

Commencez à développer dans OpenJDK 8 sous Ubuntu (partie 1)

PITTON Olivier
25 juillet 2013

Tables des matières

[Introduction](#)

[Installation des sources et dépendances](#)

[Installation des packages .deb](#)

[Récupération des sources](#)

[CUPS](#)

[ALSA](#)

[Freetype](#)

[Première compilation](#)

[Configuration](#)

[Compilation et création des images](#)

[Conclusion](#)

Introduction

OpenJDK est l'implémentation libre de Java SE. Il comprend un grand nombre de projets dans lequel les contributeurs sont les bienvenus. Néanmoins, démarrer dans un projet aussi important peut être un véritable calvaire, qu'il s'agisse du code, de l'installation ou de la compilation. Fort heureusement, la compilation a été grandement simplifiée par rapport aux versions antérieures, qui ne possédaient pas de script *configure*, des variables d'environnement dupliquées, notamment pour la parallélisation du build. Dans cet article, nous verrons pas à pas comment installer intégralement OpenJDK 8 et cela jusqu'à la compilation finale. Cet article fait partie d'une série dédiée à OpenJDK 8.

Installation des sources et dépendances

Installation des packages .deb

OpenJDK requiert un grand nombre de dépendances systèmes. Cela requiert d'avoir l'accès root sur votre machine. Avant de passer à la récupération des sources, nous allons en premier lieu les installer. Pour cela, exécuter les commandes suivantes :

```
sudo apt-get install gcc g++ build-essential ccache openjdk-7-jdk zip tar  
unzip mercurial  
sudo apt-get install libxrender-dev libxtst-dev libX11-dev libxext-dev  
libxrender1
```

Voici la description des paquets que vous avez installé :

1. **gcc et g++** : Compilateurs C et C++. Ils sont requis pour la compilation de tous les fichiers en langage C et C++, comme la VM Hotspot
2. **build-essential** : Paquet contenant la référence vers tous les paquets requis pour compiler un package debian.
3. **libX11-dev, libxext-dev, libxrender-dev, libxtst-dev et libxrender1** : Bibliothèques graphiques pour le système X Window et X11.
4. **openjdk-7-jdk** : Afin de compiler toutes les classes Java, nous avons besoin d'un compilateur. Si vous l'avez déjà installé, sachez qu'il vous faut au moins un JDK de version 7 ou supérieure.
5. **zip, tar, unzip** : Afin de compresser certaines ressources durant la compilation, comme les sources du JDK.
6. **ccache** : Agit comme un cache préprocesseur pour les compilateurs C / C++. Il permet de réduire de manière drastique la durée de compilation.
7. **mercurial** : Mercurial est un gestionnaire de version décentralisé. Il est utilisé par OpenJDK et nous permettra de récupérer les sources et les mises à jour des contributeurs.

Récupération des sources

Les sources d'OpenJDK sont situées sur un dépôt Mercurial à l'adresse : <http://hg.openjdk.java.net/jdk8/jdk8>.

Il y a deux étapes pour la récupération des sources, tout d'abord récupérer les scripts de mise à jour, puis les sources grâce à ces scripts.

L'extension Mercurial *hgforest* peut être utilisée pour faciliter le téléchargement, mais cela ne sera pas décrit ici.

Afin de réaliser une installation propre, la création de répertoire, modification de droits de fichier ... seront aussi explicités. Nous considérons que toute l'installation se fera dans le répertoire *Interne* situé dans le répertoire *home* de l'utilisateur courant.

OpenJDK 8 sera placé dans *~/Interne/jdk8* et les bibliothèques dans *~/Interne/lib*.

Passons maintenant à la récupération des scripts de configuration et à la création des répertoires.

```
cd ~
mkdir Interne
cd Interne
mkdir lib
hg clone http://hg.openjdk.java.net/jdk8/jdk8
cd jdk8
```

Une fois cette étape effectuée, nous allons modifier les permissions des fichiers permettant de configurer l'installation et de récupérer les sources de tous les projets. Ces fichiers n'ont pas les droits d'exécution, nous allons donc leur donner.

```
chmod 700 get_source.sh configure
```

Enfin, l'étape finale, nous allons récupérer toutes les sources via Mercurial d'OpenJDK. Cette étape peut prendre un certain temps. Je vous invite donc à effectuer les autres installations en parallèle.

De plus, si vous désirez recevoir les mises à jours des contributeurs régulièrement, il suffit simplement de relancer ce script pour cela. Ou d'utiliser manuellement Mercurial.

```
./get_source.sh
```

Maintenant que les sources ont été récupérées, voyons ce qui a été mis à jour et ce que contiennent les répertoires.

Le projet OpenJDK est décomposé de la manière suivante :

1. **.(racine)** : Contient l'ensemble des projets ainsi que la configuration standard et le makefile pour compiler OpenJDK.
2. **hotspot** : Contient le code la machine virtuelle Hotspot, incluant les sources et les makefiles.
3. **langtools** : Contient le code source du compilateur javac et des outils pour les langues.
4. **jdk** : Contient le code source d'OpenJDK runtime, c'est à dire toutes les classes utilisées couramment, les makefiles etc... On y trouve notamment le code natif qui

est ensuite lié à la machine virtuelle Hotspot.

5. **jaxp** : Contient le code source de JAXP, l'API pour traiter du XML.
6. **jaxws** : Contient le code source de JAXWS, l'API pour générer des web services.
7. **corba** : Contient le code source de l'implémentation CORBA d'OpenJDK.
8. **nashorn** : Contient le code source du nouveau moteur JavaScript remplaçant Rhino.

Dans ce tutoriel, nous ne nous intéresserons pas au code même, mais simplement à l'installation d'OpenJDK 8. Sachez que le code source de tous ces projets est très bien documenté, et que chaque fonction ou parcelle de code un peu complexe contient des explications.

CUPS

CUPS est une bibliothèque standard et open-source pour l'impression développée par Apple Inc. for OS® X et d'autres systèmes d'exploitation UNIX®. L'installation de cette bibliothèque, tout comme les deux suivantes, est triviale.

Penser à remplacer *version* et *name* par, respectivement, la version de CUPS / ALSA / Freetype que vous avez téléchargé et par le nom de l'utilisateur courant.

Tout d'abord récupérer les sources de la dernière version de CUPS sur le site <http://cups.org/software.php>.

Placer le fichier téléchargé dans le dossier `~/Interne/lib`. Pour lancer l'installation, effectuer les commandes suivantes.

Attention, CUPS requiert un accès root pour l'installation.

```
tar -xf cups-<version>.tar.bz2
mv cups-<version> cups
cd cups
./configure --prefix=/home/<name>/Interne/lib/build/cups
make
sudo make install
```

ALSA

ALSA, pour *Advanced Linux Sound Architecture*, est une bibliothèque offrant des fonctionnalités pour le traitement audio et MIDI.

ALSA contient beaucoup de bibliothèques spécifiques (driver, lib, ...), seule la bibliothèque nommée *lib* nous intéresse.

Vous pouvez récupérer les sources de cette bibliothèque sur le ftp dédié <ftp://ftp.alsa-project.org/pub/lib/>. Pour ce tutoriel, j'ai utilisé la version 1.0.9.

```
tar -xf alsa-<version>.tar.bz2
mv alsa-<version> alsa
cd alsa
./configure --prefix=/home/<name>/Interne/lib/build/alsa
make
make install
```

Freetype

Le projet Freetype est une bibliothèque d'affichage de polices écrit en C. Cette bibliothèque est petite et très portable.

Vous pouvez récupérer les sources sur <http://sourceforge.net/projects/freetype/files/>. Pour ce tutoriel, la version 2.5.0.1 a été utilisée. La compilation de la bibliothèque se fait de la même manière que précédemment.

```
tar -xf freetype-<version>.tar.bz2
mv freetype-<version> freetype
cd freetype
./configure --prefix=/home/<name>/Interne/lib/build/freetype
make
make install
```

Première compilation

Configuration

Nous rentrons enfin dans le vif du sujet. Nous allons maintenant configurer OpenJDK, afin de préparer la compilation de ce-dernier. Cette étape va nous permettre de paramétrer l'étape de compilation en spécifiant la mémoire allouée, les répertoires où trouver les bibliothèques installées précédemment, ...

Si vous désirez changer la configuration, il vous suffit simplement de réaliser à nouveau cette partie du tutoriel.

Voici une description des paramètres à spécifier pour le build. Attention, ces paramètres ne sont pas tous obligatoires. Les paramètres spécifiques à d'autres systèmes d'exploitation ne sont pas indiqués, qui sont principalement ceux de Windows.

Option de configuration	Description
--enable-debug	Modifie le niveau de debug à fastdebug. Il est identique à l'utilisation de --with-debug-level=fastdebug.
--with-alsa=path	Sélectionne le répertoire d'installation dans lequel se trouve Advanced Linux Sound Architecture (ALSA)
--with-boot-jdk=path	Sélectionne le Bootstrap JDK permettant de compiler OpenJDK
--with-boot-jdk-jvmargs="args"	Ajoute les options JVM à utiliser lors de l'utilisation du Bootstrap JDK pour la compilation.
--with-cacerts=path	Sélectionne le chemin où trouver le fichier cacerts.
--with-cups=path	Sélectionne le répertoire d'installation dans lequel se trouve CUPS.
--with-cups-include=path	Sélectionne le répertoire d'installation dans lequel se trouve les fichiers d'en-tête de CUPS.
--with-debug-level=level	Sélectionne le niveau d'information de debug. Les valeurs possibles sont : release,

	fastdebug, ou slowdebug.
--with-freetype=path	Sélectionne le répertoire d'installation dans lequel se trouve Freetype.
--with-import-hotspot=path	Sélectionne le répertoire d'installation d'Hotspot. Cette option est utile si vous avez déjà compiler Hotspot et que vous voulez éviter de la compiler de nouveau.
--with-target-bits=arg	Sélectionne l'architecture du build. Les valeurs possibles sont 32 ou 64
--with-jvm-variants=variants	Sélectionne les modes dans lequel compiler OpenJDK. Il s'agit d'une liste séparée par des virgules dont les valeurs possibles sont : server, client, kernel, zero et zeroshark
--with-memory-size=size	Sélectionne la mémoire (en MB) à allouer pour la compilation. Plus cette valeur est haute, plus la compilation sera rapide.
--with-num-cores=cores	Sélectionne le nombre de coeurs à utiliser pour la compilation. Plus cette valeur est haute, plus la compilation sera rapide.
--with-x=path	Sélectionne le répertoire d'installation des fichiers X11 et xrender.

Pour une compilation standard, beaucoup de ces options ne nous intéresseront pas. Puisque nous avons installé les fichiers de X11 et xrender grâce à apt-get, ceux-ci sont déjà situés dans les répertoires qu'ira vérifier OpenJDK. Voici un exemple de configuration avec le moins d'arguments possibles :

```
./configure --with-alsa=/home/<name>/Interne/lib/build/alsa
--with-cups=/home/<name>/Interne/lib/build/cups
--with-cups-include=/home/<name>/Interne/lib/build/cups/include
--with-freetype=/home/<name>/Interne/lib/build/freetype
```

Si vous désirez, et avez la possibilité, d'utiliser plus de ressources, n'hésitez pas à utiliser plus d'options. Sur une machine 48 coeurs, voici la ligne de commandes que j'utilise :

```
./configure --with-cups=/home/<name>/Interne/lib/build/cups
--with-cups-include=/home/<name>/Interne/lib/build/cups/include
--with-freetype=/home/<name>/Interne/lib/build/freetype/
--with-memory-size=32600 --with-num-cores=48 --with-target-bits=64
```

Une fois cette étape effectuée, nous pouvons passer à l'étape de compilation.

Compilation et création des images

Cette étape est probablement la plus simple et la plus chronophage. Sachez que seule la première compilation est longue. Grâce à l'utilitaire `ccache`, uniquement les fichiers modifiés seront recompilés, et leurs dépendances.

Ainsi, si vous avez modifié le JDK, vous n'aurez pas à recompiler entièrement CORBA.

La compilation se résume à utiliser l'utilitaire `make` :

```
make all
```

Si vous ne spécifiez aucun argument, vous compilerez tout, mais ne créerez pas d'images. Vous ne pourrez pas donc pas tester vos modifications, puisque vous n'allez pas générer les binaires `java`, `javac`, ...

Si vous désirez rebuild simplement, après une modification, utiliser plutôt :

```
make images
```

A la fin de l'emploi de *make*, vous devriez avoir un répertoire *build* complet. Afin de tester la compilation, allez dans le répertoire suivant :

```
/home/<name>/Interne/jdk8/build/linux-x86_64-normal-server-release/images/j2sd  
k-image/bin  
./java -version  
openjdk version "1.8.0-internal"  
OpenJDK Runtime Environment (build  
1.8.0-internal-olivier_2013_06_30_00_34-b00)  
OpenJDK 64-Bit Server VM (build 25.0-b38, mixed mode)
```

Conclusion

En conclusion, nous avons installé et configuré OpenJDK 8. De l'installation des paquets debian à la fin de la compilation. Vos modifications du code source d'OpenJDK seront bel et bien prises en compte, et vous pouvez commencer à travailler dans les entrailles de la bête. Vous pouvez d'ores et déjà aller regarder le contenu des différents projets, et nous verrons dans un prochain article une partie de l'intérieur du JDK et la machine virtuelle Hotspot.