OpenJDK Contribution Hands On

Contents

Introduction

In this hands-on lab we will create a production-ready patch for the OpenJDK along with some tests. Throughout the lab we will execute the following steps in order to prepare the patch (note: if you are already a contributor and have a development environment you can skip to step 4):

- step 1: sign OCA in order to accept the Oracle terms for contributing to OpenJDK
- step 2: contact the proper JDK mailing list in order to take work on a relevant piece of the OpenJDK (such as a small code enhancement, bug fix or a larger enhancement)
- step 3: create a development environment for OpenJDK
- step 4: build OpenJDK from latest sources
- step 5: contribute to the OpenJDK codebase by implementing some enhancements
- step 6: prepare JTReg tests for the contribution
- step 7: prepare WebRev patch for the contribution
- step 8: send the patch to the appropriate OpenJDK mailing list

In the following sections we will give a step-by-step overview of the separate steps in order to create an OpenJDK contribution. Note that this is the typical working flow for contributors that have read-only access to the OpenJDK repositories and are not allowed to commit patches in the OpenJDK repositories.

Step 1: Sign the OCA

Go to http://www.oracle.com/technetwork/community/oca-486395.html and download the PDF of the OCA. Use a PDF editing tool such as PDFBuddy (https://www.pdfbuddy.com/) in order to edit and sign the PDF. After the PDF is signed send it to oracle-ca_us@oracle.com. Please note that the process may take up to 2-3 weeks – you should receive a confirmation. If no confirmation is received during that period – continue sending emails until you receive a reply for OCA acceptance.

Step 2: Contact the proper OpenJDK mailing list

After you have signed the OCA you can contact the proper mailing list that is relevant to your expertise and ask on stuff you can contribute to. If you have a particular enhancement in mind or want to fix an OpenJDK bug (list in JIRA: https://bugs.openjdk.java.net). You then have to inform the relevant mailing list (http://mail.openjdk.java.net) about your intentions to implement the enhancement/bugfix/feature and check if there is someone else already working on the same stuff.

For this hands-on lab we have selected three OpenJDK issues to work on:

- [enhancement request] https://bugs.openjdk.java.net/browse/JDK-5050783 - add getStackTraceString() method to Throwable class
- [enhancement request] https://bugs.openjdk.java.net/browse/JDK-6201180 - add generics and collections support for generic arrays
- [enhancement request] https://bugs.openjdk.java.net/browse/JDK-4974893 - replace array parameters with varargs where possible

We have also informed the **core-libs-dev** mailing list on our intentions to work on the issues since they all fall into the category of JDK core classes.

Step 3: Create a development environment for OpenJDK

In order to create your own development environment for OpenJDK you have to set up a Linux environment – for Windows environments building OpenJDK is much more difficult and that is why this option is not recommended and considered for this lab. An OpenJDK development environment consists of the following:

- **Bootstrap JDK** – we need a bootstrap JDK in order to be able to build the JDK core classes. For the purpose of this hands-on lab we need JDK 8 installed (either OpenJDK 8 or Oracle JDK 8);

- **Mercurial** – this is the VCS of choice for OpenJDK sources. All of the OpenJDK repositories are located in http://hg.openjdk.java.net/. The repository we will be using is **jdk9/dev** since it contains the latest sources of OpenJDK used for stuff that will go into release 9 of the Java platform. (http://hg.openjdk.java.net/jdk9/dev/). For JDK9 development at least version 2.6.3 of mercurial is needed;

- **JTReg** – this is a command line tool used to execute tests for the OpenJDK platform (also called JTReg test harness). The tool was written before JUnit or TestNG existed as libraries for unit-testing of Java code and so this tool has a different usage that the fore-mentioned. Support for executing JUnit and TestNG was added to later version of JTReg so now you can also write JUnit/TestNG tests and execute them with JTReg. JTreg tests are typically three types: Java tests (the most preferable type of tests), shell script tests and applet tests. There is no IDE support (in terms of plug-ins) for JTReg – if you want you can run JTReg as an external tool from the IDE of your choice. There are currently more than 10000 JTReg tests for OpenJDK. Each bugfix/enhancement or feature must be covered with proper JTReg tests;

- **webrev.ksh** – this is a script used to generate a patch from the commits in the local OpenJDK repository. The webrev script is executed under a ksh shell – so you must have ksh installed. Patches generated from webrev can sent directly to the proper mailing list for review, approval and inclusion to the OpenJDK codebase;

- **IDE with OpenJDK sources** – it is convenient to use an IDE for making changes to the OpenJDK codebase. If you want to do modification on the Hotspot JVM in OpenJDK itself – you must have C/C++ support in your IDE as well. IDE such as Eclipse and Netbeans provide currently better support for C/C++ development than IntelliJ IDEA. However for the purpose of this hands-on lab any of the three: Eclipse, Netbeans or IntelliJ can be used because we will be making changes to the OpenJDK core classes (which are written in pure Java). Currently there are no guidelines for development of OpenJDK using any of the three IDEs – there are projects files (in the OpenJDK jdk9/dev repository) for development of OpenJDK using Netbeans. However one should typically import the sources in his IDE of choice and organize them properly for navigation. Since JDK 9 source are split into module directories (such as java.base, java.sql and so on) we have to import them in a separate project/module and link them based on the support provided by the IDE. For example in IntelliJ you have to option to create a separate project and import the entire JDK 9 repository folder as a source root. Since IntelliJ provides pretty good support for automatic detection of source folders it will include the proper subfolders from the JDK core classes that are package roots. A better option is to create a separate IntelliJ IDEA module that corresponds to an OpenJDK module in the OpenJDK repository and add other dependent modules - you can use the dependency diagram provided by JEP 200 (http://openjdk.java.net/jeps/200) in order to create IntelliJ modules in the proper order.

There are already prepared VirtualBox images with OpenJDK 9 development environment already set-up and ready for usage.

First download and install the latest version of VirtualBox suitable for your operating system from: https://www.virtualbox.org/wiki/Downloads

Then download the virtual machine export for OpenJDK:

- Fedora: ftp://bgjug.sty-consulting.com/openjdk-fedora/openjdk-fedora.ova (user/pass: openjdk/openjdk) – contains only IntelliJ as an IDE with imported OpenJDK sources

- Ubuntu: ftp://bgjug.sty-consulting.com/openjdk-ubuntu/Ubuntu_12.04_OpenJDK_dev.ova (user/pass: openjdk/j1a2v3a4) – contains IntelliJ, Netbeans and Eclipse as IDEs with imported OpenJDK sources

Double click on the OVA file of the downloaded virtual machine export and start the import into VirtualBox – it will take some time. After the virtual machine is imported – start it.

You can use the tips provided in Appendix B in order to check the state of your Linux VM and possibly - how to optimize it.

Step 4: Build OpenJDK from latest sources

In order to build an OpenJDK image execute the following commands:

Using the Fedora VM:

```
cd ~/sources/jdk9/dev
./get_sourse.sh
bash configure
make images
```

Using the Ubuntu VM:

```
cd ~/dev/jdk9_dev
./get_sourse.sh
bash configure
make images
```

The above commands perform the following:

- Navigate to the root of the OpenJDK Mercurial repositories;
- Update to the latest sources from the remote Mercurial repositories (OpenJDK uses the Mercurial forest plug-in in order to retrieve latest sources from all OpenJDK sub-repositories);
- Configure the environment to the latest changes (these are scripts that are executed in order to configure your environment based on latest OpenJDK environment configuration);
- Perform an increment build of the OpenJDK images (created images are located in the **build** folder).

You can refer to Appendix A for details on how to build a debuggable version of OpenJDK that can be used in order to debug OpenJDK sources from an IDE or a command-line Java debugger such as JDB.

Step 5: Implement enhancements to OpenJDK

JDK-5050783

Read carefully enhancement description at: https://bugs.openjdk.java.net/browse/JDK-5050783

Add getStackTraceString() to Throwable.

Usually people do:

```
catch (MyException e) {
    StringWriter sw = new StringWriter();
    e.printStackTrace(
        new PrintWriter(sw));
}
```

Think about adding default/static method to Throwable interface. Provide proper method implementation and discuss your solution.

JDK-6201180

Read carefully enhancement description at: https://bugs.openjdk.java.net/browse/JDK-6201180

Add generic support to Collection bulk methods. The methods addAll(), containsAll(), removeAll(), retainAll() can only receive other collections

You have to use an intermediate Arrays.asList:

```
collection.addAll(Arrays.asList("a", "b", "c"));
```

Consider using default methods for the purpose. Provide proper implementation and discuss your solution.

JDK-4974893

Read carefully enhancement description at: https://bugs.openjdk.java.net/browse/JDK-4974893

Replace array parameters with varargs. Look throughout all internal APIs. Provide a list of such APIs and implement enhancement.

Step 6: Prepare JTReg tests

JTReg is installed under ~/dev/jtreg. In order to execute a test with JTReg you can run the **jtreg** command and specify the test to execute. In order to try it out do the following:

```
cd ~/dev/jdk9_dev/jdk/test
jtreg -verbose:fail java/lang/invoke/AccessControlTest.java
```

You should see the following output in the terminal:

```
TEST: java/lang/invoke/AccessControlTest.java

TEST RESULT: Passed. Execution successful

--------------------------------------------------

Test results: passed: 1
```

Now create proper tests for the above three enhancements. In order to write JTReg tests use the following resources as a reference guide:

JTReg tutorial - https://wiki.openjdk.java.net/download/attachments/10747958/jtreg-tutorial.pdf
JTReg guidelines - http://openjdk.java.net/jtreg/writetests.html
JTReg command line options - http://openjdk.java.net/jtreg/command-help.html
JTReg tag specification - http://openjdk.java.net/jtreg/tag-spec.html
Running tests with JTReg - http://openjdk.java.net/jtreg/runtests.html

Step 7: Prepare a webrev patch

In order to create the patch with webrev run the following commands:

```
cd ~/dev/jdk9_dev
hg commit
ksh ~/dev/webrev.ksh
```

A patch file with name **webrev** is generated in the ~/dev/jdk9_dev directory that contains the enhancements along with the JTReg tests. Verify the contents of the patch.

Step 8: Send the patch to the proper mailing list

Send the three patches in a mail to the **core-libs-dev** mailing list.

Appendix A: Build a debug version of OpenJDK

In order to build a debug version of OpenJDK you can execute the following commands:

```
bash ./configure –enable-debug
make all CONF=linux-x86_64-normal-server-fastdebug
```

The debug version of OpenJDK is created in the **build** folder along with the normal OpenJDK images build.

Appendix B: Linux utilities for resource monitoring and optimization

You can use the following commands to inspect resource utilization on your VM:

- **top** – memory and CPU utilization
- **df** – used disk space
- du –s – used disk space (can be used
- **ps aux** – currently running processes

In order to release more space on your Linux environment you can do the following:

- remove unnecessary packages
- remove old kernels and kernel headers
- remove unused locales
- clear temporary and cache folders
- stop unused services that are running in the background
- keep at most only or two images of OpenJDK in the **build** directory - they are space consuming

For that purpose there are a number of utilities that can be used (they are installed in the Ubuntu VM with OpenJDK development environment):

- sudo gjig - wajig is a simple GUI wrapper arround apt-get, apt-cache and so on
- sudo debfoster - check what is additionally installed on the VM using apt-get or dpkg
- sudo xdiskusage - check current disk usage
- sudo localepurge - remove unnecessary linux locale packages
- sudo sysv-rc-conf - check services that are started in Ubuntu
- bleachbit - allows cleaning of browser cache, old kernels

For an OpenJDK development environment the following locales might be left on the Linux environment (unless you do any locale-specific changes that require locales to be installed on your system):

(The following are left on the VM)

- en
- en_GB
- en_US
- en_GB.UTF-8
- en_US.UTF-8

In order to check installed Linux kernels along with kernel headers you can use the following commands:

- uname -r - list currently installed kernel

- dpkg --list | grep linux-image - list all kernel version installed
- ls /usr/src | grep 'linux-headers' - list linux headers installed