Developpez Le Mag

Édition d'avril – mai 2014
Numéro 51
Magazine en ligne gratuit
Diffusion de copies conformes à l'original autorisée
Réalisation : Alexandre Pottiez & Sébastien Lataix
Rédaction : la rédaction de Developpez
Contact : magazine@redaction-developpez.com

Sommaire

Éditorial

Ce mois-ci votre magazine se refait un lifting et améliore sa présentation. Retrouvez notamment la lecture des vidéos des news directement dans votre magazine, une meilleure disposition des articles afin de faciliter sa lecture. Profitez-en bien !

La rédaction

Article 2D/3D/Jeux

Programmer un émulateur
Chapitre 1 : Un tour d'horizon

Nous allons commencer notre aventure avec un voyage dans les coulisses de l'émulation. Comment est-il possible de créer un émulateur ? Comment faut-il s'y prendre pour espérer en créer un ? Cette partie nous permettra de répondre à ces différentes interrogations.

par BestCoder

Page 2

Article Conception

Design Pattern : les mémentos

Cet article vous redonne les points clés pour utiliser les patterns les plus utiles. Il vous propose surtout de télécharger des mémentos à imprimer au bureau.

par Thierry Leriche-Dessirier

Page 38

2D/3D/Jeux

Les derniers tutoriels et articles

Programmer un émulateur :

Un tour d'horizon

Nous allons commencer notre aventure avec un voyage dans les coulisses de l'émulation. Comment est-il possible de créer un émulateur ? Comment faut-il s'y prendre pour espérer en créer un ? Cette partie nous permettra de répondre à ces différentes interrogations.

1 Définition de l'émulation

Je parie que vous avez, plusieurs fois, fait des recherches pour savoir comment programmer un émulateur et que vous vous êtes retrouvés sur des sites vous montrant comment en configurer un, déjà existant. Votre calvaire vient de prendre fin. Grâce à ce tutoriel, vous serez en mesure de programmer le vôtre de A à Z. Avant de commencer notre passionnante aventure, nous allons définir les différentes notions que nous utiliserons afin de mieux cerner nos objectifs.

1.1 Émulation

1.1.a définition

D'après Wikipédia (lien 1), en informatique, l'émulation consiste à substituer un élément de matériel informatique - tels un terminal informatique, un ordinateur ou une console de jeu - par un logiciel. La définition du terme émuler est « chercher à imiter ». Il faut voir dans l'émulation une imitation du comportement physique d'un matériel par un logiciel, et ne pas la confondre avec la simulation.

Théoriquement, il est donc possible de créer un émulateur pour toutes les machines électroniques. Pour ce faire, il suffit juste de :

- connaître ses caractéristiques ;
- trouver un support au moins aussi puissant que la machine à émuler ;
- traduire les caractéristiques.

1.1.b Difficultés

Vous verrez tout au long de ce tutoriel que programmer un émulateur n'a rien d'extraordinaire. Le plus difficile est de connaître le fonctionnement exact de la machine que l'on désire émuler. Eh oui, les constructeurs n'ont aucun intérêt à publier les caractéristiques de leurs consoles et ce n'est pas aujourd'hui qu'ils le feront Image non disponible.

Pour trouver les informations qu'il leur faut, les programmeurs doivent donc avoir recours à diverses méthodes que l'on abordera dans le prochain chapitre.

1.1.c Comment fonctionne un émulateur ?

Un émulateur fonctionne avec une facilité qui pourrait même faire pleurer. Voyez par vous-mêmes :

```
1  while("Machine Fonctionne")
2  {
3      regarder_ce_qu_il_faut_faire();
4      Le_faire();
5  }
```

Tous les émulateurs ont la même structure de base. Le principe consiste tout simplement à regarder l'opération qu'il faut effectuer, puis l'effectuer. Cette routine sera exécutée tant que la machine - la console de jeu - est en marche.

1.2 Les ROM

1.2.a Définition

Littéralement, ROM signifie « Read Only Memory » , ou en français « mémoire à lecture seule ». C'est un support qui ne permet que la lecture des données qu'il contient. C'est le cas pour presque tous les supports de jeux vidéo. Il existe divers moyens de copier le contenu de ces supports sur votre ordinateur. Pour ce faire, on utilise des ROM Dumper. Les fichiers binaires ainsi obtenus sont communément appelés : ROM.

Les ROM sont donc aux émulateurs ce que les cartouches de jeu – CD, DVD, etc. – sont aux consoles de jeu.

1.2.b Utilité

Je vous ai dit plus haut que la première étape pour créer un émulateur est de regarder ce qu'il faut faire. C'est là que les ROM interviennent. Elles contiennent toutes les instructions à exécuter. C'est-à-dire que par « regarder ce qu'il faut faire », il faut comprendre « lire le contenu du fichier ROM ».

1.3 Les consoles de jeux

1.3.a Définition

Une console de jeu est un appareil électronique conçu pour permettre de lire, interpréter et afficher les informations contenues dans un support conçu à cet effet (les cartouches, CD ou DVD de jeu). Il existe deux principaux types de consoles :

— les consoles de salon, qui se branchent sur un écran pour afficher le jeu, et auxquelles on connecte accessoirement des manettes ;
— les consoles portables, de petite taille, qui possèdent leur propre écran et sont de ce fait autonomes et facilement transportables.

1.3.b Caractéristiques

La définition importe peu (je sais que vous le saviez déjà), mais je veux attirer votre attention sur les caractéristiques des consoles. Toutes les consoles de jeu sont basées sur le même principe. Elles sont constituées :

— d'un microprocesseur qui effectue les calculs ;
— de mémoire vive pour stocker les données ;
— d'une carte graphique pour afficher le rendu graphique ;
— de périphériques de contrôle et le plus souvent de manettes de jeu pour interagir avec la console.

Notre travail sera donc de remplacer tous ces éléments par ceux de notre ordinateur.

microprocesseur	processeur
mémoire vive	RAM
carte graphique	carte graphique
périphériques de contrôle	clavier, souris, joystick

Ça y est, nous venons de faire le tour de l'émulation console. Vous êtes donc en mesure de programmer votre émulateur.

Avec cette petite présentation, nous venons de voir en gros le travail que nous aurons à faire pour réaliser notre émulateur. Cette définition assez simpliste nous permet de mieux cerner la notion d'émulation.

2 Les informations sur les consoles de jeux

Maintenant que nous savons à peu près comment programmer notre émulateur, certains se sont rués sur leur moteur de recherche. Bon réflexe ! Pour simuler une machine, il faut connaître ses caractéristiques au maximum pour espérer des résultats satisfaisants. Ici, « connaître » veut juste dire qu'il faut avoir sous la main des documents qui décrivent exhaustivement la machine à émuler. Si vous possédez toutes les informations qu'il vous faut, il ne restera plus qu'à traduire ces informations dans un langage de programmation, sans même les comprendre si cela vous arrange.Mais la compréhension sera un atout majeur si vous ne voulez pas passer des journées à chercher un bogue.

2.1 Par où commencer ?

C'est beau de dire « chercher », mais que faut-il chercher ? Si vous vous rendez sur un moteur de recherche et que vous tapez « caractéristiques de la Gameboy », vous obtiendrez des informations loin d'être suffisantes pour programmer votre émulateur. Si vous êtes malchanceux, vous verrez des liens vers des sites pour configurer des émulateurs déjà exis-

tants.

2.1.a Savoir ce que vous voulez

C'est bizarre de savoir ce qu'on est censé chercher, mais c'est la vie. Imaginez-vous dans un supermarché et que vous ne savez pas ce que vous souhaitez acheter. C'est la même chose que de lancer son moteur de recherche sans savoir ce que vous cherchez. Donc pour obtenir des résultats qui correspondent à vos attentes, il faut connaître un tout petit peu le jargon de l'émulation.

2.1.b Prendre le maximum possible

Si vous faites vos recherches, prenez le maximum d'informations possible : soyez « boulimiques ». Cela vous permettra de faire des comparaisons et de voir celles qui sont le plus utilisées. En outre, la plupart des documentations ne sont pas claires sur tous les points. Elles seront donc complémentaires, ce qui facilitera grandement l'implémentation de notre émulateur.

Jetons maintenant un coup d'œil sur quelques mots clefs pour trouver notre bonheur :

— opcode ;
— CPU.

2.2 Ce qu'il faut rechercher

Dans la section précédente, j'avais dit :

> C'est-à-dire que par « regarder ce qu'il faut faire », il faut comprendre « lire le contenu du fichier ROM ».

Lorsque le fichier **binaire** est à notre disposition, on le lit en entier et on le stocke dans un tableau. Je précise quand même que cela ne se passera pas comme cela pour un DVD : on n'interprétera pas le contenu d'un seul coup mais bout par bout, étape par étape. Ces « bouts » sont appelés opcodes.

Par exemple, nous pouvons avoir un fichier de 1 000 ko, mais les informations seront lues par morceau de 1 ko. Chaque ko représentera notre opcode.

2.2.a Opcode : Operation Code

L'opcode définit en quelque sorte toute action que peut effectuer une console du point de vue calcul et rendu graphique. Si vous regardez par exemple les opcodes de la Chip 8, vous en verrez 35. La Chip 8 ne peut donc effectuer que 35 opérations, et c'est tout. Ce sera au programmeur de voir comment faire son jeu avec 35 opérations. Il va de soi qu'une console récente aura des centaines d'opcodes ! Eh oui, plus c'est récent, plus cela sera difficile d'élaborer un émulateur.

Selon la console qu'on émule, les opcodes ne sont pas de la même taille (en bits), mais le principe d'interprétation reste le même.

2.2.b CPU : Central Processing Unit

Comme son nom l'indique, le CPU gère le fonctionnement de votre console de jeu. Il effectue les différentes opérations définies par les opcodes. Cette exécution se fait toujours à une vitesse donnée : on parle de cadencement de la console. C'est une valeur qui est donnée en hertz (Hz). Par exemple, mon PC est cadencé à 2.0 GHz, soit 2×10^9 cycles d'horloge par seconde.

Si vous désirez des informations portant sur la vitesse d'interprétation des opcodes, jetez un coup d'œil sur le CPU. Si vous avez déjà testé des émulateurs et qu'ils vont trop vite ou trop lentement, le problème vient du fait que le CPU est mal implémenté (à moins que votre PC ne soit du dixième siècle).

Il faudra toujours axer ses recherches autour de ces mots. J'utilise en priorité « opcode » parce que les résultats sont le plus souvent très fructueux. Mais « CPU » donne aussi des informations très complètes.

Preuves à l'appui, recherchez :

— « Gameboy caractéristiques » ;
— puis « Gameboy opcode » ;
— et enfin « Gameboy CPU ».

Vous verrez par vous-mêmes qu'il n'y a aucune comparaison à faire. Si vous trouvez des documents avec des choses peu courantes, n'ayez pas peur, vous serez en mesure de tout déchiffrer à la fin de ce tutoriel.

2.3 Les obstacles liés aux nouvelles consoles

2.3.a Le manque d'informations

Un gros obstacle à la programmation d'émulateurs reste le manque d'informations concernant la machine à émuler. En effet, pour bien développer son programme, il faut réunir un maximum d'informations avant de débuter. Pour ce faire, il existe des moyens comme la rétro-ingénierie (reverse engineering) que l'on ne va pas aborder ici. Je ne la maîtrise d'ailleurs pas. En gros, la rétro-ingénierie permet de retrouver les caractéristiques d'une machine en effectuant divers tests sur celle-ci. Pour les machines anciennes, vous trouverez votre bonheur sur le Net,

étant donné que d'autres auront déjà fait les investigations à votre place. Cependant, vous pourrez trouver des caractéristiques différentes pour une même machine, chaque auteur pouvant avoir une compréhension différente de son fonctionnement. Malgré ces divergences, les principales caractéristiques restent le plus souvent les mêmes dans la plupart des documentations disponibles.

Maintenant, pour les machines récentes, le problème est tout autre. Vous pourrez obtenir des résultats, mais il n'y aura sûrement pas assez d'informations pour créer votre émulateur. Des séances de tâtonnement ne seront pas à exclure, ce qui rendra le travail particulièrement difficile. À moins que vous ne fassiez le *reverse engineering* vous-mêmes.

2.3.b Des consoles de plus en plus performantes

Mis à part la récupération des informations essentielles, il existe un plus gros obstacle devant vous : l'évolution des consoles. Faisons une petite comparaison :

Caractéristique	Game Boy	PSP
Cadence	2,2 Mhz	333 Mhz
Résolution	160 × 144 pixels	480 × 272 pixels
Couleur	14 nuances de gris	16,77 millions de couleurs

Ces chiffres parlent d'eux-mêmes : les nouvelles consoles sont de plus en plus puissantes. Donc, pour que votre émulateur ne tourne pas à deux à l'heure, il faudra une machine très, très puissante. Je doute même que l'émulation puisse suivre l'évolution fulgurante des consoles de jeu (avis personnel) : la XBOX 360 et la PS3 sont déjà cadencées à 3,2 GHz (plus que mon ordinateur à 2 GHz).

En plus de cela, l'organisation de ces consoles est tellement complexe qu'il est très difficile de tout implémenter en solo. Mais avec tout ce qu'il existe comme consoles à émuler, je vous assure qu'une vie entière ne suffirait pas. Jetez un coup d'œil ici pour en avoir le cœur net.

Voilà, nous savons maintenant ce qu'il faudra rechercher comme documentation autour des consoles de jeu pour les émuler toutes, une par une.

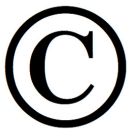
3 Législation

Est-ce que vous avez déjà vu un domaine dans lequel les juristes ne sont pas impliqués ? Bien qu'il n'existe pas encore, à ma connaissance, de lois spécifiques à l'émulation, il en existe qui défendent les droits d'auteur et la propriété intellectuelle. Les consoles de jeu sont toutes sous licences propriétaires et sources de revenus. Le marché est très lucratif et ne cesse de s'étendre. Dès lors, créer un outil gratuit pour les substituer peut susciter des débats.

A-t-on réellement le droit de créer un émulateur console ?

3.1 Les obstacles

L'un des plus gros problèmes pour l'émulation est sans doute le respect du droit d'auteur (*copyright*). En plus de ne divulguer aucune information sur les caractéristiques techniques de leurs consoles, les constructeurs les protègent jalousement. Ainsi, avec les licences utilisées, il est formellement interdit de prendre ou d'utiliser une partie ou la totalité de leur travail. D'ailleurs, les discussions sur la légalité de l'émulation font couler beaucoup d'encre.



Nous pouvons cependant retenir que la programmation d'un émulateur est **totalement légale** tant qu'on ne fait pas usage de **ressources sous droits d'auteur** (le BIOS, par exemple).

Ce genre de symboles vous est sûrement familier, ils sont présents sur presque tous les jeux vidéo pour signifier qu'ils sont sous droits d'auteur.

Tout au long de ce tutoriel, nous ferons du HLE (High Level Emulation), c'est-à-dire que nous programmerons tout ce dont nous aurons besoin.

En résumé, votre émulateur restera légal tant que vous le développez avec vos propres ressources et que vous le redistribuez sans aucun jeu sous droits d'auteur.

Cependant, l'utilisation d'un émulateur nécessite des jeux ou ROM qui sont le plus souvent difficiles d'accès et non libres d'utilisation. (Sinon : illégalité).

3.2 Une lueur d'espoir

Après ce que l'on vient de voir, il est légitime de se poser cette question. Mais il existe toujours des exceptions à toute règle. Certains jeux (notamment ceux que nous allons utiliser) sont dans le domaine public et sont dès lors libres d'utilisation et de redistribution. En plus de cela, vous êtes autorisés à posséder une ROM à condition d'être en possession du jeu original, cette ROM étant considérée comme une sauvegarde.

À défaut de tout cela, vous pourrez utiliser un homebrew. Un homebrew est un jeu de console créé par un amateur et ces jeux sont le plus souvent libres. Pour éviter de faire un cours de droit ici, je vous laisse creuser si cela vous intéresse. Pour de plus amples explications ou preuves, vous pouvez consulter :

— Wikipédia : Émulation (lien 2) ;
— Wikipédia : Propriété intellectuelle (lien 3) ;
— et pour les plus téméraires, Legifrance : lien 4.

Il existe des jeux avec des mentions « Non utilisable en dehors du support d'origine », donc attention !

Une dernière chose, je vous laisse le fameux :

Je, BestCoder, décline toute responsabilité sur des agissements qui pourraient survenir à la suite de la lecture de ce tutoriel.
BestCoder ne pourrait être tenu responsable sur aucun plan et ce en aucun cas.

Je vous invite à accorder une grande importance à la législation pour éviter des ennuis inutiles. Après ce long discours, j'espère que vous avez bien cerné le sujet et que votre curiosité a été bien assouvie. À présent, nous allons voir ce qu'il nous faut comme bagage pour nous lancer dans notre périlleuse aventure.

Nous voilà fin prêts et avertis pour nous lancer dans notre aventure. Puisque nous savons à la fois ce que nous voulons et comment l'obtenir, la moitié du travail est déjà effectuée, il ne reste plus qu'à passer à l'action.

Je signale qu'il faut une certaine maîtrise de l'hexadécimal, du binaire et des opérations logiques pour suivre ce tutoriel sans trop de difficultés.

*Retrouvez l'article de **BestCoder** en ligne : lien 5*
Cet article fait partie d'une série de tutoriels : lien 6

OpenGL Moderne : le premier triangle.

OpenGL 3 facilite l'écriture des choses compliquées, mais possède l'inconvénient de rendre l'affichage d'un simple triangle relativement difficile.

1 Introduction

Cela va être un autre long tutoriel. OpenGL 3 facilite l'écriture des choses compliquées, mais possède l'inconvénient de rendre l'affichage d'un simple triangle relativement difficile.
N'oubliez pas de copier/coller le code régulièrement.

Si le programme crashe au démarrage, vous l'exécutez certainement à partir du mauvais répertoire. Lisez PRÉCAUTIONNEUSEMENT *le premier tutoriel sur comment configurer Visual Studio : lien 7*

2 Les Vertex Array Object

Je ne vais pas m'enfoncer dans les détails maintenant, mais vous devez créer un Vertex Array Object (VAO) et le définir comment objet courant.

```
1    GLuint VertexArrayID;
2    glGenVertexArrays(1, &VertexArrayID)
         ;
3    glBindVertexArray(VertexArrayID);
```

Faites-le une fois que votre fenêtre est créée (= après la création du contexte OpenGL) et avant tout autre appel OpenGL. Si vous souhaitez vraiment en apprendre plus sur les VAO, il y a quelques autres tutoriels sur le Web, mais ce n'est pas très important.

3 Coordonnées écran

Un triangle est défini par trois points. Lorsque l'on parle de « points » en graphismes 3D, on utilise habituellement le terme de « sommet » (en anglais « vertex », « vertices » au pluriel). Un sommet possède trois coordonnées : X, Y et Z. Vous pouvez imaginer ces trois coordonnées de la manière suivante :

— X est sur votre droite ;

— Y, vers le haut ;

— Z est derrière vous (oui, derrière et non devant).

Mais voici une meilleure méthode pour les visualiser : utilisez la règle de la main droite :

— X est votre pouce ;

— Y, votre index ;

— Z est votre majeur. Si vous placez votre pouce sur la droite et votre index vers le ciel, votre majeur pointera aussi derrière votre dos.

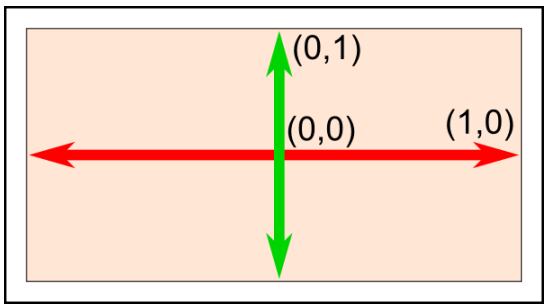
Il est étrange d'avoir l'axe Z dans cette direction. Pourquoi est-ce ainsi ? Pour faire court : car cent années de « règle de la main droite » vous donneront des outils pratiques. La seule contrepartie est un axe Z contre-intuitif.

Mis à part cela, remarquez aussi que vous pouvez bouger votre main librement : votre X, Y et Z suivront. On reviendra sur ce point.

Donc, on a besoin de trois points 3D afin de faire un triangle ; les voici :

```
1  // Un tableau de trois vecteurs qui repr
       ésentent trois sommets
2  static const GLfloat
       g_vertex_buffer_data[] = {
3     -1.0f, -1.0f, 0.0f,
4     1.0f, -1.0f, 0.0f,
5     0.0f,  1.0f, 0.0f,
6  };
```

Le premier sommet est $(-1, -1, 0)$. Cela signifie que, sauf si nous le transformons d'une quelconque manière, il sera affiché à la position $(-1, -1)$ à l'écran. Qu'est-ce que cela donne ? L'origine de l'écran est au centre, l'axe X va vers la droite, comme toujours et l'axe Y vers le haut. Voici ce que cela donne sur un écran large :



C'est une chose que vous ne pouvez modifier, c'est intégré dans votre carte graphique. Donc $(-1, -1)$ est le coin inférieur gauche de votre écran. $(1, -1)$ est le coin inférieur droit et $(0, 1)$, le milieu haut. Donc ce triangle devrait couvrir la majorité de l'écran.

4 Dessiner notre triangle

La prochaine étape est de fournir ce triangle à OpenGL. Pour cela il faut créer un tampon [1] :

```
1  //Ceci identifiera notre tampon de
       sommets
2  GLuint vertexbuffer;
3  // Génère un tampon et place l'
       identifiant dans 'vertexbuffer'
4  glGenBuffers(1, &vertexbuffer);
5  // Les commandes suivantes vont parler
       de notre tampon 'vertexbuffer'
6  glBindBuffer(GL_ARRAY_BUFFER,
       vertexbuffer);
7  // Fournit les sommets à OpenGL.
8  glBufferData(GL_ARRAY_BUFFER, sizeof(
       g_vertex_buffer_data),
       g_vertex_buffer_data, GL_STATIC_DRAW
       );
```

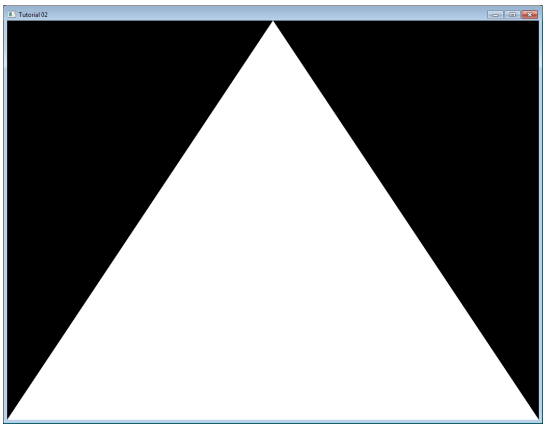
Cela n'est nécessaire qu'une seule fois au lancement du programme.

Maintenant, la boucle principale, où on a l'habitude de ne « rien » dessiner. On peut maintenant dessiner un fantastique triangle :

```
1  // premier tampon d'attributs : les
       sommets
2  glEnableVertexAttribArray(0);
3  glBindBuffer(GL_ARRAY_BUFFER,
       vertexbuffer);
4  glVertexAttribPointer(
5     0,                  // attribut 0.
          Aucune raison particulière pour 0
          , mais cela doit correspondre au
          "layout" dans le shader
6     3,                  // taille
7     GL_FLOAT,           // type
8     GL_FALSE,           // normalisé ?
9     0,                  // nombre d'
          octets séparant deux sommets dans
           le tampon
10    (void*)0            // décalage du
          tableau de tampon
11 );
12 // Dessine le triangle !
13 glDrawArrays(GL_TRIANGLES, 0, 3); // Dé
       marre à partir du sommet 0; 3
       sommets au total -> 1 triangle
14 glDisableVertexAttribArray(0);
```

1. Note du traducteur : en anglais « buffer » : un espace mémoire qui contiendra des données

Si vous avec une carte NVIDIA, vous pouvez voir le résultat (pour les autres cartes, continuez de lire) :



Ce blanc est bien ennuyeux. Voyons voir comment l'améliorer en l'affichant en rouge. Cela peut être fait avec quelque chose appelé « shader ».

5 Shaders

5.1 Compilation de shader

Dans la configuration la plus simple, vous avez besoin de deux shaders : un appelé « Vertex Shader », qui sera exécuté pour chaque sommet et l'autre appelé « Fragment Shader », qui sera exécuté pour chaque fragment. Comme on utilise un antialiasing 4x, on a quatre échantillons pour chaque pixel.

Les shaders se programment avec un langage appelé GLSL : GL Shader Language, qui est intégré à OpenGL. Contrairement au C ou au Java, le GLSL doit être compilé durant l'exécution du programme, ce qui signifie que chaque fois que vous lancez votre application, tous vos shaders sont recompilés.

Les deux shaders sont généralement dans des fichiers distincts. Dans cet exemple, nous avons *SimpleFragmentShader.fragmentshader* et *SimpleVertexShader.vertexshader*. L'extension importe peu, cela aurait pu être .txt ou .glsl.

Voici enfin le code pour charger des shaders. Il n'est pas très important de le comprendre entièrement, car vous ne le faites qu'une seule fois dans le programme, les commentaires suffiront. Comme cette fonction va être utilisée dans tous les autres tutoriels, elle est placée dans un fichier à part : *common/loadShader.cpp*. Notez que tout comme les tampons, les shaders ne sont pas directement accessibles : nous n'avons qu'un identifiant. L'implémentation actuelle est cachée par le pilote.

```
1  GLuint LoadShaders(const char * vertex_file_path,const char * fragment_file_path){
2
3      // Crée les shaders
4      GLuint VertexShaderID = glCreateShader(GL_VERTEX_SHADER);
5      GLuint FragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);
6
7      // Lit le code du vertex shader à partir du fichier
8      std::string VertexShaderCode;
9      std::ifstream VertexShaderStream(vertex_file_path, std::ios::in);
10     if(VertexShaderStream.is_open())
11     {
12         std::string Line = "";
13         while(getline(VertexShaderStream, Line))
14             VertexShaderCode += "\n" + Line;
15         VertexShaderStream.close();
16     }
17
18     // Lit le code du fragment shader à partir du fichier
19     std::string FragmentShaderCode;
20     std::ifstream FragmentShaderStream(fragment_file_path, std::ios::in);
21     if(FragmentShaderStream.is_open()){
22         std::string Line = "";
23         while(getline(FragmentShaderStream, Line))
24             FragmentShaderCode += "\n" + Line;
25         FragmentShaderStream.close();
26     }
27
28     GLint Result = GL_FALSE;
29     int InfoLogLength;
```

```
30
31      // Compile le vertex shader
32      printf("Compiling shader : %s\n", vertex_file_path);
33      char const * VertexSourcePointer = VertexShaderCode.c_str();
34      glShaderSource(VertexShaderID, 1, &VertexSourcePointer , NULL);
35      glCompileShader(VertexShaderID);
36
37      // Vérifie le vertex shader
38      glGetShaderiv(VertexShaderID, GL_COMPILE_STATUS, &Result);
39      glGetShaderiv(VertexShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
40      std::vector<char> VertexShaderErrorMessage(InfoLogLength);
41      glGetShaderInfoLog(VertexShaderID, InfoLogLength, NULL, &VertexShaderErrorMessage[
            0]);
42      fprintf(stdout, "%s\n", &VertexShaderErrorMessage[0]);
43
44      // Compile le fragment shader
45      printf("Compiling shader : %s\n", fragment_file_path);
46      char const * FragmentSourcePointer = FragmentShaderCode.c_str();
47      glShaderSource(FragmentShaderID, 1, &FragmentSourcePointer , NULL);
48      glCompileShader(FragmentShaderID);
49
50      // Vérifie le fragment shader
51      glGetShaderiv(FragmentShaderID, GL_COMPILE_STATUS, &Result);
52      glGetShaderiv(FragmentShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
53      std::vector<char> FragmentShaderErrorMessage(InfoLogLength);
54      glGetShaderInfoLog(FragmentShaderID, InfoLogLength, NULL, &
            FragmentShaderErrorMessage[0]);
55      fprintf(stdout, "%s\n", &FragmentShaderErrorMessage[0]);
56
57      // Lit le programme
58      fprintf(stdout, "Linking program\n");
59      GLuint ProgramID = glCreateProgram();
60      glAttachShader(ProgramID, VertexShaderID);
61      glAttachShader(ProgramID, FragmentShaderID);
62      glLinkProgram(ProgramID);
63
64      // Vérifie le programme
65      glGetProgramiv(ProgramID, GL_LINK_STATUS, &Result);
66      glGetProgramiv(ProgramID, GL_INFO_LOG_LENGTH, &InfoLogLength);
67      std::vector<char> ProgramErrorMessage( max(InfoLogLength, int(1)) );
68      glGetProgramInfoLog(ProgramID, InfoLogLength, NULL, &ProgramErrorMessage[0]);
69      fprintf(stdout, "%s\n", &ProgramErrorMessage[0]);
70
71      glDeleteShader(VertexShaderID);
72      glDeleteShader(FragmentShaderID);
73
74      return ProgramID;
75  }
```

5.2 Notre vertex shader

Écrivons le vertex shader.
La première ligne indique au compilateur que l'on va utiliser la syntaxe de OpenGL 3.

```
1  Écrivons le vertex shader.
2
3  La première ligne indique au compilateur
       que l'on va utiliser la syntaxe de
       OpenGL 3.
```

La seconde ligne déclare les données d'entrées :

```
1  layout(location = 0) in vec3
       vertexPosition_modelspace;
```

Expliquons-la en détail :

— « vec3 » est un vecteur de trois composantes dans le GLSL. Il est similaire (mais différent) au `glm::vec3` que nous avons utilisé pour définir notre triangle. Le point important est que si on utilise trois composantes en C++, nous utilisons aussi trois composantes dans le GLSL ;

— « layout(location = 0) » se réfère au tampon que l'on fournit à l'attribut `vertexPosition_modelspace`. Chaque sommet peut avoir de nombreux attributs : une position, une ou plusieurs couleurs, une ou plusieurs coordonnées de texture et plein d'autres choses. OpenGL ne sait pas ce qu'est une couleur : il ne voit qu'un `vec3`. Donc on doit lui dire quel tampon correspond à quelle entrée. Nous le faisons en définissant le « layout » à la même valeur que le premier paramètre de la fonction `glVertexAttribPointer`. La valeur « 0 » n'est pas importante, cela aurait pu être 12 (mais, elle ne peut être supérieure à `glGetIntegerv(GL_MAX_VERTEX_ATTRIBS, &v)`),

la chose importante est qu'elle soit la même des deux côtés ;

— « vertexPosition_modelspace » aurait pu être n'importe quoi. Il contiendra la position des sommets pour chaque exécution du vertex shader ;

— « in » signifie que ce sont des données d'entrée. Bientôt on va voir le mot clé « out ».

La fonction qui est appelée pour chaque sommet est appelée `main`, tout comme en C :

```
1 void main(){
```

La fonction principale va simplement définir la position du vertex à ce qui est dans le tampon. Donc, si on donne $(1, 1)$, le triangle aura l'un de ces sommets au coin supérieur droit de l'écran. On verra dans le prochain tutoriel comment effectuer des calculs plus intéressants sur les positions passées au shader.

```
1     gl_Position.xyz =
          vertexPosition_modelspace;
2     gl_Position.w = 1.0;
3 }
```

`gl_Position` est l'une des variables du langage : vous devez assigner une valeur à celle-ci. Tout le reste est optionnel ; on verra « tout le reste » dans le quatrième tutoriel.

5.3 Notre fragment shader

Pour le premier fragment shader, on va faire quelque chose de très simple : définir la couleur de chaque fragment à rouge. (Rappelez-vous, il y a quatre fragments dans un pixel, car nous utilisons l'antialiasing 4x.)

```
1 #version 330 core
2 out vec3 color;
3
4 void main(){
5     color = vec3(1,0,0);
6 }
```

Donc voilà, `vec3(1,0,0)` signifie rouge. Cela est dû aux écrans d'ordinateur. La couleur est représentée par un triplet rouge, vert, bleu, dans cet ordre. Donc $(1, 0, 0)$ indique complètement rouge, pas de vert et pas de bleu.

5.4 Rassembler le tout

Avant la boucle principale, on appelle la fonction `LoadShaders` :

```
1 // Crée et compile notre programme GLSL
      à partir des shaders
2 GLuint programID = LoadShaders( "
      SimpleVertexShader.vertexshader", "
      SimpleFragmentShader.fragmentshader"
       );
```

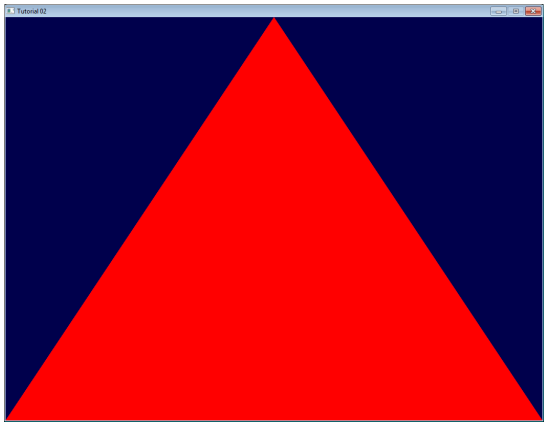
En premier, dans la boucle principale, on nettoie l'écran. Cela changera la couleur de fond en bleu foncé à cause de l'appel `glClearColor (0.0f, 0.0f, 0.4f, 0.0f)` au-dessus de la boucle principale.

```
1 glClear(GL_COLOR_BUFFER_BIT |
      GL_DEPTH_BUFFER_BIT);
```

Puis on indique à OpenGL que l'on souhaite les shaders vus précédemment :

```
1 // Utilise notre shader
2 glUseProgram(programID);
3
4 // Dessine le triangle...
```

...et voilà, un triangle rouge !



Dans le prochain tutoriel (lien 8), on étudiera les transformations : comment définir une caméra, déplacer les objets, etc.

6 Remerciements

Cet article est une traduction autorisée dont le texte original peut être trouvé sur opengl-tutorials.org : lien 9.
Retrouvez l'article d' ***Alexandre Laurent*** *en ligne : lien 10*
Cet article fait partie d'une série de tutoriels : lien 11

Java

Les dernières news Java

Java 8 est disponible

La plate-forme se met aux expressions lambdas, tour d'horizon des nouveautés

Si les versions 6 et 7 de Java étaient des évolutions douces : Java 8 est d'un tout autre ordre. Plus de 56 nouvelles fonctionnalités ont été ajoutées (http ://openjdk.java.net/projects/jdk8/features). Les arrivées des lambdas, des méthodes par défaut, des interfaces fonctionnelles et de Stream vont modifier en profondeur le langage et donc l'écosystème Java tout entier. Nous pouvons aussi citer l'incorporation d'une nouvelle API pour gérer les dates, de nouvelles annotations et d'un nouveau moteur d'exécution JavaScript.

Java 8 devrait ainsi avoir un impact au moins aussi important que Java 5 à son époque (rappelez-vous l'apparition des Generics). Il faut donc s'y préparer dès à présent. Voici quelques nouveautés plus en détail.

Les nouveautés du langage

Interfaces fonctionnelles : connues précédemment sous le nom de Single Abstract Method interfaces (SAM Interfaces), cette nouveauté introduit les interfaces qui possèdent uniquement une seule méthode d'instance abstraite. Les plus connues sont java.lang.Runnable, java.awt.event.ActionListener, java.util.Comparator. Avec Java 8, elles portent le nom d'interfaces fonctionnelles. Dès lors qu'une interface possède une seule méthode d'instance abstraite, elle est désignée comme interface fonctionnelle. Il est aussi possible d'annoter l'interface par @FunctionalInterface. Si une interface est annotée ainsi et possède plus d'une méthode d'instance abstraite, une erreur de compilation sera produite. C'est un peu le même principe qu'avec l'annotation @Override.

L'interface ci-dessous Runnable possède une méthode et est annotée @FunctionalInterface.

```
1  @FunctionalInterface
2  public interface Runnable {
3      void run();
4  }
```

Le nouveau package java.util.function propose d'ailleurs un certain nombre d'interfaces fonctionnelles répondant à divers usages.

Lambdas : il s'agit de la plus grosse nouveauté de Java 8. Décrite depuis la JSR 335 (https ://-jcp.org/en/jsr/detail ?id=335), cette fonctionnalité permet d'apporter la puissance de la programmation fonctionnelle dans Java. Une expression lambda peut être assimilée à une fonction anonyme, ayant potentiellement accès au contexte (variables locales et/ou d'instance) du code appelant. Ces "fonctions anonymes" peuvent être affectées dans une interface fonctionnelle. Le code de l'expression lambda servira ainsi d'implémentation pour la méthode abstraite de l'interface. On peut donc les utiliser avec n'importe quel code Java utilisant une telle interface, à condition que les signatures de la méthode correspondent à celle de l'expression lambda.

La syntaxe utilisée est la suivante : (paramètres) -> code ou (paramètres) -> code quand il y a plus d'une instruction.

Prenons l'exemple du tri des éléments d'une collection.

```
1  Arrays.sort(testStrings, new Comparator<
       String>() {
2      @Override
3      public int compare(String s1, String
           s2) {
4          return(s1.length() - s2.length()
               );
5      }
6  });
```

En utilisant les lambdas, la nouvelle écriture sera :

```
1  // Forme longue :
```

```
2  Arrays.sort(testStrings, (String s1,
       String s2) -> { return s1.length() -
        s2.length(); });
3
4  // Forme courte (possible uniquement s'
       il n'y a qu'une instruction) :
5  Arrays.sort(testStrings, (String s1,
       String s2) -> s1.length() - s2.
       length());
6
7  // Forme courte avec type implicite des
       paramètres
8  // (le type est déduit par le
       compilateur via l'inférence)
9  Arrays.sort(testStrings, (s1, s2) -> s1.
       length() - s2.length());
```

Les interfaces fonctionnelles servent aux lambdas, facilitant ainsi l'écriture puisqu'elles permettent d'écrire l'implémentation de façon plus concise. Nous montrons ci-dessous un exemple d'implémentation de l'interface Runnable.

```
1  Runnable r1 = () -> { System.out.println
       ("My Runnable"); };
```

Plus de détails sur les lambdas sont disponibles dans un tutoriel publié récemment : lien 12

Références de méthode : une référence de méthode est utilisée pour définir une méthode en tant qu'implémentation de la méthode abstraite d'une interface fonctionnelle. La notation utilise le nom de la classe ou une instance de la classe, suivi de l'opérateur « : : » et du nom de la méthode à référencer. Le type des paramètres sera déduit du contexte selon l'interface fonctionnelle vers laquelle on affecte la référence.

On peut distinguer quatre types de méthodes références :

— Les références vers une méthode static, qui s'utilisent toujours avec le nom de la classe en préfixe. La signature de la référence correspond alors à la signature de la méthode.

```
1  Supplier<Double> random = Math::
       random;
2  double result = random.get(); //
       Math.random();
```

— Les références vers une méthode d'instance, liées à une instance spécifique, qui s'utilisent toujours avec l'instance en préfixe. Ici également, la signature de la référence correspond à la signature de la méthode, et tous les appels s'appliqueront sur l'instance définie dans la référence de méthode :

```
1  Random r = new Random();
2  Supplier<Double> random2 = r::
       nextDouble;
3  double result2 = random2.get(); //
       r.nextDouble();
```

— Les références vers une méthode d'instance, mais sans lien avec une instance précise. Comme pour les méthodes static, on utilisera comme préfixe le nom de la classe. La signature de la référence correspond alors à la signature de la méthode, précédée par un argument du type de la classe, qui correspondra à l'instance sur laquelle on appellera la méthode :

```
1  Function<Random,Double> random3 =
       Random::nextDouble;
2  Random r1 = new Random();
3  Random r2 = new Random();
4  Random r3 = new Random();
5  double result1 = random3.apply(r1);
       // r1.nextDouble();
6  double result2 = random3.apply(r2);
       // r2.nextDouble();
7  double result3 = random3.apply(r3);
       // r2.nextDouble();
```

— Enfin, il est possible de référencer un constructeur en utilisant le mot-clef "new" comme nom de méthode. Très pratique pour créer une fabrique :

```
1  Function<String, Thread> factory =
       Thread::new;
2  Thread t = factory.apply("name");
       // new Thread("name");
```

Les références de méthodes sont une alternative aux expressions lambdas, lorsqu'il n'y a qu'une seule et unique méthode à exécuter, pour une syntaxe encore plus claire :

```
1  Random r = new Random();
2
3  Supplier<Double> random = Math::random;
4  Supplier<Double> random2 = r::nextDouble
       ;
5  Function<Random,Double> random3 = Random
       ::nextDouble;
6  Function<String, Thread> factory =
       Thread::new;
7
8  Supplier<Double> random = () -> Math.
       random();
9  Supplier<Double> random2 = () -> r->
       nextDouble();
10 Function<Random,Double> random3 = (
       Random random) -> random.nextDouble
       ();
11 Function<String, Thread> factory = (
       String name) -> new Thread(name);
```

Cela peut également être une alternative intéressante à l'API de reflection, puisque cela permet un code sécurisé.

Méthodes par défaut (Defender Methods) : cette fonctionnalité permet de proposer une implémentation dite par "défaut" aux méthodes déclarées dans les interfaces. Par conséquent, depuis Java 8, une interface Java contient du code. L'avantage est de pouvoir faire évoluer les interfaces sans avoir à tout casser.

Dans l'exemple ci-dessous, une interface Person déclare deux méthodes. La méthode sayHello est dite par défaut via le mot clé default. Toute implémentation de Person imposera que la méthode sayGoodBye() soit implémentée. Pour sayHello, l'implémentation ne sera pas obligatoire, même si elle reste bien sûr possible.

```
interface Person {
    void sayGoodBye();
    default void sayHello() {
        System.out.println("Hello there!
            ");
    }
}
```

Les méthodes par défaut permettent ainsi de faire évoluer l'API des interfaces sans provoquer de grosses incompatibilités dues à l'absence d'implémentation dans les classes qui les implémentent. L'API de base en profite grandement en enrichissant certaines de ses interfaces (en particulier dans l'API de Collections dont les interfaces s'enrichissent de plusieurs méthodes).

Plus de détails sur cette nouveauté peuvent être trouvés sur le tutoriel publié dernièrement : lien 13

Méthodes static dans les interfaces : Java 8 propose également la possibilité de créer des méthodes statiques depuis une interface Java.

Dans l'exemple ci-dessous, une interface Person déclare une méthode statique.

```
interface Person {
    static void sayHello() {
        System.out.println("Hello there!
            ");
    }
}
```

Cela peut s'avérer utile pour proposer des méthodes utilitaires liées à l'interface (comme une "fabrique" par exemple).

Les mises à jour de l'API

Stream et parallèles streams sur les collections : Java 8 apporte également la notion de Stream, qui représente un flux de données que l'on peut manipuler à la volée. L'utilisation d'un Stream se compose de trois parties :

- la création du flux à partir d'une source de données qui peut être très variée (un tableau, une collection, un flux d'entrée/sortie, des données générés à la volée, etc.) ;
- des opérations intermédiaires, qui permettent de transformer le flux en un autre flux (en filtrant des données ou en les transformant par exemple) ;
- une opération terminale, qui permet d'obtenir un résultat ou d'effectuer une opération spécifique.

Par exemple, pour créer un flux à partir d'une collection, on utilisera tout simplement la nouvelle méthode stream() de Collection. On peut alors appliquer autant d'opérations intermédiaires que nécessaire, comme filter(), qui permet de filtrer certaines données, map() qui permet de modifier la donnée à la volée, distinct() qui permet d'éviter les doublons, sorted() qui permet de les trier, ou encore limit() qui restreint la taille des données. Une fois toutes nos opérations intermédiaires effectuées, il faut alors appliquer la méthode finale qui va déterminer notre action. On peut par exemple utiliser forEach() pour itérer sur toutes ces valeurs, min()/max() pour récupérer seulement la valeur min/max, finAny()/findFirst() pour récupérer un élément correspondant aux critères, etc.

Bien sûr, toutes ces opérations sont pensées pour être utilisées avec des expressions lambdas ou des références de méthodes.

API java.time (lien 14) : la nouvelle API date, heure et calendrier basée sur la JSR 310 est disponible dans les packages java.time.*. Les classes sont désormais immuables et thread-safe. L'utilisation des méthodes chaînables rendent le code plus lisible. La nouvelle API différencie également deux modèles de temps : le temps machine et le temps humain. Pour une machine, le temps n'est qu'un entier augmentant depuis l'epoch (01 janvier 1970 00h00min00s0ms0ns). Pour un humain en revanche, il s'agit d'une succession de champs ayant une unité (année, mois, jours, heure, etc.). À noter également que le mois "0" n'existe plus et le mois "1" correspond au mois de janvier. Plus de détails sur cette nouveauté peuvent être trouvés sur le tutoriel publié dernièrement : lien 15.

Annotations multiples (lien 16) : Java 8 offre maintenant la possibilité de répéter plusieurs fois une annotation de même type pour un élément donné d'un programme.

Dans l'exemple ci-dessous, l'annotation @Schedule est utilisée deux fois, ce qui n'était pas permis dans les versions antérieures.

```
@Schedule(dayOfMonth="last")
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }
```

Nashorn, le nouveau moteur Javascript (lien 17) : un moteur JavaScript proposé par défaut depuis le JDK. Plus de détails sur cette nouveauté peuvent être trouvés sur le tutoriel publié récemment : lien 18

Nous pourrions continuer la liste des modifications en précisant également la suppression du PermGen pour la machine virtuelle, les nombreux ajouts pour JavaFX (lien 19) ou finalement l'ajout de classes dont l'intérêt va se découvrir au fil du temps. L'on peut citer, par exemple, les classes LongAdder et DoubleAdder (lien 20).

Vous l'aurez compris, Java 8 nous apporte une vraie évolution dans le paysage Java. Pour plus de détails, n'hésitez pas à faire un tour sur :

- le site OpenJDK : lien 21
- la discussion sur le forum proposée par Adi-Guba : lien 22

Les avis de l'équipe Java de Developpez.com

Fabrice Bouyé (bouye)

> Le JDK 8 est enfin là !

À la vue des expressions lambdas, certains diront qu'avec quelques années de retard, Java met enfin à niveau sa syntaxe événementielle pour rattraper celle de C#, Groovy ou encore Scala. Cependant, on aurait tort de penser qu'il s'agit là d'un simple sucre syntaxique destiné à donner un aspect moderne au langage et à le rendre plus attrayant !

Ces nouvelles fonctionnalités arrivent, en effet, avec une refonte majeure de la JVM, que ce soit au niveau du support des expressions lambdas elles-mêmes, des références de fonctions (similaires aux pointeurs de fonctions pour ceux ayant fait du C/C++) ou encore de la refonte des collections par l'introduction des streams. Outre l'aspect extérieur (les API que nous, programmeurs, sommes amenés à manipuler), désormais la JVM s'adapte aux traitements en parallèle et au support du multicœur, et le tout sans devoir manuellement manipuler des threads ou des workers !

De plus, l'excellente bibliothèque des gestions du temps JODA-Time a fini par évoluer pour devenir le JSR 310, la nouvelle API de temps et de date du JDK 8. Simple, efficace et pratique d'emploi, elle promet de nous faire rapidement oublier les fastidieuses opérations sur java.util.Calendar et java.util.Date, toujours promptes à créer toutes sortes de fiascos temporels si on n'y prend pas garde.

J'attends beaucoup de ce premier pack de fonctionnalités. Cependant, je garde également l'œil ouvert sur ce qui est encore à venir, ces autres fonctionnalités qui n'ont pu être intégrées au JDK 8 (jigsaw, etc.). Les plans pour le futur JDK 9 semblent toujours aussi appétissants...

Mickael Baron (Mickael Baron)

Je suis très enthousiaste de l'arrivée de Java 8 et des nombreuses fonctionnalités qui sont proposées. À mon avis, il faudra un certains temps afin de maîtriser toutes les subtilités des améliorations. Je compte sur les nombreux rédacteurs de Developpez.com pour nous faire découvrir tous les secrets de cette version.

Je suis également curieux de voir comment cette nouvelle version va s'imposer dans le monde de l'entreprise où on croise encore des JDK 1.4.

La fonctionnalité la plus intéressante de mon point de vue reste les implémentations par défaut, je regrette même qu'elle arrive aussi tard. Les évolutions des interfaces seront plus faciles à réaliser. Pour les lambdas, je ne suis pas encore assez expérimenté pour en comprendre toutes les subtilités mais je compte bien m'y mettre quand mon Eclipse préféré sera disponible.

Bienvenue Java 8 !!!

Frédéric Martini (AdiGuba)

Java continue son évolution.

D'aucuns diront qu'il s'agit d'une évolution bien lente. Je parlerais plutôt d'évolution prudente et intelligente.

Certes Java ne fait pas la course aux fonctionnalités, mais elles ont le mérite d'être réfléchies et pesées.

Je trouve par exemple la notion d'interface fonctionnelle particulièrement ingénieuse, car non seulement cela permet de ne pas s'embêter à manipuler un nouveau type de données (des "fonctions-types" ou des types delegate), mais cela permet également une retro-compatibilité avec de nombreuses APIs existantes, et cela sans le moindre effort !

Les expressions Lambdas (et les références de méthode) permettront également de concevoir de nouveaux types d'APIs, mais n'oublions pas non plus les méthodes par défaut, qui permettront une meilleure évolution des APIs existantes et futures (et d'ailleurs l'API de Collections en profite bien).

Je pense vraiment qu'il s'agit d'une version qui marquera sans doute un tournant aussi important que J2SE 5.0...

Thierry Leriche-Dessirier (thierryler)

Selon moi, les trois points les plus importants de cette nouvelle version sont 1) les Lambdas 2) les streams et 3) la disparition du permgen.

L'ajout des interfaces fonctionnelles et des méthodes par défaut a été imposé par les Lambdas. Ça reste très intéressant malgré tout. À mon sens, c'est tout de même au second plan.

Les Lambdas sont synonymes de programmation fonctionnelle. Si ce n'était qu'une question de syntaxe, ça aurait eu moins d'impact. Des frameworks tels que Guava le proposent déjà, comme je le montre dans l'article intitulé "Tutoriel d'introduction à la programmation fonctionnelle avec Guava" (lien 23).

Avec Java 8, on va bien plus loin. On a aussi accès au "reduce" du "filter-map-reduce" mais, et surtout, cela travaille pour de vrai dans un contexte multi-thread/multicoeur. On pouvait déjà travailler en multicoeur en Java, surtout depuis Java 7 qui a bien défriché le terrain pour Java 8, mais il fallait se coltiner les points techniques à chaque fois, avec les risques que cela comporte et avec une syntaxe non adaptée. Java 8 rend cela "simple".

Il ne faut pas oublier que les fabricants de processeurs ne cherchent plus à augmenter la fréquence. C'est désormais une course aux cœurs. L'avenir est donc aux traitements parallèles et aux lambdas. Encore faudra-t-il bien comprendre les patterns de la programmation fonctionnelle, pour ne pas les utiliser de travers et à tout-va.

Le permgen, quant à lui, disparaît et ce n'est pas trop tôt. On pointe les projecteurs sur le langage en oubliant qu'il se passe des choses en coulisse et plus spécifiquement du côté de la JVM. N'oublions pas que Java, c'est un langage et une Virtual Machine. C'est très important. La VM est conçue pour optimiser les programmes (plan d'exécution) à l'utilisation. Ce n'est pas juste un lanceur de classe Main. Avec la disparition du permgen, on voit le rapprochement des plusieurs éditeurs de JVM/GC qui porte ses fruits.

Télécharger la nouvelle version de Java : lien 24

Commentez la news d' ***Hinault Romaric*** *en ligne : lien 25*

Les derniers tutoriels et articles

Tutoriel Guava sur les fonctions de hashage et les I/O

Guava est une bibliothèque, de chez Google, proposant de nombreux outils pour améliorer les codes des programmes Java. Elle permet, entre autres, de manipuler les collections, de jouer efficacement avec les immutables, d'éviter la gestion des beans nuls, de s'essayer à la programmation fonctionnelle, de cacher les objets, de les simplifier, et bien d'autres choses...
Dans ce huitième article sur Guava, vous allez voir des trucs rigolos (mais pas simples) sur les hash et vous réconcilier avec les I/O.

1 Introduction

Cet article est le huitième d'une série consacrée à la bibliothèque Guava :

- introduction et installation : lien 26 ;
- collections : lien 27 ;
- programmation fonctionnelle : lien 28 ;
- utilitaires : lien 29 ;
- cache et concurrence : lien 30 ;
- tout pour vos Strings et primitifs : lien 31 ;
- un peu de maths : lien 32 ;
- hash et I/O : lien 33.

1.1 Versions des logiciels et bibliothèques utilisées

Pour écrire ce document, j'ai utilisé les versions suivantes :

- Java JDK 1.6.0_24-b07 ;
- Eclipse Indigo 3.7 JEE 64b ;
- Maven 3.0.3 ;
- JUnit 4.10 ;
- Guava 14.0.

J'utilise Java 6, car Java 7 n'est pas encore très répandu en entreprise. C'est ce que je vérifie durant mes conférences lorsque je demande qui utilise Java 7 sur ses serveurs de production, mais que très peu de mains se lèvent...

2 Le coin des bûcherons (hash)

On a déjà un peu parlé, plus haut, des fonctions de hash. C'est à la fois simple et complexe. Un des objectifs des implémentations de la méthode « hashCode() » est d'être très rapide. Mais, du coup, les algorithmes n'empêchent pas (toujours) les collisions, ce qui peut être gênant, par exemple dans des tables. Heureusement, le JDK sait retomber sur ses pattes. Là où ça se complique, c'est lorsqu'on veut créer des hash un poil plus sophistiqués.

2.1 Fonctions de hashage

Pour commencer, Guava propose plusieurs algorithmes standards, dont vous connaissez sans doute déjà le nom :

- MD5 ;
- Murmur 3 (en 32 et 128 bits) ;
- Sha 1 (1, 256 et 512) ;
- Good Fast Hash.

Ce dernier hash part du principe que les développeurs n'ont en réalité que très rarement besoin d'un hash spécifique. Ils utilisent MD5 ou Sha1 parce qu'il faut bien en utiliser un. Tout ce qu'ils veulent, c'est juste un hash rapide, sans avoir besoin de connaître les détails. C'est précisément à cela que sert « goodFastHash() ».

Pour profiter des fonctions de Guava, on doit employer une fonction (MD5, Sha1, etc.) d'une part et un « hasher » d'autre part :

```
1 final HashFunction hf = Hashing.md5();
2 HashCode hc = hf.newHasher()
3     .putLong(id)
4     .putString(tatouage, Charsets.UTF_8)
5     .putObject(chien, chienFunnel)
6     .hash();
```

Les « hashers » savent gérer à peu près tous les types.

En complément, un « funnel » sert à décomposer un type bien spécifique comme « Chien » :

```
1 Funnel<Chien> funnel = new Funnel<Chien>
      () {
2     @Override
3     public void chienFunnel(Chien chien,
          PrimitiveSink into) {
4     into.putInt(chien.id)
5         .putString(chien.getPrenom(),
              Charsets.UTF_8)
6         .putString(chien.getRace(),
              Charsets.UTF_8)
7         .putDouble(chien.getPoids());
8     }
9 };
```

2.2 Filtre probabiliste

Quand on recherche un élément dans une liste, ou dans une collection de manière générale, il n'y a pas trente-six solutions : Java doit parcourir tous les éléments jusqu'à avoir trouvé le bon. Prenons un exemple avec une liste assez grande dans laquelle on cherche le chien Milou :

```
1 final int NB = 100000;
2 final List<Chien> chiens = newArrayList
      ();
3 final Random rand = new Random();
4 for (int i = 0; i < NB; i++) {
5     Chien chien = new Chien();
6     chien.setPrenom("abc" + rand.nextInt
          (999));
7     ...
8     chiens.add(chien);
9 }
```

```
1 final Chien milou = new Dog("Milou");
2 boolean trouve = chiens.contains(milou);
      // -> false
```

De la façon dont est construite la liste (i.e. avec des prénoms préfixés par « abc »), on devine que Java ne va pas trouver Milou. Pour autant, Java ne peut pas le deviner et est obligé de tester les cent mille éléments les uns après les autres. Cela prend du temps. Sur MON ordinateur portable, cela prend précisément quatorze millisecondes. Autant dire que c'est éternité du point de vue du processeur.

Ajoutons maintenant Milou à la liste :

```
1 chiens.add(milou);
2 boolean trouve = chiens.contains(milou);
      // -> true
```

Cette fois, Milou est bien trouvé dans la liste. Mais comme il a été ajouté en fin de liste (l'élément qu'on cherche est bizarrement toujours situé à la fin), la recherche dure autant de temps.

Cet exemple a l'air trivial, mais correspond à un cas qu'on a souvent : simplement vérifier qu'un élément est ou non dans la liste. Or il n'est pas admissible d'attendre 14ms pour le déterminer... Les filtres (caches) probabilistes résolvent en partie ce problème.

Le principe de la méthode « contains() » est qu'elle renvoie « true » ou « false » selon que Milou est ou non dans la liste. Dans le cadre d'un filtre probabiliste, la méthode « mightContain() » parfois renvoie « true » (pour dire que Milou est trouvé) en se trompant. Par contre, quand elle retourne « false » (pour dire que Milou n'est pas trouvé), elle ne se trompe jamais :

```
1 final BloomFilter<Dog> bloom =
      BloomFilter.create(chienFunnel, NB,
      0.01);
2 for (int i = 0; i < NB; i++) {
3     ...
4     bloom.put(chien);
5 }
6
7 boolean trouve = bloom.mightContain(
      milou);
```

L'idée est que le filtre probabiliste est capable de donner une réponse en zéro milliseconde. Ce qu'il faut garder en tête, c'est qu'il ne se trompe jamais quand il répond « false » et que, dans les rares cas où il répond « true », il faudra comprendre que la réponse peut être incorrecte. Il faudra peut-être alors faire une vérification classique. Mais ce n'est pas grave, car, en général, lorsqu'on vérifie qu'un élément est dans la liste, on espère qu'il n'y est pas (par exemple parce qu'on veut l'y ajouter).

En fait, le « bloom filter » est basé sur des probabilités de collision des hash utilisés, puisque ce sont des hash qui sont en réalité stockés et non les éléments (un hash, c'est un « long » et ça ne consomme donc pas autant de mémoire qu'un Chien). La probabilité de collision est spécifiée lors de la création du bloom filter :

```
1  final int NB = 100000;
2  final BloomFilter<Dog> bloom =
       BloomFilter.create(chienFunnel, NB,
       0.01);
```

Dans cet exemple, on indique au bloom filter qu'il attend « 100000 » chiens et qu'il aura le droit de se tromper avec une probabilité de « 0.01 » soit « 1 % » des cas. L'idée est que plus cette probabilité est élevée et plus le filtre est rapide.

À lire, un billet de blog intitulé « Bloom Filter de Guava 13 » : lien 34.

3 I/O

La plupart des programmes doivent traiter les I/O, par exemple pour lire des fichiers CSV (lien 35), des propriétés ou encore des données sérialisées. Guava apporte des solutions optimisées pour tout ça.

Les fonctionnalités d'IO de Guava ont été conçues avant l'arrivée de Java 7. Si vous utilisez une version 7 ou supérieur de Java, je vous conseille de regarder d'abord du côté du JDK (même si Guava est plus complet), par exemple du côté du « try-with-resources ».

3.1 I/O Suppliers

La plupart des méthodes I/O du JDK travaillent avec des « streams ». Dans le cadre de Guava, on va plutôt utiliser les types « InputSupplier » et « OutputSupplier ». Ces deux classes sont plus pratiques que celles de Java, notamment dans la mesure où elles gèrent des problématiques telles que la fermeture des ressources. En effet, le code Java classique pour la copie de fichier par exemple, même en utilisant les utilitaires de Guava, est assez verbeux techniquement :

```
1  final File f1 = new File("in.txt");
2  final File f2 = new File("out.txt");
3
4  FileInputStream fis = null;
5  FileOutputStream fos = null;
6
7  boolean ok = false;
8
9  try {
10     fis = new FileInputStream(f1);
11     fos = new FileOutputStream(f2);
12
13     ByteStreams.copy(fis, fos);
14     ok = true;
15 } catch (FileNotFoundException e) {
16     ...
17 } catch (IOException e) {
18     ...
19 } finally {
20     try {
21         Closeables.close(fos, !ok);
22     } catch (IOException e) {
23         ...
24     }
25     try {
26         Closeables.close(fis, !ok);
27     } catch (IOException e) {
28         ...
29     }
30 }
```

L'utilisation de « Closeables.close() » avec un second paramètre booléen permet d'éviter de polluer encore plus le code avec l'ajout d'une condition sur la nullité...

En utilisant des « suppliers », on s'épargne une bonne partie du code purement technique :

```
1  final File f1 = new File("in.txt");
2  final File f2 = new File("out.txt");
3
4  try {
5      ByteStreams.copy(Files.
           newInputStreamSupplier(f1),
           Files.newOutputStreamSupplier(f2
           ));
6  } catch (IOException e) {
7      ...
8  }
```

Dans les deux cas, on conserve la gestion des « IOExceptions » mais c'est un autre sujet.

3.2 Byte/Char Streams

Comme en « Java standard », on va faire une différence entre les flux binaires (bytes) et les flux de texte. Les premiers nécessiteront l'emploi de « ByteStreams » et les seconds celui de « CharStreams ».

Les méthodes de « ByteStreams » sont :

— toByteArray(InputStream) / toByteArray(InputSupplier) ;
— readFully(InputStream, byte[]) ;
— write(byte[], OutputSupplier) ;
— copy(InputStream, OutputStream) / copy(InputSupplier, OutputSupplier) ;
— length(InputSupplier) ;
— equal(InputSupplier, InputSupplier) ;
— join(InputSupplier...) join(InputSupplier...) ;
— newInputStreamSupplier(byte[]) ;
— readBytes(InputSupplier, ByteProcessor).

Les méthodes de « CharStreams » sont :

— toString(Readable) / toString(InputSupplier) ;
— write(CharSequence, OutputSupplier) ;
— copy(Readable, Appendable) / copy(InputSupplier, OutputSupplier) ;
— newReaderSupplier(String) ;
— readLines(InputSupplier, LineProcessor).

Les noms sont assez clairs pour se passer d'explication. Une précision s'impose néanmoins : les méthodes qui travaillent avec des « streams classiques » ne ferment pas les ressources alors que les méthodes travaillant avec des suppliers les ferment.

3.3 Fichiers

Pour créer des suppliers spécifiques aux fichiers, on va utiliser « Files » qui possède également des méthodes spécialisées pour les fichiers binaires et les fichiers texte :

— newInputStreamSupplier(File) ;
— newOutputStreamSupplier(File, boolean append) ;
— newReaderSupplier(File, Charset) ;
— newWriterSupplier(File, Charset, boolean append).

Quant aux opérations disponibles, elles sont assez simples. Pour un fichier binaire :

— toByteArray(File) ;
— write(byte[], File) ;
— copy(File, File) ;
— copy(File, OutputSupplier) ;
— readBytes(File, ByteProcessor).

Et pour un fichier texte :

— toString(File, Charset) ;
— write(CharSequence, File, Charset) ;
— copy(File, Charset, OutputSupplier) ;
— readLines(File, Charset, LineProcessor).

```java
@Test
public void testRead() {
    // Arrange
    final File file = new File("src/test/resources/prenoms.txt");
    final String expected = "Milou";

    // Act
    String s = null;
    try {
        s = CharStreams.toString(Files.newReaderSupplier(file, Charsets.ISO_8859_1));
    } catch (IOException e) {
        e.printStackTrace();
    }

    // Assert
    Assert.assertTrue(s.startsWith(expected));
}
```

Ou même, directement :

```java
s = Files.toString(file, Charsets.ISO_8859_1);
```

On peut profiter de la lecture pour lancer des traitements/filtres sur les lignes lues :

```java
@Test
public void testLineProcessor() {
    // Arrange
    final File file = new File("src/test/resources/prenoms.txt");
    final int expected = 3;

    // Act
    final LineProcessor<List<String>> lp = new LineProcessor<List<String>>() {
        final ImmutableList.Builder<String> builder = ImmutableList.builder();

        @Override
        public List<String> getResult() {
            return builder.build();
        }

        @Override
        public boolean processLine(String line) throws IOException {
            if (line.contains("o")) {
                builder.add(line);
            }
            return true;
        }
    };
    List<String> lines = null;
    try {
        lines = Files.readLines(file, Charsets.ISO_8859_1, lp);
    } catch (IOException e) {
        e.printStackTrace();
    }

    // Assert
    Assert.assertEquals(expected, lines.size());
}
```

L'utilitaire « Files » propose quelques autres méthodes assez pratiques, dont :

— createParentDirs(File), pour créer la structure de dossier dans laquelle doit se trouver un fichier ;

— getFileExtension(String), qui renvoie l'extension d'un fichier ;

— simplifyPath(String), qui renvoie un chemin nettoyé.

```
1  @Test
2  public void testExtension() {
3      // Arrange
4      final String fileName = "src/test/
           resources/prenoms.txt";
5      final String expected = "txt";
6
7      // Act
8      final String extension = Files.
           getFileExtension(fileName);
9
10     // Assert
11     Assert.assertEquals(expected,
           extension);
12 }
```

4 Conclusion

Les hash et les I/O sont loin d'être des sujets triviaux. Avec Guava, sans devenir facile, c'est bien plus abordable. N'hésitez pas à consulter les autres épisodes de cette série pour découvrir les fonctionnalités fantastiques de la bibliothèque.

Retrouvez l'article de ***Thierry Leriche-Dessirier*** *en ligne : lien 36*

Eclipse

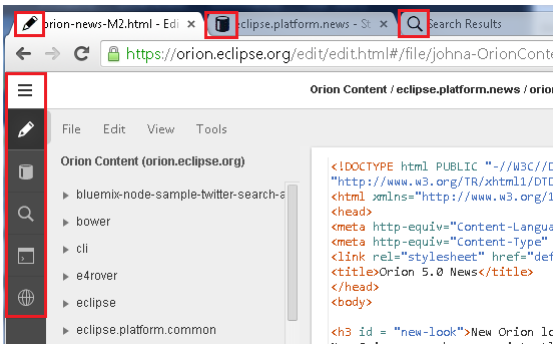
Les dernières news Eclipse

Eclipse Orion 5 sort avec un nouveau look et plusieurs changements

L'EDI Web dans le Cloud permet de déployer directement sur les plates-formes Cloud

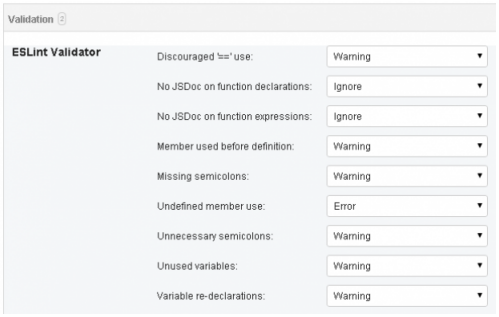
Quatre mois après la sortie d'Orion 4, l'équipe d'Eclipse continue le développement de sa solution de développement web dans le Cloud, avec cette fois-ci, un nouveau look et plusieurs améliorations.

Orion 5 arbore une interface plus conviviale suite à la demande des utilisateurs, en intégrant une barre de menu et un menu contextuel, ce qui facilitera les tâches les plus simples et récurrentes comme le copier-coller ou le renommage d'un fichier.



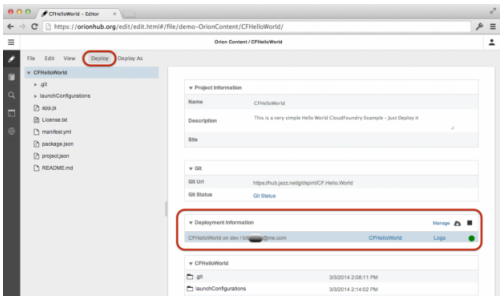
Selon John Arthorne, membre de l'équipe d'Orion, ces choix ont été dictés par les utilisateurs qui sont familiarisés avec les précédents outils de développement qu'ils utilisaient, de ce fait Orion devait répondre à ces attentes.

Outre les changements graphiques, Orion embarque plusieurs nouveautés à commencer par l'assistant de contenu qui a été revu sous trois aspects : rapidité, facilité et meilleure présentation. La rapidité sous-entend rapidité de chargement, d'exécution et de rendu de l'assistant, alors que la facilité est mise en lumière grâce au déclenchement automatique et intelligent de l'assistant par exemple après un « . » en JavaScript ou un tag en HTML, allant plus loin encore avec l'insertion automatique de la suggestion si elle est unique. En ce qui concerne la présentation, cela se traduit par la mise en avant de la suggestion au profit des autres.



Orion 5 ne s'arrête pas en si bon chemin puisqu'il propose le support de l'assistant de contenu pour Node.js ainsi que pour les bases de données suivantes : Redis, MySQL, Postgres, MongoDB. De plus, il inclut le validateur syntaxique JavaScript eslint, un second validateur pour JSON et un support amélioré de Gerrit.

Enfin, l'une des fonctionnalités les plus notables apportées sous cette dernière version est la possibilité de déployer son application directement sur la plateforme d'un provider de type PaaS (Plateform as a Service) en se basant sur la version 6 de l'API Cloud Foundry.



Télécharger Orion 5 : lien 37

*Commentez la news d'**Arslan Hamza** en ligne : lien 38*

NetBeans

Les dernières news NetBeans

NetBeans s'aligne sur Java 8

La version 8.0 de l'EDI open source améliore ses outils pour supporter les expressions Lambdas, Streams et Profiles

NetBeans passe de la version 7.4 à la version 8 pour s'aligner avec Java 8. À la suite de la publication de la plateforme de développement, Oracle a sorti une nouvelle version de son environnement de développement open source.

Cette nouvelle mouture apporte un nombre important de nouvelles fonctionnalités pour les développeurs Java, C/C++, HTML5, JavaScript et PHP.

NetBeans 8 se présente comme l'outil idéal pour découvrir les nouveautés de Java 8, grâce aux améliorations qui ont été apportées à l'outil et à l'éditeur de code pour travailler convenablement avec les expressions Lambdas, Profiles et Streams. La prise en charge de Java ME Embedded 8 est également au rendez-vous.

L'EDI supporte Java SE Embedded, permettant aux développeurs de créer, déployer, exécuter, déboguer et profiler les applications Java SE sur des dispositifs embarqués, à l'instar de Raspberry Pi. Plusieurs améliorations ont été apportées à l'éditeur de code Java. On va noter également une meilleure intégration avec JavaFX Scene Builder.

Les développeurs Java EE apprécieront l'intégration de nouveaux générateurs de code pour PrimeFaces, permettant de générer des modèles d'applications complètes PrimeFaces avec opérations CRUD (Create, Read, Udapte et Delete) et connexion aux bases de données, la prise en charge de Tomcat 8.0, la complétion de code pour des composants JSF, les classes alternatives et les stéréotypes.

En ce qui concerne JavaScript, NetBeans 8.0 améliore le support du framework AngularJS, essentiellement au niveau de la complétion de code. L'éditeur de code prend en charge la création de widgets et plugins jQuery. À ces nouveautés, s'ajoute la prise en charge du débogage du code JavaScript exécuté dans le moteur JavaScript Nashorn, embarqué avec le JDK 8.

Du côté de HTML5, le débogage pour Android 4.4 WebKit est au rendez-vous pour Cordova (PhoneGap), ainsi que la prise en charge de Karma pour les tests et l'intégration de Avatar.js dans le gestionnaire de plugins. Après son installation, les développeurs pourront créer des projets Avatar.js dans NetBeans.

NetBeans 8.0 s'enrichit du support de PHP 5.5, PHP CS Fixer et des améliorations pour Twig, Latte et Neon.

Les développeurs C/C++ n'ont pas été oubliés. L'outil a été amélioré pour offrir de meilleures performances à ceux-ci.

Télécharger NetBeans 8.0 : lien 39

*Commentez la news d'**Hinault Romaric** en ligne : lien 40*

Android

Les dernières news Android

Android Wear

Google se lance à l'assaut du marché des montres connectées. Une preview du SDK est déjà disponible

Dans un billet blog, Sundar Pichai, le responsable d'Android et des produits Chrome chez Google, indique la commercialisation avant la fin de l'année de montres connectées. L'éditeur travaillerait de concert avec plusieurs fabricants d'électronique, dont Samsung, Asus, HTC et LG, ainsi que « des marques de mode comme le groupe Fossil », groupe américain spécialisé dans les vêtements et les accessoires.

Ses montres connectées fonctionneront sous Android Wear, une version spéciale d'Android qui propose une interface repensée pour les petits écrans. Au cœur du système d'exploitation figure Google Now, l'assistant personnel intelligent déjà disponible sur les versions mobiles d'Android.

La montre affichera les informations les plus pertinentes à l'écran en fonction de certains paramètres comme le moment de la journée ou la localisation mais également d'autres informations comme les SMS reçus ou même les notifications réseaux. Pour activer l'assistant vocal, il suffira juste de dire « Ok Google ». Dès lors l'utilisateur pourra effectuer une recherche sur internet, envoyer des messages, commander son smartphone à distance, ouvrir la porte du garage. Regardez donc la démo.

Cliquez pour visualiser la vidéo.

Comme l'entreprise l'avait annoncé à la SXSW le 09 mars dernier, le tout premier kit de développement Android Wear est d'ores et déjà disponible pour les développeurs. Il permet notamment d'adapter les notifications en provenance de vos applications déjà existantes à Android Wear. Les notifications sont interactives. De plus, il est possible de dicter sa réponse à un message. Les notifications peuvent comporter plusieurs pages, qui se consultent en glissant son doigt sur l'écran tactile, ou peuvent être empilées. Comme autres fonctions de bases figurent la synchronisation avec un téléphone ainsi que l'assistant personnel Google Now. Bien entendu il s'agit d'une Preview.

Cliquez pour visualiser la vidéo.

Télécharger la Preview du kit de développement Android Wear lien 41
Source : Le blog Androïd. lien 42.

Fragmentation Android :

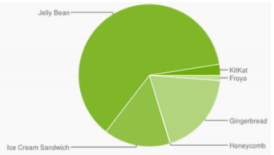
Kitkat continue lentement sa progression, Jelly Bean domine toujours mais s'essouffle

Google a présenté les chiffres de son Dashboard Android, qui s'intéresse aux différentes données concernant les terminaux qui se connectent à sa vitrine Google Play. Avec 62% de parts de marché, Jelly Bean, qui regroupe les versions 4.1.x, 4.2.x et 4.3 domine les autres versions. D'ailleurs il bénéficie de la plus forte progression mensuelle en gagnant ses 1,3 points de pourcentage.

Dans le détail, Android 4.1, la première version de Jelly Bean, subit une première perte de 0,2 point mais reste tout de même majoritaire. Seules les versions 4.2 (17,1%) et 4.3 (9,6%) ont assuré la croissance avec respectivement 0,8 point et 0,7 point de pourcentage.

Pour l'heure, Kitkat n'occupe que 2,5% des terminaux mobiles bien que cela représente une hausse de 0,7% depuis début février. En prenant en considération l'arrivée de plus de smartphones sous l'OS mobile, ce résultat est somme toute assez logique. La version prend donc lentement ses marques.

Version	Codename	API	Distribution
2.2	Froyo	8	1.2%
2.3.3 - 2.3.7	Gingerbread	10	19.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	15.2%
4.1.x	Jelly Bean	16	35.3%
4.2.x		17	17.1%
4.3		18	9.6%
4.4	KitKat	19	2.5%



Data collected during a 7-day period ending on March 3, 2014.
Any versions with less than 0.1% distribution are not shown.

Ice Cream Sandwich s'essouffle progressivement. En février les versions 4.0.3 et 4.0.4 représentaient 16,1% d'utilisation. Elles ne comptent plus que pour 15,2%, soit une perte de 0,9 point. Gingerbread perd également un point mais avec 19% la version demeure relativement utilisée. Honeycomb (3.2) et Froyo (2.2) semblent ne pas être disposés à partir même si à eux deux ils constituent à peine 1,3% des versions utilisées.

*Commentez la news de **Stéphane Le Calme** en ligne : lien 43*

Les derniers tutoriels et articles

vogella : Utiliser les fragments sous Android

Cet article décrit comment utiliser la classe Fragment dans une application Android pour créer des mises en page avec plusieurs panneaux, par exemple des applications qui se dimensionnent automatiquement par rapport au périphérique. Cet article se base sur Eclipse 4.3 (Kepler), Java 1.6 et Android 4.3.
Nous remercions Lars Vogel qui nous a aimablement autorisés à traduire et héberger cet article.

1 Les bases d'Android

La suite de cet article suppose que vous avez déjà une connaissance basique du développement sous Android.

Lisez le tutoriel du développement Android pour plus de connaissances à ce sujet : lien 44.

2 Les fragments

2.1 Qu'est ce qu'un fragment ?

Un fragment est un composant indépendant qui peut être utilisé dans une activité. Un fragment encapsule des fonctionnalités qui le rendent facile à réutiliser dans une activité ou dans une mise en page.

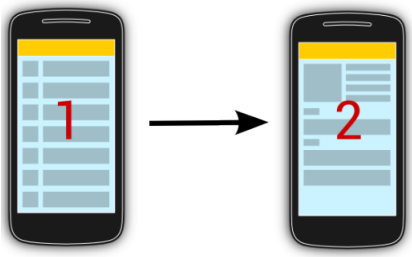
Un fragment s'exécute dans le contexte d'une activité mais possède son propre cycle de vie et typiquement, il possède une interface utilisateur. Il est aussi possible de définir des fragments sans interface utilisateur, par exemple des fragments sans entête.

Les fragments peuvent être ajoutés à une activité de manière statique ou dynamique.

2.2 Avantages de l'utilisation des fragments

Les fragments permettent une réutilisation plus facile dans différentes mises en page. Par exemple, vous pouvez construire des mises en page avec un seul panneau pour des téléphones et plusieurs panneaux pour des tablettes. Ce n'est pas limité aux tablettes, vous pouvez aussi supporter les différentes mises en page suivant l'orientation (portrait ou paysage) du téléphone.

L'exemple typique est une liste dans une activité. Sur une tablette vous pouvez voir les détails immédiatement sur la partie droite si vous cliquez sur un élément. Sur un téléphone, vous affichez une nouvelle fenêtre avec les détails. Ceci est représenté par le schéma suivant :



La discussion suivante suppose que vous avez deux fragments (principal et détail) mais vous pouvez aussi en avoir plus. Nous aurons aussi une activité principale et une activité détaillée. Sur une tablette, l'activité principale contient les deux fragments dans sa mise en page, sur un téléphone, elle ne contient que l'activité principale.

Le schéma suivant montre cette utilisation :

2.3 Comment utiliser les fragments ?

Pour créer différentes mises en page avec des fragments, vous pouvez :

— utiliser une activité qui affiche deux fragments sur les tablettes et un seul sur les téléphones. Dans ce cas, vous basculerez les fragments dans l'activité en cas de besoin. Cela requiert que le fragment ne soit pas déclaré dans le fichier de mise en page de manière à ce qu'il

puisse être supprimé de façon dynamique. Cela requiert aussi des modifications dans la barre d'actions si le statut de la barre d'actions dépend du fragment affiché.

— utiliser des activités séparées pour héberger chacun des fragments sur un téléphone. Par exemple quand l'interface utilisateur d'une tablette utilise deux fragments dans une activité, utilisez la même activité pour un téléphone mais en fournissant une mise en page alternative qui ne comprend qu'un seul fragment. Quand vous devez changer de fragment, vous démarrez une autre activité qui héberge l'autre fragment.

La deuxième approche est la plus flexible et en général la méthode privilégiée pour utiliser des fragments. Dans ce cas, l'activité principale vérifie que le fragment détaillé est disponible dans la mise en page. S'il est présent, l'activité principale l'informe de se mettre à jour. Si le fragment détaillé n'est pas présent, l'activité principale démarre l'activité détaillée.

3 Définir et utiliser des fragments

3.1 Définir des fragments

Pour définir un nouveau fragment, vous surchargez soit la classe android.app.Fragment, soit une de ses sous-classes, par exemple ListFragment, DialogFragment, PreferenceFragment or WebViewFragment. Le code suivant vous montre un exemple d'implémentation :

```
1  package com.example.android.rssfeed;
2
3  import android.app.Fragment;
4  import android.os.Bundle;
5  import android.view.LayoutInflater;
6  import android.view.View;
7  import android.view.ViewGroup;
8  import android.widget.TextView;
9
10 public class DetailFragment extends
       Fragment {
11
12   @Override
13   public View onCreateView(
         LayoutInflater inflater, ViewGroup
          container,
14       Bundle savedInstanceState) {
15     View view = inflater.inflate(R.
           layout.fragment_rssitem_detail,
16         container, false);
17     return view;
18   }
19
20   public void setText(String item) {
21     TextView view = (TextView) getView()
           .findViewById(R.id.detailsText);
22     view.setText(item);
23   }
24 }
```

3.2 Ajouter un fragment de manière statique

Pour utiliser votre nouveau fragment, vous pouvez l'ajouter de manière statique au fichier de mise en page XML.

Pour vérifier si le fragment est présent dans votre mise en page, vous pouvez utiliser la classe FragmentManager.

```
1  DetailFragment fragment = (
       DetailFragment) getFragmentManager()
       .
2    findFragmentById(R.id.detail_frag);
3  if (fragment==null || ! fragment.
       isInLayout()) {
4    // start new Activity
5    }
6  else {
7    fragment.update(...);
8  }
```

Si un fragment est défini dans le fichier XML, l'attribut android :name indique le nom de la classe correspondante.

3.3 Cycle de vie d'un fragment

Un fragment possède son propre cycle de vie mais il est toujours connecté au cycle de vie de l'activité qui l'utilise.

La méthode onCreate() est appelée après l'appel à la méthode onCreate() de l'activité mais avant la méthode onCreateView() du fragment.

La méthode onCreateView() est appelée par Android dès que le fragment doit créer son interface utilisateur. Ici, vous pouvez afficher la mise en page avec la méthode inflate() ou l'objet Inflator passé en paramètre à cette méthode. Il n'y a aucun besoin d'implémenter cette méthode pour les fragments sans en-tête.

La méthode onActivityCreated() est appelée après la méthode onCreateView() quand l'activité est créée. Ici vous pouvez instancier les objets qui nécessitent un objet Context.

Les fragments ne sous classent pas le Context. Vous devez utiliser la méthode getActivity() pour obtenir l'activité parente.

La méthode onStart() est appelée une fois que le fragment devient visible.

Si une activité s'arrête, ses fragments sont aussi arrêtés. Si une activité est détruite, ses fragments sont aussi détruits.

3.4 Communication entre l'application et les fragments

Pour améliorer la ré-utilisabilité, les fragments ne devraient pas communiquer directement entre eux. Toutes les communications avec les fragments devraient se faire via l'activité qui les héberge.

Pour cela, un fragment devrait définir une interface comme un type interne pour imposer à l'activité qui veut l'utiliser d'implémenter cette interface. Ceci permet d'éviter que le fragment connaisse l'activité qui l'utilise. Dans la méthode onAttach(), le fragment peut vérifier que l'activité implémente cette interface.

Par exemple, si un fragment doit communiquer une valeur à son activité, cela peut être fait de cette façon :

```
1  package com.example.android.rssfeed;
2
3  import android.app.Activity;
4  import android.app.Fragment;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.View;
8  import android.view.ViewGroup;
9  import android.widget.Button;
10
11 public class MyListFragment extends
       Fragment {
12
13   private OnItemSelectedListener
       listener;
14
15   @Override
16   public View onCreateView(
       LayoutInflater inflater, ViewGroup
        container,
17       Bundle savedInstanceState) {
18     View view = inflater.inflate(R.
         layout.fragment_rsslist_overview
         ,
19         container, false);
20     Button button = (Button) view.
         findViewById(R.id.button1);
21     button.setOnClickListener(new View.
         OnClickListener() {
22       @Override
23       public void onClick(View v) {
24         updateDetail();
25       }
26     });
27     return view;
28   }
29
30   public interface
       OnItemSelectedListener {
31     public void onRssItemSelected(String
         link);
32   }
33
34   @Override
35   public void onAttach(Activity activity
       ) {
36     super.onAttach(activity);
37     if (activity instanceof
         OnItemSelectedListener) {
38       listener = (OnItemSelectedListener
           ) activity;
39     } else {
40       throw new ClassCastException(
           activity.toString()
41           + " must implemenet
               MyListFragment.
               OnItemSelectedListener");
42     }
43   }
44
45   @Override
46   public void onDetach() {
47     super.onDetach();
48     listener = null;
49   }
50
51   // May also be triggered from the
       Activity
52   public void updateDetail() {
53     // create a string, just for testing
54     String newTime = String.valueOf(
         System.currentTimeMillis());
55
56     // Inform the Activity about the
         change based
57     // interface defintion
58     listener.onRssItemSelected(newTime);
59   }
60 }
```

4 Données persistantes dans les fragments

4.1 Données persistantes entre deux redémarrages de l'application

Avec les fragments, vous pouvez aussi avoir besoin de stocker des données applicatives. Pour cela, vous pouvez stocker ces données de manière centralisée. Par exemple :

- dans une base de données Sqlite ;
- dans un fichier ;
- dans l'objet application. Dans ce cas, l'application doit supporter ce type de stockage.

4.2 Données persistantes entre deux changements de configuration

Si vous devez stocker des données de configuration, vous pouvez aussi utiliser l'objet application.

En plus, vous pouvez utiliser la méthode setRetainState(true) sur le fragment. Cela sauvegarde l'instance du fragment entre deux changements de configuration mais cela marche uniquement si le fragment n'est pas ajouté à la pile des tâches de fond (backstack). Cette méthode n'est pas recommandée par Google pour les fragments qui possèdent une interface utilisateur. Dans ce cas, les données doivent être stockées comme des membres (champs).

Si les données qui doivent être sauvegardées sont supportées par la classe Bundle, vous pouvez utiliser la méthode onSaveInstanceState() pour mettre ces données dans le Bundle et les retrouver dans la méthode onActivityCreated().

5 Modifier les fragments dynamiquement

Les classes FragmentManager et FragmentTransaction vous permettent d'ajouter, de supprimer et de remplacer des fragments dans la mise en page de votre activité.

Les fragments peuvent être modifiés dynamiquement avec des transactions. Pour ajouter dynamiquement des transactions à une mise en page déjà existante, vous définissez un container dans le fichier XML dans lequel vous pouvez ajouter le fragment. Pour cela, vous pouvez utiliser l'élément FrameLayout.

```
1 FragmentTransaction ft =
      getFragmentManager().
      beginTransaction();
2 ft.replace(R.id.your_placehodler, new
      YourFragment());
3 ft.commit();
```

Un nouveau fragment va remplacer un fragment existant qui a été préalablement ajouté au container.

Si vous voulez ajouter une transition sur la pile des tâches de fond Android (backstack) vous devez utiliser la méthode addToBackStack(). Ceci rajoute l'action dans l'historique de la pile de l'activité et vous pourrez annuler les changements effectués sur le fragment avec le bouton « Back ».

6 Animation des transitions de fragments

Durant une transition de fragment, vous pouvez définir des animations basées sur l'API « Property Animation » avec la méthode setCustomAnimations().

Vous pouvez aussi utiliser les animations standard fournies par Android avec la méthode setTransition(). Elles sont définies avec les constantes commençant par FragmentTransaction.TRANSIT_FRAGMENT_*.

Ces deux méthodes permettent de définir une animation en début et en fin.

7 Ajouter des transitions de fragments sur la pile des tâches de fond (backstack)

Vous pouvez ajouter une FragmentTransition à la pile des tâches de fond (backstack) pour permettre à l'utilisateur d'utiliser le bouton « back » pour annuler la transition.

Pour cela, il faut utiliser la méthode addToBackStack() sur l'objet FragmentTransition.

8 Fragments pour les traitements en tâche de fond

8.1 Fragments sans entête

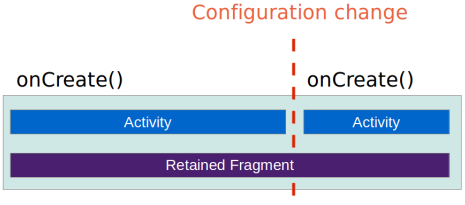
Les fragments peuvent être utilisés sans définir d'interface utilisateur.

Pour créer un fragment sans en-tête, il faut tout simplement retourner NULL dans la méthode onCreateView() de votre fragment.

Il est recommandé d'utiliser des fragments sans entête pour le traitement en tâche de fond en combinaison avec la méthode setRetainInstance(). De cette manière, vous n'avez pas à gérer vous-même les changements durant le traitement asynchrone.

8.2 Mémoriser les fragments sans en-tête pour gérer les changements de configuration

Les fragments sans en-tête sont typiquement utilisés pour mémoriser un état avant des changements de configuration ou alors pour des traitements en tâche de fond. Pour cela, vous devez mémoriser votre fragment sans entête. Un fragment mémorisé n'est pas détruit lors des changements de configuration.



Pour mémoriser votre fragment, il faut appeler sa méthode setRetainInstance().

Pour ajouter un tel fragment à votre activité, il faut utiliser la méthode add de la classe FragmentManager. Si vous devez vous référer à ce segment plus tard, vous devez l'ajouter avec un tag de façon à pouvoir le retrouver ultérieurement avec la méthode findFragmentByTag() de la classe FragmentManager.

L'usage de onRetainNonConfigurationInstance() est obsolète et devrait être remplacé par un fragment sans en-tête.

9 Contribuer à la barre d'actions

Les fragments peuvent aussi influer sur la barre d'actions. Pour cela, appelez setHasOptionsMenu() dans la méthode onCreate() du fragment. Dans ce cas, le système Android appelle la méthode onCreateOptionsMenu() du fragment et ajoute les options du menu à celles déjà ajoutées par l'activité.

10 Exercice avec les fragments

10.1 Présentation

Le tutoriel suivant montre comment utiliser les fragments. L'application utilise une mise en page différente avec des fragments dépendant du mode portrait ou paysage.

Dans le mode portrait, l'activité RssfeedActivity montre un seul fragment. Depuis ce fragment, l'utilisateur peut naviguer vers une autre activité qui contient un autre fragment.

Dans le mode paysage, l'activité RssfeedActivity montre les deux fragments côte à côte.

10.2 Création du projet

Créez un nouveau projet Android avec les données suivantes :

Propriété	Valeur
Nom de l'application	RSS Reader
Nom du projet	com.example.android.rssfeed
Nom du paquetage	com.example.android.rssfeed
Modèle	BlankActivity
Activité	RssfeedActivity
Mise en page	activity_rssfeed

Table 1. Android project

10.3 Création des mises en page

Créez ou modifiez le fichier de mise en page dans le répertoire res/layout/.

Créez un nouveau fichier de mise en page nommé fragment_rssitem_detail.xml. Ce fichier de mise en page sera utilisé par le fragment DetailFragment.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://
       schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical" >
6
7      <TextView
8          android:id="@+id/detailsText"
9          android:layout_width="
               wrap_content"
10         android:layout_height="
               match_parent"
11         android:layout_gravity="
               center_horizontal|
               center_vertical"
12         android:layout_marginTop="20dip"
13         android:text="Default Text"
14         android:textAppearance="?android
               :attr/textAppearanceLarge"
15         android:textSize="30dip" />
16
17 </LinearLayout>
```

Créez un nouveau fichier de mise en page nommé fragment_rsslist_overview.xml. Ce fichier de mise en page sera utilisé par le fragment MyListFragment.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://
       schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical" >
6
7      <Button
8          android:id="@+id/button1"
9          android:layout_width="
               wrap_content"
10         android:layout_height="
               wrap_content"
11         android:text="Press to update"
12          />
13
14 </LinearLayout>
```

Modifiez le fichier activity_rssfeed.xml existant. Ce fichier est la mise en page par défaut pour l'activité RssfeedActivity et elle affiche les deux fragments.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://
       schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:orientation="horizontal" >
6
7      <fragment
8          android:id="@+id/listFragment"
9          android:layout_width="0dp"
10         android:layout_weight="1"
```

```
        android:layout_height="
            match_parent"
        android:layout_marginTop="?
            android:attr/actionBarSize"
        class="com.example.android.
            rssfeed.MyListFragment" ></
            fragment>

    <fragment
        android:id="@+id/detailFragment"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="
            match_parent"
        class="com.example.android.
            rssfeed.DetailFragment" >
        <!-- Preview: layout=@layout/
            details -->
    </fragment>

</LinearLayout>
```

10.4 Création des classes Fragment

Maintenant, il faut créer les classes Fragment. Commençons par la classe DetailFragment.

```
package com.example.android.rssfeed;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailFragment extends
    Fragment {

  @Override
  public View onCreateView(
      LayoutInflater inflater, ViewGroup
       container,
      Bundle savedInstanceState) {
    View view = inflater.inflate(R.
        layout.fragment_rssitem_detail,
        container, false);
    return view;
  }

  public void setText(String item) {
    TextView view = (TextView) getView()
        .findViewById(R.id.detailsText);
    view.setText(item);
  }
}
```

Créons la classe MyListFragment. Malgré son nom, elle ne va pas afficher une liste, elle aura juste un bouton permettant d'envoyer la date courante au fragment détaillé.

```
package com.example.android.rssfeed;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

public class MyListFragment extends
    Fragment {

  private OnItemSelectedListener
      listener;

  @Override
  public View onCreateView(
      LayoutInflater inflater, ViewGroup
       container,
      Bundle savedInstanceState) {
    View view = inflater.inflate(R.
        layout.fragment_rsslist_overview
        ,
        container, false);
    Button button = (Button) view.
        findViewById(R.id.button1);
    button.setOnClickListener(new View.
        OnClickListener() {
      @Override
      public void onClick(View v) {
        updateDetail();
      }
    });
    return view;
  }

  public interface
      OnItemSelectedListener {
      public void onRssItemSelected(
          String link);
    }

  @Override
    public void onAttach(Activity
        activity) {
      super.onAttach(activity);
      if (activity instanceof
          OnItemSelectedListener) {
        listener = (
            OnItemSelectedListener)
            activity;
      } else {
        throw new ClassCastException(
            activity.toString()
            + " must implemenet
                MyListFragment.
                OnItemSelectedListener")
                ;
      }
    }


  // May also be triggered from the
      Activity
  public void updateDetail() {
    // create fake data
    String newTime = String.valueOf(
        System.currentTimeMillis());
    // Send data to Activity
    listener.onRssItemSelected(newTime);
  }
}
```

10.5 L'activité RssfeedActivity

Modifiez la classe RssfeedActivity avec le code suivant :

```
package com.example.android.rssfeed;

import android.os.Bundle;
import android.app.Activity;
```

```
5  import android.view.Menu;
6
7  public class RssfeedActivity extends
       Activity implements MyListFragment.
       OnItemSelectedListener{
8
9      @Override
10     protected void onCreate(Bundle
           savedInstanceState) {
11         super.onCreate(
               savedInstanceState);
12         setContentView(R.layout.
               activity_rssfeed);
13     }
14
15     // if the wizard generated an
           onCreateOptionsMenu you can
           delete
16     // it, not needed for this tutorial
17
18   @Override
19   public void onRssItemSelected(String
         link) {
20       DetailFragment fragment = (
             DetailFragment)
             getFragmentManager()
21               .findFragmentById(R.id.
                     detailFragment);
22           if (fragment != null && fragment
                 .isInLayout()) {
23             fragment.setText(link);
24           }
25   }
26
27 }
```

10.6 Exécution

Exécutez votre exemple. Les deux fragments devraient être affichés en mode portrait et paysage. Si vous appuyez sur le bouton dans le fragment ListFragment, le fragment DetailFragment devrait être mis à jour.

11 Exercice avec les fragments, utilisation du mode portrait

11.1 Création des mises en page pour le mode portrait

L'activité RssfeedActivity doit utiliser un fichier de mise en page particulier en mode portrait. En mode portrait, Android va vérifier le répertoire layout-port pour les fichiers correspondants. Si Android ne trouve pas de fichier correspondant, il utilise le répertoire layout.

Pour cela, il faut créer un répertoire res/layout-port. Après cela, créez le fichier de mise en page activity_rssfeed.xml dans le répertoire res/layout-port.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://
       schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="horizontal" >
6
7      <fragment
8          android:id="@+id/listFragment"
9          android:layout_width="
               match_parent"
10         android:layout_height="
               match_parent"
11         android:layout_marginTop="?
               android:attr/actionBarSize"
12         class="com.example.android.
               rssfeed.MyListFragment" />
13 </LinearLayout>
```

Créez aussi le fichier de mise en page activity_detail.xml. Cette mise en page sera utilisée par l'activité DetailActivity. Notez que l'on aurait aussi pu le créer dans le répertoire res/layout mais il n'est utilisé qu'en mode portrait c'est pourquoi nous le plaçons dans ce répertoire.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://
       schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical" >
6
7      <fragment
8          android:id="@+id/detailFragment"
9          android:layout_width="
               match_parent"
10         android:layout_height="
               match_parent"
11         class="com.example.android.
               rssfeed.DetailFragment" />
12
13 </LinearLayout>
```

11.2 L'activité DetailActivity

Créez une nouvelle activité nommée DetailActivity avec le code suivant :

```
1  package com.example.android.rssfeed;
2
3  import android.app.Activity;
4  import android.content.res.Configuration
       ;
5  import android.os.Bundle;
6  import android.widget.TextView;
7
8  public class DetailActivity extends
       Activity {
9
10   public static final String EXTRA_URL =
         "url";
11
12   @Override
13   protected void onCreate(Bundle
         savedInstanceState) {
14     super.onCreate(savedInstanceState);
15
16     // Need to check if Activity has
           been switched to landscape mode
17     // If yes, finished and go back to
           the start Activity
18     if (getResources().getConfiguration
           ().orientation == Configuration.
           ORIENTATION_LANDSCAPE) {
```

```
19        finish();
20        return;
21      }
22      setContentView(R.layout.
            activity_detail);
23      Bundle extras = getIntent().
            getExtras();
24      if (extras != null) {
25        String s = extras.getString(
              EXTRA_URL);
26        TextView view = (TextView)
              findViewById(R.id.detailsText)
              ;
27        view.setText(s);
28      }
29    }
30  }
```

Assurez-vous aussi d'enregistrer cette activité dans le fichier AndroidManifest.xml.

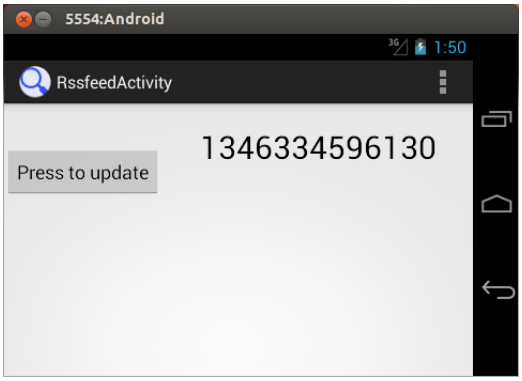
11.3 Modification de l'activité RssfeedActivity

Modifiez l'activité RssfeedActivity pour afficher l'activité DetailActivity au cas où l'autre fragment n'est pas présent dans la mise en page.

```
1   package com.example.android.rssfeed;
2
3   import android.app.Activity;
4   import android.content.Intent;
5   import android.os.Bundle;
6   import android.view.Menu;
7
8   public class RssfeedActivity extends
        Activity implements
9       MyListFragment.
            OnItemSelectedListener {
10
11    @Override
12    protected void onCreate(Bundle
          savedInstanceState) {
13      super.onCreate(savedInstanceState);
14      setContentView(R.layout.
            activity_rssfeed);
15    }
16
17    @Override
18    public void onRssItemSelected(String
          link) {
19      DetailFragment fragment = (
            DetailFragment)
            getFragmentManager()
20          .findFragmentById(R.id.
                detailFragment);
21      if (fragment != null && fragment.
            isInLayout()) {
22        fragment.setText(link);
23      } else {
24        Intent intent = new Intent(
              getApplicationContext(),
25            DetailActivity.class);
26        intent.putExtra(DetailActivity.
              EXTRA_URL, link);
27        startActivity(intent);
28
29      }
30    }
31
32  }
```

11.4 Exécution

Lancez votre exemple. Si vous lancez votre application en mode portrait, vous ne devriez voir qu'un seul fragment. Utilisez le raccourci Ctrl+F11 pour changer l'orientation. En mode paysage, vous devriez voir les deux fragments. Si vous appuyez sur le bouton en mode portrait, une nouvelle activité DetailActivity est démarrée et montre la date courante. En mode paysage, vous voyez les deux fragments.



12 Exercice avec les fragments dynamiques

12.1 But

Modifiez l'exemple RssReader de manière à ce que des fragments dynamiques soient utilisés.

Une mise en page à plusieurs panneaux doit être utilisée dans le cas des écrans larges, par exemple pour les tablettes. Le point de basculement doit être de 600dpi.

Implémentez la solution avec une activité et plusieurs fragments, supprimez l'activité DetailsActivity à la fin de cet exercice.

12.2 Utilisation d'un espace réservé pour les fragments

Remplacez le fragment statique dans le répertoire avec un FrameLayout comme espace réservé.

12.3 Utilisation du gestionnaire de fragments pour remplacer un fragment

Implémentez une solution de manière à ce que vous n'ayez qu'une seule activité et remplacez les fragments à la demande.

Retrouvez l'article de ***Lars Vogel*** *traduit par* ***ram-0000*** *en ligne : lien 45*

Accélération de l'émulateur Android sur l'architecture Intel®

Si vous êtes un développeur Android mécontent des performances de l'émulateur Android, ce document est pour vous. Nous avons fréquemment entendu de nombreux développeurs Android se plaindre de la lenteur et du manque de convivialité de l'émulateur, mais ça ne devrait pas être le cas...

1 Introduction

Ce document va vous guider dans l'installation d'Intel® HAXM (Intel® Hardware Accelerated Execution Manager), un moteur de virtualisation (hyperviseur) assisté par le matériel qui utilise la technologie de virtualisation Intel® pour accélérer le développement Android sous Windows. Il explique également comment configurer une KVM assistée par le matériel sous Linux et les méthodes les mieux connues de compilation de code natif et de soumission d'applications à Google Play Store pour x86.

2 Installation

2.1 Conditions préliminaires

- Le SDK Android doit être installé.
- Votre ordinateur doit posséder un processeur Intel prenant en charge la technologie de virtualisation Intel, EM64T et le bit XD (Execute Disable), activés dans le BIOS.

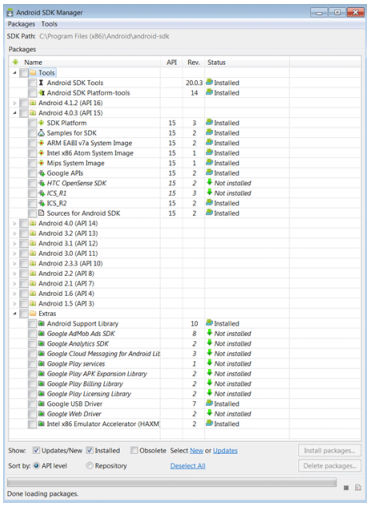
2.2 Installation sur Windows

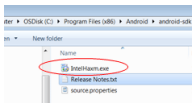
Après avoir installé le SDK Android, ouvrez le gestionnaire du SDK. Intel HAXM se trouve dans la section des extras.

Cochez la case et cliquez sur le bouton « Install packages... » (Installer les packages). Après avoir installé le package, l'état s'affiche comme « Installed » (Installé), ce qui est trompeur car ce n'est pas le cas. Le SDK copie uniquement le fichier exécutable Intel HAXM sur l'ordinateur et c'est à vous d'installer.

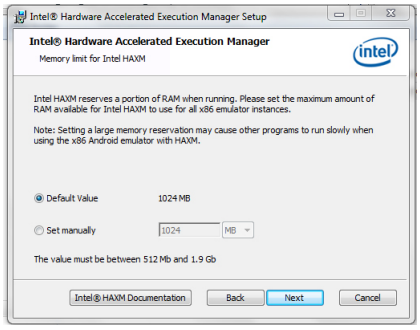
Pour installer le fichier exécutable Intel HAXM, recherchez le fichier IntelHaxm.exe (ou IntelHAXM.dmg sur Mac OS X) sur votre disque dur. Si vous avez conservé les valeurs par défaut, il devrait se trouver sous C : \Program Files \Android \android-sdk \extras \Intel \Hardware_Accelerated_Execution_Manager \IntelHaxm.exe.

Intel HAXM fonctionne uniquement en association avec une des images système x86 du processeur Intel® Atom™, disponibles pour Android 2.3.3 (API 10), 4.0.3 (API 15), 4.1.2 (API 16), 4.2.2 (API 17). Ces images système Intel s'installent exactement comme les images ARM, par le gestionnaire du SDK.

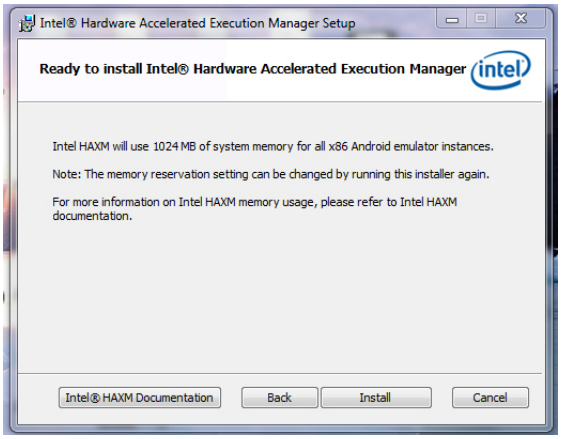




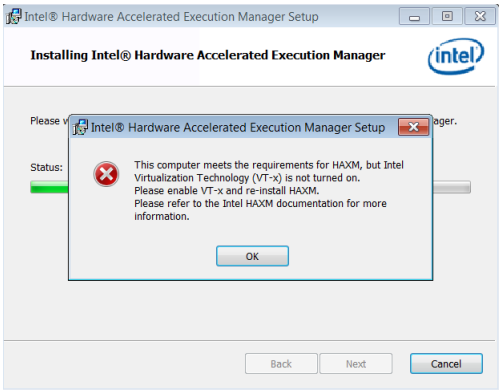
Lorsque vous cliquez sur le fichier exécutable IntelHaxm, un écran d'accueil s'affiche comme suit :



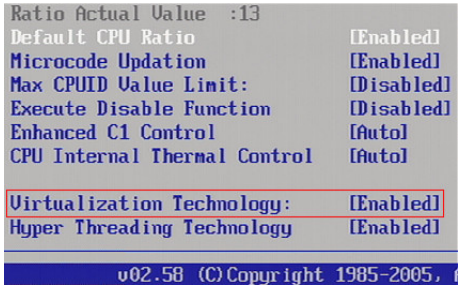
Vous pouvez spécifier la quantité de mémoire allouée à Intel HAXM. Après cela, cliquez sur Next (Suivant). L'écran suivant confirme l'allocation de mémoire. Si la configuration vous convient, cliquez sur Install (Installer).



Pour pouvoir installer Intel HAXM, la technologie de virtualisation Intel doit être activée dans le BIOS, autrement, vous obtenez une erreur comme celle-ci pendant l'installation :



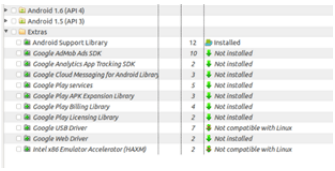
Si vous obtenez cette erreur, accédez au BIOS et activez cette fonctionnalité :



La deuxième option de téléchargement d'Intel HAXM et de l'image système de l'émulateur x86 consiste à accéder directement au site Web : http ://software.intel.com/fr-fr/android et à télécharger les composants nécessaires.

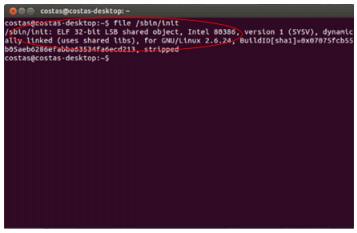
2.3 Installation sur Linux

Les étapes permettant d'accélérer l'émulateur Android pour Linux sont différentes de celles de Windows et de Mac OS X car Intel HAXM n'est pas compatible avec Linux et vous devez donc utiliser une KVM (kernel-based virtual machine) à la place. Les étapes ci-dessous ont été réalisées à l'aide d'Ubuntu 12.04 et peuvent être légèrement différentes avec d'autres distributions Linux.



Comme sous Windows (et Mac OS X), vous devez tout d'abord télécharger le SDK Android depuis le site des développeurs Android. Vous trouverez une archive ADT (Android Developer Tool) qui contient l'IDE Eclipse et le SDK Android. Téléchargez le fichier zip et décompressez-le sur votre ordinateur Linux. Veillez à bien télécharger la version correspondant à votre distribution Linux, 32 bits ou 64 bits. Vous pouvez le faire facilement à l'aide de la commande :

file /sbin/init



Avant d'installer les packages nécessaires à KVM, nous vous recommandons de vérifier que vous disposez du dernier référentiel, ce que vous pouvez faire en tapant :

sudo apt-get update

2.3.a Installation de KVM

Pour installer et exécuter KVM, qui est une solution de virtualisation complète pour Linux sur du matériel x86 (à savoir, technologie de virtualisation Intel), vous devez tout d'abord vérifier si votre UC prend en charge la virtualisation matérielle, ce que vous pouvez faire en tapant :

egrep -c '(vmx|svm)' /proc/cpuinfo

Si le résultat est 0, votre UC ne prend pas en charge la virtualisation matérielle, nécessaire pour l'exécution de KVM. Si le résultat est 1 ou plus, vous pouvez y aller, mais veillez quand même à ce qu'elle soit activée dans le BIOS .

Ensuite, vous devez installer KVM, sauf si c'est déjà fait. Vous pouvez vérifier si votre processeur prend en charge KVM en tapant :

kvm-ok

Si vous avez KVM, vous verrez ça :

"INFO : Your CPU supports KVM extensions (Votre UC prend en charge les extensions KVM)INFO : /dev/kvm exists (/dev/kvm existe)KVM acceleration can be used" (L'accélération KVM peut être utilisée.)

Autrement, si vous voyez ce qui suit, vous devez accéder au BIOS et activer la technologie de virtualisation Intel :

"INFO : KVM is disabled by your BIOS (KVM est désactivé dans votre BIOS)HINT : Enter your BIOS setup and enable Virtualization Technology (VT),and then hard poweroff/poweron your systemKVM acceleration can NOT be used" (CONSEIL : accédez à la configuration du BIOS et activez la technologie de virtualisation, puis arrêtez et redémarrez votre système avec le bouton matériel. L'accélération KVM ne peut PAS être utilisée.)

L'étape suivante consiste à installer la KVM et quelques autres packages nécessaires. Pour cela, tapez :

sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils

Ensuite, ajoutez votre utilisateur au groupe KVM et au groupe libvirtd. Pour cela, tapez :

sudo adduser your_user_name kvm sudo adduser your_user_name libvirtd

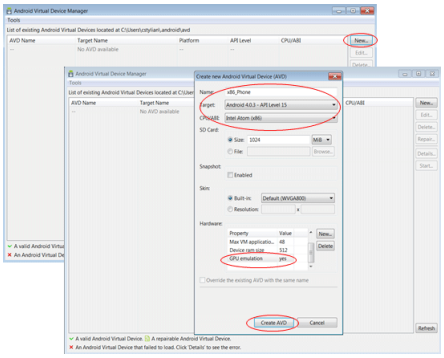
Après l'installation, ouvrez une nouvelle session afin que les modifications prennent effet. Vous pouvez tester l'installation en tapant :

sudo virsh -c qemu :///system list

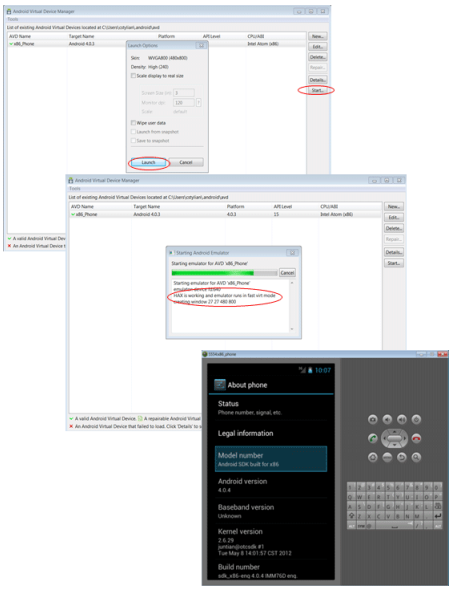
Vous êtes maintenant prêt à passer à l'étape suivante, qui consiste à créer et exécuter l'AVD (Android Virtual Device). Cette procédure est la même pour Linux et Windows.

2.4 Création d'un AVD (Android Virtual Device, appareil virtuel Android)

Après avoir installé le SDK et Intel HAXM (ou KVM pour Linux), vous pouvez créer un appareil virtuel possédant une émulation accélérée par le matériel. Pour cela, accédez au gestionnaire AVD Manager et créez un nouvel appareil. Veillez à bien sélectionner Intel Atom (x86) comme UC/ABI. La sélection s'affiche dans la liste déroulante uniquement si l'image système Intel x86 est installée. Pour que les graphiques soient plus lisses, activez l'émulation du GPU lors de la création de l'AVD.



Cliquez sur New (Nouveau) et créez votre AVD x86. Veillez à sélectionner une API prise en charge par une image système x86, que l'UC/ABI est définie sur x86, et que vous avez activé l'émulation du GPU (OpenGL ES). Après cela, cliquez sur Create AVD (Créer l'AVD) pour créer l'AVD.



Vous pouvez lancer l'AVD x86 en cliquant sur Start (Démarrer), puis sur Launch (Lancer).

Si l'installation est réussie, lorsque l'émulateur démarre, une boîte de dialogue s'affiche montrant qu'Intel HAXM est exécuté en mode virtuel rapide.

Si vous n'êtes toujours pas convaincu que vous utilisez une image système x86, vous pouvez toujours examiner les détails dans la section « About phone » (À propos du téléphone) dans l'émulateur.

Les gains de performances que vous constaterez avec Intel HAXM ou KVM dépendent de votre PC, lecteur, mémoire, etc., mais les performances devraient être entre 5 et 10 fois supérieures. La capture d'écran ci-dessous montre une comparaison côte à côte d'un AVD x86/HAXM et d'un AVD ARM. L'AVD x86 a démarré jusqu'à l'écran de verrouillage en 15 secondes alors que l'AVD non Intel a pris 40 secondes, une grande différence.

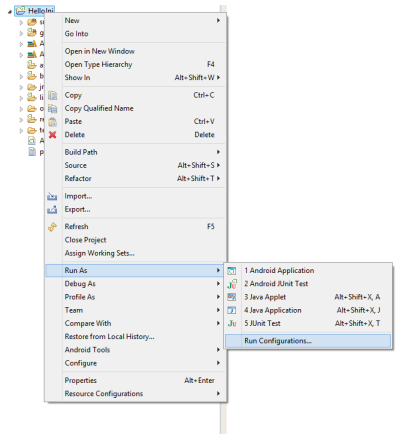
Les performances obtenues avec Intel HAXM (ou KVM) doivent être de 5 à 10 fois supérieures, selon la configuration de votre système : les logiciels et les charges de travail utilisés dans les tests de performances peuvent n'avoir été optimisés que pour des performances sur microprocesseurs Intel. Les tests de performances, comme SYSmark et MobileMark, sont mesurés à l'aide d'éléments spécifiques : systèmes informatiques, composants, logiciels, opérations et fonctions. Toute modification de l'un de ces facteurs peut provoquer une variation des résultats. Vous devez consulter d'autres infos et d'autres tests de performances pour évaluer en connaissance de cause les achats que vous envisagez de faire, et notamment les performances de ce produit combiné à d'autres. Configuration : un Mac Book Pro a été utilisé pour le test dans ce cas. Pour davantage d'informations, accédez ici : lien 46]

3 Méthodes les mieux connues

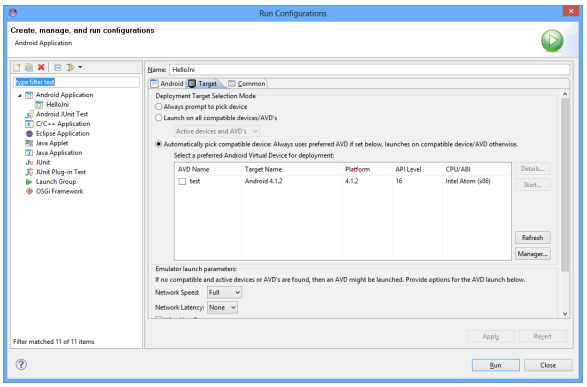
3.1 Tester votre application avec l'émulateur d'Eclipse

Qu'il s'agisse d'une application NDK ou d'une application Dalvik, vous pouvez utiliser Intel HAXM pour accélérer l'émulateur lors des tests. Si vous développez avec Eclipse, vous pouvez suivre ces simples étapes pour vous assurer que vous utilisez Intel HAXM lorsque vous démarrez l'émulateur.

Tout d'abord, veillez à bien créer votre AVD comme décrit dans l'étape 2. Si l'AVD est prêt, accédez à Run As -> Run Config (Exécuter comme -> Configuration d'exécution) comme illustré ci-dessous :



Vous devriez atterrir sur une page comme celle-ci :



Ici, vous pouvez sélectionner l'AVD que vous voulez en en cochant la case. Après avoir créé votre AVD et défini votre configuration, commencez à compiler votre projet et à le déboguer avec l'émulateur en sélectionnant Run As -> Android Application (Exécuter comme -> Application Android). Cela lancera automatiquement l'AVD avec accélération matérielle.

Après le démarrage de l'AVD, vous devriez voir l'écran d'accueil de votre application (après avoir déverrouillé l'écran).

3.2 Soumettre plusieurs APK pour différentes ABI vs. soumettre des fichiers binaires universels à Google Play

Dans le passé, vous soumettiez le fichier binaire universel de l'application développée, contenant toutes les bibliothèques et fichiers NDK, sans pouvoir différencier entre les architectures. Cela signifiait que les utilisateurs devaient télécharger le fichier APK entier contenant des fichiers qui n'étaient pas pertinents pour des architectures spécifiques, par ex., les utilisateurs x86 avaient un code ARM et inversement. Le problème était que si le fichier binaire était vraiment volumineux, l'utilisateur était forcé de télécharger une grande quantité de données qui ne concernaient pas leur appareil. Habituellement, cela est toujours acceptable si le volume de l'APK est inférieur à 10 ou 20 Mo.

La boutique d'applications de Google supporte la soumission de plusieurs APK contenant les différentes bibliothèques spécifiques à chaque architecture. Les APK envoyés doivent avoir des « versionCode » différents, et ce numéro doit être le plus élevé pour l'APK contenant les binaires x86. Vous pouvez par exemple utiliser cette convention :

Le premier chiffre se réfère à l'ABI, à savoir, 6 pour x86 ; ensuite le niveau d'API que vous ciblez, à savoir, 11 ; la taille de l'écran, 13 ; ensuite le numéro de version de votre application : 3.1.0.

Dans l'exemple ci-dessus, vous auriez 6 pour x86, 2 pour ARMv7 et 1 pour ARMv5TE. Cela permet de sélectionner de préférence les versions x86 pour les appareils x86 et les versions ARM pour les appareils ARM.

En suivant ces directives, vous pouvez vous assurer que vos utilisateurs tirent les meilleures performances de l'appareil qu'il possède. De plus, vous pouvez éviter que les utilisateurs tentent d'exécuter des applications sur des appareils spécifiques en raison de problèmes de traduction de code.

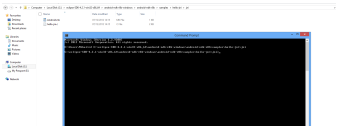
Vous trouverez davantage d'informations ici : lien 47.

3.3 Compiler votre code natif pour X86 en utilisant le NDK

Cette section vous montrera comment compiler la partie NDK de votre application pour x86.

Pour que votre application NDK s'exécute sur un AVD x86, vous devez compiler la bibliothèque de votre NDK pour l'architecture x86. Pour cela, suivez ces simples étapes :

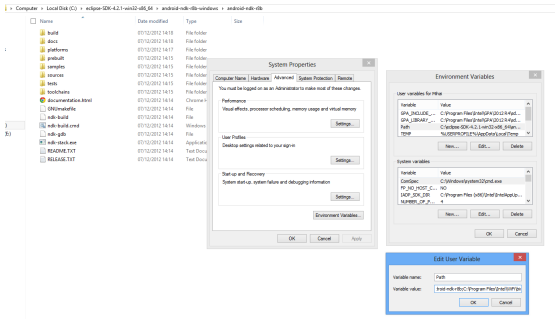
Ouvrez une invite de commande et naviguez jusqu'au dossier de vos fichiers NDK, comme suit :



Veillez à configurer le chemin d'accès de la variable d'environnement afin de pouvoir utiliser le script ndk-build depuis n'importe où.

3.3.a Ajouter le chemin d'accès du NDK à votre variable d'environnement

Pour configurer votre variable d'environnement pour le NDK, cliquez avec le bouton droit sur Ordinateur, puis sélectionnez Propriétés. Accédez à Paramètres système avancés et trouvez les variables d'environnement. Sélectionnez Path et cliquez sur Modifier. À la fin de la chaîne « Valeur de la variable », ajoutez le chemin d'accès au dossier racine du NDK, celui qui contient le fichier ndk-build.cmd comme dans l'image ci-dessous.



3.3.b Compiler avec le NDK

Après avoir navigué par l'invite de commande jusqu'au dossier NDK, exécutez : ***ndk-build APP_ABI :=all Cela compilera votre fichier NDK pour chaque architecture disponible, à savoir, ARMv5TE, ARMv7, x86 et mips. Pour compiler pour une architecture spécifique, remplacez « all » par les différentes architectures. Par exemple : ndk-build APP_ABI :=armeabi armeabi-v7a x86 mips Veillez à actualiser votre projet dans Eclipse pour prendre en compte vos derniers paramètres, à savoir, les derniers dossiers créés par le script ndk-build. Dans le dossier des***

bibliothèques de votre projet, vous devriez maintenant voir quatre dossiers, un pour chacune des architectures. Vous êtes maintenant prêt à utiliser l'AVD x86 avec votre application NDK.

3.3.c Autre manière de compiler avec le NDK

Une autre manière de compiler votre code natif pour toutes les architectures, y compris x86, consiste à modifier le fichier Application.mk qui se trouve dans le dossier jni. Si vous n'avez pas de fichier Application.mk, vous pouvez en créer un vous-même et ajouter l'instruction ci-dessous :

APP_ABI :=armeabi armeabi-v7a x86 mips

Ainsi, lorsque vous exécutez le fichier de commandes, à savoir, le script ndk-build, il compilera les bibliothèques pour toutes les architectures disponibles.

Pour une utilisation plus facile, vous pouvez également indiquer « all » au lieu d'énumérer toutes les architectures :

APP_ABI := all

Retrouvez l'article de ***beat-stauber*** *en ligne : lien 48*

Conception

Les derniers tutoriels et articles

Design Pattern : les mémentos

Un design pattern décrit une solution standard, utilisable dans la conception de logiciels, à des questions classiques et récurrentes.
Cet article vous redonne les points clés pour utiliser les patterns les plus utiles. Il vous propose surtout de télécharger des mémentos à imprimer au bureau.
Pour le moment, il n'y a que le mémento pour le Singleton mais d'autres mémentos viendront prochainement.

1 Introduction

Un design pattern décrit une solution standard, utilisable dans la conception de logiciels, à des questions classiques et récurrentes.

Les patrons de conception les plus connus sont au nombre de vingt-trois : Abstract factory, Builder, Factory, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy, Memento, Strategy, Command, Chain of responsability, Interpreter, Iterator, Mediator, Template, Visitor, State et Observer. On les classe généralement en trois familles : Création, Structure et Comportement. Ils sont désignés sous le nom de « Gang of Four » (GOF) en référence aux quatre créateurs du concept.

Cet article vous redonne les points clés pour bien comprendre/utiliser les patterns les plus utiles. Il vous propose surtout de télécharger des mémentos à imprimer vous-même au bureau.

1.1 Patterns par ordre alphabétique

Voici tous les patterns de cet article, classés par ordre alphabétique :

— singleton lien 50

1.2 Patterns par famille

Patterns de création :

— singleton lien 51

Patterns de structure :

— bientôt

Patterns de comportement :

— bientôt

1.3 Mises à jour

La première version de ce document a été écrite avec un seul pattern : le singleton. D'autres patterns (et les mémentos associés) seront ajoutés au fur et à mesure.

29 janvier 2014 : création du document (avec seulement le Singleton)

2 Design patterns

2.1 Singleton

2.1.a Principe

Le singleton permet de s'assurer qu'il n'existe qu'une seule instance d'une classe dans un environnement et pour une durée d'exécution donnée.

Points clés :

— instance privée et statique ;

— constructeur privé ;

— méthode d'accès publique et statique.

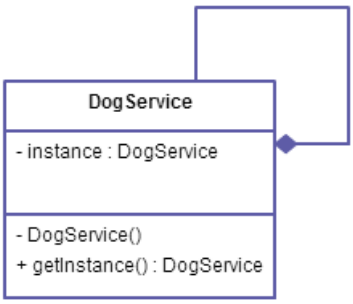


Diagramme de classe du Singleton

```
1  public class DogService {
2      private static DogService instance;
3
4      private DogService() {
5          ...
6      }
7
8      public static DogService getInstance
           () {
9          if (instance == null) {
10             instance = new DogService();
11         }
12         return instance;
13     }
14
15     public void truc() {
16         ...
17     }
18
19     ...
20 }
```

```
1  DogService dogService = DogService.
       getInstance();
2
3  dogService.truc();
```

2.1.b Synchronisation

Dans un contexte multithread, l'utilisation du pattern Singleton nécessite des précautions pour limiter les problèmes d'accès concurrents. Il faut synchroniser les appels et les initialisations.

```
1  public static synchronized DogService
       getInstance() {
2      if (instance == null) {
3          instance = new DogService();
4      }
5      return instance;
6  }
```

Malheureusement, on paie le coût de la synchronisation réalisée grâce au mot-clé « synchronized », à chaque appel et non pas seulement au premier (i.e. initialisation).

Des variantes (plus ou moins efficaces) permettent d'assurer la synchronisation ; vous en retrouverez une sélection ci-dessous. Je vous recommande également/vivement de consulter l'article de Christophe Jollivet intitulé « Le Singleton en environnement Multithread » lien 52 dont je me suis allègrement inspiré pour écrire cet article. ***Double-checked locking***

```
1  public static DogService getInstance() {
2      if (instance == null) {
3          synchronized (DogService.class)
               {
4              if (instance == null) {
5                  instance = new
                       DogService();
6              }
7          }
8      }
9      return instance;
10 }
```

La théorie semble parfaite et de nombreux sites Web recommandent cette solution. Le DCL n'apporte pourtant aucune garantie que cela fonctionnera. Ceci n'est pas inhérent à un bogue de la JVM mais au modèle de mémoire qui autorise l'écriture dans le désordre (« out-of-order writes »).

Mot-clé volatile

```
1  private static volatile DogService
       instance;
```

Là encore, c'est une « fausse bonne idée ». Dans les anciennes JVM, ça n'offre pas de garantie. Cela fonctionne dans les nouvelles versions, mais provoque un flush complet du registre du processeur. Le gain de performance recherché en n'utilisant pas la synchronisation est alors nul.

Thread local

```
1  public class DogService {
2      private static final ThreadLocal tl
           = new ThreadLocal();
3      private static DogService instance;
4
5      private DogService() {
6          ...
7      }
8
9      public static DogService getInstance
           () {
10         if (tl.get() == null) {
11             synchronized (DogService.
                   class) {
12                 if (instance == null) {
13                     instance = new
                           DogService();
14                 }
15                 tl.set(Boolean.TRUE);
16             }
17         }
18         return instance;
19     }
20 }
```

Défaut : l'utilisation du thread local est un peu lente.

Initialisation statique

```
1  \textbf{Initialisation statique}
```

L'initialisation est faite au lancement, même si le service n'est finalement jamais utilisé.

Inner class

```
1  public class DogService {
2      private static class InstanceHolder
           {
3          public static final DogService
               instance     = new DogService
               ();
4      }
5
6      private DogService() {
7          ...
8      }
9
10     public static DogService getInstance
           () {
11         return InstanceHolder.instance;
12     }
13 }
```

Le chargement des classes est « thread-safe » ; la classe encapsulée n'est initialisée que lors de l'appel de la méthode.

Enum

```
1
2  public enum DogService {
3      INSTANCE;
4
5      public static DogService getInstance
           () {
6          return INSTANCE;
7      }
8  }
```

```
1  final DogService service = DogService.
       getInstance();
2
3  // ou comme un enum
4  final DogService service = DogService.
       INSTANCE;
```

L'emploi d'un enum permet de prendre en compte le cas de la sérialisation facilement et empêche complètement l'initialisation par réflexion.

2.1.c JSR330

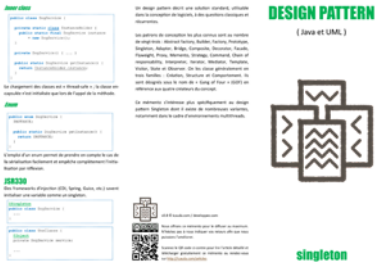
Des frameworks d'injection (CDI, Spring, Guice, etc.) savent initialiser une variable comme un singleton.

```
1  @Singleton
2  public class DogService {
3      ...
4  }
```

```
1  public class UneClasse {
2      @Inject
3      private DogService service;
4          ...
5      }
6
7      ...
```

Dans une application professionnelle, il est fort probable que vous utiliserez une bibliothèque comme Spring dont le Singleton est le « scope » par défaut. Avec ce type de framework, on n'a pas besoin (et on ne doit pas) programmer soi-même le Singleton.

2.1.d Mémento



Mémento sur le Singleton face 1

Mémento sur le Singleton face 2

3 Tous les mémentos

memento_singleton.pdf : lien 53

4 Conclusion

Et voilà...Revenez de temps en temps pour découvrir les nouveaux mémentos qui seront ajoutés.
Retrouvez l'article de ***Thierry Leriche-DessirierProfil Pro*** *en ligne : lien 54*

PyQt

Les dernières news PyQt

Sortie de PyQt 5.2.1 et première version de pyqtdeploy

Riverbank sort son outil d'aide au déploiement d'applications PyQt pour PC et mobiles et continue le développement de son binding

Le 14 mars dernier, Riverbank annonçait la sortie de la dernière version de son binding Qt pour Python. Cette version permet le support complet de Qt 5.2.1, mais on trouvera aussi comme nouveautés :

- les propriétés, signaux et slots peuvent dorénavant être définis différemment (par exemple des classes non QObject) ;
- le support à la création d'instances QSGGeometry.AttributeSet ;
- les valeurs primitives peuvent maintenant être données à chaque fois qu'un QJSValue est attendue ;
- la compilation de bibliothèques statiques est maintenant fonctionnelle ;
- le support pour la compilation sans OpenGL.

Les nouveautés proposées par Riverbank peuvent toucher aussi le déploiement des applications.

En effet le 26 mars dernier, Riverbank a annoncé la première mouture de pyqtdepoly. Cet outil doit faciliter le déploiement des applications pour Windows, Linux et OS X, mais aussi les applications mobiles. Cet outil crée le code C++ contenant l'interpréteur Python, l'application Python et les modules additionnels nécessaires. Pour finir, les outils Qt sont compilés en C++ pour la plateforme cible.

Commentez la news de ***Charlie Gentil*** *en ligne : lien 55*

Liste des liens

Page 2
lien 1 : ... http://fr.wikipedia.org/wiki/%C3%89mulation

Page 6
lien 2 : ... http://fr.wikipedia.org/wiki/%C3%89mulation
lien 3 : ... http://fr.wikipedia.org/wiki/Domaine_public_en_droit_de_la_propri%C3%A9t%C3%A9_intellectuelle_fran%C3%A7ais
lien 4 : ... http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=LEGITEXT000006055121&dateTexte=20110206
lien 5 : ... http://jeux.developpez.com/tutoriels/programmer-emulateur-console/tutoriel-1-tour-d-horizon/
lien 6 : ... http://jeux.developpez.com/tutoriels/programmer-emulateur-console/tutoriel-1-tour-d-horizon/
lien 7 : ... http://jeux.developpez.com/tutoriels/OpenGL-moderne/ouvrir-une-fenetre/

Page 10
lien 8 : ... http://jeux.developpez.com/tutoriels/OpenGL-moderne/matrices/
lien 9 : ... http://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/
lien 10 : ... http://jeux.developpez.com/tutoriels/OpenGL-moderne/tutoriel-2-premier-triangle/
lien 11 : ... http://jeux.developpez.com/tutoriels/OpenGL-moderne/tutoriel-2-premier-triangle/

Page 12
lien 12 : ... http://soat.developpez.com/tutoriels/java/projet-lambda-java8

Page 13
lien 13 : ... http://oliviercroisier.developpez.com/tutoriels/java/java8-nouveautes-interfaces/
lien 14 : ... http://openjdk.java.net/jeps/150
lien 15 : ... http://soat.developpez.com/tutoriels/java/time-date-java8/
lien 16 : ... http://openjdk.java.net/jeps/120
lien 17 : ... http://openjdk.java.net/jeps/174
lien 18 : ... http://soat.developpez.com/tutoriels/java/type-annotations-nashorn-java8/
lien 19 : ... http://pixelduke.wordpress.com/2013/...part-i-javafx/
lien 20 : ... http://blog.palominolabs.com/2014/02...vs-atomiclong/
lien 21 : ... http://openjdk.java.net/projects/jdk8/features
lien 22 : ... http://www.developpez.net/forums/d1376854-2/java/general-java/langage/decouvrons-java-8-ensemble/

Page 14
lien 23 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/programmation-fonctionnelle/

Page 15
lien 24 : ... http://www.oracle.com/technetwork/java/javase/downloads/index.html
lien 25 : ... http://www.developpez.net/forums/d1424910/java/general-java/java-8-disponible-plate-forme-se-met-aux-expressions-lambdas/
lien 26 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/introduction-et-installation/
lien 27 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/collections/
lien 28 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/programmation-fonctionnelle/
lien 29 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/utilitaires/
lien 30 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/cache-concurrence/
lien 31 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava-strings-primitifs/
lien 32 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/math/
lien 33 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/hash-io/

Page 17
lien 34 : ... http://blog.developpez.com/guava/p11149/collection/bloom_filter_de_guava_13

lien 35 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/csv-avec-java/

Page 19

lien 36 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/guava/hash-io/

Page 20

lien 37 : ... http://download.eclipse.org/orion/drops/R-5.0-201402262325/

lien 38 : ... http://www.developpez.net/forums/d1423751/environnements-developpement/eclipse/eclipse-orion-5-plus-vers-cloud/

Page 21

lien 39 : ... https://netbeans.org/

lien 40 : ... http://www.developpez.net/forums/d1425729/java/edi-outils-java/netbeans/netbeans-s-aligne-java-8-a/

Page 23

lien 41 : ... http://developer.android.com/wear/preview/start.html

lien 42 : ... http://officialandroid.blogspot.fr/2014/03/sharing-whats-up-our-sleeve-android.html

lien 43 : ... http://www.developpez.net/forums/d1317844-3/java/general-java/java-mobiles/android/android-4-0-devient-plus-populaire-gingerbread/#post7724420

Page 24

lien 44 : ... http://www.vogella.com/articles/Android/article.html

Page 31

lien 45 : ... http://vogella.developpez.com/tutoriels/vogella/Fragments-Android/

Page 35

lien 46 : ... http://www.intel.com/performance

Page 36

lien 47 : ... http://software.intel.com/fr-fr/articles/google-play-supports-cpu-architecture-filtering-for-multiple-apk

Page 37

lien 48 : ... http://intel.developpez.com/tutoriels/acceleration-emulateur-android/

Page 38

lien 49 : ... http://creativecommons.org/licenses/by-nc-sa/3.0/fr/

lien 50 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/design-pattern-mementos/#singleton

lien 51 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/design-pattern-mementos/#singleton

Page 39

lien 52 : ... http://christophej.developpez.com/tutoriel/java/singleton/multithread/

Page 40

lien 53 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/design-pattern-mementos/fichiers/memento_singleton.pdf

lien 54 : ... http://thierry-leriche-dessirier.developpez.com/tutoriels/java/design-pattern-mementos/

Page 41

lien 55 : ... http://www.developpez.net/forums/d1427667/autres-langages/python-zope/gui/pyside-pyqt/sortie-pyqt-5-2-1-premiere-version-pyqtdeploy/