Guide du webmestre et du bidouilleur

Rechercher

français

tout le site

Rechercher

Modifications récentes

- Le calendrier de SPIP 1.8.2
- Internationaliser les

SPIP-Contrib : des outils pour les webmestres

L'espace des contributions externes, qui recense l'ensemble des scripts, filtres, squelettes, documentations à imprimer, etc., fournis à la communauté par les utilisateurs de SPIP.

- Fichier CSV vers SPIP
- Etiquetage ICRA automatique
- Des fils RSS 1.0, RSS 2.0 et Atom 1.0 pour votre site SPIP
- Un graphique de popularité
- Galerie SPIP + PHP, simple et prête à l'emploi
- Abandonnons RSS 0.91 !
- csv2spip : gestion des utilisateurs de SPIP à partir de fichiers CSV

SPIP-Contrib : mes_fonctions

SPIP-Contrib : les squelettes

Les articles de documentation de SPIP-Contrib

Mise en page : manuel de référence

Comment créer sa propre mise en page pour un site géré sous SPIP.

- Principe général
- Des boucles et des balises
- La syntaxe des boucles
- La syntaxe des balises SPIP
- La boucle ARTICLES
- La boucle RUBRIQUES
- La boucle BREVES

[...]

SPIP pas à pas

Pas à pas, comment créer un SPIP qui défie les limites.

- Mon premier squelette

- Un squelette, plusieurs articles
- Une rubrique
- Boucles en boucles
- Gérer le cache
- Des filtres

Guide des fonctions avancées

Au-delà du manuel de référence, vous trouverez ici une description détaillée des fonctions plus avancées à la disposition du webmestre.

- Spip et les feuilles de style
- <INCLURE> d'autres squelettes
- Réaliser un site multilingue
- Internationaliser les squelettes
- Utiliser des URLs personnalisées
- Le moteur de recherche
- Les variables de personnalisation

[...]

Initiation : utiliser les feuilles de style avec

SPIP

Pour tirer parti de toute la souplesse de SPIP, il est recommandé d'utiliser les feuilles de style. Pas de panique, cette petite initiation permettra aux débutants de raccrocher les wagons...

- Introduction
- Des styles qui ont de la « class »
- Une typographie personnalisée
- Ils sont beaux, mes formulaires !
- Pour en savoir plus

Trucs et astuces

- Afficher automatiquement selon la date ou selon un ordre imposé
- Trier des articles par ordre alphabétique, sauf un qu'il faut afficher en premier
- Plusieurs logos pour un article
- Afficher les derniers articles de vos rédacteurs par rubrique
- Afficher des éléments par lignes dans un tableau
- Ne pas afficher les articles publiés depuis plus d'un an
- Présenter les résultats d'une recherche par secteurs

[...]

Le développement de SPIP et ses outils

Les différents outils de communication utilisés pour développer SPIP.

Tutorial : utilisation avancée des boucles et des mots-clés

Sécurité : SPIP et IIS

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.

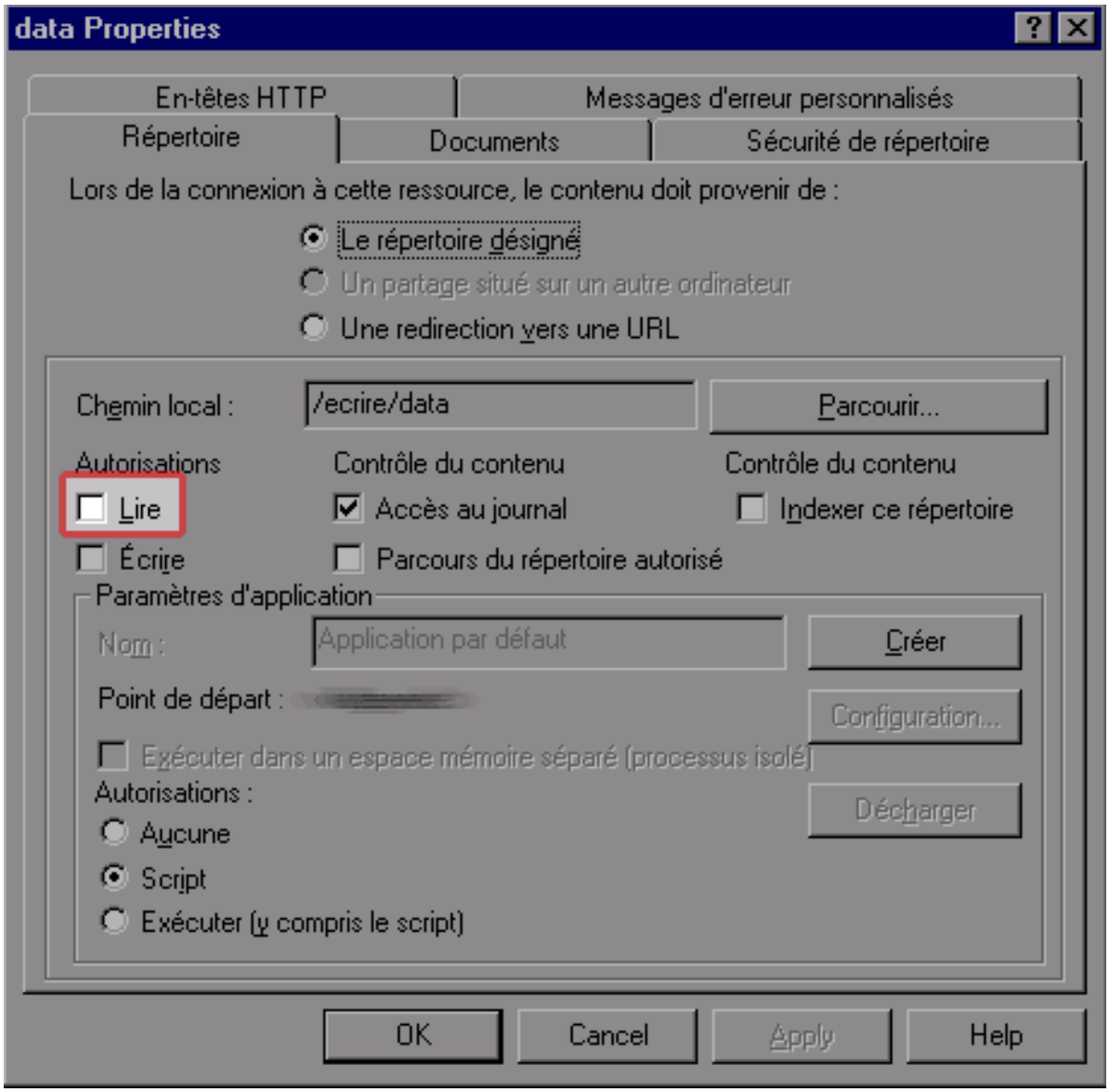
La sécurité de tous ces fichiers est assurée traditionnellement par des fichiers de configuration d'accès nommés `.htaccess`. SPIP génère automatiquement ces fichiers pour empêcher l'accès aux données sensibles stockées sur le serveur : vous pouvez vérifier que `CACHE` et `ecrire/data` contiennent chacun 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 ».

[1] Les mots de passe sont chiffrés par SPIP, mais gardez bien à l'esprit qu'aucune protection n'est inviolable.

Qu'est-ce que les fichiers « dist » ?

Comme vous le savez sûrement déjà (sinon, lisez le tutorial ou le manuel de référence), le système de squelettes est basé sur des fichiers `.html` contenant la présentation graphique du site. Par exemple, « `article.html` » présente les articles, « `rubrique.html` » présente les rubriques...

Or, nous avons remarqué que fréquemment, les utilisateurs qui manipulaient leur site public en modifiant ces fichiers `.html` fournis avec SPIP rencontraient des problèmes lors des mises à jour, s'ils n'avaient pas pris leurs précautions en sauvegardant les fichiers modifiés.

En effet, en réinstallant tous les nouveaux fichiers livrés avec SPIP, ils écrasaient purement et simplement leurs fichiers modifiés (oubliant de faire une copie de sauvegarde de leurs modifications).

Depuis [SPIP 1.8], les fichiers `.html` sont mieux rangés. Un répertoire **« dist »** est destiné aux fichiers fournis avec la distribution de spip. Ces fichiers contiennent les informations sur la mise en page par défaut du site et ne devraient pas être modifiés.

Ainsi, un utilisateur qui veut créer sa propre mise en page, développera ses fichiers `article.html`, `rubrique.html`, etc. à la racine du site, ou mieux, dans le répertoire **« squelettes »** dédié. A la prochaine réinstallation de SPIP, seuls les fichiers dans **« dist »** seront écrasés et le webmestre ne perdra pas ses personnalisations.

> **Remarque :** Les fichiers `.php3`, pendants des fichiers `.html` dans un squelette SPIP, doivent toujours rester à la racine du site [1].

> *Historique :* Depuis [SPIP 1.3] et jusqu'à [SPIP 1.7.2], les fichiers de squelettes fournis dans la distribution de SPIP étaient nommés « `article-dist.html` », « `rubrique-dist.html` », et ainsi de suite. Pour personnaliser ces fichiers, il suffisait de les renommer d'abord « `article.html` », « `rubrique.html` », etc. (sans le **`-dist`**).

Pour aller plus loin

Depuis [SPIP 1.8], SPIP recherche les fichiers `.html` des squelettes d'abord à la racine du site, puis s'ils n'y existent pas, dans le répertoire **« squelettes »**. En dernier recours, SPIP prend les fichiers par défaut dans le répertoire **« dist »**.

> **Remarque :** De plus, si on définit une variable de personnalisation `$dossier_squelettes` comme expliqué dans la documentation correspondante, SPIP cherchera les fichiers de squelette en priorité dans ce sous-dossier, avant tous les précédents.

Voici l'ordre (par priorité décroissante) dans lequel sont utilisés les fichiers de squelettes selon leur nom :

- **`rubrique=10.html`** : si ce fichier existe, il ne s'applique qu'à la rubrique numéro 10 ;

- si ce fichier n'existe pas, SPIP regarde si il n'y a pas un fichier **`rubrique-10.html`**, si ce fichier existe, la rubrique 10 ainsi que ses sous-rubriques l'utilisent, c'est donc « récursif » ;

> *Note :* pour que ces fichiers soient pris en compte il faut que le fichier par défaut (`rubrique.html`) se trouve dans le même

répertoire.

- si ce fichier n'existe pas, SPIP regarde s'il n'y a pas un fichier **`rubrique.html`**, qui s'applique à toutes les rubriques du site qui ne sont pas concernées par les fichiers indiqués ci-dessus ;

> *Historique :* Jusqu'à [SPIP 1.7.2], si ce fichier n'existe pas, SPIP utilise alors le fichier **`rubrique-dist.html`** qui est le fichier fourni par défaut. Si vous voulez modifier ce fichier, renommez-le en `rubrique.html`, de façon à ne pas écraser vos modifications à la prochaine mise à jour de SPIP.

[1] sauf, éventuellement, ceux qu'on n'utiliserait que dans un appel `<INCLURE(squel.php3)>`. Dans ce cas, le fichier squel.php3 peut *aussi* être dans le dossier `squelettes`.

FAQ webmestre

Les bases

1. Comment fais-je pour modifier la mise en page du site public ?

La gestion de la mise en page s'appuie sur des fichiers à l'extension `.html` appelés *squelettes* de mise en page. Leur rôle correspond grosso modo à ce que d'autres logiciels nomment « modèles » « gabarits », ou en anglais, « *templates* ».

Chaque fichier est associé à un type de page différent : ainsi un *squelette* pour le sommaire, un pour l'affichage des articles, un pour l'affichage des rubriques, etc. Un squelette contient du HTML standard définissant l'habillage de la page, dans lequel on insère des « codes » spécifiques à SPIP afin de définir quelles informations vont venir « habiter » cet habillage.

Le langage des squelettes de SPIP est très souple et permet de réaliser des mises en page très variées : un simple coup d'oeil à uZine, Vacarme, Hacktivist News Service ainsi que les sites enregistrés par leurs créateurs sur cette page saura vous

en convaincre. Il est donc dommage de garder la mise en page d'origine, même si celle-ci est très utile pour se mettre le pied à l'étrier.

2. Est-il possible d'écrire ces squelettes soi-même ?

Oui, c'est un des intérêts majeurs de SPIP. Pour cela allez voir :

- le tutorial, pour comprendre les bases de la programmation des squelettes.
- le manuel de référence, qui liste toutes les possibilités de programmation.

3. Je ne sais pas / ne veux pas apprendre à programmer. Peut-on utiliser des mises en pages déjà existantes ?

Oui. En dehors de la mise en page par défaut, d'autres jeux de squelettes sont disponibles sur le site des contributions à SPIP, dans la rubrique « Squelettes ».

Il suffit en général de récupérer l'archive voulue (le fichier au format .zip ou .tar.gz, au choix), de la décompresser chez vous, et de transférer son contenu par FTP à la racine de votre site SPIP. Vous pouvez faire une sauvegarde de vos fichiers .html actuels, au cas où vous voulez revenir en arrière.

4. Il n'y a pas beaucoup de jeux de squelettes disponibles. Pourquoi ?

Ces jeux de squelettes sont alimentés par les webmestres SPIP qui nous fournissent leurs créations. Nous comptons donc sur les webmestres pour compléter cette base de squelettes afin d'encourager l'entraide et la richesse des sites SPIP. (cf. section « Partager » plus bas dans cette FAQ)

Créer ses squelettes

1. Peut-on utiliser un éditeur textuel pour créer et modifier ses squelettes ?

Oui, comme on le ferait pour du HTML classique.

2. Peut-on utiliser un éditeur graphique (WYSIWYG) pour créer et modifier ses squelettes ?

Oui, comme on le ferait pour du HTML classique. Voir cependant la question suivante.

3. J'essaie d'utiliser un éditeur graphique pour créer mes pages, mais

il modifie les tags SPIP. Peut-on résoudre ce problème ?

Certains éditeurs graphiques « corrigent » automatiquement les tags qu'ils ne comprennent pas. La plupart ont toutefois une option permettant de désactiver cette fonctionnalité. Nous avons consacré un article spécifique à DreamWeaver, mais la démarche est équivalente pour les autres éditeurs (GoLive...).

Partager vos créations

1. J'ai écrit des squelettes pour mon site. Comment fais-je pour qu'ils soient disponibles à tous ?

N'hésitez pas à vous inscrire sur le site SPIP-Contrib mentionné plus haut, afin de proposer vos squelettes au téléchargement et que d'autres puissent à leur tour s'en inspirer pour créer leur propre site.

Rapidité du site public

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.

Contribuer au développement de SPIP

Si vous voulez contribuer à la programmation de SPIP, l'idée la plus importante à retenir est la suivante : *vous arrivez sur un projet qui est déjà fonctionnel*. Ce projet est muni d'un ensemble de règles qui, toutes arbitraires qu'elles peuvent paraître, assurent sa cohérence. Ces règles n'ont pas besoin d'être énoncées explicitement pour exister : certaines sont clairement visibles après un examen plus ou moins détaillé du code, et les règles tacites doivent être respectées au même titre que les autres.

Il est formellement conseillé de bien suivre ces règles. Ce respect n'a pas à être lié ou non à vos goûts personnels : il permet de garder sa cohérence et son unité au projet, et de le conserver aussi lisible qu'il l'était auparavant. N'oubliez pas que d'autres personnes que vous sont amenées à lire, comprendre, voire modifier votre code.

Par exemple, il est évident que les fonctions SPIP sont écrites sous la forme ma_fonction(). Dans le cadre de ce projet, il serait donc parfaitement déplacé d'ajouter des fonctions en les écrivant MaFonction() - même si dans l'absolu cette forme n'est pas plus critiquable que l'autre.

Tout ceci n'empêche pas évidemment de critiquer une règle et d'en proposer une meilleure, le cas échéant. N'hésitez pas à le faire ; mais il doit y avoir de véritables raisons à cela.

Enfin, tout règle souffre des exceptions. Mais celles-ci doivent avoir une véritable justification, non uniquement la paresse du programmeur ; elles doivent être le plus rares possible. Notamment, garder à l'esprit que le « provisoire » a souvent tendance à devenir définitif quand personne n'a envie de le corriger ; or il est logique et juste que tout programmeur soit responsable de la finition de son propre code, mais non de celui des autres.

Règles de présentation et d'écriture

Les règles qui suivent sont communes à un nombre plus ou moins grand de langages de programmation : au minimum tous les langages présentant une syntaxe similaire à PHP (c'est-à-dire, outre PHP lui-même, C, C++, Java....).

Ces règles sont communément acceptées, de façon aussi naturelle que les règles de présentation et de typographie d'un texte en langage naturel ; d'ailleurs elles sont fréquemment similaires.

Présentation

- Le code doit être espacé et indenté de manière à mettre en valeur sa structure et la séparation entre les différents blocs logiques (fonctions notamment). L'espacement et l'indentation doivent être suffisants pour rendre la structure compréhensible dès le premier regard ; ils ne doivent pas être excessifs. On doit y apporter le même soin qu'à la division en paragraphes d'un texte en langage naturel.

- L'indentation sera faite de préférence avec le caractère de tabulation. Cela permet de choisir librement la profondeur d'indentation dans les options de son éditeur de texte, tout en n'imposant pas ce choix aux autres développeurs.

- Tout bloc contenu à l'intérieur d'une paire d'accolades sera indenté d'une et une seule tabulation. De même, récursivement, pour les sous-blocs : ajout d'une et une seule tabulation supplémentaire à chaque entrée dans un niveau de profondeur supplémentaire. Cette règle vaut bien aussi pour la déclaration de fonctions.

- Le code qui ne fait pas partie d'une fonction ne doit pas être indenté.

- L'utilisation des transitions PHP-HTML (<?php et ?>) doit être minimisée.

L'éviter lors qu'il s'agit d'afficher uniquement de petits morceaux de HTML. Ne pas oublier qu'un petit morceau de code PHP inséré au milieu d'un océan de HTML est invisible sans un examen très attentionné.

Typographie

- Lors de l'utilisation de parenthèses ou de crochets, il ne faut pas laisser d'espace après la parenthèse ouvrante ni avant la parenthèse fermante.

- Lors de l'utilisation d'opérateurs binaires (+, =, *, AND, ...), il faut laisser un espace de part et d'autre de l'opérateur. Exception manifeste dans cette phrase, où les opérateurs sont mentionnés et non utilisés en tant que tels.

- Les opérateurs unaires (!, ...) doivent être collés au paramètre auquel ils s'appliquent.

- Par convention, lors d'un appel de fonction, il n'y a pas d'espace devant la parenthèse ouvrante : « f($x) » et non « f ($x) ». A contrario, et pour bien distinguer, on laisse un espace devant la parenthèse quand il s'agit d'une structure de contrôle intégrée au langage : « if (!$x) » et non « if(!$x) ».

- Les virgules et points-virgules sont suivis mais non précédés d'un espace.

Règles de programmation

Réfléchir

Avant de programmer une nouvelle fonctionnalité, réfléchir...

- méthodes et algorithmes utilisés pour l'implémentation : légèreté, performance, robustesse (ne pas hésiter à faire quelques calculs grossiers pour valider les choix) ;
- adéquation au projet : portabilité, sécurité, souplesse ;
- implications sur les autres fonctionnalités : modifications et ajouts à faire sur les fonctionnalités existantes ;
- place « naturelle » pour cette fonctionnalité dans le projet : en matière d'interface, de fichiers...

Ne pas négliger la factorisation ou mise en commun du code (par des fonctions, notamment dans des fichiers à inclure). Eviter par contre le plus possible les fichiers inclus contenant du code hors fonctions (sauf lorsque c'est « naturel » et voulu).

Nommer

- *Variables et fonctions* :

Quel que soit le projet, le nommage doit rester homogène pour que le code soit facile à lire. Ainsi, sous SPIP, les noms de variables et de fonctions seront en minuscules ; les noms composés, de la forme variable_composee.

D'une manière générale, les noms seront ni trop brefs, ni trop longs ; ile seront suffisamment explicites. Cette règle est particulièrement importante pour les variables globales, qui peuvent être partagées entre plusieurs fichiers et de nombreuses fonctions. Pour les variables locales (i.e. à une fonction), la règle est plus souple. Notamment, on peut employer des variables d'une lettre, par exemple pour obtenir des expressions plus compactes. Remarquer que dans tous les langages de programmation, un certain nombre de lettres sont par tradition associées à certains usages (exemples : $i, $j pour des compteurs de boucles, $n pour un dénombrement, $t pour un instant ou une durée en secondes...). Ne pas détourner ces conventions permet à vos lecteurs d'être plus vite dans le bain.

- *Fichiers* :

Pour des raisons historiques, les fichiers à inclure dans l'espace public seront appelés *inc-fichier.php3*. Dans l'espace privé, ce sera *ecrire/inc_fichier.php3* (noter le tiret bas à la place du tiret normal). Les fichiers de l'espace public appelés par redirection HTTP depuis l'espace privé sont appelés *spip_fichier.php3*. Tous les autres fichiers ont un nom qui ne commence ni par "inc", ni par "spip".

Tester

Une fois une modification importante apportée, il est bon de la tester soi-même, sans attendre que quelqu'un d'autre le fasse à sa place. Dans le cadre de SPIP, cela veut dire vérifier que le programme marche de manière correcte sur un certain nombre d'hébergeurs (par exemple : Altern, Free...) et de configurations (par exemple : différentes versions de PHP, de MySQL, restriction plus ou moins grande des droits d'accès aux répertoires...) ; mais aussi qu'un certain nombre de situations parmi les plus courantes (dans le cas d'une interface graphique notamment) sont traitées correctement.

Partager vos modifications

Une fois que vous êtes satisfait de votre modification du code, il est grand temps

d'en parler avec les autres développeurs de SPIP, et de voir s'il mérite d'être intégré à la distribution officielle de SPIP... Rendez-vous sur la liste de diffusion spip-dev. A bientôt !

Mise en page : manuel de référence

Comment créer sa propre mise en page pour un site géré sous SPIP.

Principe général

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.

Cette opération est traditionnellement assez pénible :
- 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 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 pages HTML nommées *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.

Pour chaque type de document, un couple de fichiers

L'intérêt (et la limite) d'un système de publication automatisé, c'est que l'on ne va pas redéfinir une interface différente en HTML pour chaque page isolée. Par exemple, toutes les articles bénéficieront de la même interface, simplement le système placera des informations différentes dans ce graphisme (on verra plus loin que SPIP autorise cependant une certaine souplesse).

L'avantage de cette manière de procéder est évident : on définit un format-type (squelette) pour, par exemple, tous les articles, et le système fabriquera chaque page individuelle en plaçant automatiquement le titre, le texte, les liens de navigation... de chaque article.

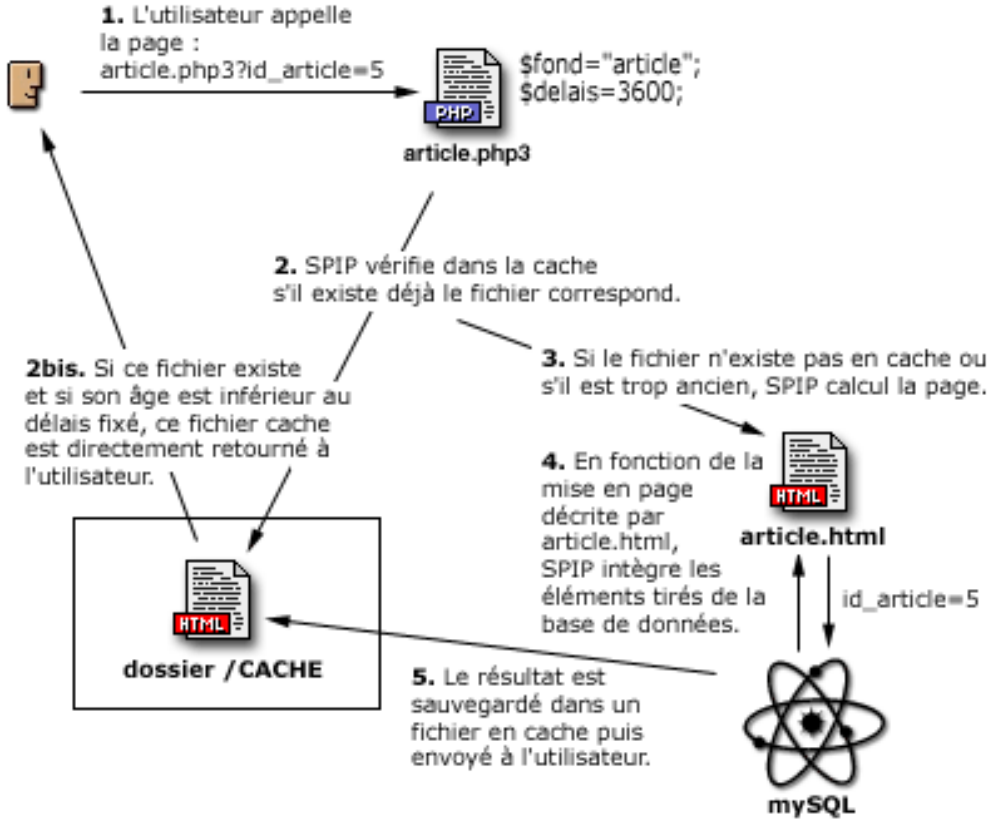
Pour chaque type de document, SPIP vous demande *deux fichiers* : un fichier **.php3** et un fichier **.html**. Lors de l'installation de SPIP, vous trouverez ainsi les couples : « article.php3 / article.html », « rubrique.php3 / rubrique.html », etc.

Vous pouvez naturellement modifier ces couples, et en créer d'autres. Depuis [SPIP 1.8.2], néanmoins, il existe un fichier **page.php3** utilisable pour tous les types de documents que l'on pourra créer.

Le principe de fonctionnement du cache

L'appel d'une page spécifique se fait par l'intermédiaire du fichier **.php3**. Par exemple, pour appeler l'article n°5, l'URL correspondante est :

```
http://monsite.net/article.php3?id_article=5
```



1. Le fichier appelé est donc `article.php3`, avec en paramètre `id_article=5`.

2. Le fichier `article.php3` est un fichier PHP ; sa première tâche consiste à vérifier dans le dossier `/CACHE` sur votre serveur, s'il existe déjà un fichier correspondant à cet article.

2bis. Si un tel fichier existe dans `/CACHE`, `article.php3` vérifie sa date de création. Si ce fichier est suffisamment récent, il le retourne directement à l'utilisateur. Le processus de consultation est alors terminé.

3. S'il n'existe pas un tel fichier dans `/CACHE` (première visite sur cet article, par exemple), ou si son âge est trop ancien, SPIP démarre le calcul de cette page.

4. C'est alors la page `article.html` qui est chargée et analysée. Cette page contient la mise en page correspondant à ce type de document. Il s'agit de HTML complété d'indications permettant de placer les éléments tirés de la base de données. En fonction des éléments requis par `article.html`, SPIP va chercher les informations nécessaires tirées de la base de données mySQL et les insérer aux endroits prévus.

5. Un fichier est ainsi fabriqué par `article.php3`, à partir de la description contenue dans `article.html`, avec les éléments tirés de la base de données. Ce fichier est alors sauvegardé dans le dossier `/CACHE` et renvoyé au visiteur.

Lors d'une visite suivante, si le délai entre les deux visites est suffisamment court, c'est donc ce nouveau fichier stocké dans `/CACHE` qui est retourné, 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 le fichier en cache qui est retourné, même s'il est « trop âgé ».

> *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 manoeuvre » ayant le bon goût de surcharger le serveur et de le faire parfois planter).

Le fichier .PHP3

Le fichier « `.php3` » est très simple. Par exemple, `article.php3` contient uniquement :

```
<?php
$fond = "article";
$delais = 24 * 3600;

include ("inc-public.php3");
?>
```

Son seul but est donc de fixer deux variables (`$fond` et `$delais`) et d'appeler le fichier qui déclenche le fonctionnement de SPIP (`inc-public.php3`).

La variable `$fond` est le nom du fichier qui contient la description de la mise en page (le *squelette*). Ici, puisque `$fond="article"`, le fichier de description sera contenu dans `article.html` [1]. Notez bien que, dans la variable `$fond`, on n'indique pas la terminaison « `.html` ».

> *Remarque.* L'intérêt de choisir soi-même le nom du fichier de squelette (que l'on aurait pu déduire automatiquement du nom du fichier `.php3`) est, si nécessaire, d'utiliser un autre nom. Cela pour ne pas écraser, éventuellement, des fichiers HTML qui subsisteraient d'une ancienne version du site que l'on ne souhaite pas supprimer. S'il existe, d'une ancienne version du site, un fichier `article.html` que l'on ne souhaite pas effacer, on utilisera par exemple un fichier squelette pour SPIP intitulé `article-nouveau.html`, et on fixera dans `article.php3` : `$fond="article-nouveau"`.

La variable `$delais` est l'âge maximum pour l'utilisation du fichier stocké en `/CACHE`. Ce délai est fixé en secondes. Un délai de 3600 correspond donc à une heure ; un délai de 24*3600 est donc de 24 heures.

On jouera sur cette valeur en fonction de la fréquence des ajouts de contenu du site (nouveaux articles, nouvelles brèves...). Un site actualisé plusieurs fois par jour pourra adopter un délai d'une heure ; un site publiant quelques articles par semaine pourra adopter un délai nettement plus long. De même, le contenu des pages est important : si vous insérez la syndication du contenu de sites fréquemment mis à jour, vous souhaiterez sans doute adapter votre propre délais à celui des sites référencés.

> *Remarque.* Certains webmestre succombent à la tentation de fixer des délais dérisoires (quelques secondes), pour que le site corresponde très exactement, à chaque instant, aux dernières modifications de la base de données. Dans ce cas, vous perdez tous les avantages du système de cache : les visites sont nettement ralenties et, à l'extrême, sur des sites très fréquentés, vous pouvez provoquer des plantages de la base de données (ou vous faire virer par votre hébergeur parce que vous monopolisez la puissance de sa machine...).
>
> *Remarque.* Une autre raison qui semble pousser les webmestres à fixer des délais très courts est la présence des forums sur leur site. En effet, pour que les contributions s'affichent dès qu'elles sont postées, ils pensent nécessaire de réduire le délais de recalcul. C'est inutile : SPIP gère cet aspect automatiquement ; lorsqu'une contribution à un forum est postée, la page correspondante est effacée du cache, et recalculée immédiatement, quel que soit le délai fixé pour cette page.

Depuis [SPIP 1.8.2], on peut utiliser un même fichier `.php3` pour tous les squelettes. Il s'agit du fichier **`page.php3`** qui est à la racine du site. Il est un peu plus complexe que celui décrit ci-dessus, mais il joue le même rôle. Lorsqu'on l'utilise, les variables **`$fond`** et **`$fond`** peuvent être passées en paramètres, par exemple :

```
http://monsite.org/page.php3?
```

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 d'uZine (visualisez le code source pour obtenir le texte du squelette).

Une interface différente dans le même site

Tout d'abord, notez qu'il est possible de créer des couples de fichiers pour le même élément logique (articles, rubriques, ...). Par exemple, vous pouvez créer des fichiers pour visiter un même article avec des interfaces différentes : `article.php3/html` pour le format normal, `imprimer.php3/html` pour le même article dans un format adapté à l'impression, `article-texte.php3/html` pour l'article dans un format texte (adapté aux mal-voyants par exemple), `article-lourd.php/html` avec une interface lourdingue adaptée au haut-débit, etc.

- **Une interface différente selon les rubriques.** Vous pouvez, pour un même type de document, créer des squelettes différents selon les rubriques du site. Il s'agit de créer simplement de nouveaux fichiers `.html` en fonction des rubriques (inutile, ici, de modifier le fichier `.php3`, on se contente de jouer sur les noms des fichiers squelettes).

Il suffit de compléter le nom du fichier squelette de « **-**numéro » (un tiret suivi d'un numéro de rubrique). Par exemple, si vous créez un fichier : `article-60.html`, tous articles contenus dans la rubrique n°60 utiliseront ce squelette (et non

plus le squelette par défaut `article.html`). Notez bien : le numéro indiqué est celui d'une *rubrique*. Si cette rubrique 60 contient des sous-rubriques, les articles contenus dans ces sous-rubriques utiliseront également le nouveau squelette `article-60.html`.

> *Remarque.* Dans notre exemple, on aura certainement également intérêt à créer un `rubrique-60.html`, voire un `breve-60.html`, etc. pour accompagner le changement de mise en page de cette rubrique.

- **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 créer un fichier : `article=60.html`, qui s'appliquera uniquement aux articles de la rubrique 60, mais pas à ses sous-rubriques.

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.php3/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 squelette des instructions en PHP, elles seront exécutées lors de la visite. Utilisée de manière assez fine, cette possibilité permet 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.

[1] Si `article.html` n'existe pas, le fichier « dist » est pris à la place. Lire à ce propos « Qu'est-ce que les fichiers « dist » ? ».

Des boucles et des balises

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

Syntaxe de base

La syntaxe simplifiée d'une boucle est la suivante :

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

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 :

```
<BOUCLEn(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, au choix, le nom de la boucle, ou le numéro de la boucle. Cet élément est 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 manoeuvre) d'utiliser plusieurs boucles dans un même squelette : leur donner un nom 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). Cet élément 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}... Ils 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 ».

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

Les différents critères et leur syntaxe seront explicités dans la suite, pour chaque type de boucle (certains critères fonctionnent pour tous les types de boucles, certains 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 :

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

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. 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 :

- `rubrique.php3?id_rubrique=15`

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

Clairement, avec le même code de squelette, si on appelle l'adresse :

- `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>
```

> 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>
```

La syntaxe des balises SPIP

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. A 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}>
<li> #TITRE
</BOUCLE_articles>
```

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

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

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}>
<li> #SURTITRE<br>
   #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 : « <LI>
 », c'est-à-dire une petite puce suivie d'une ligne blanche.

Ce que nous devons faire : n'afficher le code «
 » 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}>
<li> [(#SURTITRE)<br>]
    #TITRE
</BOUCLE_articles>
```

On obtient alors le résultat recherché : s'il existe un surtitre pour cet article, il est affiché et suivi du
 ; 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}>
[<li> [(#LOGO_ARTICLE)<br>]
    (#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
```

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
```

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|...|filtre*n*)** 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.

[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.

[2] **#ENV** et **#EXPOSER**

La boucle ARTICLES

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

```
<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} retourne 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} retourne la liste des articles contenus dans la rubrique `id_rubrique`.

{id_secteur} retourne 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).

[SPIP 1.4] **{branche}** : le critère `{branche}` retourne 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} retourne les articles correspondant à cet identifiant d'auteur (utile pour indiquer la liste des articles écrits par un auteur).

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

[SPIP 1.3] **{titre_mot=xxxx}**, ou **{type_mot=yyyy}** retourne les articles liés au mot-clé dont le nom est « xxxx », ou liés 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.

[SPIP 1.4] **{id_groupe=zzzz}** 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}`]

[SPIP 1.7.1] **{lang}** sélectionne les articles de la langue demandée dans l'adresse de la page.

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

{recherche} retourne 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 article « 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***}** [SPIP 1.8.2] sélectionne les articles en fonction de leur statut de publication :

- **{statut=prepa}** sont les articles en cours de rédaction dans l'espace privé ;
- **{statut=prop}** sont les articles proposés à la publication ;
- **{statut=publie}** sont les articles publiés sur le site, y compris les articles « post-datés » ;
- **{statut=refuse}** sont les articles qui ont été refusés à la publication ;
- **{statut=poubelle}** sont 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.

▸ **{exclus}** permet d'exclure du résultat l'article 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 trouver déjà).

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** retourne le surtitre.

- **#TITRE** retourne le titre de l'article.

- **#SOUSTITRE** retourne le soustitre.

- **#DESCRIPTIF** retourne le descriptif.

- **#CHAPO** retourne le texte d'introduction (chapeau).

- **#TEXTE** retourne le texte principal de l'article.

- **#PS** retourne le post-scriptum.

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

- **#ID_RUBRIQUE** est l'identifiant de la rubrique dont dépend l'article.

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

- **#NOM_SITE** et **#URL_SITE** correspondent aux données du « lien hypertexte » de l'article (si vous avez activé cette option).

- **#VISITES** est le nombre de visites sur cet article.

- **#POPULARITE** donne le pourcentage de popularité de cet article, voir la documentation La « popularité » des articles.

- **#LANG** donne 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.)

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

- **#INTRODUCTION** : [SPIP 1.4] si l'article contient un descriptif, c'est celui-ci qui est utilisé ici ; sinon, SPIP affiche les 600 premiers caractères du début de l'article (chapeau puis texte). [SPIP 1.3] Dans les versions précédentes de SPIP, ce sont systématiquement les premiers caractères de l'article (chapeau puis texte) qui sont pris en compte (le descriptif n'est pas utilisé).

- **#LESAUTEURS** les auteurs de cet article. Cela permet d'éviter de créer une boucle AUTEURS pour obtenir le même résultat.

- **#PETITION** 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).

- **#URL_ARTICLE** est l'URL de la page de l'article.

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

- **#FORMULAIRE_SIGNATURE** fabrique l'interface permettant de signer la pétition associée à cet article.

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

```
[<A HREF="forum.php3?(#PARAMETRES_FORUM)">Répondre à cet article</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="forum.php3?(#PARAMETRES_FORUM{#SELF})">Répondre à cet article</A>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

Les logos

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

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

- **#LOGO_RUBRIQUE** le logo de la rubrique de 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** est le logo sans survol ;
- **#LOGO_ARTICLE_SURVOL** est le logo de survol.

La boucle RUBRIQUES

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).

Les critères de sélection

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

▸ **{id_rubrique}** retourne 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}** retourne les rubriques de ce secteur. (On peut également, par extension, utiliser le critère `{branche}` décrit dans La boucle ARTICLES).

▸ **{id_parent}** retourne la liste des rubriques contenues dans une rubrique.

▸ **{racine}** retourne la liste des secteurs (rigoureusement identique à `{id_parent=0}`).

▸ **{id_enfant}** retourne 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}** retourne 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.

▸ **{recherche}** retourne 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.

▸ 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*.

▸ [SPIP 1.7.1] **{tout}** affiche 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).

▸ [SPIP 1.7.1] **{lang}** sélectionne les rubriques de la langue demandée dans l'adresse de la page.

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 lequel 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** retourne le titre de la rubrique.

- **#DESCRIPTIF** retourne le descriptif.

- **#TEXTE** retourne le texte principal de la rubrique.

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

- **#LANG** retourne 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** les notes de bas de page (calculées à partir de l'analyse du texte).

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

#URL_RUBRIQUE est l'URL de la page de la rubrique.

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

#FORMULAIRE_FORUM fabrique l'interface 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 :

```
[<A HREF="forum.php3?(#PARAMETRES_FORUM)">Répondre à cette rubrique</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="forum.php3?(#PARAMETRES_FORUM{#SELF})">Répondre à cette rubrique</A>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

#FORMULAIRE_SITE [SPIP 1.4] Le #FORMULAIRE_SITE affiche une interface 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)]
```

[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

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}`** retourne 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}`** retourne toutes les brèves contenues dans la rubrique en cours.

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

[SPIP 1.3] **{titre_mot=xxxx}**, ou **{type_mot=yyyy}** retourne 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.

[SPIP 1.4]**{id_groupe=zzzz}** 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} retourne 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 retourne le titre de la brève.

#DATE retourne la date de publication de la brève.

#TEXTE retourne le texte de la brève.

#NOM_SITE le nom du site indiqué en références.

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

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

#LANG donne 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 les notes de bas de page (calculées à partir de l'analyse du texte).

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

#URL_BREVE est l'URL de la page de la brève.

#FORMULAIRE_FORUM fabrique l'interface 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. Par exemple : `[<A HREF="forum.php3?(#PARAMETRES_FORUM)">Répondre à cette brève</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="forum.php3?(#PARAMETRES_FORUM{#SELF})">Répondre à cette brève</A>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

Le logo

#LOGO_BREVE 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 affiche, si il existe, le logo de la brève ; si ce logo n'a pas été attribué, SPIP affiche le logo de la rubrique [SPIP 1.4].

La boucle AUTEURS

La boucle AUTEURS, comme son nom l'indique, retourne une liste d'auteurs. Si l'on ne précise pas de critère de sélection, la boucle retournera tous les auteurs *ayant un article publié*.

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

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 [1]. 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** retourne le nom de l'auteur.

- **#BIO** retourne la biographie de l'auteur.

- **#EMAIL** retourne son adresse email.

- **#NOM_SITE** le nom de son site Web.

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

- **#PGP** sa clé publique pour PGP.

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

- #FORMULAIRE_ECRIRE_AUTEUR [SPIP 1.4] 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.

Les balises calculées par SPIP

- **#NOTES** les notes de bas de page (calculées à partir de l'analyse du texte).
- **#URL_AUTEUR** l'adresse de la page `auteur.php3?id_auteur=...`

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)]
```

[SPIP 1.6] : les variantes #LOGO_AUTEUR_NORMAL et #LOGO_AUTEUR_SURVOL permettent un affichage plus fin de ces deux variantes du logo.

[1] de la boucle englobante ou de l'url.

La boucle FORUMS

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.

{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.

- **#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 :

```
[<a href="forum.php3?
```

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="forum.php3?(#PARAMETRES_FORUM{#SELF})">Répondre à ce message</A>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

La boucle MOTS

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** est le titre (le mot-clé lui-même).

- **#DESCRIPTIF** est le descriptif du mot.

- **#TEXTE** est le texte associé au mot.

- **#TYPE** est 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** donne l'adresse de ce mot

La boucle (GROUPES_MOTS)

D'une utilisation marginale, la boucle **GROUPES_MOTS** [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
```

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 SITES (ou SYNDICATION)

La boucle `SITES` (SPIP 1.3) retourne une liste de sites référencés. (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 article de ces sites.)

```
<BOUCLEn(SITES){critères...}>
```

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

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

Les critères de sélection

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

- **{tout}**, tous les sites référencés.

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

- **{id_rubrique}** retourne les sites référencés dans cette rubrique.

- **{id_secteur}** retourne les sites référencés dans ce secteur.

- [SPIP 1.3] **{id_mot}** retourne toutes les sites liés au mot-clé en cours (à l'intérieur d'une boucle de type (MOTS)).

- [SPIP 1.3] **{titre_mot=xxxx}**, ou **{type_mot=yyyy}** retourne les sites liés au mot-clé dont le nom est « xxxx », ou liés 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.

- [SPIP 1.4]**{id_groupe=zzzz}** 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.

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

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

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 syndiqué.
- **#NOM_SITE** est le nom du site syndiqué.
- **#URL_SITE** est l'adresse (URL) du site syndiqué.
- **#DESCRIPTIF** est le descriptif du site syndiqué.
- **#ID_RUBRIQUE** est le numéro de la rubrique contenant cette syndication.
- **#ID_SECTEUR** est le numéro de la rubrique-secteur (à la racine du site) contenant cette syndication.
- **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 l'interface 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="forum.php3?(#PARAMETRES_FORUM)">Réagir à ce site.
</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="forum.php3?(#PARAMETRES_FORUM{#SELF})">Réagir à ce site.</A>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

La boucle DOCUMENTS

[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)
```

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_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)
```

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 ».

- `{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. Une utilisation intéressante de cette balise est combinée avec le filtre *reduire_image*, 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|
```

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

- `#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.

- #ID_DOCUMENT affiche le numéro du document.

- #EMBED_DOCUMENT est une balise à l'utilisation très spécifique : elle permet d'inclure directement les fichiers de formats autorisés (vidéo, sons) directement dans la page Web ; il faut éviter d'utiliser systématiquement cette balise, car il est déconseillé d'insérer systématiquement les documents dans les pages sans un contrôle strict (sauf à faire exploser la durée de chargement de vos pages Web...). La balise peut être complétée de paramètres propres aux formats utilisés (encore une fois : utilisation très spécifique), par exemple :

```
[(#EMBED_DOCUMENT|autostart=true)]
```

[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 SYNDIC_ARTICLES

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

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

(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** est 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** est l'adresse (URL) de l'article syndiqué (sur son site original).

- **#DATE** est la date de publication de cet article.
- **#LESAUTEURS**, les auteurs de l'article syndiqué.
- **#DESCRIPTIF** le descriptif de l'article syndiqué.
- **#NOM_SITE** est le nom du site syndiqué contenant cet article.
- **#URL_SITE** est l'adresse (URL) du site

La boucle SIGNATURES

La boucle `SIGNATURES` retourne une liste de signataires d'une pétition associée à un article.

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

Les critères de sélection

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

- **`{tout}`** toutes les signatures sont sélectionnés dans l'intégralité du site.
- **`{id_signature}`**, la signature correspondant à l'identifiant courant.
- **`{id_article}`** retourne les signatures de la pétition de cet article.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Attention. Dans ce type de boucles, certains critères de classement de la boucle 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 titre}`).

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

La boucle HIERARCHIE

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(xxx.php3)>`.)

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.

Les critères communs à toutes les boucles

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.

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}
```

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}
```

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

```
<BOUCLE_articles(ARTICLES){id_rubrique}
```

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).

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}>
```

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}
```

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
```

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.

- **`{`***`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'}`.

- **`{`***`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 :

```
petition.php3?id_article=13&debut_signatures=200
```

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

```
<BOUCLE_signatures(SIGNATURES)
```

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

```
petition.php3?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}>
```

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}{",
```

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

Les balises propres au site

Les balises suivantes sont disponibles à n'importe quel endroit du squelette, même en dehors d'une boucle.

Balises définies à la configuration

Le contenu de ces balises est défini lors de la configuration de votre site.

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

#NOM_SITE_SPIP est le nom du site.

#EMAIL_WEBMASTER [SPIP 1.5] est l'adresse du webmestre. Par défaut, SPIP prend l'adresse de celui qui a installé le site (le premier administrateur). (Si vous préférez un formulaire « écrire au webmestre », *cf.* « Les formulaires »).

#LOGO_SITE_SPIP à partir de [SPIP 1.8], est le logo du site. C'est en fait le logo

de la racine, c'est-à-dire la rubrique 0.

▸ **#CHARSET** [SPIP 1.5] est le jeu de caractères utilisé par le site. Sa valeur par défaut est `iso-8859-1`, jeu de caractères dit « iso-latin ». *Cf. www.uzine.net/article1785.html pour une introduction aux charsets, en attendant une documentation plus complète de cette fonctionnalité de SPIP.*

▸ **#LANG** [SPIP 1.7] utilisée en dehors des boucles ARTICLES, RUBRIQUES, BREVES et AUTEURS retourne la langue principale du site.

▸ **#LANG_DIR**, **#LANG_LEFT**, **#LANG_RIGHT** [SPIP 1.7] : 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**) [SPIP 1.7] 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

▸ **#DOSSIER_SQUELETTE** [SPIP 1.8.2] introduit cette balise 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"
```

▸ **#PUCE** [SPIP 1.5], qui affiche devinez-quoi ;

▸ **#FORMULAIRE_ADMIN** [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** indique à SPIP dans quelle partie de la page colorer les mots clefs recherchés.

Quand l'utilisateur arrive sur une page depuis une page de recherche SPIP, les mots clefs cherchés sont colorés automatiquement 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(menu.php3) {id_rubrique}>
#DEBUT_SURLIGNE
<h1>#TITRE</h1>
#CHAPO
#TEXTE
#NOTES
#FIN_SURLIGNE
</BOUCLE_article>
```

Balises techniques

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

- **#SELF** à partir de [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).

- **[(#ENV{*xxxx*,*zzzz*})]** à partir de [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.

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 ==
```

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`** introduite par [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.

Les formulaires

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{'merci.php3'})]` renverra vers la page *merci.php3*.
- `[(#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

Le #FORMULAIRE_SIGNATURE autorise la signature des pétitions associées aux articles (ce formulaire se place donc dans une boucle ARTICLES.

N.B. 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

[SPIP 1.4] Le #FORMULAIRE_SITE affiche une interface 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] Placé à l'intérieur d'une boucle AUTEURS, ce formulaire permet d'envoyer un mail à l'auteur (d'un article). Cela permet, en modifiant les squelettes (qui, par défaut, affichent les liens contenant les adresses email des auteurs des articles), de pouvoir écrire aux auteurs sans afficher 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 important, le #FORMULAIRE_INSCRIPTION gère l'inscription des nouveaux rédacteurs. Il n'affiche une interface d'inscription 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 ; donc ce formulaire ne fonctionne que si votre hébergeur autorise l'envoi de mails par PHP.

#LOGIN_PRIVE

[SPIP 1.4] Tout aussi important (sinon plus), le #LOGIN_PRIVE 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_login.php3`, c'est-à-dire en standard par le squelette nommé `login-dist.html`. En effet, lors des accès directs à l'adresse « /ecrire » de votre site, c'est vers `spip_login.php3` que SPIP va vous rediriger.

#LOGIN_PUBLIC

[SPIP 1.4] D'une utilisation beaucoup plus spécifique, #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|mapage.php3)]
```

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{sommaire.php3})]` renverra vers la page de sommaire.

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, <a
href='#URL_LOGOUT'>cliquez ici pour vous
déconnecter</a>

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

<?php } else { ?>

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

Feuilles de style

On peut notablement modifier l'interface graphique des formulaires par l'intérmédiaire des feuilles de style, notamment les classes **`forml`**, **`spip_encadrer`** et **`spip_bouton`**.

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

Les boucles de recherche

SPIP dispose d'un moteur de recherche intégré. Il faut donc prévoir une page permettant d'afficher les résultats des recherches.

L'interface de recherche

Pour afficher le formulaire de l'interface de recherche, il suffit d'insérer la balise :

#FORMULAIRE_RECHERCHE

Par défaut, le formulaire enverra les requêtes vers une page `recherche.php3` ; 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|adresse.php3**)]**

où `adresse.php3` est la page vers laquelle vous désirez envoyer l'utilisateur.

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. Vous pouvez en effet effectuer des recherches non seulement sur les articles, mais aussi sur les rubriques et les brèves.

La seule différence, par rapport à ce qui est documenté sur les pages de ces boucles, est le choix du critère de sélection, qui doit être **`{recherche}`**. Les critères d'affichage et les balises de ces boucles sont inchangées.

Cependant, afin de classer les résultats par pertinence, on utilisera de préférence ce nouveau critère d'affichage : **`{par points}`**.

Enfin, on pourra utiliser la balise **#POINTS**, qui indique la pertinence des résultats (attention, dans l'absolu cette valeur n'est pas très explicite, elle est surtout utile pour le classement des résultats).

Pour afficher la requête formulée par le visiteur, on peut utiliser la balise #RECHERCHE [SPIP 1.5.1].

Les filtres de SPIP

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|...|filtren) 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>]
```

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>]
```

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. Utilisation, par exemple, pour faire un descriptif META à partir du #DESCRIPTIF : `[<meta name='description' content='(#DESCRIPTIF|textebrut)'>]`

texte_backend peut être utiliser 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. La longueur par défaut est de 50 caractères. Mais on peut spécifier une autre longueur en passant un paramètre au filtre, par exemple : `[(#TEXTE|couper{80})]`.

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 paramètre :

- *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|>
```

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.php3 ;
- 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 `reduire_image` 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 ...>`.

- **|reduire_image{largeur, hauteur}** : SPIP 1.7.1 introduit le filtre |reduire_image, qui 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||reduire_image
```

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|reduire_image{600,0})]
```

affiche totues 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.

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','grey'})]` affichera « white » à la première itération de la boucle, « grey » à la deuxième, « white » à la troisième, « grey » 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','grey'})]">#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 *reduire_image* : `<div style="background: url([(#LOGO_ARTICLE||reduire_image {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. Si *valeur* n'est pas spécifié, le filtre retourne la valeur actuelle du paramètre. Par exemple, la balise #SELF retourne l'url de la page actuelle, donc :

- `[(#SELF|parametre_url{'id_article'})]` récupérera l'id_article présent dans l'url,
- `[(#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|reduire_image{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|reduire_image{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|
supprimer_tags|attribut_html)" ]>#TITRE</a>.
```

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.

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.php3` : si SPIP repère un fichier ayant ce nom, il l'inclut automatiquement.

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.php3` 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.php3 :

```
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.php3`, 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

Les boucles récursives sont une fonction très puissante pour gérer la mise en forme de l'interface. Leur programmation est particulièrement simple, mais leur utilisation demande une bonne maîtrise logique de l'enchaînement des boucles.

L'appel d'une boucle récursive est très simple : il suffit d'indiquer dans le TYPE de la boucle le nom d'une autre boucle :

```
<BOUCLEn(bouclex)></BOUCLEn>
```

Il n'y a ici aucun critère : en réalité, la boucle *n* correspond à une copie pure et simple de la boucle *x*. L'ensemble de la boucle 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*. (Il faut bien entendu que la boucle *x* précède la boucle *n*.)

L'utilisation la plus simple consiste à « dupliquer » une boucle sans avoir à la recopier. Ainsi, toute modification de la boucle d'origine *x* sera automatiquement dupliquée dans la boucle *n*.

Tout l'intérêt, en réalité, consiste à placer la boucle *n* à l'intérieur de la boucle *x* : on obtient ainsi 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.

Cette technique permet de créer notamment l'affichage des thread des forums. Cela devient très simple : 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}>
  <p>#TITRE
  <B_reponses>
  <UL>
  <BOUCLE_reponses(FORUMS){id_parent}
>
    <LI>#TITRE
    <BOUCLE_recursive(boucle_reponses)>
    </BOUCLE_recursive>
  </BOUCLE_reponses>
  </UL>
```

On peut ainsi, en très peu de lignes, provoquer l'affichage de l'intégralité de la structure (rubriques, sous-rubriques...) du site.

La « popularité » des articles

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.
```

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 %)</
```

(On enlèvera `{id_rubrique}` pour afficher un hit-parade du site entier.)

La gestion des dates

[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** [SPIP 1.5] est la date de dernière édition de l'article : précisément, il s'agit de la dernière date à laquelle cet article a été ouvert en édition *qu'il ait été modifié ou pas.* Pratique dans de nombreux cas, mais pas d'une rigueur scientifique... (Non modifiable, sauf si on veut la fixer à « maintenant » : il suffit alors... d'ouvrir l'article en édition.)

#DATE_NOUVEAUTES [SPIP 1.6] permet d'afficher la date du dernier envoi du mail présentant les nouveautés.

Les filtres `|annee`, `|mois`, `|jour`, `|heures`, `|minutes`, `|secondes`, mais aussi `|affdate`, `|nom_mois`, `|nom_jour`, `|saison`, etc. s'appliquent pour permettre tous les affichages habituels sous divers formats. (Sans ces filtres, les balises #DATE... s'affichent en effet au format MySQL : `"2001-12-01 03:25:02"`.)

Formater les dates

Une liste complète des filtres pouvant être appliqués aux dates pour les formater est fournie dans l'article Les filtres de SPIP.

Si les balises #DATE... sont utilisées sans filtres, alors toutes les informations de date sont affichées dans un format numérique [1] : « 2001-12-01 03:25:02 ».

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)
```

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 />"}>
```

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}>
<BR><UL><b> [(#DATE|nom_mois|
majuscules)] [(#DATE|annee)] </b>
   <li><a href="#URL_ARTICLE">[(#TITRE|
couper{50})]</a> - [(#DATE|jour)]/[(#DATE|
mois)]</li>
</BOUCLE_premierdumois>
```

[1] le format MySQL.

Exposer un article dans une liste

SPIP 1.8.2 introduit la balise **#EXPOSE**. Elle permet de mettre en évidence, dans un menu ou dans une liste, l'objet principal de la page où l'on se trouve.

Utilisée simplement, elle permet de changer le mode d'affichage du titre de l'article principal ; par exemple, dans le squelette `article.html`, on modifiera le style du lien de la manière suivante :

```
<BOUCLE_principale(ARTICLES){id_article}
>
	<BOUCLE_menu(ARTICLES)
{id_rubrique}>
	<li class='lien#EXPOSE'><a
href="#URL_ARTICLE">#TITRE<a></li>
	</BOUCLE_menu>
```

avec les styles suivants :

```
a.lien {
    color: red;
}
a.lienon {
    color: blue;
    font-weight: bold;
}
```

L'objet qui est ainsi « exposé » par un affichage différent est l'article, 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 sur l'article exposé, et ce qui doit s'afficher sur les autres articles. Ainsi `[(#EXPOSE{oui,non})]` affichera « oui » sur l'article exposé, et « non » sur les autres.

Avec un peu d'astuce il est possible de désactiver le lien sur l'article exposé et dans le même temps de choisir la feuille de style :

```
<BOUCLE_menu(ARTICLES){id_rubrique}
{"<br>"}>
<a [(#EXPOSE{id,href})]="#URL_ARTICLE"
```

créera le HTML suivant :

```
<a href="article1.html" class="lien">Tout sur
ma soeur</a><br>
<a id="article2.html" class="lien-
```

ce qui s'affiche ainsi :

Tout sur ma soeur
Tout sur moi
Tout sur mon frère

Historique :

Cette fonctionnalité a été introduite par SPIP 1.7.1 avec la balise **#EXPOSER**. La syntaxe complète de #EXPOSER était la suivante :

```
[(#EXPOSER|oui,non)]
```

Celle-ci est désormais obsolète. Il est donc conseillé d'utiliser **#EXPOSE**, dont la syntaxe complète est plus conforme au modèle général des balises de SPIP.

SPIP pas à pas

Pas à pas, comment créer un SPIP qui défie les limites.

Modifications récentes

- Le calendrier de SPIP 1.8.2
- Internationaliser les squelettes
- Principe général
- <INCLURE> d'autres

Le cache, ou comment faire un site dynamique qui ne bouge pas trop.

Des filtres

Subtilités squelettiques

Les filtres transforment le contenu de la base de données en code HTML présentable.

Mon premier squelette

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).

Avant d'utiliser un squelette, il faut pouvoir l'appeler : 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.

A la racine du site, déposez un fichier `tutoriel.html`, qui contient ce qui suit :

```
<BOUCLE_article(ARTICLES){id_article=1}>
#TITRE
</BOUCLE_article>
```

Puis rechargez la page `http://votresite.net/tutoriel.php3`. C'est mieux, n'est-ce pas ? 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>
<b>#CHAPO</b>
<div align="justify">#TEXTE</div>
</BOUCLE_article>
```

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

Bien !

Un squelette, plusieurs articles

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 « `http://votresite.net/tutoriel.php3?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/tutoriel.php3?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).

Cliquez maintenant sur :

- http://votresite.net/tutoriel.php3?id_article=1,
- http://votresite.net/tutoriel.php3?id_article=2 et
- http://votresite.net/tutoriel.php3.

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.

Bravo ! Votre squelette est maintenant « contextuel ».

[1] Non ? Il devrait...

Une rubrique

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<BR>
</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/tutoriel.php3` (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. A 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}>
<a href="#URL_ARTICLE">#TITRE</a><br>
```

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 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/tutoriel.php3?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 !

Boucles en boucles

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}>
<a href="#URL_ARTICLE">#TITRE</a><br>
</BOUCLE_article>
```

On appelle la page avec `http://votresite.net/tutoriel.php3?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 ?

Gérer le cache

Dans les leçons précédentes nous avons commencé à élaborer des squelettes. Le succès de notre site risque d'être fulgurant. Pensons tout de suite aux pauvres neurones de notre ordinateur. Dans cette leçon, rien d'amusant, rien d'essentiel non plus. Les flemmards en profiteront pour roupiller au fond près du radiateur...

Résumé pour ceux-ci et pour les gens pressés : dans les fichiers d'appel de type `tutoriel.php3`, *réglez* `$delais = 3600 ;` *au lieu de* `0`.

...

Au moment où une page est demandée à SPIP, celui-ci regarde si, par hasard, il n'aurait pas déjà calculé cette page auparavant. Si l'URL demandée est `http://votresite.net/tutoriel.php3?id_article=12`, SPIP regarde dans son sous-répertoire `CACHE/` si ce fichier existe, et, le cas échéant, compare l'âge du fichier caché aux `$delais` fixés dans le fichier d'appel `tutoriel.php3`.

Dans notre exemple nous avions fixé des `$delais=0 ;` - d'où un recalcul systématique des pages à chaque consultation du site. Passons à `$delais=3600 ;`

(c'est en secondes).

Notre page web n'est donc recalculée que si, lorsqu'un visiteur la demande, sa version cachée date de plus d'une heure (soit 3600 s.). Sinon, SPIP lit simplement le contenu du fichier caché [1], et renvoie le résultat sans se connecter à la base de données (sauf pour y insérer un « hit » dans les statistiques).

Comment fixer ces `$delais` de manière à optimiser le rapport réactivité/charge du serveur ? Pas de solution miracle, mais n'hésitez pas à fixer un délai d'une journée (i.e. `$delais=24*3600 ;`) ou plus pour les articles et les rubriques. Les pages de navigation les plus importantes peuvent avoir des `$delais` plus courts (vingt minutes ou une heure par exemple) si votre site est censé réagir à la validation fréquente de nouvelles brèves et de sites syndiqués... Si vous êtes sur un serveur partagé avec d'autres sites, soyez respectueux des autres et ne prenez pas tout le temps de calcul pour des pages qui changent rarement : ce serait d'autant plus idiot que, sur les gros articles ou sur les sommaires, le calcul des pages peut prendre quelques secondes, ce qui ralentit la consultation de vos pages...

Comment provoquer une mise à jour hors délai ? Nous venons de décider de `$delais` extrêmement longs, et nous repérons une fôte d'ortografe dans une page. Correction dans l'espace privé... Comment effacer tout de suite cette vilaine cicatrice du site ?

- Depuis l'espace privé, cliquer sur « Voir en ligne » déclenche le recalcul pour les pages correspondant à `#URL_ARTICLE` ou `#URL_RUBRIQUE` de l'article ou de la rubrique correspondante. C'est le cas le plus courant. Mais sinon ?

- Dans la partie « Sauvegarde/Restauration » de l'espace privé, un bouton « vider le cache » efface *tous* les fichiers cachés (utile si vous faites plein de modifications et avez un site très complexe, à éviter sinon).

- Toutefois, la solution la plus simple est de demander à SPIP, dans la page d'accueil de l'espace privé, de vous « poser un cookie d'administration ». Ce cookie s'incrustera sur votre navigateur, et SPIP vous reconnaîtra au moment de vous envoyer la page dans le site public : il vous proposera alors, en bas de page, un bouton « Recalculer cette page ».

Retour au contexte : On revient ici à la notion de contexte. Si le squelette est appelé avec un contexte d'`id_article`, d'`id_rubrique` ou encore d'`id_breve`, un autre bouton vous est proposé quand SPIP détecte le cookie : « Modifier cet article (ou rubrique, ou brève) », qui vous mène directement sur la page correspondante dans le back-office. Merci qui ?

Derniers détails :

- pour des raisons évidentes, le moteur de recherche ne déclenche pas de cache, et les pages avec forum sont réactualisées dès qu'une nouvelle contribution est envoyée.
- le répertoire `CACHE/` dans l'arborescence du site est découpé en 16 sous-répertoires numérotés `0, 1, 2... 9, A, B... F`, dans lesquels les fichiers cachés se distribuent quasi-aléatoirement ; cela s'appelle « *hasher le cache* » et rien que pour cela mérite bien qu'on le mentionne.
- les fichiers cachés sont exploités même si la base de données est « tombée », ce qui garantit le site contre des pannes transitoires du serveur mySQL.

[1] Pour les spécialistes, il s'agit en fait d'un *include* PHP du fichier correspondant, permettant d'exécuter du code depuis le cache...

Des filtres

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.

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

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 *Spip et les feuilles de style* pour plus de détails sur les styles CSS offerts par SPIP.

Guide des fonctions avancées

Au-delà du manuel de référence, vous trouverez ici une description détaillée des fonctions plus avancées à la disposition du webmestre.

Modifications récentes

- Le calendrier de SPIP 1.8.2
- Internationaliser les

Spip et les feuilles de style

SPIP 1.2 Lorsque l'on utilise les raccourcis typographiques dans les articles dans SPIP (permettant par exemple de mettre en gras, en italique, créer des liens hypertextes, des intertitres, etc.), SPIP produit les balises HTML nécessaires à ces effets, chacune de ces balises étant alors associée à une classe de style CSS.

Par exemple,

```
Ceci est un [lien->http://www.uzine.net]
```

est transformé en code HTML ainsi :

```
Ceci est un <a href="http://www.uzine.net"
class="spip_out">lien</a>
```

Le code HTML est ainsi complété par l'appel à un style CSS intitulé « spip_out ». L'utilisateur peut donc pousser la personnalisation de son interface graphique en définissant ce style « spip_out » (couleur différente, fond coloré, police utilisée...).

La plupart des raccourcis typographiques de SPIP peuvent ainsi être paramétrés avec des feuilles de style ; certains sont très utiles, d'autres seront réservés aux webmestres qui souhaitent obtenir des effets exotiques...

Où se trouve la définition de ces feuilles de style ?

Lors de l'installation de SPIP, avec les squelettes fournis en standard, la définition des feuilles de style se trouve dans le fichier :

- spip_style.css

Vous pouvez modifier ces styles (c'est même conseillé), 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. Vous pouvez aussi bien sûr intégrer directement des définitions de styles dans vos squelettes.

> *Notez bien : la notion de feuille de style, ou* cascading style sheets, *n'est pas une norme propre à SPIP, il s'agit d'un standard du Web. De très nombreuses documentations existent sur ce sujet par ailleurs ; consulter par exemple la page du W3C à ce sujet.*

Afin de suivre la suite de la présente explication, il est vivement conseillé d'ouvrir le fichier « spip_style.css » dans un éditeur de texte.

Les liens hypertextes

Les deux premières définitions permettent de modifier le comportement de « a » et « a :hover » ; très classiques, elles concernent tous les liens affichés sur votre page Web (afficher les liens sans soulignement, et régler le « survol » des liens hypertextes).

Viennent ensuite trois définitions propres aux raccourcis typographiques de SPIP : « `a.spip_in` », « `a.spip_out` », « `a.spip_url` ».

- **`a.spip_in`** concerne les liens à l'intérieur de votre propre site. Par exemple :

```
Ceci est un [lien interne->article1177]
```

- **`a.spip_out`** concerne les liens vers l'extérieur de votre site. Par exemple :

```
Ceci est un [lien externe->http://www.uzine.net]
```

- **`a.spip_url`** traite les adresses URL transformées en lien hypertexte. Par exemple :

```
[->http://www.uzine.net]
```

(ce raccourci affiche directement l'URL, avec un lien hypertexte vers cette adresse, ainsi : http://www.uzine.net).

Le principal intérêt de ces trois styles différents est de permettre de différencier graphiquement les liens internes au site et les liens vers d'autres sites.

Les intertitres

Les intertitres, créés par le raccourci suivant :

```
{{{Un intertitre}}}
```

peuvent être définis par le 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.

Par défaut, la définition en est :

```
h3.spip {
	font-family: Verdana,Arial,Helvetica,
sans-serif;
	font-weight: bold;
	font-size: 120%;
	text-align: center;
	margin-top: 2em;
	margin-bottom: 1.5em;
	padding: 0em;
```

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, il y aurait de fortes chances que l'intertitre serait soit trop « collé » au reste du texte, soit trop espacé (selon les goûts...).

Code et cadre

Les éléments de code, définis par le raccourci :

```
<code>Du code dans le texte</code>
```

sont paramétrés par le style **.spip_code**. Peu utilisé, 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...

Introduit dans SPIP 1.3, la balise `<cadre>...</cadre>` permet de présenter du code source dans un tableau (élément de formulaire) dans lequel il est facile copier-coller le texte. La feuille de style associée est : **.spip_cadre**, définie ainsi par défaut :

```
.spip_cadre {
	width : 100%;
	background-color: #FFFFFF;
	padding: 5px;
}
```

Les notes de bas de page

Les notes de bas de page, définies par le raccourci :

```
Le texte[[Une note de bas de page]]
```

sont paramétrés par le style **p.spip_note**. Souvent inutile, puisque les notes peuvent être modifiées directement en HTML lors de l'emploi de la balise #NOTES dans vos squelettes.

Les tableaux

Les tableaux sont définis dans SPIP de la façon suivante :

```
| {{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 donne :

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 feuilles de style permettent de paramétrer finement l'affichage de tels tableaux :

```
table.spip {
}

table.spip tr.row_first {
	background-color: #FCF4D0;
}

table.spip tr.row_odd {
	background-color: #C0C0C0;
}

table.spip tr.row_even {
	background-color: #F0F0F0;
}

table.spip td {
	padding: 1px;
	text-align: left;
	vertical-align: center;
```

- **`table.spip`** permet de modifier le comportement général du tableau (notamment sa position, à gauche, centré...) ;

- **`table.spip tr.row_first`** définit le comportement de la « première ligne » du tableau (ici en jaune) (pour que la « première ligne » soit prise en compte, il faut que les éléments qu'elle contient soient placés en gras) ;

- **`table.spip tr.row_odd`** pour les lignes impaires ;

- **`table.spip tr.row_even`** pour les lignes paires ;

- **`table.spip td`** permet de modifier le comportement des cases du tableau.

Un des intérêts repose sur le choix de couleurs différentes pour « row_odd » et « row_even », permettant de faire une présentation de couleurs alternées pour les lignes d'un tableau (ici, gris clair et gris foncé).

Ligne de séparation horizontale

Une ligne de séparation horizontale, définie par :

```
----
```

peut être modifiée par : **hr.spip**.

Gras et italique

Le gras et l'italique sont définis par les raccourcis :

```
Du texte {{en gras}}, du texte {en italique}
```

Ils peuvent être modifiés par les styles : **strong.spip** et **i.spip**. Styles peu utiles.

> *historique :* dans les versions antèrieures à [SPIP 1.8], le texte **en gras** est déclaré par le style b.spip>.

Les paragraphes

Les paragraphes créés par SPIP (en laissant des lignes vides entre les paragraphes) peuvent être modifiés par le style : **p.spip**.

A priori, peu utile, car on peut directement paramétrer le comportement des éléments de texte en HTML.

Les formulaires

Dans l'espace public, différents formulaires sont utilisés pour le moteur de recherche interne, l'interface de rédaction des messages des forums, les inscriptions à l'espace privé...

Les feuilles de style sont : **.forml**, **.spip_encadrer**, **.spip_bouton**, **.formrecherche**.

Par défaut, ils sont définis ainsi :

```
.forml { width: 100% ; background-color:
#FFDDAA;}
.spip_encadrer { background-color:
#EEEEEE; }
```

- **.forml** définit les « cases » de texte des formulaires ; utile pour définir la largeur de ces cases, et la couleur du fond ;

- **.spip_encadrer** ; lorsqu'un formulaire propose différentes « parties », la séparation entre ces différentes parties peut être paramétrées avec ce style (par exemple, encadrer chaque partie, créer un espace avant ou après...) ;

- **.spip_bouton** modifie l'aspect du bouton de validation du formulaire ;

- **.formrecherche** modifie l'aspect de la case « Rechercher » du moteur de recherche.

Les images et les documents

Depuis [SPIP 1.8], le style des images et des documents insérés automatiquement avec les raccourcis `<docXX|left>` et `<imgXX|right>` peut être controlé avec les classes :
- **.spip_documents** pour la boîte qui contient la vignette et les informations du document,
- **.spip_doc_titre** qui contrôle l'affichage du titre du document,
- **.spip_doc_descriptif** pour le descriptif du document.

Conclusion

Vous remarquerez que, par défaut, certaines feuille de style ne sont pas définies. Elles peuvent être considérées comme très accessoires (réservées aux webmestres voulant obtenir des effets graphiques très spécifiques).

En règle générale, les styles qui provoquent des modifications graphiques spectaculaires sur un site, par ailleurs simples à paramétrer, sont celles qui concernent :
- les liens de l'ensemble de la page, **a** et **a:hover**,
- le comportement des intertitres, **h3.spip**,
- les formulaires.

<INCLURE> d'autres squelettes

[SPIP 1.4] 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.

> *Les habitués de PHP connaissent la fonction `include`, dont le principe est similaire à ce qui est présenté ici.*

Dans SPIP, 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}>
```

Pour une réflexion plus large sur ce qu'apporte cette nouvelle possibilité d'appel de squelettes ainsi que la balise `#DOSSIER_SQUELETTE`, on pourra consulter cet article de Spip Contrib.

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 : « `hierarchie.php3?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 installera le code suivant à l'intérieur de cette boucle :

```
<INCLURE(hierarchie.php3){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(hierarchie.php3){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 multilingüisme de SPIP est activé depuis SPIP 1.7.1 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 (pied.php3)>`, le squelette inclus est appelé en utilisant la langue par défaut du site,

- `<INCLURE(pied.php3){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),

- et `<INCLURE(pied.php3){lang}>` appelle le squelette dans la langue courante du contexte d'inclusion.

Il convient de noter que cela rend possible d'utiliser des codes de fichiers de langue dans les squelettes inclus (voir Internationaliser les squelettes).

Les squelettes inclus supportent les mêmes mechanismes 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 details.

[1] Rappelons que la variable `$delais` définit la périodicité de mise à jour du cache. Voir la section « Le fichier .php3 » dans « Principe général ».

Réaliser un site multilingue

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 (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 rubrique de la racine du site correspond une langue modifiable par les administrateurs, qui concerne toutes ses sous-rubriques ainsi que les articles et les brèves y 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 nouveau raccourci de SPIP, dont la structure est relativement intuitive :

```
<multi>chaîne 1 [xx] chaîne 2 [yy] chaîne 3 ...
```

Pour reprendre l'exemple du mot-clé, son titre serait entré sous la forme :

```
<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 :

```
<BOUCLE_sommaire(ARTICLES){par date}
{inverse}{0,10}>
  <li dir="#LANG_DIR">[(#DATE|affdate)]:
  <a href="#URL_ARTICLE">#TITRE</a></li>
```

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 donner la langue de l'élément demandé, et la direction générale de la page :

```
<html lang="#LANG">
<body dir="#LANG_DIR">
...
</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}>
[<a href="#URL_ARTICLE"
```

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` »...).

- 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/spikini....

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, 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
```

```
<BOUCLE_recherche(ARTICLES){lang?}
```

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

[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

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 :

```
<a href="#URL_DOCUMENT"><:
```

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 :

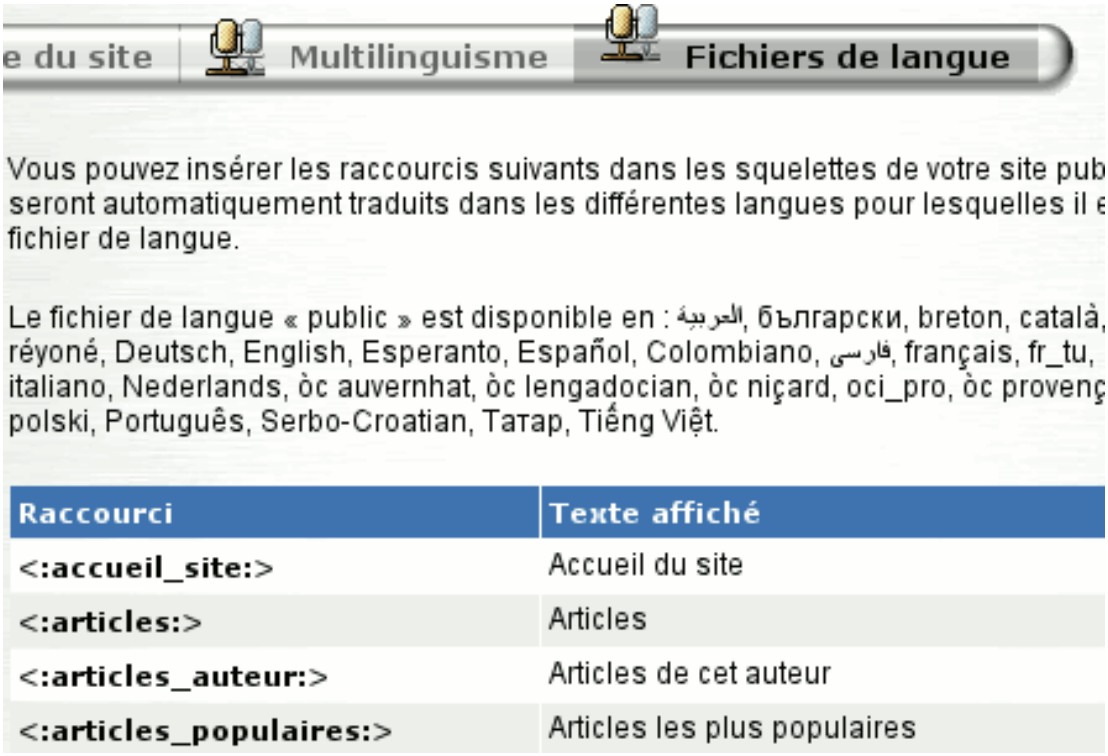
```
<a href="/IMG/jpg/mondocument.jpg">download this file</a>
```

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.php3` contient les chaînes en français,
— `/ecrire/lang/public_en.php3` 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.php3` pour définir les chaînes en français,
— `local_en.php3` 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_...php3`.

> 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' => 'T&eacute;l&eacute;charger la derni&egrave;re version',
	'quoideneuf' => 'Modifications r&eacute;centes'
```

Version catalane :

```
<?php
$GLOBALS[$GLOBALS['idx_lang']] = array(
	'telechargement' => 'Descarregar la
darrera versi&oacute;',
	'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 » soit 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...php3`) 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.

Utiliser des URLs personnalisées

Après l'installation, les pages générées par SPIP utilisent des adresses relatives ressemblant à `article.php3?id_article=123`, donnant des URLs du type `http://www.spip.net/article.php3?id_article=123`.

Ce type de syntaxe, courant chez les sites « dynamiques », n'est cependant pas très joli ni très évocateur. 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/article.php3?id_article=765` ; 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.php3` [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 = "standard";
```

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 `inc-urls-standard.php3`.

Vous remarquerez que SPIP propose aussi les fichiers `inc-urls-html.php3`, `inc-urls-propres.php3` et `inc-urls-propres2.php3`.

- Le fichier `inc-urls-html.php3` permet de traiter des adresses du type (« article123.html »). Vous pouvez décider d'utiliser les « URLs "html" » en mettant dans `ecrire/mes_options.php3` la ligne :
  `$type_urls = "html";`

- Le fichier `inc-urls-propres.php3` permet de traiter des adresses du type (« Titre-de-l-article »). Il faut alors ajouter :
  `$type_urls = "propres";`

- Le fichier `inc-urls-propres2.php3` est une variation du précédent, qui donne des adresses du type (« Titre-de-l-article.html »). Il faut alors ajouter :
  `$type_urls = "propres2";`

> 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 : `inc-urls-XXX.php3`. Il est par exemple très aisé de modifier la fonction `_generer_url_propre()` dans `inc-urls-propres.php3` 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

article.php3 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, [SPIP 1.8.1] fournit 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 1.8.1], 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 facilliter 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.

[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...

Le moteur de recherche

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).

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.

[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 variables de personnalisation

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.php3`, placé à la racine du site. (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 **`$puce`**. Lorsque vous commencez une nouvelle ligne par un tiret, SPIP le remplace par une petite « puce » graphique. Cette puce est constituée par le fichier `puce.gif` installé à la racine du site ; vous pouvez modifier ce fichier selon vos besoins. Mais vous pouvez aussi décider de fixer vous-même le choix de la puce, au travers de la variable `$puce`. Par exemple pour indiquer un autre fichier graphique :

```
$puce = "<img src='mapuce.gif' alt='-'
```

ou par un élément HTML non graphique :

```
$puce = "<li>";
```

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

[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 dans `mes_fonctions.php3` la variable **`$dossier_squelettes`**.

```
<?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(xxx.php3)>`, SPIP ira chercher le fichier `xxx.php3` 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 ».

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.php3`.

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;
```

Tidy : validation XHTML 1.0

[SPIP 1.8.1] permet aux webmestres qui le désirent d'utiliser sur leurs pages l'outil Tidy, introduisant ainsi une fonction de validation XHTML sur leur site.

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. SPIP exploite cet outil pour permettre aux webmestres de proposer des sites conformes aux recommandations du XHTML 1.0 transitional.

> **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. L'outil est donc intégré dans SPIP pour, à la fois, créer du code conforme, et permettre de « traquer » les erreurs de code dans vos pages.

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* ») ; notez bien : SPIP vise la conformité *HTML 4.01 transitional* ;
— les squelettes du site public doivent eux-mêmes être aussi conformes que possible (pour rester cohérent, on visera la conformité HTML 4.01).

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 au XHTML 1.0 transitional.

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, dont on peut d'ailleurs faire valider la conformité par le W3C validator ; 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 que SPIP ne se contente pas de se reposer sur un outil (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, avec Tidy, 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/info.php3` de son site pour obtenir la configuration de son serveur et la liste des extensions de PHP

disponibles.

N.B. : À l'heure actuelle, l'utilisation de Tidy en tant qu'extension de PHP n'a pas pu être testée en situation de production dans SPIP. *En théorie*, cela fonctionne, mais 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 `/ecrire/mes_options.php3` 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.php3` et non à la racine du site dans `/mes_fonctions.php3`. 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 (eux-mêmes déjà conformes HTML 4) 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 au Validator en visant le conformité HTML 4.01 transitional (SPIP visant, dans ses traitements typographiques, cette conformité, autant rester cohérent). Le W3C Validator est un outil très pratique pour nettoyer son code.

Une fois vos squelettes aussi proches que possible du HTML 4, Tidy n'aura pas de difficulté à produire du XHTML très compliant. Si ces pages sont complètement conformes, c'est encore mieux ! (Et pas impossible : les squelettes livrés avec SPIP sont déjà conformes).

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).

Encore une fois, ces fonctionnalités sont toutes récentes, et demandent sans doute des tests supplémentaires. N'hésitez pas à faire part de votre expérience sur spip-dev.

[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.

Le support LDAP

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.

Le traitement des images

[SPIP 1.8] 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 `reduire_image` 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...

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/info.php3`, 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.

Insérer des formules mathématiques en LaTEX

[SPIP 1.8] introduit une puissante fonctionnalité permettant d'insérer dans les textes des formules de mathématique complexes, en utilisant la syntaxe de TEX/ LaTEX.

Cette fonctionnalité permet par exemple d'afficher une formule comme celle-ci :

$$z = \left(\frac{e^{i\theta} + e^{-i\theta}}{2}\right)^2 + \left(\frac{e^{i\theta} - e^{-i\theta}}{2i}\right)^2$$

en la codant directement dans le texte, comme on peut le faire avec TEX.

> **Notez bien :** l'utilisation de cette méthode nécessite, évidemment, de connaître la syntaxe des formules dans TEX. Syntaxe qui n'est pas simple...

Des images dans le texte

Le principe technique de cette méthode consiste à transformer chacune des formules en *image*, image qui est ensuite affichée dans le texte. C'est actuellement la méthode la plus simple et efficace pour afficher des formules mathématiques complexes sur une page Web.

> Pour l'heure, l'affichage de formules mathématiques sur des pages Web grâce au standard MathML n'est absolument pas viable, l'implémentation de MathML dans les navigateurs étant totalement erratique. La solution retenue par SPIP (l'intégration d'images représentant les formules) est actuellement la seule qui garanti que *tous* les visiteurs d'un site verront correctement les formules mathématiques.

Il est important de comprendre que, dans SPIP, *seules les formules* sont transformées. Il n'est pas question, en particulier, d'utiliser les macro-fonctions de T_EX pour réaliser la mise en forme du document. Il s'agit ici d'un outil destiné à intégrer des formules mathématiques à l'intérieur d'un document codé selon les habitudes de SPIP.

Syntaxe dans SPIP

La syntaxe dans SPIP consiste à placer la partie de texte concernée par le traitement entre les pseudo-tags suivants :

```
<math>
    ...
    Ici  on extrait les formules mathématiques...
    ...
</math>
```

> Puisque seules les formules mathématiques sont traitées, on peut en réalité ajouter `<math>...</math>` de manière très large (en clair : on peut ajouter `<math>` tout au début du texte, et `</math>` tout à la fin...).
>
> La seule incompatibilité sera le cas où l'on souhaite afficher le symbole « dollar » ($) dans le texte, ce symbole étant utilisé pour délimiter les formules. (C'est la raison, en réalité, de l'existence des codes `<math></math>`.)

À l'intérieur de ces pseudo-tags, on code ensuite les formules de mathématiques selon les normes de TEX, en les encadrant de dollars ($) ou de double-dollars ($$) pour les formules centrées.

Voici un exemple :

On peut insérer des matrices : $\begin{pmatrix} 1 & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & b \end{pmatrix}$; on peut placer des fractions, telles que $\frac{1}{z}$, $\frac{1}{\displaystyle 1+\frac{1}{x}}$; utiliser des lettres grecques α, β, γ, Γ, φ ; centrer des formules complexes :

$$\left|\frac{1}{N}\sum_{n=1}^{N}\gamma(u_n) - \frac{1}{2\pi}\int_0^{2\pi}\gamma(t)\mathrm{d}t\right| \leq \frac{\varepsilon}{3}.$$

que l'on pourra coder ainsi :

```
On peut insérer des matrices:
$\pmatrix{1&0&0\cr 0&a&0\cr 0&0&b\cr}$;
on peut placer des fractions, telles que ${1
\over z}$,
${1\over\displaystyle 1+{1\over x}}$;
utiliser des lettres grecques $\alpha$, $\beta
$,
$\gamma$, $\Gamma$, $\varphi$;
centrer des formules complexes:
$$\left|{1\over N}\sum_{n=1}^N\gamma(u_n)
```

Le système est limité à l'affichage de formules mathématiques. De ce fait, toutes les autres fonctions de T_EX sont désactivées. Parmi la plus importante interdiction : *il n'est pas possible de définir ses propres macros* (`\def...{...}` est désactivé) *et les macros utilisées en dehors des formules mathématiques ne seront pas reconnues*. À l'usage, on trouvera d'autres limitations, en se souvenant toujours que le but est d'intégrer des formules mathématiques dans ses textes, et rien de plus...

Pour les webmestres

Le traitement des équations se fait sur le mode client-serveur : les formules sont envoyées à un serveur centralisé, qui retourne à votre site les fichiers graphiques de ces équations. (Évidemment, les fichiers sont sauvegardés sur votre système, et l'échange n'a lieu qu'une seule fois par équation.)

Pour plus d'informations sur le système utilisé, vous pouvez consulter la page du Wiki de SPIP. Vous y trouverez notamment les explications pour monter votre propre serveur d'équations, afin de ne plus dépendre de notre serveur central.

SPIP 1.8 : l'interface graphique

L'interface graphique de l'espace privé a été, dans la version 1.8, largement remaniée. Au délà de l'aspect purement graphique de l'interface (qu'il est inutile de détailler), nous présentons ci-dessous quelques-uns des éléments d'*ergonomie* introduits, destinés à faciliter le travail à la fois des utilisateurs débutants et des utilisateurs confirmés.

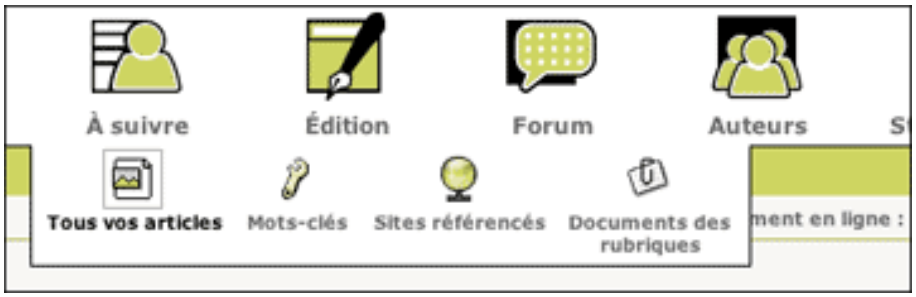
« The "Un-seul-click"™ interface for the SPIP-user »

L'une des principales tâches effectuées sur cette version a consisté à tenter de réduire le nombre de clics nécessaires pour accéder aux différentes pages. Il est destiné à la fois aux utilisateurs confirmés (ils « gagnent » du temps) mais aussi aux utilisateurs débutants en rendant visible toutes les informations nécessaires (telle page n'est pas « cachée » comme sous-option d'une autre page, elle est accessible directement).

Évidemment, on a tenté ici, selon l'habitude de SPIP, de ne pas rendre plus confuse l'interface pour les débutants en voulant faciliter le travail des utilisateurs confirmés.

La modification la plus visible par rapport aux versions précédentes (depuis la version 1.4) est la disparition du second bandeau de navigation. Désormais tous les éléments de ce « second bandeau » apparaissent en « popup » par survol des

éléments principaux (eux-mêmes réorganisés selon des principes simplifiés — comme un retour à l'interface de SPIP 1.0 !). Ainsi chacune des pages de ce « second bandeau » devient accessible en un clic depuis toutes les pages de l'espace privé.

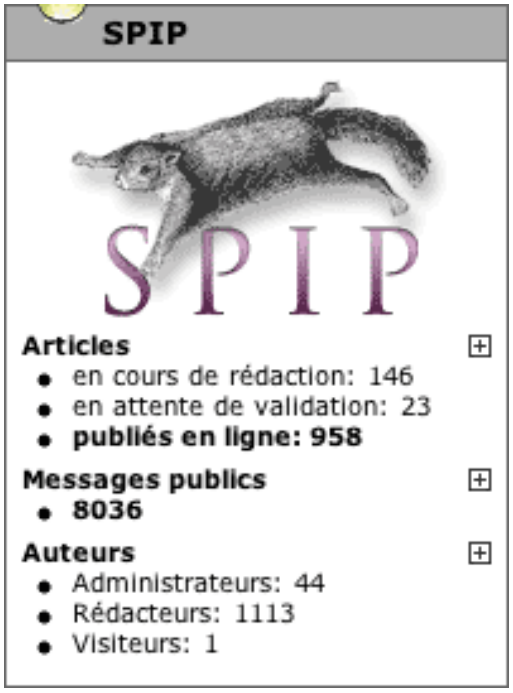


Selon la même logique, le « bandeau coloré » accueille, outre les options d'interface (couleur, taille de l'écran, etc.), quelques popups récapitulant en permanence certaines informations utiles :

- une navigation dans l'intégralité de la hiérarchie des rubriques, par un système de menus déroulants,
- un rappel des articles en cours de rédaction et proposés à la publication, ainsi qu'un des boutons « créer un article, une brève... »,
- l'interface de recherche,
- les raccourcis liés au calendrier interne et à la messagerie.

À différents endroits de l'interface (notamment sur la page « À suivre »), on pourra repérer des petits boutons « Plus ». Destinés aux utilisateurs confirmés (l'absence de descriptif explicite de ces boutons faisant qu'on en découvre l'usage au fur et à mesure), ils envoient à des pages spécifiques. Par exemple, à côté d'une liste d'articles, ce bouton mène à la page « Tous vos articles » ; à côté d'une liste

de brèves, le bouton mène à la page de gestion des brèves, etc.



Changements de statut dans les listes

Une interface de changement de statut à l'intérieur même des listes (d'articles, notamment) permet aux administrateurs de modifier le statut d'une série d'éléments sans changer de page. Cette manipulation, immédiate, permet de changer, par exemple, le statut de tous les articles d'une même rubrique (tous les publier, tous les dépublier) sans changer de page.

Des vignettes et des rotations

Si vous disposez de fonctions de traitement d'images sur votre serveur, SPIP peut en tirer partie pour « illustrer » graphique l'espace privé :

- vignettes des logos dans les listes d'articles, brèves, sous-rubriques,
- vignettes des auteurs dans les forums internes.

Ces fonctions graphiques rendent l'interface plus agréable, mais aussi plus lisible (les éléments des listes étant ainsi mieux différenciés graphiquement par l'affichage d'un petit logo).

Ces fonctions permettent aussi d'effectuer des rotations de photographies dans la nouvelle interface de portfolio.

Le calendrier

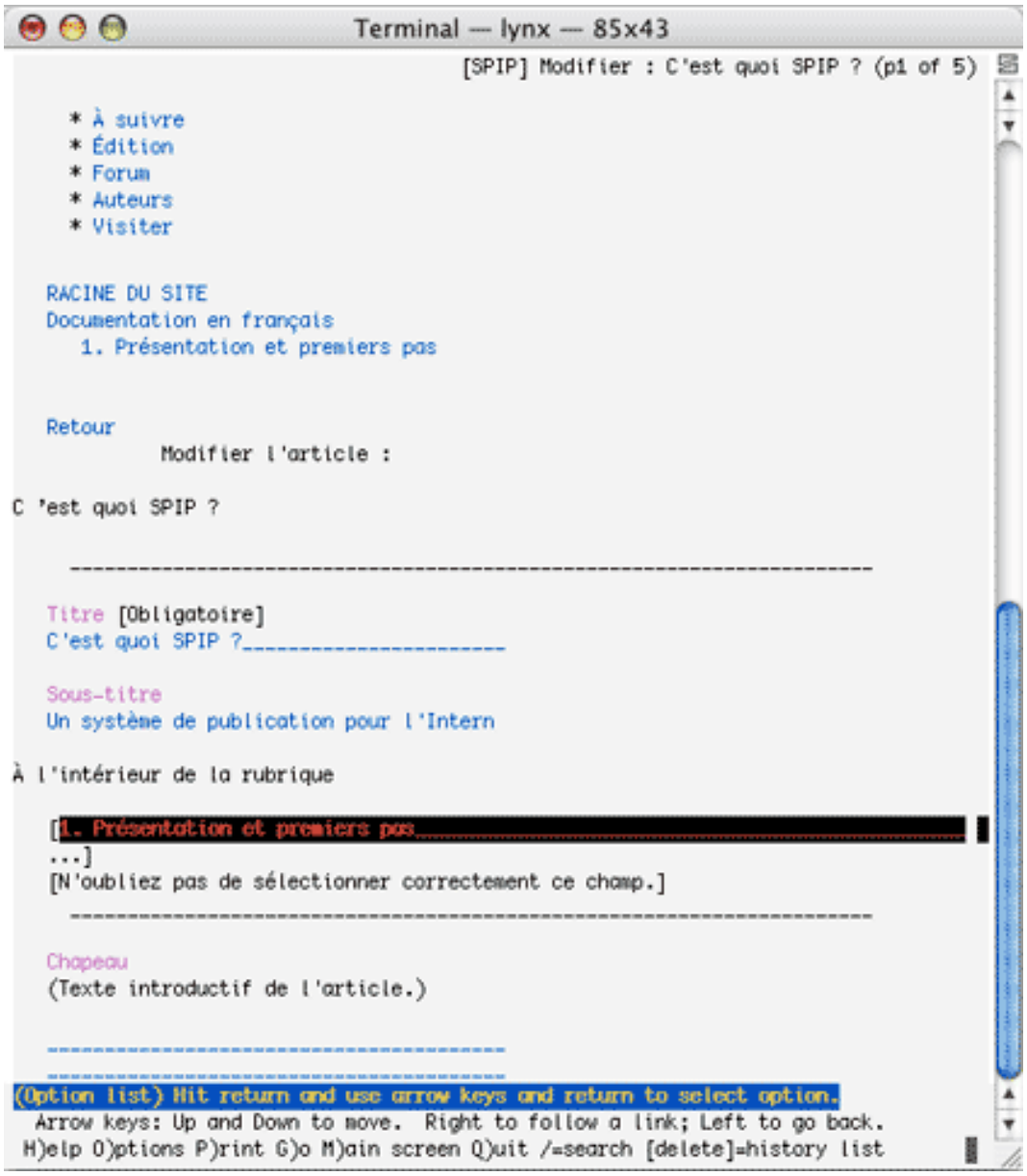
Outre les changements graphiques de l'interface du calendrier interne, on remarquera l'apparition de cinq petits boutons de réglage ergonomique :

- les petites horloges permettent de développer plus ou moins l'affichage de la matinée ou de l'après-midi ;
- des loupes « plus » et « moins » permettent de zoomer sur cet affichage.

Interface purement textuelle

SPIP 1.8 introduit une version nouvelle de l'interface, destinée aux malvoyants. Par rapport à la version « texte » de l'interface (introduite dans SPIP 1.4), cette nouvelle interface est nettement « allégée » (l'interface « texte » de l'interface, elle, convient aux connexions relativement lentes, mais pour une lecture à l'écran par un utilisateur voyant).

On peut accéder à cette interface par l'adresse `/ecrire/oo`, ou bien en suivant le lien « Afficher l'interface textuelle simplifiée » qui apparaît sur les butineurs ne comprenant pas Javascript.



Il est possible de revenir à l'interface habituelle en suivant le lien proposé en bas de page (« Retour à l'interface graphique complète »). Notez qu'alors vous revenez en interface dite « simplifiée ».

> L'équipe des développeurs (liste de discussion `spip-dev`) est très intéressée par des retours d'expérience de la part d'utilisateurs concernés par cette interface. Nous manquons cruellement de témoignages pour améliorer le système en fonction de tels utilisateurs.

On notera que cette interface est également particulièrement adaptée aux connexions sur supports mobiles dotés de connexions *très* lentes et/ou d'écrans de petite taille (smartphones, PDA communiquants).

Espaces plus grands (ou pas...)

La zone « Texte » du formulaire de modification d'articles s'adapte pour occuper toute la hauteur de la fenêtre. Ce qui, dès qu'on utilise une définition d'écran supérieure à 800x600, permet d'agrandir notablement l'espace pour saisir son texte.

Selon le même principe, les fenêtres de « Navigation rapide » occupent toute la hauteur de l'écran.

Formulaires dynamiques

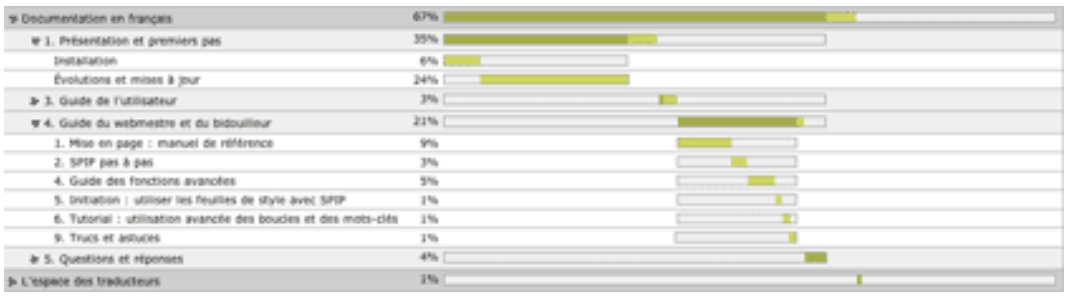
De nombreux formulaires deviennent dynamiques : des boutons de validation, des cases de sélection, ainsi que des zones de texte, apparaissent ou disparaissent en fonction des choix des utilisateurs.

Ainsi, dans l'affichage d'un article, certains boutons de « Validation » n'apparaissent que lorsqu'on modifie le menu déroulant attenant. On pourra également voir de tels formulaires dans l'espace de configuration du site (par exemple, dans la zone « Référencement de sites », selon qu'on sélectionne « Gérer un annuaire de sites Web » ou « Déselectionner », et « Utiliser la syndication » ou non, on voit les options de configuration changer immédiatement).

Statistiques

Les pages d'affichage des statistiques ne changent pas dans leur principe. On notera l'apparition de boutons de « zoom » pour l'évolution des statistiques.

Les pages des « répartitions de visites » devient nettement plus compréhensible !



Pour les webmestres...

Les adeptes de HTML pourront se demander comment les grandes icones de navigation de l'espace privé changent de couleur selon le choix de l'utilisateur. Cet effet est réalisé en utilisant des icones au format PNG 24, et en profitant de la transparence fine autorisée par ce format (couche alpha).

Or, il est habituellement expliqué que Microsoft Explorer est incapable d'exploiter la couche alpha du PNG24. L'espace privé de SPIP utilise une astuce permettant cependant de faire utiliser cette couche alpha par Microsoft Explorer. Les webmestres intéressés pourront avantageusement l'exploiter pour la partie publique de leur site, lors de la création de leurs propres squelettes...

Il convient de créer une classe de style, classe que l'on attribuera aux images concernées (notez bien : on peut appliquer sans risque cette classe à des images dans d'autres formats que le PNG, la classe sera alors simplement inutile, mais sans déterioration de ces images).

Cette classe pourra se définir ainsi :

```
.format_png {
        behavior: url("win_png.htc");
}
```

qu'on applique dans les balises `<img src="..." class="format_png">`.

On constate que cette classe utilise une caractéristique de Microsoft Explorer, l'appel d'un *behavior*. Celui-ci est un fichier Javascript, présent dans l'espace privé (`/ecrire/win_png.htc`), que l'on recopie à la racine du site pour l'exploiter en dehors de l'espace privé.

En ouvrant ce fichier de comportement (*behavior*), on verra qu'il fait appel à un fichier `/img_pack/rien.gif` dans son fonctionnement. On doit donc recopier ce fichier à la racine du site (depuis `/ecrire/img_pack/rien.gif`), et remplacer le chemin dans la version de `win_png.htc` installée à la racine du site.

Vous n'aimez pas le vert ?

Les goûts et les couleurs...

Les couleurs de l'interface peuvent être complétées d'autres couleurs, ou remplacées. Pour cela, il suffit d'installer des variables dans un fichier `/ecrire/mes_options.php3` :

```
global $couleurs_spip;

$couleurs_spip[1] = array (
        "couleur_foncee" => "#508A72",
        "couleur_claire" => "#A5DFC7",
        "couleur_lien" => "#657701",
        "couleur_lien_off" => "#A6C113"
);


$couleurs_spip[] = array (
        "couleur_foncee" => "#949064",
        "couleur_claire" => "#DFDBA5",
        "couleur_lien" => "#657701",
        "couleur_lien_off" => "#A6C113"
```

Ici, on a remplacé le vert (variable « 1 » ; les couleurs de l'installation normale sont numérotées de 1 à 6, on peut donc toutes les modifier), et ajouté une nouvelle couleur (`$couleurs_spip[]` indiquant qu'on ajoute une valeur au tableau (*array*) `$couleurs_spip`.

Le calendrier de SPIP 1.8.2

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>
<ul>
<BOUCLE_semaine(ARTICLES) {agenda
date, semaine} {par date}>
<li>#TITRE</li>
</BOUCLE_semaine>
</ul>
```

affiche une liste des articles publiés dans la semaine actuelle.

```
<BOUCLE_art_principale(ARTICLES)
{id_article}>
<B_mememois>
<ul>
<BOUCLE_mememois(ARTICLES) {agenda
date, mois, (#DATE|annee), (#DATE|mois)}
{par date}>
<li>#TITRE</li>
</BOUCLE_mememois>
```

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>
<ul>
<BOUCLE_apres(ARTICLES) {agenda date,
periode, (#DATE|annee), (#DATE|mois),
(#DATE|jour)} {par date}>
<li>#TITRE</li>
</BOUCLE_apres>
```

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>
```

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.

Images typographiques

[SPIP 1.9 et GD2]

Si GD2 est installé 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.

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.
```

(**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 `/ecrire`, ou du dossier contenant vos squelettes.

Si nous installons, par exemple, un fichier TrueType ainsi :
`/ecrire/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,
```

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,
```

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})]
```

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,
```

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,
```

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,
```

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})]
```

[1] GD2 est une extension *graphique* de PHP, qui permet de nombreuses manipulations d'images. En cas de doute, demandez à votre hébergeur si GD2 est installé.

[2] Attention : si vous ne protégez pas ce dossier, 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.

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.

Couleurs automatiques

[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
```

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 :

```
<div style="background-color: #
```

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
```

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)]
```

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 :

```
<div style="background-color: #
```

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|
```

— 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: #
```

— 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

[SPIP 1.9 et GD2]

SPIP permet 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, pour interface graphique, 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 travaillerait autrement : les auteurs

installeraient un simple logo d'article (par exemple, une photographie), sans aucun traitement spécifique ; et, automatiquement, les squelettes de SPIP fabriqueraient :
— une vignette à la bonne taille ;
— la même vignette en noir et blanc, légèrement foncée ;
— les reflets de l'image sur le sol.

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, même interrompu, le script aura certainement calculé quelques images

avant de s'arrêter, et ces images seront sauvegardées en cache. Ainsi, quand les pages seront recalculées, les images seront directement réutilisées depuis le cache et on n'aura plus d'erreurs.

Cet avertissement concerne, en priorité, les hébergeurs mutualisés.

Transparences

Dans [SPIP 1.9], si l'on utilise GD2, outre les fonctions exposées dans cet article, vous constaterez que les réductions d'image (`reduire_image`) 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.

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.

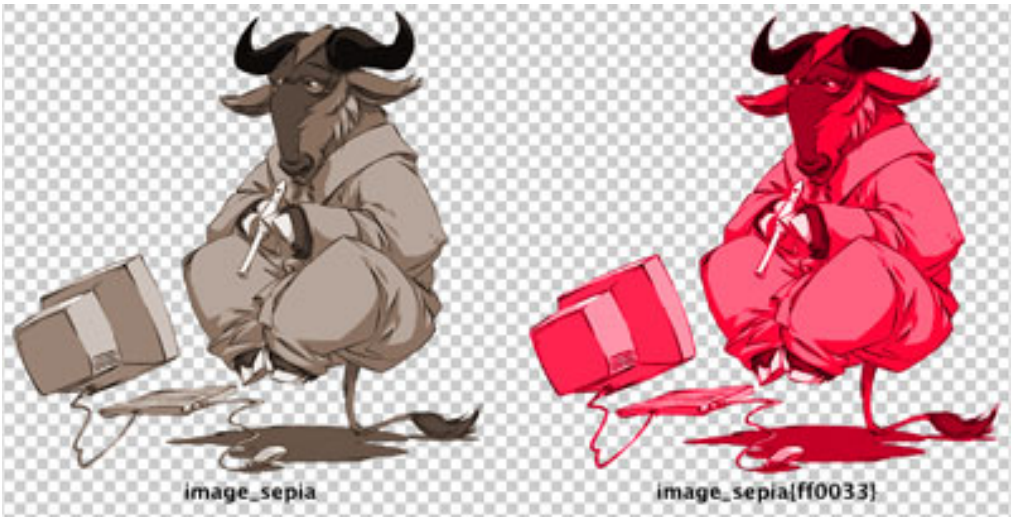
On peut forcer la pondération des trois composantes RVB en passant les valeurs en pour-mille. Par exemple (image de droite) :

```
[(#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 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)]
```

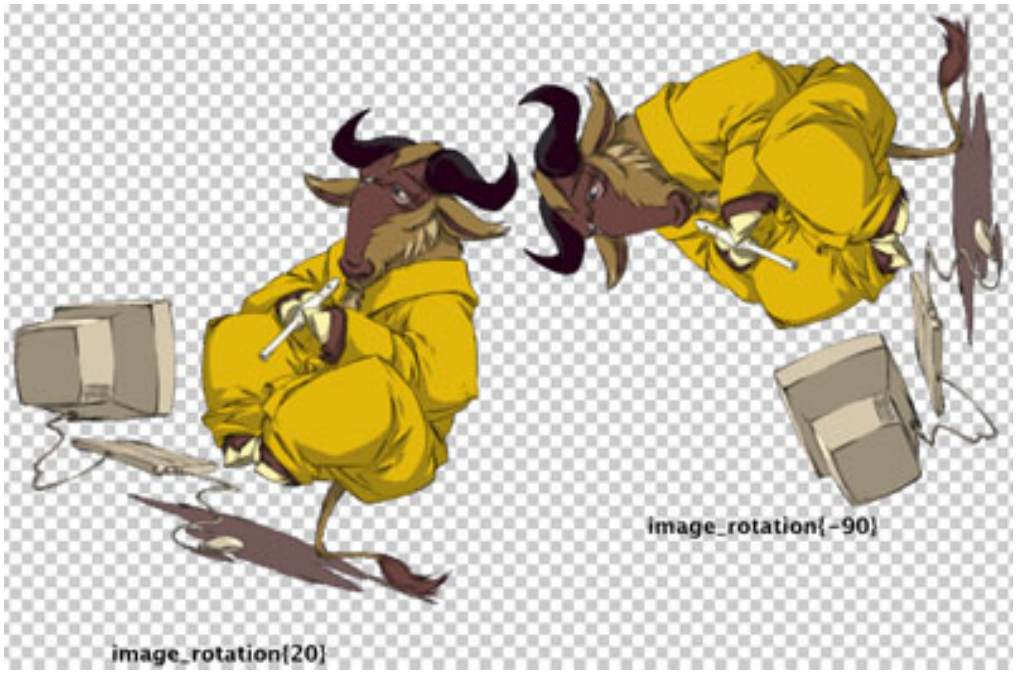
Au pire, le calcul se fera en deux « recalcul » de squelette, le premier floutage

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 !

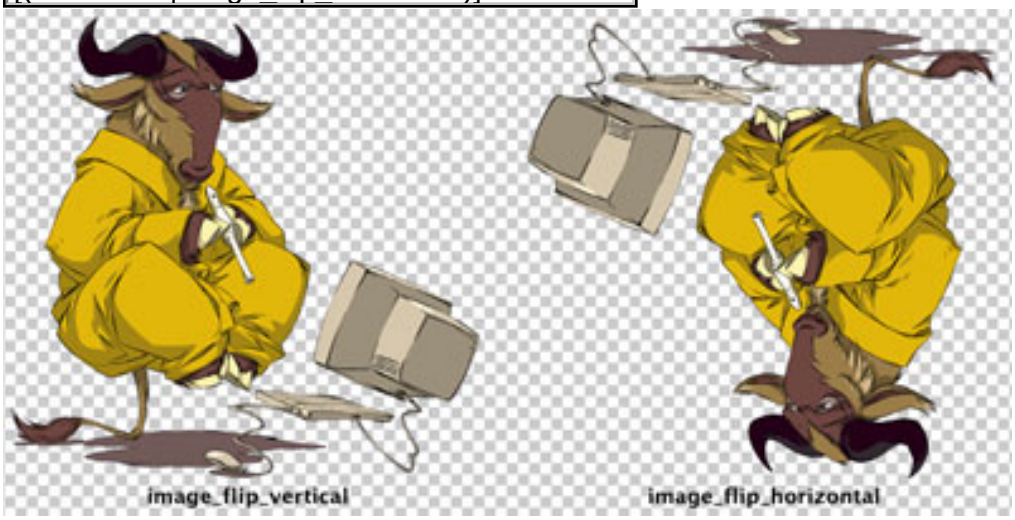
Attention (2) : ce filtre modifie les dimensions de l'image.

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.

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.

La structure de la base de données

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.

Initiation : utiliser les feuilles de style avec SPIP

Pour tirer parti de toute la souplesse de SPIP, il est recommandé d'utiliser les feuilles de style. Pas de panique, cette petite initiation permettra aux débutants de raccrocher les wagons...

Introduction

Cette présentation considère que vous connaissez le système des squelettes SPIP et que vous êtes capables de comprendre des squelettes simples. Dans le cas contraire, nous vous conseillons de relire le tutorial et/ou le manuel de référence.

Pourquoi les feuilles de style ?

Si vous réalisez des pages Web de manière « traditionnelle », les indications graphiques sont insérées 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 dans tous les fichiers HTML rechercher les morceaux de HTML à modifier, et effectuer les 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, 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'élements 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. 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

La feuille de style, lorsque vous l'appliquez à un fichier HTML (qui peut être un squelette SPIP), doit être déclarée dans le tag `<head>` du fichier HTML - aux côtés du titre et autres champs `<meta>`. De la façon suivante :

```
<head>
   ...
   <link rel="stylesheet" type="text/css" href="mes_styles.
css">
</head>
```

Ici le fichier `mes_styles.css` contient les propriétés graphiques que je veux appliquer à la page HTML (dans toute la suite du tutorial, 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** [1] 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.

> Note : une feuille de style peut s'appliquer aussi bien à une page HTML classique (« statique ») qu'à un squelette SPIP. 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 se mettre le pied à l'étrier...

[1] *Cascading Style Sheets* : littéralement, « feuilles de style en cascade ».

Des styles qui ont de la « class »

Dans l'article précédent, nous avons exposé de manière générale les avantages des feuilles de style. Ici nous expliquons les apports supplémentaires des feuilles de style lorsqu'elles sont utilisées en conjonction avec SPIP.

Les styles définis par SPIP

Comme vous le savez désormais, 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. Cela rend leur utilisation appréciable, sans être forcément indispensable : vous pouvez continuer à insérer, par endroits, des indications graphiques directement dans le HTML tout en utilisant aussi une feuille de style.

Dans SPIP, les feuilles de style acquièrent une fonction supplémentaire, fondamentale : elles servent à modifier les propriétés graphiques des éléments qui ne sont *pas* définis dans votre HTML (celui de votre squelette) ! En effet, SPIP génère de lui-même une multitude de styles d'affichage divers et variés. Ainsi, les raccourcis typographiques (liens hypertexte, intertitres, gras, italique, tableaux...) sont transformés en code HTML afin de les représenter à l'écran. Cela vaut aussi pour les formulaires automatiques (répondre à un forum, signer une pétition...) et d'autres encore.

Afin que vous puissiez tout de même modifier l'apparence graphique de ces styles, SPIP leur donne un nom spécifique, immuable (ces noms font l'objet d'une liste exhaustive dans « Spip et les feuilles de style »). Par exemple, les `{{{intertitres}}}` ne génèrent pas un simple tag `<h3>`, mais un tag **`<h3 class="spip">`**. Quel est l'intérêt ? Ces tags 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.

Comment fait-on alors pour changer l'apparence de tous les intertitres SPIP ? C'est très simple, il suffit d'ouvrir votre fichier `mes_styles.css` (ou tout autre nom que vous avez décidé de lui donner) dans un éditeur de texte et d'y ajouter les lignes suivantes :

```
h3.spip {
   color: red;
   font-size: 18px;
}
```

Rechargez la page et tous les intertitres SPIP apparaîtront comme par magie en rouge ; remarquez de plus que les autres tags `<h3>` de votre page, s'il y en a, ne sont *pas* affichés en rouge... Si rien de tout cela n'apparait, vérifiez bien que vous avez déclaré la feuille de style dans le squelette (dans le tag `<head>` comme expliqué dans l'article précédent), et recalculez la page...

Expliquons brièvement la syntaxe de cette règle de mise en page :

- `h3.spip` juste avant les accolades signifie que la règle qui suit ne s'appliquera qu'aux tags `<h3>` qui ont un attribut `class` égal à « `spip` ». Notez bien : *ni les tags `<h3>` n'ayant pas cet attribut, ni les tags ayant cet attribut sans être des tags `<h3>`, ne seront concernés.*

- Les accolades contiennent la liste des propriétés graphiques associées au style ainsi définies. Notons que toutes les propriétés non définies dans cette liste garderont leur valeur habituelle pour le tag considéré ; dans le cas présent, le tag `<h3>` générera toujours un texte en gras, car rien dans le style ne dit le contraire.

Les propriétés listées entre accolades sont chacunes terminées par un point-virgule. Elles sont constituées d'un nom (ce nom est standardisé par le langage CSS), suivi d'un deux-points et d'une ou plusieurs valeurs. 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.

> **Important :** si vous ajoutez vos propres styles, sachez que la valeur donnée à l'attribut `class` est totalement arbitraire. Votre navigateur ne fera aucune différence, que cet attribut soit nommé `spip`, `menu-rubriques` ou `cheval321`. La seule chose qui compte est que cette valeur corresponde bien à la règle que vous aurez énoncée dans votre feuille de style.

Comme vous le remarquerez, le langage CSS est très simple et utilise le même type de vocabulaire que les attributs HTML classiques. Quand vous progresserez dans le langage des feuilles de style, vous continuerez à retrouver des notions plus ou moins héritées du HTML traditionnel (`border`, `width`, `height`...).

La gestion du cache

Le fait que votre feuille de style soit définie dans un fichier séparé (le fameux `mes_styles.css`) 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 votre 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 : la ligne `<link rel="stylesheet" type="text/css" href="mes_styles.css">` doit se trouver « dans le cache » pour que votre navigateur puisse en tenir compte.

Une typographie personnalisée

Après une introduction générale sur les feuilles de style, nous allons maintenant passer en revue quelques-uns de leurs usages les plus courants sous SPIP.

Intéressons-nous ici aux styles créés lorsque des raccourcis typographiques sont insérés dans un texte entré sous SPIP. Que l'on affiche un article, une brève, une rubrique ou autre n'a aucune importance : les styles portent toujours les mêmes noms. Cela ne nous empêchera pas de découvrir comment y appliquer éventuellement des habillages graphiques distincts...

Le texte de base

Même le texte de base sous SPIP, celui que vous entrez dans un formulaire en tapant au kilomètre, génère des tags HTML particuliers. En effet, il est découpé en paragraphes ; à chaque paragraphe correspond un tag `<p class="spip">`. Nous pouvons donc associer à ces paragraphes un style bien précis, qui ne sera pas appliqué au reste du texte de la page.

Commençons par choisir une police de caractères. Pour cela on utilise la propriété `font-family`, qui prend pour valeur un ou plusieurs noms de polices de caractères à utiliser. Pour un corps de texte comme celui d'un article il vaut mieux

une fonte à empattements ; mettons que vous vous décidiez pour « Bookman Old Style ». Ajoutez donc à votre fichier `mes_styles.css` la règle suivante :

```
p.spip {
   font-family: "Bookman Old Style";
}
```

(notez qu'un nom de fonte en plusieurs mots doit être entouré de guillemets...) Un problème peut survenir si la police « Bookman Old Style » n'est pas installée sur l'ordinateur de vos visiteurs : chaque ordinateur a une configuration différente, et n'oubliez pas que les fontes « gratuites » de Microsoft sont rarement installées sur Linux et Macintosh... Il est préférable de prévoir ce cas et spécifier une ou plusieurs polices de remplacement successives :

```
p.spip {
   font-family: "Bookman Old Style", "Times New Roman",
serif;
}
```

On spécifie ici comme remplaçantes successives de Bookman, la classique « Times New Roman », et en dernier recours « `serif` ». « `serif` » n'est pas une police en soi, c'est un code générique qui indique au navigateur de prendre la police à empattements par défaut de l'ordinateur ; de même « `sans-serif` » spécifie la police sans empattements par défaut (en général Arial ou Helvetica).

> On retiendra cette règle importante : dans la propriété font-family, il convient toujours de proposer plusieurs choix successifs pour s'adapter aux polices de caractères installées sur l'ordinateur du visiteur. Notons que cette règle est aussi valable pour le tag `<font face="...">` du HTML traditionnel.

Bien évidemment d'autres propriétés sont à votre disposition. Vous pouvez par exemple régler la taille du texte avec la propriété `font-size`. Notez cependant que les navigateurs disposent d'un réglage pour configurer la taille du texte par défaut, et le texte principal de vos pages ne devrait pas outrepasser ce réglage, pour des raisons de confort visuel : c'est l'utilisateur qui choisit la taille de base, non le webmestre.

Notez bien que les styles que vous appliquez aux tags <p> s'appliquent à chaque paragraphe en tant qu'objet autonome. Cela autorise certains effets intéressants, comme par exemple d'indenter la première ligne des paragraphes en utilisant la propriété `text-indent`. Par défaut, cette propriété prend pour valeur zéro, c'est-à-dire qu'il n'y a pas d'indentation. On peut la modifier pour obtenir, sur chaque première ligne, un décalage de soixante pixels à droite :

```
p.spip {
   text-indent: 60px;
}
```

Les liens hypertexte

Ceux qui ont déjà réalisé leurs premières armes en CSS savent qu'on peut modifier l'habillage des liens de façon globale :

```
a {
   color: green;
   text-decoration: none;
}
```

Cette règle de style spécifie que tous les liens hypertexte (c'est-à-dire tous les tags <a ...>, qu'ils aient ou non un attribut `class`) seront affichés en vert sans soulignement.

SPIP permet d'aller plus loin. Les liens hypertexte, lorsqu'ils sont générés par les raccourcis typographiques, utilisent en effet plusieurs styles différents :

- lorsque le lien est interne (il renvoie vers une autre page de votre site), le tag est <a class="spip_in">;
- lorsque le lien est externe (il renvoie vers un autre site Web), le tag est <a class="spip_out"> [1];
- enfin, lorsque l'URL est entrée sans titre, le tag est <a class="spip_url">.

Il devient donc très simple de manifester par une apparence graphique différente ces trois types de liens. Ainsi :

```
a {
   color: green;
   text-decoration: none;
```

```
}
a.spip_in {
   color: blue;
}
a.spip_out {
   color: red;
}
```

affiche les liens internes en bleu et les liens externes en rouge. Tous les autres liens, y compris ceux qui ne sont pas générés par SPIP, s'afficheront en vert. De plus tous les liens seront affichées sans surlignement : en effet la propriété `text-decoration`, spécifiée dans la première règle, n'a pas été modifiée dans les suivantes ; elle s'applique donc automatiquement à tous les éléments de type `<a ...>`.

> 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.

Appliquer un traitement différencié

Plutôt que de nous étendre sur la panoplie de styles générés automatiquement par les raccourcis typographiques de SPIP, et que vous pourrez habillez à votre guise (ils sont énumérés dans « Spip et les feuilles de style »), étudions ici le cas où vous voulez appliquer un habillage différent à un même style, selon sa position dans ce squelette. Ce besoin est légitime : on veut par exemple afficher le corps de texte dans une police à empattements avec indentation en début de paragraphe, mais le post-scriptum dans une police plus « lisse » (sans empattements) et plus petite, sans indentation.

Cette opération est en réalité très simple. Il faut d'abord modifier votre squelette afin d'introduire les éléments qui permettront de discriminer le texte et le post-scriptum. Cela prendra par exemple, à l'intérieur de la boucle `ARTICLES` principale, la forme suivante :

```
<div class="texte">#TEXTE</div>
<div class="ps">#PS</div>
```

Il faut ici recalculer la page (car on a modifié le HTML...). L'affichage du navigateur est toujours le même : normal, nos nouveaux styles ne faisant l'objet d'aucune règle dans la feuille de style, ils sont ignorés par le navigateur. Remédions-y :

```
.texte p.spip {
   font-family: "Times New Roman", serif;
   text-indent: 50px;
}
.ps p.spip {
   font-family: Tahoma, Arial, sans-serif;
   font-size: 90%;
}
```

La grande nouveauté ici ne réside pas dans les propriétés graphiques mais dans la façon dont on les applique au code HTML. En effet, « `.texte p.spip` » signifie : « cette règle s'applique à tous les tags `<p class="spip">` qui sont contenus dans un tag ayant un attribut `class` égal à "`texte`" ». On pourrait restreindre un peu cette règle en spécifiant que le tag parent doit en plus être un tag `<div>` (le début de la règle s'écrirait alors « `div.texte p.spip` ») ; mais comme nous maîtrisons la structure de nos propres squelettes, il n'est pas utile de rendre la feuille de style très restrictive.

Toujours est-il que cette feuille de style a le résultat voulu : les paragraphes du corps de texte s'affichent avec une indentation, ceux du post-scriptum dans une fonte plus petite et sans indentation. Pour vérifier que ces règles s'appliquent bien à chacun des paragraphes `<p class="spip">` et non au `<div class="...">` englobant, on peut s'amuser à définir un cadre noir :

```
.texte p.spip {
   border: 1px solid black;
}
```

On note que chaque paragraphe du corps de texte (mais pas du post-scriptum) est entouré de son propre cadre noir. Si on avait simplement écrit « `.texte` » au lieu de « `.texte p.spip` », c'est le texte tout entier qui serait entouré d'un unique cadre noir englobant. Remarquons en passant l'apparition de la propriété `border`...

> Note : cette astuce, consistant à tracer un cadre de couleur pour savoir à quels éléments s'applique précisément une règle, peut être très utile quand votre feuille de style s'enrichit. N'hésitez pas à l'utiliser si vous commencez à perdre pied...

Cette méthode est très puissante et se généralise avec profit pour la structuration de votre mise en page.

[1] Que les amateurs de liens ouvrants ne se réjouissent pas trop vite : il est impossible, dans une feuille de style, de spécifier qu'un lien doit s'ouvrir une nouvelle fenêtre... Raté !

Ils sont beaux, mes formulaires !

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.

Vous vous souvenez que les feuilles de style, utilisées en conjonction avec SPIP, permettent d'adapter le rendu des raccourcis typographiques à votre mise en page. Eh bien, il en va de même pour les formulaires générés par SPIP : leur aspect graphique peut être modulé afin de s'insérer sans hiatus dans votre design.

Encore une fois, pas question ici de passer en revue l'ensemble des styles fournis par SPIP. Sachez cependant que vous pouvez altérer l'aspect de tous les formulaires de l'espace public : formulaire de réponse aux forums, de recherche, de signature de pétition... [1] Nous nous bornons ici à donner quelques recettes.

Des couleurs et du relief

Une nouvelle qui ravira les débutants en CSS : on peut changer la couleur des champs et des boutons des formulaires [2].

Par exemple, pour que les boutons aient un fond bleu clair, ajoutez la règle suivante à votre fichier CSS (qui, si vous avez suivi cette initiation à la lettre, s'appelle `mes_styles.css`) :

```
.spip_bouton {
   background-color: #b0d0FF;
   color: black;
}
```

Les boutons apparaissent désormais avec un fond bleu clair (propriété `background-color`), et un texte noir. Notez que **`.spip_bouton`** est le style utilisé par les boutons et champs des formulaires SPIP.

Modifions maintenant l'aspect du *bord* des boutons. Le relief traditionnel des boutons HTML est un peu vieillot. On peut décider d'aplatir les bords, et de les épaissir en contrepartie pour qu'ils restent bien visibles. Par exemple :

```
.spip_bouton {
   background-color: #b0d0FF;
   color: black;
   border: 2px solid #000060;
}
```

La propriété `border` ainsi définie trace un bord épais de 2 pixels, à l'aspect plat (« `solid` » en langage CSS) et de couleur bleu foncé autour des boutons. On pourra également modifier la police de caractères du bouton (avec les propriétés `font-size` et `font-family` comme vu aux étapes précédentes de cette initiation).

Et pour les champs ? Il suffit d'appliquer vos choix au style **`.formul`** au lieu de `.spip_bouton`.

Un peu d'espace

Les feuilles de style permettent non seulement de changer les couleurs et les polices de caractères, mais aussi de gérer le positionnement relatif des objets dans la page. Sans aller très loin, montrons comment aérer un peu les formulaires :

```
.formulaire {
   background-color: #e8f4ff;
   font-family: Verdana, Arial, sans-serif;
```

```
    font-size: 90%;
    font-weight: normal;
    border: 1px solid black;

    padding: 10px;


}
```

Nous modifions ici l'affichage du style `.formulaire`, qui est le style principal de tous les formulaires générés par SPIP. L'intérêt de modifier ce style est que l'on peut gérer l'affichage de tous les formulaires. Ici l'on applique une couleur de fond, très claire, et une police de caractères. Mais surtout, on modifie l'espacement intérieur de chaque formulaire pris individuellement. C'est la propriété `padding` qui permet cet effet d'aération. Notez bien que cette aération se produit précisément à l'intérieur du bord défini par la propriété `border` : le formulaire est ici considéré comme un *bloc rectangulaire autonome*.

> *Remarque :* les feuilles de style permettent même de définir précisément la disposition de ces blocs rectangulaires entre eux, sans utiliser de tableaux pour la mise en page. C'est cependant un sujet trop vaste pour cette initiation.

[1] Et même les boutons d'administration qui s'affichent en bas des pages lorsque vous êtes connecté.

[2] Notons cependant que certains navigateurs imposent leurs propres boutons stylisés et ne vous laisseront pas en changer l'aspect.

Pour en savoir plus

Cette initiation n'a fait qu'effleurer la puissance des feuilles de style. Vous êtes libres de vous contenter du niveau abordé ici, ou d'explorer les nombreux documents disponibles sur le Web afin d'aller plus loin.

Citons quelques portes de sortie intéressantes.

En français

- OpenWeb, site mêlant argumentaires idéologiques et articles techniques sur les CSS et autres standards du Web ;
- un cours « CSS débutant » ;
- « Techniques et astuces pratiques pour une mise en page CSS » vous expliquera comment gérer le positionnement d'éléments avec les feuilles de style ;
- une collection de recettes pour une utilisation efficace des CSS ;
- un pense-bête, pour ne pas se perdre dans le feu de l'action ;

- une traduction en français du standard CSS2 (assez aride).

En anglais

- 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

- Si vous utilisez l'excellent navigateur Firefox, le plug-in EditCss vous permet de modifier et tester les feuilles de style à la volée, depuis votre brouteur.

Trucs et astuces

Remplir les meta-tags HTML des pages d'article

Afficher automatiquement selon la date ou selon un ordre imposé

La situation est la suivante : dans un même site, la présentation des articles dans différentes rubriques doit être différenciée :

- dans certaines rubriques, les articles sont publiés les uns après les 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 veut « forcer » l'affichage des articles en les numérotant (« 1. Le premier article », « 2. Le deuxième article »...) ; sur le site public, on veut donc les présenter selon cet ordre indiqué par la numérotation.

Voici une méthode pour réaliser cet effet.

Nous sommes à l'intérieur d'une boucle de rubrique (par exemple, la page « rubrique.html ») :

```
<BOUCLE_rubrique_principale
(RUBRIQUES){id_rubrique}>

    ...
```

À l'intérieur de cette boucle, 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>
```

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}>
        <li> [(#TITRE|supprimer_numero)]
    </BOUCLE_ordre_numeros>
</BOUCLE_test_numero>
    <BOUCLE_ordre_date(ARTICLES)
{id_rubrique}{par date}{inverse}>
```

Trier des articles par ordre alphabétique, sauf un qu'il faut afficher en premier

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 de boucle* `{par titre}`*). Tout autre classement ne fonctionnera pas.*

Remarque : cela s'applique également aux rubriques, brèves, sites référencés, etc.

tableau

- Ne pas afficher les articles publiés depuis plus d'un an
- Présenter les résultats d'une recherche par secteurs
- Afficher le nombre de messages du forum lié à un article
- Un menu déroulant pour présenter une liste d'articles
- Remplir les meta-tags HTML des pages d'article

Plusieurs logos pour un article

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.

Jusqu'à récemment, les utilisateurs avaient créé des méthodes personnelles basées sur l'utilisation différenciée du logo « normal » et du logo « pour survol ». Dans notre 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 la version [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 normalement (colonne de gauche) ; 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.

- en bas de la page de l'article, nous installons 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>
```

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>
```

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 (encore peu répandu parmi les sites utilisant SPIP) 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>
```

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)
```

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)
```

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
```

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
```

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
```

(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}>
```

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
```

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>
      [<div>(#SURTITRE|majuscules)</div>]
      <div><font size=6>#TITRE</font></div>
      [<div>(#SOUSTITRE|majuscules)</div>]
      <br>[(#DATE|nom_jour)] [(#DATE|
```

* * *

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

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 deux fichiers : un fichier myauteur.php3 et un fichier myauteur.html

Création du fichier myauteur.php3

Dans le fichier myauteur.php3 mettre le code suivant :

```
<?php

$fond = "myauteur";
$delais = 24*3600;

include ("inc-public.php3");

?>
```

Création du fichier myauteur.html

Dans le fichier myauteur.php3 mettre les codes suivants :

- Juste après la balise <body>, mettre

```
<BOUCLE_principale(AUTEURS){id_auteur}
```

- 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">
```

Code pour article choisi au hasard

```
<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 hasard}
{doublons}{0,15}>
                  <li><a href="#URL_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="myauteur.php3?id_auteur=#ID_AUTEUR">nom du lien</a>
```

Afficher des éléments par lignes dans un tableau

Par exemple, on peut vouloir créer un tableau contenant les titres des articles d'une rubrique agencés sur trois colonnes, 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

L'astuce consiste à jouer à la fois avec les doublons et avec les boucles récursives. On construit une première boucle qui affiche les trois premiers articles de la rubrique *une fois les doublons éliminés*. On voit qu'il suffit ensuite de réafficher cette boucle à chaque fois qu'il reste des articles pour afficher graduellement tous les articles, ceux déjà affichés venant à chaque fois grossir les rangs des doublons. Pour cela, dans le code conditionnel de cette boucle, on ajoute un appel récursif vers la boucle elle-même : elle sera affichée tant qu'elle produit des résultats.

```
<table>
<B_ligne>
	<tr>
<BOUCLE_ligne (ARTICLES) {id_rubrique}
{doublons} {par titre} {0,3}>
	<td width="33%">
	<a href="#URL_ARTICLE">#TITRE</a>
	</td>
</BOUCLE_ligne>
	</tr>

	<BOUCLE_ligne_suite (BOUCLE_ligne)
></BOUCLE_ligne_suite>
```

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 oeuvres exposées.

Ne pas afficher les articles publiés depuis plus d'un an

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. Le critère de sélection qui nous intéresse ici est : « *age* ».

```
<B_articles_recents>
        <ul>
<BOUCLE_articles_recents(ARTICLES)
{id_rubrique}{age < 365}{par titre}>
        <li>#TITRE</li>
</BOUCLE_articles_recents>
        </ul>
```

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 faut utiliser le critère « *age_redac* » au lieu de « *age* ». L'âge est indiqué en nombre de jours.

Notons que cette manipulation est possible avec tous les types de données auxquels est associée une date (brèves, forums...).

d’un an

- Présenter les résultats d’une recherche par secteurs
- Afficher le nombre de messages du forum lié à un article
- Un menu déroulant pour présenter une liste d’articles
- Remplir les meta-tags HTML des pages d’article

Présenter les résultats d'une recherche par secteurs

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>
```

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.

plus d’un an

- **Présenter les résultats d’une recherche par secteurs**
- Afficher le nombre de messages du forum lié à un article
- Un menu déroulant pour présenter une liste d’articles
- Remplir les meta-tags HTML des pages d’article

Afficher le nombre de messages du forum lié à un article

C'est un poil acrobatique.

À première vue, il est très simple de connaître le nombre d'éléments d'une boucle : il suffit d'utiliser le code SPIP : #TOTAL_BOUCLE. Ce code peut s'utiliser non seulement à l'intérieur de la boucle, mais aussi (c'est le seul 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).

Nous devons créer une boucle de type FORUMS, liée à un article, de façon à compter son nombre de résultats.

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}>
```

contient uniquement les messages qui répondent à l'article. Habituellement, pour accéder aux réponses à ces messages, on inclut une seconde boucle à l'intérieur de

celle-ci... Ici, nous voulons que la boucle sélectionne absolument tous les messages attachés 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}
```

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 pas le titre des messages, on évitera même de lui faire afficher des espaces ou des retours à la ligne (sinon votre page HTML contiendra des dizaines de lignes vides, inélégantes) ; l'intérieur de la boucle n'affiche donc rigoureusement rien, mais on doit afficher, après la boucle, le nombre de résultats.

Une subtilité à bien comprendre : le texte conditionnel alternatif s'affiche *si la boucle n'affiche rien* ; 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 contributions au forum ») dans ce cas.

```
<BOUCLE_nb_forums(FORUMS){id_article}
{plat}></BOUCLE_nb_forums>
[Il y a (#TOTAL_BOUCLE) contribution(s) au
```

Un menu déroulant pour présenter une liste d'articles

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 `article.php3?id_article=123`, le bout de code suivant conviendra :

```
<FORM ACTION="article.php3"
METHOD="get">

      <SELECT NAME="id_article">
            <BOUCLE_menu_articles
(ARTICLES) {id_rubrique} {par titre}>
            <OPTION
VALUE="#ID_ARTICLE">#TITRE</OPTION>
            </BOUCLE_menu_articles>
      </SELECT>
```

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.

plus d'un an

- Présenter les résultats d'une recherche par secteurs
- Afficher le nombre de messages du forum lié à un article
- **Un menu déroulant pour présenter une liste d'articles**
- Remplir les meta-tags HTML des pages d'article

Remplir les meta-tags HTML des pages d'article

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.

- Afficher automatiquement selon la date ou selon un ordre imposé
- Trier des articles par ordre alphabétique, sauf un qu'il faut afficher en premier
- Plusieurs logos pour un article
- Afficher les derniers articles de vos rédacteurs par rubrique
- Afficher des éléments par lignes dans un tableau

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.

```
<head>
<BOUCLE_head(ARTICLES){id_article}>
<title>#TITRE</title>
<meta name="Description"
content="#DESCRIPTIF">
<meta name="Keywords"
content="<BOUCLE_keywords(MOTS)
{id_article}{", "}>#TITRE </
```

On remarquera que pour les mots-clés et l'auteur, 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).

Attention ! le code donné ci-dessus à titre d'exemple est un peu « naïf » : si le `#NOM` d'un auteur ou le `#DESCRIPTIF` d'un article peuvent contenir des tags html

(mise en italiques, saut de paragraphe...) la page qui en résultera sera en effet pleine d'erreurs. Pour éviter cela, il faut penser à passer un filtre comme `|supprimer_tags` sur le champ en question (remplacer `#DESCRIPTIF` par `[(#DESCRIPTIF|supprimer_tags)]`)...

Le développement de SPIP et ses outils

Les différents outils de communication utilisés pour développer SPIP.

Rechercher

français

tout le site

Rechercher

dev.spip

Le développement de SPIP est désormais fait sous SVN. Il est facile à suivre à travers différents outils, dont l'interface Web proposée sur ce site.

SPIP Zone

Un outil communautaire pour développer des squelettes, des outils, des scripts... autour de SPIP.

Modifications récentes

- Le calendrier de SPIP 1.8.2
- Internationaliser les

La liste des annonces des développements : spip-core@rezo.net

La liste spip-core n'est pas une liste de discussion : personne ne peut y poster de message. Elle est destinée à recevoir les informations sur les plus récents développements.

On peut néanmoins s'y abonner et consulter ses archives si l'on veut tester les tous derniers développements de SPIP et disposer des informations nécessaires.

N.B. Tous les messages de spip-core sont également disponibles sur spip-dev.

La liste des développeurs : spip-dev@rezo.net

Attention : **cette liste de travail est destinée à discuter de la *programmation* de SPIP** (et non pas la programmation *avec* SPIP).

Le fait que votre problème soit compliqué et implique l'utilisation de PHP et MySQL *n'est pas* une raison pour poster sur spip-dev : toutes les questions liées à l'utilisation de SPIP, y compris les plus techniques, doivent être postées sur la liste des utilisateurs mentionnée plus haut. Merci d'avance de votre compréhension.

> Merci de consulter attentivement les archives de spip-dev avant de poster sur la liste : ceux qui y participent ont fourni de gros efforts pour développer le système, il est normal que vous fassiez à votre tour un petit effort de documentation pour que l'échange d'information soit aussi efficace

que possible.