The WhiteBox testing API

The HotSpot WhiteBox API

One of the not so well-known tools of the HotSpot VM is its WhiteBox testing API. Introduced in Java 7 it has been significantly improved and extended in Java 8 and 9. It can be used to query or change HotSpot internals which are not otherwise exposed to Java-land. While its features make it an indispensable tool for writing good HotSpot regression tests, it can also be used for experiments or for the mere fun of peeking into the VM. This entry will focus on the usage of the WhiteBox API in Java 8 and 9.

The WhiteBox API is implemented as a Java class (called sun.hotspot.WhiteBox) which defines various entry points into the HotSpot VM. Most of the functionality is implemented natively, directly in the HotSpot VM. The API is implemented as a singleton which can be easily retrieved by calling the static method WhiteBox.getWhiteBox().

Unfortunately, currently even a simple JavaDoc documentation of the API doesn't exist, so in order to make full use of its functionality, you'll have to peek right into WhiteBox.java.

In Java 8 the sources were located in the hotspot repository under:

http://hg.openjdk.java.net/jdk8u/jdk8u-dev/hotspot/file/tip/test/testlibrary/whitebox/sun/hotspot/WhiteBox.java

In Java 9 the Java API was moved to the top-level repository in order to make it accessible for regression tests from other repositories as well. It is now located under:

http://hg.openjdk.java.net/jdk9/dev/file/tip/test/lib/sun/hotspot/WhiteBox.java

Building

In Java 8 the WhiteBox API can be easily built with a special Makefile:

```
cd hotspot/test/testlibrary/whitebox
make BOOTDIR=<path_to_jdk>
ls -la wb.jar
```

This will create wb.jar which contains all the required classes in the current directory.

In Java 9 the wb.jar file is created as a part of the build-test-lib target:

```
mkdir output && cd output
bash <jdk9_src_root>/configure
make build-test-lib
ls -la support/test/lib/wb.jar
```

This will create the library in support/test/lib/wb.jar. (It is also possible to use the the build-test-lib-only target if the jdk has been built before.)

Usage

Because the WhiteBox API can be used to query and alter internal data structures of the VM its usage is not recommended for productive scenarios. In order to enable its functionality, one hast to place the WhiteBox jar file (i.e. wb.jar) into the bootstrap class path and one also has to enable the API in the HotSpot VM with the special diagnostic option -XX:+WhiteBoxAPI. A complete command line may look as follows:

```
java -Xbootclasspath/a:wb.jar -XX:+UnlockDiagnosticVMOptions -XX:+WhiteBoxAPI <Program_using_WhiteBox>
```

Notice that the WhiteBox API is also working with the commercial Oracle JDK, although it is not shipped with it. In general, older versions of wb.jar will most probably work fine with more recent HotSpot versions while newer versions of the WhiteBox API will usually not work together with an older VM.

WhiteBox JTreg tests

Many HotSpot regression tests already use the WhiteBox API under the hood and the JTreg framework makes it easy to write new ones without having to care about where the API is located and how it can be built. A minimal JTreg test for jdk9 which uses the WhiteBox API looks as follows (taken from http://hg.openjdk.java.net/jdk9/dev/hotspot/file/tip/test/sanity/WBApi.java):

```
/*
 * @test WBApi
 * @summary verify that whitebox functions can be linked and executed
 * @library /testlibrary /test/lib
 * @build WBApi
 * @run main ClassFileInstaller sun.hotspot.WhiteBox
 *                              sun.hotspot.WhiteBox$WhiteBoxPermission
 * @run main/othervm -Xbootclasspath/a:.
 *                   -XX:+UnlockDiagnosticVMOptions -XX:+WhiteBoxAPI WBApi
 */

import sun.hotspot.WhiteBox;

public class WBApi {
    public static void main(String... args) {
        System.out.printf("args at: %x\n",
          WhiteBox.getWhiteBox().getObjectAddress(args));
    }
}
```

The test uses the @library tag to specify the location of the WhiteBox sources (i.e. /test/lib). In this context an absolute path either refers to the root directory of the JTreg regression tests (i.e. hotspot/test) or to the external.lib.roots property in the TEST.ROOT file which is located in the root directory of the test suite (e.g. http://hg.openjdk.java.net/jdk9/dev/hotspot/file/tip/test/TEST.ROOT). By convention, the external.lib.roots property contains the relative path to the top-level jdk directory (e.g. external.lib.roots is defined as "../../" in hotspot/test/TEST.ROOT). This means that the WhiteBox source used by the HotSpot regression tests can be found under hotspot/test/../../test/lib which is exactly the location in the OpenJDK top level directory where the sources have been moved to in Java 9.

The "/testlibrary" path is required to find the sources of the ClassFileInstaller which are located under http://hg.openjdk.java.net/jdk9/dev/hotspot/file/tip/test/testlibrary/ClassFileInstaller.java. The "@run main ClassFileInstaller" directive instructs JTreg to copy the WhiteBox classes to the temporary scratch directory in which the actual test will be executed. This is necessary in order to make it possible to easily add the WhiteBox classes to the boot class path with the help of the "-Xbootclasspath/a:." option.

The second "@run" directive finally executes the regression test in a new VM (because of the attribute "main/othervm") with the required extra options described before (i.e. -Xbootclasspath/a:. -XX:+UnlockDiagnosticVMOptions -XX:+WhiteBoxAPI).

Implementation

As mentioned before, the WhiteBox class only contains Java wrappers for functionality actually implemented right in the HotSpot virtual machine. The implementation is located in the file http://hg.openjdk.java.net/jdk9/dev/hotspot/file/tip/src/share/vm/prims/whitebox.cpp

Notice that the native method WhiteBox.registerNatives() has no default JNI implementation in the VM but is handled implicitly by the class NativeLookup (see NativeLookup::lookup_entry() -> NativeLookup::lookup_style() -> lookup_special_native()) and bound to JVM_RegisterWhiteBoxMethods() which does the actual registration of all the native methods from the WhiteBox class.

Currently there is one method (WhiteBox.deoptimize()) which is implemented directly as a HotSpot compiler intrinsic.

Available methods

Following a short summary of the methods provided by the WhiteBox class. Please feel free to extend this list and to document individual methods more thoroughly.