Developpez

Le Mag

Edition de Août - Septembre 2009.
Numéro 23.
Magazine en ligne gratuit.

Réalisation : Alexandre Pottiez
Rédaction : la rédaction de Developpez
Contact : magazine@redaction-developpez.com

Sommaire

Editorial

Voici votre nouveau manuel scolaire pour une rentrée Developpez.com

Retrouvez une sélection de nos meilleurs articles, critiques de livres, et questions/réponses sur diverses technologies.

Profitez-en bien !

La rédaction

Article Webmarketing

Le référencement / SEO pour les débutants

Découvrez les aspects du référencement web et améliorez le référencement de vos sites.

par **Jean-François Lépine**
Page 33

Article 2D/3D/Jeux

Une introduction à CUDA

Ecrivez vos premiers kernels avec CUDA.

par **Thibaut Cuvelier**
Page 53

JavaOne 2009

Un événement mondial attendu par tous les fans de Java qui se respectent : JavaOne.
Scindé en plusieurs événements, en 5 jours, Sun présentera Java University, CommunityOne et JavaOne.

Cependant, cet événement était beaucoup plus important que tous les autres.
Le rachat de SUN MicroSystems par Oracle, submerge les esprits de questions et de doutes.

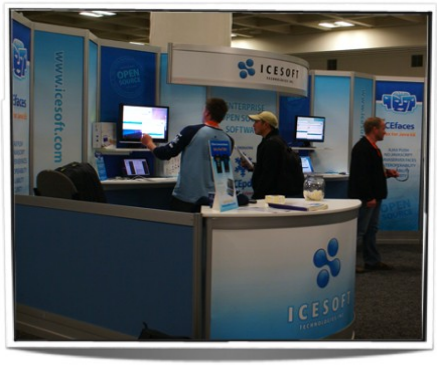
Oracle arrivera-t-il à rassurer la communauté ?

1. Les lieux

Un événement au Moscone Center est ... un événement.
C'est le cas de le dire avec les trois bâtiments dont deux qui communiquent en passant sous la route et un troisième situé au carrefour à proximité des deux autres.
Avec 65 000 m2 où plus de 200 sessions techniques ont été présentées, l'organisation ainsi que la logistique sont de taille.

Un espace d'exposition où les plus grandes compagnies exposent leurs produites.

Ci-dessous, **IceFaces**



Spring

Fatigué de marcher et de participer à des sessions ???

Sun a tout prévu.
Vous pouvez trouver des aires de jeux et de détente. L'ordre des termes est volontaire car nous trouvons bon nombre de consoles de jeux où nous pouvons échanger d'une autre manière avec les personnes présentes.
Ici, l'aire de jeu dans le hall d'entrée :

Ici, l'aire de jeu au milieu de la salle d'exposition des entreprises.

2. L'organisation et la logistique

Sun n'en est pas à sa première et on le ressent dès l'arrivée à l'enregistrement.
Certes, au départ c'est assez désorientant mais avec les personnes postées pour surveiller (à qui on peut demander un renseignement et qui sont toujours prêtes à vous informer voire discuter) et les cabines où les informations sur le lieu et les évènements résident, plus aucun doute ni aucune excuse pour se perdre.

Avec tous les panneaux aux portes d'entrée qui indiquent actuellement et prochainement les sessions en cours dans les salles ainsi que les panneaux de 3x1m indiquant les endroits BOF, Sessions générales... on ne peut que trouver son bonheur.

Armé d'un "livre" comportant le plan et les sessions avec un descriptif du contenu, il suffisait de prévoir les horaires et de partir à la conquête du Moscone Center d'une soif importante de connaissances au royaume de Java.
Peut-être est-ce excessif et pourtant je pèse mes mots. Un endroit où toute une communauté se joint pour partager une passion qui fait partis intégrante de chacun ne peut être que le paradis pour le plus addict et Sun nous le rend bien.

Du côté de la restauration :

La salle de restauration représente aisément l'importance de l'évènement et le nombre de personnes attendues.

Un flot de goodies


Comme on peut le constater, les goodies sont nombreux. Nous pouvons voir un sac, un tee-shirt, un jeu de carte...

3. Planning général

Voici le planning général de l'évènement où les sessions les plus importantes sont notées.

Lundi 1er juin - CommunityOne
- 9:30am to 10:30am: Keynotes by Dave Douglas (cloud) and John Fowler (OpenSolaris)
- 10:35am to 11:15am: Post keynote press conference
- Panel session: Security in the Cloud
- Panel session: Social media developer
- 7:00pm: International media and analyst dinner at Scala's. Cocktails at 7:00pm, dinner at 7:30pm

Mardi 2 juin - JavaOne Day 1
- 8:30am to 10:30am: General session, keynotes (including by Jonathan Schwartz joined by industry luminaries)
- 10:35am to 11:15am: News advance/product news overview (executives TBD)
- 1:30pm to 3:00pm: Afternoon general session/keynote
- 5:00pm to 6:30pm: Co-sponsors cocktail reception

Mercredi 3 juin - JavaOne Day 2
- 8:30am to 9:30am: Sun Mobility General Session
- 9:40am to 10:00am: Sun Mobility post keynote press conference
- Panel session: Internet Archive customer
- Panel session: Java Utopia - partners, mobility and entertainment
- Panel session: Java Community Process/International media session

Jeudi 4 juin - JavaOne Day 3
- 8:30am to 9:15am: Oracle general session/keynote (TBD)
- 9:20am to 10:00am: Oracle post keynote press conference (TBD) (**Annulé et remplacé par Microsoft dont le thème était interopérabilité .NET/Java**)
- Panel session: NHIN Connect/Enterprise Java (open source, standards for governments)
- Panel session: 21st century skills for next generation developers and students
- 5:30pm to 6:15pm: IBM general session (TBD)
- 6:15pm to 7:15pm: IBM post keynote press conference (TBD)
- International media depart

4. Les informations à retenir

Le thème général :
JavaFx 1.2

Les «nouveautés» présentées :

Java EE 6
Avec =>
Servlet 3.0
JPA 2.0
EJB 3.1
JSF 2.0
JAX-WS (De la JSR 101 Ã la 204)
...

Micro Edition 3.0

Et biensûr JavaFx 1.2

Des questions

Des réponses à de nombreuses questions :

Les développeurs Java rassurés par le discours de Larry Ellison («Toutes nos applications sans compter les bases des données, utilisent Java et c'est pourquoi nous comptons investir davantage dans la technologie»). Tout autant, Oracle voit dans le marché du mobile (Netbooks, smartphone...), une place indéniable pour Java et Java Fx et a annoncé qu'ils ne s'attarderaient pas à s'y atteler.

Des questions qui subsistent :

MySQL, Glassfish, et autres produit Sun. Mais n'oublions pas que 90% des sources de revenus provenait du matériel.
Les points attendus non présentés :
Java SE 7 et Netbeans 6.7

JavaFx ou le maître mot de l'événement :

Sun et Oracle croient au marché que pourrait offrir JavaFx et offre à cette JavaOne beaucoup d'annonces basées sur celui-ci.

Avec 250 Millions de machines et un kit téléchargé plus de 400 000 fois, Sun croit en la part de marché de JavaFx et à son pouvoir à concurrencer des leaders déjà installés depuis quelques années.

Pour ce faire, le langage de script s'arme d'une nouvelle boîte à outils (la gestion du stockage local de données pour les applications hors connexion,de nouveaux contrôles (boutons, listes, histogrammes, barre de progression...)), d'une intégration plus stable et plus ancrée pour les EDIs...

Bref, Sun tente d'imposer son langage malgré la compétitivité.

Enfin un outil concret pour les WebDesigners :

D'ici fin 2009, un nouvel outil (pour concurrencer Adobe Flash CS4) nommé JavaFX Authoring Tool.

Un langage = de nouveaux horizons ?

Avec JavaFx, la firme de Menlo Park se fixe pour objectif d'intégrer un panel d'outils au sein des différents médias.

- JavaFx TV :
Une nouveauté qui va révolutionner le monde du huitième art.
Cette partie reprenait la démonstration de la keynote de CommunityOne sur Cloud.
Le petit écran de 119 cm (quand même) laissa apparaître une grille de vidéos du type Cloud DVR connecté à Sun Cloud.

La démonstration portait sur comment commander un film et/ou pouvoir discuter avec d'autres personnes.

En résumé, c'est la fusion entre les ondes et Internet.

- Intégration d'application dans les disques Blue-Ray :
Blue-Ray est le support d'une technologie qui permet d'accéder au contenu d'internet par un lecteur connecté Blue-Ray dont le projet se nomme BD-J.
Ceci permettrait de pouvoir télécharger les dernières sorties...
Il a même été évoqué de pouvoir voir naître la possibilité de télécharger des contenus, type sonneries de téléphone...

- La communication vue par JavaFx avec un téléphone JavaFx

- Tru2Way, le projet d'intégration de la plate-forme dans les télévisions.

Enfin, lors de l'apparition de Larry Allison, il confia qu'il aimerait voir l'utilisation de certaines librairies du langage de script dans OpenOffice.

Pour Java EE 6, ME 3.0, et les autres features, il fallait se rendre dans les sessions techniques pour vouloir connaître les quelques nouveautés.

D'autres nouveautés ?

Oui, et pas des moindres avec :

Java Store : A l'image de l'Apple Store, ceci serait un espace où nous pourrions retrouver les différentes applications java de tous types, payantes ou gratuites.

Java Warehouse : Il pourrait y avoir un espace pour les échanges entre entreprises partenaires ou autres.

Cependant, pour les deux types de stockage, rien n'est défini.

Java Store est disponible avec une adresse IP américaine, mais quant à son fonctionnement sur la validation des applications, rien n'est moins sûr.

5. Les reportages

Dès le début de la JavaOne, plusieurs grands noms de la communauté ont voulu vous dire un mot.

James Gosling : Créateur de Java et Java Fellow : Lien1

Bruno Hourdel : Directeur Marketing Europe de SUN MicroSystems : Lien2

Chris Melissinos : Chef évangéliste SUN MicroSystems : Lien3

6. Liens sur l'événement

Couverture Blog : Lien4
Les news JavaOne commentées sur le forum : Lien5
Les PDFs des sessions : Lien6

Retrouvez l'article de Mike François en ligne : Lien7

Installation et utilisation d'eclipse pour le développement en Java

La version originale de cet article peut être trouvé ici (Lien8). Copyright Henri Garreta , Université de la Méditerranée (Aix-Marseille Université) et Faculté des Sciences de Luminy

La documentation en ligne d' eclipse est abondante mais son approche n'est pas immédiate. Sans souci d'exhaustivité, nous donnons ici quelques indications pour débuter rapidement dans l'utilisation de cet excellent outil de développement.

A la date du 16 juillet 2009, les versions courantes des logiciels en question sont :

* Java JDK 1.6.0 (ou 6) update 14
* Eclipse platform 3.5.0 « Galileo »

Les versions décrites ici ne sont peut-être pas exactement celles-là mais elles leur sont fonctionnellement équivalentes

1. Se procurer le kit de développement Java

Eclipse ne contient ni le compilateur Java ni les autres outils basiques. Pour développer des programmes en Java il faut donc installer au préalable un kit de développement. Nous conseillons celui de Sun Microsystems, la maison mère de Java, qui est complet, à jour (par définition) et gratuit.

Vous pouvez l'obtenir chez Sun : site java.sun.com (Lien9), menu Downloads, rubrique Java SE (comme Standard Edition). Le produit à télécharger s'appelle, lors de la publication de cette notice, JDK 6 update 14 .

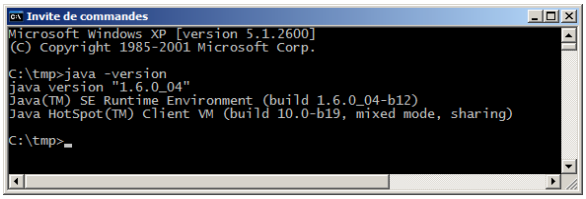
Attention, ne confondez pas le JDK (Java Development Kit) avec le JRE (Java Runtime Environment), appelé parfois « plugin Java », qui ne contient que le nécessaire pour exécuter les programmes Java. Ne vous occupez pas de télécharger le JRE, à l'intérieur du JDK il y en a un exemplaire.

Le fichier qui vous concerne se nomme :

- dans le cas de Windows : **jdk-6u14-windows-i586.exe** (73,5 Mo)
- dans le cas de Linux : **jdk-6u14-linux-i586.bin** (77 Mo)

Il s'agit dans les deux cas d'un installateur auto-extractible : après le téléchargement il suffit de le lancer et de suivre les instructions qui s'affichent. Au besoin, des renseignements supplémentaires sur l'installation du JDK sont donnés sur le site de Sun, aussi bien pour Windows (Lien10) que pour Linux (Lien11).

Une fois l'installation terminée, vérifiez sa réussite en tapant « **java -version** » dans une console de commandes. Vous devez obtenir un message vous annonçant le numéro de version de la machine Java mise en place. Dans le cas de Windows cela ressemblera à ceci :



Pour développer des programmes en Java il vous faut disposer également de la documentation de l' *API (Application Programmer Interface*, c'est-à-dire le volumineux ensemble de paquetages, classes, méthodes et variables qui constituent la bibliothèque système). Vous pouvez la consulter (Lien12) en ligne ou bien la télécharger depuis le site java.sun.com (Lien9), menu ***Downloads***, rubrique ***Java SE*** , produit ***Java SE 6 Documentation*** (quel que soit votre système d'exploitation, le fichier s'appelle **jdk-6u10-docs.zip** et pèse 56 Mo).

Note (cas de *Windows*). Si vous souhaitez pouvoir employer le compilateur et les autres outils Java en dehors d' *eclipse*, c'est-à-dire en tapant des commandes dans une console *Invite de commandes*, alors vous devez procéder à la manipulation supplémentaire suivante : repérer le répertoire ® d'installation de Java et ajouter le chemin ® \ **bin** dans la définition de la variable *Path* .
Si vous avez laissé l'installateur de Java faire à sa guise, ® doit être quelque chose comme **C:\Program Files\Java\jdk1.6.0_04** .
Vous pouvez examiner et modifier la valeur de la variable *Path* en cliquant avec le bouton droit sur l'icône du *Poste de travail* , puis *Propriétés* > *Avancé* > *Variables d'environnement* > *Variables système* ; sélectionner la ligne *Path* puis faire *Modifier*.

Dans le cas de *Linux*, une manipulation analogue est nécessaire après l'installation du *JDK* . Nous ne l'expliquons pas car elle fait partie des opérations courantes sur ce système.

2. Télécharger eclipse

Eclipse est un logiciel libre que vous pouvez télécharger depuis le site www.eclipse.org (Lien13), onglet *Downloads*. Le produit qui nous intéresse est *Eclipse IDE for Java Developers (92 MB)* .

Le fichier téléchargé se nomme

- dans le cas de Windows : **eclipse-java-galileo-win32.zip**
- dans le cas de Linux : **eclipse-java-galileo-linux-gtk.tar.gz**
- dans le cas de Max OS X : **eclipse-java-galileo-macosx-carbon.tar.gz**

Nous ne vous conseillons pas de télécharger une version française d' *eclipse*. Il existe bien des plugin de francisations de l'interface, mais outre le fait qu'elles sont assez imparfaites, elles servent surtout à vous empêcher d'utiliser la dernière version du logiciel.

3. Installer eclipse

A partir d'ici, les explications sont communes aux divers systèmes d'exploitation, ou bien ne concernent que Windows XP et Vista.

Pour installer *eclipse* il suffit de décompresser l'archive *zip* ou *tar.gz* téléchargée. Cela crée un dossier, nommé **eclipse**, que nous vous conseillons de placer aussi haut que vous le pouvez dans la hiérarchie de fichiers de votre système.
Dans la suite de cette note nous supposons que vous avez fait ainsi et que vous avez donc un dossier nommé **C:\eclipse** .

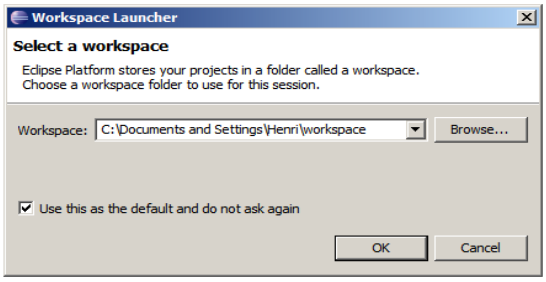
Pour faciliter le lancement d' *eclipse* créez un raccourci vers le fichier **C:\eclipse\eclipse.exe** et placez-le sur le bureau, dans le menu démarrer ou ailleurs, selon vos goûts.

L'installation d' *eclipse* est donc bien plus légère que celle de beaucoup de logiciels ; en particulier, sous Windows elle ne produit pas d'inscription dans la base de registres. Par conséquent, pour désinstaller complètement eclipse il suffira, le moment venu, de mettre à la corbeille le dossier **C:\eclipse** et les divers espaces de travail (dossiers **workspace**, voir ci-dessous) créés ultérieurement.

4. Premier lancement d'eclipse

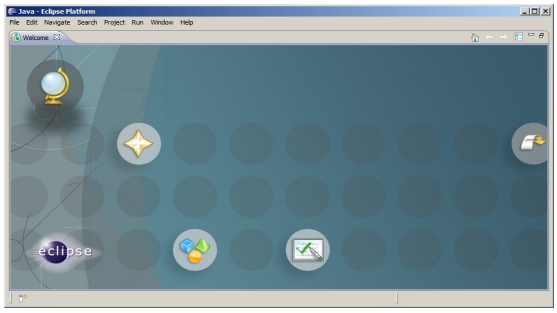
Lancez *eclipse*, par exemple en double-cliquant sur le raccourci que vous venez de créer. Au bout de quelques instants, on vous demandera de situer l'espace de travail dans lequel seront vos fichiers. Si vous travaillez sur un ordinateur partagé il est conseillé de mettre l'espace de travail dans votre dossier Documents (sur Windows XP il se trouve dans le dossier Documents and Settings). Si vous êtes le seul utilisateur de votre système, mettez l'espace de travail où bon vous semble.

Sauf indication contraire, les fichiers sources de vos programmes se trouveront dans l'espace de travail. Il est donc important de se souvenir de l'emplacement de ce dernier pour accéder aux sources (par exemple, pour les transporter, les copier, etc.)

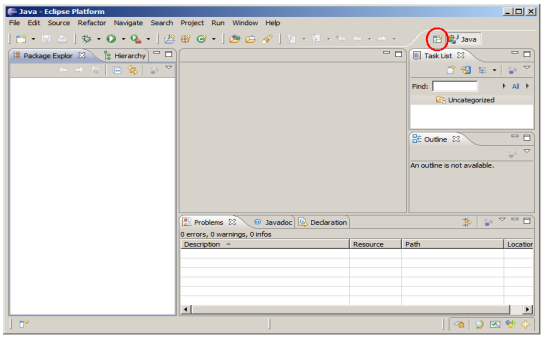


Si vous cochez la case « *Use this as the default and do not ask again* » *eclipse* ne vous posera plus cette question (mais il y a toujours un moyen pour changer ultérieurement l'espace de travail : *File* > *Switch Workspace* > *Other...*).

Au bout de quelques instants (la première fois ce n'est pas très rapide) vous obtenez un écran qui présente le produit, comme ceci :



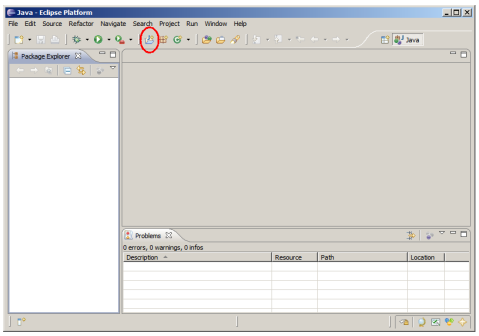
Vous pouvez feuilleter cette présentation, elle est faite pour cela. Quand vous en aurez assez, cliquez sur le lien *Workbench* (la flèche représentée à droite de l'écran). Le contenu de la fenêtre devient tout de suite beaucoup plus sérieux :



5. Configurer eclipse pour faire du Java

Eclipse est un environnement qui permet une grande variété d'activités de développement (pour vous en convaincre, faites un tour chez eclipse plugin central (Lien14), chez eclipse plugins (Lien15) ou bien sur le site francophone eclipsetotale.com (Lien16)). En standard, *eclipse* est prêt pour le développement en Java, encore faut-il veiller à ce que la *perspective* (c'est-à-dire l'arrangement des vues montrées à l'écran) soit celle qui convient le mieux à Java. Si ce n'est pas le cas, agissez sur la petite icône en haut à droite cerclée de rouge sur la figure 4, étiquetée *Open perspective* et choisissez *Java*.

Fermez les vues *Task List* et *Outline* (à droite) ; pour afficher la structure des classes, la vue *Package explorer* (à gauche) suffit. Vous obtenez un cadre de travail tout à fait commode pour développer en *Java* :

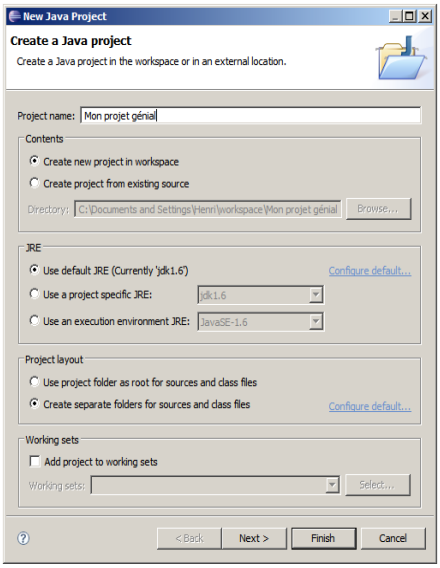


6. Développer un programme Java

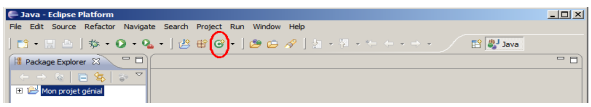
Pour commencer, créer un projet.

Note pour les étudiants. Ne vous sentez pas obligés de créer un nouveau projet chaque fois que vous commencez un nouvel exercice de programmation : vous pouvez très bien avoir un seul projet, contenant tous les exercices que vous faites dans le cadre d'un enseignement. D'autant plus que cela ne vous empêchera pas de bien ranger vos fichiers : un projet peut contenir plusieurs packages java (qui se traduiront dans le système de fichiers par des répertoires différents).

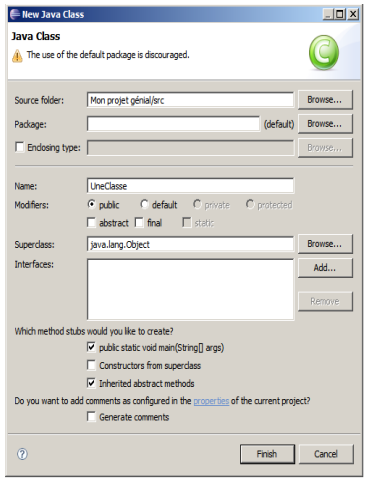
Pour créer un projet, cliquez sur le premier des boutons d'assistants Java (cerclé de rouge sur la figure 5) etiqueté « *New Java Project* ».
Vous obtenez le panneau *New Java Project* où, au minimum, vous devez donner un nom pour votre projet :



La méthode rapide consiste à donner un nom de projet (si possible, moins bête que *Mon projet génial*...) et cliquer sur le bouton *Finish*. Notez que les autres « questions » posées dans ce panneau sont intéressantes. La troisième, notamment, permet de conserver *séparément* les fichiers sources (précieux) et les fichiers classes (qu'en cas de perte on peut toujours refaire).

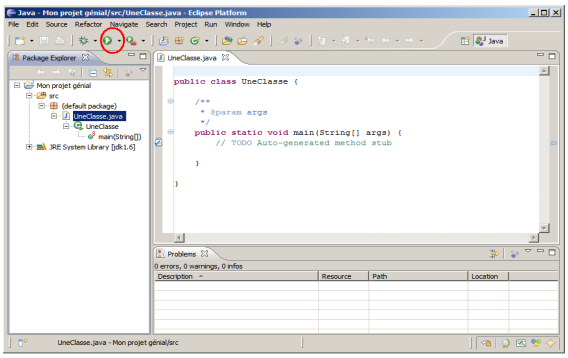


Dans un projet sérieux nous commencerions par créer des packages (deuxième bouton des assistants Java, « *New Java Package* »). Mais, puisque nous débutons, allons à l'essentiel et ajoutons directement une ou plusieurs classes au projet : c'est le troisième des boutons d'assistants Java, « *New Java Class* » (cerclé de rouge dans la figure 7), qui fait cela. La méthode rapide consiste à donner le nom de la classe et cocher la case étiquetée *public static void main(String[] args)* :



Notez qu'*eclipse* critique notre démarche, nous indiquant que *l'emploi du package par défaut (sans nom) est découragé*. Cela ne fait rien, nous construisons ici une application de débutant.

Eclipse crée alors un fichier source contenant une classe rudimentaire, correcte mais creuse, que vous n'avez plus qu'à compléter pour en faire le programme voulu :

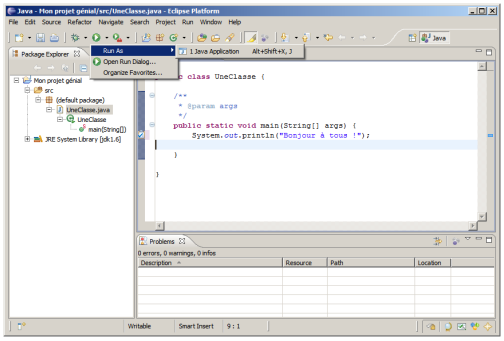


Note. Lorsqu'un commentaire contient l'expresson *TODO*, *eclipse* affiche une marque bleue dans la marge qui permet de se rendre rapidement à cet endroit. C'est très pratique pour retrouver dans les gros fichiers ces commentaires qui signalent des morceaux en chantier. Pour essayer votre programme vous allez taper le classique **System.out.println("Bonjour à tous!");** à l'intérieur de la fonction **main** . Au fur et à mesure que vous tapez, remarquez comment :

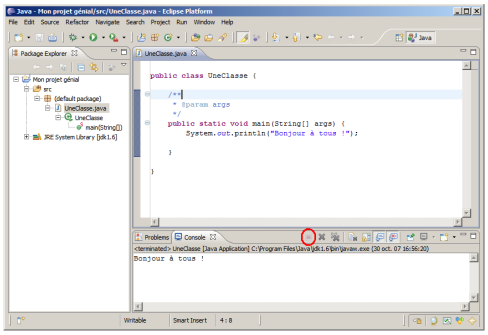
- la vue *Package Explorer* montre les packages (répertoires) qui composent votre projet, les classes que ces paquetages contiennent, les membres de ces classes, etc. Bien entendu, double cliquer sur une de ces entités vous positionne dessus dans le texte source.
- si vous marquez une pause lorsque vous tapez un

point, *eclipse* vous montre la liste de ce que vous pouvez taper ensuite,
- si vous laissez traîner le curseur sur un identificateur, *eclipse* affiche la documentation correspondante,
- si vous faites une faute *eclipse* vous la signale immédiatement et, dans le cas d'erreurs sémantiques, vous suggère des corrections,
- le simple fait de sauver le programme en provoque la compilation

Pour exécuter le programme assurez-vous que la vue éditeur contient une classe exécutable (c'est-à-dire une classe *publique* avec une méthode **public static void main(String[] args);**) et alors activez la commande *Run as > Java Application* du menu attaché au bouton cerclé de rouge sur la figure 9 :



L'application s'exécute et, si des sorties sont à afficher, une vue Console apparaît au-dessous de la vue éditeur :



Notez que dans la vue *Console* il y a un bouton, cerclé de rouge sur la figure 11, qui permet d'arrêter une application qui bouclerait indéfiniment. Ce bouton est rouge quand l'application est vivante, gris (estompé) lorsque l'application est morte.

7. Où sont mes fichiers sources ?

Cette question se pose par exemple lorsque, après avoir développé une application dans *eclipse*, vous souhaitez récupérer vos fichier sources pour les amener sur un autre système, les compiler dans un autre environnement ou tout simplement les ranger dans vos archives.

La réponse se trouve dans les figures 2 et 6 : si lors de la création du projet vous avez laissé l'option par défaut « *Create new project in workspace* » (cf. figure 6) alors les sources, rangés dans des dossiers correspondant aux packages, sont dans le dossier *workspace*, lui-même placé à l'endroit que vous avez indiqué au lancement d'*eclipse* (cf. figure 2).

8. Comment amener dans eclipse des fichiers créés ailleurs ?

Deux cas possibles : ces fichiers forment déjà un projet *eclipse* (par exemple créé sur un autre système), ou bien il ne s'agit que d'un ensemble de fichiers sources en vrac.

8.1. Vous avez déjà un projet eclipse

Copiez le dossier du projet *eclipse* où vous voulez (par exemple dans le dossier *workspace*, mais ce n'est pas une obligation), puis faites la commande *File > Import...* Ensuite, choisissez *General* puis *Existing Projects into Workspace*. Le projet que vous venez d'importer apparaît dans la fenêtre *Package explorer*, c'est terminé.

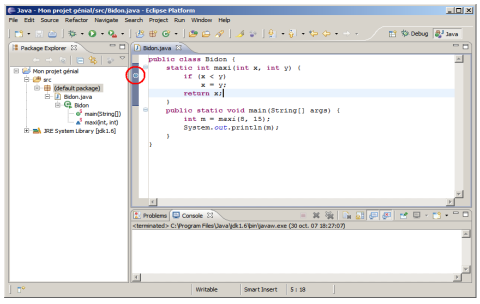
8.2. Vous n'avez qu'un ensemble de fichiers sources Java

Prenez un projet qui existe déjà, ou bien créez un noveau projet. Ensuite :
- soit, à l'aide de la commande *File > Import... > General > File system* ; vous naviguez à la recherche du(des) fichier(s) en question et vous les importez dans ce projet,
- soit, plus simplement : vous copiez les fichiers dans le dossier où sont les sources d'un des projets connus dans eclipse vous sélectionnez ce projet dans la vue Package Explorer vous exécutez la commande *File > Refresh*

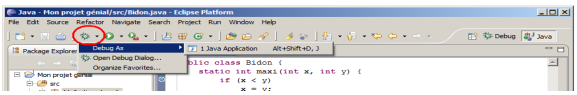
9. Déboguer les programmes

Un programme « bogué » est un programme qui ne donne pas les résultats qu'il devrait. « Déboguer » un programme c'est chercher les erreurs de programmation à l'origine de tels dysfonctionnements. Pour aider le programmeur dans cette recherche, eclipse offre un mode debug permettant, entre autres choses :
- la pose de marques, appelées *points d'arrêt*, sur des lignes du programme source, de telle manière que l'exécution s'arrêtera lorsque ces instructions seront atteintes,
- lors de tels arrêts, l'examen des valeurs qu'ont alors les variables locales et les membres des objets,
- à partir de là, l'exécution du programme pas à pas (c'est-à-dire ligne à ligne)

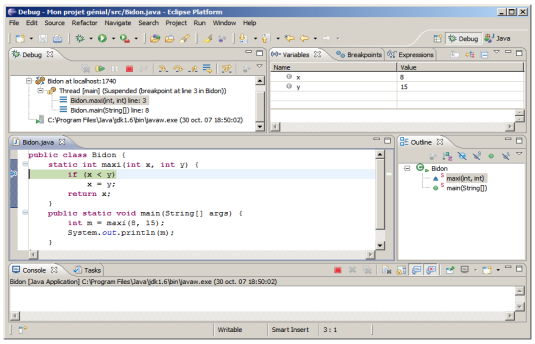
Pour déboguer simplement un programme il suffit de poser un point d'arrêt au début de l'endroit qu'on souhaite examiner en détail. Pour cela il faut double-cliquer dans la marge, à gauche de la ligne en question, ce qui fait apparaître un disque bleu (cerclé de rouge dans la figure 12) qui représente le point d'arrêt.



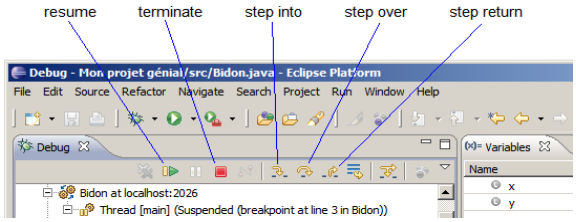
Il faut ensuite lancer le débogage, à l'aide du bouton à gauche de celui qui lance l'exécution, représentant une punaise (bug) :



L'exécution est alors lancée et se déroule normalement jusqu'à atteindre le point d'arrêt. *Eclipse* demande alors la permission de changer de *perspective* (ensemble et disposition des vues montrées) et adopte l'apparence de la figure 14 :



La vue *Debug*, en haut à gauche de la fenêtre, montre la *pile d'exécution*, c'est-à-dire, pour chaque thread, l'empilement des méthodes qui se sont mutuellement appelées (méthodes commencées et non terminées). Dans la figure 14, par exemple, on attire notre attention sur la méthode *Bidon.main*, plus précisément la ligne 8 du fichier source, où a été appelée la méthode *Bidon.maxi*, dans laquelle l'exécution est arretée, à la ligne 3.



En haut de cette vue (figure 15) se trouvent des boutons très utiles. Parmi les principaux :

- *step over* : faire avancer l'exécution d'une ligne. Si cette dernière contient un appel de méthode, *ne pas détailler* l'activation de celle-ci, c'est-à-dire considérer l'appel comme une instruction indivisible,
- *step into* : avancer l'exécution d'une ligne. Si un appel de méthode est concerné, détailler son activation, c'est-à-dire aller dans la méthode et s'arrêter sur sa première ligne,
- *step return* : relancer l'exécution normale, jusqu'à la fin de la méthode dans laquelle on est arrêté et le retour à la méthode qui a appelé celle-ci,
- *resume* : relancer l'exécution normale, jusqu'à la fin du programme ou le prochain point d'arrêt,
- *terminate* : terminer l'exécution.

En haut et à droite de la fenêtre principale se trouvent les vues *Variables* et *Expressions*. La première affiche les valeurs courantes des variables locales de la méthode en cours, la deuxième affiche les valeurs courantes des expressions sélectionnées avec la commande *Watch* (cliquer avec le bouton droit sur l'expression à surveiller).

Le debogueur d'*eclipse* possède bien d'autres commandes très puissantes, comme les *points d'arrêt conditionnels* et la possibilité de *modifier* les valeurs des variables du programme. Prenez un peu de temps pour les explorer, c'est payant.

10. Le « refactoring »

Réusiner (refactor) un programme correct c'est modifier son texte source sans changer son fonctionnement et ses résultats. Par exemple, changer le nom d'une variable ou d'une méthode parce que, par suite de l'évolution du programme (ou du programmeur), le nom initialement choisi est devenu moins adapté ou expressif qu'un autre.

Le *refactoring* se traduit généralement par des opérations globales, fastidieuses, qu'il n'est pas facile d'automatiser. Heureusement *eclipse* offre de puissantes fonctions pour effectuer ce travail. Voici leur description, directement traduite de l'aide en ligne du logiciel.

Commandes du menu Refactor : voir le tableau sur l'article en ligne (Lien17)

Retrouvez l'article d'Henri Garreta en ligne : Lien17

Les dernières news

Naissance de la rubrique Android avec son portail dynamique et communautaire

Nous avons le plaisir de vous annoncer la naissance de la rubrique Android : Lien18.

Vous trouverez sur le portail dédié à Android :

- Les actualités touchant à Android (aspects Mobile, développement, etc.)
- Une section "A la une" permet de mettre en évidence les news importantes ou d'actualité.
- Une section "Les news les plus lues" vous permet de retrouver les annonces ayant suscité le plus d'intérêt auprès des lecteurs
- Vous avez également la possibilité de proposer des annonces avec le lien "Proposer une actualité", par exemple pour un sujet du forum qui vous semble intéressant, un évènement à venir, etc.

En outre, vous trouverez des liens vers :
- les ressources Android (Lien19)
- le(s) forum(s) Android (Lien20)
- Un récapitulatif blogs Android (Lien21)

N'hésitez pas à être acteurs de ce portail et d'apporter votre soutien à la communauté francophone autour d'Android

Commentez cette news en ligne : Lien22

Spring par la pratique - Spring 2.5 et 3.0

Cet ouvrage montre comment développer des applications Java EE professionnelles performantes à l'aide du framework Spring. L'ouvrage présente les concepts sur lesquels reposent Spring (conteneur léger, injection de dépendances, programmation orienté aspect) avant de détailler les différentes facettes du développement d'applications d'entreprise avec Spring : couche présentation, persistance des données et gestion des transactions, intégration avec d'autres applications et sécurité applicative.

Cette seconde édition présente en détail les nouveautés majeures des versions 2.5 et 3.0 de Spring et de ses modules annexes : modèle de programmation basé sur les annotations, Spring Dynamic Modules for OSGi, Spring Batch, Spring Security, SpringSource dm Server, etc. L'accent est mis tout particulièrement sur les bonnes pratiques de conception et de développement, qui sont illustrées à travers une étude de cas détaillée, le projet Open Source Tudu Lists.

Critique du livre par Gildas Cuisinier

Spring par la pratique, première édition, fut un excellent livre sur Spring et son portfolio. A sa sortie, il couvrait directement Spring 2.0, fraichement sorti à l'époque. Mais l'eau a coulé sous les ponts et le livre est devenu un peu désuet de par les nouveautés de Spring 2.5 et 3.0 ou encore l'arrivée de projet tels que Spring Dm ou Spring Batch.. C'est là qu'intervient la nouvelle édition, couvrant justement ceux-ci, alors que Spring 3.0 n'est pas encore officiellement sorti.

Sur la forme, le livre garde le même style que l'ancienne édition, chaque chapitre étant divisé en deux parties. La première étant une présentation théorique d'un sujet, la deuxième étant sa mise en pratique dans un exemple concret (le même qu'auparavant : Tudu List). Concernant le contenu par contre, les changements sont majeurs, allant de grosse mises à jour à de nouveaux chapitres complets.

Ce qui change ?

Les chapitre concernant les bases de Spring (Inversion du controle, injection de dépendances, AOP) ont été complètement revus afin de prendre en compte les nouvelles possiblités de configuration apportées par Spring 2.5 et Spring 3.0 : annotations, expression language, nouveaux schéma XML.
La chapitre sur la partie Web à aussi ré-écrit afin de présenter le nouveau framework @MVC et ses annotations ainsi que le support de REST. C'est aussi le cas du chapitre Spring WebFlow, qui se base sur la version 2.0, ou sur la sécurité basé sur Spring Security et non plus sur Acegi.

Quoi de neuf ?

Parmi les nouveautés, plusieurs chapitres sont dédiés à OSGi et les solutions apportées par SpringSource : Spring Dynamic Modules qui rend le développement OSGi des plus simples, et Spring Dm Server qui propose une alternative à JEE en permettant le déploiement d'application OSGi.
Spring Batch, une solution de traitements de lots de fichiers en Java, se voit aussi consacrer un chapitre.

Une autre nouveauté est le nouveau site (Lien23) qui accompagne le livre qui permet de discuter avec les auteurs ou récupérer le code source.

Je conseille donc vivement la lecture de ce livre, aussi bien au débutant voulant apprendre à utiliser Spring en se basant sur la dernière version qu'a des personnes souhaitant simplement se mettre à jour et découvrir les nouveautés du portfolio. Le contenu est des plus complet et très bien expliqué !

Retrouvez ces critiques de livre sur la page livres Java : Lien24

Créer une application basée sur Zend Framework avec Zend_Tool

Cet article a pour objectif de vous donner une introduction à l'utilisation de Zend_Tool pour échafauder un projet basé sur Zend Framework

1. Introduction

Avec l'avènement de la version 1.8 du Zend Framework, vous avez certainement remarqué que plusieurs nouveaux composants sont venus embellir le Framework, parmi eux on distingue Zend_Tool qui est particulièrement différent des autres.

Sa particularité réside dans le fait que les autres composants sont destinés à agir au cœur de votre application pour vous faciliter certaines tâches d'exécution courante, ce qui n'est pas le cas avec Zend_Tool qui est un espace de nom qui regroupe un certain nombre de classes destinées à vous aider dans la construction du squelette de votre projet. Il peut être apparenté beaucoup plus à un outil de travail à l'instar de votre IDE.

2. Vue d'ensemble

Le développement d'application basée sur Zend Framework et particulièrement lorsque une architecture MVC est adoptée, nécessite au préalable de construire l'architecture initiale de votre projet (structure des répertoires, bootstraping...etc.) et d'y ajouter par la suite les modules, contrôleurs, actions et vues...etc.

Ces tâches qui prennent l'allure de routines, absorbent énormément de temps et qui sont sans intérêt d'usage pour votre projet, deviennent vite fastidieuses au fil du temps, c'est pour ce fait qu'est né le concept d'automatiser toutes ces tâches à l'aide d'outil interne ou externe au Framework, cette méthode de metaprogrammation est communément appelé le scaffolding (échafaudage).

Le scaffolding est une pratique vulgarisé par le Framework non PHP Ruby on Rails et adapté par la suite par d'autres Framework tel que CakePHP, Symphony...etc. et depuis bien longtemps, ce n'est que tardivement que Zend Framework l'a introduite avec l'outil Zend_Tool qui lui manquait considérablement.

3. Présentation de Zend_Tool

Zend_Tool a été développé dans la perspective de promouvoir l'aspect RAD du Framework afin d'offrir aux développeurs un gain de temps qui sera consacré aux parties utiles de leur code qui font la particularité et l'utilité de leurs projets.

Initialement l'outil a été développé pour être utilisé avec la ligne de commande, mais avec l'abstraction de la classe Zend_Tool_Framework_Client, il est désormais possible aux développeurs d'implémenter leur propre client avec des protocoles comme XML-RPC, SOAP...etc.

Depuis quelques temps déjà Zend Studio pour eclipse est le seul IDE à offrir un support et des options pour échafauder des projets basés sur Zend Framework. Mais avec la popularité grandissante de Zend Framework et l'introduction de Zend_tool on verra certainement dans les jours à venir d'autres IDE également proposer de telles fonctionnalités.

4. Installation et configuration de l'outil ligne de commande (CLI Tool).

Le CLI Tool (Commande ligne tool) est l'interface principale avec laquelle le développeur peut requêter avec Zend_tool via l'invite de commande, afin de pouvoir utiliser cet outil, vous devrez préalablement installer Zend Framework et faire une petite configuration.

Téléchargez la dernière version de Zend Framework si vous ne l'avez pas encore fait, décompressez le package vers le dossier de votre choix.

4.1. Installation sous Windows

Dans le package que vous avez décompressé et dans le répertoire bin, copiez les deux fichiers **zf.bat** et **zf.php** dans le même dossier où se trouve votre **php.exe**, qui se trouve généralement dans le dossier c:\wamp\bin\php\php5.x.x avec une installation WAMP soit php5.x.x votre version PHP.

Rajoutez le chemin vers votre php.exe dans votre variable d'environnement PATH, pour que la commande soit accessible où que vous soyez.

Mettez votre Library Zend Framework dans votre dossier PHP include_path de votre système, généralement dans le dossier C:\PHP5\Pear avec une installation WAMP. Si vous ne savez pas où se trouve votre include_path, vous pouvez facilement le récupérer avec la commande :

```
C:\> php -i | find " include_path "
```

Si pour une raison quelconque vous n'avez pas envie de mettre la librairie ZF dans votre include_path vous pouvez alternativement définir l'une des variables d'environnement ZEND_TOOL_INCLUDE_PATH_PREPEND ou ZEND_TOOL_INCLUDE_PATH vers le dossier où vous avez mis la librairie ZF.

4.2. Installation sous Linux

Dans le package que vous avez décompressé et dans le répertoire bin, copiez les deux fichiers zf.sh et zf.php dans

le même dossier où se trouve votre php.bin que vous pouvez retrouver avec la commande

```
mypc@user$ which php
```

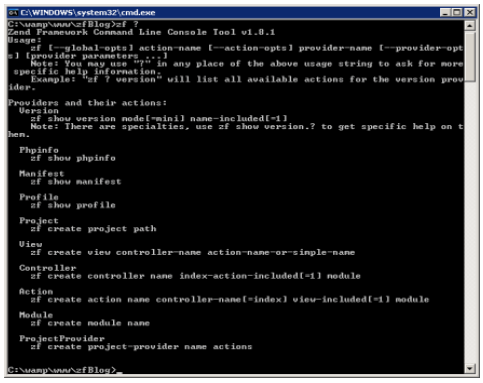
Mettez la Librairie Zend Framework dans votre dossier PHP include_path que vous pouvez retrouver avec la commande suivante

```
mypc@user$  Php  -i | grep include_path
```

5. Utilisation de la ligne de commande

Une fois que votre outil est bien configuré, vous pouvez afficher l'écran d'aide avec la commande suivante si vous obtenez l'écran suivant c'est que vous avez correctement configuré votre outil et pouvez commencer à l'utiliser.

```
C:\> zf --help
 ou
C:\> zf ?
```



Console help

5.1. Créer un projet

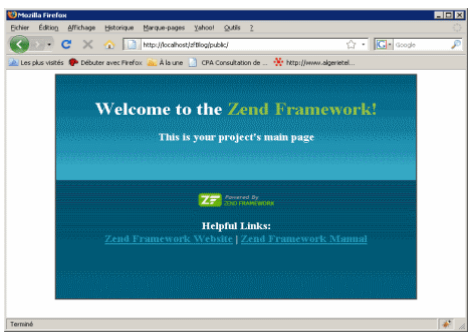
La première étape d'utilisation de Zend_Tool consiste à créer un projet, supposant que nous voulons créer un nouveau projet zfProject, dans l'invite de commande positionnez vous dans le dossier où vous voulez créer votre projet et tapez la commande suivante :

```
C:\> zf create project zfProject
```

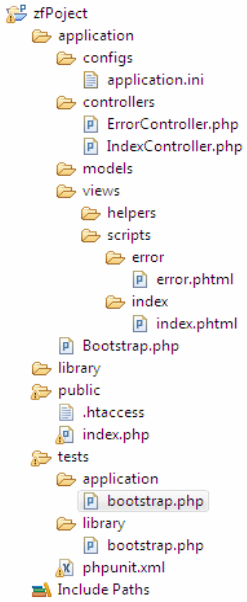
Comme vous pouvez directement indiquer le chemin complet où sera créé votre projet

```
C:\> zf create project  c:\wamp\www\zfProject
```

Essayez maintenant de voir le résultat dans votre navigateur, vous devrez avoir quelque chose qui ressemble à ceci.



Essayez maintenant de voir le contenu de notre dossier zfProject, vous devrez trouver une structure similaire a celle-ci

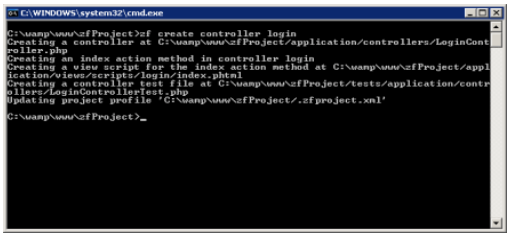


Remarquez que la commande create project nous a crée la structure des répertoires, le bootstrap, les contrôleurs index et Error avec les vues correspondantes. N'est-ce pas pratique.

5.2. Ajouter un contrôleur

Supposant que vous voulez ajouter un Contrôleur login a votre projet, comme pour la création de projet, il suffit de se positionner à la racine de votre projet et de taper la commande suivante :

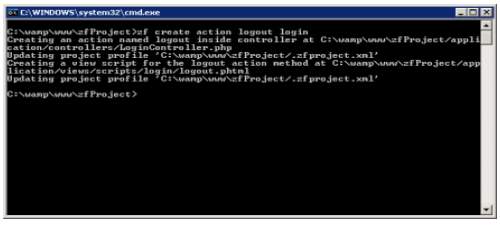
```
C:\> Zf create controller login
```



5.3. Ajouter une action

Supposant que maintenant vous voulez ajouter une action logout a votre Controller login précédemment créé, toujours dans l'invite de commande tapez la commande suivante :

```
C:\> zf create action logout login
```



Il est important à savoir que vous ne pouvez pas ajouter de composant (Modules, Contrôleurs, actions?) à un projet qui n'est pas initialement créé avec Zend_Tool, car lors de

sa création, sa structure est indexé dans un fichier caché .zfproject.xml qui permet à Zend_Tool_Project de gérer la hiérarchie des différentes ressources. Ce fichier est désigné comme étant le profil du projet et il est mis à jour à chaque fois qu'un nouveau composant vient s'ajouter au projet.

6. Conclusion

Je trouve que cet outil est une valeur ajoutée au Framework, puisqu'il permet non seulement d'offrir un gain de temps au développeur, mais favorise également le maintien d'une structure propre et commune aux projets basés sur Zend Framework.

Retrouvez l'article d'Idir Ait Yahia en ligne : Lien25

Présentation des nouveautés Zend Studio 7.0

Zend Studio est sans doute l'un des meilleurs IDE pour PHP. Dans cet article nous allons essayer de vous présenter les améliorations et les nouveautés de la version 7.0

1. Introduction

Zend Studio est l'environnement de développement par excellence des adeptes de PHP et de Zend Framework, la version 7.0 de Zend Studio vient d'être mise à la disposition de la communauté Zend. Comme de coutume cette version est une évaluation de 30 jours.

Vous pouvez récupérer librement la version de démonstration de la version 7.0 sur le site de Zend.

Téléchargez Zend Studio 7.0 (Lien26)

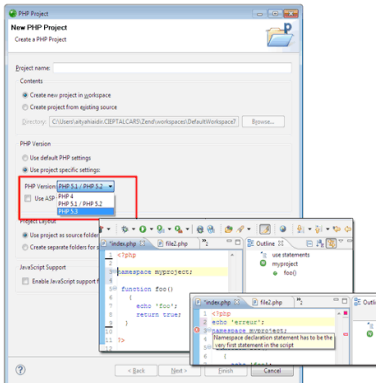
2. Nouveautés

Basée sur la toute dernière version d'Eclipse Galileo (3.5), la nouvelle version de Zend Studio apporte un tas de nouveautés hormis celle de la plateforme Galileo.

2.1. Support de PHP 5.3

Zend studio est d'ores et déjà prêt à accueillir la future version de PHP en offrant une aide à la saisie, la coloration syntaxique,les fonctions lambdas, la gestion des espaces de nom et le débogage.

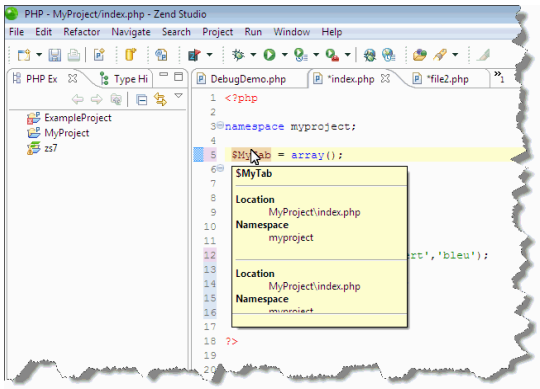
Pour avoir un aperçu d'ensemble sur les nouveautés de PHP 5.3, je vous invite à lire cet excellent article PHP 5.3 : nouveautés, migration (Lien27)



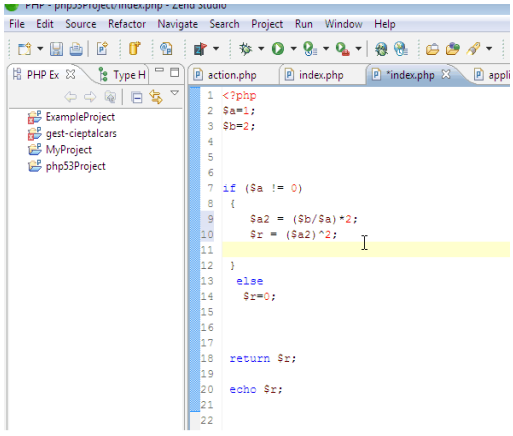
2.2. Amélioration de l'édition de code

De nombreuses fonctionnalités ont été ajoutées afin améliorer l'édition de code.

Amélioration de l'outil de refactoring qui nous permet de renommer rapidement toutes les occurrences d'un élément comme vous pouvez le voir dans l'animation ci-dessous.

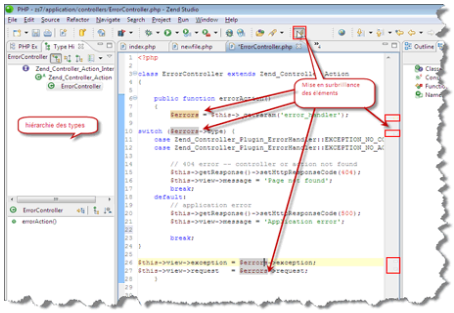


Création de variables ou de fonctions à partir de bloc de code déterminé avec la fonction **Extract variable /Extract Method**.

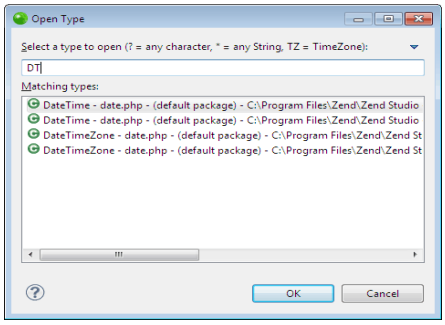


Mise en surbrillance de toutes les occurrences d'un élément de code.

Hiérarchie des types structurés.



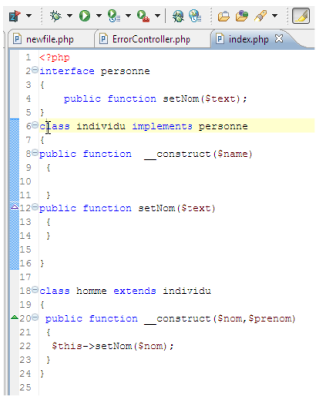
Open Type et Open Method sont deux nouveaux outils, accessibles depuis le Menu Navigation, qui vont vous permettre de chercher des types ou méthodes avec des mots clefs substitués ou en CamelCase (exp: DT pour DateTime).



L'assistance à la saisie a été améliorée et supporte la casse en CamelCase.

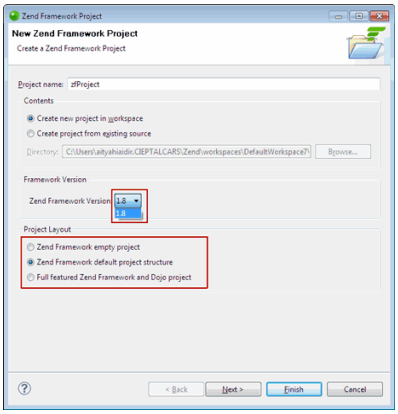
Deux nouveaux indicateurs ont fait leur apparition dans cette nouvelle version : L'Indicateur de surcharge et l'implémentation de méthodes. Ces indicateurs sont affichés sous forme de triangle dans la barre d'indication.

Le triangle vert désigne une méthode surchargée quand au blanc, une méthode implémentée.

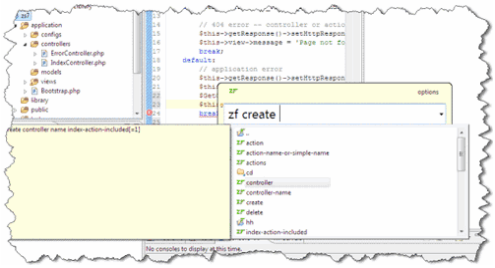


2.3. Support de Zend Framework

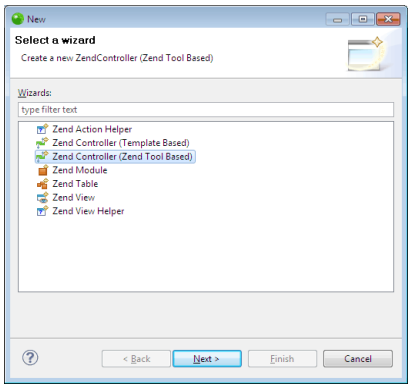
Le support de Zend Framework n'est pas une nouveauté pour Zend Studio, mais avec cette nouvelle version nous avons en plus l'intégration avec Zend_Tool et le support de la version 1.8. Les anciennes versions du Framework ne sont plus supportées.



Vous pouvez utiliser Zend_Tool soit en mode console soit en mode GUI. La console zf est accessible par la combinaison de touches Ctrl+2 ou par le menu *Project | Zend Tool*



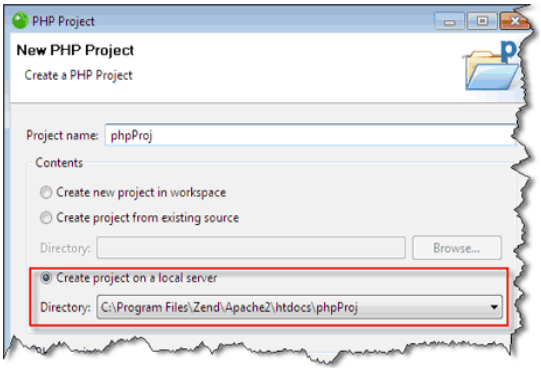
Comme vous pouvez le voir sur l'image ci-dessous, pour la création d'un contrôleur nous avons le choix d'utiliser zend_tool ou de se basé sur un modèle.



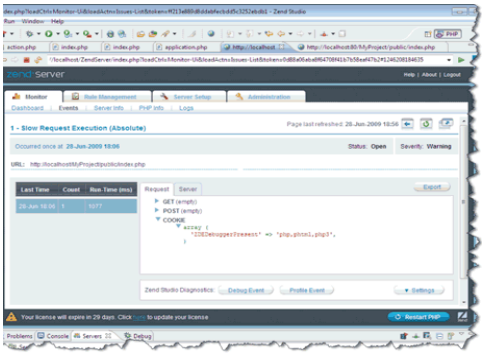
2.4. Intégration avec Zend Server

Zend Studio détecte automatiquement l'installation de Zend Server et configure l'environnement pour l'intégration sans aucune intervention de cotre part.

Avec ce nouveau duo, le déploiement et le débogage d'applications sur votre serveur local ne sera jamais aussi simple.



Vous pouvez directement afficher et déboguer les événements enregistrés par le serveur. Pour plus de détails sur cette fonctionnalité, je vous invite à visionner la vidéo *detecting and resloving application problem's* dans la rubrique liens.

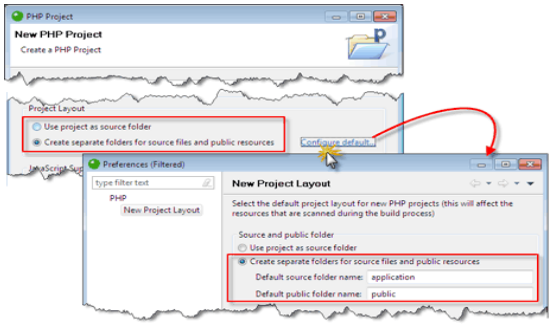


2.5. Amélioration des performances

Amélioration des performances exprimée par la réduction de la mémoire footprint (ressource mémoire utilisée lors de l'exécution), suppression des dépendances inutilisées de la plateforme et l'utilisation de l'architecture du cache et de l'indexation d'Eclipse

2.6. Autres

Il vous est possible de définir une structure basique pour tous vos nouveaux projets PHP.



3. Conclusion

Sur l'ensemble des nouvelles améliorations et fonctionnalités, le support de PHP 5.3 et l'intégration avec Zend Server me semblent les plus importants. Les autres restent des améliorations de surface, certe utiles mais facultatives.

Retrouvez l'article d'Idir Ait Yahia en ligne : Lien28

Le système de build de Dojo

Cet article décrit le système de build de Dojo permettant d'optimiser les performances d'une application Dojo. Il explique les raisons du build et en donne un exemple en utilisant Ant.

1. Introduction

Dojo est un framework javascript Open Source très complet. J'ai rédigé un précédent article (Lien29) décrivant ce framework.

Dans sa dernière release (1.3), Dojo contient de très très nombreux fichiers :

Type de fichier	Quantité
Javascript	1513
Style css	200
Gif / Jpg	210
HTML	730
Autre	581
Total	3234

Cette multiplicité de fichiers est due au fait que chaque classe et chaque widget dans Dojo est décrit dans un fichier javascript. En plus du fichier javascript, un widget peut être accompagné d'un template (fichier HTML), d'une feuille de style (fichier css), de fichiers javascript pour l'internationalisation et éventuellement d'images.

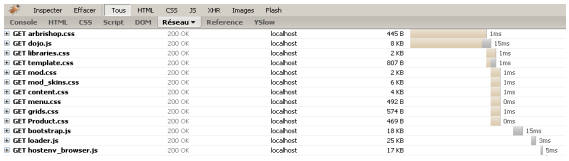
2. Focus sur les performances des applications WEB 2.0

Ce paragraphe n'a pas l'ambition de faire le tour de la problématique de performance pour les applications Web 2.0 (plusieurs articles n'y suffiraient pas) mais va mettre en lumière une problématique importante des applications Web 2.0 : le nombre très important de fichiers à charger sur le navigateur de l'utilisateur.

Tout d'abord, il faut bien comprendre de quelle manière est architecturée une application Web 2.0 réalisée en JavaScript : le fichier HTML qui contient normalement le contenu de la page n'est utilisé que pour charger un noyau d'application en JavaScript. Ce noyau va ensuite charger différents fichiers JavaScript qui vont générer l'interface utilisateur.

Au niveau de Dojo, le chargement des différents widgets est assuré par une instruction "dojo.require('nom de la classe');" à chaque appel, cette instruction va effectuer une requête HTTP au serveur pour obtenir les fichiers JavaScript qui ne sont pas dans le cache du navigateur. Bien entendu, pour chaque fichier demandé (CSS, image,...) une nouvelle requête sera effectuée.

L'image suivante illustre la situation avec un simple chargement de Dojo. Firebug est utilisé pour tracer les éléments chargés.



Trace réseau du chargement de Dojo

La masse de fichiers à charger induit des temps de chargement prohibitifs d'une simple application Dojo : de l'ordre de 1 seconde sur un poste en local mais plusieurs secondes sur un vrai serveur distant !

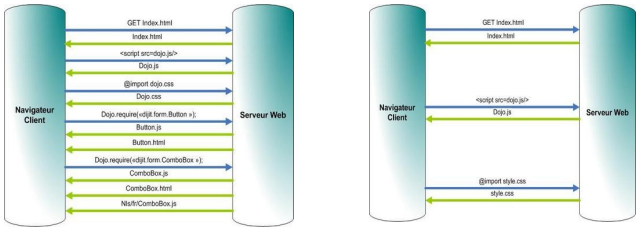
Cette problématique a bien évidemment été prise en compte par les équipes de développement de Dojo et leur solution a été la création du système de build de Dojo.

3. Description du build de Dojo

Le build de dojo a plusieurs objectifs :

- Compresser les fichiers JavaScript
- Grouper plusieurs fichiers JavaScript en un seul
- Internaliser les fichiers non JavaScript (template HTML, fichiers d'internationalisation)
- Regrouper les fichiers CSS contenant la clause @import
- Créer une release de votre application

Les deux images suivantes illustrent les différences entre une application Web 2.0 non buildée et une application Web 2.0 buildée :



Comme vous pouvez le constater, la différence est flagrante ! On passe du chargement d'une centaine de fichiers au chargement de trois fichiers. Bien entendu, certaines ressources comme les images ne pourront pas être buildées.

4. Description du fonctionnement du système de build

Le système de build inclus une version customisée de Rhino, l'interpréteur JavaScript de Mozilla. Cette version est modifiée pour permettre la compression de code JavaScript. Etant donné que cet interpréteur est codé en

Java, il est nécessaire d'avoir un JRE sur son poste (> 1.4.2).

En plus de Rhino, le système de build comprend un programme JavaScript nommé "build". Ce programme a pour but la concaténation des JavaScripts en un seul fichier.

Ces deux outils se situent dans le répertoire /util de dojo : util/shrinksafe/shrinksafe.jar et util/buildscript/build.js

En sortie de build, plusieurs actions auront été réalisées :

- Création d'un répertoire pour la version
- Copie sélective de fichiers dans le répertoire de la version
- Concaténation des fichiers en un seul fichier
- Compression du fichier JavaScript

Pour fonctionner, le build se base sur un fichier de profile au format JSON contenant :

- Le nom du fichier JavaScript en sortie (name),
- Les widgets à prendre en compte (dependencies)
- Les répertoires à prendre en compte (prefixes)

Exemple de fichier profile

```
dependencies = {
  layers: [
  {
    name: "../dojox/storage/storage-browser.js",
    layerDependencies: [
    ],
    dependencies: [
      "dojox.storage",
      "dojox.storage.GearsStorageProvider",
      "dojox.storage.WhatWGStorageProvider",
      "dojox.storage.FlashStorageProvider",
      "dojox.flash"
    ]
  }
  ],

  prefixes: [
    [ "dijit", "../dijit" ],
    [ "dojox", "../dojox" ]
  ]
}
```

Pour lancer le build, la ligne de commande suivante doit être utilisée:

Ligne de commande pour le build (avec des retours chariots pour la lisibilité)

```
java -classpath
../shrinksafe/js.jar;../shrinksafe/shrinksafe.jar
org.mozilla.javascript.tools.shell.Main build.js
        action=clean,release,
        dojodir=C:\Prog\Dojo\dojo-release-1.3.0-
src,
        profileFile=C:\Prog\Dojo\dojo-release-
1.3.0-
src\util\buildscripts\profiles\storage.profile.js
,
        releaseDir=C:\Prog\Dojo\dojo-release-
1.3.0-src/release,
        version=0.0.0.dev,
        optimize=none
```

Cette mise en œuvre manuelle étant fastidieuse, je vais maintenant vous proposer une mise en œuvre avec Ant.

5. Mise en œuvre avec Ant

Ant est un outil de construction d'application écrit en Java. Je ne vais pas le décrire plus avant car il y a un excellent tutoriel (Lien30) qui vous expliquera ça dans le détail.

Afin d'illustrer le build, on va réaliser un projet Dojo à l'aide d'Eclipse et on réalisera le script Ant permettant d'automatiser le build.

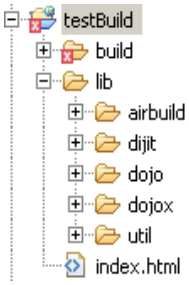
5.1. Création du projet Eclipse

Si vous avez suivi mon précédent article (Lien29), la création d'un projet Dojo ne doit plus avoir de secret pour vous.

Ici par contre, on va devoir modifier légèrement la procédure pour ne pas prendre la librairie proposée par défaut par Aptana. En effet, celle-ci est une librairie partielle ne comprenant pas tous les outils de build de Dojo. Il faut utiliser la version source de Dojo.

On peut la trouver à l'adresse suivante : Téléchargement de Dojo source 1.3.1 (Lien31)

Pour créer le projet d'exemple, créez un simple projet Web à l'aide d'Aptana (Default Web Project) sans préciser de bibliothèque Ajax. Ensuite décompressez la version source de Dojo dans un répertoire lib de votre projet. Enfin créez un répertoire build.



Répertoire projet que vous devez obtenir

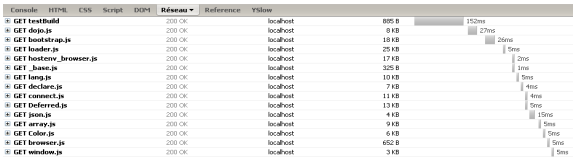
On va coder un petit fichier HTML permettant l'affichage de 2 widgets Dijit.

fichier index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Projet de test de build</title>
        <!-- Import du source de Dojo -->
        <script  type="text/javascript"
                src="lib/dojo/dojo.js"
djConfig="parseOnLoad:  true">
        </script>
        <!-- Import des css-->
        <style type="text/css">
        @import
"lib/dijit/themes/tundra/tundra.css";
        @import "lib/dojo/resources/dojo.css";
    </style>
        <!-- Import des bibliothèques nécéssaires
```

```
à l'affichage des widgets -->
        <script  type="text/javascript">
                dojo.require("dijit.form.Button")
;
                dojo.require("dijit.form.CheckBox
");
        </script>
</head>
<body  class="tundra">
        <div  dojoType="dijit.form.Button">Click
Button</div>
        <div>Une checkBox</div>
        <div
dojoType="dijit.form.CheckBox"></div>
</body>
</html>
```

Un rapide coup d'œil sur Firebug après l'exécution de la page :



Pas terrible : 79 requêtes, 1,3 secondes et 588 Kb sur mon poste.

5.2. Création du fichier de profile

Notre application étant très simple, le fichier de profil le sera également :

Le fichier profile.json dans le répertoire build

```
/**
 * @author Mikael Morvan
 * 20 juin 2009
 */
dependencies = {
  layers: [
  {
    name: "dojo.js",
    layerDependencies: [
    ],
    dependencies: [
      "dijit.form.Button",
      "dijit.form.CheckBox"

    ]
  }
  ],

  prefixes: [
    [ "dijit", "../dijit" ],
    [ "dojox", "../dojox"]
  ]
}
```

Comme vous pouvez le constater, les deux widgets que j'utilise sont ajoutés à la section dependencies. Le nom de mon build "dojo.js" sera mixé avec le "dojo.js" de la bibliothèque de base et je n'aurai donc qu'un seul fichier JavaScript à charger ! La section "prefixes" est utilisée dans le cas où vous avez une bibliothèque personnelle.

5.3. Création du fichier Ant

Avec Eclipse, il faut commencer par ouvrir la vue Ant (Menu Window/Show View/Ant). Créez un nouveau fichier xml nommé "build.xml" et faites un "drag and drop" de la vue fichier vers la vue Ant. (Normalement votre vue Ant indiquera une erreur puisque votre fichier "build.xml" est mal construit).

J'ai séparé mon "projet Ant" en trois parties : la première partie permet de lancer le système de build de Dojo, la deuxième effectue une copie sélective des fichiers "buildés" vers un répertoire final et enfin la dernière partie effectue un peu de nettoyage.

Le fichier build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ======================================
  20 juin 2009

  testBuild
  Build du projet testBuild

  Mikaël Morvan
     ====================================== -->
<project name="testBuild" default="nettoyage"
basedir=".." >
  <description>
    Build du projet d'exemple
  </description>

  <property name="build.name" value="buildDojo"/>
  <property name="buildscripts.dir" value="$
{basedir}/lib/util/buildscripts/"/>
  <property name="release.dir" value="${basedir}/
build/release/${build.name}/"/>
  <property name="final.dir" value="$
{basedir}/build/lib/"/>

    <!-- ================================
          target: packaging
         ================================ -->

  <target name="packaging" depends=""
description="Build du projet">
    <java
     classname="org.mozilla.javascript.tools.shel
l.Main"
     dir="${buildscripts.dir}"
     fork="true"
     failonerror="true"
     maxmemory="128m">
      <arg line="build.js"/>
      <arg line="action=clean,release"/>
      <arg line="dojodir=${basedir}/lib/"/>
      <arg line="profileFile=$
{basedir}/build/profile.json"/>
      <arg line="releaseName=${build.name}"/>
      <arg line="releaseDir=$
{basedir}/build/release/"/>
      <arg line="version=1.0.0.dev"/>
      <arg line="cssOptimize=comments"/>
      <arg line="layerOptimize=shrinksafe"/>
      <arg line="copyTests=false"/>
      <classpath>
        <pathelement location="$
{buildscripts.dir}\..\shrinksafe\shrinksafe.jar"/
>
        <pathelement location="$
{buildscripts.dir}\..\shrinksafe\js.jar"/>
```

```
        <pathelement path="${java.class.path}"/>
      </classpath>

    </java>
  </target>

        <!-- - - - - - - - - - - - - - - - - -
          target: optimisation
         - - - - - - - - - - - - - - - - - - -->
  <target name="optimisation"
depends="packaging" description="optimisation de
la release">

    <delete dir="${final.dir}"/>
    <mkdir dir="${final.dir}"/>

    <!-- Répertoire dojo -->
    <mkdir dir="${final.dir}dojo"/>
    <copy file="${release.dir}dojo/dojo.js"
todir="${final.dir}dojo/"/>
    <!-- Uniquement si on veux pouvoir utiliser
firebug light-->
    <copy todir="${final.dir}dojo/_firebug/">
      <fileset dir="$
{release.dir}dojo/_firebug/"/>
    </copy>
    <!-- Les images et les css doivent être
gargées-->
    <copy todir="${final.dir}dojo/resources/">
      <fileset dir="$
{release.dir}dojo/resources/">
        <include name="**/*.css"/>
        <include name="**/*.png"/>
        <include name="**/*.gif"/>
        <!-- Les fichiers css non optimisés ont
été gardés dans l'arborescence -->
        <exclude name="**/*.commented.css"/>
      </fileset>
    </copy>

    <!-- Répertoire dijit -->
    <mkdir dir="${final.dir}dijit"/>
    <copy todir="${final.dir}dijit/themes/">
      <!-- On ne copie que les themes et rien
d'autre (tout a été internalisé dans dojo.js)-->
      <fileset dir="${release.dir}dijit/themes/">
        <exclude name="**/*.commented.css"/>
        <exclude name="**/*.psd"/>
      </fileset>
    </copy>

    <!-- Répertoire dojox -->
    <!-- A ne garder que si c'est nécessaire-->
    <!--
      <mkdir dir="${final.dir}dojox"/>
      -->
    <!-- Copie des css et des images uniquement
-->
    <!--
    <copy todir="${final.dir}dojox/">
      <fileset dir="${release.dir}dojox/">
        <include name="**/*.css"/>
        <include name="**/*.png"/>
        <include name="**/*.gif"/>
        <include name="**/*.jpg"/>
        <exclude name="**/*.commented.css"/>
      </fileset>
    </copy>
    -->
    <!-- Nettoyage des fichiers utilisés pour les
tests -->
    <!--
    <delete dir="${final.dir}dojox/data/"/>
    <delete dir="${final.dir}dojox/fx/"/>
    <delete dir="${final.dir}dojox/gfx/"/>
    <delete dir="${final.dir}dojox/grid/tests/"/>
    <delete dir="${final.dir}dojox/image/tests/"/
>
    <delete dir="$
{final.dir}dojox/layout/tests/"/>
    <delete dir="$
{final.dir}dojox/widget/tests/"/>
    <delete dir="${final.dir}dojox/wire/"/>
    -->
  </target>

  <!-- - - - - - - - - - - - - - - - - -
    target: nettoyage
    - - - - - - - - - - - - - - - - - -->
  <target name="nettoyage" depends="optimisation"
description="nettoyage de la distribution">
    <delete dir="${release.dir}"/>
  </target>

</project>
```

Un petit focus sur la partie "Optimisation" : il faut tout d'abord comprendre que le build de Dojo copie l'ensemble des fichiers de la source dans le répertoire de sortie. Donc, si jamais on a oublié de builder un fichier, il est présent malgré tout dans la version buildée. Pour ma part, je pense que la version buildée doit être optimisée et que la phase de test est là pour palier à l'oubli d'inclusion d'un widget dans le fichier de profil. C'est pourquoi j'ai réalisé cette partie optimisation.

En premier lieu, le fichier "dojo.js" est copié vers la destination : c'est ce seul fichier JavaSript qui comprend tout notre code source. Même les fichiers de template HTML ont été internalisés.

Ensuite, un recopie sélective des ressources CSS, PNG et GIF est réalisée pour le répertoire "dojo". On exclu les fichiers terminant par "commented.css" car ce sont les fichiers css non optimisés qui sont gardés par le build.

Pour le répertoire "Dijit", c'est très simple : tous les fichiers ont été internalisés dans "dojo.js" dont il ne reste que les ressources css, png et gif situés dans le répertoire "themes". Et c'est tout !

Pour finir, les ressources du répertoire "Dojox" peuvent être copiées si nécessaire.

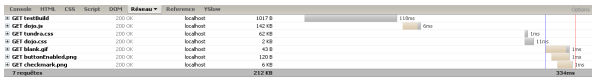
5.4. Lancement du projet optimisé

Une fois le fichier buildé, un nouveau répertoire /build/lib contient notre bibliothèque Dojo optimisée. On va donc modifier notre fichier HTML afin de le prendre en compte :

Le fichier build.xml

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
  <title>Projet de test de build (version
optimisée)</title>
  <!-- Import du source optimisé de Dojo -->
  <script  type="text/javascript"
                src="build/lib/dojo/dojo.js"
djConfig="parseOnLoad:  true">
  </script>
  <!-- Import des css-->
  <style type="text/css">
    @import
"build/lib/dijit/themes/tundra/tundra.css";
    @import "build/lib/dojo/resources/dojo.css";
  </style>
  <!-- Import des bibliothèques nécéssaires à
l'affichage des widgets -->
  <!-- Dans cette version optimisée, il n'y aura
pas de chargement de JavaScript -->
  <script  type="text/javascript">
    dojo.require("dijit.form.Button");
    dojo.require("dijit.form.CheckBox");
  </script>
</head>
<body  class="tundra">
  <div  dojoType="dijit.form.Button">Click
Button</div>
  <div>Une checkBox</div>
  <div  dojoType="dijit.form.CheckBox"></div>
</body>
</html>
```



Et voilà ! 7 requêtes, 334 ms et 212 Kb sur mon poste. L'optimisation est bien réelle et le gain de performance en production sera vraiment énorme.

6. Conclusion

Comme vous avez pu le constater, le build de Dojo est vraiment nécessaire si on veut obtenir une application performante. La mise en œuvre à l'aide de Ant est simple et permet, dans le cadre d'un projet réel, d'automatiser le build (l'intégration continue peut même être envisagée).

Le système de build trouve un intérêt supplémentaire dans le cas où on crée des widgets personnalisés. En effet, dans ce cas là, les widgets pourront également être buildés de la même manière que les widgets de Dijit ou Dojox. On pourra dès lors obtenir une application Dojo personnalisée et performante.

Mais pour personnaliser cette application, il faudra apprendre à réaliser des widgets personnalisés. Ce sera le sujet d'un futur article ...

7. Le code source

Voici le code source pour refaire les exemples du système de build : Lien32

Retrouvez l'article de Mikaël Morvan en ligne : Lien33

Comprendre les mécanismes d'AJAX

AJAX est encore trop souvent mal compris par les débutants. Qu'il s'agisse de sa mise en œuvre, de ses capacités ou de sa nature même. Cet article va tenter de dégonfler les différentes baudruches concernant cette "technologie". Nous verrons aussi en quoi AJAX se différencie des différentes techniques d'inclusion de contenu.

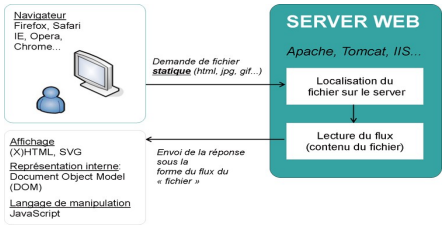
1. Préambule

Cet article n'a pas pour objet de vous apprendre à utiliser AJAX, mais de vous faire comprendre ses mécanismes.
Pour ceux qui souhaitent en savoir plus, je vous invite à consulter :

- Les tutoriels AJAX de developpez.com (Lien34).
- En particulier, pour les débutants et en complément de cet article : Ajax : Vos premiers pas dans les nouvelles technologies (Lien35) et Web 2.0, allez plus loin avec AJAX et XMLHttpRequest (Lien36).
- La FAQ AJAX (Lien37).

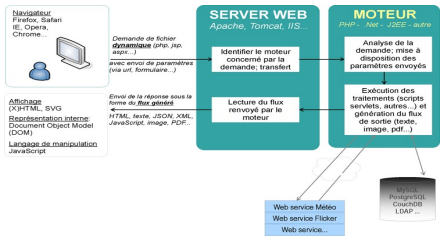
Avant toute chose, il est important de bien comprendre comment est architecturée une communication client / serveur.
Il existe deux types de communication entre le client (habituellement, le navigateur) et le serveur, les communications dites "statiques", c'est-à-dire qui se contentent d'envoyer vers le client un flux de données présentes sur le serveur et les communications dites "dynamiques" qui effectuent un traitement sur le serveur pour renvoyer au client le résultat de ce traitement.
Un schéma valant mieux que de longs discours, voici des représentations des architectures client / serveur statiques et dynamiques :

Schéma représentant le fonctionnement d'un site statique :



Relation client / serveur statique

Schéma représentant le fonctionnement d'un site dynamique :



Relation client / serveur dynamique

2. AJAX : késako ?

Ce qui fait généralement le plus peur quand on commence à aborder AJAX, c'est l'apparente complexité pour obtenir une instance *XMLHttpRequest*. *XMLHttpRequest* est la clé de voûte d'une requête AJAX, sans lequel il est impossible de faire quoi que ce soit.
Historiquement, la technologie AJAX est apparue en 1999 avec IE5 ; à l'époque aucune norme AJAX du W3C n'existe et donc aucune notion d'objet *XMLHttpRequest*. Microsoft choisit alors de s'appuyer sur sa technologie *ActiveX MSXML*. Un an et demi plus tard Mozilla propose l'objet *XMLHttpRequest*, dont les noms des fonctions sont calqués sur celles de l'*ActiveX* de IE5 pour conserver un maximum de compatibilité entre les navigateurs. De facto *XMLHttRequest* devient une norme (Safari, Opera, Chrome l'adoptent) et c'est en 2006 que le W3C publie ses premières spécifications. L'objet *XMLHttpRequest* apparait ainsi dans IE7. Une seconde version des spécifications est finalement publiée en 2008.
Toujours est-il que la complexité n'est qu'apparente et que l'implémentation se fait assez facilement (voir un exemple (Lien38)).

Cependant, AJAX n'est rien d'autre qu'un acronyme ! Pour bien le comprendre, décomposons-le :
AJAX : Asynchronous JavaScript And XML, soit en français : JavaScript et XML asynchrones. Cependant, nous allons voir que les termes employés sont parfois là uniquement pour générer l'acronyme !

- **Asynchrone** : il s'agit de la capacité de communiquer avec une ressource coté serveur sans figer le navigateur (c'est-à-dire que le reste de la page reste actif), il est à noter que bien que l'aspect asynchrone d'une requête Ajax en fait souvent un avantage non négligeable, cela ne reste qu'une option, il est tout à fait possible de créer des requêtes synchrones, qui bloqueront donc votre page tant que le serveur n'aura pas répondu.
- **JavaScript** : et bien oui, AJAX n'est rien d'autre que l'utilisation de l'objet (ou ActiveX) XMLHttpRequest de JavaScript. Tout ce que vous aurez à savoir pour créer une requête Ajax efficace, ce seront les différentes propriétés de cet objet.
- **Et XML** : si le XML se justifiait à l'époque des premières évocations d'AJAX (la mode était au XML et les formats alternatifs comme JSON n'existaient pas), il est aujourd'hui controversé. Toujours est-il que le retour d'une requête AJAX est supposé être soit du texte, soit du XML, on

comprend donc bien que ce X est surtout là pour l'acronyme !

3. Mécanisme de création d'une page HTML

Avant de rentrer plus en détail dans le déroulement d'une requête AJAX, il est important d'avoir une idée claire du déroulement de création d'une page Web et en particulier de la communication client / serveur.
Dans cet article, j'ai pris comme référence de langage serveur le PHP. Sachez que c'est un choix arbitraire et que tout ce qui suit est valable **quel que soit** le langage serveur utilisé !

Lorsqu'un internaute demande une page Web via son navigateur, il va chercher un fichier sur un serveur. Une connexion est donc ouverte vers le serveur pour pouvoir communiquer avec lui. Si le fichier en question est identifié comme du HTML (par son extension essentiellement), alors, il "comprend" que le résultat sera directement exploitable par le navigateur et renvoie donc le contenu du fichier au navigateur qui ferme la connexion.
En revanche, s'il comprend que le contenu est du PHP (j'insiste, ou tout autre langage serveur), alors, il sait qu'il doit d'abord interpréter le code contenu dans le fichier pour pouvoir renvoyer au navigateurs du code HTML compréhensible par le navigateur, que l'on appelle du coup "généré". Bien entendu, quand je dis que le code généré sera compris par le navigateur, cela implique que le développeur a bien codé sa page !
Une fois le code interprété, le résultat est envoyé vers le navigateur qui l'affiche, sans savoir qu'il s'agit d'un document PHP, ce qui du reste ne servirait à rien car un navigateur ne sait pas évaluer du code PHP, puis ferme la connexion.

4. Interaction PHP / JavaScript

Les notions abordées ici se limitent aux cas des requêtes HTTP de type GET ou POST (les plus courantes), s'il existe des méthodes autorisant les connexions dites persistantes, elles dépassent largement le cadre de cet article.

JavaScript (dans le cadre d'une requête AJAX) est lui interprété sur le navigateur.
La conséquence du fonctionnement décrit précédemment est que les interpréteurs PHP et JavaScript ne fonctionnent **jamais** en même temps !
En effet, lorsque le serveur interprète le PHP, le document HTML n'existe pas, il n'existera qu'une fois le contenu reçu et affiché sur le navigateur, donc une fois la connexion client / serveur fermée. A l'inverse, lorsque l'interpréteur JavaScript entre en action, c'est-à-dire dès que le navigateur aura rencontré une balise *<script>* alors le document PHP n'existe plus.

Il est donc absolument impossible de faire interagir PHP et JavaScript !

5. Comment ça impossible !?

Je vous entends déjà pester ! Comment ça c'est impossible, je l'ai pourtant déjà fait !
Et bien non, c'est faux ! Vous avez déjà certainement transmis des données du client (navigateur) vers le serveur, en effet, JavaScript est exécuté coté client, PHP coté serveur, le client et le serveur savent communiquer entre eux et se transmettre des données, donc JavaScript est capable de dire au navigateur de transmettre des données au serveur puis le serveur est capable de dire à PHP qu'il a reçu telles données du navigateur. De même, PHP est capable de dire au serveur d'envoyer des données au navigateur qui les transmettra à JavaScript, mais à **aucun moment** PHP et JavaScript n'ont communiqué entre eux.
Vous trouvez peut-être que je chipote, mais cela a une importance majeure : l'intérêt de faire interagir les deux serait d'échanger des données typées (un nombre, un tableau, un booléen, un objet etc.) mais le client et le serveur ne sont pas des langages de programmation et ne connaissent pas ce genre de données ! Le protocole HTTP impose que les seules informations qui peuvent transiter et être exploitées soient au format texte, éventuellement sous forme de texte structuré (XML par exemple) reconnu à la fois par JavaScript et PHP bref côté client et côté serveur.

6. Communications serveur -> client

Il est donc possible de faire parvenir au client ou au serveur des données de type textuel.
Pour passer de PHP à JavaScript, la solution est plutôt simple. Puisque c'est le navigateur qui créera le contexte JavaScript et qu'il lui suffit de rencontrer une balise *<script>* pour savoir qu'il faut interpréter son contenu comme du JavaScript (à condition bien sûr, que l'attribut *type* soit bien renseigné), il n'y a qu'à insérer ces balises dans la page générée. Voici un exemple pour créer un tableau JavaScript depuis un tableau PHP :

```
<?php
$tableau = array('toto', 'tata', 5);
echo '<script type="text/javascript">
var tableau = ["'.$tableau[0].'","'.
$tableau[1].'",'.$tableau[2].'];
</script>';
?>
```

Vous noterez bien que l'on ne transmet pas directement le tableau, on construit un tableau JavaScript depuis les éléments d'un tableau PHP.
De même, concernant la valeur numérique, elle n'est pas insérée comme étant un entier. Elle est ajoutée sous forme de texte formaté de façon à ce que JavaScript puisse interpréter qu'il s'agit d'un nombre.

Attention, dans le cadre d'Ajax, le code JavaScript ne sera pas interprété, nous verrons pourquoi plus tard.

Concernant la communication entre JavaScript et PHP, les modalités sont plus complexes et dépendent de différents cas, c'est ce que nous allons détailler maintenant.

7. Communications client -> serveur classique

J'entends par communication client / serveur classique la navigation sans utiliser AJAX.
Il existe trois façons de communiquer entre le navigateur et le serveur sans utiliser AJAX.

7.1. Les formulaires

Lorsque l'on soumet un formulaire, le navigateur récupère les valeurs des éléments du formulaire soumis possédant

un attribut *name* puis transmet ces couples nom / valeur à la page définie par l'attribut *action* via une requête HTTP soit en les ajoutant à l'URL soit dans le corps de la requête selon la valeur de l'attribut *method* ("GET" dans le premier cas, "POST" dans le second).
PHP reçoit ces paramètres dans les tableaux *$_GET* ou *$_POST*. Les valeurs sont alors nécessairement de type chaîne. Il ne peut pas en être autrement pour deux raisons :

- Comme nous l'avons déjà évoqué, le protocole ne connait pas les types.
- Il est nécessaire de passer par une couche supplémentaire, le HTML, pour transmettre les données, or HTML n'accepte pas les variables et ne reconnait donc pas les types non plus.

PHP peut alors construire une nouvelle page en prenant ces informations en compte puis renvoyer un nouveau document HTML au navigateur. Le navigateur recharge alors la page pour afficher le nouveau contenu.
Vous noterez toutefois que les champs de formulaires peuvent êtres remplis en JavaScript. Il est donc possible de transmettre des représentations textuelles des variables JavaScript à PHP (voire de les réintégrer dans la nouvelle page). Mais il s'agira tout de même d'un contexte JavaScript différent.

7.2. Les liens

Comme pour les formulaires, il est possible de transmettre des données depuis un lien. Cependant, nous sommes ici limités à la méthode GET, c'est-à-dire à les faire passer par l'URL. Ces paramètres sont à inclure dans l'URL après un point d'interrogation ("?") sous forme de couples nom / valeur séparés par le caractère "&" :

```
www.developpez.com?toto=tata&auteur=bovino
```

Là encore, nous sommes limités à des données textuelles, mais nous pouvons ajouter avec JavaScript des valeurs à transmettre :

```
<a href="www.developpez.com" id="test">Lien</a>
<script type="text/javascript">
	var variable = 'bovino';
	var parametres = '?toto=tata&auteur=" +
variable;
	document.getElementById('test').href +=
parametres;
</script>
```

Vous constaterez que la couche HTML est ici aussi nécessaire pour communiquer avec le serveur.

7.3. Les redirections JavaScript

L'objet *location* de JavaScript comprend quatre méthodes permettant de faire une redirection. Il est donc possible d'utiliser ces méthodes pour transmettre des paramètres.

méthode	description
location.href	Il ne s'agit en fait pas d'une méthode mais d'une propriété qui fait référence à l'URL de la page. Elle permet de récupérer ou d'affecter l'URL de la page.
location.assign()	Cette méthode charge l'URL passée en argument avec inscription dans l'historique.
location.replace()	Cette méthode charge l'URL passée en argument sans inscription dans l'historique.
location.reload()	Cette méthode recharge la page en cours, elle n'est pas utile pour transférer des paramètres JavaScript.

Méthodes de l'objet location.

Exemples

```
location.href = 'www.developpez.com?
toto=tata&auteur=bovino';
location.assign('www.developpez.com?
toto=tata&auteur=bovino');
location.replace('www.developpez.com?
toto=tata&auteur=bovino');
```

Il faut noter qu'ici, c'est JavaScript qui appelle directement la requête HTTP, il n'y a plus d'intermédiaire HTML, en revanche, c'est le moteur HTML qui reçoit la réponse du serveur.

Le point commun de toutes ces méthodes est que l'on envoie des informations au serveur qui renvoie un document complet. Il y a donc nécessairement rechargement de la page. AJAX va nous permettre de modifier uniquement des portions de page lorsque le rechargement complet n'est pas nécessaire. Nous allons voir ce que cela change par rapport aux modèles précédents.

8. Le cas AJAX

8.1. Premier principe

L'utilité essentielle d'AJAX est la capacité d'utiliser des informations présentes sur le serveur dans un script JavaScript sans avoir à recharger la page. Les exemples typiques d'utilisation sont nombreux, par exemple :

- Modifier une partie de la page sans avoir à la recharger entièrement comme c'était le cas dans les techniques présentées auparavant.
- Enregistrer à la volée des informations sur le serveur.
- Vérifier la validité et/ou la disponibilité (pseudos, mots de passe...) de champs de formulaires avant de soumettre ceux-ci (ATTENTION : cela n'exonère pas des vérifications coté serveur !)
- Proposer des suggestions lors du remplissage d'un champ de formulaire (autocomplétion).
- Etc.

La première conséquence est donc que lors d'une requête AJAX, le serveur **ne doit pas** renvoyer un document HTML.

En effet, dans les communications client / serveur sans utiliser AJAX, on parle de navigation, c'est-à-dire que le serveur doit renvoyer une page HTML complète au navigateur qui se charge ensuite de l'afficher. Le navigateur a besoin d'une page complète (et si possible

valide) pour l'afficher correctement.
A l'inverse, dans le cas d'AJAX, nous souhaitons juste mettre à jour une partie de la page, y insérer une page HTML complète constituerait donc une erreur grave de conception et risquerait de provoquer des erreurs d'affichage.
Il est cependant possible de créer coté serveur un fragment HTML (par exemple le contenu HTML que l'on veut insérer et qui peut lui-même comporter des balises). Mais ce fragment ne sera pas interprété comme du HTML mais comme du texte simple.

8.2. Que doit-on donc faire côté serveur ?

JavaScript ne pourra interpréter la réponse que comme du texte. Cependant, la méthode de création est identique à celle utilisée pour la création d'une page. En PHP, le résultat retourné par le serveur sera celui qui aura été affiché avec des instructions *echo* par exemple. Le script PHP devra donc organiser le résultat obtenu de façon a être facilement exploitable par JavaScript. Il existe différentes techniques pour cela :

- Renvoyer du texte simple, par exemple si vous avez juste un message à renvoyer.

```
echo 'Ce login est invalide';
```

- Un document XML.

Attention, pour que la réponse du serveur puisse être correctement interprétée, il est important de le préciser dans le *header* du script PHP et que le XML retourné soit valide.
Il est aussi à noter que le format XML est considéré comme plus sécurisé que les autres modes d'échange.

- Un objet JSON (JavaScript Object Notation (Lien39)).

S'il est un langage de balises permettant de structurer parfaitement les données, XML génère en contrepartie des échanges conséquents de données souvent inadaptées et qui nécessitent d'être ensuite analysées ("*parsées*") dès réception. Le format d'échange JSON est ainsi naturellement apparu ; il présente l'avantage d'organiser les données de façon comparable, mais beaucoup plus légère, en utilisant la syntaxe des objets JavaScript, et de les rendre directement utilisables :

```
{nom1: valeur1,
nom2: valeur2,
nom3: {
        sousnom1: sousvaleur1,
        sousnom2: sousvaleur2
}
}
```

On utilise des couples nom / valeur, chaque couple est séparé par une virgule (mais il ne faut pas de virgule après le dernier), les noms et les valeurs sont séparés par deux points (caractère ":"). Les valeurs peuvent elles-mêmes être des objets.

Le format JSON offre plusieurs avantages :
- Il est léger et facilement manipulable.
- Il est normalisé, cela signifie que n'importe qu'elle application supportant ce format peut accéder à son contenu sans avoir à ajouter quoi que ce soit.

En revanche, il présente aussi un inconvénient : s'agissant au final d'une donnée typée, il ne pourra pas être envoyé en tant que tel mais sous forme de texte, il faudra donc l'évaluer avec JavaScript avant de l'utiliser.

- Un système de pseudo XML en séparant les différents résultats avec des séparateurs prédéfinis.

```
$nom1 = $valeur1;
$nom2 = $valeur2;
$nom3 = $valeur3;
echo $nom1.'<>'.$valeur1.'<arg>'.$nom2.'<>'.
$valeur2.'<arg>'.$nom3.'<>'.$valeur3;
```

Les avantages de ce type de format sont : - Il est relativement léger et facile à mettre en œuvre. - Il est directement exploitable en JavaScript et chaque élément facilement récupérable au moyen d'appels à la méthode *split()* de l'objet *String*.

Les inconvénients :
- Il n'est pas normalisé, il est donc nécessaire de bien documenter sa structure.
- Il devient vite lourd à gérer en cas de structure trop complexe (imbrications multiples par exemple).

8.3. Etapes de la requête

Une requête AJAX se décompose en différentes étapes :

- Créer un objet XMLHttpRequest (Voir un exemple dans la FAQ (Lien38)).
- Définir les actions à réaliser lors des différentes réponses du serveur (méthode onreadystatechange()).
- Récupérer les éventuels paramètres à envoyer.
- Ouvrir une connexion avec le serveur (méthode open()).
- Envoyer la requête au serveur (méthode send()).

Vous constaterez qu'avec AJAX, c'est JavaScript qui envoie et qui reçoit les données. Il s'agit d'une différence essentielle avec la navigation classique.
En effet, nous avons vu que c'est le moteur HTML qui insère le JavaScript dans le contexte du document et cela lorsqu'il rencontre une balise <*script*>. Donc, comme le moteur HTML n'intervient pas dans une requête AJAX, ce qui signifie que :

Il n'est pas possible de récupérer du JavaScript avec AJAX.

Bon d'accord, j'exagère un peu... Mais il faut bien comprendre que le code JavaScript contenu dans la réponse du serveur est, comme tout le reste, considéré comme du texte, il est donc nécessaire de l'évaluer (avec la méthode *eval()*) avant de pouvoir l'utiliser, tout comme les réponses au format JSON.

```
eval(XMLHttpRequest.responseText);
```

Il est cependant à noter que l'utilisation de *eval()* est plutôt déconseillée en règle générale.
Cette méthode oblige à appeler le moteur JavaScript, ce qui réduit considérablement les performances.
Cette méthode est aussi considérée comme peu sécurisée car elle donne une importance exagérée au texte évalué.
Dans le cadre d'AJAX, on peut se dire (avec bon sens) qu'étant donné la *Same Origin Policy* (SOA : politique de même origine) inhérente à JavaScript et donc à AJAX, les risques sont minimes. C'est logiquement le cas si l'on considère que le développeur est compétent, ce qui n'est pas toujours le cas... mais qu'en est-il si la page appelée par votre requête fonctionne à la manière d'un proxy, qui est encore aujourd'hui la méthode la plus courante pour faire de l'AJAX inter domaine ? Avez-vous vraiment confiance dans les intermédiaires chez lesquels vous récupérez vos données ?
Il faut toutefois être conscient qu'il est impossible d'éviter l'utilisation de *eval()* pour interpréter du JavaScript récupéré par AJAX.
De ce fait, si vous avez à inclure du JavaScript dans un document via AJAX, demandez-vous au préalable si vous ne faites pas de fautes de conception et donc s'il n'y a pas un moyen plus académique de procéder.
D'après mon expérience, les cas où il est impossible de procéder différemment sont assez rares.

Une utilisation détournée de *eval()* serait de passer directement par le DOM, l'intérêt étant de rendre le script accessible dans le contexte global si vous l'évaluez par exemple dans le cadre d'une fonction anonyme de rappel type *callback* :

```
var lafonction = XMLHttpRequest.responseText;
var lescript = document.createElement('script');
lescript.type='text/javascript';
lescript.appendChild(document.createTextNode(lafonction));
document.getElementsByTagName('head')[0].appendChild(lescript);
```

Attention, cette méthode n'a pas été testée avec tous les navigateurs !

8.4. Asynchrone ou synchrone ?

Il est aussi important de comprendre à quoi correspondent les notions de requêtes synchrones ou asynchrones.
Le principe est relativement simple et bien compris, mais les implications sont plus difficiles à appréhender.

Lors d'une requête synchrone, la fonction qui la contient bloque son exécution tant que la réponse du serveur n'a pas été reçue. A titre de comparaison, c'est presque le même fonctionnement qu'avec une boîte d'alerte. Les implications sont que de ce fait, aucun code JavaScript ne peut être exécuté pendant ce temps, la page n'est plus interactive.

Lors d'une requête asynchrone au contraire, la requête est lancée mais le code de la fonction continue à s'exécuter. C'est habituellement un intérêt majeur d'AJAX, en revanche, il est important de notifier au visiteur que l'action qu'il a déclenchée est en cours d'exécution, car rien n'est plus agaçant que de cliquer sur un contrôle devant effectuer une action et de ne rien voir arriver...
Une autre conséquence, au niveau du développeur, est de bien comprendre que pendant le cours de la requête, la fonction qui l'a créée continue son exécution et bien souvent termine son exécution avant que la réponse du serveur ne revienne. Or, quand une fonction termine son exécution, son contexte est détruit.
De ce fait, il est important de savoir que si vous avez besoin des résultats de la requête dans la fonction qui la crée, vous risquez d'être obligé de passer par une requête synchrone.
En général, on préfèrera envisager tous les traitements de retour dans la fonction appelée par la réussite de la requête (ce que l'on appelle habituellement le *callback*) via l'événement *onreadystatechange* en prenant bien soin de rendre disponibles les éléments du contexte requis.

L'aspect asynchrone ne représente que la valeur par défaut d'une requête AJAX, elle est cependant celle que je préconiserais.
Tout comme pour les évaluations de scripts, les cas où l'utilisation de requêtes synchrones est indispensable sont plutôt rares et je préfère tenter de privilégier les actions des utilisateurs au maximum.

Retrouvez l'article de Didier Mouronval en ligne : Lien40

Savoir comment utiliser les sélecteurs CSS 2.1 et les nouveautés CSS 3

Les sélecteurs CSS sont certainement un des aspects les plus puissants en CSS. Ils permettent de cibler un élément spécifique du balisage HTML pour lui affecter un style. Les sélecteurs CSS sont variés, allant du simple nom d'un élément jusqu'à une expression de sélection riche d'une liste d'éléments HTML. Cet article vous fait un rappel sur les possibilités des sélecteurs CSS et vous présente les nouveautés CSS 3 qui s'avèrent très puissantes pour éviter des id, classes ou JavaScript non nécessaires.

1. Sélecteur CSS 2.1

1.1. Syntaxe des sélecteurs

Les sélecteurs peuvent être de type simple (cas du sélecteur universel), de type sélecteurs d'attributs ou encore de type sélecteurs d'id.

Les sélecteurs de type simple peuvent être combinés à l'aide de caractères spéciaux comme un espace, un "+" ou un ">".

Enfin on peut combiner les sélecteurs dans une liste quand on veut appliquer un même style aux éléments ciblés par plusieurs sélecteurs.

```
h1 { color: red; }
h2 { color: red; }
h1, h2 { color: red; } /* équivalent aux 2 lignes ci-dessus */
```

1.2. Sélecteur universel

Le sélecteur universel correspond au caractère "*", il représente l'ensemble des éléments de l'arbre HTML et peut être omis.

```
*.warning
.warning
/* les 2 syntaxes sont équivalentes */
```

1.3. Sélecteur de type

Un sélecteur de type correspond au nom de l'élément HTML. Il permet d'appliquer un style à tous les éléments de ce type dans le document HTML. La règle suivante concerne tous les éléments *h1*.

```
h1 { font-family: sans-serif; }
```

1.4. Sélecteur descendant

Un sélecteur descendant permet de cibler les éléments enfants d'un autre élément.

L'exemple suivant appliquera une couleur jaune au texte de tous les éléments *strong* contenus dans un paragraphe.

```
p strong { color: yellow; }
```

1.5. Sélecteur d'enfant

Un sélecteur d'enfant est utilisé pour vérifier qu'un élément est un enfant d'un autre élément. Pour utiliser le sélecteur d'enfant on utilise le caractère ">" avec à gauche le parent et à droite l'enfant.

L'exemple suivant appliquera un fond bleu aux éléments *strong* enfants d'un paragraphe.

```
p > strong { background-color: blue; }
```

La différence entre le sélecteur d'enfant et le sélecteur descendant est que le sélecteur descendant correspond aux éléments *strong* où qu'ils soient dans le paragraphe, alors que le sélecteur d'enfant est plus spécifique et correspond uniquement aux éléments *strong* qui sont enfants du paragraphe.

Dans l'exemple suivant :

```
<p>Martin dit :
    <q>super site sur le référencement naturel à <strong>Marseille</strong></q>,
    en plus il connaissait bien les sujets <strong>SEO</strong>!
</p>
```

Le code sélecteur descendant précédent mettra en jaune les deux textes entre les balises *strong* alors que le code sélecteur d'enfant précédent appliquera un fond bleu uniquement au 2e *strong* soit le mot SEO car seulement cet élément est un enfant du paragraphe. L'élément *strong* Marseille est un enfant de la citation *q*.

Les sélecteurs d'enfants ne sont pas supportés par IE6.

1.6. Sélecteur d'enfant adjacent

Les enfants adjacents sont les éléments qui suivent immédiatement un autre élément qui partage le même parent. On utilise le signe "+" pour les enfants adjacents.

L'exemple suivant appliquera un fond vert à tous les éléments *h2* qui suivent immédiatement un élément *h1* si ses éléments partagent le même parent.

```
h1 + h2 { background-color: green; }
```

Les sélecteurs d'enfants adjacents ne sont pas supportés par IE6.

1.7. Sélecteur d'attribut

Les sélecteurs d'attribut correspondent aux éléments qui ont des attributs définis dans l'arbre HTML. Il y a 4 types de sélécteurs d'attribut.

1.7.1. [att]

Le sélecteur d'attribut basique correspond aux éléments qui ont l'attribut spécifié indépendamment de la valeur de l'attribut.

L'exemple suivant appliquera un fond rouge à tous les liens qui possèdent un attribut *title*.

```
a[title] { background-color: red; }
```

1.7.2. [att=val]

Ce sélecteur d'attribut est utilisé pour sélectionner les éléments qui contiennent un attribut spécifique avec une valeur particulière.

L'exemple suivant appliquera un fond rouge à tous les liens qui possèdent un attribut *rel* avec la valeur *external*.

```
a[rel=external] { background-color: red; }
```

1.7.3. [att~=val]

Certains attributs autorisent une liste de valeurs séparées par des espaces. En utilisant le "~" avant le caractère "=", vous pouvez positionner un style sur les éléments qui possèdent un attribut spécifique avec une valeur particulière présente dans la liste des valeurs.

L'exemple suivant appliquera un fond rouge à tous les paragraphes qui contiennent la classe *marseille*.

```
p[class~=marseille] { background-color: red; }
/* p[class~=marseille] est équivalent à
p.marseille */
```

1.7.4. [att|=val]

Ce sélecteur d'attribut est utilisé pour sélectionner les éléments qui possèdent un attribut spécifique et dont la valeur est une liste de mots séparés par des tirets et commençant par val.

L'exemple suivant appliquera le style sur les liens avec un *hreflang* dont la valeur commence par "en", incluant "en-US", "en-GB", etc.

```
a[hreflang|=en] { background-color: red; }
```

Plusieurs sélecteurs d'attributs peuvent être utilisés dans la même instruction.

L'exemple suivant utilise 2 sélecteurs d'attribut pour appliquer un fond rouge à tous les liens qui ont un attribut *title* et dont l'attribut *rel* à la valeur *external*.

```
a[title][rel~=external] { background-color:
red; }
```

1.8. Sélecteur de classe

Comme vu précédemment, on utilise le caractère "." comme sélecteur de classe.

Le sélecteur suivant représente un élément *h1* portant la class "referenceur".

```
h1.referenceur
```

1.9. Sélecteur d'ID

Le sélecteur d'ID correspond au caractère "#". Dans l'exemple suivant, le sélecteur d'ID représente un élément *h1* dont l'attribut *id* a la valeur "referencement".

```
h1#referencement
```

1.10. Pseudo-éléments et pseudo-classes

Les sélecteurs de pseudo-classes et de pseudo-éléments sont utilisés dans les scénarios où sélectionner la position de l'élément dans l'arbre HTML n'est pas suffisant. Pour utiliser les pseudo-classes et les pseudo-éléments on utilise le caractère ":".

1.10.1. La pseudo-classe :first-child

La pseudo-classe *:first-child* correspond au premier élément enfant d'un autre élément.

```
div > p:first-child { text-indent: 0; }
```

1.10.2. Les pseudo-classes d'ancre :link et :visited

La pseudo-classe *:link* s'applique aux liens qui n'ont pas été visités et la pseudo-classe *:visited* s'applique lorsque le lien a été visité par l'utilisateur.

1.10.3. Les pseudo-classes dynamiques :hover, :active et :focus

Les pseudo-classes dynamiques appliquent un changement de style en fonction d'une action utilisateur. La pseudo-classe *:hover* applique un style quand l'utilisateur désigne un élément (par exemple au survol de la souris), la pseudo-classe *:active* applique un style quand l'utilisateur active un élément (par exemple au moment où l'utilisateur presse le bouton de la souris et le relâche) et la pseudo-classe *:focus* applique un style quand un élément reçoit l'attention (par exemple quand l'utilisateur se positionne sur l'élément à l'aide du clavier).

```
a:link { color: red }        /* lien non-visité */
a:visited { color: blue }    /* lien visité */
a:hover { color: yellow }    /* lien survolé */
a:active { color: lime }     /* lien activé */
```

1.10.4. La pseudo-classe de langue :lang

La pseudo-classe *:lang* permet de cibler un élément dont la langue correspond à une certaine valeur.

```
p:lang(en) { background-color: red; }
```

1.10.5. Le pseudo-élément :first-line

Le pseudo-élément *:first-line* associé à un paragraphe

produit un style particulier sur la première ligne formatée d'un paragraphe.

```
p:first-line { text-transform: uppercase; }
```

1.10.6. Le pseudo-élément :first-letter

Le pseudo-élément *:first-letter* peut être employé pour faire des capitales initiales et des lettrines.

1.10.7. Les pseudo-éléments :before et :after

Les pseudo-élément *:before* et *:after* servent à insérer un contenu généré avant ou après celui d'un élément.

2. Nouveautés CSS 3 sur les sélecteurs

2.1. Sélecteur d'attribut

3 nouveaux sélecteurs d'attribut sont introduits par la norme CSS 3 :

- [att^="val"] : Représente un attribut dont la valeur commence exactement par le préfixe "val".
- [att$=ident] : Représente un attribut dont la valeur finit exactement par le suffixe "ident".
- [att*="val"] : Représente un attribut dont la valeur contient au moins une fois la sous-chaîne "val".

Le seul navigateur qui ne supporte pas les sélecteurs d'attribut CSS3 est IE6. IE7, IE8, Opera et les navigateurs basés les moteurs Webkit (Safari et Chrome) et Gecko (Firefox) supportent ces sélecteurs.

2.2. Combinateur d'adjacence indirecte

Les combinateurs d'adjacence indirecte sont composés du caractère "~" séparant deux séquences de sélecteurs simples.

L'exemple suivant permet d'ajouter une bordure grise à toutes les images qui suivent une *div* particulière (l'image et la *div* doivent avoir le même parent).

```
div~img { border: 1px solid #ccc; }
```

Tous les navigateurs supportent le combinateur d'adjacence indirecte sauf votre navigateur favori : Internet Explorer 6.

2.3. Pseudo-classes

Contient les plus grosses nouveautés de la norme CSS 3.

2.3.1. La pseudo-classe :nth-child()

Ce sélecteur vous permet de cibler les éléments en se basant sur leur position dans la liste des enfants de leur parent. Vous pouvez utiliser un numéro, une expression numérique ou les mots "odd" et "even" correspondant à impair et pair (parfait pour faire un style alternatif pour vos tableaux).

```
tr:nth-child(2n+1) /* correspond aux lignes
paires */
tr:nth-child(odd) /* pareil */
```

2.3.2. La pseudo-classe :nth-last-child()

Ce sélecteur ressemble beaucoup au sélecteur précédent sauf qu'il correspond au dernier enfant d'un élément parent.

```
tr:nth-last-child(-n+2) /* correspond aux 2
dernières lignes du tableau */
```

2.3.3. La pseudo-classe :last-child

Identique à *:nth-last-child(1)*. La pseudo-classe *:last-child* représente un élément qui est le dernier fils d'un autre élément.

2.3.4. La pseudo-classe :checked

Correspond aux éléments qui sont cochés.

2.3.5. La pseudo-classe :empty

Correspond aux éléments qui n'ont pas d'enfants ou qui sont vides.

2.3.6. La pseudo-classe de negation

La pseudo-classe de négation est une notation fonctionnelle prenant un sélecteur simple pour argument. Elle représente un élément qui n'est pas représenté par l'argument.

L'exemple ci-dessous permet d'afficher le texte des paragraphes qui n'ont pas la classe "lead" en noir.

```
p:not([class*="lead"]) { color: black; }
```

Les navigateurs basés sur le moteur Webkit ou Opera supportent toutes les nouveautés CSS 3 des sélecteurs de pseudo-classe. Firefox 2 et 3 supporte uniquement *:not(s)*, *:last-child*, *:only-child*, *:root*, *:empty*, *:target*, *:checked*, *:enabled* et *:disabled*, mais Firefox 3.5 aura un support complet des sélecteurs CSS 3. Internet Explorer n'a actuellement aucun support de ces pseudo-classes.

2.4. Pseudo-élément

2.4.1. Les pseudo-éléments fragments d'éléments d'interface

Le nouveau pseudo-élément *::selection* s'applique à la portion du document qui a été mise en exergue par l'utilisateur.

L'exemple suivant permet de modifier la couleur de fond du texte en cours de sélection.

```
::selection { background-color: blue; }
```

3. Pour en savoir plus

- Module CSS3 : Sélecteurs W3C – Traduction officielle du brouillon de la norme (Lien41)
- Selectors Level3 – W3C Working Draft (Lien42)
- Selectutorial – CSS selectors (Lien43)
- CSS3: Attribute selectors – CSS3 info (Lien44)
- CSS selectors and pseudo selectors browser compatibility (Lien45), sur cette page vous retrouverez en détails le support des sélecteurs par navigateur.

Retrouvez l'article de Julien en ligne : Lien46

Compte rendu de la présentation de la nouvelle plateforme Flash Adobe

Ce compte rendu a pour but de présenter la nouvelle plateforme Flash d'Adobe présentée à Paris le 3 Juin 2009.

1. Introduction

A l'occasion de la sortie en version béta de leur nouvelle solution web, Adobe a organisé une conférence de presse suivi d'une présentation plus technique de Flash Builder (pour ceux qui ont participé à la session Développeur).

Cette plateforme regroupe donc un ensemble de logiciel Adobe qui permet de construire un projet web. Nous allons donc voir dans cet article comment fonctionne cette plateforme puis nous présenterons les deux stars de cette conférence Flash Catalyst et Flash Builder. Enfin nous ferons le bilan de ce que peut donner toutes ces nouveautés dans l'avenir.

2. La plateforme Flash

La plateforme Flash est une solution proposée par Adobe pour faciliter la collaboration entre designer et développeur.

2.1. Les solutions pour les Designers

Tous les designers connaissent sûrement la gamme Creative Suite 4 (CS 4) composée de Photoshop, Illustrator et Fireworks. Habituellement, le designer dessine une maquette et la transmet au développeur. Ce dernier tentera tant bien que mal de respecter au mieux la maquette tout en développant les différentes fonctionnalités de l'application.

Adobe revoit ce workflow (Lien47) en ajoutant Flash Catalyst entre le designer et le développeur.

2.2. Flash Catalyst : le médiateur entre designer et développeur ?

Flash Catalyst est un outil capable de récupérer une image sous Photoshop incluant tous ses calques et de les convertir en Flex sans écrire une seule ligne de code. Pour nous montrer toutes les capacités de Flash Catalyst, les consultants d'Adobe ont mis en place une simple application de consultation représentant un annuaire téléphonique.

En partant, d'une image représentant l'interface, ils ont réussi en une vingtaine de minutes à en faire une application web dynamique interfacée avec un service PHP.

La démonstration est tout à fait bluffante et est disponible ici (Lien48).

Afin de faciliter cette transformation, l'image doit être conçue avec des calques bien définis. Ainsi, il sera plus facile de convertir les composants. Avec Flash Catalyst, il est possible de réaliser une maquette entièrement dynamique. Elle gère les évènements, les états et les effets.

2.3. Comment Flash Catalyst peut-il générer du Flex ?

Il faut d'abord savoir que Flash Catalyst génère du Flex 4 et que bien des choses ont été modifées entre Flex 3 et 4. Et la principale modification est l'ajout d'une nouvelle librairie de composants graphiques nommée **spark** qui vient compléter celle déjà existante **halo**. Elle ajoute un aspect manquant à halo :

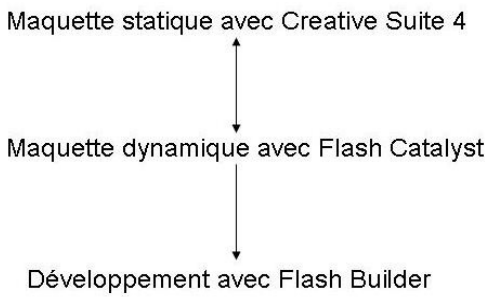
- la séparation entre le skinning et le comportement du composant,
- une gestion différente des effets,
- une librairie graphique (FXG) permettant de réaliser les dessins vectoriels et qui facilitera le passage entre la maquette sous forme d'image et l'application Flex.

Une fois l'application réalisée, le designer l'exporte au format fxp. Format reconnu par Flash Builder qui va l'ouvrir comme un projet Flex classique.

2.4. Un nouveau workflow en place

Avec cette solution le workflow (Lien47) est complètement revu. Ce qui donne plus d'importance au rôle du designer qui peut aller plus loin dans la conception de sa maquette.

Actuellement le fonctionnement se passe de la façon suivante :



Comme vous pouvez le voir la flèche de développement vers maquette dynamique n'est que dans un sens. Car une fois le code modifié dans Flash Builder il n'est pas possible de revenir dans Flash Catalyst. D'après les consultants Adobe présents, il faudra attendre la sortie des prochaines versions.

Durant la session Développeur, les consultants Adobe ont

présenté la version Béta 1 de FlashBuilder.

3. Les nouveautés de Flash Builder

Les améliorations de Flash Builder suivent 3 grands axes :

- amélioration de la productivité : avec génération automatique de getter et setter, génération automatique de gestionnaire d'évènements,
- intégration des services distants qui proposent entre autres BlazeDS, LCDS et Zend AMF,
- applications plus expressives grâce à l'exploitation de Flash 10.

3.1. Productivité améliorée

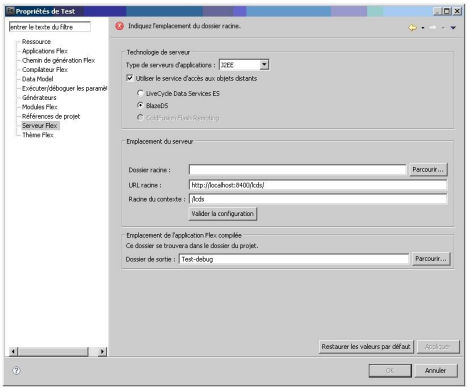
Flash Builder améliore la productivité du développeur avec l'ajout de fonctionnalités bien pratiques pour le développement. Parmi ces fonctionnalités, il y a :

- la génération automatique de getter et setter,
- la génération automatique d'évènements : lors de l'ajout d'un bouton par exemple avec l'évènement click, Flash Builder va proposer la génération de la fonction correspondante. Un gain de temps non négligeable sachant que l'on peut être amené à le faire plusieurs fois par jour,
- Le refactoring (Lien49) a été amélioré. Il peut maintenant être effectué sur les composants MXML.

Flash Builder le prend en charge et applique les modifications là où il le faut. L'IDE intègre aussi FlexUnit qui permet de réaliser des suites de tests et de l'intégration continue.

3.2. Intégration des services distants

Flash Builder prend en charge un certain nombre de serveurs distants. L'IDE fournit une fenêtre de configuration de serveurs :



Il est possible à tout moment de configurer son application afin qu'elle prenne en charge :

- les applications J2EE avec BlazeDS et LiveCycle Data Services ES,
- les applications PHP. Adobe a d'ailleurs réalisé un partenariat avec Zend afin de permettre l'utilisation de ZendAMF et AMFPHP.

Enfin, Flash Builder est maintenant capable de réaliser de l'introspection sur les services distants. Il peut donc récupérer le profil des méthodes, reconnaître le contenu d'un objet retourné par une méthode.

4. Conclusion

Avec cette nouvelle plateforme, le designer peut aller plus loin dans la conception de sa maquette et le développeur pourra se concentrer sur le développement de fonctionnalités. Pour le moment, aucun prix a été annoncé mais nous pouvons penser qu'elle sera vendue avec la gamme Creative Suite 4 (sinon cela n'aura pas d'intérêt). La release de Flash Catalyst devrait être prévue pour début 2010 mais aucune date précise.

Suite à la présentation, quelqu'un a posé la question suivante : " Est-ce la fin des développeurs ? ". La fin des développeurs, peut-être pas, mais une meilleure attribution des rôles probablement. De plus, les DSI vont-elles accepter demain d'investir dans le design ?

On ne peut que constater également les efforts d'Adobe pour répondre aux besoins de la communauté Flex. Car toutes ces nouvelles fonctionnalités sont issues de demandes de la communauté. D'ailleurs les consultants ont beaucoup insisté sur ce point en demandant aux développeurs Flex de ne pas hésiter à faire des retours sur les prochaines versions de Flash Builder.

5. Liens

Pour télécharger ces différents outils :

- Lien50
- Lien51

Documentation sur Flex 4 :

- Lien52

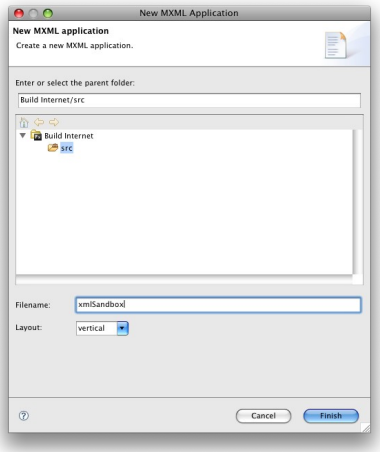
Retrouvez l'article d'Ellène Dijoux en ligne : Lien53

Les fondamentaux XML avec Flex 3

Si vous utilisez toujours des tableaux et des fichiers textes pour charger du contenu, il est temps d'évoluer. Avec XML, vous pouvez concentrer un fouillis complet d'informations apparemment sans lien dans un fichier bien organisé et facile à lire. Nous allons voir aujourd'hui comment en tirer des avantages avec Flex 3. Ce tutoriel vous apprendra comment charger un fichier XML externe dans Flex et ensuite comment faire passer les résultats dans des étiquettes.

1. Mettre en place l'espace de travail

Ouvrez Flex 3 Builder et commencez une nouvelle application MXML. J'ai appelé la mienne fondamentauxXML pour cet exemple. Choisissez un layout vertical et cliquez sur *finish*.



Avant de charger un fichier XML, nous devons en avoir un. J'en ai fait un à télécharger pour ce tutoriel (Lien54). Créez un nouveau répertoire dans votre répertoire 'src' par clic droit sur l'icône et la sélection de *New* > *Folder*. Appelez ce nouveau répertoire 'assets' et importez le fichier XML téléchargé dedans. Une fois que c'est fait, vous êtes prêt à le charger dans le document Flex.

2. Charger le XML

MXML rend simple le chargement de fichiers externes avec la balise HTTPService. Ecrivez le code suivant immédiatement sous la balise application ouvrante :

```
<mx:HTTPService url="assets/contenu.xml"
id="donneesSite" resultFormat="e4x"/>
```

Vous venez de créer un service HTTP avec l'*id* 'donneesSite' qui pointe sur le fichier XML dans le répertoire 'assets'. Les résultats sont formatés en *e4x*, qui nous permet de naviguer vers les contenus du fichier XML. Pour que cette url soit chargée, nous devons demander à l'application d'envoyer la requête. Nous le faisons en ajoutant l'événement 'creationComplete' à la balise application ouvrante.

```
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical"
creationComplete="donneesSite.send()">
```

L'événement creationComplete est appelé après le chargement de la page. Dans le code ci-dessus nous indiquons que quand le chargement de la page est terminé, le service HTTP appelé 'donneesSite' doit être envoyé. Prenez un instant pour tester votre application. Elle doit s'exécuter sans erreur si le XML est chargé correctement.

3. Récupérer les données XML

Les données XML auront besoin d'être stockées en attendant d'être utilisées. Ajoutez une balise script juste sous la balise application ouvrante et déclarez la variable suivante :

```
<mx:Script>
    <![CDATA[
    private var contenuSite:XMLList;
    ]]>
</mx:Script>
```

Le type XMLList nous permet de charger des données XML et de travailler avec. Pour le démontrer nous allons créer une fonction qui chargera les données du fichier XML et les affichera dans des étiquettes. Ajoutez la fonction suivante à votre balise script :

```
private function
afficherPage(pageCible:String):void
{
  // Recherche du titre de la page cible
  contenuSite = donneesSite.lastResult.page.
(@id==pageCible).titre;
  // Affiche les résultats dans l'étiquette
  enTete.text = contenuSite;

  // Recherche du corps de la page cible
  contenuSite = donneesSite.lastResult.page.
(@id==pageCible).texte;
  // Affiche les résultats dans l'étiquette
  corps.text = contenuSite;
}
```

Maintenant, sous la balise HTTPService, nous allons ajouter quelques composants dans lesquels les résultats seront chargés. Nous allons aussi ajouter des boutons pour

contrôler la navigation.

```
<mx:VBox>
  <mx:Label fontSize="24" id="enTete"
text="Cliquez sur un bouton"/>
  <mx:Label id="corps" text="A vous de
choisir !"/>

  <mx:HBox>
    <mx:Button label="Page d'accueil"
click="afficherPage('accueil')"/>
    <mx:Button label="Page à propos"
click="afficherPage('apropos')"/>
    <mx:Button label="Page de contact"
click="afficherPage('contact')"/>
  </mx:HBox>
</mx:VBox>
```

Que venons-nous de faire au juste ? Voici un aperçu de ce qui arrive :

1. Quand un des boutons est pressé il passe une chaîne de caractères à la fonction afficherPage.
2. afficherPage reçoit ce paramètre dans une chaîne de caractères nommée pageCible.
3. Cette pageCible déterminera le noeud XML à afficher.
4. Chaque noeud du fichier XML a un attribut appelé "id" qui correspond à la valeur du paramètre pageCible.
5. La XMLList contenuSite fait correspondre un noeud à pageCible et charge alors les bons textes de titre et de corps dans leurs étiquettes respectives.

4. Conclusion

Maintenant vous y êtes ! Exécutez votre application. Si tout se passe comme ça devrait, les données du fichier XML sont chargées pour chaque page quand un bouton est pressé. Pas mal ! Vous n'êtes pas limité à du texte. Essayez de charger l'url d'une image d'un fichier XML avec Flex. Le XML a un grand nombre de possibilités à explorer.

Regardez la démo en ligne (Lien55).

Téléchargez les sources (Lien56).

Retrouvez l'article de Zach Dunn traduit par Sylvain Jorge Do Marco en ligne : Lien57

Le référencement / SEO pour les débutants

Le bon référencement d'un site est aujourd'hui une priorité pour le webmaster, et on a ainsi vu apparaître de nouveaux métiers : référenceur, rédacteur web...

L'optimisation du référencement d'un site (ou Search Engine Optimisation) est en effet complexe et prend en compte de très nombreux paramètres; je vous propose de découvrir quelques uns de ces aspects, afin de vous aider à améliorer le référencement de vos sites web.

1. Search Engine Optimisation - Enjeux et définitions

1.1. La SEO En un mot

La SEO, ou " Optimisation des Moteurs de recherche ", consiste à améliorer le positionnement d'un mot sur les moteurs de recherche, c'est-à-dire à faire en sorte que, lorsqu'un utilisateur effectue une recherche sur un moteur de recherche sur une expression précise, un site précis apparaisse le plus haut possible sur la première page dans la liste des résultats.

1.2. Pourquoi optimiser ?

Parce qu'être bien positionné dans les résultats des moteurs de recherche apporte une plus-value très importante à un site web : un nombre de visiteurs (trafic) très important. Il est en effet évident que les résultats qui apparaissent en première page génèrent plus de trafic pour les sites internet associés que ceux situés sur les pages suivantes. De même, les résultats positionnés en haut de la première page génèrent également plus de trafic.

En effet, " 54% des internautes ne visualisent que la première page de résultat " (REUSSIR SON REFERENCEMENT WEB - O. Andrieu (Eyrolles, 2007 - chap.1))

Cette situation se voit très bien dans le Triangle d'Or de Google :

Le Triangle d'or - Oseox.fr - Plus une zone est rouge, plus l'oeil s'y attarde longtemps

Sur cette image, les points représentent les zones observées par les utilisateurs. On voit très facilement l'avantage qu'il y a à être bien positionné : les quelques premiers résultats couvrent à eux seuls la majorité des clics des utilisateurs. Notons que le concept de Triangle d'or évolue depuis peu très rapidement. Nous nous attarderons sur cette évolution dans la section 9.3 (La recherche localisée / universelle : impacts sur le référencement).

1.3. Un enjeu double

1.3.1. Un enjeu technique

Un des enjeux de la SEO est de rendre accessible l'ensemble des pages (publiques) d'un site aux moteurs de recherches (absence de liens morts, fichier d'indexation des pages…), et l'ensemble du contenu de ces pages. (L'utilisation de certaines technologies empêche les moteurs de recherche d'accéder au même contenu que le visiteur (JavaScript, Ajax...))

1.3.2. Un enjeu sémantique

L'autre enjeu de la SEO est de permettre aux moteurs d'identifier facilement et efficacement les types de contenus sur une page (titres, images, contenus…). Pour cela, les pages doivent contenir certaines balises qui vont renseigner au mieux le type de contenu qu'elles contiennent. Par exemple, un contenu placé entre des balises <h1> est un titre, placé entre des balises <dd> il s'agit une définition…

1.4. SEO, SEA, Webmarketing

Si le terme de " SEO " est généralement connu, l'erreur est fréquente d'y superposer deux autres termes, pourtant bien différents : la SEA et le Webmarketing. Contrairement à la SEO (Search Engine Optimization), la SEA, ou Search Engine Advertising a pour objectif de travailler uniquement sur l'aspect publicitaire d'un moteur de recherche : les résultats " non-naturels ", c'est-à-dire les publicités. L'illustration suivante permet de bien les distinguer :

Search Engine Optimization et Search Engine Advertising - Oseox.fr

La SEA concerne toutes les zones d'affichages des publicités, tandis que la SEO concerne la zone de résultats en elle-même. Le Webmarketing est, lui, parallèle à la SEO : il s'agira alors d'organiser des " campagnes " pour générer des trafics, campagnes publicitaires auprès d'annonceurs, diffusions de vidéos virales ou encore campagnes d'emailing…

2. Le contexte de la SEO en 2009

2.1. Acteurs et Historique

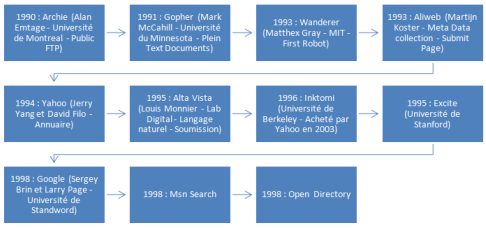
2.1.1. Acteurs

Malgré de très nombreux moteurs de recherche, le nombre réel de moteurs est très faible. En effet, seuls cinq algorithmes constituent la quasi-totalité des moteurs de recherches importants :

Moteur de recherche	Est utilisé par
Google	Google, Free, AOL...
Yahoo!	Yahoo!, Lycos, Altavista
Exalead	Exalead
Voila	Voila, Orange
MSN	Live Search (et désormais Bing)

2.1.2. Apparition des premiers moteurs de recherche

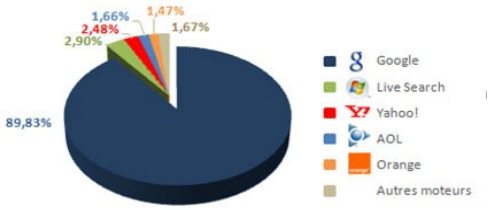
Le premier moteur de recherche apparaît en 1990. Très rapidement, le marché prend de l'essor. Dès 1998, les principaux acteurs de marchés sont présents.



Apparition des premiers moteurs

2.1.3. Parts de Marché des différents moteurs

Google est aujourd'hui, de très loin, le premier moteur de recherche en France, avec près de 90% des parts de marché, et une croissance ininterrompue depuis plusieurs mois.



Top 5 des moteurs de recherche en parts de visites en France en Avril 2009

Source : Baromètre AT Internet Institute (Lien58) (auparavant XitiMonitor)

C'est pour cette raison que Google est la principale priorité dans le monde de la SEO en France.

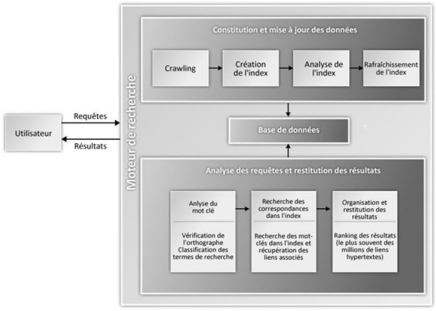
3. Moteurs de recherche : compréhension et approche

3.1. Principe général : fonctionnement des moteurs recherche

Pour bien comprendre en quoi et comment il est possible d'optimiser un site internet pour améliorer son référencement, il convient tout d'abord de comprendre comment fonctionne, globalement, un moteur de recherche. L'affichage d'une page de résultats (sur Google, Yahoo !) n'est que la partie émergée d'un mécanisme complexe, commun à l'ensemble des principaux moteurs de recherche.

Ce mécanisme repose sur deux étapes (REUSSIR SON REFERENCEMENT WEB, chap. 2 - Olivier Andrieu (Eyrolles - 2007)) :

1. La constitution et mise à jour des données
2. L'analyse des requêtes et la restitution des résultats



Les différentes étapes du fonctionnement des moteurs de recherche

Voici une explication rapide de ce processus :

3.1.1. Constitution et Mise à jour des données

3.1.1.1. Le crawling

le moteur explore le web et stocke les informations.

3.1.1.2. Le moteur d'indexation :

le moteur traite les données recueillies et les ajoute à un index géant.

3.1.2. Analyse des requêtes et restitution des résultats

3.1.2.1. Analyse du mot-clef :

un interpréteur analyse l'expression tapée par l'utilisateur.

3.1.2.2. Recherche dans l'index :

le moteur recherche les résultats associés dans l'index.

3.1.2.3. Organisation des résultats :

le moteur détermine l'ordre d'apparition des résultats selon un ensemble de règles de tri.

Ces processus sont communs aux principaux moteurs de recherche du web (notons qu'on ne parle pas des annuaires ici). Par contre, les différents algorithmes utilisés (indexation, règles de tri...) sont généralement tenus secrets et dépendent du moteur utilisé.

3.2." Dark Hat " versus " White Hat "

Le référenceur dispose d'un ensemble de techniques très variées pour son optimisation. Mais si certaines sont acceptées et encouragées par les moteurs de recherche (techniques " White Hat "), d'autres sont largement déconseillées voires interdites (techniques " Dark Hat "), car elles entraînent des pages au contenu " non naturel " pour l'utilisateur. L'utilisation de ces techniques frauduleuses, si elle est détectée, peut entraîner des pertes importantes pour le site concerné, allant d'un affaiblissement du positionnement au bannissement total du site dans l'index du moteur de recherche. Nous ne parlerons pour l'instant que des techniques approuvées par les moteurs de recherches : la White SEO. Google publie régulièrement sur son Blog la liste des techniques qu'il considère comme frauduleuses, et Il existe plusieurs documents de référence, comme LA CHARTE DE QUALITE ET DE DEONTOLOGIE SUR LE REFERENCEMENT DE SITES WEB (Lien59), du réseau Abondance, acteur majeur du référencement français.

3.3. le Page Rank

Pendant longtemps, on a considéré (et c'était vrai à l'époque) que le critère principal du positionnement d'une page était son Page Rank, c'est-à-dire une note sur dix, attribuée par Google, prenant compte de nombreux critères mais dont le principal était le nombre de liens pointant vers ladite page (" Back Links"). Aujourd'hui, le Page Rank tend à avoir de moins en moins d'effet. Google reconnaît même que les données qu'il fournit aux webmestres sur le Page Rank de leur page ne sont qu'approximatives. En effet, c'est la qualité du contenu de la page qui prime : pertinence et originalité. Et même si le nombre de Back Link continue d'influer sur le positionnement de la page sur un mot donné, là encore la qualité est de mise : plus le site qui pointe vers la page est " digne de confiance " (Indice généralement nommé " Trust Rank " par Yahoo!, ou " Indice de Confiance ". Un site qui propose des contenus de très bonne qualité se verra attribuer un très bon indice de confiance, à l'inverse d'un site qui, par exemple, contiendra des contenus plagiés. Google attribue de base une confiance maximale au site dont l'extension est .gov ou .edu, .gouv… (sites officiels ou de l'enseignement)), plus la page est valorisée. Même si Google se défend de donner trop d'importance au Trust Rank (Mythes et réalités du moteur de recherche Google (Lien60), par Google : Le Trust Rank est à la base un concept de Yahoo!), il semble bien qu'aujourd'hui le Page Rank tend à laisser la place à la qualité, au détriment de la quantité.

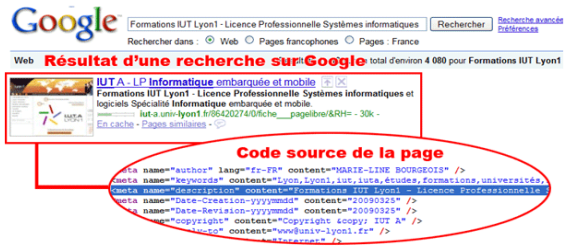
4. Les techniques de base de la SEO : optimisation interne

4.1. Introduction

Une page web est composée de contenu (images, textes), et de balises qui indiquent au navigateur (comme Internet Explorer ou Firefox) la manière dont doit être mis en forme ce contenu. Il est possible d'utiliser ces balises pour améliorer le positionnement d'un site ; c'est même la première chose à faire pour espérer placer une page de manière correcte sur un moteur de recherche.

4.2. Les balises <meta>

Chaque page web est identifiée par des balises <meta>. Ces balises attribuent un titre, une description, une liste de mots clefs… à la page, autant d'éléments qui seront utilisés par les moteurs de recherche pour analyser votre page. Par exemple, dans l'illustration suivante, on remarque bien que la balise <meta> " description " est utilisée par Google pour décrire la page :



La balise <meta> de description est utilisée par Google pour identifier la page

Il convient donc de renseigner au mieux ces balises pour faciliter au maximum la tâche aux moteurs de recherche lors de l'indexation du contenu.

4.2.1. Remarque

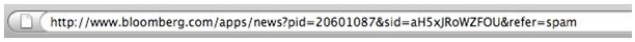
On considère généralement aujourd'hui que la balise <meta> de mots clefs (keywords) n'a quasiment plus aucun impact. En effet, les différents abus réalisés par de nombreux sites aux alentours des années 2000 ont poussé les moteurs de recherche à diminuer le poids de cette balise dans leurs algorithmes.

4.3. L'URL

L'URL est l'adresse de la page. Elle correspond à ce que l'on tape dans la barre d'adresse pour accéder à la page (par exemple, l'url de Google France est " http://www.google.fr "). Contrairement à ce que l'on pourrait penser en observant l'arborescence des pages, une url ne correspond pas à une adresse physique : on peut naviguer dans une url comme on navigue dans des dossiers (en allant dans le dossier enfant, etc.), mais il est possible de redéfinir des url, de les retravailler : c'est l'URL Rewriting. L'objectif de cette réécriture sera alors de créer une URL optimisée pour les moteurs de recherche, c'est-à-dire une url :

- Qui contient maximum 5 paramètres (idéalement, 0) (Un paramètre d'URL est une liste de valeurs apparaissant dans l'adresse de la page, précédée d'un point d'interrogation (?) ou un Et commercial (&) ; par exemple : www.monsite.fr/mapage.html**?login=test&p=1**) dans la mesure du possible
- Qui contient les mots clefs de la page concernée

Les seuls séparateurs de mots reconnus par l'ensemble des moteurs de recherche sont le tiret (-) et le slash(/).



Exemple d'URL non optimisée

Exemple d'URL optimisée

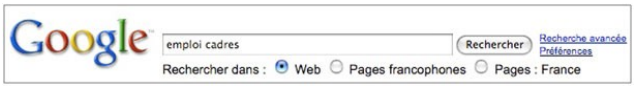
4.4. Mots en gras

Les mots en gras (entre balises et <b>, et non par mise en forme CSS) sont valorisés par les moteurs de recherche, et considérés comme plus importants que le reste du texte. Les expressions les plus importantes doivent donc être mises entre des balises de mise en exergue du texte, dans la mesure du raisonnable.

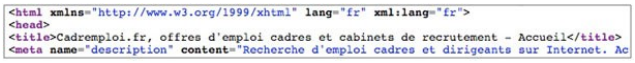
Les mots en gras sont un des aspects importants de la SEO.

4.5. Le titre

Le titre qui apparaît dans la barre de titre de la fenêtre est défini par la balise <title>. C'est l'élément généralement considéré aujourd'hui comme déterminant pour le bon positionnement d'une page. Il faut donc que les mots clefs importants, sur lesquels on souhaite être bien positionné sur Google, soient présents dans le <title> de la page.



L'utilisateur rentre un ensemble de mots-clefs dans le champ de recherche



Le <title> du site CadreEmploi

La page d'accueil apparaît dans les résultats, avec le contenu de la balise title en première ligne

Bien sûr, il ne suffit pas de juxtaposer les mots clefs pour être bien positionné. Généralement, on considère qu'un <title> doit avoir une taille maximale de 200 caractères. De plus, les deux ou trois premiers mots ont plus de poids pour Google. Il faut donc passer l'expression la plus importante en premier dans le titre.

Pour avoir du poids dans l'algorithme des moteurs de recherche, chaque <title> doit être unique sur le site (chaque page doit avoir son propre titre).

4.6. Bots, Robots.txt et Sitemap.xml

Il est possible d'aider les moteurs de recherche dans leur indexation. En effet, ces derniers utilisent des robots (Bots), comme le GoogleBot, qui parcourent l'ensemble d'internet en suivant tous les liens qu'ils trouvent sur leur passage (" Crawling "). Certains de ces bots suivent les instructions contenues dans des fichiers d'instruction, placés à la racine du site :

4.6.1. Le fichier robots.txt

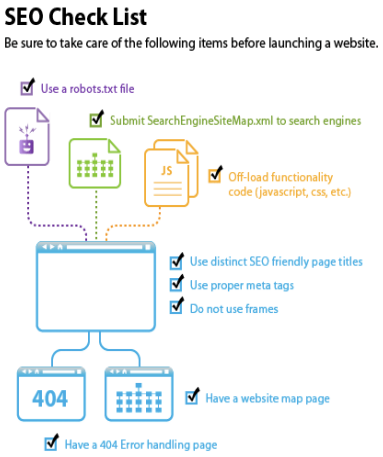
Le fichier robots.txt contient permet de préciser au Bot si la page doit être indexée (répertoriée) ou non. Cela peut-être utile pour cacher certaines pages sur Internet. Malheureusement, seul Google semble respecter ces instructions. Dans la pratique, interdire l'indexation d'une page (ou d'un dossier) aux bots à l'aide d'un robots.txt est illusoire.

4.6.2. Le fichier sitemap.xml

Ce fichier contient la liste des pages d'un site web. Cela permet d'éviter que des pages orphelines (des pages auxquelles on ne peut pas accéder directement en cliquant sur un lien, mais uniquement en tapant l'URL dans la barre d'adresse) ne soient pas indexées par les moteurs de recherche. Il ne doit pas y avoir de lien mort sur un site (" page 404 "). En effet, Google pénalise largement les sites sur lesquels il existe des erreurs de ce type.

4.7. Synthèse

Pour simplifier la compréhension de ces techniques de base de la SEO d'optimisation technique, nous pouvons consulter l'illustration suivante, qui synthétise cette section :



Technical SEO Check List (Elliance)

Source : SEO CHECK LIST - Elliance (Lien61)

5. Les techniques de base de la SEO : optimisation sémantique

5.1. Qualité du contenu

Plus un contenu est pertinent, plus il est valorisé par les moteurs de recherche (REUSSIR SON REFERENCEMENT WEB de O. Andrieu (Eyrolles, 2007) - Chap. 2 p.33 et suivantes).

De manière très générale, la pertinence d'une page web pour une requête précise est calculée :

- par la présence du mot dans la page et par sa position (dans le titre, au début de la page...)
- par la densité du mot et des mots proches (synonymes,...), c'est-à-dire le nombre d'occurrences par rapport au nombre de mots de la page.
- par la manière dont le mot est mis en exergue

(balises de titre, gras)

- par le poids dans la base du moteur : moins le contenu est fréquent dans la base de données du moteur de recherche, plus la page est favorisée
- par la correspondance exacte ou proche avec les termes recherchés (L'algorithme de Google est suffisamment évolué pour prendre en compte des mots voisins. Il se base plus sur l'analyse de lexèmes que sur une analyse syntaxique stricte. Pour plus d'information, Cf. Comment Google traite-t-il le sens des lexemes ? (Lien62) - Webankinfo)
- par la proximité des mots importants de la page avec ceux de la requête

A ces critères s'ajoute un indice de popularité, introduit par Google. Cet indice est calculé de manière complexe et récursive, de sorte que plus il y a de pages pertinentes et populaires pointant vers un site, plus son indice de popularité (page Rank) est élevé, la pertinence et la popularité de ces pages étant elles-mêmes calculées de la même façon.

5.2. Mise à jour des données

Une page figée reflète un contenu figé. Si un contenu figé peut être très bien positionné, il est plus facile d'optimiser une page si celle-ci évolue de temps en temps (le délai des modifications varie selon le type de page ; il va de soi qu'une page d'actualité est mise à jour plus souvent qu'une page d'encyclopédie. La première n'en sera pas pour autant privilégiée).

5.3. Une page par thématique

Google (et les moteurs de recherche en général) analyse la pertinence d'une page selon son contexte. Il convient donc que ce contexte soit le plus ciblé et précis possible. Plus la thématique de la page sera définie, plus elle répondra à une requête déterminée des utilisateurs des moteurs, plus elle remontera dans la page de résultats (Search engine optimization starter guide (Lien63) - Google). Il sera d'ailleurs plus facile de gérer les balises <title> et <meta> sur de telles pages.

5.4. Position du texte

Google privilégie les premiers contenus de la page (on estime généralement que les trente à quarante premiers mots ont le plus de poids). A l'inverse de la " presse " traditionnelle, ou le contenu le plus important est placé à la fin pour attirer le lecteur le plus longtemps possible, sur Internet, le contenu pertinent doit toujours être placé devant. Une page optimisée ne contiendra donc pas un menu comme premier élément, mais bel et bien du texte (une phrase d'accroche, un slogan pertinent…), contrairement à ce que l'on voit le plus souvent.

5.5. Smart pages

Depuis quelques temps, une pratique devient de plus en plus courante, même si elle existe depuis longtemps : celle de l'emploi de Smart Pages (ou " Landing Pages "). Il s'agit de pages optimisées pour les moteurs de recherche, qui ressortiront comme résultats privilégiés sur ces derniers. La particularité de ces pages est qu'elles sont généralement orphelines pour le visiteur, c'est-à-dire que, sur le site, aucun lien ne guide le visiteur vers une Smart Page, et aucune Smart Page n'est mise en avant. Ces pages ne sont destinées qu'à rediriger l'utilisateur vers une page interne du site.

ANOTHER SEO BLOG (LES LANDING PAGES – Another SEO Blog (Lien64)) propose une image intéressante : les Smarts Pages seraient l'équivalent des Bandes annonces des films :

1. " Ambassadrice du film auprès du public, elle va le convaincre d'aller le voir, sans elle le film n'aura certainement pas le même succès, elle est donc le premier point de contact dont tout dépend. "
2. " Economique à réaliser, elle permet aussi de "recycler" les acteurs du film, les décors, les costumes, les dialogues et les effets spéciaux, tout ce qui coûte cher et sans quoi elle n'aurait aucun impact. La bande annonce contient donc la quintessence des ressources d'un film. "

Voici un exemple d'une smart page, pour le site franchise-cargo.fr. L'objectif de cette page est d'être bien positionnée dans les résultats de recherche de l'expression " franchise location automobile " :

Exemple de Smart Page ; l'expression franchise en location automobile est valorisée

5.6. Le Duplicate Content

Un élément majeur de la SEO est le Duplicate Content (ou contenu dupliqué). Il s'agit de pages non uniques, c'est-à-dire des pages dont le contenu peut se retrouver en allant à une autre URL. Dans le cas de page en Duplicate Content, seule la première source trouvée est valorisée. Les autres pages sont pénalisées par Google, qui baisse leur Indice de Confiance. De très nombreuses pages sont en Duplicate Content. En effet, même si les moteurs de recherche parviennent à isoler les contenus et à ne pas prendre en compte les éléments non déterminants (menus, actualités… qui sont communs à l'ensemble des pages d'un site), les webmasters n'hésitent pas à copier des contenus d'autres sites (Google détecte la proximité entre les textes, même si des mots ou phrases ont été changées). De même, la structure d'un site peut générer du Duplicate Content. Par exemple, un blog qui archive l'ensemble de ses pages se retrouve avec des URL différentes qui pointent vers le même contenu.

6. Les techniques de base de la SEO : optimisation externe

6.1. Les liens entrants (Back Link)

L'algorithme de Google a longtemps été critiqué pour l'importance qu'il donnait aux " **Back Links** ", ou " Liens Entrants ". Ces termes désignent les liens d'un site qui pointent vers une page internet. Tous les liens qui pointent vers un site A sont comptabilisés par les moteurs. Plus il y

a de liens vers une page d'un site A, plus cette page est " populaire ", et plus son Page Rank est élevé. Avant 2008, de très nombreux sites abusaient de ces Back Links et multipliaient les sites fantômes (sites satellites) pour créer des fermes de liens (" Farm Links ") vers d'autres sites, afin d'améliorer le Page Rank de leurs pages. Aujourd'hui, Google limite le poids du Back Link de deux manières :

- Tout d'abord, en tenant compte de l'Indice de Confiance du site qui contient le lien.
- Ensuite, en intégrant la notion de " jus " (" **Seo Juice** " - cf ci-dessous en 6.1.2) à son algorithme.

6.1.1. Back Links et Indice de Confiance

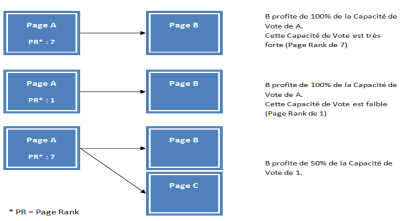
En effet, plus une page a un Indice de Confiance élevé, plus ses liens sont valorisés par Google. Il existe ainsi différents types de liens, plus ou moins valorisés (Tutoriel référencement naturel sur moteurs (Lien65), III-E-3 - G. Gregoires) :

Type de lien	Comportement de Google
Les liens externes	Back Link " standard "
Les liens externes de sites situés sur le même serveur	Back Link défavorisé
Les liens circulaires (un site A pointe vers B, qui lui aussi pointe vers A, soit : A = B = A)	Back Link non pris en compte
Les liens circulaires larges (A = B = C = A)	Back Link " standard "
Les liens nuisibles (liens sur des sites black listés pour fraude)	Même si Google affirme le contraire, le Back Link semble pénalisant
Les liens en or (sur des sites .edu, .org, .gov, .gouv…)	Back Link très valorisé

6.1.2. Back Links et SEO Juice

La notion de " jus " est très couramment employée par les référenceurs. Elle désigne l'impact qu'a un lien sur le positionnement d'une autre page. Plus l'impact d'un lien est important, plus ce lien transmet de " jus " à la page (O. Andrieu (REUSSIR SON REFERENCEMENT WEB, Eyrolles, 2007 - p.105) parle parfois de " Capacité de Vote ").

Comme l'image le laisse entendre, plus il y a des liens sur la page, plus le jus est réparti sur l'ensemble de ces liens. Un lien unique donnera 100% du jus à la page pointée, deux liens en donneront chacun 50%, etc.



Le concept de SEO Juice : chaque flèche représente un lien. Le jus est réparti entre les liens de la page

6.1.3. Connaître le nombre de Back Links d'un site

Pour avoir une estimation du nombre de Back Links d'un site, il suffit de taper : " link:www.adressedusite.fr " dans Google ou dans Yahoo pour accéder aux statistiques. De même, pour voir quel est l'ensemble du contenu indexé d'un site, il suffira de taper " site:www.adressedusite.fr "

6.2. Le Link Baiting

Le Link Baiting consiste à attirer le plus possible de liens entrants (back Links) vers une page grâce au contenu de cette page. C'est le principe du " Buzz ". C'est pourquoi la plupart des sites importants organisent des jeux concours, des sondages, des guides de consommateurs, des comparatifs… En plus de l'effet médiatique que cela peut avoir, ces concours, sondages… créent autant de liens qui pointent sur le site qu'il y a de personnes à en parler sur Internet. Plus le " Buzz " est important, plus le nombre de Back Links est elevé, et plus le poids (en Page Rank) des pages d'un site est important.

7. Autres critères importants

7.1. Les textes alternatifs et titres des images

Une image est insérée dans une page web à l'aide la balise <img>. Celle-ci comporte deux attributs importants :

- L'attribut " alt ", c'est-à-dire le texte alternatif, qui apparaît au cas où l'image ne se charge pas
- L'attribut " title ", qui apparaît quand le visiteur passe la souris sur l'image

Il est donc possible d'insérer des mots clefs, de manière efficace (tant que cela reste cohérent) dans ces attributs des images.

7.2. Ancienneté du nom de domaine

Plus un nom de domaine est ancien, plus l'Indice de Confiance des pages qui y sont hébergées est important. Le positionnement d'un site s'améliore donc automatiquement avec le temps (c'est pour cela que certains noms de domaines sont achetés et inutilisés durant des années, puis revendus).

7.3. Type d'adresse

Comme nous l'avons évoqué, les pages sur une adresse en .gov, .org, .gouv, .edu… c'est-à-dire une adresse d'un site officiel ou institutionnel, sont largement valorisées par Google.

7.4. Vitesse d'affichage

Plus une page s'affiche rapidement, plus son Page Rank augmente (même si cette augmentation est peu conséquente).

Retrouvez la suite de l'article de Jean-François Lépine en ligne : Lien66

Active Record : les migrations

Dans cet article, faites connaissance avec Active Record en commençant par manipuler des structures de bases de données grâce aux migrations.

1. Qu'est ce qu'ActiveRecord ?

ActiveRecord est le composant qui permet la manipulation des données dans les applications Ruby on Rails. ActiveRecord est aussi et avant tout un design pattern pour lier les données d'une base de données à des modèles de programmation. Voici la définition très juste de ce design pattern trouvée sur Wikipedia :

« En génie logiciel le patron de conception (design pattern) active record (enregistrement actif en anglais) est une approche pour lire les données d'une base de données. Les attributs d'une table ou d'une vue sont encapsulés dans une classe. Ainsi, l'objet instance de la classe est lié à un tuple de la base. Après l'instanciation d'un objet, un nouveau tuple est ajouté à la base au moment de l'enregistrement. Chaque objet récupère ses données depuis la base; quand un objet est mis à jour, le tuple auquel il est lié l'est aussi. La classe implémente des accesseurs pour chaque attribut. »

Voici un exemple concret :
Si nous avons une table « contacts » dans la base de données de notre application Rails, alors nous avons forcément une classe Contact qui peut être manipulée via ActiveRecord. Voici un exemple de code :

```
  c = Contact.new(:first_name => "Dubois",
:last_name => "Vincent")
  c.save    # création du tuple
```

2. Anatomie d'une migration

Avant de rentrer dans le détail des migrations Active Record, voyons un exemple de migration que nous avons créé dans le tutoriel précédent (Lien67) :

db/migrate/20090422185716_create_contacts.rb

```
  class CreateContacts < ActiveRecord::Migration
    def self.up
      create_table :contacts do |t|
        t.string :name
        t.string :email

        t.timestamps
      end
    end

    def self.down
      drop_table :contacts
    end
  end
```

Cette migration ajoute une table « contacts », possédant deux colonnes « name » et « email » de type texte. La méthode « timestamps » quant à elle, ajoute des colonnes d'informations sur les dates de mises à jour des enregistrements. Nous détaillerons tout cela un peu plus tard.

2.1. Les migrations sont des classes

Une migration est une sous-classe de la classe **ActiveRecord::Migration**. A ce titre, elle définit deux méthodes : une méthode « **up** », qui permet d'ajouter une modification à la base de données, et une méthode « **down** », qui permet de retirer cette modification de la base de données.

Active Record fournit des méthodes qui permettent de définir des tâches de modification de la structure de la base de données :

- create_table
- change_table
- drop_table
- add_column
- change_column
- rename_column
- remove_column
- add_index
- remove_index

Les migrations sont effectuées lors de l'exécution de la commande « **rake db:migrate** ». Si une opération ne se déroule pas correctement, la migration qui a échoué fait l'objet d'un rollback.

Les noms des fichiers de migrations sont du type **YYYYMMDDHHMMSS_action_entite.rb**. Dans le cas de ce tutoriel, le nom du fichier est : 20090418171815_create_contacts.rb. Le nom du fichier est sûrement différent chez vous, il dépend de l'heure de la création de l'entité « Contact ». Tous les fichiers de migrations se situent dans le répertoire : **db/migrate/**. La date de chaque nouveau fichier de migration étant postérieure à celle du dernier, chaque migration sera toujours exécutée dans l'ordre que vous aurez pré-établi et « rollbackée » dans l'ordre inverse.

Si vous souhaitez effectuer un « rollback » sur la base de données, il existe une commande qui opère le contraire de « rake db:migrate ». C'est la commande « **rake db:rollback** ». N'hésitez pas à l'utiliser en cas de besoin.

3. Créer une migration

3.1. Créer un modèle

Lors de la création d'un modèle, la migration adéquate est automatiquement créée. Par exemple, si on exécute la commande (ne le faites pas, car l'entité « Contact » existe déjà) :

```
  ruby script/generate model Contact name:string
email:string
```

Alors, automatiquement, un fichier **[yyyymmddhhmmss]_create_contacts.rb** est créé (cf. §1). Vous pouvez ajouter à la main autant de colonnes que vous le souhaitez au sein du bloc create_table.

3.2. Créer une migration seule

Vous pouvez si vous le souhaitez créer seulement une migration, hors du cadre de la création d'une entité ou d'un « scaffold ». Procédons par exemple à l'ajout d'un attribut « first_name » correspondant au prénom du contact et changeons l'attribut « name » en « last_name » à cette occasion.

```
  ruby script/generate migration
AddFirstNameToContacts
```

Voici le contenu du fichier de migration par défaut :

```
  class AddFirstNameToContacts <
ActiveRecord::Migration
    def self.up
    end

    def self.down
    end
  end
```

Evidemment, ça ne fait pas grand chose pour l'instant. Précisons tout d'abord que les méthodes « up » et « down » opèrent chacune respectivement une montée et une descente de version de migration. Dans ce but, nous allons donc prévoir de quoi revenir sur nos pas, c'est à dire envisager la suppression de la colonne « first_name » et le renommage de la colonne « last_name » en « name ». Commençons par la méthode « up » :

```
  def self.up
    add_column :contacts, :first_name, :string
    rename_column :contacts, :name, :last_name
  end
```

Observons le code. Nous avons tout d'abord ajouté la colonne « first_name » grâce à la méthode « add_column » qui prend en paramètre le nom de la table, le nom de la colonne, le type de colonne et d'autres paramètres éventuels non précisés ici. Ensuite, la méthode « rename_column » nous permet de renommer la colonne « name ». Nous devons maintenant implémenter l'opération inverse dans la méthode « down » :

```
  def self.down
    remove_column :contacts, :first_name
    rename_column :contacts, :last_name, :name
  end
```

Facile, n'est ce pas ? Si nous avions seulement voulu ajouter la colonne « first_name », nous aurions même pu le faire en tapant :

```
  ruby script/generate migration
AddFirstNameToContacts first_name:string
```

Et Rails aurait généré pour nous les lignes contenant « add_column » et « remove_column ». A l'inverse, une migration nommée « RemoveFirstNameToContacts » aurait généré une méthode « remove_column » dans la méthode « up » et une méthode « add_column » dans la méthode « down ». Le principe de « **convention over configuration** » est poussé ici à son paroxisme. Nous pouvons d'ailleurs aller un peu plus loin si nous le souhaitons en ajoutant plusieurs descriptions de colonnes à la commande : elles seront toutes ajoutées à la migration.

4. Ecriture d'une migration

4.1. Créer une table

L'écriture d'une table passe par le bloc de code **create_table**. Voici un exemple de code typique :

```
  create_table :contacts, :force => true do |t|
    t.string :name
  end
```

L'objet table « t » manipulé par le bloc nous permet de créer des colonnes dans la table. Notons au passage l'usage d'un deuxième paramètre nommé « :force », qui permet de recréer la table dans le cas où elle existe déjà.
Il y a deux façons de créer des colonnes. Il y a l'ancienne façon (à l'origine dans les versions antérieures de Rails) :

```
  t.column :name, :string, :null => false
```

Et il y a la nouvelle manière introduite depuis peu dans Rails, du moins de façon officielle. Les migrations qui l'utilisent sont surnommées « sexy migrations ». C'est celle que nous appliquerons par la suite :

```
  t.string :name, :null => false
```

Il est intéressant de préciser que nous n'avons pas de colonne d'identifiant à créer. En effet, Rails crée pour nous automatiquement une colonne de nom « id » qui joue le rôle de clé primaire dans la table. Cela dit, il existe des moyens d'outrepasser ceci, mais là encore, il vaut mieux suivre le principe « convention over configuration ».

Les types supportés par Active Record lors de la création d'une colonne sont :primary_key, :string, :text, :integer, :float, :decimal, :datetime, :timestamp, :time, :date, :binary et :boolean. Il est possible d'utiliser d'autres types de données non supportés de la façon suivante :

```
  t.column :name, 'autre_type', :null => false
```

Mais le type ne sera supporté probablement que par la base de données que vous utiliserez. C'est donc à utiliser avec précaution.

4.2. Modifier une table

La méthode **change_table**, proche cousin de la méthode **create_table**, est utilisée pour les tables déjà existantes. L'objet manipulé par le bloc de code dispose de plus de méthodes. En voici un exemple :

```
  change_table :contacts do |t|
    t.remove :email, :country
    t.string :address
    t.index :first_name
    t.rename :first_name, :firstname
  end
```

Toutes ces méthodes sont des raccourcis. Si nous avions dû créer ceci de façon standard, nous aurions fait :

```
  remove_column :contacts, :email
  remove_column :contacts, :country
  add_column :contacts, :address, :string
  add_index :contacts, :first_name
  rename_column :contacts, :first_name,
:firstname
```

Tout cela nous permet de ne pas réécrire plusieurs fois le même code et donc de respecter le principe « DRY » de Rails.

4.3. Helpers spéciaux

Rails fournit un certain nombre de méthodes « helpers » spéciales qui permettent de manipuler la structure des tables ou encore les relations entre les tables. La première, que nous avons déjà vu plus haut dans ce tutoriel, est la méthode « **timestamps** ». Cette méthode déclare automatiquement deux colonnes, nommées « **created_at** » et « **updated_at** », qui permettent respectivement de tracer les dates de création et de mise à jour des enregistrements dans la table.

La deuxième méthode que nous allons voir et qui est très importante est la méthode « **references** ». C'est cette méthode qui permet à Rails de structurer les tables afin d'établir le lien entre deux d'entre elles. Voici un exemple d'utilisation :

```
  t.references :category
```

L'exemple ci-dessus permet de générer automatiquement le lien entre la table actuelle et la table « category » via un champ « category_id » qui va être automatiquement créé.

5. Exécuter des migrations

Rails nous fournit un certain nombre de tâches Rake qui permettent de manipuler les migrations. Celle que nous utiliserons le plus souvent est la commande « **rake db:migrate** ». Elle peut être agrémentée du numéro de version de migration à atteindre. Dans ce cas, les migrations suivantes ne sont pas effectuées. Par exemple :

```
  rake db:migrate VERSION=20090420203025
```

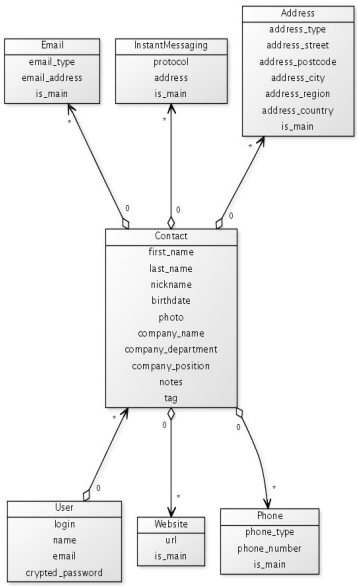
Dans l'exemple ci-dessus, la version correspond au préfixe du fichier de migration. Nous pouvons également exécuter l'opération inverse, c'est-à-dire décrémenter le niveau des migrations. Pour cela, Rails met à notre disposition la méthode « **rake db:rollback** ». Seule, la commande permet de descendre d'un niveau de migration à chaque exécution. On peut, comme pour la commande « migrate », préciser « VERSION=... ». Dans ce cas, on descend le niveau de migration jusqu'à la version donnée en paramètre. Le mieux pour l'opération rollback, est quand même de préciser le paramètre « STEP » pour descendre plusieurs niveaux de migrations à la fois. Par exemple :

```
  rake db:rollback STEP=3
```

Lorsque vous invoquez les commandes « rake db:migrate » ou « rake db:rollback », la tâche « **rake db:schema:dump** » est exécutée automatiquement et met à jour le fichier schema.rb qui se situe dans le répertoire db. Ce fichier reflète alors la structure de la base de données. Il est possible de recharger la structure de la base de données à partir du fichier schema.rb à l'aide de la commande « **rake db:schema:load** ».

6. Modéliser l'application « Mes Contacts »

Voici le diagramme de classes de notre application de gestion des contacts : nous souhaitons qu'un utilisateur puisse gérer ses propres contacts. Chaque contact pourra avoir plusieurs numéros de téléphone, messageries instantanées, sites web, adresses postales, et adresses email.



Avant de créer les migrations qui vont nous permettre de modéliser la base de données, nous devons supprimer les migrations que nous avons ajouté précédemment :

```
rake db:rollback STEP=2
rm db/migrate/*
```

Maintenant que la base de données est vierge, nous pouvons créer nos entités. Il est néanmoins temps de faire un point sur le modèle. Certaines choses peuvent être mutualisées. C'est le cas par exemple de la gestion des utilisateurs. Plutôt que de créer de zéro une gestion des utilisateurs, nous allons appliquer le principe « DRY » et utiliser le plugin « **restful_authentication** ». Ce plugin va nous permettre de gérer tous les enregistrements et

connexions des utilisateurs à l'application. Commençons par l'installer :

```
  ruby script/plugin install
git://github.com/technoweenie/restful-
authentication
```

Le plugin est maintenant installé dans notre répertoire *vendor/plugins*. Pour créer l'entité « User » et la structure des tables utilisateurs en base de données, lançons la commande :

```
  ruby script/generate authenticated user
sessions --include-activation
```

Cette commande crée beaucoup de choses, notamment l'entité « User » (fichier *app/models/user.rb*) et le fichier de migration correspondant *db/migrate/[VERSION]_create_users.rb*. Sachez, pour finir, que le plugin génère des contrôleurs et vues qui permettent de gérer les utilisateurs et les sessions d'authentification. Il rajoute au fichier *routes.rb* des routes qui facilitent l'utilisation de ces contrôleurs (exemples : /signup, /register, /login, /logout). Nous verrons en détail ce plugin dans un autre chapitre quand nous en saurons plus sur les contrôleurs. Le but ici était d'initialiser l'entité « User ».

Concentrons-nous maintenant sur la création des autres entités. A cette fin, lançons les commandes suivantes :

```
  ruby script/generate model Phone
phone_type:integer phone_number:string
is_main:boolean

  ruby script/generate model InstantMessaging
protocol:integer address:string is_main:boolean

  ruby script/generate model Website url:string
is_main:boolean

  ruby script/generate model Address
address_type:integer address_street:string
address_postcode:string
  address_city:string address_region:string
address_country:string is_main:boolean

  ruby script/generate model Email
email_type:integer email_address:string
is_main:boolean

  ruby script/generate model Contact
first_name:string last_name:string
nickname:string birthdate:date
  photo:string company_name:string
company_department:string company_position:string
notes:text
```

Nous devons ensuite rajouter à la main les liens entre les différentes entités. Pour cela, nous allons utiliser la méthode « **references** » que nous avons étudié dans le paragraphe 3.3. Ajoutons la ligne :

```
  t.references :contact
```

dans les méthodes **create_table** des fichiers de migrations correspondant aux entités « Phone », « InstantMessaging », « Website », « Address » et « Email » pour faire le lien entre ces entités et l'entité « Contact ». Le lien entre les deux entités « Phone » et « Contact » se lit : un contact possède de zéro à plusieurs numéros de téléphone. Pour finir, il reste à faire le lien entre les utilisateurs et les contacts. Dans le fichier de migration de l'entité « Contact », ajoutons :

```
  t.references :user
```

Une dernière chose maintenant. Si vous êtes observateur, vous vous êtes probablement aperçu qu'il manque un attribut à l'ensemble du modèle. Il y a un attribut « tag » appartenant à l'entité « Contact ». Cet attribut veut seulement dire que l'on doit pouvoir « tagger » un contact avec un à plusieurs mots. On pourra générer par exemple avec ce genre de données un « *nuage de tags* » des contacts. Cette fonctionnalité existe déjà dans Rails par l'intermédiaire de plugins. **ActsAsTaggableOnSteroids** en est un. Nous l'étudierons plus en détails dans un autre chapitre. Nous allons l'installer et l'utiliser pour le raccorder à notre modèle :

```
  ruby script/plugin install
http://svn.viney.net.nz/things/rails/plugins/acts
_as_taggable_on_steroids
```

Ensuite nous devons créer la migration correspondante à l'aide de la commande :

```
  ruby script/generate acts_as_taggable_migration
```

La dernière chose à faire est de rajouter la fonctionnalité de « tagging » aux contacts. Nous devons ouvrir le fichier *app/models/contact.rb* et le compléter comme suit :

```
  class Contact < ActiveRecord::Base
    acts_as_taggable
  end
```

Voilà, tous nos modèles sont décrits, toutes les migrations correspondantes sont écrites et l'apport de certains plugins a permis de nous pré-mâcher beaucoup de travail. N'oublions pas de mettre à jour la base de données avant de passer à la suite :

```
  rake db:migrate
```

7. Conclusion

ActiveRecord est LE composant de Rails qu'on se doit de connaître sur le bout des doigts. Une bonne conception de la base de données passera forcément par une écriture correcte des migrations. Les migrations sont un outil puissant car elles permettent de gagner un temps fou lors de l'initialisation d'un projet Rails. On ne doit pas les négliger.

Dans la prochaine partie, nous aborderons le développement par les tests dans Ruby on Rails. Nous verrons quel est leur rôle et comment en tirer un maximum de bénéfice.

Vous pouvez retrouver le source de ce tutoriel (Lien68).

Retrouvez l'article de Vincent Dubois en ligne : Lien69

Mariage de la Programmation Orientée Objet et de la Programmation Générique : Type Erasure

Si vous utilisez les templates du C++ pour écrire des composants génériques, vous avez déjà peut-être voulu les combiner à la programmation orientée objet, via de l'héritage par exemple. Peut-être avez-vous eu des difficultés ; cet article va vous présenter une technique répandue, nommée Type Erasure, qui vous permettra de tirer profit des deux mondes sans perdre en flexibilité ni en maintenabilité.

1. Introduction

Le C++ est un langage riche. On peut résoudre des problèmes identiques avec des approches différentes. Certains utiliseront des classes, d'autres un style plus basé sur les fonctions, etc. En effet, le C++ supporte plusieurs paradigmes, comme la programmation orientée objet, la programmation générique, la programmation fonctionnelle, la programmation logique, etc. Toutefois, seuls les deux premiers ici sont nativement supportés par le C++ (c'est à dire sans utiliser de bibliothèque). Seulement, il peut arriver que l'on ait besoin de mélanger les styles, afin de bénéficier des avantages de l'un et de l'autre et l'on aimerait bien ne pas avoir de difficultés à les mélanger. En effet, lorsque l'on écrit des composants génériques (paramétrés via le mécanisme de templates), on veut pouvoir les injecter dans du code Orienté Objet ne serait-ce par exemple qu'en agissant uniformément sur une liste de composants via le polymorphisme d'héritage... Or ceci n'est pas trivial et possède ses pièges.

Ce n'est toutefois pas si élémentaire de tirer profit, dans notre cas, du mélange de la programmation orientée objet avec la programmation générique, de par les règles qui régissent le langage C++. Par exemple, la plupart d'entre nous ont probablement déjà fait face au problème suivant : soit une classe template définie comme suit.

```
template <class T>
class A
{
        T t;
        // ...
};
```

Comment peut-on alors stocker ensemble des objets A<T1>, A<T2>, etc, où T1, T2 et autres sont des types différents ? C'est l'une des questions qui va être au centre de cet article. Nous allons dans un premier temps mettre en avant le problème auquel répond le principe de *Type Erasure*, puis nous introduirons ce dernier en donnant une solution à notre exemple de l'étude de cas. Enfin, nous appliquerons le principe dans un exemple plus concret et utile, puis nous réécrirons la classe boost::any avant de conclure cet article sur les limites de cette approche.

2. Etude de cas

Nous allons commencer par un problème trivial : prenons la classe template suivante.

```
template <typename T>
class A
{
        T t;
        public:
        A(const T& t_) : t(t_) { }
        void print(std::ostream& o) const { o <<
t; }
};
```

A première vue, rien de très compliqué. Une classe template qui renferme une valeur d'un certain type, et qui permet d'envoyer cette valeur dans un flux. Elle n'est pas terriblement utile, mais sera une excellente base.

Désormais, imaginons vouloir stocker un nombre arbitraire d'objets de ce type, avec des paramètres *T* différents, dans un même conteneur. Ainsi, nous voudrions avoir par exemple un std::vector contenant des A<int>, A<std::string>, A<char>, etc.

Contrairement à ce que l'on peut penser parfois, le type A<int> est différent de A<char>, et on ne peut pas faire un std::vector<A> ou autre std::vector< A<int> > qui permettrait de stocker des A<T> quels que soient les types T. N'oubliez pas en effet que écrire A<int> génère une classe A<int> en remplaçant T par int dans le code de la classe. A<char> et A<int> n'ont en commun que le code qui a permis de les générer, mais ce sont deux types incompatibles. Toutefois, le fait qu'elles proviennent d'un même "modèle", "patron", va nous servir ici. Voyons maintenant la solution qui utilise, vous vous en doutez étant donné le titre de l'article, l'héritage.

3. Principe de Type Erasure

La clé ici est que nous voulons d'une part avoir des A<T> à manipuler en tant que tels, et d'autre part nous voulons pouvoir stocker des A<> avec des types T différents en manipulant l'ensemble de manière uniforme. Dans notre cas, nous voulons stocker les A<T> pour pouvoir les afficher à travers un flux *std::ostream*. Vous vous en doutez, c'est une mission pour l'héritage et la virtualité. Ecrivons donc une classe abstraite qu'implémenteront toutes les classes A<T> quel que soit le type T.

```
class A_base
{
        virtual void do_print(std::ostream&)
const = 0;
        public:
```

```
        void print(std::ostream& o) const
{ do_print(o); }
};
```

Il nous suffit ensuite de modifier notre classe template A<T> pour qu'elle hérite et implémente cette classe abstraite.

```
template <typname T>
class A : A_base
{
        T t;
        void do_print(std::ostream& o) const { o
<< t; }
        public:
        A(const T& t_) : t(t_) { }
};
```

Et voilà, nous pouvons désormais d'une part stocker tous les A<T> dans un même conteneur via la classe abstraite A_base, et d'un autre côté préserver leur utilisation sans passer par l'interface A_base. Le coût n'est absolument pas grand. Il y aura simplement une *vtable* contenant toute l'interface publique par laquelle on manipulera uniformément les différents A<T>. Il y a un deuxième coût toutefois : vous serez obligé de stocker des pointeurs au lieu de stocker des valeurs simples. Voici donc notre std::vector et une manipulation uniforme sur les A<T>, dans un code complet que vous pourrez compiler si vous avez Boost. Sinon, remplacez les pointeurs intelligents par des A_base* et faites des **delete** à la fin de main sur ces *A_base**.

```
#include <iostream>
#include <string>
#include <vector>
#include <tr1/shared_ptr.h>
// si votre compilateur ne fournit pas
tr1/shared_ptr.h
// vous pouvez obtenir la même chose depuis
// <boost/tr1/memory.hpp>
// qui est donc inclus dans Boost

class A_base
{

        virtual void do_print(std::ostream&)
const = 0;
        public:
        void print(std::ostream& o) const
{ do_print(o); }
};

template <typename T>
class A : public A_base
{
        T t;
        void do_print(std::ostream& o) const { o
<< t; }
        public:
        A(const T& t_) : t(t_) { }
};

// permet d'avoir des A< A<T> >
std::ostream& operator<<(std::ostream& o, const
A_base& a)
{
        a.print(o);
        return o;
}

int main()
{
  std::vector< std::tr1::shared_ptr<A_base> >
vec; // on utilise std::tr1::shared_ptr, un des
pointeurs intelligents de Boost/TR1, cf [1] [2]
  vec.push_back(std::tr1::shared_ptr<A_base>(new
A<int>(42)));
  vec.push_back(std::tr1::shared_ptr<A_base>(new
A<std::string>("Type Erasure")));
  vec.push_back(std::tr1::shared_ptr<A_base>(new
A<char>('c')));
  std::vector< std::tr1::shared_ptr<A_base>
>::iterator it = vec.begin(); // vivement auto de
C++0x ;-)
  for ( ; it != vec.end(); it++ )
    {
      (*it)->print(std::cout);
    }
  return 0;
}
```

[1] Pointeurs Intelligents (Lien70), Loïc Joly
[2] Boost.SmartPtr, les pointeurs intelligents de Boost (Lien71), par Matthieu Brucher

Ce code affiche donc
42Type Erasurec.

Certes, vous voyez que ce code marche très bien dans notre cas simple, mais vous vous demandez quelle utilité il peut avoir dans la pratique, dans nos projets de tous les jours. C'est pourquoi dans la section suivante nous allons nous attaquer à un problème plus sérieux et plus réaliste.

4. Application dans un cas concret

En C++ moderne, l'une des bonnes pratiques lorsque l'on écrit des composants modulaires (souples, paramétrables) est de découper ses composants en classes de politiques (*Policy Classes*) comme décrit dans l'article Classes de Traits et de Politiques (Lien72) ou encore dans le livre *Modern C++ Design* de Andrei Alexandrescu, évangéliste de cette pratique.

C'est pourquoi nous allons avoir affaire à un système de génération de widgets paramétré par des politiques. Celui-ci sera minimal pour garder le fil de l'article, mais tout de même consistant.

```
#include <iostream>
#include <string>

#define CHECK_FUN(Policy, Member) enum
{ Policy##Member = sizeof(&Policy::Member) > 0 };

template
  <
  class SizePolicy,
  class ClickAwarenessPolicy,
  class TextPolicy,
  class DrawingPolicy
  >
class widget_impl :
  public SizePolicy,
  public ClickAwarenessPolicy,
```

```
  public TextPolicy,
  public DrawingPolicy
{
    /* Par sûreté, on vérifie la présence des
fonctions nécessaires
       dans les politiques */
  CHECK_FUN(DrawingPolicy, draw)

  public:
  widget_impl(unsigned int width = 800, unsigned
int height = 600, const std::string& text = "")
    :  SizePolicy(width, height),
TextPolicy(text)
  {
  }

  void print_size() const
  {
    std::cout << SizePolicy::width << "x" <<
SizePolicy::height << std::endl;
  }
};

struct NoText { };
struct NonClickAware { };

struct NonResizable
{
  protected:
  unsigned int width;
  unsigned int height;

  public:
  NonResizable(unsigned int w, unsigned int h) :
width(w), height(h) { }
};

struct Resizable
{
  protected:
  unsigned int width;
  unsigned int height;

  public:
  Resizable(unsigned int w, unsigned int h) :
width(w), height(h) { }

  void resize(unsigned int w, unsigned int h)
  {
    width = w;
    height = h;
  }
};

struct ClickAware
{
  void on_click()
  {
    std::cout << "Clicked" << std::endl;
  }
};

struct ReadOnlyText
{
  protected:
  std::string text;

  public:
  ReadOnlyText(const std::string& t) : text(t)
{ }
  std::string get_text() { return text; }
};

struct EditableText
{
  protected:
  std::string text;

  public:
  EditableText(const std::string& t) : text(t)
{ }
  std::string get_text() { return text; }
  void set_text(const std::string& s) { text = s;
}
};

struct RectangularDraw
{
  void draw()
  {
    std::cout << " __________ " << std::endl;
    std::cout << "|          |" << std::endl;
    std::cout << "|__________|" << std::endl;
  }
};

struct SquareDraw
{
  void draw()
  {
    std::cout << " _____ " << std::endl;
    std::cout << "|     |" << std::endl;
    std::cout << "|     |" << std::endl;
    std::cout << "|_____|" << std::endl;
  }
};

int main()
{
  typedef widget_impl <
    NonResizable,
    NonClickAware,
    ReadOnlyText,
    SquareDraw
      > widget_t;

  widget_t w(1024, 780, "The best widget of the
world");
  w.draw();
  std::cout << "Widget's text : " << w.get_text()
<< std::endl;
  w.print_size();
  // w.set_text("foo"); ne compile par, car
ReadOnlyText
  // w.resize(); ne compile pas, car NonResizable
  // w.on_click(); ne compile pas, car
NonClickAware

  return 0;
}
```

Résumons... Nous avons une classe template widget_impl, paramétrée par différentes politiques :

- Politique de redimensionnement (Resizable ou NonResizable),
- Politique de réaction aux clicks (ClickAware ou NonClickAware),
- Politique textuelle (Aucun texte inclus, Texte

inclus en lecture, Texte inclus et modifiable),
- Politique de dessin (Dessin de carré, Dessin de rectangle).

Nous avons ensuite défini des implémentations de ces politiques. Puis dans la fonction main, nous définissons un certain type de widget *widget_t* qui n'est pas redimensionnable, ne réagit pas aux clics, affiche du texte sans possibilité de l'éditer et dont le dessin représente un carré. Déjà, quel intérêt n'est-ce pas ? Tout simplement, nous avons décomposé notre widget en fonctionnalités orthogonales (i.e. indépendantes) et pouvons ainsi soit utiliser des implémentations de fonctionnalités déjà définies, ou bien définir notre propre implémentation d'une politique donnée ! Par exemple, nous pourrions définir une implémentation de politique s'occupant de définir une fonction *draw* pour afficher un widget circulaire et passer cette implémentation de politique, sans avoir à hériter de notre type existant ou à redéfinir toute la classe widget, ou autres (comme c'est le cas avec beaucoup de toolkits GUI).

Maintenant que nous avons démontré l'utilité d'un tel design, il est temps de mettre le doigt sur un problème... fort embêtant.

Dans beaucoup de bibliothèques GUI, il y a un système de parent/enfants entre widgets. En effet, il n'est pas rare qu'une fenêtre ait la responsabilité de détruire tous les composants qu'elle comporte. Le mécanisme utilisé pour ce faire est de passer le composant parent au constructeur du composant enfant. Le composant enfant peut ainsi demander à son parent de le rajouter dans sa liste des *widgets sous sa responsabilité*. Mais comment faire pour qu'un type de widget donné, comme *widget_t* ci-dessus (qui correspond, je le rappelle, à *widget_impl<NonResizable, NonClickAware, ReadOnlyText, SquareDraw>* , puisse stocker des *widget_impl<SomeSizePolicy, SomeClickAwarenessPolicy, SomeTextPolicy, SomeDrawingPolicy>* , où chacune des implémentations de politique peut être différente d'un objet à l'autre qui doit être stocké dans la liste ?

Nous allons, comme dans la section précédente, introduire une classe de base permettant de profiter du polymorphisme d'héritage. Ainsi, dans l'interface (publique) de la classe de base, nous y définirons les fonctions virtuelles, éventuellement pures, qu'il nous faut pour que chaque widget puisse être sereinement responsable d'une liste de widgets enfants et puisse gérer leur durée de vie correctement. Nous allons d'abord définir un foncteur utilitaire, *deleter*.

```
// foncteur utilitaire qui permet de détruire en
série tous les widgets enfants via std::for_each
struct deleter
{
  template <typename T>
  void operator() (T* t)
  {
    t->destroy();
  }
};
```

Puis définissons la classe de base, *widget*.

```
class widget
{
        std::list<widget*> children; // liste des
widgets enfant, dont le widget courant est
responsable
        widget* parent; // widget parent

        void register_child(widget* w)
        {
                children.push_back(w); //
enregistre un widget enfant
        }

        widget(const widget& other); // widget
est ainsi non-copiable
        widget& operator=(const widget&
other); // ni assignable

        public:
        widget(widget* parent_) : parent(parent_)
        {
                if(parent_ != NULL)
                {
                        parent_-
>register_child(this); // si on donne un parent,
alors c'est le parent qui devient responsable du
widget courant
                }
        }

        virtual ~widget()
        {
                if(parent == NULL)
                        widget::destroy(); //
s'il n'y a pas de parent, le destructeur détruit
le widget courant
                        // sinon, c'est le parent
qui s'en charge via la fonction destroy
                        // cela permet d'éviter
un double-appel à destroy
        }

        virtual void destroy()
        {
                std::for_each(children.begin(),
children.end(), deleter());
                // appelle destroy sur tous les
widgets enfants
        }
};
```

Biensûr, il faut désormais apporter de légères modifications à la classe template *widget_impl*. Tout d'abord, il faut la faire hériter de la classe *widget* et prendre cela en compte dans le constructeur de *widget_impl*.

```
template
  <
  class SizePolicy,
  class ClickAwarenessPolicy,
  class TextPolicy,
  class DrawingPolicy
  >
class widget_impl :
  public widget, /* NOUVEAU */
  public SizePolicy,
  public ClickAwarenessPolicy,
  public TextPolicy,
```

```
  public DrawingPolicy
{
// ...
  public:
  widget_impl(widget* parent /* NOUVEAU */ ,
unsigned int width = 800, unsigned int height =
600, const std::string& text = "")
  : widget(parent) /* NOUVEAU */ ,
SizePolicy(width, height), TextPolicy(text)
  {
  }
// ...
};
```

Il reste un dernier détail à régler. La fonction membre *widget::destroy* détruit les widgets enfants. Toutefois, imaginez que l'on ait des choses à détruire dans les politiques. Comment faire ? Il nous suffirait de procéder comme suit. Il faudrait définir une fonction membre destroy dans *widget_impl* qui aurait l'allure suivante.

```
template
  <
  class SizePolicy,
  class ClickAwarenessPolicy,
  class TextPolicy,
  class DrawingPolicy
  >
class widget_impl :
  public widget, /* NOUVEAU */
  public SizePolicy,
  public ClickAwarenessPolicy,
  public TextPolicy,
  public DrawingPolicy
{
// ...
  public:
  // ...
  void destroy()
  {
        // ici on fait par exemple
DrawingPolicy::destroy(); s'il y a quelque chose
à détruire
        // et autres
        // enfin, on détruit les widget fils en
délégant le travail à widget::destroy
        widget::destroy();
  }
  // ...
};
```

Nous avons désormais appliqué le *Type Erasure* pour manipuler indifféremment des widgets basés sur une classe template. Si l'on voulait pouvoir gérer par exemple des redimensionnement en cascade, il faudrait rajouter le nécessaire (une fonction virtuelle pure *resize(int, int)* par exemple) dans l'interface publique de *widget*. En rajoutant une fonction qui permet de lister les widgets enfants dans l'interface de widget, ainsi qu'un peu d'affichage dans le constructeur et le destructeur de widget_impl, le code suivant permet de montrer que notre système marche très bien.

```
int main()
{
  typedef widget_impl <
    NonResizable,
    NonClickAware,
    ReadOnlyText,
    SquareDraw
      > widget_t;

  widget_t w(NULL, 1024, 780, "The best widget of
the world");
  widget_t w2(&w, 1024, 780, "The second best
widget of the world");
  w.print_children(); // fonction qui parcourt la
liste des enfants et affiche le texte obtenu via
get_text()
  // ...
  return 0;
}
```

Le code suivant affiche une construction, celle de w ; une autre, celle de w2 ; puis cela affiche "The second best widget of the world" lors de l'appel à print_children. Et enfin, cela affiche deux destructions, celle de w2 en premier, puis celle de w. Ainsi, w2 ne se détruit pas tout seul mais laisse w s'en charger en appelant destroy sur w2.

J'ai donc ici montré l'utilité du principe de *Type Erasure* sur un exemple réel et concret. Le principe doit désormais être clair pour vous, et c'est pourquoi nous allons désormais passer à la création d'un outil qui existe déjà, mais dont vous n'aviez peut-être pas idée du fonctionnement avant : **boost::any**, une classe qui peut contenir des valeurs de type quelconque, et dont on peut changer la valeur et le type de valeur à l'exécution.

5. Réécrivons boost::any

Déjà, peut-être ne connaissez vous pas le module Boost.Any de Boost, qui définit la classe boost::any. Elle appartient depuis un bon nombre d'années à l'ensemble de bibliothèques Boost. Le problème résolu par *boost::any* est le suivant : je voudrais disposer d'un type qui puisse stocker des valeurs de (presque) n'importe quel autre type. La documentation, pour en découvrir plus, se situe ici (Lien73).

Regardons dans un premier temps comment s'utilise *boost::any*.

```
boost::any a(13); // a encapsule la valeur 13, de
type int donc
a = std::string("salut"); // a encapsule une
std::string contenant "salut"
```

Pour récupérer la valeur encapsulée par un *boost::any*, il existe une fonction (plusieurs en fait, mais du même nom), *boost::any_cast*. Exemple :

```
boost::any a(13);
int i = boost::any_cast<int>(a);
```

Si l'on utilise l'une des versions travaillant sur des références et que l'on donne un type incompatible (vers lequel on ne peut pas convertir la valeur encapsulée), alors une exception *boost::bad_any_cast* est lancée. Si l'on utilise l'une des versions travaillant sur des pointeurs et que l'on donne un type incompatible vers lequel convertir, *boost::any_cast* retourne un pointeur nul.

Voilà donc ce que nous allons ici reproduire, tout simplement, avec notre technique de *Type Erasure*.

Nous allons dans un premier temps créer une classe template qui permettra de stocker une valeur de n'importe quel type.

```
template <class T>
class value
{
  T t;

  public:
  value(const T& t_) : t(t_) { }
};

// exemple d'utilisation
value<int> v(42);
```

Le problème ici est que v ne pourra stocker que des valeurs de type int. Or comme vu précédemment, un objet de type *boost::any* peut stocker un entier puis à la ligne suivante stocker une chaîne de caractères par exemple. Il nous faut donc pouvoir stocker des *value<T>* avec T variant d'un coup sur l'autre. Commençons par écrire une classe qui aura un *value<T>* et qui essayera de pouvoir charger une valeur de type différent sur demande.

```
class any
{
        value<T> v;
```

Voilà un problème. Quel T mettre ? Comment faire varier T d'un coup sur l'autre ? La programmation générique en elle même ne suffit plus. C'est là qu'intervient le principe de *Type Erasure*. Nous allons faire hériter tous les *value<T>* (donc la classe template) d'une même classe, et stocker un pointeur vers cette classe de base dans la classe *any*. Voilà donc ce que l'on obtient.

```
class value_base
{
        public:
        virtual ~value_base() { }
};

template <class T>
class value : public value_base /* NOUVEAU */
{
        T t;

        public:
        value(const T& t_) : t(t_) { }
};

class any
{
        value_base* v;

        public:
        any() : v(0) { }

        template <class value_type>
        any(const value_type& v_) : v(new
value<value_type>(v_)) { }

        ~any() { delete v; }
};
```

```
// exemple d'utilisation
{
        any a = 4;
        a = 'c';
}
```

Lors de la première ligne de l'exemple d'utilisation, on appelle le constructeur template. Pas de problème. Que se passe-t-il toutefois lors de la deuxième ligne ? En fait, un objet temporaire de type *any* va être construit pour encapsuler 'c' avec le constructeur template . Il est évident que ce "transfert" n'est pas très sécurisé du fait que l'opérateur d'assignation généré par défaut va "partager" le pointeur au lieu d'en retourner une copie. C'est pourquoi pour faciliter de telles opérations il va nous falloir une fonction dans value<T> pour cloner, c'est à dire construire une copie mais qui ne stockera pas un objet au même endroit de la mémoire, ainsi que le nécessaire pour l'assignation. Introduisons donc nos quelques modifications.

```
class value_base
{
        public:
        virtual ~value_base() { }
        virtual value_base* clone() const = 0; /*
NOUVEAU */
};

template <class T>
class value : public value_base
{
        T t;

        public:
        value(const T& t_) : t(t_) { }
        value_base* clone() const /* NOUVEAU */
        {
                return new value(t);
        }
};

class any
{
        value_base* v;

        public:
        any() : v(0) { }

        template <class value_type>
        any(const value_type& v_) : v(new
value<value_type>(v_)) { }

        any(any const & other) : v(other.v ?
other.v->clone() : 0) {}

        any& operator=(const any& other)
        {
                if(&other != this)
                {
                        any copy(other);
                        swap(copy);
                }
        return *this;
        }

        void swap(any& other)
        {
```

```
        std::swap(v, other.v);
        }

        ~any() { delete v; }
};
```

A noter que nous utilisons l'idiome *Copy and Swap*, tel que présenté ici (Lien74).

Notre *any* réagit désormais correctement lors de copie depuis un autre any, ainsi que lors d'une assignation depuis un autre any. En effet, si vous avez testé le code présenté avant nos modifications, vous auriez vu que le problème mentionné plus haut menait à une erreur de segmentation. Désormais, tout se passe bien.

Ne reste plus maintenant qu'à implémenter le fameux *any_cast* qui permet d'essayer de récupérer la valeur contenue en explicitant le type de destination (puisque toute information de type a été "perdue", du moins publiquement). Il s'agit simplement ici de rendre cette fonction any_cast amie (plutôt que d'exposer notre *value_base* v* via une fonction publique) de sorte à ce qu'elle puisse tenter un dynamic_cast de *v* vers un value<*le type demandé*>. Nous allons donc ici n'écrire qu'une version de *any_cast*, celle qui prend un any& et renvoie le type demandé si la récupération réussit, une exception *bad_any_cast* le cas échéant.

```
class any; // pour permettre la déclaration
suivate

template <class T>
T any_cast(any& a); // pour permettre les
'friend' dans les classes qui suivent

// modification de la classe template value
template <class T>
class value : public value_base
{
        friend T any_cast<>(any& a);
        // ...
};

// modification de la classe any
class any
{
        template <class T>
        friend T any_cast(any& a);
        // ...
};

// classe bad_any_cast
class bad_any_cast : public std::exception
{
        public:
        const char* what() const throw()
        {
        return "Bad any_cast exception";
        }
};

// fonction template any_cast, version
travaillant sur des références non constantes
template <class T>
T any_cast(any& a)
{
        value<T>* v =
dynamic_cast<value<T>*>(a.v);
        if(v == 0)
        {
                throw bad_any_cast();
        }
        else
        {
                return v->t;
        }
}
```

Pour terminer, voici un code qui utilise tout ce que nous avons créé :

```
int main()
{
        any a = 42;
        any b = 'c';
        std::cout << "[1] a=" << any_cast<int>(a)
<< " b='" << any_cast<char>(b) << "'" <<
std::endl;
        a.swap(b);
        std::cout << "[2] a='" <<
any_cast<char>(a) << "' b=" << any_cast<int>(b)
<< std::endl;
        try
        {
                std::string s =
any_cast<std::string>(b);
        }
        catch(const std::exception& e)
        {
                std::cout << "[3] " << e.what()
<< std::endl;
        }
        any c(a);
        std::cout << "[4] c='" <<
any_cast<char>(c) << "'" << std::endl;
        return 0;
}
/*
alp@mestan:~/cpp$ g++ -o te3 type_erasure3.cpp
alp@mestan:~/cpp$ ./te3
[1] a=42 b='c'
[2] a='c' b=42
[3] Bad any_cast exception
[4] c='c'
*/
```

Les 3 autres versions (références constantes, pointeurs non constants, pointeurs constants) sont laissées comme exercice pour le lecteur.

Nous sommes donc parvenus à une classe *any* faite maison accompagnée de la fonction template *any_cast* en appliquant simplement le principe de *Type Erasure*. Normalement, cela n'a pas été bien difficile, car une fois le principe connu, on sait qu'il nous suffit d'exposer l'interface minimale dans la classe de base de notre classe template afin de pouvoir parvenir à nos fins ensuite. Nous allons maintenant conclure quand aux limites de ce principe et à ce qui a été décrit ici.

6. Limites et conclusion

Il n'y a qu'une vraie limite à ce type d'approche, qui est une fausse limite : on ne peut pas récupérer l'information que l'on a perdu sur le type précis de départ, comme

value<T> pour notre classe *any*, que l'on se retrouve à traiter comme un *value_base*. Dans le cas des widgets, aucun moyen de savoir quels sont les types précis des widgets fils d'un certain widget à partir du moment où l'on les ajoute en tant que widget* dans la liste. Mais, bien évidemment, là est tout l'intérêt du polymorphisme de substitution ! Si l'on a créé cet héritage, c'est que que justement, quelque part, on voulait uniformiser un ensemble de valeurs et les traiter indifféremment, qu'il s'agisse de les stocker ensemble dans une collection comme pour les widgets, ou bien d'avoir une seule valeur mais qui peut prendre plusieurs valeurs de types concrets générés par une même classe template, pendant l'exécution.

Si vous voulez vous documenter un peu plus sur le principe de *Type Erasure* (sachez toutefois que cet article a couvert plus que l'essentiel sur le sujet), voici deux liens qui vous seront utiles.

- Documentation du monde Any de Boost 1.39 (Lien75)
- On the Tension Between Object-Oriented and Generic Programming in C++, par Thomas Becker (Lien76)
- An Efficient Variant Type, par Christopher Diggins (Lien77)

Retrouvez l'article d'Alp Mestan en ligne : Lien78

Coder proprement

Nettoyez votre code et devenez plus performant !

Si un code sale peut fonctionner, il peut également compromettre la pérennité d'une entreprise de développement de logiciels. Chaque année, du temps et des ressources sont gaspillés à cause d'un code mal écrit. Toutefois, ce n'est pas une fatalité.

Grâce à cet ouvrage, vous apprendrez à rédiger du bon code, ainsi qu'à le nettoyer "à la volée", et vous obtiendrez des applications plus robustes, plus évolutives et donc plus durables. Concret et pédagogique, ce manuel se base sur les bonnes pratiques d'une équipe de développeurs aguerris réunie autour de Robert C. Martin, expert logiciel reconnu. Il vous inculquera les valeurs d'un artisan du logiciel et fera de vous un meilleur programmeur.

Coder proprement est décomposé en trois parties. La première décrit les principes, les pratiques et les motifs employés dans l'écriture d'un code propre. La deuxième est constituée de plusieurs études de cas à la complexité croissante. Chacune d'elles est un exercice de nettoyage : vous partirez d'un exemple de code présentant certains problèmes, et l'auteur vous expliquera comment en obtenir une version saine et performante. La troisième partie, enfin, sera votre récompense. Son unique chapitre contient une liste d'indicateurs éprouvés par l'auteur qui vous seront précieux pour repérer efficacement les défauts de votre code.

Après avoir lu ce livre, vous saurez

- faire la différence entre du bon et du mauvais code ;
- écrire du bon code et transformer le mauvais code en bon code ;
- choisir des noms, des fonctions, des objets et des classes appropriés ;
- mettre en forme le code pour une lisibilité maximale ;
- implémenter le traitement des erreurs sans perturber la logique du code ;
- mener des tests unitaires et pratiquer le développement piloté par les tests.

Véritable manuel du savoir-faire en développement agile, cet ouvrage est un outil indispensable à tout développeur, ingénieur logiciel, chef de projet, responsable d'équipe ou analyste des systèmes dont l'objectif est de produire un meilleur code.

Critique du livre par Baptiste Wicht

Un livre tout simplement indispensable pour apprendre à coder proprement. En plus de nous apprendre à coder proprement, il nous apprend également pourquoi il faut le faire, ce qui est indispensable pour motiver quelqu'un à programmer proprement. Ce livre m'a beaucoup appris sur la bonne manière de coder. Tous les éléments théoriques du livre sont mis en pratique via des exemples concrets de code. En plus de cela, un chapitre entier est consacré au remaniement d'une classe et un autre au remaniement d'un petit programme.

Le premier chapitre tente de spécifier ce qu'est un code propre. Vous y trouverez l'avis de nombreuses personnes (Bjarne Stroustrup ou Grady Booch) sur ce qu'est un code propre. Ensuite, le chapitre suivant traite de l'importance d'utiliser des noms significatifs pour les variables, classes et méthodes. Le chapitre 3 est à mon avis l'un des chapitres les plus importants du livre. En effet, il traite des fonctions et de l'importance de les faire le plus court possible, de leur faire faire une seule chose et à un seul niveau d'abstraction, ce qui augmente clairement la lisibilité des méthodes.

On passe ensuite aux commentaires. On y apprend notamment quels sont les commentaires vraiment utiles et quels sont ceux qui ne font que surcharger le code. On passe ensuite à la mise en forme de votre code, c'est à dire l'indentation, la taille des fichiers, la largeur d'une ligne, ...

Après cela, on en vient aux objets et structures de données, à la gestion des erreurs et à la gestion des limites. Puis on passe à un autre chapitre très important, les tests unitaires. Ce chapitre m'a fait redécouvrir les tests unitaires et leur utilité.

On apprend ensuite à concevoir de bonnes classes. Avant de passer au niveau supérieur, celui du système. Avec lequel, on apprendra à concevoir également les préoccupations transversales. Puis on passe à un ensemble de règles permettant de faciliter l'émergence d'un code propre.

Le chapitre suivant est consacré à la concurrence et à la conception d'un code concurrent propre et fonctionnel. Ce chapitre sera poursuivi en annexe par un deuxième chapitre sur la concurrence.

Puis, on passe à 3 chapitres pratiques. Le premier traitant des améliorations successives d'un programme fonctionnel, mais pas propre. Le chapitre suivant est entièrement consacré au refactoring d'une classe de JUnit et enfin le troisième chapitre va traiter du remaniement de la classe SerialDate de JCommon.

Le dernier chapitre fournit une liste d'indicateurs et d'heuristiques permettant d'avoir rapidement à portée de mains une liste de bonnes pratiques pour coder.

En conclusion, ce livre vous permettra, soit de commencer à bien coder, soit de redécouvrir la façon de coder, suivant que vous soyez débutant ou expérimenté. L'auteur pouvant

parfois être très radical dans ces dires, il est possible que certaines de ses recommandations ne vous plaisent pas du tout ou vous semblent trop extrémistes. Néanmoins, ces recommandations viennent, comme l'auteur l'indique, d'une école de pensée. Il est donc possible que d'autres systèmes de code propre vous satisfassent plus que celui-là, mais pour ma part, j'ai trouvé ce système de code propre des plus intéressants.

Retrouvez ces critiques de livre sur la page livres Conception : Lien79

Une introduction à CUDA

Une introduction à CUDA et au calcul sur GPU, comparativement avec les CPU.
Avant la fin, vous pourrez écrire vos premiers kernels.
Cette introduction se base sur CUDA 2.1 et 2.2.

Introduction théorique à CUDA

1. GPGPU

Les constructeurs ont décidé de créer des langages qui permettent d'exploiter les possibilités de ces processeurs graphiques. Ils n'ont pas été les seuls.

Par exemple, l'université de Stanford a créé le *BrookGPU*, le tout premier langage, un dérivé du C, qui permet d'utiliser les API *DirectX* et *OpenGL*, ainsi que GLSL ou CG. L'avantage de ces solutions est qu'elles sont utilisables sur tous les GPU qui supportent DirectX et/ou OpenGL, c'est-à-dire la plus grande majorité d'entre eux, et la totalité ces dernières années. Cependant, cette universalité se traduit aussi par un manque de performances par rapport à d'autres librairies plus proches du matériel.

Ainsi, ATI a développé Close to Metal, une librairie très bas niveau. Cette librairie sera suppléée par Stream, mais cette dernière est plus spécifiquement dirigée vers les processeurs *FireStream*, prévus pour le calcul.

Ensuite vient NVIDIA, avec CUDA, une technologie disponible sur toutes les cartes graphiques grand public depuis la série des GeForce 8000 et sur tous les supercalculateurs Tesla.

2. CUDA

Ou *C*ompute *U*nified *D*evice *A*rchitecture.

C'est la réponse de NVIDIA aux demandes sans cesse croissantes de puissance de calcul. Cette librairie, dévoilée en 2007, permet d'employer la puissance de calcul des GPU. Elle n'est que la partie logicielle du tout : il faut encore une carte graphique compatible.

CUDA supporte plusieurs langages : le C, le C++ et le Fortran. Vous pouvez donc utiliser conjointement ces trois langages dans vos fonctions et vos kernels.

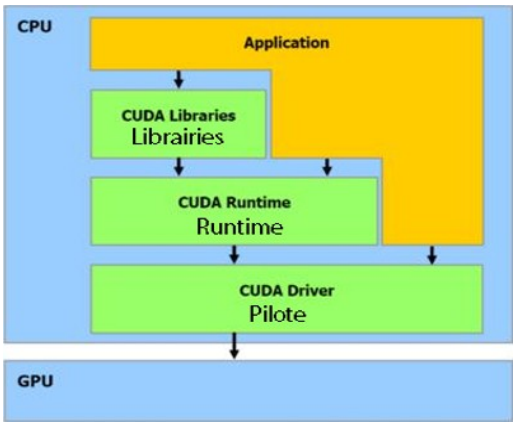
Il existe déjà quelques wrapper pour CUDA : PyCUDA (Lien80), destiné à Python, ainsi que JCublas (Lien81), JCufft (Lien82) et JCudpp (Lien83), sans oublier CUDA lui-même, avec jCUDA (Lien84) pour Java, sans oublier CuBLAS.Net (Lien85), un wrapper de CuBLAS pour le CLR .Net.

CUDA est constitué d'un pilote, déjà intégré aux ForceWare les plus récents ; d'un runtime ; et de quelques librairies. CUDA est aussi un langage, dérivé du C (mais n'apportant que peu de modifications : 9 nouveaux mots clés, 24 nouveaux types et 62 nouvelles fonctions). Ces extensions nécessitent leur compilateur, lui aussi fourni.

CUDA est prévu pour s'exécuter sur un GPU, mais il est aussi disponible sur CPU, en émulation. Les performances sont alors bien moindres, mais cela peut être utile pour tester ses applications sans GPU compatible.

L'API CUDA est de haut niveau : vous ne vous occupez donc pas du GPU directement. CUDA en est une couche d'abstraction.

Voici, graphiquement représentées, toutes les composantes de CUDA et de son utilisation.



2.1. Pilote

- **Rôle** : transmettre les calculs de l'application au GPU ;
- **Distribution** avec les ForceWare 178.08 et plus récents ;
- **Inconvénient** : pas d'automatismes.

2.2. Runtime

- **Rôle** : interface entre le GPU et l'application, en fournissant quelques automatismes ;
- **Distribution** : en même temps que le pilote ;
- **Inconvénient** : impossibilité d'optimiser à partir d'un certain point.

2.3. Bibliothèques

Pour le moment, CUDA est livré avec CuBLAS et CuFFt, respectivement les implémentations de BLAS (une bibliothèque d'algèbre) et de la transformation rapide de Fourier (utilisée en analyse de Fourier et en traitement du signal). La dernière n'est pas inspirée d'une bibliothèque

préexistante.

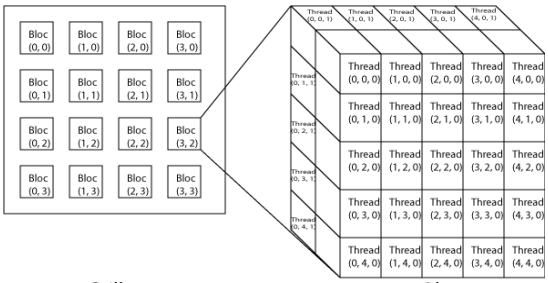
Ces implémentations reprennent le fonctionnement des bibliothèques originelles (CuBLAS), ou bien des algorithmes les plus performants (CuFFT) et les optimisent au maximum pour CUDA.

3. Un peu de vocabulaire

Nous allons continuer cette introduction avec un peu de vocabulaire inhérent à la programmation avec CUDA.

L'**hôte** est le CPU, c'est lui qui demande au **périphérique** (le GPU) d'effectuer les calculs.

Un **kernel** est une portion parallèle de code à exécuter sur le périphérique. Chacune de ses instances s'appelle un **thread**.



Une **grille** est constituée de blocs. Chaque **bloc** est constitué de threads.

Un bloc est un élément des calculs, dissociable d'autres blocs : les blocs ne doivent donc pas être exécutés dans un certain ordre : parallèlement, consécutivement ou toute autre combinaison est possible. C'est pourquoi les threads ne peuvent communiquer qu'avec des threads du même bloc.

Un **warp** est un ensemble de 32 threads, envoyés ensemble à l'exécution, et exécutés *simultanément*. Quel que soit le GPU utilisé, quel que soit la quantité de données à traiter, dans n'importe quel cas, un warp sera exécuté sur deux cycles. On peut être sûr et certain qu'il le seront. Ceci pourra vous aider lors de la conception de vos algorithmes. Par exemple, Mark Harris, chercheur pour NVIDIA dans le rendu graphique en temps réel, fondateur du site GPGPU (Lien86), utilise cette donnée pour dérouler ses boucles.

Un petit parallèle avec le matériel. Un thread est exécuté par un processeur : posons donc l'égalité entre le thread et le processeur. Ainsi, le bloc est le multiprocesseur, tandis que la grille représente l'entièreté de la carte.

Le **calcul hétérogène** est l'utilisation des deux types de processeur disponibles sur nos ordinateurs : les CPU et les GPU. Il s'agit donc d'utiliser le bon type de processeur pour la bonne tâche.

Vous voici prêt pour partir à l'attaque !

4. CPU et GPU

4.1. Survol de quelques différences

La puissance de nos GPU n'a de cesse d'augmenter depuis quelques années. À un point qu'il est désormais possible de les utiliser pour réaliser des calculs autres que pour des jeux. En effet, parmi les CPU, un Intel Pentium 4 cadencé à 3 GHz fournit 4,8 GFlops, un Intel Core 2 Duo E6750 (2,66 GHz), 14,2 GFlops ; chez les GPU, on change de catégorie : la GeForce 9800 GTX, 420 GFlops, pour 675 MHz seulement.

Cependant, ces différences énormes s'expliquent très facilement, explications dans ce tableau.

	CPU (hors SIMD)	GPU
Nombre de tâches	Une seule et unique	Le plus grand nombre
Variété des tâches	Toutes possibles	Restreinte
Subdivision de la tâche	Aucune : tout en un coup	Maximale, pour mieux la répartir sur les différentes unités de calcul

Il ne faut pas oublier de préciser que les GPU préfèrent travailler avec des vecteurs. Dans le cas contraire, les gains sont réellement minimes.

Les deux types de processeur travaillent de façon radicalement différente. L'emploi de GPU à la place de CPU ne se fait donc pas en un tour de main : il faut repenser le calcul pour l'adapter au type de processeurs désiré. Si l'on ne change pas sa manière de penser, autant continuer de produire son électricité à la pomme de terre, qui permet quand même de produire assez pour éclairer quelques centimètres ; tandis que la centrale électrique permet d'éclairer des villes entières.

Pour le grand public, les prix se tiennent : un E6750 coûte, actuellement, 140 € ; une 9800 GTX, 150 €. Leurs éditions professionnelles sont légèrement différentes : 1 500 $ pour un NVIDIA Tesla S870 plafonnant à 2 Tflops, contre 200 000 $ pour un IBM BlueGene de même puissance. Ici, on remarque bien l'un des grands avantages du GPGPU.

On peut considérer des racks de cartes Tesla comme des supercalculateurs. En effet, ce sont eux qui calculent. Cependant, un ou plusieurs CPU les orchestrent, en plus de leur donner la masse de travail.

Aussi, les GPU ont été, à la base, destinés à et spécialisés pour des calculs intensifs. Ceci leur permet de réserver plus de transistors au traitement des données, au lieu de les utiliser pour le cache ou bien pour la gestion des flux d'entrée ou de sortie.

Ainsi, un GPU doit être constitué de beaucoup de processeurs pour ces calculs : un GPU comporte au strict minimum 32 processeurs (240 pour le T10, 128 en moyenne), et ce, depuis plus qu'un temps certain. Ces processeurs sont les équivalents des coeurs de nos CPU,

qui en comportent, en moyenne, 2 depuis quelques années, et, dans les années à venir, 80. Nous sommes donc bien loin des GPU !

4.2. Précision des calculs

Les GPU actuels, avec CUDA, n'ont qu'une précision FP32, sur 32 bits. Il faut se tourner vers les solutions d'ATI/AMD pour une précision double sur 64 bits, ou bien vers des GPU plus chers, comme les Tesla ou les Quadro, ou bien récents, comme tous les GPU basés sur le GT200 (GeForce GTX260 à GTX295).

Tous les processeurs ne fonctionnent pas à la même précision : sur les premières GeForce compatibles CUDA, tous sont FP32. Sur un T10, 8 unités sont FP32, et une seule FP64. Chez AMD, pour 8 unités FP64, il y a 4 unités FP32.

Le peu d'unités dédiées au calcul à double précision sur les Tesla et autres explique leur faible puissance à ce niveau de précision, en comparaison de la simple précision ou bien des solutions d'AMD. Ainsi, pour du calcul en haute précision, les solutions NVIDIA tous publics ne sont pas encore au point (AMD ne propose plus de GPGPU pour la même gamme).

Actuellement, tous les processeurs supportent la double précision sur 64 bits.

Plus précisément, NVIDIA met à disposition la liste des écarts avec les standards, ainsi que ses limitations.

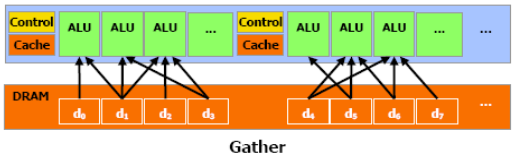
1. Les additions et soustractions sont souvent associées en une seule instruction ;
2. La division et la racine carrée sont implémentées par la réciproque, non conformément aux standards ;
3. Pour la multiplication et l'addition, il n'est possible que d'arrondir vers le nombre pair le plus proche ;
4. Il n'y a pas de possibilité d'arrondi configurable dynamiquement ;
5. Il n'y a pas de signalisation de NaN (*N*ot *a N*umber) ;
6. Il n'y a pas de mécanisme de détection d'exception, qui sera masquée selon les standards ;
7. Les opérandes de source dénormalisée tendent vers 0 ;
8. Le résultat d'une opération avec NaN est un NaN canonique de la forme 0x7fffffff ;
9. En accord avec les standards, si un NaN est passé à min() ou à max(), l'autre sera retourné.

4.3. GPU

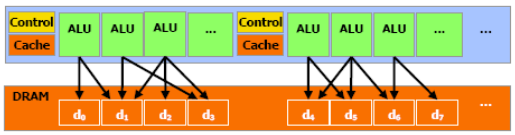
4.3.1. Mémoires

4.3.1.1. Mémoire globale

CUDA est capable de lire et d'écrire sur la mémoire embarquée dans la carte graphique. Ces opérations portent, respectivement, les doux noms de *gathering* et de *scattering*.



Gather



Scatter

La mémoire globale est la mémoire utilisable de n'importe quel endroit de CUDA, avec les mêmes performances à la clé : cette mémoire n'est pas cachée, et il faut attendre 400 à 600 cycles avant d'y accéder. Ce qui laisse un multiprocesseur inactif pendant ce temps.

Pourquoi une telle latence ?
La mémoire globale est, en général (dans tous les cas, jusqu'à présent), de la DRAM.
Cette mémoire est très bon marché : 1,50$ en septembre 2008, pour les intégrateurs ! Ceci lui permet d'être utilisée comme mémoire principale de nos ordinateurs.
De plus, elle se révèle compacte : on en fait tenir des Go sans problème sur des cartes !
Pourtant, cette mémoire a un problème, et il s'agit de la latence. Elle monte sans problème jusqu'à 30 ns, ce qui représente quand même déjà 30 cycles ! Et sans compter les bus entre le multiprocesseur et la mémoire.
Finalement, cette mémoire n'est pas cachée.

4.3.1.2. Mémoire locale

Cette mémoire est, à l'*instar* de la mémoire globale, non cachée, et avec une latence très élevée.

Cette mémoire n'est utilisée que pour certaines variables, qui y sont placées automatiquement. En effet, certains tableaux, normalement placés dans les registres, sont trop grands : il leur faut donc un espace plus grand, qu'offre la mémoire locale.

4.3.1.3. Mémoire constante

La mémoire constante est cachée : la lecture depuis cette mémoire ne coûte qu'un cycle. Pour tous les threads d'un demi-warp, la lecture depuis la mémoire constante est aussi rapide que depuis un registre, aussi longtemps que tous les threads lisent le même emplacement mémoire. Le coût de lecture augmente linéairement avec le nombre d'adresses différentes demandées par les threads. Il est recommandé que tous les threads d'un warp utilisent la même adresse, et non seulement ceux de demi-warps, vu que les périphériques futurs le requerront pour un fonctionnement optimal.

Chaque multiprocesseur dispose d'une mémoire réservée aux constantes, d'une taille de 8 ko, dans le cas des GeForce 8800.

4.3.1.4. Mémoire des textures

Cet espace mémoire est caché, le coût de la lecture est donc très faible.

Cette mémoire est optimisée pour un espace à deux dimensions, ainsi, les threads d'un même warp qui lisent à des adresses proches auront des performances optimales.

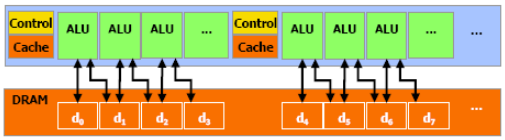
Aussi, elle est prévue pour des demandes de flux avec une latence constante.

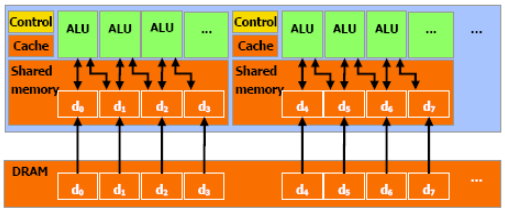
La lecture des mémoires du périphérique par le mécanisme des textures peut être une alternative avantageuse à la lecture depuis les mémoires globale ou constante.

Les textures seront approfondies plus tard, mais voici un avant-goût.

Les textures permettent vraiment de simplifier le traitement d'images : elles permettent la mise en oeuvre de filtrages bilinéaires et trilinéaires très facilement, et l'accès aléatoire aisé aux pixels.

4.3.1.5. Mémoire partagée





Cette mémoire est présente sur le chipset, ce qui lui permet d'être assez rapide, plus que la mémoire locale.
En fait, pour tous les threads d'un warp, accéder à cette mémoire est aussi rapide que d'accéder à un registre, tant qu'il n'y a pas de conflit entre les threads.

Pour permettre une bande-passante assez élevée, la mémoire partagée est divisée en modules de mémoire, les banques, qui peuvent être accédée simultanément. Ainsi, *n* lectures ou écritures qui tombent dans des banques différentes peuvent être exécutées simultanément dans un warp, ce qui permet d'augmenter sensiblement la bande passante, qui devient *n* fois plus élevée que celle d'un module.

Cependant, si deux demandes tombent dans la même banque, il y a un conflit de banques, et l'accès doit être sérialisé. Le matériel divise ces requêtes problématiques en autant de requêtes que nécessaire pour qu'aucun problème n'ait lieu, ce qui diminue la bande passante d'un facteur équivalent au nombre de requêtes total à effectuer.

Pour des performances maximales, il est donc très important de comprendre comment les adresses mémoires sont reliées aux banques, pour pouvoir prévoir les requêtes, et, ainsi, minimiser les conflits.

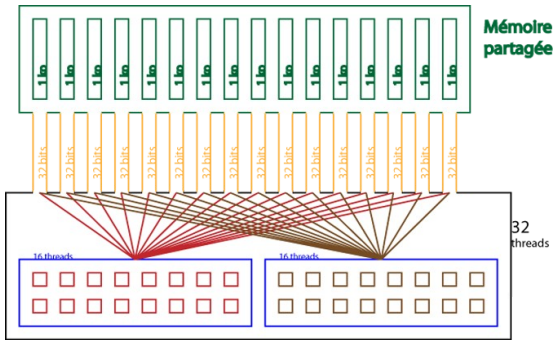
Dans le cas d'un espace en mémoire partagée, les banques sont organisées pour que des mots successifs de 32 bits soient assignés à des banques successives. Chaque banque a une bande passante de 32 bits tous les deux cycles d'horloge.

Pour le moment, un warp a une taille de 32 threads, et il y a 16 banques.

Une requête en mémoire partagée pour un warp est divisée en deux : une partie pour le premier demi-warp, une autre, pour l'autre moitié. Ce qui a pour conséquence qu'il ne peut y avoir de conflit entre chaque demi-warp. Les conflits seront détaillés plus tard.

Actuellement, la mémoire partagée atteint un total de 16 ko, 1 ko pour chaque banque.

Pour résumer ceci, voici un schéma qui reprend l'essentiel des caractéristiques présentées ici.



4.3.1.6. Registres

Généralement, l'accès à un registre ne prend pas un seul cycle supplémentaire par instruction, mais des retards peuvent apparaître, suite aux dépendances de lecture après écriture, et des conflits qui peuvent se produire.

Les retards introduits pas les dépendances peuvent être ignorés, dès qu'il y a au moins 192 threads actifs par multiprocesseur, qui permettent de les cacher.

Le compilateur et l'organisateur des threads organisent les instructions pour des performances optimales, qui nécessitent d'éviter les conflits avec les banques. Le meilleur moyen d'obtenir de bonnes performances est d'utiliser un multiple de 64 comme nombre de threads par bloc. Une application n'a strictement aucun moyen de contrôler ces conflits.

Chaque multiprocesseur dispose de 8192 registres.

4.3.1.7. Mémoire système

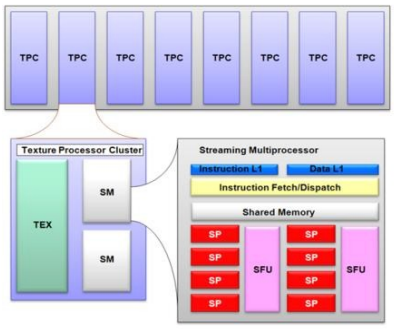
Depuis les GT200 (GeForce GTX 260 à 295), il est désormais possible d'utiliser la mémoire principale du système, alias RAM, grâce à CUDA 2.2.

Les appels à cette mémoire ne peuvent être fréquents : ils sont encore plus lents que les appels à la mémoire locale (700 à 800 cycles de latence !). Mais la RAM est disponible, de nos jours, en quantités plus grandes que celle disponible sur nos GPU.

4.3.2. Shaders

Les calculs demandés à CUDA sont, pour le moment, effectués sur les unités de shaders, les processeurs les plus

rapides sur les GPU. Par exemple, les GeForce 8800 GTX ont des unités cadencées à 1,2 GHz.



Chaque unité de traitement des shaders est, comme montré ci-dessus, constituée de **T**exture **P**rocessor **C**lusters (TPC).

Chacun de ces clusters est fait d'une unité de traitement des textures (TEX) et de deux unités de traitement des flux (SM, **S**treaming **M**ultiprocessor).

Vous n'avez pas vraiment besoin d'en savoir beaucoup plus pour pouvoir aborder CUDA. Cependant, si vous en voulez encore, faites-vous plaisir avec la section suivante !

4.3.2.1. Plus de précisions

Chacun de ces deux processeurs contient une interface qui code et décode les instructions, et qui les lance. Derrière l'interface, plusieurs unités exécutent les instructions. Les calculateurs fonctionnent deux fois plus vite que l'interface !

Ces calculateurs sont 8 unités de calcul (SP) et 2 unités superfonctionnelles (SFU).

À chaque cycle, l'interface choisit un warp prêt à être exécuté.

Pour exécuter toutes les instructions des 32 threads, il faudra 4 cycles. Cependant, vu de l'interface, cela prendra 2 cycles.

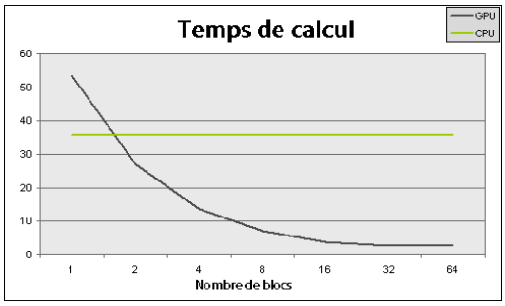
Pour éviter que l'interface reste inactive pendant un cycle, l'idéal est d'alterner les types de warps : un premier pour les SP, un second pour les SFU.

4.3.2.2. Limites

Un SM étant composé de 8 SP, on sera donc limité à l'exécution de 8 blocs en simultané. De plus, l'exécution est limitée à 65536 blocs et 512 threads par bloc au total.

Vous n'avez pas encore eu un aperçu du temps consacré au calcul en fonction des différents paramètres.

En faisant varier le nombre de blocs de calcul sur un même problème, voici les résultats que l'on peut obtenir, avec de simples opérations d'entrée/sortie dans une table. Un bloc correspond à un thread sur un CPU, que l'on peut affecter à un coeur.



Le processeur utilisé ici est un simple coeur, ses performances en fonction du nombre de threads restent donc stables. S'il s'agissait d'un quad-core, le minimum serait situé à 4 threads.

La carte graphique, une GeForce 8800 GTX, possède 16 processeurs, qui ne donnent leur pleine puissance qu'à deux blocs chacun. NVIDIA recommande toutefois d'utiliser au moins une centaine de blocs, afin de pouvoir utiliser la puissance de chipsets plus récents à venir.

4.4. CPU

Il exécute uniquement les instructions dans l'ordre assigné, sans parallélisation (sauf architectures multi-cores et multi-CPU, qui nécessitent quand même une action à la conception).

Les instructions sont aussi écrites en mémoire, pour exécution. Cependant, les données avec lesquelles il faudra travailler sont souvent dans la même mémoire !

D'habitude, il ne travaille pas directement en mémoire : les données sont copiées dans des registres puis manipulées et enfin stockées en mémoire.

À l'origine, CPU et mémoire partageaient les mêmes fréquences. Mais le premier a accéléré, et la seconde ne l'a pas rattrapé : au point que, si les processeurs actuels lisaient directement dans la mémoire, ils ne seraient utilisés que 10% du temps.

4.4.1. Mémoire cache

C'est pour cela que des caches ont été installés : il s'agit de petites quantités de mémoire, mais très rapide, qui se place entre le CPU et la mémoire centrale. Ils ne sont utilisés que pour les instructions fréquemment utilisées et les données. Il en existe deux niveaux : L1 et L2, exceptionnellement un troisième, L3, sur les processeurs les plus chers (réservés généralement aux serveurs).

Cependant, ces mémoires très rapides ne sont pas présentes en grande quantités sur nos CPU, vu leur prix : en moyenne, le mégaoctet de cache coûte 100 fois plus cher que le mégaoctet de RAM ! Le cache fonctionne aussi 10 fois plus vite que la RAM, avec un temps d'accès de 5 à 10 fois inférieur.

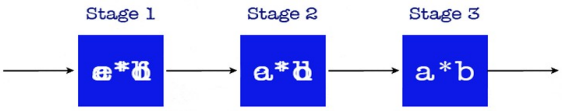
Les caches sont utilisés de manière transparente par le matériel. Ils se font les mirroirs des données en mémoire. Ils transportent les données où elles sont nécessaires quand cela est demandé. Ces données ne sont remplacées que quand des données plus urgentes arrivent.

Si les données demandées par le CPU sont disponibles sur le cache, celui-ci les lui envoie, le CPU ne doit pas attendre. Par contre, si elles ne le sont pas, la demande est effectuée en aval, sur des mémoires plus lentes, et le CPU doit attendre.

4.4.2. Pipelines d'instructions

Supposons qu'un CPU prenne 3 cycles pour une multiplication de paire. Combien de temps prendra-t-il pour multiplier n paires ? Nous pourrions dire 3 n cycles. Il est possible de réduire ce nombre.

La multiplication aura lieu dans une ligne de production. Nous pouvons avoir plus d'une paire de nombres en calcul en même temps. Dans ce cas, les multiplications prendront n + 2 cycles.



Notre but, pour atteindre cette vitesse, est de garder le pipeline rempli.

Dans une architecture avec pipelines, il est préférable d'avoir le moins possible de branches.

Moins de branches
```
do i=start,end
    a(i) = b * c(i)
end do
```

Moins de multiplications
```
do i=start,end
    if( c(i) == 0 )
        a(i) = 0
    else if( c(i) == 1 )
        a(i) = b
    else
        a(i) = b * c(i)
    end if
end do
```

4.4.3. Exécution superscalaire

Les CPU modernes ont plusieurs unités de calcul, qui peuvent effectuer un nombre limité d'instructions en parallèle.

Le matériel examine les instructions pour repérer des opportunités d'optimisation.

Le pipeline
```
i = i + 1;
j = j + 1;
a = b * c;
```

Les branches limitent ces opportunités, et les unités d'exécution sont laissées en attente pendant l'évaluation des conditions.

Les CPU essayent toutes sortes d'autres astuces, comme la prédiction de branches, l'exécution spéculative ou autres, dont le compilateur et le CPU s'occupent.

Introduction plus pratique à CUDA

5. Les mains dans le cambouis

```
Il n'y a pas de langage informatique dans lequel
vous ne puissiez écrire de mauvais programme.
```

```
Si vous ne savez pas ce que votre programme est
censé faire, vous feriez bien de ne pas commencer
à l'écrire.
```

(Extraits de Les lois de Murphy (Lien87)).

5.1. Les kernels

Très simplement, un kernel est une fonction exécutée sur le GPU.

Il en existe différent types, qualifiés de :
1. __global__
2. __device__
3. __host__

Le premier correspond à un kernel exécuté sur le GPU mais appelé par le CPU ; le deuxième, à un kernel exécuté et appelé par le GPU ; le troisième, à une fonction exécutée et appelée par le CPU. Ce dernier n'est pas obligatoire : c'est le mode de fonctionnement par défaut.

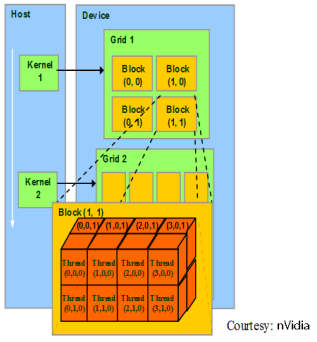
Un kernel ne s'appelle pas de la même manière qu'une fonction. Voici un appel de fonction.

Appel de fonction
```
fonction(parametre, parametre);
```

Mais avant de vous parler de l'appel d'un kernel, il faut que vous compreniez bien le mode de fonctionnement d'un GPU.

Une grille représente la totalité de la tâche à effectuer. Chaque grille peut être divisée en un ou plusieurs blocs, chacun exécutant plusieurs threads.

Un thread sur un GPU n'a pas le même sens qu'un thread sur le CPU.
Sur un GPU, il s'agit de la plus petite subdivision de la tâche à effectuer.



Un appel de kernel se fait en spécifiant 2 paramètres entre triples chevrons précédant les paramètres passés au kernel.

```
kernel <<< nBlocs, threadsParBloc >>>
(arguments);
```

nBlocs est le nombre de subdivisions appliquées à la grille à calculer et est de type dim3 (le cast à partir d'un entier N initialise le *dim3* à *{N, 0, 0}*).
threadsParBloc indique le nombre de threads à exécuter simultanément pour chaque bloc. Ici encore, cette valeur est de type dim3.

Les valeurs à appliquer dépendent simultanément du problème à résoudre (choix des dimensions des blocs) et du matériel utilisé (nombre de threads par bloc). Choisir un nombre de threads supérieurs à la quantité nativement supportée entraînera une perte de performances. Cette notation permet ainsi d'adapter dynamiquement le programme aux matériels passés, présents et futurs.

Chaque kernel dispose de variables implicites en lecture seule (toutes de type dim3).

1. **blockIdx** : index du bloc dans la grille,
2. **threadIdx** : index du thread dans le bloc,
3. **blockDim** : nombre de threads par bloc (valeur de *threadsParBloc* du paramétrage du kernel).

La grille est ici considérée comme un seul et unique bloc à une seule dimension.

```
__global__ void vecAdd(float * A, float * B,
float * C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    // utilisation du kernel
    vecAdd<<<1, N>>>(A, B, C);
        //     |-> vecteurs additionnés une seule
fois
        //        |-> nombre de composante des
vecteurs
}
```

Dans le cas où la grille est sous-divisée en N blocs (tous de 1 dimension), l'index pourrait être trouvé de la manière suivante.

```
__global__ void vecAdd(float * A, float * B,
float * C)
{
    int i = blockIdx.x * blockDim.x +
threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    // utilisation du kernel
    const int nThreadsPerBlocks  = 4;
    const int nBlocks            = (arraySize /
nThreadsPerBlocks) + ( (arraySize %
nThreadsPerBlocks) == 0 ? 0 : 1);
    vecAdd<<<nBlocks, nThreadsPerBlocks>>>(A, B,
C);
}
```

Les variables doivent être qualifiées, pour définir leur lieu de résidence : voyez la section qui y est réservée.

Les paramètres entre chevrons sont requis, car le kernel est de type __global__. S'il était d'un autre type, ils n'auraient pas dû être précisés !

5.2. Qualifieurs de kernels

5.2.1. __global__

- Exécuté sur le périphérique,
- Appelable de l'hôte.
- Pas de récursion possible,
- Pas de variables statiques,
- Pas de liste de paramètres variable.
- On ne peut demander leur adresse mémoire,
- Incompatible avec __device__,
- Ne peut rien retourner,
- À l'exécution, on doit préciser la configuration,
- Appel asynchrone (le kernel retourne avant d'avoir effectué les calculs),
- Paramètres stockés dans la mémoire partagée, limités à 256 octets,
- Dure aussi longtemps que le kernel.

5.2.2. __device__

- Exécuté sur le périphérique,
- Appelable du périphérique.
- Pas de récursion possible,
- Pas de variables statiques,
- Pas de liste de paramètres variable.
- On ne peut demander leur adresse mémoire,
- Incompatible avec __global__,
- Dure aussi longtemps que l'application.

5.2.3. __host__

- Exécuté sur l'hôte,
- Appelable de l'hôte.
- Appliqué par défaut.
- Compatible avec __device__ (dans ce cas, le kernel pourra être exécuté sur l'hôte et sur le périphérique),
- Incompatible avec __global__,
- Dure aussi longtemps que le kernel.

5.3. Configuration de l'exécution

Ceci n'est ***requis*** que pour les kernels __global__ ! Requis signifie bien que l'on ne peut s'en passer, sans quoi rien ne fonctionne (avec, à la clé, beaux plantages) !

Cette configuration doit être passée entre triples chevrons avant les paramètres.

```
//Définition du kernel
__global__ void func(float * parameter);
//Utilisation du kernel
func <<< Dg, Db, Ns, S >>> (parameter);
```

5.3.1. Dg

- **Type** : dim3 ;
- **Utilité** : spécifier la taille et la dimension de la

grille (le produit des trois composantes est le nombre de blocs lancés) ;
- **Remarque** : *z* n'est pas encore utilisé, et doit valoir 1.

5.3.2. Db

- **Type** : dim3 ;
- **Utilité** : spécifier la taille et la dimension de chaque bloc (le produit des trois composantes est le nombre de threads par bloc)

5.3.3. Ns

- **Type** : size_t ;
- **Utilité** : spécifier le nombre d'octets en mémoire partagée alloués dynamiquement par bloc en plus de la mémoire allouée statiquement ;
- **Remarque** : paramètre optionnel, valeur par défaut : 0.

5.3.4. S

- **Type** : cudaStream_t ;
- **Utilité** : spécifier le flux associé ;
- **Remarque** : paramètre optionnel, valeur par défaut : 0 ;
- La notion de flux sera abordée plus tard : sachez simplement qu'il s'agit d'une suite d'éléments de même type (comme une texture).

5.4. Qualificateurs de variables

5.4.1. __device__

Cette variable est et restera sur le périphérique. Elle ne vivra pas plus longtemps que l'application, et est accessible à tous les threads de la grille, et à l'hôte grâce au runtime.

Ce type peut se marier avec un des deux suivants.

5.4.2. __constant__

Ce type peut être utilisé avec __device__.

La variable restera en mémoire constante. Elle ne vivra pas plus longtemps que l'application, et est accessible à tous les threads de la grille, et à l'hôte par le runtime.

Ces variables ne peuvent être déclarées que de l'hôte, pas du périphérique !

5.4.3. __shared__

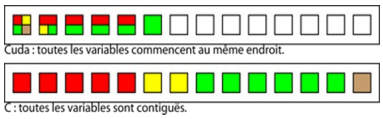
Ce type peut être utilisé avec __device__.

La variable résidera dans la mémoire partagée, et ne survivra pas au bloc. Elle ne sera accessible qu'aux threads du bloc.

Avant que les modifications soient écrites dans la variable, et visibles pour tous les autres threads, il faut appeler *__syncthreads();*. À noter que cet appel ne sert qu'à le garantir, il est *possible* que les modifications soient visibles avant.

Tableau externe
```
extern __shared__ float shared[];
```

Quand la variable est déclarée en tant que tableau externe, comme précédemment, sa taille sera fixée à l'exécution. Toutes les variables déclarées de cette manière ne sont pas contiguës : le premier bit de la première correspond au premier bit des autres, contrairement aux autres langages comme le C ou le C++.



C'est pourquoi il faut préciser l'offset de début. Pour avoir l'équivalent de ce premier code, il faut écrire le contenu du second.

Code C++
```
short array0[128];
float array1[64];
int array2[256];
```

Équivalent CUDA
```
extern __shared__ char array[];
__device__ void func()        // kernel
__device__ ou bien __global__
{
    short* array0 = (short*) array;
    float* array1 = (float*)&array0[128];
    int  * array2 = (int*)  &array1[64];
}
```

Ceci est le seul moyen d'utiliser le mot-clé *extern* sur des variables : tous les autres emplois sont interdits.

Ces variables ne peuvent pas être initialisées en même temps que leur déclaration !

Si nous avions utilisé le premier code dans CUDA, en écrivant une valeur dans le premier tableau, une partie de cette variable aurait été imputée au deuxième et au troisième tableau. Ce qui pourrait donner des résultats très aberrants.

5.4.4. Généralités

Ces paramètres ne sont pas permis sur des unions ou des structures.

En définissant une variable *__shared__* ou *__constant__*, elle sera définie statique.

5.5. Compilation

NVIDIA, dans son immense bonté, nous fournit un compilateur prévu pour CUDA. Celui-ci dispose d'une interface en ligne de commande simple et comparable à celles que nous connaissons, cl, de Visual Studio, ou gcc, l'interface de GCC. Ce compilateur, nvcc, s'occupe de toutes les étapes de la compilation.

Pour pouvoir définir les portions de code spécifiques à ce compilateur, il définit la macro __CUDACC__.

Comme dit précédemment, il s'occupe de toutes les phases de la compilation : l'assemblage, la compilation, et l'édition des liens. Vous pouvez choisir ces parties grâce à la ligne de commande.

Ce compilateur fonctionne très bien avec les Makefiles, c'est d'ailleurs cette technique qui va être ici développée, compatible avec les chaînes de compilation GNU (make) et Microsoft (nmake).

```
# Précise le compilateur précis à utiliser
ifdef ON_WINDOWS
    export compiler-bindir := "a:/program
files/microsoft visual studio 9.0/vc/bin"
endif

export NVCC := a:/cuda/bin/nvcc.exe

cpp.obj : cpp.cpp
    $(NVCC) -c cpp.cpp $(CFLAGS) -o cpp.obj

c.o : c.c
    $(NVCC) -c c.c $(CFLAGS) -o c.obj

cu.o : cu.cu
    $(NVCC) -c cu.cu $(CFLAGS) -o cu.obj

OBJECTS = cpp.obj c.obj cu.obj

all : $(OBJECTS)
    $(NVCC) $(OBJECTS) $(LDFLAGS) -o app.exe

clean :
    $(RM) $(OBJECTS)
```

Ce Makefile doit être utilisé après avoir appelé le script vsvars.bat s'il est utilisé avec Visual Studio !

Si vous utilisez un make d'origine GNU, vous pouvez utiliser ce Makefile

```
# Précise le compilateur précis à utiliser
ifdef ON_WINDOWS
    export compiler-bindir := "a:/program
files/microsoft visual studio 9.0/vc/bin"
endif

export NVCC := a:/cuda/bin/nvcc.exe

%.o : %.cpp
    $(NVCC) -c %^ $(CFLAGS) -o $@
    $(NVCC) -M %^ $(CFLAGS) > $@.dep

%.o : %.c
    $(NVCC) -c %^ $(CFLAGS) -o $@
    $(NVCC) -M %^ $(CFLAGS) > $@.dep

%.o : %.cu
    $(NVCC) -c %^ $(CFLAGS) -o $@
    $(NVCC) -M %^ $(CFLAGS) > $@.dep

include $(wildcard *.dep) /dev/null

all : $(OBJECTS)
    $(NVCC) $(OBJECTS) $(LDFLAGS) -o app.exe

clean :
    $(RM) $(OBJECTS) *.dep
```

Vous pouvez aussi utiliser la ligne de commande directement :

```
nvcc -c cu.cu -o cu.obj
nvcc cu.obj -o app.exe
```

Vous pouvez aussi décider que le code CUDA sera exécuté sur le processeur, qui émulera alors un GPU. Il suffit d'ajouter emu=1 à la ligne de commandes, comme ceci.

```
make emu=1
```

Conclusions

6. Le modèle de programmation

Pour un développeur CUDA, l'ordinateur consiste en un ou plusieurs *hôtes*, un traditionnel CPU, et un ou plusieurs *périphériques*, des non moins traditionnels GPU, des processeurs massivement parallèles.

Dans les applications modernes, certaines parties utilisent du calcul qui peut facilement devenir parallèle, sans aucun problème. Ces parties peuvent être déportées de l'hôte vers le périphérique.

6.1. Parallélisme des données

Les applications actuelles qui doivent traiter de grandes quantités de données prennent beaucoup de temps à l'exécution. Ce temps pourrait être réduit en parallélisant les opérations : des phénomènes physiques peuvent être calculés indépendamment les uns des autres, des images à analyser peuvent être découpées en portions, et un flux vidéo peut être découpé image par image.

La parallélisation des données réfère à la propriété du programme de gérer parallèlement et indépendamment ces instructions arithmétiques.

Par exemple, pour des multiplications de matrices de taille 1000 x 1000, il s'agit de 1.000.000 de multiplications, sans rapport les unes avec les autres, qui peuvent donc être parallélisées sans problème. Un GPU peut fortement améliorer les performances en exécutant toutes ces opérations simultanément.

6.2. Structure du programme

Un programme CUDA est constitué d'une partie qui s'exécute sur l'hôte et d'une partie qui s'exécute sur le périphérique.

Les phases peu ou pas parallèles sont exécutées sur l'hôte.

Les phases massivement parallèles sont exécutées sur le périphérique.

Le programme peut tenir en un seul fichier, comprenant ces deux phases et environnements. Le compilateur se charge de les séparer : le code pour l'hôte est du standard C ANSI/ISO, et est compilé par le compilateur principal du système, il sera lancé comme un simple processus. Le code pour le périphérique est aussi écrit en C ANSI/ISO,

avec quelques extensions CUDA, mais il est compilé par NVCC, et sera exécuté sur le périphérique.

Les kernels génèrent généralement beaucoup de threads pour exploiter au mieux le parallélisme des données.

Dans notre exemple de produit matriciel, il y a autant de threads que de cellules dans la matrice résultante. Chacun de ces threads prend, généralement, très peu de cycles, vu le peu de tâches qui leur sont demandées.

L'exécution commence avec le CPU, qui prépare l'appel au kernel. Le GPU prend le relais pour le kernel, qui sera, lui, massivement multithread. Quand le kernel a fini sa tâche, il renvoie le résultat au CPU, et son exécution continue.

6.3. L'exemple : la multiplication de matrices carrées

Pour commencer par clarifier la situation, voici le fonctionnement que décrira notre programme.

1. **CPU** : Initialisation des matrices M, N et P, toutes carrées ;
2. **CPU** : Remplissage des matrices d'entrée M et N ;
3. **GPU** : Calcul du produit matriciel de M et de N, dont le résultat est stocké sur P ;
4. **CPU** : Écriture de la matrice P ;
5. **CPU** : Nettoyage de la mémoire et fin de l'exécution du programme.

Nous avons vu les différents types de mémoire, mais pas la manière d'y accéder. Or, cela sera nécessaire pour permettre l'exécution du programme. Nous allons ici nous concentrer sur l'utilisation de la mémoire globale, le but étant de montrer le fonctionnement d'un programme CUDA et non d'optimiser au maximum une application.

Ces fonctions se trouvent, heureusement, dans l'API CUDA. Leurs noms sont très recherchés : cudaMalloc() et cudaFree().

Voici un bref exemple d'utilisation de ces deux fonctions, très proches des fonctions malloc() et free() du C. On considère que *Width* est le nombre de ligne et de colonne de la matrice pour laquelle on crée l'espace mémoire.

```
float *Md;
float *Nd;
float *Pd;
const int size = Width * Width * sizeof(float);

cudaMalloc( (void**) & Md, size);
cudaMalloc( (void**) & Nd, size);
cudaMalloc( (void**) & Pd, size);

cudaFree   (            Md      );
cudaFree   (            Nd      );
cudaFree   (            Pd      );
```

cudaMalloc() prend deux paramètres, pour définir la mémoire à allouer en mémoire globale.

1. L'adresse d'un pointeur vers la mémoire allouée,
2. La taille de la mémoire à allouer.

cudaFree() ne prend qu'un paramètre, pour désallouer cette mémoire en mémoire globale.

1. Un pointeur vers la mémoire à désallouer.

Une fois que le programme a alloué sa mémoire, il peut demander les données des matrices à stocker en mémoire.

Ceci s'obtient avec une fonction de copie de mémoire : cudaMemcpy(). Cette fonction requiert 4 paramètres.

1. Un pointeur vers les données source à copier,
2. La destination des données,
3. Le nombre d'octets à copier,
4. Le type de mémoire vers laquelle copier.

Concernant le quatrième paramètre, il peut prendre une de ces valeurs.

- cudaMemcpyHostToDevice : copie de l'hôte vers le périphérique
- cudaMemcpyHostToHost : copie de l'hôte vers l'hôte
- cudaMemcpyDeviceToHost : copie du périphérique vers l'hôte
- cudaMemcpyDeviceToDevice : copie du périphérique vers le périphérique

Voici les appels réalisés pour copier les matrices sur lesquelles nous allons travailler, et pour envoyer le résultat à l'endroit souhaité. *M*, *N*, *P*, *Md*, *Nd*, *Pd* et *size* gardent leurs valeurs précédentes.

```
cudaMemcpy(Md, M, size, cudaMemcpyHostToDevice);
cudaMemcpy(Nd, N, size, cudaMemcpyHostToDevice);

cudaMemcpy(P, Pd, size, cudaMemcpyDeviceToHost);
```

Tous ces transferts de mémoire sont asynchrones !

Maintenant que nous savons ce que nous pouvons faire avec la mémoire et comment le faire, nous pouvons commencer l'implémentation de notre exemple. Normalement, à ce stade de votre étude de CUDA, vous devriez pouvoir écrire correctement ce kernel (ce qui est fortement recommandé, il s'agit d'un exercice comme un autre), et voici la correction.

```
__global__ void MatrixMulKernel(float* Md, float*
Nd, float* Pd, int Width)
{
    // identifiant de thread à deux dimensions,
comme la matrice
    int tx = threadIdx.x;
    int ty = threadIdx.y;
    // Pvaleur sert au stockage de la valeur
calculée par le thread
    float Pvaleur = 0;
    for (int i = 0; i < Width; ++i)
    {
        float MdElement = Md[ty * Md.width + i];
        float NdElement = Nd[i * Nd.width + tx];
        Pvaleur        += MdElement * NdElement;
    }
    // écrit la valeur calculée dans la matrice
de résultat
    // chaque thread ne peut écrire qu'une valeur
!
    Pd[ty * Width + tx] = Pvaleur;
}
```

Et voici l'utilisation de ce kernel, que vous devriez aussi

pouvoir écrire.

```
void MatrixMulOnDevice(float * M, float * N,
float * P, int Width)
{
    //calcul de la taille des matrices
    int size = Width * Width * sizeof(float);

    //allocation des matrices et leur remplissage
    cudaMalloc(Md, size);
    cudaMemcpy(Md, M, size,
cudaMemcpyHostToDevice) ;
    cudaMalloc(Nd, size);
    cudaMemcpy(Nd, N, size,
cudaMemcpyHostToDevice);

    //allocation de la matrice de résultat
    cudaMalloc(Pd, size);

    //multiplication d'une seule matrice
    dim3 dimGrid(1, 1);
    //matrice carrée
    dim3 dimBlock(WIDTH, WIDTH);

    //produit matriciel proprement dit
    MatrixMulKernel<<<dimGrid, dimBlock>>>(Md,
Nd, Pd);

    //récupération du résultat du calcul
    cudaMemcpy(P, Pd, size,
cudaMemcpyDeviceToHost);

    //destruction des matrices, désormais
inutilisées
    cudaFree(Md);
    cudaFree(Nd);
    cudaFree(Pd);
}
```

Pour que ce code compile, vous devez inclure les fichiers cuda.h et cuda_runtime.h.

6.4. Séparer les opérations à effectuer

Comme souvent dit plus haut, le périphérique exécute beaucoup de calculs en même temps. Il est donc bien nécessaire de découper ses calculs en autant de petits morceaux, algorithmiquement identiques.

Prenons un exemple : le traitement de données sismiques. Il s'agit d'analyser une image, à trois dimensions, pixel par pixel, pour vérifier l'évolution de la situation. Nous allons nous concentrer sur l'imagerie de Kirchhoff.

Dans ce cas, une grille représente une image à analyser, avec deux dimensions, qui correspondent aux largeur et hauteur de l'image. Un bloc aura donc l'abscisse et l'ordonnée déjà fixées, la seule dimension à encore faire varier est la cote : voici donc le nombre de threads par bloc. Chaque thread aura donc ses trois coordonnées définies par ses places dans la grille et dans le bloc, il comparera ce pixel avec celui de l'image précédente.

Ceci correspond, plus ou moins, à l'algorithme utilisé sur un CPU : il est constitué de trois boucles imbriquées, chacune faisant varier une coordonnée. Basiquement, ces boucles ne sont pas parallélisées, mais cela peut être effectué sans problème, ce qui améliore sensiblement les performances.

7. Conclusions

Nous avons fini une très brève introduction aux possibilités offertes par CUDA. Nous n'avons vu que le strict nécessaire pour commencer à écrire des programmes CUDA, et les compiler.

Vous avez pu vous rendre compte de la simplicité d'écrire du code avec CUDA. Cette approche, relativement haut niveau, ne permet pas au premier abord une optimisation approfondie. Pour ce genre d'exercice, il faut se tourner vers une solution plus proche du matériel, comme l'antique CTM. Cependant, optimiser à ce niveau augmente considérablement le temps de développement (si vous avez déjà utilisé l'assembleur, vous savez ce que c'est).

7.1. Et chez AMD/ATI ?

Cette société a été la première à dégainer avec CTM, mais elle a pris du retard : son langage haut-niveau a été lancé en 2008, soit deux ans après CUDA. Donc, le langage Brook a été repris. Mais il produisait du code OpenGL. AMD l'a donc amélioré pour qu'il produise du CAL (proche de l'assembleur). Et s'est arrêté là.

Dernièrement, ils ont abandonné Brook+ (leur version de Brook) pour OpenCL, qui est en passe de devenir un standard accepté par tous (y compris NVIDIA, dont le support est disponible en beta privée pour le moment).

Il n'est pas possible d'utiliser le GPU comme processeur sans le remarquer à toutes les lignes : il faut commencer par une initialisation, qui doit préciser la version des Shaders à utiliser. Certaines commandes existent en version DirectX9 et DirectX10.

La documentation n'est pas là pour aider : aussi brouillonne que l'interface, elle est plus qu'illisible. Principal grief : l'emploi des noms pour les GPU. Nous entendons parler de Radeon HD 2900, de R600, de Pele. Qui ne sont, en fait, que les mêmes GPU ...

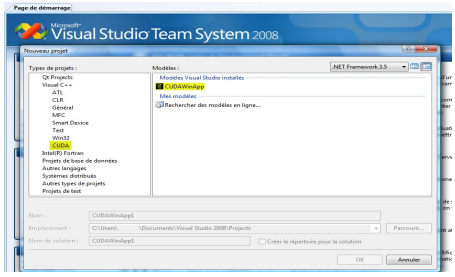
De quoi rebuter facilement du monde. Même si l'architecture peut être plus performante, et permet une double précision (FP64) depuis plus longtemps.

Dernier point : le SDK pour AMD Stream est réservé à ces GPU, réservés aux professionnels. La firme se détourne complètement des GPU grand public, contrairement à son opposant.

7.2. Intégration à Visual Studio

CUDA est prévu pour le compilateur de Visual Studio, sous Windows, alors que NVIDIA ne propose strictement aucun moyen d'intégrer le SDK dans l'IDE. C'est pourquoi Kaiyong Zhao (Lien88) a créé un modèle de projet pour cet IDE. Ce modèle est compatible avec Visual Studio 2005 (8.0) et 2008 (9.0).

Il vous suffit d'aller sur le site web du projet (Lien89) pour télécharger le modèle, puis de l'installer. Ainsi, un nouveau type de projet sera disponible à la création.



Cependant, cela nous vous apportera pas la coloration syntaxique du code. Pour ce faire, ouvrez le fichier C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\usertype.dat (chemin à modifier selon votre installation). S'il n'existe pas, créez-le. Ajoutez-y le contenu de C:\Program Files\NVIDIA Corporation\NVIDIA CUDA SDK\doc\syntax_highlighting\visual_studio_8\usertype.dat. Il s'agit de l'ensemble des mots-clés, des types, des variables prédéfinies, et des fonctions mathématiques de base de CUDA. Ce fichier convient aux versions 7.0, 7.1, 8.0 et 9.0.

Maintenant, il faut que Visual Studio utilise une coloration syntaxique, semblable à celle du C et du C++, avec les quelques mots-clés que nous venons d'ajouter, pour les fichiers CUDA.

Pour les versions 7.0 et 7.1, utilisez le fichier C:\Program Files\NVIDIA Corporation\NVIDIA CUDA SDK\doc\syntax_highlighting\visual_studio_7\install_cuda_highlighting_vs7.reg.

Pour les autres versions, *Outils > Options > Éditeur de texte > Extension de fichier*, ajoutez les extensions *cu* et *cuh*, avec l'éditeur *Visual C++*

Et voilà ! Visual Studio est maintenant fin prêt à créer de nouveaux projets CUDA et à les mettre en couleur comme il le faut !

Il existe une autre solution : le fichier de règles pour CUDA. Il s'utilise comme tous les autres fichiers de règles. Placez ce fichier dans le dossier Microsoft Visual Studio 9.0\VC\VCProjectDefaults\ : cuda.rules (Lien90). De cette manière, tous les projets disposeront des règles pour construire des fichiers .cu.

7.3. Déploiement

Vous ne devrez déployer strictement aucune DLL ou autre à côté de votre application : CUDA est intégré aux pilotes graphiques, depuis CUDA 1.1 et les pilotes de génération 177. Par contre, pour que votre application fonctionne, il faut que le client dispose d'une carte graphique compatible et de pilotes supportant CUDA.

Cependant, ceci n'est valable que si vous n'utilisez que le driver de CUDA. Si vous utilisez le runtime (présenté ici), vous devrez aussi en distribuer la DLL (cudart.dll).

Si le client ne dispose pas d'une carte graphique compatible, vous ne devrez pas livrer de DLL particulière si vous vous limitez à CUDA : si vous utilisez une autre librairie, vous devrez aussi en livrer la DLL d'émulation (suffixée par _emu).

7.4. Quels fichiers inclure ?

Ici, je n'ai parlé que du runtime : il est composé de deux fichiers, cuda_runtime.h et cuda_runtime_api.h. Tous deux sont nécessaires pour pouvoir utiliser l'ensemble des fonctionnalités du runtime.

8. Divers

8.1. Références

- ECE Illinois – cours 498 AL : Programming Massively Parallel Processors (Lien91)
- Tesla 10 & Cuda 2.0 (Lien92)
- CUDA, Supercomputing for the Masses (Lien93)
- The End of the CPU ? (Lien94)
- GPU Workshop (Lien95)
- *CUDA Textbook*, disponible sur le site du MIT (Lien96)
- *NVIDIA CUDA Programming Guide* (versions 2.1 et 2.2), distribué avec le SDK
- *NVIDIA NVCC Guide* (versions 2.1 et 2.2), distribué avec le SDK

8.2. Voir aussi

8.2.1. GPGPU

- La FAQ GPGPU (Lien97)
- BrookGPU (Stanford) (Lien98)
- Close to Metal (ATI) (Lien99)
- Stream (ATI/AMD) (Lien100)
- CUDA (NVIDIA) (Lien101)
- OpenCL (Apple/Khronos) (Lien102)
- Larrabee (Intel) (Lien103)

8.2.2. CUDA

- Site officiel (Lien104)
- Téléchargements (Lien105)
- Produits compatibles (Lien106)
- CUDA Visual Studio Wizard (Lien107)

8.2.3. Bibliothèques

- BLAS (Lien108)
- Transformée rapide de Fourier (FFT) (Lien109)

8.2.4. Matériel

- DRAM (fr) (Lien110)
- DRAM (en) (Lien111)

8.2.5. Autres

- Précision des nombres à virgule flottante (FP32 et FP64) (Lien112)
- Matrices (Lien113)
- Produit matriciel (Lien114)

Retrouvez l'article de Thibaut Cuvelier en ligne : Lien115

Liens

Lien1 : http://www.youtube.com/watch?v=ihclINC-R-U
Lien2 : http://www.youtube.com/watch?v=Jxizq-gdgrs
Lien3 : http://www.youtube.com/watch?v=fVKfXVK-bdE
Lien4 : http://blog.developpez.com/x-plode/c2108/javaone-2009/
Lien5 : http://www.developpez.net/forums/d754055/java/communaute-java/news-annonces-pendant-javaone-2009-contribuez-commentez/
Lien6 : http://www.developpez.net/forums/redirect-to/?redirect=http%3A%2F%2Fdevelopers.sun.com%2Flearning%2Fjavaoneonline%2Fj1online.jsp%3Ftrack%3Dembedded%26yr%3D2009
Lien7 : http://x-plode.developpez.com/reportages/javaone2009/
Lien8 : http://www.dil.univ-mrs.fr/~garreta/javaGen/eclipse/index.html
Lien9 : http://java.sun.com/
Lien10 : http://java.sun.com/javase/6/webnotes/install/jdk/install-windows.html
Lien11 : http://java.sun.com/javase/6/webnotes/install/jdk/install-linux.html
Lien12 : http://java.sun.com/javase/6/docs/api/
Lien13 : http://www.eclipse.org/
Lien14 : http://www.eclipseplugincentral.com/
Lien15 : http://www.eclipse-plugins.2y.net/
Lien16 : http://www.eclipsetotale.com/
Lien17 : http://henri-garreta.developpez.com/tutoriels/eclipse/installation-utilisation-eclipse-developpement-java/
Lien18 : http://android.developpez.com/
Lien19 : http://java.developpez.com/cours/?page=java-me-cat#android
Lien20 : http://www.developpez.net/forums/f1238/java/general-java/java-mobiles/android/
Lien21 : http://blog.developpez.com/recap/java/c2133/recapitulatif-java/android/
Lien22 : http://www.developpez.net/forums/d775719/java/general-java/java-mobiles/android/naissance-rubrique-android-portail-dynamique-communautaire/
Lien23 : http://www.springparlapratique.org/
Lien24 : http://java.developpez.com/livres/
Lien25 : http://aityahia.developpez.com/tutoriels/zend-framework/zend-tool/
Lien26 : https://www.zend.com/en/products/studio/downloads
Lien27 : http://g-rossolini.developpez.com/tutoriels/php/5.3/
Lien28 : http://aityahia.developpez.com/tutoriels/php/zend-studio/
Lien29 : http://mikael-morvan.developpez.com/tutoriels/javascript/dojo/introduction/
Lien30 : http://skebir.developpez.com/tutoriels/java/ant/
Lien31 : http://download.dojotoolkit.org/release-1.3.1/dojo-release-1.3.1-src.zip
Lien32 : http://mikael-morvan.developpez.com/tutoriels/javascript/dojo/build/fichier/Exemple.zip
Lien33 : http://mikael-morvan.developpez.com/tutoriels/javascript/dojo/build/
Lien34 : http://ajax.developpez.com/cours/
Lien35 : http://nicolaspied.developpez.com/ajax-premiers-pas/
Lien36 : http://siddh.developpez.com/articles/ajax/
Lien37 : http://javascript.developpez.com/faq/?page=Ajax
Lien38 : http://javascript.developpez.com/faq/?page=Ajax#ajax.xmlHttpRequest
Lien39 : http://javascript.developpez.com/faq/?page=Ajax#ajax.json.introduction
Lien40 : http://dmouronval.developpez.com/tutoriels/ajax/comprendre-requete-ajax/
Lien41 : http://www.w3.org/Style/css3-selectors-updates/WD-css3-selectors-20010126.fr.html
Lien42 : http://www.w3.org/TR/css3-selectors/
Lien43 : http://css.maxdesign.com.au/selectutorial/index.htm
Lien44 : http://www.css3.info/preview/attribute-selectors/
Lien45 : http://kimblim.dk/css-tests/selectors/
Lien46 : http://css.developpez.com/tutoriels/utiliser-nouveaux-selecteur-css-3/
Lien47 : http://dico.developpez.com/html/2230-Conception-workflow.php
Lien48 : https://admin.na3.acrobat.com/_a183912/p11813592/
Lien49 : http://fr.wikipedia.org/wiki/Refactorisation
Lien50 : http://labs.adobe.com/technologies/flashbuilder4/
Lien51 : http://labs.adobe.com/technologies/flashcatalyst/
Lien52 : http://livedocs.adobe.com/flex/gumbo/langref/
Lien53 : http://ellene-dijoux.developpez.com/tutoriels/web/plateforme-flash-adobe/
Lien54 : http://s-jdm.developpez.com/tutoriels/flex/traductions/fondamentaux-xml/fichiers/assets/contenu.xml
Lien55 : http://s-jdm.developpez.com/tutoriels/flex/traductions/fondamentaux-xml/fichiers/fondamentaux-xml.html
Lien56 : http://s-jdm.developpez.com/tutoriels/flex/traductions/fondamentaux-xml/fichiers/fondamentaux-xml.zip
Lien57 : http://s-jdm.developpez.com/tutoriels/flex/traductions/fondamentaux-xml/
Lien58 : http://www.atinternet-institute.com/fr-fr/barometre-des-moteurs/barometre-des-moteurs-avril-2009/index-1-1-6-170.html
Lien59 : http://docs.abondance.com/charte.html
Lien60 : http://adsense-fr.blogspot.com/2009/03/editeurs-webmasters-mythes-et-realites.html
Lien61 : http://searchengineoptimization.elliance.com/search-marketing-resources/seo-infographics.aspx?title=SEO-Checklist
Lien62 : http://www.webrankinfo.com/dossiers/redaction/lexemes-et-semes
Lien63 : http://www.google.com/webmasters/docs/search-engine-optimization-starter-guide.pdf
Lien64 : http://www.pink-seo.com/blog/les-landing-pages-37
Lien65 : http://g-gregoire.developpez.com/tutoriels/referencement/moteurs/
Lien66 : http://jf-lepine.developpez.com/tutoriels/seo/cours-referencement/
Lien67 : http://v-dubois.developpez.com/ruby-on-rails/introduction/
Lien68 : ftp://ftp-developpez.com/v-dubois/ruby-on-rails/active-record-migrations/mycontacts-chapitre2.zip
Lien69 : http://v-dubois.developpez.com/ruby-on-rails/active-record-migrations/
Lien70 : http://loic-joly.developpez.com/tutoriels/cpp/smart-pointers/
Lien71 : http://matthieu-brucher.developpez.com/tutoriels/cpp/boost/smartptrs/
Lien72 : http://alp.developpez.com/tutoriels/traitspolicies/
Lien73 : http://www.boost.org/doc/html/any.html

Lien74 : http://en.wikibooks.org/wiki/More_C%2B%2B_Idioms/Copy-and-swap
Lien75 : http://www.boost.org/doc/html/any.html
Lien76 : http://www.artima.com/cppsource/type_erasure.html
Lien77 : http://www.ddj.com/cpp/184402027
Lien78 : http://alp.developpez.com/tutoriels/type-erasure/
Lien79 : http://conception.developpez.com/livres/
Lien80 : http://mathema.tician.de/software/pycuda
Lien81 : http://jcuda.de/jcuda/jcublas/JCublas.html
Lien82 : http://jcuda.de/jcuda/jcufft/JCufft.html
Lien83 : http://jcuda.de/jcuda/jcudpp/JCudpp.html
Lien84 : http://jcuda.de/jcuda/JCuda.html
Lien85 : http://cublasnet.codeplex.com/
Lien86 : http://www.gpgpu.org/
Lien87 : http://courtois.cc/murphy/murphy_informatique.html#progsbugs
Lien88 : http://www.comp.hkbu.edu.hk/~kyzhao/
Lien89 : http://sourceforge.net/projects/cudavswizard
Lien90 : ftp://ftp-developpez.com/tcuvelier/gpgpu/cuda/introduction/fichiers/Cuda.rules
Lien91 : http://courses.ece.illinois.edu/ece498/al/
Lien92 : http://www.beyond3d.com/content/articles/106/4
Lien93 : http://www.ddj.com/cpp/207402986
Lien94 : http://www.tomshardware.com/reviews/NVIDIA-cuda-gpu,1954.html
Lien95 : http://astro.pas.rochester.edu/~aquillen/gpuworkshop.html
Lien96 : http://www.mit.edu/
Lien97 : http://tcuvelier.developpez.com/gpgpu/faq
Lien98 : http://en.wikipedia.org/wiki/BrookGPU
Lien99 : http://en.wikipedia.org/wiki/Close_to_Metal
Lien100 : http://en.wikipedia.org/wiki/AMD_Stream_Processor#Software_Development_Kit
Lien101 : http://en.wikipedia.org/wiki/CUDA
Lien102 : http://en.wikipedia.org/wiki/OpenCL
Lien103 : http://en.wikipedia.org/wiki/Larrabee_(GPU)
Lien104 : http://www.nvidia.com/object/cuda_home.html
Lien105 : http://www.nvidia.com/object/cuda_get.html
Lien106 : http://www.nvidia.com/object/cuda_learn_products.html
Lien107 : http://sourceforge.net/projects/cudavswizard
Lien108 : http://en.wikipedia.org/wiki/BLAS
Lien109 : http://fr.wikipedia.org/wiki/Transformée_de_Fourier_rapide
Lien110 : http://fr.wikipedia.org/wiki/DRAM
Lien111 : http://en.wikipedia.org/wiki/Dynamic_random_access_memory
Lien112 : http://fr.wikipedia.org/wiki/Virgule_flottante
Lien113 : http://fr.wikipedia.org/wiki/Matrice_(mathématiques)
Lien114 : http://fr.wikipedia.org/wiki/Produit_matriciel
Lien115 : http://tcuvelier.developpez.com/gpgpu/cuda/introduction/