SPIP WEBMESTRES

Table des matières

Où placer les fichiers de squelettes ?

Dans le dossier `squelettes/` :-)

Août 2006 — maj : 9 janvier

Depuis SPIP 1.8, les squelettes sont rangés dans un dossier dédié, nommé `dist/`. Le dossier `squelettes/` accueillera vos squelettes personnalisés.

Les avantages de ce rangement sont évidents : meilleure séparation du code de SPIP et de la structure du site, possibilité de changer tout un ensemble de squelettes d'un seul coup, etc.

> *Historique :* Dans les versions antérieures à SPIP 1.8, les fichiers de squelettes fournis dans la distribution de SPIP étaient placés à la racine. Voir : « Qu'est-ce que les fichiers « dist » ? ».

Les squelettes par défaut : `dist/`

Depuis SPIP 1.8, les squelettes de la distribution — c'est-à-dire ceux fournis en standard à l'installation de SPIP — sont regroupés dans le répertoire `dist/`. Ces fichiers contiennent les informations sur la mise en page par défaut du site et ne doivent pas être modifiés. Vous pouvez examiner le contenu de ce répertoire et partir de ce jeu de squelettes pour adapter la mise en page à vos besoins [1].

Toutefois, **il ne faut surtout pas modifier les fichiers du répertoire** `dist/`, sinon vous risqueriez de perdre toutes vos modifications à chaque mise à jour de SPIP !
Pour éviter cela, **faites une copie des fichiers que vous souhaitez modifier**, et placez-les dans un autre répertoire, comme indiqué ci-après.

Votre dossier `squelettes/`

Depuis SPIP 1.8, les squelettes personnalisés doivent être rangés dans un répertoire nommé `squelettes/` (attention au « s » final !), que vous créerez à la racine de votre site SPIP. Que vous souhaitiez installer un jeu complet de squelettes (pris sur SPIP - Contrib ou ailleurs), ou apporter une légère modification aux squelettes par défaut, placez vos squelettes dans ce répertoire.

Ainsi, un utilisateur qui veut créer sa propre mise en page, développera ses propres fichiers `article.html`, `rubrique.html`, etc. dans le répertoire `squelettes/`. Notez bien qu'il n'est pas indispensable de placer un jeu de squelettes complet dans ce répertoire.

Pour afficher les pages du site, SPIP cherche les squelettes prioritairement dans le dossier `squelettes/` ; si SPIP n'y trouve pas un fichier `.html` qui lui est nécessaire, il ira chercher celui de la distribution dans le dossier `dist/`.
Ainsi, si vous n'avez placé qu'un seul fichier dans le dossier `squelettes`, par exemple `article.html`, SPIP utilisera ce squelette pour afficher les articles, et ceux de la `dist` pour toutes les autres pages du site.

Le dossier `squelettes/` **est destiné à recevoir tous les fichiers nécessaires à la mise en page d'un site**. On y rangera donc :

- les squelettes, c'est-à-dire les fichiers `.html` avec du code SPIP ;
- les fichiers inclus dans les squelettes (ainsi que, pour les versions antérieures à SPIP 1.9, leur

fichier php3 correspondant) et les modèles (depuis SPIP 1.9) ;
- les formulaires modifiés, de préférence dans un sous-répertoire `formulaires/`
- les feuilles de style CSS qui produisent l'habillage graphique ; les personnaliser permet en effet de varier, parfois spectaculairement, l'habillage d'un site sans intervenir dans les squelettes. Voir : « Mettez-y votre style ! » ;
- les images utilisées dans les squelettes ;
- le fichier `mes_fonctions.php` contenant les filtres et variables de personnalisation propres à ce jeu de squelettes ;
- les fichiers javascripts ;
- les fichiers de langue personnalisés (cf. : « Internationaliser les squelettes », méthode des fichiers de langues), de préférence dans un sous-répertoire `lang/` ;
- etc...

Utiliser un autre dossier de squelettes

Depuis SPIP 1.5 il est possible de ranger les squelettes dans un répertoire portant le nom de votre choix, en le déclarant dans le fichier `ecrire/mes_options.php`, avec la variable de personnalisation `$dossier_squelettes`, comme expliqué dans la documentation correspondante. SPIP ira chercher les squelettes en priorité dans le répertoire ainsi déclaré.

Ceci vous permet, par exemple, d'essayer un nouveau jeu de squelettes sans écraser l'ancien, ou de gérer dynamiquement plusieurs jeux de squelettes.

Priorité des dossiers de squelettes

Soyons plus exhaustifs et résumons. Grosso modo, lorsque SPIP doit utiliser un fichier, il le cherche dans différents répertoires dans l'ordre suivant :

1. en premier lieu dans liste de dossiers désignés dans variable `$dossier_squelettes`, si celle-ci est définie ;
2. ensuite dans le dossier `squelettes/` situé à a racine du site ;
3. puis (depuis SPIP 1.9) dans la liste de dossiers de la variable `$plugins` ;
4. ensuite à la racine du site ;
5. dans le répertoire `dist/` ;
6. et enfin dans le répertoire `ecrire/`.

« Grosso modo », car à cela s'ajoutent quelques subtilités [2], dont un ordre de priorité par fichier de squelette, qui permet des variantes plus fines : par rubrique, par branche ou par langue, comme expliqué dans la documentation correspondante.

> *Remarque :* Depuis SPIP 1.9 : En fait, le mécanisme décrit ci-dessus pour choisir l'emplacement d'un fichier ne s'applique pas seulement aux squelettes, mais aussi à l'ensemble du code de SPIP. Dans le jargon des développeurs on parle de « surcharger du code », l'ordre de choix des dossiers étant dans le « SPIP_PATH ». Cela met en place le cadre et les normes pour le développement des « plug-ins », extensions des fonctionnalités de SPIP que tout un chacun dans la communauté peut apporter.
>
> Ainsi, on peut modifier à souhait n'importe quelle caractéristique du comportement de SPIP sans pour cela se priver à l'avenir des évolutions de la distribution et du support de la communauté. SPIP est devenu modulaire !

Historique : Le fait que SPIP recherche des squelettes à la racine est historique, car c'était le premier endroit où ils étaient placés. Cela avait l'avantage de rendre les squelettes « visibles » dans un navigateur, puisque les liens aux `.css` et autres habillages graphiques étaient forcément saisis « en dur » depuis la racine.
De plus, jusqu'à SPIP 1.8.3, SPIP cherchait les squelettes à la racine *avant* le dossier `squelettes/`.

Et les fichiers `.php3` (ou `.php`) dans tout ça ?

Rappelons avant tout qu'à partir de SPIP 1.9, il n'y a plus de fichier `.php3` (ou `.php`) pour les squelettes : SPIP calcule toutes ses pages à partir du script unique `spip.php`. Tout ce qui suit est donc historique.

Dans SPIP 1.8.2 et SPIP 1.8.3, il existait un fichier **page.php3** qui préfigurait le spip.php et la forme d'inclusion de SPIP 1.9. En effet, **page.php3** permettait, à lui seul, d'appeler n'importe quel squelette en passant en paramètre la variable de fond, c'est-à-dire le fichier squelette .html à utiliser :

www.monsite.org/page.php3?fond=mon_squelette&...

Le plus simple était alors d'utiliser un appel de ce type [3] pour tout squelette autre que ceux des objets de base (article, breve, rubrique, sommaire,...) pour lesquels un fichier `.php3` existe à la racine, ce qui permettait l'appel habituel de la forme :

www.monsite.org/article.php3?...

Jusqu'à SPIP 1.8 il fallait créer un fichier `.php3`, qui soit le pendant de votre `.html` dans l'ancienne structure des squelettes SPIP, celui-ci devait forcément être placé à la racine du site [4].

Notes

[1] C'est même une méthode vivement conseillée, car ce jeu de squelettes a été pensé pour être aussi modulaire que possible.

[2] Comme une nomenclature des fichiers en fonction de leur rôle, ce qui fait, par exemple, que SPIP ira chercher les fichiers de langue dans un sous répertoire `lang/`, comme nous l'avons vu plus haut.

[3] Il est à noter que cette méthode fonctionne aussi pour les appels avec la balise `<INCLURE(page.php3){fond=inc-entete}>`.

[4] Sauf, éventuellement, ceux qu'on n'utilisait que dans un appel `<INCLURE(squel.php3)>` et pas directement par une URL. Dans ce cas, le fichier squel.php3 pouvait *aussi* être déplacé dans le dossier de squelettes.

Principe général

Mai 2007 — maj : Juillet 2007

Tout le contenu d'un site géré sous SPIP est installé dans une base de données MySQL. Pour présenter ces informations aux visiteurs du site, il faut donc réaliser l'opération qui consiste à lire les informations, à les organiser et à les mettre en page, afin d'afficher une page HTML dans le navigateur Web.

A moins d'utiliser un gestionnaire de contenu avancé comme SPIP, cette opération est assez fastidieuse :
- il faut connaître la programmation PHP et MySQL, et écrire des « routines » relativement complexes ;
- l'intégration de telles routines dans une mise en page HTML élaborée est assez pénible ;
- il faut développer toute une interface pour que les utilisateurs autorisés modifient le contenu de la base de données ;
- il faut prendre en compte des problèmes de performances : le recours systématique à du code MySQL et PHP est gourmand en ressources, ralentit la visite et, dans des cas extrêmes, provoque des plantages du serveur Web.

SPIP propose une solution complète pour contourner ces difficultés :
- la mise en page du site est effectuée au moyen de gabarits au format HTML nommés *squelettes*, contenant des instructions simplifiées permettant d'indiquer où et comment se placent les informations tirées de la base de données dans la page ;
- un système de cache permet de stocker chaque page et ainsi d'éviter de provoquer des appels à la base de données à chaque visite. Non seulement la charge sur le serveur est réduite, la vitesse très largement accélérée, de plus un site sous SPIP reste consultable même lorsque la base MySQL est plantée ;
- un « espace privé » permettant aux administrateurs et aux rédacteurs de gérer l'ensemble des données du site.

Pour chaque type de page, un squelette

L'intérêt (et la limite) d'un système de publication automatisé, est de définir un gabarit pour, par exemple, tous les articles. On indique dans ce gabarit (le squelette) où viendront se placer, par exemple, le titre, le texte, les liens de navigation... de l'article, et le système fabriquera chaque page individuelle d'article en plaçant automatiquement ces éléments tirés de la base de données, dans la mise en page conçue par la/le webmestre.

Ce système automatisé permet donc une présentation cohérente à l'intérieur d'un site... Et c'est aussi sa limite : il ne permet pas de définir une interface différente pour chaque page isolée (on verra plus loin que SPIP autorise cependant une certaine souplesse).

Lorsque vous installez SPIP, un jeu de squelettes est proposé par défaut. Il se trouve dans le répertoire **dist/**, à la racine de votre site. Dès que vous modifiez ces fichiers pour les adapter à vos besoins, ou si vous voulez installer un autre jeu de squelettes, il convient de créer un répertoire nommé **squelettes/** au même niveau. Pour plus de détails sur cette question, lire l'article Où placer ses squelettes .

Lorsqu'un visiteur demande la page `http://exemple.org/spip.php?article3437`, SPIP va chercher un squelette nommé « article.html ». SPIP se base donc sur l'adresse URL de la

page pour déterminer le squelette à utiliser :

L'URL	appellera le squelette
`spip.php?article3437`	`article.html`
`spip.php?rubrique143`	`rubrique.html`
`spip.php?mot12`	`mot.html`
`spip.php?auteur5`	`auteur.html`
`spip.php?site364`	`site.html`

Avec deux cas particuliers :

- L'URL `spip.php` appelle le squelette sommaire.html. Il s'agit de la page d'accueil du site.
- L'URL `spip.php?page=abcd` appelle le squelette abcd.html. En d'autres termes, vous pouvez créer des squelettes qui ne sont pas prévus par le système et les nommer comme vous le souhaitez.

Cette syntaxe sert également pour les pages telles que le plan du site ou les résultats de recherche par exemple : `spip.php?page=plan`, `spip.php?page=recherche&recherche=ecureuil`.

Lorsque SPIP appelle un squelette, il lui passe un contexte

Par ailleurs, vous aurez constaté que l'URL fournit d'autres éléments que le nom du squelette. Par exemple, dans `spip.php?article3437`, le numéro de l'article demandé (3437) ; dans `spip.php?page=recherche&recherche=ecureuil`, le mot recherché (ecureuil).

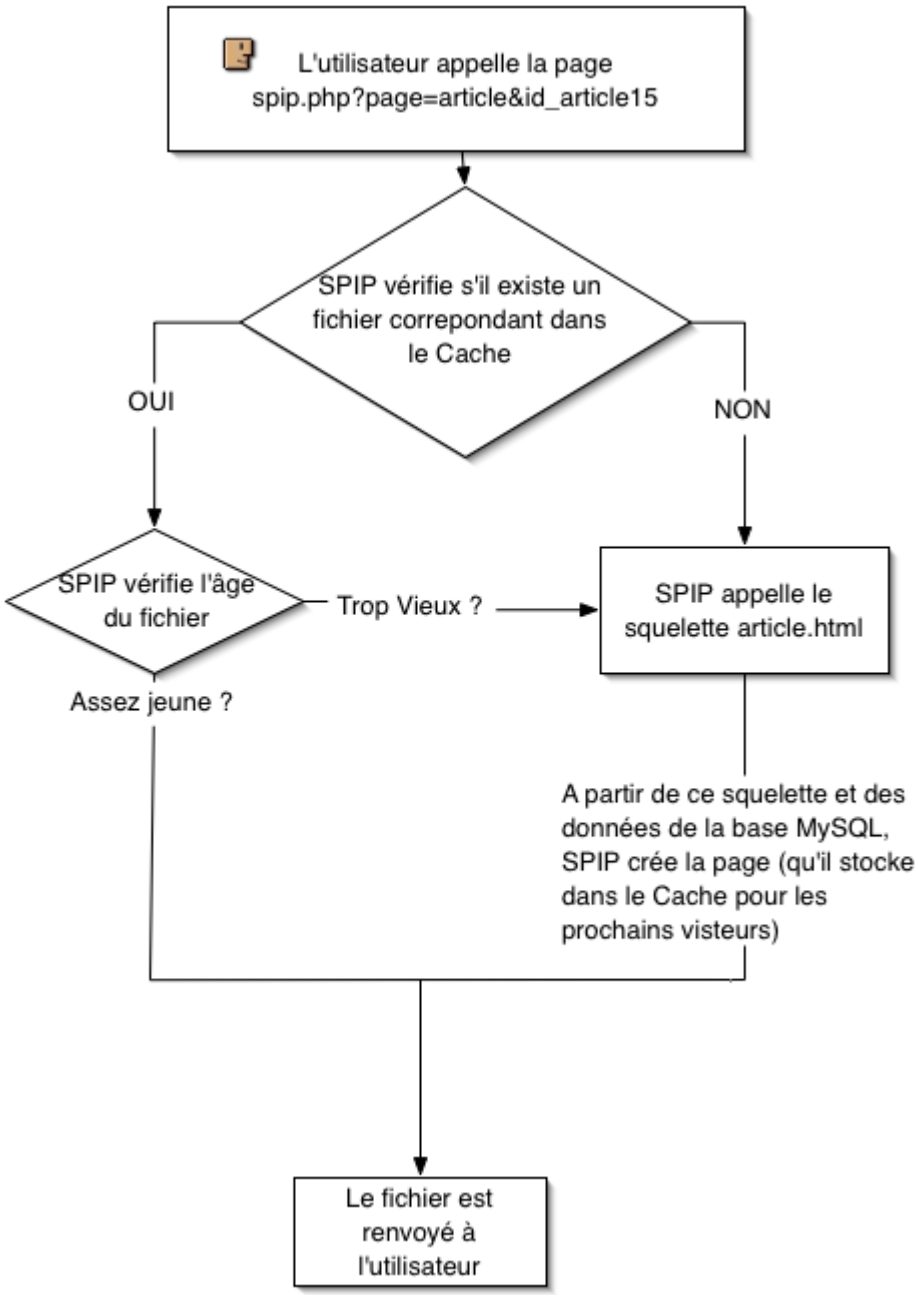
Il s'agit d'un « contexte », c'est-à-dire, une ou plusieurs « variables d'environnement », que SPIP va fournir au squelette pour qu'elles puissent être utilisées dans la composition de la page. En effet, le squelette article.html a besoin de connaître le numéro de l'article demandé pour rechercher son titre, son texte,... dans la base de données. De même, le squelette recherche.html doit connaitre les mots recherchés par le visiteur pour trouver les enregistrements de la base de données qui contiennent ces termes.

Dans toute URL, les variables d'environnement apparaissent après le « ? ». Lorsqu'il y en a plusieurs, elles sont séparées par des « & ».

Dans l'URL `spip.php?page=recherche&recherche=ecureuil`, on a donc deux variables : page et recherche, auxquelles on attribue les valeurs respectives « recherche » et « écureuil ».

Dans le cas de `spip.php?article3437`, SPIP a simplifié l'URL qui correspond en fait à : `spip.php?page=article&id_article=3437` (si si, vous pouvez essayer !). On a donc ici aussi deux variables : page a la valeur "article" et id_article a la valeur "3437". Ces variables permettent à SPIP d'utiliser les données de l'article 3437 dans le squelette article.html.

Le principe de fonctionnement du cache, de manière simplifiée



SPIP regarde si une page correspondante à l'URL demandée se situe dans le **CACHE**

1. Si la page existe, SPIP vérifie qu'elle n'est pas trop ancienne.
    1. Si la page est trop ancienne, il la recalcule à partir du squelette et de la base MySQL. Puis il la stocke dans le **CACHE**, avant de l'envoyer à l'utilisateur.
    2. Si la page est assez récente, il la retourne à l'utilisateur.
2. Si la page n'existe pas, il le calcule à partir du squelette et de la base MySQL. Puis il la stocke dans le **CACHE**, avant de l'envoyée à l'utilisateur.

Lors d'une visite suivante, si le délai entre les deux visites est suffisamment court, c'est donc cette nouvelle page stockée dans `/CACHE` qui est retournée, sans avoir à faire un nouveau calcul à partir de la base de données. En cas de plantage de la base de données, c'est forcément la page en cache qui est retournée, même si elle est « trop âgée ».

Remarque. On voit ici que chaque page du site est mise en cache individuellement, et chaque recalcul est provoqué par les visites du site. Il n'y a pas, en particulier, un recalcul de toutes les pages du site d'un seul coup à échéance régulière (ce genre de « grosse manœuvres » ayant le bon goût de surcharger le serveur et de le faire parfois planter).

Par défaut, une page est considérée comme trop vieille au bout de 3600 secondes. [1]

Le fichier .html

Dans SPIP, nous appelons les fichiers `.html` les **squelettes**. Ce sont eux qui décrivent l'interface graphique de vos pages.

Ces fichiers sont rédigés directement en HTML, auquel on ajoute des instructions permettant d'indiquer à SPIP où il devra placer les éléments tirés de la base de données (du genre : « placer le titre ici », « indiquer à cet endroit la liste des articles portant sur le même thème »...).

Les instructions de placement des éléments sont rédigées dans un langage spécifique, qui fait l'objet du présent manuel d'utilisation. Ce langage constitue par ailleurs la seule difficulté de SPIP.

« Encore un langage ? » Hé oui, il va vous falloir apprendre un nouveau langage. Il n'est cependant pas très compliqué, et il permet de créer des interfaces complexes très rapidement. Par rapport au couple PHP/MySQL, vous verrez qu'il vous fait gagner un temps fou (surtout : il est beaucoup plus simple). C'est un *markup language*, c'est-à-dire un langage utilisant des balises similaires à celles du HTML.

Remarque. De la même façon que l'on apprend le HTML en s'inspirant du code source des sites que l'on visite, vous pouvez vous inspirer des squelettes utilisés sur d'autres sites fonctionnant sous SPIP. Il suffit d'aller chercher le fichier « .html » correspondant. Par exemple, vous pouvez voir le squelette des articles de ce présent cycle (visualisez le code source pour obtenir le texte du squelette).

Une interface différente dans le même site

En plus de la mise en page par défaut des différents contenus du site (rubrique.html, article.html, etc.), vous pouvez créer des squelettes différents pour des rubriques particulières du site et leurs contenus. Il suffit de créer de nouveaux fichiers .html et de les nommer selon le principe suivant :

- Une interface différente pour une rubrique et ses contenus (sous-rubriques, articles, brèves, etc.) : il faut compléter le nom du fichier squelette correspondant par « -numéro » (un tiret suivi d'un numéro de rubrique).

Par exemple, si vous créez un fichier rubrique-60.html, il s'appliquera à la rubrique n°60 et à ses sous-rubriques en lieu et place de rubrique.html. Le squelette article-60.html s'appliquera aux articles de la rubrique n°60. Si cette rubrique contient des sous-rubriques, leurs articles adopteront également le nouveau squelette. Idem pour breve-60.html... et ainsi de suite.

- Une interface pour une seule rubrique. (SPIP 1.3) On peut créer une interface qui s'applique à une rubrique, mais pas à ses sous-rubriques. Pour cela, il faut compléter le nom du fichier squelette correspondant par « =numéro ».

Par exemple, il faut créer un fichier rubrique=60.html, qui s'appliquera uniquement à la rubrique n °60, mais pas à ses sous-rubriques. Idem pour article=60.html, breve=60.html, etc. Ces squelettes s'appliquent aux contenus de la rubrique n°60 mais ceux de ses sous-rubriques retrouvent

l'habillage par défaut.

> Notez bien : le numéro indiqué est celui d'une rubrique. Les squelettes article-60.html ou article=60.html ne concernent donc pas l'article n°60 mais bien tous les articles de la rubrique n°60.

Que peut-on mettre dans un fichier .HTML

Les fichiers `.html` sont essentiellement des fichiers « texte », complétés d'instructions de placement des éléments de la base de données.

SPIP analyse uniquement les instructions de placement des éléments de la base de données (codées selon le langage spécifique de SPIP) ; il se contrefiche de ce qui est placé dans ce fichier et qui ne correspond pas à ces instructions.

Leur contenu essentiel est donc du HTML. Vous déterminez la mise en page, la version du HTML désiré, etc. Vous pouvez évidemment y inclure des feuilles de style (CSS), mais également du JavaScript, du Flash... en gros : tout ce qu'on place habituellement dans une page Web.

Mais vous pouvez également (tout cela n'est jamais que du texte) créer du XML (par exemple, « backend.html » génère du XML).

Plus original : toutes les pages retournées au visiteur sont tirées du **CACHE** par une page écrite en PHP. Vous pouvez donc inclure dans vos squelettes des instructions en PHP, elles seront exécutées lors de la visite. Utilisée de manière assez fine, cette possibilité apporte une grande souplesse à SPIP, que vous pouvez ainsi compléter (par exemple ajouter un compteur, etc.), ou même faire évoluer certains éléments de mise en page en fonction des informations tirées de la base de données.

P.-S.

Cet article n'est valable qu'à partir de SPIP 1.9. Pour les versions antérieures, lire la version archivée.

Notes

[1] Une fois que vous aurez compris le langage SPIP, vous pourrez modifier ce délai en vous servant de la balise #CACHE.

Créer mon premier squelette

Mon premier squelette

(je le sors du placard)

Juin 2001 — maj : 27 juin

Si le système de squelettes peut de prime abord paraître intimidant, c'est que ce qu'on lui demande est suffisamment riche pour l'obliger à être complexe. Mais ! Complexe ne veut pas dire compliqué. Voici un exemple minimal de squelette.

Matériel requis pour ce tutoriel

- Un SPIP installé quelque part. On supposera, pour commencer, que votre base SPIP contient au **minimum une rubrique et deux articles publiés**. Si ce n'est pas le cas, vous pouvez très vite y remédier en copiant-collant les premiers textes qui vous passent sous la main (vérifiez quand même qu'il ne s'agit pas de votre déclaration enflammée au petit ami de votre voisin de bureau).

- Un éditeur de texte pour créer et modifier les fichiers utilisés par SPIP. *Note :* certaines personnes auront le réflexe de vouloir utiliser DreamWeaver (ou autre logiciel graphique) pour modifier les fichiers `.html`. Cependant pour des exemples simples DreamWeaver compliquera la tâche et risque même de modifier vos fichiers dans votre dos. Il est donc vraiment préférable d'utiliser un éditeur de texte classique (par exemple le bloc-notes sous Windows).

1. Dans les versions antérieures à SPIP 1.9, avant d'utiliser un squelette, il faut pouvoir l'appeler. Si vous utilisez une version supérieure de SPIP, passez directement à l'étape suivante, ci-après. Sinon, pour appeler le squelette, créez à la racine de votre site un fichier `tutoriel.php3` contenant les lignes suivantes :

```
<?php
$fond = "tutoriel";
$delais = 0;
include "inc-public.php3";
?>
```

Puis testez dans votre navigateur : `http://votresite.net/tutoriel.php3`. Pas très glorieux, n'est-ce pas ? Le message d'erreur vous informe qu'il manque un fichier. C'est le fameux squelette, que nous allons maintenant créer.

SPIP 1.9 a nettement simplifié la création de squelettes, en vous épargnant cette première étape d'appel du squelette. En effet, à partir de la version 1.9, il n'y a plus de fichier `.php3` (ou `.php`) pour les squelettes, qui sont tous calculés à partir du script unique `spip.php`.
La suite de ce tutorial reste valable quelque soit la version de SPIP utilisée.

2. Dans le dossier `squelettes/` (à créer manuellement s'il n'existe pas à la racine du site),

déposez un fichier `tutoriel.html`, qui contient ce qui suit :

<BOUCLE_article(ARTICLES){id_article=1}>

#TITRE

</BOUCLE_article>

Puis affichez la page `http://votresite.net/spip.php?page=tutoriel` (ou rechargez la page `http://votresite.net/tutoriel.php3`, si vous utilisez une version antérieure à SPIP 1.9) :

SPIP est allé chercher le titre de l'article n°1 de votre base, et l'a inscrit à la place de `#TITRE`.

Si ça ne fonctionne pas, vérifiez que votre article n°1 est bien « publié » (et pas « en attente » ou « en cours de rédaction »).

Puis ajoutez du HTML et d'autres appels de « champs » SPIP, et vous obtenez rapidement votre article n° 1 :

<BOUCLE_article(ARTICLES){id_article=1}>

<h1>#TITRE</h1>

#CHAPO

#TEXTE

</BOUCLE_article>

Ajoutez ensuite les champs manquants pour parfaire l'affichage de l'article : `#SURTITRE`, `#LESAUTEURS`, `#SOUSTITRE`, `#NOTES`, etc.

Bien !

Un squelette, plusieurs articles...

c'est à ça que ça sert !

Juin 2001 — maj : 17 mars

La leçon précédente nous a permis d'extraire des données de l'article n°1 de la base et d'en faire une page Web. Généralisons...

Notre squelette est bien inutile s'il ne sert qu'à afficher l'article n°1. **Apprenons-lui à afficher n'importe quel article :**

Pour cela nous allons appeler notre page Web avec un paramètre, du type `id_article=2` : dirigez votre navigateur sur l'URL suivante :
« `http://votresite.net/spip.php?page=tutoriel&id_article=2` » [*]

S'affiche... toujours l'article 1 (et pas 2). Modifions dans le squelette `tutoriel.html` la ligne qui définit la « boucle article » :
`<BOUCLE_article(ARTICLES){id_article}>`
Comme vous le voyez, on remplace simplement `{id_article=1}` par `{id_article}` tout court.

Voilà : `http://votresite.net/spip.php?page=tutoriel&id_article=2` vous donne maintenant l'article 2. [1]

La `BOUCLE_article` s'exécute dans un « *contexte* » où `id_article` est égal à 2 (c'est la valeur qui est passée dans l'URL). Si on lui précise `{id_article=1}` elle va chercher l'article n ° 1, mais si on lui demande juste `{id_article}`, elle va chercher l'article dont le numéro est indiqué par le contexte (ici l'URL).

Visitez maintenant ces pages :
- `http://votresite.net/spip.php?page=tutoriel&id_article=1`,
- `http://votresite.net/spip.php?page=tutoriel&id_article=2` et
- `http://votresite.net/spip.php?page=tutoriel` [*].

Voyez-vous la différence ? Les deux premières pages vous donnent les articles n°1 et 2, la troisième n'a pas d'`id_article` dans son contexte, et génère une erreur.

NB : N'oubliez pas de recalculer la page (bouton en haut à droite de votre page) pour prendre en compte les modifications de votre squelette.

Bravo ! Votre squelette est maintenant « contextuel ».

Notes

[*] <*>Rappelons que dans les versions antérieures à SPIP 1.9, l'URL pour afficher notre tutorial est : `http://votresite.net/tutoriel.php3`. Lorsqu'on lui passe un paramètre : `http://votresite.net/tutoriel.php3?id_article=2`, etc.

[1] Non ? Il devrait...

[*] <*>

Une rubrique

ou comment faire des listes du contenu de la base

Juin 2001 — maj : Janvier 2007

La leçon précédente nous a appris à afficher des éléments en fonction du contexte. Nous allons ici voir comment ce contexte varie au fur et à mesure des `BOUCLES` rencontrées.

Modifions notre squelette « `tutoriel.html` » de la manière suivante :

<BOUCLE_article(ARTICLES)>

#TITRE

</BOUCLE_article>

Là, on supprime carrément la condition `{id_article}`. Attention : cette `BOUCLE` peut générer une page énorme si votre base contient déjà pas mal d'articles : mieux vaut prendre nos précautions et ajouter tout de suite `{0,10}` pour limiter aux 10 premiers articles...

`<BOUCLE_article(ARTICLES){0,10}>`

Résultat : en appelant simplement `http://votresite.net/spip.php?page=tutoriel` [*] (plus besoin d'`id_article` désormais, puisque cette condition a été

supprimée) les titres des 10 premiers articles publiés s'affichent, séparés chacun par un saut de ligne. À partir de là, on voit comment on peut produire le sommaire d'une rubrique : affichons les 10 articles les plus récents appartenant à cette rubrique.

<BOUCLE_article(ARTICLES){id_rubrique}{par date}{inverse}{0,10}>

#TITRE

</BOUCLE_article>

Prenons dans l'ordre :

- `id_rubrique` : ne prend que les articles appartenant à la rubrique `id_rubrique` (*cf.* ci-dessous pour que cette variable soit définie dans le contexte de notre `BOUCLE_article`) ;
- `{par date}{inverse}` : trie par date dans l'ordre chronologique décroissant...
- `{0,10}` : ... et prend les 10 premiers résultats.
- Enfin, `<a href="#URL_ARTICLE">#TITRE</a>` va afficher non seulement le titre de l'article mais en plus créer un lien vers cet article.

Reste à invoquer le squelette, *en lui passant le contexte* `id_rubrique=1` :

`http://votresite.net/spip.php?page=tutoriel&id_rubrique=1` [*]

La magie de SPIP tient dans la combinaison de ce type de fonctionnalités. Si vous êtes arrivé jusqu'ici, c'est gagné !

Notes

[*] <*>Rappelons que dans les versions antérieures à SPIP 1.9, l'URL pour afficher notre tutorial est : `http://votresite.net/tutoriel.php3`. Lorsqu'on lui passe un paramètre : `http://votresite.net/tutoriel.php3?id_rubrique=1`, etc.

[*] <*>

Boucles en boucles

plusieurs niveaux de lecture

Juin 2001 — maj : Décembre 2007

Nous savons générer une liste de titres dans une rubrique. Maintenant, nous allons afficher, sur la même page, les éléments de la rubrique elle-même : son titre, son texte de présentation, etc.

Essayez !

Et voici une solution :

<BOUCLE_rubrique(RUBRIQUES){id_rubrique}>

<h1>#TITRE</h1>

<BOUCLE_article(ARTICLES){id_rubrique}{par date}{inverse}{0,10}>

#TITRE

</BOUCLE_article>

[(#TEXTE|justifier)]

</BOUCLE_rubrique>

On appelle la page avec `http://votresite.net/spip.php?page=tutoriel&id_rubrique=1` [*].
Que s'est-il passé ici ?

Notre boucle `ARTICLES` est intégrée dans une boucle `RUBRIQUES`. Le contexte de la boucle `ARTICLES` est l'`id_rubrique` donné par la boucle `RUBRIQUES`, qui elle-même va chercher le contexte donné par l'URL (`id_rubrique=1`). Donc nous sommes bien, au niveau des `ARTICLES`, avec l'`id_rubrique` demandé. De ce point de vue rien ne change.

En revanche, la boucle `RUBRIQUES` a permis à SPIP de sélectionner les valeurs des champs de la rubrique en question : on peut donc afficher le `#TITRE` et le `#TEXTE` de cette rubrique. Notez bien que ce `#TEXTE` serait celui de la rubrique *même si* on appelait aussi `#TEXTE` dans la boucle `ARTICLES`. Le fonctionnement arborescent de SPIP garantit que le `#TEXTE` d'un article ne déborde pas de la boucle `ARTICLES`...

Dernière remarque : on a introduit un *filtre* `|justifier` sur le champ `#TEXTE`. Ce filtre modifie le contenu du texte avant de l'installer dans la page finale. Ca vous fait saliver ?

Notes

[*] <*>Rappelons que dans les versions antérieures à SPIP 1.9, l'URL pour afficher notre tutorial est : `http://votresite.net/tutoriel.php3`. Lorsqu'on lui passe un paramètre : `http://votresite.net/tutoriel.php3?id_rubrique=1`, etc.

Des filtres

Subtilités squelettiques

Juin 2001 — maj : Octobre 2003

Si les `BOUCLES` permettent de structurer la page de manière logique, reste à présenter les données de manière esthétique. Question dizahïgne SPIP ne peut rien pour vous, mais sachez user de ses philtres...

Une donnée stockée dans la base de données se présente comme un bloc de texte, et on peut avoir envie de manipuler sa valeur avant de l'afficher à l'écran. Les *filtres* sont faits pour ça :

- les filtres les plus utilisés (ils sont appelés automatiquement) sont `|typo` et `|propre` ; le premier est un correcteur typographique, dont la mission principale est d'ajouter des espaces insécables où il en faut (*cf.* l'aide en ligne de SPIP) ; le second s'intéresse aux paragraphes, aux raccourcis SPIP (italiques, gras, intertitres, etc.) - il n'est appliqué par défaut qu'aux textes longs (`#TEXTE`, `#CHAPO`, etc.)

- d'autres filtres sont très utiles : citons `|majuscules` (à la fonctionnalité évidente), `|justifier` ou `|aligner_droite` (qui définissent l'alignement du texte par rapport aux bords verticaux), ou encore l'ésotérique `|saison` (qui affiche « été » si la variable est une date comprise entre le 21 juin et le 20 septembre)...

Pour utiliser un filtre il faut entourer la variable de parenthèses et de crochets (on verra plus tard les implications) : `[blah blah (#VARIABLE|filtre) bloh bloh]`

On peut enchaîner les filtres les uns à la suite des autres [1] : ainsi `[(#DATE|saison|majuscules)]` affichera-t-il « HIVER ».

Exercice portant sur l'ensemble des leçons précédentes : Afficher en majuscules les titres des 10 articles les plus récents de la rubrique passée en contexte, et mettre en tête de page la saison courante (c'est-à-dire la saison à laquelle a été publié l'article le plus récent de toute la base).

...

Pourquoi ces crochets ? Supposons que votre base contient des articles datés et d'autres non datés. La variable #DATE vaut « 2001-07-01 10-53-01 » (date au format mySQL) dans le premier cas, et « 0000-00-00 00-00-00 » dans le second. Pour afficher la date dans un joli (?) cadre, on va utiliser, dans le squelette, les lignes suivantes :

[<table border="1"><tr><td>

(#DATE|affdate)

</td></tr></table>]

Ici le filtre `|affdate` affiche la date en lettres (au format « 1er juillet 2001 »), mais renvoie une chaîne vide si la date est inconnue (égale à « 0000... »). Les crochets délimitent ce qu'il faut afficher autour de la date *si le resultat entre parenthèses n'est pas une chaîne vide*.

Résultat : seuls les articles datés provoquent l'affichage d'un tableau contenant la date. Un squelette bien construit définira précisément ce qu'il faut afficher *ou pas* en fonction du contenu... Les filtres servent aussi à ça.

P.-S.

Notons que certains filtres de présentation peuvent être avantageusement remplacés par des feuilles de style. Ainsi `|majuscules` est équivalent à l'attribut CSS « `text-transform: uppercase` », et `|justifier` à « `text-align: justify` ».

Lire *Les feuilles de style de SPIP* pour plus de détails sur les styles CSS offerts par SPIP.

Notes

[1] On peut appeler ça un « *pipeline* »...

Manuel de référence des boucles et balises

Des boucles et des balises

Juin 2001 — maj : Novembre 2003

Tout ce qui suit concerne désormais le langage de description de la mise en page des *squelettes* dans SPIP ; si vous avez bien compris l'explication précédente, vous savez que nous travaillons donc dans les fichiers « .html ».

La présente documentation est volontairement technique (il s'agit ici de références techniques) ; vous pouvez préférer commencer par notre guide Pas à pas, plus didactique, et revenir ensuite ici pour une documentation plus précise.

Des boucles

La notion de base du langage de SPIP est la *boucle*.

- La logique de la boucle

Une base de données, classiquement, c'est une liste d'éléments : ici, une liste des articles, une liste des rubriques, une liste des auteurs, etc. Pour « fabriquer » le site, on va donc extraire de cette liste certains de ses éléments :
- à la base, on veut extraire *un seul élément* d'une liste ; par exemple, afficher l'article désiré ;
- mais il est fréquent d'extraire *plusieurs éléments* d'une liste ; par exemple, dans la page d'une rubrique, on veut afficher *tous* les articles contenus dans cette rubrique, ainsi que *toutes* les sous-rubriques contenues dans cette rubrique ;
- plus subtil : il arrive fréquemment qu'il n'y ait pas d'éléments satisfaisants à tel ou tel endroit ; SPIP doit alors pouvoir gérer l'éventualité de l'absence de ces éléments ; par exemple, le squelette de l'affichage des rubriques demande l'affichage de toutes les sous-rubriques contenues dans une rubrique ; que faire, alors, s'il n'y a pas sous-rubriques dans cette rubrique spécifique ?

Ces trois situations sont traitées par la notion unique de *boucle*, qui permet à la fois de gérer l'affichage d'un seul élément, de plusieurs éléments successifs, ou l'absence d'éléments.

Le système de boucle permet, dans un code unique :
- d'indiquer à quel endroit du code HTML on a besoin de quel type d'élément (à tel endroit on veut récupérer la liste des articles, à tel endroit on veut inclure la liste des sous-rubriques...) ;
- de prévoir l'affichage d'un élément unique ;
- d'indiquer comment est affichée une liste de plusieurs éléments ;
- de déterminer ce qu'on affiche lorsqu'il n'y a aucun élément correspondant.

Analogie avec la programmation en PHP/mySQL

Ceux qui ont déjà programmé des requêtes mySQL en PHP savent que le traitement se déroule en deux temps :
- la construction de la syntaxe de la requête (qui consiste à dire « je veux récupérer la liste des articles contenus dans telle rubrique... ») ;
- l'analyse et l'affichage des résultats au travers d'une boucle.

Ce sont ces deux événements qui sont gérés, dans SPIP, au travers des boucles.

Les balises SPIP

Grâce aux boucles, on a donc récupéré des éléments uniques ou des listes d'éléments : par exemple

une liste d'articles ou une liste de rubriques...

Cependant, chaque élément de telles listes est composé de plusieurs éléments précis : par exemple un article se compose d'un titre, d'un surtitre, d'un sous-titre, d'un texte d'introduction (chapeau), d'un texte principal, d'un post-scriptum, etc. Il existe ainsi des balises spécifiques à SPIP, permettant d'indiquer précisément à quel endroit on affiche des éléments : « placer le titre ici », « placer le texte ici »...

Les balises à l'intérieur des boucles

Voici, au travers d'un cas classique, le principe de fonctionnement général d'une boucle accompagnée de ses balises (attention, ça n'est pas du langage SPIP, c'est une description logique) :

```
BOUCLE : afficher la liste des articles de cette rubrique
```

- `afficher ici le titre de l'article`
- `afficher le sous-titre`
- `afficher le texte`

```
Fin de la BOUCLE
```

Cette boucle, analysée par SPIP, peut donner trois résultats différents.

- **Il n'y a aucun article dans cette rubrique.**

Dans ce cas, bien évidemment, aucun des éléments « afficher ici... (titre, sous-titre...) » n'est utilisé. En revanche, si on l'a prévu, on peut afficher un message du genre « Il n'y a pas d'article ».

- **Il y a un seul article dans cette rubrique.**

Dans ce cas, très simplement, la page HTML est construite sur le modèle de la boucle :

- `Titre de l'article`
- `Sous-titre`
- `Texte de l'article`

- **Il y a plusieurs articles dans cette rubrique.**

La description de la mise en page (« placer ici... ») va alors être calculée successivement pour chacun des articles. Ce qui donne simplement :

- `Titre de l'article 1`
- `Sous-titre de l'article 1`
- `Texte de l'article 1`
- `Titre de l'article 2`
- `Sous-titre de l'article 2`
- `Texte de l'article 2`

  `...`
- `Titre du dernier article`
- `Sous-titre du dernier article`
- `Texte du dernier article`

La suite de ce guide de référence se construira donc de la manière suivante :
- syntaxe générale des boucles ;
- syntaxe générale des balises de SPIP ;

- et, ensuite, une page spécifique à chaque type de boucles, indiquant quelles balises on peut y utiliser.

La syntaxe des boucles

Mai 2001 — maj : 28 juillet

Syntaxe de base

La syntaxe simplifiée d'une boucle est la suivante :

<BOUCLE*n*`(TYPE){critère1}{critère2}...{critèrex}`**>**
`* Code HTML + balises SPIP`
</BOUCLE*n***>**

On a vu, dans l'explication sur les boucles et les balises, que le « `Code HTML + balises SPIP` » se répétait autant de fois que la boucle obtenait d'éléments tirés de la base de données (c'est-à-dire une fois, plusieurs fois, ou zéro fois).

La ligne importante, ici, est :

<BOUCLE*n*`(TYPE){critère1}{critère2}...{critèrex}`**>**

- **L'élément** `BOUCLE` est l'ordre indiquant qu'il s'agit d'une boucle SPIP ; on ne peut donc pas le modifier ; dit autrement, toutes les boucles de SPIP commencent par l'instruction `BOUCLE`.

- **L'élément *n*** est le nom ou le numéro de la boucle, librement choisi par le webmestre, pour chaque boucle qu'il utilise. On verra plus loin qu'il est possible (c'est même tout l'intérêt de la manœuvre) d'utiliser plusieurs boucles dans un même squelette : les nommer est donc indispensable pour les identifier.

Si vous décidez de numéroter vos boucles, la syntaxe devient par exemple (pour la boucle 5) :
`<BOUCLE5...> ... </BOUCLE5>`

Si vous décidez de donner un nom à vos boucles (c'est généralement plus pratique, votre code est plus lisible), il faut impérativement faire précéder ce nom par le symbole « _ » (que l'on appelle habituellement *underscore*). Par exemple :
`<BOUCLE_sousrubriques...> ... </BOUCLE_sousrubriques>`

- **L'élément** `(TYPE)` est primordial : il indique quel type d'éléments on veut récupérer. La syntaxe est importante : le TYPE est indiqué entre parenthèses (sans espaces), en majuscules, et ce TYPE doit correspondre obligatoirement à l'un des types prévus dans SPIP (qu'on trouvera dans la présente documentation) : ARTICLES, RUBRIQUES, AUTEURS, BREVES, etc.

Pour l'exemple précédent, on aurait donc :

```
<BOUCLE_sousrubriques(RUBRIQUES)...>
...
</BOUCLE_sousrubriques>
```

- **Les critères** `{critère1}{critère2}...{critèrex}` indiquent à la fois selon quels critères on veut sélectionner les éléments de la base de données (afficher les sous-rubriques incluses dans cette rubrique, afficher les autres rubriques installées au même niveau hiérarchique que la présente rubrique...), et la façon dont on va classer ou sélectionner les éléments (classer les articles selon leur date, selon leur titre... afficher uniquement les 3 premiers articles, afficher la moitié des articles...). Comme on peut combiner les critères, on peut très aisément fabriquer des requêtes très puissantes, du genre « afficher la liste des 5 articles les plus récents écrits par cet auteur ».

Les critères sont entre acolades ; ils peuvent être séparés les uns des autres par un espace. Exemple :

```
<BOUCLE_meme_auteur(ARTICLES) {id_auteur} {par date}{inverse}
{0,5}>
...
</BOUCLE_meme_auteur>
```

Les différents critères et leur syntaxe seront explicités par la suite, pour chaque type de boucle (certains critères fonctionnent pour tous les types de boucles, d'autres sont spécifiques à certaines boucles).

Syntaxe complète

Le syntaxe indiquée précédemment peut être complétée par des éléments conditionnels. En effet, la boucle précédente affiche successivement les éléments contenus *à l'intérieur* de la boucle. SPIP permet de plus d'indiquer ce qu'on affiche avant et après la boucle au cas où elle contient un ou plusieurs résultats, et ce qu'on affiche s'il n'y a aucun élément.

Cela donne :

<B*n***>**
`* Code HTML optionnel avant`
<BOUCLE*n***(TYPE){critère1}{critère2}...{critèrex}>**
`* Code HTML + balises SPIP`
</BOUCLE*n***>**
`* Code HTML optionnel après`
</B*n***>**
`* Code HTML alternatif`
<//B*n***>**

Le *code optionnel avant* (précédé de <Bn>) n'est affiché que si la boucle contient au moins une réponse. Il est affiché avant les résultats de la boucle.

Le *code optionnel après* (terminé par </Bn>) n'est affiché que si la boucle contient au moins une réponse. Il est affiché après les résultats de la boucle.

Le *code alternatif* (terminé par <//Bn>) est affiché à la place de la boucle (et donc également à la place des codes optionnels avant et après) si la boucle n'a trouvé aucune réponse.

Par exemple, le code :

```
<B1>
 Cette rubrique contient les éléments suivants:
 <ul>
   <BOUCLE1(ARTICLES){id_rubrique}>
   <li>#TITRE</li>
   </BOUCLE1>
 </ul>
</B1>
 Cette rubrique ne contient pas d'article.
<//B1>
```

donne les résultats suivants :

- **s'il y a un seul article :**

```
    Cette rubrique contient les éléments suivants:
    <ul>
      <li>Titre de l'article</li>
```

```
</ul>
```

- **s'il y a plusieurs articles :**

```
Cette rubrique contient les éléments suivants:
<ul>
  <li>Titre de l'article 1</li>
  <li>Titre de l'article 2</li>
  ...
  <li>Titre du dernier article</li>
</ul>
```

- **s'il n'y a aucun article :**

```
Cette rubrique ne contient pas d'article.
```

Historique : Jusqu'à [SPIP 1.7.2], La manière dont SPIP interprétait les boucles interdisait de mettre une boucle entre `<Bn>` et `<BOUCLEn>`. Par contre, il restait possible de mettre des boucles supplémentaires dans les parties optionnelles situées *après* la définition `<BOUCLEn...>`. Si vous deviez vraiment installer une boucle dans la partie optionnelle *avant*, il fallait passer par une commande `<INCLURE()>`.

Des critères d'environnement en cascade

Chaque boucle effectue la sélection des éléments tirés de la base de données en fonction de critères. Certains de ces critères correspondent à l'environnement dans lequel se trouve la boucle.

Par exemple : si on prévoit une boucle du genre « Afficher les articles inclus dans cette rubrique », il faut savoir de quelle rubrique il s'agit. C'est ce que l'on nomme l'environnement.

- **L'environnement fourni par l'URL**

Lorsque l'on visite une page d'un site SPIP, son adresse contient généralement une variable. Par exemple : `spip.php?rubrique15` (avant SPIP 1.9, l'url s'écrit : `rubrique.php3?id_rubrique=15`)

Cette variable définit donc un premier environnement : la boucle « Afficher les articles inclus dans cette rubrique » doit alors être comprise comme « Afficher les articles de la rubrique 15 ».

Clairement, avec le même code de squelette, si on appelle l'adresse : `spip.php?rubrique7` (avant SPIP 1.9, l'url s'écrit : `rubrique.php3?id_rubrique=7`) ; l'interprétation de cette boucle deviendra « Afficher les articles de la rubrique 7 ».

- **L'environnement fourni par les autres boucles**

À l'intérieur d'une boucle, l'environnement est modifié par chaque élément de la boucle. En plaçant des boucles les unes à l'intérieur des autres, on hérite ainsi d'environnements imbriqués les uns dans les autres.

Ainsi, dans la structure suivante :

```
<BOUCLE_articles: afficher les articles de cette rubrique>
 Afficher le titre de l'article
 <BOUCLE_auteurs: afficher les auteurs de cet article>
   Nom de l'auteur
 </BOUCLE_auteurs>
```

```
</BOUCLE_articles>
```

On doit comprendre que :
- la première boucle (`BOUCLE_articles`) affiche les articles en fonction de la rubrique, selon l'environnement fournit par l'URL (`id_rubrique=15` par exemple) ;
- dans cette boucle, on obtient un ou plusieurs articles ;
- « à l'intérieur » de chacun de ces articles, on a un environnement différent (celui de l'article, c'est-à-dire, par exemple, `id_article=199`) ;
- la seconde boucle (`BOUCLE_auteurs`), qui est installée *à l'intérieur* de la première boucle, dépend pour chacune de ses exécutions successives (elle est exécutée pour chaque article de la première boucle) : « afficher les auteurs de cet article » devient successivement « afficher les auteurs du premier article », « du deuxième article » et ainsi de suite.

On voit que, par l'imbrication de boucles successives, on obtient différentes boucles, incluses les unes dans les autres, qui dépendent du résultat des boucles dans lesquelles elles sont situées. Et finalement, la toute première boucle (celle qui contient toutes les autres) dépend d'un paramètre fixé dans l'adresse de la page.

Boucles incluses et boucles successives

Si l'on peut inclure des boucles les unes à l'intérieur des autres (chaque boucle incluse dépendant alors du résultat de la boucle à l'intérieur de laquelle elle est installée), on peut tout aussi bien installer des boucles les unes à la suite des autres ; des boucles successives n'influent pas les unes sur les autres.

Par exemple, la page d'une rubrique est typiquement constituée des éléments suivants :

```
<BOUCLE_rubrique(RUBRIQUES){id_rubrique}>
 <ul>Titre de la rubrique
 <BOUCLE_articles(ARTICLES){id_rubrique}>
   <li> Titre de l'article</li>
 </BOUCLE_articles>
 <BOUCLE_sous_rubriques(RUBRIQUES){id_rubrique}>
   <li> Titre de la sous-rubrique </li>
 </BOUCLE_sous_rubriques>
 </ul>
</BOUCLE_rubrique>
 <ul>Il n'y a pas de rubrique à cette adresse.</ul>
<//B_rubrique>
```

La première boucle (`BOUCLE_rubrique`) dépend de la variable passée dans l'URL de la page (`id_rubrique=15` par exemple).

Les boucles suivantes (`BOUCLE_articles` et `BOUCLE_sous_rubriques`) sont installées *à l'intérieur* de la première boucle. Ainsi, s'il n'existe pas de rubrique 15, la première boucle ne donne aucun résultat (le code alternatif « Il n'y a pas de rubrique... » est affiché), et donc les deux boucles incluses sont totalement ignorées. Mais s'il existe une rubrique 15, ces deux sous-boucles seront analysées.

On constate également que ces deux boucles se présentent *l'une après l'autre*. Ainsi, elles fonctionnent en fonction de la première boucle, mais indépendamment l'une de l'autre. S'il n'y a pas d'articles dans la rubrique 15 (`BOUCLE_articles`), on affichera tout de même la liste des sous-rubriques de la rubrique 15 (`BOUCLE_sous_rubriques`) ; et inversement.

Compteurs

Deux balises permettent de compter les résultats dans les boucles.

- **#TOTAL_BOUCLE** retourne le nombre total de résultats affichés par la boucle. On peut l'utiliser dans la boucle, dans ses parties *optionnelles* — avant et après — ou même dans la partie *alternative* après la boucle.
Par exemple, pour afficher le nombre de documents associés à un article :

```
<BOUCLE_art(ARTICLES){id_article}>
  <BOUCLE_doc(DOCUMENTS) {id_article}></BOUCLE_doc>
   [il y a (#TOTAL_BOUCLE) document(s).]
  <//B_doc>
</BOUCLE_art>
```

> Attention : si la partie centrale de la boucle ne retourne rien (c'est le cas avec la boucle `<BOUCLE_doc>` ci-dessus, qui ne sert qu'à compter le nombre de résultats), le `#TOTAL_BOUCLE` ne pourra être affiché que dans la partie alternative *après* de la boucle (`<//B_doc>`).

- **#COMPTEUR_BOUCLE** retourne le numéro de l'itération actuelle de la boucle. On peut par exemple l'utiliser pour numéroter des résultats :

```
<BOUCLE_art(ARTICLES) {par date} {inverse} {0,10}>
#COMPTEUR_BOUCLE - #TITRE<br>
</BOUCLE_art>
```

La syntaxe des balises SPIP

Mai 2001 — maj : 6 septembre

Chaque type de boucle permet de sélectionner des éléments de la base de données de SPIP : des articles, des rubriques, des brèves, etc. Chacun de ces éléments est lui-même constitué d'éléments précis : un titre, une date, un texte, etc. À l'intérieur d'une boucle, il faut donc pouvoir indiquer à quel endroit du code HTML on place tel ou tel de ces éléments précis.

Pour cela, on va utiliser des balises SPIP.

Fonctionnement simplifié

Une balise SPIP se place à l'intérieur d'une boucle (puisqu'il faut savoir si l'on veut récupérer un élément d'un article, d'une rubrique, etc.). Le nom de ces balises est généralement simple, et nous fournirons, pour chaque type de boucle, la liste complète des balises que l'on peut utiliser.

Une balise est toujours précédée du signe dièse (#).

Par exemple, affichons une liste de noms d'articles :

```
<BOUCLE_articles(ARTICLES){id_rubrique}>
#TITRE<br />
```

</BOUCLE_articles>

Lorsque la boucle sera exécutée, la balise SPIP `#TITRE` sera à chaque fois remplacée par le titre de l'article en question :

```
Titre de l'article 1<br />
Titre de l'article 2<br />
...
Titre du dernier article<br />
```

Rien de bien compliqué : on se contente d'indiquer à l'intérieur du code HTML le nom de l'élément désiré, et celui-ci est remplacé par le contenu tiré de la base de données.

Codes optionnels

Dans la pratique, un élément de contenu est souvent accompagné de code HTML *qui ne doit s'afficher que si cet élément existe*, faute de quoi la mise en page devient imprécise.

Par exemple : il existe une balise SPIP pour indiquer le surtitre d'un article. Or de nombreux articles n'ont pas de surtitre.

Complétons l'exemple précédent :

<BOUCLE_articles(ARTICLES){id_rubrique}>

#SURTITRE

#TITRE

</BOUCLE_articles>

qui, classiquement, nous donne une liste d'articles, avec désormais l'indication du titre et du surtitre de chaque article. Mais que se passe-t-il si l'article n'a pas de surtitre ? On obtient le code : « `<br />` », c'est-à-dire une ligne blanche (un retour chariot).

Ce que nous devons faire : n'afficher le code « `<br />` » que si un surtitre existe pour l'article.

La syntaxe de la balise SPIP devient alors :

[`texte optionnel avant` **(#BALISE)** `texte optionnel après` **]**

La balise qui détermine l'option est placée entre parenthèses, et l'ensemble du texte conditionnel entre crochets. Le *texte optionnel avant* et le *texte optionnel après* ne s'affichent que s'il existe, dans la base de données, un élément correspondant à cette balise.

Notre exemple devient :

<BOUCLE_articles(ARTICLES){id_rubrique}>

[(#SURTITRE)
]

#TITRE

</BOUCLE_articles>

On obtient alors le résultat recherché : s'il existe un surtitre pour cet article, il est affiché et suivi du `<br />` ; s'il n'existe pas de surtitre, même le `<br />` est occulté.

Utilisations avancées

A partir de [SPIP 1.8] on peut imbriquer des balises étendues les unes dans les autres. Ainsi, si dans notre exemple on voulait n'afficher le logo de l'article que si le surtitre est défini, on pourrait

écrire :

<BOUCLE_articles(ARTICLES){id_rubrique}>

[[(#LOGO_ARTICLE)
](#SURTITRE)
]

</BOUCLE_articles>

> *Note:On ne peut jamais mettre une boucle dans le code optionnel d'une balise. Mais si on veut faire n'afficher une boucle qu'en fonction d'une certaine balise, on peut utiliser* `<INCLURE{...}>` *à l'intérieur d'un code optionnel.*

Balises non ambiguës

Quand on imbrique des boucles les unes dans les autres, il peut arriver que deux boucles aient des balises homonymes.

Par exemple, dans le code suivant :

<BOUCLE_rubriques(RUBRIQUES){id_rubrique}>

  <BOUCLE_articles(ARTICLES){id_rubrique}>

    #TITRE

  </BOUCLE_articles>

</BOUCLE_rubriques>

la balise **#TITRE** désigne le titre d'un article. Ainsi, si on voulait afficher le titre de la rubrique à l'intérieur de la boucle **_articles**, on ne pourrait pas utiliser `#TITRE`.

Depuis [SPIP 1.8], on peut appeler une balise homonyme de l'une des boucles englobantes en explicitant le nom de la boucle à laquelle la balise appartient. Il faut alors spécifier le nom de la boucle entre le # et le nom de la balise.

On écrira alors la balise `#BALISE` de la boucle `_boucle` [1] de la façon suivante : **#_*boucle*:BALISE**. Par exemple :

<BOUCLE_rubriques(RUBRIQUES){id_rubrique}>

  <BOUCLE_articles(ARTICLES){id_rubrique}>

    #_rubriques:TITRE > #TITRE

  </BOUCLE_articles>

</BOUCLE_rubriques>

affichera le titre de la rubrique, puis le titre de l'article : la balise `#TITRE` pour la boucle *_rubriques* devient `#_rubriques:TITRE` pour ne pas être confondue avec la balise `#TITRE` de la boucle *_articles*.

Filtrer les résultats

Il est fréquent de vouloir modifier un élément tiré de la base de données, soit pour obtenir un affichage différent (par exemple, afficher le titre entièrement en majuscules), ou pour récupérer une valeur découlant de cet élément (par exemple, afficher le jour de la semaine correspondant à une date).

Dans SPIP, on peut directement appliquer des *filtres* aux éléments récupérés de la base de données,

en les indiquant dans la syntaxe des balises SPIP, qui devient :

```
[ option avant (#BALISE|filtre1|filtre2|...|filtren) option après ]
```

La syntaxe est donc de faire suivre le nom de la balise, entre les parenthèses, par les filtres succesifs, séparés par une barre verticale (nommée habituellement *pipe*).

Voici quelques filtres fournis par SPIP :

- *majuscules*, passe le texte en majuscules (plus puissant que la fonction de PHP correspondante, qui ne fonctionne pas correctement avec les caractères accentués) ; par exemple :

```
[(#TITRE|majuscules)]
```

- *justifier*, affiche le texte en justification totale (c'est-à-dire `<P align=justify>`) ; par exemple :

```
[(#TEXTE|justifier)]
```

La présente documentation consacre un article aux différents filtres livrés avec SPIP.

Court-circuiter le traitement par SPIP

SPIP applique un traitement typographique à tous les textes tirés de la base de données. En particulier, il place des espaces insécables avant certains symboles (point-virgule, point d'interrogation, etc.), et analyse des raccourcis de mise en page.

Dans certains cas, vous pouvez avoir besoin de court-circuiter ce traitement, afin de récupérer directement le texte brut tel qu'il est placé dans la base de données. Pour cela, il suffit d'ajouter une astérisque (*) à la suite de la balise SPIP. Ce qui donne :

```
[ option avant (#BALISE*|filtre1|filtre2|...|filtren) option après ]
```

Les paramètres des balises

Depuis [SPIP 1.8], certaines balises [2] acceptent des paramètres. On passera alors une liste de paramètres entre accolade «{» et «}» avec des virgules « , » pour séparer chaque paramètre. Par exemple : `#ENV{lang,fr}`.

Un paramètre peut être une constante ou une autre balise. Seulement les balises de forme simple peuvent être passées en paramètres (i.e. pas de code optionnel ou de filtres). On peut mettre les paramètres entre guillemets simples « '...' » si l'on ne veut pas qu'ils soient interprétés par SPIP.

Notes

[1] *Précision* : n'oubliez pas le cas échéant, l'underscore "_" initial dans le nom de la boucle si celui ci ne commence pas par un numéro.

La boucle ARTICLES

Mai 2001 — maj : 21 mars

Une boucle d'articles se code en plaçant entre parenthèses `ARTICLES` (avec un « s ») :

```
<BOUCLEn(ARTICLES){critères...}>
```

Les éléments contenus dans une telle boucle sont des articles.

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- `{tout}` : les articles sont sélectionnés dans l'intégralité du site (dans toutes les rubriques). Utile notamment pour afficher les articles les plus récents (dans l'intégralité du site) sur la page d'accueil. [En réalité, le critère « tout » n'est pas traité de manière informatique : c'est un aide-mémoire pour le webmestre ; on obtient le même résultat en n'indiquant aucun des critères suivants.]

- `{id_article}` sélectionne l'article dont l'identifiant est `id_article`. Comme l'identifiant de chaque article est unique, ce critère ne retourne qu'une ou zéro réponse.

- `{id_rubrique}` sélectionne les articles contenus dans la rubrique dont l'identifiant est `id_rubrique`.

- `{id_secteur}` sélectionne les articles dans ce secteur (un secteur est une rubrique qui ne dépend d'aucune autre rubrique, c'est-à-dire située à la racine du site).

- `{branche}` (depuis SPIP 1.4) sélectionne l'ensemble des articles de la rubrique ET de ses sous-rubriques. (C'est une sorte d'extension du critère `{id_secteur}`. Toutefois, à l'inverse de `{id_secteur=2}`, il n'est pas possible d'appeler directement une *branche* en faisant par exemple `{branche=2}` : techniquement parlant, il faut que la rubrique en question figure dans le contexte courant. Ce critère est à utiliser avec parcimonie : si votre site est bien structuré, vous ne devriez pas en avoir besoin, sauf dans des cas très particuliers.)

- `{id_auteur}` sélectionne les articles correspondant à cet identifiant d'auteur (utile pour indiquer la liste des articles écrits par un auteur).

- `{id_mot}` sélectionne les articles correspondant à cet identifiant de mot-clé (utile pour indiquer la liste des articles traitant d'un sujet donné).

- `{titre_mot=xxxx}`, ou `{type_mot=yyyy}` (depuis SPIP 1.3) sélectionne respectivement les articles liés au mot-clé dont le nom est « xxxx », ou liés à des mots-clés du groupe de mots-clés « yyyy ». Si l'on donne plusieurs plusieurs critères `{titre_mot=xxxx}` (ou plusieurs `{type_mot=yyyy}`), on sélectionnera ceux qui auront tous ces mots à la fois (nouveauté de SPIP 1.9).

- `{id_groupe=zzzz}` (depuis SPIP 1.4) permet de sélectionner les articles liés à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ». [Nota : Ce critère n'est pas (en l'état actuel du développement de SPIP) cumulable avec le précédent `{type_mot=yyyy}`]

- `{lang}` (depuis SPIP 1.7.1) sélectionne les articles de la langue demandée dans l'adresse de la page.

- `{traduction}` (depuis SPIP 1.7.1) sélectionne les traductions de l'article courant en différentes langues.

- `{origine_traduction}` (depuis SPIP 1.7.1) sélectionne l'article de référence dont l'article courant est une traduction.

- Les critères `{date}` (ou `{date=...}` ou `{date==...}`) permettent de sélectionner un article en fonction de la date passée dans l'URL (depuis SPIP 1.7.2).

- `{recherche}` sélectionne les articles correspondant aux mots indiqués dans l'interface de

recherche (moteur de recherche incorporé à SPIP). Voir la page consacrée au moteur de recherche.

Le statut de l'article

Comme toutes les boucles de SPIP, une boucle `ARTICLES` ne retourne que des articles *publiés* ; dans le cas où le site est réglé de manière à ne pas publier les articles « post-datés », un autre test est fait sur la date de l'article. Jusqu'à SPIP 1.8.2 il n'existait aucun moyen de débrayer ce système et d'afficher les articles « en cours de rédaction », « proposés à la publication » ou « refusés ». C'est désormais possible grâce au critère `{statut}` :

- `{statut=prop|prepa|publie|refuse|poubelle}` (depuis SPIP 1.8.2) sélectionne les articles en fonction de leur statut de publication :
- `{statut=prepa}` sélectionne les articles en cours de rédaction dans l'espace privé ;
- `{statut=prop}` sélectionne les articles proposés à la publication ;
- `{statut=publie}` sélectionne les articles publiés sur le site, y compris les articles « post-datés » ;
- `{statut=refuse}` sélectionne les articles qui ont été refusés à la publication ;
- `{statut=poubelle}` sélectionne les articles qui ont été mis à la poubelle.

Les critères d'affichage

Une fois fixé l'un des critères ci-dessus, on pourra ajouter les critères suivants pour restreindre le nombre d'éléments affichés.

Les critères communs à toutes les boucles s'appliquent évidemment.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (par exemple : `{par date}` ou `{par titre}`).

- `#ID_ARTICLE` affiche l'identifiant unique de l'article. Utile pour fabriquer des liens hypertextes non prévus (par exemple vers une page « Afficher au format impression »).

- `#SURTITRE` affiche le surtitre.

- `#TITRE` affiche le titre de l'article.

- `#SOUSTITRE` affiche le soustitre.

- `#DESCRIPTIF` affiche le descriptif.

- `#CHAPO` affiche le texte d'introduction (chapeau).

- `#TEXTE` affiche le texte principal de l'article.

- `#PS` affiche le post-scriptum.

- Les balises de dates : `#DATE, #DATE_REDAC, #DATE_MODIF` sont explicitées dans la documentation sur « La gestion des dates ».

- `#ID_RUBRIQUE` affiche l'identifiant de la rubrique dont dépend l'article.

- `#ID_SECTEUR` affiche l'identifiant du secteur dont dépend l'article (le secteur étant la rubrique

parente située à la racine du site).

- `#NOM_SITE` et `#URL_SITE` affichent le nom et l'url du « lien hypertexte » de l'article (si vous avez activé cette option).

- `#VISITES` affiche le nombre total de visites sur cet article.

- `#POPULARITE` affiche le pourcentage de popularité de cet article ; voir la documentation « La « popularité » des articles ».

- `#LANG` affiche la langue de cet article.

Les balises calculées par SPIP

Les éléments suivants sont calculés par SPIP (Ils ne peuvent pas être utilisés comme critère de classement).

- `#URL_ARTICLE` affiche l'URL de la page de l'article.

- `#NOTES` affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- `#INTRODUCTION` (depuis SPIP 1.4) affiche le descriptif de l'article, sinon affiche les 600 premiers caractères du début de l'article (chapeau puis texte). Dans les versions antérieures à SPIP 1.3, ce sont systématiquement les premiers caractères de l'article (chapeau puis texte) qui sont pris en compte (le descriptif n'est pas utilisé).

- `#LESAUTEURS` affiche les auteurs de cet article, avec lien vers leur propre page publique (afin de pouvoir directement leur écrire ou de consulter la liste des articles qu'ils ont publié). Cela évite de créer une boucle AUTEURS pour obtenir le même résultat. Dans les versions antérieures à SPIP 1.9, cette balise affiche les auteurs de l'article avec lien vers leur adresse e-mail.

- `#PETITION` affiche le texte de la pétition si elle existe. Si elle existe mais que le texte est vide, retourne un espace (une chaîne non vide sans incidence dans une page html).

- `#FORMULAIRE_SIGNATURE` fabrique et affiche le formulaire permettant de signer la pétition associée à cet article.

- `#FORMULAIRE_FORUM` fabrique et affiche le formulaire permettant de poster un message répondant à cet article. Pour en savoir plus, voir aussi « Les formulaires ».

- `#PARAMETRES_FORUM` fabrique et affiche la liste des variables exploitées par le formulaire permettant de répondre à cet article. Par exemple :

[Répondre à cet article]

Depuis SPIP 1.8.2 on peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à cet article</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

> *Historique :* Dans les versions antérieures à SPIP 1.9 il aurait fallu écrire `forum.php3?` et non `spip.php?page=forum&`
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme `http://monsite.net/xxx.php3` et non pas `http://monsite.net/spip.php?page=xxx`.

Les logos

- `#LOGO_ARTICLE` affiche le logo de l'article, éventuellement avec la gestion du survol.

- `#LOGO_RUBRIQUE` affiche le logo de la rubrique de l'article.

- `#LOGO_ARTICLE_RUBRIQUE` affiche le logo de l'article, éventuellement remplacé par le logo de la rubrique s'il n'existe pas de logo spécifique à l'article.

Les logos s'installent de la manière suivante : `[(#LOGO_ARTICLE|alignement|adresse)]`

L'alignement peut être `left` ou `right`. L'adresse est l'URL de destination du lien de ce logo (par exemple `#URL_ARTICLE`). Si l'on n'indique pas d'adresse, le bouton n'est pas cliquable.

Si l'on veut récupérer directement le nom du fichier du logo (alors que les balises précédentes fabriquent le code HTML complet pour insérer l'image dans la page), par exemple pour afficher une image en fond de tableau, on utilisera le filtre `|fichier` comme suit : `[(#LOGO_ARTICLE|fichier)]`

Par ailleurs deux balises permettent de récupérer un seul des deux logos :
- `#LOGO_ARTICLE_NORMAL` affiche le logo sans survol ;
- `#LOGO_ARTICLE_SURVOL` affiche le logo de survol.

La boucle RUBRIQUES

Mai 2001 — maj : 2 mai

La boucle `RUBRIQUES` retourne une liste de... rubriques (étonnant, non ?)

```
<BOUCLEn(RUBRIQUES){critères...}>
```

> *Remarque.* Une boucle `RUBRIQUES` n'affiche que des rubriques « actives », c'est-à-dire contenant des articles publiés, des documents joints (à partir de SPIP 1.4), des sites publiés - ou des sous-rubriques elles-mêmes actives. De cette façon, on évite de se trouver dans des rubriques « culs de sac » n'offrant aucun élément de navigation. À partir de la version SPIP 1.7.1, il est possible de forcer l'affichage des rubriques vides (voir ci-dessous, le critère `{tout}`).

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- `{id_rubrique}` sélectionne la rubrique dont l'identifiant est `id_rubrique`. Comme l'identifiant de chaque rubrique est unique, ce critère retourne une ou zéro réponse.

- `{id_secteur}` sélectionne les rubriques de ce secteur. (On peut également, par extension, utiliser le critère `{branche}` décrit dans La boucle ARTICLES).

- `{id_parent}` sélectionne la liste des rubriques contenues dans une rubrique.

- `{racine}` sélectionne la liste des secteurs (rigoureusement identique à `{id_parent=0}`).

- `{id_enfant}` sélectionne la rubrique qui contient la rubrique (une seule réponse ; ou zéro réponse si la présente rubrique est située à la racine du site).

- `{meme_parent}` sélectionne la liste des rubriques dépendant de la même rubrique que la rubrique en cours. Permet d'afficher les rubriques « sœurs » qui se trouvent au même niveau dans la hiérarchie.

- À partir de la version SPIP 1.4 , les rubriques peuvent être liées à des mots-clés. Les critères de mots-clés peuvent donc être désormais utilisés dans les boucles `(RUBRIQUES)` :

  - `{id_mot}`, `{titre_mot=xxx}` récupèrent les rubriques liées au mot dont le numéro est *id_mot* ou dont le titre est *titre_mot* ;
  - `{id_groupe}`, `{type_mot=yyyy}` récupèrent les rubriques liées à des mots du groupe *id_groupe*, ou du groupe dont le titre est *type_mot*.

- `{recherche}` sélectionne les rubriques correspondant aux mots indiqués dans l'interface de recherche (moteur de recherche incorporé à SPIP). Voir la page consacrée au moteur de recherche.

- `{lang}` (depuis SPIP 1.7.1) sélectionne les rubriques de la langue demandée dans l'adresse de la page.

- `{tout}` (depuis SPIP 1.7.1) sélectionne *toutes* les rubriques, c'est-à-dire : les rubriques vides *en plus* des rubriques contenant des éléments publiés.
On réservera ce choix à des besoins très spécifiques ; en effet, par défaut, SPIP n'affiche pas sur le site public les rubriques qui ne contiennent aucun élément actif, afin de garantir que le site ne propose pas de « culs de sac » (navigation vers des pages ne proposant aucun contenu).

Les critères d'affichage

Une fois fixé l'un des critères ci-dessus, on pourra ajouter les critères suivants pour restreindre le nombre d'éléments affichés.

- Les critères communs à toutes les boucles s'appliquent évidemment.

- `{exclus}` permet d'exclure du résultat la rubrique dans laquelle on se trouve déjà (utile avec `meme_parent`).

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- `#ID_RUBRIQUE` affiche l'identifiant unique de la rubrique.

- `#TITRE` affiche le titre de la rubrique.

- `#DESCRIPTIF` affiche le descriptif.

- `#TEXTE` affiche le texte principal de la rubrique.

- `#ID_SECTEUR` affiche l'identifiant du secteur dont dépend la rubrique (le secteur étant la rubrique située à la racine du site).

- `#ID_PARENT` affiche l'identifiant de la rubrique qui contient la rubrique actuelle. Si la rubrique est située à la racine du site, retourne 0.

- `#LANG` affiche la langue de cette rubrique.

Les balises calculées par SPIP

Les éléments suivants sont calculés par SPIP. (Ils ne peuvent pas être utilisés comme critère de classement.)

- `#NOTES` affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- `#INTRODUCTION` affiche les 600 premiers caractères du texte, les enrichissements typographiques (gras, italique) sont supprimés.

- `#URL_RUBRIQUE` affiche l'URL de la page de la rubrique.

- `#DATE` (depuis SPIP 1.4) affiche la date de la dernière publication effectuée dans la rubrique et/ou ses sous-rubriques (articles, brèves...).

- `#FORMULAIRE_FORUM` fabrique et affiche le formulaire permettant de poster un message répondant à cette rubrique. Pour en savoir plus, voir aussi « Les formulaires ».

- `#PARAMETRES_FORUM` fabrique la liste des variables exploitées par l'interface du formulaire permettant de répondre à cette rubrique. Par exemple :

[Répondre à cette rubrique]

Depuis SPIP 1.8.2 on peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à cette rubrique</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

> *Historique :* Dans les versions antérieures à SPIP 1.9 il aurait fallu écrire `forum.php3?` et non `spip.php?page=forum&`
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme `http://monsite.net/xxx.php3` et non pas `http://monsite.net/spip.php?page=xxx`.

- `#FORMULAIRE_SITE` (depuis SPIP 1.4) fabrique et affiche un formulaire permettant aux visiteurs du site de proposer des référencements de sites. Ces sites apparaîtront comme « proposés » dans l'espace privé, en attendant une validation par les administrateurs.

Ce formulaire ne s'affiche que si vous avez activé l'option « Gérer un annuaire de sites » dans la *Configuration sur site* dans l'espace privé, et si vous avez réglé « Qui peut proposer des sites référencés » sur « les visiteurs du site public ».

Le logo

- `#LOGO_RUBRIQUE` le logo de la rubrique, éventuellement avec la gestion du survol. S'il n'y a pas de logo pour cette rubrique, SPIP va automatiquement chercher s'il existe un logo pour la rubrique dont elle dépend, et ainsi de suite de manière récursive.

Le logo s'installe de la manière suivante :

`[(#LOGO_RUBRIQUE|`***alignement***`|`***adresse***`)]`

Où :

- *alignement* est l'alignement du logo (left ou right)
- *adresse* est une url si on veut ajouter un lien directement sur le logo (par exemple `#URL_RUBRIQUE`).

Depuis SPIP 1.4 : `#LOGO_RUBRIQUE_NORMAL` affiche le logo « sans survol » ; `#LOGO_RUBRIQUE_SURVOL` affiche le logo de survol : ces deux balises permettent par exemple, quand on est dans une rubrique, de gérer un logo « avec survol » pour les liens vers les autres rubriques, et de laisser le logo de survol seul dans la rubrique active.

La boucle BREVES

Mai 2001 — maj : 9 juin

La boucle `BREVES`, comme son nom l'indique, retourne une liste de brèves.

`<BOUCLEn(BREVES){critères...}>`

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- `{tout}` : les brèves sont sélectionnées dans l'intégralité du site.

- `{id_breve}` sélectionne la brève dont l'identifiant est `id_breve`. Comme l'identifiant de chaque brève est unique, ce critère retourne une ou zéro réponse.

- `{id_rubrique}` sélectionne toutes les brèves contenues dans la rubrique en cours.

- `{id_mot}` (depuis SPIP 1.2) sélectionne toutes les brèves liées au mot-clé en cours (à l'intérieur d'une boucle de type MOTS).

- `{titre_mot=xxxx}`, ou `{type_mot=yyyy}` (depuis SPIP 1.3) sélectionne les brèves liées au mot-clé dont le nom est « xxxx », ou liées à des mots-clés du groupe de mots-clés « yyyy ». Attention, on ne peut pas utiliser plusieurs critères `{titre_mot=xxxx}` ou `{type_mot=yyyy}` dans une même boucle.

- `{id_groupe=zzzz}` (depuis SPIP 1.4) permet de sélectionner les brèves liées à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ».

- `{lang}` sélectionne les brèves de la langue demandée dans l'adresse de la page.

- `{recherche}` sélectionne les brèves correspondant aux mots indiqués dans l'interface de recherche (moteur de recherche incorporé à SPIP). Voir la page consacrée au moteur de recherche.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- `#ID_BREVE` affiche l'identifiant unique de la brève.

- `#TITRE` affiche le titre de la brève.

- `#DATE` affiche la date de publication de la brève.

- `#TEXTE` affiche le texte de la brève.

- `#NOM_SITE` affiche le nom du site indiqué en références.

- `#URL_SITE` affiche l'adresse (URL) du site indiqué en références.

- `#ID_RUBRIQUE` affiche l'identifiant de la rubrique dont dépend cette brève.

- `#LANG` affiche la langue de cette brève. Par défaut, la langue d'une brève est la langue du secteur dans lequel elle se trouve.

Les balises calculées par SPIP

Les éléments suivants sont calculés par SPIP (Ils ne peuvent pas être utilisés comme critère de classement).

- `#NOTES` affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- `#INTRODUCTION` affiche les 600 premiers caractères du texte, les enrichissements typographiques (gras, italique) sont supprimés.

- `#URL_BREVE` affiche l'URL de la page de la brève.

- `#FORMULAIRE_FORUM` fabrique et affiche le formulaire permettant de poster un message répondant à cette brève. Pour en savoir plus, voir aussi « Les formulaires ».

- `#PARAMETRES_FORUM` fabrique la liste des variables exploitées par l'interface du formulaire permettant de répondre à cette brève.

[Répondre à cette bréve]

Depuis SPIP 1.8.2 on peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à cette bréve</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

> *Historique :* Dans les versions antérieures à SPIP 1.9 il aurait fallu écrire `forum.php3?` et non `spip.php?page=forum&`
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme `http://monsite.net/xxx.php3` et non pas `http://monsite.net/spip.php?page=xxx`.

Le logo

- `#LOGO_BREVE` affiche le logo de la brève, éventuellement avec la gestion du survol.

Le logo s'installe de la manière suivante : `[(#LOGO_BREVE|alignement|adresse)]`

- `#LOGO_BREVE_RUBRIQUE` (depuis SPIP 1.4) affiche, si il existe, le logo de la brève ; si ce logo n'a pas été attribué, SPIP affiche le logo de la rubrique.

La boucle AUTEURS

Mai 2001 — maj : 13 août

La boucle `AUTEURS`, comme son nom l'indique, retourne une liste d'auteurs.

`<BOUCLEn(AUTEURS){critères...}>`

Si l'on ne précise pas de critère de sélection, la boucle retournera tous les auteurs *ayant un article publié*.

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- `{tout}` : les auteurs sont sélectionnés, qu'ils aient écrit un article ou non.

- `{id_auteur}` retourne l'auteur dont l'identifiant est `id_auteur`. Comme l'identifiant de chaque auteur est unique, ce critère retourne une ou zéro réponse.

- `{id_article}` retourne tous les auteurs de cet article.

- `{lang}` sélectionne les auteurs qui ont choisi, dans l'espace privé, la langue demandée dans l'adresse de la page. Si un auteur ne s'est jamais connecté dans l'espace privé, il ne sera pas trouvé par ce critère.

- `{lang_select}` Par défaut, une boucle AUTEURS affiche les balises et les chaînes localisées dans la langue du contexte (de la boucle englobante ou de l'url). Si on utilise ce critère, ces informations seront localisées dans la langue choisie par l'auteur.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par nom}`).

- `#ID_AUTEUR` affiche l'identifiant unique de l'auteur.

- `#NOM` affiche le nom de l'auteur.

- `#BIO` affiche la biographie de l'auteur.

- `#EMAIL` affiche son adresse email.

- `#NOM_SITE` affiche le nom de son site Web.

- `#URL_SITE` affiche l'adresse (URL) de son site.

- `#PGP` affiche sa clé publique pour PGP.

- `#LANG` affiche la langue de l'auteur (c'est-à-dire celle qu'il a choisie dans l'espace privé).

Les balises calculées par SPIP

- `#FORMULAIRE_ECRIRE_AUTEUR` (depuis SPIP 1.4) fabrique et affiche un formulaire permettant d'écrire à l'auteur. Il faut que le serveur hébergeant le site accepte d'envoyer des mails. Ce système permet de ne pas divulguer l'adresse email de l'auteur.

- `#NOTES` affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- `#URL_AUTEUR` affiche l'adresse de la page `spip.php?auteurxxx`.

Le logo

- `#LOGO_AUTEUR` le logo de l'auteur, éventuellement avec la gestion du survol.

Le logo s'installe de la manière suivante : `[(#LOGO_AUTEUR|alignement|adresse)]`

Les variantes `#LOGO_AUTEUR_NORMAL` et `#LOGO_AUTEUR_SURVOL` (depuis SPIP 1.6) permettent un affichage plus fin de ces deux variantes du logo.

La boucle FORUMS

Mai 2001 — maj : Décembre 2007

La boucle `FORUMS` retourne une liste de messages de forums.

`<BOUCLEn(FORUMS){critères...}>`

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{id_forum}`** retourne le message dont l'identifiant est `id_forum`. Comme l'identifiant de chaque message est unique, ce critère retourne une ou zéro réponse.

- **`{id_article}`** retourne les messages correspondant à cet article.

- **`{id_rubrique}`** retourne les messages correspondant à cette rubrique. Attention, il ne s'agit pas de messages des articles de cette rubrique, mais bien des messages de cette rubrique. En effet, il est possible d'activer dans l'espace privé des forums pour chaque rubrique.

- **`{id_breve}`** retourne les messages correspondant à cette brève.

- **`{id_syndic}`** retourne les messages correspondant à ce site.

- **`{id_thread}`** introduit dans [SPIP 1.8], retourne les messages appartenant à ce fil de discussion.
Note : `id_thread` n'est rien d'autre que l'identifiant `id_forum` du message qui démarre le fil de discussion (aussi appelé « pied » de la discussion).

- **`{id_parent}`** retourne les messages dépendant d'un autre message. Indispensable pour gérer des fils de discussion (« *threads* ») dans les forums.

- **`{id_enfant}`** retourne le message dont dépend le message actuel (permet de « remonter » dans la hiérachie des fils de discussion). (SPIP 1.3)

- **`{meme_parent}`** retourne les autres messages répondant à un même message. (SPIP 1.3)

- **`{plat}`** ou **`{tout}`** : affiche tous les messages de forum sans prendre en compte leur hiérarchie : avec ce critère, vous pouvez sélectionner tous les messages quelle que soit leur position dans un thread (dans la limite des autres critères, bien sûr). Cela permet par exemple d'afficher les messages par ordre strictement chronologique par exemple, ou de compter le nombre total de contributions dans un forum.

N.B. En l'absence de critère `{id_forum}` ou `{id_parent}`, lorsque `{plat}` n'est pas utilisé, seuls les messages n'ayant pas de parent (i.e. à la racine d'un thread) sont affichés.

- **`{id_secteur}`** retourne les messages correspondant au secteur. A priori, peu utile ; mais cela permet par exemple de faire un grand forum thématique regroupant tous les messages d'un secteur, quel que soit l'endroit où l'on se trouve.

- À partir de la version SPIP 1.4, les messages des forums peuvent être liées à des mots-clés. Les critères de mots-clés peuvent donc être désormais utilisés dans les boucles `(FORUMS)` :

- `{id_mot}`, `{titre_mot=xxx}` récupèrent les messages liés au mot dont le numéro est *id_mot* ou dont le titre est *titre_mot* ;
- `{id_groupe}`, `{type_mot=yyyy}` récupèrent les messages liés à des mots du groupe *id_groupe*, ou du groupe dont le titre est *type_mot*.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

- **Les balises tirées de la base de données**

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- **`#ID_FORUM`** affiche l'identifiant unique du message.

- **`#ID_THREAD`** introduit dans [SPIP 1.8], affiche l'identifiant du fil de discussion auquel appartient ce message. (Il s'agit de l'`id_forum` du *pied* de la discussion.)

- **`#URL_FORUM`** donne, depuis [SPIP 1.8], l'adresse canonique de la page qui affiche le message de forum (par exemple, avec les URLs normales de SPIP, `article.php3?id_article=8#forum15` pour le message 15 associé à l'article 8).

- **`#ID_BREVE`** affiche l'identifiant de la brève à laquelle ce message est attaché. Attention, cela n'est pas récursif : un message qui répond à un message attaché à une brève ne contient pas lui-même le numéro de la brève.

- **`#ID_ARTICLE`** est l'identifiant de l'article auquel répond le message.

- **`#ID_RUBRIQUE`** est l'identifiant de la rubrique à laquelle le message répond.

- **`#ID_SYNDIC`** est l'identifiant du site auquel le message répond.

- **`#DATE`** est la date de publication.

- **`#TITRE`** est le titre.

- **`#TEXTE`** est le texte du message.

- **`#NOM_SITE`** le nom du site Web indiqué par l'auteur.

- **`#URL_SITE`** l'adresse (URL) de ce site Web.

- **`#NOM`** est le nom de l'auteur du message.

- **`#EMAIL`** est l'adresse email de l'auteur, sous forme directe d'envoi (le mailto: est intégré). Elle s'utilise donc ainsi `[(#EMAIL)]` pour avoir à la fois l'adresse mail apparente et, directement, le lien à cliquer sur cette adresse.

- **`#IP`** est l'adresse IP de l'auteur du message au moment de l'envoi de sa contribution.

- **Les balises calculées par SPIP**

- **`#FORMULAIRE_FORUM`** fabrique l'interface permettant de poster un message de réponse. Pour en savoir plus, voir aussi « Les formulaires ».

- **#PARAMETRES_FORUM** fabrique la liste des variables exploitées par l'interface du formulaire permettant de répondre à ce message. Par exemple :

[Répondre à ce message]

Depuis [SPIP 1.8.2] on peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à ce message</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

> *Historique :* Dans les versions antérieures à [SPIP 1.9] il aurait fallu écrire `forum.php3?` et non `spip.php?page=forum&`
>
> De façon générale jusqu'à [SPIP 1.9], les urls des pages générées par spip étaient de la forme http://monsite.net/xxx.php3 et non pas `http://monsite.net/spip.php?page=xxx`.

La boucle MOTS

Mai 2001 — maj : Janvier 2007

La boucle `MOTS` retourne une liste de mots-clés.

`<BOUCLEn(MOTS){critères...}>`

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- `{tout}` les mots sont sélectionnés dans l'intégralité du site.

- `{id_mot}` retourne le mot-clé dont l'identifiant est *`id_mot`*.

- `{id_groupe}` retourne les mots-clés associés au groupe de mots dont le numéro est *id_groupe* SPIP 1.4 .

- `{id_article}` retourne les mots-clés associés à cet article (c'est l'utilisation la plus courante de cette boucle).

- `{id_rubrique}` retourne les mots-clés associés à une rubrique SPIP 1.4 .

- `{id_breve}` retourne les mots associés à une brève SPIP 1.4 .

- `{id_syndic}` retourne les mots associés à un site référencé SPIP 1.4 .

- `{id_forum}` retourne les mots associés à un message de forum SPIP 1.4 (attention, utilisation très spécifique).

- `{titre=france}` retourne le mot-clé intitulé `france` (par exemple).

- `{type=pays}` retourne les mots-clés du groupe de mots-clés intitulé `pays` (par exemple).

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- `#ID_MOT` affiche l'identifiant unique du mot.
- `#TITRE` affiche le titre (le mot-clé lui-même).
- `#DESCRIPTIF` affiche le descriptif du mot.
- `#TEXTE` affiche le texte associé au mot.
- `#TYPE` affiche la catégorie dans laquelle est installé ce mot-clé (par exemple, le mot-clé « France » pourrait être associé à la catégorie « Pays »).
- `#LOGO_MOT` SPIP 1.4 affiche le logo associé au mot-clé.
- `#URL_MOT` affiche l'adresse de ce mot

La boucle (GROUPES_MOTS)

D'une utilisation marginale, la boucle `GROUPES_MOTS`, introduite avec SPIP 1.5, mérite d'être citée ici : elle permet, si vous avez plusieurs groupes de mots-clés, de sélectionner ces groupes, et d'organiser par exemple une page récapitulative de tous les mots-clés classés par groupe, puis par ordre alphabétique à l'intérieur de chaque groupe, par exemple via le code suivant :

```
<BOUCLE_groupes(GROUPES_MOTS){par titre}>
<h1>#TITRE</H1>
<BOUCLE_mots(MOTS){id_groupe}{par titre}{" - "}>
#TITRE
</BOUCLE_mots>
</BOUCLE_groupes>
```

Les balises et critères associés à cette boucle sont :
- `#ID_GROUPE`, l'identifiant du groupe de mots [également disponible dans la boucle(MOTS)] ;
- `#TITRE`, le titre du groupe [à l'intérieur de la boucle(MOTS), vous pouvez utiliser `#TYPE` pour afficher cette valeur].

La boucle DOCUMENTS

Juin 2002 — maj : 15 décembre

Introduite avec SPIP 1.4 , la boucle DOCUMENTS retourne une liste de documents multimédia associés (à un article, à une rubrique, éventuellement les images incluses dans une brève).

`<BOUCLEn(DOCUMENTS){critères...}>`

Cette boucle gère non seulement les documents joints non installés dans le texte d'un article, mais peut aussi accéder aux *images* (depuis la version 1.4, les images sont gérées, au niveau du programme, comme un genre spécifique de documents), aux vignettes de prévisualisation et aux documents déjà insérés dans le corps de l'article.

Pour mémoire, on utilisera donc le plus fréquemment (utilisation courante) la boucle DOCUMENTS avec, au minimum, les critères suivants (explications ci-après) :

<BOUCLEn(DOCUMENTS){mode=document}{doublons}>

Les critères de sélection

Une boucle DOCUMENTS s'utilise en général à l'intérieur d'un article ou d'une rubrique (éventuellement dans une brève, mais ici l'utilisation sera réservée à la récupération d'images, ce qui sera très spécifique).

- `{id_document}` sélectionne le document dont l'identifiant est id_document. Comme l'identifiant de chaque document est unique, ce critère ne retourne qu'une ou zéro réponse.

- `{id_article}` retourne les documents de l'article dont l'identifiant est *id_article*.

- `{id_rubrique}` retourne les documents de la rubrique *id_rubrique*.

- `{id_breve}` retourne les documents de la brève *id_breve* (il n'est pas possible d'associer des documents multimédia à une brève, seulement des images ; l'utilisation d'une boucle DOCUMENTS dans ce cadre sera donc très spécifique).

Notez bien : il n'est pas possible d'utiliser ici le critère `{id_secteur}` ; les documents sont conçus pour être intimement liés aux articles et aux rubriques, et non à être appelés seuls sans ces éléments (on parle dans SPIP de « documents joints »).

Les critères d'affichage

- `{mode=document}` ou `{mode=image}` permet d'indiquer si l'on veut appeler les documents multimédia, ou les images (en effet, désormais les images associées à l'article et éventuellement insérées dans l'article sont traités comme des *documents* en *mode=image*).

N.B. Dans les sites SPIP existant avant la version 1.4, l'habitude a été prise de ne pas pouvoir afficher les images qui ne sont pas insérées *à l'intérieur* du texte de l'article. De fait, si vous ajoutez un boucle DOCUMENTS en *mode=image* sur un site déjà existant, vous risquez de voir réapparaître dans cette boucle des images qui n'étaient pas destinées à être publiées sur le site public. Donc, n'utilisez une telle boucle que sur un site créé avec la version 1.4, ou bien procédez avec beaucoup de précautions (vérifiez les anciens articles pour éviter la publication d'images parasites).

- `{extension=...}` permet de sélectionner les documents selon leur terminaison (terminaison du fichier multimédia, par exemple « mov », « ra », « avi »...). Cela peut être utilisé par exemple pour réaliser un *porfolio*, c'est-à-dire une boucle n'affichant que les documents de type image, une seconde boucle ensuite, avec une présentation graphique différente, les autres types de documents :

<BOUCLE_portfolio(DOCUMENTS){id_article}{extension==jpg|png|gif}{mode=document}{doublons}>

Cette BOUCLE_portfolio récupère les documents joints à un article, non déjà affichés dans le texte de l'article, et donc les extensions des fichiers peuvent être « jpg », « png » ou « gif ».

- `{distant}` permet, depuis SPIP 1.8.2, de sélectionner les documents selon qu'ils soient distant ou non. C'est à dire stockés sur un autre site ou téléchargés dans l'espace web du site. On précisera `{distant=oui}` et `{distant=non}` respectivement. (voire SPIP 1.8.2)

- `{doublons}` prend ici une importance particulière : elle permet non seulement de ne pas réafficher des documents déjà affichés par une autre boucle, mais également de ne pas réafficher les

documents déjà intégrés à l'intérieur d'un article. Si l'on oublie ce critère, on affichera *tous* les documents associés à un article, *y compris* ceux qui auraient déjà été affichés à l'intérieur du texte [1].

Les balises

- `#LOGO_DOCUMENT` affiche le logo (vignette de prévisualisation) associé à cet article ; si une vignette personnalisée n'a pas été installée manuellement par l'auteur de l'article, SPIP utilise une vignette standard selon le type du fichier.

- `#URL_DOCUMENT` est l'URL du fichier multimédia. Pour afficher une vignette cliquable pointant vers le document multimédia, on utilisera donc le code suivant :

[(#LOGO_DOCUMENT|#URL_DOCUMENT)]

- `#TITRE` affiche le titre du document.

- `#DESCRIPTIF` affiche le descriptif du document.

- `#FICHIER` SPIP 1.8.2 affiche le nom de fichier du document, plus précisément son URL reltive. Pour obtenir le nom de fichier seul, on passera ce filtre : `[(#FICHIER|basename)]`.

Une utilisation intéressante de cette balise est combinée avec le filtre `image_reduire`, dans le cadre d'un portfolio pour afficher une réduction de l'image plutôt que de son logo ; par exemple en utilisant

[<a

href="#URL_DOCUMENT">(#FICHIER|image_reduire{500})]

- `#TYPE_DOCUMENT` affiche le type (fichier Quicktime, fichier Real...) du document multimédia.

- `#EXTENSION` SPIP 2.0 comme son nom l'indique, affiche l'extension du format du fichier, par exemple : `pdf`, `jpeg`, `move`, `ra`.

- `#TAILLE` affiche la taille du fichier multimédia. Ce chiffre est fourni en octets. Pour de gros fichiers, cette valeur devient rapidement inutilisable ; on pourra donc lui appliquer le filtre `taille_en_octets`, qui affichera successivement en octets, en kilooctets, ou même en mégaoctets :

[(#TAILLE|taille_en_octets)]

- `#LARGEUR` et `#HAUTEUR` fournissent les dimensions en pixels.

- `#MIME_TYPE` affiche le type MIME du fichier — par exemple : `image/jpeg` —, *cf.* Type de média internet.

- `#DATE` est la date de mise en ligne du document. (Modifiable après l'ajout). Voir « La gestion des dates » pour des détails sur l'utilisation du champ `#DATE`.

- `#ID_DOCUMENT` affiche le numéro du document.

- `#DISTANT` est une balise qui affiche « oui » ou « non » selon que le document est distant (référencé par une url) ou pas.

- `#EMBED_DOCUMENT` est une balise permettant d'incruster le document dans la page produite plutôt que de le référencer. Cette balise est aujourd'hui dépréciée, étant un cas particulier des *modèles* (voir Utiliser les modèles) qui permettent de moduler cette incrustation de manière plus souple et plus efficace.

Cette balise peut être complétée de paramètres propres aux formats utilisés, par exemple :

[(#EMBED_DOCUMENT|autostart=true)]

mais on lui préférera ce en quoi elle se traduit automatiquement à présent :

#MODELE{emb, autostart=true}

Notes

[1] Si on utilise un critère avec un nom (`{doublons unnom}`), celui ci n'exclura pas les documents intégrés dans le texte de l'article.

La boucle SITES (ou SYNDICATION)

Mai 2001 — maj : 6 janvier

La boucle `SITES` retourne une liste de sites référencés.

```
<BOUCLEn(SITES){critères...}>
```

Si l'on a syndiqué des sites référencés, cette boucle s'utilise, naturellement, associée à une boucle `SYNDIC_ARTICLES` qui permet de récupérer la liste des articles de ces sites.

> *Historique :* Avant SPIP 1.3, cette boucle était nommée `SYNDICATION`, car seuls des sites syndiqués pouvaient être référencés.
> `<BOUCLEn(SYNDICATION){critères...}>`
> Les deux dénominations sont rigoureusement équivalentes (mais « SITES » correspond mieux au fait que, depuis la version 1.3 de SPIP, il s'agit d'un système de *référencement* de sites, la syndication étant une option).

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- `{tout}` sélectionne *tous* les sites référencés.

- `{id_syndic}` sélectionne le site référencé dont l'identifiant est *`id_syndic`*.

- `{id_rubrique}` sélectionne les sites référencés dans cette rubrique.

- `{id_secteur}` sélectionne les sites référencés dans ce secteur.

- `{id_mot}` (depuis SPIP 1.3) sélectionne toutes les sites liés au mot-clé indiqué par le contexte (boucle `(MOTS)` englobante, paramètre d'URL etc).

- `{titre_mot=xxxx}`, ou `{type_mot=yyyy}` (depuis SPIP 1.3) sélectionne les sites liés au mot-clé dont le nom est « xxxx », ou liés à des mots-clés du groupe de mots-clés « yyyy ». Si l'on donne plusieurs plusieurs critères `{titre_mot=xxxx}` (ou plusieurs `{type_mot=yyyy}`), on sélectionnera ceux qui auront tous ces mots à la fois (nouveauté de SPIP 1.9).

  – `{id_groupe=zzzz}` (depuis SPIP 1.4) permet de sélectionner les sites liés à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ».

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

- `{syndication=oui}`, ou `{syndication=non}` (depuis SPIP 1.3) permet de n'afficher que les sites référencés faisant l'objet d'une syndication, ou les sites non syndiqués.

- `{moderation=oui}` (depuis SPIP 1.4) affiche les sites syndiqués dont les liens sont bloqués a priori (« modérés ») ; l'inverse de ce critère est `{moderation!=oui}`.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par nom_site}`).

- `#ID_SYNDIC` affiche l'identifiant unique du site référencé. Par exemple pour renvoyer vers la page décrivant le site *(site.html sur la /dist)* avec le code suivant :

```
<BOUCLE_sites(SITES) {id_rubrique} {par nom_site}>
<li><a href="[(#ID_SYNDIC|generer_url_site)]">#NOM_SITE</a>
</BOUCLE_sites>
```

- `#NOM_SITE` affiche le nom du site référencé.

- `#URL_SITE` affiche l'adresse (URL) du site référencé.

- `#DESCRIPTIF` affiche le descriptif du site référencé.

- `#ID_RUBRIQUE` affiche le numéro de la rubrique contenant ce site.

- `#ID_SECTEUR` affiche le numéro de la rubrique-secteur (à la racine du site) contenant ce site.

Autres balises

- `#LOGO_SITE` affiche le logo attribué au site.

- `#URL_SYNDIC` affiche l'adresse (URL) du fichier de syndication de ce site.

- `#FORMULAIRE_FORUM` fabrique et affiche le formulaire permettant de poster un message de forum à propos de ce site. Pour en savoir plus, voir aussi « Les formulaires ».

- `#PARAMETRES_FORUM` fabrique la liste des variables exploitées par l'interface du formulaire permettant de poster un message de forum à propos de ce site. Par exemple : `[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">Répondre à ce forum</a>]`

Depuis [SPIP 1.8.2] on peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à ce message</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

> *Historique :* Dans les versions antérieures à SPIP 1.9 il aurait fallu écrire `forum.php3?` et non `spip.php?page=forum&`
>
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme `http://monsite.net/xxx.php3` et non pas

http://monsite.net/spip.php?page=xxx.

La boucle SYNDIC_ARTICLES

Mai 2001 — maj : Janvier 2007

La boucle SYNDIC_ARTICLES retourne une liste des articles des sites syndiqués.

```
<BOUCLEn(SYNDIC_ARTICLES){critères...}>
```

On peut soit l'utiliser à l'intérieur d'une boucle SITES (cette dernière récupère une liste de sites référencés, ensuite on récupère chaque article de ces sites), soit directement à l'intérieur d'une rubrique (on récupère directement tous les articles syndiqués dans une rubrique, en court-circuitant le passage par la liste des sites).

(SPIP 1.3) À partir de la version 1.3 de SPIP, la boucle SITES (ou SYNDICATION) n'affiche plus uniquement des sites syndiqués, mais plus généralement des sites référencés (la syndication de certains sites référencés étant une option). On pourra donc, pour obtenir une présentation graphique plus précise, utiliser une boucle SYNDIC_ARTICLES uniquement à l'intérieur d'une boucle SITES utilisant le critère {syndication=oui}.

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- {tout}, tous les sites syndiqués.

- {id_syndic_article} retourne l'article syndiqué dont l'identifiant est id_syndic_article. (Dans la pratique, il y très peu d'intérêt à fabriquer une page pour un article syndiqué, puisqu'on préférera renvoyer directement vers l'article en question.)

- {id_syndic} retourne la liste des articles du site syndiqué dont l'identifiant est id_syndic.

- {id_rubrique} retourne la liste des articles syndiqués dans cette rubrique.

- {id_secteur} retourne la liste des articles syndiqués dans ce secteur.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : {par titre}).

- #ID_SYNDIC_ARTICLE affiche l'identifiant unique de l'article syndiqué.

- #ID_SYNDIC affiche l'identifiant unique du site syndiqué contenant cet article.

- #TITRE affiche le titre de l'article.

Remarque : il est préférable d'utiliser ici le titre « brut » de l'article syndiqué - via le code

`[(#TITRE*)]` -, pour éviter le moteur typographique. En effet les titres sont censés être déjà « typographiquement corrects » dans les backends, et on ne souhaite pas passer la correction typographique sur des titres en anglais ou sur des titres comprenant des expressions du genre « Les fichiers ~/.tcshrc ».

- `#URL_ARTICLE` affiche l'adresse (URL) de l'article syndiqué (sur son site original).
- `#DATE` affiche la date de publication de cet article.
- `#LESAUTEURS` affiche les auteurs de l'article syndiqué.
- `#DESCRIPTIF` affiche le descriptif de l'article syndiqué.
- `#NOM_SITE` affiche le nom du site syndiqué contenant cet article.
- `#URL_SITE` affiche l'adresse (URL) du site

La boucle SIGNATURES

Mai 2001 — maj : 16 octobre

La boucle SIGNATURES retourne une liste de signatures de pétition.

`<BOUCLEn(SIGNATURES){critères...}>`

Les critères de sélection

Les critères disponibles sont les suivants :

- `{id_article}` retourne les signatures de la pétition de cet article.
- `{id_signature}`, la signature correspondant à l'identifiant indiqué.
- `{id_trad}`, retourne les signatures des pétitions associées à l'article indiqué ou à ses traductions (SPIP2).
- `{tout}` toutes les signatures sont sélectionnées dans l'intégralité du site.

Tous ces critères peuvent être conditionnels, et éventuellement combinés. Par exemple `{id_trad ?}{id_article ?}` permet de sélectionner les signatures pour un article ou pour tout un jeu de traduction d'un article, selon le contexte fourni (SPIP2).

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Attention. Dans ce type de boucles, certains critères de classement ne sont pas identiques aux balises SPIP indiquées ci-dessous :
- `{par nom_email}` classe les résultats selon le `#NOM` du signataire ;
- `{par ad_email}` classe selon l'`#EMAIL` du signataire.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par`

nom_email}).

- #ID_SIGNATURE affiche l'identifiant unique du message.
- #ID_ARTICLE affiche l'identifiant de l'article pour cette pétition.
- #DATE affiche la date de publication.
- #MESSAGE affiche le texte du message.
- #NOM_EMAIL affiche le nom de l'auteur du message.
- #EMAIL affiche l'adresse email de l'auteur.
- #NOM_SITE affiche le nom du site Web indiqué par l'auteur.
- #URL_SITE affiche l'adresse (URL) de ce site Web.

La boucle HIERARCHIE

Mai 2001 — maj : 14 août

La boucle HIERARCHIE retourne la liste des RUBRIQUES qui mènent de la racine du site à la rubrique ou à l'article en cours.

<BOUCLEn(HIERARCHIE){critères...}>

Les critères de sélection

On utilisera obligatoirement l'un des deux critères suivants pour indiquer comment on sélectionne les éléments :

- {id_article} retourne la liste des rubriques depuis la racine jusqu'à la rubrique contenant l'article correspondant à cet identifiant.

- {id_rubrique} retourne la liste des rubriques depuis la racine jusqu'à la rubrique correspondant à cet identifiant (exclue).

Note : Depuis [SPIP 1.8], {tout} permet d'obtenir **aussi** la rubrique correspondant à l'identifiant spécifié.

Les critères {id_article} ou {id_rubrique} ne peuvent pas être utilisés avec une comparaison. Par exemple, <BOUCLE_hi(HIERARCHIE) {id_article=12}> retournera une erreur.

> *Attention :* cette boucle sera obligatoirement placée à l'intérieur d'une boucle ARTICLES ou RUBRIQUES — elle ne va pas par elle-même « chercher » l'id_article ou id_rubrique indiquée dans l'URL. (Le même principe vaut pour les boucles HIERARCHIE des squelettes inclus par la commande <INCLURE{fond=xxx}> ou <INCLURE(xxx.php3)> pour les version antérieure à 1.9)

Les critères d'affichage

Depuis [SPIP 1.8], tous les critères de La boucle RUBRIQUES peuvent être utilisés avec cette boucle, y compris les critères de tri (il devient possible par exemple de trier une <BOUCLE_x(HIERARCHIE){id_article}{par hasard}>).

Historique : Jusqu'à la version [SPIP 1.7.2], Les critères communs à toutes les boucles ne s'appliquent pas tous à ce type de boucle. Seuls les critères `{"inter"}` et `{a,b}` étaient utilisables.

Les balises de cette boucle

Les éléments obtenus avec une boucle `HIERARCHIE` sont des rubriques. On peut donc utiliser toutes les balises proposées pour les boucles `RUBRIQUES`.

Note : Il n'y a pas de critère id_breve dans HIERARCHIE mais, dans le cas d'une brève, l'usage d'id_article retournera quand même la bonne rubrique.

Les critères communs à toutes les boucles

Mai 2001 — maj : 15 juillet

Certains critères s'appliquent à (presque) tous les types de boucles. Ce sont des critères destinés à restreindre le nombre de résultats affichés ou à indiquer l'ordre d'affichage. On peut sans difficulté combiner plusieurs de ces critères de sélection.

Classer les résultats

`{par` *`critère_de_classement`***`}`** indique l'ordre de présentation des résultats. Ce critère de classement correspond à l'une des balises tirées de la base de données pour chaque type de boucle. Par exemple, on pourra classer les articles `{par date}`, `{par date_redac}` ou `{par titre}`. (Notez que, si les balises sont en majuscules, les critères de classement sont en minuscules.)

Cas particulier : `{par hasard}` permet d'obtenir une liste présentée dans un ordre aléatoire.

Inverser le classement. De plus, `{inverse}` provoque l'affichage du classement inversé. Par exemple `{par date}` commence par les articles les plus anciens ; avec `{par date} {inverse}` on commence la liste avec les articles les plus récents.

Depuis SPIP 1.9, le critère *inverse* peut prendre en paramètre n'importe quelle balise pour varier dynamiquement le sens du tri. Par exemple, il est possible d'écrire : `<BOUCLE_exemple(ARTICLES){par #ENV{tri}}{inverse #ENV{senstri}}>`, ce qui permet de choisir la colonne de tri et le sens du tri par l'url (&senstri=1 ou &senstri=0)

Classer par numéro. [SPIP 1.3] Lorsqu'on réalise le classement selon un élément de texte (par exemple le *titre*), le classement est réalisé par ordre *alphabétique*. Cependant, pour forcer un ordre d'affichage, on peut indiquer un numéro devant le titre, par exemple : « 1. Mon premier article », « 2. Deuxième article », « 3. Troisième... », etc ; avec un classement alphabétique, le classement de ces éléments donnerait la série « 1, 10, 11, 2, 3... ». Pour rétablir le classement selon les numéros, on peut utiliser le critère :

`{par num` `critère`**`}`**

Par exemple :

<BOUCLE_articles(ARTICLES){id_rubrique}{par date}{inverse}>

affiche les articles d'une rubrique classés selon l'ordre chronologique inversé (les plus récents au début, les plus anciens à la fin), et :

<BOUCLE_articles(ARTICLES){id_rubrique}{par titre}>

les affiche selon l'ordre alphabétique de leur titre ; enfin :

<BOUCLE_articles(ARTICLES){id_rubrique}{par num titre}>

les affiche selon l'ordre du numéro de leur titre (remarque : l'option `{par num titre}` ne fonctionne pas pour les plus anciennes versions de MySQL, antérieures à la version 3.23).

<BOUCLE_articles(ARTICLES){id_rubrique}{par multi titre}>

Dans le cadre d'un site multilingue le critère **`{par multi critère}`** permet de trier par ordre alphabétique dans chaque langue. Sans l'ajout de "multi" la boucle renvoie le même classement pour chaque langue.

Classer selon plusieurs critères A partir de SPIP 1.8, on peut classer selon plusieurs critères : **`{par critère1, critère2}`**. On indique ainsi des ordres de classement consécutifs. Les résultats seront d'abord triés selon le *critère1*, puis le *critère2* pour les résultats ayant le même *critère1*. On peut spécifier autant de critères que nescessaire.

Par exemple `{par date, titre}` triera les résultats par *date* puis les résultats ayant la même *date* seront triés par *titre*.

Avec [SPIP 1.8.2] on peut spécifier plusieurs critères **`{par ...}`** pour une boucle pour arriver au même résultat. Par exemple : `{par date} {par titre}` est équivalent à l'exemple précédent.

> *Remarque :* Quand on utilise plusieurs critères de tri, le critère **`{inverse}`** ne s'applique qu'au critère de tri placé juste avant.
>
> C'est pourquoi [SPIP 1.8.2] introduit la notation **`{!par ...}`** qui inverse un critère de tri en particulier. Par exemple : `{!par date} {par num titre}` tri par *date* décroissantes puis par numéros croissants dans le *titre* pour les résultats ayant la même *date*.

Comparaisons, égalités

`{`*`critère`*** `<` *`valeur`***`}`** Comparaison avec une valeur fixée (on peut utiliser « > », « < », « = », « >= », « <= ». Tous les *critères de classement* (tels que tirés de la base de données) peuvent également être utilisés pour limiter le nombre de résultats.

La valeur à droite de l'opérateur peut être :

- Une valeur constante fixée dans le squelette. Par exemple :
- <BOUCLE_art(ARTICLES){id_article=5}>

<BOUCLE_art(ARTICLES){id_article=5}>

affiche l'article dont le numéro est 5 (utile pour mettre en vedette un article précis sur la page d'accueil).

<BOUCLE_art(ARTICLES){id_secteur=2}>

affiche les articles du secteur numéro 2.

- A partir de [SPIP 1.8], une balise disponible dans le contexte de la boucle. Par exemple :

<BOUCLE_art(ARTICLES){id_article=5}>

<BOUCLE_titre(ARTICLES) {titre=#TITRE}>

...

</BOUCLE_titre>

</BOUCLE_art>

sert à trouver les articles qui ont le même titre que l'article 5.

> *Attention :* On ne peut utiliser q'une balise simple. Il n'est pas permis de la filtrer ou de mettre du code optionnel.
>
> Spécialement, si on veut utiliser la balise **#ENV** — ou tout autre balise prenant des paramètres —, on doit utiliser la notation : `{titre = #ENV{titre}}` et **pas** : `{titre = [(#ENV{titre})]}`.

Expressions régulières :

Très puissant (mais nettement plus complexe à manipuler), le terme de comparaison « == » introduit une comparaison selon une expression régulière. Par exemple :

<BOUCLE_art(ARTICLES){titre==^[aA]}>

sélectionne les articles dont le titre commence par « a » ou « A ».

Négation :

A partir de [SPIP 1.2] On peut utiliser la notation `{xxx != yyy}` et `{xxx !== yyy}`, le ! correspondant à la négation (opérateur logique NOT).

<BOUCLE_art(ARTICLES){id_secteur != 2}>

sélectionne les articles qui n'appartiennent pas au secteur numéro 2.

<BOUCLE_art(ARTICLES){titre!==^[aA]}>

sélectionne les articles dont le titre ne commence pas par « a » ou « A ».

Affichage en fonction de la date

Pour faciliter l'utilisation des comparaisons sur les dates, on a ajouté des critères :
- `age` et `age_redac` correspondent respectivement à l'ancienneté de la publication et de la première publication d'un article, en jours : `{age<30}` sélectionne les éléménts publiés depuis un mois ;
- les critères `mois`, `mois_redac`, `annee`, `annee_redac` permettent de comparer avec des valeurs fixes (`{annee<=2000}` pour les éléments publiés avant la fin de l'année 2000).

On peut combiner plusieurs de ces critères pour effectuer des sélections très précises. Par exemple :

<BOUCLE_art(ARTICLES){id_secteur=2}{id_rubrique!=3}{age<30}

affiche les articles du secteur 2, à l'exclusion de ceux de la rubrique 3, et publiés depuis moins de 30 jours.

Astuce. Le critère `age` est très pratique pour afficher les articles ou les brèves dont la date est située « dans le futur », avec des valeurs négatives (à condition d'avoir sélectionné, dans la Configuration précise du site, l'option « Publier les articles post-datés »). Par exemple, ce critère permet de mettre en valeur des événements futurs. `{age<0}` sélectionne les articles ou les brèves dont la date est

située dans le futur (« après » aujourd'hui)...

[SPIP 1.3] *Âge par rapport à une date fixée.* Le critère `age` est calculé par rapport à la date d'aujourd'hui (ainsi `{age<30}` correspond aux articles publiés depuis moins d'un mois par rapport à aujourd'hui). Le critère **`age_relatif`** compare la date d'un article ou d'une brève à une date « courante » ; par exemple, à l'intérieur d'une boucle ARTICLES, on connaît déjà une date pour chaque résultat de la boucle, on peut donc sélectionner par rapport à cette date (et non plus par rapport à aujourd'hui).

Par exemple :

<BOUCLE_article_principal(ARTICLES){id_article}>

<h1>#TITRE</h1>

<BOUCLE_suivant(ARTICLES){id_rubrique}{age_relatif<=0}{exclus}{par date}{0,1}>

Article suivant: #TITRE

</BOUCLE_suivant>

</BOUCLE_article_principal>

la BOUCLE_suivant affiche un seul article de la même rubrique, classé par date, dont la date de publication est inférieure ou égale à la date de l'« article_principal » ; c'est-à-dire l'article de la même rubrique publié après l'article principal.

De plus amples informations sur l'utilisation des dates se trouvent dans l'article sur « La gestion des dates ».

Affichage d'une partie des résultats

- `{branche}` A partir de [SPIP 1.8.2], limite les résultats — pour des boucles ayant un #ID_RUBRIQUE — à la branche actuelle (la rubrique actuelle et ses sous-rubriques). Par exemple :

`<BOUCLE_articles(ARTICLES) {branche}>` retournera tous les articles de la rubrique actuelle et de ces sous-rubriques,
`<BOUCLE_articles(ARTICLES) {!branche}>` retournera tous les articles qui ne sont pas dans la rubrique actuelle ou ses sous-rubriques,

On peut utiliser le critère **`{branche?}`** *optionnel* pour ne l'appliquer que si une rubrique est sélectionnée dans le contexte (une boucle englobante ou l'url fournie un id_rubrique). Par exemple :

`<BOUCLE_articles(ARTICLES) {branche?}>` retournera tous les articles de la rubrique actuelle et de ces sous-rubriques si il y a un id_rubrique dans le contexte, sinon, tous les articles du site.

- `{doublons}` ou `{unique}` (ces deux critères sont rigoureusement identiques) permettent d'interdire l'affichage des résultats déjà affichés dans d'autres boucles utilisant ce critère.

historique : A partir de [SPIP 1.2] et jusqu'à [SPIP 1.7.2], seules les boucles

ARTICLES, RUBRIQUES, DOCUMENTS et SITES acceptaient ce critère.

- {doublons *xxxx*} à partir de [SPIP 1.8], on peut avoir plusieurs jeux de critères {doublons} indépendants. Les boucles ayant {doublons rouge} n'auront aucune incidence sur les boucles ayant {doublons bleu} comme critère.

- {exclus} permet d'exclure du résultat l'élément (article, brève, rubrique, etc.) dans lequel on se trouve déjà. Par exemple, lorsque l'on affiche les articles contenus dans la même rubrique, on ne veut pas afficher un lien vers l'article dans lequel on se trouve déjà.

- **{***xxxx* **IN** *a,b,c,d***}** à partir de [SPIP 1.8], limite l'affichage aux résultats ayant le critère *xxxx* égal à *a, b, c* **ou** *d.* Les résultats sont triés dans l'ordre indiqué (sauf demande explicite d'un autre critère de tri). Il est aussi possible de sélectionner des chaînes de caractères, par exemple avec {titre IN 'Chine', 'Japon'}.

Avec SPIP 1.9, les balises sont admises dans les arguments de IN, et notamment la balise ENV, à laquelle sont appliqués les filtres d'analyse pour assurer que la requête SQL sera bien écrite. De manière dérogatoire, SPIP testera si l'argument de ENV désigne un tableau (venant par exemple de saisies de formulaire dont l'attribut name se termine par []). Si c'est le cas, et si les filtres d'analyse ont été désactivés en suffixant cette balise par une double étoile, alors chaque élément du tableau sera considéré comme argument de IN, SPIP appliquant les filtres de sécurité sur chacun d'eux.

Le squelette standard formulaire_forum_previsu fournit un exemple d'utilisation avec une boucle MOTS ayant le critère {id_mot IN #ENV**{ajouter_mot}} : cette boucle sélectionne seulement les mots-clés appartenant à un ensemble indiqué dynamiquement. Ici, cet ensemble aura été construit par le formulaire du squelette standard choix_mots, qui utilise des attributs name=ajouter_mot[].

- **{***a,b***}** où *a* et *b* sont des chiffres. Ce critère permet de limiter le nombre de résultats. *a* indique le résultat à partir duquel on commence l'affichage (attention, le premier résultat est numéroté 0 - zéro) ; *b* indique le nombre de résultats affichés.

Par exemple {0,10} affiche les dix premiers résultats ; {4,2} affiche les deux résultats à partir du cinquième (inclus).

- **{debut_***xxx,b***}** est une variante très élaborée de la précédente. Elle permet de faire commencer la limitation des résultats par une variable passée dans l'URL (cette variable remplace ainsi le *a* que l'on indiquait précédemment). C'est un fonctionnement un peu compliqué, que fort heureusement on n'a pas besoin d'utiliser trop souvent.

La variable passée dans l'URL commence forcément par debut_*xxx* (où *xxx* est un mot choisi par le webmestre) . Ainsi, pour une page dont l'URL est :

spip.php?page=petition&id_article=13&debut_signatures=200

avec un squelette (petition.html) contenant par exemple :

<BOUCLE_signatures(SIGNATURES){id_article}{debut_signatures,100}>

on obtiendra la liste des 100 signatures à partir de la 201-ième [rappel]. Avec l'URL :

spip.php?page=petition&id_article=13&debut_signatures=300

on obtient la liste des 100 signatures à partir de la 301-ième [rappel].

- **{***a*,n-*b***}** à partir de [SPIP 1.8], est une variante de {a,b} qui limite l'affichage en fonction du nombre de résultats dans la boucle. *a* est le résultat à partir duquel commencer à faire l'affichage ; *b* indique le nombre de résultats à **ne pas afficher** à la fin de la boucle.

{0,n-10} affichera tous les résultats de la boucle sauf les 10 derniers.

- **{n-***a*,*b***}** à partir de [SPIP 1.8], est le pendant de {a, n-b}. On limite à *b* résultats en commençant l'affichage au a^e résultat avant la fin de la boucle.

Par exemple : {n-20,10} affichera au 10 résultats en partant du 20^e résultat avant la fin de la boucle.

- **{***a***/***b***}** où *a* et *b* sont des chiffres. Ce critère permet d'afficher une partie *a* (proportionnellement) des résultats en fonction d'un nombre de « tranches » *b*.

Par exemple : {1/3} affiche le premier tiers des résultats. Ce critère est surtout utile pour présenter des listes sur plusieurs colonnes. Pour obtenir un affichage sur deux colonnes, il suffit de créer une première boucle, affichée dans une case de tableau, avec le critère {1/2} (la première moitié des résultats), puis une seconde boucle dans une seconde case, avec le critère {2/2} (la seconde moitié des résultats).

Attention. L'utilisation du critère {doublons} avec ce critère est périlleuse. Par exemple :

```
<BOUCLE_prem(ARTICLES){id_rubrique}{1/2}{doublons}>
 <li> #TITRE
</BOUCLE_prem>
<BOUCLE_deux(ARTICLES){id_rubrique}{2/2}{doublons}>
 <li> #TITRE
</BOUCLE_deux>
```

n'affichera pas tous les articles de la rubrique ! Imaginons par exemple qu'il y ait au total 20 articles dans notre rubrique. La BOUCLE_prem va afficher la première moitié des articles, c'est-à-dire les 10 premiers, et interdire (à cause de {doublons}) de les réutiliser. La BOUCLE_deux, elle, va récupérer la deuxième moitié des articles de cette rubrique *qui n'ont pas encore été affichés* par la BOUCLE_prem ; donc, la moitié des 10 articles suivants, c'est-à-dire les 5 derniers articles de la rubrique. Vous avez donc « perdu » 5 articles dans l'opération...

Affichage entre les résultats

{"*inter***"}** permet d'indiquer un code HTML (ici, *inter*) inséré *entre* les résultats de la boucle. Par exemple, pour séparer une liste d'auteurs par une virgule, on indiquera :

```
<BOUCLE_auteurs(AUTEURS){id_article}{", "}>
```

Divers

{logo} permet de ne sélectionner que les articles (ou rubriques, etc) qui disposent d'un logo. Il fonctionne aussi dans la boucle (HIERARCHIE). Le critère inverse {!logo} liste les objets qui n'ont pas de logo.

Notes

[rappel] le premier résultat est numéroté 0, donc le 200e résultat représente réellement la 201 e signature

Les balises propres au site

Décembre 2001 — maj : 25 mai

Les balises suivantes sont disponibles à n'importe quel endroit du squelette, même en dehors d'une boucle (hors « contexte »).

Balises définies à la configuration

Le contenu de ces balises est défini dans l'espace privé, lors de la configuration de votre site.

- `#NOM_SITE_SPIP` affiche le nom du site.

- `#URL_SITE_SPIP` affiche l'adresse du site. Elle ne comprend pas le `/` final, ainsi vous pouvez créer un lien du type `#URL_SITE_SPIP/sommaire.php3`

- `#DESCRIPTIF_SITE_SPIP` (depuis SPIP 1.9) affiche, comme son nom l'indique, le descriptif du site, que l'on renseigne dans la page de configuration générale du site.

- `#EMAIL_WEBMASTER` (depuis SPIP 1.5) affiche l'adresse du webmestre. Par défaut, SPIP prend l'adresse de celui qui a installé le site (le premier administrateur).

- `#LOGO_SITE_SPIP` (depuis SPIP 1.8) affiche le logo du site. Depuis SPIP 1.9 cette balise renvoie le logo du site 0. Il ne faut pas confondre avec le logo de la racine, aussi désigné sous le nom de logo standard des rubriques, c'est-à-dire celui de la rubrique 0.

- `#CHARSET` (depuis SPIP 1.5) affiche le jeu de caractères utilisé par le site. Sa valeur par défaut est `iso-8859-1`, jeu de caractères dit « iso-latin » [1].

- `#LANG` (depuis SPIP 1.7, SPIP 1.7.2) : utilisée en dehors des boucles ARTICLES, RUBRIQUES, BREVES et AUTEURS, cette balise affiche la langue principale du site.

- `#LANG_DIR`, `#LANG_LEFT`, `#LANG_RIGHT` (depuis SPIP 1.7, SPIP 1.7.2) : ces balises définissent le sens d'écriture de la langue du contexte actuel (par exemple, de l'article qu'on est en train d'afficher). Voir l'article « Réaliser un site multilingue » pour plus d'information.

- `#MENU_LANG` (et `#MENU_LANG_ECRIRE`) (depuis SPIP 1.7, SPIP 1.7.2) : ces balises fabriquent et affichent un menu de langues permettant au visiteur d'obtenir la page en cours dans la langue choisie. La première balise affiche la liste des langues du site ; la seconde la liste des langues de l'espace privé (elle est utilisée sur la page de connexion à l'espace privé).

Balises de mise en page

- SPIP 1.8.2 introduit la balise `#DOSSIER_SQUELETTE` pour pouvoir développer des squelettes facilement transportables et échangeables. Elle permet d'obtenir le chemin du dossier dans lequel est installé le squelette utilisé.

On peut ainsi placer les fichiers « accessoires » (feuille de style, javascript, etc...) au squelette dans le répertoire du squelette et donc simplement distribuer ce dossier pour échanger ses squelettes. On

écrira donc, par exemple, pour inclure une feuille de style du répertoire squelette :

<link rel="stylesheet" href="#DOSSIER_SQUELETTE/mon_style.css" type="text/css" />

Depuis SPIP 1.9, la balise `#CHEMIN` remplace et améliore `#DOSSIER_SQUELETTE`. `#CHEMIN{xxx}` donnera le chemin complet vers le fichier `xxx`, qu'il se trouve à la racine, dans le dossier des squelettes, dans dist/ etc.

<link rel="stylesheet" href="#CHEMIN{mon_style.css}" type="text/css" />

- `#PUCE` (depuis SPIP 1.5) qui affiche devinez-quoi...

- `#FORMULAIRE_ADMIN` (depuis SPIP 1.5) est une balise optionnelle qui permet de placer les boutons d'administration (« recalculer cette page », etc.) dans ses squelettes. Lorsqu'un administrateur parcours le site public, si cette balise est présente, elle sera remplacée par les boutons d'administration, sinon, les boutons seront placés à la fin de la page.

Depuis SPIP 1.8, on peut aussi modifier la feuille de style *spip_admin.css* pour contrôler la position des boutons.

- `#DEBUT_SURLIGNE`, `#FIN_SURLIGNE` sont deux balises qui indiquent à SPIP dans quelle partie de la page colorer les mots clefs recherchés. Voir : « Les boucles et balises de recherche ».

- La balise `#INSERT_HEAD` (depuis SPIP 1.9.1) doit se situer entre les balises `<head>` et `</head>` de vos squelettes. Elle permet à SPIP, ainsi qu'aux plugins éventuels, d'ajouter du contenu entre ces deux balises html.

Balises techniques

Attention, ces balises s'adressent à des utilisateurs avertis de SPIP.

- La balise `#REM` ne produit aucun affichage : elle permet de commenter le code des squelettes, de cette façon : `[(#REM) Ceci est un commentaire. ]`. Ces commentaires n'apparaissent pas dans le code généré pour le site public.

- `#SELF` (depuis SPIP 1.8) retourne l'URL de la page appelée, nettoyée des variables propres à l'exécution de SPIP. Par exemple, pour une page avec l'url : `article.php3?id_article=25&var_mode=recalcul` la balise #SELF retournera : `article.php3?id_article=25`

Par exemple pour faire un formulaire :

<form action="#SELF" method="get">

Remarque : la balise `#SELF` représentant l'adresse de la page, elle n'est pas compatible avec les `<INCLURE()>` (sauf si le `$delais` de l'inclusion est mis à 0).

- `#URL_PAGE` (depuis SPIP 1.9) retourne une url de type « page » (cf. les urls de spip), vers la page passée en paramètre et qui pourra être utilisée dans un lien. Par exemple, pour accéder à la page générée par le squelette `toto.html`, située dans votre dossier-squelette, `#URL_PAGE{toto}` généra automatiquement l'url `spip.php?page=toto`. Un second paramètre est autorisé pour ajouter des paramètres à l'url. Exemple `#URL_PAGE{toto,id_article=#ID_ARTICLE}` génèrera l'url `spip.php?page=toto&id_article=XXX`.

- `[(#ENV{xxxx,zzzz})]` (depuis SPIP 1.8) permet d'accéder à la variable de nom *xxxx* passée par la requête HTTP. *zzzz* est une partie optionnelle qui permet de retourner une valeur même si la

variable *xxxx* n'existe pas. On trouve une explication détaillée sur Spip-Contrib

Par défaut, la balise #ENV est filtrée par `htmlspecialchars`. Si on veut avoir le résultat brut, l'étoile « * » peut être utilisée comme pour les autres balises : `[(#ENV*{xxxx})]`.

Par exemple pour limiter la liste d'auteurs affichés :

<BOUCLE_auteurs(AUTEURS) {nom == #ENV{lettre,^A}}>

Retourne la liste d'auteur ayant le nom correspondant à l'expression régulière passé dans l'url par la variable lettre (`liste_auteur.php3?lettre=^Z`) ou les auteurs qui ont un nom commençant par un 'A' s'il n'y a pas de variable dans l'url.

- La balise `#SPIP_CRON` (depuis SPIP 1.8) est liée à la gestion par SPIP des calculs qu'il doit faire périodiquement (statistiques, indexation pour le moteur de recherche, syndication de sites etc.).

Si cette balise n'est pas présente sur le site, le moteur de SPIP effectue ses calculs, en temps utile, après avoir envoyé une page à un visiteur ; malheureusement php ne permet pas de fermer la connexion à la fin de la page, et dans certains cas cela peut conduire certains visiteurs malchanceux (ceux dont le passage déclenche une procédure un peu longue, notamment la syndication) à constater une certaine lenteur dans l'affichage de la page demandée.

La balise `#SPIP_CRON` permet de contourner ce problème : son rôle est de générer un marqueur `<div>`

invisible dont la propriété « background » pointe sur le script `spip_background.php3` ; ce script à son tour effectue les calculs nécessaires « en tâche de fond », et renvoie une image transparente de 1×1 pixel. Cette astuce permet donc d'éviter tout sentiment de « ralentissement » en déportant les éventuelles lenteurs sur un script annexe.

> *A noter :* cette balise n'est pas stratégique, et sa présence ou son absence ne modifient en rien la régularité du calcul des tâches périodiques du site.

- La balise `#SET{`*`variable,valeur`*`}` et son pendant `#GET{`*`variable`*`}` ont été introduites par SPIP 1.9.1. La balise `#SET{xxx,yyy}` affecte une valeur yyy à une variable xxx **propre au squelette calculé**. Cette valeur peut être récupérée par la balise `#GET{xxx}`. Les variables créées ainsi ne sont pas transmises au squelette inclus.

Attention ! Si l'on affecte une valeur à une variable dans la partie facultative avant d'une boucle, il ne sera pas possible de récupérer cette valeur dans la boucle. Cela tient à la manière dont Spip calcule les squelettes.

- La balise `#HTTP_HEADER{`*`argument`*`}` (depuis SPIP 1.9) permet de modifier l'entête HTTP de la page retournée par SPIP. Exemple : `#HTTP_HEADER{Content-Type: text/css}`. **Attention !** Le fait d'utiliser cette balise supprime les boutons d'administration. Cette balise ne peut pas être utilisée dans des squelettes inclus via la syntaxe `<INCLURE>`.

- La balise `#EVAL{`*`argument`*`}` (depuis SPIP 1.9) évalue l'expression PHP mise en accolade. Par exemple `#EVAL{1+1}` affichera 2, `#EVAL{_DIR_IMG_PACK}` affichera ainsi le chemin vers le répertoire `ecrire/img_pack/`. Attention, il est fortement conseillé de s'en servir avec modération.

- La balise `#CACHE{`*`temps`*`}` permet de déterminer le délai au bout duquel le squelette est réinterprété. Le temps est exprimé en secondes. Il peut se mettre sous forme de calcul. Par exemple : `#CACHE{24*3600}`.

Notes

[1] Cf. http://www.uzine.net/article1785.html pour une introduction aux charsets, en attendant une documentation plus complète de cette fonctionnalité de SPIP.

Les formulaires

Août 2002 — maj : Mai 2007

SPIP permet une grande interaction du site avec les visiteurs ; pour cela, il propose de nombreux formulaires sur le site public, permettant tantôt de gérer les accès à l'espace privé, tantôt d'autoriser l'ajout de messages et signatures.

Les formulaires s'insèrent dans les squelettes par une simple balise ; SPIP se charge ensuite de gérer le comportement (souvent complexe) de ces formulaires en fonction de l'environnement et des configurations effectuées dans l'espace privé.

Fonctions interactives

- `#FORMULAIRE_RECHERCHE`

Il s'agit du formulaire du moteur de recherche intégré à SPIP. Il est présenté dans l'article sur les boucles de recherche.

- `#FORMULAIRE_FORUM`

Le `#FORMULAIRE_FORUM` gère l'interface permettant de poster des messages dans les forums publics. Il concerne donc en premier chef la boucle FORUMS mais peut être utilisé dans toutes les boucles acceptant un forum :

- La boucle ARTICLES,
- La boucle RUBRIQUES,
- La boucle BREVES,
- La boucle SITES (ou SYNDICATION).

Le formulaire dépend évidemment du choix des forums modérés a posteriori, a priori ou sur abonnement.

[SPIP 1.8.2] Par défaut, une fois le message posté, le visiteur est renvoyé vers la page de l'élément [1] auquel il a répondu. On peut décider de renvoyer le visiteur vers une autre page en passant une url en paramètre à cette balise. Par exemple :

- `[(#FORMULAIRE_FORUM{'spip.php?page=merci'})]` renverra vers la page *spip ?page=merci*.

  Historique : Pour les versions antérieures à [SPIP 1.9], il aurait fallu écrire `merci.php3`.

  Jusqu'à [SPIP 1.9] , les fichiers de spip avaient une extension en `.php3` et non en `.php`.

- `[(#FORMULAIRE_FORUM{#SELF})]` renverra vers la page où le formulaire de forum est placé (voir la balise #SELF).

Dans le cas (très spécifique) où l'on a autorisé la présence de mots-clés dans les forums publics, on

peut affiner le comportement de ce formulaire avec des variables de personnalisation.

- #FORMULAIRE_SIGNATURE

La balise #FORMULAIRE_SIGNATURE affiche un formulaire permettant aux visiteurs du site de signer les pétitions associées aux articles. Cette balise se place donc dans une boucle ARTICLES.

La signature des pétitions réclame obligatoirement une validation des signataires par email. Ce formulaire n'a donc d'intérêt que si votre hébergeur autorise l'envoi de mails par PHP.

- #FORMULAIRE_SITE

Introduite dans [SPIP 1.4], la balise #FORMULAIRE_SITE affiche un formulaire permettant aux visiteurs du site de proposer des référencements de sites. Ces sites apparaîtront comme « proposés » dans l'espace privé, en attendant une validation par les administrateurs.

Ce formulaire ne s'affiche que si vous avez activé l'option « Gérer un annuaire de sites » dans la *Configuration sur site* dans l'espace privé, et si vous avez réglé « Qui peut proposer des sites référencés » sur « les visiteurs du site public ».

Les sites référencés étant, dans SPIP, attachés aux rubriques, on ne peut placer ce #FORMULAIRE_SITE qu'à l'intérieur d'une boucle RUBRIQUES.

- #FORMULAIRE_ECRIRE_AUTEUR

[SPIP 1.4] Placée à l'intérieur d'une boucle AUTEURS, cette balise affiche le formulaire qui permet d'envoyer un mail à l'auteur. Cela permet d'écrire aux auteurs sans divulguer leur adresse email sur le site public.

[SPIP 1.8.2] Placé dans une boucle ARTICLES, ce formulaire permet d'envoyer un mail à tous les auteurs de cet article.

[SPIP 1.8.2] Placé dans une boucle FORUMS, ce formulaire permet d'envoyer un mail directement à l'auteur du message si l'auteur est enregistré sur le site.

Inscription, authentification...

- #FORMULAIRE_INSCRIPTION

Sans doute le plus importante, la balise #FORMULAIRE_INSCRIPTION affiche le formulaire permettant l'inscription de nouveaux rédacteurs. Celui-ci ne s'affiche que si vous avez autorisé l'inscription automatique depuis le site public (sinon, cette balise n'affiche rigoureusement rien).

L'inscription nécessite l'envoi des informations de connexion (login et mot de passe) par email ; ce formulaire ne fonctionne donc que si votre hébergeur autorise l'envoi de mails par PHP.

- [(#FORMULAIRE_INSCRIPTION{forum})]

Est l'équivalente de la précédente, pour l'inscription des visiteurs, appelés à écrire dans les forums (réservés aux visiteurs enregistrés), option qui se détermine dans la partie privée configuration/interactivité/Mode de fonctionnement par défaut des forums publics.

Après la validation, un message avertit le visiteur : "Votre nouvel identifiant vient de vous être envoyé par email."

- #LOGIN_PRIVE

SPIP 1.4 Tout aussi importante (sinon plus), cette balise affiche le formulaire d'accès à l'espace privé (la partie « /ecrire » du site).

Important : cette balise doit impérativement être présente dans le squelette appelé par la page

`spip.php?page=login`, c'est-à-dire dans le squelette nommé `login.html`. En effet, lors des accès directs à l'adresse « /ecrire » de votre site, c'est vers `spip.php?page=login` que SPIP va vous rediriger.

> *Historique :* Pour les versions antérieures à SPIP 1.9, il s'agit du squelette appelé par la page `spip_login.php3`, c'est-à-dire en standard par le squelette nommé `spip_login.html`
>
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme http://monsite.net/xxx.php3 et non pas `http://monsite.net/spip.php?page=xxx`.

- `#LOGIN_PUBLIC`

SPIP 1.4 D'une utilisation beaucoup plus spécifique, la balise `#LOGIN_PUBLIC` affiche un formulaire permettant à vos utilisateurs de s'identifier tout en restant sur le site public (sans entrer dans l'espace privé). Cette balise sert notamment à authentifier les visiteurs pour les sites proposant des forums *modérés sur abonnement*. Elle peut aussi servir de brique de base pour restreindre l'accès à certains contenus sur le site public : mais cela reste d'un maniement complexe, et nécessitera encore des développements et la rédaction de tutoriels complets avant d'être facilement utilisable par tous ; néanmoins, un exemple d'utilisation avancée est donné plus bas.

Le `#LOGIN_PUBLIC`, par défaut, « boucle sur lui-même », c'est-à-dire que le formulaire revient sur la page où il se trouve. On peut cependant indiquer une page vers laquelle le formulaire mènera, sous la forme :

[(#LOGIN_PUBLIC|spip.php?page=mapage)]

> *Historique :* Dans les versions antérieures à SPIP 1.9 il aurait fallu écrire `mapage.php3` et non `spip.php?page=mapage`
>
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme http://monsite.net/xxx.php3 et non pas `http://monsite.net/spip.php?page=xxx` .

Si votre site offre une inscription automatique à l'espace privé, les données de connexion à l'espace public sont identiques à celles de l'espace privé ; c'est-à-dire que les données envoyées à l'utilisateur pour s'identifier à l'espace public lui permettent également d'accéder à l'espace privé. Si, au contraire, vous avez interdit l'inscription automatique à l'espace privé, *il faut impérativement avoir au moins un article dont les forums seront réglés en mode « sur abonnement »* pour activer cette balise ; dès lors, SPIP pourra fournir des informations de connexion pour le site public sans accès à l'espace privé.

- `#URL_LOGOUT` [SPIP 1.5] est le pendant de `#LOGIN_PUBLIC` ; il donne une URL permettant à un visiteur authentifié de se déconnecter.

[SPIP 1.8.2] On peut passer un paramètre à cette balise pour spécifier l'adresse de retour après la déconnection. Par exemple `[(#URL_LOGOUT{spip.php?page=sommaire})]` renverra vers la page de sommaire.

> *Historique :* Dans les versions antérieur à SPIP 1.9, il aurait fallu écrire `[(#URL_LOGOUT{sommaire.php3})]` et non pas `[(#URL_LOGOUT{spip.php?page=sommaire})]`

> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme http://monsite.net/xxx.php3 et non pas `http://monsite.net/spip.php?page=xxx`.

Voici un exemple simple, mais complet, d'utilisation de ces deux balises. Il faut passer par un peu de php pour tester la variable `$auteur_session`, qui indique qu'un auteur est identifié ou non. Si c'est le cas, on peut récupérer (voire tester) son statut, son login, etc., via `$auteur_session['statut']`....

Notez bien que le contenu n'est « sécurisé » que sur ce squelette. Si votre squelette « imprimer cet article », par exemple, ne vérifie par `$auteur_session`, tout le monde (y compris les moteurs de recherche !) pourra avoir accès à ce fameux contenu que vous souhaitez protéger.

<?php if ($auteur_session) { ?>

Vous êtes authentifié, cliquez ici pour vous déconnecter

... ici le contenu en accès restreint....

<?php } else { ?>

<h2>Cette partie est en accès restreint</h2>

#LOGIN_PUBLIC

<?php } ?

Styles

On peut sensiblement modifier l'interface graphique des formulaires par l'intermédiaire des feuilles de style. Voir : « Ils sont beaux, mes formulaires ! ».

Notes

[1] article, rubrique, brève, site ou forum

Les filtres de SPIP

Mai 2001 — maj : 4 novembre

Nous avons vu dans la syntaxe des balises SPIP qu'il était possible de modifier le comportement et l'affichage des balises en leur attribuant des filtres.

[option avant **(#BALISE**|filtre1|filtre2|...|filtre*n***)** option après **]**

Les filtres 1, 2, ..., n sont appliqués successivement à la `#BALISE`.

Les filtres de mise en page

Les filtres de mise en page suivants (majuscules, justifier...) ne sont plus conseillés. Il est recommandé de leur préférer, désormais, l'utilisation des styles CSS correspondants.

- **majuscules** fait passer le texte en majuscules. Par rapport à la fonction de PHP, *majuscules* s'applique également aux lettres accentuées.

- **justifier** fait passer le texte en justification totale (`<P align=justify>`).

- **aligner_droite** fait passer le texte en justification à droite (`<P align=right>`).

- **aligner_gauche** fait passer le texte en justification à gauche (`<P align=left>`).

- **centrer** centre le texte (`<P align=center>`).

Les filtres des dates

Les filtres suivants s'appliquent aux dates (`[(#DATE|affdate)]` par exemple).

- **affdate** affiche la date sous forme de texte, par exemple « 13 janvier 2001 ».

[SPIP 1.8] étend la notation de ce filtre. On peut lui passer un paramètre de formatage de la date, correspondant à un format spip (« `'saison'` », etc.) ou à un format de la commande php date (« `'Y-m-d'` »). Par exemple :

- `[(#DATE|affdate{'Y-m'})]` affichera numériquement l'année et le mois de la date filtrée séparés par un tiret,
- la notation `[(#DATE|affdate{'saison'})]` est totalement équivalente à : `[(#DATE|saison)]`.

- Il existe aussi des variantes de *affdate* qui fournissent des raccourcis :

affdate_jourcourt affiche le nom du mois et la valeur numérique du jour, e.g. « 19 Avril ». Si la date n'est pas dans l'année actuelle, alors l'année est aussi affichée : « 1 Novembre 2004 ».

affdate_court affiche le nom du mois et le numéros du jour, e.g. « 19 Avril ». Si la date n'est pas dans l'année actuelle, alors on affiche seulement le mois et l'année sans le numéros du jour : « Novembre 2004 ».

affdate_mois_annee affiche seulement le mois et l'année : « Avril 2005 », « Novembre 2003 ».

- **jour** affiche le jour (en nombre).

- **mois** affiche le mois (en nombre).

- **annee** affiche l'année.

- [SPIP 1.0.2] **heures** affiche les heures d'une date (les dates fournies par SPIP contiennent non seulement le jour, mais également les horaires).

- [SPIP 1.0.2] **minutes** affiche les minutes d'une date.

- [SPIP 1.0.2] **secondes** affiche les secondes.

- **nom_jour** affiche le nom du jour (lundi, mardi...).

- **nom_mois** affiche le nom du mois (janvier, février...).

- **saison** affiche la saison (hiver, été...).

- [SPIP 1.8] introduit le filtre **unique** qui retourne la valeur de l'élément filtré seulement si c'est la première fois qu'elle est rencontrée. Ce filtre n'est pas limité aux dates mais est intéressant pour,

par exemple, afficher une liste d'articles par date :

<BOUCLE_blog(ARTICLES){par date}{inverse}{"<br>"}>

[<hr /><h1>(#DATE|affdate_mois_annee|unique)</h1>]

#TITRE ...

</BOUCLE_blog>

cette balise n'affichera la date qu'à chaque changement de mois.

Voici un autre exemple :

<BOUCLE_blog2(ARTICLES){par date}{inverse}>

[<hr /><h1>(#DATE|annee|unique)</h1>]

[<h2>(#DATE|affdate{'Y-m'}|unique|nom_mois)</h2>]

#TITRE

</BOUCLE_blog2>

affichera une liste ressemblant à :

```
2005
     mars
             article de mars
             autre article de mars
     février
             article de février
2004
     décembre
             un article
```

On utilise la notation `affdate{'Y-m'}` pour afficher le nom du mois à chaque année. En effet :

- si l'on ne faisait que `#DATE|nom_mois|unique`, les noms de mois ne seraient affichés que la première année.
- si le filtrage était : `#DATE|unique|nom_mois`, on afficherait toutes les dates. En effet, `#DATE` retourne une date complète qui contient aussi l'heure. Il y a donc une grande chance que les dates complètes de deux articles publiés le même jours soient différentes.

C'est pourquoi on garde juste le mois et l'année de la date avant de la passer au filtre *unique*.

On peut passer un argument optionnel à ce filtre pour différencier deux utilisations indépendantes du filtre. Par exemple : `[(#DATE|affdate_mois_annee|unique{ici})]` n'aura pas d'incidence sur `[(#DATE|affdate_mois_annee|unique{la})]`.

Filtres de texte

La plupart de ces filtres ont été introduits dans la version [SPIP 1.4]

- **liens_ouvrants** transforme les liens SPIP qui donnent vers des sites extérieurs en liens de type « popup », qui ouvrent dans une nouvelle fenetre ; c'est l'équivalent du *target=blank* du HTML. *N.B. : les développeurs de SPIP estiment qu'il s'agit en général d'une impolitesse, car les internautes savent très bien s'ils ont envie ou pas d'ouvrir une nouvelle fenêtre - or ce système le leur impose. Mais la demande était trop forte, et nous avons craqué ;-)*

- **supprimer_numero** sert à éliminer le numéro d'un titre, si par exemple on veut faire des tris

d'articles `{par num titre}` mais ne pas afficher les numéros (car ils ne servent qu'à ordonner les articles). Le format des préfixes numerotés est « XX. titre », XX étant un nombre à n chiffres (illimité).

- **PtoBR** transforme les sauts de paragraphe en simples passages a la ligne, ce qui permet de « resserrer » une mise en page, par exemple à l'intérieur d'un sommaire

- **taille_en_octets** permet de transformer un nombre d'octets (25678906) en une chaîne de caractères plus explicite (« 24.4 Mo »).

- **supprimer_tags** est une suppression basique et brutale de tous les `<...>`

- **textebrut** s'apparente au filtre `supprimer_tags`, mais il agit de manière un peu plus subtile, transformant notamment les paragraphes et `<br>` en sauts de ligne, et les espaces insécables en espaces simples. Il s'utilise, par exemple, pour faire un descriptif META : `[<meta name="description" content="(#DESCRIPTIF|textebrut)">]`

- **texte_backend** peut être utilisé pour transformer un texte et le rendre compatible avec des flux XML. Ce filtre est utilisé, par exemple, dans le squelette *backend.html* qui génére le fil RSS du site.

- **couper** coupe un texte après un certain nombre de caractères. Il essaie de ne pas couper les mots et enlève le formatage du texte. Si le texte est trop long, alors « (...) » est ajouté à la fin. Ce filtre coupe par défaut à 50 caractères, mais on peut spécifier une autre longueur en passant un paramètre au filtre, par exemple : `[(#TEXTE|couper{80})]`.

- **lignes_longues**, introduit par SPIP 1.9, coupe les mots « trops longs » (utile si l'on a, par exemple, des urls à afficher dans une colone étroite). Ce filtre coupe par défaut à 70 caractères, mais on peut spécifier une autre longueur en passant un paramètre au filtre, par exemple : `[(#URL_SITE|lignes_longues{40})]`.

- **match** utilise une expression régulière (cf. *preg_match()*) pour extraire un motif dans le texte si il est présent, ne retourne rien sinon. Par exemple pour récupérer le premier mot du titre `[(#TITRE|match{^\w+})]`. Ce peut être un texte simple, afficher "toto" si dans le titre : `[(#TITRE|match{toto})]`

- **replace** utilise aussi une expression régulière (cf. *preg_replace()*) pour supprimer ou remplacer toutes les occurences d'un motif dans le texte. Avec un seul parametre, une expression régulière, le motif sera remplacé par une chaîne, c'est à dire supprimé. Par exemple pour supprimer tous les "notaXX" du texte `[(#TEXTE|replace{nota\d*})]`. Lorsqu'un deuxiéme paramètre est fourni, les occurences du motif seront remplacées par cette valeur. Par exemple pour remplacer tous les 2005 ou 2006 du texte en 2007 `[(#TEXTE|replace{200[56],2007})]`. Ce peut être des textes simples, remplacer tous les "au temps" par "autant" : `[(#TEXTE|replace{au temps,autant})]`

Filtres de test

- [SPIP 1.6] introduit le filtre `|sinon`, qui indique ce qu'il faut afficher si l'élément « filtré » est vide : ainsi `[(#TEXTE|sinon{"pas de texte"})]` affiche le texte ; si celui-ci est vide, affiche *« pas de texte »*.

- [SPIP 1.8] introduit le filtre **`|?{`***`sioui,sinon`***`}`** qui est une version évoluée de `|sinon`. Il prend un ou deux paramètres :

  - *sioui* est la valeur à afficher à la place de l'élément filtré si celui-ci est non vide.
  - *sinon* est optionnel. C'est la valeur à afficher si l'élément filtré est vide. `[(#TEXTE|?`

`{#TEXTE,"pas de texte"})]` est équivalent à l'exemple donné pour le filtre `|sinon`.

- [SPIP 1.8] introduit un jeu de filtres pour faire des comparaisons avec des valeurs :

  - **`|=={`***`valeur`***`}`** et **`|!={`***`valeur`***`}`** permettent de vérifier, respectivement, l'égalité ou l'inégalité entre l'élément filtré et *valeur*. Par exemple : `<li [(#TITRE|=={édito}|?{'id="edito"',''})]>#TITRE</li>`

  - **`|>{`***`valeur`***`}`**, **`|>={`***`valeur`***`}`**, **`|<{`***`valeur`***`}`** et **`|<={`***`valeur`***`}`** comparent l'élément filtré (qui doit être numérique) avec une valeur numérique.
    Par exemple :

    [(#TOTAL_BOUCLE) [(#TOTAL_BOUCLE|>{1}|?{'articles','article'})] dans cette rubrique.]

  Remarque : De manière générale, tous les opérateurs de comparaison de php peuvent être utilisés comme filtres dans [SPIP 1.8].

Filtres de logos

- **fichier** [SPIP 1.4]. Affecté à un logo, ce filtre permet de récupérer directement le nom de fichier correspondant au logo.

- ||**autres filtres** Contrairement aux versions précédentes, [SPIP 1.4] permet de passer des filtres « maison » sur les logos : la logique est un peu tordue, car il fallait respecter la compatibilité avec SPIP 1.3. L'analyse se déroule comme suit :

  - si le premier « filtre » n'est pas un alignement, SPIP considère qu'il s'agit d'un URL et fait un lien du logo vers cette adresse ;
  - si le premier filtre est un alignement, SPIP considère que le deuxième « filtre » est un URL ;
  - les filtres suivants sont de vrais filtres au sens habituel (y compris des filtres « maison » déclarés dans mes_fonctions.php ;
  - pour appliquer un filtre quelconque sans mettre d'URL, il faut mettre deux barres. Par exemple : `<?php $logo = '[(#LOGO_RUBRIQUE||texte_script)]'; ?>` permet de récupérer le logo dans la variable php `$logo`, pour traitement ultérieur (voir ci-dessous pour la signification de `|texte_script`).

- [SPIP 1.8] introduit les filtres `hauteur` et `largeur` qui retournent les informations sur la taille (i.e. Hauteur et Largeur) de l'élément filtré si c'est une image.

Ces filtres n'ont qu'un intérêt moyen à être appliqués directement à un logo de document puisqu'il y a déjà #HAUTEUR et #LARGEUR à disposition pour les documents. Par contre, on peut les appliquer après le filtre `image_reduire` pour connaître la taille exacte de l'image réduite.

Plus généralement, on peut les appliquer sur n'importe quelles balises (ou filtre) retournant un balise HTML `<img ...>`.

- `|image_reduire{largeur,hauteur}` permet de forcer une taille maximale d'affichage des images et des logos.

Ce filtre s'utilise par exemple sur un logo d'article de la façon suivante :

[(#LOGO_ARTICLE|right||image_reduire{130})]

Dans cet exemple, logo de l'article apparaît aligné à droite, à une taille maximale de 130 pixels.

Depuis [SPIP 1.8.2], ce filtre peut prendre deux arguments : *largeur* et *hauteur*. Si l'un de ces deux arguments est égal à 0, SPIP ne tient compte que de l'autre et calcule cette dimension en conservant les proportions de l'image. De plus, ce filtre s'applique aussi à la balise `#TEXTE` et ce sont alors toutes les images que le rédacteur introduit dans le texte grâce aux raccourcis SPIP qui sont alors réduites.

Ainsi, par exemple,

[(#TEXTE|image_reduire{600,0})]

affiche toutes les images insérées dans le fil du texte à une largeur maximale de 600 pixels. Cela permet de préserver la mise en page même sans que le rédacteur ait à se soucier de la taille des images qu'il télécharge sur le site.

NB. Si l'option « création de vignettes » est activée dans la configuration du site, ces logos réduits seront des fichiers d'images spécifiques calculés automatiquement par le serveur (idéalement, avec l'extension GD2 installée sur le serveur), pour les formats acceptés par le serveur (avec GD2, habituellement, les formats JPG et PNG). Sinon, c'est une version complète de l'image qui est affichée, mais avec une taille d'affichage fixée directement en HTML.

> *Historique :* SPIP 1.7.1 introduit le filtre `|reduire_image`. Avec SPIP 1.9, celui-ci devient `|image_reduire` (les filtres de traitement d'image commencent ainsi tous par `|image_`).

Les filtres mathématiques

SPIP 1.9 introduit une série de filtres d'opérations mathématiques.

- `|plus{xx}`, `|moins{xx}` et `|mult{xx}` correspondent respectivement à l'addition, la soustraction et la multiplication.

- `|div{xx}` correspond à la division *non-euclidienne* ("après la virgule").

- `|modulo{xx}` correspond au reste de la division euclidienne par xx d'un nombre.

Par exemple `[(#COMPTEUR_BOUCLE|modulo{5})]` compte de 0 à 4 puis revient à 0 etc

De plus l'ensemble des fonctions mathématiques PHP peuvent être utilisées comme filtres.

Autres Filtres

- **traduire_nom_langue** s'applique à la balise #LANG et retourne un traduction du code de langue qu'elle retourne (fr, en, it, etc.) dans cette langue.

> *Remarque :* Les traductions des codes sont faites dans la langue que représente ce code et suivent les conventions d'écriture de cette langue.
>
> Ainsi « fr » sera traduit « français » en minuscule, alors que « es » sera traduit « Español » avec une majuscule.

- **alterner{*a*,*b*,*c*,...}** [SPIP 1.8.2] s'applique à une balise numérique (en général #COMPTEUR_BOUCLE ou #TOTAL_BOUCLE) et affiche l'argument correspondant à la valeur de cette balise . On peut ainsi alterner un affichage dans une boucle. Par exemple, `[(#COMPTEUR_BOUCLE|alterner{'white','yellow'})]` affichera « white » à la première itération de la boucle, « yellow » à la deuxième, « white » à la troisième, « yellow » à la quatrième, etc. Ainsi, on peut faire une liste d'article qui utilise une couleur différente pour les

lignes paires et impaires :

```
<B_lesarticles>
  <ul>
<BOUCLE_lesarticles(ARTICLES) {par titre}>
  <li style="background: [(#COMPTEUR_BOUCLE|alterner{'white','yellow'})]">#TITRE</li>
</BOUCLE_lesarticles>
  </ul>
</B_lesarticles>
```

- **inserer_attribut{*attribut*,*valeur*}** [SPIP 1.8.2] permet d'ajouter un attribut html dans une balise html générée par SPIP. Par exemple : `[(#LOGO_DOCUMENT|| inserer_attribut{'alt',#TITRE})]` va ajouter un attribut « alt » avec le titre du document dans la balise « img » du logo.

- **extraire_attribut{*attribut*}** [SPIP 1.8.2] est l'inverse du filtre précédent. Il permet de récupérer un attribut d'une balise html générée par SPIP. Par exemple, on peut trouver le chemin de la vignette générée par le filtre *image_reduire* : `<div style="background: url([(#LOGO_ARTICLE||image_reduire{90}|extraire_attribut{src})]) left;"]>#TEXTE</div>`

- **parametre_url{*parametre*,*valeur*}** [SPIP 1.8.2] est un filtre qui permet d'ajouter des paramètres dans une url générée par une balise SPIP. Si *valeur* vaut `''` le filtre supprime un paramètre actuellement dans l'url. Par exemple, la balise `#SELF` retourne l'url de la page actuelle, donc :

  - `[(#SELF|parametre_url{'id_article','12'})]` placera une variable id_article égale à 12 dans l'url,
  - `[(#SELF|parametre_url{'id_article',''})]` effacera l'id_article actuellement dans l'url.

On peut par exemple l'utiliser pour faire des boutons pour naviguer parmi les documents sur une page :

```
<BOUCLE_actuel(DOCUMENTS) {id_document}>
 #LOGO_DOCUMENT
 <ul>
 <BOUCLE_precede(DOCUMENTS) {par date} {age_relatif <= 0} {0,1} {exclus}>
 <li>
  <a href="[(#SELF|parametre_url{'id_document',#ID_DOCUMENT})]" title="précédent">
   [(#LOGO_DOCUMENT||image_reduire{70})]
  </a>
 </li>
 </BOUCLE_precede>
 <BOUCLE_suivant(DOCUMENTS) {par date} {age_relatif > 0} {0,1}>
 <li>
  <a href="[(#SELF|parametre_url{'id_document',#ID_DOCUMENT})]" title="suivant">
```

```
    [(#LOGO_DOCUMENT||image_reduire{70})]
   </a>
  </li>
  </BOUCLE_suivant>
  </ul>
</BOUCLE_actuel>
```

Filtres techniques

Ces filtres ont été introduits par [SPIP 1.4].

- **entites_html** transforme un texte en entités HTML, que l'on peut donc implanter dans un formulaire, exemple : `[<textarea>(#DESCRIPTIF|entites_html)</textarea>]`

- **texte_script** transforme n'importe quel champ en une chaîne utilisable en PHP ou Javascript en toute securité, exemple : `<?php $x = '[(#TEXTE|texte_script)]'; ?>`. Attention : utilisez bien le caractère `'` et non `"` : en effet, dans le second cas, si votre texte contient le symbole `$`, le résultat peut être catastrophique (affichage partiel, affichage d'autre chose, plantage php, etc.).

- **attribut_html** rend une chaîne utilisable sans dommage comme attribut HTML ; par exemple, si l'on veut ajouter un texte de survol au lien normal vers un article, on utilisera `<a href="#URL_ARTICLE"[ title="(#DESCRIPTIF|attribut_html|couper{80})"]>#TITRE</a>`. Dans les versions antérieures à SPIP 1.9.2, il fallait ajouter le filtre `|supprimer_tags`.

- **liens_absolus** [SPIP 1.8.2] s'applique sur une balise de texte et transforme tous les liens que celui-ci contient en liens absolus (avec l'url complète du site). Ce filtre est particulièrement utile dans des squelettes de fil rss par exemple.

- **url_absolue** [SPIP 1.8.2] marche de la même façon que le filtre précédent, mais s'applique à une balise qui retourne une url (par exemple #URL_ARTICLE).

- **abs_url** [SPIP 1.8.2] combine les deux balises précédentes et peut donc s'appliquer à un texte ou à une balise d'url.

- **form_hidden** SPIP 1.9 Si on fait un formulaire qui utilise comme action un lien comprenant des arguments (par exemple, quand on utilise la balise `#SELF` avec le type d'url par defaut), il faut remettre ces valeurs dans des champs *hidden* ; cette fonction calcule les champs en question. A utiliser par exemple :

```
<form action="#SELF">
[(#SELF|form_hidden)]
...
</form>
```

- Le filtre **|compacte** permet de réduire la taille d'un CSS ou d'un javascript en supprimant tout les commentaires. Le filtre prend en entrée le nom du fichier, et produit un nouveau fichier dont il renvoie le nom `<link rel="stylesheet" href="[(#CHEMIN{spip_style.css}|compacte)]" type="text/css" media="all" />`. (Filtre ajouté dans SPIP 1.9.2)

- Un nombre d'autres filtres techniques sont référencés ici.

Ajouter ses propres fonctions

Les filtres de SPIP sont des fonctions PHP qui reçoivent la balise sur laquelle ils sont appliqués en premier paramètre et retournent le texte à afficher. Vous pouvez utiliser directement les fonctions habituelles de PHP, mais également créer les vôtres, sur le modèle :

```
<?php
function mon_filtre($texte){
   $texte = (bidouillages en PHP) ...;
   return $texte;
}
?>
```

Afin de ne pas avoir à modifier des fichiers de SPIP (qui risqueraient d'être écrasés lors d'une prochaine mise à jour), vous pouvez installer vos fonctions personnelles dans un fichier `mes_fonctions.php` : si SPIP repère un fichier ayant ce nom, il l'inclut automatiquement.

> *Historique :* Dans les versions antérieures à [SPIP 1.9] ce fichier doit s'appeler `mes_fonctions.php3`
>
> De façon générale jusqu'à [SPIP 1.9] , les fichiers de spip avaient une extension en .php3 et non en .php.

Depuis [SPIP 1.8] il peut se situer :

- dans le dossier où sont stockés vos squelettes
- à la racine du site Mais **jamais** dans les deux à la fois.

Par exemple, ARNO* a développé le filtre `enlettres`, qui n'est pas inclus dans la distribution standard de SPIP. Ce filtre écrit un nombre en toutes lettres (`[(#DATE|annee|enlettres)]` = « `deux mille deux` ») ; ce filtre peut être téléchargé sur http://www.uzine.net/spip_contrib/a... ; il suffit de l'ajouter dans votre fichier `mes_fonctions.php` pour l'utiliser.

Filtres avec des paramètres

Depuis [SPIP 1.5], il est possible de passer des paramètres dans les filtres. La syntaxe est :

[(#BALISE|filtre{arg1, arg2}|...)]

Le filtre doit être défini de la manière suivante dans mes_fonctions.php :

function filtre($texte, $arg1='valeur par defaut1', $arg2='valeur par défaut 2')

{

....calculs....

return (une chaine de caractères);

}

On peut ainsi appeler n'importe quelle fonction php, ou s'appuyer sur des fonctions définies dans SPIP ou dans `mes_fonctions.php`, pour peu qu'elles respectent l'ordre des arguments (le texte à traiter doit être impérativement le premier argument). Par exemple, pour enlever les points à la fin d'un texte, on pourra faire : `[(#TEXTE|rtrim{'.?!'})]`.

Depuis [SPIP 1.8], les arguments des filtres peuvent être des balises (sans codes optionnels ni

filtres). Par exemple :
`[(#TOTAL_BOUCLE|=={#COMPTEUR_BOUCLE}|?{'Fin.',''})]`
Depuis [SPIP 1.8.2] on peut mettre un balise avec notation étendue en paramètre. Par exemple :
`[(#DESCRIPTIF|sinon{[(#CHAPO|sinon{#TEXTE}|couper{300})]})]`

Les boucles récursives

Mai 2001 — maj : Janvier 2007

Les boucles récursives offrent une fonctionnalité très puissante de mise en page de structure hiérarchique. Leur écriture est concise, mais leur utilisation demande une bonne maîtrise logique de l'enchaînement des boucles.

Pour construire une boucle récursive, il suffit d'indiquer dans son `TYPE` le nom d'une boucle contenant celle qu'on écrit :

```
<BOUCLEx ....>
....
<BOUCLEn(BOUCLEx)></BOUCLEn>
....
</BOUCLEx>
```

La boucle *n* fonctionne comme si l'on avait recopié l'intégralité de la boucle *x* (toutes les balises et le code HTML, ainsi que les textes conditionnels avant, après et alternatif) à l'endroit où l'on insère la boucle *n*. La boucle *n* étant à l'intérieur de la boucle *x*, on obtient un comportement récursif : la boucle *x* contient une boucle *n*, qui elle-même reproduit la boucle *x* qui contient la boucle *n*, et ainsi de suite, jusqu'à ce que la boucle *x* ne donne plus aucun résultat. Aucun critère ne figure dans la boucle *n*, le changement de contexte à chaque appel de la boucle *x* devant conduire les critères de celle-ci à ne plus trouver aucun élément.

Cette technique permet de créer notamment l'affichage des threads des forums. Une première boucle « fabrique » l'entrée des threads (les messages qui répondent directement à un article), une seconde boucle affiche les réponses à ces messages, et une boucle récursive provoque la récursivité sur cette seconde boucle :

```
<BOUCLE_forum(FORUMS){id_article}>
  #TITRE
  <B_reponses>
  <UL>
  <BOUCLE_reponses(FORUMS){id_parent}>
    <LI>#TITRE
    <BOUCLE_recursive(BOUCLE_reponses)>
    </BOUCLE_recursive>
    </LI>
  </BOUCLE_reponses>
  </UL>
  </B_reponses>
```

</BOUCLE_forum>

Plus généralement, cette fonctionnalité provoque un affichage graphiquement très clair de structures arborescentes, en particulier la hiérarchie des rubriques de votre site.

Remarque 1 : Le nouveau compilateur introduit par SPIP 1.8 considère que la notation `<BOUCLEn(BOUCLEx)>` à l'extérieur de la boucle *x* est vide de sens, ce qui lui permet d'atteindre l'optimalité du nombre de champs des requêtes SQL qu'il produit, pour tout type de boucles.
Cet emploi n'est pas une récursion mais une inclusion, fonctionnalité à laquelle répond la balise INCLURE, nettement préférable.

Remarque 2 : Dans l'état actuel (SPIP 1.9) du compilateur de SPIP, la séquence `<BOUCLEn(BOUCLEx)></BOUCLEn>` doit figurer au premier niveau de la boucle *x*, autrement dit la boucle *n* doit être immédiatement englobée par la boucle *x*, non par une autre boucle elle-même à l'intérieur de la boucle *x*.
La levée de cette restriction est à l'étude.

La gestion des dates

Mai 2003 — maj : 4 novembre

SPIP 1.6 introduit une série de critères et de balises pour mieux gérer les dates des articles. En voici une liste.

Afficher les dates

- `#DATE` est la date de mise en ligne. (Modifiable après la mise en ligne de l'article, de la brève, etc. La date d'une rubrique est celle de son élément le plus récent.)

- `#DATE_REDAC` est la date de première publication. (Modifiable à volonté, disponible sur les articles seulement.)

- `#DATE_MODIF` Apparue avec [SPIP 1.5], cette balise désigne la date de dernière modification de l'article.

- `#DATE_NOUVEAUTES` [SPIP 1.6] permet d'afficher la date du dernier envoi du mail présentant les nouveautés.

Formater les dates

Si les balises `#DATE...` sont utilisées sans filtre, alors toutes les informations de date sont affichées dans un format numérique (au format MySQL) : « `2001-12-01 03:25:02` ».

Les filtres `|annee`, `|mois`, `|jour`, `|heures`, `|minutes`, `|secondes`, mais aussi `|affdate`, `|date_relative`, `|nom_mois`, `|nom_jour`, `|saison`, etc. s'appliquent pour permettre tous les affichages habituels sous divers formats. Une liste complète des filtres pouvant être appliqués aux dates est fournie dans l'article Les filtres de SPIP.

Contexte de date

[SPIP 1.6] fournit à toutes les boucles un contexte de date. Si l'on se trouve à l'intérieur d'une boucle `(ARTICLES)`, `(BREVES)` ou `(RUBRIQUES)`, la date en question est la date de publication de l'article, de la brève ou la date de dernière modification de la rubrique.

Si en revanche on se trouve au premier niveau du squelette (c'est-à-dire en-dehors de toute boucle), la date considérée est la date du jour - à moins qu'on ait passé une date dans l'URL de la page (voir l'exemple plus bas).

Dans ce dernier cas, et pour les versions de php supérieures à 3.0.12, la date passée dans l'URL est analysée avec la fonction `strtotime` : ainsi `?date=2003`, `?date=2003/01` fonctionneront, mais aussi `date=-1year` (il y a un an), `?date=1march1970` (articles publiés le 1er mars 1970), etc.

Critère de date, d'âge, et d'âge relatif

Le critère `{age}` permet de sélectionner les articles en fonction de la durée qui sépare leur date de publication en ligne avec la date courante. Ainsi `{age<30}` permettra de ne pas afficher les articles âgés de plus de 30 jours.

L'`{age_relatif}` permet de comparer les dates de publication de deux articles : si l'on vient de sélectionner un article dans une boucle, une seconde boucle placée à l'intérieur de la première pourra demander les articles publiés dans la semaine qui précède celui-ci, via `{age_relatif<=7}{age_relatif>=0}`, etc.

Les critères `{age}` et `{age_relatif}` permettent de distinguer deux articles publiés le même jour (ce n'était pas le cas avant [SPIP 1.6]). On peut donc désormais programmer des boucles pour obtenir l'article « précédent » ou le « suivant » :

```
<BOUCLE_art(ARTICLES){id_article}>
<BOUCLE_precedent(ARTICLES){age_relatif>=0}{par date}{inverse}{1,1}>
précédent : <a href='#URL_ARTICLE'>#TITRE</a> #DATE
</BOUCLE_precedent>
<br />
<b>#TITRE</b> - #DATE
<br />
<BOUCLE_suivant(ARTICLES){age_relatif<0}{par date}{0,1}>
suivant : <a href='#URL_ARTICLE'>#TITRE</a> #DATE
</BOUCLE_suivant>
</BOUCLE_art>
```

Attention ! Malgré les apparences les comparaisons de date sont d'un maniement délicat : en effet, à cause des « dates floues » (un article publié un mois donné, sans que le jour soit précisé), le calcul de l'age_relatif peut donner la valeur zéro dans un sens, et pas dans l'autre ! D'où la dissymétrie des boucles présentées ci-dessus : dans un sens on cherche le « second plus récent » des articles `{age_relatif>=0}` (car le plus récent, avec la comparaison non-stricte, ne peut être que l'article lui-même) ; dans l'autre le plus âgé des articles publiés strictement plus tard.

Les critères `{jour_relatif}`, `{mois_relatif}` et `{annee_relatif}` fonctionnent

comme l'*age_relatif*, mais prennent en compte des dates arrondies au jour, au mois et à l'année respectivement ; par exemple, si l'URL comporte la variable `?date=2003-01-01`, la boucle suivante donnera « tous les les articles du mois de mars 2003 »

```
<h3>Articles de [(#DATE|nom_mois)] [(#DATE|annee)] :</h3>
<BOUCLE_blog(ARTICLES){mois_relatif=0}{par date}{"<br />"}>
<a href='#URL_ARTICLE'>#TITRE</a> ([(#DATE|jour)]/[(#DATE|nom_mois)]
</BOUCLE_blog>
```

La date de rédaction antérieure

Si vous avez activé l'utilisation des dates de publication antérieure, la plupart des critères présentés ci-dessus fonctionnent : il suffit d'ajouter `_redac` au critère. Ainsi `{age_redac>365}` affichera les articles dont la date de publication antérieure remonte à plus d'un an.

Si une boucle sélectionne un article dont la date_redac est définie, une boucle interne comportant le critère `{annee_relatif_redac=0}` ira chercher les articles dont la date de publication antérieure appartient à la même année.

Un exemple de sommaire de site trié par date

A titre d'exemple, voici comment on peut afficher tous les articles d'un site, triés par mois de publication :

```
<BOUCLE_articlem(ARTICLES){par date}{inverse}>
<BOUCLE_premierdumois(ARTICLES){id_article}{doublons}>
<b> [(#DATE|nom_mois|majuscules)] [(#DATE|annee)] </b>
<ul>
  <li><a href="#URL_ARTICLE">[(#TITRE|couper{50})]</a> - [(#DATE|jour)]/[(#DATE|
mois)]</li>
</BOUCLE_premierdumois>
  <BOUCLE_MOIS(ARTICLES) {mois_relatif=0}{doublons}{par date}{inverse} >
    <li><a href="#URL_ARTICLE">[(#TITRE|couper{50})]</a> - [(#DATE|jour)]/[(#DATE|
mois)]</li>
  </BOUCLE_MOIS>
</ul>
</BOUCLE_articlem>
```

Quelques exemples de boucles

Classer selon la date ou selon un ordre imposé

Septembre 2003 — maj : Novembre 2007

Nous souhaitons trier les articles de façon différenciée, de cette façon :

- dans certaines rubriques, les articles sont publiés les uns à la suite des autres ; on veut donc les présenter selon l'ordre chronologique : les plus récents en début de liste, les plus anciens en fin de liste ;
- dans d'autres rubriques, on souhaite afficher les articles dans un ordre précis, en les numérotant ; sur le site public, on veut donc les présenter selon cet ordre indiqué par la numérotation ;
- enfin, dans les autres rubriques, les articles doivent être classés dans l'ordre chronologique, *sauf certains* que l'on souhaite placer en tête de liste dans un ordre précis.

Il y a plusieurs méthodes pour réaliser ce classement. Notez bien que les deux méthodes présentées ci-après sont **valables tant pour ordonner les articles que les autres objets de SPIP** (rubriques, brèves, etc.).

Méthode simple

Rappel : Pour classer les articles selon un ordre imposé, on numérote leurs titres dans l'espace privé, avec un numéro suivi d'un point et d'un espace : « 1. Premier article », « 2. Deuxième article », etc.
Dans les squelettes, pour ne pas afficher ces numéros sur le site public, on passe le filtre `|supprimer_numero` sur la balise `#TITRE`, et on utilise le critère `{par num titre}` sur la boucle ARTICLES pour ordonner selon les numéros indiqués.

Pour classer selon un ordre imposé ET/OU selon la date, on écrira donc ceci (à l'intérieur d'une boucle RUBRIQUES) :

```
<BOUCLE_articles(ARTICLES) {id_rubrique} {par num titre}{!par date}>
[(#TITRE|supprimer_numero)]
</BOUCLE_articles>
```

Ainsi les articles de la rubrique seront tout d'abord classés par numéros, puis ceux portant un numéro identique seront classés par date inverse.

Remarque importante : Cela est tout à fait satisfaisant tant que, dans une rubrique donnée, tous les articles sont numérotés ou qu'aucun ne l'est. Par contre, si une rubrique contient des articles non numérotés, sauf un par exemple, celui-ci s'affichera en dernier. En effet, **non numérotés, les articles sont considérés comme ayant zéro pour numéro**. Pour éviter ce classement, on emploie alors la méthode suivante.

Méthode fine

À l'intérieur d'une boucle RUBRIQUES, nous allons effectuer le test suivant : est-ce qu'il existe, dans cette rubrique, au moins un article dont le titre commence par un numéro suivi d'un point ?

```
<BOUCLE_test_numero(ARTICLES) {id_rubrique} {titre==^[0-9]+\.} {0,1}>
```

Il existe un article numéroté dans cette rubrique

</BOUCLE_test_numero>

Il n'y a pas d'article numéroté

<//B_test_numero>

Le critère intéressant ici est : `{titre==^[0-9]+\.}`

Il s'agit d'une sélection sur le `titre`, selon une *expression régulière* (« == » indique une sélection selon une expression régulière) dont la syntaxe est : au début du `titre` (« ^ » indique le début de la chaîne testée), il y a un ou plusieurs (« + » indique « au moins un des caractères précédents ») caractères compris entre 0 et 9 (« `[0-9]` » signifie « caractères compris entre 0 et 9 inclus »), suivis du caractère « point » (« `\.` »).

Notez enfin qu'on ne sélectionne qu'un seul article ainsi numéroté (`{0,1}`) ; de cette façon, l'intérieur de la boucle ne sera effectué qu'une seule fois. De plus, il suffit qu'il existe un seul article numéroté pour provoquer l'affichage d'une liste par ordre « numéroté ».

Cette boucle affiche ainsi « Il existe un article numéroté dans cette rubrique » s'il y a au moins un article dont le titre est précédé d'un numéro dans la rubrique, et « Il n'y a pas d'article numéroté » sinon.

Il suffit maintenant d'installer à la place de ces mentions des boucles d'affichage des articles selon l'ordre de présentation désiré :

<BOUCLE_test_numero(ARTICLES) {id_rubrique} {titre==^[0-9]+\.} {0,1}>

<BOUCLE_ordre_numeros(ARTICLES) {id_rubrique} {par num titre}>

[(#TITRE|supprimer_numero)]

</BOUCLE_ordre_numeros>

</BOUCLE_test_numero>

<BOUCLE_ordre_date(ARTICLES) {id_rubrique} {par date}{inverse}>

#TITRE

</BOUCLE_ordre_date>

<//B_test_numero>

Ainsi les articles numérotés de la rubrique sont affichés en premier, classés par numéros, suivis de ceux ne portant pas de numéro, classés du plus récent au plus ancien.

Depuis SPIP 1.9, il est possible de faire plus simple :

<BOUCLE_Classement(ARTICLES){par num titre, titre}>

[(#TITRE|supprimer_numero)]

</BOUCLE_Classement>

Classera les articles par ordre de titre puis par numéro.

Trier par ordre alphabétique, sauf un article qu'il faut afficher en premier

Septembre 2003 — maj : Février 2007

Mettons que vous voulez présenter une série d'articles par ordre alphabétique, sauf le prologue que vous voulez légitimement afficher au début de la liste...

Il existe une astuce pour « duper » le tri effectué par SPIP : il suffit d'ajouter un espace au début du titre de l'article (par exemple, transformer « Prologue » en « Prologue »). Le tri alphabétique pensera alors que ce titre doit « passer avant les autres », et l'affichera en premier. Cependant l'espace supplémentaire du titre sera ignoré par le navigateur Web, et ne provoquera pas de décalage disgrâcieux dans la mise en page.

Notez bien : l'astuce ne fonctionnera que si vous utilisez un classement alphabétique sur le titre des articles (c'est-à-dire le critère `{par titre}`). Tout autre classement ne fonctionnera pas.

Remarque : cela s'applique également aux rubriques, brèves, sites référencés, etc.

Plusieurs logos pour un article

Août 2003 — maj : Janvier 2007

Il est fréquent, pour rythmer la navigation sur son site, de vouloir utiliser des logos différents (notamment de tailles différentes) pour un même article en fonction de l'endroit où il apparaît.

Par exemple, utiliser un « gros » logo sur la page d'accueil du site qui permette de bien mettre en valeur l'article principal du moment, et un « petit » logo pour la navigation générale du site.

Rappelons que depuis SPIP 1.7, SPIP 1.7.2 , les webmestres disposent d'une fonction `image_reduire` qui permet de créer différentes versions (de différentes tailles) d'un *même* logo d'article. Mais il faut une autre méthode pour utiliser des logos *différents* pour un même article.

Les webmestres ont créé des méthodes personnelles basées sur l'utilisation différenciée du logo « normal » et du logo « pour survol ». Par exemple : le logo « normal » utilisé comme « petit logo » (appelé par la balise `#LOGO_ARTICLE_NORMAL`), et sur le sommaire, le logo « pour survol » (appelé par la balise `#LOGO_ARTICLE_SURVOL`) pour afficher la « grande » version du logo. Cette méthode complique souvent le code des squelettes, et interdit l'utilisation habituelle des logos « avec survol » que SPIP fournit automatiquement. Elle est de plus d'une souplesse très limitée.

Depuis SPIP 1.4 , il est possible de joindre des *documents* aux articles (et, accessoirement, aux rubriques). Nous allons expliquer ci-dessous comment utiliser ces documents joints pour créer plusieurs logos pour un même article.

Principe général

- Nous continuerons à utiliser les deux logos de l'article pour afficher les logos « normaux » (ceux qui apparaissent dans les liens de navigation les plus fréquents, par exemple sur les pages des rubriques), ce qui permet de conserver la simplicité de gestion des logos avec SPIP et la gestion automatique du survol (on revient à l'utilisation évidente de la balise `#LOGO_ARTICLE`, ou de `#LOGO_ARTICLE_RUBRIQUE`).

- Nous déciderons de joindre aux articles un document (généralement une image aux formats GIF,

JPEG ou PNG) auquel nous donnerons systématiquement le même nom. Il nous suffira d'afficher ce document (en l'appelant par son nom) à la place du logo « normal » lorsque nous le désirerons.

- Cette méthode permet ainsi de créer autant de logos différents que nécessaire pour un même article (pas seulement un grand logo et un petit logo, mais pourquoi pas une image pixelisée avec un travail typographique élaboré pour afficher le titre, etc.).

- Nous verrons de plus que, grâce aux boucles de SPIP, on pourra très facilement dans les squelettes déterminer si un tel « grand » logo (document portant le nom choisi par nous) est présent, et agir en conséquence (afficher à la place le logo « normal », du texte spécifique, ou carrément un autre élément graphique).

- Miracle de la technologie moderne, des formats propriétaires et de l'accès par haut-débit, de telles versions spécifiques des logos, étant des documents joints, pourront être d'un autre format que des images. On pourra ainsi afficher, en tant que « grands » logos, des animations Flash ou Shockwave, des animations vidéo...

Mise en place des documents et choix des noms

- Nous décidons (arbitrairement, mais faites selon vos besoins) que les documents joints utilisés en tant que « gros » logo seront tous intitulés « spip_logo » ; ce document « spip_logo » sera affiché sur la page du sommaire de notre site à la place du logo normal.

Nous utiliserons d'autres noms dans la suite de cet exemple pour créer des effets plus fins : décidons immédiatement qu'ils auront tous des noms commençant par « spip_... ». (Cela nous permettra, dans l'affichage habituel des documents joints à un article d'exclure tous les documents dont le nom commence par « spip_... ». De cette façon, l'utilisation de documents en tant que logos alternatifs n'interfèrera pas avec l'affichage, par exemple, d'un portfolio.)

- Sur un article publié en ligne (de façon à pouvoir bidouiller nos squelettes en les testant), nous installons simplement :

- un logo normal ; nous pouvons, si nous le voulons, installer une version de l'image « pour survol » pour la gestion automatique du changement d'image lors du survol avec la souris ;

Le logo « normal »

Le logo est installé classiquement. A priori, il s'agit d'une image de taille modeste.

- un « document joint » (par le pavé « JOINDRE UN DOCUMENT ») ; pour faire simple, installons une image (JPEG, GIF, PNG) ; une fois ce document installé (« uploadé »), nous lui donnons pour titre « spip_logo ». Voilà, c'est la seule manip nécessaire... SPIP affiche ce document en bas de la page de l'article dans l'espace privé, en donnant son titre (« spip_logo ») et en indiquant ses dimensions en pixels.

Le document « spip_logo »

Le seul impératif est de donner à ce document le titre « spip_logo ». Il est inutile d'installer une vignette de prévisualisation.
Dans le cas d'un document multimédia (Flash, Shockwave...), il faut indiquer à la main ses dimensions en pixels.

- Nous décidons de l'usage de ce document intitulé « spip_logo » : il sera affiché sur la page d'accueil du site à la place du logo normal du dernier article publié. De cette façon, la page de Une du site peut afficher une « grande » image pour mettre en valeur l'article en vedette.

Afficher « spip_logo » en Une du site

- Commençons par insérer une boucle toute simple pour afficher le dernier article publié sur le site et son logo « normal ». (Dans tous les exemples qui suivent, le code HTML est réduit à son strict minimum ; à vous d'enrober cela avec la mise-en-pages graphique qui vous convient.)

```
<BOUCLE_article_vedette(ARTICLES){doublons}{par date}{inverse}{0,1}>
 #LOGO_ARTICLE
 <h1>#TITRE</h1>
</BOUCLE_article_vedette>
```

Cette boucle très simple affiche le premier article (`{0,1}`) parmi tous les ARTICLES, sélectionnés par date de publication (`{par date}`) du plus récent au plus ancien (`{inverse}`). On affiche donc bien le dernier article publié sur le site. À l'intérieur de la boucle, on affiche le logo de l'article suivi du titre de l'article.

- Nous avons dit que nous voulions afficher, à la place du logo normal, le document joint à cet article dont le titre est « spip_logo ». Le code devient :

```
<BOUCLE_article_vedette(ARTICLES){doublons}{par date}{inverse}{0,1}>
 <BOUCLE_logo_article_vedette(DOCUMENTS){id_article}{titre=spip_logo}>
  [(#EMBED_DOCUMENT)]
 </BOUCLE_logo_article_vedette>


 <h1>#TITRE</h1>
</BOUCLE_article_vedette>
```

La `BOUCLE_logo_article_vedette` sélectionne parmi les documents joints à cet article (`{id_article}`) celui dont le titre est « spip_logo » (`{titre=spip_logo}`). À l'intérieur de la boucle, on demande l'affichage de ce document joint (`#EMBED_DOCUMENT`).

L'usage de `#EMBED_DOCUMENT` dans les squelettes permet d'insérer, via le système de boucles, directement le document à l'intérieur de la page. SPIP se charge de créer le code correspondant à des images ou à des fichiers multimédia.

- Inconvénient : si l'article n'a pas de document joint intitulé « spip_logo », le code précédent n'affiche que le titre. On va donc effectuer une nouvelle modification, qui permet d'afficher le logo « normal » de l'article s'il n'existe pas de document joint pour cet usage. *Notez bien :* une fois cette méthode comprise, il n'y aura plus d'autres subtilités pour réaliser tous les effets suivants...

```
<BOUCLE_article_vedette(ARTICLES){doublons}{par date}{inverse}{0,1}>
 <BOUCLE_logo_article_vedette(DOCUMENTS){id_article}{titre=spip_logo}>
  [(#EMBED_DOCUMENT)]
 </BOUCLE_logo_article_vedette>
  #LOGO_ARTICLE
 <//B_logo_article_vedette>


 <h1>#TITRE</h1>
</BOUCLE_article_vedette>
```

Nous avons tout simplement ajouté l'appel au logo « normal » (`#LOGO_ARTICLE`) en texte alternatif (ce qui se trouve avant `<//B...>` d'une boucle s'affichant si la boucle ne fournit pas de résultat - ici, s'il n'y a pas de document joint à l'article portant le titre « spip_logo »).

Nous avons obtenu le résultat désiré :
- s'il existe un document joint associé à l'article auquel nous avons donné le titre « spip_logo », il est directement affiché ;
- *sinon*, c'est le logo « normal » qui est affiché.

Exclure ces documents spécifiques de l'affichage normal des documents joints

Dans le squelette des articles, on affiche les documents joints grâce à la `BOUCLE_documents_joints`, dont les critères essentiels sont :

```
<BOUCLE_documents_joints(DOCUMENTS){id_article}{mode=document}
{doublons}>
```

On appelle les DOCUMENTS liés à cet article (`{id_article}`), qui sont bien des documents joints et non des images (`{mode=document}`) et qu'on n'a pas déjà affichés à l'intérieur du texte de l'article en utilisant le raccourci `<EMBxx>` (`{doublons}`).

Modifions ce code pour interdire l'affichage, dans cette boucle (qui est une sorte de « portfolio »), des documents dont le nom commence par « spip_... » (on ne veut pas afficher ici le « gros » logo utilisé en page de Une du site) :

<BOUCLE_documents_joints(DOCUMENTS){id_article}{titre!==^spip_}{mode=document}
{doublons}>

Le critère `{titre!==^spip\_}` est une expression régulière, dont la syntaxe est très codifiée. On sélectionne les documents dont le titre n'est pas formé ainsi (le `!==` signifie « qui ne correspond pas à l'expression régulière ») : les premiers caractères (le symbole ^ indique le début de la chaîne de caractères) sont « spip » suivi de « _ » (dans la syntaxe des expressions régulières, « _ » indique le caractère « _ », de la même façon que « \. » indique le caractère « . »).

Ce critère sélectionne donc les documents joints dont le titre *ne commence pas* par « spip_ ».

L'exposé du principe général est terminé, vous avez largement de quoi vous amuser avec ça sur votre propre site. Les exemples suivants n'en sont que des variations.

Afficher toujours un gros logo en haut de page

Je décide, toujours de manière arbitraire, qu'il doit toujours y avoir une grosse image en haut de page de mes articles. Il s'agit d'un choix graphique de ma part : pour assurer l'unité graphique de mon site, j'affiche en haut de page une version de grand format liée à l'article (une variation du principe du « grand » logo) et, à défaut, une image stockée ailleurs sur mon site.

- Toujours le même principe : je joins à mon article un document dont je fixe le titre à « spip_haut ». (Pour éviter que ce document ne s'affiche dans le « portfolio » de la `BOUCLE_documents_joints` précédente, je fais commencer son titre par « spip_... ».)

Dans mon squelette des articles, j'affiche simplement en haut de page ce document :

<BOUCLE_doc_haut(DOCUMENTS){id_article}{titre=spip_haut}>

#EMBED_DOCUMENT

</BOUCLE_doc_haut>

Comme dans l'exemple précédent, j'affiche le document, lié à l'article de cette page, et dont le titre est « spip_haut ». Fastoche.

- Comme dans le premier exemple, je pourrais décider d'afficher le logo de l'article si ce document n'existe pas :

<BOUCLE_doc_haut(DOCUMENTS){id_article}{titre=spip_haut}>

#EMBED_DOCUMENT

</BOUCLE_doc_haut>

#LOGO_ARTICLE

<//B_doc_haut>

- Mais ça n'est pas le résultat désiré. Je veux, pour des impératifs graphiques, toujours afficher une

grande image aux dimensions prédéterminées.

Je vais donc (toujours un choix arbitraire de ma part) créer des images de substitution, utilisées « par défaut », au cas où un article n'aurait pas d'image en propre. Ces images répondent à mes impératifs graphiques (par exemple, elles ont toutes les mêmes dimensions que les documents que j'utilise d'habitude en « spip_haut »).

Sur mon site, je créée une rubrique pour accueillir « en vrac » ces documents de substitution. J'active les documents associés aux rubriques. (Je peux aussi créer un article qui accueillerait tous ces documents, et ainsi ne pas activer les documents joints aux rubriques. Le code serait à peine différent.) Admettons que cette rubrique porte le numéro 18 (c'est SPIP qui va fixer automatiquement le numéro de la rubrique lors de sa création). J'installe tous mes documents de substitution à l'intérieur de cette rubrique numéro 18. (Il est inutile de leur donner un titre.)

Pour appeler, au hasard, un document installé dans cette rubrique, il me suffit d'invoquer les critères suivants :

<BOUCLE_doc_substitution(DOCUMENTS){id_rubrique=18}{0,1}{par hasard}>

#EMBED_DOCUMENT

</BOUCLE_doc_substitution>

(Notez bien : le critère `{par hasard}` ne signifie pas que l'image sera différente à chaque visite de la page, mais qu'elle sera sélectionnée au hasard à chaque recalcul de la page.)

On prendra soin, dans la navigation du site, d'interdire l'affichage de la rubrique 18, qui n'a pas besoin d'être présentée aux visiteurs. Le critère `{id_rubrique!=18}` *fera l'affaire.*

- Pour terminer la mise en place du dispositif, il nous suffit d'insérer cette boucle affichant un document de substitution pris au hasard dans la rubrique 18 en tant que texte alternatif à notre `BOUCLE_doc_haut` (à la place du `#LOGO_ARTICLE`) :

<BOUCLE_doc_haut(DOCUMENTS){id_article}{titre=spip_haut}>

#EMBED_DOCUMENT

</BOUCLE_doc_haut>

<BOUCLE_doc_substitution(DOCUMENTS){id_rubrique=18}{0,1}{par hasard}>

#EMBED_DOCUMENT

</BOUCLE_doc_substitution>

<//B_doc_haut>

Afficher un titre graphique

Toujours sur le même principe, nous allons afficher une version graphique du titre de l'article. Il s'agit d'une image réalisée avec un logiciel de dessin, où apparaît , avec une typographie particulièrement soignée (effets de relief, de dégradés de couleurs, avec des polices de caractère exotiques...) le titre de l'article.

- Décrétons qu'il s'agira d'un document joint associé à l'article, que nous titrerons « spip_titre ».

- Pour appeler ce document, à l'endroit où doit être affiché le `#TITRE` de l'article, installons la boucle désormais connue :

<BOUCLE_doc_titre(DOCUMENTS){id_article}{titre=spip_titre}>

#EMBED_DOCUMENT

</BOUCLE_doc_titre>

Notez à nouveau que cette méthode permet non seulement d'utiliser une image pour afficher le titre, mais aussi une animation Flash, un film... Dans ces cas, il vous faudra indiquer à la main pour votre document joint quelles sont ses dimensions en pixels.

- Complétons le dispositif : s'il n'existe pas de document joint portant le titre « spip_titre », il faut afficher en tant que texte HTML classique les informations nécessaires :

<BOUCLE_doc_titre(DOCUMENTS){id_article}{titre=spip_titre}>

#EMBED_DOCUMENT

</BOUCLE_doc_titre>

[(#SURTITRE|majuscules)]

#TITRE

[(#SOUSTITRE|majuscules)]

[(#DATE|nom_jour)] [(#DATE|affdate)]</p>

<//B_doc_titre>

* * *

Signalons un dernier avantage à cette méthode...

Elle permet de faire par la suite encore évoluer radicalement l'interface graphique de votre site. Sans avoir à supprimer un par un tous les documents intitulés « spip_haut », « spip_titre »..., il vous suffit de créer de nouveaux squelettes qui ne les appellent tout simplement pas.

Par exemple, si les documents « spip_haut » étaient précédemment tous conçus pour une largeur de 450 pixels, et que la nouvelle interface graphique requiert des images d'une largeur de 600 pixels, vous n'aurez pas besoin de modifier un par un tous vos fichiers « spip_haut ». Il vous suffit, dans les squelettes, de ne plus faire appel aux documents intitulés « spip_haut », mais d'utiliser un nouveau nom (par exemple « spip_large ») et d'installer au fur et à mesure vos nouvelles versions de documents en les titrant « spip_large ». Pendant la transition, il n'y aura pas d'incohérences graphiques.

Les plus jetés d'entre vous peuvent même imaginer toutes sortes de tests sur le type de document (`{extension=...}`) ou sur leur taille (`{largeur=...}`, `{hauteur=...}`) (aucun PHP nécessaire) pour réaliser des interfaces selon ces tests (par exemple, une certaine interface graphique si « spip_haut » fait 450 pixels de largeur, et une autre s'il fait 600 pixels de largeur...).

Afficher les derniers articles de vos rédacteurs par rubrique

Mai 2002 — maj : Novembre 2007

Par défaut SPIP vous propose une page auteur qui vous permet de montrer la liste des auteurs/rédacteurs participant à votre site, ainsi que leurs dernières contributions.

Mais un problème vient à se poser quand vous avez plusieurs rédacteurs et que ceux-ci participent activement à votre site. Cela finit par être une page à rallonge.

Cependant il existe un moyen de montrer les dernières contributions de vos auteurs/redacteurs et ce pour chacun d'eux.

Comment procéder ?

Tout d'abord, on va créer un fichier :

myauteur.html

Création du fichier myauteur.html

Dans le fichier mettre le code suivant :

- Juste après la balise `<body>`, mettre

<BOUCLE_principale(AUTEURS){id_auteur}{unique}>

- Juste avant la balise `</body>`, mettre

</BOUCLE_principale>

- Dans le corps de la page HTML, voici le code à installer (on ne peut déterminer une rubrique car par défaut l'auteur n'est pas associé à une rubrique mais à un article, le code peut parait biscornu mais on va donc retrouver la rubrique par rapport à l'article) :

Code pour le dernier article

```
<B_appel_article>


Dernier article écrit par <BOUCLE_nom_auteur(AUTEURS)
{id_auteur}>[(#NOM)]</BOUCLE_nom_auteur><br>


<BOUCLE_appel_article(ARTICLES){id_auteur}>
        <BOUCLE_appel_rubrique_article(RUBRIQUES){id_rubrique}{par titre}{doublons}>
                [(#TITRE|majuscules)]
                <ul>
                <BOUCLE_rappel_article(ARTICLES){id_rubrique}{par date}{inverse}{doublons}
{0,15}>
                   <li><a href="#URL_ARTICLE">[(#TITRE)<br></a>]
                </BOUCLE_rappel_article>
                </ul>
        </BOUCLE_appel_rubrique_article>
</BOUCLE_appel_article>
</B_appel_article>


Cette auteur n'a pour l'instant écrit aucun article


<//B_appel_article>
```

Code pour article choisi au hasard

```
<B_appel_article>
```

Dernier article écrit par <BOUCLE_nom_auteur(AUTEURS)
{id_auteur}>[(#NOM)]</BOUCLE_nom_auteur>

<BOUCLE_appel_article(ARTICLES){id_auteur}>
<BOUCLE_appel_rubrique_article(RUBRIQUES){id_rubrique}{par titre}{doublons}>
[(#TITRE|majuscules)]

<BOUCLE_rappel_article(ARTICLES){id_rubrique}{par hasard}{doublons}
{0,15}>
[(#TITRE)
]
</BOUCLE_rappel_article>

</BOUCLE_appel_rubrique_article>
</BOUCLE_appel_article>
</B_appel_article>

Cette auteur n'a pour l'instant écrit aucun article

<//B_appel_article>

Et enfin

Maintenant, il faut configurer votre page auteur (page où vous énumérez vos différents auteurs) pour que, en cliquant sur le lien auteur, celui-ci, dirigera vers la page *myauteur* où sera inscrit les derniers articles écrits par l'auteur.

Le lien devra être écrit de la manière suivante :

```
<a href="#URL_PAGE{myauteur, id_auteur=#ID_AUTEUR}">nom du
lien</a>
```

Afficher des éléments par lignes dans un tableau

Avril 2002 — maj : Juillet 2007

Soit à créer un tableau de trois colonnes contenant les titres des articles d'une rubrique, le nombre de lignes dépendant du nombre total d'articles ; sur le principe :

article 1 article 2 article 3
article 4 article 5 article 6
article 7 article 8 article 9

Il faut évidemment utiliser une boucle Articles, mais la difficulté réside dans le placement de balises `tr` ouvrante et fermante tous les trois passages dans la boucle. L'astuce consiste à utiliser la balise `#COMPTEUR_BOUCLE` et le filtre `alterner`. Cette balise indique le nombre de passages déjà

effectués dans la boucle : si le reste de sa division par 3 vaut 0 il faut produire une balise `tr` ouvrante au début du code, s'il vaut 2, il faut produire une balise `tr` fermante à la fin.

Pour cela, point n'est besoin d'opérations arithmétiques dans le texte du squelette. Il suffit d'utiliser le filtre `alterner`, qui accepte un nombre quelconque d'arguments, disons *n*, le premier devant être un entier, appelons-le `i`. Ce filtre calcule le reste *k* de la division de *n-1* par *i* et retourne alors son *k*+unième argument. Il suffit donc de lui donner 3 arguments vides sauf un pour ne produire les balises `tr` qu'aux passages appropriés.

Si l'on veut un code HTML irréprochable, il faut produire une séquence de `<td></td>` après la boucle pour compléter la dernière ligne du tableau lorsque le nombre total d'articles n'est pas divisible par 3. Il suffit à nouveau d'utiliser le filtre `alterner`, mais appliqué cette fois à la balise `#TOTAL_BOUCLE`.

Ce qui donne :

```
<B_ligne>
<table>
<BOUCLE_ligne (ARTICLES) {id_rubrique} {par titre}>[
(#COMPTEUR_BOUCLE|alterner{'<tr>',''})]
 <td width="33%">
 <a href="#URL_ARTICLE">#TITRE</a>
 </td>[
(#COMPTEUR_BOUCLE|alterner{'','','</tr>'})
]</BOUCLE_ligne>
[(#TOTAL_BOUCLE|alterner{
   '<td></td><td></td></tr>','<td></td></tr>', ''})]
</table>
</B_ligne>
```

Pour améliorer la lisibilité, on peut colorer différemment les lignes paires et impaires en mettant un attribut `style` dans la balise `tr` ouvrante. Il faut alors remplacer `'<tr>'` ci-dessus par une nouvelle utilisation de `#COMPTEUR_BOUCLE|alterner`, l'argument d'un filtre pouvant lui-même utiliser des balises et des filtres. Le squelette suivant donnera ainsi le tableau visualisé au début de cet article :

```
<B_ligne>
<table>
<BOUCLE_ligne (ARTICLES) {id_rubrique} {par titre}{0,7}>[
(#COMPTEUR_BOUCLE|
 alterner{[(#COMPTEUR_BOUCLE|
  alterner{'<tr style="background: #eee;">',
    '<tr style="background: #ddd;">'})],
     ''})]
```

```
 <td width="33%">
 <a href="#URL_ARTICLE">#TITRE</a>
 </td>[
(#COMPTEUR_BOUCLE|alterner{'','</tr>'})
]</BOUCLE_ligne>
[(#TOTAL_BOUCLE|alterner{'<td></td><td></td></tr>','<td></td></tr>', ''})]
</table>
</B_ligne>
```

Le même type de boucle, en remplaçant l'appel du titre par le logo (avec la balise #LOGO_ARTICLE), permet d'afficher une galerie où chaque logo d'article donne un aperçu (dont la taille sera de préférence fixée afin d'avoir une belle mise en page), et le texte de l'article contient la ou les œuvres exposées.

P.-S.

Pour obtenir ce résultat, la version initiale de cet article utilisait les boucles récursives de manière détournée. Ce détournement ne sera plus possible à terme dans SPIP, et est en outre moins efficace que la méthode ci-dessus qui n'effectue qu'une seule requête au serveur SQL.

Ne pas afficher les articles publiés depuis plus d'un an

Avril 2002 — maj : Janvier 2007

Cela s'effectue avec le critère « *age* », qui est l'âge de l'article (calculé depuis sa date de mise en ligne dans l'espace public) en nombre de jours.

Ainsi pour conserver tous les articles de moins d'un an dans la rubrique courante, nous utiliserons le critère `{age < 365}` :

```
<B_articles_recents>
 <ul>
  <BOUCLE_articles_recents(ARTICLES) {id_rubrique} {age < 365} {par titre}>
  <li>#TITRE</li>
  </BOUCLE_articles_recents>
 </ul>
</B_articles_recents>
```

Pour prendre en compte l'âge vis-à-vis de la date de première publication au lieu de la date de mise en ligne, il faudra utiliser le critère « *age_redac* » au lieu de « *age* ». L'âge est indiqué en nombre de jours. Voir : « La gestion des dates ».

Notons que cette manipulation est possible avec tous les types de données auxquels est associée une date (brèves, forums...).

Présenter les résultats d'une recherche par secteurs

Avril 2002 — maj : Septembre 2003

Il suffit d'inclure la boucle de recherche dans une boucle de type rubriques sélectionnant les rubriques de premier niveau ; dans la boucle de recherche, on ajoute alors le critère « *id_secteur* » pour se limiter au secteur courant.

```
<BOUCLE_secteurs(RUBRIQUES){racine}>

<B_recherche>
 <b>#TITRE</b>
 <ul>
<BOUCLE_recherche(ARTICLES){recherche}{id_secteur}{par points}{inverse}{0,5}>
 <li><a href="#URL_ARTICLE">#TITRE </a>
</BOUCLE_recherche>
 </ul><hr>
</B_recherche>

</BOUCLE_secteurs>
```

On remarquera que le titre du secteur n'est affiché que si la recherche a donné des résultats pour ce secteur. D'autre part, pour chaque secteur on n'affiche que les cinq articles les mieux classés, par ordre décroissant de pertinence.

Attention cependant, comme la recherche est effectuée autant de fois qu'il y a de secteurs, le calcul risque d'être ralenti.

Afficher le nombre de messages répondant à un article

Avril 2002 — maj : Février 2007

Pour afficher le nombre de messages du forum lié à un article, nous devons créer une boucle de type FORUMS, liée à un article, de façon à compter son nombre de résultats.

À première vue, il est très simple de connaître le nombre d'éléments d'une boucle : il suffit d'utiliser la balise : `#TOTAL_BOUCLE`. Mais son usage est un poil acrobatique.

Bien sélectionner tous les messages

Première subtilité : nous voulons *tous* les messages des forums liés à l'article, en comptant les réponses aux messages, les réponses aux réponses...

Une simple boucle de type : `<BOUCLE_forum(FORUMS) {id_article}>` sélectionne uniquement les messages qui répondent directement à l'article. Pour accéder aux réponses à ces messages, on inclut habituellement une seconde boucle à l'intérieur de celle-ci, mais ce n'est pas ce que nous souhaitons faire ici.
Pour pouvoir tous les compter, nous voulons que la boucle sélectionne absolument tous les messages du forum lié à l'article, sans tenir compte de leur hiérarchie. Pour cela, il faut spécifier le critère `{plat}`, qui comme son nom l'indique sert à afficher un forum à plat. Ce qui donne :
`<BOUCLE_forum(FORUMS) {id_article} {plat}>`

N'afficher que le total

Voyons maintenant comment compter les éléments qu'elle contient. La difficulté, ici, c'est que justement cette boucle ne doit rien afficher ! Elle n'affiche ni les messages ni leur titre, et on évitera même de lui faire afficher des espaces ou des retours à la ligne (sinon votre code HTML final contiendra des dizaines de lignes vides, inélégantes), en ne laissant pas même un espace entre les deux tags ouvrants et fermants de la boucle : `<BOUCLE_forum(FORUMS) {id_article} {plat}></BOUCLE_forum>`.

L'intérieur de la boucle n'affiche donc rigoureusement rien, mais on doit afficher, après la boucle, le nombre de résultats. La balise `#TOTAL_BOUCLE` peut s'utiliser non seulement à l'intérieur de la boucle, mais aussi (c'est la seule dans ce cas) dans le *texte conditionnel après* (le texte qui s'affiche après la boucle si elle contient des éléments) et le *texte conditionnel alternatif* (le texte qui s'affiche si la boucle est vide).

Une subtilité à bien comprendre : le texte conditionnel alternatif s'affiche *si la boucle ne trouve aucune réponse* ; il est donc affiché même si la boucle sélectionne des éléments (ici des messages de forum) mais qu'elle ne contient aucun affichage.

Nous devons donc placer `#TOTAL_BOUCLE` dans le texte conditionnel alternatif. S'il n'y aucun message de forum attaché à l'article, `#TOTAL_BOUCLE` sera vide, il ne faut donc pas afficher le texte englobant (« il y a N message(s) de forum ») dans ce cas.

<BOUCLE_combien(FORUMS) {id_article} {plat}></BOUCLE_combien>

[Il y a (#TOTAL_BOUCLE) message(s) de forum.]

</B_combien>

Un message, des messages...

Nous affichons désormais le nombre de messages, s'il en existe. Mais nous voulons faire mieux : nous souhaitons que le terme « message » soit automatiquement accordé au singulier lorsque la boucle ne trouve qu'un seul résultat, et accordé au pluriel lorsqu'elle en trouve plusieurs.

Pour cela, nous allons utiliser les filtres de test `|=={...}` et `|?{...,...}` (introduits avec SPIP 1.8) pour accorder « message » selon la valeur de `#TOTAL_BOUCLE` : `[(#TOTAL_BOUCLE|=={1}|?{message,messages})]`

Ce qui nous donne donc :

<BOUCLE_combien(FORUMS) {id_article} {plat}></BOUCLE_combien>

[Il y a (#TOTAL_BOUCLE) [(#TOTAL_BOUCLE|=={1}|?{message,messages})] de forum.]

</B_combien>

Un menu déroulant pour présenter une liste d'articles

Avril 2002 — maj : Novembre 2007

On souhaite réaliser un menu déroulant en utilisant les commandes HTML adaptées à la création de formulaire ; de plus on veut que ce menu serve à aller à l'URL de l'article sélectionné. Si l'URL des articles est du type `#URL_PAGE{article, id_article=123}`, le bout de code suivant conviendra :

<FORM ACTION="#URL_PAGE{article}" METHOD="get">

```
<SELECT NAME="id_article">
 <BOUCLE_menu_articles(ARTICLES) {id_rubrique} {par titre}>
 <OPTION VALUE="#ID_ARTICLE">#TITRE</OPTION>
 </BOUCLE_menu_articles>
</SELECT>

 <INPUT TYPE="submit" NAME="Valider" VALUE="Afficher l'article">
</FORM>
```

Les critères de la boucle articles (ici : les articles de la rubrique courante, triés par titre) seront modifiés selon vos besoins. Ce type de construction marche bien sûr aussi pour les brèves, rubriques...

Selon le même principe, il est tout aussi facile de présenter une liste de rubriques, de brèves... ou même l'intégralité de la structure du site.

Remplir les meta-tags HTML des pages d'article

Avril 2002 — maj : Décembre 2007

Le but de cet exemple est d'installer dans les méta-tags de notre page, la liste des mots-clés associés à l'article ainsi que le nom des auteurs.

Si l'on veut optimiser le référencement du site par les moteurs de recherche, on peut par exemple mentionner le descriptif de l'article, les mots-clés associés, ainsi que le nom du ou des auteurs dans des balises spéciales, appelées « Métas NAME », situées dans l'en-tête du document HTML.

Exemples de Métas NAME remplis par SPIP

Rôle	**Syntaxe HTML/SPIP**
Titre de la page	`<title>[(#NOM_SITE_SPIP|textebrut)]</title>`
Description	`[<meta name="Description" content="(#INTRODUCTION|couper{200}|textebrut)" />]`

Mots-clés	`<B_keywords>`
	`<meta name="Keywords" content="<BOUCLE_keywords(MOTS){id_article} {","}>[(#TITRE|textebrut)]</BOUCLE_keywords>" />`
	`</B_keywords>`

Auteurs	`[<meta name="Author" content="(#LESAUTEURS|textebrut)" />]`
Nom du logiciel	`<meta name="Generator" content="SPIP[ (#SPIP_VERSION)]" />`
Courriel du webmestre	`[<meta name="Reply-to" content="(#EMAIL_WEBMASTER|textebrut)" />]`

N'oubliez pas de passer le filtre `textebrut` sur les balises SPIP pour supprimer les tags, paragraphes et espaces insécables qui n'ont rien à faire ici. Limitez également le nombre de

caractères, conformément aux limitations propres à chaque meta-tags, avec le filtre `couper`.

On remarquera que pour les mots-clés on utilise une boucle imbriquée pour aller chercher ces informations à partir de l'*id_article* courant. De plus, on spécifie une virgule comme séparateur afin que le contenu du meta-tag soit compréhensible (y compris par un moteur de recherche).

Ces meta-tags ne sont pas indispensables pour assurer le bon référencement d'un site. Par contre, n'oubliez pas le titre de la page, qui reste important, notamment parce qu'il permet d'identifier clairement chaque page de votre site dans les résultats de recherche.

Voici donc l'exemple complet pour le squelette article (à placer dans une boucle ARTICLES, entre les balises `head` de la page) :

```
<head>

<title>[(#TITRE|textebrut) - ][(#NOM_SITE_SPIP|textebrut)]</title>

[<meta name="Description" content="(#INTRODUCTION|couper{200}|textebrut|
sinon{[(#DESCRIPTIF_SITE_SPIP|couper{200}|textebrut)]})" />]

<B_keywords><meta name="Keywords" content="<BOUCLE_keywords(MOTS) {id_article}
{","}>[(#TITRE|textebrut)]</BOUCLE_keywords>" /></B_keywords>

[<meta name="Author" content="(#LESAUTEURS|textebrut)" />]

<meta name="Generator" content="SPIP[ (#SPIP_VERSION)]" />

[<meta name="Reply-to" content="(#EMAIL_WEBMASTER|textebrut)" />]

</head>
```

Vous pouvez adapter cet exemple à chaque type d'élément : rubriques, brèves, etc.

Guide des fonctions avancées

Je voudrais une nouvelle fonctionnalité...

Avril 2002 — maj : 6 mars

Depuis SPIP 1.9, il est possible d'intégrer des « plugins », qui rajoutent des fonctionnalités à SPIP.

Toutefois, si vous ne trouvez pas votre bonheur, vous pouvez écrire à la liste des développeurs pour la proposer, en donnant moults détails et en précisant les raisons qui vous poussent à vouloir une telle fonctionnalité.

N'oubliez pas cependant que SPIP est un logiciel libre, et que vous pouvez implémenter cette fonctionnalité vous-même si vous avez les compétences requises — auquel cas vous feriez des heureux en nous la communiquant !

Multimedia et traitements graphiques

Les modèles d'incrustation de documents et leurs filtres

12 décembre — maj : 31 juillet

Les documents joints à un article sont généralement présentés en dehors de son texte, dans une espèce de portfolio séparé. SPIP permet de mentionner chacun de ces documents en un endroit arbitraire du texte. Cette mention peut se faire soit par un lien soit par une incrustation. Jusqu'à SPIP 1.9.2 inclus, le lien était fourni par le modèle `doc` et l'incrustation par le modèle `emb`. Depuis, l'incrustation est également disponible par autant de modèles que de groupes de types simple dans la *Multipurpose Internet Mail Extension* (soit 5 en MIME 1.0, seule version qu'a jamais connu cette nomenclature). Chacun de ces modèles gère différemment l'incrustation, souvent en appliquant au contenu d'un document un filtre spécifique redéfinissable déduit de son type.

Rappelons (Utiliser les modèles) qu'un modèle est un squelette appelable directement à partir du texte présent dans la base de données, afin d'obtenir un placement précis du fragment HTML décrit par le squelette. Pour le placement des pièces jointes, il en existe essentiellement deux, `doc` (pour référencer le document) et `emb` (pour l'incruster). L'incrustation offre le plus de possibilités.

Le modèle `emb` délègue à présent le travail à cinq modèles nommés par le nom du groupe MIME du document concerné : `text, image, audio, video, application`. Ces modèles sont utilisables directement (par exemple `<audio44|center>` est équivalent à `<emb44|center>` si le document relève du groupe audio) et il est d'ailleurs plus efficace de le faire, le modèle `emb` n'étant plus là qu'en cas de doute sur le groupe du document. L'utilisation directe a aussi l'avantage de forcer l'incrustation selon le modèle choisi, même si le document est officiellement d'un autre groupe. Cela permet de contourner certaines incongruités de la classification MIME : beaucoup de documents purement textuels y sont officiellement une `application`, par exemple le XHTML qui est pourtant un sous-ensemble du HTML, qui appartient au groupe `text`.

Le modèle `text`

Le modèle `text` incruste le contenu d'un document textuel dans un article, en appliquant au préalable si elle existe une fonction `filtre_`*T* où *T* est un nom déduit du type MIME du document. Cette déduction consiste à remplacer dans le nom du type tous les caractères non alphanumériques par un souligné. Ainsi, pour le type `text/txt` le modèle cherchera s'il existe la fonction `filtre_text_txt`.

La fonction `filtre_text_txt` est fournie d'avance par SPIP, et elle est utilisée par défaut pour les documents du groupe `text`. Elle remplace les chevrons dans le corps du document par les entités HTML correspondantes, et l'entoure par une balise `pre`, ce qui permet de présenter le contenu du document directement.

Est fournie également la fonction `filtre_text_csv`, destinée aux documents `text/csv`. Le RFC4180 étant très diversement suivi par les tableurs, cette fonction compte le nombre de tabulation, de virgule et de points-virgule dans le document et considère que le plus fréquent des trois est le séparateur utilisé. Elle le remplace par les raccourcis SPIP appropriés, ce qui fournit une table HTML correspondant à celle fournie par le tableur. L'usage des guillemets pour introduire le séparateur lui-même ou un saut de ligne est traité correctement (par une entité HTML et une balise `br` respectivement). Le saut de ligne final optionnel est également bien géré. Si la première ligne n'a que sa première colonne de remplie, elle fournira la balise `caption` de la table. La seconde ligne (ou la première sinon) sera vue comme le nom des colonnes, et sera donc typographiée avec des balises `th`. Le format CSV n'ayant pas prévu d'indication d'encodage, il faut s'assurer que celui du document est le même que celui du site avant d'utiliser ce filtre, SPIP ne pouvant savoir si un ré-encodage est nécessaire.

Enfin, il existe `filtre_text_html`, destinée aux documents `text/html`. Elle incruste le corps du document (donc ce qui est délimité par la balise `body`). Ce corps sera toutefois expurgé de ses scripts, par mesure de sécurité. Les balises `style` éventuellement présentes dans l'en-tête, seront regroupées en une seule au début de l'incrustation. Les feuilles de styles référencées dans l'en-tête par des balises `link` de type `text/css` seront recherchées sur le WEB et leur contenu s'ajoutera à la balise `style` mentionnée ci-dessus. Toutefois cela ne sera possible que pour les feuilles dont l'URL est absolue (autrement dit complète). A cette condition seulement SPIP pourra présenter le document de manière équivalente à l'originale, et sous réserve que les balises `img` présentes dans le document possèdent elles aussi des URLs absolues.

Le modèle `audio`

Le modèle `audio` permet d'incruster le contenu d'un document audio dans un article, sous la forme d'une balise `object`. Comme le précédent, ce modèle cherche s'il existe une fonction `filtre_`*T* où *T* est le nom du type après substitution des caractères non alpha-numériques par des soulignés. Mais il l'applique alors sur l'ID du document. Le résultat de cette fonction est inséré dans la balise `object`, ce qui permet en particulier de préciser les balises `param` souvent spécifiques au type audio concerné. De plus, les couples passés comme argument du modèle (syntaxe `nom=val`) sont eux aussi convertis en balise `param`.

Le modèle `image`

Le modèle `image` est une petite extension du raccourci `img`, présent de longue date dans SPIP. Il en diffère seulement en produisant une balise `object` plutôt qu'une balise `img` lorsque cette balise ne gère pas le type MIME du document concerné. Pour de tels types, il se comporte donc comme le

modèle audio, à cette différence que les couples passés en arguments seront traités comme attributs de la balise object. Il est donc à choisir en particulier pour l'incrustation de documents de type SVG, lequel relève officiellement du groupe application.

Le modèle video

Le modèle video incruste le document en l'entourant d'un panneau de contrôle permettant de la faire défiler.

Le modèle application

Ce modèle correspondant au groupe fourre-tout de la spécification MIME, il n'a pas de description synthétique bien claire. Il évoluera certainement beaucoup avec le temps.

Utilisation

Comme pour les autres modèles, on écrit par exemple <text67> pour appliquer le modèle text au document numéro 67, et donc voir son contenu directement dans la page.

Ces modèles sont également implicitement utilisés par le squelette article standard de SPIP lorsque l'article a un corps vide et possède un seul document joint. Ce squelette se comporte alors comme si le texte de l'article était réduit à <embN>, où N est le numéro du document, ce qui a pour effet de présenter le corps du document comme contenu de l'article.

Pour afficher sur un site la mise en page opérée par un tableur, il suffit donc de créer un article et d'y joindre la version csv de ce type de document, sans aucune manipulation supplémentaire. De même si l'on veut mettre en ligne une vidéo ou un fichier audio.

On peut aussi facilement passer sous SPIP un site HTML statique en associant chacune de ses pages à un article vide : si les URLs de leurs feuilles de style et de leurs images sont complètes, SPIP les laissera conduire la mise en page.

Images typographiques

Titres graphiques avec la police de son choix

Mars 2006 — maj : Décembre 2007

[SPIP 1.9, GD2 et Freetype]
Si GD2 et Freetype sont installés sur votre site [1], SPIP peut fabriquer, de lui-même, des images à partir de titres (ou tout élément de la base de donnée) en utilisant une police de caractères de votre choix.

La fonction qui réalise cet effet est image_typo. La présente page va vous présenter les différentes variables que l'on peut utiliser avec cette fonction.

[(#TITRE|image_typo)]

Si vous ne précisez aucune variable, image_typo va utiliser la police par défaut fournie avec SPIP : **Dustismo,** une police GPL de Dustin Norlander. Elle figure dans le sous-dossier polices du dossier dist (dans les versions précédentes de SPIP, il s'agissait du dossier ecrire).

police

Vous pouvez préciser le nom d'une autre police, que vous aurez pris soin d'installer sur votre site.

[(#TITRE|image_typo{police=dustismo_bold.ttf})]

(**Dustismo-bold** est également livré avec SPIP.)

En théorie, vous pouvez utiliser de nombreux formats de police : TrueType, PostScript Type 1, OpenType... Selon la configuration de votre site, il est possible que certains formats ne soient pas acceptés.

Les polices doivent être installées dans un sous-dossier `/polices` du dossier contenant vos squelettes.

Si nous installons, par exemple, un fichier TrueType ainsi :
`polices/stencil.ttf`
il est possible d'utiliser cette nouvelle police [2].

[(#TITRE|image_typo{police=stencil.ttf})]

taille

On peut préciser la taille d'affichage de la police. Cela s'utilise avec la variable `taille`.

[(#TITRE|image_typo{police=stencil.ttf,taille=36})]

Note : on ne précise pas « 36pt », on indique seulement « 36 », sans indication de l'unité.

couleur

Cette variable permet d'indiquer la couleur. Par défaut, le rendu est noir. Cette variable est une couleur RVB hexadécimale, toujours de la forme « 3399BB ». Notez : on omet le « # » qui précède habituellement ce type de code couleur.

[(#TITRE|image_typo{police=stencil.ttf,taille=36,couleur=4433bb})]

largeur

La variable `largeur` permet de fixer la largeur *maximale* de l'image. Notez bien : c'est une valeur maximale ; l'image réelle est « recadrée » automatiquement, ensuite, pour adopter les dimensions du texte réellement composé.

Le premier pavé ci-dessous est composé avec une largeur maximale de 300 pixels, le second avec une largeur de 400 pixels.

[(#TITRE|image_typo{police=stencil.ttf,largeur=300})]

[(#TITRE|image_typo{police=stencil.ttf,largeur=400})]

align

La variable `align` permet de forcer l'alignement de plusieurs lignes de texte (lorsque c'est le cas) à gauche, droite, ou au centre. Exceptionnellement, on utilise ici une syntaxe anglaise, proche de ce qui se fait pour les feuilles de style.

[(#TITRE|image_typo{police=stencil.ttf,align=left})]

[(#TITRE|image_typo{police=stencil.ttf,align=center})]

[(#TITRE|image_typo{police=stencil.ttf,align=right})]

hauteur_ligne

`hauteur_ligne` permet de fixer la hauteur entre chaque ligne de texte (dans le cas où l'image comporte plusieurs lignes).

[(#TITRE|image_typo{police=stencil.ttf,taille=36,hauteur_ligne=80})]

padding

Certaines polices « dépassent » de leur boîte de rendu, et on obtient un effet désastreux (polices « coupées »). La variable `padding` permet, exceptionnellement, de forcer un espace supplémentaire *autour* du rendu typographique.

[(#TITRE|image_typo{police=stencil.ttf,padding=5})]

Filtrer l'image

Le résultat de `image_typo` étant une image, il est tout à fait possible de lui appliquer des filtres d'images. Par exemple, ci-après, on rend l'image semi-transparente, ou on lui applique une texture.

[(#TITRE|image_typo{police=stencil.ttf,couleur=aa2244}|image_alpha{60})]

[(#TITRE|image_typo{police=stencil.ttf,couleur=aa2244}|image_masque{carre-mur.png})]

P.-S.

N.B.1. L'image créée par `image_typo` est au format PNG 24 avec une couche alpha pour réaliser la transparence. Pour forcer Microsoft Explorer à afficher correctement cette transparence, SPIP utilise une classe de feuille de style spécifique, `format_png`, définie dans `spip_style.css` ; celle-ci appelle un « comportement » (*behavior*) rendant l'affichage possible sous MSIE. On a donc, encore une fois, tout intérêt à intégrer le `spip_style.css` standard dans ses propres squelettes, quitte à le surcharger avec ses propres styles.

N.B.2. L'affichage de certaines polices (notamment les anglaises et certaines italiques) est problématique. Les techniques de rendu typographique dans GD2 sont, visiblement, encore en développement (nous rencontrons bugs sur bugs de ce côté). Espérons que les fonctions GD2 progresseront rapidement.

N.B.3. De l'arabe, du farsi, de l'hébreu ? Malheureusement : non ! Nous rencontrons pour l'heure

deux difficultés qui ne permettent pas de proposer de solution « propre » pour l'affichage de l'hébreu et de l'arabe.
— Tout d'abord, la gestion de l'affichage bidirectionnel n'est pas assuré par GD2 ; il n'est donc pas possible pour l'instant de créer des images typographiques pour des chaînes s'écrivant de droite à gauche.
— Pour l'arabe (et le farsi), les ligatures ne sont pas gérées. En particulier : les ligatures d'OpenType sont purement et simplement ignorées.

Notes

[1] GD2 est une extension *graphique* de PHP, qui permet de nombreuses manipulations d'images. Freetype, habituellement installé avec GD2, est l'extension qui insère du texte dans une image à partir d'un fichier de police, Truetype ou Postscript. En cas de doute, demandez à votre hébergeur si GD2 est installé.

[2] Attention : si vous ne protégez pas ce dossier (avec un htaccess par exemple), votre fichier de police sera accessible par le Web. Si vous utilisez des polices commerciales, faites attention à ne pas vous retrouver, ainsi, à diffuser des polices pour lesquelles cela n'est pas autorisé

Couleurs automatiques

Mars 2006 — maj : 10 août

SPIP 1.9

[SPIP 1.9 et GD2]
SPIP permet d'extraire automatiquement une couleur d'une image, afin de l'appliquer à d'autres éléments d'interface.

Par exemple, à côté du logo d'un article, nous allons afficher le titre du même article *dans une couleur* tirée de ce logo. De cette façon, tout en rendant l'affichage plus varié (d'un article à l'autre, la couleur utilisée change en fonction du logo), le fait que la couleur soit extraite de l'image assure une certaine cohérence graphique.

Cette fonction consistant à récupérer une couleur dans une image est complétée par toute une série de fonctions permettant de manipuler cette couleur, principalement éclaircir et foncer la couleur. La liste de fonctions est longue, de façon à permettre un nombre très important d'effets.

couleur_extraire

À partir d'une image (logo d'article, logo de rubrique..., mais aussi images de portfolio), on demande à SPIP de tirer une couleur.

[(#LOGO_RUBRIQUE||couleur_extraire)]

Attention : il ne s'agit pas pour SPIP de déterminer la couleur *dominante* de l'image, mais d'extraire une couleur de l'image. Pour que cette couleur soit réellement « représentative », l'image est réduite à une taille de 20 pixels maximum ; ainsi les différentes couleurs de l'image sont relativement « moyennées ». Cette valeur de 20 pixels est expérimentale : elle est suffisamment basse pour éviter d'extraire une couleur très peu présente dans l'image ; elle est suffisamment élevée pour éviter que la couleur soit systématiquement grisâtre.

Utilisée sans paramètres, la fonction `couleur_extraire` retourne une couleur située légèrement au-dessus du centre de l'image. Il est possible d'indiquer un point préféré pour le

sondage, en passant deux valeurs (*x* et *y*) comprises entre 0 et 20 (conseil : entre 1 et 19 pour éviter les effets de marges).

Par exemple :

[(#LOGO_RUBRIQUE||couleur_extraire{15,5})]

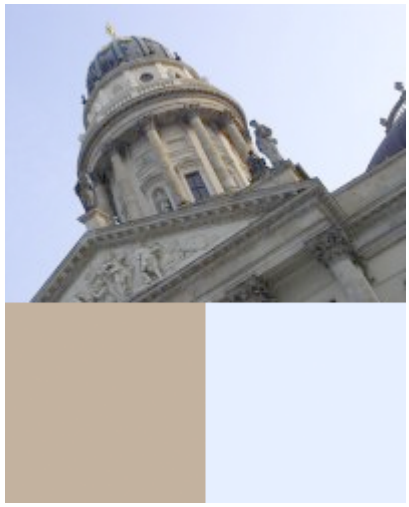
retourne une couleur située en haut à droite du centre de l'image.

Pour bien comprendre le principe, appliquons ce filtre sur un logo de couleur uniforme :

Le résultat est : `ff9200`.

Notez bien : les valeurs retournées sont systématiquement en codage RVB hexadécimal, en omettant le « # » qui précède habituellement ces codes. On pensera donc à insérer ce dièse quand on utilise ces valeurs.

On peut, par exemple, appliquer cette couleur au fond d'un pavé :

Appliquons ce filtre à une photographie : 

En bas à gauche, la couleur extraire sans paramètres, c'est-à-dire présente un peu au dessus du centre de l'image (marron clair de la pierre du bâtiment). En bas à droite, on force une couleur située en haut à droite de l'image (bleu clair du ciel).

[(#LOGO_RUBRIQUE||couleur_extraire)]

[(#LOGO_RUBRIQUE||couleur_extraire{15,5})]

L'utilisation de ce filtre est, techniquement, très simple. En revanche, créativité et inventivité seront nécessaires pour l'exploiter... Voici quelques utilisations :
— couleur de texte ;
— couleur de fond d'un pavé ;
— couleur d'une image typographique ;
— modifier la couleur d'une image (`image_sepia`)...

Modifier la couleur

Une fois la couleur extraite, il est utile de la manipuler, afin de jouer avec différentes variantes de la couleur, tout en respectant la cohérence graphique.

- **couleur_foncer**, **couleur_eclaircir**

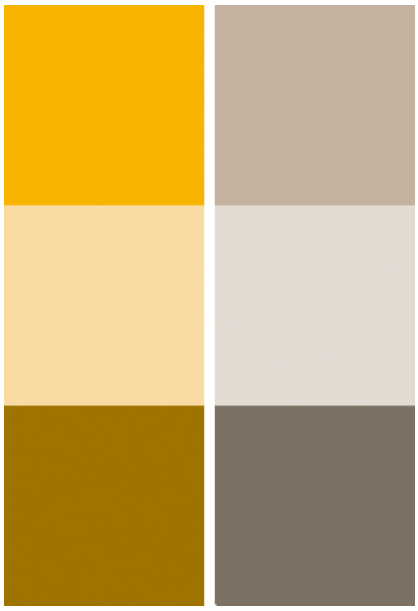
À partir de la couleur extraite d'une image, nous souhaitons afficher des couleurs plus foncées et plus claires.

[(#LOGO_RUBRIQUE||couleur_extraire)]

[(#LOGO_RUBRIQUE||couleur_extraire|couleur_foncer)]

[(#LOGO_RUBRIQUE||couleur_extraire|couleur_eclaircir)]

Appliqué aux couleurs extraites des exemples précédents, cela donne :



On constate qu'on a ainsi des camaïeux de couleurs faciles à obtenir, l'ensemble étant très cohérent.

- **couleur_foncer_si_claire**, **couleur_eclaircir_si_foncee**

Si nous appliquons la couleur extraite au fond d'un pavé de texte, il faut déterminer dans quelle couleur nous voulons écrire ce texte (par exemple : noir sur orange ou blanc sur orange ?) ; c'est ce que nous verrons avec les fonctions suivantes.

Pour l'instant, nous décidons que le texte sera d'une certaine couleur. Nous voulons par exemple que le texte soit noir. Il faut donc choisir la couleur du fond en fonction de ce texte noir : il faut que la couleur du fond soit claire (donc : texte noir sur fond clair).

Si nous appliquons le filtre `couleur_eclaircir` à notre couleur extraite, nous avons deux cas :

— si la couleur est foncée, alors elle est éclaircie et nous obtenons l'effet voulu ;
— si la couleur est déjà claire, alors nous l'éclaircissons encore, et nous obtenons un fond qui peut devenir quasiment blanc. Or, la couleur étant déjà claire , nous aurions voulu l'utiliser telle quelle.

C'est ici que nous appliquons le filtre `couleur_eclaircir_si_foncee` :
— si la couleur est foncée, nous l'éclaircissons ;
— si la couleur est claire, nous l'utilisons telle quelle.

Le filtre `couleur_foncer_si_claire` a la logique exactement inverse. Il est très utile, par exemple, pour écrire en blanc sur un fond systématiquement foncé, mais en évitant de rendre ce fond quasiment noir quand la couleur d'origine est déjà foncée.

— **couleur_extreme**, **couleur_inverser**

Le filtre `couleur_extreme` passe une couleur foncée en noir, et une couleur claire en blanc. Cela est utile pour écrire en noir ou blanc sur un fond coloré.

En réalité, récupérer la couleur « extrême » est habituellement utilisé avec `couleur_inverser`, il inverse la couleur RVB. Elle transforme notamment du noir en blanc, et du blanc en noir.

En pratique, cela permet d'assurer un bon contraste, quelle que soit la couleur du fond du bloc (alors que, dans l'exemple précédent, nous choisissions la couleur du fond du bloc en fonction d'une couleur de texte).

Appliquons, en couleur de fond, la couleur extraite de l'image :

 ...

On obtient donc, selon le logo de l'article, soit un fond foncé, soit un fond clair.

Appliquons, pour la couleur du texte, la couleur extraite, rendue « extrême » :

[(#LOGO_ARTICLE||couleur_extraire|couleur_extreme)]

— Si la couleur est foncée, la couleur extrême est noire ; nous écrivons en noir sur fond foncé.
— Si la couleur est claire, la couleur extrême est blanche ; nous écrivons en blanc sur fond clair.

Dans les deux cas, c'est peu lisible. On pourra utiliser cette couleur pour un autre effet (par exemple : une bordure autour du `div`).

Il nous reste à inverser cette couleur pour l'appliquer au texte ;

```
<div
 style="color: #[(#LOGO_ARTICLE||couleur_extraire|couleur_extreme|couleur_inverser)];
  background-color: #[(#LOGO_ARTICLE||couleur_extraire)];">
   ...
</div>
```

— Si la couleur extraite est foncée, la couleur extrême est noire, et l'inverse est alors blanche. On écrit en blanc sur fond foncé.
— Si la couleur extraite est claire, la couleur extrême est blanche, et l'inverse est noire. On écrit en blanc sur fond clair.

Dans les deux cas, le contraste assure une bonne lisibilité.

Traitement automatisé des images

Mars 2006 — maj : 20 août

[SPIP 1.9 et GD2]
SPIP permet de faire subir aux images des effets automatisés. Ces effets ont deux vocations :
— tout simplement assurer la cohérence graphique du site, en fabriquant automatiquement des éléments de navigation qui seront toujours réalisés selon les désirs du graphiste ;
— créer des effets relativement spectaculaires, sans pour autant demander aux auteurs des articles de traiter les images eux-mêmes, et sans non plus interdire les évolutions graphiques du site par la suite.

Par exemple : on veut, dans l'interface graphique du site public, que les logos de navigation des articles aient deux aspects :
— dans tous les cas, ils sont « posés sur le sol », avec un reflet sous eux ;
— au repos, ils sont en noir et blanc, assez foncés ; survolés, ils sont en couleur.

Sans les automatismes qui suivent, les webmestres ont pris l'habitude de créer, à la main, deux versions de ces images, et d'installer deux logos sur le site. Deux inconvénients (particulièrement gênants) :
— la manipulation est longue ; par ailleurs elle ne peut pas être confiée à un tiers, qui serait de toute façon incapable de la réaliser correctement ;
— lorsqu'on voudra faire évoluer l'interface graphique du site, on sera bloqué avec ces logos très typés.

Avec les automatismes qui suivent, on peut travailler autrement : les auteurs installent un simple logo d'article (par exemple, une photographie), sans aucun traitement spécifique ; et, automatiquement, les squelettes de SPIP fabriquent :
— une vignette à la bonne taille ;
— la même vignette en noir et blanc, légèrement foncée ;
— les reflets de l'image sur le sol.

Certains des filtres présentés utilisent les fonctions de redimensionnement des images. Il est impératif, après l'installation, de se rendre dans l'espace privé, « Configuration », puis dans l'onglet « Fonctions avancées » : là, sélectionnez « GD2 » pour la « Méthode de fabrication des vignettes ».

Avertissement lenteur

Avant de commencer, signalons que ces fonctions sont *lourdes*. Voire très lourdes si vous utilisez de grosses images. Le calcul d'une image est relativement long, et s'il faut calculer, dans vos squelettes, plusieurs images, voire plusieurs effets pour chaque image, vous risquez d'obtenir des erreurs de timeout (temps maximum d'exécution des scripts dépassé).

Cela dit, il est important de noter que les filtres de traitement des images ont leur propre système de cache : *une fois calculée, une image « filtrée » est sauvegardée, et ne sera plus recalculée*. La charge sur le serveur est donc limitée au premier calcul.

Cet avertissement concerne, en priorité, les hébergeurs mutualisés.

La page « Vider le cache » de l'espace privé affiche la taille utilisée par SPIP pour stocker ces images calculées. On peut ainsi vider ce cache indépendamment du cache des squelettes et des pages HTML.

Transparences

Dans [SPIP 1.9], si l'on utilise GD2, outre les fonctions exposées dans cet article, vous constaterez que les réductions d'image (`image_reduire`) respectent la transparence des fichiers GIF et PNG 24 (transparence par couche alpha).

Dans la configuration du site, pensez à sélectionner GD2 comme méthode de réduction d'image.

L'image d'origine

Toute image gérée par SPIP peut se voir appliquer les filtres suivants. Sont donc concernés les logos (d'articles, de rubriques...), mais aussi les images de portfolio (images en tant que fichiers joints aux articles), sans oublier les nouvelles images typographiques.

Voici, pour nos exemples, une image à traiter.

Réduire les dimensions d'une image

La fonction `reduire_image` devient `image_reduire`, pour adopter la même logique de noms que les nouvelles fonctions graphiques. L'ancienne dénomination est toujours fonctionnelle.

Elle est désormais complétée par une `image_reduire_par`, qui permet de réduire une image selon une certaine échelle. Là où `reduire_image` réduit une image à des dimensions fixées à l'avance, `image_reduire_par` réduit l'image proportionnellement.

Par exemple :

[(#TITRE|image_typo{taille=24}|image_reduire_par{2})]

réduit l'image typographique d'un facteur 2 (l'image devient deux fois plus petite).

Recadrer une image

La fonction `image_recadre{largeur,hauteur,position}` permet de recadrer une image (équivalent du crop des logiciels de traitement d'image) avec les combinaisons de left/center/right et top/center/bottom pour la position (ex 'left center').

Par exemple :

[(#FICHIER|image_recadre{90,90,center})]

recadre l'image originale en un carré de 90 px de largeur et hauteur ne gardant en se basant sur le centre de l'image

Supprimer la transparence et forcer le format de l'image

La fonction `image_aplatir` réalise deux opérations :
— elle sauvegarde une image dans un format prédéfini (par exemple, transformer une image PNG en une image GIF) ;
— elle supprime les informations de transparence, et remplace les zones transparentes par une couleur.

Par exemple :

```
[(#TITRE
    |image_typo{police=stencil.ttf,couleur=000000,taille=40}
    |image_aplatir{gif,ff0000})]
```

Le titre transformé en image typographique est un fichier PNG avec des zones transparentes. En passant cette image par le filtre `image_aplatir`, on la transforme en GIF, en remplaçant les zones transparentes par du rouge (`ff0000`).

image_nb

Le filtre `image_nb` passe une image en niveaux de gris (ce qu'on appelle « noir et blanc » lorsqu'on évoque des photographies).

Sans paramètres (image de gauche), le filtre calcule les niveaux de gris en pondérant les composantes de l'image d'origine ainsi :
luminosité = 0,299 × rouge + 0,587 × vert + 0,114 × bleu.

On peut forcer la pondération des trois composantes RVB en passant les valeurs en pour-mille. Par exemple (image de droite) :

```
[(#FICHIER|image_nb{330,330,330})]
```

On a pris chaque composante R, V et B à niveau égal.

image_sepia

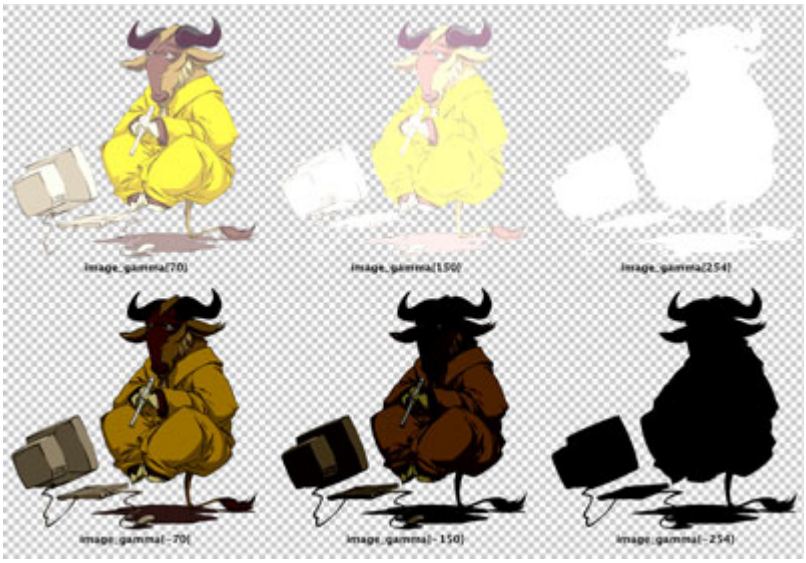
Le filtre `image_sepia` applique un filtre « Sépia ». Appliqué à une photographie, ce genre d'effet donne une tonalité de vieille photographie.

Sans paramètres (image de gauche), la valeur sépia est, par défaut, « 896f5e » (en RVB hexadécimal). On peut passer la valeur de la couleur de sépia en paramètre. Par exemple (image de droite) :

[(#FICHIER|image_sepia{ff0033})]

image_gamma

Le filtre `image_gamma` change la luminosité d'une image. Il rend une image plus claire ou plus foncée. Son paramètre est compris entre -254 et 254. Les valeurs supérieures à zéro rendent l'image plus claire (254 rend toute l'image entièrement blanche) ; les valeurs négatives rendent l'image plus foncée (-254 rend l'image complètement noire).



[(#FICHIER|image_gamma{70})]

[(#FICHIER|image_gamma{150})]

[(#FICHIER|image_gamma{254})]

[(#FICHIER|image_gamma{-70})]

[(#FICHIER|image_gamma{-150})]

[(#FICHIER|image_gamma{-254})]

image_alpha

Le filtre `image_alpha` rend l'image semi-transparente, en PNG 24 avec couche alpha. Si l'image

était déjà semi-transparente, les deux informations sont mélangées.

[(#FICHIER|image_alpha{50})]

[(#FICHIER|image_alpha{90})]

Le paramètre est une valeur entre 0 et 127 : 0 laisse l'image inchangée (aucune transparence), 127 rend l'image complètement transparente.

image_flou

Le filtre `image_flou` rend l'image... floue. On peut lui passer en paramètre un nombre compris entre 1 et 11, définissant l'intensité du floutage (de 1 pixel de floutage à 11 pixels de floutage).

[(#FICHIER|image_flou)]

[(#FICHIER|image_flou{6})]<

Sans paramètre, la valeur de floutage est 3.

Attention : ce filtre est particulièrement lourd (c'est-à-dire nécessite beaucoup de puissance). Plutôt que de tenter un floutage important, on peut préférer flouter plusieurs fois avec des valeurs faibles. Par exemple, remplacer :

[(#FICHIER|image_flou{6})]

par :

[(#FICHIER|image_flou|image_flou)]

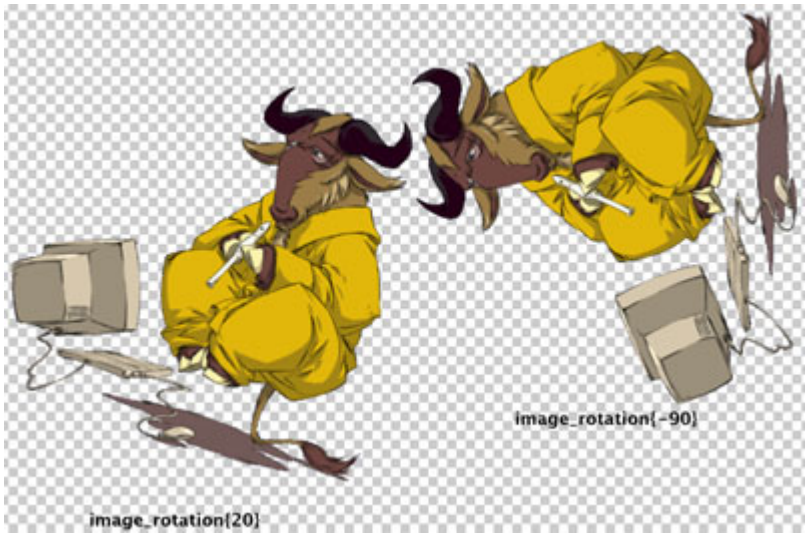
Au pire, le calcul se fera en deux « recalcul » de squelette, le premier floutage étant sauvegardé en cache.

Attention (2) : ce filtre agrandit l'image, en ajoutant tout autour de l'image une « marge »

équivalente à la valeur de floutage. Ainsi, avec le paramètre « 3 » (par défaut), on ajoute 3 pixels de chaque côté de l'image, et le résultat aura donc 6 pixels de large et de haut de plus que l'image d'origine.

image_rotation

Le filtre `image_rotation` fait tourner l'image d'un angle égal au paramètre passé. Les valeurs positives sont dans le sens des aiguilles d'une montre.



[(#FICHIER|image_rotation{20})]

[(#FICHIER|image_rotation{-90})]

Sauf pour les rotations à angle droit, la rotation provoque un effet d'escalier. Nous avons tenté de le limiter, mais il reste toujours présent. Une solution pour réduire cet effet consiste à réduire l'image après avoir appliqué la rotation.

Attention : ce filtre est relativement lourd !

Attention (2) : ce filtre modifie les dimensions de l'image.

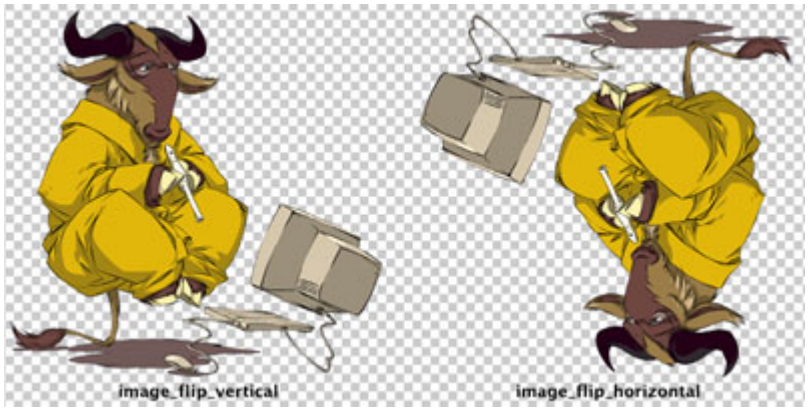
image_flip_vertical et image_flip_horizontal

Le filtre `image_flip_vertical` applique un effet de « miroir » selon un axe vertical ; `image_flip_horizontal` selon un axe horizontal.

Très simple d'utilisation, il n'y a pas de paramètre.

[(#FICHIER|image_flip_vertical)]

[(#FICHIER|image_flip_horizontal)]



image_masque

`image_masque` est le filtre le plus puissant de cette série. De fait, sa logique est nettement plus complexe que les autres filtres. Il permet, à partir d'un fichier PNG 24 en niveaux de gris et avec couche de transparence alpha, de modifier :
— le cadrage de l'image ;
— la transparence d'une image ;
— la luminosité d'une image.

- **Dimensions de l'image**

Si l'image d'origine est plus grande que le fichier masque, alors l'image d'origine est réduite et découpée au format du masque, puis on applique les informations de transparence et de luminosité du masque. Utile pour créer les vignettes de navigation.

Si l'image d'origine est plus petite que le masque, alors on ne recadre pas, on applique simplement les informations de luminosité et de transparence du masque (lui-même non redimensionné).

Voici notre image d'origine :

- **Masque de transparence**

Les informations de transparence du masque sont directement appliquées à l'image d'origine. Un pixel transparent du masque rend le pixel correspondant de l'image d'origine transparent, selon la même valeur de transparence. (Si l'image d'origine est déjà transparente, les informations sont

mélangées de façon à conserver les deux infos de transparence.)

Si on a le fichier masque suivant, nommé « decoupe1.png » :

qu'on applique ainsi :

[(#FICHIER|image_masque{decoupe1.png})]

on obtient l'image suivante :

L'image d'origine a été redimensionnée aux dimensions de « decoupe1.png », et les zones transparentes du masque sont devenues les zones transparentes du résultat.

> **Attention !** Le filtre utilise la réduction d'image. Pour qu'il fonctionne correctement, il est impératif de choisir la méthode de réduction GD2 dans la configuration du site, onglet « Fonctions avancées ».

- **Masque de luminosité**

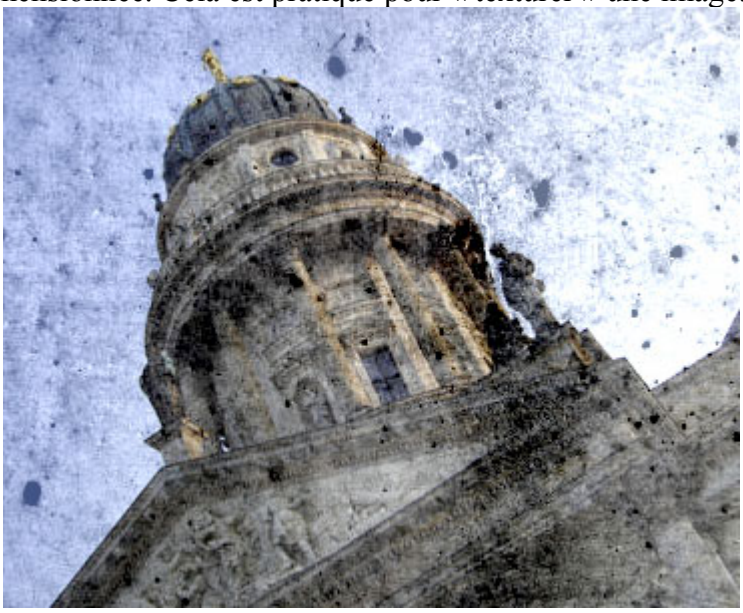
Dans l'exemple ci-dessus, l'image masque est entièrement en gris à 50%. Les couleurs de l'image d'origine sont alors laissées inchangées (on s'est contenté de « découper » l'image.

En faisant varier les couleurs du masque, on va appliquer ces différences de luminosité à l'image traitée. Lorsqu'un pixel du masque est plus clair, alors le fichier résultant est éclairci ; si le pixel du masque est foncé, alors on fonce le fichier résultant.

Par exemple, si notre masque est « decoupe2.png » :

[(#FICHIER|image_masque{decoupe2.png})]

on obtient l'image suivante :

Dans ces deux exemples, le masque est plus petit que l'image d'origine, on obtient donc une sorte de vignette de l'image. Si le masque est plus grand que l'image d'origine, on l'applique à l'image non redimensionnée. Cela est pratique pour « texturer » une image. On peut ainsi réaliser l'effet suivant :  en appliquant un masque en niveau de gris, masque que nous avons créé plus grand que l'image d'origine.

Attention : l'impact sur la luminosité est plus important sur l'image finale que dans le fichier masque.

Attention (2) : en réalité, le masque de luminosité est un masque de coloration. Si l'image masque est colorée, alors on modifiera non seulement la luminosité, mais aussi les couleurs de l'image. Mais cet effet est particulièrement difficile à maîtriser, notamment en partant d'images en couleur.

P.-S.

Sur son site Paris—Beyrouth, ARNO* donne de nombreux exemples d'utilisation de ces filtres graphiques, et explique notamment la philosophie qui les sous-tend : « Pourquoi utiliser les filtres graphiques de SPIP 1.9 ? ».

Le traitement des images

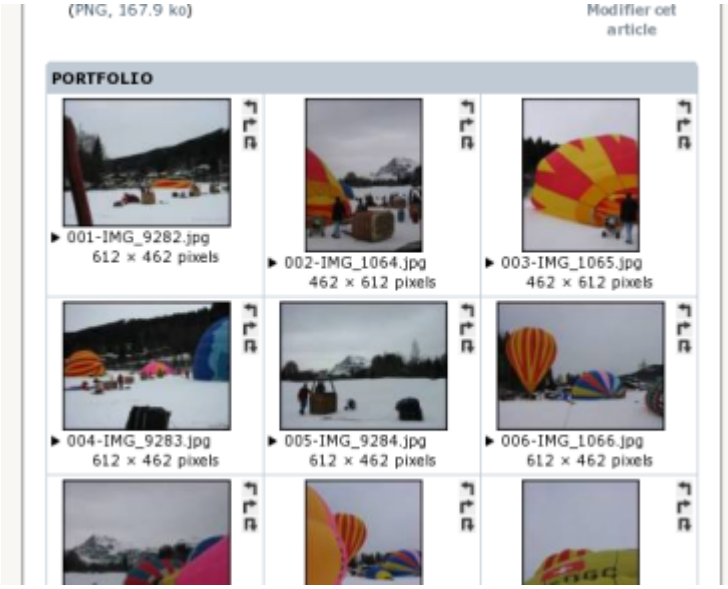
via GD, GD2 ou Imagemagick

Avril 2005 — maj : Décembre 2007

SPIP permet d'utiliser les systèmes de traitement d'images installés sur votre site d'hébergement. Introduites dans la version 1.7, ces fonctions de SPIP sont complétées et plus puissantes.

SPIP utilise le traitement d'images de trois manières différentes :

— la création de vignettes de prévisualition pour les images installées comme « documents joints » ; cela était déjà présent dans la version 1.7 ; [SPIP 1.8] permet de plus, dans ces « portfolios », de faire subir à chaque image une rotation à 90° (cela est particulièrement intéressant lorsqu'on installe une série de photographies depuis un appareil photo numérique) ;



— à de nombreux endroits dans l'espace privé, l'affichage de « vignettes » destinées à illustrer la navigation, à partir des logos des articles, des rubriques et même des auteurs (par exemple, si l'on dote les participants à un site de logos d'auteurs, des vignettes de ces logos accompagneront tous les messages de ces auteurs dans les forums de l'espace privé) ;

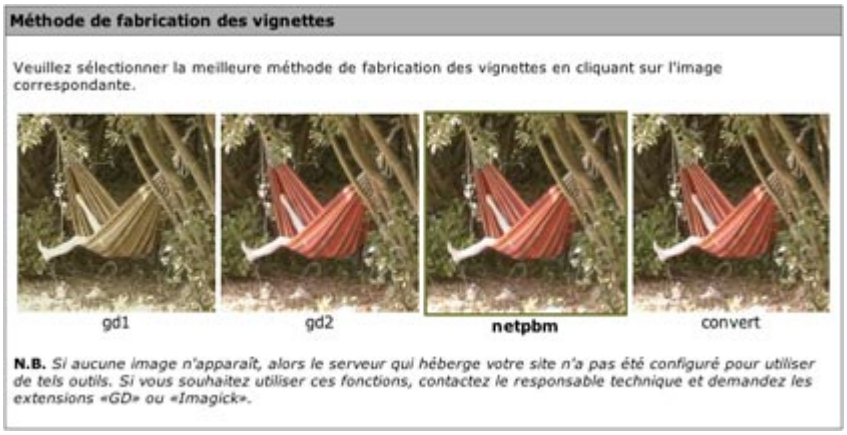
— dans les squelettes, les webmestres disposent d'une fonction `image_reduire`

particulièrement utile pour contrôler sa mise en page, et créer différentes versions (de différentes tailles) d'une même image. Nous ne pouvons qu'encourager les webmestres à « jouer » avec cette fonction pour enrichir et contrôler leur interface graphique ; on en tirera avantage pour obtenir :

- des alignements d'images parfaits (par exemple : toutes les images de la même largeur), sans s'obliger à installer des images de dimensions prédéfinies ;
- la garantie de ne pas faire « exploser » sa mise en page lorsqu'une image trop grande est installée par un rédacteur,
- des effets de survol et d'animation réalisés simplement en utilisant la même image à des tailles différentes (sans devoir jouer avec les « logos de survol »),
- des interfaces de portfolios (galeries de photos) épatantes...

Choix du système de traitement d'images

Mais, pour réaliser ces opérations de traitement d'images, SPIP fait appel à des systèmes qui ne peuvent pas être installés automatiquement avec SPIP, mais doivent être présents sur le serveur qui héberge votre site. Il faut donc que ces systèmes soient présents *en plus de SPIP* (dit autrement : il ne suffit pas que SPIP soit installé pour que les fonctions de traitement d'images soient disponibles ; il faut en fait que ces fonctions soient présentes par ailleurs).



Le choix d'un système de traitement d'images se fait dans la partie « Configuration » (configuration avancée) de l'espace privé. SPIP permet de choisir parmi 5 méthodes différentes de traitement des images.

Imagemagick

Imagemagick en tant qu'extension de PHP (php-imagick) est le choix privilégié par SPIP. SPIP est capable de déterminer seul sa présence. Si Imagemagick est présent sur votre serveur, alors SPIP l'utilisera automatiquement.

Si Imagemagick n'est pas présent sur votre serveur, alors SPIP vous proposera de choisir parmi d'autres méthodes. Ces méthodes n'étant pas détectables par SPIP (et, en tout cas, pas parfaitement), une vignette vous est proposée pour chaque méthode ou, éventuellement, pas de vignette si la méthode ne fonctionne pas sur votre site. Vous êtes alors invité à sélectionner votre méthode préférée (parfois : sélectionner la seule réellement disponible !).

GD, GD2

GD (et sa version 2, nettement plus puissante) est une extension de PHP désormais fréquemment présente sur les serveurs, y compris les hébergeurs mutualisés.

Si GD2 est présente, vous pouvez l'utiliser, elle donne des résultats de bonne qualité.

En revanche, GD (comprendre : « version 1 de GD ») est proposée comme pis-aller : le traitement des images se fait en 256 couleurs, et introduit de fortes dégradation des images ; elle n'est donc à sélectionner que *si aucune autre méthode ne fonctionne sur votre site*.

Imagemagick par convert

Convert est le logiciel en ligne de commande de Imagemagick. La qualité est absolument épatante, cependant son installation est relativement complexe.

Une fois *convert* installé sur votre site, vous devez configurer le chemin d'accès dans `mes_options.php3` (il s'agit d'un appel en ligne de commande) par la variable suivante :

$convert_command = '/bin/convert';

Il convient ici d'indiquer le chemin complet d'accès au programme. Sous Linux, ce chemin est souvent

$convert_command = '/bin/convert';

sous MacOS X, s'il est installé avec Fink :

$convert_command = '/sw/bin/convert';

(ces valeurs sont fournies à titre indicatif ; comme tout programme, il peut être installé quasiment n'importe où...).

NetPBM

Cette méthode consiste en trois programmes, déjà anciens, qui permettent de réaliser le redimensionnement de l'image. L'avantage de cette méthode est que ces programmes peuvent être installés sans accès *root* sur la plupart des hébergements.

On trouvera, sur le site du logiciel *gallery*, une explication claire et des versions précompilées de NetPBM.

Dans SPIP, on configure le chemin d'accès à pnmscale (l'un seulement des trois programmes installés - les deux autres chemins s'en déduiront, puisque les programmes sont installés dans le même répertoire) par la variable suivante :

$pnmscale_command = '/bin/pnmscale';

(encore une fois, c'est-à-vous de déterminer le chemin d'accès réel de votre installation).

* *

Pour rappel, vous pouvez obtenir nombre d'informations utiles sur votre système via la page /ecrire/ ?exec=info, notamment : — le système utilisé (utile pour installer NetPBM précompilé) ; — la version de PHP ; — la présence éventuelle des extensions GD, GD2 et Imagemagick.

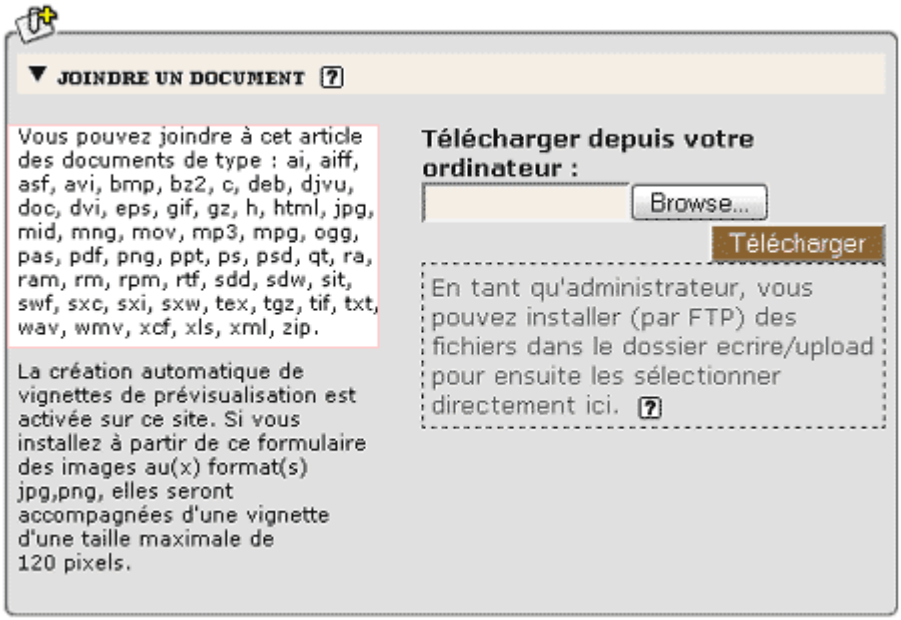
Enfin, en cas de difficulté, la meilleure solution consiste à contacter votre hébergeur pour qu'il installe les extensions nécessaires si aucune n'est présente. La présence d'au moins une extension graphique de PHP est désormais une norme chez les hébergeurs, n'hésitez pas à demander leur installation pour en bénéficier sur votre site.

Ajouter un type de document

Août 2003 — maj : Décembre 2007

Depuis la version [SPIP 1.4], il est possible d'installer des documents joints aux articles (et, en option, aux rubriques).

Pour des raisons de sécurité, SPIP n'autorise pas l'installation de n'importe quels types de documents. En effet, permettre l'installation de documents sur un serveur distant à partir d'une interface Web peut poser de sérieux problèmes de sécurité. C'est pourquoi cette liste d'autorisations existe, et pour cette même raison SPIP ne propose pas d'interface pour modifier cette liste.



Comme vous pouvez le constater, la liste des types de fichiers autorisés est déjà relativement fournie, et nous l'enrichissons régulièrement lorsque le besoin est exprimé sur les listes.

Avant de poursuivre, prenez le temps de bien lire ce qui suit :

- Cette manipulation est *potentiellement dangereuse* et peut introduire un énorme trou de sécurité dans votre site. Certains formats de fichiers (exécutables sur le serveur) ne doivent surtout pas être acceptés. En particulier, **n'acceptez jamais l'installation de fichiers de type PHP** (.php, .php3...) sur votre site, la sécurité de votre site serait *totalement compromise*. N'hésitez pas à vous renseigner avant d'ajouter des types de fichiers.

- Volontairement, nous n'avons pas installé dans SPIP d'interface pour modifier la liste des types de fichiers autorisés. Cela pour réserver cette modification aux utilisateurs confirmés. Pour effectuer la manipulation, vous devez utiliser un gestionnaire de base de données (par exemple : phpMyAdmin). Or s'attaquer directement à la base de données de SPIP « à la main » (sans utiliser l'interface et les automatismes de SPIP) est potentiellement destructeur pour votre site. Si vous ne savez pas exactement ce que vous faites, ne le faites pas. Si vous n'avez pas l'habitude de phpMyAdmin, ne l'utilisez pas sur la base de SPIP. Dans tous les cas, effectuez une sauvegarde de votre site SPIP avant de procéder à des interventions manuelles dans la base.

- Accessoirement, l'ajout d'un type de fichier ne se justifie que si les visiteurs de votre site peuvent utiliser (« lire ») ce type de fichier. Utiliser un format de fichier que l'on ne peut lire qu'avec un logiciel ultra-spécialisé sur un site grand public n'a pas vraiment de sens. Avant d'ajouter un type de fichier, vérifiez qu'il peut bien être utilisé par vos

utilisateurs (notamment : lecteur ou plug-in gratuit et facile à installer ; ce format est-il lisible sur tous les ordinateurs, Mac, PC, Linux... ?).

- Pour toutes ces raisons, nous insistons sur le fait que **cette manipulation ne doit être effectuée qu'en parfaite connaissance de cause**. Si vous avez des doutes, renseignez-vous auprès de personnes compétentes ou, plus simplement encore, convertissez vos fichiers dans un format déjà autorisé et largement répandu (par exemple, un document FrameMaker ne pourra pas être lu directement par le grand public ; s'il s'agit simplement de diffuser son contenu, vous pouvez aussi bien en faire un fichier PDF directement téléchargeable par vos visiteurs plutôt que de vouloir ajouter le type « FrameMaker » dans la liste des fichiers autorisés par SPIP).

* *

Pour ajouter un type de fichier autorisé sur votre site sous SPIP, utilisez phpMyAdmin (ou tout logiciel équivalent) pour accéder à la gestion de la base de données.

Il vous faut ajouter un nouvel élément dans la table `spip_types_documents`.

←T→		id_type	titre	descriptif	extension	mime_type	inclus	upload	maj
Edit	Delete	59	RealAudio		rm		embed	oui	20020829183952
Edit	Delete	58	RealAudio		ram		embed	oui	20020829183952
Edit	Delete	53	PDF		pdf		non	oui	20020829183952
Edit	Delete	3	GIF		gif		image	oui	20020829183952

- `id_type`. Laissez ce champ vide. La numérotation des `id_type` est effectuée automatiquement par mySQL.

- `titre`. Indiquez ici le nom du type de fichier (souvent il s'agit du nom du programme qui permet de créer et lire ce type de fichier). Choisissez un nom court et aussi générique que possible. Sur un site multilingue, prenez soin à ne pas utiliser un titre propre à une langue (par exemple, un visiteur anglophone ne sera pas tellement intéressé par un fichier indiqué comme étant une « image vectorielle pour Illustrator » ; on se contentera donc d'indiquer « Adobe Illustrator »).

- `descriptif`. Laissez vide ; ce champ n'est pas utilisé.

- `extension`. Ce champ est le plus important : c'est là qu'on indique le type de fichier identifié par son extension. Par exemple, « rm » et « ram » pour du Real, « pdf » pour un fichier Acrobat PDF, « gif » pour une image au format GIF...

- `mime_type`. Laissez vide ; ce champ n'est pas utilisé.

- `inclus`. Le choix est laissé entre : « embed », « non », « image ». Prenez bien soin à effectuer ici le bon choix (c'est très important pour le bon fonctionnement de votre site) ; il détermine de quelle manière ce type de document sera « appelé » dans votre site pour être présenté aux visiteurs :

  - « non » : ce type de fichier ne peut pas être inclus directement *à l'intérieur* d'une page HTML ; on ne peut que le présenter au travers d'un lien hypertexte. Par exemple, un fichier PDF ne peut pas être affiché à l'intérieur d'une page HTML : le seul moyen de le joindre est de créer un lien hypertexte permettant d'ouvrir le fichier dans une nouvelle fenêtre (ou de le télécharger sur le disque dur du visiteur) ; un fichier de type « pdf » se voit donc attribuer le champ `inclus` fixé à « non » ;
  - « embed » : ce type de fichier peut être directement affiché à l'intérieur d'une page HTML où il sera lu grâce à une extension du butineur (plug-in...). C'est le cas de la majorité des formats multimédia utilisés sur le Web : Flash, Shockwave, films vidéo...

- « image » : il s'agit de formats d'image affichés directement dans la page HTML sans extension particulière (avec le code HTML `<img src=...>`). A priori, vous n'aurez pas besoin d'ajouter de tels types de fichiers, la liste fournie par SPIP étant déjà exhaustive. (Notez bien : certains formats d'images réalisés avec des logiciels de « dessin » ne peuvent pas être affichés directement en tant que « image » et nécessitent une extension pour être lus ; de tels formats seront alors « embed », voire « non ».)

- `upload`. Indique que vous autorisez l'installation de ce type de fichier via l'interface Web de SPIP. On choisit donc « oui ».

- `maj`. Ce champ est géré automatiquement par SPIP. Laissez vide.

* *

Cette opération effectuée, vous pouvez créer une nouvelle icone (vignette de prévisualisation) correspondant à ce type de fichier. Pour assurer la cohérence graphique avec les icones livrées par défaut avec SPIP, cette vignette aura une taille d'environ 48 pixels de large et 52 pixels de haut.

Vous pouvez utiliser tout type de format (GIF, PNG, JPG) ; préférez un format autorisant un fond transparent.

- Le nom de votre fichier sera formé ainsi :
  - l'extension du type de fichier autorisé ;
  - le format graphique de cette vignette (« .gif », « .png »...) ;
  - par exemple, une vignette sauvegardée en PNG, créée pour le format PDF, sera nommée : « pdf.png ».
  - N.B. Les noms se terminant par « -dist » sont réservés aux fichiers distribués avec SPIP. Ainsi, n'utilisez pas le nom « pdf-dist.png », ce nom est réservé au fichier créé par les développeurs de SPIP ; si vous utilisez tout de même ce nom, votre fichier risque d'être écrasé lors de votre prochaine mise-à-jour de SPIP.
- Cette vignette s'installe par FTP dans le dossier `/IMG/icones`.

Rechercher avec SPIP

Comment fonctionne le moteur de recherche de SPIP ?

Mai 2002 — maj : Décembre 2007

Le moteur de recherche intégré à SPIP est très simple d'utilisation, et cependant relativement puissant. Même si la plupart des utilisateurs n'ont aucune raison de se demander « comment ça marche ? », nombreux sont les courriers qui demandent des précisions sur son fonctionnement...

Voici les principes sur lesquels repose le moteur de SPIP.

Afin d'être rapide et efficace (c'est-à-dire pour qu'il donne des réponses pertinentes), le moteur de SPIP utilise un système d'**indexation** des contenus. L'indexation consiste à analyser tous les textes contenus dans SPIP, à en extraire tous les mots, et à sauvegarder pour chaque mot l'endroit où il apparaît.

Comme l'index dans un livre vous présente les mots importants du livre, suivis des numéros des pages où les retrouver, l'index du moteur de recherche fait correspondre à chaque mot utilisé sur le site le numéro de l'article, de la brève... où le retrouver.

Ensuite, lorsqu'on utilise le moteur pour effectuer une recherche, SPIP n'a plus besoin de consulter l'ensemble des textes du site, mais simplement d'aller consulter l'index pour savoir où se trouvent ces mots. On gagne ainsi énormément de temps (et plus le site est gros, plus on gagne de temps).

L'indexation

Le principe est donc le suivant : prendre un texte (plus ou moins long), en extraire tous les mots, et noter chacun de ces mots dans une base de données, en faisant correspondre ce mot à l'endroit où il se trouve.

Par exemple, notre site a trois articles, dont les textes (très courts) sont :
- *article 1* : « Le petit chat est mort de froid et de faim. »
- *article 2* : « Le gros chat est rentré à la maison. »
- *article 3* : « La maison résiste au froid. »

Nous allons extraire les mots de chaque article, et enregistrer pour chaque mot à quel article il correspond (nous ne prendrons que les mots de plus de trois lettres, nous expliquerons plus loin pourquoi) :
- *petit* : 1
- *chat* : 1, 2
- *mort* : 1
- *froid* : 1, 3
- *faim* : 1
- *gros* : 1
- *rentré* : 2
- *maison* : 2, 3
- *résiste* : 3

Et ainsi de suite, en considérant que notre site est certainement beaucoup plus gros, et les articles beaucoup plus long.

Si l'on recherche le mot *chat* :
- une solution sans indexation consisterait à relire tous les articles, pour y trouver le mot *chat* ; sur un gros site, même pour un ordinateur, cela prend beaucoup de temps ;
- puisque nous avons un index, il suffit de consulter l'entrée *chat* : on sait immédiatement qu'il se trouve dans les articles 1 et 2.

La pondération

À l'indexation s'ajoute un deuxième principe : la pondération. Il s'agit d'essayer de rendre le moteur plus pertinent. Par exemple, si un mot apparaît dans le titre d'un article, et dans le corps du texte d'un autre article, on considère que si l'on recherche ce mot, il faut en premier indiquer celui où il apparaît dans le titre. De plus, si un mot apparaît 25 fois dans un article, et seulement deux fois dans un autre, on veut afficher en premier l'article où le mot est le plus fréquent.

On voit que l'indexation simple ne suffit pas. Si on recherche *chat*, on trouvera les articles où il apparaît, mais sans pouvoir ensuite classer ces articles entre eux (selon que le mot chat apparaît une fois ou 20 fois, ou s'il se trouve dans le titre ou seulement dans le texte...).

SPIP va donc calculer une pondération pour chaque mot dans chaque article. C'est-à-dire donner des points à ce mot en fonction de l'endroit où il se trouve, et du nombre de fois où il apparaît :

dans le titre	8 points
dans le soustitre	5 points
dans le surtitre	5 points

dans le descriptif	4 points
dans le chapo	3 points
dans le texte	1 point
dans le post-scriptum	1 point

Si le mot apparaît plusieurs fois, on additionne les occurences.

Par exemple, si dans un article, le mot *chat* apparaît :
- une fois dans le titre : 8 points
- deux fois dans le descriptif : 2 fois 4 = 8 points
- six fois dans le texte : 6 fois 1 = 6 points
- *total* : 8 + 8 + 6 = 22 points.

Le mot *chat*, dans l'index, est donc ainsi enregistré :
- *chat*, dans l'article numéro X, 22 points ;
- *chat*, dans l'article numéro Y, 15 points ;
- ...

Si l'on recherche le mot *chat*, grâce à l'index, on saura donc qu'il se trouve dans les articles X et Y, et de plus on peut classer ces articles entre eux : 22 points dans X, 15 points dans Y, donc on considère que l'article X répond mieux à la recherche.

Les mots-clés : beaucoup d'utilisateurs confondent les mots-clés et l'indexation. Les mots-clés n'ont, par nature, aucun rapport avec l'indexation : quand SPIP effectue une recherche, il ne recherche pas dans les mots-clés associés à des articles (ce serait très limité), il recherche dans l'index qu'il a fabriqué à partir du texte exact des articles. Donc, dans le principe même de l'indexation, les mots-clés n'interviennent pas du tout.

En revanche, les mots-clés sont tout de même utilisés pour fabriquer la pondération des articles. Si un mot-clé est associé à un article, il entre alors dans l'indexation de l'article lui-même, avec une forte pondération (12 points pour le nom du mot-clé, 3 points pour son descriptif). Ainsi, si notre article Y (15 points en prenant simplement compte son contenu propre) est associé au mot-clé *chat* (parce qu'on indique par ce mot que c'est le sujet de cet article), il faut ajouter à l'indexation de cet article, pour le mot *chat*, les 12 points du mot-clé : total 27 points. Dans notre recherche sur *chat*, l'article Y devance désormais l'article X.

Notons enfin que tous les éléments de contenu de SPIP font ainsi l'objet d'une indexation : les articles, les brèves, les rubriques, les auteur, les mots-clés, les sites référencés (si le site est une syndication, les titres des articles récupérés sur ce site entrent également dans l'indexation).

Où est-ce stocké ?

Les données de l'indexation sont directement stockées dans la base de données. Cette partie est un peu technique, aussi je ne m'étendrai pas longtemps sur le sujet.

Sachez simplement qu'il y a plusieurs index (listes des mots), correspondant chacun à un type de contenu (un index pour les articles, un index pour les rubriques, un index pour les brèves...). De plus, contrairement à l'explication ci-dessus, où les entrées de l'index sont des mots, dans SPIP les entrées des index sont des nombres calculés à partir de ces mots (des *hachages*) ; une autre table de la base de données stockant par ailleurs les correspondance entre les véritables mots et ces nombres ; cette méthode permet d'accélérer les recherches dans les index (il est plus rapide pour le logiciel de rechercher des nombres plutôt que des mots).

Notez surtout que la taille des index est très importante. Sur uZine, par exemple, la partie de la base de données consacrée au stockage des articles pèse 9,7 Mo. La partie qui stocke l'index correspond aux articles pèse 10,5 Mo. Et la partie qui stocke la correspondance entre les mots et leur traduction

en nombres 4,1 Mo. Ainsi, pour un site dont les articles occupent 9,7 Mo, l'indexation de ces articles représente, à elle seule, près de 14,6 Mo. L'espace nécessaire à ces articles et à la recherche a donc plus que doublé la taille occupée par le site. C'est l'une des raisons, notamment, pour lesquelles on peut préférer désactiver le moteur de recherche, si l'on a d'importantes limitations en matière d'hébergement.

Quels mots sont indexés ?

Nous l'avons vu, tous les mots de tous les éléments de contenu du site sont extraits, puis analysés (pour pondération). Cependant, SPIP ne va pas retenir tous les mots.

- Les codes HTML sont exclus de l'indexation. Cela est évidemment nécessaire pour obtenir des recherches « propres »...

- Les mots de moins de 4 lettres ne sont pas retenus (en réalité, de moins de 3 lettres, mais les mots de 3 lettres ne sont pour l'instant pas exploités réellement lors des recherches). Ce point soulève beaucoup de questions de la part des utilisateurs...

Le problème est d'obtenir des résultats aussi pertinents que possible. Il faut donc privilégier, dans les recherches, les mots réellement importants dans le sens de la recherche. Par exemple, si l'on recherche les mots *le chat*, le mot important est *chat*, et non *le*...

Reprenons notre premier exemple (avec trois articles constitués chacun d'un phrase). Si l'on avait indexé tous les mots, nous aurions notamment les mots :
- *le* : 1, 2
- *est* : 1, 2
- ...

En recherchant les mot *le froid est dangereux*, nous trouverions les entrées :
- *le* : 1, 2
- *froid* : 1,3
- *est* : 1, 2
- *dangereux* : nulle part.

En ajoutant les résultats de ces mots pour chaque article (en réalité, la pondération de chaque article, mais considérons pour notre exemple que chaque mot a une pondération d'un seul point), on obtiendrait :
- article 1 : 3 mots
- article 2 : 2 mots
- article 3 : 1 mot.

Le classement mettrait donc en tête l'article 1, puis l'article 2, puis l'article 3. Or, l'article 2 ne parle pas de froid, contrairement à l'article 3. À cause de l'utilisation des mots sans importance pour le sens (*le*, *est*), le classement est faussé.

D'où le besoin, dans l'indexation, de ne pas tenir compte des mots qui n'entrent pas dans le sens de la recherche. La méthode la plus propre consiste à fournir au système une liste de mots à ne pas indexer ; cependant, cela nécessite un travail énorme, c'est-à-dire la constitution de dictionnaires d'exclusion (et cela dans toutes les langues)... Donc, plus simplement, dans SPIP nous choisissons de considérer que les mots de trois lettres et moins ne sont pas « importants » ; évidemment, il y a de la casse, puisque des mots comme *Val*, *mer*, *sud*... ne sont plus pris en compte ; c'est donc un compromis, qui se juge sur l'efficacité des recherches (qui sont globalement de bonne qualité).

N.B. Depuis la version 1.6 de SPIP, les sigles de deux lettres et plus, y compris ceux contenant des chiffres (G8, CNT, PHP, ONU...), sont indexés. Un sigle est un mot ne comprenant aucune minuscule.

Quand a lieu l'indexation ?

L'indexation a lieu à trois moments différents :

- lorsque vous validez un article, celui-ci est immédiatement indexé ;

- lorsque vous modifiez un article déjà publié, il est à nouveau indexé ;

- lors de la visite du site public, si un élément accessible au public n'est pas indexé (par exemple, lorsque vous venez d'effacer les données d'indexation depuis l'espace privé, ou lorsque vous venez de rétablir une sauvegarde de votre site - les index ne sont pas sauvegardés -, ou si vous avez activé le moteur de recherche après avoir déjà publié des articles), alors il est indexé (en tâche de fond).

Retenez que l'opération d'indexation est relativement lourde : elle demande de nombreux calculs (calculs peu complexes, mais effectués sur tous les mots de l'article), et provoque de très nombreux appels à la base de données. Là aussi, chez un hébergeur très lent (vraiment très lent !), il peut être préférable de désactiver le moteur de recherche.

Retenez également que, si vous activez le moteur après avoir déjà publié des articles, ceux-ci ne sont pas immédiatement indexés : ce seront les visites sur le site public qui provoqueront leur indexation. Sur un gros site, cela peut prendre un certain temps.

La recherche

Puisque tous les documents sont indexés, on peut désormais effectuer des recherches.

Si vous recherchez un seul mot...

SPIP va consulter l'index, et trouver l'entrée correspond à ce mot. Pour le mot *chat*, nous avions trouvé les articles X et Y. SPIP va de plus récupérer le nombre de points attribués à ce mot pour chaque article (22 points dans X, et 27 points pour Y). On peut donc classer nos résultats : l'article Y, puis l'article X.

Si vous recherchez plusieurs mots...

SPIP n'autorise pas les constructions du type « ET », « OU », il ne fonctionne pas de cette manière.

Lorsque vous recherches plusieurs mots, SPIP va effectuer l'opération de recherche pour chaque mot, récupérer les points de chaque article (ou brève, ou rubrique, etc.), et ajouter ces points.

Recherchons par exemple les mots *chat*, *gros*, *maison*. On obtient les résultats suivants pour chaque mot :
- *chat* : article X (22 points), article Y (27 points),
- *gros* : article X (12 points), article Y (2 points), article Z (5 points),
- *maison* : article Y (3 points), article Z (17 points).

SPIP fait l'addition de tous ces points pour chaque article :
- article X : 22 + 12 = 34 points,
- article Y : 27 + 2 + 3 = 32 points,
- article Z : 5 + 17 = 22 points.

Le classement des articles, pour la recherche *chat, gros, maison*, est : article X, puis article Y, puis article Z. (Dans les squelettes, on peut d'ailleurs demander l'affichage des points de chaque résultat grâce à la balise `#POINTS`, voir l'article « Les boucles et balises de recherche ».)

Ca n'est donc pas une recherche de type « ET » ni « OU », c'est une addition de points. Et à l'usage, cela se montre plutôt efficace (on trouve ce que l'on cherche, ce qui est bien le but d'un moteur...).

Le moteur de recherche

Juin 2001 — maj : Décembre 2007

SPIP intègre un moteur de recherche, désactivé par défaut. Ce moteur, lorsqu'il est activé par un administrateur dans la page de configuration, permet d'effectuer des recherches sur différents types d'informations présentes dans la base de données : les articles, les rubriques, les brèves, les mots-clés et les auteurs. Depuis SPIP 1.7.1 les fils de discussion des forums (threads) et les signatures de pétitions sont également indexés.

Principe

Il y a deux grandes façons de faire un moteur de recherche. La première est de chercher tout bêtement dans le type de stockage existant (fichiers HTML, base de données... selon le type de site). Cette méthode est très lente car le type de stockage n'est pas prévu à cet effet.

La seconde méthode, qui a été choisie pour SPIP (et qui est aussi celle de tous les moteurs professionnels), est d'établir un mode de stockage spécifique aux besoins de la recherche. Par exemple, le score de chaque mots d'un article peut être stocké directement afin d'être facilement retrouvé, et d'avoir le score total d'une recherche par une simple addition. L'avantage est que la recherche est très rapide : presque aussi rapide que n'importe quel calcul de page. L'inconvénient est qu'il faut une phase de construction du dit stockage des informations : cela s'appelle l'indexation. L'indexation a un coût en termes de ressources (temps de calcul et espace disque), et elle introduit également un léger décalage temporel entre l'ajout ou la modification d'un contenu, et la répercussion de cet ajout ou de cette modification sur les résultats de recherche.

D'autre part, dans le cas de SPIP, nous sommes obligés d'utiliser PHP et MySQL comme pour le reste du logiciel, ce qui ne permet pas de réaliser un moteur très performant, en termes de rapidité, mais aussi de pertinence ou d'enrichissements divers (indexation de documents extérieurs au site, création de champs sémantiques permettant de proposer des recherches plus fines, etc.).

L'avantage du moteur interne, cependant, c'est qu'il permet de gérer l'affichage des résultats à travers les mêmes méthodes (squelettes) que le reste des pages de SPIP, et à l'intérieur du même environnement visuel.

L'indexation

L'indexation est réalisée lors des visites du site public. Afin d'éviter que le cumul d'une indexation et d'un calcul de page ne mène à un *timeout* sur les serveurs particulièrement lents, SPIP attend qu'une page soit affichée en utilisant le cache [1].

L'indexation traite une à une les différentes données textuelles d'un contenu donné : par exemple, pour un article, le chapo, le descriptif, le titre, le texte... Pour chaque donnée textuelle, le score de chaque mot est calculé en comptant simplement le nombre d'occurrences. A cet effet, les mots de trois caractères ou moins sont ignorés (ils sont, pour la plupart, non significatifs, et alourdiraient la base de données) ; d'autre part, les caractères accentués sont translittérés (convertis en leurs équivalents non-accentués), pour éviter les problèmes de jeux de caractères et aussi pour permettre d'effectuer des recherches en version non accentuée.

Ensuite, les scores de chaque mot sont cumulés, de façon pondérée, entre les différentes données textuelles du contenu indexé. La pondération permet, par exemple, de donner plus de poids aux mots présents dans le titre d'un article que dans le corps du texte ou le post-scriptum...

Les fonctions d'indexation peuvent être étudiées au sein du fichier `ecrire/inc_index.php3`.

Pour mieux visualiser la dynamique d'indexation du site, vous pouvez ouvrir le fichier `ecrire/data/spip.log`, ou encore regarder la page `ecrire/admin_index.php3` (nota : cette page, encore expérimentale, n'est pas livrée avec toutes les versions de SPIP, et n'existe qu'en français. Elle a été retirée de [SPIP 1.9] et est désormais disponible dans le plugin recherche_etendue avec d'autres fonctions des gestion de l'indexation).

Dans la version [SPIP 1.6], d'importantes modifications ont été apportées au comportement du moteur :
- meilleur comportement dans un environnement multilingue ;
- le tiret bas (*underscore*) n'est plus considéré comme un séparateur de mot, mais comme un caractère alphabétique (utile pour de la documentation informatique) ;
- les mots de deux lettres (et plus) ne contenant que des majuscules et des chiffres sont considérés comme des sigles, et sont indexés, ce qui supprime l'un des principaux inconvénients de la limitation de l'indexation aux mots de plus de 3 lettres (G8, CNT, ONU sont désormais indexés).

La recherche

La recherche s'effectue simplement en séparant le texte de recherche en ses différents mots ; le même filtre est appliqué que lors de l'indexation : suppression des mots de trois lettres ou moins (sauf sigles), et translittération.

Pour chaque contenu recherché, le score des différents mots est ensuite récupéré puis additionné afin d'obtenir le score total. Enfin, les résultats sont en général affichés par ordre décroissant de score (`{par points}{inverse}`), c'est-à-dire de pertinence (mais cela est laissé à la volonté de la personne qui écrit les squelettes de mise en page).

Performances

Rapidité

Sur un serveur récent et pas trop chargé, l'indexation d'un texte long (plusieurs dizaines de milliers de caractères) prendra entre une et deux secondes : l'attente est presque imperceptible, comparée aux délais de chargement via le réseau. Les contenus courts sont indexés de façon quasi-instantanée. Bien sûr, ces affirmations doivent être modulées selon la taille du site. Un site vraiment très gros risque de voir les temps d'indexation s'allonger légèrement ; pour relativiser, signalons qu'un site comme Le Courrier des Balkans comporte, à la date d'écriture de ce texte, environ 3 800 articles publiés, et plus de 7500 messages de forum, et que le moteur de recherche de SPIP ne donne aucun signe de faiblesse.

Par ailleurs, statistiquement, on peut considérer de façon approximative que chaque contenu ne sera indexé qu'une seule fois : compte tenu qu'il y a en général beaucoup plus de visites sur un site que de mises à jour de contenus, le surcroît de charge du serveur apparaît négligeable.

Qualité

La qualité de l'indexation est plus faible que sous des moteurs de recherche professionnels. PHP étant un langage plutôt lent, la phase d'extraction des mots a dû être simplifiée au maximum pour que les temps d'indexation restent minimes. Par conséquent, les données d'indexation comportent quelques « déchets », c'est-à-dire des morceaux de texte qui ne correspondent pas à de « vrais » mots, mais ont été indexés comme tels (il s'agit souvent de contenus techniques comme des noms de fichiers, ou de passages à la ponctuation malmenée). L'exemple d'uZine, où l'on constate environ 2 % de tels « déchets », nous laisse cependant penser que ces données sont quantité négligeable, d'autant qu'il y a peu de chance qu'elles déclenchent un résultat positif lors d'une recherche.

La recherche n'offre pas d'opérateurs booléens, l'opérateur implicite étant grosso modo un « OU » logique. Cependant, depuis SPIP 1.7.1, les articles trouvés s'affichent dans un ordre qui privilégie les résultats contenant le plus de mots orthographiés précisément selon la requête. Ainsi, une requête sur « la main rouge » mettra en évidence les articles contenant « main » et « rouge », loin devant les articles ne contenant que « maintenance » ou « rouget » - ceux-ci apparaîtront, mais plus loin dans le classement.

Espace disque

MySQL n'étant pas spécialement conçu pour le stockage de données d'indexation, l'utilisation du moteur de recherche a tendance à faire beaucoup grossir l'espace disque utilisé par la base de données. Pour donner quelque précision, disons qu'un contenu génère des données d'indexation de taille comprise entre la taille du contenu et le double de celle-ci. Donc, si l'on fait abstraction des données ne donnant pas lieu à indexation (les forums par exemple), l'indexation fait entre doubler et tripler la place prise par la base de données. Cela peut être gênant si la place vous est très comptée.

Si jamais vous désactivez le moteur de recherche afin d'économiser de l'espace disque, n'oubliez pas ensuite d'effacer les données d'indexation (dans la page de sauvegarde/restauration de la base de données) afin de réellement libérer l'espace disque occupé par ces données.

Notes

[1] Si, donc, vous avez mis tous les `$delais` à zéro, ou si votre site n'est pas visité (site de test), l'indexation n'aura pas lieu.

Les boucles et balises de recherche

Mai 2001 — maj : Décembre 2007

SPIP dispose d'un moteur de recherche intégré. Il faut donc prévoir une page permettant d'afficher les résultats des recherches.

Le formulaire de recherche

Pour afficher le formulaire de recherche, il suffit d'insérer la balise :

```
#FORMULAIRE_RECHERCHE
```

Le formulaire renvoit par défaut vers `spip.php?page=recherche` ; vous devez donc réaliser un squelette `recherche.html` permettant d'afficher les résultats.

Vous pouvez décider d'utiliser une autre page d'affichage des résultats. Pour cela, il faut utiliser la balise de la manière suivante :

```
[(#FORMULAIRE_RECHERCHE|spip.php?page=xxx)]
```

où `spip.php?page=xxx` est la page vers laquelle vous désirez envoyer l'utilisateur, et `xxx.html` est son squelette.

> *Historique :* Dans les versions antérieures à SPIP 1.9 il faut écrire `xxx.php3` et non `spip.php?page=xxx`
>
> De façon générale jusqu'à SPIP 1.9, les urls des pages générées par SPIP étaient de la forme `http://monsite.net/xxx.php3` et non pas `http://monsite.net/spip.php?page=xxx`.

Le squelette des résultats

Les boucles permettant d'afficher les résultats de la recherche sont, en réalité, des boucles déjà abordées ici : ARTICLES, RUBRIQUES, BREVES et depuis [SPIP 1.8.2] FORUMS. Vous pouvez en effet effectuer des recherches non seulement sur les articles, mais aussi sur les rubriques, les brèves et les forums.

- La seule différence, par rapport à ce qui est documenté dans le manuel de référence des boucles, est le choix du critère de sélection, qui doit être `{recherche}`. Les autres critères d'affichage et les balises de ces boucles restent ici valables.

- Cependant, afin de classer les résultats par pertinence, on utilisera de préférence ce nouveau critère d'affichage : `{par points}`. Les résultats sont en général affichés par ordre décroissant de score (`{par points}{inverse}`), c'est-à-dire de pertinence.

- Enfin, on pourra utiliser la balise `#POINTS`, qui affiche la pertinence des résultats (attention, dans l'absolu cette valeur n'est pas très explicite, elle est surtout utile pour le classement des résultats).

- Introduite par [SPIP 1.5.1], la balise `#RECHERCHE` affiche la requête formulée par le visiteur.

Exemple de boucle complète :

```
<h1><:resultats_recherche:>[ (#RECHERCHE)]</h1>


<B_articles>
<h2><:info_articles_trouves:></h2>
<ul>
 <BOUCLE_articles(ARTICLES) {recherche} {par points}{inverse}>
 <li>#POINTS <a href="#URL_ARTICLE">#TITRE</a></li>
 </BOUCLE_articles>
</ul>
</B_articles>
```

Surligner la requête dans les autres pages

- `#DEBUT_SURLIGNE`, `#FIN_SURLIGNE` sont deux balises qui indiquent à SPIP dans quelle partie de la page colorer les mots recherchés. Ces balises propres au site sont disponibles à n'importe quel endroit des squelettes.

Quand l'internaute arrive sur une page depuis la page des résultats de recherche, les mots cherchés sont automatiquement colorés dans la page trouvée. SPIP ne distingue pas, par exemple, entre un menu, du code javascript ou le texte de l'article, il colore les mots partout, ce qui peut être moche ou même créer des problèmes dans les scripts. On peut utiliser ces deux balises pour limiter la coloration à une partie de la page. Par exemple :

```
<BOUCLE_article(ARTICLES) {id_article}>
<INCLURE{fond=inc-menu-rubriques}>
#DEBUT_SURLIGNE
 <h1>#TITRE</h1>
```

#CHAPO

#TEXTE

#FIN_SURLIGNE

#NOTES

</BOUCLE_article>

- La variable de personnalisation `$nombre_surligne` représente le nombre maximum de fois où un mot recherché sera surligné dans le texte. Par défaut, cette valeur est définie à 4.

- L'apparence visuelle du surlignement peut être changée en modifiant la définition de style de `.spip_surligne` (voir : « Modifier l'habillage graphique »).

Multilinguisme

Réaliser un site multilingue

Octobre 2003 — maj : 14 avril

La modification la plus importante qu'apporte SPIP 1.7 est sa gestion « naturelle » des sites multilingues. La totalité de cet article de documentation concerne SPIP 1.7.

Préalable : qu'est-ce qu'un site multilingue ?

Il n'est pas question dans cet article de rédiger un tutoriel complet sur les sites multilingues : d'une part, il y a certainement plusieurs « visions » de ce qu'on appelle « multilinguisme » ; d'autre part, nous manquons toujours de recul pour pouvoir définir « la meilleure méthode ».

Vous trouverez donc ci-dessous une revue des différents outils que SPIP propose pour gérer des sites multilingues ; à vous de les utiliser, et discutons-en dans les espaces prévus à cet effet (forums, wiki, listes de discussion, etc.)

Mais avant de lire, oubliez un peu votre projet du jour, et pensez aux situations suivantes :
- un site de poésies, classées par thèmes (rubriques) ;
- un site de documentation pour, par exemple, un logiciel comme SPIP ;
- un site institutionnel en 25 langues ;
- un site corporate bilingue ;
- le site d'une association bulgare avec quelques pages en anglais.

Le site de poésie choisira plutôt ses langues article par article ; la documentation de SPIP, pour sa part, les ventile par « secteurs » (rubriques de premier niveau), et affiche les traductions disponibles pour chaque article, quand ces traductions sont disponibles. Le site institutionnel en 25 langues ne pourra sans doute pas fournir les 25 traductions en même temps, mais cherchera tout de même à conserver des arborescences parallèles ; le site corporate bilingue aura obligatoirement une traduction en face de chacun des articles, et une arborescence de rubriques en deux parties, la partie anglaise étant « clonée » sur la partie en auvergnat ; l'association bulgare affectera l'anglais à un secteur particulier de son site, le reste des secteurs étant tous en bulgare (par défaut).

Principe

Pour pouvoir autoriser ces situations différentes, et d'autres encore, le modèle mis en place dans SPIP consiste à déterminer une langue pour chaque article, chaque rubrique et chaque brève. Dans l'espace public comme dans l'espace privé, cette langue détermine le mode de correction typographique qui est appliqué aux textes ; dans l'espace public cela détermine également la langue des élements insérés par SPIP autour de ces « objets » : dates et formulaires principalement.

Pour créer un site multilingue avec SPIP, il faut d'abord configurer le site en conséquence : dans la configuration du site, à la section « langues » bien entendu. Là vous pourrez activer la gestion du multilingue, et choisir les langues que vous utiliserez sur votre site.

Configurer l'espace privé

Pour gérer plus facilement le site, on peut choisir dans la configuration du site avec quelle précision s'effectuera le réglage des langues, ce qui permet de masquer l'interface là où elle n'est pas nécessaire et de limiter les risques d'erreurs [1]. SPIP propose trois niveaux d'interface différents pour choisir les langues affectées aux articles (et brèves, etc.) ; par ordre croissant de complexité :

- **Par secteur** (rubrique de premier niveau) : à chaque secteur du site correspond une langue modifiable par les administrateurs, qui concerne toutes ses sous-rubriques ainsi que les articles et les brèves qui y sont publiés ; **ce réglage devrait satisfaire les besoins de la plupart des sites multilingues tout en conservant une structure et une interface simples.**

- **Par rubrique** : de manière plus fine, avec ce réglage, on peut changer la langue pour chacune des rubriques du site, pas seulement celles de premier niveau.

- **Par article** : la langue peut être modifiée au niveau de chaque article ; ce choix est compatible avec les précédents (on peut par exemple choisir la langue par rubrique mais appliquer des exceptions de-ci de-là à certains articles) et permet toutes les finesses imaginables, mais attention à ne pas produire un site à la structure incompréhensible...

Blocs multilingues

[SPIP 1.7.2] Certains objets, comme les auteurs ou les mots-clés, peuvent s'orthographier différemment selon qu'ils sont affectés à un article dans une langue ou dans une autre. Cependant, il serait absurde de concevoir des « traduction d'un mot-clé » ou « traduction d'un auteur », car c'est bien le même auteur qui signe les deux articles, ou le même mot-clé (même « concept ») qu'on leur attache. Ces objets n'ont donc pas de langue au sens de SPIP, mais il est tout de même possible, à l'aide des « blocs multi », de les faire s'afficher dans la langue du contexte dans lequel ils sont invoqués (pour le dire plus simplement : faire que le mot-clé « Irak » s'affiche « Iraq » quand il est affecté à un article en anglais).

Le « bloc multi » est un raccourci SPIP, dont la structure est relativement intuitive :
`<multi>chaîne 1 [xx] chaîne 2 [yy] chaîne 3 ...</multi>`

Pour reprendre l'exemple du mot-clé, son titre serait entré sous cette forme dans l'espace privé :

<multi>[fr]Irak [en]Iraq</multi>

Si un bloc multi est appelé à s'afficher dans une langue qui n'est pas prévue, c'est toujours la première partie du bloc qui s'affiche (« chaîne 1 » dans le premier exemple, « Irak » dans le second). Cela, afin de ne jamais avoir d'affichage vide [2].

NB : les blocs multi peuvent aussi être utilisés, selon la même structure, dans les squelettes, *cf.* Internationaliser les squelettes.

Boucles et balises : comment faire

Une fois l'espace privé réglé aux petits oignons, passons maintenant au site public. Hé oui, même si chaque article dispose maintenant de sa propre langue judicieusement choisie (selon le mécanisme expliqué plus haut), les squelettes doivent bien pouvoir en tenir compte dans l'affichage du site.

1. Une bonne nouvelle pour commencer : **le multilinguisme des squelettes est pour la plus grande part totalement naturel ; il n'est pas nécessaire de faire des squelettes différents pour afficher des articles de langues différentes. Un même squelette adapte *automatiquement* son affichage à la lange courante.**

Ainsi tous les éléments affichés autour et dans un article d'une langue donnée, seront affichés dans cette langue. Cela concerne aussi bien la date de publication de l'article que les formulaires de réponse au forum, de signature d'une pétition, etc. Plus généralement : toute balise SPIP incluse dans une boucle `ARTICLES` sera affichée dans la langue de l'article (de même pour les rubriques et les brèves).

Exemple : si votre page d'accueil contient un sommaire affichant les dix derniers articles publiés ainsi que leur date de publication, la date des articles en vietnamien s'affichera en vietnamien, celle des articles en créole de la Réunion s'afficheront en créole de la Réunion, etc.

> *Note : ce fonctionnement suppose que la langue de l'article fait l'objet d'une traduction dans SPIP. Ainsi, si un article est écrit en volapück mais que votre version de SPIP n'est pas encore traduite en volapück (nous vous invitons bien évidemment à corriger cette lacune en participant à l'effort de traduction), la date de l'article s'affichera en toutes lettres certes, mais dans une langue par défaut - le français probablement.*

2. Le sens de l'écriture

Si votre site contient des langues s'écrivant de gauche à droite (la plupart des langues) mais aussi des langues s'écrivant de droite à gauche (notamment l'arabe, l'hébreu ou le farsi), il faudra de petits compléments au code HTML pour que l'affichage se fasse sans accroc [3].

SPIP offre à cet effet une balise spécifique : `#LANG_DIR`, qui définit le sens d'écriture de la langue courante. Cette balise est utilisable comme valeur de l'attribut `dir` dans la plupart des tags HTML (cela donne donc « `ltr` » pour les langues s'écrivant de gauche à droite, et « `rtl` » pour les autres [4]).

Une boucle d'affichage du sommaire devient donc :

```
<ul>
<BOUCLE_sommaire(ARTICLES){par date}{inverse}{0,10}>
  <li dir="#LANG_DIR">[(#DATE|affdate)]:
  <a href="#URL_ARTICLE">#TITRE</a></li>
</BOUCLE_sommaire>
</ul>
```

Si la mise en page repose sur des éléments alignés à droite ou à gauche, ceux-ci devront être inversés pour les langues écrites de la droite vers la gauche : on peut tout de suite penser à remplacer *tous* [5] les éléments du squelette marqués `left` ou `right` par les balises `#LANG_LEFT` et `#LANG_RIGHT`. Pour ce qui est de la définition de la page elle-même, il est alors judicieux de commencer par indiquer la langue de l'élément demandé, et la direction générale de la page :

<html dir="#LANG_DIR" lang="#LANG">

<head>

...

</head>

<body>

...

</body>

</html>

3. Les liens de traduction

SPIP propose un système de traduction entre articles : on peut spécifier quelles sont les différentes traductions d'un article (note : ces traductions sont elles-mêmes des articles à part entière). Le critère `{traduction}` permet alors, dans une boucle `ARTICLES`, de récupérer toutes les versions d'un même article.

Par exemple, pour afficher toutes les traductions de l'article courant :

<BOUCLE_traductions(ARTICLES){traduction} {exclus}>

[(#LANG| traduire_nom_langue)]

</BOUCLE_traductions>

Notons le critère `{exclus}`, qui permet de ne pas afficher la version courante, et le filtre `| traduire_nom_langue` qui fournit le nom véritable de la langue à partir de son code informatique (cela permet d'afficher « français » au lieu de « `fr` », « English » au lieu de « `en` », etc.).

- Un critère complémentaire `{origine_traduction}` (pour les plus acharnés) permet de sélectionner uniquement la « version originale » de l'article courant.

Une page du wiki de spip-contrib rassemble des exemples de boucles utilisant ces critères : http://www.spip-contrib.net/-Carnet....

4. Éléments supplémentaires

[SPIP 1.7.2] introduit d'autres éléments permettant de fabriquer des sites multilingues :

— le critère `{lang_select}` sert à forcer la sélection de la langue pour la boucle (AUTEURS), qui normalement ne le fait pas (à l'inverse, le critère `{lang_select=non}` permet de dire aux boucles (ARTICLES), (RUBRIQUES) ou (BREVES) de ne pas sélectionner la langue).

— la variable de personnalisation `$forcer_lang` indique à SPIP qu'il doit vérifier si le visiteur dispose d'un cookie de langue, et si oui le renvoyer vers la page correspondante. C'est ce que fait la page de connexion à l'espace privé livrée en standard avec SPIP.

— les balises `#MENU_LANG` (et `#MENU_LANG_ECRIRE`) affichent un menu de langue qui permet au visiteur de choisir « cette page en ... ». La première balise affiche la liste des langues du site ; la seconde la liste des langues de l'espace privé (elle est utilisée sur la page de connexion à l'espace privé).

— enfin, les critères optionnels (*cf.* SPIP 1.7, SPIP 1.7.2) permettent d'utiliser une même boucle (en fait, un même squelette) pour afficher soit tous les articles du site dans toutes les langues, soit

seulement les articles dans la langue passée dans l'URL. Ça peut être utile dans les backend, par exemple, ou dans les boucles de recherche :

<BOUCLE_recents(ARTICLES){lang?}{par date}{inverse}{0,10}>

<BOUCLE_recherche(ARTICLES){lang?}{recherche}{par points}{inverse}{0,10}>

Des squelettes internationaux pour un site massivement multilingue

Ce qui précède nous a permis de rendre multilingue la partie proprement SPIP de notre squelette : tout ce qui est issu des boucles s'affiche dans le bon sens, avec la bonne typographie, et les éléments produits par SPIP (formulaires, dates...) sont dans la langue demandée.

Pour un site présentant un nombre modeste de langues (bilingue par exemple), ou pour lequel il existe une langue principale et quelques langues annexes, on pourrait en rester là. Les textes figés présents dans les squelettes, c'est-à-dire les mentions écrites directement dans le HTML comme « Plan du site », « Espace de rédaction », « Répondre à ce message »... peuvent dans certains cas rester dans une seule langue ; ou alors, un site bilingue pourra utiliser des squelettes séparés pour chacune des deux langues.

Cependant, si vous voulez réaliser et gérer efficacement un site présentant beaucoup de langues à part entière, il devient illusoire de maintenir des squelettes séparés, ou d'imposer une navigation dans une langue unique (même en anglais ou en espéranto...). Pour réaliser un jeu de squelettes unique fonctionnant dans toutes les langues, il faut internationaliser les squelettes afin de modifier les textes indépendamment du code HTML qui les contient (qui reste, lui, figé d'une langue à l'autre). Cette tâche nécessite de mettre un peu « les mains dans le cambouis » et fait l'objet d'un article séparé.

Détails annexes

- Les raccourcis typographiques `<code>` et `<cadre>` produisent toujours un texte écrit de gauche à droite, même si la langue de l'article s'écrit normalement de droite à gauche. En effet ces deux raccourcis sont destinés à afficher du code ou des données informatiques, qui sont à peu près toujours écrits de gauche à droite (et, la plupart du temps, en caractères occidentaux).

- Toujours en ce qui concerne le sens d'écriture, notons que les attributs `left` et `right` du HTML sont aussi souvent présents dans les feuilles de style. Cela veut dire que vous devrez peut-être inclure la partie correspondante de la feuille de style dans vos squelettes (pour utiliser les balises `#LANG_LEFT` et `#LANG_RIGHT`) plutôt que de la placer dans un fichier séparé.

Voici un récapitulatif du comportement des balises SPIP liées au sens de l'écriture :

Langue	#LANG_LEFT	#LANG_RIGHT	#LANG_DIR
langues écrites de gauche à droite	left	right	ltr
arabe,farsi,hébreu...	right	left	rtl

- Un filtre `|direction_css` permet d'« inverser » un fichier CSS pour les langues s'écrivant de droite à gauche. Si la feuille de style CSS à inverser s'appelle par exemple *style.css*, ce filtre utilise (dans le cas où la langue courante s'écrit de droite à gauche) une éventuelle feuille *style_rtl.css* ; si celle-ci n'existe pas, il crée automatiquement une feuille RTL en remplaçant toutes les occurrences de *left* par *right* et vice-versa (et la stocke dans le répertoire `IMG/cache-css/`). Il s'applique le plus souvent sur une balise `#CHEMIN`, de cette façon : `[(#CHEMIN{style.css}|direction_css)]`.

Notes

[1] On précise ici que dans la configuration livrée d'origine, SPIP reste monolingue, afin de ne pas compliquer du tout l'interface.

[2] Si l'on veut au contraire un affichage vide, il faut créer explicitement une première partie vide avec un nom de langue quelconque.

[3] Théoriquement le HTML devrait régler tous ces détails automatiquement, mais le résultat n'est pas toujours à la hauteur, surtout lors des passages à la ligne ou si l'on mélange des langues utilisant un sens d'écriture différent.

[4] Malheureusement, les instances de normalisation semblent pour l'instant ignorer le boustrophédon, ce qui empêche son utilisation en HTML.

[5] Tous, ou presque tous, nous vous laissons découvrir si votre mise en page présente des cas particuliers...

Internationaliser les squelettes

Octobre 2003 — maj : Décembre 2007

L'internationalisation des squelettes, abordée dans cet article est disponible à partir de SPIP 1.7.

Pourquoi créer des squelettes multilingues ?

Dès la mise en ligne de documents, SPIP adapte certaines informations « automatiques » dans la langue désirée. Notamment les dates sont affichées dans la langue du site, ou d'un article, ou d'une rubrique, les formulaires sont affichés dans la langue correspondante (par exemple l'interface pour poster des messages de forums)... Tout cela est d'ores et déjà traduit par SPIP.

Ce n'est cependant pas suffisant : les webmestres insèrent dans leurs squelettes un certain nombre d'informations, décrivant notamment les principes de navigation dans le site. Il est nécessaire, par exemple, d'afficher des textes du style « Plan du site », « Répondre à cet article », « Articles du même auteur », « Dans la même rubrique »... Pour un site dans une seule langue, ces différents éléments sont faciles à insérer : on les insère tels quels dans le code HTML des squelettes. Le problème apparaît lorsque le site est multilingue : sous un article en français, on veut afficher *« Répondre à cet article »*, mais sous un article en anglais, on a besoin d'afficher un autre texte (*« Comment on this article »*).

[SPIP 1.7.2] propose trois méthodes pour gérer ces éléments de texte différents selon les langues :

— (1) une méthode consistant à stocker les éléments de texte des squelettes dans des *fichiers de langue* (un fichier différent par langue utilisée sur le site), séparés des squelettes ; un squelette *unique* (par exemple `article.html` appelant, selon des codes définis par le webmestre, ces éléments de texte en fonction de la langue utilisée ; de cette façon, un même squelette `article.html` affichera automatiquement le texte « Répondre à cet article » ou « Comment on this document » en fonction de la langue de l'article. Cette méthode est vivement conseillée, elle offre le plus de souplesse, elle facilite les mises à jour du site (on travaille avec un squelette unique qui gère automatiquement plusieurs langues), et à terme des outils seront ajoutés à SPIP facilitant le travail collectif de traduction de l'interface de votre site (plusieurs administrateurs, parlant chacun

une langue différente, pourront *traduire* l'interface d'un même site depuis l'espace privé, sans avoir besoin d'intervenir sur les fichiers des squelettes) ;

— (2) une méthode plus rapidement accessible, techniquement très simple, reposant sur la création de fichiers de squelettes différents pour chaque langue. Dans cette méthode, on fabrique un fichier `article.html` pour gérer les articles en français, et un fichier `article.en.html` pour les articles en anglais (note : `article.html` gère en réalité toutes les langues sauf l'anglais). Cette méthode est déconseillée si le site utilise de nombreuses langues et/ou si on utilise des squelettes distincts selon les rubriques. Par ailleurs, elle est dans tous les cas avantageusement remplacée par la méthode 3 décrite ci-dessous :

— (3) la méthode des « blocs multilingues », introduite par [SPIP 1.7.2], fonctionne aussi bien dans les contenus que dans les squelettes. Il suffit de mettre dans le squelette la construction suivante :

<multi>

[fr] Répondre à cet article

[en] Comment on this article

</multi>

et la phrase s'affichera dans la langue voulue. Si ce système est très souple, il atteint toutefois ses limites dès que le nombre de langues est important, et que l'on souhaite que différents traducteurs interviennent dans le site (il faut en effet qu'ils puissent modifier le squelette, ce que la méthode des fichiers de langue permet d'éviter).

1. Méthode des fichiers de langue

Le principe des fichiers de langue consiste à insérer dans un squelette unique un *code*, lequel correspondra dans chaque langue à un élément de texte (une « chaîne ») ; entre les différentes langues le code ne varie pas, mais le texte est traduit.

Par exemple, nous pouvons décider que le *code* `telechargement` correspond :
— en français, à la chaîne *« télécharger ce fichier »*,
— en anglais, à la chaîne *« download this file »*,
— en espagnol, à la chaîne *« descargar este archivo »*,
— etc.

Dans le fichier de squelette des articles (un seul fichier gérera toutes les langues), `article.html`, il suffit d'insérer le code (notez la syntaxe) :

<:telechargement:>

Lors de l'affichage d'un article, ce *code* sera remplacé par sa traduction dans la langue de l'article.

Par exemple, dans le squelette `article.html`, nous insérons dans la boucle affichant les documents associés à l'article, le code suivant :

<:telechargement:>

Si l'article en question est en français, cela produira :

```
<a href="/IMG/jpg/mondocument.jpg">télécharger ce fichier</a>
```

si l'article est en anglais :

```
<a href="/IMG/jpg/mondocument.jpg">download this file</a>
```

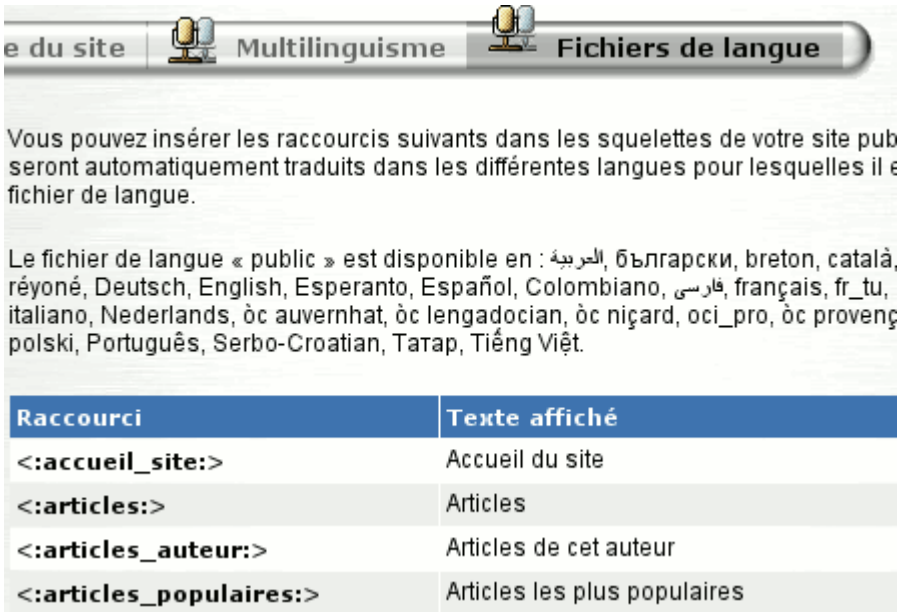
et ainsi de suite. Un unique squelette, contenant un unique code, affiche un texte traduit dans toutes

les langues utilisées sur le site.

- **Utiliser des textes déjà traduits**

Pour faciliter le travail des webmestres, SPIP fournit un ensemble de chaînes déjà traduites (par les traducteurs de SPIP). En utilisant ces chaînes, correspondant à des éléments de texte fréquemment utilisés sur des sites Web, le webmestre peut rapidement réaliser une interface fonctionnant dans différentes langues, mêmes celles qu'il ne parle pas lui-même.

Vous pouvez lister les chaînes disponibles depuis l'espace privé : allez dans la section « Gestion des langues » de la partie « Administration du site », puis cliquez sur l'onglet « Fichiers de langues ». Vous n'avez plus qu'à y piocher les codes de votre choix pour la réalisation de vos squelettes.



Les fichiers de langue dans l'espace privé

Exemple : un webmestre veut réaliser l'interface pour un site en français, en espagnol et en arabe, mais il ne parle pas lui-même l'espagnol et l'arabe. En insérant dans ses squelettes les codes livrés avec SPIP, il n'a pas à se soucier d'obtenir des traductions en espagnol et en arabe, car le travail de traduction a déjà été effectué en amont par les traducteurs de SPIP ; ainsi, en mettant au point l'interface en français avec les codes fournis par SPIP, il sait que ses pages s'afficheront immédiatement en espagnol et en arabe.

Si par la suite on ajoute des articles en polonais, ils seront immédiatement affichés avec les éléments de texte traduits en polonais, sans que le webmestre ait à intervenir à nouveau.

> Autre avantage de cette méthode : elle facilite la création de squelettes « à distribuer » immédiatement multilingues. Des squelettes réalisés selon cette méthode seront immédiatement utilisables dans toutes les langues dans lesquelles sera traduit SPIP.

D'un point de vue technique, les éléments de texte fournis en standard avec SPIP sont stockés dans les fichiers de langue « public » :
— `/ecrire/lang/public_fr.php` contient les chaînes en français,
— `/ecrire/lang/public_en.php` en anglais
— etc.

- **Créer ses propres codes**

Il est de plus possible de créer ses propres codes, correspondant à des chaînes que l'on désire

ajouter soi-même.

Il s'agit alors de créer des fichiers de langue personnels, sur le modèle des fichiers `public...`. Pour créer ses propres fichiers, on installera, dans votre répertoire de squelette ou dans le répertoire `/ecrire/lang` :

— `local_fr.php` pour définir les chaînes en français,
— `local_en.php` en anglais,
— ...

> *historique :* Dans les versions antérieures à [SPIP 1.8], les fichiers de langue personnels se plaçaient seulement dans le répertoire `/ecrire/lang`.

Par exemple, on pourra créer les chaînes suivantes :
— `telechargement` pour afficher « Télécharger la dernière version »,
— `quoideneuf` pour afficher « Modifications récentes ».

Selon cette méthode, on insère dans les squelettes les codes `<:telechargement:>` et `<:quoideneuf:>`, ils seront ensuite affichés avec les traductions correspondantes, telles qu'elles sont définies dans les fichiers `local_...php`.

> Notons que les codes sont arbitraires : c'est vous qui les choisissez. Nous recommandons bien évidemment de choisir des codes qui vous permettent de les retenir facilement (plutôt que des numéros par exemple). Comme souvent avec les codes informatiques, il est préférable de n'utiliser que des lettres de l'alphabet latin, et sans accents...

Les *fichiers de langue* contiennent les différentes traductions des codes que vous utiliserez ; ce sont des fichiers PHP contenant chacun un tableau associant aux codes les chaînes correspondantes dans chaque langue.

Ils contiendront par exemple :

- *Version française :*
- *<?php*
- *$GLOBALS[$GLOBALS['idx_lang']] = array(*
- *'telechargement' => 'Télécharger la dernière version',*
- *'quoideneuf' => 'Modifications récentes'*
- *);*
- *?>*
- *Version catalane :*
- *<?php*
- *$GLOBALS[$GLOBALS['idx_lang']] = array(*
- *'telechargement' => 'Descarregar la darrera versió',*
- *'quoideneuf' => 'Modificacions recents'*
- *);*
- *?>*

La construction est la suivante :

— au début du fichier :

<?php

$GLOBALS[$GLOBALS['idx_lang']] = array(

— à la fin du fichier :

);

?>

— la partie qu'il faut enrichir soit-même consiste en plusieurs lignes de *définitions*, sur le modèle :

'code' => 'La chaîne à afficher',

> **N.B.** Chaque ligne de définition se termine par une virgule, sauf la dernière ligne.
>
> **N.B.2.** Le texte de la chaîne à traduire doit être convertie en codes HTML (les caractères accentués, par exemple, sont convertis en leur équivalent HTML, du type `&eacute;`).
>
> Les apostrophes à l'intérieur de la chaîne doivent être *échappées*, c'est-à-dire précédées d'un antislash. Par exemple, la chaîne « sur l'internet » doit être inscrit : `sur l\'internet`.
>
> *Note :* à terme, il est prévu d'inclure un outil permettant de gérer et créer ses propres fichiers de langue sans avoir à modifier « à la main » des fichiers PHP. Un tel outil facilitera de plus l'utilisation de caractères « spéciaux » (caractères accentués, caractères dans des alphabets non occidentaux, échappement des apostrophes...), ainsi que la collaboration de plusieurs personnes au processus de traduction de l'interface du site public.
>
> Pour l'instant, l'outil permettant de gérer la traduction des chaînes de texte n'est pas livré directement avec SPIP, et son utilisation très généraliste (nous l'utilisons pour traduire toute l'interface de SPIP, et pas seulement des fichiers de langue de type `local...php`) le rend un peu complexe par rapport à cette tâche. Ce programme, **trad-lang**, qui nous sert à traduire le logiciel SPIP, le site spip.net, etc., est lui-même disponible sous licence GNU/GPL, mais il n'est pas intégré en standard à SPIP. Vous pourrez le télécharger, pour l'utiliser pour votre site ou pour d'autres projets de logiciels. Si vous l'améliorez, ou avez des idées pour le transformer, venez en discuter sur la liste des traducteurs de SPIP, spip-trad.

2. Des squelettes séparés pour chaque langue

La seconde méthode, plus accessible aux webmestres débutants, consiste à créer des squelettes différents pour chaque langue. Un peu sur le même principe qui consiste à créer des squelettes spécifiques pour différentes rubriques pour obtenir des interfaces graphiques différentes.

Nous voulons réaliser un site en français (langue par défaut), en anglais et en espagnol. Nous réalisons trois fichiers de squelette différents :
— `article.html` pour le français (en réalité, pour toutes les langues qui n'ont pas de fichier de langue spécifique ci-après),

— `article.en.html` pour l'anglais,
— `article.es.html` pour l'espagnol.

(Note : si on publie un article en allemand, alors qu'il n'existe pas de squelette `article.de.html` sur notre site, c'est le squelette `article.html` qui sera utilisé.)

> *Important :* pour que les squelettes « par langue », définis par l'ajout d'un `.lang` à la fin du nom, soient pris en compte, il faut obligatoirement qu'il existe la version « par défaut » sur le site.
>
> Ici, si `article.html` n'existe pas (on aurait préféré nommer directement un `article.fr.html`), les fichiers anglais et espagnol ne seront pas pris en compte.

On peut combiner cela avec le nommage « par rubriques », et l'ordre suivant sera pris en compte :
— `article=8.es.html` (le squelette pour les articles en espagnol de la rubrique 8, mais pas ses sous-rubriques),
— `article=8.html` (le squelette pour les articles de la rubrique 8, mais pas ses sous-rubriques),

— `article-2.es.html` (le squelette pour les articles en espagnol de la rubrique 2 et ses sous-rubriques),
— `article-2.html` (le squelette pour les articles de la rubrique 2 et ses sous-rubriques),
— `article.es.html` (le squelette pour les articles en espagnol),
— `article.html` (le squelette pour les articles),
— `article-dist.html` (le squelette pour les articles livré avec SPIP).

> *Note :* sauf pour quelques exceptions, il faut utiliser ici les codes de langues à deux lettres normalisés par l'ISO, comme « `es` ». On en trouvera notamment la liste sur le site de la Bibliothèque du Congrès des Etats-Unis (eh oui !). Pour s'assurer d'un maximum de compatibilité, il faut utiliser en priorité les codes ISO à deux lettres (« iso 639-1 »), quand ils existent, et pour une désignation plus spécialisée d'une langue dans sa famille linguistique les codes ISO à trois lettres (« iso 639-2 T »). Par exemple, l'allemand sera désigné comme « `de` ». Mais l'ISO ne prend en compte que 182 langues sur les 5 000 à 7 000 langues parlées dans le monde ; pour les langues non encore répertoriées, on peut s'inspirer du répertoire proposé par le site ethnologue.com ; pour en savoir plus, rendez-vous sur la liste spip-trad.
>
> *Se simplifier la vie.* Une des méthodes de structuration très simple qu'autorise SPIP pour gérer un site multilingue consiste à associer directement les langues aux rubriques (et non article par article). Ainsi, dans le cas où les articles d'une même langue sont regroupés dans une même rubrique (voire par secteurs), on peut se contenter de créer les squelettes spécifiques *par rubrique*, sans utiliser alors les noms de fichiers par langues.
>
> De cette façon, si, tous les articles en espagnol sont regroupés dans la rubrique 8 (et ses sous-rubriques), on peut se contenter de nommer le fichier adapté à l'espagnol `rubrique-8.html` plutôt que `rubrique.es.html`.

On devine les problèmes qui peuvent se poser avec cette méthode :
— le fichier `article-2.html` *doit* exister si on veut que `article-2.es.html` puisse être sélectionné ;
— on ne peut pas décider, à l'inverse, que `article.es.html` doit être utilisé à la place

d’`article-2.html` si `article-2.es.html` n’existe pas : la sélection par rubriques a toujours la priorité sur la sélection par langues ;
— une correction de mise en page dans un squelette implique de faire la même correction dans les autres versions,
— on doit pouvoir insérer soi-même des éléments de texte dans les fichiers des squelettes, alors qu’on ne comprend pas forcément la langue (imaginez-vous créer ainsi les squelettes en arabe alors que vous parlez un peu le français de l’ouest et assez mal l’occitan du nord...).

Cette méthode est donc destinée à travailler rapidement sur des sites peu complexes (peu ou pas de squelettes spécifiques aux rubriques) et/ou comportant peu de langues différentes. Dès que votre projet multilingue devient un peu plus ambitieux, il est vivement conseillé de travailler avec la méthode des fichiers de langue.

3. Les blocs multilingues

Les blocs multi définis dans l’article Réaliser un site multilingue fonctionnent aussi bien dans le texte des auteurs ou mots-clés que dans les squelettes. Attention, ces blocs ne gèrent *que* du texte, et pas des boucles SPIP !

Pour gérer rapidement un site multilingue, c’est sans doute, dans un premier temps, la meilleure méthode, à la fois accessible et efficace ; ensuite, une fois votre site stabilisé, si vous avez besoin d’affiner vos squelettes (par exemple, pour les ouvrir à plus de langues ; ou pour les distribuer sous forme de contrib ; ou encore pour les « professionnaliser »), il faudra transformer vos blocs multi en fichiers de langue.

Gestion des squelettes

La gestion des pages 404

Février 2007 — maj : Décembre 2007

Depuis [Spip 1.8] il est possible de créer facilement un squelette pour la page d'erreurs 404 qui sera affichée si l'internaute demande une page qui n'existe pas.

Page d'erreur 404 , quesaco ?

Lorsqu'une personne navigue dans le Web, il peut lui arriver d'appeler une page web qui n'existe pas. Lorsqu'un serveur web reçoit la demande pour cette page, il répond alors par une page web spécifique (pour signaler que la page demandée n'existe pas), définie par le propriétaire du site (ou par défaut celle fournie avec le serveur), et qui contient généralement un message d'explication, et est accompagnée d'un code de réponse « 404 » du protocole http (qui, seul, ne serait pas très explicite), d'où le nom de « page 404 » parfois donné à ce genre de page.

Et SPIP dans tout ça ?

Il peut également arriver qu'un utilisateur demande un fichier qui existe, mais qui, faute d'information, n'apporte pas de contenu. Pour prendre un exemple avec SPIP, si quelqu'un tape dans son navigateur un adresse comme comme http://adresse-d-un-site-spip/spip.php?article520, et que l'article 520 n'existe pas encore — ou n'est pas publié en ligne —, il serait logique que SPIP retourne un message d'erreur de la même façon que la procédure précédemment décrite. On pourrait qualifier cela de « pseudo erreur 404 ».

C'est effectivement ce que fait SPIP. Il retourne une page 404 construite à partir d'un squelette. Pour la personnaliser, il suffit donc de créer son propre fichier 404.html, comme n'importe quel squelette SPIP. Ce fichier 404.html sera donc enregistré avec les autres squelettes que vous avez créés (voir à ce sujet l'article « Où placer les fichiers de squelettes ? »).

Réglage de SPIP

SPIP ne peut pas savoir tout seul quand retourner cette « pseudo erreur 404 », il faut donc lui expliciter dans chacun des squelettes qui pourrait la générer.

Par convention, SPIP va retourner la page d'erreur 404 seulement quand le contenu retourné par les boucles du squelette est complètement vide.

Le principe est assez simple, par exemple :

- le squelette de recherche, s'il ne trouve pas de résultats, affichera la page de recherche, probablement avec un message informant l'utilisateur de l'échec de la recherche. Mais il ne retournera pas une « pseudo erreur 404 ».
- par contre, un squelette d'article, si on lui demande un article qui n'existe pas ne doit pas afficher de page article, mais la « pseudo erreur 404 ».

Pour reprendre l'exemple de l'article 520 non publié ou inexistant et de l'appel d'une l'url http://adresse-d-un-site-spip/spip.php?article520, il faudrait concevoir le squelette article de la manière suivante :

```
<BOUCLE_principale(ARTICLES) {id_article}>
code html, y compris entete
</BOUCLE_principale>
```

La boucle _principale ne retourne rien si l'on demande un article inexistant ou non publié, le résultat produit sera alors une page vide, et SPIP engendrera un message d'erreur et retournera une « pseudo erreur 404 ».

Réglage du serveur Web

Cette procédure est nécessaire pour les pages d'erreurs 404 « normales », c'est à dire dues à l'absence du fichier demandé par l'utilisateur.

La méthode la plus simple est sans doute de se servir du fichier .htaccess fournit par SPIP, même si vous ne vous servez pas des URLs propres. Pour cela, renommez le fichier htaccess.txt livré en standard en .htaccess. [1]. Et c'est tout !

Attention ! il peut arriver que votre hébergeur désactive l'usage du fichier .htaccess, auquel cas il faudra le contacter pour résoudre ce problème.

Notes

[1] il est possible que vous ne puissiez pas appeler ainsi votre fichier sur votre ordinateur. Auquel cas nommer le htacces.txt, et renommer le .htaccess lorsque vous l'aurez mis en ligne.

Utiliser les modèles

Août 2006 — maj : Décembre 2007

Qu'est-ce qu'un modèle ? Notion introduite avec SPIP 1.9.1, un *modèle* est un petit squelette SPIP qui décrit un fragment de HTML facile à insérer dans un autre squelette ou — et c'est là la principale nouveauté — dans le texte d'un article.

Inspiré des modèles de Wikipédia, le système des *modèles* offre de nouvelles capacités aux webmestres et aux rédacteurs.

Les modèles sont une extension des classiques raccourcis `<img1>` et `<doc1>`. Ceux-ci correspondent désormais aux modèles `dist/modeles/img.html` et `dist/modeles/doc.html`.

Leur syntaxe a été étendue, et il est possible de spécifier, outre l'alignement (`<img1|left>`, `<img1|right>` ou `<img1|center>`), une classe plus générique, qui soit correspond à un squelette précis, s'il existe, soit à une classe CSS (`<img1|classe>` ira chercher le modele `modeles/img_classe.html`, si ce fichier existe, sinon utilisera le modèle `modeles/img.html`, mais avec un paramètre `class="classe"`).

Mais les modèles ne se limitent pas aux images et documents ; il est ainsi possible de créer de nouveaux raccourcis de la forme `<modele1>`, simplement en ajoutant un squelette dans le répertoire `modeles/` de son dossier de squelettes !

> En l'absence de tout modèle correspondant au raccourci indiqué (par exemple, `<breve1>`), le gestionnaire de modèle de SPIP regarde s'il connaît l'objet demandé (ici, `breve`), et si ce dernier a une URL.

> breve 1 Dans ce cas (vérifié ici, car les brèves sont connues du système et disposent d'une fonction d'URL), SPIP remplace le raccourci `<breve1>` par un petit encadré flottant, avec un lien vers la brève, et l'affichage de son titre, comme si l'on avait indiqué `[->breve1]`.
>
> Si l'objet n'est pas connu, SPIP laisse le raccourci intact, pour qu'il soit éventuellement traité par la suite (par un plugin ou un filtre supplémentaire), ou tout simplement ignoré.

Il est par ailleurs possible de passer des paramètres supplémentaires aux modèles (comme on le faisait pour les documents flash « embed », avec le raccourci `<emb1|autostart=true>`). La syntaxe en est généralisée et accepte même du HTML, comme dans l'exemple :

<son19|couleur=#ff0000

|legende=Le grand Count Basie

|photo=12>

qui pourrait appeler un modèle `modeles/son.html`, lequel exploiterait les paramètres pour afficher la photo numéro 12, dans un cadre de couleur #ff0000, avec une légende comportant des mots en italique.

Des usages multiples

Nous sommes loin d'avoir exploré toutes les applications possibles des modèles. En voici quelques unes auxquelles nous avons pensé. Si vous en trouvez d'autres, n'hésitez pas à la partager en proposant vos modèles sur SPIP Zone/SPIP Contrib', qui disposent désormais d'une rubrique dédiée.

- **Changer l'aspect des raccourcis de documents.** Souvent demandée, cette fonctionnalité était jusqu'ici difficile à mettre en place, puisqu'il fallait éditer du code php dans les fichiers du noyau de SPIP. Désormais, il suffit de recopier `dist/modeles/img.html` dans un sous-répertoire `modeles/` de son répertoire de squelettes, et de modifier ce fichier. Idem pour les raccourcis `<doc1>` et `<emb1>`, bien que ce dernier soit d'une complexité qui peut rebuter...
Attention : ne vous lancez pas dans la modification des modèles pour des modifications mineures du rendu de ces raccourcis — il est souvent plus facile de modifier les styles spip_documents_xx à partir des fichiers CSS de votre site.

- **Jouer un son avec un player flash.** Un modèle `modeles/son_player.html` pourrait donner un raccourci `<son12|player>`.

- **Associer un site à un article.** En ajoutant dans ses squelettes un modèle `modeles/site_box.html`, on crée immédiatement un nouveau raccourci `<site1|box>`. On écrit alors le modèle (avec des boucles classiques) de manière à lui faire afficher le nom du site, suivi de liens vers les 3 derniers articles syndiqués, dans une boîte flottant sur la droite, et voici une infobox facile à placer dans un article. Un paramètre pourrait indiquer le nombre d'articles à afficher, la présence ou non de résumés, etc.

- **Afficher une image timbrée.** Une fois qu'on sait faire un squelette qui affiche une photo sous forme de timbre-poste (voir Un site dûment timbré), il suffit de le nommer `modeles/timbre.html` pour créer le raccourci `<timbre12>`. Avec, pourquoi pas, des paramètres de taille, de couleur, de choix du tampon, etc [1].

Bien entendu on peut imaginer de la même façon des modèles affichant les images en sépia, en version réduite, etc. Gageons que tout cela sera rapidement disponible sous forme de plugins.

- **Créer un article composite.** Supposons qu'on ait besoin d'un article composé du « chapo » de l'article 1 et du texte de l'article 2. Rien de plus simple : on crée deux modèles, qui renvoient tout simplement, pour l'un, le chapo de l'article demandé, pour l'autre son texte, et dans notre article composite on indique, dans le champ chapo : `<article1|chapo>`, et dans le champ texte : `<article2|texte>`. On peut y ajouter filtres, balises et bidouilles à volonté...

- **Installer un article dans plusieurs rubriques.** En programmant des modèles `<article1|chapo>`, `<article1|texte>` etc, il devient envisageable de mettre une copie d'un article dans un autre article, sans dupliquer les données. On obtient ainsi un article « fantôme » qu'on peut installer dans une nouvelle rubrique (avec, en plus, la possibilité de le titrer différement ou de lui ajouter des éléments).

- **Faire un sondage.** Le plugin *Forms*, qui permet de créer des formulaires et de les exploiter dans des articles avec le raccourci `<form1>`, a été réécrit à partir des modèles.

- **Afficher une citation aléatoire.** Si on a mis des citations dans ses brèves, un raccourci `<citation|aleatoire>` peut en extraire une au hasard (critère `{par hasard}{0,1}` sur une boucle de brèves), et l'afficher dans un encadré flottant à côté du paragraphe courant.

- **Insérer un document dans une autre langue** que la langue de l'article. Les modèles fonctionnant comme des inclusions, le paramètre `lang=xx` y est toujours disponible. Si l'un de vos documents est bilingue (par exemple avec un « bloc multi » dans le descriptif), vous pouvez afficher le descriptif en espéranto, dans un article en japonais, en appelant `<doc1|left|lang=eo>`. Si le squelette du modèle contient des chaînes de langue, elles seront interprêtées, le cas échéant, dans la langue passée en paramètre.

- **Afficher un graphique.** On passe une table de données au modèle, qui se débrouille ensuite pour créer le graphique et l'insérer dans le flux de texte.

- **Un intertitre sous forme d'image.** Pourquoi ne pas envisager un raccourci `<imagetypo|texte=Mon intertitre>` ?

Des paramètres à foison

La syntaxe des raccourcis de modèles est de la forme `<modele12>`, `<modele|parametre1=truc|parametre2=chose>` ou `<modele12|alignement|parametre1=etc>`. Les paramètres peuvent être composés de HTML et de raccourcis SPIP (à condition, bien sûr, que les modèles appelés aient prévu de les traiter).

On notera que, pour éviter toute collision avec les balises HTML, un modèle ne peut pas être appelé par un raccourci comme `<modele>`, qui ne contient ni chiffre ni le symbole `|`.

Les paramètres peuvent s'étendre sur plusieurs lignes, ce qui permet l'écriture relativement aérée :

```
<modele 10
 |pays=Allemagne
 |population=82000000
 |superficie=357027
 |classement=63
 |hymne=<i>Das Lied der Deutschen</i>
 |url=http://fr.wikipedia.org/wiki/Allemagne
>
```

Le squelette récupère tous ces paramètres dans la balise `#ENV`, ainsi `#ENV{population}` vaut 82000000. Cette balise étant sécurisée contre toute injection de javascript, si on veut permettre du

HTML dans un paramètre, il convient d'utiliser la notation `#ENV*{hymne}` ; si l'on veut en plus appliquer la typographie de SPIP, on peut employer `[(#ENV*{hymne}|typo)]` ou `[(#ENV*{hymne}|propre)]`.

Le paramètre principal (ici, 10), est passé sous deux formes : `#ENV{id}=10`, et `#ENV{id_modele}=10`. Ce qui permet d'accéder, pour un modèle appelé par `<article3|chapo>`, au chapo de l'article 3 en faisant : `<BOUCLE_a(ARTICLES){id_article}>#CHAPO</BOUCLE_a>`

ou bien aux brèves liées au mot-clé numéro 3, par ; `<BOUCLE_b(BREVES){id_mot=#ENV{id}}>#TITRE</BOUCLE_b>`

Comme on le voit, chaque modèle devra avoir sa propre documentation, car le raccourci n'indique rien en lui-même sur l'exploitation faite des éléments passés par le raccourci.

Un emploi possible dans les squelettes

Les modèles ne sont pas limités à des raccourcis dans les articles. Il est possible de les appeler depuis un squelette, en utilisant la balise `#MODELE{modele}` ou `[(#MODELE{modele}{p1=truc,p2=chose}{p3=etc}|filtre...)]` ; cela est toutefois moins nouveau, car c'est équivalent à une inclusion (statique) de squelette (également permise par la nouvelle balise `#INCLURE`).

Les modèles par défaut

SPIP est livré avec les modèles suivants :
- `img`, `doc`, `emb`,
- `article_mots` et `article_traductions`, qui donnent respectivement la liste des mots-clés associés à un article, et de ses traductions (raccourcis : `<article1|mots>` et `<article1|traductions>`, mais ces modèles sont aussi appelés par le squelette par `dist/article.html`) ;
- `lesauteurs`, qui définit le produit de la balise `#LESAUTEURS`, mais ne peut pas être appelé comme un raccourci ;
- et une série de modèles de pagination (voir Le système de pagination).

Quelques conseils pour écrire un modèle

Il est conseillé de commencer par réfléchir à la syntaxe que l'on veut adopter : ce modèle est-il lié à un article précis ? Si oui, on partira sur un `<article1|xxx>`.

Une fois cette syntaxe établie, on crée le fichier `modeles/article_xxx.html`, et on y entre simplement la balise suivante : `#ENV`. Puis, dans un article de test, on tape notre raccourci `<article1|xxx|param=x...>` ; on voit alors (sous une forme un peu cryptique, il s'agit d'un tableau sérialisé) l'environnement tel qu'il est passé au modèle. On peut par exemple y distinguer notre identifiant d'article (sous le nom `#ENV{id}` et `#ENV{id_article}`.

Ensuite on peut commencer à entrer le squelette de notre fragment de page : `<BOUCLE_x(ARTICLES){id_article}>` ou `<BOUCLE_x(ARTICLES){id_article=#ENV{id}}>`. Et c'est parti...

Pour que notre modèle soit complet, il faut essayer les syntaxes `<article1|xxx|left>` et `[<article1|xxx>->www.spip.net]`.

Dans le premier cas, c'est le paramètre `align=left` qui est passé au modèle (et il est souhaitable que le modèle s'aligne alors du côté demandé).

Dans le second cas, le lien est passé dans un paramètre `lien=http://www.spip.net`, et la classe du lien dans `lien_classe=spip_out`. Il est recommandé de prendre en compte l'url demandée, en la transformant en lien, quelque part dans le modèle (par exemple sur le titre ou l'icone) ; dans ce cas, il *faut* ajouter dans la première balise HTML du modèle une `class="spip_lien_ok"`, qui signalera au gestionnaire de modèle que le lien a été pris en compte (faute de quoi le gestionnaire ajoutera un `<a href=...>...</a>` autour du modèle produit.

En ce qui concerne les paramètres, la balise `#ENV{x}` a été conçue de manière à éviter toute injection de HTML ou de javascript non désirée. Dans un modèle, on peut souhaiter autoriser le HTML dans les paramètres : il faut alors utiliser `#ENV*{x}` pour récupérer les données, et éventuellement les filtrer par `|propre` ou `|typo`, selon le type de données (texte pouvant comporter plusieurs paragraphes, ou simple ligne de texte).

Sur le plan de la programmation, il est recommandé de concevoir ses modèles tout en boucles, sans aucun code php ni balise dynamique. A noter toutefois : dans un modèle comportant du php, c'est le résultat du calcul qui est mis en cache, et non le script lui-même (comme c'est le cas avec les squelettes).

Notes

[1] Si l'on préfère que le raccourci s'appelle `<img12|timbre>`, on nommera le modèle `modeles/img_timbre.html`.

Les variantes de squelette

par langue, par rubrique ou par branche

Août 2006 — maj : Décembre 2007

SPIP permet de gérer des variantes de squelettes par rubrique, par branche ou par langue.

Comme mentionné au début du manuel de réference et dans la documentation sur le multilinguisme, SPIP permet de gérer des *variantes des squelettes*, par langue et pour certaines rubriques uniquement.

Des mises en page différentes

On peut souhaiter, par exemple, que tous les articles d'une rubrique aient une mise en page différente : couleur de fond et taille de texte différente, informations relatives aux mots clés mises en évidence, etc. Ou encore que le contenu d'une rubrique donnée soit présentée différement parce qu'il correspond à un type de données différents : par exemple en listant tous les articles par numéro, y compris le contenu qui serait court, plutôt que les derniers en dates suivis d'une pagination de tous les articles, avec des liens vers les pages d'articles. On pourra aussi vouloir que l'interface du site soit différente selon la langue de l'article ou de la rubrique.

Les variantes de squelettes sont une manière simple — mais du coup, pas forcément très souple — permettent à SPIP de faire cela. Pour lui indiquer d'utiliser des mises en pages différentes, il suffit de réaliser des squelettes différents auxquels on donnera des noms de fichier qui indiquent quand il faut les utiliser :

- `rubrique=22.html` : squelette spécial pour la rubrique numéro 22,
- `article-2.html` : squelette pour tous les articles contenus dans la rubrique 2 et ses sous-rubriques,
- `article.en.html` : squelette pour les articles anglais,
- `rubrique.html` : squelette par défaut, s'appliquant à toutes les rubriques n'ayant pas de squelette particulier, **mais dont la présence est obligatoire dans un répertoire pour que les variantes de squelettes soient prises en compte**.

Ordre exhaustif des variantes de squelette

Prenons l'exemple des squelettes d'article, avec des valeurs données de langue et de rubriques (mais l'explication ci-dessous reste valable pour les squelettes de rubriques ou les brèves, et bien sûr, quelles que soient les langues et les numéros de rubriques).

Rappellons avant tout que SPIP recherche *d'abord* le répertoire où il prendra le squelette, comme détaillé dans « Où placer les fichiers de squelettes ? », en cherchant s'il existe un fichier `article.html`. Il ne faut donc pas créer de variante de squelette (par exemple un fichier `article.cs.html`) sans créer *dans le même répertoire* un fichier `article.html`, au risque de voir notre variante ignorée au profit du squelette générique d'un répertoire de priorité moindre.

Si ce fichier existe, SPIP utilise les fichiers de squelettes selon leur nom en commençant par « les plus précis », avec précédence du rubricage par rapport à la langue. Voici donc l'ordre (par priorité décroissante) dans lequel sont utilisés les fichiers de squelettes selon leur nom :

- `article=8.cs.html` : si ce fichier existe, il ne s'applique qu'aux articles en langue tchèque de la rubrique numéro 8 (mais pas aux articles contenus par ses sous-rubriques).

- `article=8.html` : si ce fichier existe, il s'applique aux articles de la rubrique 8 (sauf aux articles concernés par un fichier spécifiant aussi la langue, tel que précedemment).

- `article-2.es.html` : si ce fichier existe, il s'applique aux articles en espagnol contenus dans la rubrique 2 et ses sous-rubriques. Si la rubrique 8 est une sous-rubrique de la 2, si les fichiers ci-dessus existent, ils prévaudront.

- `article-2.html` : si ce fichier existe, il s'applique aux articles contenus dans la rubrique 2 et ses sous-rubriques (sauf aux articles concernés par les fichiers indiqués ci-dessus).

- `article.vi.html` : si ce fichier existe, il s'applique aux articles en vietnamien, dans toutes les rubriques (sauf aux articles concernés par les fichiers indiqués ci-dessus).

- Enfin, le fichier `article.html` s'applique à tous les articles qui ne sont pas concernés par les fichiers indiqués ci-dessus. Et, répétons-le, il est quoi qu'il en soit nécessaire que ce fichier existe.

> *Historique :* Jusqu'à SPIP 1.7, SPIP 1.7.2 , si le fichier `article.html` n'existe pas, SPIP utilise alors le fichier `article-dist.html` qui est le fichier fourni par défaut. Voir « Qu'est-ce que les fichiers « dist » ? ».

<INCLURE> d'autres squelettes

Août 2002 — maj : 30 juin

Lorsque l'on a des éléments de texte et des boucles communs à plusieurs fichiers, on peut vouloir extraire ces éléments des pages où ils se trouvent, les installer dans un fichier séparé, et les appeler

depuis les autres squelettes. De cette façon, le code commun est regroupé dans un unique fichier, ce qui facilite notamment les modifications qui concernent plusieurs squelettes d'un seul coup.

On installe ainsi typiquement, dans de tels fichiers séparés appelés depuis de nombreux squelettes, des éléments tels que :
— les déclarations d'entête des pages HTML (appel des javascript, des feuilles de style...),
— les éléments de navigation communs à la plupart des pages (en haut de page, en colonne de gauche...),
— la bannière supérieure (logo, liens vers les crédits du site, la page de contact...),
— un pied de page...

Syntaxe d'inclusion

> *Les habitués de PHP connaissent la fonction `include`, dont le principe est similaire à ce qui est présenté ici.*

Depuis SPIP 1.4 , on peut appeler un squelette depuis un autre squelette grâce à la balise `<INCLURE>` (on peut aussi utiliser `<INCLUDE>`, qui est identique). Sa syntaxe générale est :

```
<INCLURE(fichier.php3){paramètre}...>
```

Le « fichier.php3 » est le nom du fichier que l'on veut intégrer dans sa page. Par exemple, imaginons que toutes les pages du site affichent les mêmes informations en bas de page. On regroupe alors le code SPIP et HTML de ce « pied de page » dans un fichier « `pied.html` », squelette lui-même appelé par le fichier « `pied.php3` » (toujours selon le principe des couples de fichiers destinés à appeler des squelettes). Il suffit d'ajouter la ligne suivante, à l'endroit voulu, dans chacun des squelettes voulant afficher le bas de page :

<INCLURE(pied.php3)>

Depuis SPIP 1.8.2 la distribution inclut un fichier `page.php3` qui permet d'appeler tout squelette en passant en paramètre le « fond ». Ainsi on pourra s'éviter le fichier `pied.php3` et remplacer avantageusement l'appel ci-dessus par :

<INCLURE(page.php3){fond=pied}>

Cette syntaxe est encore simplifiée depuis SPIP 1.9, puisqu'on ne précise plus désormais que le nom du squelette à inclure, sous la forme :

<INCLURE{fond=pied}>

Inclusions dépendant du contexte

Certaines inclusions peuvent dépendre du contexte. Par exemple, imaginons un squelette « `hierarchie` », qui affiche le chemin menant à une rubrique depuis la racine du site ; on appelerait cette page par une URL de la forme : « `spip.php?page=hierarchie&id_rubrique=xxx` ».

Dans les squelettes voulant afficher la hiérarchie à partir de la rubrique courante, il faut donc indiquer que le paramètre concerné est `{id_rubrique}` ; si nécessaire, on aura créé une boucle permettant de récupérer le numéro de la rubrique concernée, et on placera le code suivant à l'intérieur de cette boucle :

<INCLURE{fond=hierarchie}{id_rubrique}>

> *Note : dans ce cas, le squelette `hierarchie.html` commencera certainement par*

une boucle rubriques avec le critère `{id_rubrique}`...

On peut imaginer que, dans certains squelettes, on désire récupérer non pas la hiérarchie en fonction d'une rubrique « variable » (au gré du contexte, par exemple le paramètre passé dans l'URL), mais en fonction d'une rubrique dont on connaît à l'avance le numéro. Pour cela, on peut fixer la valeur du paramètre ainsi :

<INCLURE{fond=hierarchie}{id_rubrique=5}>

N.B. Il est possible d'indiquer plusieurs paramètres dans la balise `<INCLURE>` ; cependant ce cas est très rare en pratique. Evitez d'ajouter des paramètres inutiles, qui rendront le cache moins efficace et votre site plus lent.

N.B. Le fichier inclus étant lui-même un squelette, il disposera donc de sa propre valeur de `$delais` [1]. Cela peut s'avérer pratique pour séparer des éléments lourds du site, que l'on recalculera peu souvent, et quelques éléments dynamiques nécessitant une mise à jour fréquente (par exemple, syndication).

Dans un contexte multilingue

Si le multilinguisme de SPIP est activé, depuis SPIP 1.7, SPIP 1.7.2 , il est possible de définir la langue de l'environnement d'un squelette inclus en utilisant le paramètre `{lang}`.

- S'il n'y a pas de paramètre de langue utilisé, c'est-à-dire sous la forme `<INCLURE{fond=pied}>`, le squelette inclus est appelé en utilisant la langue par défaut du site ;

- `<INCLURE{fond=pied}{lang=es}>` appelle le squelette en espagnol. Bien sûr, vous pouvez remplacer « es » par le code ISO de la langue souhaitée : *en* pour l'anglais, *fr* pour le français, *vi* pour le vietnamien, etc. (voir : « Internationaliser les squelettes ») ;

- `<INCLURE{fond=pied}{lang}>` appelle le squelette dans la langue courante du contexte d'inclusion.

Il convient de noter que cela rend possible l'utilisation des codes de fichiers de langue dans les squelettes inclus (voir : « Internationaliser les squelettes »).

Les squelettes inclus supportent les mêmes mécanismes de sélection par langue que les squelettes de « premier niveau ». En d'autres termes les squelettes inclus (ici `pied.html`) peuvent être déterminés par rapport à une certaine langue (`pied.es.html`, par exemple) de la même manière que tout autre squelette. Encore une fois, voir « Internationaliser les squelettes » pour plus de détails.

`#INCLURE` en statique

La version SPIP 1.9.1 a introduit une autre méthode d'inclusion : `#INCLURE`.

La syntaxe `<INCLURE{fond=..}>` provoque l'inclusion des pages à chaque visite d'un internaute, que celle-ci concerne une page déjà en cache ou non.

Avec la nouvelle balise `[(#INCLURE{fond=...})]`, l'inclusion est réalisée lors du calcul du squelette, et son résultat est stocké dans le cache de la page appelante. Avec ce système, on ne peut plus gérer une durée de vie (`$delais` ou `#CACHE{}`) réduite pour un squelette inclus ; en revanche il devient possible d'appliquer des filtres sur le squelette inclus : `[(#INCLURE{fond=lettre}|version_texte)]`.

`<INCLURE{fond=..}>` et `[(#INCLURE{fond=..})]` ont donc des fonctionnements

différents :
— avec `<INCLURE{fond=..}>`, chaque squelette inclus a un cache indépendant ;
— avec `[(#INCLURE{fond=..})]`, la page principale appelante contient, en cache, l'intégralité du code généré, et les fichiers inclus n'ont pas de cache séparé.

Notes

[1] Rappelons que la variable `$delais` définit la périodicité de mise à jour du cache. Voir la section « Le fichier .php3 » dans « Principe général - version archivée ».

Bases de données

Les bases de données en SPIP

12 décembre — maj : 7 décembre

SPIP peut être vu comme un outil de mise en page dans un format MIME (HTML, RSS, ICS ...) d'extraits de bases de données. Une des nouveautés de la version 2.0 est de pouvoir rassembler dans une page des données en provenance de plusieurs bases SQL, accessibles par des serveurs éventuellement différents et distants les uns des autres. SPIP se charge de gérer les différentes connexions et déclarations de tables sans imposer de programmation à ses utilisateurs. Le présent article expose toutes les manipulations de bases de données offertes par cette dernière version.

Installation minimale

Voici d'abord tous les scénarios possibles d'installation de SPIP avec une seule base.

Lors de l'installation de SPIP, celui-ci teste la configuration de PHP et propose, lorsque c'est possible, un choix parmi plusieurs types de serveurs SQL (actuellement MySQL, PostgreSQL ou SQLite), qui offrent tous les mêmes fonctionnalités. À ce stade, il faut également fournir l'adresse Internet du serveur SQL, un identifiant de connexion à ce serveur et son mot de passe associé. Ces informations sont en général à saisir dans le formulaire d'installation. Toutefois, si vous voulez mettre en œuvre la Mutualisation du noyau SPIP, vous pouvez indiquer une ou plusieurs de ces valeurs dans le fichier de configuration `mes_options.php`. SPIP ne les demandera alors pas, ce qui dispense de les saisir et ne vous oblige pas à les communiquer à ceux qui utiliseront votre installation mutualisée. Voici les noms des constantes PHP à définir pour cela :

`_INSTALL_SERVER_DB` *type du serveur SQL* (Mysql ou PG ; casse indifférente)
`_INSTALL_HOST_DB` *nom Internet du serveur SQL (par exemple : localhost)*
`_INSTALL_USER_DB` *un nom d'utilisateur du serveur SQL*
`_INSTALL_PASS_DB` *son mot de passe*

SPIP se connecte alors au serveur choisi, avec les identifiants de connexion. En cas de réussite, il teste si l'utilisateur a le droit de créer des bases, ou seulement le droit de voir celles présentes et d'en choisir une, voire n'a le droit que d'utiliser une base homonyme de son nom d'utilisateur. Vous pouvez également fixer d'avance dans le fichier `mes_options.php` le nom de la base de données à utiliser, en définissant la constante PHP :

`_INSTALL_NAME_DB` *nom de la base*

SPIP poursuit alors l'installation en créant ses tables SQL (ou, en cas de réinstallation, en vérifiant que celles présentes sont utilisables). Ces tables commencent toutes par un *préfixe de table* (qui vaut par défaut `spip_`) et peut être indiqué dans `mes_options.php` de deux manières :

la variable globale `$table_prefix`
la constante `_INSTALL_TABLE_PREFIX`

Ce préfixe de table est ce qui permet d'écrire des boucles abrégées dans les squelettes, comme

```
<BOUCLE1(ARTICLES)....
```

alors que le nom SQL de la table est en fait `spip_articles`.

SPIP demande ensuite de créer le premier utilisateur du site en fournissant un nom et un mot de passe (sans rapport avec ceux du serveur SQL) ou en proposant de déléguer le service d'authentification à un serveur LDAP. Après cette déclaration, la procédure minimale d'installation est terminée, et la page suivante est la page habituelle d'entrée dans l'espace de rédaction, qui demande les identifiants venant d'être saisis. Ceux-ci ont été sauvés dans un fichier qui par défaut est `config/connect.php`.

Compléments d'installation

Voici à présent comment indiquer à SPIP plusieurs bases à exploiter, sous le même serveur ou sous des serveurs différents, et même sur des machines différentes.

Une fois le site installé, choisir le sous-menu *maintenance du site* dans le menu *configuration* en haut de l'espace privé. Ce menu fournit une page à trois onglets ; cliquer sur celui de droite, nommé *déclarer une autre base*. On accède alors à un formulaire quasiment identique au formulaire d'installation : il propose d'indiquer un type de serveur SQL, son adresse Internet, un nom d'utilisateur et son mot de passe. En cas de connexion réussie, SPIP demandera de fournir le nom d'une base, en listant celles présentes s'il en a la possibilité. Si elle existe bien, il créera un fichier de connexion supplémentaire, qui par défaut se trouve dans le répertoire `config` et porte le nom de la base de données indiquée.

Cela fait, SPIP revient à nouveau à ce formulaire, et liste les fichiers de connexion présents, autrement dit les bases accessibles. SPIP propose donc de déclarer encore une base supplémentaire : il n'y a pas de limitation au nombre de bases déclarables.

L'intérêt de ces déclarations est de pouvoir appliquer la notation des squelettes de mise en pages à ces bases supplémentaires. S'il existe un fichier de connexion nommé *B*`.php`, permettant d'interroger une base possédant une table nommée *T*, alors le squelette

```
<BOUCLE1(B:T)></BOUCLE1>#TOTAL_BOUCLE<//B1>
```

donnera le nombre de lignes de cette table. De plus, SPIP demande au serveur SQL de lui *décrire* cette table, ce qui lui permet de connaître ses champs et, éventuellement, sa *clé primaire*. Si la boucle *T* a une clé primaire nommée `id` et un champ nommé `nom`, alors le squelette :

```
<BOUCLE1(B:T){id}>#NOM</BOUCLE1>
```

donnera le champ `nom` de l'unique ligne ayant l'`id` donné par le contexte.

Enfin, il est possible d'appliquer un squelette en demandant que toutes ses boucles utilisent une autre base que celle par défaut, en ajoutant dans l'URL de la page le paramètre `connect` indiquant le nom de la base. Par exemple `http://monsite?connect=autre_site` appliquera le squelette `sommaire` standard du site `monsite` à la base indiquée par le fichier de connexion `config/autre_site.php` dans le répertoire d'installation de `mon_site`.

Il est recommandé de respecter la casse du nom du fichier de connexion, lorsqu'on l'utilise dans une boucle ou comme paramètre de `connect` : aucune conversion majuscules / minuscules n'est effectuée par SPIP (mais certains systèmes de fichiers le font).

Remarque : L'accès à des bases supplémentaires est exclusivement en lecture ; en particulier, si une base est par exemple un forum (administré par SPIP ou non) il reste impossible de poster dans ce forum autrement qu'à partir de son site d'origine. La levée de cette restriction est à l'étude.

Installations croisées

Les extraits de squelette ci-dessus reposent donc sur l'existence d'un fichier de connexion nommé

B`.php`. Un point important est que le fichier de connexion principal et ceux des bases supplémentaires ont le même format, ce qui permet en particulier à un site SPIP d'utiliser directement le fichier de connexion d'un autre site SPIP pour exploiter ses tables. Une manière de procéder est de copier le fichier `connect.php` du site B sous le nom B`.php` dans le répertoire `config` du site A.

Une manière plus astucieuse, dans le cas de sites partageant une même installation de SPIP, est d'avoir un seul répertoire `config` pour tous les sites, et d'y nommer le fichier de connexion du site A non pas `connect.php` mais A`.php`, de même pour le site B etc. De la sorte, dès qu'un site est installé sous SPIP, il est connu comme base supplémentaire de tous les autres sites partageant cette installation de SPIP, et il verra tous les sites utilisant le même répertoire `config` que lui comme autant de bases supplémentaires. Pour obtenir cet optimum dans les déclarations implicites, il faut définir soigneusement les constantes `_DIR_CONFIG` et `_FILE_CONNECT_INS`. On trouvera plus bas un exemple de mutuallisation offrant cette fonctionnalité et quelques autres.

Pour un site installé sur un serveur SQL éloigné, la copie de son fichier de connexion sur le site local peut suffire en théorie. Toutefois beaucoup d'hébergeurs refusent que leurs serveurs SQL soient interrogés par des machines en dehors de leur réseau local. Il faut aussi penser que bien souvent le « nom » indiqué pour le serveur SQL à l'installation est `localhost`, qui est une adresse relative au serveur HTTP, alors qu'ici il y a nécessité d'une adresse absolue. En bref, SPIP peut opérer dans une telle architecture, mais un minimum de connaissances, voire d'autorisation d'intervention, sur la topographie des sous-réseaux en jeu est pratiquement indispensable.

Les bases supplémentaires sous SPIP

Lorsqu'un fichier de connexion est celui d'un autre site sous SPIP (que ce soit une copie ou l'original accessible par installations croisées), il contient l'indication que ce site est sous SPIP (la globale `spip_connect_version` y est affectée). Dans ce cas, les squelettes du site principal vont s'appliquer avec un comportement spécial :

- les boucles avec nom abrégé seront interprétées comme abrégées aussi dans la base supplémentaire ; autrement dit, `<BOUCLE1(B:ARTICLES)...` (ou `<BOUCLE1(ARTICLES)...` avec une URL comportant `connect=B.`) référencera la table `spip_articles` dans la base B, plus précisément la table *prefixe*`articles` où *prefixe* est celui utilisé par le site B (ce préfixe est indiqué dans le fichier de connexion) ;

- à l'intérieur d'une boucle, les balises `#URL_` (voir Utiliser des URLs personnalisées) utiliseront le `$type_urls` du site principal, et non du site distant, et l'URL produite référencera le site principal mais en rajoutant la variable d'URL `connect=`*site_distant* ; cette stratégie permet donc de naviguer dans la base du site distant sans cesser d'utiliser les squelettes du site principal, autrement dit de tester l'apparence que donne ce jeu de squelettes et son `$type_urls` sur le site distant sans modifier l'installation de celui-ci ou travailler sur une copie.

- Les raccourcis SPIP des balises #URL qui peuvent figurer dans les champs de la base de données distante sont eux aussi interprétés de cette manière : `[->art3681]` référencera bien l'article 3681 de la base distante, mais à travers une URL qui fournira la mise en page par les squelettes du site principal.

Ce mode de fonctionnement permet donc de présenter d'autres bases SPIP sans même avoir à écrire des squelettes de mises en page, puisque les noms des tables dans ceux-ci sont les mêmes que ceux du site distant. En revanche, une base qui n'est pas sous SPIP a besoin de squelettes de mises en page nommant explicitement les tables de cette autre base et ses champs. Pour répondre à ce besoin, les administrateurs du site bénéficient d'un traitement spécial : lorsqu'ils donnent à leur navigateur

l'URL de leur site suivi de `?page=table:`*table* où *table* est le nom d'une table de la base, SPIP va automatiquement construire un squelette spécifique à cette table, permettant d'en examiner le contenu avec une grande ergonomie. Le squelette produit est affichable à travers le lien *squelette* en bas de page. Il est possible alors de le copier à la souris (cliquer en haut de colonne pour cacher la numérotation des lignes) et de l'améliorer en le travaillant sous un éditeur approprié.

À noter qu'avec cette production automatique, SPIP pourrait, à la limite, fonctionner sans aucun squelette prédéfini.

Sauvegardes et Fusions

Avec le sous-menu **maintenance du site**, SPIP donne accès depuis toujours à deux outils de sauvegarde et restauration de la base de données locale (voir Sauvegarder vos données). Malgré quelques cas exceptionnellement favorables, on ne peut restaurer une base que dans une installation de SPIP de même numéro. D'autre part, jusqu'à la version 1.8, une sauvegarde était totale, et la restauration également. Après quelques essais fondés sur le statut d'administrateur restreint, qui se sont révélés finalement malcommodes, SPIP 1.9.3 offre une nouvelle fonctionnalité plus harmonieuse de sauvegarde partielle et de restauration par fusion.

Le formulaire de sauvegarde propose de limiter la sauvegarde à une rubrique donnée. Le fichier produit contiendra les articles, sites, brèves et sous rubriques contenu dans la rubrique indiquée, ainsi que les mots-clés et groupes de mots-clés utilisés par toutes ces données. Le nom de la sauvegarde sera par défaut le nom de la rubrique, mais cela peut être changé. La création des fichiers de sauvegarde peut être si longue que le serveur HTTP peut vous déconnecter pour cause d'inactivité apparente. Recharger simplement la page : SPIP retrouvera à quel endroit il en était et poursuivra son travail. Le grand nombre de fichiers que vous pourrez temporairement apercevoir dans un sous-répertoire du répertoire `tmp` est normal, car lié à l'anticipation de cette situation de reprise après déconnexion.

Symétriquement, le formulaire de restauration propose à présent deux fonctionnalités. La première est l'habituel remplacement de la base courante par la base sauvegardée. La deuxième consiste à considérer le fichier de sauvegarde comme une base incomplète venant s'ajouter à la base déjà installée. Comme il peut y avoir conflit entre la base installée et la base incomplète en ce qui concerne la numérotation de leurs tables (d'articles, de brèves etc), SPIP commence par renuméroter les éléments de la base incomplète en leur affectant des numéros immédiatement supérieurs aux maximum de la base installée. Dans une deuxième passe, SPIP importe effectivement les éléments renumérotés, mais lors de la première passe, il a comparé les éléments à importer avec ceux de la base installée et a ignoré les doublons définis par les règles suivantes :

- s'il existe dans la base installée un groupe de mots-clés de même nom qu'un groupe de mots-clés à importer, ce groupe est ignoré ;

- si un mot-clé dont le groupe de mot-clés n'a pas été importé possède un homonyme dans un groupe de mot-clés portant le même nom que son groupe d'origine, ce mot-clé est ignoré ;

- s'il existe dans la base installée une rubrique de secteur portant même nom qu'une rubrique de secteur à importer, cette rubrique est ignorée ;

- si une sous-rubrique dont la rubrique parente n'a pas été importée possède un homonyme de même parenté, cette sous-rubrique est ignorée ; il en est de même pour les articles, les brèves et les sites référencés ;

- s'il existe dans la base installée un document portant même nom qu'un document à importer et qu'ils ont même taille, ce document est ignoré.

Dit de manière plus synthétique, l'opération de fusion est l'union de deux bases, le contenu de la base installée ayant priorité sur la base importée en ce qui concerne les doublons (par exemple, pour deux secteurs de même nom, les champs *texte* et *descriptif* de celui déjà installé seront conservés et les autres seront ignorés).

En ce qui concerne les documents joints, ceux-ci seront importés en tant que documents distants, ce qui dispense de recenser et copier la partie du répertoire IMG/ concernée par la sauvegarde partielle. On peut ensuite utilliser, sur chaque document, le bouton de copie locale pour s'affranchir du site d'origine. Si celui-ci a déjà disparu ou possède des restrictions d'accès non contournables par le site récepteur, il faudra installer le répertoire IMG/ originel à une URL accessible, et la donner dans la dernière case de saisie du formulaire d'importation.

On notera la possibilité de rabattre les statuts de valeur *publié* à la valeur *proposé*, ce qui permet de garder une politique de publication au cas par cas, malgré cette importation massive.

Un exemple complet

L'installation mutualisée de SPIP ci-dessous permet d'avoir un seul répertoire de configuration (`_DIR_CONNECT`) et un seul de sauvegardes (`_DIR_DUMP`). Chaque site peut ainsi donner des aperçus de la base de chacun des autres, et utiliser leurs sauvegardes pour les fusionner avec sa propre base. N'est utilisé ausi qu'un seul répertoire pour le cache de l'aide en ligne (`_DIR_AIDE`) car SPIP l'importe au coup par coup à partir du serveur spipnet, et un seul répertoire pour les *Document Type Definitions* que Le validateur XML intégré va chercher sur le site du W3C et autres (`_DIR_XML`).

Les fichiers de droits et de journalisation sont également rassemblés dans seulement deux répertoires (`_DIR_CHMOD` et `_DIR_LOG`).

Seront donc créés à la racine config/connect, config/chmod, tmp/log, tmp/dump, tmp/xml et tmp/aide. Le premier contiendra A.php pour la connexion à la base A, B.php etc.

```
if ( preg_match(',/([a-zA-Z0-9_-]*)[/?],',$_SERVER['REQUEST_URI'],
$r)) {
        if (is_dir($e = _DIR_RACINE . 'Ajouts/' . $r[1]. '/')) {
                $cookie_prefix = $table_prefix = $r[1];

                define('_SPIP_PATH',
                _DIR_RACINE. 'Ajouts/' . $table_prefix  .
'/dist/:' .
                _DIR_RACINE .'Ajouts/' . $table_prefix  . '/:' .
                _DIR_RACINE .'dist/:' .
                _DIR_RACINE .'dist/javascript/:' .
                _DIR_RESTREINT);

                $pi = $e . _NOM_PERMANENTS_INACCESSIBLES;
                $pa = $e . _NOM_PERMANENTS_ACCESSIBLES;
                $ti = $e . _NOM_TEMPORAIRES_INACCESSIBLES;
                $ta = $e . _NOM_TEMPORAIRES_ACCESSIBLES;

                $pig = _DIR_RACINE .
_NOM_PERMANENTS_INACCESSIBLES;
                $tig = _DIR_RACINE .
_NOM_TEMPORAIRES_INACCESSIBLES;
```

```
                define('_DIR_DUMP', $tig . 'dump/');
                define('_DIR_AIDE', $tig . 'aide/');
                define('_DIR_CACHE_XML', $tig . "xml/");
                define('_DIR_LOG', $tig . 'log/');
                define('_DIR_CONNECT', $pig . 'connect/');
                define('_DIR_CHMOD', $pig . 'chmod/');
                define('_FILE_CONNECT_INS', $table_prefix);
                define('_FILE_CHMOD_INS', $table_prefix);
                define('_FILE_LOG_SUFFIX',
                         '_' . $table_prefix . '.log');

                $GLOBALS['test_dirs'] =
                  array($pa, $ta, $ti, $pig, $tig,
                _DIR_DUMP, _DIR_LOG, _DIR_CONNECT, _DIR_CHMOD);

                spip_initialisation($pi, $pa, $ti, $ta);
        }
)
```

P.-S.

Point d'histoire. Dans ses premières versions, SPIP ne donnait accès qu'à certains champs d'une unique base de données (celle qu'il créait à sa première installation). Tout juste était-il possible de simuler plusieurs bases SPIP dans une seule (mais qui s'ignoraient mutuellement) en préfixant le nom des tables de chaque base simulée avec un préfixe spécifique.

La version 1.8 et le nouveau compilateur de squelettes, ont ouvert l'accès à tous les champs de toutes les bases accessibles par le serveur HTTP. Mais cette possibilité exigeait d'écrire des fichiers PHP par imitation des fichiers standards, peu intuitifs. De plus, SPIP ne distinguait pas les notions de *type de serveurs* et de *base de données* : pour adresser plusieurs bases d'un même serveur, il fallait dupliquer ces fichiers et les adpater. Cette architecture très insatisfaisante explique pourquoi ces fonctionnalités n'ont jamais été documentées, bien que plusieurs extensions de SPIP s'en soient servi.

L'interface de SPIP avec SQL

12 décembre — maj : 7 décembre

Rappelons d'abord que le *Structured Query Langage* ne s'est standardisé que très progressivement, chaque implémentation comblant les lacunes de la spécification avec ses propres solutions. Les premières versions de SPIP ne connaissaient qu'une seule implémentation de SQL ce qui lui permettait d'en centraliser l'utilisation à travers une seule fonction (spip_query) dont l'unique argument était la requête. Le portage de SPIP sur différentes implémentations de SQL a imposé de renoncer à ce modèle ; cette fonction reste disponible par souci de compatibilité, mais son usage doit être désormais évité. On lui préférera le jeu de fonctions suivant, qui a de plus l'avantage de neutraliser la plupart des techniques d'attaque par injection de code.

Le premier portage de SPIP sur un autre serveur que MySQL3 a été réalisé pour le serveur PostGreSQL version 8.2. Il a été immédiatement suivi d'un double portage en SQLite 2 et 3. Ces portages se fondent sur le remplacement de la fonction spip_query par autant de fonctions que d'*instructions* SQL (select, update etc) ayant chacune autant d'arguments que l'instruction admet de clauses (where, limit etc), plus un argument optionnel précisant la base SQL. Les *opérandes* SQL

(en particulier les dates et les nombres en hexadécimal) restent écrits en syntaxe MySQL, les fonctions d'interface se chargeant de les réécrire au besoin. Quelques fonctions supplémentaires sont proposées : des indispensables (l'accès aux lignes successives d'un résultat de **select**) et des abréviations (décompte, accès à une ligne qu'on sait unique etc). Des informations générales sont également fournies : alphabets utilisés, présence d'un LDAP etc.

Le double portage en SQLite n'a pas nécessité de réviser le jeu de fonctions défini d'abord pour le seul portage PostGres, ce qui tend à prouver la perennité de l'interface ainsi définie. Toutefois, de tels outils sont tôt ou tard condamnés à évoluer, aussi cette nouvelle l'interface proposée avec SPIP 2.0 intègre dès à présent un gestionnaire de ses propres versions, lui permettant d'utiliser une connexion SQL avec plusieurs versions de l'interface *simultanément*. Ainsi, les extensions de SPIP respectant cette nouvelle interface sont-elles assurées de fonctionner dans les versions ultérieures de SPIP.

Signalons enfin que le choix de développer cette architecture a été imposé par l'inexistence d'une bibliothèque de niveau d'abstraction équivalent disponible aussi librement que SPIP : les extensions de PHP couramment offertes se contentent d'uniformiser les appels aux fonctions de base de celui-ci, ce qui n'est que la partie émergée de l'iceberg auquel est confronté le développeur d'applications sur bases de données hétérogènes.

Cet article comporte trois parties. Les deux premières sont destinées aux développeurs désireux d'écrire des extensions de SPIP ; elles présentent l'architecture générale, puis les fonctions disponibles. La troisième détaille l'implémentation et est destinée aux contributeurs de SPIP, désireux de le porter sur d'autres implémentation de SQL ou améliorer les portages existants. Dans tout l'article, on parlera de *serveur SQL* pour désigner une implémentation de SQL utilisée par SPIP, bien qu'à proprement parler certaines implémentations ne sont pas des serveurs.

Architecture générale

Dans un premier temps, on peut considérer que l'interface de SPIP aux implémentations de SQL se réduit à l'unique fonction suivante, définie dans le fichier `ecrire/base/abstract_sql.php` :

```
sql_serveur($ins_sql, $serveur='', $continue=false)
```

Cette fonction commence, si elle ne l'a pas déjà fait auparavant, par se connecter au serveur spécifié en deuxième argument. Cet argument est souvent omis, ce qui désigne alors l'implémentation SQL choisie à l'installation, et mémorisé par SPIP dans un *fichier de connexion* (voir Les bases de données en SPIP). Sinon, l'argument doit indiquer explicitement le nom du fichier de connexion à utiliser, l'extension `.php` étant omise. Le résultat retourné est une autre fonction, réalisant le type d'action demandée par la chaîne passée en premier argument (par exemple `select` ou `fetch`) sur la base SQL indiquée par le fichier de connexion. Le troisième argument indique ce qu'il faut faire lorsqu'un telle fonction ne peut être retournée. S'il est absent ou égal à `false`, une erreur fatale sera déclenchée. Autrement, deux autres situations sont distinguées. Si la connexion indiquée est inconnue ou inopérante, la valeur `false` sera retournée. Autrement, sera retournée une struture de données décrivant la connexion (elle sera décrite dans la dernière partie), ce qui permet d'une part de vérifier qu'une fonction existe sans risquer l'erreur fatale, et d'autre part d'obtenir plusieurs informations avant utilisation.

Cette vision minimaliste de l'interface permet de tout faire, mais avec une syntaxe assez opaque. Si par exemple `$db` est le nom d'une base dans le serveur principal, on le sélectionnera pas

```
$f = sql_serveur('selectdb');
```

```
$f($db);
```

Pour clarifier l'écriture, il existe un jeu de fonctions enchaînant les deux instructions ci-dessus pour les cas les plus fréquents. Par exemple, il existe

```
sql_selectdb($nom, $serveur='')
```

Ce qui permet de réécrire plus simplement l'opération précédente :

```
sql_selectdb($db)
```

De manière générale, l'interface de SPIP aux serveurs SQL est un jeu de fonctions dont le nom est `sql_`*f* et dont le dernier argument, optionnel, est le nom du serveur. Elles appellent `sql_serveur` pour obtenir une fonction *f* qu'elles appliquent sur leurs arguments, y compris le nom du serveur. Toutes les fonctions dont le nom est ainsi construit sont réservées à l'interface de SPIP aux serveurs SQL.

Dans une vision *orientée objet*, ce jeu de fonctions représente les *méthodes* de l'objet `sql`, mais on écrit `sql_`*f* à la place de `sql->`*f*, et il n'est pas nécessaire d'*instancier une classe*. La présence du nom du serveur dans tous les appels permet de simuler les opérations impliquant l'objet *moi-même*.

Ce jeu de fonctions dispense donc la plupart du temps d'utiliser `sql_serveur`. Signalons également la fonction (présente dans SPIP de longue date) :

```
spip_connect($serveur='')
```

qui simplement ouvre la connexion au serveur, et est donc équivalente à

```
sql_serveur('', $serveur, true)
```

et qui retourne *false* si le serveur est indisponible, et sinon la structure de données décrivant les possibilités du serveur.

Fonctions disponibles

Les fonctions de l'interface SQL peuvent se classer en plusieurs groupes, pour lesquels on donnera à chaque fois un tableau présentant leurs arguments. Pour les exemples on se reportera au code de SPIP.

Un premier groupe de fonctions concernent la lecture des tables SQL. Les fonctions incontournables sont :

- `sql_select` dont les arguments sont les clauses SQL habituelles de cette instruction, et dont le résultat est une *ressource Select* ;

- `sql_fetch`, utilisée presque toujours dans une boucle, qui permet de récupérer les lignes successives d'une ressource Select ; son résultat est un tableau indexé par le nom des champs ;

- `sql_free` qui signale au serveur de libérer une ressource.

D'autres fonctions offrent des compositions fréquentes de ces opérations :

- `sql_fetsel` de mêmes arguments que `sql_select`, qui applique celle-ci sur ses arguments puis `sql_fetch` sur la ressource retournée ; pratique pour les requêtes dont le résultat ne comporte qu'une ligne ;

- `sql_getfetsel` de mêmes arguments que `sql_select`, qui applique celle-ci sur ses arguments puis `sql_fetch` sur la ressource retournée, et enfin extrait du tableau retourné le champ dont le nom est donné comme premier argument (la liste des champs n'en comporte donc qu'un) ; pratique pour les requêtes dont le résultat ne comporte qu'une ligne d'un seul champ ;

- `sql_allfetsel` de mêmes arguments que `sql_select`, qui applique celle-ci sur ses arguments puis `sql_fetch` sur la ressource retournée tant que celui-ci retourne un tableau non vide ; `sql_allfetsel` retourne pour finir le tableau de tous les tableaux retournés par `sql_fetch` (attention à ne pas saturer la mémoire avec cette fonction) ;

- `sql_countsel` de mêmes arguments que `sql_select` moins le premier, qui applique celle-ci sur `COUNT(*)` et ses autres arguments et retourne le nombre calculé ; pratique pour connaître le nombre de lignes que retournerait la requête Select ainsi décrite.

- `sql_count` qui retourne le nombre de lignes d'une ressource Select, comme si on avait ajouté `COUNT(*)` dans la requête ;

- `sql_get_select` utilise les mêmes arguments que `sql_select` , mais retourne le code SQL de la requête, sans l'executer. Cette fonctoin peut-être utile pour les besoins de certains plugins ou pour créer facilement des vues SQL.

Les quatre premières appellent pour finir `sql_free`, et sont donc à préférer autant que possible au trio `select-fetch-free` dont on oublie facilement le dernier membre.

Le tableau ci-dessous précise le contenu et l'ordre des arguments attendus par ces fonctions.

Fonction	Arguments
sql_select	1. liste des champs : chaîne ou tableau 2. liste des tables : chaîne ou tableau 3. clause Where : chaîne ou tableau 4. clause Groupby : chaîne avec virgule séparatice ou tableau 5. clause Orderby : chaîne avec virgule séparatice ou tableau 6. clause Limit : un ou deux entiers séparés par des virgules 7. clause Having : chaîne ou tableau 8. serveur
sql_fetch	1. ressource 2. serveur
sql_free	1. ressource 2. serveur
sql_count	1. ressource 2. serveur
sql_countsel	1. liste des tables : chaîne ou tableau 2. clause Where : chaîne ou tableau 3. clause Groupby : chaîne ou tableau 4. clause Orderby : chaîne avec virgule séparatice ou tableau 5. clause Limit : un ou deux entiers séparés par des virgules 6. clause Having : chaîne ou tableau 7. serveur
sql_fetsel	1. liste des champs : chaîne ou tableau 2. liste des tables : chaîne ou tableau 3. clause Where : chaîne ou tableau

	4. clause Groupby : chaîne avec virgule séparatice ou tableau 5. clause Orderby : chaîne avec virgule séparatice ou tableau 6. clause Limit : un ou deux entiers séparés par des virgules 7. clause Having : chaîne ou tableau 8. serveur
sql_allfetsel	1. liste des champs : chaîne ou tableau 2. liste des tables : chaîne ou tableau 3. clause Where : chaîne ou tableau 4. clause Groupby : chaîne avec virgule séparatice ou tableau 5. clause Orderby : chaîne avec virgule séparatice ou tableau 6. clause Limit : un ou deux entiers séparés par des virgules 7. clause Having : chaîne ou tableau 8. serveur
sql_getfetsel	1. nom d'un champ : chaîne 2. liste des tables : chaîne ou tableau 3. clause Where : chaîne ou tableau 4. clause Groupby : chaîne ou tableau 5. clause Orderby : chaîne avec virgule séparatice ou tableau 6. clause Limit : un ou deux entiers séparés par des virgules 7. clause Having : chaîne ou tableau 8. serveur
sql_get_select	1. liste des champs : chaîne ou tableau 2. liste des tables : chaîne ou tableau 3. clause Where : chaîne ou tableau 4. clause Groupby : chaîne avec virgule séparatice ou tableau 5. clause Orderby : chaîne avec virgule séparatice ou tableau 6. clause Limit : un ou deux entiers séparés par des virgules 7. clause Having : chaîne ou tableau 8. serveur

Dans les fonctions ci-dessus, si la clause Select est fournie sous forme de tableau, ses éléments seront concaténés, séparés par des virgules. En cas de tableau pour les clauses Where et Having, les éléments doivent être des chaînes (des sous-tableaux en notation préfixée sont également pris en charge, mais réservés au compilateur). Ces chaînes seront réunies en une grande conjonction (i.e, elles seront concaténées avec AND comme séparateur).

La clause From est une chaîne (le cas du tableau est réservé au compilateur de SPIP). Attention : s'il est nécessaire de référencer les tables dans les autres clauses, il faut en définir des alias dans ce paramètre et les utiliser systématiquement. Ainsi, on écrira :

```
sql_countsel('spip_articles AS a, spip_rubriques AS r',
"a.id_secteur=r.id_rubrique AND r.titre='monsecteur')
```

ou

```
sql_countsel('spip_articles AS a JOIN spip_rubriques AS r ON
a.id_secteur=r.id_rubrique", "r.titre='monsecteur'")
```

alors que l'écriture suivante ne sera pas comprise :

```
sql_countsel('spip_articles, spip_rubriques',
"spip_articles.id_rubrique=spip_rubriques.id_secteur AND
spip_rubriques.titre='monsecteur'")
```

Un deuxième groupe de fonctions est constitué par celles modifiant le contenu des tables. Ces fonctions sont délicates à définir car la syntaxe des valeurs à introduire dans les tables change d'un serveur SQL à un autre (notamment les dates). Pour cette raison, ces fonctions doivent disposer de la description de la table à modifier, afin de connaître le type des valeurs attendues par le serveur SQL. SPIP retrouve automatiquement ces informations (données au moment de la création de la table) mais il est possible de fournir une description arbitraire (avant-dernier argument de ces fonctions, optionnel et d'ailleurs rarement utile).

SPIP fournit donc une fonction d'insertion, `sql_insertq`, et une fonction de mise à jour, `sql_updateq`, qui prennent un tableau *champ=>valeur* et s'occupent de *citer* les valeurs en fonction du type (avec la fonction `sql_quote` spécifiée ci-dessous). Est également disponible `sql_insertq_multi` permettant de faire des insertions de plusieurs entrées en prenant un tableau de tableau *champ=>valeur*. Pour les mises à jour où les nouvelles valeurs dépendent des anciennes (comme dans `cpt=cpt+1`), utiliser `sql_update` où les valeurs seront prises littérallement, mais il faudra interdire soigneusement les possibilités d'attaque par injection de code. Il existe également `sql_replace`, fonction effectuant une mise à jour sur une ligne correspondant à une clé primaire, ou insérant les valeurs si cette ligne n'existe pas ainsi qu'une fonction `sql_replace_multi` pour des mises à jour ou insertions multiples. Enfin, `sql_delete` efface d'une table les lignes répondant à une clause Where.

Fonction	Arguments
sql_updateq	1. table 2. tableau *champ=>valeur à citer* 3. clause Where 4. description 5. serveur
sql_update	1. table 2. tableau *champ=>valeur* 3. clause Where 4. description 5. serveur
sql_insertq	1. table 2. tableau *champ=>valeur à citer* 3. description 4. serveur
sql_insertq_multi	1. table 2. tableau de tableau *champ=>valeur à citer* 3. description 4. serveur
sql_replace	1. table 2. tableau *champ=>valeur à citer* 3. description

	4. serveur
sql_replace_multi	1. table 2. tableau de tableau *champ=>valeur à citer* 3. description 4. serveur
sql_delete	1. $table 2. $where 3. serveur

Un groupe un peu à part est formé de fonctions traitant spécifiquement des opérandes. : elles ne se connectent pas au serveur, mais retournent des chaînes dépendant de celui-ci :

- `sql_quote` prend une chaîne ou un nombre, retourne un nombre si l'argument était un nombre ou une chaîne représentant un entier, sinon retourne la chaîne initiale entourée d'apostrophes et avec les apostrophes protégées selon la syntaxe propre au serveur (un \ devant pour certains, une deuxième apostrophe pour d'autres) ;

- `sql_hex` prend une chaîne de chiffres hexadécimaux et retourne sa représentation dans le serveur SQL visé ;

- `sql_in` construit un appel à l'opérande IN, en traitant les éventuelles valeurs hexadécimales y figurant ;

- `sql_test_int` prédicat retournant Vrai si le type SQL fourni désigne un entier ;

- `sql_test_date` prédicat retournant Vrai si le type SQL fourni désigne une date ;

- `sql_multi` applique une expression SQL sur champ contenant un *bloc multi* (voir Réaliser un site multilingue) pour y prendre la partie correspondant à la langue indiquée ; l'intérêt d'effectuer cette opération au niveau SQL est essentiellement de demander simultanément un tri sur cette colonne.

Fonction	Arguments
sql_quote	1. valeur 2. serveur
sql_hex	1. valeur 2. serveur
sql_in	1. colonne 2. valeurs 3. vrai si négation souhaitée 4. serveur
sql_multi	1. colonne 2. langue 3. serveur
sql_test_date	1. type

	2. serveur
sql_test_int	1. type 2. serveur

Un groupe important est constitué par les fonctions manipulant les déclarations de bases et de tables. Pour des raisons historiques, cette première version de l'interface reprend quasi littéralement la syntaxe de MySQL3 et devra certainement être revue, en particulier pour y faire apparaitre la déclaration des jointures. Les fonctions `sql_create`, `sql_alter`, `sql_showtable` et `sql_drop_table` permettent de créer, modifier, voir et supprimer une table. Les fonctions `sql_create_view`, `sql_drop_view` permettent de créer ou supprimer une vue. Les fonctions `sql_listdbs`, `sql_showbase` et `sql_selectdb`permettent de voir les bases accessibles, de voir leur contenu et d'en sélectionner une. À noter que tous les hébergeurs n'autorisent pas forcément de telles actions ; SPIP essaiera de le deviner, en notant ses essais dans le fichier `spip.log`.

Fonction	Arguments
sql_create	1. nom de la table 2. tableau nom de colonne => type SQL et valeur par défaut 3. tableau nom d'index => colonne(s) 4. vrai si auto-incrément 5. vrai si temporaire 6. serveur
sql_alter	1. requête MYSQL Alter 2. serveur
sql_showtable	1. RegExp 2. vrai si table déclaré par SPIP 3. serveur
sql_drop_table	1. nom de la table 2. vrai s'il faut insérer la condtion *existe* 3. serveur
sql_create_view	1. nom de la vue 2. requête de selection de champs (créé par exemple avec `sql_get_select`) 3. serveur
sql_drop_view	1. nom de la vue 2. vrai s'il faut insérer la condtion *existe* 3. serveur
sql_listdbs	
sql_selectdb	1. nom de la base 2. serveur

sql_showbase	1. RegExp 2. serveur

Deux fonctions permettent de régler le codage des caractères lors des communications avec le serveur :

- `sql_set_charset`, demande d'utiliser le codage indiqué ;
- `sql_get_charset`, demande si un codage de nom donné est disponible sur le serveur.

Fonction	Arguments
sql_get_charset	1. RegExp 2. serveur
sql_set_charset	1. codage 2. serveur

Un dernier groupe de fonctions offre quelques outils de gestion des requêtes et des tables ; on se reportera à leurs homonymes dans la documentation des serveurs SQL. Signalons toutefois que `sql_explain` est utilisée implicitement par le débusqueur de SPIP, accessible par les boutons d'administrations de l'espace public, lorsqu'on lui demande le plan de calcul d'un boucle (ou de tout un squelette).

Fonction	Arguments
sql_optimize	1. requête 2. serveur
sql_repair	1. table 2. serveur
sql_explain	1. requête 2. serveur
sql_error	1. requête 2. serveur
sql_errno	
sql_version	

Hors groupe, la fonction générale `sql_query`, nom qu'aurait dû porter l'historique `spip_query` ; leur utilisation est de toutes façons à éviter.

Réalisation des portages

Cette section est destinée à ceux souhaitant porter SPIP sur d'autres serveurs SQL, ou ayant besoin d'informations plus techniques, notamment sur la gestion des versions de l'interface. Les fonctions du fichier `ecrire/base/abstract_sql.php` étudiées ci-dessus se contentent d'offrir une interface homogène aux différents serveurs, mais ne procèdent elles-mêmes à aucun calcul. C'est dans le fichier `ecrire/base/connect_sql.php` que se situe le travail effectif.

La fonction essentielle est `spip_connect` qui ouvre la connexion au serveur SQL indiqué par

son argument (ou, s'il est omis, le serveur principal) en repérant les connexions déjà faites. Cette ouverture consiste à inclure un *fichier de connexion* créé lors de l'installation de SPIP par les scripts présents dans le répertoire `install`. Un fichier de connexion se réduit pour l'essentiel à appliquer la fonction `spip_connect_db` aux valeurs fournies lors de l'installation.

La fonction `spip_connect_db` reçoit en particulier comme argument le type du serveur SQL. Ce type doit être le nom d'un fichier présent dans le répertoire `req`. Ce fichier est chargé et doit définir toutes les fonctions d'interfaces définies à la section précédente, plus la fonction `req_`*type*`_dist` qui sera immédiatement appliquée sur les mêmes arguments que `spip_connect_db`, type excepté. C'est cette fonction qui doit établir effectivement la connexion.

Porter SPIP sur d'autres serveurs SQL consiste donc à définir ce jeu de fonctions et à le placer dans le répertoire `req`.

Le gestionnaire de versions d'interface repose sur le deuxième argument de `spip_connect` qui indique la version, la version courante étant prise par défaut. Toutes les fonctions de l'interface sont définies dans le fichier `abstract_sql`, se nomment sql_*X* et sont les seules à se nommées ainsi. Elles se connectent toutes en appelant une variante de `spip_connect` dont le premier argument est le numéro de version de l'interface. Au cas où le fichier `abstract_sql` nécessiterait une révision, il sera renommé abstract_sql_*N*, et le Sed suivant lui sera appliqué (*N* désigne le numéro de version) :

```
s/\(sql_[A-Za-z_0-9 ]*\)/\1_N/
```

En appliquant également ce script aux extensions de SPIP fondées sur cette version, on leur permettra d'en appeler ses fonctions, qui seront chargées sans collision de noms, le Sed ayant préfixé le noms des anciennes avec leur numéro de version. Il faudra juste rajouter une instruction include portant sur le fichier abstract_sql_*N*. Du côté du portage, il faudra renommer pareillement les fichiers du répertoire `req`, et écrire les nouvelles versions.

La coexistence de plusieurs versions de l'interface pendant l'exécution de SPIP repose sur la structure décrivant le serveur. Celle-ci est en fait un tableau, contenant :

- `link`, *ressource* indiquant la connexion ;
- `db`, nom de la base de données ;
- `prefixe`, nom du préfixe de table ;
- `ldap`, nom de l'éventuel fichier décrivant le serveur LDAP.

Les autres entrées sont les numéros de versions disponibles, et leur valeur est le tableau des fonctions implémentant cette version de l'interface.

Les autres fonctions du fichier `connect_sql` concernent essentiellement la gestion des versions et le traitement de quelques cas particuliers de déclarations des tables standards de SPIP.

P.-S.

Le remplacement de spip_query dans l'ancien code de SPIP par ce jeu de fonctions a été réalisé en partie automatiquement par des scripts Sed. Ces scripts font partie des dépots des versions successives du code, et peuvent donc être récupérés pour être appliqués aux extensions de SPIP voulant migrer vers la nouvelle architecture. En voici la liste :

9916 9918 9919 10394 10492 10493 10497 10501 10504 10508 10509 10519 10520 10707 10742

Attention cependant : ces scripts ne fonctionnent pas pour n'importe quelle utilisation de spip_query, il est indispensable de lire chacun d'eux pour vérifier qu'ils s'appliquent bien au code à faire migrer.

La structure de la base de données

Juin 2001 — maj : 15 décembre

La structure de la base de données est assez simple. Certaines conventions ont été utilisées, que vous repérerez assez facilement au cours de ce document. Par exemple, la plupart des objets sont indexés par un entier autoincrémenté dont le nom est du type `id_objet`, et qui est déclaré comme clé primaire dans la table appropriée.

NB : cet article commence à dater, et personne n'a encore pris la peine d'en faire la mise à jour. Il faut le lire comme un élément permettant de comprendre le fonctionnement de SPIP, mais plus comme un outil de référence. Si vous souhaitez contribuer à la documentation en refondant cet article, surtout n'hésitez pas !

Contenu rédactionnel

Les rubriques : spip_rubriques

id_rubrique	bigint(21)
id_parent	bigint(21)
titre	text
descriptif	text
texte	longblob
id_secteur	bigint(21)
maj	timestamp(14)
export	varchar(10)
id_import	bigint(20)

- Chaque rubrique est identifiée par son **id_rubrique**.
- **id_parent** est l'*id_rubrique* de la rubrique qui contient cette rubrique (zéro si la rubrique se trouve à la racine du site).
- **titre**, **descriptif**, **texte** parlent d'eux-mêmes.
- **id_secteur** est l'*id_rubrique* de la rubrique en tête de la hiérarchie contenant cette rubrique. Une rubrique dépend d'une rubrique qui dépend d'une rubrique... jusqu'à une rubrique placée à la racine du site ; c'est cette dernière rubrique qui détermine l'*id_secteur*. Cette valeur précalculée permet d'accélérer certains calculs de l'espace public (en effet, les brèves sont classées par secteur uniquement, et non selon toute la hiérarchie).
- **maj** est un champ technique mis à jour automatiquement par MySQL, qui contient la date de la dernière modification de l'entrée dans la table.
- **export**, **id_import** sont des champs réservés pour des fonctionnalités futures.

Les articles : spip_articles

id_article	bigint(21)
surtitre	text
titre	text
soustitre	text
id_rubrique	bigint(21)
descriptif	text
chapo	mediumtext
texte	longblob
ps	mediumtext
date	datetime
statut	varchar(10)
id_secteur	bigint(21)
maj	timestamp(14)
export	varchar(10)
images	text
date_redac	datetime
visites	int(11)
referers	blob
accepter_forum	char(3)

- Chaque article est identifié par son **id_article**.
- **id_rubrique** indique dans quelle rubrique est rangé l'article.
- **id_secteur** indique le secteur correspondant à la rubrique susmentionnée (voir le paragraphe précédent pour l'explication de la différence entre les deux).
- **titre**, **surtitre**, **soustitre**, **descriptif**, **chapo**, **texte**, **ps** parlent d'eux-mêmes.
- **date** est la date de publication de l'article (si l'article n'a pas encore été publié, c'est la date de création).
- **date_redac** est la date de publication antérieure si vous réglez cette valeur, sinon elle est égale à « `0000-00-00` ».
- **statut** est le statut actuel de l'article : `prepa` (en cours de rédaction), `prop` (proposé à la publication), `publie` (publié), `refuse` (refusé), `poubelle` (à la poubelle).
- **accepter_forum** : permet de régler manuellement si l'article accepte des forums (par défaut, oui).
- **maj** : même signification que dans la table des rubriques.

- **export** est un champ réservé pour des fonctionnalités futures.
- **images** est un champ contenant la liste des images utilisées par l'article, dans un format particulier. Ce champ est généré par `spip_image.php3`.
- **visites** et **referers** sont utilisés pour les statistiques sur les articles. Le premier est le nombre de chargements de l'article dans l'espace public ; le deuxième contient un extrait de hash des différents referers, afin de connaître le nombre de referers distincts. Voir `inc-stats.php3`.

Les auteurs : spip_auteurs

id_auteur	bigint(21)
nom	text
bio	text
email	tinytext
nom_site	tinytext
url_site	text
login	tinytext
pass	tinyblob
statut	tinytext
maj	timestamp(14)
pgp	blob
htpass	tinyblob

- Chaque auteur est identifié par son **id_auteur**.
- **nom**, **bio**, **nom_site**, **url_site**, **pgp** sont respectivement le nom de l'auteur, sa courte biographie, son adresse e-mail, le nom et l'URL de son site Web, sa clé PGP. Informations modifiables librement par l'auteur.
- **email**, **login** sont son e-mail d'inscription et son login. Ils ne sont modifiables que par un administrateur.
- **pass** est le hash MD5 du mot de passe.
- **htpass** est la valeur cryptée (i.e. générée par crypt()) du mot de passe pour le .htpasswd.
- **statut** est le statut de l'auteur : `0minirezo` (administrateur), `1comite` (rédacteur), `5poubelle` (à la poubelle), `6forum` (abonné aux forums, lorque ceux-ci sont réglés en mode « par abonnement »).
- **maj** a la même signification que dans les autres tables.

Les brèves : spip_breves

id_breve	bigint(21)
date_heure	datetime
titre	text
texte	longblob
lien_titre	text
lien_url	text
statut	varchar(6)
id_rubrique	bigint(21)
maj	timestamp(14)

- Chaque brève est identifiée par son **id_breve**.
- **id_rubrique** est la rubrique (en fait, le secteur) dans laquelle est classée la brève.
- **titre**, **texte**, **lien_titre**, **lien_url** sont le titre, le texte, le nom et l'adresse du lien associé à la brève.
- **date_heure** est la date de la brève.
- **statut** est le statut de la brève : `prop` (proposée à la publication), `publie` (publiée), `refuse` (refusée).
- **maj** : idem que dans les autres tables.

Les mots-clés : spip_mots

id_mot	bigint(21)
type	varchar(100)
titre	text
descriptif	text
texte	longblob
maj	timestamp(14)

- Chaque mot-clé est identifié par son **id_mot**.
- Le **type** du mot-clé est le type, ou groupe, choisi pour le mot-clé. En définissant plusieurs types, on définit plusieurs classifications indépendantes (par exemple « sujet », « époque », « pays »...).
- **titre**, **descriptif**, **texte** parlent d'eux-mêmes.
- **maj** : idem que dans les autres tables.

Les sites syndiqués : spip_syndic

id_syndic	bigint(20)
id_rubrique	bigint(20)
id_secteur	bigint(20)
nom_site	blob
url_site	blob
url_syndic	blob
descriptif	blob

- Chaque site syndiqué est identifié par son **id_syndic**.
- **id_rubrique** et **id_secteur** définissent l'endroit dans la hiérarchie du site où viennent s'insérer les contenus syndiqués.

- **nom_site**, **url_site**, **descriptif** sont le nom, l'adresse et le descriptif du site syndiqué.
- **url_syndic** est l'adresse du fichier dynamique utilisé pour récupérer les contenus syndiqués (souvent il s'agit de *url_site* suivi de `backend.php3`).

Les articles syndiqués : spip_syndic_articles

id_syndic_article	bigint(20)
id_syndic	bigint(20)
titre	text
url	text
date	datetime
lesauteurs	text

- Chaque article syndiqué est identifié par son **id_syndic_article**.
- **id_syndic** réfère au site syndiqué d'où est tiré l'article.
- **titre**, **url**, **date**, **lesauteurs** parlent d'eux-mêmes.

Eléments interactifs

Les messages de forums : spip_forum

id_forum	bigint(21)
id_parent	bigint(21)
id_rubrique	bigint(21)
id_article	bigint(21)
id_breve	bigint(21)
date_heure	datetime
titre	text
texte	mediumtext
auteur	text
email_auteur	text
nom_site	text
url_site	text
statut	varchar(8)
ip	varchar(16)
maj	timestamp(14)
id_auteur	bigint(20)
id_message	bigint(21)

- Chaque message de forum est identifié par son **id_forum**.
- L'objet auquel est attaché le forum est identifié par son **id_rubrique**, **id_article** ou **id_breve**. Par défaut, ces valeurs sont égales à zéro.
- Le message parent (c'est-à-dire le message auquel répond ce message) est identifié par **id_parent**. Si le message ne répond à aucun autre message, cette valeur est égale à zéro.
- **titre**, **texte**, **nom_site**, **url_site** sont le titre et le texte du message, le nom et l'adresse du lien y attaché.
- **auteur** et **email_auteur** sont le nom et l'e-mail déclarés par l'auteur. Dans le cas des forums par abonnement, ils ne sont pas forcément identiques aux données enregistrées dans la fiche de l'auteur (i.e. dans la table spip_auteurs).
- **id_auteur** identifie l'auteur du message dans le cas de forums par abonnement.
- **statut** est le statut du message : `publie` (lisible dans l'espace public), `prive` (écrit en réaction à un article dans l'espace privé), `privrac` (écrit dans le forum interne dans l'espace privé), `off` (supprimé ou à valider, selon la modération des forums - a priori ou a posteriori).
- **ip** est l'adresse IP de l'auteur, dans les forums publics.
- **maj** a la même signification que dans les autres tables.

Les pétitions : spip_petitions

id_article	bigint(21)
email_unique	char(3)
site_obli	char(3)
site_unique	char(3)
message	char(3)
texte	longblob
maj	timestamp(14)

- **id_article** identifie l'article auquel est associée la pétition (une seule pétition par article).
- **email_unique**, **site_obli**, **site_unique**, **message** définissent la configuration de la pétition : l'adresse e-mail des signataires doit-elle être unique dans les signatures, l'adresse Web est-elle obligatoire, est-elle unique, un message attenant aux signatures est-il autorisé (`oui` ou `non`).
- **texte** est le texte de la pétition.
- **maj** : pareil que dans les autres tables.

Les signatures de pétitions : spip_signatures

id_signature	bigint(21)
id_article	bigint(21)
date_time	datetime
nom_email	text
ad_email	text
nom_site	text
url_site	text
message	mediumtext
statut	varchar(10)
maj	timestamp(14)

- Chaque signature est identifiée par son **id_signature**.

- **id_article** identifie l'article, donc la pétition sur laquelle est apposée la signature.
- **nom_email**, **ad_email**, **nom_site**, **url_site** sont le nom, l'adresse e-mail, ainsi que le site Web déclarés par le signataire.
- **message** est le message éventuellement entré par le signataire.
- **statut** est le statut de la signature : `publie` (acceptée), `poubelle` (supprimée) ; toute autre valeur donne la valeur de la clé de validation utilisée pour la confirmation par e-mail.
- **maj** a la même signification que dans les autres tables.

Les relations entre objets

Ces tables ne gèrent aucun contenu, simplement une relation entre les objets présents dans d'autres tables. Ainsi :

- **spip_auteurs_articles** spécifie la relation entre auteurs et articles. Si un *id_auteur* y est associé à un *id_article*, cela veut dire que l'auteur en question a écrit ou co-écrit l'article (il peut y avoir plusieurs auteurs par article, et vice-versa bien sûr).

- **spip_mots_articles** définit de même la relation de référencement des articles par des mots-clés.

Gestion du site

La table **spip_meta** est primordiale. Elle contient des couples (`nom`, `valeur`) indexés par le nom (clé primaire) ; ces couples permettent de stocker différentes informations telles que la configuration du site, ou la version installée de SPIP.

La table **spip_forum_cache** est utilisée afin d'adapter le système de cache à l'instantanéité des forums. Pour chaque fichier du cache ayant donné lieu à une requête sur la table spip_forum, la table spip_forum_cache stocke les paramètres de la requête (article, rubrique, brève et éventuel message parent du forum). Lorsqu'un message est posté, les paramètres du message sont comparés avec ceux présents dans spip_forum_cache, et pour chaque correspondance trouvée le fichier cache indiqué dans la table est effacé. Ainsi les messages n'attendent pas le recalcul régulier de la page dans laquelle ils s'insèrent pour apparaître dans l'espace public.

Indexation (moteur de recherche)

Six tables sont utilisées par le moteur de recherche. Elles se divisent en deux catégories.

Le dictionnaire d'indexation : spip_index_dico

Chaque mot rencontré au cours de l'indexation est stocké dans cette table, ainsi que les 64 premiers bits de son hash MD5. C'est le mot qui sert de clé primaire, permettant ainsi d'effectuer très rapidement des requêtes sur un début de mot ; on récupère alors le(s) hash satisfaisant à la requête, afin d'effectuer la recherche proprement dite dans les tables d'indexation.

Les tables d'indexation : spip_index_*

Ces tables, au nombre de cinq, gèrent chacune l'indexation d'un type d'objet : articles, rubriques, brèves, auteurs, mots-clés. Une entrée par mot et par objet est stockée. Chaque entrée contient le hash du mot (en fait les 64 bits de poids fort du hash MD5, cf. ci-dessus), l'identifiant de l'objet indexé (par exemple l'*id_article* pour un article), et le nombre de points associé à l'indexation du mot dans l'objet. Ce nombre de points est calculé en fonction du nombre d'occurences du mot, pondéré par le champ où ont lieu les occurences : une occurence dans le titre d'un article génère plus de points qu'une occurence dans le corps de l'article.

Le mécanisme d'indexation est expliqué plus en détail ici.

Autres fonctions avancées

Mutualisation du noyau SPIP

Février 2007 — maj : 11 août

La mutualisation du noyau de SPIP est possible depuis SPIP 1.9. Il s'agit de pouvoir gérer plusieurs sites SPIP sur un seul serveur ou hébergement en n'utilisant qu'une seule fois les fichiers du noyau de SPIP.

Cela permet un gain d'espace disque important, ainsi qu'une possibilité de mise à jour de SPIP simple de l'ensemble des sites en ne mettant à jour que le noyau.

Les évolutions de SPIP 1.9.1 simplifiaient un peu la procédure, mais c'est avec SPIP 1.9.2 et ses améliorations [1] que la mutualisation devient plus robuste permettant la mise en place d'un partage du noyau de SPIP.

Cet article explique la procédure pour SPIP 1.9.2, sur des serveurs Apache [2] autorisant la réécriture d'url (url_rewriting).

Il y a plusieurs méthodes pour arriver aux mêmes résultats, selon que l'on souhaite configurer directement la mutualisation depuis un hébergement ou depuis un serveur.

Le concept...

Depuis SPIP 1.9.2 les dossiers nécessaires au fonctionnement du noyau SPIP (ecrire, dist, oo), et ceux marquant l'activité d'un site (config, IMG, tmp, local) sont clairement identifiables et séparés. C'est cette séparation qui permet d'avoir plusieurs sites SPIP autonomes pour un même noyau de SPIP.

Cette autonomie repose sur l'existence d'un quatuor de répertoires par site, dans lesquels Spip va écrire les données résultant de l'activité du site. Ces répertoires sont au nombre de quatre, car on distingue les données temporaires et permanentes d'une part, et les données accessibles par http et celles qui ne le sont pas d'autre part, d'où quatre types de données. Les deux répertoires inaccessibles par http seront protégées par un .htaccess installé automatiquement par SPIP (les administrateurs du serveur pourront même mettre ces répertoires en dehors de l'arborescence servie par http).

Avant SPIP 1.9.2, ces quatre types de données n'étaient pas clairement distingués, et se retrouvaient dans les répertoires IMG, CACHE, ecrire et ecrire/data. Avec SPIP 1.9.2, ces quatre répertoires sont distincts et nommés par les constantes PHP suivantes :

```
define('_NOM_TEMPORAIRES_INACCESSIBLES', "tmp/");
define('_NOM_TEMPORAIRES_ACCESSIBLES', "local/");
define('_NOM_PERMANENTS_INACCESSIBLES', "config/");
define('_NOM_PERMANENTS_ACCESSIBLES', "IMG/");
```

Dans une installation de Spip non mutualisée, ces répertoires sont créés à la racine du site, lors de

l'application de la fonction `spip_initialisation` sur les quatre valeurs ci-dessus. Pour obtenir la mutualisation des sources de SPIP, il suffit de savoir associer une URL spécifique à son quatuor de répertoires spécifiques, en appelant la fonction `spip_initialisation` sur quatre autres valeurs, déduites de l'URL.

Créer les bons répertoires

Pour démarrer la mutualisation, il faut tout d'abord partir d'un site fonctionnel. Pour les exemples, nous partirons d'un site appelé par l'url http://example.org/ stocké physiquement dans /home/toto/public_html/.

Il faut créer un répertoire (par exemple nommé 'sites') qui va contenir les répertoires des sites mutualisés, à la racine de la distribution (au même niveau que /ecrire).

- mutualisation d'un répertoire :
Si l'on souhaite que les adresses http://example.org/premier_site/ et http://example.org/deuxieme_site/ appellent chacun un site spip, il faut créer alors les sous-répertoires /premier_site et /deuxieme_site dans le répertoire /sites, puis dans chacun d'eux créer les répertoires /config, /IMG, /tmp, /local. Ces quatre répertoires doivent être accessibles en écriture. Il sera possible par la suite d'ajouter un répertoire /squelettes aussi, à côté de ces répertoires.

- mutualisation sous-domaine et domaine (quelques idées) :
Si l'on souhaite que l'adresse http://toto.example.org/, http://exemple.tld/ et http://utilisateur.example.com/ appellent chacun un site spip, on peut envisager de créer dans /sites les répertoires /toto, /exemple.tld, et /example.com/utilisateur

Remarque : Toutes les url doivent pointer à la racine de la distribution SPIP, c'est-à-dire dans /home/toto/public_html/. C'est le rôle que vont remplir soit un fichier .htaccess soit la configuration du serveur Apache expliqués plus loin.

Des redirections bien faites !

Pour que SPIP reconnaisse qu'un site est mutualisé, il faut que le script spip.php (appelé par index.php) soit executé. Celui-ci cherchera, grâce à un code ajouté dans /config/mes_options.php si l'URL appelée correspond à un site mutualisé ou non. Pour cela, il faut qu'une adresse http://example.org/premier_site/ ou http://example.org/deuxieme_site/ soit redirigée vers http://example.org/ pour executer alors index.php...

C'est le rôle attribué au fichier .htaccess (ou directement dans la configuration du serveur Apache)

Il faut copier et renommer le fichier htaccess.txt à la racine de la distribution en .htaccess, puis le modifier :

Pour autoriser la réécriture d'URL (rien ne change normalement) : `RewriteEngine On`

Si la distribution SPIP est dans un sous-répertoire, modifier rewritebase. Ici, le site est à la racine donc : `RewriteBase /`

Enfin, dans réglages personnalisés, ajouter le code suivant pour que les répertoires /premier_site , /deuxieme_site et /troisieme_site soient traités depuis la racine de la distribution :

```
#Mutualisation
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)$ /$1/
[R,L]
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)/(.*) /$2
[QSA,L]
```

Le premier rewriterule redirige les adresses http://example.org/premier_site vers http://example.org/premier_site/. Le deuxième redirige tout ce qu'il y a derrière /premier_site/ à la racine, par exemple : http://example.org/premier_site/art... est redirigé vers http://example.org/article112. (Cependant, cette redirection est transparente, le nom de l'url ne change pas, il est toujours http://example.org/premier_site/art..., même et surtout pour php !)

Et si SPIP est dans un sous répertoire ?
Dans ce cas là, les fichiers de spip se trouvent dans /home/toto/public_html/spip/ , SPIP est appelé par http://example.org/spip/, les sites mutualisés par http://example.org/spip/premier_site/ ou http://example.org/spip/deuxieme_site/.

Il faut alors mettre dans le .htaccess :

```
RewriteEngine On
RewriteBase /spip/

#Mutualisation
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)$ /spip/
$1/ [R,L]
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)/(.*)
/spip/$2 [QSA,L]
```

Et pour rediriger tous les répertoires comme des sites mutualisés ?

Il vous est possible **à la place** des rewriterules ci-dessus d'utiliser un code générique, utilisable quelque-soit le nom du répertoire du site mutualisé :

- à la racine

```
RewriteCond %{REQUEST_URI} !^/(config|dist|ecrire|IMG|oo|plugins|
sites|squelettes|tmp|local)/(.*)
RewriteRule ^[^/]+/(.*) /$1 [QSA,L]
```

- ou dans un dossier /spip :

```
RewriteCond %{REQUEST_URI} !^/spip/(config|dist|ecrire|IMG|oo|
plugins|sites|squelettes|tmp|local)/(.*)
RewriteRule ^[^/]+/(.*) /spip/$1 [QSA,L]
```

Et pour les domaines et sous domaines ?
Par un heureux hasard, il n'y a rien à faire ici puisqu'ils pointent normalement déjà vers la racine de la distribution SPIP...

Mutualiser selon l'URL grâce à mes_options.php

C'est le fichier /config/mes_options.php à la racine de la distribution qui va réaliser la majeure partie du travail : il doit chercher si une URL est à mutualiser ou non, et initialiser SPIP en fonction.

De nombreux cas peuvent se présenter, entre les répertoires, les domaines et sous domaines. PHP peut utiliser deux variables pour tester les URLs qui ont appelé le script :

```
$_SERVER['REQUEST_URI']; // contient tout ce qu'il y a derrière le
nom de domaine : /premier_site/article112 par exemple...
$_SERVER['SERVER_NAME']; // contient le nom du domaine et sous
domaine : utilisateur.example.org par exemple
```

Ce sont ces deux variables qui vont être comparées à une expression régulière pour extraire le nom du répertoire qui contient la mutualisation.

Un moyen simple de mutualiser tous les répertoires est de copier le code suivant :

```
<?php
if ( preg_match(',/([a-zA-Z0-9_-]+)/?,',$_SERVER['REQUEST_URI'],
$r)) {

   if (is_dir($e = _DIR_RACINE . 'sites/' . $r[1]. '/')) {

       $cookie_prefix = $table_prefix = $r[1];

       define('_SPIP_PATH',
           $e . ':' .
           _DIR_RACINE .':' .
           _DIR_RACINE .'dist/:' .
           _DIR_RESTREINT);

       spip_initialisation(
           ($e . _NOM_PERMANENTS_INACCESSIBLES),
           ($e . _NOM_PERMANENTS_ACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_INACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_ACCESSIBLES)
           );

      $GLOBALS['dossier_squelettes'] = $e.'squelettes';

       if (is_readable($f =
$e._NOM_PERMANENTS_INACCESSIBLES._NOM_CONFIG.'.php')) include($f);
   }
}
?>
```

La ligne preg_match récupère le nom d'un dossier dans l'arborescence de l'URL, par exemple 'premier_site' dans http://example.org/premier_site/ ... Puis le script tente une mutualisation si le répertoire /sites/premier_site/ existe.

Si SPIP est dans un répertoire /spip, il faut modifier la première ligne par : `if (preg_match(',/spip/([a-zA-Z0-9_-]+)/?,', $_SERVER['REQUEST_URI'], $r)) {`

Il faut dire dire à SPIP quel est le préfixe de table de base de données utilisé, ce que fait `$cookie_prefix = $table_prefix = $r[1];`
Cette ligne va créer des tables MySQL avec des préfixes ayant le nom du répertoire contenant le site : mon_site_article à la place de spip_article par exemple. Cela permet d'héberger tous les sites sur une seule et même base de données. Vous pouvez garder le préfixe par défaut (spip), mais il faut penser à avoir plusieurs bases de données différentes pour chaque site.
Pour cela, mettre à la place :

```
$cookie_prefix = $r[1];
$table_prefix='spip';
```

La fonction spip_initialisation admet quatre paramètres qui sont l'adresse de chacun des répertoires nécessaires au fonctionnement du site mutualisé.

`$GLOBALS['dossier_squelettes'] = $e.'squelettes';` définit l'emplacement du

dossier squelettes du site mutualisé.

Enfin les dernières lignes chargent un éventuel sites/premier_site/config/mes_options.php.

Information :

Toute modification du fichier config/mes_options.php du noyau SPIP affectera les options de tous les sites hébergés.

Par exemple, mettre dans ce fichier : `$type_urls = 'propres' ;`

Donnera par défaut à tous les sites ce type d'url... Mais chaque site peut le changer dans son propre /sites/mon_site/config/mes_options.php.

Configurer apache pour les domaines et sous-domaines

La mutualisation côté serveur, pour ce qui concerne la gestion des sous-domaines ou des domaines reste simple, mais nécessite de créer quelques redirections d'URL dans la configuration du serveur Apache pour tenir compte de ces sites.

Voici un exemple de configuration pour un serveur nommé 'exemple.tld' (ici un SPIP mutualisé) utilisant des sous-domaines (SPIP appelé par http://utilisateur.example.org/spip/).

Le noyau SPIP est dans '/home/toto/public_html/spip/'.

Il faut alors créer les répertoires /sites/exemple.tld/ , /sites/exemple.tld/utilisteur/.

Le fichier de configuration se situe dans Apache 2/linux dans /etc/apache2/sites_availables/default (vous pouvez aussi créer un nouveau fichier dans le dossier sites_availables et l'activer).

```
# SERVEUR exemple.tld
# SPIP par sous domaine...
<VirtualHost *>
       ServerName exemple.tld
        ServerAlias *.exemple.tld

       # Redirection vers le SPIP noyau
        DocumentRoot "/home/toto/public_html"
        <Directory "/home/toto/public_html/">
                AllowOverride All
                Order allow,deny
                Allow from all
        </Directory>

        # Seule l'adresse http://utilisateur.exemple.tld/spip/*
doit être redirigée

        # (utilisateur.exemple.tld/spip/* ->
/home/toto/public_html/*)
        RewriteCond %{SERVER_NAME} (www\.)?([^.]+)\.example\.net$
        RewriteRule ^/spip/(.*) /home/toto/public_html/spip/$1
[QSA,L]

        # (utilisateur.exemple.tld/* ->
/home/toto/public_html/sites/exemple.tld/utilisateur/*)
        RewriteCond %{SERVER_NAME} (www\.)?([^.]+)\.example\.net$
```

```
        RewriteRule (.*) /home/toto/public_html/sites/exemple.tld/
%1/$1 [QSA,L]

</VirtualHost>
```

Il est par contre nécessaire de tester ces mutualisations possibles dans /config/mes_options.php :

```
<?php
# pour utilisateur.exemple.tld/spip/
if ( preg_match(',(.*)\.exemple\.tld/spip/?,',
$_SERVER['SERVER_NAME'].$_SERVER['REQUEST_URI'],$r)) {

   if (is_dir($e = _DIR_RACINE . 'sites/exemple.tld/' . $r[1].
'/')) {

      $cookie_prefix = $table_prefix = $r[1];

      define('_SPIP_PATH',
           $e . ':' .
           _DIR_RACINE .':' .
           _DIR_RACINE .'dist/:' .
           _DIR_RESTREINT);

       spip_initialisation(
           ($e . _NOM_PERMANENTS_INACCESSIBLES),
           ($e . _NOM_PERMANENTS_ACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_INACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_ACCESSIBLES)
           );

      $GLOBALS['dossier_squelettes'] = $e.'squelettes';

       if (is_readable($f =
$e._NOM_PERMANENTS_INACCESSIBLES._NOM_CONFIG.'.php')) include($f);
   }
}
?>
```

Note sur les sauvegardes et les restaurations

Chaque site copie ses fichiers de sauvegardes dans le répertoire /sites/premier_site/tmp/dump (ou /sites/premier_site/tmp/upload/*login* pour les sauvegardes d'un administrateur restreint). Les restaurations se font par le même dossier.

Attention :

A l'heure actuelle, SPIP enregistre dans la base de données, pour ce qui concerne le dossier /IMG des sites mutualisés une adresse sites/premier_site/IMG/* au lieu de IMG/* comme pour un site SPIP seul.

Une restauration ne fonctionnera donc que si elle s'effectue dans le site qui a créé la sauvegarde.

Une astuce pour restaurer le site ailleurs consiste à éditer le fichier dump.xml et à remplacer toutes les occurrences (à l'exception de celles des déclarations dir_img et dir_logo dans l'ouverture de la

balise spip) :
- sites/premier_site/IMG/

  - par (SPIP seul) : IMG/
  - ou (SPIP mutualisé) : sites/mon_nouveau_site/IMG/

Inversement, pour passer un SPIP seul dans un répertoire mutualisé, il faut remplacer toutes les occurrences de :
- IMG/
- par : sites/mon_site/IMG/

(Cette difficulté sera corrigée dans la prochaine version de SPIP.)

Notes

[1] Nouvelle distribution des répertoires pour clarifier la mutualisation, correction des problèmes de logins et de noms de sites qui se mélangeaient parfois entre les sites mutualisés, fonction d'initialisation d'un site à mutualiser plus claire

[2] Pour information : ces procédures ont été testées avec Apache 2.0.55 et PHP 5.1.2 sur serveur Ubuntu Dapper Drake ainsi que sur PHP 5.1.6 sur serveur Ubuntu Edgy Eft

Le validateur XML intégré

Février 2007 — maj : Décembre 2007

Depuis que le World Wide Web Consortium a lancé la Web Accessibility Initiative, les problématiques de validation XML et d'accessibilité du Web aux déficients visuels ont convergé. SPIP s'est très tôt intéressé aux problèmes d'accessibilité, avec la version 1.5 en 2002. En revanche, il a longtemps récusé XML eu égard à la rareté des pages conformes : l'abondance de pages HTML non XHTML fait que les navigateurs ont leur propre analyseur, et les langages basés sur XML, comme SVG, MathML et XQuery, ont connu un démarrage très lent. Toutefois, la convergence des problématiques d'accessibilité et de validation d'une part, et l'implémentation en natif de SVG dans plusieurs navigateurs de l'autre, a amené SPIP, dans sa version 1.8.1, à offrir une interface avec des outils de validation comme Tidy ou le validateur officiel du W3C. Ces outils, comme l'indiquent sans détour les pages d'accueil de leur site, souffrent de quelques limitations, révèlent des divergences dans leurs messages d'erreur, et ne sont pas installables par l'internaute moyen [1] Ils sont de plus inopérants face aux nouvelles technologies comme Ajax, qui modifient incrémentalement le contenu d'une page Web. Plus grave encore, les *Document Type Definition* du W3C ont elles aussi des limitations, y compris la DTD XHTML 1.0 dite *stricte* : alors que la spécification officielle interdit l'emboîtement des balises `a`, `label` ou `form`, la définition formelle de la grammaire décrite considère comme valides les constructions `<label for='x'><label...` ou `<form action='.'><div><form...` par exemple [2].

Face à cette situation, SPIP 1.9.2 innove radicalement en proposant un *validateur extensible, intégré, incrémental et optionnel*, fondé sur le *Simple Analyzer for XML* fourni en standard par PHP. Cette pluralité d'aspects le destine autant aux webmestres qu'aux graphistes et aux développeurs d'extensions de SPIP avec évidemment des manipulations différentes.

Validation pour les webmestres

Pour les webmestres, mettre simplement dans le fichier `mes_options.php` l'instruction `$xhtml = 'sax';` provoquera une remise en page du code HTML produit par un squelette, chaque ligne comportant une seule balise ouvrante et au plus un attribut, avec renfoncement de la

marge gauche proportionnellement au nombre de balises ouvrantes non fermées. Dans le cas où la page se révèle invalide, ce travail sera finalement ignoré, et le texte initial sera envoyé au client HTTP. Dans les ceux cas, la mise en cache éventuelle a évidemment lieu après l'action du validateur, qui se contente donc ici d'une simple mise en page.

Quelques précisions en ce qui concerne le traitement des attributs. Leur valeur sera systématiquement entourée d'une apostrophe, sauf si elle contient une apostrophe auquel cas elle sera entourée de guillemets (et si elle contient aussi des guillemets, ils seront écrits `&quot;`). En raison d'une erreur de conception de SAX [3], les entités XML seront préalablement converties dans le *charset* du site à l'exception de `& < >` et `&quot;` qui sont correctement traitées par SAX (car c'est indispensable). La liste des entités et leur valeur seront déduites du DOCTYPE indiqué par le squelette ; à défaut SPIP prendra un ensemble prédéfini équivalent à la DTD du latin1 enrichie de `&euro;`, `&oelig;` et `&OElig;` définis dans la DTD des symboles spéciaux.

Validation pour les graphistes

Pour les créateurs de squelettes, le validateur est disponible à travers le débusqueur introduit en SPIP 1.8. Cet outil n'est visible que des administrateurs du site, car ils disposent des boutons d'administration lorsqu'ils visitent les pages de l'espace public. L'un de ces boutons se nomme *Analyse XML* et déclenche explicitement la demande de validation de la page visitée. Cette analyse est d'abord confiée à SAX qui, même en l'absence de `DOCTYPE`, repère :

- les mauvais emboîtements de balises ouvrantes et fermantes ;
- les attributs sans valeur ;
- les attributs sans délimiteur ;
- les attributs sans espace avant le suivant ;
- les entités XML mal écrites (notamment un `&` littéral, alors que XML impose d'écrire `&`).

À la première erreur rencontrée, le débusqueur de SPIP essaiera immédiatement de retrouver de quel squelette provient l'erreur (il y en a plusieurs en cas d'inclusion) et à quelle ligne, en donnant des liens vers les fichiers sources (cette détermination ne peut pas toujours aboutir, une marge d'erreur est à prendre en compte).

Si cette première analyse est correcte, cette nouvelle version de SPIP va plus loin en prenant en compte le DOCTYPE de la page. Il peut être de type `PUBLIC` ou `SYSTEM`, et dans le premier cas la DTD sera mise en cache pour accélérer les analyses ultérieures. Chaque DTD peut en inclure d'autres, qui seront chargées pareillement. Afin d'éviter les redondances de calcul tout en tenant compte du système d'inclusion conditionnelle des DTD, SPIP met également en cache une structure de données spécifique qu'il déduit de la lecture de toutes les définitions d'éléments, d'attributs et d'entités issus du DOCTYPE indiqué. A l'aide de cette structure, SPIP *valide* la page, autrement dit vérifie que :

- tous les noms d'attributs utilisés dans une balise sont acceptés par la DTD ;
- tous les attributs obligatoires d'une balise sont présents ;
- toutes les valeurs d'attributs sont conformes au *motif* indiqué éventuellement dans la DTD ;
- tous les attributs de type ID ont une valeur composée de lettres, chiffres, souligné, tiret et deux-points ;
- tous les attributs de type IDREF ou IDREFS référencent des ID effectivement présents dans la page ;
- toutes les entités XML utilisées sont définies dans la DTD ;

- tous les noms de balises utilisées sont définis dans la DTD ;
- toute balise non vide est autorisée à l'être par la DTD (par exemple un `img` non vide sera dénoncé) ;
- toute balise vide est autorisée à l'être par la DTD (par exemple un `tr` vide sera dénoncé) ;
- toute balise utilisée est fille d'une balise l'admettant comme telle dans la DTD ;
- toute balise devant apparaître avant une de ces sœurs apparaît effectivement ainsi (par exemple `head` avant `body`)
- toute balise devant apparaître un nombre fixe de fois apparait effectivement ainsi (par exemple `title` une unique fois dans `head`).

Si des erreurs sont détectées, le validateur affichera un tableau de toutes les erreurs, avec leur nombre d'occurrences, des liens vers les lignes incriminées et des suggestions de corrections déduites automatiquement des constructions autorisées par la DTD. En l'absence d'erreur, le débusqueur affichera le code selon la mise en page décrite précédemment.

Validation pour les développeurs

Les pages Web en accès restreint ont particulièrement besoin d'un validateur intégré pour être mises au point, puisque les outils de validation externes n'ont par définition pas accès à ces pages. En ce qui concerne les scripts, standards ou venant d'extension, de l'espace de rédaction de SPIP on leur applique la validateur XML en plaçant `$GLOBALS['transformer_xml'] = 'valider_xml';` dans le fichier mes_options.php. L'espace privé de SPIP 1.9.2 a été rendu conforme XHTML 1.0 grâce à ce mécanisme. Affecter cette variable globale à `'indenter_xml'` provoquera l'indentation du source HTML si celui-ci est conforme XML, sans chercher à le valider.

Il est également possible de lancer, avec la souris, l'analyse XML du résultat d'un script Ajax activé dans l'espace de rédaction. Un tel script ne retournant pas une page HTML complète mais seulement un fragment, le validateur intégré fabriquera une page avec le DOCTYPE courant, un en-tête réduit à la balise Title, et une balise Body composée de ce fragment. Une fenêtre s'ouvrira avec le résultat de l'analyse, comme pour une page normale, pendant que la fenêtre initiale recevra le résultat du script Ajax comme d'habitude. En raison d'une spécification du W3C inutilisable quant au modèle d'événements [4], ce lancement du validateur n'est pas provoqué par un bouton spécifique de la souris, mais par un clic pendant lequel une au moins des deux touches Alt ou Meta est enfoncée.

Par ailleurs, le validateur de SPIP peut être appliqué à n'importe page présente sur le Web. Tout site SPIP installé à l'URL `http://`*u* contient la page `http://`*u*`ecrire/valider_xml` que les auteurs du site peuvent invoquer explicitement pour valider des pages non limitées à celle du site. Ce validateur ne s'applique cependant qu'à des documents XML ; en l'absence de DOCTYPE, la DTD XHTML1.0 transitionnal sera prise.

Comme le suggèrent les lignes ci-dessus, le validateur s'applique à n'importe quel DOCTYPE, y compris ceux référençant des DTD résidantes sur le site. Rendre accessibles des pages Web, c'est être plus rigoureux dans l'utilisation des attributs et des balises, on peut donc facilement se construire un outil de validation d'accessibilité en écrivant des DTD moins laxistes que la XHTML 1.0 dite stricte. Remplacer `#IMPLIED` par `#REQUIRED` dans les attributs jugés indispensables est immédiat. Forcer `input, select, textarea` et `button` a être exclusivement des filles de la balise `label` est à peine plus difficile. En outre, le validateur accepte n'importe quel motif (au sens de PCRE) comme type d'attribut, il sera alors appliqué à chaque occurrence de cet attribut dans la page analysée.

Enfin, il est possible de définir ses propres règles de validation associées à un attribut. Les types

usuels ID, IDREF et IDREFS sont implémentés par les fonctions validerAttribut_ID, validerAttribut_IDREF, validerAttribut_IDREFS. Il suffit d'introduire de nouveau type S1 ... Sn dans la DTD, et de définir les fonctions associées dans le fichier mes_options pour provoquer des contrôles personnalisés. En fin d'analyse, la fonction surchargeable `inc_valider_passe2` est appelée, afin d'effectuer les contrôles rétrospectifs (c'est ici que les attributs IDREF sont vérifiés comme référençant des attributs effectivement présents). Cette interface de programmation est encore un peu frustre et sera revisée après expérimentation. Mais elle permet dès à présent de mettre sur pied très rapidement de nouvelles spécifications d'accessibilité.

Notes

[1] Le validateur du W3C se présente sous la forme d'un archive de plus de 1Mo, comportant essentiellement un script CGI en Perl, nécessitant plusieurs modules de ce langage et son interface avec le serveur HTTP. En outre, l'analyse syntaxique et la validation ne sont pas effectuées par ce programme, mais déléguées à l'analyseur *onsgmls* programmé en C++ et nécessitant donc d'être compilé après chargement des sources du projet OpenSP de 1Mo également (certains systèmes d'exploitation incluent cependant déjà l'exécutable de 128Ko résultant). Cette difficulté explique sans doute la rareté de ses installations, qui du coup sont souvent saturées.

[2] Cette carence est justifiée par ce passage de la recommandation XHTML qui affirme : *The HTML 4 Strict DTD forbids the nesting of an 'a' element within another 'a' element to any descendant depth. It is not possible to spell out such prohibitions in XML.* Cette affirmation n'est etayée par aucune démonstration ou référence scientifique. Les théorèmes établissant les pouvoirs de l'analyse syntaxique automatique ont été énoncés et démontrés dès les années 1950, et contredisent une telle affirmation. W3C, go to school ?

[3] À la rencontre de ce qu'il considère comme un lexème, SAX appelle un *preneur d'événements*. Faute de considérer les attributs comme des lexèmes, il provoquera la même séquence d'appel pour les 3 textes suivants :

```
&eolig;&EOlig;<a ...
&eolig;<a title='&EOlig;' ...
<a title='&eolig;' href='&EOlig;' ....
```

En conséquence, le preneur d'événements *Entité XML* ne saura pas si l'entité provoquant son appel fait partie de l'élément de texte précédant la prochaine balise, ou si elle fait partie de l'un des attributs de cette balise, et lequel. L'ambiguïté ne peut même pas être levée à l'aide des numéros de ligne et de colonne (ou de caractère du flux), qui indiquent dans tous les cas l'emplacement précédant la balise.

[4] Alors que le système d'exploitation le plus répandu dans le monde suivait pour une fois sagement les leçons d'Unix et plus précisément de X-Window, en décrivant un bouton de souris par un masque de bits, le W3C a cru bon de produire une spécification incompatible et mal conçue, que ne suit pratiquement aucun navigateur.

Le système de pagination

Juillet 2006 — maj : 7 mai

Lorsqu'une boucle renvoie plusieurs dizaines d'articles (ou, pour une pétition, plusieurs milliers de signatures), il n'est pas souhaitable, voire impossible, de tout afficher sur une seule page.

On préférera alors répartir les résultats en plusieurs pages, avec un système de navigation page à

page. S'il est possible de réaliser cela avec des boucles SPIP ordinaires, c'est tout de même relativement complexe.

Aussi SPIP 1.9 introduit-il un système simplifié de pagination des résultats d'une boucle.

Exemple

Au plus simple, ce système est composé d'un critère et d'une balise :

- le critère `{pagination}` s'ajoute sur la boucle à paginer ;
- la balise `#PAGINATION`, placée dans une des parties optionnelles (« avant » ou « après ») de la boucle, affiche la « pagination ».

- <B_page>
- #PAGINATION
-
- <BOUCLE_page(ARTICLES) {par date} {pagination}>
- #TITRE
- </BOUCLE_page>
-
- </B_page>

Si le site comporte 90 articles publiés, cette boucle affichera la liste des dix plus anciens articles, surplombée de liens conduisant vers la page qui affiche les dix suivants, les dix d'après, etc. Ces liens sont numérotés comme suit :

0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | ...

Le numéro à partir duquel les résultats sont affichés est passé dans l'url via un paramètre `{debut_page=x}` portant le même nom (ici, « page ») que la boucle concernée. (Ce paramètre est exploitable dans une autre boucle via le critère classique `{debut_page,10}`.)

A noter : le nombre total de liens affichés est limité ; des points de suspension permettent, le cas échéant, d'aller directement à la toute fin de la liste, ou de revenir au tout-début.

Ancres de pagination

La balise `#PAGINATION` comporte une ancre html qui permet au navigateur d'afficher directement la partie de la page qui est paginée ; toutefois si l'on veut mettre les liens de pagination *en dessous* de la liste des articles, il faut pouvoir placer l'ancre *au-dessus* de la liste.

C'est à cela que sert la balise `#ANCRE_PAGINATION`, qui retourne l'ancre en question, et interdit à la balise `#PAGINATION` suivante d'afficher son ancre.

<B_page>

#ANCRE_PAGINATION

<BOUCLE_page(ARTICLES) {par date} {pagination}>

#TITRE

</BOUCLE_page>

#PAGINATION

</B_page>

Le nombre total de résultats

Dans une boucle avec le critère `{pagination}`, `#TOTAL_BOUCLE` affiche le nombre d'éléments effectivement retournés, c'est-à-dire 10 sur les pages pleines, et 10 ou moins sur la dernière page de résultats.

Pour afficher le nombre d'éléments qui **auraient été retournés** si le critère `{pagination}` n'avait pas été là, utilisez la balise `#GRAND_TOTAL`.

<B_pagination>

#ANCRE_PAGINATION

<BOUCLE_pagination (ARTICLES) {pagination}>

#TITRE

</BOUCLE_pagination>

Il y a au total #GRAND_TOTAL articles, cette page en affiche #TOTAL_BOUCLE

</B_pagination>

indiquera : « Il y a au total 1578 articles, cette page en affiche 10. »

Changer le pas de la `{pagination}`

Le nombre standard de 10 éléments par page peut être modifié par un paramètre supplémentaire dans le critère.

Ainsi

```
<BOUCLE_page (ARTICLES) {pagination 5}>
#TITRE <br />
</BOUCLE_page>
```

retournera les titres de cinq articles à partir de *debut_page*.

Le paramètre en question peut lui-même être composé comme on le souhaite à partir d'autres balises, notamment `#ENV{xx}`, ce qui permet de faire un affichage à la demande très complet.

La pagination dans les squelettes inclus

Si votre pagination doit fonctionner dans un squelette inclus, vous **devez** passer en paramètre de la commande `INCLURE` la formulation `{self=#SELF}` ; ceci permet au squelette inclus de se calculer avec la bonne valeur du paramètre `debut_xxx`, et se justifie qui plus est par un besoin de sécurité (la balise `#PAGINATION` est en effet calculée à partir de l'URL complète de la page).

Styles de la pagination

La pagination est constituée d'une série de liens, et d'un numéro de page correspondant à la page actuelle doté de la class « .on » : on définira donc les styles pour `a` et `.on` pour en personnaliser

l'apparence.

Choisir le modèle de pagination

Depuis SPIP 1.9.1, la balise #PAGINATION accepte un paramètre `{modele}`, qui permet de modifier le résultat de la balise.

Ainsi `#PAGINATION{precedent_suivant}` affichera des liens vers les pages précédentes et suivantes. Les liens seront les suivants

> page précédente | page suivante

`#PAGINATION{page}` affichera quelque chose de la forme suivante

> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ...

`#PAGINATION{page_precedent_suivant}` affichera quelquechose comme

> ≤ 1 | 2 | 3 | 4 | 5 | 6 | ≥

Il est possible de définir d'autres modèles de pagination, qui devront s'appeler pagination_modele. Pour plus d'info, lire la documentation sur les modèles.

P.-S.

> **Attention :** beaucoup plus simple, ce mécanisme de pagination est aussi *incompatible* avec celui SPIP-Contrib', qui lui a servi de matrice. Si vous utilisiez la contrib, il vous faudra revoir les squelettes concernés.

La syndication de contenus

Juillet 2006 — maj : 3 janvier

Avec SPIP 1.9, le système de syndication (voir l'article « Les fichiers backend ») s'enrichit : il est désormais possible d'échanger l'adresse des documents joints aux articles (*podcasting*), de transporter d'un site à l'autre les mots-clés (*tags*) associés aux articles ainsi que leur rubrique (ou *catégorie*). On peut aussi, si on le désire, syndiquer le *contenu intégral* des articles.

Dans tout ce qui suit, on considère que le flux de syndication offert par le site source est suffisamment riche pour avoir prévu toutes les possibilités qu'offre notamment le squelette *dist/backend.html* de SPIP.

Référencement rapide du site

On repère d'abord sur le site source l'URL de son flux de syndication au format RSS (ou Atom). Selon les cas, cette adresse est indiquée directement sur la page, et/ou est « découverte » automatiquement par le navigateur, qui affiche alors une icône caractéristique. Si les squelettes du site source le prévoient, SPIP 1.9 peut aussi découvrir cette adresse de syndication, et il suffit d'indiquer l'adresse du site pour se voir proposer de le syndiquer.

Une fois la syndication activée, les articles présents dans le flux de syndication du site source sont repris sur le site récepteur.

Décider ce que l'on veut émettre

Le webmestre du site source peut décider de ce qu'il met dans son flux RSS : ce choix peut bien évidemment se faire en modifiant le squelette *dist/backend.html*, ou en s'en inspirant pour créer un nouveau flux.

Mais la page de configuration dans l'espace privé propose une option très importante pour la syndication : elle permet de décider si le flux RSS du site comportera l'intégralité du contenu des articles, au format HTML, ou seulement un petit résumé (au format texte). Dans le premier cas (qui est la configuration par défaut du système), les articles récents du site sont entièrement lisibles avec un lecteur RSS : ils sont aussi entièrement « recopiables » d'un site à l'autre. Cela permet par exemple de réaliser automatiquement des sites miroirs, ou des sites composés d'un contenu produit sur d'autres sites.

Décider ce que l'on veut récupérer

Si le site source ne diffuse pas son contenu intégral, il est bien évident que la syndication ne pourra pas en récupérer plus. Mais dans le cas d'une diffusion intégrale, SPIP peut récupérer ce contenu HTML et l'afficher, images comprises.

Site par site, on peut choisir ce que l'on veut récupérer par syndication : le contenu HTML des articles (si disponible) ou un simple résumé au format texte.

On peut également décider de ce que l'on fait des articles qui disparaissent des flux (qui sont en général limités aux 15 articles les plus récents du site) : on peut décider que SPIP les élimine de la base de données (après une période de deux mois), et/ou les passe immédiatement en mode « refusé ». Ces dernières options permettent par exemple de gérer un portail sur des flux très rapides (agences de presse, tags très populaires de sites de photographie, etc) ; ou encore de maintenir un miroir fidèle d'un site (en éliminant les articles qui seraient dépubliés).

SPIP considère que si la date d'un élément de flux RSS et « très vieux » (plus d'un an) ou trop loin dans le futur (plus de 48 heures), c'est qu'il s'agit probablement d'une erreur. Il inscrit donc la date du jour à la place de celle qu'il a trouvé. On pourrait avoir besoin, cependant, de syndiquer des informations qui, à dessein, fournissent une date située à plus de 2 jours dans le futur. Par exemple, on peut se servir de fuls RSS pour annoncer des évènements qui auront lieu dans le futur. C'est le cas, dans la galaxie SPIP, du site SPIP Party.

Pour syndiquer ses rendez-vous via le flux RSS dédié, une variable de personnalisation nommée `$controler_dates_rss`, qui vaut vrai par défaut.

Si dans votre fichier `config/mes_options.php`, ajouter la ligne `$controler_dates_rss = false;`, le test n'est plus effectué et les dates du futur, ou du « lointain » passé, seront bien prises en compte.

Décider ce que l'on veut afficher

- la *source* :

De manière habituelle, `#NOM_SITE` représente le nom du site syndiqué, qui est donc la « source » de l'article. Cependant avec le développement des agrégateurs de contenu (les portails par exemple), la véritable source de l'article peut être un autre site. Le format RSS a prévu ce cas en définissant un champ `<source>` indiquant la véritable source de l'article. Le moteur de syndication de SPIP récupère ces données quand elles sont présentes, et les balises `#SOURCE` et `#URL_SOURCE` permettent d'afficher respectivement le nom et l'adresse de la source.

- **les *tags* :**

Si les mots-clés affectés aux articles sont correctement renseignés dans le flux RSS, ils sont récupérés par le site récepteur ; ils n'arrivent pas individuellement dans la table des mots-clés, mais sont stockés en vrac dans le champ `tags` de la table `spip_syndic_articles`.

Si le flux de syndication utilise la notation standard `<dc:subject>Tag</dc:subject>`, le tag est noté tel quel. SPIP pousse cette notion un peu plus loin en transmettant aussi l'adresse de la page du mot-clé, grâce aux microformats. Le tag est alors récupéré avec son lien, sous la forme : `<a rel="tag" href="adresse de la page du mot-clé">Mot-clé</a>`.

Sur le site destinataire, ces tags sont affichés dans l'espace privé sous le descriptif de l'article, et sont affichables sur le site public grâce à la balise `#TAGS` (on verra plus loin comment la filtrer pour faire un affichage sélectif des tags).

Note : SPIP prévoit une gestion spécifique des tags en provenance des sites del.icio.us (bookmarks collectifs), flickr (photographie) et connotea (annotation d'articles scientifiques), de manière à pouvoir leur attribuer un URL (qui n'est pas prévu dans leurs flux RSS respectifs).

- **la rubrique :**

Dans de nombreuses applications (systèmes de blogs, répertoire de liens...) la catégorie (ou encore *directory*) d'un article est l'équivalent de ce que SPIP appelle la *rubrique*. Il était donc naturel d'utiliser la notion RSS standard de `<category>...</category>` pour faire connaître l'appartenance de notre article à telle ou telle rubrique.

De même qu'avec les tags, SPIP récupère cette information et l'affiche via `#TAGS`.

- **les documents joints :**

Les documents qui, dans l'espace privé, figurent en bas de la page de visualisation d'un article, soit dans la section « Documents » soit, pour les images, dans la section « Portfolio », sont eux aussi transmis dans le flux RSS, en utilisant la notion `<enclosure ... />`. C'est ce qu'on appelle le *podcasting*, désormais une fonction standard de SPIP [1].

L'information sur les *enclosures* est elle aussi récupérée dans la balise `#TAGS`, mais elle est affichée de façon différenciée dans l'espace privée, sous forme d'un petit trombone pour chaque fichier.

Utiliser la balise `#TAGS`

Comme on l'a vu ci-dessus, la balise `#TAGS` affiche en vrac les liens vers les mots-clés, la rubrique et les documents joints. Mais par la grâce des microformats, les liens vers chacun de ces concepts sont « marqués » de la façon suivante :
- `<a rel="tag" ...>` pour les tags/mots-clés
- `<a rel="directory" ...>` pour la rubrique/catégorie
- `<a rel="enclosure" ...>` pour les documents joints/podcast

Si l'on veut n'afficher que l'un de ces types de tags, on utilisera le filtre `afficher_tags` en précisant le type souhaité en argument :

[(#TAGS|afficher_tags{directory})]

[(#TAGS|afficher_tags{tag})]

[(#TAGS|afficher_tags{enclosure})]

(Par défaut `[(#TAGS|afficher_tags)]` est équivalent à `[(#TAGS|afficher_tags{'tag,directory'})]`.)

Pour les documents joints, le filtre spécifique `afficher_enclosures` permet d'afficher les trombones plutôt que des liens classiques :

[(#TAGS|afficher_enclosures)]

* * *

Manipuler le contenu HTML des articles syndiqués

Cas pratique : notre site syndique un photoblog, qui diffuse systématiquement un petit commentaire suivi de la photographie. Cette dernière se présente sous la forme d'une balise HTML `<img ... />`. Une fois ce photoblog syndiqué en version HTML complète dans notre site, nous pouvons décider de n'afficher que la photo, sans le commentaire. Il nous faut alors extraire la balise `<img />` ; cela peut se réaliser grâce au filtre `extraire_balise{xxx}`, qui récupère la première balise HTML `<xxx />` d'un contenu.

A partir de là tout est possible :
- `[(#DESCRIPTIF|extraire_balise{img})]` affiche la photo ;
- `[(#DESCRIPTIF|extraire_balise{img}|extraire_attribut{src})]` son URL ;
- `[(#DESCRIPTIF|extraire_balise{img}|extraire_attribut{width})]` sa largeur ;
- on peut même en modifier le style avec, par exemple :

```
[(#DESCRIPTIF|extraire_balise{img}|
        inserer_attribut{style,'border: double red 4px;'})]
```

> Remarque : les contenus HTML provenant d'un site extérieur sont par définition considérés comme « non contrôlés », et donc potentiellement problématiques s'ils contiennent des balises mal fermées, ou du javascript ; SPIP leur applique donc systématiquement le filtre `safehtml` avant affichage.

Autres filtres liés à la syndication

Ces filtres permettent de convertir les tags de syndication d'un format à un autre, afin de pouvoir par exemple remettre au format RSS des « tags » récupérés par syndication, et enregistrés dans notre base de données sous forme de microformats : `tags2dcsubject`, `enclosure2microformat`, `microformat2enclosure`.

Références

- RSS 2.0
- microformats
- rel=tag
- rel=enclosure
- rel=directory

Notes

[1] Attention les fichiers eux-mêmes ne sont pas recopiés : seules leur URL et les données associées (titre, taille, format) sont récupérées. Par ailleurs il faut signaler, pour les puristes, qu'on prend ici quelque liberté avec la norme RSS, qui interdit normalement d'avoir plusieurs *enclosures* dans un même article

Le calendrier de [SPIP 1.8.2]

Août 2005 — maj : Décembre 2007

SPIP 1.8.2 permet de visualiser dans l'espace public des calendriers avec le même affichage que celui de l'espace privé, et plus généralement de construire des agendas quelconques bénéficiant des outils de mises en pages de ces calendriers. Cette possibilité est fournie par un nouveau critère de boucle et trois nouveaux filtres.

Le critère `agenda` possède un nombre variable d'arguments et peut donc s'écrire de deux façons :
- {agenda *datetime*, *type*, *AAAA*, *MM*, *JJ*}
- {agenda *datetime*, periode, *AAAA*, *MM*, *JJ*, *AAAA2*, *MM2*, *JJ2*}

Les deux premiers arguments sont :

1. un nom de champ SQL de type *datetime* (par exemple `date` ou `date_redac` pour la table Articles et `date_time` pour les Breves) ; cet argument est obligatoire.
2. un *type* de calendrier (`jour`, `semaine`, `mois` ou `periode`) ; la valeur par défaut est `mois`.

Ces deux arguments doivent être littéraux (autrement dit ils ne peuvent être calculés dynamiquement par `#ENV` ou toute autre balise).

Les trois prochains arguments sont optionnels et peuvent être indiqués par des balises (avec et sans filtre) :

1. *YYYY* une chaîne d'exactement 4 chiffres indiquant une année ;
2. *MM* une chaîne d'exactement 2 chiffres indiquant un mois ;
3. *JJ* une chaîne d'exactement 2 chiffres indiquant un jour.

Si ces valeurs sont omises ou nulles, elles sont remplacées par celle de la date courante. Ces paramètres représentent la date d'un jour dans la période choisie :
- si le type est `jour`, on affichera les éléments dont la date correspond au jour spécifié,
- si le type est `semaine`, on affichera les éléments dont la date est dans la semaine qui contient le jour spécifié,
- si le type est `mois`, on affichera les éléments dont la date est dans le mois qui contient le mois spécifié (dans ce cas, le paramètre de jour *JJ* est facultatif).

Par exemple :

<B_semaine>

<BOUCLE_semaine(ARTICLES) {agenda date, semaine} {par date}>

#TITRE

</BOUCLE_semaine>

</B_semaine>

affiche une liste des articles publiés dans la semaine actuelle.

<BOUCLE_art_principale(ARTICLES) {id_article}>

<B_mememois>

<BOUCLE_mememois(ARTICLES) {agenda date, mois, (#DATE|annee), (#DATE|mois)} {par date}>

#TITRE

</BOUCLE_mememois>

</B_mememois>

</BOUCLE_art_principale>

affiche une liste des articles publiés le même mois que l'article actuel.

Dans le cas où l'argument *type* (le deuxième argument) vaut `periode`, trois autres arguments peuvent être spécifiés à la fin du critère. Alors :
- *YYYY*, *MM*, *JJ* correspondront à la date de début de la période,
- *YYYY2*, *MM2*, *JJ2* correspondront à la date de fin de la période.

Si le deuxième trio spécifiant la date de fin est omis, la date courante sera prise comme date de fin, et si le premier trio est également absent, la période de sélection couvrira toute la vie du site (pour les sites avec beaucoup d'articles, le temps d'exécution risque d'être excessif si d'autres critères n'en limitent pas le nombre).

Par exemple :

<BOUCLE_art_principale(ARTICLES) {id_article}>

<B_apres>

<BOUCLE_apres(ARTICLES) {agenda date, periode, (#DATE|annee), (#DATE|mois), (#DATE|jour)} {par date}>

#TITRE

</BOUCLE_apres>

</B_apres>

</BOUCLE_art_principale>

liste les articles qui ont été publiés après l'article actuel.

Pour mettre en page les éléments sélectionnés par une boucle (en particulier une boucle avec un critère **agenda**) sous forme de calendrier, SPIP fournit trois filtres. Ces filtres fournissent un affichage similaire au calendrier de l'espace privé avec le même système de navigation.

- Le filtre `agenda_memo` s'applique sur une balise de date (par exemple `#DATE` ou `#DATE_MODIF`) et prend quatre paramètres :

1. un descriptif
2. un titre
3. une URL représentant l'élément ayant ce titre et ce descriptif (par exemple `#URL_ARTICLE`)
4. un nom de *classe* CSS

Si la balise sur laquelle `agenda_memo` s'applique n'est pas nulle, il se contente de mémoriser la date et les trois premiers arguments dans un tableau indexé par le dernier argument (le nom de

classe CSS) et ne retourne rien (aucun affichage).

L'utilisation du dernier argument comme index pour l'élément à mémoriser permet d'avoir plusieurs calendriers par page. De plus, la classe spécifiée ici sera attribuée à cet élément dans l'affichage en calendrier fourni par **agenda_affiche**. Ainsi, on peut donner des styles différents aux éléments. La feuille *calendrier.css* fournit 28 styles différents qui donnent un exemple de différents styles de calendrier.

- Le filtre `agenda_affiche` s'applique sur une balise retournant le nombre d'éléments à afficher (en général `#TOTAL_BOUCLE`) et prend trois paramètres :

1. un texte qui sera affiché si la balise filtrée ne retourne rien (0 élément à afficher) ;
2. un type de calendrier (`jour`, `semaine`, `mois` ou `periode`) ;
3. des noms de *classes* CSS utilisées dans l'appel du filtre précédent qui permettent de filtrer les éléments à afficher.

Si la balise filtrée est nulle, ce filtre retourne son premier argument. Sinon il retourne les éléments mémorisés par le filtre **agenda_memo** mis en page dans un calendrier du type demandé.

Seul les éléments indexés par **agenda_memo** avec une des classes CSS indiquées dans le dernier argument seront affichés (ou alors tous les éléments si ce paramètre est omis). Ainsi on peut filtrer les éléments pour les répartir dans plusieurs calendriers sur la même page.

Le type `periode` restreindra l'affichage à la période comprise entre le plus vieil élément et le plus récent effectivement trouvés, ce qui permet de donner dans le critère une période très large sans récupérer un affichage trop long.

Exemple :

<BOUCLE_memorise(ARTICLES) {agenda date, semaine}{par date}>[

(#DATE|agenda_memo{#DESCRIPTIF,#TITRE,#URL_ARTICLE})

]</BOUCLE_memorise>

[(#TOTAL_BOUCLE|agenda_affiche{<:aucun_article:>,semaine})]

<//B_memorise>

affiche les articles publiés dans la semaine actuelle sous forme de calendrier.

- Enfin, le filtre `agenda_connu` teste si son argument est l'un des quatre types de calendriers connus (`jour`, `semaine`, `mois` ou `periode`).

> Ce critère et ces filtres sont utilisés par les nouveaux squelettes `agenda_jour.html`, `agenda_semaine.html`, `agenda_mois.html` et `agenda_periode.html`, appelés à partir du squelette `agenda.html` qui indique dans son en-tête les feuilles de style et fonctions javascript nécessaires (mais remplaçables à volonté). Ces squelettes fournissent donc un exemple représentatif d'utilisation.

Tidy : Correction XHTML 1.0

Avril 2005 — maj : 16 décembre

Les pages produites par SPIP peuvent être analysées par le Le validateur XML intégré disponible depuis SPIP 1.9. Il existe aussi, depuis SPIP 1.8, une interface avec Tidy ; elle exige une installation séparée mais a l'avantage d'essayer, souvent avec succès, de corriger automatiquement les erreurs de validation.

Principe général

Tidy est un outil (externe à SPIP) qui permet de transformer du code HTML 4 *relativement propre* en code XHTML 1.0 transitional valide. Cet outil aide les webmestres à conformer leurs sites aux recommandations XHTML, même dans les messages de forums librement composés par les visiteurs.

> **Important :** Tidy n'est pas un outil « magique » : il est incapable de transformer du code « très sale » en code conforme. Face à certaines « erreurs » de code, il refuse purement et simplement de fonctionner. Des interventions manuelles sur les erreurs qu'il dénonce sont donc à prévoir..

- **Étape 1 :** il convient, avant tout, de créer du code aussi propre et conforme que possible, avant même le passage par Tidy. Cela se fait à plusieurs niveaux :
— tout d'abord, SPIP produit, dans ses traitements typographiques, du code propre, et à chaque version plus conforme (« *compliant* ») ; l'espace privé et les squelettes de SPIP 1.8 sont conformes XHTML transitionnal, dans [spip19] ils sont conformes XHTML strict.

- **Étape 2 :** si Tidy est présent sur le serveur, et si le webmestre active cette option (voir plus loin), alors SPIP fait passer les pages produites par Tidy, qui va alors tenter de nettoyer les pages et de les transformer en pages conformes.

- **Étape 3 :** si le traitement a bien fonctionné (Tidy n'a pas rencontré d'erreur « bloquante »), alors la page affichée est bien du XHTML 1.0 ; en revanche, si Tidy n'a pas fonctionné (voir plus loin), alors c'est la page d'origine qui est affichée — dans ce cas (c'est un outil important à prendre en compte), SPIP affiche un bouton d'administration signalant l'« erreur Tidy », et créé un fichier récapitulant les différentes pages ayant rencontré des erreurs.

> Encore une fois, afin de ne pas préter à l'outil des possibilités « magiques » qu'il n'a pas, et à ses développeurs des intentions qui ne sont pas les leurs, il est important de noter qu'il ne s'agit pas de se reposer entièrement sur Tidy, dont les limites sont connues, mais de l'intégrer dans une logique plus large de mise en conformité :
> — d'abord en améliorant le code produit par SPIP,
> — en utilisant ensuite Tidy à la fois comme outil de « nettoyage » du code, mais aussi *en proposant une indication des erreurs* pour permettre au webmestre d'améliorer son code.
>
> La mise en conformité du code ne peut reposer sur une solution purement technique, mais sur un travail personnel pour lequel SPIP offre un outil de suivi.

Installer Tidy

- **Tidy, extension de PHP**

Tidy existe en tant qu'extension de PHP. C'est la façon la plus simple de l'utiliser, l'outil étant alors directement utilisable par le webmestre. Pour déterminer sa présence, on pourra consulter la page `ecrire/?exec=info` de son site pour obtenir la configuration de son serveur et la liste des extensions de PHP disponibles.

N.B. : Tout retour d'expérience de la part de webmestres intéresse les développeurs — sur la liste spip-dev. (Nous avons besoin de retours d'expérience sur les versions 1 et 2 de Tidy, c'est-à-dire sur des sites en PHP 4, en PHP 5, mais également des installations via PEAR.)

- **Tidy comme programme indépendant**

Il est par ailleurs possible d'utiliser Tidy en ligne de commande (c'est-à-dire en tant que programme indépendant de PHP s'exécutant directement sur le serveur).

> Cette version est particulièrement pratique, puisque :
> — il existe des versions de Tidy déjà compilées pour la plupart des systèmes d'exploitation,
> — il est souvent possible et simple d'installer ces versions de Tidy sur un hébergement sans avoir d'accès *root*,
> — certains administrateurs de sites ont subi des incompatibilités lors de l'installation de Tidy en extension PHP (avec, semble-t-il, ImageMagick) ; la version en ligne de commande ne provoque pas ce genre de problème.

Avant toute chose, vérifiez que Tidy n'est pas déjà présent sur votre serveur. Pour cela, installez dans le fichier `mes_options.php` les lignes suivantes :

```
define('_TIDY_COMMAND', 'tidy');

$xhtml = true;
```

Et vérifiez sur votre site public si les pages sont modifiées (soit transformées en XHTML, soit affichage du message « Erreur tidy »). Si cela ne fonctionne pas, pensez à supprimer ces deux lignes, et essayez d'installer Tidy selon la méthode suivante (ou demandez à la personne responsable de votre hébergement de le faire).

Vous pouvez installer une version déjà compilée de Tidy correspondant à votre système.
— On trouvera ces versions sur le site officiel de Tidy ; il en existe pour Linux, divers *BSD, MacOS X, etc.
— Décompressez l'archive téléchargée, et installez le fichier « tidy » sur votre site.
— Vérifiez les droits d'exécution de ce fichier sur le serveur (si nécessaire, passez les droits en « 777 »). (Si vous avez un accès SSH à votre serveur, vous pouvez tester le programme directement depuis le terminal. Si vous n'avez pas un tel accès, rien de grave, passez aux étapes suivantes, sachant que vous serez plus démuni si cela ne fonctionne pas du premier coup.)
— Configurez l'accès à ce fichier en indiquant le chemin d'accès en le définissant ainsi :

```
define('_TIDY_COMMAND', '/usr/bin/tidy');

$xhtml = true
```

Si le chemin indiqué dans `_TIDY_COMMAND` est correct, alors Tidy sera déclenché lors des affichages des pages de votre site public.

> **Important.** La définition de `_TIDY_COMMAND` doit se trouver dans `/ecrire/mes_options.php` et non à la racine du site dans `/mes_fonctions.php`. Cela est dû au fonctionnement assez spécifique du système (post-traitement des fichiers tirés du cache de SPIP).
>
> La partie `$xhtml = true;`, en revanche, fonctionne comme une « variable de personnalisation » ; vous pouvez, si vous souhaitez faire des essais ou restreindre le fonctionnement à une partie du site, définir cette variable au niveau du fichier d'appel, par exemple `article.php3` si vous ne voulez passer tidy que sur les articles.

Nettoyez votre code...

Encore une fois, il faut bien comprendre que Tidy n'est capable de rendre conforme que du code qui

est déjà, à l'origine, très propre. Avec les squelettes livrés avec SPIP et le code produit par défaut par SPIP, cela ne pose aucun problème : le code est très proche du HTML 4 conforme (« *compliant* »), aussi Tidy n'a aucun mal à en faire du XHTML 1.0 transitional parfaitement conforme.

Lorsque Tidy a bien fonctionné, vous constatez dans le code source de vos pages :
— que le « DOCTYPE » de votre page est devenu « XHTML... »,
— que le code est joliment indenté,
— que l'ensemble passe la validation W3C sans difficulté.

Si le DOCTYPE n'est pas modifié, c'est que Tidy a renoncé à corriger la page, dans laquelle il a rencontré des erreurs impossibles (pour lui) à corriger.

Les erreurs peuvent avoir deux sources : les squelettes, et le texte des articles.

- **Vos squelettes ne sont eux-mêmes pas conformes ;** dans ce cas, il faut les corriger. C'est le cas le plus fréquent.

Vous pouvez, ici, commencer par désactiver Tidy (passer la variable `$xhtml` à `false`), et passer vos pages à un validateur (le Le validateur XML intégré ou le W3C Validator) en visant le conformité HTML 4.01 transitional au minimum.

Une fois vos squelettes aussi proches que possible du HTML 4, Tidy n'aura pas de difficulté à produire du XHTML conforme. Si ces pages sont complètement conformes, c'est encore mieux ! (Et pas impossible : les squelettes livrés avec SPIP le sont déjà).

- **Certaines articles contiennent des codes erronés**

SPIP laissant les rédacteurs travailler en « code source », ceux-ci peuvent insérer du code non conforme à l'intérieur de leurs articles. (Par exemple, dans la documentation de `www.spip.net`, on trouve à certains endroits l'utilisation de balises HTML `<tt>...</tt>` que Tidy considère comme inacceptables.)

Aussi, une fois les squelettes nettoyés, on pourra chercher à corriger les textes de certains articles (cela concerne, donc, les insertions de HTML directement dans les articles ; encore une fois, le code produit par SPIP est essentiellement conforme, et ne provoque pas de « blocage » de Tidy).

Pour cela, outre l'apparition, sur les pages concernées, d'une bouton d'aministration intitulé « Erreur Tidy », SPIP tient à jour en permanence un fichier `/ecrire/data/w3c-go-home.txt` qui contient la liste des pages impossibles à valider [1]. Une fois vos squelettes rendus propres (paragraphe précédent), le nombre de pages défaillantes devrait être limité aux articles contenant du code HTML non accepté par Tidy.

> Il est assez difficile de définir précisément ce que Tidy considère comme une « erreur insurmontable ». Par exemple, les balises mal fermées ne sont pas réellement considérées comme impossible à corriger (par exemple, passer en italique dans un paragraphe et fermer l'italique dans un autre paragraphe fabrique du code HTML non conforme, que Tidy parvient cependant à bien corriger). Le plus souvent, il s'agit de balises insérées à la main totalement inexistantes (par exemple : taper `<bt>` à la place de `<br>`), ou de balises HTML considérées comme obsolètes dans le HTML 4 (telles que `<tt>` ou `<blink>`), que Tidy refusera purement et simplement de traiter.

Conclusion

Encore une fois, l'outil Tidy ne doit surtout pas être considéré comme un produit « miracle » : il ne transforme pas du code sale en code conforme. Son intégration dans SPIP suit donc une logique d'accompagnement d'un processus de nettoyage :

— SPIP lui-même continue à produire du code de plus en plus propre ;

— Tidy sert alors à mettre la dernière touche de nettoyage sur du code déjà très « compliant » (au passage, en résolvant quelques incompatibilités difficiles à gérer avec un code unique entre le HTML et le XHTML, telles que certaines balises auto-fermantes en XHTML, et non fermées en HTML, telles que `<br />`) ;

— Tidy est ensuite utilisé pour identifier les erreurs de codage dans le code source des articles eux-mêmes (lors de l'insertion, assez fréquent, de code HTML « à la main » dans le corps des articles).

Notes

[1] Ce fichier titre son nom de l'article W3C go home!, publié sur sur uZine, qui critiquait l'acharnement des prophètes de la *compliance* à emm... les webmestres qui se contentent de faire des pages Web ; l'essentiel, rappelons-le, est de publier sans se casser la tête. Si en plus on peut le faire de manière conforme, tant mieux, et c'est l'objet de cette passerelle SPIP-Tidy.

Sécurité : SPIP et IIS

Empêcher l'accès aux données confidentielles de SPIP sous Microsoft IIS

Août 2004 — maj : Décembre 2007

Sécurité par défaut de SPIP

Il existe deux dossiers « sensibles » dans SPIP, ce sont `CACHE` et `ecrire/data`. Le premier comporte tous les fichiers qu'utilise votre cache pour accélérer l'affichage des pages, il est donc moyennement sensible, mais le deuxième stocke les journaux d'activité de spip (les `spip.log`) et vous permet notamment de créer `dump.xml`, le fichier de sauvegarde de la base de données.

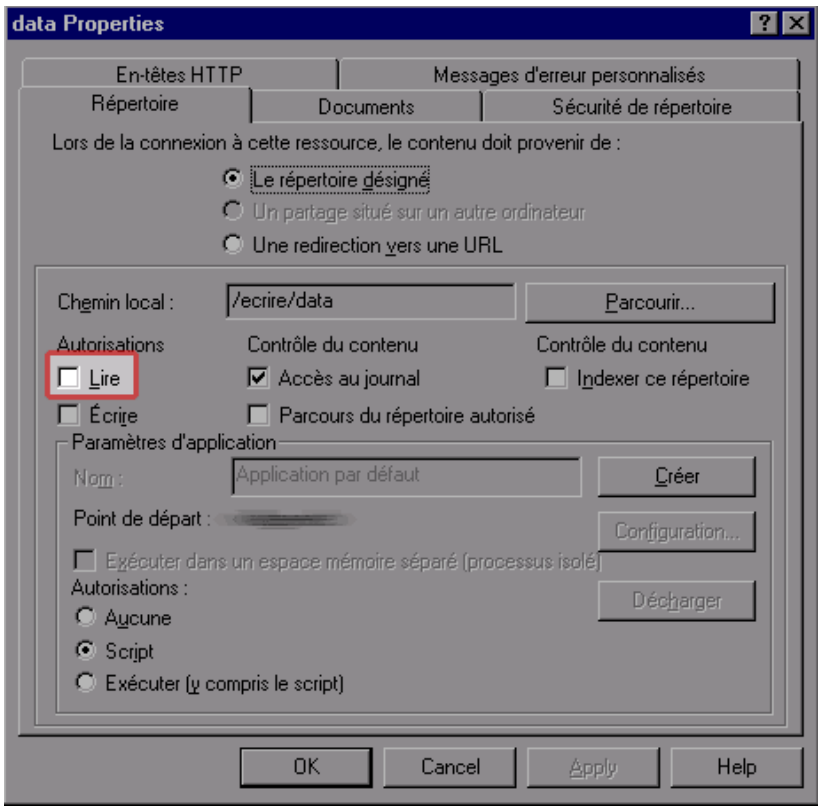
Or le fichier `dump.xml` contient des données très sensibles : en particulier on peut y voir *tous* les articles, même s'ils ne sont pas rendus publics sur le site, sans compter qu'il liste également les identifiants et les mots de passe [1] des rédacteurs et administrateurs du site.

La sécurité de tous ces fichiers est assurée traditionnellement par des fichiers de configuration d'accès nommés `.htaccess`. SPIP génère automatiquement ces fichiers pour empêcher l'accès aux données sensibles stockées sur le serveur : vous pouvez vérifier que `CACHE` et `ecrire/data` contiennent chacun un fichier `.htaccess`. Hélas, ces fichiers fonctionnent sous Apache (le serveur Web libre faisant tourner la majorité des sites Web de l'Internet) mais pas sous IIS (Internet Information Services, le serveur Web de Microsoft).

Protéger ses données sous IIS : une étape de plus

Si votre site SPIP est installé sur un IIS, n'importe qui peut donc voir les dossiers censément sécurisés via `.htaccess` : il faut donc les protéger.

Pour protéger un dossier sur votre site : allez dans le panneau d'administration de votre serveur Web, faites un clic droit sur le dossier concerné, cliquez sur « propriétés », et dans l'onglet « Répertoire » décochez la case « Lire ».



Le panneau de propriétés du dossier /ecrire/data/

Décocher la case "Lire" suffit à protéger le dossier exactement comme le fait Apache avec les fichiers .htaccess

Faites cette opération pour chacun des deux dossiers `CACHE` et `ecrire/data`. Si la manipulation est bonne, vous ne devriez plus pouvoir accéder aux fichiers de ces dossiers à travers le serveur web. Testez votre configuration en essayant d'afficher `http://www.votresite.com/ecrire/data/spip.log` avec votre navigateur. Vous devriez obtenir un message du type « Accès refusé ».

Notes

[1] Les mots de passe sont chiffrés par SPIP, mais gardez bien à l'esprit qu'aucune protection n'est inviolable

Exposer un article dans une liste

Avril 2004 — maj : 10 janvier

La balise `#EXPOSE` permet de mettre en évidence, dans un menu ou dans une liste, l'objet principal de la page où l'on se trouve.

L'objet qui est « exposé » est l'article (ou la brève, la rubrique, le mot-clé ou l'auteur) qui appartient au « contexte » courant. Dans le cas des rubriques, la traversée de la hiérarchie est gérée, ce qui permet d'« exposer » l'arborescence des rubriques qui contient l'article affiché.

Par défaut, SPIP remplace la balise `#EXPOSE` par « **on** » si l'objet correspond au contexte ; sinon la

balise est simplement ignorée.

Toutefois la balise `#EXPOSE` accepte un ou deux arguments, qui permettent de préciser ce qui doit s'afficher pour l'objet exposé, et ce qui doit s'afficher pour les autres objets. Ainsi `[(#EXPOSE{oui,non})]` affichera « oui » pour l'objet exposé, et « non » pour les autres.

Afficher différemment l'article exposé

Utilisée simplement, la balise `#EXPOSE` permet, par exemple, dans un menu de navigation, de changer l'apparence du lien vers l'article que l'on est précisément en train de consulter. Pour modifier le style des liens de ce menu, on placera la balise `#EXPOSE` de la manière suivante dans le squelette `article.html` :

```
<BOUCLE_principale (ARTICLES) {id_article}>

 <B_menu>
 <ul>
  <BOUCLE_menu (ARTICLES) {id_rubrique}>
  <li>
   <a href="#URL_ARTICLE"[ class="(#EXPOSE)"]>
   #TITRE
   <a>
  </li>
  </BOUCLE_menu>
 </ul>
 </B_menu>

 #TEXTE

</BOUCLE_principale>
```

avec les styles suivants :

```
a { color: blue; }
a.on { color: red; font-weight: bold; }
```

L'article ainsi exposé se distinguera dans la liste par un affichage visuellement différent.

Désactiver le lien exposé

Avec un peu d'astuce il est possible de désactiver le lien vers l'article exposé et dans le même temps de choisir le style à appliquer :

```
<B_menu>
<ul>
 <BOUCLE_menu(ARTICLES){id_rubrique}>
```


<#EXPOSE{span,a href="#URL_ARTICLE"}[class="(#EXPOSE)"]>
#TITRE
</#EXPOSE{span,a}>

</BOUCLE_menu>

</B_menu>

créera le code HTML suivant, où les balises `<a>` sont remplacées par des `<span>` :

```
<ul>
<li><a href="article1.html">Tout sur ma soeur</a></li>
<li><span class="on">Tout sur moi</span></li>
<li><a href="article3.html">Tout sur mon frère</a></li>
</ul>
```

ce qui s'affiche ainsi :

- Tout sur ma soeur
- Tout sur moi
- Tout sur mon frère

P.-S.

Historique :

Cette fonctionnalité a été introduite par SPIP 1.7.1 avec la balise `#EXPOSER` dont la syntaxe complète était la suivante : `[(#EXPOSER|oui,non)]`

Depuis SPIP 1.8.2, celle-ci est désormais obsolète. Il est donc conseillé d'utiliser `#EXPOSE`, dont la syntaxe complète, ci-dessus présentée, est plus conforme au modèle général des balises de SPIP.

Les variables de personnalisation

Août 2002 — maj : 16 décembre

Certains comportements des pages de votre site peuvent être modifiés au moyen de variables PHP. Ces variables sont normalement définies par SPIP, mais, pour obtenir une personnalisation plus fine du site, le webmestre peut les modifier.

Où indiquer ces variables ?

Inutile d'entrer dans le code source de SPIP lui-même pour fixer ces variables (ouf !).

- **Pour l'ensemble du site**

Si vous voulez fixer ces variables pour l'intégralité du site, vous pouvez les indiquer comme globales, avec une syntaxe un peu différente, dans un fichier intitulé `mes_fonctions.php` (`mes_fonctions.php3` dans les versions antérieurs à SPIP 1.9), placé à la racine du site, ou, **de préférence**, dans votre dossier `squelettes/` (cf. Où placer les fichiers de squelettes ?).
Il faudra éventuellement créer ce fichier, et entourer les définitions de vos variables par les marqueurs `<?php` et `?>`, voir les exemples ci-dessous.

- **Pour chaque type de squelette**

[SPIP 1.4] Vous pouvez aussi définir ces variables squelette par squelette. Pour cela, il faut les installer *au début* du fichier PHP appelant le squelette (par exemple `article.php3`, `rubrique.php3`...). Elles s'insèrent naturellement à côté des variables obligatoires `$fond` et `$delais`. Voir les exemples.

Les variables du texte

Ces variables sont utilisées lors du calcul de la mise en page (correction typographique) par SPIP.

- `$debut_intertitre` fixe le code HTML inséré en ouverture des intertitres (par le raccourci `{{{`). En standard, sa valeur est :

$debut_intertitre = "\n<h3 class=\"spip\">\n";

- `$fin_intertitre` est le code HTML inséré en fermeture des intertitres (raccourci `}}}`). Sa valeur normale est :

$fin_intertitre = "</h3>\n";

- `$ouvre_ref` est le code d'ouverture des appels des notes de bas de page ; par défaut, c'est une espace insécable et un crochet ouvrant ;

- `$ferme_ref` est le code de fermeture des appels des notes de bas de page ; par défaut, c'est un crochet fermant.

- `$ouvre_note` est le code d'ouverture de la note de bas de page (telle qu'elle apparaît dans `#NOTES`) ; par défaut, un crochet ouvrant ;

- `$ferme_note` est le code de fermeture des notes de bas de page (un crochet fermant).

Des choix alternatifs pourront être par exemple d'utiliser des parenthèses ; ou, plus joliment, d'ouvrir avec le tag HTML `<sup>`, et de fermer avec `</sup>`.

- Le fichier `puce.gif` et la variable `$puce`. Lorsque vous commencez une nouvelle ligne par un tiret, SPIP le remplace par une petite « puce » graphique. Cette puce est constituée par le fichier `puce.gif` installé dans le dossier `dist/` à la racine du site ; vous pouvez modifier ce fichier selon vos besoins. Mais vous pouvez aussi décider de fixer vous-même le choix de la puce, au travers de la variable `$puce`. Par exemple pour indiquer un autre fichier graphique :

$puce = "<img src='mapuce.gif' alt='-' align='top' border='0'>";

ou par un élément HTML non graphique :

$puce = "<li>";

A partir de la version 1.9.2, et en cas de mutualisation, on peut aussi mettre dans config/mes_options.php, et non pas dans squelettes/mes_fonctions la variable suivante :

$GLOBALS['puce'];

- `$nombre_surligne` représente le nombre maximum de fois où un mot recherché sera surligné dans le texte. Par défaut, cette valeur est définie à **4**.

- `$ligne_horizontale` est le code de remplacement du raccourci typographique `----` (quatre tirets) ou `____` (quatre caractères de soulignement) qui permet d'insérer une ligne horizontale dans le texte. Par défaut, c'est le code `<hr class="spip" />`

.

- `$url_glossaire_externe` est l'adresse utilisé pour le raccourcis automatiques `[?SPIP]` vers un glossaire. Par défaut, le glossaire externe renvoie vers l'encyclopédie libre wikipedia.org.

Les variables pour les forums publics

Il existe des variables permettant de fixer le comportement des forums publics *avec des mots-clés*.

> *N.B. : Ces variables ne sont utilisées que lorsque vous créez des forums publics dans lesquels les visiteurs peuvent sélectionner des mots-clés ; leur utilisation est donc extrêmement spécifique (et pas évidente...).*

- `$afficher_texte` (« oui »/« non »). Par défaut, les forums publics sont conçus pour permettre aux visiteurs d'entrer le texte de leur message ; mais lorsque l'on propose le choix de mots-clés dans ces forums, on peut décider qu'aucun message n'est utile, seul la sélection des mots-clés importe. Dans ce cas, on pourra indiquer :

$afficher_texte = "non";

- `$afficher_groupe` permet d'indiquer les différents groupes de mots-clés que l'on souhaite proposer dans tel forum. En effet, tous les forums sur un site ne sont pas forcément identiques, et si, à certains endroits, on peut vouloir afficher une sélection de tous les groupes de mots-clés (ceux que l'ont a rendu accessibles aux visiteurs depuis l'espace privé), à d'autres endroits, on peut vouloir n'utiliser que certains groupes, voire aucune groupe (pas de sélection de mots-clés du tout).

La variable `$afficher_groupe` est un tableau (*array*), et se construit donc de la façon suivante :

$afficher_groupe[] = 3;

$afficher_groupe[] = 5;

impose l'affichage *uniquement* des groupes 3 et 5.

$afficher_groupe[] = 0;

interdit l'utiliser des mots-clés dans ces forums (puisqu'il n'existe pas de groupe de mots-clés numéroté 0).

Si l'on n'indique rien (on ne précise pas `$afficher_groupe`), tous les groupes de mots-clés indiqués, dans l'espace privé, comme « proposés aux visiteurs du site public » sont utilisés.

Interdire l'affichage des boutons d'admin

Toutes les pages de squelette se voient ajouter des « boutons d'amin » (notamment : « recalculer cette page ») lorsqu'on est administrateur et qu'on a activé le cookie de correspondance. Cette fonctionnalité, très pratique pour gérer le site, peut s'avérer malpratique dans certains cas ; par exemple pour des fichiers XML, que l'on ne veut en aucun cas voir perturbés par de tels ajouts.

[SPIP 1.7] La variable `flag_preserver` permet d'interdire ces affichages.

$flag_preserver = true;

On verra par exemple l'utilisation de cette variable dans `backend.php3`.

Le dossier des squelettes

Depuis SPIP 1.5 : Si l'on souhaite mettre les squelettes de son site dans un dossier particulier, par exemple pour faire des essais de différents jeux de squelettes trouvés sur Internet, ou parce qu'on aime que les choses soient bien rangées, etc., il est possible de fixer la variable `$dossier_squelettes`, dans le fichier `mes_fonctions.php` placé à la racine du site, ou **de préférence**, dans `ecrire/mes_options.php`.

```
<?php
   $GLOBALS['dossier_squelettes'] = 'design';
?>
```

À partir de ce moment-là, SPIP ira chercher en priorité les squelettes présents dans le dossier `design/` (que vous aurez créé à la racine du site). Si, de plus, vous utilisez `<INCLURE{fond=xxx}>`, SPIP ira chercher le fichier `xxx.html` d'abord dans `design/`, puis, s'il n'y figure pas, à la racine du site.

Les avantages de ce rangement peuvent sembler évidents (meilleure séparation du code de spip et de la structure du site, possibilité de changer tout un ensemble de squelettes d'un seul coup, etc.) ; l'inconvénient principal est qu'il sera plus difficile de visualiser les squelettes via un simple navigateur. En effet, même s'ils sont situés dans ce sous-dossier, l'HTML contenu dans ces fichiers de squelette doit être conçu comme s'ils étaient à la racine. Ainsi, les liens vers les images ou les CSS, notamment, risquent de « casser ».

Cette variable permet d'indiquer un ou plusieurs dossiers où SPIP cherchera les squelettes en priorité :

```
<?php
   $GLOBALS['dossier_squelettes'] = 'mes_skel1:mes_skel2';
?>
```

Ceci ouvre différentes possibilités, comme :
- essayer un nouveau jeu de squelettes sans écraser l'ancien,
- organiser vos squelettes en une arborescence de sous-répertoires : `mes_skel:mes_skel/rss:mes_skel/formulaires:mes_skel/mailing`,
- gérer dynamiquement plusieurs jeux de squelettes grâce à des *plug-ins* comme le switcher,
- etc.

Exemples

- Pour modifier des variables uniquement pour un certain type de squelettes (par exemples pour les pages de rubriques), il suffit de les définir dans le fichier d'appel de ces squelettes. Par exemple, pour les rubriques, on peut fixer des valeurs directement dans `rubrique.php3` :

```
<?php
   $fond = "rubrique";
   $delais = 2 * 3600;
   $espace_logos = 20;
```

```
include ("inc-public.php3");
?>
```

Ici, on a modifié la valeur de l'espace autour des logos.

- Pour modifier des valeurs de variables pour l'ensemble du site, on peut les définir dans le fichier `mes_fonctions.php`.

Attention, lorsqu'on définit des valeurs dans ce fichier, il faut impérativement utiliser la syntaxe `$GLOBALS['xxx']` pour chacune des variables à personnaliser. Par exemple, pour définir la valeur de `$debut_intertitre`, on utilise la syntaxe `$GLOBALS['debut_intertitre']`.

L'utilisation de cette syntaxe est imposée par des impératifs de sécurité des sites.

```
<?php
   $GLOBALS['debut_intertitre'] = "<h3 class='mon_style_h3'>";
   $GLOBALS['fin_intertitre'] = "</h3>";

   $GLOBALS['ouvre_ref'] = ' (';
   $GLOBALS['ferme_ref'] = ')';
   $GLOBALS['ouvre_note'] = '(';
   $GLOBALS['ferme_note'] = ') ';

   $GLOBALS['espace_logos'] = 0;
?>
```

Il existe d'autres variables permettant de modifier le comportement de SPIP au niveau technique. Ces variables sont décrites sur le site de documentation technique.

La « popularité » des articles

Juillet 2002 — maj : Décembre 2007

La notion de *popularité*, exposée ci-dessous, apparaît dans SPIP 1.4.

Comment décompter des visites

Des centaines de méthodes statistiques existent pour décompter des visites sur un site donné. La plupart donnent des courbes horaires, ou par jour, qui permettent de savoir si son site « monte » ou « descend », et de vérifier qu'il y a plus de gens sur le net en fin d'après-midi et dans la semaine, que le week-end ou la nuit...

Notre objectif est un peu différent : il s'agit d'attribuer à chaque article une valeur de « popularité » reflétant assez rapidement une tendance, et permettant de comparer l'activité de différents articles, soit de manière globale sur tout le site (hit-parade), soit à l'intérieur d'une rubrique, soit parmi les articles d'un même auteur, etc.

La méthode retenue est la suivante (rassurez-vous, vous pouvez sauter cette explication si vous n'êtes pas à l'aise en maths) :
- chaque visite sur un article ajoute un certain nombre de points à cet article ; 1 point si c'est un article que l'on consulte depuis le site lui-même en suivant un lien, et 2 points si c'est une « entrée directe » depuis un site extérieur (moteur de recherche, lien hypertexte, syndication...)
- toutes les 10 minutes, le score obtenu est multiplié par un petit facteur d'escompte, qui fait qu'un point attribué par une visite à 10h12 le mercredi ne vaut plus, le lendemain à la même heure, qu'un demi-point, et, le vendredi à 10h12, un quart de point... ;
- le tout est calculé de manière à ce que, dans l'hypothèse où l'article reçoit toujours le même nombre *x* de visites par unité de temps, son score se stabilise sur cette valeur *x*. Autrement dit, si la fréquentation de l'article est stationnaire, sa popularité finira par refléter exactement son nombre de visites par jour (modulo le score 2 donné pour les entrées directes) ;
- cette popularité s'exprime de deux manières : l'une, la popularité_absolue, exprime le score en question (évaluation de la fréquentation quotidienne de l'article) ; l'autre, la popularité_relative, un pourcentage relatif à l'article du site ayant la plus forte popularité (popularité_max) ;
- enfin, la somme de toutes ces valeurs (absolues) sur le site donne la popularité_site, qui permet de comparer la fréquentation de deux sites sous spip...

Boucles et balises

Des balises permettent de récupérer et d'afficher ces valeurs dans vos squelettes. La boucle ci-dessous résume l'ensemble de ces balises :

```
<BOUCLE_pop(ARTICLES){id_article}{popularite>0}>
<h5>Popularité</h5>
Cet article a une popularité absolue égale à #POPULARITE_ABSOLUE, soit
#POPULARITE % de #POPULARITE_MAX. Au total, ce site fait environ
#POPULARITE_SITE visites par jour.
</BOUCLE_pop>
```

La balise la plus utile est `#POPULARITE` puisqu'elle donne un pourcentage représentant la popularité de l'article relativement à l'article le plus populaire du site. Cela permet ainsi de réaliser facilement des classements compréhensibles par tous (avec des valeurs allant de 0 à 100). Les autres balises donnent des valeurs absolues, plus difficiles à interpréter par les visiteurs du site.

> *Note :* bien que les données soient représentées, dans la base de spip, sous forme de nombres réels, le rendu de toutes ces balises est toujours donné sous la forme d'un nombre entier, ce qui donnera, sur des sites *très* peu fréquentés (sites de tests, notamment), des choses amusantes du genre :
>
> *« Cet article a une popularité absolue égale à 1, soit 17 % de 2. Au total, ce site fait environ 5 visites par jour. »*

Enfin, un critère de tri peut se révéler utile : `{par popularite}`, que l'on utilisera par exemple de la manière suivante pour afficher la liste des 10 articles les plus populaires de la rubrique courante :

```
<BOUCLE_hitparade(ARTICLES){id_rubrique}{par popularite}{inverse}{0,10}>
<li>#TITRE (popularité : #POPULARITE %)</li>
</BOUCLE_hitparade>
```

(On enlèvera `{id_rubrique}` pour afficher un hit-parade du site entier.)

Le support LDAP

Décembre 2001 — maj : Décembre 2007

Attention, cet article est vraiment destiné à des utilisateurs avancés, qui maîtrisent l'usage de LDAP et souhaitent appuyer SPIP sur un annuaire LDAP existant.

LDAP (Lightweight Directory Access Protocol) est un protocole permettant d'interroger un annuaire contenant des informations d'utilisateurs (nom, login, authentification...). Depuis la version [SPIP 1.5] il est possible de vérifier si un rédacteur est dans la base LDAP avant de lui donner accès à l'espace privé.

A l'installation, SPIP détecte si PHP a été compilé avec le support LDAP. Si oui, à la cinquième étape (« créer un accès »), un bouton permet d'ajouter un annuaire LDAP à la configuration SPIP. La configuration qui suit est relativement simple, elle essaie de deviner les paramètres au maximum. Notamment, elle permet de choisir le statut par défaut des auteurs venant de l'annuaire : ceux-ci peuvent être rédacteurs (conseillé), administrateurs ou simples visiteurs.

> *Note : par défaut, l'extension LDAP de PHP n'est généralement pas activée, donc SPIP n'affichera pas le formulaire correspondant lors de l'installation. Pensez à activer l'extension LDAP dans votre installation de PHP si vous voulez utiliser LDAP avec SPIP.*

Si SPIP est déjà installé et que vous voulez configurer l'annuaire LDAP, il faudra reprendre l'installation en effaçant le fichier `ecrire/inc_connect.php3`.

Une fois la configuration correctement effectuée, tous les utilisateurs de l'annuaire LDAP seront identifiés en tapant leur login (ou nom) dans l'annuaire LDAP, puis leur mot de passe. Notez que cela n'empêche pas de créer directement des auteurs dans SPIP ; ces auteurs ne seront pas recopiés dans l'annuaire mais gérés directement par SPIP. D'autre part les informations personnelles (biographie, clé PGP...) des auteurs authentifiés depuis LDAP ne seront pas non plus recopiées dans l'annuaire. Ainsi SPIP n'a besoin que d'un accès *en lecture seule* à l'annuaire LDAP.

Important : créez toujours un premier administrateur « normal » (non LDAP) lors de l'installation de SPIP. C'est préférable pour éviter d'être bloqué en cas de panne du serveur LDAP.

Pour en savoir plus

Les infos de connexion au serveur LDAP sont écrites dans `inc_connect.php3`. Corollaire : il faut supprimer ce fichier et relancer l'installation pour activer LDAP sur un site SPIP existant.

Dans la table spip_auteurs, est ajouté un champ "source" qui indique d'où viennent les infos sur l'auteur. Par défaut, c'est "spip", mais ça peut aussi prendre la valeur "ldap". Ca permet de savoir notamment quels champs ne doivent pas être changés : en particulier, on ne doit pas autoriser la modification du login, car sinon il y a une perte de synchronisation entre SPIP et LDAP.

A l'authentification, les deux méthodes sont testées à la suite : SPIP puis LDAP. En fait un auteur LDAP ne pourra pas être authentifié par la méthode SPIP (méthode standard avec challenge md5) car le pass est laissé vide dans la table spip_auteurs. Un auteur SPIP, quant à lui, sera authentifié directement depuis la table spip_auteurs. D'autre part, si le login entré ne vient pas de SPIP, le mot de passe est transmis en clair.

Quand un auteur LDAP se connecte pour la première fois, son entrée est ajoutée dans la table spip_auteurs. Les champs remplis sont : nom, login et email qui viennent de LDAP (champs 'cn', 'uid' et 'mail' respectivement) et le statut dont la valeur par défaut a été définie à l'installation (rédacteur, admin ou visiteur). Important : on peut modifier le statut par la suite, afin de choisir ses admins à la main par exemple.

Une fois un auteur connecté, il est authentifié par la voie classique, c'est-à-dire simplement avec le cookie de session. Ainsi on ne se connecte à LDAP que lors du login (spip_cookie.php3). De même, les infos prises en compte dans l'affichage et les boucles sont celles de spip_auteurs, pas celles de l'annuaire.

Pour les auteurs SPIP, rien ne change. On peut les créer et les modifier comme à l'habitude.

Rapidité du site public

Juin 2001 — maj : Décembre 2007

Contrairement à la plupart des systèmes de publication gratuits, SPIP intègre un système de cache permettant d'accélérer l'affichage du site public. Quelques pistes pour comprendre ce qui influe sur la rapidité de votre site...

Optimiser un site

Si vous vous inquiétez pour la rapidité de votre site, il est bon de vous intéresser aux pistes suivantes :

- Votre hébergement Web offre-t-il des performances de bonne qualité ? Evidemment, c'est subjectif. L'expression « mauvaise qualité » recouvre à coup sûr la plupart des hébergeurs gratuits (notamment Free). « Bonne qualité » inclut forcément une machine dédiée (i.e. qui ne sert qu'à votre site) de fabrication récente, mais aussi des hébergeurs commerciaux pas trop au rabais. Entre les deux, ça devient très subjectif, en fonction de vos exigences, de la taille de votre site....

- Si la qualité de votre hébergement laisse à désirer, vous aurez intérêt à ne pas créer de squelettes trop complexes, i.e. qui demandent à SPIP d'afficher trop d'informations différentes. Cela vaut pour tout type d'informations : tout ce qui, dans les squelettes, est susceptible d'être transformé par SPIP en données affichables. Notez, en particulier, que les squelettes fournis par défaut démontrent au maximum les possibilités de SPIP, et par conséquent génèrent des pages assez lourdes.

- N'oubliez pas non plus de régler les délais d'expiration des différents types de pages. Ainsi, si votre site contient un grand nombre d'articles en archives, vous avez peut-être intérêt à augmenter la durée d'expiration des articles, sinon les articles consultés peu souvent ne bénéficieraient pas du système de cache.

L'influence du cache

La présence du cache change quelque peu la donne en matière de rapidité. Ce n'est pas tant le nombre de visites de votre site qui sera le point critique, que la capacité de votre serveur à recalculer les pages dans le temps imparti au script PHP (en effet, sur la plupart des serveurs, une limite de durée d'exécution par appel de script est fixée afin d'éviter les abus et les erreurs de programmation). Par contre, si la page demandée est dans le cache et n'a pas expiré, la réponse du serveur devrait être quasi-instantanée (dans le cas contraire, votre serveur est vraiment très chargé).

La qualité des performances devient ainsi objectivement mesurable si, lors du recalcul d'une page

du site, on obtient un « *timeout* », c'est-à-dire que le serveur a dépassé le temps maximal d'exécution d'un script PHP. Alors il faut soit changer d'hébergement, soit se résoudre à afficher des pages plus simples : pour cela, modifier les squelettes pour afficher moins d'informations sur une même page.

Sur une machine dédiée

Si vous utilisez votre propre machine, il faut vous assurer qu'elle pourra tenir la charge. N'importe quelle machine pas trop vieille (moins de trois ans environ) devrait en être capable.

Par contre, l'utilisation de SPIP, par rapport à d'autres systèmes de publication, permet de mutualiser les ressources techniques entre plusieurs sites. En effet, tant que le cache est utilisé, la machine est peu sollicitée, donc plusieurs sites peuvent cohabiter sans problème (sauf s'il y a vraiment un très grand nombre de visites). Le problème est donc surtout de prévenir qu'il y ait trop de passagers à bord, c'est-à-dire qu'un trop grand nombre de « services » hébergés (sites Web, boîtes à e-mail...) mette en péril la qualité du service.

Utiliser des URLs personnalisées

Mai 2001 — maj : Décembre 2007

Après l'installation, les pages générées par SPIP utilisent des adresses relatives ressemblant à `spip.php?article765`, donnant des URLs du type `http://www.spip.net/spip.php?article765`.

Il y a possibilité d'avoir des adresses plus à votre goût — par exemple `article123.html` ou `Titre-de-l-article.html`, et SPIP vous aide en partie dans cette tâche.

Cette fonctionnalité fait appel à la distinction entre deux types d'URLs :

- *l'URL apparente* d'une page, c'est-à-dire telle qu'elle est tapée et/ou affichée dans la barre d'adresse du navigateur. Par exemple `http://www.spip.net/fr_article765.html`. Ce sont ces URLs qu'on cherche à rendre plus « jolies » ou plus « signifiantes » ;

- *l'URL réelle* de la page, c'est-à-dire l'URL qui est « vue » par SPIP lorsque la page est calculée sur le serveur. Par exemple `http://www.spip.net/spip.php?article765` ; en général, cette URL peut aussi être tapée directement dans le navigateur (vous pouvez vérifier).

Choisir le type d'URLs apparentes

Dans le fichier `ecrire/mes_options.php` [1] (à créer le cas échéant), vous pouvez déclarer une variable PHP contenant le type d'URLs à utiliser. En l'absence de ce réglage, SPIP utilisera :

```
$type_urls = "page";
```

La variable `$type_urls` détermine le nom du fichier PHP qui est appelé pour gérer les URLs. Avec la déclaration par défaut ci-dessus, c'est le premier `urls/page.php` trouvé dans SPIP_PATH.

Vous remarquerez que SPIP propose aussi les fichiers `urls/standard.php`, `urls/html.php`, `urls/propres.php`, `urls/propres-qs.php` et `urls/propres2.php` dans le répertoire `ecrire/`

- Le fichier `urls/html.php` permet de traiter des adresses du type (« article123.html »). Vous pouvez décider d'utiliser les « URLs "html" » en mettant dans `ecrire/mes_options.php` la ligne :

`$type_urls = "html";`

- Le fichier `urls/propres.php` permet de traiter des adresses du type (« Titre-de-l-article »). Il faut alors ajouter :
`$type_urls = "propres";`

- Le fichier `urls/propres2.php` est une variation du précédent, qui donne des adresses du type (« Titre-de-l-article.html »). Il faut alors ajouter :
`$type_urls = "propres2";`

- Le fichier `urls/propres-qs.php` est une variation du précédent, qui donne des adresses du type (« `./?Titre-de-l-article` »). Il faut alors ajouter :
`$type_urls = "propres-qs";`. Ce dernier est à utiliser si votre hébergeur ne vous permet pas d'utiliser le module de réecriture d'urls d'apache (cf. mod_rewrite)

- Enfin, le fichier `urls/standard.php` permet aux nostalgiques des versions précédentes de spip de donner des adresses du type (« `article.php3?id_article=765` »). Il faut alors ajouter :
`$type_urls = "standard";`

> Si vous voulez plutôt utiliser vos propres adresses (ce pour quoi vous devez savoir programmer en PHP), il est fortement conseillé de partir d'un des fichiers existants et de le recopier sous le nom que vous aurez choisi : `urls/XXX.php`. Il est par exemple très aisé de modifier la fonction `_generer_url_propre()` dans `urls/propres.php` pour obtenir des variations très intéressantes ; si vous faites cela, merci de partager vos modifications sur le site SPIP Contrib'.

Programmer la traduction des adresses apparentes en adresses réelles

Pour que l'adresse `article123.html` appelle bien en réalité le fichier PHP spip.php avec comme paramètre `id_article=123`, il va falloir configurer le serveur Web qui héberge votre site, soit dans un fichier *.htaccess* (ça ne marche pas toujours), soit dans le fichier de configuration centrale du serveur si vous y avez accès. Cela utilise, sous le serveur Apache (le plus utilisé), ce qu'on appelle des *Rewrite Rules* : des règles de réécriture d'adresses Web.

Savoir écrire ces règles n'est pas simple pour les non-programmeurs, et nous ne pouvons pas vous donner de solutions infaillibles car cela dépend de votre configuration : cette partie est entièrement entre vos mains (ou celles de votre hébergeur).

Néanmoins, depuis SPIP 1.8 est fourni un fichier `htaccess.txt` à titre d'exemple, qui fonctionne sur la plupart des hébergeurs avec les types d'URLs cités précédemment (« standard », « html », « propres » et « propres2 »). Pour l'activer il faut le recopier à la racine du site sous le nom `.htaccess`. Il est fortement conseillé de l'ouvrir au préalable pour vérifier quelques aspects de configuration.

Vous devrez ensuite tester la validité de ces adresses, en appelant la page « Voir en ligne » sur un article, un auteur, une brève, une rubrique, etc.

Générer les URLs apparentes dans les pages SPIP

Afin d'afficher partout les URLs du type choisi, utilisez dans vos squelettes les balises `#URL_ARTICLE`, `#URL_RUBRIQUE`, `#URL_BREVE`, etc.

Transition d'un type d'URLs à l'autre

Depuis SPIP 1.8, tout est prévu pour que la transition d'un type d'adresses à l'autre se fasse en douceur : installez le fichier `htaccess.txt`, et vous pouvez ensuite librement basculer des adresses « standard » aux adresses « propres2 », « propres » ou « html », et vice-versa, sans jamais provoquer d'erreur 404 pour les visiteurs (ou les moteurs de recherche) qui auraient mémorisé les anciennes adresses.

Dernier détail pour faciliter la transition, si vous choisissez les URLs propres ou propres2, les visites des pages portant les anciennes adresses (standard ou html) sont redirigées automatiquement vers les nouvelles adresses.

Notes

[1] Remarque : les versions précédentes de SPIP incluaient le fichier `inc-urls.php3` à la racine du site s'il était présent ; cette méthode est encore valable mais est considérée comme obsolète...

Modifier l’habillage graphique

Pour réaliser l’habillage de votre site, il est recommandé d’utiliser les feuilles de style CSS. Pas de panique, ces quelques pages permettront aux débutants de raccrocher les wagons...

Introduction aux feuilles de style

Avril 2004 — maj : 22 juillet

Les feuilles de style permettent de centraliser et de gérer de manière beaucoup plus aisée les indications graphiques que l’on insérait traditionnellement dans le HTML. Elles s’écrivent dans un langage spécifique : le CSS.

Comme vous le savez déjà, SPIP traite séparément le contenu de sa mise en page et son habillage graphique : les squelettes trient et affichent les contenus souhaités en pages HTML, dont l’habillage graphique est réalisé par des feuilles de style CSS. Passez à la vitesse supérieure pour habiller vos squelettes : utilisez les feuilles de style avec SPIP !

Pourquoi les feuilles de style ?

Si vous réalisez des pages Web de manière « traditionnelle », les indications graphiques sont insérées directement dans le code HTML de votre page. Ainsi à chaque fois que vous voulez mettre un texte en rouge, vous écrivez `<font color="red">`. Pour afficher un tableau avec des bordures épaisses, vous écrivez `<table border="2">`.

Avec cette méthode et un site statique (où chaque article a une page HTML spécifique), changer la maquette de tout un site est un cauchemar : il faut rechercher dans tous les fichiers HTML, les portions de code à modifier, et effectuer ces modifications une par une (par exemple remplacer `<font color="red">` par `<b>` si l’on décide que les éléments anciennement affichés en rouge seront désormais en gras).

Comme vous le savez déjà, SPIP améliore beaucoup la situation : vous n’avez plus à modifier des centaines de fichiers HTML, mais juste quelques squelettes ; et votre mise en page est remise à jour automatiquement sur l’ensemble du site.
Cependant le problème n’est pas entièrement résolu. Par exemple, mettons que vous ayez décidé d’employer un certain bleu pastel sur beaucoup d’éléments du site, afin de donner une identité graphique à votre site : les liens, les encarts, certains éléments de navigation... sont affichés en bleu pastel. Le jour où vous voudrez remplacer ce bleu pastel par un vert pâle, vous devrez modifier tous les endroits du squelette où ce bleu apparaissait pour le remplacer par le vert pâle. Cela peut être décourageant : il n’est pas aisé dans ces conditions de changer rapidement le rendu des pages, ne serait-ce que pour faire des essais.

La solution réside dans l’utilisation des « **feuilles de style externes** ». Une feuille de style est un fichier où vous définissez un ensemble de propriétés graphiques, et les endroits où elles s’appliquent. On note deux avantages capitaux des feuilles de style :

- **la feuille de style est un fichier unique et centralisé**, que vous pouvez appliquer à autant de fichiers HTML (et de squelettes SPIP) que vous le désirez ;
- **les propriétés graphiques sont définies une seule fois dans la feuille de style**, quel que soit le nombre d’endroits où ces propriétés sont appliquées dans le HTML.

Concrètement

Pour être appliquée à un fichier HTML (qui peut être un squelette SPIP), la feuille de style doit être déclarée dans l'entête de votre page (entre les balises `<head>`), de la façon suivante :

```
<link rel="stylesheet" type="text/css" href="mes_styles.css" />
```

- Ici le fichier `mes_styles.css` contient les propriétés graphiques que vous voulez appliquer à la page HTML (dans la suite de cette rubrique, on supposera que `mes_styles.css` est le nom que vous avez choisi pour ce fichier).

- Ce fichier porte l'extension « `.css` ». En effet, **CSS** est le nom du langage utilisé pour les feuilles de style, de la même manière que HTML est le nom du langage utilisé pour la réalisation de pages web. **Notez bien que le CSS n'est pas propre à SPIP, il s'agit d'un standard du Web** [1].

> Note : une feuille de style peut s'appliquer aussi bien à une page HTML classique (« statique ») qu'à un squelette SPIP (« dynamique »). Cela veut dire que toute astuce CSS valable dans du HTML classique sera aussi utilisable dans un squelette de votre site.

Si vous avez bien lu les paragraphes précédents, vous serez peut-être dubitatifs : oui, il faut apprendre un nouveau langage pour utiliser les feuilles de styles (SPIP n'y est pour rien !). Les CSS n'utilisent pas, en effet, la syntaxe du HTML. Cependant ce langage est très simple, et il suffit de quelques exemples pour mettre le pied à l'étrier. De très nombreuses documentations existent sur ce sujet par ailleurs ; consultez les ressources proposées à cette page : « Ressources CSS pour en savoir plus ».

Notes

[1] La première version de CSS a vu le jour en 1996. C'est un langage de feuille de style, approuvé comme *Recommandation du W3C*.

Les feuilles de style de SPIP

Septembre 2001 — maj : 27 juillet

Le code généré par SPIP est doté de certains styles qu'il convient de définir.

Dans l'article précédent, nous avons vu quels étaient les avantages des feuilles de style CSS. Voyons maintenant quel usage spécifique SPIP fait des feuilles de style.

Des styles définis par SPIP

Dans SPIP, certains styles jouent un rôle important : ils servent à modifier les propriétés graphiques des éléments qui ne sont *pas* définis dans *votre* HTML (celui de votre squelette), mais dans le code *généré par SPIP*. En effet, **SPIP associe de lui-même plusieurs styles au code qu'il génère**.

Ainsi, lorsque l'on utilise les raccourcis SPIP dans les articles (permettant par exemple de mettre en gras, en italique, créer des liens hypertextes, des intertitres, des tableaux, etc.), SPIP produit les balises HTML nécessaires à ces effets, chacune de ces balises étant alors dotée d'un sélecteur CSS.

> Par exemple, le raccourci suivant :
>
> ```
> Ceci est un [lien->http://www.uzine.net]
> ```

est ainsi transformé en code HTML :

```
Ceci est un <a href="http://www.uzine.net"
class="spip_out">lien</a>
```

Quel est l'intérêt ? Ces balises portent un nom spécifique dans l'attribut `class` : ce nom définit à quelle « classe » ils appartiennent, c'est-à-dire un ensemble d'éléments HTML qui hériteront des mêmes propriétés graphiques définies dans la feuille de style.
Dans notre exemple, le code HTML est complété par le sélecteur CSS intitulé « `spip_out` ». Le webmestre peut donc pousser la personnalisation graphique des liens sortants en modifiant la définition stylistique de « `spip_out` » (couleur différente, fond coloré, police utilisée, etc.).

L'apparence de la plupart des raccourcis SPIP peut ainsi être paramétrée dans les feuilles de style. Cela vaut aussi pour les formulaires automatiques (répondre à un forum, signer une pétition...) et d'autres encore. Certains de ces styles sont très utiles, voire indispensables, d'autres seront réservés aux webmestres qui souhaitent obtenir des effets exotiques.

Où se trouvent ces définitions de style ?

Les propriétés graphiques appliquées aux pages HTML sont regroupées dans les fichiers `.css` qui accompagnent les squelettes. Depuis SPIP 1.8, les squelettes et leurs feuilles de style sont regroupés dans le répertoire `dist/`. Dans les versions antérieures, ils se trouvent à la racine.

- Les définitions de style propres à SPIP se trouvent dans la « feuille de style externe » nommée `spip_style.css`. Celle-ci regroupe les définitions des styles associées au code généré par SPIP (passées en revue dans cette rubrique).

- Une autre feuille de style, `spip_admin.css` (disponible depuis SPIP 1.8), permet de contrôler l'apparence des boutons d'administration (« recalculer cette page », etc.).

Vous pouvez les modifier (c'est même conseillé : « Mettez-y votre style ! »), mais notez bien que vous ne pouvez pas les renommer. **Ces styles sont indispensables et doivent nécessairement être définis pour un bon affichage de vos squelettes**.

> *Historique :* Dans les versions antérieures à SPIP 1.9, certaines définitions de styles (concernant les images ou les formulaires) ne sont pas disponibles dans les feuilles de style externes et sont donc difficilement personnalisables.

Mettez-y votre style !

Avril 2004 — maj : 22 juillet

Vous pouvez modifier les styles fournis avec SPIP et ajouter les votres, en créant votre propre feuille de style. Voici comment.

Si vous connaissez le langage CSS (sinon lisez d'abord cet article : « Introduction aux feuilles de style »), vous pouvez très facilement modifier l'apparence de votre site, sans même avoir besoin de connaître le langage des boucles et balises de SPIP.

Créez votre feuille de style

Lors de l'installation de SPIP, les squelettes sont distribués avec plusieurs « feuilles de style

externes » qui regroupent les indications produisant l'habillage graphique du site. Vous pouvez modifier ces fichiers et ajouter vos propres définitions de style, mais il est préférable de le faire dans votre propre fichier CSS afin de pas voir vos ajouts « écrasés » lorsque vous installerez une nouvelle version de SPIP.

> **Important :** ne travaillez jamais directement dans les fichiers fournis par défaut, sinon vous risqueriez de perdre toutes vos modifications à chaque mise à jour de SPIP ! Pour éviter cela, faites une copie des fichiers que vous souhaitez modifier.

1. Créez un fichier `mes_styles.css` (ou tout autre nom que vous avez décidé de lui donner) et rangez-le dans votre dossier « squelettes ». Vous copierez dans ce fichier les définitions de styles que vous souhaitez utiliser et modifier ; mais pour la suite de ce tutorial, nous allons considérer que vous partez d'une feuille vierge.

2. Appelez cette feuille de style dans l'entête de votre squelette, c'est-à-dire entre les balises `<head>` du fichier HTML (aux côtés du `title` et autres `meta`). De la façon suivante :

<link rel="stylesheet" type="text/css" href="#CHEMIN{mes_styles.css}" />

Bien souvent une seule feuille de style suffit pour tout l'habillage graphique d'un site, mais vous pouvez déclarer de cette façon autant de feuilles de style que nécessaire.

> *Historique :* À partir de SPIP 1.8.2, on utilise :
>
> <link rel="stylesheet" type="text/css" href="#DOSSIER_SQUELETTE/mes_styles.css" />
>
> Depuis SPIP 1.9, la balise `#CHEMIN` remplace et améliore `#DOSSIER_SQUELETTE`.

Respectez la « cascade »

Il est important de garder à l'esprit le fonctionnement « **en cascade** » du CSS (*Cascading Style Sheets* signifie littéralement « feuilles de style en cascade ») : lorsque plusieurs définitions de style concernent un même élément, **est appliqué en priorité le style le plus proche de l'élément**. L'ordre dans lequel les styles sont « lus » a donc une importance.

- ***Feuilles de style externes***

Si vous utilisez plusieurs feuilles de style, notez que l'ordre dans lequel celles-ci sont appelées dans l'entête de la page a une importance. Si vous appelez *d'abord* `mes_styles.css` et *ensuite* `spip_style.css` : ce sont les styles de cette dernière, plus *proches*, qui s'appliqueront prioritairement aux votres. Pensez donc à faire l'inverse :

```
<link rel="stylesheet" type="text/css"
href="#CHEMIN{spip_style.css}">
<link rel="stylesheet" type="text/css"
href="#CHEMIN{mes_styles.css}">
```

- ***Styles définis dans le code HTML***

Si vous ne souhaitez pas toucher aux fichiers CSS, vous pouvez continuer à insérer, par endroits, des indications graphiques directement dans le code HTML de vos squelettes : en définissant quelques styles dans le `head`, et/ou en plaçant des indications de style directement dans les balises HTML de la page.

Les styles placés directement dans les balises, étant au plus *proches* des éléments concernés, seront

prioritaires sur ceux définis dans le `head`, eux-mêmes prioritaires sur ceux des feuilles de style *externes*.

Des styles qui ont de la « class »

Comment fait-on alors pour changer, par exemple, l'apparence de tous les intertitres SPIP ? C'est très simple. Ouvrez votre fichier `mes_styles.css` dans un éditeur de texte et ajoutez-y la ligne suivante :

```
h3.spip { color: red; font-size: 18px; }
```

Rechargez la page : tous les intertitres apparaissent comme par magie en rouge ; remarquez de plus que les autres `h3` de votre page, s'il y en a, ne sont *pas* affichés en rouge.

Expliquons brièvement la syntaxe de cette règle de mise en page :

- `h3.spip` juste avant les accolades signifie que la règle ne s'applique qu'aux `<h3>`

dotés d'un attribut `class` égal à « `spip` ». Notez bien : ni les `<h3>`

n'ayant pas cet attribut, ni les balises ayant cet attribut sans être des `<h3>`

, ne seront concernés.

Si vous ajoutez vos propres styles, sachez que la valeur donnée à l'attribut `class` est totalement arbitraire. La seule chose qui compte, est que vous utilisiez bien le même nom dans le code HTML (`class="toto"`) et dans votre feuille de style CSS (`.toto { ... }`).
Rappelons toutefois que vous ne pouvez pas renommer les `class` associées au code généré par SPIP (dont les définitions de style sont regroupées dans `spip_style.css`).

- Les accolades contiennent la liste des propriétés graphiques associées au style ainsi défini. Ici nous voyons que la couleur est réglée à rouge et que la police de caractères doit être affichée avec une taille de 18 pixels.

Notons que toutes les propriétés non définies dans cette liste garderont leur valeur habituelle pour la balise considérée ; dans le cas présent, le `h3` générera toujours un texte en gras, car rien dans cette définition de style ne dit le contraire.

La gestion du cache

Le fait que les styles soient définis dans un fichier séparé a une conséquence importante. En effet, ce fichier, au contraire de vos squelettes n'est pas géré par SPIP (il n'en a pas besoin !). Cela signifie que **si vous modifiez une feuille de style, vous n'avez pas besoin de vider le cache de SPIP : il suffit de recharger la page dans votre navigateur.** Cela rend le réglage de la mise en page encore plus aisé.

Rappelons tout de même que votre feuille de style doit être déclarée dans vos fichiers HTML, et que ceux-ci doivent être recalculés une première fois pour que cette déclaration soit prise en compte

Styles des raccourcis typographiques de SPIP

Juin 2006 — maj : Janvier 2007

Voici les styles CSS associés aux raccourcis typographiques les plus courants et les plus simples de SPIP.

Pour faciliter la mise en forme des textes, SPIP propose un certain nombre de « raccourcis SPIP » qui font l'objet d'un traitement automatisé : ils sont remplacés par le code HTML correspondant, doté d'un style (généralement la classe `spip`). Pour affiner encore davantage la mise en forme des textes, vous pouvez modifier les styles CSS associés à ces raccourcis typographiques.

Les paragraphes de SPIP

Pour créer des paragraphes dans SPIP, il suffit de laisser une ligne vide. Les paragraphes ainsi générés par SPIP sont dotés de la classe `spip` :

```
<p class="spip">Voici le texte de mon paragraphe</p>
```

Pour ajuster, par exemple, l'espace entre chaque paragraphe, modifiez la définition de style : `p.spip`.

> *Remarque :* ceci n'est valable que pour les textes en plusieurs paragraphes ; les textes écrits d'un seul bloc, en un seul paragraphe, n'étant pas dotés par SPIP de balises `<p>`. Il est donc préférable de ne pas définir ce style.

Gras et italique

Le gras et l'italique sont générés par les raccourcis suivants :

```
Du texte {en italique}, du texte {{en gras}}
```

Leur apparence est contrôlée par les définitions de style respectives : `i.spip` et `strong.spip`. Ces styles sont peu utiles, et ne sont d'ailleurs pas définis par défaut.

> *Historique :* dans les versions antèrieures à SPIP 1.8, le texte en gras est entouré de la balise `<b>` et donc stylé par la définition `b.spip`.

Les intertitres de SPIP

Les intertitres sont créés par le raccourci suivant :

```
{{{Un intertitre}}}
```

Leur apparence est contrôlée par la définition de style `h3.spip`. Ce style est sans doute l'un des plus importants, car il permet de définir la taille, la police et le positionnement des intertitres dans les articles : vous serez certainement amenés à le modifier en fonction de vos choix graphiques et typographiques.

Notez en particulier les attributs `margin` et `padding` qui permettent d'agir sur l'espacement de l'intertitre avec les paragraphes précédent et suivant. Sans ce réglage, l'intertitre serait soit trop « collé » au reste du texte, soit trop espacé.

Le trait de séparation horizontal de SPIP

Le trait de séparation horizontal est généré par ce raccourci SPIP :

```
----
```

Son apparence est contrôlée par la définition de style : `hr.spip`.

> *Remarque :* La ligne horizontale (`<hr />`) est un élément délicat à styler, car les styles CSS qui lui sont appliqués sont interprétés differemment selon les navigateurs.

Les notes de bas de page

Les notes de bas de page, sont créées par le raccourci :

```
Le texte[[Ceci est une note de bas de page.]].
```

Leur apparence est contrôlée par plusieurs définitions de style :
- `a.spip_note` permet de différencier visuellement les appels des notes de bas de page des autres liens (c'est-à-dire leurs numéros, tant dans le corps du texte qu'en bas de page) ;
- `p.spip_note` contrôle l'affichage des notes elles-mêmes ; bien souvent inutile (on préférera en effet définir ce style plus simplement, à partir du style de la balise parente.).

Styles des liens hypertextes

Juin 2006 — maj : 22 juillet

Comment définir l'apparence de ses liens ?

Sur le web, les liens hypertextes sont traditionnellement caractérisés par la couleur bleue et le soulignement. Mais vous avez certainement remarqué que cette présentation varie d'un site à l'autre. Vous pouvez très facilement personnaliser l'apparence de vos liens avec les styles CSS. Exemple très classique :

```
a { color: green; text-decoration: none; }
a:hover { color: red; text-decoration: underline; }
```

- « `a` » concerne tous les liens affichés sur votre page web, c'est-à-dire toutes les balises `<a>`, sans exception, qu'elles aient ou non un attribut class. Dans l'exemple ci-dessus, les liens s'afficheront en vert, sans « décoration » : ils ne sont pas soulignés.

- Plusieurs états sont disponibles pour les liens avec les « pseudo classes » (`:hover`, `:visited`, etc.). Ainsi « `a:hover` » concerne les liens « survolés ». Dans l'exemple ci-dessus, les liens deviennent rouges et soulignés lorsqu'ils sont survolés par le pointeur de la souris.

Notez que la recommandation CSS2 précise que, pour être pleinement prise en compte, la règle `a:hover` doit être placée après les autres.

Les liens hypertextes de SPIP

SPIP 1.2, SPIP 1.2.1 va plus loin, en vous permettant de différencier graphiquement les différents types de liens, notamment les liens internes au site et les liens vers d'autres sites. Cela se fait avec quelques définitions de style spécifiques, que nous vous recommandons de personnaliser à votre goût :

- `a.spip_in` concerne les liens à l'intérieur de votre propre site. Par exemple :
Le raccourci `[->article1177]` génére un lien interne, vers l'article 1177 de votre site, ainsi : Les feuilles de style de SPIP

- `a.spip_out` concerne les liens vers l'extérieur de votre site. Par exemple :
Le raccourci `[uZine->http://www.uzine.net]` affiche le lien externe suivant : uZine

- `a.spip_url` traite les adresses URL transformées en lien hypertexte. Par exemple :
Le raccourci `[->http://www.uzine.net]` affiche directement l'URL, avec un lien hypertexte vers cette adresse, ainsi : http://www.uzine.net

- `a.spip_glossaire` concerne les liens vers le glossaire externe (en l'occurrence

l'encyclopédie en ligne Wikipédia). Par exemple :
Le raccourci `[?SPIP]` génére le lien suivant : SPIP

Pour ne pas s'emmêler les liens

Notez bien que, pour un lien donné, plusieurs définitions de style interviennent. Par exemple, si vous avez précisé :

```
a { color: green; text-decoration: underline; }
a.spip_in { color: orange; }
```

Les liens internes (dotés de la classe `spip_in`) sont de couleur orange, mais héritent également du soulignement appliqué à `<a>`. Les autres liens, y compris ceux qui ne sont pas générés par SPIP, s'afficheront en vert souligné.

> On note ici une propriété fondamentale des feuilles de style : les règles graphiques s'appliquent dans l'ordre allant de la plus générique à la plus spécifique. Cela permet de spécifier un comportement général pour la plupart des éléments, et de modifier ce comportement pour un plus petit sous-ensemble d'éléments. Cette caractéristique fait toute la puissance des feuilles de style.

Exposer le lien activé

Depuis SPIP 1.7.1, on peut mettre en évidence, dans une liste de liens, celui concernant la page où l'on se trouve. Le style que l'on utilisera pour ce faire est : `.on`. Voir : « Exposer un article dans une liste ».

Styles des citations dans SPIP

Juin 2006 — maj : Janvier 2007

Citations d'auteurs, de code informatique ou de poésie, SPIP facilite l'insertion de citations dans vos textes. *Code is poetry !*

Les citations de SPIP

Pour simplifier l'insertion de citations dans vos textes, SPIP 1.7, SPIP 1.7.2 propose le raccourci suivant :

```
<quote>Ceci est une citation.</quote>
```

ce qui donne :

> Ceci est une citation.

Son apparence est contrôlée par la définition de style `blockquote.spip`.

Signalons l'existence d'une autre balise HTML, que vous pouvez également utiliser dans vos textes : à la différence de celle utilisée dans le raccourci de SPIP (`<blockquote>`) qui « va à la ligne », `<q>` permet de faire une citation au fil du texte.

Poésie

SPIP 1.7, SPIP 1.7.2 propose de plus une mise en forme particulière pour la poèsie, avec le raccourci suivant :

```
<poesie>Voici de la poésie.</poesie>
```

ce qui donne :

> Voici de la poésie.

Son apparence est contrôlée par la définition de style `div.spip_poesie`.

Le code informatique affiché dans SPIP

Pour citer du code informatique au fil du texte, on utilise le raccourci SPIP suivant :

```
<code>Du code dans le texte</code>
```

Le style associé est `.spip_code`.

Introduit dans SPIP 1.3, le très astucieux raccourci `<cadre>...</cadre>` permet de présenter du code informatique dans un bloc de formulaire, ce qui facilite le copier-coller du code. Son apparence est contrôlée par la définition de style : `.spip_cadre`.

Ces deux derniers raccourcis, et leurs définitions de style correspondantes, sont peu utilisés, sauf dans le cas d'une documentation technique (comme celle-ci) où l'on doit citer des morceaux de code informatique, des noms de fichiers ou de répertoires.

Styles des logos, images et documents

Juin 2006 — maj : Décembre 2007

Les styles CSS associés par SPIP aux logos, aux images et autres documents permettent de contrôler leur affichage dans la page.

Style des logos

Les logos des éléments (rubriques, articles, brèves, auteurs, sites) sont systématiquement dotés du style `.spip_logos`, placé sur la balise HTML `<img>`. Ce style peut être très utile pour, par exemple, déterminer la position du logo.

Ainsi pour aligner à droite les logos placés dans le cartouche, par exemple, tout en ménageant un espace entre l'image et le texte, vous ferez simplement :

```
.cartouche .spip_logos {
   float: right;
   margin-left: 1em;
}
```

Styles des images et documents

Depuis SPIP 1.8, l'insertion avec les raccourcis `<docXX|left>` et `<imgXX|right>`, de documents et images dans le corps de texte d'un article (ou d'une brève), est contrôlée par des styles CSS.

Trois définitions de style permettent de personnaliser l'affichage des documents, de leurs titres et légendes :
- `.spip_documents` concerne la boîte qui contient la vignette et les informations du document (`<docXX|left>`) ou l'image insérée sans titre ni descriptif (`<imgXX|right>`) ;
- `.spip_doc_titre` contrôle l'affichage du titre du document ;
- `.spip_doc_descriptif` contrôle l'affichage du descriptif du document.

Trois styles sont utilisés en complément, indispensables pour définir le positionnement du document ou de l'image dans la page :
- `.spip_documents_center` quand le document est centré (`<docXX|center>`) ;
- `.spip_documents_left` quand le document est aligné à gauche (`<docXX|left>`) ;
- `.spip_documents_right` quand le document est aligné à droite (`<docXX|right>`).

Notez bien que ces styles concernent de la même façon les images (`<imgXX>`) et les documents (`<docXX>`).

NB : Avant de modifier radicalement ces définitions de style, notez que celles-ci sont également utilisées, dans les squelettes par défaut (depuis SPIP 1.9), pour styler les éléments du portfolio et des listes de documents joints (aux articles et/ou rubriques).

Images avec ou sans bordure

Depuis SPIP 1.9 il est possible d'appliquer une bordure aux images. Par exemple : `.spip_documents img { border: 1px solid #666; }` encadrera vos images d'une fine bordure grise.

Inversement, pour ne pas voir vos logos encadrés de cette horrible bordure bleue qui signale traditionnellement les « images cliquables » sur certains navigateurs (comme FireFox), n'oubliez pas de régler leur bordure à zéro : `.spip_logos { border: 0; }`.

Styles des tableaux de SPIP

Juin 2006 — maj : 27 juillet

Les styles CSS permettent de paramétrer finement l'affichage des tableaux générés par SPIP.

Les tableaux constituent un moyen facile de présenter l'information en rangées et colonnes de cellules.
Pour tirer profit de cet article, mieux vaut connaître la syntaxe HTML des tableaux. La balise `<table>`, qui permet d'insérer un tableau dans la page, est l'une des balises les plus utilisées en HTML.

Les tableaux sont créés dans SPIP de la façon suivante :

```
||Légende du Tableau|Résumé du Tableau||
| {{Nom}} | {{Date de naissance}} | {{Ville}} |
| Jacques | 5/10/1970 | Paris |
| Claire | 12/2/1975 | Belfort |
| Martin | 1/31/1957 | Nice |
| Marie | 23/12/1948 | Perpignan |
```

ce qui produit cet affichage :

Légende du Tableau

Nom	Date de naissance	Ville
Jacques	5/10/1970	Paris
Claire	12/2/1975	Belfort
Martin	1/31/1957	Nice
Marie	23/12/1948	Perpignan

Les styles permettent de paramétrer finement l'affichage des tableaux :

- `table.spip` permet de modifier le comportement général du tableau, notamment ses dimensions, ses marges et sa position (calé à gauche, centré, etc.) ;

- `table.spip caption` concerne la légende (optionnelle) du tableau ;

- `table.spip tr.row_first` concerne la « première ligne » du tableau (ici en jaune). Pour que la première ligne soit prise en compte comme rangée de titres, il faut que chacun des éléments qu'elle contient soient en gras ;

- `table.spip tr.row_odd` et `table.spip tr.row_even` pour les autres lignes. Un des intérêts de ces styles est la possibilité d'appliquer deux couleurs différentes via « `row_odd` » et « `row_even` » (ici, gris clair et gris foncé), permettant d'alterner les couleurs d'une ligne à l'autre, ce qui facilite agréablement la lecture du tableau.

- `table.spip th` et `table.spip td` concernent les cellules du tableau, et permettent, par exemple, de contrôler leur espacement intérieur (`padding`), afin d'aérer la présentation.

Ils sont beaux, mes formulaires !

Avril 2004 — maj : 23 juillet

Vous avez personnalisé la mise en page et la typographie de votre site, mais maintenant ce sont les formulaires SPIP qui jurent totalement sur le reste ! Pas de panique, là aussi les feuilles de style remédient au problème.

Différents formulaires sont utilisés dans le site public, pour le moteur de recherche interne, la rédaction des messages des forums, les inscriptions à l'espace privé, etc. Il en va pour ces formulaires SPIP comme pour le reste : leur aspect graphique peut être modifié via CSS afin de s'insérer sans hiatus dans votre design.

Pour tirer profit de cet article, mieux vaut connaître les balises HTML propres aux formulaires.

À chaque formulaire son style

Tous les formulaires SPIP utilisés dans le site public sont contenus dans une `div` dotée du même style, `.formulaire_spip`, qui permet d'appliquer facilement une modification devant concerner l'ensemble de ces formulaires.

Par exemple, pour mettre en gras tous les descriptifs des champs de saisie (`<label>`) de vos formulaires, vous stylerez ainsi :

```
.formulaire_spip label {
   font-weight: bold;
}
```

Depuis SPIP 1.9, chaque formulaire est, de plus, doté d'un style qui lui est propre. Celui-ci permet, inversemment, de modifier l'apparence de certains formulaires en particulier, sans interférer sur les

autres.
Chacun de ces styles est nommé de même que la balise appelant le formulaire et son squelette. Par exemple, le fichier HTML du formulaire de recherche est `formulaire_recherche.html` ; celui-ci est inséré dans les squelettes grâce à la balise `#FORMULAIRE_RECHERCHE` ; et le style qui lui est associé est donc `.formulaire_recherche`.
Depuis SPIP 1.9.2, les squelettes des formulaires sont déplacés et ainsi renommés : `dist/formulaires/recherche.html`.

Styles des champs de saisie

Le style `.forml` est appliqué aux champs de saisie des formulaires. Il permet de définir la couleur du fond et la largeur des champs de saisie, mais aussi leur taille et police de caractère. Par exemple :

```
.forml {
   width: 99%;
   padding: 1px;
   border: 1px solid #666;
   font-family: Verdana;
   font-size: 11px;
}
```

Ce style est particulièrement utile pour homogénéiser l'ensemble des champs de saisie, quelques soient les balises qui le reçoivent, telles que `<input>` et `<textarea>`, toutes deux présentes dans le formulaire de forum.

Depuis SPIP 1.9, chaque champ de saisie est étiqueté d'un terme explicatif, enveloppé dans une balise HTML `<label>`. On modifiera l'apparence de ces étiquettes par cette définition de style : `.formulaire_spip label`.

Les styles CSS permettent non seulement de changer les couleurs et les polices de caractères, mais aussi de gérer le positionnement relatif des objets dans la page. Ils permettent même de définir précisément la disposition des éléments entre eux (par exemple : `<input>`, `<textarea>` et `<label>`), sans utiliser de `<table>`

pour la mise en page.

Des boutons à vos couleurs

Une nouvelle qui ravira les débutant-e-s en CSS : on peut changer la couleur des champs mais aussi des boutons des formulaires [1]. Le style `.spip_bouton` est celui utilisé pour les boutons des formulaires SPIP.

Par exemple, pour que les boutons aient un fond bleu clair, et une bordure suffisemment visible (épaisse, en relief et bleue foncée), modifiez la règle suivante dans votre feuille de style :

```
.spip_bouton {
   background-color: #b0d0FF;
   border: 2px outset #000060;
   color: black;
}
```

Depuis SPIP 1.7, SPIP 1.7.2 , le formulaire de forum propose une barre de raccourcis typographiques. L'apparence de celle-ci est contrôlée par le style `.spip_barre`. Ainsi, pour modifier l'apparence des icônes qui la composent, et, par exemple, distinguer celles-ci au survol de la souris, vous définirez les styles pour `.spip_barre a img` puis `.spip_barre a:hover`

`img`.

Organiser ces éléments de façon visiblement logique

Depuis SPIP 1.9, les différents éléments d'un formulaire sont plus rigoureusement regroupés en « blocs logiques » grâce aux balises HTML dédiées `<fieldset>` et `<legend>`. Leur apparence est donc contrôlée par `.formulaire_spip fieldset` et `.formulaire_spip legend`, tout simplement (ceci était autrefois réalisé avec `.spip_encadrer`). Utile pour encadrer chaque partie d'une bordure, et aérer vos formulaires en espaçant ces blocs les uns des autres, par exemple.

Quelques styles complémentaires permettent d'affiner encore la présentation de vos formulaires :

- L'apparence des messages d'erreur et autres réponses renvoyées par les formulaires peut être personnalisée par le style `.reponse_formulaire`.

- Certains formulaires permettent de prévisualiser les informations saisies avant de les envoyer. Depuis SPIP 1.9, l'apparence de cette prévisualisation est contrôlée par le style `.previsu`.

- Enfin, ce dernier style n'est pas utilisé dans les formulaires, mais est lié au formulaire de recherche interne au site : `.spip_surligne` permet de mettre en évidence les termes recherchés dans les pages de résultats.

P.-S.

Cet article de Pompage.net introduit au style des formulaires : « CSS : on reprend tout à zéro ! (13ème épisode) ».
Lire aussi : Comment ne pas styler les éléments de formulaire ?

Notes

[1] Notons cependant que certains navigateurs imposent leurs propres boutons stylisés et ne vous laisseront pas en changer l'aspect.

Ressources CSS pour en savoir plus

Avril 2004 — maj : Décembre 2007

Le CSS n'est pas propre à SPIP. C'est un langage normalisé par le W3C et très utilisé sur le Web pour l'habillage graphique. Vous êtes libres de vous contenter du niveau abordé dans ces pages, mais nous vous invitons à consulter la très nombreuse documentation existant sur le CSS.

Citons quelques ressources intéressantes.

Ressources francophones sur le CSS

- **OpenWeb** est un site offrant à la fois un regard expert sur le Web et des exemples concrets d'utilisation sur les CSS et autres standards du Web.
- un cours « CSS débutant », de Mammouthland ;
- un excellent tutorial : « **CSS : on reprend tout à zéro !** », par pompage.net, qui permet de débuter en douceur et en toute beauté ;
- une collection de recettes pour une utilisation efficace des CSS, et un forum CSS et standards W3C pour s'entre-aider, chez Alsacreations ;
- un pense-bête, pour ne pas se perdre dans le feu de l'action ;
- et surtout, la traduction française de la documentation officielle, assez aride mais

incontournable : les **spécifications CSS2** du W3C.

Ressources anglophones

- Listes d'éléments : des exemples d'effets graphiques à foison ;
- de très bons tutoriaux sur le positionnement d'objets avec les CSS ;
- le W3C a ses propres *tips 'n 'tricks*...
- et pour ceux qui ont le goût du risque, l'intégrale des spécifications originales du W3C !

Petits outils bien utiles

Si vous utilisez l'excellent navigateur Firefox, des extensions telles que *CSS Viewer*, le plug-in *EditCSS*, ou la barre d'outils *Web Developer* vous rendront des services appréciables en matière de CSS.