Developpez Magazine

Edition de Juin-Juillet 2008.
Numéro 16.
Magazine en ligne gratuit.
Diffusion de copies conformes à l'original autorisée.
Réalisation : Alexandre Pottiez
Rédaction : la rédaction de Developpez
Contact : magazine@redaction-developpez.com

Index

Java	Page 2
PHP	Page 8
(X)HTML/CSS	Page 14
DotNet	Page 20
C/C++/GTK/Qt	Page 26
Office & VB	Page 32
Linux	Page 38
Mac	Page 44
2D/3D/jeux	Page 50
Algo	Page 56
SharePoint	Page 62
Liens	Page 68

Editorial

Quoi de mieux qu'un magazine de votre site préféré pour s'occuper sur la plage ?

Découvrez dans ce nouveau magazine, une série d'articles, de critiques de livres, de questions/réponses sur diverses technologies.

La rédaction

Article Java

Compte-rendu de JavaOne 2008

Voici un compte rendu du plus gros rendez-vous annuel autour des technologies Java.

par **Yann d'Isanto**
Page 2

Article PHP

PHP5 : La gestion avancée des dates

Faisons un point sur es nouvelles fonctions qui ont fait leur apparition et apportent, entres autres, la gestion des fuseaux et décalages horaires (heure d'été), autant de notions intéressantes en développement web et sur lesquelles il serait dommage de faire l'impasse.

par **Julien Pauli**
Page 8

Compte-rendu de JavaOne 2008

Voici un compte rendu de ma semaine passée à assister à JavaOne 2008, le plus gros rendez-vous annuel autour des technologies Java.

1. Avant propos

Comme chaque année, Sun organise la plus grosse conférence mondiale Java au Moscone (prononcez "mosconi") Center à San Francisco. Cette année, elle a eu lieu la semaine du 5 au 9 mai et j'ai eu la chance d'être le représentant de developpez.com pour couvrir cette conférence.

L'inscription à la conférence donne accès à une application web permettant de construire son planning avec les sessions auxquelles on veut assister. Il y a énormément de sessions et cela peut donner lieu à des choix cornéliens.

Un peu partout dans le bâtiment, des stations de travail sont accessibles mais elles sont littéralement prises d'assaut.

2. JavaOne - J1 : arrivée et CommunityOne

Comme le veut la tradition, le premier jour est réservé au Community One. Malheureusement je n'ai pas pu assister au keynote du matin, mon avion ayant atterri à San Francisco à 12h30 heure locale.

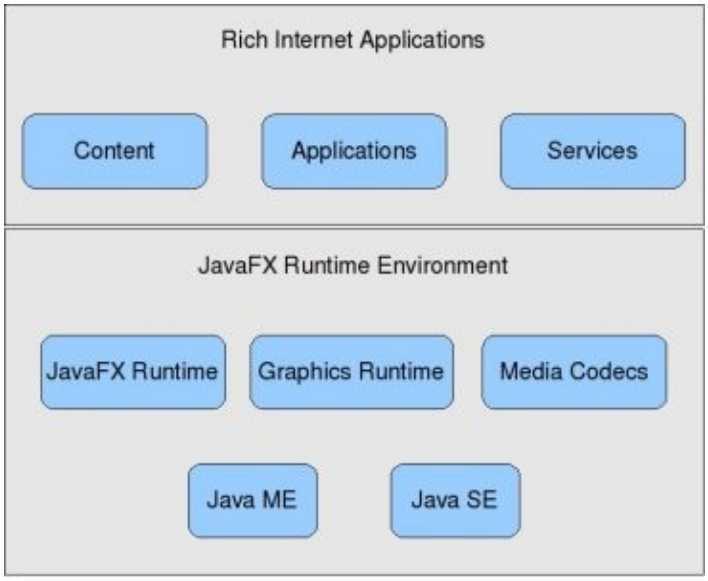
Après avoir déposé mes affaires à l'hôtel je suis allé récupérer mon badge permettant l'accès aux salles de la conférence et j'ai également eu droit à cette occasion à quelques goodies (un sac à dos, un stylo, un carnet, un CD).

J'ai ensuite assisté à deux tables rondes respectivement menées par Simon Phipps (Open Source) et Zack Urlocker (MySQL).
Lors de la première, il a beaucoup été parlé du virage amorcé par Sun il y a quelques années vers l'Open Source. Sa place s'y fait de plus en plus importante (avec notamment le récent rachat de MySQL) et son objectif est de devenir le premier contributeur du monde Open Source. Jonathan Schwartz, le CEO de Sun, a par exemple indiqué que ZFS (système de fichiers de Solaris) devrait bientôt passer sous licence GPL.

Comme on pouvait s'en douter, Sun compte améliorer le support de MySQL dans le monde Java, notamment en rapprochant l'équipe de MySQL de celles du monde Java (on pense notamment aux grands frameworks tels que Spring, etc.). Il a clairement été indiqué que Sun ne vise pas, avec MySQL, le marché des SGBDs client/serveur tel que Oracle et SQL Server (pour par exemple les ERPs) mais reste dans l'optique des applications orientées web. La question de l'intégration de Java à MySQL a également été soulevée (procédures stockées en Java ?) mais cela ne reste à l'heure actuelle qu'au stade d'idée.

3. JavaOne - J2

Cette journée a commencé avec une session générale dont la vedette était JavaFX et où nous avons eu droit à de superbes démonstrations (malgré quelques petits plantages).
Une première démonstration montrait une application tournant dans un navigateur, celle-ci a alors été "transférée" sur le bureau par un simple drag&drop. Nous avons également pu voir une application lançant simultanément la lecture de 200 vidéos en HD et dont les miniatures se déplaçaient en formant une sphère (multimédia + 3D). Voilà ce qui nous attend avec JavaFX. SUN a clairement de grosses ambitions pour JavaFX qui va plus loin que les RIA déjà présents sur le marché. JavaFX est destinée à devenir une plateforme universelle tournant au dessus de Java SE et Java ME.



Cependant il va falloir attendre encore un peu avant de pouvoir y goûter. La version 1.0 de JavaFX Desktop ne sera disponible qu'en automne 2008 et la version 1.0 de JavaFX pour Mobile et TV ne sera, quant à elle, disponible qu'au printemps 2009.

J'ai ensuite pu assister à plusieurs sessions de questions/réponses avec Param Singh (JavaFX), puis Jonathan Schwartz, et enfin James Gosling. Bien entendu le sujet principal de conversation a été JavaFX. Sun souhaite toucher un maximum d'appareils et James Gosling lui-même a avoué que, en ce qui concerne les appareils mobiles, cela n'était pas une tâche aisée et nécessitait encore beaucoup de travail.
JavaFX arrive donc assez tard, et la bataille sur le marché des RIA bat déjà sont plein avec SilverLight (Microsoft) et AIR (Adobe). JavaFX a de quoi séduire, son framework est très riche et orienté multimédia (avec JavaFX, Sun veut "rapprocher le designer du développeur"). JavaFX devrait également supporter un large ensemble de codecs (bien que l'on n'en connaisse pas encore la liste) et intégrer un système de base de données (JavaDB pour Java SE et SQLite pour Java ME).
Un autre point intéressant est le fait que JavaFX soit 100% open source (cela devient une habitude chez Sun) au contraire de ses deux concurrents.

Cependant, son retard est son principal point faible, beaucoup de

zones d'ombres subsistent : on ne connaît pas encore les spécifications nécessaires pour faire fonctionner JavaFX et, même s'ils sont prévus, les outils de développement (qui seront sûrement intégrés à NetBeans sous la forme de plugin) n'ont pas de roadmap bien définie.

Sun compte néanmoins sur la large présence de la JVM sur les PC (90%), les mobiles (85%), mais aussi sur tous les lecteurs Blu-Ray pour faciliter l'adoption de JavaFX.

4. JavaOne - J3

4.1. TS-6605 : Deep Inside JSR 296, the Swing Application Framework

J'ai commencé ma journée par la session technique TS-6605 : "Deep Inside JSR 296, the Swing Application Framework" présentée par Hans Muller et Tomas Pavek.
La JSR 296 propose un framework pour le développement d'application Swing. Le framework permet de construire facilement une application Swing en se basant sur une structure commune à la plupart des applications desktop. La session était plutôt convaincante notamment grâce à une démonstration s'appuyant sur l'IDE NetBeans qui permet de créer un projet basé sur le Swing Application Framework.

Ce framework apporte un support relativement avancé pour :

- le cycle de vie de l'application,
- la gestion des ressources de l'application (injection),
- la simplification pour la création des actions (annotation),
- l'exécution des tâches de fond (avec notamment la gestion de ce qui s'exécute ou pas dans l'EDT en se basant sur la classe SwingWorker),
- la sauvegarde de l'état graphique (position et taille de la fenêtre, ...).

Son utilisation n'est pas très compliquée, plutôt efficace, et permet d'éviter pas mal de "pièges" relatifs au développement d'une application Swing.

4.2. Scripting panel

J'ai ensuite assisté à une session de questions/réponses sur le thème des langages de script et langages dynamiques à laquelle participait Charles Nutter (JRuby), Tor Norbye (NetBeans), Ted Leung (Jython), Guillaume Laforge (Groovy), Greg Murray (jMaki) et Tim Bray (Sun).

En effet, les langages dynamiques supporté par la JVM fleurissent (Scala, ..., JavaFX qui possède lui aussi sont propre langage de scripting) et leur nombre grandissant soulève des interrogations. Pourquoi multiplier les langages supportés par la JVM ? Quel est l'intérêt des langages dynamiques ? Comment choisir quel langage utiliser ? ...

Avec l'ouverture de la plateforme Java, il est logique de voir le support d'autres langages arriver. L'avantage est que chaque langage apporte sa spécificité, par exemple : Ruby a une syntaxe plus flexible (rapide à écrire), Python est un langage plus concis (rapide à lire), Groovy apporte une haute intégration des fonctionnalités "entreprises", etc.
De plus ces langages peuvent dès lors profiter de toute la plateforme Java (par exemple JRuby est plus rapide que Ruby). Les langages dynamiques sont plus souples, apportent une syntaxe plus simple, et génèrent moins de code. Le choix du langage peut dépendre pour une petite part de spécificités techniques mais se résume généralement plus à une question de goût.

4.3. TS-6050 : Comparing JRuby and Groovy

Sur le même sujet, la journée a continué avec la session technique TS-6050 sur la comparaison entre Groovy et JRuby animée par Neal Ford. Je ne connaissais pas ces deux langages et cela a été un bon moyen pour moi de les découvrir.

Leur comparaison a été faite sur les points suivants :

- la syntaxe,
- la philosophie,
- les méthodes de test,
- la meta programmation.

Ces deux langages se ressemblent assez malgré quelques différences qui se révèlent surtout être des différences de philosophie. Ce qui est faisable avec l'un l'est généralement aussi avec l'autre, c'est plutôt dans la façon de faire qu'il y a une différence. Même en ce qui concerne les performances ces deux langages se révèlent semblables.

4.4. Table ronde NetBeans

J'ai ensuite participé à une table ronde sur NetBeans avec David Folk. Il a d'abord été question de la sortie de NetBeans 6.1 avec notamment le support de PHP et Javascript, l'amélioration du support de MySQL, des WebServices et de Spring.

Tout comme la plateforme Java, NetBeans s'ouvre de plus en plus aux autres langages. Après Ruby c'est au tour de Python dont le support est actuellement en cours de travail. Le support d'autres langages (Groovy, Scala, etc.) dépendra de la demande de la communauté.
Bien entendu il a également été question du support de JavaFX, la star de ce JavaOne. NetBeans offre déjà un support pour JavaFX Script. Comme la plateforme JavaFX cible les designers, il a été demandé si Sun fournira des outils de création pour JavaFX dans NetBeans. La réponse est oui : Sun a bien prévu de fournir une suite d'outils, malheureusement le quand n'est pas encore défini. A noter que cela ne fera pas partie du projet NetBeans, mais que cela se présentera plutôt sous la forme de plugin pour l'IDE.

4.5. TS-5199 : JMX : Java Management Extensions (JMX) Technology Update

Changement de sujet avec ma dernière session technique de la journée qui portait sur JMX (TS-5199), session présentée par Jean-François Denise et Eamonn McManus.
La session a commencé par une présentation de l'API (ce qui tombait bien puisque je ne la connaissais pas) puis par une description des nouveautés prévues pour la prochaine version.
Pour ceux qui ne connaissent pas, JMX est une API (présente depuis Java 5) qui permet de gérer et monitorer une application qui s'exécute dans la JVM. Par exemple, la console Java JConsole exploite JMX pour restituer ces informations et administrer une application. Une version 2.0 de cette API devrait normalement être intégré à Java 7 via les JSR 255 et 262. Les possibilités de cette API nous ont été montrées au travers d'une démonstration plutôt convaincante utilisant le projet CSPoker (Lien1).

5. JavaOne - J4

5.1. TS-4895 : The NetBeans IDE Compared to the Eclipse Rich Client Platform

Fervent utilisateur de NetBeans, je m'étais déjà intéressé à sa plateforme mais avais vite abandonné, surtout par manque de temps. J'ai donc choisi la session technique TS-4895 présentée par Kai Toedter (Eclipse) et Geertjan Wielenga (NetBeans) afin

d'en apprendre un peu plus.

La comparaison a été illustré par une application de démo, "MP3 Manager". Les deux plateformes se ressemblent beaucoup et ont une architecture très semblable (principe des plugins), dans les deux cas il est possible de créer des projet pour développer une application utilisant la plateforme considérée. En fait, leur principale différence réside dans le fait qu'Eclipse utilise SWT là où NetBeans utilise Swing. Mais sinon à peu près tout ce qui est faisable avec l'une l'est aussi avec l'autre, et ceci de façon assez similaire.

5.2. TS-5541 : Creating Better Applications at Boeing with the NetBeans Platform Application Framework

Toujours afin de mieux connaître la plateforme NetBeans, j'ai choisi d'assister à la session technique TS-5541 animée par Bruce Shimel et Tom Wheeler.

Malheureusement je dois avouer être quelque peu resté sur ma faim, cette session étant bien moins technique que ce à quoi je m'attendais.
En effet, c'est surtout la question du pourquoi choisir la plateforme NetBeans qui a été traitée : ne pas réinventer la roue et se concentrer sur les développements spécifiques à l'entreprise.

Boeing a donc construit une plateforme (Boeing NetBeans Platform (Lien2)) par dessus celle de NetBeans qui apporte notamment le support de JFreechart et d'un viewer Java3D. C'est cette plateforme qui est utilisée par les ingénieurs de Boeing et qui leur permet de produire une application riche en approximativement une heure.

5.3. TS-6656 : Extreme GUI Makeover

J'ai continué ma journée très orientée "desktop" avec la session technique TS-6656 animée par Christopher Campbell et Ben Galbraith. Cette session (ainsi que la suivante) est à recommander aux personnes qui pensent que Java est lent et moche ainsi qu'aux fans d'interfaces graphiques conviviales et évoluées. C'est le genre de session où l'on apprend, ou revoit, plein de petites techniques (en l'occurrence sur Swing) telles que

- choisir entre réécrire la méthode paintComponent où implémenter un UIDelegate pour personnaliser un composant graphique,
- faire son propre Look and Feel, en choisir un déjà existant et le plus proche du voulu pour ensuite le modifier,
- utilisation du glasspane et des JLayeredPane pour les animations,
- utilisation le timing framework pour les animations,
- ...

La démonstration s'appuyait sur une très jolie interface aux effets vraiment 'wouaw !' et montrait pas mal de code.
Un point qui a été souligné est que le design des interactions est différent du design visuel et que c'est souvent là qu'une application pèche.

5.4. TS-6611 : Filthy-Rich Clients: Filthier, Richer, Clientier

En continuité de la session précédente, j'ai terminé ma journée par la session technique TS-6611 animée par Romain Guy et Chet Haase. Là encore on en prend plein les mirettes et tout développeur Swing est aux anges.

Cette session peut être considérée comme un "add-on" à leur livre Filthy Rich Clients (Lien3), que personnellement je recommande à tous les développeurs Swing.
Il a d'abord été question du problème engendré par un nombre d'animations trop élevé. En effet, chaque animation se base généralement sur un Timer et leur multiplication entraîne une chute dramatique des performances. La solution est de n'utiliser qu'un seul Timer pour toutes les animations en utilisant soit la classe TimingSource du Timing Framework (Lien4), soit le projet Scene Graph (Lien5). Le tout était illustré par du code et une démonstration.
Nous avons ensuite assisté à la mise en place d'effets graphiques relativement simples mais très efficaces pour le drag&drop d'images, là encore démonstration et code à l'appui.

6. JavaOne - J5

Ce cinquième et dernier jour a commencé par un hommage à John Gage puis par le traditionnel "Gosling's Toy Show" aussi nommé "Extreme Innovation" où James Gosling présente son classement des meilleurs innovations technologiques utilisant Java.

- **Java VisualVM** : *VisualVM* et une application de monitoring, de profiling et d'analyse de vos applications Java. Une sorte de JConsole avancée.
  -> https://visualvm.dev.java.net/ (Lien6)
- **JavaScript Support in Netbeans** : *NetBeans* offre, depuis sa version 6.1, un support complet de JavaScript avec notamment la complétion de code et un debuggeur.
  -> http://www.netbeans.org/ (Lien7)
- **Java on NVidia APX 2500** : Support de Java par l'*APX 2500* qui est un processeur pour smartphone pouvant traiter des images d'une résolution de 12 mégapixels et possédant un coeur graphique 3D de classe GeForce 6 avec lequel il devient possible d'exécuter une application OpenGL écrite pour le desktop en utilisant de simples bindings OpenGL.
- **Java Games and Project Darkstar** : Présentation du projet *Darkstar* pour le développement de jeux vidéos en Java, avec en exemple le jeu *Call of the Kings*.
  -> http://www.projectdarkstar.com/ (Lien8)
  -> http://calloftheKings.com (Lien9)
- **Extreme Java Card Innovation** : Mise en oeuvre de l'API *Robocode* sur une carte à puce au travers d'un concours.
- **Pervasive Java** : L'entreprise *Sentilla* propose une plateforme logicielle basée sur Java pour des applications embarquées.
  -> http://www.sentilla.com (Lien10)
- **Never Miss a Word** : Un stylo intelligent capable d'enregistrer ce que vous écrivez, de le traduire, de jouer sur un piano que vous auriez dessiné ...
  -> http://www.livescribe.com (Lien11)
- **Industrial Strength Java** : Présentation de l'architecture *Blue Wonder* qui repose sur une combinaison de PC x86, Solaris et Java RTS pour une application de contrôle industriel.
- **Java : Licence to Drive** : *Tommy Junior* est une voiture autonome qui utilise une plateforme logicielle 100% Java (Java RTS, Java SE, Java ME).
  -> http://www.teamjefferson.com (Lien12)
- **Java Rocks on Mars** : *JMars* est un système d'information géospatial basé sur la technologie Java utilisé pour faire de l'analyse de données et de la planification de mission par la NASA (Mars Odyssey, Mars Reconnaissance Orbiter et bientôt Lunar Reconnaissance Orbiter).
  -> http://jmars.asu.edu/ (Lien13)
- **CERN : Accelerating Java** : L'*organisation européenne pour la recherche nucléaire* utilise Java pour gérer le LHC (*Large Hadron Collider* ou *Grand collisionneur de hadrons* en français), un gigantesque accélérateur de particules digne des meilleurs films de

science-fiction. Avec le détecteur ATLAS on parle d'un flux de données de 2 pétas-octets/seconde (soit environ 3 millions de CDs par seconde).
-> http://public.web.cern.ch/Public/fr/LHC/LHC-fr.html (Lien14)

Malheureusement, mon avion décollant de San Francisco à 12h, mon JavaOne s'est terminé là.

7. Le mot de la fin

Ce JavaOne a donc beaucoup tourné autour de JavaFX qui a énormément d'atouts pour faire une entrée remarquée dans le monde des RIA. Cependant, son retard est une pénalité sévère et tout retard supplémentaire pourrait bien lui être fatal (notamment en ce qui concerne les outils de développement).
Un autre point sur lequel Sun a beaucoup communiqué est l'ouverture (au sens le plus général du terme) de la plateforme Java avec notamment un support accru pour d'autres langages sur la JVM.

A noter que Sun commence à publier l'ensemble des sessions sur http://developers.sun.com/learning/javaoneonline/ (Lien15) (toutes ne sont pas encore disponibles). Des vidéos des différentes sessions générales sont également disponibles : http://java.sun.com/javaone/sf/sessions/general/ (Lien16)

Enfin, il y a des photos :
- sur le site de Sun (Lien17)
- sur FlickR (Lien18)

Sur un plan plus personnel, ce fût une semaine vraiment inoubliable et j'espère bien avoir la chance d'y retourner une prochaine fois.

Retrouvez l'article de Yann d'Isanto en ligne : Lien19

Stratégies de chargement avec Hibernate

Ce petit tutoriel n'a nulle vocation à être parfait ni même exhaustif. Il n'est que le reflet de mon expérience avec Hibernate, des préconisations glanées ci et là, et mises en œuvre dans mon cadre professionnel quotidien. Certains points peuvent prêter à débat, mais de manière générale, les principes appliqués ici m'ont toujours permis de livrer une application fonctionnelle ayant des performances honorables en temps et en heure.

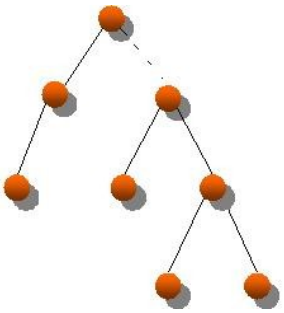
1. Introduction

Hibernate permet aux développeurs de s'affranchir des nombreuses problématiques liées à la gestion d'une base de données. Le niveau d'abstraction est tel (utilisation de POJO, sauvegarde automatique, etc…) qu'un mythe veut que l'utilisation de la librairie de persistance ne nécessite peu ou pas de connaissance des mécanismes sous-jacents.

C'est hélas non seulement faux, mais également dangereux pour la survie de votre application. En effet, sur un domaine aussi sensible que le chargement de données, se reposer aveuglément sur les mécanismes natifs d'Hibernate peut vous mener droit à la catastrophe. Petite revue des options disponibles…

2. Notions fondamentales

La manipulation des entités Hibernate se limite rarement à des objets sans lien les uns avec les autres. Au contraire, les bases de données relationnelles mettent en œuvre toute une panoplie d'index reliant les enregistrement les uns aux autres. Hibernate permet la manipulation de ces enregistrements sous forme de **graphe d'objets**, que l'on peut représenter sous la forme d'un arbre :



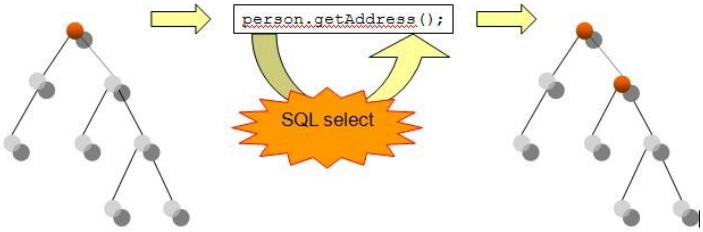
Or le chargement d'un tel graphe d'objet peut s'avérer coûteux en temps d'exécution (bien plus qu'en occupation mémoire), d'où l'impérieuse nécessité de les remplir avec soin et parcimonie.

3. Le chargement à la demande

3.1. Principe

Le chargement à la demande (ou " lazy loading ") est la stratégie native mise en œuvre par Hibernate. Elle consiste à ne charger que le minimum de données, puis de générer une nouvelle requête SQL pour récupérer les données supplémentaires lorsque celles-ci seront demandées par le programme.

Dans la pratique, Hibernate charge tous champs de la table principale, et les clés étrangères sont stockées sous forme simplifiée (seul l'ID est renseigné), ce que l'on nomme un proxy. Lorsque le programme essaiera d'accéder aux membres de ce proxy, Hibernate générera une requête SQL et récupérera les données nécessaires afin de le remplir…



3.2. Avantage

Le principal avantage du lazy-loading est bien entendu sa transparence. Le programme utilise les objets du Domaine en toute simplicité, et seules les données vraiment nécessaires sont chargées.

3.3. Inconvénients

Si séduisante qu'elle puisse paraître, cette approche présente les défauts suivants :

- Absence de maîtrise du chargement : chaque appel peut potentiellement provoquer une requête en base de données, et ce, de manière complètement transparente et incontrôlable. On peut ainsi assister à des effondrements spectaculaires des performances de l'application, dus à un excès de requêtes SQL. De plus, l'utilisation de certaines librairies, basées notamment sur l'introspection _ telles que JAXB ou Dozer_ provoque le chargement de la totalité du graphe d'objet lors de la sérialization du graphe d'objets
- N+1 select : corollaire du point précédent, le chargement de collections se traduit par un nombre rédhibitoire de requêtes. Regardons l'exemple suivant :

```
List<Address> addressList = person.getAddressList() ; (1)
for (Address address : addressList)
{
    displayCity(address.getCity(), address.getZipCode()) ; (2)
…
}
```

L'appel (1) génère une requête SQL remontant la liste des identifiants des adresses composant la liste, et que ***chaque*** appel (2) génère une autre requête SQL afin de remplir la totalité de l'objet Address, soit au total N+1 requêtes (N étant la cardinalité de la liste).

Seule solution, non pas pour maîtriser mais pour diagnostiquer ce genre de problématique : passer le paramètre de configuration " show_sql " à 'true' afin de visualiser en développement les requêtes générées automatiquement par les proxies Hibernate.

4. Eager fetching

4.1. Principe

Solution diamétralement opposée au chargement à la demande, le " eager-fetching " consiste à systématiquement charger l'intégralité du graphe d'objets associés.

Techniquement, il suffit de d'ajouter l'attribut " lazy " à 'false' (ou fetch='join') pour toutes les associations (one-to-one, many-to-one, one-to-many) et le tour est joué.

4.2. Avantage

Tout comme le chargement à la demande, cette stratégie de chargement présente l'avantage de la simplicité. De plus, une fois la donnée chargée, cette dernière est entièrement et immédiatement disponible, sans génération de requête SQL supplémentaire.

4.3. Inconvénient

Vous l'aurez deviné, là où le bât blesse, c'est justement au temps de chargement de votre graphe d'objets, qui peut vite s'avérer rédhibitoire avec un schéma classique (quelques associations suffisent à faire exploser les performances).

Si on y ajoute une grande propension au syndrome du produit cartésien (cf. encadré), vous comprendrez bien que cette approche est à bannir de toute application manipulant un graphe d'objet un tant soit peu conséquent.

Règle n°1 : pas plus d'une collection à la fois

Le chargement des collections (association many-to-one, coté " one ") est un point délicat de la programmation avec Hibernate.

En effet le chargement de plusieurs collections en parallèle risque de fortement dégrader les performances de votre application car une telle requête remonte en général trop de données (*produit cartésien*). En effet, le résultat de la requête SQL est composé de la **combinaison** des résultats possibles sur chaque jointure, comportant une multitude de doublons qu'Hibernate devra éliminer.

A titre d'exemple, une requête renvoyant 100 éléments, chacun associé à deux collections de 10 éléments chacun, serait constituée de (100*10*10) = 10 000 lignes de résultats.

Outre le temps de traitement par Hibernate, le temps de transfert d'un tel volume de résultat achèvera de réduire les performances de votre application à néant !

5. Chargement explicite

5.1. Principe

L'idée directrice est ici de définir à l'avance les associations qui seront nécessaires à l'affichage ou au traitement d'une entité, puis de les charger en une seule requête. Cela tombe plus ou moins sous le (bon) sens : on définit une fois pour toutes ce dont on a besoin et on le charge, sans recours à une fonctionnalité " magique " qui fait le boulot en aveugle.

Ainsi, la couche d'accès aux données se décompose en plusieurs méthodes de chargement (par exemple loadPersonAndAddress, loadPersonAndFamily, loadPersonAddressAndFamily, …), chacune décrivant le chargement associé.

Techniquement, les requêtes sont plus complexes qu'un simple chargement Hibernate : il faut définir explicitement les jointures qui seront chargées par la méthode :

```
StringBuilder hqlQuery = new StringBuilder();
hqlQuery.append("from A a");
hqlQuery.append(" inner join fetch a.b1");
hqlQuery.append(" inner join fetch a.b2");
hqlQuery.append(" inner join fetch a.c");
hqlQuery.append(" inner join fetch a.c.d");
hqlQuery.append(" where a.id = :id");

Query query = session.createQuery(hqlQuery.toString());
query.setLong("id", idList.get(0));

return (A) query.uniqueResult();
```

5.2. Avantage

Comme vous vous en doutez si vous m'avez suivi jusqu'ici, cette solution est de loin la plus performante des trois. De plus, elle permet de contrôler au plus près le graphe des objets chargés.

5.3. Inconvénient

Si le chargement explicite apporte enfin des performances maîtrisables et acceptables, elle présente un prix à payer relativement important :

- Le plus évident réside dans le nombre de méthodes à écrire. Alors qu'avec les deux premières méthodes, une seule méthode de chargement suffit, le chargement explicite se traduit généralement par des DAO conséquentes aux nombreuses méthodes
- L'écriture des méthodes de chargement devient plus technique qu'un simple load ou get Hibernate : la maîtrise d'Hibernate (différents types de jointures, HQL, criterias, ...)
- Enfin, comme il n'est pas possible de débrayer le chargement à la demande natif d'Hibernate, tout accès à des propriétés non chargées se traduit soit par une requête en base de données, soit par une *LazyInitialisationException* si la session Hibernate a été fermée.

Règle n°2 : Ne mesurez jamais les performances en local

L'un des conforts de développement les plus trompeurs consiste à estimer les performances avec une base de données locale à l'application. Les temps de transfert réseaux sont alors nuls, et les règles (notamment la première) et observations de cet article

risquent fort de passer inaperçues… jusqu'au moment du déploiement !

Les mesures de performance doivent être effectuées de manière régulière avec un environnement de test réaliste (base de données sur un serveur distinct + jeu de données avec une volumétrie proche de celle estimée en déploiement).

6. Chargement par interface

6.1. Principe

Avec le chargement par interface, on sort ici des sentiers battus, puisque cette technique est le fruit de mon expérience. Elle permet de résoudre une bonne partie des inconvénients du chargement explicite, en gardant la maîtrise des associations chargées.

Le chargement par interface repose sur une interface de chargement, qui représente la vue de l'objet à charger.

Exemple : soit la classe du Domaine suivante :

```
public class Person
{
	// Attributes
	private Long id;
	private String firstName;
	private String lastName;
	private Address address;
	private Job job;

	//	Getters and setters
	public Long getId() { return id; }
	public void setId(Long id) { this.id = id; }

	public String getFirstName() { return
firstName; }
	public void setFirstName(String firstName)
{ this.firstName = firstName; }

	public String getLastName() { return
lastName; }
	public void setLastName(String lastName)
{ this.lastName = lastName;}

	public Address getAddress() { return address; }
	public void setAddress(Address address)
{ this.address = address; }

	public Job getJob() { return job; }
	public void setJob(Job job) { this.job = job; }
}
```

On peut en extraire l'interface de chargement suivante :

```
public interface IWorker
{
	public Long getId();
	public void setId(Long id);

	public String getFirstName();
	public void setFirstName(String firstName);

	public String getLastName();
	public void setLastName(String lastName);

	public Job getJob();
	public void setJob(Job job);
}
```

Il suffit alors de générer les requêtes de jointures par introspection afin de charger les associations contenues dans l'interface de chargement (méthode 'addFetchingStrategy'):

Retrouvez la suite de l'article de Bruno Marchesson en ligne : Lien20

PHP5 : La gestion avancée des dates

Depuis PHP 5.1, la gestion des dates en PHP a profondément changé. Certaines fonctions ont été réécrites, la gestion interne des dates s'est agrandie et elle est devenue indépendante de l'OS sous-jascent.
De nouveaux objets/fonctions ont fait leur apparition qui apportent, entres autres, la gestion des fuseaux et décalages horaires (heure d'été), autant de notions intéréssantes en développement web et sur lesquelles il serait dommage de faire l'impasse. Faisons un point dessus.

1. Introduction aux dates en PHP

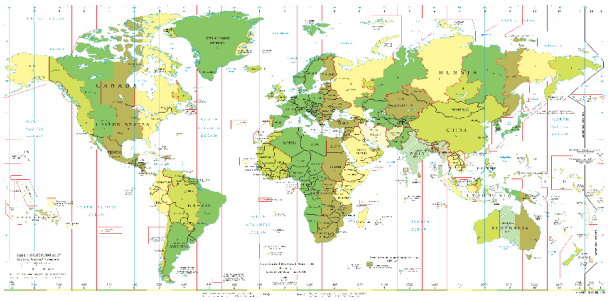
Tout langage web doit être doté de fonctionnalités concernant les dates. C'est le cas de PHP depuis bien longtemps. Cependant sur Internet, qu'est ce qu'une date ?
La date est globalement la représentation d'un jour, d'un mois, d'une année, d'une heure, de minutes, et de secondes. Le problème vient d'Internet : l'International Network.
La Terre n'est pas plate, et n'est pas une communauté unique qui parle la même langue et utilise les mêmes représentations pour des notions comme les dates.
Une date doit être fixée à un point de la planète, et écrite d'une certaine manière selon ce point.

2. Les problèmes de dates

- Chaque lieu de la planète est assimilé à un fuseau horaire amenant à un décalage positif ou négatif par rapport à UTC (Lien21), (anciennement GMT (Lien22) : le fuseau zéro passant par Greenwich)
- Certains pays sont traversés par plusieurs fuseaux (4 , 5... c'est le cas des USA ou de la Russie).
- Le décalage d'un fuseau horaire peut être d'une heure, mais aussi (à certains endroits) d'une demie ou un quart d'heure
- Des pays ont des fuseaux horaires horizontaux : l'Australie, le découpage est donc sous forme de carrés
- Certains lieux changent de fuseau pendant l'année
- Certains pays utilisent une notion "d'heure d'été" (DST : Daylight Saving Time), en plus de la notion de fuseau horaire
- Ces pays ne passent pas à "l'heure d'été" tous à la même période
- Le décalage d'"heure d'été" n'est pas le même pour tous, il peut être d'une heure, mais aussi d'une demie ou un quart d'heure

Cela fait déja beaucoup, mais ça n'est pas terminé :

- Les abréviations des noms des fuseaux horaires utilisées en informatique, ne sont pas les mêmes d'un OS à un autre
- Ces abréviations peuvent être confuses
- Les représentations des dates sont très différentes : September 16th, 2005; 20060225040821; 2003-08-11 16:12:07.11403+01; 2001-12-17T12:10:27.123-05:00...



timezones

3. Les dates en PHP <= 5.0

En PH4 et 5.0, les fonctions de dates sont les suivantes :

1. ***gettimeofday()*** - Récupère le temps actuel (tableau PHP)
2. ***checkdate()*** Valide une date Grégorienne
3. ***localtime()*** - Récupère le temps local (tableau PHP)
4. ***date()*** / ***gmdate()*** - Formate une date locale/GMT
5. ***mktime()*** / ***gmmktime()*** - Récupère le timestamp Unix d'une date
6. ***strftime()*** / ***gmstrftime()*** - Formate une date/temps locale/GMT en fonction de paramètres locaux
7. ***getdate()*** - Récupère des infos de date/temps depuis un timestamp
8. ***strtotime()*** - Récupère un timestamp depuis un texte écrit en anglais (now, yesterday ...)

Toutes ces fonctions utilisent la notion de timestamp UNIX : le nombre de secondes écoulées depuis "EPOCH" : le 1er Janvier 1970 à 0h00.
Le problème est que ce chiffre ne représente rien pour un utilisateur d'un site lambda : il faut le convertir en date, dans le format que l'utilisateur a l'habitude de voir.
Et c'est bien là tout le problème : relisez la liste dans la section précédente. Pour formater une date d'un point donné (le serveur), en une date d'une autre point (le client), il faut du courage...

Ces fonctions-là répondent néanmoins à beaucoup de besoins, mais elles ne sont pas suffisantes :
Les timestamps sont stockés sur 32bits, et vont de 1902 à 2038, mais certains OS ne traitent que les entiers positifs : la limite est donc de 1970 à 2038.
Aussi, ces fonctions ne gèrent pas correctement (voire pas du tout pour certaines) les notions de fuseaux horaires et de décalage d'heure d'été, et beaucoup d'entre elles dépendent de l'OS sur lequel elles sont utilisées, elles renvoient donc des résultats différents en fonction de celui-ci.

4. PHP5.1 et supérieur apporte une solution

Heureusement, depuis PHP5.1, une solution intérssante existe grâce à l'apparition de 2 objets simplifiant la gestion des dates, et surtout proposant d'autres avantages

Ces fonctions sont disponibles dans la distribution standard, à partir de PHP 5.2
Il est possible d'ajouter un support expérimental dans PHP 5.1.x en utilisant le drapeau CFLAGS=-DEXPERIMENTAL_DATE_SUPPORT=1 lors de la compilation.

- Calculs effectués autour d'un entier 64bits signé
- Plus de dépendances envers l'OS : embarquement d'une base timezonedb complète, mise à jour régulièrement
- Support complet des fuseaux et des décalages (plus de 550 fuseaux)
- Abréviations (Lien23) normalisées des fuseaux : Continent/Lieu/SousLieu
- Calculs sur les dates (ajouts, retranchements)
- Formats normalisés : ISO8601, RSS, W3C, Cookie

4.1. Les objets DateTime et DateTimezone

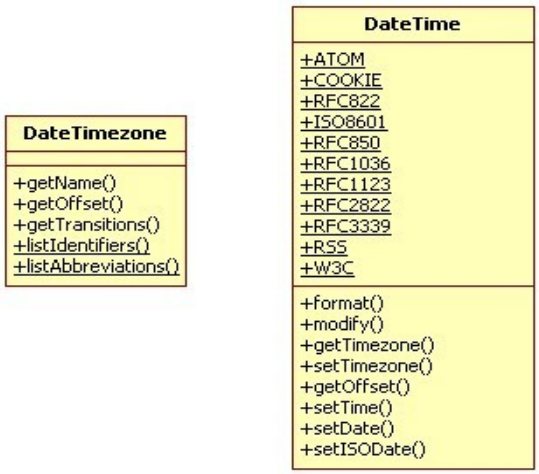


Diagramme de classes concernant la gestion des dates en PHP5.1 et supérieurs

Dans PHP5.1 et ultérieurs, avec le nouveau support des dates, il n'est plus question de manipuler des timestamps signés.

Utilisation des fonctions classiques

```
<?php
$date = strtotime("1969-12-31 12:13:40");
echo $date;
// affiche -13390
```

Mais plutot des objets de date, qui sont des représentations sur un entier 64bits signé, et qui peuvent être obtenus soit par instanciation de la nouvelle classe **DateTime**, soit par utilisation des nouvelles fonctions apparues, comme ***date_create()*** :

utilisation des fonctionnalités avancées

```
<?php
$date = new DateTime("1969-12-31 12:13:40");
// équivalent strictement à $date =
date_create("1969-12-31 12:13:40");
echo $date->format('U'); // affiche -13390
echo $date->format(DateTime::RSS); // affiche Wed, 31
Dec 1969 12:13:40 +0100
```

L'objet **DateTime** ou ***date_create()***, prennent un paramètre tel qu'accepté par ***strtotime()***, elle même calée sur les recommandations GNU des dates (Lien24)

Si l'envie vous prend d'utiliser un timestamp tout de même, veillez à le faire précéder du symbole '@' :

```
$date = new DateTime('@' time());
```

Notez que ceci n'est pas la procédure recommandée, mais si le besoin se présente... new DateTime() est valide (sans paramètres) : le défaut est 'now'.
Mais attention tout de même : pour que tout cela fonctionne, il est necéssaire d'affecter un fuseau à une date. Pour que PHP ne se repose plus sur l'OS, il faut obligatoirement lui dire quel est le fuseau sur lequel il devra caler toutes ses dates (le fuseau par défaut).
Ceci est possible au moyen de la fonction ***date_default_timezone_set()*** (Lien25), ou alors via la directive date_timezone (Lien26) du fichier php.ini

Tout appel a une fonction de date, alors qu'aucun fuseau (timezone) n'est défini explicitement, lèvera une erreur PHP non fatale (E_WARNING ou E_STRICT selon la fonction utilisée) Par la suite, nous supposerons la timezone par défaut comme étant 'Europe/Paris'.

Ainsi, travailler avec les anciennes fonctions PHP sur les dates, n'a plus grand interêt car on ne profite aucunement des nouvelles fonctionnalités. Il ne faut plus penser timestamp, mais bien objet date.
Les options de la méthode ***format()*** sont les mêmes que celles de la fonction PHP ***date()***, à laquelle l'option 'U' a été rajoutée depuis PHP5.1 donc.
De la même manière, cette méthode ***format()*** sur un objet **DateTime**, est équivalente à la fonction ***date_format($objetDateTime)***.

L'ajout des fonctionnalités relatives aux dates à partir de PHP 5.1 a été faite autant avec un style objet que procédural. Nous préfèrerons pour la suite le style objet.
Ce double support procédural/objet se retrouve dans d'autres fonctionnalités de PHP5 comme MySQLi

La base de données des fuseaux et des décalages, sur laquelle est basé le désormais standard (>=PHP5.1) support des dates, se trouve dans PECL (Lien27). Il n'est pas spécialement nécessaire de la mettre à jour, une nouvelle version est de toute façon intégrée lors de la migration de version de PHP.
Cette base est une compilation de la base OLSON (Lien28).

formats normalisés

```
<?php
$date = new DateTime("1819-02-14 10:51");
echo $date->format('D Y-m-d H:i:s - \s\e\m\a\i\n\e W');
// Sun 1819-02-14 10:51:00 - semaine 06

echo $date->format(DateTime::ATOM); //
1819-02-14T10:51:00+00:09
echo $date->format(DateTime::COOKIE); //Sunday, 14-
Feb-19 10:51:00 PMT
echo $date-
>format(DateTime::ISO8601); //1819-02-14T10:51:00+0009
echo $date->format(DateTime::RSS); //Sun, 14 Feb 1819
10:51:00 +0009
echo $date-
>format(DateTime::W3C); //1819-02-14T10:51:00+00:09
echo $date-
>format(DateTime::RFC3339); //1819-02-14T10:51:00+00:09
```

Remarquez le +00:09. Mon fuseau est pourtant bien Europe/Paris, pourquoi n'ai-je pas +01:00 comme je pourrais m'y attendre ? Très simple : en 1819, à cette date-là précisémmment, le décalage de mon fuseau n'était que de 00:09 !

La base de données timezonedb comporte vraiment toutes ces infos.

```
<?php
$date = new DateTime("2019-01-14 10:51");
echo $date-
>format(DateTime::ISO8601); //2019-07-14T10:51:00+0100

$date = new DateTime("2019-07-14 10:51");
echo $date-
>format(DateTime::ISO8601); //2019-07-14T10:51:00+0200
```

L'heure d'été est aussi prise en compte. En juillet nous sommes à +02, alors qu'en janvier nous sommes bien à +01, ça sent bon :)

4.2. La gestion des fuseaux horaires

Toute date est associée à une timezone : un fuseau horaire. Celui-ci permet de mesurer le décalage par rapport à GMT (ou UTC si on préfère, l'abus de langage est courant...).
Ce décalage est la somme du décalage du fuseau (la Terre n'est pas plate) ajouté au décalage éventuel de l'"heure d'été".
La gestion des fuseaux est possible grâce à l'objet **DateTimezone** (on a des fonctions PHP aux effets similaires). Cet objet est séparé de l'objet **DateTime**, mais les 2 restent très proches :

```
<?php
$date = new DateTime("now"); // 'now' n'est pas
nécéssaire, c'est la valeur par défaut
echo $date->format(DateTime::ISO8601) //
2008-05-27T20:30:46+0200

$tz = new DateTimeZone('America/New_York');
$date->setTimezone($tz);
echo $date->format(DateTime::ISO8601) //
2008-05-27T14:30:46-0400

echo $date->getOffset(); // -14400
// équivalent à $ts->getOffset($date)
```

Une date peut donc changer de fuseau, son décalage sera géré automatiquement. Lorsqu'on interroge le fuseau via ***getOffset()***, nous sommes obligés de lui passer un objet **DateTime**. C'est logique : le décalage varie en fonction du fuseau certes, mais de la date que l'on cherche (heure d'été par exemple).
Ainsi, il n'y a plus besoin de calculer quoi que ce soit en additionnant ou soustrayant des timestamps, ici tout est géré, quels que soient le fuseau et la date.

Affectation directe d'un fuseau

```
<?php
// affectation directe d'un fuseau précis (autre que
celui par défaut) à une date
$date = new DateTime('now',new
DateTimeZone('Europe/London'));
```

La classe **DateTimeZone** possède encore quelques méthodes, la méthode statique ***listAbbreviations()*** va lister toutes les abréviations des fuseaux horaires que les OS ou les programmes avaient l'habitude d'utiliser (attention, liste extrêmement longue et coûteuse). Il est déprécié d'utiliser de telles abréviations à présent, le nom d'un fuseau doit toujours être indiqué selon une liste précise (Lien29)
Cette liste est d'ailleurs obtenue par la méthode statique ***DateTimeZone::listIdentifiers()***
Plus intéressante, la méthode ***getTransitions()*** va retourner un tableau indiquant tous les décalages programmés dans le temps :

décalages programmés par fuseau

```
<?php
$tz = new DateTimeZone(date_default_timezone_get());
var_dump($tz->getTransitions());
```

```
/* affiche
[0]=>
  array(5) {
    ["ts"]=>
    int(-1855958901)
    ["time"]=>
    string(24) "1911-03-10T23:51:39+0000"
    ["offset"]=>
    int(0)
    ["isdst"]=>
    bool(false)
    ["abbr"]=>
    string(3) "WET"
  }
  [1]=>
  array(5) {
    ["ts"]=>
    int(-1689814800)
    ["time"]=>
    string(24) "1916-06-14T23:00:00+0000"
    ["offset"]=>
    int(3600)
    ["isdst"]=>
    bool(true)
    ["abbr"]=>
    string(4) "WEST"
  }
  il y a 182 entrées comme cela ...
*/
```

4.3. Les opérations sur les dates

Maintenant que nous connaissons bien les objets DateTime et DateTimeZone, voyons de plus près les calculs possibles :

```
<?php
$d = new DateTime();
$d->setDate(2007,10,03);
echo $d->format("d/m/Y à H\hi:s"); // 03/10/2007 à
19h39:53
$d->setTime(2,30);
echo $d->format("d/m/Y à H\hi:s"); // 03/10/2007 à
02h30:00
```

Si l'on ne veut changer que les minutes, ou alors les mois, il faut s'y prendre comme celà :

```
<?php
$d = new DateTime();
$d->setDate($d->format('Y'),10,$d->format('d'));
```

Un peu bizarre, ***setIsoDate()***, comme son nom ne l'indique pas, permet de spécifier [année, no de semaine, jour] :

```
<?php
$d = new DateTime();
$d->setIsoDate(2007,03,05); // année 2007, et 5ème jour
de sa 3ème semaine
echo $d->format(DateTime::W3C); //
2007-01-19T19:46:30+01:00
```

La méthode ***modify()*** va faire un calcul. Comme le constructeur de **DateTime**, elle prend un paramètre accepté par strtotime() (Lien30) :

```
<?php
$d = new DateTime("2008/03/12 15:30");
$d->modify("-3 months +1 hour");
echo $d->format(DateTime::W3C); //
2007-12-12T16:30:00+01:00
$d->modify('2 years ago'); // "il y a 2 ans"
echo $d->format(DateTime::W3C); //
2005-12-12T16:30:00+01:00
```

modify() va modifier l'objet actuel (un peu logique non ?) - Si vous voulez garder une copie de l'ancien objet, clonez-le simplement.

Le fuseau n'a pas bougé, il s'agit de celui par défaut, car je ne l'ai spécifié nulle part. Je rappelle que mon php.ini contient le fuseau par défaut de PHP : Europe/Paris.
Pour ces deux dates-là : le décalage était à ce moment-là de +1.
De la même manière, on peut comparer des dates, mais apparament uniquement depuis PHP 5.2 (à confirmer...) :

```
<?php
$d1 = new DateTime();
$d2 = clone $d1;
$d2->modify("tomorrow");
var_dump($d2 < $d1); // false
```

5. Aller plus loin avec les dates

Cette API des dates, (à partir de PHP5.1) est très pratique, surtout pour la gestion automatique des fuseaux et de leurs décalages. La possibilité de changer une date de fuseau est aussi interéssante : la date en question change bien en fonction du décalage de son fuseau. Qu'en est-t-il de l'intervalle de validité ?
C'est le point d'interrogation. Je peux créer une date de l'année 910 par exemple, et demander le jour de la semaine correspondant, ça fonctionne (du moins ça semble, je n'ai pas vérifié la réalité)
Sur cette même date 'exotique' (car très en arrière, largement sous 1900), le décalage avec mon fuseau est noté à +00:09, cependant, l'offset ne renvoie pas 540 secondes mais 561... ?

```
<?php
$d = new DateTime("0910/11/01"); // l'année est
toujours exprimée sur 4 chiffres
echo $d->format("D"); // Sun -> il s'agirait donc d'un
dimanche ?
echo $d->format(DATE_ISO8601)  //
0910-11-01T00:00:00+0009
echo $d->getOffset(); // 561
```

Pour une date très supérieure à 2038, tout semble OK :

```
<?php
$d = new DateTime("8910/11/01"); // an 8910 !
echo $d->format("D"); // Sat : ce sera un Samedi
echo $d->format(DATE_ISO8601); //
8910-11-01T00:00:00+0100
echo $d->getOffset(); // 3600 : OK
```

L'intervalle semble donc virtuellement infini, mais les dates avant le 10/03/1911 retournent une information de décalage de +00:09 et un offset non pas de 540 mais 561 secondes (sur le fuseau Europe/Paris donc).
Je n'ai pas eu le courage de lire tout Wikipedia pour me renseigner, ni de me pencher plus sur la source de cette fameuse timezonedb, inclue dans PHP.

Quoiqu'il en soit, si on veut aller plus loin avec les calculs sur les dates, je conseille le composant Zend_Date (Lien31), du Zend_Framwork (Lien32)
Cette classe fait abstraction du fuseau (timezone) de PHP, et n'utilise pas les objets DateTime et DateTimezone car actuellement, Zend Framework requiert PHP 5.1.4 pour fonctionner, or le support de ces 2 objets n'est pas inclu systématiquement dans la distribution de PHP5.1, ainsi Zend_Date ne fonctionnerait pas.
Au lieu de cela, Zend_Date se repose sur la bibliothèque BCMath de PHP (inclue), afin de faire des calculs très précis. Zend_Date sait donc faire tout ce que PHP sait faire, et même plus (Lien33)...

Zend_Date se pilote différemment des objets dates de PHP, et gère tout dans un objet (dates, temps, fuseau, et même localisation). Ce composant permet aussi des calculs avancés, et gère l'internationalisation des dates. Voyez plutôt quelques exemples :

```
<?php
require_once 'Zend/Date.php';
$date = new Zend_Date('06/10/2004 11h47', false,
'fr_FR'); // oui carrément écrit en bon Français !

echo $date->toString("GGGG, EEEE dd MMMM yy à
hh:mm"); // après Jésus-Christ, mercredi 06 octobre 04
à 11:47

print_r (Zend_Date_Cities::City('Paris'));
/*
Array(
    [latitude] => 48.8666667
    [longitude] => 2.3333333
    [horizon] =>
)
*/

$date = new Zend_Date('12/18/1882');
echo $date->getIso(); // 1882-12-18T00:00:00+0100

$date->addHour(3); // ajoutons 3 heures
echo $date->get(Zend_Date::TIMES); // 03:00:00
```

L'API de Zend_Date (Lien34) vous en dira plus ;-)

6. Conclusion

La gestion des dates n'est pas une mince affaire. En fait elle n'est pas spécialement difficile non plus, il suffit de savoir comment elle fonctionne en PHP, et de ne pas oublier que toute date est reliée obligatoirement à un fuseau, qui va pouvoir modifier la manière dont la date est perçue, en instaurant des décalages.
Cette notion est bien gérée à partir de PHP5.1, mais a quand même des limites que tentent de repousser les Frameworks objets de plus en plus utilisés pour bâtir des applications complexes.

Retrouvez l'article de Julien Pauli en ligne : Lien35

Bloguer par SMS avec Wordpress - Plugin de micro-blogging avec l'API Orange

Ce tutorial a pour but de créer un plugin wordpress pour le micro-blogging en utilisant les API d'Orange.

1. Introduction

Wordpress est un système de gestion de contenu écrit en PHP et l'un des moteurs de blog les plus populaires. Ce tutorial a pour but de créer un plugin wordpress pour le micro-blogging en utilisant les API d'Orange. Le micro-blogging est une pratique consistant à envoyer des messages courts (moins de 200 caractères) et à partager ce contenu. Twitter est l'application de micro-blogging la plus connue. Dans la suite, nous présenterons rapidement le système de plugin de Wordpress et l'API SMS d'Orange. La seconde partie sera consacrée à la description de la structure d'un plugin Wordpress. Puis, nous détaillerons les quelques fonctions essentielles en PHP du plugin

Comme indiqué précédemment, notre outil de micro-blogging est une extension ou plugin de Wordpress. Il existe une multitude de plugins wordpress, on trouve pas loin de 2000 plugins disponibles sur le site wordpress.org. Il faut souligner que le but de ce tutorial n'est pas de présenter Wordpress mais de combiner sur un

exemple : plugin et web-services. Pour une description précise de Wordpress, on peut se référer au site wordpress.org

L'API SMS d'Orange sera la seconde brique de base de l'outil de micro-blogging. Les API d'Orange ont déjà fait l'objet d'un article (Lien36) auquel je vous invite à vous référer pour davantage de détails sur les API d'Orange (L'auteur de ce tutorial travaille pour Orange et a contribué aux développements des API.). Pour poster de nouveaux articles, le blogueur enverra simplement un SMS à un numéro court. Pour utiliser les API d'Orange, l'utilisateur devra s'enregistrer sur le site d'Orange Partner (Lien37) pour obtenir sa clé d'accès au service et configurer (Lien38) les différents paramètres des API.

Le fonctionnement du plugin est très simple et l'ergonomie minimaliste. Le blogueur envoie un SMS au 20345 (prix d'un SMS 'normal') dont le premier mot sera utilisé uniquement pour l'aiguillage du SMS vers votre boîte mail. Le titre sera délimité par un dièse (#) et le reste du SMS correspond au contenu proprement dit du SMS. Le blogueur enverra le SMS suivant au 20345 pour poster un article (on admet qu'il a configuré 'post' comme mot clé) : 'post nouveau tuto#Je suis en train de terminer un plugin de micro-blogging' ce qui donnera sous Wordpress un article avec comme titre 'nouveau tuto', le corps de l'article 'je suis en train de terminer un plugin de micro-blogging' et la date du post sera l'arrivée du SMS dans la boîte mail.

2. Structure d'un plugin Wordpress

Ce plugin a été testé avec la version 2.3.3 de Wordpress (mais devrait fonctionner avec toute version supérieure à 2.1 de Wordpress) et PHP 5.2.4. Les plugins Wordpress sont des scripts PHP situés dans la répertoire wordpress\wp-content\plugins. Vous pouvez au choix créer un nouveau répertoire ou mettre directement votre script dans le répertoire plugins. Chaque plugin (i.e. le fichier php) commence par un en-tête particulier (indispensable, autrement votre plugin ne sera pas référencé par Wordpress).

En-tête plugin

```
/*
Plugin Name: Post by SMS
Plugin URI: http://www.developpez.com
Description: take the SMS forwarded in your mailbox and post it on your wordpress blog
Author: Francois Marx
Version: 0.3
Author URI: http://www.orangepartner.com
*/
```

Vous vous demandez peut-être maintenant comment votre script va être capable d'intéragir avec Wordpress. La solution est d'utiliser un hook qui permet d'exécuter une fonction (définie par l'utilisateur) lorsque Wordpress effectue certaines actions (ajout d'un post, enregistrement d'un utilisateur...). Le site de wordpress (Lien39) fournit une liste de tous les hooks. Par exemple ce bout de code (extrait du plugin exemple Hello.php disponible dans le répertoire wordpress\wp-content\plugins) affiche les paroles de la chanson 'Hello, Dolly'. Chaque fois que Wordpress charge la page d'adminstration, Wordpress exécute la fonction hello_dolly. Plus précisement, la fonction est exécutée juste avant la fermeture du tag </body> dans la page d'administration. L'assignation du hook à la fonction hello_dolly se fait grâce la fonction add_action avec comme paramètre le nom du hook et la fonction à exécuter.

Plugin hook

```
// This just echoes the chosen line, we'll position it later
function hello_dolly() {
        global $chosen;
        echo "<p id='dolly'>$chosen</p>";
}

// Now we set that function up to execute when the admin_footer action is called
add_action('admin_footer', 'hello_dolly');
```

Il est souvent nécessaire de configurer des options dans les plugins. L'ajout d'option de configuration se fait par l'intermédiaire de la fonction add_options_page qui prend comme argument le titre de la page d'options, le titre du menu, le niveau d'accès minimum de l'utilisateur, le fichier et la fonction qui affiche le contenu de la page d'options.

Définition page d'administration

```
function sms_add_option_pages() {
        if (function_exists('add_options_page')) {
                add_options_page('Post by SMS', 'Post by SMS', 8, __FILE__, 'sms_options_page');
        }
}

function sms_options_page() {
  // code if needed
  // html form if needed
}
```

Après cette brève présentation de l'architecture d'un plugin Wordpress, nous allons entrer dans le vif du sujet en décrivant l'utilisation du plugin et la configuration des API.

3. Principe du plugin et configuration des API

Comme indiqué précédement, le but est de pouvoir poster des messages sur un blog via des SMS. Le principe est le suivant:

- Envoi d'un SMS au 20345 commençant par un mot clé (choisi par l'utilisateur des API d'Orange)
- Le SMS est alors aiguillé vers la boîte mail en api-xxxx@orange.fr de l'utilisateur
- Un timer est activé pour sonder régulièrement la boîte mail et détecter l'arrivée de nouveaux SMS. Nous profitons de l'équivalent 'cron' de Wordpress. Note: il est déconseillé de vérifier l'arrivée de nouveaux messages trop fréquemment.
- Lorsque un SMS arrive, le contenu est extrait et 'posté' comme un article dans Wordpress

Nous allons maintenant configurer les API pour utiliser notre plugin. Tout d'abord, accédez à l'IHM d'administration (Lien40) pour configurer le routage du SMS. Je suppose que vous êtes déjà inscrit sur le site d'orangepartner dans le cas contraire, il faut d'abord s'enregistrer. Dans l'onglet email, notez l'adresse mail qui vous a été attribuée api-xxxx@orange.fr Dans l'onglet SMS, configurez le routage de vos SMS vers votre adresse mail API Orange (api-xxxx@orange.fr).

Tout est maintenant prêt pour développer notre plugin. Bien entendu, je suppose que vous avez installé Wordpress. Si ce n'est pas fait, c'est le moment.

4. Code du plugin

4.1. Principales fonctions du plugin

Les fonctions sms_add_option_pages et sms_options_page permettent de configurer les options. Les options sont :

- L'intervalle de temps entre chaque interrogation de la boîte mail pour vérifier l'arrivée de nouveaux messages
- La clé d'accès aux APIs

Définir une page de configuration est aisé. Nous avons déjà indiqué que la fonction add_options_pages ajoute un onglet

d'options pour votre plugin dans la page options de Wordpress. Le contenu de cet onglet est défini dans une fonction. Il s'agit le plus souvent d'un formulaire pour récupérer les paramètres de configuration de votre plugin. Le code ci-dessous illustre la page d'option du plugin de micro-blogging.

Code HTML pour l'affichage de la page d'options

```
function sms_options_page() {
  // config scheduler
  ?>
        <div class=wrap>
         <h2>Scheduled SMS</h2>
         <form method="post" action="<?php echo
$_SERVER["REQUEST_URI"]; ?>">
    <input type="hidden" name="info_update"
id="info_update" value="true" />
    Number of minutes between SMS checks<br>
          <input name="sms_delay" type="text" size="10"
value="<?php echo get_option('sms_delay'); ?>"/><br>
    API access key<br>
          <input name="sms_api_access_key" type="text"
size="20" value="<?php echo
get_option('sms_api_access_key'); ?>"/>
    <div class="submit">
                  <input type="submit"
name="info_update" value="<?php _e('Update options'); ?
> &raquo;" />
          </div>
         </form>
        </div>
  <?php
}
```

Copie d'écran de la page de configuration du plugin de micro-blogging

Dans la fonction sms_options_pages nous configurons aussi le planificateur avec le code suivant:

Initialisation planificateur code PHP

```
<?php
if (isset($_POST['info_update'])) {
    update_option('sms_api_access_key',(string)
$_POST['sms_api_access_key']);
    update_option('sms_delay', (int)
$_POST['sms_delay']);
    wp_clear_scheduled_hook('sms_check_post');
    wp_schedule_event(time(),'sms_schedule','sms_check_
post');
  }
?>
```

Le planificateur a été introduit dans Wordpress 2.1. A chaque fois qu'une page est chargée, Wordpress compare l'heure courante a une liste de tâches planifiées et exécute les fonctions spécifiées. Il est possible de définir un évènement simple ou périodique. Dans l'exemple choisi, la fonction wp_schedule_event planifie l'exécution de la fonction sms_check_post avec la périodicité sms_schedule. sms_schedule est une pérodicité définie par l'utilisateur. Wordpress définit deux options 'hourly' and 'daily'. Il est possible d'ajouter d'autres périodicités avec le bout de code suivant (la durée de l'intervalle est exprimé en secondes):

periodicity addition

```
function sms_more_reccurences() {
  return array(
    'sms_schedule' => array('interval' =>
60*(int)get_option('sms_delay'), 'display' => 'SMS
Schedule')
  );
}

add_filter('cron_schedules', 'sms_more_reccurences');
```

Le planificateur déclenche l'exécution de la fonction sms_check_post. La fonction sms_check_post récupère la liste des emails dans la boîte mail. Pour récupérer la liste des emails, nous utilisons la méthode getMailList des API d'Orange. Le code ci-dessous récupère la liste des emails et parcourt cette liste:

```
$url_list = "http://mail.alpha.orange-
api.net/mail/getMailList.xml?
id=$api_access_key";
$url_delete = "http://mail.alpha.orange-
api.net/mail/deleteMail.xml?id=$api_access_key";
// get the list of all emails
$response = file_get_contents($url_list);
$xml_list = simplexml_load_string($response);

// loop on all emails
for ($n = count($xml_list->list->message); $n >= 1;
$n--) {
// TODO emails and SMS processing
}
```

Les mails dont le sujet commence par "Sms from" sont considérés comme des articles du blog. On récupère alors le contenu de ce mail à l'aide de la méthode getMail. Le contenu du message est alors 'parsé' pour supprimer le premier mot qui correspond au mot clé nécessaire pour aiguiller le SMS dans votre boîte mail, extraire le titre de l'article et enfin son contenu. La chaîne de caractères comprise entre le premier espace et le premier dièse est considérée comme le titre, le reste du message correspond au contenu proprement dit de l'article. Le message est alors posté sur le blog à l'aide de la fonction wp_insert_post. Cette fonction prend comme paramètre le titre du 'post', le contenu du 'post', la date et le status. L'email est alors supprimé définitement de la boîte mail avec la méthode deleteMail.

Post message

```
$response = file_get_contents($url_content.'&number='.
$n);
$xml_content = simplexml_load_string($response);
$body = $xml_content->message->body;
// remove the first words
$post_title = substr($body,strpos($body,'
'),strpos($body,'#')-strpos($body,' '));
$post_title = $post_title.' (par SMS du '.substr($m-
>subject,9,11).')';
$post_content = substr($body,strpos($body,'#')+1);
$post_date = date('Y-m-d H:i:s',strtotime($xml_content-
>message->date)+get_option('gmt_offset')*3600);
$post_status = 'publish';
$post_data =
compact('post_content','post_title','post_date','post_s
tatus');
$post_ID = wp_insert_post($post_data);
// delete emails
$response = file_get_contents($url_delete.'&number='.
$n);
```

Retrouvez la suite de l'article de François Marx en ligne : Lien41

Créer une liste de texte survolable et cliquable

Une technique émerge de plus en plus du net, c'est la création d'une liste, ayant un contenu plus que conséquent, survolable en utilisant des balises **a**. Ce tutoriel aura donc pour unique but de vous la présenter.

1. Introduction

Depuis peu, il existe une technique, que j'appellerai *blocklist*, est de plus en plus utilisée des concepteurs de sites web. Par *blocklist*, je veux dire une liste de liens, empilés verticalement, avec un contenu assez conséquent. Pour vous donner une idée, Veerle Pieters (Lien42) a gentiment acceptée, à titre d'exemple, de me laisser utiliser la rubrique Approved (Lien43) de son blog.

Je n'ai pas inclus tout le code nécessaire à l'obtention de la version finale, étant donné que c'est assez compliqué. Cependant, vous pouvez voir un exemple (Lien44) d'utilisation de cette technique et jeter un œil au code source. Même s'il est assez détaillé, je vous recommande de lire l'explication qui suit plutôt que de vous précipiter sur la démo; ça vous aidera à vous faire une idée sur le fonctionnement en cas de problèmes ou quand vous voudrez faire des modifications ultérieures.

2. L'approche

Allez voir l'implémentation (Lien43) de Veerle : interagissez avec elle (survolez et cliquez sur les éléments de la liste). N'est-elle pas belle ? Cette large zone cliquable apporte du caractère à ce qui serait qu'un simple titre lié à une description positionnée à sa suite, sans oublier les avantages que cela apporte en termes d'accessibilité pour les personnes à motricité réduite.

Jetons un coup d'oeil au code :

```
<h2>Approved</h2>
<ul>
  <li><a href="#">
  FF Meta Serif<br>
  <em>A gorgeous, all-purpose typeface.</em><br>
  <span> December 11 at 09.47 am</span>
  </a></li>
</ul>
```

Actuellement, la seule manière de rendre le bloc entièrement cliquable est d'entourer chaque élément contenu dans le lien, c'est-à-dire le titre, la description et la date, d'une seule et unique balise **a**.

Comme **a** est un élément *inline*, il ne peut pas contenir de balises de type bloc (*block-level*), ce qui signifie que le titre ne peut être codé avec une entête (**hN**) et qu'il en va de même pour les paragraphes (**p**). Pour personnaliser le *blocklist*, chacun des trois composants a besoin de son propre élément. Avec des choix limités, Veerle a optée pour un élément **em** afin d'y mettre la description, et pour la date, un élément **span** et des balises **br** pour effectuer un retour à la ligne.

Veerle n'est certainement pas la seule à choisir cette approche. J'espère que Elliot Jay Stocks (Lien45) ne me tiendra pas rigueur d'utiliser la rubrique *Recent Reads* de son site comme un autre exemple. En effet, sur tous les sites où j'ai pu trouver une liste similaire, les auteurs ont dû employer des balises **strong**, **em**, **span**, **small** voire même **q** pour marquer le contenu de leurs liens.

3. Quel est le problème ?

Coder la liste de cette manière soulève quelques interrogations. Faut-il vraiment souligner la description par rapport au reste du contenu du lien ? Y a-t-il une quelconque relation entre le titre de l'élément et sa description ? Qu'en est-il de la date ?

Deuxièmement, appliquer un style à la liste a été rendu plus difficile par le type inline des éléments. Des sauts de ligne ont dû être utilisés pour séparer ce qui aurait été une ligne continue de texte avec le CSS désactivé (sans les avantages du **display: block**).

Troisièmement, utiliser un élément comme la balise **em** juste pour *contenir* la description, empêche son utilisation pour du texte qui devrait être mis en valeur. Il en va de même pour les liens. Comment faire si vous souhaitez un lien supplémentaire dans la description ? Vous ne pouvez pas avoir un lien imbriqué.

4. Un regard nouveau sur ces éléments

Quand je développais mon site (Lien46), j'avais prévu d'y intégrer une liste de Quicklinks (Lien47) (liens rapides), c'est-à-dire un ensemble d'éléments externes à mon site renvoyant vers des sujets intéressant sur le Web, essentiellement le même que celui de la rubrique Approved (Lien43) de Veerle. J'ai pensé qu'il devait avoir une meilleure façon de construire une liste avec de larges zones cliquables, et donc je me suis mis à faire des tests.

Pour démarrer, j'ai retranscris la manière dont je voulais voir s'afficher les balises :

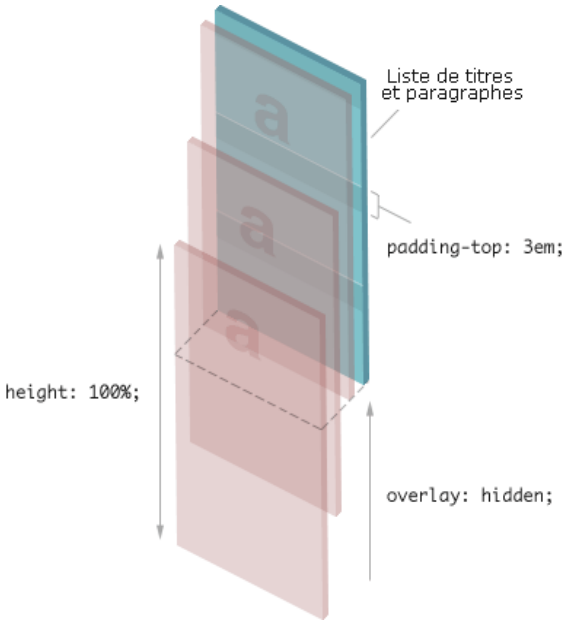
```
<h2>Liens Rapides</h2>
<ul>
    <li>
        <h3><a href="#">Titre du lien rapide</a><h3>
        <p>Une courte description du lien</p>
        <p>1er Janvier</p>
    </li>
</ul>
```

L'astuce de cette technique est que les liens se trouvent uniquement sur les titres. Les éléments **a** deviennent ensuite des éléments de type block (*display: block;*) qui recouvrent les éléments qui les suivent à la même manière qu'un calque transparent, qui rend l'ensemble cliquable.

Je tiens à souligner que tout le texte couvert par la balise étendue **a** ne pourra pas être sélectionné. Si cela vous pose un gros problème alors cette technique ne correspond peut-être pas à ce que vous recherchez.

Pour que le lien soit en haut du contenu qui lui est consécutif sans

tout remettre en bas, il faut le retirer du flux du document (positionnement absolu) et lui attribuer un z-index assez important.



Le plus gros obstacle à surmonter est de prendre en compte un contenu variable ou l'augmentation de la taille du texte. Il serait facile de donner à chaque balise **a** une hauteur fixe et de la laisser telle quelle, mais elle ne a) couvrirait pas l'ensemble du contenu ou b) limiterait la quantité de texte qu'un lien pourrait avoir. Pour surmonter ce problème, la liste en elle-même doit avoir une position relative et chaque élément **a** une hauteur de 100%. Cela permet à chaque lien de s'étendre sur toute la hauteur de la liste. Les éléments qui apparaissent plus bas dans le code source ont à la base un z-index plus important, de ce fait chaque lien recouvre convenablement le précédent, ce qui coupe la zone cliquable depuis son origine (le haut du titre étant lié) jusqu'au point où le lien suivant débute (le haut du prochain titre).

En touche finale, on peut appliquer à la liste l'instruction " *overlay: hidden;* ". Attribuer à chaque lien une hauteur de 100% fait que ces derniers surplombent la liste qui doit se terminer à la hauteur des cases précédentes, cela donne une liste de 200% par rapport à la hauteur de base. Mettre la propriété *overlay* de liste à *hidden* découpe l'ensemble et la rétablit à la bonne hauteur. (Si vous avez du mal à assimiler tout cela, ne vous inquiétez pas. Reprenez le code de la page de démonstration (Lien48) et supprimez l'instruction *overflow: hidden;* pour voir ce que je veux dire).

Une des caractéristiques de la technique dont je ne me satisfais pas est l'obligation d'attribuer une certaine hauteur au titre du lien étant donné qu'il a été supprimé du flux du document. Pour y remédier, un padding d'environ 3em doit être ajouté en haut de l'élément qui suit immédiatement le titre. On fait cela en appliquant une classe à l'élément liste si son titre est susceptible de s'étendre sur deux ou plusieurs lignes et on applique un style à l'élément qui le suit de la manière suivante:

```
.longtitle h3 + p
{
    padding-top: 5em;
}

.verylongtitle h3 + p
{
    padding-top: 7em;
}
```

Si vous pensez qu'il y a une meilleure façon de surmonter cette difficulté, n'hésitez pas à manipuler le code et à me le faire savoir.

Eh bien, nous y sommes. La structure de base derrière cette technique est terminée. Tout ce qui nous reste à faire est de lui donner un style agréable.

5. Astuces pour styliser votre Blocklist

- Utilisez le sélecteur de parenté pour cibler un élément consécutif à un autre (cf le deuxième paragraphe dans chaque élément de la liste : **p + p { color: red; }** ou, comme nous avons pu l'utiliser, la 'description' dans chaque lien : **h3 + p { padding-top: 3em; }**). Ces sélecteurs de parenté sont puissants, jusqu'à présent ils sont peu utilisés du fait de leur compatibilité plus que limité par les navigateurs (je compte sur vous Microsoft).
- Pour appliquer un style à un lien lorsqu'il est survolé il faut appliquer la pseudo-class **:hover** à l'élément de la liste (**li**), pas au **a** lui-même. Par exemple, **li:hover { background-color: green; }**. Si vous appliquez **:hover** au **a**, vous ne serez pas en mesure de changer l'arrière-plan par crainte de masquer tout ce qui se trouve en dessous, et le style n'affectera pas le reste du contenu du lien.
- Si vous envisagez de mettre des liens supplémentaires dans les éléments regroupés dans **a** (un des avantages de cette technique), stylisez-les pour qu'ils apparaissent comme du texte normal étant donné qu'**ils ne seront pas cliquables**. N'importe quel utilisateur qui lira votre contenu avec le CSS désactivé ou dans un lecteur de flux RSS continuera à bénéficier de leur mise en forme mais je n'ai trouvé aucun moyen pour qu'ils restent cliquables par l'intermédiaire du lien qui les recouvre. Encore une fois, toutes suggestions relatives à une solution sur ce problème serait grandement appréciée dans vos commentaires.
- Amusez-vous avec cette technique ! Elle vous donne bien plus de flexibilité comparée à l'ancienne méthode et vous laisse un balisage propre pour styliser le tout à votre guise.

6. Le Bon, le Méchant et IE

Eh bien, voici un petit tour d'horizon des avantages et inconvénients de cette technique :

	Avantages	Inconvénients
Ancienne Méthode	Grande zone cliquable sous IE6	Difficile à styliser Oblige à utiliser les retours à la ligne Exploite de façon incorrecte des éléments qui pourraient être mieux utilisés
Nouvelle Méthode	Propre, balisage sémantique Ne vous restreint pas à une utilisation des éléments inline Pas de balisage superflu Peut contenir des liens supplémentaires	La hauteur approximative du titre doit être connue A part le titre, le texte n'est pas sélectionnable Petite zone cliquable sous IE6

Comme vous pouvez le constater, c'est loin d'être parfait mais les avantages sont notables en comparaison de la méthode traditionnelle. Votre *blocklist* ne fonctionnera pas aussi bien sous IE6. Cependant n'ayez crainte, tel quel (en utilisant le code de la démonstration (Lien49)) le *blocklist* devrait s'afficher parfaitement, avec seulement des zones cliquables réduites à la taille de leurs titres.

L'un des avantages liés à l'utilisation de sélecteurs de parenté pour styliser le *blocklist* est qu'IE6 ignore la plupart des choses sur lesquelles il s'arrête habituellement. Bien sûr, les résultats peuvent varier si vous personnalisez le code en fonction de vos besoins, mais avec une ligne ou deux d'instructions CSS spécifiques cela devrait suffire à IE pour résoudre n'importe quel problème. Par défaut, IE7 le gère plutôt bien.

Et voilà ! Encore une fois, je tiens à remercier Veerle (Lien42) pour m'avoir permis d'utiliser son site à titre d'exemple. Assurez-vous d'avoir regardé la démonstration pour voir la technique en action. Elle inclut deux *blocklists* : l'un dans le même style que mes Quicklinks et l'autre, qui reprend à titre de comparaison la toute récente technique de Veerle dans sa rubrique Approved. Cependant, gardez à l'esprit que n'importe quel CSS fourni au-delà de la structure de base est seulement à titre de démonstration et ne doit pas être recopié aveuglément. J'espère que les styles que j'ai pu inclure vous donneront de quoi démarrer les vôtres.

Si vous avez un quelconque avis sur cette technique, positif ou négatif, n'hésitez pas à me le faire savoir.

Retrouvez l'article de Sam Rayner en ligne : Lien50

Créer sa propre info-bulle en CSS

Ce tutoriel a pour but de vous donner une technique pour créer vos propres info-bulles en utilisant que du CSS.

1. Introduction

Dans nos développements Web, il se peut que nous voulions afficher un bloc d'information au survol d'un lien. Malheureusement, l'attribut title de l'élément **<a>** ne permet que d'afficher quelques mots et rien d'autres.

Suite à l'insuffisance de cette balise, je vous propose une méthode purement CSS (Lien51) qui nous permettra non seulement d'afficher un paragraphe mais également n'importe quel élément du moment qu'il soit un *enfant* de **<a>** (par exemple : une image, un champ texte (input), *etc.*). Nous aurons ainsi une info-bulle personnalisée.

Avant de commencer, je vous suggère de jeter un oeil à cet outil (Lien52) qui vous permettra de connaitre les balises *enfants* que la balise <a> (Lien53) peut contenir. Ceci vous donnera une idée des possibilités de contenu de vos bulles d'informations.

Dans ce tutoriel nous verrons deux méthodes. La première consistera à créer une info-bulle avec une simple couleur comme arrière-plan, la deuxième, elle, contiendra une image.

Mais avant toutes choses, je tiens à signaler que cette astuce n'est faîte que pour contenir du texte. L'insertion d'un nouveau lien **a** n'est pas possible.

2. Info-bulle avec fond coloré

Au cours de ce chapitre nous allons tenter de réaliser une simple info-bulle qui aura un fond coloré et l'apparence de l'image qui suit.

Info-bulle avec fond coloré

2.1. Partie (X)HTML

Nous utiliserons, ici, un élément de type **span**, pour ajouter notre texte. On pourra donc avoir un code du style :

Code (X)HTML de notre bulle d'information textuelle

```
<a class="info_bulle" href="http://developpez.com">
  Developpez.com
  <span>
    www.developez.com est la communauté en langue
française qui concentre le
    plus de développeurs professionnels avec plus de
130 000 visites par jour.
  </span>
</a>
```

Avec ce code, nous avons un texte sur la même ligne et sans effet de survol, ce qui est tout à fait normal vu que la balise **** est de type **inline** (c'est-à-dire que son positionnement se fait sur la même ligne).

La présence de **class="info_bulle"** est indispensable pour la suite car c'est grâce à elle que nos effets de style seront appliqués à notre lien.

Pour le moment, nous n'avons pas encore notre info-bulle. Voir le résultat (Lien54) en ligne.

2.2. Partie CSS

Cette partie est consacrée à la mise en oeuvre du code de personnalisation des balises qui constituent notre bulle d'informations.

Avant toutes choses, il nous faut rendre invisible la balise qui contiendra l'information à transmettre pour n'obtenir que notre lien.

Code CSS rendant invisible notre texte

```
a.info_bulle span
{
  display : none; /* Rend invisible tout notre bloc
span */
}
```

Maintenant, nous pouvons personnaliser, aisément, notre lien en lui donnant une apparrence permettant de communiquer aux lecteurs qu'ils auront un supplément d'informations en le survolant. Par exemple, on peut lui attribuer une image représentant une bulle en guise de fond.

Code CSS de personnalisation de notre lien

```
a.info_bulle
{
  color           : #2F368A; /* Couleur de notre lien
*/
  font-size       : 1.2em;   /* Taille de la police */

  text-decoration : none;    /* Aucun soulignement du
texte */
```

```
  padding         : 2px 12px 2px 2px; /*Définition des
marges intérieures de notre lien */

  /* Définition de l'arrière plan de notre lien */
  background      : transparent url('comment.gif') no-
repeat right center;
}
```

Une fois ces deux étapes réalisées, il ne nous reste plus qu'à ajouter le code qui permettra d'afficher notre info-bulle au survol.

Code CSS d'affichage de notre info-bulle

```
a.info_bulle:hover span
{
  display         :  block;    /* Rend visible notre
bloc span */
  position        :  absolute; /* Sort notre bloc de son
conteneur afin de le positionner */

  background      : #DDEEFF;   /* Définition du fond,
sinon on a le même que notre lien */
  border          : 1px solid #6699FF; /* Définition des
bordures */

  padding         : 6px;       /* Définition des marges
intérieures */
  font-size       : 12px;

  width           : 180px;     /* On fixe une largeur
par défaut */

  color           : #000;      /* Réinitialisation de la
couleur du texte */

  text-align      : justify;   /* Justification du texte
*/

  cursor          : default;   /* Réinitialisation de
notre curseur, sinon par défaut on a le même que notre
lien */

  /* Positionnement de notre info-bulle */
  top             : 1.8em;
  left            : 1px;
}
```

Malheureusement, avec ce code nous constatons que Internet Explorer 6 (et inférieur) n'affiche pas notre info-bulle contrairement à Firefox, Internet Explorer 7, Opéra, Safari, *etc.*. Par conséquent, nous devons ajouter la ligne suivante :

Code CSS qui corrige le bug d'IE6

```
a.info_bulle:hover
{
  border : 0;  /* ligne qui corrige le bug d'IE6 et
inférieur */
}
```

Tel que le code est présenté actuellement, nous aurons un affichage incorrect de notre info-bulle. En effet, elle sera positionnée en bas à droite de notre page et non en dessous de notre lien. Pour y remédier, nous devons ajouter une propriété supplémentaire au code CSS de notre lien.

Code CSS pour le bon positionnement de notre info-bulle

```
position : relative; /* Indispensable pour le bon
positionnement de l'info-bulle */
```

Avec le code qui vous est présenté jusqu'à maintenant, il se peut que vous fassiez face à un mauvais affichage de l'info-bulle en présence d'une animation flash, par exemple. Dans ce cas, il vous faut rajouter une nouvelle propriété à son code CSS.

Code CSS permettant la mise au premier plan de notre info-bulle

```
z-index  : 1000; /* Positionne au premier plan l'info-
bulle en cas de chevauchement */
```

Visualisation de notre info-bulle finale (Lien55)

2.3. Exemple d'utilisation

Nous avons, enfin, notre bulle d'informations CSS. Il ne nous reste plus qu'à voir le résultat avec un exemple concret.

Code (X)HTML d'exemple d'utilisation dans un texte

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="fr" lang="fr">
  <head>
        <meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1" />
        <title>Info-bulle personnalisée</title>
        <link rel="stylesheet" href="style.css"
type="text/css" />
  </head>
  <body>
        <h3>Exemple d'info-bulle textuelle dans un
paragraphe</h3>
    <p>
        Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim
        veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea
        commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate
        velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint
        occaecat cupidatat non proident, sunt in culpa
qui officia deserunt
        mollit anim id est laborum.
    </p>
    <p>
        Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim
        veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea
        commodo consequat.
        <a id="info_bulle" class="info_bulle"
href="http://www.developpez.com/">developpez.com
           <span>
              www.developez.com est la communauté en
langue française qui concentre le
              plus de développeurs professionnels avec
plus de 130 000 visites par jour.
           </span>
        </a>
        Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore
        eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in
        culpa qui officia deserunt mollit anim id est
laborum.
    </p>
    <p>
        Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim
        veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea
```

```
        commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate
        velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint
        occaecat cupidatat non proident, sunt in culpa
qui officia deserunt
        mollit anim id est laborum.
    </p>
  </body>
</html>
```

Code CSS de notre info-bulle

```
p
{
  font-size       : 1.1em; /* Définition de la taille
de la police de nos paragraphes */
}

a.info_bulle span
{
  display          : none; /* Rend invisible tout notre
bloc span */
}

a.info_bulle
{
  color           : #2F368A;

  text-decoration : none;

  padding         : 2px 16px 2px 2px; /*Définition des
marges intérieures de notre lien */

  /* Définition de l'arrière plan de notre lien */
  background      : transparent url('comment.gif') no-
repeat right center;

  position        : relative; /* Indispensable pour le
bon positionnement de l'info-bulle */
}

a.info_bulle:hover
{
  border          : 0;  /* ligne qui corrige le bug
d'IE6 et inférieur */
}

a.info_bulle:hover span
{
  display        :  block;    /* Rend visible notre
bloc span */
  position       :  absolute; /* Sort notre bloc de son
conteneur afin de le positionner */

  background     : #DDEEFF;   /* Définition du fond,
sinon on a le même que notre lien */
  border         : 1px solid #6699FF; /* Définition des
bordures */

  padding        : 6px;       /* Définition des marges
intérieures */
  font-size      : 12px;

  width          : 180px;     /* On fixe une largeur
par défaut */

  color          : #000;      /* Réinitialisation de la
couleur du texte */

  text-align     : justify;   /* Justification du texte
*/

  cursor         : default;   /* Réinitialisation de
notre curseur, sinon par défaut on a le même que notre
lien */

  /* Positionnement de notre info-bulle */
  top            : 1.8em;
  left           : 1px;

  z-index        : 1000; /* Positionne au premier plan
l'info-bulle en cas de chevauchement */
}
```

Visualisation de notre info-bulle dans un environnement textuel (Lien56)

3. Info-bulle avec fond imagé

Ici nous allons tenter de réaliser une info-bulle qui aura comme arrière plan une image. Voici un petit aperçu du résultat final

Info-bulle avec image comme arrière plan

3.1. Partie (X)HTML

Etant donné que la seule différence avec l'info-bulle précédente est son arrière-plan, nous aurons, par conséquent, pratiquement le même code.

Code (X)HTML de notre info-bulle

```
<a class="info_bulle" href="http://developpez.com">
  Developpez.com
  <span class="info_bulle">
    <span class="header"></span>
    <span class="content">
      www.developez.com est la communauté en langue
française qui concentre le
      plus de développeurs professionnels avec plus de
130 000 visites par jour.
    </span>
    <span class="footer"></span>
  </span>
</a>
```

On remarque l'apparition de trois nouvelles balises ****. C'est grâce à elles que nous allons construire notre arrière-plan, à partir d'images préalablement découpées. Vous me demanderez certainement pourquoi utiliser des balises ****. Tout simplement parce que c'est l'une des balises enfants de **<a>** et que c'est une balise adaptée à ce genre de création.

3.2. Partie CSS

Nous avons vu précédemment l'apparition de nouvelles balises qui consituent l'arrière-plan de notre info-bulle. Au cours de ce chapitre, nous allons utiliser le même code CSS que celui de la partie II-B tout en introduisant le CSS des nouveaux éléments :

- le *header* sera l'élément qui représentera la partie haute de notre info-bulle ;

Code CSS de notre header

```
span.header
{
   display         : block;
   height          : 35px;  /* Hauteur correspondant à
celle de notre image */
   line-height     : 220%;  /* Propriété qui centrera
le texte verticalement */
   text-align      : center;
```

```
   background      : transparent url('./top.gif') no-
repeat 0 0;
   font-size       : 15px;
   font-weight     : bold;
}
```

- le *content* qui contiendra tout le texte de notre info-bulle ;

Code CSS de notre content

```
span.content
{
   display     : block;
   background  : transparent url('./centre.gif')
repeat-y;
   padding     : 0 8px;
}
```

Nous sommes obligés de mettre un **repeat-y** afin d'avoir une répétition verticale de notre image. Dans le cas contraire nous n'aurions qu'une partie qui la contiendrait.

- et le *footer* qui, comme son nom l'indique, sera sa partie basse.

Code CSS de notre footer

```
span.footer
{
   display     : block;
   height      : 5px;
   background  : url('./bot.gif') no-repeat bottom
left;
   font-size   : 0; /* Corrige l'espacement inutile
sous IE */
}
```

Avec un tel code, nous n'aurons pas encore notre info-bulle au survol du lien. Pour y remedier, il faut rajouter un petit élément au code CSS de la balise **** principale. En effet, vous avez du remarquer l'apparition d'un **class="info_bulle"** dans le code (X)HTML de cette balise. Il ne reste donc plus qu'à définir cette classe.

Ainsi le code initial devient donc le suivant :

Code CSS de notre info-bulle

```
a.info_bulle span.info_bulle
{
  display        : none; /* Rend invisible tout notre
bloc span */
}

a.info_bulle:hover span.info_bulle
{
  display        :  block;    /* Rend visible notre
bloc span */
  position       :  absolute; /* Sort notre bloc de son
conteneur afin de le positionner */

  font-size      : 12px;

  width          : 220px;     /* On fixe une largeur
par défaut */

  color          : #000;      /* Réinitialisation de la
couleur du texte */

  text-align     : justify;   /* Justification du texte
*/

  cursor         : default;   /* Réinitialisation de
notre curseur, sinon par défaut on a le même que notre
lien */

  /* Positionnement de notre info-bulle */
  top            : 1.8em;
  left           : 1px;

  z-index        : 1000; /* Positionne au premier plan
l'info-bulle en cas de chevauchement */
}
```

On obtient, enfin, notre nouvelle info-bulle : voir le résultat (Lien57)

Retrouvez la suite de l'article de Rodrigue Hunel en ligne : Lien58

Développer du code performant et de qualité avec Visual Studio 2008

Visual Studio est, depuis plusieurs années, l'outil numéro un pour les développeurs souhaitant utilisant la plateforme .NET ainsi que les différentes solutions Microsoft. Cette nouvelle version de l'environnement de développement reste la référence en la matière tant elle offre des nouvelles fonctionnalités. Cette fois, ce n'est pas seulement le développeur qui y trouvera son compte puisque Visual Studio 2008 propose un nombre important de nouveautés pour les architectes, les développeurs de bases de données ou encore les chefs de projets par l'intermédiaire de Team Foundation Server, l'outil de centralisation des sources et de compilation de la plateforme Team System.

1. Visual Studio 2008, l'environnement de développement tout terrain

1.1. Un seul outil pour plusieurs versions du Framework .NET et d'Office

Visual Studio 2008 permet de développer des applications ayant pour cible les différentes versions de .NET sorties après octobre 2005 (c'est-à-dire les versions 2.0, 3.0 et 3.5) et non plus une seule version comme ses prédécesseurs.

Qu'en est-il du développement d'add-ins pour la suite Office ? Il y a également du neuf vu que, par défaut, Visual Studio inclus Visual Studio Tools for Office et nous propose donc des projets de types add-in pour les versions 2003 et 2007 d'Office.

1.2. Des designers parfaitement intégrés

Désormais, il est encore plus aisé de développer des applications utilisant une interface graphique basée sur Windows Presentation Foundation (WPF) et de mettre en place de la communication entre différentes applications à l'aide de Windows Communication Foundation (WCF).

Les designers pour Windows Workflow Foundation (WF) se retrouvent également complètement intégrés à Visual Studio 2008, ce qui réjouira bon nombre de développeurs dont les nombreux développeurs SharePoint.

Vous aurez peut-être noté que ces designers étaient déjà présents dans Visual Studio 2005 sous la forme d'extensions. C'est effectivement vrai et, bien que ce ne soit pas fondé, un certain nombre de développeurs avait des craintes autour de l'installation de ces extensions, cette installation étant vue par certains comme du bricolage. Ces craintes peuvent désormais s'estomper et même disparaître : les designers font bel et bien partie intégrante de Visual Studio 2008.

1.3. Ajoutez du dynamisme dans vos applications Web

Inclus dans le Framework .NET 3.5 et disponible en tant qu'extension pour la version 2.0 du Framework, ASP.NET AJAX est un Framework s'intégrant parfaitement avec ASP.NET. ASP.NET AJAX permet d'implémenter facilement des contrôles utilisant JavaScript et permettant d'effectuer des requêtes vers le serveur de manière asynchrone, c'est-à-dire sans recharger la page web ni la bloquer durant l'exécution de ces requêtes. Mais cela ne s'arrête pas là : ASP.NET AJAX propose des composants permettant d'étendre les possibilités de plusieurs composants ASP.NET classiques et d'y ajouter des fonctionnalités du côté client.

Visual Studio supporte totalement ces différents contrôles.

1.4. De l'aide pour le développement d'applications Web utilisant du JavaScript

Avec l'utilisation en pleine croissance du JavaScript, tout environnement digne de ce nom se doit de posséder les outils permettant le développement avec du JavaScript. Visual Studio 2008 propose de nouveaux outils, permettant ainsi de regrouper tous les outils nécessaires au sein d'un même environnement. On retrouve ainsi véritablement de l'IntelliSense pour JavaScript, un vérificateur syntaxique et la possibilité de déboguer le JavaScript de manière simple et efficace directement dans Visual Studio.

Il ne nous reste plus qu'à utiliser abondamment ces outils pour rendre nos applications web encore plus attrayantes.

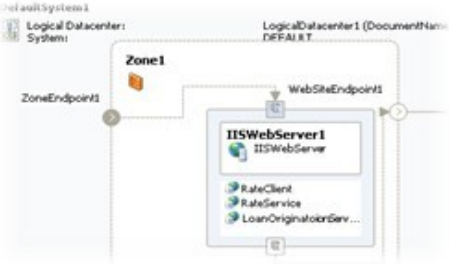
2. La gestion du cycle de vie d'une application

Les besoins et les exigences des utilisateurs grandissent jour après jour. Ainsi, gérer le cycle de vie d'une application devient une nécessité dans les développements d'aujourd'hui.

En utilisant Visual Studio et Team Foundation Server, la gestion du cycle de l'application fait partie du processus de développement, ce qui assure une maintenance plus aisée.

2.1. Quoi de mieux que des designers pour dessiner l'architecture des applications ?

Visual Studio 2008 nous permet de réaliser une approche " top-down ". Les architectes peuvent réaliser facilement le design d'applications, que ce soit du point de vue logique ou technique. Les architectes peuvent également utiliser un designer pour spécifier comment doit se faire le déploiement d'une application.



2.2. Une méthode donne-t-elle le résultat attendu ?

Lors du développement, une des premières choses qu'importe un développeur est d'écrire du code qui réponde à des spécifications techniques et qui permette donc de réaliser les opérations indiquées au sein de ces spécifications. Ainsi, pour arriver à ses fins, le développeur peut créer des tests et développer la méthode

jusqu'à ce qu'elle réponde entièrement aux différents cas présents dans les documents d'analyse grâce à l'utilisation de tests dit " unitaires ", c'est-à-dire ayant pour but de tester un cas particulier. Par ailleurs, par cette technique, nous pouvons vérifier que notre méthode fonctionnera toujours même si nous modifions d'autres méthodes ou classes.

2.2.1. Les tests unitaires

La première étape lors du développement d'une méthode, ou d'une fonction, en dehors de l'analyse et du développement proprement dit, est de vérifier que cette méthode répond aux spécifications. Il nous faudra donc tester les différents cas qui sont censés être gérés au sein de cette méthode.

Visual Studio nous permet, au travers des différents menus, qu'ils soient contextuels ou non, et des différents fenêtres mises à notre disposition, de créer, de gérer et de lancer les tests unitaires.

Le Framework utilisé par Visual Studio a été également fortement amélioré. Un test unitaire peut désormais hériter d'un autre test unitaire, ce qui permet, entre autres, de réaliser les différentes initialisations dans une classe de base et évite les doublons de code pour les tests unitaires.

Les tests unitaires peuvent également être appliqués à du code T-SQL.

Parfait, nous avons créé des tests unitaires et ceux-ci sont tous au vert. Enfin une application sans bugs, sans comportements non contrôlés et qui donc satisferont pleinement l'utilisateur final ! Mais… est-on sur de cela ? Non bien sûr, effectivement, nous venons de tester des morceaux d'application, mais quel est le pourcentage de code qui est réellement couverte par les tests unitaires? Pour le savoir, Visual Studio nous propose une fonctionnalité appelée " couverture de code ". Cet outil nous donne ce pourcentage méthode par méthode. Enfin, pour identifier les lignes de code qui ont été testées et celles qui ne l'ont pas été, cette fonctionnalité surligne chacune de nos lignes de code dans une couleur différente selon si un test couvre ou non celle-ci.

2.3. Le code est-il de bonne qualité ?

Maintenant, nous avons du code qui répond aux attentes vis-à-vis des opérations qu'il effectue. Cependant, est-il simple à maintenir ?

2.3.1. L'analyse du code

L'analyse du code permet d'identifier les erreurs de codage et les vulnérabilités au niveau de la sécurité. Cette analyse utilise la source de notre code afin de vérifier que nous respectons les différentes guidelines. Dans le cas contraire, nous recevons un avertissement dans la fenêtre " Liste des erreurs " si bien connue des utilisateurs de Visual Studio 2005.

De quel type sont les messages que l'analyseur nous renvoie ? Il vérifie par exemple qu'une librairie de classes soit bel et bien signée, que les variables sont bien initialisées avant leur utilisation et bien d'autres choses encore.

Par rapport à son prédécesseur, Visual Studio 2008 propose plus de 20 nouvelles règles dont principalement des règles autour des conventions de nommage et de l'orthographe, ce qui est très utile étant donné que parfois, les fautes d'orthographe dans une application peuvent provoquer un retour disproportionné de la part des utilisateurs par rapport à ces fautes. Pour éviter de recevoir des messages au sujet de l'orthographe des termes liés au métier auquel l'application est destinée, il vous est possible d'utiliser un dictionnaire personnel.

Mais le code d'une application s'arrête-t-il à la partie .NET de celle-ci ? Bien sur que non. D'ailleurs, souvent, le rôle de développeur .NET et de développeur SQL coïncide. C'est pourquoi ce type d'analyse s'applique également sur du code T-SQL. Ainsi, si nous avons utilisé un " SELECT * " dans une requête, l'analyseur nous indiquera qu'il est préférable de donner le nom des colonnes à retourner. Il sera ensuite très simple de réaliser la modification en utilisant du refactoring sur le code T-SQL.

Enfin, on notera une véritable évolution du point de vue de la gestion de ces messages puisque la suppression des messages est bien plus efficace qu'auparavant. Il est par ailleurs possible d'indiquer que les messages nous indiquant des améliorations possibles dans du code généré ne soient pas pris en compte.

2.3.2. Les métriques de code

Maintenir du code peut s'avérer une tâche ardue si les bonnes pratiques n'ont pas été mises en place. L'analyse statique du code ne permet pas de vérifier si les bonnes pratiques du point de vue architectural (faible cohésion entre les objets, …) sont respectées ou non.

Ainsi, sur base de valeurs telles que le nombre d'objets liés à une classe et le nombre de lignes présentes dans une méthode, un indice de maintenabilité est calculé. Le développeur peut ainsi aisément identifier les zones de code à refactoriser.

2.4. Et les performances dans tout cela ?

Grâce aux tests unitaires, à la couverture du code, aux métriques et à l'analyse du code, notre application est bien plus facile à maintenir et à modifier qu'auparavant. Mais, par quoi l'utilisateur final est-il le plus intéressé ? Du code facile à maintenir ou une application qui réponde à ses besoins, simple à utiliser et performante ?

2.4.1. L'analyse de performance

Un des critères important pour les utilisateurs d'une application est la vitesse d'exécution de celle-ci.

Ainsi, il est intéressant de réaliser des tests de performance durant le processus de développement. Identifier un goulot d'étranglement dans le code et cela avec des chiffres obtenus ligne par ligne permet de mettre le doigt sur une méthode qui rendra l'utilisation insatisfait et qui demande donc du refactoring.

Afin d'identifier ce qui s'avère être la meilleure solution, Il est également important de pouvoir comparer deux ensembles de résultats de tests de performance. Cela semble anodin mais il s'agit d'une des nouveautés les plus appréciées dans cette nouvelle version.

Le choix des résultats à comparer se doit d'être simple afin de permettre d'obtenir des résultats de comparaison réellement utiles. Cela est possible grâce à l'utilisation de filtres basés sur le timestamp ou le processus, entre autres.

2.5. L'application répond-t-elle, dans sa globalité, aux attentes techniques et fonctionnelles ?

2.5.1. Les tests de charge

Comment notre application va-t-elle se comporter si plusieurs centaines ou plusieurs milliers de personne s'y connectent simultanément ?

Souvent, les tests sur les applications sont effectués par quelques personnes maximum et l'application ne subit pas le stress qu'elle subira le jour de la mise en production. Or, nous savons tous qu'une application qui n'est pas accessible le jour de sa sortie ne reçoit que difficilement pleine confiance des utilisateurs qui doivent s'y connecter.

Ainsi, il est important de réaliser des simulations réalistes basées sur différents critères comme la durée du test, le nombre d'utilisateurs et bien d'autres critères.

Enfin, si le nombre d'utilisateurs simulés est insuffisant, il nous est possible d'utiliser des agents à installer sur d'autres ordinateurs qui simuleront des utilisateurs supplémentaires.

2.5.2. Les tests d'applications Web

Pour tester nos applications web, nous allons utiliser la notion de scénario. Pour réaliser ces scénarios, rien de plus simple, il suffit d'enregistrer ceux-ci dans Internet Explorer grâce au plugin Test Recorder. Cette nouvelle version du plugin est capable de capturer le trafic généré par les requêtes effectuées à l'aide de XMLHttpRequest (et donc ce qu'il est communément appelé " AJAX "). Test Recorder peut également gérer les paramètres dynamiques qui peuvent être utilisés dans notre application.

Ces scénarios peuvent évoluer au fur et à mesure que le développement avance. Dès lors, le refactoring des tests d'applications s'avère d'une grande utilité. Ce refactoring permet d'extraire des sous-tests qui peuvent être exécutés seuls ou être liés entre eux. Grâce à cette possibilité de faire appel à d'autres tests, il est aisé de créer de nouveaux scénarios.

2.5.3. XML, le format pour le stockage d'informations concernant des tests

Les métadonnées et les résultats sont stockés dans des fichiers XML. Grâce à cela, vous pouvez manipuler ces fichiers à la main ou par programmation. Il est par ailleurs aisé de partager des fichiers contenant les définitions de tests, par l'intermédiaire de Team Foundation Server par exemple.

Toutes les définitions de tests doivent respecter un schéma qui est décrit dans le fichier TestTypes.xsd. En respectant ce schéma, nous pouvons créer de nouvelles définitions de tests qui pourront être exécutés par Visual Studio.

2.6. Travailler en équipe

Dès que vous pensez au mot " équipe ", pensez à Team Foundation Server. C'est lui qui est l'élément central de votre équipe permettant de partager des informations et du code source.

2.6.1. Le contrôle des sources partagées par un serveur de source

Eviter tout conflit et faciliter la manipulation des sources sur le serveur, voici ce qui ressort des nouvelles fonctionnalités et menus présents dans Visual Studio. Ainsi, il est par exemple possible d'automatiser le téléchargement de la dernière version lors de chaque " check-out ", ce qui assure d'éviter la perte de modifications réalisées par un membre de l'équipe sur ce même fichier.

2.6.2. Les annotations de fichiers

Plusieurs personnes peuvent intervenir sur un même fichier, à des moments différents. Ainsi, bien que l'on puisse forcer la déclaration du lien entre des fichiers et un Work Item lors de l'envoi des fichiers sources sur le serveur (check-in), comment pouvons-nous savoir qui a fait quelle modification ?

Tout simplement en utilisant des annotations. Celles-ci nous donnent la date et le nom du développeur qui a fait des modifications, bloc de lignes par bloc de lignes. Ces informations proviennent directement de Team Foundation Server puisque, à nouveau, c'est lui qui centralise ce type d'informations.

Cette fonctionnalité était disponible au sein de Visual Studio 2005 sous la forme d'un Power Tool.

2.6.3. La comparaison de fichiers ou de répertoires

A nouveau présent en tant que Power Tool dans la version précédente de Visual Studio, la comparaison de répertoires permet d'identifier rapidement les différences entre le contenu des répertoires ou des fichiers.

Cette comparaison peut être réalisée entre des répertoires ne se trouvant pas sur la même machine.

2.7. Visual Studio pour les professionnels des bases de données

Au-delà des tests et analyses présentés précédemment et applicables au T-SQL, nous pouvons noter plusieurs améliorations particulièrement utiles.

2.7.1. Le refactoring du T-SQL

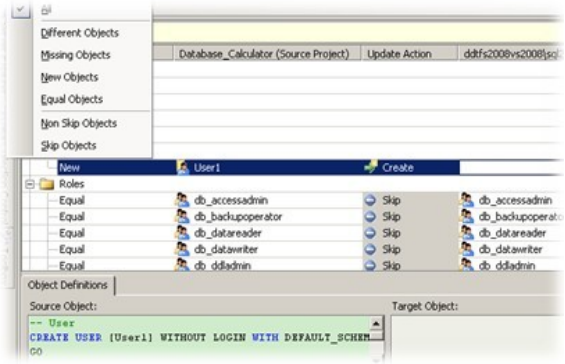
Nous utiliserons la " Wildcard Expansion " pour remplacer un " SELECT * " par un SELECT avec les noms des colonnes. Notez que ce refactoring est capable de gérer les alias, évitant ainsi toute ambigüité et ainsi toute erreur dans les requêtes.

On notera également que lors du refactoring de la procédure stockée, tout DataSet utilisant cette procédure pour son chargement est automatiquement refactorisé.

2.7.2. La comparaison de schémas ou de données entre deux bases

Cet ensemble de fenêtres permet non seulement de réaliser des comparaisons entre des données ou des schémas de bases de données mais également de générer les scripts permettant de synchroniser deux bases de données.

Les scripts générés peuvent être également sauvegardés pour une exécution ultérieure. Cette fonctionnalité est très utile pour faciliter les mises en production, que ce soit vis-à-vis des changements sur le schéma ou des configurations stockées dans une table par exemple. Nous sommes évidemment libres de modifier ces scripts si le besoin s'en fait sentir.



3. Visual Studio Team Foundation Server

3.1. Des performances en nette évolution

Team Foundation Server 2008 est beaucoup moins gourmand en ressources que son prédécesseur. Cela sous-entend que plus d'utilisateurs peuvent accéder à Team Foundation Server simultanément pour accéder à un nombre grandissant de projets et ne bloquant pas pour autant les compilations et tests en cours.

3.2. Microsoft Office SharePoint Server 2007, distant ou non, pour la gestion des documents

MOSS 2007 n'est plus à présenter tant il est devenu une des références dans le monde de la gestion documentaire.

Team Foundation Server 2008 utilise pleinement les possibilités de Windows SharePoint Services 3.0 et de Microsoft Office SharePoint Server 2007. SharePoint peut être installé localement ou sur un autre serveur.

3.3. Check-in policies

Sauvegarder ses sources sur un serveur central et les partager entre différentes personnes est nécessaire pour un développement efficace en équipe. Cependant, pour cela, il faut encore que les sources mises en commun soient de qualité sans quoi, le travail de l'équipe s'en voit fort contrarié.

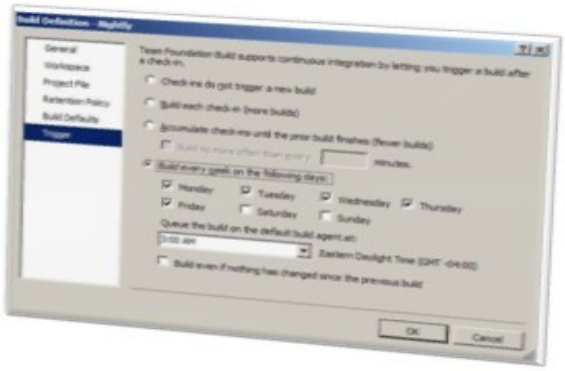
C'est sur ce constat que se base la notion de check-in policies. Effectivement, à chaque envoi des sources sur le serveur, c'est-à-dire à chaque check-in, Team Foundation Server vérifiera que nos sources respectent bien certaines règles. Ainsi, il est fréquent d'obliger le développeur à associer un changement dans ses sources à un Work Item, facilitant ainsi la maintenance.

D'autres règles peuvent être établies comme par exemple vérifier que le code est bien couvert entièrement à des tests unitaires ou que l'analyse de code ne renvoie pas tel ou tel type d'avertissements.

3.4. Faire de l'intégration continue sur base des déclencheurs dans Team Foundation Server

Sur base de déclencheurs, une compilation (et tous les tests qui en dépendent) peut être lancée à chaque check-in avec ou non un délai d'attente indiquant le délai minimum entre deux compilations. La compilation périodique reste toujours possible.

Notons également que la gestion, la configuration et la planification des compilations peuvent se faire directement dans Visual Studio, amenant de la convivialité pour l'utilisateur.



Retrouvez l'article de Didier Danse en ligne : Lien59

Les animations avec Silverlight

Dans cet article, vous allez découvrir les différents types d'animation proposés dans Silverlight

1. Introduction

Les animations doivent être placées dans le fichier .xaml. Elles doivent se trouver dans la section **x.Resources** (où x est au choix un Canvas, Grid ou StackPanel), et dans une section **Storyboard**.

```
<Canvas.Resources>
  <Storyboard x:Name="Animate">
    <!-- Animations here -->
  </Storyboard>
</Canvas.Resources>
```

L'attribut *x:Name* permet de donner un nom à votre **Storyboard** pour pouvoir y accéder à partir du javascript ou du code-behind.

On pourra notamment faire

```
Animate.Begin();
```

On pourra également accéder et changer les valeurs des propriétés de nos storyboard via leur nom.

```
Animate.Duration = new
Duration(TimeSpan.FromSeconds(5));
```

On peut également depuis la version 2 de Silverlight rajouter des animations et des storyboard dynamiquement en code.

```
DoubleAnimation SimpleAnimation = new
DoubleAnimation();
[...]
Animate.Children.Add(SimpleAnimation);
```

```
StoryBoard storyBoard = new StoryBoard();
[...]
MonCanvas.Resources.Add(storyBoard);
```

Un **Storyboard** peut bien entendu contenir plusieurs animations.

Il est également possible de lancer une animation à l'évènement **Loaded**, pour cela on peut placer le **Storyboard** dans les balises du contrôle (ici un **Rectangle**) de cette façon.

```
<Rectangle>
    <Rectangle.Triggers>
        <EventTrigger>
            <BeginStoryboard>
                <Storyboard>
                    <!-- Animations here -->
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Rectangle.Triggers>
</Rectangle>
```

Ici l'animation sera lancée après le chargement de l'objet **Rectangle**.

On peut aussi imbriquer les **Storyboard**, vous en verrez l'utilité en regardant les exemples présentés plus bas.

2. From - To Animations

L'animation dite *From - To* est l'animation de base, elle modifie la propriété d'un objet de la valeur **From** à la valeur **To**.

Le type de la propriété pouvant changer, nous avons à notre disposition plusieurs type d'animation *From - To*.

- DoubleAnimation
- ColorAnimation
- PointAnimation

La **DoubleAnimation** permet d'animer une propriété de type double (Height, Width, Canvas.Top, Canvas.Left...).

La **ColorAnimation** permet d'animer une propriété de type couleur (Backgroud, Foreground...).

La **PointAnimation** permet d'animer une propriété de type Point (RenderTransformOrigin...).

Commençons par spécifier la durée de l'animation, pour cela utilisons la propriété **Duration** qui a la syntaxe suivante : hh:mm:ss (où hh désigne le nombre d'heures, mm de minutes et ss de secondes).

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <DoubleAnimation Duration="00:00:05" />
    </Storyboard>
</Canvas.Resources>
```

Ici mon animation durera donc 5 secondes.

Pour spécifier des millisecondes (ici 50) : 00:00:00.050 !

Ensuite spécifions les valeurs de début et de fin de l'animation, il faut comme vous vous en doutez, utiliser les propriétés **From** et **To**.

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <DoubleAnimation Duration="00:00:05" From="10"
To="100" />
    </Storyboard>
</Canvas.Resources>
```

Ici la propriété (que nous n'avons pas définie), variera de 10 à 100 en 5 secondes.

Une autre propriété peut être spécifiée en rapport avec **From** et **To** : **By**.

Cette propriété sert à animer la valeur **par** (**by**) une certaine valeur, il ne s'agit en aucun cas de définir par quoi va passer l'animation.

Voici un petit tableau expliquant le fonctionnement des propriétés **From**, **To** et **By**. La valeur initiale est la valeur spécifiée à la création de l'objet qu'on veut animer.

Propriétés spécifiées	Résultat
From	La propriété changera en partant de la valeur **From** jusqu'à la valeur initiale
From et **To**	La propriété changera en partant de la valeur **From** jusqu'à la valeur **To**
To	La propriété changera en partant de la valeur initiale jusqu'à la valeur **To**
By	La propriété changera en partant de la valeur initiale jusqu'à la valeur **By** + la valeur initiale
From et **By**	La propriété changera en partant de la valeur **From** jusqu'à la valeur **From** + **By**

Si les propriétés **To** et **By** sont spécifiées en même temps, la valeur **To** est prioritaire.

Exemples :

DoubleAnimation

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <DoubleAnimation Duration="00:00:05" From="10"
To="100" />
    </Storyboard>
</Canvas.Resources>
```

ColorAnimation

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <ColorAnimation Duration="00:00:05" From="Red"
By="Blue" />
    </Storyboard>
</Canvas.Resources>
```

PointAnimation

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <PointAnimation Duration="00:00:05" To="1,1"/>
    </Storyboard>
</Canvas.Resources>
```

Alors bien évidemment ici, nous ne spécifions nulle part quelle propriété animer, et surtout quel objet animer. Pour cela nous devons utiliser les propriétés **Storyboard.TargetName** et **Storyboard.TargetProperty**.

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <DoubleAnimation
Storyboard.TargetName="GreenSquare"
Storyboard.TargetProperty="(Canvas.Left)"
                        Duration="00:00:05" From="10"
To="100" />
    </Storyboard>
</Canvas.Resources>
```

Ici nous avons spéficié d'animer la propriété **Canvas.Left** de l'objet **GreenSquare**.
GreenSquare est simplement un carré défini dans le XAML.

GreenSquare

```
<Rectangle x:Name="GreenSquare" Width="100"
Height="100" Fill="Green" />
```

Selon la propriété à animer la syntaxe de **Storyboard.TargetProperty** différe :
http://msdn2.microsoft.com/en-us/library/system.windows.media.animation.storyboard.targetproperty(VS.95).aspx (Lien60)

Voici donc quelques exemples d'utilisation de **Storyboard.TargetProperty**.

Canvas.Left

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <DoubleAnimation
Storyboard.TargetName="GreenSquare"
Storyboard.TargetProperty="(Canvas.Left)"
                        Duration="00:00:05" From="10"
To="100" />
    </Storyboard>
</Canvas.Resources>
```

Fill.Color

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <ColorAnimation
Storyboard.TargetName="GreenSquare"
Storyboard.TargetProperty="(Fill).(Color)"
                       Duration="00:00:05"
From="Black" To="Green" />
    </Storyboard>
</Canvas.Resources>
```

Nous allons maintenant passer en revue les autres propriétés d'une animation *From - To*.

Propriété	Valeurs possibles	Résultat
AutoReverse	True / False	**True** : après avoir terminé, l'animation fait marche arrière **False** par défaut
BeginTime	hh:mm:ss	Une fois la méthode Begin appellée, la **TimeLine** commencera après le temps spécifié.
FillBehavior	HoldEnd / Stop	Spécifie comment se comporte la propriété de l'objet une fois l'animation terminée. **HoldEnd** (*défaut*) : la propriété garde sa valeur de fin de l'animation. **Stop** : la propriété retourne à sa valeur initiale.
RepeatBehavior	nx hh:mm:ss Forever	**nx** : l'animation sera repétée *n* fois (3x par exemple). **hh:mm:ss** : l'animation sera repétée le temps spécifié, si le temps spécifié est inférieur à la propriété **Duration**, l'animation s'arrêtera au temps spécifié. **Forever** : l'animation sera répétée à l'infini.
SpeedRatio	n	Spécifie le ratio d'accélération de votre animation. Par exemple, si je spécifie 2, mon animation ira 2 fois plus vite.

Certaines propriétés peuvent être définies au niveau du **Storyboard**, comme **Duration** ou **Storyboard.TargetName**. Du coup ces propriétés sont valables pour toutes les animations définies dans le **Storyboard** (si elles ne sont pas redéfinies bien entendu).

Je rappelle également que l'on peut accéder à toutes ces propriétés via le code behind, il suffit d'ajouter l'attribut *x:Name* avec une valeur à nos animations.

```
<Canvas.Resources>
    <Storyboard x:Name="FromToAnimation">
        <DoubleAnimation x:Name="SimpleAnimation"/>
    </Storyboard>
</Canvas.Resources>
```

Et un code C# possible (Silverlight 2).

```
SimpleAnimation.Duration = new
Duration(TimeSpan.FromSeconds(5));
SimpleAnimation.From = 10;
SimpleAnimation.To = 540;
SimpleAnimation.SetValue(Storyboard.TargetNameProperty,
"GreenSquare");
SimpleAnimation.SetValue(Storyboard.TargetPropertyPrope
rty, "(Canvas.Left)");
```

Retrouvez la suite de l'article de Benjamin Roux en ligne : Lien61

Les derniers tutoriels et articles

Présentation des principaux design patterns en C++

Cet article a pour but de vous présenter la majorité des design patterns via un exemple de besoin qui sera complété par une mise en pratique en C++.

1. Introduction

Les design patterns. Un nom récurrent quand on code un minimum. On en a tous entendu parler de ces patterns, mais que sont-ils ?

Ce sont des modèles théoriques adaptables qui résolvent un problème précis.

Je vais vous montrer dans la suite de cet article les principaux patterns accompagnés à chaque fois d'une implémentation en C++.

2. Prototype

Définition : un prototype est une classe dont le but est d'être clonée.

2.1. Exemple de besoin du protoype

Pendant la création de vos programmes, vous serez sans doute amenés à instancier des objets de taille conséquente en mémoire.
Créer un gros objet ne pose guère de problèmes. Mais en créer plusieurs à la suite peut amener à tuer les performances de votre application.
La solution est alors de copier l'objet de base, le prototype, puis de modifier ce qui doit l'être pour que le nouvel objet réponde aux besoins.

De cette façon, on s'aperçoit que le pattern prototype dispose d'une méthode obligatoire, Clone, et d'une facultative Change. La première a pour but d'effectuer une copie du prototype tandis que la seconde a pour rôle de changer l'état de l'objet.
Ainsi, le diagramme UML qui modélise le pattern prototype est :

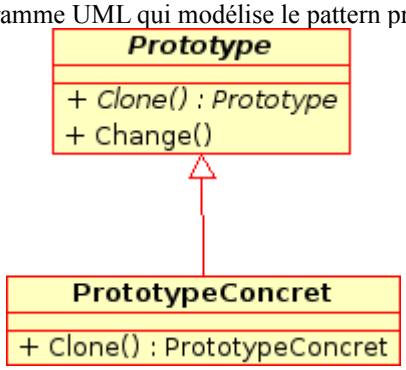


Diagramme UML du pattern prototype

2.2. Exemple naïf d'implémentation

D'après l'analyse précédente et en prenant comme exemple une connexion Mysql, une première implémentation de ce pattern pourrait être tout simplement :

```
//fichier prototype.h

#ifndef PROTOTYPE_H
#define PROTOTYPE_H

#include <string>

class Prototype
{
        public:
        virtual ~Prototype();
        virtual Prototype* Clone() const  = 0;
};

class MySQLManager: public Prototype
{
std::string m_host,m_login,m_pass;
std::string m_base,m_table;
public:
         MySQLManager(const std::string& host="",const
std::string& login="",const std::string& pass="");
         MySQLManager* Clone() const;

         void Afficher();
         void Set(const std::string& base="",const
std::string& table="");
};

#endif
```

Et dans le fichier prototype.cpp

```
#include <iostream>
#include "prototype.h"

Prototype::~Prototype()
{

}

MySQLManager::MySQLManager(const std::string&
host,const std::string& login,const std::string& pass):
m_host(host),m_login(login),m_pass(pass)
{
//on se connecte à la base Mysql.
//ceci prend du temps car il faut se connecter puis
s'identifier.
//Lourd.
}

MySQLManager* MySQLManager::Clone() const
{
       //le constructeur de recopie fait tout le
travail à notre place.
```

```
        return (new MySQLManager(*this));
}

void MySQLManager::Set(const std::string& base,const
std::string& table)
{
m_base=base;
m_table=table;
//on se connecte a la base m_base pour étudier m_table
}
void MySQLManager::Afficher() const
{
        std::cout<<m_host<<"|"<<m_login<<"|"<<m_host<<"
|"<<m_pass<<"|"<<m_base<<"|"<<m_table<<std::endl;
}

int main(void)
{
        MySQLManager* manager1=new
MySQLManager("localhost","Davidbrcz","motdepasse");
        manager1->Set("faussebase","table1");
        //ici manager1 est connecté sur localhost avec
l'identifiant Davidbrcz sur la table table1 de
faussebasse

        manager1->Afficher();

        //Maintenant, on doit travailler en parallèle
sur table2 de faussebase2 toujours sur localhost.
        //on crée donc un autre manager
        MySQLManager* manager2=manager1->Clone();
        //et ici manager2 est déjà connecté à localhost
avec l'identifiant Davidbrcz.Pas besoin de se
réidentifier.
        manager2->Afficher();
        manager2->Set("faussebase2","table2");
        manager2->Afficher();

        //on travaile ...

        delete manager1;
        delete manager2;

        return 0;
}
```

Ce code mérite quelques explications.
Tout d'abord on déclare une classe Prototype qui est une interface grâce à la méthode virtuelle pure Clone. Ensuite on déclare une classe MySQLManager qui hérite de Prototype.

On rend cette classe instanciable en définissant la méthode Clone. Mais la fonction virtuelle Prototype::Clone renvoie un Prototype. Or le type de retour ne rentre normalement pas dans la définition d'une fonction. Le compilateur devrait donc en théorie me crier dessus.

Mais si ceci compile sans problème c'est grâce aux valeurs de retour covariantes.
Pour faire simple les valeurs de retour covariantes sont les cas où la valeur de retour d'une fonction virtuelle est une référence ou un pointeur sur une classe C. Ainsi les classes qui redéfiniront cette fonction virtuelle pourront renvoyer un pointeur ou une référence sur une classe **dérivée** de C.

C'est ce qu'il se passe ici. MySQLManager dérive de Prototype donc MySQLManager::Clone peut renvoyer un pointeur vers une instance de MySQLManager.

2.3. Implémentation avec les templates

Le code précédent est bien fonctionnel mais peu modulaire. On va donc le templatiser.
Ainsi les classes Prototype et MySQLManager deviennent :

```
//fichier prototypetpl.h

#ifndef PROTOTYPE_H
#define PROTOTYPE_H

#include <string>

template <class T> class Prototype
{
        public:
        virtual ~Prototype();
        virtual T* Clone() const =0 ;
};

class MySQLManager: public Prototype<MySQLManager>
{

std::string m_host,m_login,m_pass;
std::string m_base,m_table;

public:
         MySQLManager(const std::string& host="",const
std::string& login="",const std::string& pass="");

         void Afficher() const ;
         MySQLManager* Clone() const ;
         void Set(const std::string& base="",const
std::string& table="");
};

#endif
```

Et le fichier prototype.cpp

```
#include <iostream>
#include "prototypetpl.h"

template <class T> Prototype<T>::~Prototype()
{

}

MySQLManager::MySQLManager(const std::string&
host,const std::string& login,const std::string& pass):
m_host(host),m_login(login),m_pass(pass)
{
//on se connecte à la base Mysql.
//Ceci prend du temps car il faut se connecter puis
s'identifier.
}

MySQLManager* MySQLManager::Clone() const
{
//le constructeur de recopie fait tout le travail à
notre place.
        return (new MySQLManager(*this));
}


void MySQLManager::Set(const std::string& base,const
std::string& table)
{
m_base=base;
m_table=table;
//on se connecte à la base m_base pour étudier m_table
}
```

```
void MySQLManager::Afficher() const
{
        std::cout<<m_host<<"|"<<m_login<<"|"<<m_host<<"
|"<<m_pass<<"|"<<m_base<<"|"<<m_table<<std::endl;
}
int main(void)
{
//le main reste inchangé
//...
}
```

Comme on peut le voir, les valeurs de retour covariantes sont ici remplacées par une classe template couplée à l'héritage.
Mis à part ceci, rien ne change.

3. Le singleton

Définition : le singleton permet de s'assurer qu'il n'existe qu'une unique instance d'une classe donnée.

3.1. Quand a-t-on besoin du singleton ?

Lors du développement de vos logiciels, vous souhaiterez sans doute vous assurer de n'avoir qu'une seule instance de certaines classes. En effet, imaginez que dans un jeux vidéo, un manager de son est, par mégarde, créé deux fois. Ceci pose de gros problèmes sur le plan de la gestion de la mémoire.

La solution est alors d'utiliser le pattern singleton. En effet celui-ci va, via une de ses méthodes, vous permettre de récupérer l'unique instance de la classe. Et puis, une fois votre travail fini, il ne reste plus qu'à détruire celle-ci via une autre méthode.

De cette analyse, on tire le diagramme suivant :

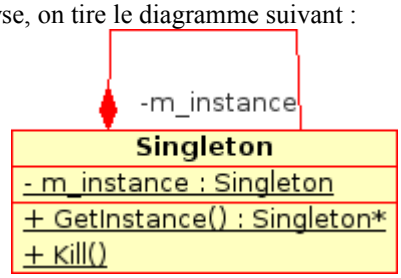


Diagramme UML du pattern singleton

3.2. Exemple simpliste d'implémentation du Singleton

Un premier exemple pour implémenter le pattern singleton peut tout simplement être le suivant (avec comme choix d'unique objet un manager de son). Le fichier singleton.h

```
#ifndef SINGLETON_H
#define SINGLETON_H

class SoundManager
{
 public:
    //les fonctions pour récupérer/tuer l'instance
    static SoundManager* Get();
    static void Kill();

 private:
     //l'instance à proprement parlé
    static SoundManager* m_instance;

    //constructeur et destructeur
    SoundManager();
   ~SoundManager();

   //constructeur de copie et operator= privé pour ne
pas pouvoir copier le singleton.
   SoundManager& operator= (const SoundManager&){}
   SoundManager (const SoundManager&){}
};
#endif
```

Et dans le fichier singleton.cpp

```
#include <iostream>
#include "singleton.h"

using namespace std;

SoundManager* SoundManager::m_instance=0;

SoundManager::SoundManager()
{
    cout<<"Création"<<endl;
}

SoundManager::~SoundManager()
{
    cout<<"Destruction"<<endl;
}

SoundManager* SoundManager::Get()
{
    if(m_instance==0) //si l'instance n'a pas encore
été créée, on la crée
        {
            m_instance=new SoundManager();
        }
    return m_instance;
}

void SoundManager::Kill()
{
    //pas besoin de vérifier si m_instance est valide,
delete 0 est sûr
    delete m_instance;
    m_instance=0;
}


int main(void)
{
    //Premier appel de Instance : on alloue le pointeur
SoundManager::m_instance
    SoundManager* ptr1=SoundManager::Get();

    //Deuxième appel : on se contente de renvoyer le
pointeur déjà alloué.
    SoundManager* ptr2=SoundManager::Get();

    //ptr1 et ptr2 pointent sur la même adresse
mémoire.
    //On voit donc qu'il n'y a bien qu'un seul objet.
    cout<<ptr1<<"|"<<ptr2<<endl;

    //On détruit l'unique instance.
    SoundManager::Kill();

    //ne fait rien
    SoundManager::Kill();

    return 0;
}
```

Il n'y a rien de bien difficile dans cette implémentation.

3.3. Implémentation templatisée

Ce singleton fonctionne bien, mais il est un peu lourd à écrire. En effet avec Get et Kill qui font la même chose d'une classe à l'autre aux types près, les templates semblent s'imposer d'elles mêmes.
Ainsi le code devient dans le fichier singleton.h :

```
#ifndef SINGLETONTPL_H
#define SINGLETONTPL_H

template <class T> class Singleton
{
public:
        static T* Get();
        static void Kill();
protected:
        static T* m_i;
        private:
        T& operator= (const T&){}
};

class SoundManager :public Singleton<SoundManager>
{
        friend SoundManager*
Singleton<SoundManager>::Get();
        friend void Singleton<SoundManager>::Kill();

private:
       SoundManager (const SoundManager&){}

        SoundManager();
       ~SoundManager();
};

#endif
```

Et dans le fichier singleton.cpp

```
#include <iostream>
#include "singletontpl.h"

template <class T> T* Singleton<T>::m_i=0;


template <class T>  T* Singleton<T>::Get()
{
        if(m_i==0)
        {
                m_i=new T();
        }
        return m_i;
}

template <class T> void Singleton<T>::Kill()
{
        delete m_i;
        m_i=0;
}

SoundManager::SoundManager()
{
std::cout<<"Création"<<std::endl;
}

SoundManager::~SoundManager()
{
std::cout<<"Destruction"<<std::endl;
}

int main(void)
{
        SoundManager*
sin=Singleton<SoundManager>::Get();
        SoundManager*
sin2=Singleton<SoundManager>::Get();
        std::cout<<sin<<"|"<<sin2<<std::endl;
        Singleton<SoundManager>::Kill();
}
```

La seule innovation de ce code vis-à-vis du précédent est l'héritage particulier de SoundManager

En effet, avec la classe singleton actuelle, il faut faire hériter de Singleton la classe que l'on souhaite avoir en unique exemplaire. Or la classe Singleton est template, le seul moyen de l'instancier est donc de passer la classe dérivée en paramètre de Singleton.

Mais avec ceci un autre problème se pose : pour assurer l'unique instance de la classe SoundManager, ses constructeurs et son destructeur doivent être privés, mais Instance et Kill doivent avoir accès à ceux-ci.
Ce problème est résolu via l'utilisation de l'amitié sur les méthodes Instance et Kill de Singleton.

3.4. Remarques sur le singleton

3.4.1. Ce singleton n'est pas thread-safe

3.4.1.1. Première solution envisageable

Le singleton que je vous propose fonctionne bien mais n'est absolument pas thread-safe.

En effet, imaginez que dans un contexte multi-thread un thread A exécute jusqu'à la ligne *if(m_i==0)* de *Get* puis qu'il soit suspendu par l'OS.
Tout ce que nous pouvons dire pour le moment, c'est qu'aucun objet de type Singleton n'a été créé mais qu'à la reprise du thread A, un objet va être créé.

Si juste après la suspension de A un autre thread (nommé B) exécute entièrement *Get*, le Singleton sera créé. Mais lorsque A reprendra son exécution, il va continuer et créer un autre Singleton.
Ce qui fait que l'on se retrouve avec deux objets de type Singleton ! Ce qui est fondamentalement contraire à ce pattern.

Une première solution est de placer un mutex (objet qui permet de bloquer l'accès à des variables se situant après sa création tant que ce premier n'a pas été détruit) dans la méthode *Get* juste avant *if(m_i==0)*. De cette façon, le code de Get devient :

```
template <class T>  T* Singleton<T>::Get()
{
        Lock lock //ici Lock est une classe symbolique.
Elle représente un mutex.
        if(m_i==0)
        {
                m_i=new T();
        }
        return m_i;
}
```

3.4.1.2. Problème lié à cette solution

Mais cette solution est très mauvaise. En effet nous avons besoin de locker seulement à la première création de m_instance.
Alors pourquoi payer le prix d'un lock à chaque appel de *Get* quand un seul est nécessaire ?

C'est pour résoudre ce problème que l'on va utiliser le principe du double check. En effet, pourquoi acquérir un lock si le singleton a déjà été créé ?
Ainsi selon cette courte réflexion, la méthode *Get* devient :

```
template <class T>  T* Singleton<T>::Get()
{
  if(m_i==0) //Premier check
  {
        Lock lock
        if(m_i==0) //Second
        {
                m_i=new T();
        }
  }
        return m_i;
}
```

Et de ce fait le nom de double check prend tous son sens !

En effet, le premier sert à vérifier que le singleton n'a pas déjà été créé, auquel cas on ne fait que renvoyer l'instance.
Et le second sert à s'assurer que le singleton n'a pas été créé par un autre thread entre le premier test et le moment d'acquisition du lock.

3.4.1.3. Problème lié au double check

Dans cette sous-section, je présume que vous disposez de connaissances du C++ assez évoluées, tel que le placement new.

Pour entrevoir le problème, il faut descendre plus bas dans le langage.
Avant toute chose regardons comment se décompose une allocation dynamique de type T pour un pointeur p :

1. Réservation de mémoire de taille suffisamment grande pour contenir un objet de type T.
2. Appel du constructeur de T.
3. Affectation de l'adresse réservée par la première opération à p.

Ce qui fait que l'on peut écrire notre singleton de cette façon :

```
template <class T>  T* Singleton<T>::Get()
{
  if(m_i==0)  //1er check
  {
     Lock lock;
        if(m_i==0)  //2eme
        {
                m_i=
//phase 3 de la liste
                static_cast<T*>(operator
new(sizeof(T))); //phase 1
                new (m_i) T;
// phase 2
        }
  }
        return m_i;
}
```

Ce code, s'il est exécuté dans cet ordre, ne pose aucun problème. Mais comme la vie n'est pas rose, les compilateurs peuvent, s'ils le souhaitent, intervertir les phases (iii) et (ii) de telle sorte que l'adresse mémoire soit affectée à m_i avant l'appel du constructeur. Or ceci pose un problème. Imaginons le scénario suivant :

1. Un thread A appelle Get, obtient un lock et effectue les étapes (i) et (iii) avant d'être suspendu.
   Au moment de sa suspension, m_i n'est pas NULL mais ne pointe pour autant pas sur un objet valide.
2. Un thread B appelle à son tour Get. À ce moment, cet appel de Get ne passe pas le premier check car m_i n'est justement pas NULL !
   Puis il renvoie l'instance non complètement construite de m_i. À l'instant où le code client va déférencer m_i, boom !

En effet, c'est là que se pose tout le problème du double check, puisqu'il impose un ordre de séquencement que les compilateurs ne sont pas obligés de respecter.
Quoi que vous fassiez,vous **ne pourrez pas** obliger le compilateur à suivre l'ordre de séquencement qui vous arrange, ce qui fait que pour certaines personnes, en contexte multi-thread, le pattern singleton peut être considéré comme un anti-pattern.

3.4.2. Le singleton est une variable globale

Je tiens à vous mettre en garde, le singleton est malgré son apparence une **variable globale** ! Il est donc à utiliser avec précaution. De plus, faites attention à ne pas attraper la singletonite-aiguë qui consiste à mettre des singletons un peu partout. Si vous ne devez construire qu'une seule instance dans un objet dans un petit programme mono-thread, ne sortez pas le singleton en prétextant le fait qu'il ne doit y avoir qu'une seule instance. S'il ne doit en avoir qu'une dans votre programme, il vous suffit de n'en créer qu'une !

Retrouvez la suite de l'article de David Come en ligne : Lien62

Comment programmer un Navigateur Internet avec Qt 4 et QtWebKit

Cet article explique comment il est possible de créer efficacement et rapidement un navigateur avec Qt 4.4 et le QtWebKit.

1. Introduction

Dans sa version 4.4, Qt a reçu un nouveau module, j'en ai suivi le développement au cours de l'année dernière tout en étant impatient de m'amuser avec en utilisant Qt. Peu après, le module a été intégré à la branche principale de Qt et j'ai commencé à travailler sur un petit navigateur pendant mon temps libre. Un outil que je pourrais utiliser pour naviguer un peu au quotidien. En me servant directement de ce que j'ai créé, j'ai pu détecter des erreurs dans WebKit, le nouveau code de mise en réseau, et repérer des zones posant problème, en particulier dans l'interface de programmation avant le lancement sur le marché.

Avis de non-responsabilité : Ce n'est qu'un petit projet de démonstration personnel, en aucun cas une substitution à FireFox.

2. Vitesse de démarrage

L'une des particularités appréciables de Qt4 par rapport à Qt3 est sa vitesse de démarrage. J'ai même lu des commentaires de KDE4 qui mentionnent le petit bond en avant reçu par les applications KDE, qui l'ont bien supporté sans nécessiter de réusinage.

Lorsque l'un de mes collègues a commencé à travailler sur l'outil d'intégration WebKit/Qt, j'ai remarqué que son application test se lançait presque instantanément quand Qt était déjà en mémoire. Donc, si WebKit peut se lancer rapidement et que Qt le peut également, j'allais à coup sûr faire en sorte qu'aucun code de démonstration idiot ne le ralentisse au lancement. Ainsi, même avec une base de données d'icônes, un historique, des cookies, des onglets, etc. la démonstration se lance tout de même rapidement. Je pense donc que la démonstration est assez rapide.

3. Gestion du réseau

La version 4.4 comprend de nouvelles classes de gestion de réseau élaborées autour de la nouvelle classe QNetworkAccessManager. J'ai créé, au-dessus de ces classes de gestion de réseau, des composants interface utilisateur graphique comprenant un gestionnaire de téléchargement minimal avec les outils habituels que sont une barre de progression, la vitesse de téléchargement, etc. Une belle petite démo pour le gestionnaire de téléchargement en bonne et due forme. L'une des nouveautés utilisées par QNetworkAccessManager que j'ai oublié de mentionner est QCookieJar qui est inclus dans Qt4. En le créant, j'ai ajouté, sauvegarder/charger, et les fonctionnalités habituelles d'un navigateur Internet.

4. Agent utilisateur

Une fois le navigateur suffisamment préparé pour être utilisé quotidiennement, j'ai réglé Gnome et KDE pour qu'ils choisissent le navigateur de démonstration comme navigateur favori. Donc, dès que je clique sur un lien contenu dans un courrier électronique ou en provenance de #qt sur irc, le navigateur de démonstration est utilisé et un nouvel agent utilisateur apparaît dans les journaux de serveur pour le mystérieux demobrowser/0.1. L'agent utilisateur qtwebkit par défaut est élaboré à partir de QCoreApplication::applicationName et des nouvelles propriétés QCoreApplication::applicationVersion. Donc, si vous intégrez QtWebKit dans votre application, il inclura automatiquement le nom et la version de l'application dans l'agent utilisateur. Il est possible de configurer cette option.

Puisqu'il ne s'agit que d'une démonstration, et par souci de simplicité, la démo se comporte comme une application simple, lorsqu'elle et lancée, elle contacte le navigateur en cours d'exécution et lui demande d'ouvrir l'URL des arguments de ligne de commande s'il y en a (il est bien entendu possible de le configurer pour qu'une nouvelle fenêtre ou un nouvel onglet s'ouvre).

5. Gestion de session

En développant la démonstration, je savais que je devrais laisser mon navigateur ouvert pendant des jours et la perte des onglets ouverts (ou tout en fait) dans un plantage n'est jamais agréable. J'ai donc voulu, lors du développement de la démonstration, redémarrer celle-ci rapidement à mesure que j'y apportais des modifications. Avec ceci en tête, la démo contient un petit outil de surveillance qui s'assure que l'on ne perd au maximum que les trois dernières secondes en cas de plantage. Ce ne sont pas seulement les onglets, mais aussi les cookies et tout le reste qui sont protégés. Au redémarrage du navigateur, en sélectionnant Historique/Restaurer la dernière session, toutes les fenêtres et onglets de premier niveau de la dernière session se rouvriront très vite.

6. X-Qt-Plugin

QtWebKit comprend un cadre de plug-in et QWebPage comporte une structure supplémentaire en support pour un type de plug-in spécifique, x-qt-plugin que le demobrowser exécute

Le navigateur de démonstration vous permet d'ajouter un gadget logiciel Qt sur la page Web. Comme avec QScript, les signaux, fentes, propriétés sont tous entièrement accessibles depuis JavaScript.

L'on pourrait même intégrer QWebView et un navigateur dans le navigateur si on le voulait. Si vous chargez une page Web avec le navigateur de démonstration avec le code suivant, vous trouverez une QProgressbar de chargement.

```
<code>
<object type="application/x-qt-plugin"
classid="QProgressBar"
name="progressbar" height=30></object>
<script>
function display(){
    if (++document.progressbar.value != 100)
        setTimeout("display()", 50)
}
display();
</script>
</code>
```

7. Plus qu'un navigateur

Certains développeurs ont déjà rédigé des blogs sur leurs projets QtWebKit intéressants comme Amarok (Lien63) et KDE Plasmoids (Lien64). Je suis impatient de voir ce qui se fait avec QtWebKit.

8. Conclusion

Vous pouvez trouver la démo dans le répertoire de démos/navigateur du progiciel Qt. Le navigateur de la version 4.4 snapshots (Lien65) est plus complet que la version bêta car il comprend la configuration du proxy, une page de recherche, des raccourcis clavier améliorés et certaines fonctionnalités citées ici.

- Ce n'est qu'une démo, j'ai essayé de faire quelque chose qui serait facile à faire avec Qt, je ne me suis donc pas servi de support bunzip, tar et bittorrent pour le gestionnaire de téléchargement. Il existe plein de choses à clarifier tout d'abord comme les signets.
- Bien que le support de plug-in ait été ajouté à QtWebKit pour la version 4.4, le support plug-in flash officiel d'Adobe devra attendre la version 4.5. Plus précisément, le support de plug-in Netscape ne sera pas ajouté avant la version 4.5.
- Gmail ne fonctionne pas actuellement.
- Aucun plantage n'a été rapporté, alors n'hésitez pas à les signaler si vous en rencontrez.
- J'ai prévu de continuer à développer cette démo. J'ai, par exemple, presque terminé les signets dans une branche et je travaille à l'ajout de tests automatiques supplémentaires avant de fusionner.
- De nombreux problèmes de cookies peuvent être contournés en activant "Accepter tous les cookies" dans les préférences, la réserve de cookies comporte des solutions n'ayant pas encore été ajoutées.
- N'hésitez pas à vérifier le code de la démo. La taille totale est relativement réduite et j'ai essayé de garder le code assez simple.

Retrouvez l'article de Benjamin Meyer en ligne : Lien66

Les derniers tutoriels et articles

Création et utilisation de classes personnalisées en VB 6.0 et VBA (partie 1)

Grâce aux classes personnalisées, créez vos propres objets, maintenez facilement votre code, réutilisez rapidement votre code dans de nouvelles applications

1. Introduction

1.1. Pourquoi utiliser des classes personnalisées

On est souvent confronté, lors de l'utilisation de VB ou de VBA, au problème de "réinventer la roue" à chaque nouveau travail que l'on développe.

Si je prends l'exemple d'applications liées à la gestion d'une entreprise, nombre de nos classeurs Excel, de nos bases de données Access, voire de nos applications VB, gèrent les données de contacts.

L'utilisation de classes personnalisées va permettre un gain de temps appréciable en favorisant la réutilisation et la maintenance du code. De plus, les classes personnalisées permettent la mise à disposition d'objets complexes à des programmeurs qui pourront "se borner" à utiliser les propriétés et méthodes qu'une classe personnalisée met à leur disposition.

1.2. Objectifs

Ce tutoriel en trois parties a pour ambition de vous expliquer le fonctionnement des classes personnalisées en VB 6.0 ou en VBA, de vous démontrer que la mise au point de classes performantes n'est pas un travail impossible, et de vous prouver que les classes personnalisées permettent la mise à disposition d'autres programmeurs d'un code qui les soulage de la "réinvention de la roue".

1.3. Notions abordées dans la partie 1

Dans cette partie, nous allons manipuler principalement les concepts liés aux propriétés d'un objet.

Pour cela, nous allons créer ce que l'on appelle une "classe métier", c'est-à-dire une classe qui reproduit en informatique un objet ou un concept que l'on voudra manipuler.

Notions abordées dans la partie 1
- Qu'est-ce qu'un objet informatique
- Qu'est-ce qu'une propriété
- Propriété en lecture seule, en écriture seule, en lecture écriture
- Encapsulation
- Génération des erreurs dans une classe et gestion par le code appelant
- Génération des évènements et gestion par le code appelant

1.4. Notions abordées dans la partie 2

Dans un deuxième temps, nous utiliserons notre classe "métier" vue en partie 1 en la liant à différentes sources de données au travers d'une nouvelle classe d'accès aux données.

Notions abordées dans la partie 2
- Création d'un objet personnalisé au travers d'une classe d'accès aux données
- Liaison d'un objet personnalisé à une source pour la lecture des données
- Liaison d'un objet personnalisé à une source pour la modification des données du fichier source
- Portabilité du jeu des classes
- Utilisation de données d'une base Access dans un fichier Excel grâce aux classes personnalisées

1.5. Notions abordées dans la partie 3

La troisième partie nous permettra d'aborder une classe liant propriétés, fonctions et méthodes.

1.6. Prérequis

L'utilisation et la compréhension de ce cours requiert une bonne connaissance de base en VB ou VBA.
Vous devez notamment savoir déclarer et utiliser des variables et comprendre la notion de portée d'utilisation d'une variable.
Vous devez savoir déclarer et utiliser des fonctions personnalisées et des procédures requérant l'emploi de paramètres.
Il est également préférable que l'environnement de développement de VB ou VBE (Visual Basic Editor) vous soit familier, notamment les possibilités de débogage du code.

2. Notions d'objet

2.1. Les objets de la vie courante

Dans la vie courante, nous utilisons quotidiennement des objets.
Nous les définissons par un ensemble de caractéristiques et, généralement, nous pouvons agir avec ces objets.

Ainsi, si je regarde une boite à chaussures, je peux la définir.
Elle est noire, elle fait 40x18x12cm, ...
Je peux l'ouvrir, la fermer, la remplir, la vider,...

Ma voiture possède quatre roues, un volant, un moteur de telle cylindrée.
Je peux la faire avancer, la stopper, ...

Dans ces exemples, nous discernons que nos objets possèdent des caractéristiques qui leurs sont propres, et que nous pouvons définir des actions réalisables sur ou avec ces objets.

2.2. Les objets informatiques

Un objet informatique est similaire à un objet de la vie courante, et nous en utilisons quotidiennement dans nos codes VBA.

Ces objets possèdent des caractéristiques qui leur sont propres et nous pouvons agir sur ou avec ces objets.

Les caractéristiques d'un objet informatique sont appelées les **propriétés** de l'objet.
Les actions que nous pouvons réaliser sur ou grâce à ces objets sont appelées des **méthodes**.

Lorsque nous souhaitons utiliser un objet, nous devons donc connaître les propriétés de l'objet mises à notre disposition, et nous devons connaître aussi les méthodes utilisables sur cet objet.

2.2.1. Quelques objets informatiques

En VB6, nous utilisons tout le temps des objets. Tous les contrôles que nous plaçons sur un formulaire (qui est lui-même un objet) sont des objets.

En VBA pour Excel, nous manipulons les objets **Range** (plage de cellules), **Worksheet** (feuille de calcul), **Workbook** (classeur), ...

En VBA pour Word, nous manipulerons les objets **Word**, **Document**, ...

En VBA pour Access, nous manipulerons les objets **Form**, **Database**, **Recordset**

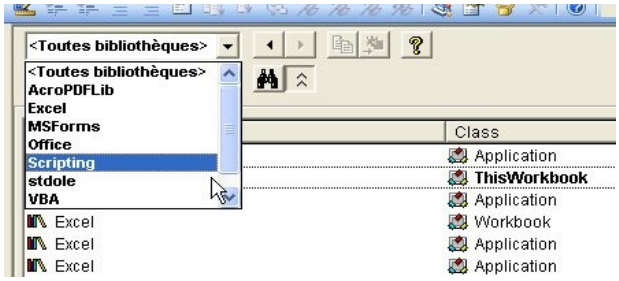
En VB6 ou VBA, nous référençons parfois des bibliothèques d'objets externes, pour manipuler des objets particuliers. Par exemple, l'utilisation de la bibliothèque **Microsoft Scripting Runtime** permet de manipuler des objets **Fichier, dossier, ...** non utilisables par défaut dans VB6 ou VBA.

2.2.2. L'explorateur d'objets en VB et VBA

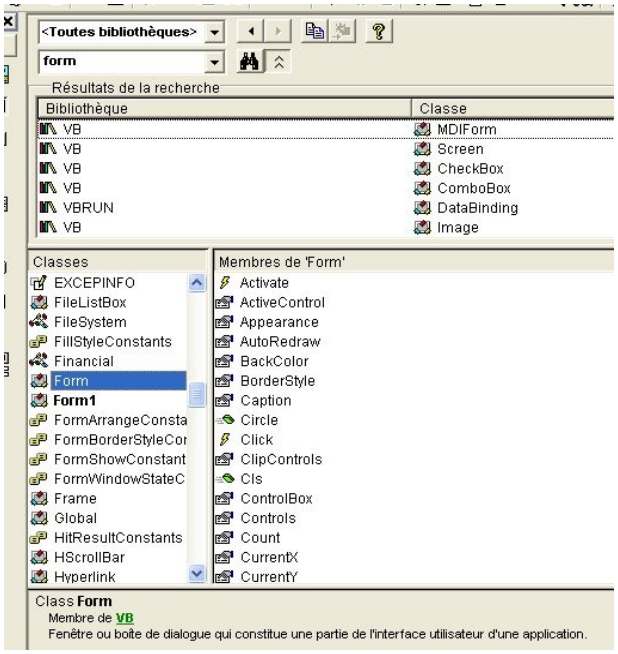
L'explorateur d'objets permet... d'explorer les objets, c'est-à-dire d'en afficher les propriétés, méthodes et évènements.

En VB 6.0 comme en VBA, l'affichage de l'explorateur d'objets est possible via le menu **Affichage/Explorateur d'objets** ou par le raccourci **F2**.

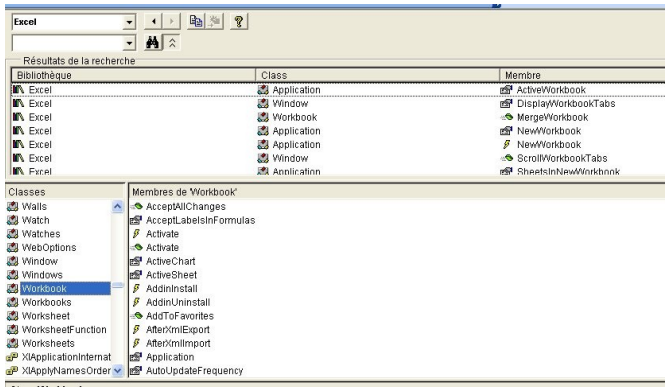
Lorsque l'explorateur d'objets est affiché, on peut alors choisir d'afficher les objets de toutes les bibliothèques disponibles du projet, ou de restreindre la liste des obets à une bibliothèque particulière.



On peut alors choisir l'objet pour lequel on souhaite afficher les propriétés, méthodes et évènements.



L'objet Form de la bibliothèque VB



L'objet Workbook de la bibliothèque Excel

3. Utilisation de variables simples, puis d'un "pseudo-objet", le type personnalisé

Lorsque l'on doit utiliser un "objet" informatique, on a trois possiblités. La première consiste à créer autant de variables que de propriétés.

La seconde utilise un type personnalisé.

La troisième, enfin, passe par l'utilisation d'une classe personnalisée.

Avant d'aborder la création d'une classe personnalisée, je vais brièvement exposer les deux premières méthodes, ne serait-ce que pour démontrer la puissance d'une classe par rapport aux variables simples et aux types personnalisés.

Pour ceux qui connaissent l'histoire des trois petits cochons, on

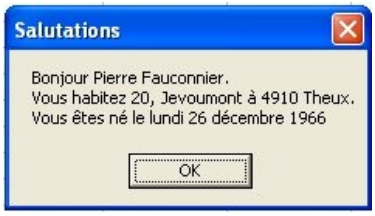
peut imaginer la première solution comme étant la maison de paille, le type personnalisé étant la maison de bois, la classe personnalisée comme la maison de briques...

3.1. Manipulation des données d'un contact sans utilisation d'un type personnalisé

Si je souhaite manipuler les données d'un contact, je dois déclarer autant de variables que j'ai de données (champs) pour mon contact, puis leur attribuer les données de mon contact, et enfin les utiliser, par exemple au sein d'une procédure qui affiche un message à l'écran.

```
Sub UtiliserContact()
    Dim ContactNom As String
    Dim ContactPrenom As String
    Dim ContactAdresse As String
    Dim ContactCP As String
    Dim ContactLocalite As String
    Dim ContactDateNaissance As Date

    ContactNom = "Fauconnier"
    ContactPrenom = "Pierre"
    ContactAdresse = "20, Jevoumont"
    ContactCP = "4910"
    ContactLocalite = "Theux"
    ContactDateNaissance = DateSerial(1966, 12, 26)

    AfficherMessageContact ContactNom, ContactPrenom,
ContactAdresse, _
        ContactCP, ContactLocalite,
ContactDateNaissance
End Sub

Sub AfficherMessageContact(Nom As String, Prenom As
String, Adresse As String, _
    CP As String, Localite As String, DateNaissance As
Date)
    MsgBox "Bonjour " & Prenom & " " & Nom & "." &
vbCrLf & _
        "Vous habitez " & Adresse & " à " & CP & " " &
Localite & "." & vbCrLf & _
        "Vous êtes né le " & Format(DateNaissance,
"dddd dd mmmm yyyy"), vbOKOnly, "Salutations"
End Sub
```

Ce qui me donnera le résultat suivant à l'écran.



3.1.1. Limites de l'utilisation de variables simples

Les limites du code présenté ci-dessus sont évidentes.

Chaque fois que je souhaite utiliser les données d'un autre contact, je dois réinitialiser mes variables, avec le risque d'en oublier, et donc de mélanger les données de mes contacts.

De plus, passer les variables à une procédure ou une fonction qui doit les utiliser n'est déjà pas pratique avec six variables...

```
...
    AfficherMessageContact ContactNom, ContactPrenom,
ContactAdresse, _
        ContactCP, ContactLocalite,
ContactDateNaissance
...
```

```
Sub AfficherMessageContact(Nom As String, Prenom As
String, Adresse As String, _
...
End Sub
```

Imaginez ce que cela donnerait avec un objet défini par vingt ou trente champs...

Enfin, si je veux utiliser simultanément les données de plusieurs contacts, je vais devoir créer autant de variables **ContactNom1**, **ContactNom2**, ... que j'ai de contacts à gérer simultanément. Mission impossible!!

3.2. Qu'est-ce qu'une variable de type personnalisé et quand créer un type personnalisé?

Une variable de type personnalisé est un groupe de plusieurs variables simples dont le type personnalisé concrétise la structure.

Un type personnalisé sera créé lorsqu'il s'avérera nécessaire de manipuler plusieurs variables comme si elles représentaient une entité.
L'utilisation de variables de type personnalisé répondra partiellement aux objections émises plus haut quant à l'utilisation de variables simples.

3.3. Création d'un type personnalisé

Créer un type personnalisé de variable, c'est définir la structure de la variable, préciser quelles sont les variables simples qui seront groupées.

Les variables utilisées plus haut dans le cours constituent la structure de mon type personnalisé.
Je vais les réutiliser en les modifiant légèrement et créer, ou plutôt définir mon type personnalisé avec le code suivant:

```
Public Type tpContact
    Nom As String
    Prenom As String
    Adresse As String
    CP As String
    Localite As String
    DateNaissance As Date
End Type
```

Remarquez qu'ici, j'ai supprimé le préfixe **Contact** pour mes variables, puisqu'elles sont définies à l'intérieur de mon type personnalisé.

Ce code se place toujours en tête de module **standard**. Il ne peut jamais être utilisé dans le corps d'une procédure ou d'une fonction.

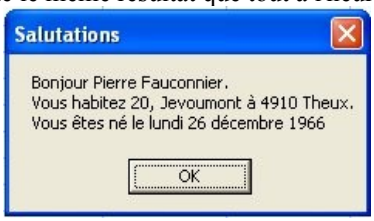
3.4. Utilisation de mon type personnalisé

Je peux maintenant utiliser ce type personnalisé dans mon code

```
Sub UtiliserContact()
    Dim Contact As tpContact

    With Contact
        .Nom = "Fauconnier"
        .Prenom = "Pierre"
        .Adresse = "20, Jevoumont"
        .CP = "4910"
        .Localite = "Theux"
        .DateNaissance = DateSerial(1966, 12, 26)
    End With

    AfficherMessageContact Contact
```

```
End Sub

Sub AfficherMessageContact(ByRef Contact As tpContact)
    With Contact
        MsgBox "Bonjour " & .Prenom & " " & .Nom & "."
& vbCrLf & _
            "Vous habitez " & .Adresse & " à " & .CP &
" " & .Localite & "." & vbCrLf & _
            "Vous êtes né le " & Format(.DateNaissance,
"dddd dd mmmm yyyy"), vbOKOnly, "Salutations"
    End With
End Sub
```

Ce qui me donne le même résultat que tout à l'heure.



3.5. Différences entre l'utilisation de variables et l'utilisation d'un type personnalisé

Après avoir créé mon type personnalisé, j'utilise une variable de ce type au travers de mon code, en étant certain d'emporter avec moi la structure complète de ma variable.

De plus, l'écriture et la lecture du code sont plus simples, notamment lorsque je dois passer ma variable en argument d'une fonction ou d'une procédure.

```
...
    AfficherMessageContact Contact
End Sub

Sub AfficherMessageContact(Contact As tpContact)
...
```

est plus simple à écrire et à lire que

```
...
    AfficherMessageContact ContactNom, ContactPrenom,
ContactAdresse, _
        ContactCP, ContactLocalite,
ContactDateNaissance
End Sub

Sub AfficherMessageContact(Nom As String, Prenom As
String, Adresse As String, _
    CP As String, Localite As String, DateNaissance As
Date)
...
```

Enfin, la maintenance de mon code est simplifiée lorsque la structure de ma variable change.

En effet, si je dois ajouter un "champ" à mon type personnalisé, les modifications se limitent à l'affectation du nouveau champ et à la récupération de sa valeur. Les passages et récupérations d'un paramètre **contact** ne changent pas.

3.6. Limites de l'utilisation d'un type personnalisé

L'utilisation d'un type personnalisé ne permet pas le traitement des données à l'intérieur de sa structure.
La vérification et le traitement des données sont délégués au code appelant, ce qui limite considérablement la portabilité du type.

En effet, dans l'exemple de l'utilisation de ma variable **Contact**, je dois déléguer au code appelant la vérification de date de naissance. Nulle part dans la structure **Type... End Type**, je n'ai la possibilité d'effectuer des actions relatives à la saisie d'informations.

De plus, la gestion des erreurs est également déléguée au code appelant.

Enfin, je ne peux gérer aucun évènement relatif à un type personnalisé.

4. Une classe pour gérer des contacts

L'utilisation d'une classe personnalisée va balayer ces restrictions et me donner la possibilité de mettre à disposition du code utilisateur de ma classe des propriétés, méthodes, erreurs et évènements.

Dans la suite du cours, nous allons avancer petit à petit dans la création d'une classe permettant de gérer des contacts.

La classe que nous allons créer dans un premier temps ne sera vraiment pas professionnelle, mais nous améliorerons le code petit à petit pour intégrer calmement les notions essentielles à la création de classes sûres et portables.

4.1. Notion de classe,notion d'objet, notion d'encapsulation

4.1.1. Notion de classe

La classe représente la structure d'un objet. En cela, elle est comparable aux lignes de définition **Type... End Type**.

4.1.2. Notion d'objet

Un objet est une variable dont le type s'appuie sur une classe. Que ce soit en VB ou en VBA, cette classe peut être personnalisée ou mise à disposition par l'application.

VB 6.0 comme VBA permettent d'utiliser des références externes. Dans VB 6.0, le menu **Projet/Références...** permet d'ajouter une référence. Dans VBA, c'est *via* le menu **Outils/Références...** que vous avez la possibilité d'ajouter une référence externe.

Ainsi, en Excel, **Dim MaFeuille As Worksheet** créera un objet appelé **MaFeuille** qui s'appuiera sur la classe qui définit les objets de type **WorkSheet**, c'est-à-dire les feuilles de calcul.

4.1.3. Notion d'encapsulation

Nous devrons toujours veiller, lorsque nous construisons une classe, à ce qu'elle soit "encapsulée". Cela veut dire qu'à aucun moment, notre classe ne doit avoir besoin de données externes pour fonctionner. Dans les faits, cela veut dire que notre classe ne peut jamais faire appel à une variable extérieure à la portée du module de classe.

De même, une classe ne devrait jamais donner la main à l'utilisateur via par exemple un **msgbox** ou un **inputbox**. Il faudra donc développer à l'intérieur du module de classe des événements et des gestions d'erreurs qui rendront la main au code appelant.

Ces notions seront vues dans la suite du cours.

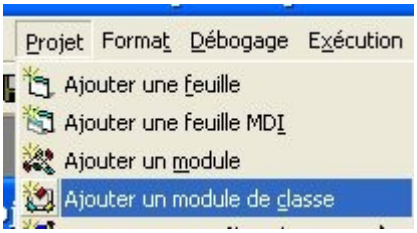
4.2. Création de notre première classe

La classe que nous allons créer va nous permettre de manipuler un objet **contact**.

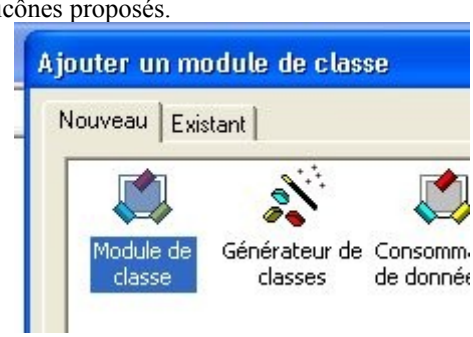
4.2.1. Module de classe

4.2.1.1. Ajout d'un module de classe en VB 6.0

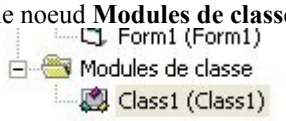
En VB 6.0, nous créerons une classe personnalisée en ajoutant un module de classe via le menu **Projet/Ajouter un module de classe**



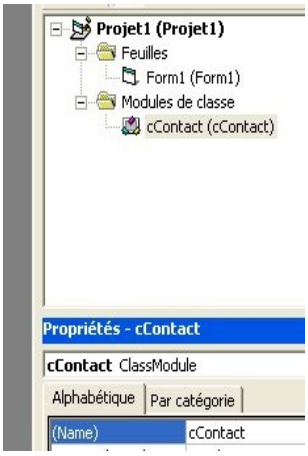
Nous choisissons alors d'ajouter le module de classe parmi les différents icônes proposés.



Le nouveau module de classe apparaît dans l'arborescence de notre projet, sous le noeud **Modules de classe**



La première chose que nous allons faire, c'est renommer le module, **car c'est lui qui déterminera le nom de notre classe**.
Ce nom sera donc choisi avec soin, en évitant notamment les noms réservés par VB 6.0.

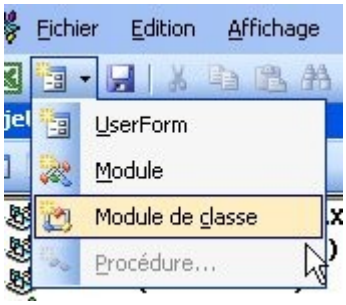


4.2.1.2. Ajout d'un module de classe en VBA

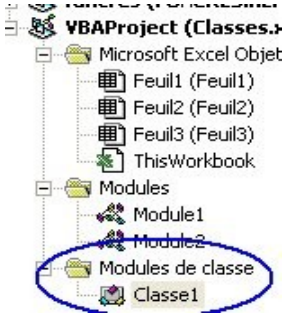
En VBA, nous créerons une classe personnalisée par l'ajout d'un module de classe au sein de notre projet.

Les illustrations proviennent d'un VBA pour EXCEL, mais vous pouvez les transposer sans problème sur Word, PowerPoint, Access, Outlook, ...

Pour ajouter un module de classe, utilisons le menu **Insertion/Module de classe**.
Nous pouvons aussi utiliser le bouton d'outil d'ajout de module.



Un nouveau module apparaît dans l'arborescence du projet, dans le noeud **Modules de classe**.



La première chose que nous allons faire, c'est renommer le module, **car c'est lui qui déterminera le nom de notre classe**.
Ce nom sera donc choisi avec soin, en évitant notamment les noms déjà utilisés par VBA.

4.2.2. Création de propriétés simples pour notre classe

Ce que nous allons voir maintenant est à proscrire en utilisation professionnelle de classes personnalisées.
La seule raison de leur mention ici est de permettre une prise en main "pas à pas" des notions nécessaires à la réalisation d'une classe performante.

Pour ajouter des propriétés simples à notre classe, il nous suffit de déclarer des variables publiques au sein de notre module de classe.

Pour cela, nous pouvons reprendre la structure de notre type personnalisé.
Le code est le suivant:

```
Public Nom As String
Public Prenom As String
Public Adresse As String
Public CP As String
Public Localite As String
Public DateNaissance As Date
Public Sexe As String
```

Ces variables publiques, parce que déclarées au sein d'un module de classe, sont des propriétés en lecture/écriture pour les objets qui s'appuieront sur notre classe.

Le terme **variable publique** est utilisé ici abusivement. La déclaration de variables via **Public ...** dans un module de classe ne crée pas des variables publiques utilisables partout dans le projet, mais bien des **propriétés publiques** mises à disposition du code appelant par la classe dont l'objet est issu.

A l'extérieur du module de classe, ces propriétés devront être préfixées du nom de l'objet issu de la classe personnalisée.

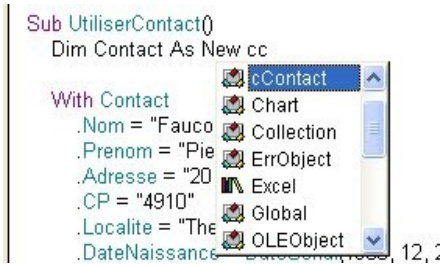
4.2.3. Utilisation de notre classe au sein d'un code

Pour utiliser un objet de type **cContact** au sein d'un code, nous utiliserons la même technique que pour notre type personnalisé.

En reprenant le code utilisé plus haut, nous avons deux modifications à réaliser pour utiliser notre classe plutôt que notre type personnalisé.
Voici le code:

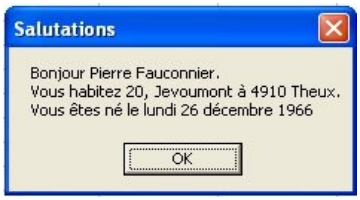
```
Sub UtiliserContact()
    Dim Contact As New cContact

    With Contact
        .Nom = "Fauconnier"
        .Prenom = "Pierre"
        .Adresse = "20, Jevoumont"
        .CP = "4910"
        .Localite = "Theux"
        .DateNaissance = DateSerial(1966, 12, 26)
        .Sexe = "masculin"
    End With

    AfficherMessageContact Contact
End Sub

Sub AfficherMessageContact(Contact As cContact)
    With Contact
        MsgBox "Bonjour " & .Prenom & " " & .Nom & "." & vbCrLf & _
            "Vous habitez " & .Adresse & " à " & .CP & " " & .Localite & "." & vbCrLf & _
            "Vous êtes né le " & Format(.DateNaissance, "dddd dd mmmm yyyy") & ".", vbOKOnly, "Salutations"
    End With
End Sub
```

La saisie semi-automatique permet, ici aussi, d'utiliser la liste des objets disponibles en fonction de la saisie.

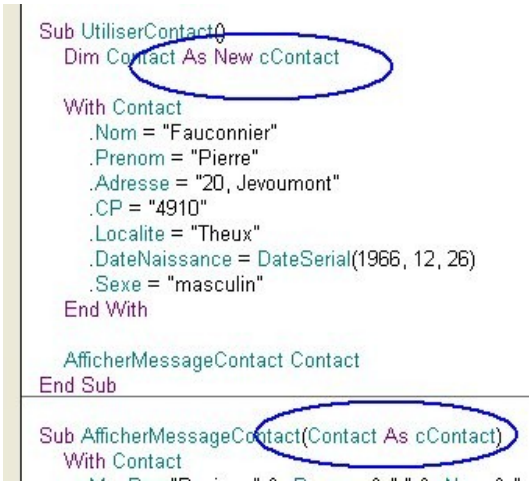


Et voici les modifications à réaliser:

Le résultat est toujours le même, à savoir l'affichage d'un message de bienvenue.



Attention! Bien que nous ayons déclaré nos variables publiques, il n'est pas possible de les utiliser telles quelles ailleurs que dans notre module de classe.
Ainsi, le code **Debug.Print DateNaissance** à l'extérieur de notre code provoquera une erreur de compilation.



4.2.4. Conclusions

Nous venons de voir comment créer notre première classe. Il n'y a pas de quoi fouetter un chat. Nous ne savons pas mieux contrôler les données avec ce code qu'avec l'utilisation d'un type personnalisé.

Néanmoins, nous avons vu comment créer un module de classe, lui affecter un nom qui sera celui de la classe que nous créons, et nous avons ajouté des variables "publiques" qui sont des propriétés en lecture/écriture disponibles pour les objets de notre classe.

4.3. Création de vraies propriétés

Déclarer des variables publiques au sein d'un module de classe est certes rapide, mais peu intéressant.
Pour pouvoir manipuler, contrôler et travailler avec les propriétés d'un objet, nous allons déclarer des ... propriétés.

4.3.1. Notions de propriétés en lecture/écriture, en lecture seule et en écriture seule

4.3.1.1. Propriétés en lecture/écriture

Une propriété en lecture/écriture est une propriété qui peut être lue ET modifiée.
Exemple: La propriété **DateNaissance** doit être en lecture pour pouvoir la récupérer et en écriture pour pouvoir la définir pour un contact particulier.

4.3.1.2. Propriétés en lecture seule

Une propriété en lecture seule ne peut être ni définie, ni modifiée.
Exemple: si je dote ma classe d'une propriété **Age**, elle sera en lecture seule car elle dépendra de la propriété **DateNaissance**, qui sera, elle, en lecture/écriture.

4.3.1.3. Propriétés en écriture seule

Une propriété pourrait être en écriture seule, et donc définissable mais non lisible.
Je ne vois pas d'exemple concret pour ce cas.

Retrouvez la suite de l'article de Pierre Fauconnier en ligne : Lien67

Installation et configuration d'un honeypot : Honeyd.

Ce tutoriel a pour but d'expliquer l'installation d'un honeypot sur une distribution Debian.

1. Prérequis

Le contenu de cet article a été rédigé en se basant sur une distribution Debian Etch. Les interfaces réseau utilisées seront lo (127.0.0.1) et ath0 (192.168.0.248). Suivant votre configuration n'oublié pas de les adapter au besoin.

2. Qu'est ce qu'un honeypot ?

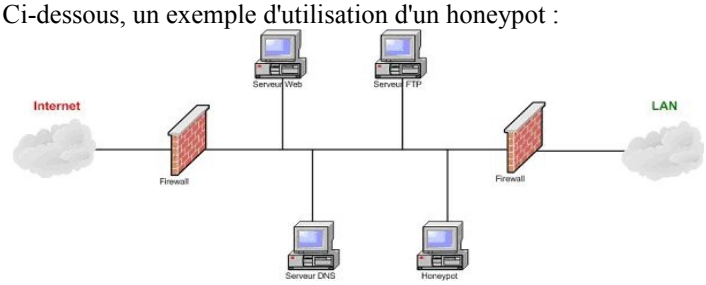
Un honeypot permet d'émuler des services sur une machine afin de simuler le véritable fonctionnement d'une machine de production. Ce système assure ainsi la surveillance du réseau par la collecte et le traitement des informations.

Les honeypots sont principalement divisés en deux catégories : les honeypots à faible interaction et les honeypots à forte interaction.

Les honeypots à faible interaction ne fournissent pas de véritables services, ils se contentent de les simuler par l'intermédiaire de script, comme Honeyd le propose. Ces honeypots ne posent que très peu de problèmes de sécurités.

A contrario, les honeypots à forte interaction fournissent des services bien réels mais non dédiés à la production. Ceux-ci sont très sensibles et pourraient mettre à mal la sécurité de votre entreprise. Il convient donc d'apporter une attention particulière à leur sécurisation.

Ci-dessous, un exemple d'utilisation d'un honeypot :



3. Installation

Dans cet article nous avons choisis d'utiliser un honeypot à faible interaction : Honeyd (Lien68).

La version d'Honeyd actuelle (Janvier 2008) fournie en paquet pour Debian Etch est la 1.5b. Pour installer Honeyd, nous allons utilisé Aptitude, en tapant la commande ci-dessous :

```
apt-get install honeyd
```

Les principaux fichiers installés sont les suivants :

```
/etc/init.d/honeyd
/etc/logrotate.d/honeyd
/etc/default/honeyd
/usr/lib/honeyd
/usr/share/honeyd
/usr/share/doc/honeyd
/usr/include/honeyd
/usr/bin/honeyd
```

Suivant l'utilisation de votre honeypot vous pourriez avoir besoin des paquets suivant :

- farp
- rrdtool

Pour plus d'informations sur les dépendances du paquet Honeyd vous pouvez visiter le site Web de Debian : http://packages.debian.org/etch/honeyd (Lien69)

4. Configuration

4.1. Fichier de configuration utilisé

Le fichier de configuration utilisé est similaire au fichier installé par défaut

```
############################
#Configuration du réseau virtuel utilisé
route entry 10.0.0.1
route 10.0.0.1 link 10.2.0.0/24
route 10.0.0.1 add net 10.3.0.0/16 10.3.0.1 latency 8ms
bandwidth 10Mbps
route 10.3.0.1 link 10.3.0.0/24
route 10.3.0.1 add net 10.3.1.0/24 10.3.1.1 latency 7ms
loss 0.5
route 10.3.1.1 link 10.3.1.0/24

############################
# Création d'un profil template
create template
set template personality "Microsoft Windows XP
Professional SP1"
set template uptime 1728650
set template maxfds 35
# For a complex IIS server
add template tcp port 80 "sh /usr/share/honeyd/scripts/
win32/web.sh"
add template tcp port 22
"/usr/share/honeyd/scripts/test.sh $ipsrc $dport"
add template tcp port 23 proxy $ipsrc:23
add template udp port 53 proxy 141.211.92.141:53
set template default tcp action reset
# Use this if you are not running honeyd as 'honeyd'
user:
# Debian-specific (use nobody = 65534 instead of 32767)
# set template uid 65534 gid 65534

############################
#Création d'un profil default
create default
set default default tcp action block
set default default udp action block
set default default icmp action block

############################
```

```
#Création d'un profil router
create router
set router personality "Cisco 1601R router running IOS 12.1(5)"
set router default tcp action reset
add router tcp port 22 "/usr/share/honeyd/scripts/test.sh"
#add router tcp port 23 "/usr/share/honeyd/scripts/router-telnet.pl"
add router tcp port 23 "/usr/share/honeyd/scripts/telnet/fake.pl"

############################
#On démarre les hotes en spécifiant l'IP et le profil
bind 10.3.0.1 router
bind 10.3.1.1 router
bind 10.3.1.12 template
bind 10.3.1.11 template
bind 10.3.1.10 template
set 10.3.1.11 personality "Microsoft Windows NT 4.0 SP3"
set 10.3.1.10 personality "IBM AIX 4.2"
```

4.2. Mise en place

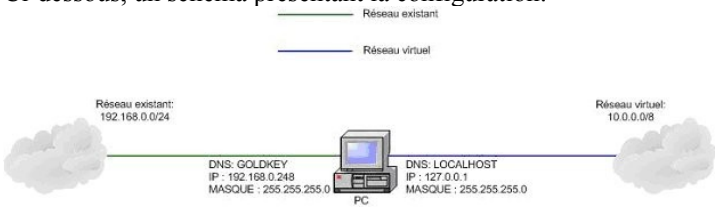
Nous allons utiliser la configuration par défaut proposer par Honeyd dans le fichier honeyd.conf:

```
route entry 10.0.0.1
route 10.0.0.1 link 10.2.0.0/24
route 10.0.0.1 add net 10.3.0.0/16 10.3.0.1 latency 8ms bandwidth 10Mbps
route 10.3.0.1 link 10.3.0.0/24
route 10.3.0.1 add net 10.3.1.0/24 10.3.1.1 latency 7ms loss 0.5
route 10.3.1.1 link 10.3.1.0/24
```

Pour faire fonctionner le réseau virtuel ci-dessus nous allons devoir déclarer une route (bien réelle) dans la table de routage pour l'atteindre. La passerelle utilisée pour cette route sera l'interface loopback (localhost) afin de ne pas perturber le réseau existant.

```
route add -net 10.0.0.0 netmask 255.0.0.0 gw localhost
```

Ci-dessous, un schéma présentant la configuration:



5. L'exécution

Pour tester notre configuration nous allons d'abord lancer Honeyd en mode interactif en lançant la commande ci-dessous dans une console :

```
honeyd -d -p /etc/honeypot/nmap.prints -l /var/log/honeypot/honeyd.log -f /etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
```

Détails des paramètres :

- -d lancer en mode interactif
- -p fichier des fingerprints
- -f fichier de configuration

```
gold@deb ~# honeyd -d -p /etc/honeypot/nmap.prints -l /var/log/honeypot/honeyd.log -f /etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
Honeyd V1.5b Copyright (c) 2002-2004 Niels Provos
honeyd[3676]: started with -d -p /etc/honeypot/nmap.prints -l /var/log/honeypot/honeyd.log -f /etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
honeyd[3676]: listening on lo: ip and (dst net 10.0.0.0/8)
```

Ok tout fonctionne...un petit ctrl+C pour arrêter la commande. Nous allons maintenant lancer notre démon Honeyd. Pour cela nous allons modifier le fichier de configuration du démon. Editer le fichier :

```
/etc/default/honeyd
```

Dans un premier temps, il faut modifié la constante RUN afin de pouvoir démarrer le démon :

```
RUN="yes"
```

Puis indiquer l'interface a utilisé et la plage d'adresses IP du réseau :

```
INTERFACE="ath0"
NETWORK=10.0.0.0/8
```

Puis démarrer le démon Honeyd avec la commande

```
/etc/init.d/honeyd start
```

S'il n'y pas d'erreur vous devez obtenir :

```
Starting Honeyd daemon: honeyd.
```

6. Les scripts d'émulation de services

Pour émuler un service fonctionnant sur une machine virtuelle, Honeyd permet l'utilisation de scripts. Ceux-ci peuvent être écrit an langage Perl ou même directement en SHELL. Des exemples de scripts sont fournis avec l'installation d'Honeyd. Les différents scripts sont situés dans le répertoire :

```
/usr/share/honeyd/script
```

Pour obtenir d'autres scripts vous pouvez visiter la rubrique "contributions" sur le site Web d'Honeyd : http://www.honeyd.org/contrib.php (Lien70)

Cela vous donnera sûrement de bonnes idées.

7. Les premiers pas

7.1. Vérification avec une commande ping

Vérifions si un de nos hôtes configurés répond à une commande ping. Pour plus de visibilité, nous allons lancer Honeyd en mode interactif :

```
gold@deb ~# honeyd -d -p /etc/honeypot/nmap.prints -l /var/log/honeypot/honeyd.log -f /etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
Honeyd V1.5b Copyright (c) 2002-2004 Niels Provos
honeyd[3753]: started with -d -p /etc/honeypot/nmap.prints -l /var/log/honeypot/honeyd.log -f /etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
honeyd[3753]: listening on lo: ip and (dst net 10.0.0.0/8)
```

Dans une autre console nous allons tenter de pinguer un hote configuré :

```
gold@deb ~# ping 10.3.0.1
PING 10.3.0.1 (10.3.0.1) 56(84) bytes of data.
64 bytes from 10.3.0.1: icmp_seq=1 ttl=63 time=10.0 ms
64 bytes from 10.3.0.1: icmp_seq=2 ttl=63 time=10.0 ms
64 bytes from 10.3.0.1: icmp_seq=3 ttl=63 time=10.0 ms
64 bytes from 10.3.0.1: icmp_seq=4 ttl=63 time=10.0 ms
64 bytes from 10.3.0.1: icmp_seq=5 ttl=63 time=10.0 ms
64 bytes from 10.3.0.1: icmp_seq=6 ttl=63 time=10.0 ms
64 bytes from 10.3.0.1: icmp_seq=7 ttl=63 time=10.0 ms
```

Honeyd a bien reçu notre ping :

```
gold@deb ~# honeyd -d -p /etc/honeypot/nmap.prints -l /
var/log/honeypot/honeyd.log -f
/etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
Honeyd V1.5b Copyright (c) 2002-2004 Niels Provos
honeyd[3753]: started with -d -p
/etc/honeypot/nmap.prints -l
/var/log/honeypot/honeyd.log -f
/etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
honeyd[3753]: listening on lo: ip and (dst net
10.0.0.0/8)
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
```

7.2. Vérification avec l'utilisation d'un script

Vérifions si un de nos hôtes configurés répond à l'appel d'un script. Pour plus de visibilité, nous allons a nouveau lancer Honeyd en mode interactif :

```
gold@deb ~# honeyd -d -p /etc/honeypot/nmap.prints -l /
var/log/honeypot/honeyd.log -f
/etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
Honeyd V1.5b Copyright (c) 2002-2004 Niels Provos
honeyd[3753]: started with -d -p
/etc/honeypot/nmap.prints -l
/var/log/honeypot/honeyd.log -f
/etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
honeyd[3753]: listening on lo: ip and (dst net
10.0.0.0/8)
```

Dans une autre console nous allons tenter d'accéder à un hôte configuré sur le port 23 :

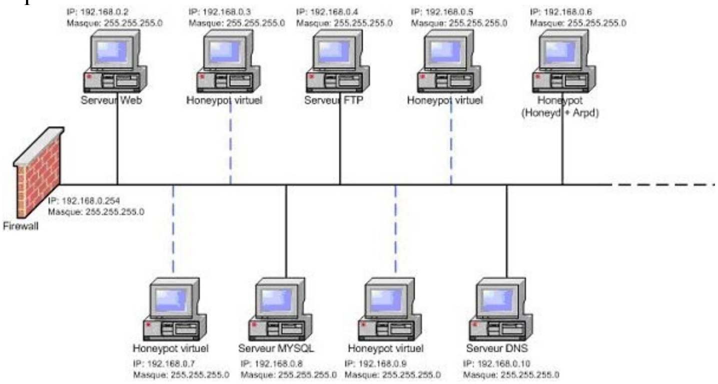
```
gold@deb ~# telnet 10.3.0.1 23
Trying 10.3.0.1...
Connected to 10.3.0.1.
```

Honeyd a bien reçu notre tentative d'accès au port 23 :

```
gold@deb ~# honeyd -d -p /etc/honeypot/nmap.prints -l /
var/log/honeypot/honeyd.log -f
/etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
Honeyd V1.5b Copyright (c) 2002-2004 Niels Provos
honeyd[3753]: started with -d -p
/etc/honeypot/nmap.prints -l
/var/log/honeypot/honeyd.log -f
/etc/honeypot/honeyd.conf -i lo 10.0.0.0/8
honeyd[3753]: listening on lo: ip and (dst net
10.0.0.0/8)
honeyd[3753]: Sending ICMP Echo Reply: 10.3.0.1 ->
192.168.0.249
honeyd[3753]: Connection request: tcp
(192.168.0.249:1584 - 10.3.0.1:23)
honeyd[3753]: Connection established: tcp
(192.168.0.249:1584 - 10.3.0.1:23) <->
/usr/share/honeyd/scripts/telnet/fake.pl
```

8. Pour aller plus loin

La configuration jusqu'ici a été conçue pour ne pas interagir avec le réseau existant. D'où l'utilisation de l'interface loopback (localhost) pour le routage vers notre réseau virtuel. Cependant grâce au démon Arpd vous pouvez interagir avec votre réseau existant. En effet celui-ci permet d'écouter les requêtes ARP et d'y répondre afin de simuler des machines inexistantes sur le réseau.



Attention cependant à son utilisation dans un réseau utilisant le protocole DHCP. Il est possible qu'Arpd interfère avec le serveur DHCP en permettant une réponse d'Honeyd au ping envoyé par le serveur DHCP pour déterminer si une adresse IP est libre sur le réseau.

Pour installer Arpd si cela n'a pas été fait au chapitre 1) taper la commande

```
apt-get install farpd
```

Le fichier de configuration du démon Arpd se situe dans le répertoire :

```
/etc/default/farpd
```

N'oublier pas de modifier les constantes suivantes :

```
INTERFACE="ath0"
NETWORK="192.168.0.0/24"
```

9. Conclusion

Honeyd est un honeypot à faible interaction complet et particulièrement souple grâce a son système de script. Attention tout de même car Honeyd n'a pas été conçu pour fonctionner dans un environnement de production mais plutôt dans un domaine de recherche afin d'améliorer la sécurité d'un réseau.

Retrouvez l'article de Goldkey en ligne : Lien71

Didacticiel sur Iptables, version 1.2.0

1. Préalables

Des connaissances préalables sur Linux/Unix sont nécessaires, en particulier l'écriture de scripts shell, la compilation de son propre noyau, et quelques notions sur son fonctionnement interne.

J'ai essayé autant que possible d'éliminer tous les préalables nécessaires pour comprendre pleinement ce document, mais en pratique il est inévitable de posséder un minimum de connaissances.

2. Introduction

2.1. Motivations

A l'origine, j'ai constaté un vide important dans les guides pratiques (<< Howto's >>) disséminés un peu partout, avec un manque d'informations notable sur les fonctions d'iptables et de

Netfilter pour les nouveaux noyaux Linux de la famille 2.4.x. Par conséquent, je vais tenter de répondre à des interrogations courantes concernant de nouvelles possibilités comme la correspondance d'état. La plupart du temps, les situations seront appuyées par un fichier d'exemple rc.firewall.txt (Lien72) que vous pourrez utiliser dans vos scripts /etc/rc.d/. Effectivement, ce fichier était à l'origine issu du guide pratique du camouflage, pour ceux d'entre-vous qui l'auraient reconnu.

Par la même occasion, il existe un petit script que j'ai écrit au cas où vous peiniez autant que moi lors de la configuration. Il est disponible sous le nom rc.flush-iptables.txt (Lien73).

2.2. Contenu

Initialement rédigé pour boingworld.com, qui fût un site de news consacré à Amiga/linux pour un petit nombre de personnes, y compris moi, il s'agissait d'un très petit didacticiel. En fonction du grand nombre de lecteurs et de commentaires que j'ai reçu, j'ai continué à écrire sur ce sujet. Le version originale faisait à peu près 10-15 pages au format A4 dans sa version imprimée. Un grand nombre de personnes m'ont aidé, pour la correction orthographique, bugs, etc. Au moment où j'écris ceci, le site http://iptables-tutorial.frozentux.net a enregistré plus de 600.000 connections.

Ce document est conçu pour vous guider pas-à-pas dans la méthode de configuration et il devrait vous aider à comprendre davantage le paquetage d'iptables. La plupart des exemples s'appuie sur le fichier rc.firewall, puisqu'il m'a semblé être un bon point de départ pour apprendre à se servir d'iptables. J'ai décidé de suivre simplement les chaînes fondamentales, et à partir de là, de poursuivre en approfondissant chacune des chaînes traversées dans l'ordre logique. Cette approche rend le didacticiel un peu plus difficile à suivre, mais elle a l'avantage d'être plus cohérente. Chaque fois que quelque-chose vous semble difficile à comprendre, replongez-vous dans ce didacticiel.

2.3. Termes spécifiques

Dans ce document, certains termes méritent des explications détaillées avant d'être abordés. Cette section cherche à couvrir les plus évidents et présente la façon dont ils sont utilisés ici.

Connexion - Se réfère généralement, dans ce document, à une série de paquets en relation entre eux. Ces paquets interragissent entre eux en établissant une sorte de connexion. Une connexion est en d'autres termes une série de paquets échangés.

DNAT - Traduction d'adresse réseau de destination ou << Destination Network Address Translation >>. Le DNAT fait référence à la technique de traduction de l'adresse IP de destination d'un paquet. On l'utilise conjointement avec du SNAT pour permettre à plusieurs hôtes de partager une même adresse IP connectée à Internet, et pour continuer à offrir des services de type serveur. Typiquement, il suffit d'attribuer des ports différents avec une adresse IP utilisable sur Internet, puis de signaler au routeur Linux où expédier le trafic.

Espace noyau - C'est plus ou moins l'opposé de l'espace utilisateur. Ceci implique les actions effectuées dans le noyau, et non en dehors du noyau.

Flux (<< Stream >>) - Ce terme fait référence à une connexion qui envoie et reçoit des paquets qui sont d'une certaine manière en relation les uns avec les autres. Typiquement, j'ai employé ce terme pour toute connexion qui envoie deux paquets ou plus dans les deux sens. Pour le protocole TCP, ce terme peut désigner une connexion qui envoie un paquet SYN, puis répond avec un autre de type SYN/ACK; mais il peut aussi désigner une connexion qui envoie un paquet SYN, puis répond avec un paquet ICPM de type hôte inaccessible (<< ICMP Host unreachable >>). Bref, j'ai souvent utilisé ce terme avec inexactitude.

SNAT - Traduction d'adresse réseau de source ou << Source Network Address Translation >>. Ce terme fait référence aux techniques mises en oeuvre pour traduire une adresse de source en une autre dans un paquet. Ceci permet à plusieurs hôtes de partager une même adresse IP connectée à Internet, c'est utile pour compenser le manque d'adresses IP disponibles avec le protocole IPv4 (mais IPv6 vient résoudre ce problème).

État - Ce terme fait référence à l'état d'un paquet, en accord avec la RFC 793 – Transmission Control Protocol (Lien74) ou avec les états utilisateur utilisés dans Netfilter/iptables. Notez que les états utilisés, en interne et en externe, ne respectent pas scrupuleusement la spécification de la RFC 793. La raison principale provient du fait que Netfilter a dû faire plusieurs hypothèses sur les connexions et les paquets.

Espace utilisateur (<< User space >>) - Cette expression permet de d'assigner tout ce qui a lieu à l'extérieur du noyau. Par exemple, la commande **iptables -h** s'exécute en dehors du noyau, alors que **iptables -A FORWARD -p tcp -j ACCEPT** se déroule (en partie) à l'intérieur, puisqu'une nouvelle règle est ajoutée à la table de règles.

Domaine de l'utilisateur - Voir Espace utilisateur.

Paquet - Une unité envoyée sur le réseau, contenant une partie en-tête et une partie de données. Par exemple, un paquet IP sur un paquet TCP. Dans les RFC (Request For Comments) un paquet n'est pas généralisé ainsi, au lieu de cela les paquets sont appelés datagrammes, tandis que les paquets TCP sont appelés segments. J'ai choisi de tout nommer paquet dans ce document pour simplifier.

Segment - Un segment TCP est à peu près la même chose qu'un paquet, c'est en fait un paquet TCP.

3. Rappel TCP/IP

Iptables est un outil d'apprentissage très puissant. Parmi d'autres choses, vous devez avoir une très bonne compréhension du protocole TCP/IP

Ce chapitre a pour but l'explication de ce que vous << devez savoir >> sur TCP/IP avant de commencer à utiliser iptables. Les choses que nous allons aborder concernent les protocoles IP, TCP, UDP et ICMP, leurs en-têtes, et l'utilisation générale de chacun des ces protocoles et comment ils sont corrélés entre eux. Iptables fonctionne au niveau des couches Internet et transport, et à cause de ça, ce chapitre mettra l'accent sur ces couches.

Iptables peut aussi fonctionner sur des couches plus hautes, comme la couche application. Cependant, il n'a pas été conçu pour ça, et ne devrait pas être utilisé pour ce genre d'usage. J'en parlerai d'avantage dans le chapitre Introduction au filtrage.

3.1. Couches TCP/IP

Comme déja établi, TCP/IP est multi-couches. Ceci indique que nous avons une fonctionnalité sur un niveau, et une autre à un autre niveau, etc. La raison pour laquelle nous avons toutes ces couches est très simple.

La raison principale est que l'architecture globale est très extensible. Nous pouvons ajouter de nouvelles fonctionnalités aux couches application, par exemple, sans avoir a réimplémenter l'ensemble du code TCP/IP, ou inclure une pile TCP/IP complète dans l'application. Ainsi il est inutile de réécrire chaque programme chaque fois que nous installons une nouvelle carte d'interface réseau.

Quand nous parlons de code TCP/IP lequel est intégré dans le noyau, nous parlons souvent de pile TCP/IP. La pile TCP/IP indique toutes les sous-couches utilisées, depuis la couche réseau jusqu'à la couche application.

Il existe deux architectures de base lorsque nous parlons de couches. Une des deux est le modèle OSI (Open System Interconnect) et consiste en 7 couches. Nous les verrons superficiellement ici, nous nous intéressons plus particulièrement aux couches TCP/IP. Cependant, sur un plan historique il est intéressant de le connaître, en particulier si vous travaillez avec plusieurs types de réseaux différents. Voir la liste dans le OSI Reference Model (Lien75).

1. Couche Application
2. Couche Présentation
3. Couche Session
4. Couche Transport
5. Couche Réseau
6. Couche Liaison
7. Couche Physique

Un paquet que nous envoyons, parcourt du sommet à la base cette liste, chaque couche ajoutant ses propres en-têtes au paquet, ce que nous appelons la phase d'encapsulation. Lorsque le paquet rejoint sa destination il parcourt en sens inverse la liste et les en-têtes sont supprimés du paquet, un à un, chaque en-tête donnant à l'hôte de destination toute l'information nécessaire jusqu'à ce que le paquet joigne l'application ou le programme pour lequel il était destiné.

Le second et plus intéressant standard est le protocole TCP/IP, comme indiqué dans la liste TCP/IP architecture (Lien76). Il n'y a pas d'accord universel en ce qui concerne le nombre de couches dans l'architecture TCP/IP. Cependant, on considère généralement qu'il y a de 3 à 5 couches disponibles, nous en verrons 4 pour simplifier.

1. Couche Application
2. Couche Transport
3. Couche Internet
4. Couche Réseau

Comme vous pouvez le voir, l'architecture du protocole TCP/IP est très proche du modèle OSI. De même qu'avec le modèle OSI, nous ajoutons et soustrayons les en-têtes pour chaque couche.

Par exemple, utilisons une des analogies les plus communes pour les machines modernes en réseau, la lettre par courrier postal. Chaque chose est effectuée par étape, identique en TCP/IP.

Vous désirez envoyer une lettre à quelqu'un en lui demandant comment il va, et qu'est-ce qu'il fait. Pour celà, vous devez poser des questions. Les questions seront situées à l'intérieur de la couche Application.

Après ceci vous écrirez les questions sur une feuille de papier que vous mettrez dans une enveloppe sur laquelle vous écrirez l'adresse de destination. Peut-être quelque chose comme :

Attn: John Doe

C'est l'équivalent de la couche Transport en TCP/IP.

À ce niveau nous écrivons l'adresse sur l'enveloppe, comme ceci :

V. Andersgardsgatan 2

41715 Gothenburg

Ça se passe dans l'analogie comme dans la couche Internet. La couche Internet contient les informations indiquant comment joindre le destinataire, ou l'hôte, dans un réseau TCP/IP. De la même façon qu'une enveloppe avec une adresse.

L'étape finale est de poster l'enveloppe dans une boîte aux lettres. Ce qui équivaut à peu près à envoyer un paquet dans la couche Réseau. La couche Réseau contient les fonctions et les routines pour accéder au réseau physique par lequel le paquet sera transporté.

Quand finalement nous recevons la lettre, nous la retirerons de l'enveloppe (décapsulation). La lettre que nous recevons peut parfois demander une réponse ou non. Dans certains cas il peut y avoir une réponse du destinataire, alors le destinataire devient expéditeur, et l'expéditeur devient destinataire.

Il est très important de comprendre que Iptables est spécifiquement construit pour travailler à l'intérieur des en-têtes des couches Internet et Transport. Il est possible de créer quelques filtres très basiques avec Iptables dans les couches Application et Réseau, mais il n'a pas été conçu pour cela, ni approprié.

Par exemple, si nous utilisons une correspondance de chaîne et l'apparions pour une chaîne spécifique dans le paquet, disons **get / index.html**. Ceci fonctionnera ? Normalement oui. Cependant, si la taille du paquet est très petite, cela ne marchera pas. La raison en est que Iptables est construit pour fonctionner sur une base *par paquet*, qui indique que si la chaîne est divisée en plusieurs paquets séparés, Iptables ne verra pas l'ensemble de la chaîne. Pour cette raison, il est mieux d'utiliser un proxy pour filtrer au niveau de la couche Application. Nous verrons ces problèmes en détail plus tard dans Introduction au filtrage IP (Lien77).

Étant donné que Iptables et Netfilter opèrent principalement sur les couches Internet et Transport, ce sont les couches sur lesquelles nous insisterons le plus dans ce chapitre. Sous la couche Internet, nous verrons presque exclusivement le protocole IP. Il existe quelques ajouts à ceci, comme, par exemple, le protocole GRE, mais ils sont très rares. À cause de tous ces facteurs nous nous concentrerons sur le protocole IP de la couche Internet, et TCP, UDP, ICMP de la couche Transport.

Le protocole ICMP est actuellement une sorte de mélange entre les deux couches. Il fonctionne dans la couche Internet, mais il possède exactement les mêmes en-têtes que le protocole IP, mais aussi quelques en-têtes supplémentaires. Nous verrons ceci plus en détail plus loin dans Caractéristiques ICMP (Lien78).

3.2. Caractéristiques IP

Le protocole IP résides dans la couche Internet, comme nous l'avons déjà dit. Le protocole IP est le protocole dans la pile TCP/IP qui permet à votre machine, routeur, switch, etc. de savoir où un paquet spécifique est envoyé. Ce protocole est véritablement le coeur de toute la pile TCP/IP, et la base de tout sur Internet.

Le protocole IP encapsule le paquet de la couche Transport avec l'information du procole de la couche Transport, ainsi que d'autres

informations utiles. Tout ceci, bien sûr, très précisément standardisé. Nous allons en parler dans ce chapitre.

Le protocole IP possède un couple de fonctionnalités de base qu'il doit être capable de traiter. Il doit être capable de définir le datagramme, lequel est le bloc de construction suivant créé par la couche Transport (ce qui en d'autres termes peut être TCP, UDP ou ICMP par esemple). Le Protocole IP définit aussi le système d'adressage Internet que nous utilisons aujourd'hui. Ceci indique que le protocole IP définit comment les hôtes peuvent se joindre entre eux, il indique aussi comment nous pouvons router les paquets, bien sûr. Les adresses dont nous parlons sont généralement appelées adresses IP. Usuellement, quand nous parlons d'adresses IP, nous parlons de chiffres avec des points (ex. 127.0.0.1). C'est principalement pour rendre l'adresse IP plus lisible pour l'oeil humain, car l'adresse IP est actuellement un champ de 32 bits de 1 et de 0 (127.0.0.1 pourrait désormais être lu comme 01111111000000000000000000000001 dans l'en-tête IP).

Le protocole IP doit aussi pouvoir décapsuler et encapsuler le datagramme IP (donnée IP) et envoyer ou recevoir le datagramme d'une couche Réseau, ou d'une couche Transport. Ceci peut sembler évident, mais parfois ce ne l'est pas. Au sommet de tout ça, il possède deux fonctions qu'il doit exécuter correctement, ce qui est particulièrement intéressant pour le pare-feu et le routage. Le protocole IP est responsable du routage des paquets depuis un hôte vers un autre. La plupart du temps sur des réseaux uniques, c'est un processus simple. Nous avons deux options différentes, soit le paquet est destiné au réseau local, soit passe par une passerelle. Mais lorsque vous commencez à travailler avec des pare-feux et des politiques de sécurité conjointement avec de multiples interfaces réseau et différentes routes, ce peut être casse-tête pour les administrateurs. La dernière des responsabilités du protocole IP est qu'il doit fragmenter et ré-assembler les datagrammes qui ont préalablement été fragmentés, ou qui nécessitent d'être fragmentés pour s'adapter à la taille du paquet pour la topologie du réseau où nous sommes connectés. Si ces fragments de paquet sont suffisamment petits, ils peuvent causer d'horribles maux de tête aux administrateurs réseau. Le problème est, qu'une fois qu'ils sont fragmentés, nous commençons à avoir des soucis pour lire même les en-têtes du paquet.

Dans les séries 2.4 du noyau Linux, et Iptables, ceci ne représente pas un problème pour la plupart des pare-feux Linux. Le système de traçage de connexion utilisé par Iptables pour la vérification d'état, la traduction d'adresse, etc. doit être capable de lire les paquets défragmentés. À cause de ça, conntrack défragmente automatiquement tous les paquets avant qu'ils rejoignent la structure netfilter/iptables dans le noyau.

Le protocole IP est aussi un protocole en mode datagramme, ce qui indique que IP ne "négocie" pas une connexion. Un protocole orienté-connexion, en d'autres termes, négocie une "connexion" (appelée *poignée de main*) et lorsque toutes les données ont été envoyées, stoppe la connexion. TCP est un exemple de ce genre de protocole, cependant, il est implémenté au sommet du protocole IP. Il y a plusieurs raisons pour lesquelles il n'est pas orienté-connexion, mais parmi d'autres, une poignée de main n'est pas nécessaire à ce moment ce qui ne ferait qu'ajouter du temps système. Comme vous pouvez le voir, envoyer une requête et ensuite attendre un moment pour la réponse est préférable à envoyer un paquet pour dire que nous voulons établir une connexion, ensuite recevoir le réponse nous disant que la connexion est ouverte, et finalement accuser réception que nous sommes au courant que la connexion est ouverte, et *alors* envoyer la requête, et après renvoyer un autre paquet pour couper la connexion et attendre une autre réponse.

IP est également connu comme un *protocole incertain*, c'est à dire, il ne permet pas de savoir si un paquet a été reçu ou non. Il reçoit simplement un paquet depuis la couche transport et le passe à la couche réseau, et ne fait rien de plus. il peut recevoir un paquet en retour, lequel passe de la couche réseau au protocole IP et ensuite à la couche transport. Cependant, il ne vérifie pas si c'est un paquet en réponse ou si le paquet a été reçu dans un autre but. La même chose s'applique en terme d'incertitude IP comme pour le mode datagramme, ce qui nécessitera l'envoi d'un paquet supplémentaire en retour pour chaque paquet envoyé. Par exemple, considérons une consultation de table DNS. Nous envoyons une requête DNS au serveur de nom. Si nous ne recevons pas de réponse, nous savons que quelque chose ne fonctionne pas et renvoyons une requête de consultation, mais dans l'usage normal nous envoyons une requête et obtenons une réponse en retour. Ajouter de la fiabilité à ce protocole signifierait que la requête nécessite deux paquets (une requête et une confirmation que le paquet a été reçu) et ensuite deux paquets pour la réponse (une réponse et un accusé-réception comme quoi le paquet a été reçu). En d'autres termes, nous doublons le nombre de paquets nécessaires, et bien sûr doublons le nombre de données à transmettre.

3.3. En-têtes IP

Comme vous avez pû le comprendre dans l'introduction sur le protocole IP, un paquet IP contient différentes parties dans l'en-tête. Celui-ci est méticuleusement divisé en plusieurs parties, et chaque partie de l'en-tête est aussi petite que possible pour faire ce travail, ceci pour limiter le temps système au minimum. Vous verrez la configuration exacte d'une en-tête IP dans l'image En-têtes IP (Lien79).

Comprenez que les explications des différents en-têtes sont très brèves et que nous ne parlerons que des bases de ceux-ci. Pour chaque type d'en-tête dont nous parlons, nous indiquerons aussi sa RFC correspondante que vous devriez lire pour une meilleure compréhension et des explications techniques du protocole en question. En note marginale, les RFC (Request For Comments), ont aujourd'hui une signification totalement différente dans la communauté Internet. Elles définissent et standardisent l'ensemble de l'Internet, par rapport à ce pourquoi elles ont été écrites à l'origine. Au départ, il ne s'agissait que de simples RFC dont le but était de poser des questions pour avoir l'avis des autres chercheurs.

Le protocole IP est décrit principalement dans RFC 791 – Internet Protocol (Lien80). Cependant, cette RFC est aussi mise à jour par la RFC 1349 – Type of Service in the Internet Protocol Suite (Lien81), rendue obsolète par RFC 2474 – Definition of the Differentiated Services Field (DS Field) in the Ipv4 and Ipv6 Headers (Lien82), mise à jour par RFC 3168 – The Addiction of Explicit Congestion Notification (ECN) to IP (Lien83) et RFC 3260 – New Terminology and Clarifications for Diffserv (Lien84).

Comme vous pouvez le voir, tous ces standards peuvent être un peu difficiles à suivre. Un tuyau pour trouver les différentes RFC est d'utiliser les fonctions recherche disponibles sur RFC-editor.org (Lien85). Dans le cas de IP, considérez que la RFC 791 est la RFC de base, toutes les autres sont simplement des mises à jour par rapport au standard. Nous parlerons de ceci plus en détail quand nous verrons les en-têtes spécifiques modifiés par ces nouvelles RFC.

Retrouvez la suite de l'article de'Oskar Andreasson en ligne : Lien86

Les derniers tutoriels et articles

De Java à Cocoa

Cet article, en plusieurs parties, permettra aux développeurs Java de pouvoir facilement passer à Cocoa et Objective-C.

1. Introduction

On présente souvent Objective-C comme un dérivé du C. C'est syntaxiquement exact, mais Objective-C tient beaucoup de son grand père Smalltalk, tout comme Java.

Java est aujourd'hui bien plus populaire que le C et attire certainement une audience un peu plus jeune.

Bien que je possède une longue expérience du C et du C++, il me semble plus approprié d'apprendre Objective-C en prenant Java comme point de référence. Tous deux sont de vrais langages à objets et tous deux offrent des comportements bien plus dynamiques que C++.

Autre différence primordiale entre C/C++ et Java/Objective-C : *les bibliothèques de développement*.

Alors que le C n'est généralement utilisé qu'avec la bibliothèque standard C. C++ fait de même avec la STL (Standard_Template_Library). Mais aucune de ces deux bibliothèques ne permet de développer une application complète.

Les bibliothèques de développement ne sont pas aussi fortement liées au langage que dans Java et Objective-C. C'est tellement vrai pour ce dernier qu'il est souvent confondu avec Cocoa, la bibliothèque de composants du Mac héritée de NeXTStep/OpenStep.

Java et Objective-C partagent donc nombre de caractéristiques.

Il me semble approprié de présenter le couple Objective-C et Cocoa en mettant en avant les similitudes avec Java et en soulignant si besoin leur différences.

2. Classes d'objets et héritage

Java comme Objective-C sont des langages à objets. Comme beaucoup des langages de cette famille, ils dépendent d'un système de classe.

Cet article présente la syntaxe de création d'une classe en Objective-C, en la comparant avec celle de Java.

2.1. Le rôle d'une classe

La classe est un moule à partir duquel on peut construire des objets : les instances d'une classe.

La classe couvre différents objectifs dont les plus significatifs sont :

- définir les données manipulées par un objet ;
- définir les services fournis par un objet ;
- masquer les détails d'implémentation en *encapsulant* les données et les services techniques ;

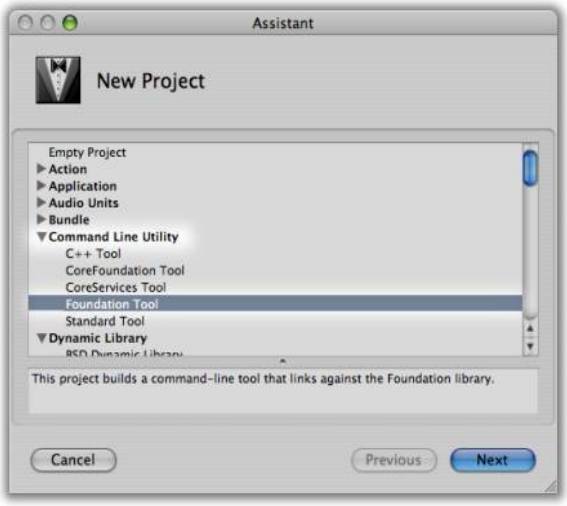
Donnée	Service
attribut, propriété, variable	méthode, message

Une classe peut atteindre ses objectifs en s'appuyant sur des outils techniques.

- des règles de visibilité pour définir qui peut voir ses attributs et méthodes ;
- l'héritage défini la généalogie d'une classe ;
- le *polymorphisme* pour se restreindre à utiliser une interface plutôt qu'une implémentation spécifique.

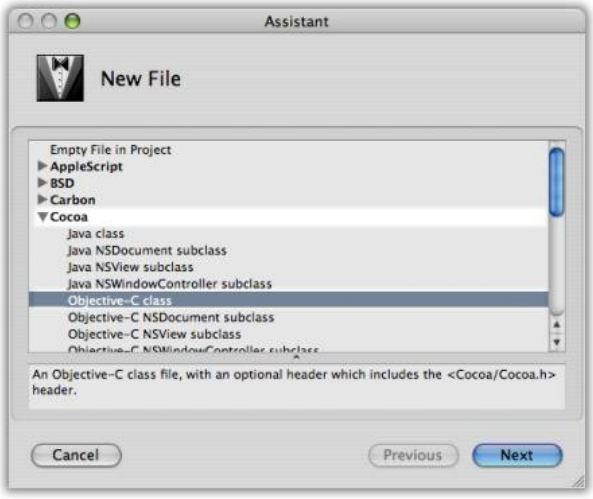
2.2. Comment créer une classe ?

La méthode la plus simple pour commencer est de créer un projet de type *Foundation Tool* dans la famille *Command Line Utility*.



Pour créer une classe utilisez l'assistant de création :

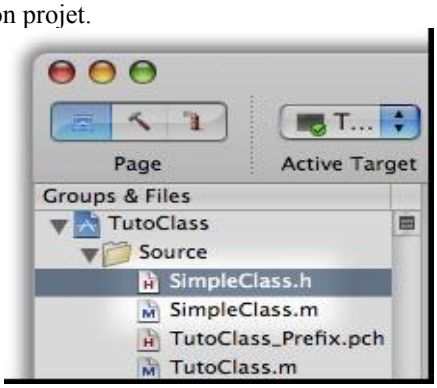
- Allez dans le menu File, entrée New File...
- Choisissez alors dans le groupe *Cocoa* l'entrée *Objective-C Class*.
- Donnez un nom à votre classe.



XCode vous crée un squelette de classe vide sous la forme de deux fichiers. Dans mon cas j'ai créé la classe *SimpleClass* :

- *SimpleClass.h* défini l'interface de ma classe.
- *SimpleClass.m* contiendra l'implémentation de ma classe.

Les nouveau fichiers apparaissent dans le groupe du projet sur la gauche. Je les ai déplacé dans le groupe " Source " pour mieux organiser mon projet.



La première différence par rapport à Java saute aux yeux : *Là où Java mélange dans un fichier unique interface et implémentation, Objective-C sépare les deux dans des fichiers indépendants*.

Cette séparation est un héritage direct du langage C au dessus duquel est construit Objective-C. L'avantage de cette solution est de pouvoir utiliser une classe sans disposer de son code source. Le fichier en-tête est suffisant pour le compilateur et décrit l'ensemble de l'interface publique de la classe et de ses objets.

L'interface d'une classe est définie dans le fichier d'en-tête *.h* (h pour header)

```
#import <Cocoa/Cocoa.h>

@interface SimpleClass : NSObject {
// Déclaration des attributs associés à la classe
}
// Déclaration des méthodes associées à la classe
@end
```

L'implémentation de la classe est dans un fichier *.m* (m pour module)

```
#import "SimpleClass.h"

@implementation SimpleClass

// Implémentation des différentes méthodes
// déclarées dans le fichier en-tête

@end
```

La directive #import indique au compilateur qu'il lui faut inclure le fichier en-tête indiqué. Ainsi, l'implémentation de notre classe importe le fichier en-tête qui lui est associé.

En Java le code aurait été réduit à un simple fichier *.java* :

```
public class SimpleClass {
// Définitions des attributs et méthodes
}
```

Deux outils indispensables à connaître :

1. dans la barre d'outils, le bouton pour passer dans l'éditeur pour le fichier sélectionné ;
2. Dans l'éditeur, en haut à gauche, un bouton en forme de double carré permet de passer de l'interface à l'implémentation et inversement.

En résumé, une classe se déclare ainsi :

Java	Objective-C

`class UneClasse`	`@interface UneClasse`
	`...`
	`@end`
	
	`@implementation UneClasse`
	`...`
	`@end`

2.3. Étendre une classe

Objective-C supporte le même modèle d'héritage que Java, à savoir une seule classe mère pour une classe donnée.

Contrairement à Java, il n'existe pas une hiérarchie unique pour la généalogie des classes. Au contraire, toute nouvelle classe peut former la racine d'une nouvelle arborescence de classes.

Le squelette de classe généré par XCode indique automatiquement que votre nouvelle classe dérive de la classe racine de Cocoa : NSObject. Hériter de NSObject n'est donc qu'une facilité et en aucun cas une obligation. Cependant, le type d'objet par défaut ne défini aucun comportement, ce qui fait que Objective-C dépend très fortement du framework applicatif utilisé.

```
@interface SimpleClass : NSObject {
 ...
```

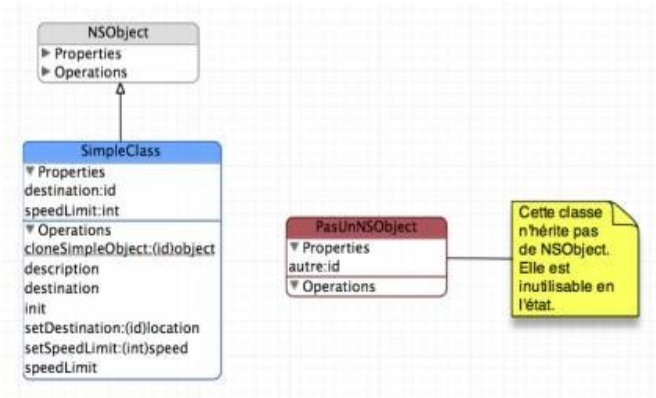
Ce qui en Java aurait pu s'écrire :

```
public class SimpleClass extends java.lang.Object {
 ...
```

Mais en Java, toute classe dérivant obligatoirement de la classe Object, cet héritage n'est généralement pas spécifié explicitement par les développeurs. Cette facilité devra être évitée à tout pris en Objective-C sous peine de voir son code ne pas réussir à fonctionner correctement.

Pour manipuler des objet le langage Objective-C utilise le type id qui est une référence sur un objet de type indéterminé.

Même si le langage permet des arbres de classes distincts de celui dont la racine est NSObject, il est fortement déconseillé de se construire sa propre hiérarchie de classes. Objective-C n'a que peu d'intérêt s'il n'est pas utilisé avec le framework Cocoa, et dériver ses classes à partir de NSObject et le meilleur moyen pour en profiter (Si vous poursuivez l'exploration de Cocoa vous verrez certainement que *NSObject* n'est pas la seule racine. Cocoa fournit également la classe *NSProxy*. En parler plus en détails sort du cadre de cet article d'introduction.)



Si l'héritage simple peut paraître trop limité par rapport à un modèle d'héritage multiple tel que celui proposé par C++, nous verrons que comme en Java, Objective-C utilise la notion d'interface. Cela permet sans compromettre la simplicité de supporter des modèles complexes.

Héritage	Java	Objective-C
Classe Racine	java.lang.object	NSObject (par facilité, mais pas forcément)
Déclaration	class MaClasse **extends** ClasseMere	@interface MaClasse : ClasseMere
Héritage Implicite	Oui, de java.lang.Object	Non

2.4. Pour finir

Nous savons maintenant définir une classe.

Cette classe ne nous sert pas à grand chose pour l'instant et il nous reste encore quelques étapes avant de la rendre réellement utilisable :

- définir des attributs pour les instances ;
- définir les méthodes (services) proposés par les objets ;
- construire et initialiser des instances de cette classe.

3. Définir les attributs d'une classe

Les variables présentent l'état d'un objet. On les appelle propriétés, attributs, données membre. Cocoa retiendra plutôt le terme propriété alors que Java réserve ce terme aux attributs des Java Beans.

Un objet fourni un jeu de services, mais pour pouvoir être implémentés ces services dépendent de l'état courant de l'objet.

C'est le rôle des attributs d'un objet que de définir son état à un instant donné.

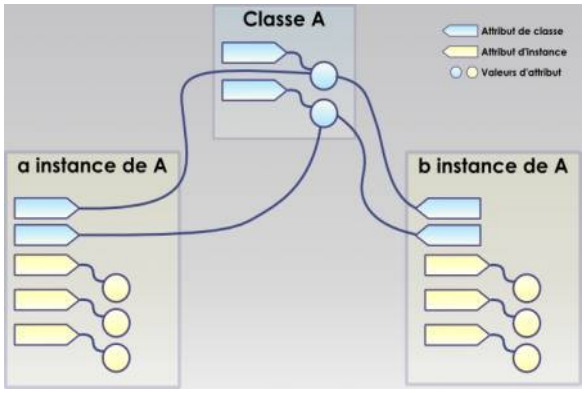
Nous allons donc voir ici comment nous allons définir les attributs de nos objets Objective-C.

Pour rappel, il est important clarifier ce qu'est une variable. *Une variable n'est qu'une étiquette qui est rattachée à une valeur.*

Un objet peut manipuler des variables d'instances et de classes :

- les variables de classes ont une valeur définie dans la classe.
- les variables d'instance ont une valeur défini dans l'objet.

Le schéma suivant représente cette structure :



On y voit clairement que si chaque instance partage le même jeu de noms de variables, ces variables ont toutes des valeurs différentes.

Au contraire, les variables de classes ont une valeur unique partagée entre toutes les instances de la classe.

3.1. Déclarer des variables d'instance

Les variables sont déclarées dans l'interface de la classe.

```
@interface MaClasse {
 // Déclaration des variables d'instance entre
 // les accolades
}
...
```

La syntaxe est celle des déclarations C normales. Pour plus de détails je vous renvoie donc à la documentation de référence disponible sur le site Apple ou dans votre installation d'XCode.

Notre classe *SimpleClass* va définir deux attributs :

- un identifiant vers un objet destination ;
- un nombre indiquant une limite de vitesse.

```
@interface SimpleClass : NSObject {
   id destination;
   int speedLimit;
}
```

En Objective-C, contrairement à Java, il est *obligatoire* d'utiliser explicitement la syntaxe C des pointeurs lorsqu'on déclare une variable de type objet :

```
(TYPE) * variable;
```

La seule exception étant l'utilisation du type id qui est implicitement un pointeur sur un type quelconque d'objet : un équivalent du void *, mais restreint aux objets.

En effet, comme nous le verrons en parlant du cycle de vie des objets, l'initialisation suivante n'est pas valide puisque les méthodes *alloc* et *init* retournent toutes deux un *id* qui est un type *pointeur* sur objet.

```
// Définition invalide
NSString chaine = [[NSString alloc] init];
```

Seule l'expression suivante permet de construire une chaîne :

```
// Définition correcte
NSString * chaine = [[NSString alloc] init];
```

3.2. Et les variables de classe ?

Pour un développeur Java, les attributs d'une classe se divisent en deux groupes :

- les *attributs des instances* de la classe : chaque instance de la classe partage le même ensemble de variables, mais

chaque instance leur associe des valeurs distinctes.
- les *attributs de la classe* : les attributs de classe sont partagés entre toutes les instances de la classe.

C'est le modèle classique présenté au début de cet article.

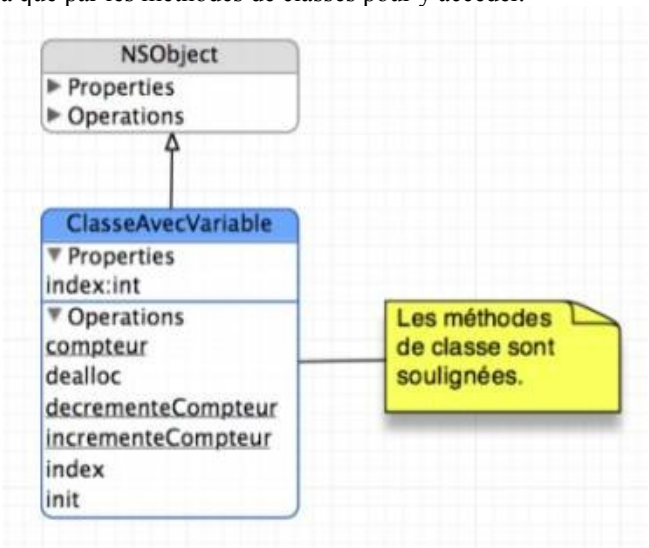
En Objective-C les variables de classe n'existent pas. Elles sont en fait inutiles car le langage C permet d'obtenir la même fonction par l'utilisation de variables privées dans un module.

Capable de posséder ses propres variables et sa propre portée, le fichier module joue le rôle de conteneur que la classe joue en Java.

Pour illustrer ce concept nous allons créer une nouvelle classe ClasseAvecVariable dont l'interface est la suivante :

```
@interface ClasseAvecVariable : NSObject {
   @private
   int index;
}
/* ... */
@end
```

La variable de classe n'est pas définie dans l'interface. Elle ne le sera que par les méthodes de classes pour y accéder.



Dans l'implémentation nous déclarons la variable de classe comme une variable globale mais dont la portée est restreinte au fichier source courant.

```
#import "ClasseAvecVariable.h"

static int compteur = 100; // Variable privée au module

@implementation ClasseAvecVariable
/* ... */
@end
```

Nous remarquerons que la variable étant d'un type simple nous pouvons combiner en une seule ligne la déclaration et la définition de la variable.

Pour les objets, il est possible de les initialiser avec une méthode d'initialisation de la classe. Ce sujet sera traité avec le cycle de vie des objets.
Cette même classe pourrait être définie de façon plus concise en Java comme suit :

```
public class ClasseAvecVariable {
 private static int compteur = 100; // Variable de
classe
 private int index; // Variable d'instance
 // ...
}
```

Il est tout de même important de remarquer que si l'utilisation de variables privée à un module semble similaire à une variable de classe, nous n'avons pas vraiment une stricte équivalence avec Java :
- Comme en Java un variable de classe si elle est surchargée, elle n'en est pas pour autant capable de polymorphisme.
- Contrairement au Java, une variable de classe est systématiquement privée. Les variables de classes ne sont ni protégées, ni publiques. Une classe peut cependant être associée à des variables globales publiques déclarées dans son fichier en-tête et définies dans le module d'implémentation.

Il est possible de restreindre la visibilité d'une variable externe à un module, mais cela implique de définir un fichier en-tête spécifique et complique peut-être inutilement le code.

Variable de classe	Java	Objective-C
Déclaration	Source Java	Fichier en-tête
Visibilité	Publique, protégée, ou privée	Privée
Syntaxe	static TYPE var ;	static TYPE var ;

3.3. Visibilité des attributs

En Java, mais aussi en Objective-C, les variables de classes peuvent profiter de trois niveaux de visibilité :
1. *Privée*, la variable n'est visible que dans la classe qui la déclare.
2. *Protégée*, la variable est visible dans la classe qui la déclare ainsi que dans ses classes dérivées.
3. *Publique*, la variable est visible à partir de toute partie de code.

Java et Objective-C définissent la visibilité d'une variable membre différemment :
- Java défini la visibilité pour *chaque variable* en ajoutant un qualificatif de portée à la déclaration.
- Objective-C défini des *sections* dans la déclaration des variables, chaque section définie le niveau de portée des variables qui y seront déclarées.

En cela, Objective-C ressemble d'avantage à C++ qu'à Java.

Les mots clefs utilisés sont similaires.

Portée	Java	Objective-C
Privée	private	@private
Protégée	protected	@protected
Publique	public	@public

3.4. Utilisation des attributs

Dans une méthode de la classe, les attributs sont accessibles comme des variables normales. Elles sont dans la portée de la classe et à se titre sont directement accessibles.

Nous pourrions donc écrire la méthode suivante pour la classe SimpleClass :

```
(id) init {
 [super init];
 destination = location;
 speedLimit = limit;
 return self;
}
```

Si l'on n'est pas dans la portée de la classe, par exemple lorsqu'on accède à une variable publique, il faut utiliser la notation classique du C pour accéder à une valeur d'un type structuré. Les objets étant déclarée par des pointeurs sur le type, c'est l'opérateur flèche d'indirection de pointeur qui sera utilisé :

```
MaClasse * objet;
// ...
objet->variable = valeur;
```

3.5. Pour finir

Nous sommes maintenant capable d'ajouter des données à nos objet en leur associant des variables d'instance ou de classe.

Mais nos classes resteront inertes tant qu'elles ne proposeront pas des services. Ces services sont fournis par l'intermédiaire de méthodes répondant à des messages. C'est le sujet de la troisième partie de cette introduction à Objective-C.

4. Définir les méthodes d'une classe

Les méthodes permettent de masquer les variables et exposent l'interface publique d'un objet ou d'une classe.

Comme Java, Objective-C connaît les méthodes d'instance et de classes, mais son interprétation du modèle objet diffère de Java.

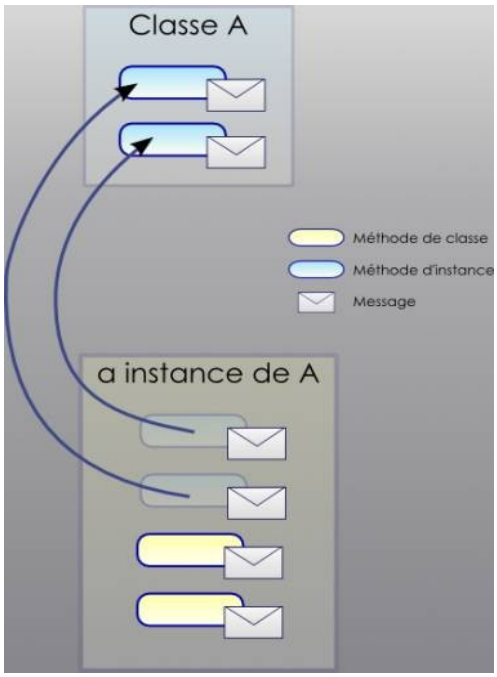
4.1. Généralités

Java parle volontiers de méthode et Objective-C adopte plutôt la terminologie de message.

Cette différence dépasse le vocabulaire et reflète une réalité concrète. Les services des objets ne sont effectivement pas des appels vers des fonctions, mais des envois de messages qui sont ensuite traités par une fonction. La différence est subtile, mais peut aider à comprendre l'esprit du langage.

Là ou Java se rapproche de C++, Objective-C est vraiment dans l'esprit de SmallTalk par cette idée de message. Le coût est limité au moteur du langage (Objective-C est implémenté au dessus de C mais utilise un petit moteur, le runtime, pour implémenter les fonctionnalités dynamiques du langage.).

Le diagramme ci-dessous illustre comment un message peut-être géré par un objet.



- La classe déclare les messages qu'elle gère.
- La classe déclare les messages gérés par ses instances.
- Les instance ou la classe reçoivent des messages correspondant à ce que la classe à déclaré.
- Les messages de classes sont gérés par l'implémentation fournie par la classe.
- Les messages d'instance sont gérés par l'implémentation fournie par l'objet.

C'est en partie cette dernière caractéristique qui explique que les méthodes, contrairement aux variables, ne bénéficie pas de règles de visibilité. Les méthodes sont soit des méthodes d'instances, soit des méthodes de la classe.

Mais comme pour les variables de classes, le module peut servir à implémenter des messages privés. Il suffit de ne définir une méthode répondant à un message dans l'implémentation, sans déclarer ce message au niveau de l'interface de l'objet. Mais même si le message n'est pas visible il peut toujours être envoyé par un objet qui n'a pas connaissance de l'implémentation privée. Je ne pense pas que cette pratique soit à conseiller.

1. Java défini des méthodes sur les objets ou les classes. Une méthode Java est une fonction qui utilise la portée (l'espace de nommage) de la classe qui la défini.
2. Objective-C défini les services d'une classe par un ensemble de messages auxquels elle est capable de répondre. La classe répond à un message en appelant une méthode.

Le mode de fonctionnement d'Objective-C permet d'introduire une forte dynamique dans le langage sans imposer le surcoût important d'un système d'introspection.

Une classe peut ainsi répondre à une message sans forcément implémenter la méthode qui y répondra. Il est très facile de déléguer la gestion du message à un objet tiers en lui faisant simplement suivre le message.

4.2. Déclaration des messages

Comme toutes les déclarations d'une classe les messages gérés par une classe sont déclarés dans le fichier en-tête de la classe.

```
@interface SimpleClass : NSObject {
        // ...
}

// Constructeur
- (id) init;

// Affichage
- (NSString *) description;

// Accès et modification sur 'destination'
- (id) destination;
- (void) setDestination: (id) location;

// Accès et modification sur 'speedLimit'
- (int) speedLimit;
- (void) setSpeedLimit: (int) speed;

// Une méthode de classe
+ (id) cloneSimpleObject: (id) object;

@end
```

Une déclaration se décompose en trois types d'éléments :

1. Un préfixe Un signe moins - pour les messages gérés par les instances. Un signe plus + pour les messages gérés par la classe.
2. Le type de retour de chaque message.

3. Le nom du message
4. Si le message accepte un paramètre il est déclaré en indiquant son type et un identifiant

Un message peut accepter plusieurs paramètres, dans ce cas la syntaxe varie largement du Java. Par exemple, pour déclarer un constructeur qui permette de définir les valeurs des variables d'instance on peut utiliser la déclaration suivante :

```
- (id) initWithDestination: (id) location
andSpeedLimit: (int) limit;
```

Dans cet exemple, le nom du message est initWithDestination:andSpeedLimit : et les deux paramètres sont location et limit. L'avantage de cette syntaxe est simple : le nom des méthodes, s'il est correctement choisit, permet d'expliciter le rôle de chaque paramètre.

En Java, le rôle de chaque paramètre ne peut être déduit que si votre éditeur offre une complétion automatique et un accès à la Javadoc au fils de l'écriture de votre code. Cela est utile lorsque vous écrivez votre code, mais absent lorsque vous le relisez.

En Objective-C, le langage permet d'exposer directement le rôle des paramètres, que vous soyez en pleine écriture du code ou en train de relire un source vieux de plusieurs mois. *Le code source est auto-suffisant pour assurer sa compréhension.*

4.3. Définition des méthodes

Les méthodes qui répondent aux message déclarés dans l'interface de la classe sont définies dans le module d'implémentation de la classe.

Pour chaque message on défini une méthode :

```
@implementation SimpleClass
//...

- (int) speedLimit
{
   return speedLimit;
}


- (void) setSpeedLimit: (int) speed
{
   speedLimit =  speed;
}
//...
@end
```

Si vous de fournissez pas de méthode pour chaque message déclaré votre classe est abstraite.

4.4. Appel de méthode

Pour invoquer un service sur un objet ou une classe il suffit de lui envoyer le message correspondant :

```
// Message sans paramètre
[object message];
/* Appel le message messageD:situeA:
* avec les paramètres expediteur et lieu
*/
[objet messageDe: expediteur situeA: lieu]
```

Voici par exemple l'implémentation de la méthode description pour notre classe simple :

```
- (NSString *) description;
{
   NSMutableString * descr =
       [[NSMutableString alloc] initWithString:
@"SimpleClass"];
   if ( nil != [self destination] ) {
       [descr appendFormat: @" destination %@", [self
destination]];
   }
   [descr appendFormat: @" speedLimit %d", [self
speedLimit]];
   return descr;
}
```

4.5. Pour finir

Nos objets viennent d'acquérir la capacité de proposer des services. Ils sont donc largement plus utiles.

Ils nous manque un dernier élément : comprendre le cycle de vie des objets pour pouvoir les animer correctement. C'est le sujet de la quatrième et dernière partie de cette introduction.

5. Initialiser classes et objets

Nous avons vu comment déclarer une classe et comment préparer l'implémentation des méthodes qui seront associées aux messages acceptés par la classe.

Pour aller plus loin il faut être capable de construire proprement des instances d'une classe, et si besoin initialiser les variables globales utilisées par une classe.

Cette partie a donc pour but de détailler :

- l'initialisation d'une classe ;
- le mécanisme de construction d'un objet ;
- la fin de vie d'un objet.

5.1. Initialisation de la classe

Les classes ont parfois besoin d'une phase d'initialisation qui leur soit propre. Ceci est particulièrement vrai si l'on utilise des variables de classes.

Java propose un mécanisme simple pour initialiser les membres d'une classe : *l'initialisation au moment de la déclaration* :

```
class UneClasse {
 // Une constante
 static public final String LABEL = "unLabel";

 // Un objet
 static private java.net.URL baseURL =
   new java.net.URL ( "http://www.google.com/" ) ;
}
```

Cette méthode, si elle fonctionne très bien pose un problème pour la création d'objets plus complexes. Dans le cas de la construction de la variable baseURL on se trouve dans un cas où une alternative doit être trouvée pour pouvoir gérer les cas d'erreurs.

En effet, la construction de l'URL peut générer une exception qui ne sera pas traitée par la classe, mais qui sera interceptée par le chargeur de classe (ClassLoader). Pour peu que la classe ne soit pas initialisée explicitement, une telle erreur peut être ensuite plus délicate à identifier correctement.

Une bien meilleure solution est proposée en Java : l'initialisation dans le bloc static.

Retrouvez la suite de l'article de Sylvain Gamel en ligne : Lien87

Les derniers tutoriels et articles

IRT : un ray tracer interactif - Partie 1

Nous allons commencer par créer un raytracer générique, qui sera optimisé par la suite.

1. Objectifs

Les objectifs de ce premier tutoriel sont simples :

- Définir une architecture souple
- Proposer une sur-couche en Python pour simplifier les tests
- Effectuer un ray tracing sans reflet
- Optimiser le code

Une bibliothèque matricielle doit être utilisée, une même classe servira pour chaque élément, vecteur point ou couleur, mais avec une taille différente si nécessaire. Pour bien séparer les responsabilités, un **typedef** sera utilisé pour chacun de ces éléments.

Pour le moment, les entrailles ne cette bibliothèque ne sont pas exposées, mais l'optimisation future de ce ray tracer passera par cette étape. Il est tout de même nécessaire d'utiliser une bibliothèque rapide.

Un vecteur aura pour type **Vector3df**, un point **Point3df**, une normale **Normal3df** et une couleur **Color**. Tous ces éléments et les autres classes vivront dans le namespace **IRT**.

2. Les classes de base

Les classes de bases, outre les rayons qui seront exposés tout de suite, sont les chevilles ouvrières du ray tracer. Les primitives décrivent quels sont les objets qui peuvent être présents dans une scène, leur couleur à un endroit, s'ils sont transparents, s'ils reflètent la lumière (mat ou brillant), ... La scène quant à elle agglomère ces primitives et permet de chercher de manière plus ou moins rapide ces éléments. Enfin, le ray tracer lui-même projette des rayons dans la scène et analyse les résultats.

Avant de commencer, la classe Ray va être exposée. Il s'agit en réalité d'une aggrégation d'un vecteur et d'un point de départ.

ray.h

```
  class Ray
  {
  private:
    Point3df origin_;
    Vector3df direction_;

  public:
    _export_tools Ray(const Point3df& origin, const
Vector3df& direction);

    _export_tools ~Ray();

    const Point3df& origin() const
    {
      return origin_;
    }
    Point3df& origin()
    {
      return origin_;
    }
    const Vector3df& direction() const
    {
      return direction_;
    }
    Vector3df& direction()
    {
      return direction_;
    }
  };
```

Ma macro **_export_tools** permet d'exporter les symboles dans une bibliothèque dynamique sous Windows. Seuls les éléments accessibles de l'extérieur de la bibliothèque seront exportés.

ray.cpp

```
Ray::Ray(const Point3df& origin, const Vector3df&
direction)
    :origin_(origin), direction_(direction)
  {
  }

  Ray::~Ray()
  {
  }
```

Cette classe ne nécessite pas plus de commentaires, tout le travail est réalisé par la bibliothèque matricielle.

2.1. Les primitives

Une primitive est tout objet qui pourra être contenu dans la scène, il s'agit donc d'une classe virtuelle pure, d'une interface, qui définit certaines fonctions virtuelles pures. Il s'agit du test de l'intersection ainsi que du calcul de la couleur :

```
  class Primitive
  {
  public:
    Primitive(){}

    virtual ~Primitive()
    {}

    virtual bool intersect(const Ray& ray, float& dist)
= 0;

    virtual void computeColorNormal(const Ray& ray,
float dist, Color& color, Vector3df& normal) = 0;
  };
```

Une primitive simple qui va être exposée est la classe **Sphere** qui permet d'instancier une sphère de couleur uniforme. Le calcul le plus complexe est celui de l'intersection entre un rayon et la sphère :

```
  Sphere::Sphere(const Point3df& center, float radius)
    :center(center), radius(radius), color(1.f)
  {}
```

```
  void Sphere::setColor(const Color& color)
  {
    this->color = color;
  }

  bool Sphere::intersect(const Ray& ray, float& dist)
  {
    float A = 1.f;
    float B = (ray.direction() * (ray.origin() -
center));
    float C = norm2(ray.origin() - center) - radius *
radius;

    float delta = (B * B - A * C);

    if(delta < 0.f)
      return false;
    float disc = sqrt(delta);
    if(dist = - (B + disc) < 0.)
      dist = - (B - disc);
    return true;
  }
```

Le principe est de considérer le rayon comme une droite paramétrée par **dist**. A partir de cette équation, on rajoute l'équation de la sphère que le point doit respecter. Une équation du second degré doit alors être résolue et on prend la racine positive la plus petite en compte.

Pour simplifier les calculs, on considère que le vecteur **direction** du rayon est unitaire.

Il ne reste plus qu'à calculer la couleur si besoin est ainsi que la normale, ce qui est très simple pour une sphère connaissant son centre et le point sur la sphère que le rayon touche.

```
  void Sphere::computeColorNormal(const Ray& ray, float
dist, Color& color, Normal3df& normal)
  {
    Vector3df collide(ray.origin() +
dist*ray.direction());
    normal = collide - center;
    normal *= 1./(sqrt(norm2(normal)));

    color = this->color;
  }
```

La normale ne sera pas utilisée ici, tout comme la couleur est constante sur la scène.

2.2. La scène

Une scène regroupe donc les primitives. Il est nécessaire de pouvoir en ajouter, en ôter ou en récupérer une. De plus, c'est la scène qui est chargée de trouver un objet touché par un rayon à l'aide de la fonction **getFirstCollision()**.

simple_scene.h

```
  class SimpleScene
  {
  private:
    std::vector<Primitive*> primitives;

  public:
    _export_tools SimpleScene();

    _export_tools ~SimpleScene();

    _export_tools Primitive* getPrimitive(unsigned long
index);

    _export_tools Primitive* removePrimitive(unsigned
long index);

    _export_tools long getFirstCollision(const Ray&
ray, float& dist);

    _export_tools bool addPrimitive(Primitive*
primitive);
  };
```

Le code se charge de détruire les primitives qui sont stockées dans la scène. A partir du moment où une primitive est ajoutée dans la scène, celle-ci se charge de sa durée de vie. En revanche, si elle est ôtée de la scène, l'utilisateur doit la détruire lui-même.

Voici le code de la scène :

```
  SimpleScene::SimpleScene()
    :primitives()
  {
  }

  SimpleScene::~SimpleScene()
  {
    for(std::vector<Primitive*>::const_iterator it =
primitives.begin(); it != primitives.end(); ++it)
      delete *it;
  }

  Primitive* SimpleScene::getPrimitive(unsigned long
index)
  {
    return primitives[index];
  }
```

La primitive qui est ôtée de la scène est renvoyée à l'utilisateur

```
  Primitive* SimpleScene::removePrimitive(unsigned long
index)
  {
    std::vector<Primitive*>::iterator it =
primitives.begin();
    std::advance(it, index);
    Primitive* primitive = *it;
    primitives.erase(it);
    return primitive;
  }
```

Ceci est le coeur de la scène. Dans le cas simple présenté ici, chaque primitive est testée, et si le rayon touche la primitive, la distance du départ du rayon à la primitive est mis à jour dans **dist** et l'indice de la primitive est retourné.

```
  long SimpleScene::getFirstCollision(const Ray& ray,
float& dist)
  {
    float min_dist = std::numeric_limits<float>::max();
    long min_primitive = -1;

    for(std::vector<Primitive*>::const_iterator it =
primitives.begin(); it != primitives.end(); ++it)
    {
      float dist;
      bool test = (*it)->intersect(ray, dist);

      if(test)
      {
        min_primitive = it - primitives.begin();
        min_dist = dist;
      }
    }

    if(min_primitive == -1)
      return -1;
```

```
    else
    {
      dist = min_dist;
      return min_primitive;
    }
  }

  bool SimpleScene::addPrimitive(Primitive* primitive)
  {
    if(std::find(primitives.begin(), primitives.end(),
primitive) != primitives.end())
      return false;

    primitives.push_back(primitive);
    return true;
  }
```

Une primitive ne peut être ajoutée deux fois dans la scène. Si l'insertion échoue, **false** est renvoyé par **addPrimitive()**.

2.3. Le ray tracer

Un ray tracer peut utiliser *a priori* n'importe quelle scène à partir du moment où l'on peut obtenir la primitive touchée par un rayon.

raytracer.h

```
  class Raytracer
  {
  private:
    void generateRay(unsigned long x, unsigned long y,
Ray& ray) const;

    void computeColor(const Ray& ray, Color& color)
const;

    void updateParameters();
  public:
    _export_tools Raytracer(unsigned long pixelWidth,
unsigned long pixelHeight, float width, float height,
float depth);

    _export_tools ~Raytracer();

    _export_tools void draw(float* screen) const;

    _export_tools void setViewer(float width, float
height, const Vector3df& origin, const Vector3df&
direction);

    _export_tools void setResolution(unsigned long
pixelWidth, unsigned long pixelHeight);

    _export_tools void setScene(SimpleScene* scene);

  private:
    Point3df origin;
    Vector3df direction;
    unsigned long pixelWidth;
    unsigned long pixelHeight;
    float width;
    float height;
    float depth;

    float precompWidth;
    float precompHeight;

    SimpleScene* scene;
  };
```

Outre les fonctions clés du ray tracer, cette classe propose une interface permettant idéalement de positionner la caméra n'importe où (même si le code n'en est pas encore capable à l'heure actuelle).

Avant de commencer, quelques détails sur l'écran sont nécessaires. Il possède une hauteur et une largeur avec une certaine précision par dimension. L'écran se situe en plus à une certaine distance (profondeur) de l'observateur (la caméra). Ces éléments doivent être stockés dans la classe.

```
  Raytracer::Raytracer(unsigned long pixelWidth,
unsigned long pixelHeight, float width, float height,
float depth)
    :origin(0.), direction(0.), pixelWidth(pixelWidth),
pixelHeight(pixelHeight), width(width), height(height),
depth(depth)
  {
    direction(2) = 1;
    updateParameters();
  }

  Raytracer::~Raytracer()
  {
  }
```

Une fois ces éléments sauvegardés, une fonction de précalcul est appelée pour optimiser le temps d'exécution.

```
  void Raytracer::generateRay(unsigned long x, unsigned
long y, Ray& ray) const
  {
    ray.direction()(0) = precompWidth * (x - pixelWidth
/ 2.f);
    ray.direction()(1) = precompHeight * (y -
pixelHeight / 2.f);
    ray.direction()(2) = depth;

    ray.direction() *= 1./
(sqrt(norm2(ray.direction())));
  }
```

Cette méthode construit un rayon unitaire sur place. A partir de la position de l'observateur, un rayon vers chaque pixel de l'écran est généré.

Générer le rayon sur place évite une copie de celui-ci à chaque étape : on gagne donc du temps.

Le code ici devrait tenir compte de la rotation de la caméra par rapport à la direction de l'écran.

```
  void Raytracer::draw(float* screen) const
  {
    Ray ray(origin, direction);
    for(unsigned long j = 0; j < pixelHeight; ++j)
    {
      for(unsigned long i = 0; i < pixelWidth; ++i)
      {
        generateRay(i, j, ray);
        Color color(0.);

        computeColor(ray, color);

        for(unsigned int k = 0; k < nbColors; ++k)
          screen[nbColors * (j* pixelWidth + i) + k] =
color(k);
      }
    }
  }
```

Cette méthode permet de lancer le rayon dans une direction et de récupérer la couleur associée.

```
  void Raytracer::setScene(SimpleScene* scene)
  {
```

```
    this->scene = scene;
  }

  void Raytracer::setResolution(unsigned long
pixelWidth, unsigned long pixelHeight)
  {
    this->pixelWidth = pixelWidth;
    this->pixelHeight = pixelHeight;

    updateParameters();
  }

  void Raytracer::setViewer(float width, float height,
const Vector3df& origin, const Vector3df& direction)
  {
    this->width = width;
    this->height = height;
    this->origin = origin;
    this->direction = direction;

    updateParameters();
  }
```

Ces différentes méthodes mettent à jour le ray tracer.

```
  void Raytracer::computeColor(const Ray& ray, Color&
color) const
  {
    float dist;
    long index = scene->getFirstCollision(ray, dist);
    if(index < 0)
      return;

    Normal3df normal;
    Primitive* primitive = scene->getPrimitive(index);
    primitive->computeColorNormal(ray, dist, color,
normal);
  }
```

Le coeur du ray tracer teste s'il a touché une primitive et si c'est le cas récupère la couleur de l'objet. Par la suite, c'est lui aussi qui gèrera les rayons réfléchis et transmis.

```
  void Raytracer::updateParameters()
  {
    precompWidth = width / pixelWidth;
    precompHeight = height / pixelHeight;
  }
```

3. Encapsulation Python

Pour simplifier les tests et le développement, une partie du code est écrit en Python.

3.1. Création du wrapper

`Le wrapper` est un module SWIG qui s'appellera **IRT**. Pour gérer les vecteurs en Python, on utilisera Numpy (Lien88).

Voici le coeur du module :

```
%{
#define SWIG_FILE_WITH_INIT
#define PY_ARRAY_UNIQUE_SYMBOL PyArray_API
%}
%include "numpy.i"
%init %{
import_array();
%}

%include "constraints.i"

%module(package="IRT", docstring="Python interface to
the Interactive RayTracer") IRT

%nodefaultdtor;
%nodefaultctor;

%include "primitives.i"
%include "simple_scene.i"
%include "raytracer.i"
```

Rien d'extraordinaire, à part qu'il n'existera pas d'encapsulation des constructeurs par défaut ou de recopie : cela évitera des bêtises.

3.1.1. Gestion des primitives

Cette partie est la plus complexe. En effet, il faut encapsuler chaque primitive, mais aussi gérer le fait que la scène gère la destruction d'une primitive si elle se trouve dans la scène.

La gestion des primitives par la scène est primordiale, en réalité. Si une primitive est ajoutée et que la scène ne gère pas sa destruction, la primitive sera détruite par Python à la sortie de la fonction de création de la scène, ce qui est gênant. Cela explique aussi la nécessité de gérer l'objet lorsqu'il est retiré de la scène.

```
%{
#include "primitives.h"
%}

%typemap(in)
    (IRT::Vector3df&)
    (PyArrayObject* array=NULL, int is_new_object=0)
{
  array =
obj_to_array_contiguous_allow_conversion($input,
NPY_FLOAT, &is_new_object);
  if (!array || !require_dimensions(array, 1) ||
(array->dimensions[0] != 3)) SWIG_fail;

  $1 = new IRT::Vector3df((float*) array->data);
}
%typemap(freearg)
    (IRT::Vector3df&)
{
  if (is_new_object$argnum && array$argnum)
Py_DECREF(array$argnum);
}

%typemap(in)
    (IRT::Color& color)
    (PyArrayObject* array=NULL, int is_new_object=0)
{
  array =
obj_to_array_contiguous_allow_conversion($input,
NPY_FLOAT, &is_new_object);
  if (!array || !require_dimensions(array, 1) ||
(array->dimensions[0] != IRT::nbColors)) SWIG_fail;

  $1 = new IRT::Color((float*) array->data);
}
%typemap(freearg)
    (IRT::Color&)
{
  if (is_new_object$argnum && array$argnum)
Py_DECREF(array$argnum);
}
```

On voit l'apparition de **IRT::nbColors**. Cette constante indique en réalité le nombre de couleurs gérées par le ray tracer.

Lorsqu'un tableau Numpy est passé en paramètre à une fonction prenant un vecteur en paramètre, une référence sur cet objet est

acquise et n'est libérée qu'à la fin de la fonction par le typemap **freearg**.

```
%typemap(in) IRT::Primitive*
{
  if ((SWIG_ConvertPtr($input,(void **)(&$1),
$1_descriptor, SWIG_POINTER_EXCEPTION |
SWIG_POINTER_DISOWN)) == -1) SWIG_fail;
}

%typemap(out) IRT::Primitive*
{
  $result = SWIG_NewPointerObj($1, $1_descriptor,
SWIG_POINTER_OWN);
}
```

Ces typemaps sont chargés de déléguer la gestion de l'objet ou de récupérer la gestion lors de l'interaction avec la scène.

```
namespace IRT
{
  class Primitive
  {
  public:
    ~Primitive();
  };

  class Sphere: public Primitive
  {
  public:
    Sphere(IRT::Vector3df& ray, float dist);
    ~Sphere();
    void setColor(IRT::Color& color);
  };
}
```

Il est à noter que toute l'interface n'est pas accessible en Python. Cela est normal, il n'est pas nécessaire et peu judicieux de laisser l'accès aux entrailles du ray tracer à tout un chacun.

3.1.2. Gestion de la scène

La scène doit permettre d'ajouter ou de retirer un élément. Seuls ces éléments sont pertinents.

```
%{
#include "simple_scene.h"
%}

%apply Pointer NONNULL{IRT::Primitive*};

namespace IRT
{
  class SimpleScene
  {
  public:
    SimpleScene();
    ~SimpleScene();
    IRT::Primitive* removePrimitive(unsigned long
index);
    bool addPrimitive(IRT::Primitive* primitive);
  };
}
```

Si un pointeur **NULL** est passé à la fonction, le wrapper SWIG lèvera lui-même une exception. Cette contrainte est contenue dans le fichier **constraints.i** ajouté dans **IRT.i**.

3.1.3. Gestion du ray tracer

```
%{
#include "raytracer.h"
%}

%typecheck(SWIG_TYPECHECK_DOUBLE_ARRAY)
  (float* INPLACE_ARRAY)
{
  $1 = is_array($input) &&
PyArray_EquivTypenums(array_type($input),NPY_FLOAT);
}
%typemap(in)
  (float* INPLACE_ARRAY)
  (PyArrayObject* array=NULL, int is_new_object=0)
{
  array =
obj_to_array_contiguous_allow_conversion($input,
NPY_FLOAT, &is_new_object);
  $1 = ($1_ltype) array->data;
}

namespace IRT
{
  class Raytracer
  {
  public:
    Raytracer(unsigned long pixelWidth, unsigned long
pixelHeight, float width, float height, float depth);
    ~Raytracer();

    void draw(float* INPLACE_ARRAY);
    void setResolution(unsigned long pixelWidth,
unsigned long pixelHeight);
    void setScene(IRT::SimpleScene* scene);
  };
}
```

Le ray tracer doit pouvoir récupérer une scène, dessiner celle-ci et décider de sa résolution. Les point le plus critique reste le dessin de la scène, ici implémenté grâce à un typemap **in** qui donne accès au tableau Numpy sous-jacent.

3.2. Création d'une scène et mesure de temps

Pour tester la vitesse d'exécution et en effectuer un profil, il est nécessaire de tout d'abord créer la scène et le raytracer puis dans une autre fonction ou méthode appeler la méthode **draw()** du raytracer.

sample.py

```
import IRT
import numpy

class Sample(object):
  def __init__(self):
    self.raytracer = IRT.Raytracer(800, 600, 8., 6.,
40.)
    self.scene = IRT.SimpleScene()

    sphere = IRT.Sphere(numpy.array((0, 0, 80.),
dtype=numpy.float32), 2.)
    sphere.setColor(numpy.array((0., 0., 1.),
dtype=numpy.float32))
    self.scene.addPrimitive(sphere)
    sphere = IRT.Sphere(numpy.array((2., 1., 60.),
dtype=numpy.float32), 1.)
    sphere.setColor(numpy.array((1., 0., 0.),
dtype=numpy.float32))
    self.scene.addPrimitive(sphere)
    sphere = IRT.Sphere(numpy.array((-2., -1., 60.),
dtype=numpy.float32), 1.)
    sphere.setColor(numpy.array((0., 1., 0.),
dtype=numpy.float32))
    self.scene.addPrimitive(sphere)

    self.raytracer.setScene(self.scene)
```

```
  def __call__(self, screen):
    self.raytracer.draw(screen)
```

L'écran est de taille 800x600, avec une résolution de 0.1 par pixel. Trois sphères seront dessinées devant l'écran, une en bleu, une en vert et une dernière en rouge. La méthode **__call__()** sera utilisée pour mesurer le temps d'exécution ou créer le profil.

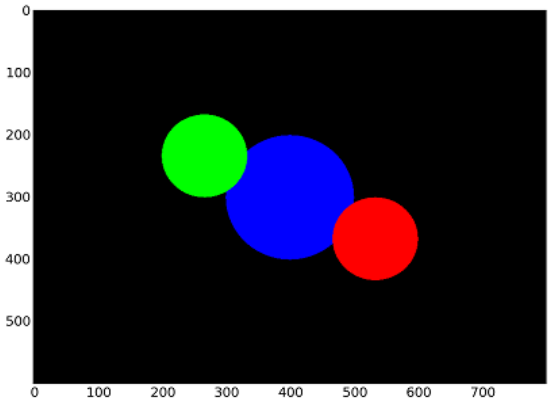
timeit_image.py

```
import timeit

setup="""import numpy
import sample
import pylab

s = sample.Sample()

screen = numpy.zeros((600, 800, 3),
dtype=numpy.float32)
"""

timer = timeit.Timer('s(screen)', setup=setup)

print timer.timeit(10)
```

4. Résultats

4.1. Résultat brut

Voici le résultat affiché par le ray tracer :

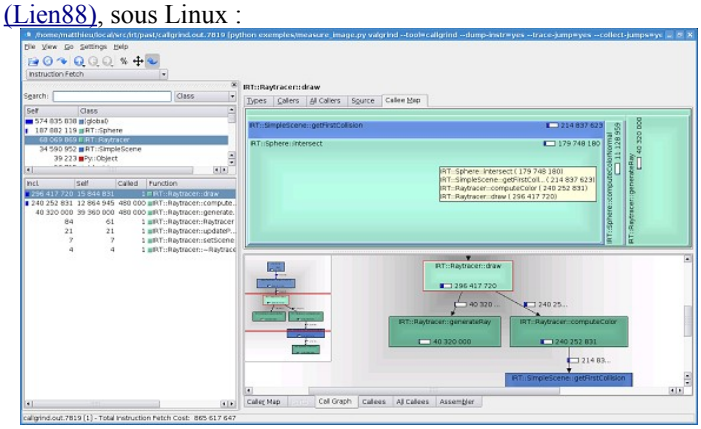


La scène avec 3 sphères

4.2. Profil du raytracer

Avant de parler profil, le ray tracer trace cette simple scène en 140 ms sur un Xeon 3.8GHz sous Linux avec GCC 4.1.2. Sous Windows, avec un Athlon 64 3000+ (1.8GHz),il faut près d'une demi-seconde pour dessiner cette scène. Malheureusement, les outils de profil sont plus rares sous Windows, il est donc plus complexe de déterminer un profil pour déterminer le point critique.

Voici donc ce profil en mode optimisé calculé par Valgrind (Lien88), sous Linux :



La majorité du temps est passé dans la fonction de calcul d'intersection, et plus particulièrement dans les fonctions de la bibliothèque matricielle d'addition ou de soustraction. Cela peut se voir en observant les coûts par instruction, grâce à KCacheGrind. Ces trois éléments sont pour le moment le principal frein de notre fonction. En particulier, aucune instruction SSE n'est générée par aucun des compilateurs testés (MSVC, GCC ou Intel).

5. Conclusion

Je vous ai présenté une base de travail pour un ray tracer interactif. Les prochains tutoriels proposeront :

- la gestion des reflets et des lumières
- le suréchantillonnage
- ...

Retrouvez l'article de Matthieu Brucher en ligne : Lien89

Les derniers tutoriels et articles

Introduction aux Réseaux de Neurones Artificiels Feed Forward

Cet article présente la théorie des réseaux de neurones feed-forward en partant de la description d'un seul neurone puis en arrivant progressivement au modèle de perceptron monocouche puis multicouche. Il présente également 3 algorithmes d'apprentissage, concernant les 2 derniers modèles.

1. Introduction

Plongeons-nous dans l'univers de la reconnaissance de formes. Plus particulièrement, nous allons nous intéresser à la reconnaissance des chiffres (0, 1, ..., 9). Imaginons un programme qui devrait reconnaître, depuis une image, un chiffre. On présente donc au programme une image d'un "1" manuscrit par exemple et lui doit pouvoir nous dire "c'est un 1". Supposons que les images que l'on montrera au programme soient toutes au format 200x300 pixels. On aurait alors 60000 informations à partir desquelles le programme déduirait le chiffre que représente cette image. L'utilisation principale des réseaux de neurones est justement de pouvoir, à partir d'une liste de *n* informations, pouvoir déterminer à laquelle des *p* classes possibles appartient cette liste.

De façon plus générale, un réseau de neurone permet l'approximation d'une fonction. Ici, il s'agit d'une fonction de classification : à chaque n-uplet d'informations en entrée la fonction associe une classe.

Dans la suite de l'article, on notera $X=(x_i)_{1 \leq i \leq n}$ un vecteur dont les composantes sont les *n* informations concernant un exemple donné. Dans l'exemple de la reconnaissance de formes, un exemple est une image représentant plus ou moins approximativement un chiffre. C'est un objet descriptible par *n* informations. De plus, on notera C_1, C_2, ..., C_p les *p* classes. Dans l'exemple précédent, les classes correspondent aux chiffres.

Voyons maintenant d'où vient la théorie des réseaux de neurones artificiels.

2. Une origine... biologique !

Comment l'homme fait-il pour raisonner, parler, calculer, apprendre... ? Comment s'y prendre pour créer une intelligence artificielle ? Deux types d'approches ont été essentiellement explorées :

Approches adoptée en recherche en Intelligence Artificielle

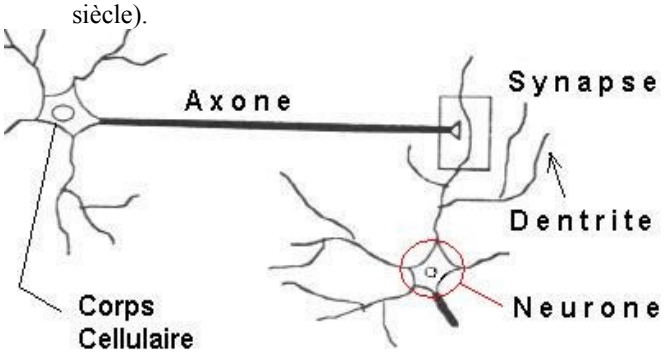
- procéder d'abord à l'analyse logique des tâches relevant de la cognition humaine et tenter de les reconstituer par programme. C'est cette approche qui a été privilégiée par l'Intelligence Artificielle symbolique et la psychologie cognitive classiques. Cette démarche est étiquetée sous le nom de cognitivisme.
- puisque la pensée est produite par le cerveau ou en est une propriété, commencer par étudier comment celui-ci fonctionne. C'est cette approche qui a conduit à l'étude des réseaux de neurones formels. On désigne par connexionnisme la démarche consistant à vouloir rendre compte de la cognition humaine par des réseaux de neurones.

La seconde approche a donc mené à la définition et à l'étude de réseaux de neurones formels qui sont des réseaux complexes d'unités de calcul élémentaires interconnectées. Il existe deux courants de recherche sur les réseaux de neurones : un premier motivé par l'étude et la modélisation des phénomènes naturels d'apprentissage pour lequel la pertinence biologique est importante ; un second motivé par l'obtention d'algorithmes efficaces ne se préoccupant pas de la pertinence biologique. Nous nous plaçons du point de vue du second groupe. En effet, bien que les réseaux de neurones formels aient été définis à partir de considérations biologiques, pour la plupart d'entre eux, et en particulier ceux étudiés dans ce cours, de nombreuses caractéristiques biologiques (le temps, la mémoire...) ne sont pas prises en compte. Toutefois, nous donnons, dans la suite de cette introduction, un bref aperçu de quelques propriétés élémentaires de neurophysiologie qui permettent au lecteur de relier neurones réels et neurones formels. Nous donnons ensuite un rapide historique des réseaux de neurones.

La physiologie du cerveau montre que celui-ci est constitué de cellules (les neurones) interconnectées. Les principales étapes de cette découverte sont :

Découvertes

- Van Leuwenhook (1718) : première description fidèle de ce qu'on appellera plus tard les axones ;
- Dutrochet (1824) : observation du corps cellulaire des neurones ;
- Valentin : découverte des dendrites ;
- Deiters (1865) : image actuelle de la cellule nerveuse ;
- Sherington (1897) : les synapses ;
- les neuro-transmetteurs (première moitié du 20ème siècle).



Modèle du neurone biologique

Les neurones reçoivent les signaux (impulsions électriques) par des extensions très ramifiées de leur corps cellulaire (les dendrites) et envoient l'information par de longs prolongements (les axones). Les impulsions électriques sont régénérées pendant le parcours le long de l'axone. La durée de chaque impulsion est de l'ordre d'1 ms et son amplitude d'environ 100 mV.

Les contacts entre deux neurones, de l'axone à une dendrite, se

font par l'intermédiaire des synapses. Lorsqu'une impulsion électrique atteint la terminaison d'un axone, des neuromédiateurs sont libérés et se lient à des récepteurs post-synaptiques présents sur les dendrites. L'effet peut être excitateur ou inhibiteur.

Chaque neurone intègre en permanence jusqu'à un millier de signaux synaptiques. Ces signaux n'opèrent pas de manière linéaire : il y a un effet de seuil.

Voici quelques informations à propos des neurones du cerveau humain :

Informations diverses sur les neurones du cerveau humain

- le cerveau contient environ 100 milliards de neurones.
- on ne dénombre que quelques dizaines de catégories distinctes de neurones.
- aucune catégorie de neurones n'est propre à l'homme (cela serait trop beau !).
- la vitesse de propagation des influx nerveux est de l'ordre de 100m/s, c'est à dire bien inférieure à la vitesse de transmission de l'information dans un circuit électronique.
- on compte de quelques centaines à plusieurs dizaines de milliers de contacts synaptiques par neurone. Le nombre total de connexions est estimé à environ 10^{15} .
- la connectique du cerveau ne peut pas être codée dans un "document biologique" tel l'ADN pour de simples raisons combinatoires. La structure du cerveau provient donc en partie des contacts avec l'environnement. L'apprentissage est donc indispensable à son développement.
- le nombre de neurones décroit après la naissance. Cependant, cette affirmation semble remise en question.
- on observe par contre une grande plasticité de l'axone, des dendrites et des contacts synaptiques. Celle-ci est surtout très importante après la naissance (on a observé chez le chat un accroissement des contacts synaptiques de quelques centaines à 12 000 entre le 10ème et le 35ème jour). Cette plasticité est conservée tout au long de l'existence, bien qu'affaiblie lors du processus de vieillesse ; on parle alors de "rigidité synaptique".
- les synapses entre des neurones qui ne sont pas simultanément actifs sont affaiblis puis éliminés.
- il semble que l'apprentissage se fasse par un double mécanisme : des connections sont établies de manière redondantes et aléatoires puis seules les connexions entre des neurones simultanément actifs sont conservés (phase de sélection) tandis que les autres sont éliminés. On parle de stabilisation sélective.

3. Un neurone seul

Un neurone est, comme vous l'avez vu, l'unité élémentaire de traitement d'un réseau de neurones. Il est connecté à des sources d'information en entrée (d'autres neurones par exemple) et renvoie une information en sortie. Voyons comment tout cela s'organise.

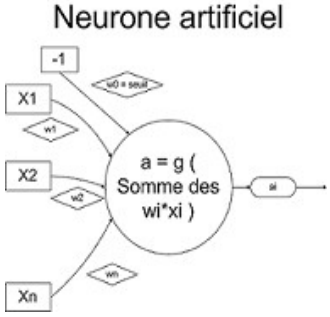
3.1. Entrées

On note $(x_i)_{1 \leq i < k}$ les k informations parvenant au neurone. De plus, chacune sera plus ou moins valorisée vis à vis du neurone par le biais d'un poids. Un poids est simplement un coefficient w_i lié à l'information x_i . La i -ème information qui parviendra au neurone sera donc en fait $w_i * x_i$. Il y a toutefois un "poids" supplémentaire, qui va représenter ce que l'on appelle le *coefficien de biais* . Nous le noterons w_0 et le supposerons lié à une information $x_0 = -1$. Nous verrons plus tard son utilité, dans la section **Fonction d'activation** .

Le neurone artificiel (qui est une modélisation des neurones du cerveau) va effectuer une somme pondérée de ses entrées putôt que de considérer séparément chacune des informations. On définit une nouvelle donnée, *in* , par :

$$in = \sum_{i=0}^{k} w_i \times x_i = (\sum_{i=1}^{k} w_i \times x_i) - w_0$$

C'est en fait cette donnée-là que va traiter le neurone. Cette donnée est passée à la fonction d'activation, qui fait l'objet de la prochaine section. C'est d'ailleurs pour ça que l'on peut parfois appeler un neurone une **unité de traitement** .



3.2. Fonction d'activation

La fonction d'activation, ou fonction de transfert, est une fonction qui doit renvoyer un réel proche de 1 quand les "bonnes" informations d'entrée sont données et un réel proche de 0 quand elles sont "mauvaises". On utilise généralement des fonctions à valeurs dans l'intervalle réel [0,1]. Quand le réel est proche de 1, on dit que l'unité (le neurone) est **active** alors que quand le réel est proche de 0, on dit que l'unité est **inactive** . Le réel en question est appelé la **sortie** du neurone et sera noté a . Si la fonction d'activation est linéaire, le réseau de neurones se réduirait à une simple fonction linéaire.

En effet, si les fonctions d'activations sont linéaires, alors le réseau est l'équivalent d'une régression multi-linéaire (méthode utilisée en statistiques) . L'utilisation du réseau de neurone est toutefois bien plus intéressante lorsque l'on utilise des fonctions d'activations non linéaires.

En notant g la fonction d'activation, on obtient donc la formule donnant la sortie d'un neurone :

$$a = g(in) = g(\sum_{i=0}^{k} w_i \times x_i)$$

Remarquez que le coefficient de biais est inclus dans la somme, d'où la formule plus explicite :

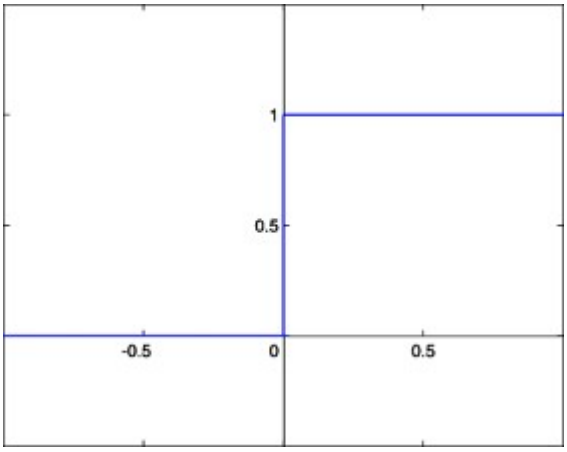
$$a = g(in) = g((\sum_{i=1}^{k} w_i \times x_i) - w_0)$$

Il y a bien sûr beaucoup de fonctions d'activations possibles, c'est à dire répondant aux critères que nous avons donnés, toutefois dans la pratique il y en a principalement 2 qui sont utilisées :

Les 2 fonctions de transfert les plus utilisées

- La fonction de Heaviside
- La fonction sigmoïde

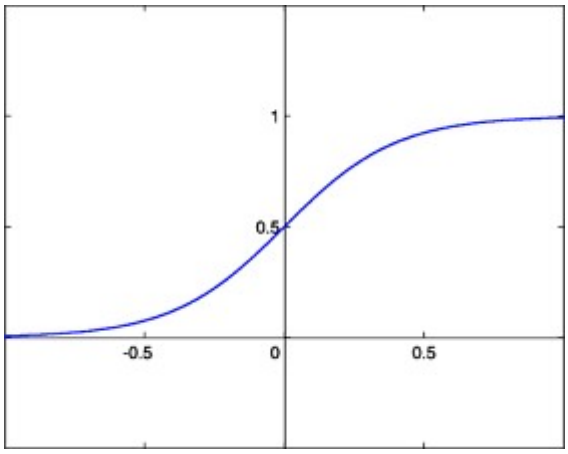
La fonction **de Heaviside** est définie par : $\forall x \in R, g(x) = 1$ si $x \geq 0$, 0 sinon.



Graphe de la fonction de Heaviside

La fonction **sigmoïde** est quand à elle définie par :

$$\forall x \in R, g(x) = \frac{1}{1+e^{-x}}$$



Graphe de la fonction Sigmoïde

La fonction sigmoïde présente l'avantage d'être dérivable (ce qui va être utile par la suite) ainsi que de donner des valeurs intermédiaires (des réels compris entre 0 et 1) par opposition à la fonction de Heaviside qui elle renvoie soit 0 soit 1. Toutefois, les deux fonctions possèdent un seuil. Celui de la fonction de Heaviside est en x = 0 et vaut 1 alors que celui de la fonction sigmoïde est en 0 également mais vaut 1/2.

Revenons à notre neurone et demandons-nous quand est-ce que le seuil est atteint, ou dépassé dans le cas de la fonction sigmoïde. Il est dans tous les cas atteint quand *in* vaut 0.

$$in = 0 \Leftrightarrow \sum_{i=0}^{k} w_i \times x_i = 0 \Leftrightarrow (\sum_{i=1}^{k} w_i \times x_i) - w_0 = 0 \Leftrightarrow \sum_{i=1}^{k} w_i \times x_i = w_0$$

C'est là qu'intervient réellement le **coefficient de biais** . Nous voyons donc que l'on atteint le seuil de la fonction d'activation lorsque la somme pondérée des informations d'entrée vaut le **coefficient de biais** . De plus :

$$in \geq 0 \Rightarrow g(in) \geq seuil$$

où seuil vaut 1 si g est la fonction seuil, 0 si g est la fonction sigmoïde. Les propriétés énoncées ci-dessus sont vraies grâce à la croissance des fonctions d'activations.

Maintenant, notons $\mathbf{W} = (w_i)_{1 \leq i \leq n}$ le vecteur dont les composantes sont les poids et $\mathbf{X} = (x_i)_{1 \leq i \leq n}$ le vecteur dont les composantes sont les informations d'entrées du neurone. Avec cette notation on obtient :

$$W \cdot X = 0$$

Ceci définit un hyperplan d'un espace de dimension *n* . En fait, l'espace dont il est question est l'espace des informations d'entrées. De la même manière que dans notre monde nous décrivons des points avec nos 3 coordonnées souvent notées (x,y,z), dans l'espace des informations d'entrée on note les coordonnées (x_1 , ..., x_n). Un hyperplan est un espace de dimension n-1. Dans notre monde, l'espace est de dimension 3 (car 3 coordonnées) et un hyperplan est donc un espace de dimension 2. En dimension 3, un hyperplan est donc simplement un plan. En dimension 2, un hyperplan est par conséquent une droite.

Plaçons-nous en dimension 2. Tracez donc 2 axes perpendiculaires. Maintenant, tracez une droite. Vous voyez que cette droite sépare le plan en 2 parties. A quoi cela sert-il ? En fait, un réseau de neurone simple (nous le verrons plus loin) va permettre de classer les points du plan dans une partie ou l'autre du plan grâce à cette droite, dans le cas de la dimension 2. Encore une fois, dans ce cas-là, ceux qui seront dans une partie du plan appartiendront à la première classe et ceux qui seront dans l'autre partie du plan (de l'autre côté de la droite) appartiendront à la deuxième classe.

3.3. Activation et condition d'activation

On dit que le neurone est actif lorsque in $\geq$ 0, autrement dit lorsque a = g(in) $\geq$ $seuil_g$ = g(0). Similairement, on dit que le neurone est inactif lorsque in $\leq$ 0, autrement dit lorsque a = g(in) $\leq$ $seuil_g$ = g(0).

3.4. Exemple en dimension 2 : la fonction booléenne OU

Dans ce cas, nous allons décrire une entrée par 2 informations. En effet, la fonction booléenne OU prend 2 informations en entrée (deux nombres qui peuvent valoir 0 ou 1 chacun) et retourne 1 si l'un des deux au moins vaut 1. Ici, nous allons grâce à un neurone simuler cette fonction, en rajoutant de la souplesse (en entrée nous accepterons des nombres **réels** compris entre 0 et 1).

Tout d'abord, il faut que nous-même sachions décrire le fonctionnement. Nous pouvons dire que le neurone pourra être actif lorsque $w_1 x_1 + w_2 x_2 \geq 0.5$, ce qui représente la partie des réels entre 0 et 1 qui est la plus proche de 1. Nous avons donc déjà le seuil du neurone, qui sera ici 0,5. De plus, on peut considérer que si $x_1 = x_2 = 0.25$, le neurone renverra 1. On va donc finalement choisir comme poids $w_1 = w_2 = 1$ et comme nous l'avons dit, le coefficient de biais, aussi appelé **seuil** , $w_0 = 0.5$.

Tracez les axes perpendiculaires puis la droite qui passe par les points (0.25,0) et (0,0.25). Tous les points qui se trouvent en-dessous de la droite seront ceux pour lesquels le neurone renverra 0, les autres seront ceux pour lesquels le neurone renverra 1.

En effet, prenons par exemple comme entrées le couple (0.1,0.8). Le calcul fait par le neurone est le suivant.

$$in = 1 \times 0.1 + 1 \times 0.8 = 0.9 \geq 0.5$$

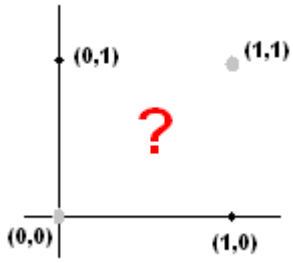
donc a = g(in) = 1.

Ainsi on remarque que le point (0.1,0.8) est un point pour lequel le neurone doit renvoyer la valeur 1 en sortie, car la somme pondérée des entrées est supérieure au seuil. Si l'on regarde sur le dessin contenant la droite séparatrice que vous avez tracé, on constate effectivement que ce point là se situe bien dans la zone supérieure du plan que délimite la droite.

Etant donné que la fonction OU renvoie 1 ou 0 généralement, et non pas un réel de [0,1], la fonction seuil suffira bien amplement pour notre neurone.

Nous avons ainsi entièrement déterminé notre neurone. Ainsi, ce neurone prend les mêmes valeurs que la fonction booléenne OU, ce qui constitue une approximation parfaite de cette dernière.

Maintenant, essayez de reproduire cette démarche pour la fonction XOR. La fonction booléenne XOR renvoie 1 quand l'une de ses deux entrées vaut 1, 0 sinon. Essayez donc de placer les points pour lesquels *x XOR y* vaut 1 en noir et ceux pour lesquels *x XOR y* vaut 0 en gris clair. Essayez maintenant de tracer **une droite** séparant les points noirs des points blancs. Vous verrez alors que c'est impossible. On voit donc les limitations du neurone seul. Qui plus est, le modèle de réseau de neurone qui va être étudié dans la partie qui suit ne permet pas non plus de représenter la fonction XOR, car il ne servira qu'à obtenir plusieurs sorties au lieu d'une seule, chaque neurone faisant sa partie du travail.



La fonction XOR n'est pas linéairement séparable

A titre informatif, le modèle qui vient d'être présenté est celui établi par McCulloch et Pitts en 1943. Nous allons justement dans la partie qui suit étudier le modèle du perceptron.

En résumé :

- Un neurone est l'unité élémentaire de traitement d'un réseau de neurones ;
- Un neurone est relié à chacune de ses informations et à chacune de ces liaisons est attaché un nombre réel, nommé **poids** ;
- Un neurone possède un seuil, qui est modélisé par une information de valeur -1 et un poids w_0 , où justement w_0 est le seuil ;
- Un neurone ne traite pas chaque information indépendamment, mais effectue la somme des produits des informations par leur poids associé et traite cette donnée ;
- Il existe plusieurs fonctions de transferts remplissant les conditions nécessaires ;
- Quelle que soit la fonction de transfert, elle prendra l'image de la somme pondérée des informations, y compris le $-w_0$;
- La fonction booléenne OU peut être approchée par un seul neurone. Il en est de même pour la fonction ET ;
- Une fonction de classification qui consiste en l'insertion d'un hyperplan séparant les points appartenant à une première classe de ceux appartenant à une seconde est approchable par un neurone seul ;
- Pour les fonctions pour lesquelles c'est impossible, il faut établir un réseau de neurones multicouche.

4. Réseau de neurones monocouche : le perceptron

Un réseau de neurones monocouche, aussi appelé perceptron, est caractérisé de la manière suivante.

- Il possède *n* informations en entrée ;
- Il est composé de *p* neurones, que l'on représente généralement alignés verticalement. Chacun peut en théorie avoir une fonction d'activation différente. En pratique, ce n'est généralement pas le cas ;
- Chacun des *p* neurones est connecté aux *n* informations d'entrée.

Le réseau de neurones possède ainsi *n* informations en entrée et *p* sorties, chaque neurone renvoyant sa sortie.

Pour la suite, on notera :

- $\mathbf{X} = (x_i)_{1 \leq i \leq n}$ les n informations d'entrée ;
- $w_{i,j}$ pour $1 \leq i \leq n$ et $1 \leq j \leq p$, le poids reliant l'information x_i et le neurone *j* puis a_j l' **activation** du *j* -ème neurone ;
- $w_{0,j}$ le **coefficient de biais** , également appelé **seuil** , du *j* -ème neurone ;
- in_j la donnée d'entrée (somme pondérée) du *j* -ème neurone.

On a donc l'équation suivante :

$$\forall 1 \leq j \leq p, a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} \times x_i) = g((\sum_{i=1}^{n} w_{i,j} \times x_i) - w_{0,j})$$

Chaque neurone de la couche donnera donc une sortie. Une utilisation courante est que chaque neurone de la couche représente une classe. Pour un exemple **X** donné, on obtient la classe de cet exemple en prenant la plus grande des *p* sorties. Essayons maintenant d'approfondir ce modèle.

4.1. Modélisation Matricielle

Avec la notation que nous avons définie pour les poids, on obtient ainsi une matrice $\mathbf{W} = (w_{i,j})_{(i,j) / 1 \leq i \leq n \text{ et } 1 \leq j \leq p}$ où le coefficient ligne *i* colonne *j* représente le coefficient liant la *i* -ème information au *p* -ème neurone. Si on représente par $\mathbf{A} = (a_j)_{1 \leq j \leq p}$ la liste des *p* sorties du réseau de neurone, on obtient l'équation matricielle suivante, en adaptant la fonction de transfert à notre nouvelle modélisation : on définit une nouvelle fonction g qui prend l'image d'un vecteur en calculant l'image de chacune de ses composantes, et retourne un vecteur dont les composantes sont les images des composantes de départ.

$$A = g({}^tW \times X)$$

Cette modélisation n'est pas fréquemment utilisée mais représenter un réseau de neurones monocouche par une matrice permet d'avoir une autre vision du problème.

4.2. Evaluation : RdN(X)

L'évaluation d'un exemple $\mathbf{X} = (x_i)_{1 \leq i \leq n}$ par un réseau de neurones donne un vecteur à *p* composantes. On définit la notation suivante.

$$RdN(X) = g({}^tW \times X)$$

où *RdN(**X**)* est un vecteur à *p* composantes, **W** une matrice à *n* lignes et *p* colonnes et **X** le vecteur dont chacune des *n* composantes représente une information d'entrée.

4.3. Les différents types de perceptrons

Il existe 2 types de perceptrons : les perceptrons *feed-forward* et les perceptrons *récurrents* . Les perceptrons *récurrents* sont ceux qui alimentent leurs entrées avec leurs sorties, alors que les perceptrons *feed-forward* non.

4.4. Séparation linéaire et conditions d'approximabilité

Pour un perceptron dont les fonctions de transferts sont toutes la fonction seuil (perceptron à seuil), en dimension 2, comme nous l'avons vu plus haut, on peut parfois tracer une droite telle que d'un côté de la droite, les points appartiennent à une première classe et de l'autre côté ils appartiennent à une seconde classe. Cette notion de **séparation** est généralisée. Pour un perceptron à *p* neurones, on parle de **séparation** de l'espace par un **hyperplan**

de cet espace. Toutefois, on ne peut pas toujours avoir un hyerplan qui sépare l'espace en 2 parties, dans les cas où la fonction qu'on veut approcher est plus complexe.

Dans le cas où l'on peut séparer l'espace avec un hyperplan, on dit que la fonction à approcher est **linéairement séparable** .

En général, les perceptrons à seuil ne permettent de représenter que des fonctions linéairement séparables. De même, les perceptrons à sigmoïde sont assez limités, à la seule différence qu'ils représentent une séparation linéaire un peu plus douce.

4.5. Comment choisir les poids ?

Malgré cette limitation, les perceptrons à seuil ont une propriété intéressante qui est celle que nous allons aborder dans la section suivante : il existe un (en fait plusieurs) algorithme(s) qui permet(tent) à un perceptron d'adapter ses poids à un ensemble d'exemples de sorte à obtenir pour cet ensemble la classification attendue. Ainsi, si l'ensemble d'exemples est assez vaste (les exemples sont assez variés), on pourra obtenir un perceptron qui donnera des résultats convenables pour des exemples non rencontrés.

5. Apprentissage simple du perceptron : méthode du gradient et algorithme de Widrow-Hoff

Il y a deux algorithmes, principalement, pour "faire apprendre" à un réseau de neurones monocouche. Le premier est la méthode simple et se nomme la **descente de gradient** . L'autre, un peu plus efficace généralement, se nomme algorithme de **Widrow-Hoff** , du nom des deux scientifiques qui ont élaboré cette technique.

Les deux méthodes consistent à comparer le résultat qui était attendu pour les exemples puis à minimiser l'erreur commise sur les exemples. Toutefois, il existe bien sûr une nuance entre les deux méthodes, qui va être expliquée plus loin.

Nous allons, pour chacune des méthodes, étudier la correction des poids concernant seulement l'un des neurones. Il suffira d'appliquer successivement la méthode de votre choix à chacun des neurones du réseau monocouche.

5.1. Apprentissage par descente de gradient

Pour comprendre cette méthode d'apprentissage, il faut définir l'erreur quadratique *E* . Si l'on est en présence de *N* exemples, alors pour $1 \leq k \leq N$, notons ($\mathbf{X_k}$,y_k) le couple *exemple - sortie attendue* , où $X_k = (x_i)_{1 \leq i \leq n}$ est le vecteur dont les coordonnées sont les *n* informations d'entrée de l'exemple et où y_k est la sortie attendue (la sortie "vraie") pour cet exemple-là de la part de notre neurone. Enfin, on note s_k la sortie obtenue pour le *k* -ème exemple avec les poids actuels. Alors, l'erreur quadratique est définie comme suit.

$$E = \frac{1}{2}\sum_{k=1}^{N}(y_k - s_k)^2$$

On voit donc que l'erreur est nulle si le réseau de neurones ne se trompe sur aucun des exemples, c'est à dire s'il parvient à calculer la bonne sortie (par exemple classifier) pour chacun des exemples correctement. C'est rarement le cas car souvent on démarre avec des poids tirés aléatoirement. Il s'agit donc de minimiser, pour un ensemble de *N* exemples donné, cette erreur quadratique. Je n'ai pas jugé nécessaire de vous présenter en milieu de ce cours le calcul qui permet d'obtenir une minimisation de l'erreur quadratique. Toutefois, le résultat étant très important, la démonstration est disponible en annexe de cet article. On obtient la variation à appliquer à chaque poids afin de classifier au mieux (parfaitement, dans le meilleur des cas) chacun des *N* exemples.

On va noter α un nombre réel auquel on donne le nom de taux d'apprentissage. C'est nous qui devons lui donner une valeur lors de la mise en pratique de l'apprentissage. Comme nous ne considérons qu'un neurone à la fois, on va noter w_i le poids reliant la *i* -ème information à notre neurone.

La méthode de descente du gradient consiste en fait à effectuer les actions suivantes :

Etapes de la méthode de descente du gradient

- Créer *n* variables dw_i, pour $1 \leq i \leq n$, égales à 0
- Prendre un exemple e_k, pour $1 \leq k \leq N$
- Calculer la sortie obtenue avec les poids actuels, notée s_k (1)
- Rajouter à dw_i, pour tout $1 \leq i \leq n$, le nombre $\alpha(y_k - s_k)x_i$ (2)
- Réitérer (1) et (2) sur chacun des exemples
- Pour $1 \leq i \leq n$, remplacer w_i par $w_i + dw_i$

Voici donc l'algorithme d'apprentissage par descente du gradient, appliqué à un seul neurone, qu'il faudra donc répéter sur chacun des neurones.

```
Entrée : n poids reliant les n informations à notre
neurone ayant des valeurs quelconques
N exemples (X_k, yk) où X_k est un vecteur à n
composantes x_i,
chacune représentant une information de cet exemple

Sortie : les n poids modifiés

POUR 1 <= i <= n
        dw_i = 0
FIN POUR

POUR TOUT exemple e = (Xk, yk)
        Calculer la sortie sk du neurone
        POUR 1 <= i <= n
                di = dw_i + alpha*(yk - sk)*x_i
        FIN POUR
FIN POUR

POUR 1 <= i <= n
        w_i = w_i + dw_i
FIN POUR
```

Afin d'obtenir de bons résultats, il faudra passer plusieurs fois les exemples à chaque neurone, de sorte que les poids convergent vers des poids "idéaux". Le problème avec cette méthode est que l'on corrige sur la globalité des exemples, ce qui fait que le réseau ne s'adaptera aux exemples qu'après un certain moment. Il y a une autre méthode qui permet de corriger sur chacun des exemples, et qui se nomme méthode d'apprentissage de Widrow-Hoff.

5.2. Apprentissage par l'algorithme de Widrow-Hoff

L'algorithme de Widrow-Hoff, ou encore "la règle delta", n'est en fait qu'une variante de l'algorithme précédent. Je vais détailler ceci.

Comme nous l'avons vu dans l'algorithme précédent, on engrange les erreurs commises sur chaque exemple puis pour terminer on corrige le poids, et ce pour chaque poids. Vous ressentez probablement cette méthode comme assez "grossière", dans le sens où elle n'est pas très précise et met beaucoup de temps avant

de tendre vers le bon coefficient. On ressent le fait qu'elle ne corrige qu'un petit peu alors qu'elle pourrait corriger beaucoup mieux pour chaque exemple. Et c'est là que l'algorithme de Widrow-Hoff intervient. En effet, la méthode élaborée par Widrow et Hoff consiste à modifier les poids après **chaque exemple** , et non pas après que tous les exemples aient défilé. Ceci va donc minimiser l'erreur de manière précise, et ce sur chaque exemple. Instinctivement, on constate bien que le réseau de neurones va s'améliorer nettement mieux et va tendre bien plus rapidement à classifier parfaitement (ou presque) chacun des exemples, bien que des méthodes plus efficaces encore existent. Voici donc l'algorithme de Widrow-Hoff.

```
Entrée : n poids reliant les n informations à notre
neurone ayant des valeurs quelconques
                        N exemples (X_k, yk) où X_k est
un vecteur à n composantes x_i,
                                chacune représentant
une information de cet exemple
                        Le taux d'apprentissage alpha

Sortie : les n poids modifiés

POUR TOUT exemple = (Xk,yk)
        Calculer la sortie sk du neurone
        POUR 1 <= i <= n
                w_i = w_i + alpha*(yk - sk)*x_i
        FIN POUR
FIN POUR
```

Cette méthode est bien sûr plus efficace, comme dit précédemment, mais l'algorithme est plus simple également ! Je vous invite donc à tester tout d'abord cet algorithme dans des cas assez simples, comme les fonctions booléennes ainsi que tout autre exemple de ce genre auquel vous pourriez penser.

Appliquer plusieurs fois cet algorithme permettra d'affiner la correction d'erreur et d'obtenir un réseau de neurones de plus en plus performant. Attention toutefois, car l'appliquer un trop grand nombre de fois mènerait à ce que l'on appelle "l'overfitting" (sur-apprentissage), c'est à dire que votre réseau devient très performant sur les exemples utilisés pour l'apprentissage, mais ne parvient peu ou pas à généraliser pour des informations quelconques.

6. Les types de réseaux de neurones

Il s'agit de généraliser ce qui a été vu en 4.3. Un réseau de neurones est simplement un ensemble de neurones liés entre eux, le poids étant partie intégrante de cette liaison. La façon de lier un neurone à un autre est de prendre la sortie de ce premier, d'affecter un poids à sa valeur et de rejoindre une entrée d'un autre neurone, comme nous l'avons déjà vu.

Les réseaux de neurones *feed-forward* sont des réseaux où les neurones ne sont connectés que dans un sens, le sens orienté de l'entrée vers la sortie. Les réseaux de neurones *récurrents* sont des réseaux où les neurones de sortie par exemple peuvent voir leur sortie utilisée comme entrée d'un neurone d'une couche précédente (nous verrons ce que sont les couches) ou de la même couche. Par conséquent, instinctivement il est évident que ce modèle est bien plus compliqué. Comme l'indique le titre de cet article, ce n'est pas ce modèle qui est étudié dans le cas d'un réseaux de neurones à plusieurs couches.

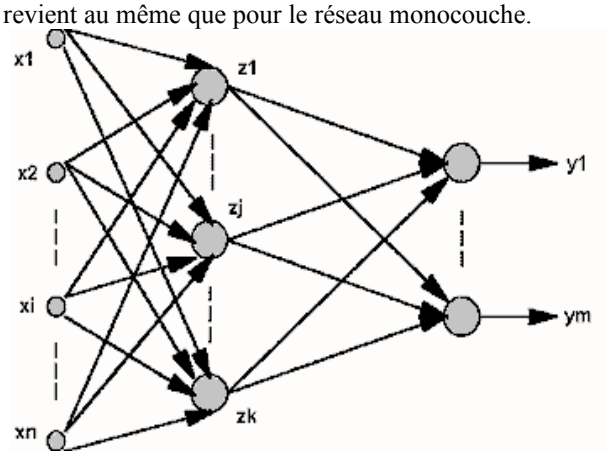
7. Perceptron multicouche

Nous avons précédemment étudié les perceptrons (réseaux monocouche) et nous avons vu que les neurones de sortie étaient chacun connectés aux mêmes informations. Nous les avons perçus comme une couche (alignés verticalement). Ainsi, une couche est constituée de neurones étant connectés aux mêmes informations mais n'étant pas connectés entre eux. Il s'agit maintenant de généraliser le perceptron. On peut ainsi disposer les neurones en plusieurs couches. Ainsi les informations en entrée sont connectées à tous les neurones de la première couche, tous les neurones de la première couche sont connectés à tous les neurones de la seconde couche, et ainsi de suite jusqu'à la dernière couche, appelée couche de sortie. Toutes les couches exceptée la couche de sortie sont considérées comme "couches cachées". Toutefois, n'ayez crainte : il a été prouvé que dans la plupart des cas, un réseau à 2 couches (informations -> une couche -> couche de sortie) où chaque neurone de la couche cachée a comme fonction d'activation la fonction sigmoïde et chaque neurone de la couche de sortie a comme fonction d'activation une fonction linéaire permet d'approximer une fonction continue. Il s'agit du Théorème de Cybenko. Pour des fonctions discontinues, nous n'avons aucune garantie.

Du fait du résultat précédent, nous allons restreindre notre étude aux réseaux neuronaux à une seule couche cachée. La couche cachée permet plus d'interaction et instinctivement, on est conscient que notre réseau de neurone pourra apprendre des "fonctions" plus complexes en rajoutant une couche. Le fonctionnement n'est pas pour autant complexe. On a toujours le choix de la fonction d'activation, et l'évaluation de la sortie d'un neurone se déroule de la même manière. La seule différence est que la sortie d'un neurone de la couche cachée sera l'une des informations d'entrées des neurones de la couche de sortie.

7.1. Evaluation : RdN(X)

Il y a par rapport au perceptron monocouche une étape supplémentaire lors de l'évaluation de la sortie du réseau de neurone. On obtient ainsi une matrice des poids pour passer des informations à la couche cachée, et une autre matrice de poids pour passer de la couche cachée à la couche de sortie. Ensuite, cela revient au même que pour le réseau monocouche.



Modélisation du perceptron multicouche

Retrouvez la suite de l'article d'Alp Mestan en ligne : Lien90

Les derniers tutoriels et articles

Développer une colonne lookup personnelle avec de l'AJAX

Dans ce tutoriel, nous allons voir comment on peut développer une colonne personnelle de type lookup en y intégrant de l'AJAX.

1. Introduction

Pour une introduction sur les colonnes personnelles, veuillez consulter ce tutoriel (Lien91). Il explique en effet toutes les démarches à suivre pour créer une colonne personnelle en terme de structure, de déploiement etc...Dans ce tutoriel, nous allons uniquement nous concentrer sur le type de colonne (lookup) et sur une intégration "manuelle" de code AJAX.

Notre composant fait en effet appel à de l'AJAX mais sans nécessiter l'installation du framework ASP.NET AJAX qui n'est pas compatible en tant que tel avec SharePoint lorsque le SP1 n'est pas installé (UpdatePannel non supporté même s'il est possible de le faire fonctionner moyennant quelques efforts). C'est d'ailleurs la raison pour laquelle, ce composant génère l'AJAX "à la main" car dans certains environnements, installer à la fois le SP1 de SharePoint et le framework AJAX est encore proscrit par crainte d'instabilité.

D'hatibude, je préfère ne pas réinventer la roue mais certains cas de figure l'imposent :). En d'autres circonstances, je me serais bien sûr reposé sur le framework ASP.NET AJAX riche, stable et très facile à utiliser.
Ceci dit, cela nous permettra de démystifier l'AJAX et de se rendre compte qu'on peut très facilement intégrer des composants avec de l'AJAX sans devoir changer quoi que ce soit à la configuration de SharePoint. Après tout, l'AJAX ce n'est qu'un peu de code client et un composant serveur qui lui répond, en l'occurrence dans notre cas, il s'agira du service web lists.asmx de SharePoint

2. Pourquoi une lookup avec de l'AJAX?

Pourquoi implémenter une colonne de type lookup et faire appel à de l'AJAX? La réponse se trouve dans cette question : avez-vous déjà essayé de créer une colonne lookup standard sur une liste contenant plusieurs milliers d'éléments?

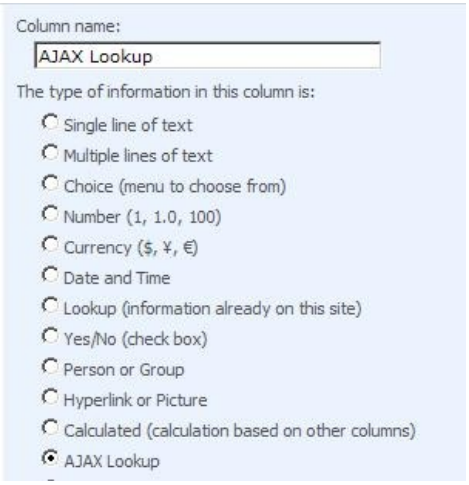
Si oui, vous aurez probablement constaté une forte dégradation des performances lors de l'encodage/édition d'un élement de liste. Sinon, faites le test. Notez toutefois que cette dégradation est uniquement dûe au navigateur, SharePoint quant à lui restitue les milliers d'enregistrements en un clin d'oeil.

L'AJAX va nous permettre de rappatrier des éléments de la liste source petit à petit sans faire de full page postback et donc de garder un confort optimal pour l'utilisateur.

3. Démonstration du projet

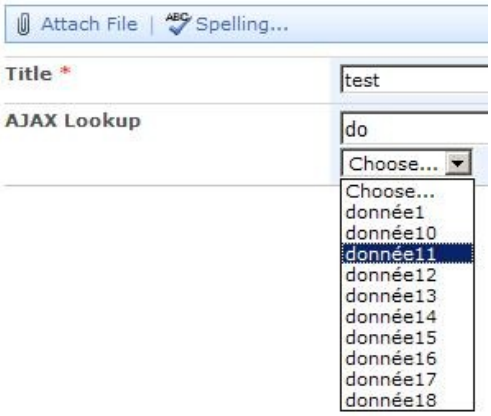
Le projet est actuellement téléchargeable sur codepelex à cette adresse (Lien92). Voici en quelques étapes comment il fonctionne

Etape 1 : Ajout de la colonne à une liste et sélection de la liste source



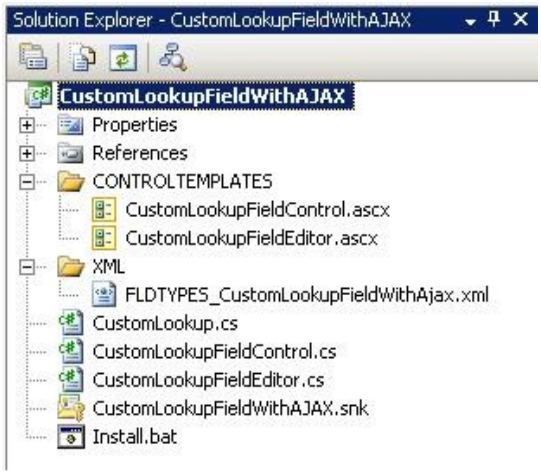
Notez que vous ne pouvez pas sélectionner de champ cible. En effet, la colonne se basera toujours sur le champ **Title** pour plus de facilité. Je donnerai les détails à ce propos plus tard.

Etape 2 : création d'un élément de liste



au fur et à mesure que vous tapez des lettres (ici ***do***), la liste du dessous retourne les 10 éléments commençant par la/les lettre(s) entrée(s).

4. Structure de notre solution



- CONTROLTEMPLATES => CustomLookupFieldControl.ascx : contrôle affiché lors de la saisie de données
- CONTROLTEMPLATES => CustomLookupFieldEditor.ascx : contrôle affiché lors de la création de la colonne
- XML => FLDTYPES_...: fichier XML décrivant la colonne personnelle et son comportement en mode "raw view"
- CustomLookup.cs => classe invoquée lors de l'addition/édition/affichage de notre colonne personnelle
- CustomLookupFieldControl.cs => classe invoquée lors de l'addition/édition/affichage de notre colonne personnelle
- CustomLookupFieldEditor.cs => classe invoquée lors de l'addition/édition de notre colonne à une liste

Le fichier Install.bat n'est là que pour déployer les composants dans un environnement de développement. Il vous faudrait idéalement générer un fichier de solution. Vous pouvez le faire très facilement avec WSPBUILDER (Lien93)

5. Communication en AJAX avec le service web lists.asmx

Tout d'abord, en préambule, il faut savoir que le service web lists.asmx out of the box permet entre-autres de récupérer des éléments de liste via des requêtes CAML.

Il est accessible depuis n'importe quel site de votre ferme, il suffit d'ajouter à la fin de l'URL de votre site courant /_vti_bin/lists.asmx. Lorsque vous le faites, vous obtenez la liste des méthodes utilisables par n'importe quel applicatif.

Vous retrouvez notamment la méthode **GetListItems** que nous utiliserons dans notre composant. Lorsque vous cliquez sur le nom de cette méthode, vous obtenez des informations cruciales telles que la **SOAPACTION**, les différentes en-têtes HTTP et les paramètres que vous allez devoir lui transmettre. En l'occurrence, on voit qu'on doit transmettre les paramètres **listName, viewName, query, viewFields, rowLimit et queryOptions**

```
POST /_vti_bin/lists.asmx HTTP/1.1
Host: sey-pc
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://schemas.microsoft.com/sharepoint/soap/GetListItems"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmln
  <soap:Body>
    <GetListItems xmlns="http://schemas.microsoft.com/sharepoint/soap/">
      <listName>string</listName>
      <viewName>string</viewName>
      <query>
        <xsd:schema>schema</xsd:schema>xml</query>
      <viewFields>
        <xsd:schema>schema</xsd:schema>xml</viewFields>
      <rowLimit>string</rowLimit>
      <queryOptions>
        <xsd:schema>schema</xsd:schema>xml</queryOptions>
      <webID>string</webID>
    </GetListItems>
  </soap:Body>
</soap:Envelope>
```

Avant de passer à l'AJAX, voyons comment appeler cette méthode depuis un programme console ou une application windows

```
XmlDocument XmlDoc = new System.Xml.XmlDocument();
XmlNode QueryNode =
XmlDoc.CreateNode(XmlNodeType.Element, "Query", "");
XmlNode ViewFieldsNode =
        XmlDoc.CreateNode(XmlNodeType.Element,
"ViewFields", "");
XmlNode QueryOptionsNode =
        XmlDoc.CreateNode(XmlNodeType.Element,
"QueryOptions", "");
QueryOptionsNode.InnerXml =
        "<IncludeMandatoryColumns>FALSE</IncludeMandato
ryColumns>";
QueryNode.InnerXml = "<OrderBy><FieldRef
Name='Title'/></OrderBy><Where>" +
        "<Eq><FieldRef Name='Title'/><Value
Type='Text'>test</Value></Eq></Where>";
list.Lists SrvInstance = new
ConsoleApplication1.list.Lists();
SrvInstance.Credentials =
System.Net.CredentialCache.DefaultNetworkCredentials;
try
{
   XmlNode Result = SrvInstance.GetListItems(
        "guid de la liste",
        null,
        QueryNode,
        null,
        "10",
        QueryOptionsNode,
        null);

   Console.WriteLine(Result.OuterXml);
}
catch (SoapException Ex)
{
   Console.WriteLine(Ex.Detail.InnerText);
}
```

Ce petit exemple nous ramène le flux XML suivant en guise de réponse

```
<listitems xmlns:s="uuid:BDC6E3F0-6DA3-11d1-
A2A3-00AA00C14882"
                   xmlns:dt="uuid:C2F41010-65B3-11d1-
A29F-00AA00C14882"
                   xmlns:rs="urn:schemas-microsoft-
com:rowset"
                   xmlns:z="#RowsetSchema"
                   xmlns="http://schemas.microsoft.com/
sharepoint/soap/">
<rs:data ItemCount="1">
   <z:row ows_Attachments="0" ows_LinkTitle="test"
                  ows_ID="1" ows_MetaInfo="1;#"
                  ows__ModerationStatus="0"
             ows__Level="1"
             ows_owshiddenversion="3"
             ows_UniqueId="1;#{6AC7A2B9-BDD0-48DC-
B73C-2389344D2619}"
             ows_FSObjType="1;#0"
```

```
            ows_Created_x0020_Date="1;#2008-05-19T06:
25:52Z"
            ows_Created="2008-05-19T06:25:52Z"
            ows_FileLeafRef="1;#1_.000"
            ows_FileRef="1;#Lists/test/1_.000" />
</rs:data>
</listitems>
```

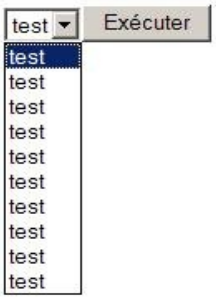
Flux qu'il faudra ensuite analyser pour récupérer les différentes valeurs de notre élément de liste. Avec l'AJAX, c'est un peu pareil sauf qu'on doit en plus passer les en-têtes HTTP.

Voici un exemple complet d'appel à la méthode GetListItems du service web lists.asmx en AJAX et le traitement de la réponse.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html>
<head>
<script language="javascript">
    function DoQuery()
    {
        var SpCAMLQuery;
        HttpObject = null;
        TargetObject =
document.getElementById("UneListe");
        SpCAMLQuery= '<?xml version="1.0"
encoding="utf-8"?>';
        SpCAMLQuery+='<soap:Envelope xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"';
        SpCAMLQuery+='xmlns:xsd="http://www.w3.org/2001
/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
';
        SpCAMLQuery+='<soap:Body>';
        SpCAMLQuery+='<GetListItems
xmlns="http://schemas.microsoft.com/sharepoint/soap/">'
;
        SpCAMLQuery+='<listName>a1420e71-84d3-4602-98e0
-643a10c4fafd</listName>' ;
        SpCAMLQuery+='<rowLimit>10</rowLimit>' ;
        SpCAMLQuery+="<query><Query><Where><Eq><FieldRe
f Name='Title'/><Value Type='Text'>test</Value>';
        SpCAMLQuery+='</Eq></Where></Query></query>";
        SpCAMLQuery+='<queryOptions><QueryOptions><Incl
udeMandatoryColumns>TRUE</IncludeMandatoryColumns>';
        SpCAMLQuery+='</QueryOptions></queryOptions>' ;
        SpCAMLQuery+='</GetListItems></soap:Body></soap
:Envelope>' ;
        var serverUrl = 'http://sey-
pc/_vti_bin/Lists.asmx?wsdl' ;
        if (HttpObject==null)
        {
                if (window.XMLHttpRequest)
                {
                        HttpObject = new
XMLHttpRequest();
                }
                else if (window.ActiveXObject)
                {
                        HttpObject = new
ActiveXObject("MSXML2.XMLHTTP.3.0");
                }
        }
        //Executing the call
        HttpObject.open("POST", "http://sey-
pc/_vti_bin/lists.asmx", true);
        HttpObject.setRequestHeader("Content-
Type","text/xml; charset=utf-8");
        HttpObject.setRequestHeader("Host", "sey-pc");
        HttpObject.setRequestHeader("SOAPAction","http:
//schemas.microsoft.com/sharepoint/soap/GetListItems");
        readyStateChangeHandler = AjaxAnswer;
        HttpObject.onreadystatechange =
readyStateChangeHandler;
        HttpObject.send(SpCAMLQuery);
    }
    function AjaxAnswer()
    {

        //When the answer is back and the http request
was successfull
        if (HttpObject.readyState==4 &&
HttpObject.status==200)
        {
            window.status = "SOAP request successfully
processed...";
            var xml = HttpObject.responseXML;
            if (xml.documentElement)
            {
               //retrieving the returned rows
                    var rows =
xml.documentElement.getElementsByTagName("z:row");
                    if (rows.length==0)
                    {
                        rows =
xml.documentElement.getElementsByTagName("row");
                    }

                    TargetObject.length = 0;
                    TargetObject.disabled=true;
                    if (rows.length > 0)
                    {

                        for (var i = 0; i <
rows.length; i++)
                        {
                           var TitleValue = "";
                           var IdValue = "";
                           if
(rows[i].getAttributeNode("ows_Title")!=null)
                           {
                               TitleValue =
rows[i].getAttributeNode("ows_Title").nodeValue;
                           }
                           if
(rows[i].getAttributeNode("ows_ID")!=null)
                           {
                               IdValue =
rows[i].getAttributeNode("ows_ID").nodeValue;
                           }
                           NewOption =
document.createElement("option");
                           NewOption.value =
IdValue+";#"+TitleValue;
                           NewOption.text = TitleValue;
                           try
                           {
                               TargetObject.add(NewOpti
on,null);
                           }
                           catch(ex)
                           {
                               TargetObject.add(NewOpti
on);
                           }
                           TargetObject.disabled=false;
                        }
                    }
            }
        }
        else
        {
            window.status = "SOAP request failed...";
        }
}
```

```
</script>
</head>
<body>
<select id="UneListe"></select><input type="button"
onclick="DoQuery()" value="Exécuter"/>
</body>
</html>
```

Si vous copiez/collez ce code dans un fichier html, que vous remplacez bien sûr **sey-pc** par votre serveur et votre url et que vous remplacez le guid de la liste par l'un de vos guids et pour autant que votre liste contienne bien les données requêtes par la requête CAML, vous devriez obtenir ceci :))



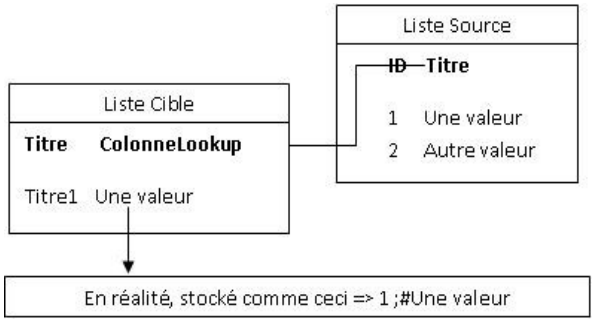
Je ne rentrerai pas dans le détail du code AJAX (similaire) de mon composant, vous aurez tout le loisir de le découvrir si cela vous intéresse en examinant le code source (contrôle CustomLookupFieldControl.ascx). Il y a juste un peu de code client supplémentaire pour traiter l'évènement **onkeyup** de la zone de texte où l'utilisateur saisit les données et un traitement spécifique de celle-ci. En effet, dans le contexte d'un contrôle SharePoint en général (webpart, custom field etc...), il est toujours possible que plusieurs instances d'un même contrôle soient déployées en même temps sur une même page. Il faut donc s'assurer de travailler avec des ID uniques lorsque l'on effectue des opérations du côté client.

Au sein de notre solution, c'est le contrôle utilisateur CustomLookupFieldControl.ascx qui contient le code AJAX très similaire à celui présenté ci-dessus

6. Dériver de SPFieldLookup

Comme pour un SPFieldText, lorsque l'on crée un custom field, il faut dériver d'un type existant. La grosse différence entre le SPFieldText et le SPFieldLookup est que SPFieldText fonctionne en mode "isolé/autonome" alors que SPFieldLookup maintient une référence vers un élément d' une liste de données source. La manière dont on stockera la valeur du champ sera donc différente. Il faut également spécifier au custom field vers quelle liste source il doit pointer.

Avant de se lancer dans le code, il est toujours bon de se rappeler comment une colonne de type lookup fonctionne en standard dans SharePoint. Pour pouvoir maintenir sa référence vers l'élément source, la valeur stockée au sein d'une lookup correspond à l'ID de l'élément source + le libellé de la colonne qu'on a décidé d'afficher comme le montre le schéma ci-dessous



Notre tâche principale va donc consister d'une part à indiquer à notre champ quelle est la liste source. Celle-ci peut-être choisie par l'utilisateur. C'est donc le contrôle utilisateur **CustomLookupFieldEditor.ascx** et son code-behind **CustomLookupFieldEditor.cs** qui s'en chargeront et d'autre part, nous devons stocker la valeur de l'ID+libellé de l'élément source.

En standard, une colonne lookup peut-être utilisée en mode multi-valué, je n'ai pas implémenté celui-ci sur notre composant, donc je ne l'aborderai pas.

6.1. L'héritage en question

```
public class CustomLookup : SPFieldLookup
    {

        #region default contructors
        public CustomLookup(SPFieldCollection fields,
string fieldName)
            : base(fields, fieldName)
        {



        }

        public
CustomLookup(Microsoft.SharePoint.SPFieldCollection
fields, string typeName, string displayName)
            : base(fields, typeName, displayName)
        {




        }
        #endregion

        /// <summary>
        /// This method ensures that a value is
provided if the field is mandatory
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        public override string
GetValidatedString(object value)
        {

            if (this.Required)
            {
                if (value == null || value.ToString()
== "")
                    throw new
Microsoft.SharePoint.SPFieldValidationException("Please
fill in this mandatory field");
                else
                    return value.ToString();
            }
            else
            {
                if (value != null)
                    return value.ToString();
                else
                    return null;
            }

        }

        /// <summary>
        /// Rendering control is called when the field
renders.
        /// </summary>
        public override
Microsoft.SharePoint.WebControls.BaseFieldControl
```

```
FieldRenderingControl
        {
            get
            {
                Microsoft.SharePoint.WebControls.BaseFi
eldControl CtField = new
CustomLookupFieldWithAJAX.CustomLookupFieldControl();
                CtField.FieldName = InternalName;
                return CtField;
            }

        }
    }
```

Héritage tout à fait classique, on indique simplement à SharePoint qu'on dérive de SPFieldLookup au lieu de SPFieldText. Dans la méthode **FieldRenderingControl** invoquée par les interfaces d'addition/édition/affichage web de SharePoint, nous indiquons qu'on retourne une instance de **CustomLookupFieldControl**, qui n'est ni plus ni moins que la classe qui décrit le comportement de notre colonne.

6.2. Le contrôle en question

Là encore, pas énormément de différence par rapport à SPFieldText. Je ne vais pas copier/coller tout le code de cette classe, simplement voici le code de la seule propriété changeante.

```
public override object Value
{
        get
        {
                EnsureChildControls();
                if (LookupHiddenValue.Text != "")
                        return new
SPFieldLookupValue(LookupHiddenValue.Text);
                else
                        return "";
        }
        set
        {
                if (ItemFieldValue != null)
                {
                        LookupHiddenValue.Text =
ItemFieldValue.ToString();
                        LookupValue.Text = new
SPFieldLookupValue(ItemFieldValue.ToString()).LookupVal
ue;
                }
        }
}
```

Ce qui change est donc la manière dont on construit la valeur qui je vous le rappelle a le format ID;#Libellé. Dans ce code, Le contrôle LookupHiddenValue est un simple TextBox caché qui contient la valeur ID;#Libellé. La classe **SPFieldLookupValue** permet de générer une valeur lookup compréhensible par SharePoint.

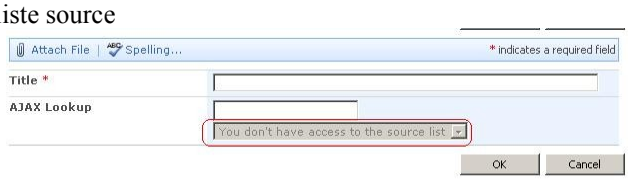
Au cas où vous implémentez un custom lookup field multi-valué, vous travaillerez avec la classe **SPFieldLookupValueCollection**

Une autre différence encore entre un SPFieldText et un SPFieldLookup est la sécurité. Lorsque vous créez un custom field dérivé de SPFieldText, vous ne vous souciez pas le moins du monde de savoir si l'utilisateur a le droit ou non de saisir une valeur pour votre colonne puisque SharePoint se chargera de lui interdire l'accès. Par contre, dans le cadre d'une colonne de type **Lookup**, le comportement de SharePoint est différent.
Si l'utilisateur n'a pas d'accès en lecture à la liste source pointée par la colonne, celle-ci reste vide et aucune donnée n'est sélectionnable. Il faut donc que nous adoptions le même comportement. J'ai ajouté un contrôle de sécurité au niveau du composant pour afficher un message à l'utilisateur si il n'a pas le droit d'accéder à la liste source. En plus de cette notification, j'éviterai de générer le code client faisant appel aux fonctions AJAX.

Ce contrôle se passe dans la méthode **CreateChildControls** que tout développeur connaît bien :)

```
using (SPWeb Web = SPContext.Current.Site.OpenWeb())
{
        try
        {
                Web.Site.CatchAccessDeniedException =
false;
                if (Web.Lists[new
Guid(CurrentField.LookupList)].DoesUserHavePermissions(
                        SPBasePermissions.ViewListItems
))
                        AccessListOk = true;
        }
        catch(UnauthorizedAccessException)
    {
                AccessListOk = false;
        }
}
```

Ce code vérifie si oui ou non l'utilisateur courant a le droit d'accéder en lecture à la liste source pointée par notre colonne de type lookup.
La variable **AccessListOk** est exploitée ultérieurement pour afficher le message d'erreur et ne pas générer le code client le cas échéant. L'utilisateur verra donc ceci en cas de non accessibilité à la liste source



6.3. Le contrôle d'addition/édition de la colonne à une liste

Ce contrôle permet de gérer le comportement de notre colonne lors de l'addition/édition de celle-ci à une liste. Dans notre solution, il s'agit du contrôle **CustomLookupFieldEditor.ascx** dont le code est le suivant

```
<wssuc:InputFormControl runat="server"
        LabelText="<%$Resources:wss,fldedit_getinfofrom
%>"
        >
        <Template_Control>
                <asp:DropDownList id="TargetLookupList"
runat="server"
                        Title = "<%
$Resources:wss,fldedit_getinfofrom%>"
                    Enabled="false"
                        >
                </asp:DropDownList>
        </Template_Control>
</wssuc:InputFormControl>
```

Ce code est relativement restreint et sert à déclarer notre contrôle DropDown qui contiendra l'ensemble des listes/bibliothèques du site courant.

Une fois de plus, c'est plutôt dans le code-behind qu'il faut travailler davantage.

```
public class CustomLookupFieldEditor : UserControl,
IFieldEditor
```

On dérive de UserControl et on implémente IFieldEditor.

```
public void InitializeWithField(SPField field)
{
CurrentField = field as SPFieldLookup;
using (SPWeb Web = SPContext.Current.Site.OpenWeb())
{
    //If the field is being created
    if (CurrentField == null)
    {

        TargetLookupList.Enabled = true;

        foreach (SPList List in Web.Lists)
        {
            if (!List.Hidden)
                TargetLookupList.Items.Add(new
ListItem(List.Title));
        }
    }
        //The field exists, so just show the list name
and does not allow
        //the selection of another list
    else
    {
        try
        {
            TargetLookupList.Items.Add(new
ListItem(Web.Lists[new
Guid(CurrentField.LookupList)].Title));

        }
        catch
        {
            TargetLookupList.Items.Add(new
ListItem("The source list does not exist"));
        }
        TargetLookupList.Enabled = false;
        return;
    }
}

}
```

Si **CurrentField** est null, cela signifie qu'on ajoute le champ à une liste et dans ce cas, on ajoute à la dropdown, la liste de toutes les listes/bibliothèques non cachées du site. Sinon, le champ a déjà été ajouté à la liste et a donc forcément été lié à une liste source, donc on affiche le nom de celle-ci et on verrouille la dropdown.

```
public void OnSaveChange(SPField field, bool bNewField)
{
        SPFieldLookup CurrentLookupField =
(SPFieldLookup)field;

        if (bNewField)
        {
                using (SPWeb Web =
SPContext.Current.Site.OpenWeb())
                {
                        CurrentLookupField.LookupList =
Web.Lists[TargetLookupList.SelectedItem.Text].ID.ToStri
ng();
                        CurrentLookupField.LookupWebId
= Web.ID;
                        CurrentLookupField.LookupField
= "Title";
                }
        }
}
```

La méthode **OnSaveChange** est appelée lorsque l'utilisateur clique sur le bouton Ok. On ne traite dans ce cas présent que l'addition de la colonne à une liste puisqu'on ne permet pas de la modifier. En cas d'ajout, on spécifie à la propriété **LookupList** le GUID de la liste choisie, le GUID du site courant et le champ cible. Dans ce cas, j'ai hard-codé le champ **Title** qui existe toujours quelle que soit la liste et la langue du site.

La raison pour laquelle j'ai hard-codé ce champ est tout simplement pour faciliter la génération de la requête CAML qu'on envoie vers le service web de SharePoint. En effet, en sachant qu'on travaille avec **Title**, il est aisé de créer une requête CAML travaillant avec l'opérateur **BeginsWith** puisqu'on sait qu'on traite toujours une colonne de type **String**. Si on permet à l'utilisateur de choisir la colonne cible, on va devoir traiter tous les types de champs, ce qui complique grandement la tâche.

7. Petits désavantages liées aux custom fields

Bien que les custom fields sont des composants très intéressants, ils ont néanmoins leur lot de désavantages, parmi ceux-ci

- ils sont en lecture seule en mode feuille de données (datasheet)
- ils sont en lecture seule en mode intégration cliente avec les produits Office (dans word par ex, un custom field est grisé et sa valeur ne peut être modifiée)
- ils sont actifs ou non pour la ferme entière, en effet, on ne peut pas décider d'activer un custom field pour une collection particulière. On peut toutefois créer un custom field, travailler avec et le rendre invisible par la suite mais ce n'est pas idéal
- ils ne sont parfois pas pris en charge par certains produits tiers tels que des webparts etc...car ils ont un type interne **Invalid**

8. Téléchargement

Vous trouverez le fichier de solution ainsi que les sources du projet à cette adresse (Lien94)

Retrouvez l'article de Stéphane Eyskens en ligne : Lien95

Liens

Lien1 : http://code.google.com/p/cspoker/
Lien2 : http://www.netbeans.org/community/articles/boeing-netbeans-platform.html
Lien3 : http://filthyrichclients.org/
Lien4 : https://timingframework.dev.java.net/
Lien5 : https://scenegraph.dev.java.net/
Lien6 : https://visualvm.dev.java.net/
Lien7 : http://www.netbeans.org/
Lien8 : http://www.projectdarkstar.com/
Lien9 : http://callofthekings.com/
Lien10 : http://www.sentilla.com/
Lien11 : http://www.livescribe.com/
Lien12 : http://www.teamjefferson.com/
Lien13 : http://jmars.asu.edu/
Lien14 : http://public.web.cern.ch/Public/fr/LHC/LHC-fr.html
Lien15 : http://developers.sun.com/learning/javaoneonline/
Lien16 : http://java.sun.com/javaone/sf/sessions/general/
Lien17 : http://photos.sun.com/page/2850
Lien18 : http://www.flickr.com/search/?q=javaone2008+OR+(javaone+and+2008)&z=t
Lien19 : http://ydisanto.developpez.com/reportages/java/javaone2008/
Lien20 : http://bmarchesson.developpez.com/tutoriels/java/hibernate/chargement/
Lien21 : http://fr.wikipedia.org/wiki/Temps_universel_coordonné
Lien22 : http://fr.wikipedia.org/wiki/Greenwich_Mean_Time
Lien23 : http://www.php.net/manual/fr/timezones.php
Lien24 : http://www.gnu.org/software/tar/manual/html_node/Date-input-formats.html
Lien25 : http://www.php.net/manual/fr/function.date-default-timezone-set.php
Lien26 : http://www.php.net/manual/fr/datetime.configuration.php#ini.date.timezone
Lien27 : http://pecl.php.net/package/timezonedb/
Lien28 : http://en.wikipedia.org/wiki/Zoneinfo
Lien29 : http://www.php.net/timezones
Lien30 : http://www.php.net/strtotime
Lien31 : http://framework.zend.com/manual/en/zend.date.html
Lien32 : http://zend-framework.developpez.com/
Lien33 : http://framework.zend.com/manual/fr/zend.date.html#zend.date.why
Lien34 : http://framework.zend.com/apidoc/core/Zend_Date/Zend_Date_DateObject/Zend_Date.html
Lien35 : http://julien-pauli.developpez.com/tutoriels/php/dates/
Lien36 : http://f-marx.developpez.com/tutoriels/webservices/sms-email/
Lien37 : http://www.orangepartner.com/
Lien38 : http://www.orangepartner.com/site/enuk/develop/devplayzone/l_admin_web_interface.jsp
Lien39 : http://codex.wordpress.org/Plugin_API/Action_Reference
Lien40 : http://www.orangepartner.com/site/enuk/develop/devplayzone/l_admin_web_interface.jsp
Lien41 : http://f-marx.developpez.com/tutoriels/webservices/wordpress-sms/
Lien42 : http://veerle.duoh.com/
Lien43 : http://veerle.duoh.com/#approved
Lien44 : http://css.developpez.com/tutoriels/liste-survolable-cliquable/fichiers/demo.html
Lien45 : http://elliotjaystocks.com/
Lien46 : http://samrayner.com/
Lien47 : http://samayner.com/archives/quicklinks/
Lien48 : http://css.developpez.com/tutoriels/liste-survolable-cliquable/fichiers/demo.html
Lien49 : http://css.developpez.com/tutoriels/liste-survolable-cliquable/
Lien50 : http://dico.developpez.com/html/1693-Internet-CSS-Cascading-Style-Sheets.php
Lien51 : http://giminik.developpez.com/xhtml/
Lien52 : http://giminik.developpez.com/xhtml/a.html
Lien53 : http://r-hunel.developpez.com/tutoriels/css/info-bulle/fichiers/xhtml.html
Lien54 : http://r-hunel.developpez.com/tutoriels/css/info-bulle/fichiers/info-bulle1.html
Lien55 : http://r-hunel.developpez.com/tutoriels/css/info-bulle/fichiers/exemple.html
Lien56 : http://r-hunel.developpez.com/tutoriels/css/info-bulle/fichiers/info-bulle2.html
Lien57 : http://r-hunel.developpez.com/tutoriels/css/info-bulle/
Lien58 : http://ditch.developpez.com/articles/visualstudio/2008/presentation/
Lien59 : http://broux.developpez.com/articles/csharp/animations-silverlight/
Lien60 : http://msdn2.microsoft.com/en-us/library/system.windows.media.animation.storyboard.targetproperty(VS.95).aspx
Lien61 : http://come-david.developpez.com/tutoriels/dps/
Lien62 : http://amarok.kde.org/blog/archives/597-A-few-days-worth-of-work.html
Lien63 : http://aseigo.blogspot.com/2008/02/loading-random-widgets.html
Lien64 : http://trolltech.com/developer/downloads/qt/snapshots
Lien65 : http://qt.developpez.com/tutoriels/navigateur-avec-qtwebkit/
Lien66 : http://fauconnier.developpez.com/articles/vba/general/classes/
Lien67 : http://www.honeyd.org/
Lien68 : http://packages.debian.org/etch/honeyd
Lien69 : http://www.honeyd.org/contrib.php
Lien70 : http://goldkey.developpez.com/tutoriels/linux/installation-configuration-honeypot/
Lien71 : http://iptables-tutorial.frozentux.net/fr/x5248.html
Lien72 : http://iptables-tutorial.frozentux.net/fr/x5422.html
Lien73 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFC793
Lien74: http://iptables-tutorial.frozentux.net/fr/c96.html#OSIREFERENCE
Lien75 : http://iptables-tutorial.frozentux.net/fr/c96.html#TCPIPLAYERS.IMG
Lien76 : http://iptables-tutorial.frozentux.net/fr/c461.html
Lien77 : http://iptables-tutorial.frozentux.net/fr/x265.html

Lien78 : http://iptables-tutorial.frozentux.net/fr/x173.html
Lien79 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFC791
Lien80 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFC1349
Lien81 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFC2474
Lien82 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFC3168
Lien83 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFC3260
Lien84 : http://iptables-tutorial.frozentux.net/fr/a6206.html#RFCEDITOR.ORG
Lien85 : http://linux.developpez.com/iptables/
Lien86 : http://sylvain-gamel.developpez.com/tutoriel/mac/cocoa/java/
Lien87 : http://matthieu-brucher.developpez.com/tutoriels/python/swig-numpy/
Lien88 : http://valgrind.org/
Lien89 : http://matthieu-brucher.developpez.com/tutoriels/3D/raytracer/01-introduction/
Lien90 : http://alp.developpez.com/tutoriels/intelligence-artificielle/reseaux-de-neurones/
Lien91 : http://stephaneey.developpez.com/tutoriel/sharepoint/customfields/
Lien92 : http://www.codeplex.com/autocompletelookupax
Lien93 : http://www.codeplex.com/wspbuilder
Lien94 : http://www.codeplex.com/autocompletelookupax
Lien95 : http://stephaneey.developpez.com/tutoriel/sharepoint/customlookupfield/