APPENDIX C

Miscellaneous Changes in Java 10

In this chapter, you will learn:

- About the removal of the `javah` tool in JDK10
- How to use the `-h` option with the `javac` command to generate C/C++ header files
- What class data sharing (CDS) and application class data sharing (AppCDS) are
- How to use CDS and AppCDS in your applications

The javah Tool Is Gone

The JDK used to ship a tool named `javah`. The tool was used to generate C/C++ header native header files for all native method declarations in your Java code. The `javah` tool was deprecated in JDK9 for future removal. In JDK10, the tool has been removed.

JDK8 had added a `-h` option to the Java compiler (`javac`) to generate the C/C++ header files. The `-h` option accepts the directory name where it will place the generated header files. The following command compiles the `HelloJNI.java` file and places any C/C++ header files in the `jni_headers` directory:

```
javac -h jni_headers HelloJNI.java
```

When the `-h` option is used with the Java compiler, the compiler scans all source files being compiled for native method declarations and generates the C/C++ header files in the specified directory.

Data Sharing Among JVMs

When a JVM starts, it typically loads thousands of classes from the Java core library. Loading so many classes at startup takes time. If you start multiple JVMs, each JVM performs the same task—it loads the same copy of thousands of system classes at startup. This has two drawbacks:

- JVMs start slower.
- Each JVM loads a copy of the same system classes into memory, thus increasing the footprint.

Class Data Sharing (CDS), introduced in JDK5, is a JVM feature that allows pre-processing of system classes and sharing them across JVMs, thus reducing the startup time and footprint. CDS allowed sharing of Java core classes loaded by the bootstrap class loader.

K. Sharan, *Java Language Features*, https://doi.org/10.1007/978-1-4842-3348-1

Application Class Data Sharing (AppCDS), added in JDK10, extended the CDS feature, which allows sharing the application classes on the class path across JVMs. From JDK10, all types of class loaders (bootstrap, platform, application, and custom) can load classes from AppCDS. I explain CDS and AppCDS in detail in the subsequent sections.

■ **Tip** AppCDS was available in Oracle JDK8 and JDK9, which needed commercial licenses from Oracle. In JDK10, AppCDS is available in the OpenJDK.

Class Data Sharing

Class Data Sharing (CDS) was introduced in JDK5. The JRE installer loads the commonly used core Java classes into memory and dumps the loaded classes into a file called a *shared archive*. A shared archive contains classes in a pre-processed format to be memory mapped later. The shared archive is a `classes.jsa` file in your JDK/JRE installation. On Solaris, Linux, and MacOS platforms, the shared archive is stored in `/lib/[arch]/server/classes.jsa`. On Windows platforms, the shared archive is stored in `/bin/server/classes.jsa`. If you do not use the installer to install the JDK, the shared archive may not exist on your machine. If you do not find the shared archive on your machine, you can generate it at any time. Sometimes, you need to regenerate it when you have made changes to system classes. I show you how to generate the shared archive shortly.

When a JVM is launched, it looks for shared archive and memory-maps the shared archive in read-only mode. When the JVM needs to load a class, it attempts to load the class from the shared archive in memory first instead from the disk, thus reducing the class loading time. If another JVM is launched, the JVM first checks if the shared archive data is already loaded in memory by another JVM and reuses the same classes from the shared archive in memory, thus reducing the startup time further for the second JVM onward and significantly reducing the footprint by sharing the same class data in the same shared archive in memory.

CDS is supported with the G1, serial, parallel, and parallelOldGC garbage collectors. The shared string feature (part of CDS) supports only the G1 garbage collector on 64-bit non-Windows platforms.

You can create a shared archive using the `java` command with a `-Xshare:dump` option as follows. The command will create a shared archive (a `classes.jsa` file) in the `JAVA_HOME` directory, as described earlier in this section.

```
C:\Java10NewFeatures>java -Xshare:dump
```

```
narrow_klass_base = 0x0000000800000000, narrow_klass_shift = 3
Allocated temporary class space: 1073741824 bytes at 0x00000008c0000000
Allocated shared space: 3221225472 bytes at 0x0000000800000000
Loading classes to share ...
Loading classes to share: done.
Rewriting and linking classes ...
Rewriting and linking classes: done
Number of classes 1287
    instance classes   =  1227
    obj array classes  =    52
    type array classes =     8
Updating ConstMethods ... done.
Removing unshareable information ... done.
Scanning all metaspace objects ...
```

```
Allocating RW objects ...
Allocating RO objects ...
Relocating embedded pointers ...
Relocating external roots ...
Dumping symbol table ...
Relocating SystemDictionary::_well_known_klasses[] ...
Removing java_mirror ... done.
mc  space:      8272 [  0.0% of total] out of     65536 bytes [ 12.6% used] at
0x0000000800000000
rw  space:   4010056 [ 22.0% of total] out of   4063232 bytes [ 98.7% used] at
0x0000000800010000
ro  space:   7367280 [ 40.4% of total] out of   7405568 bytes [ 99.5% used] at
0x00000008003f0000
md  space:      6160 [  0.0% of total] out of     65536 bytes [  9.4% used] at
0x0000000800b00000
od  space:   6593552 [ 36.2% of total] out of   6619136 bytes [ 99.6% used] at
0x0000000800b10000
total    :  17985320 [100.0% of total] out of  18219008 bytes [ 98.7% used]
```

Let's create a simple Java class that you will run to verify the CDS features. Listing C-1 contains the code for a `CDSTest` class. The class is very simple. It prints a message to the standard output.

Listing C-1. A CDSTest Class

```
// CDSTest.java
package com.jdojo.java10.newfeatures;

public class CDSTest {
    public static void main(String[] args) {
        System.out.println("Hello CDS and AppCDS");
    }
}
```

You can use the one of the following three options to run your application with or without shared archive:

- `-Xshare:on`
- `-Xshare:off`
- `-Xshare:auto`

The `-Xshare:on` option requires using shared class data. If it is not possible to share class data, this option will fail launching the application. The `-Xshare:off` option does not attempt to use shared class data. The `-Xshare:auto` option uses shared class data if possible. This option does not fail if it cannot use shared class data. It's best to use the `-Xshare:auto` option.

The following commands run the `CDSTest` class without sharing class data and with sharing class data:

```
C:\Java10NewFeatures>java -cp dist\* com.jdojo.java10.newfeatures.CDSTest

Hello CDS and AppCDS
```

```
C:\Java10NewFeatures>java -Xshare:on -cp dist\* com.jdojo.java10.newfeatures.CDSTest
```

```
Hello CDS and AppCDS
```

You did not see any difference. However, the second time, the JVM startup time is quicker. How do you prove this? Let's run the same command on Linux using the `time` command as follows. The `time` command prints the time taken to run the application.

```
[/home/ksharan/Java10NewFeatures] $ time java -cp dist/* com.jdojo.java10.newfeatures.
CDSTest
```

```
Hello CDS and AppCDS

real 0m0.215s
user 0m0.124s
sys 0m0.034s
```

```
[/home/ksharan/Java10NewFeatures] $ time java -Xshare:on -cp dist/* com.jdojo.java10.
newfeatures.CDSTest
```

```
Hello CDS and AppCDS

real 0m0.108s
user 0m0.098s
sys 0m0.023s
```

Notice the time taken to run the program the second time, which was less compared to the first time. When CDS is enabled, core system classes are loaded from the shared archive. You can print the location of the loaded class using the `-Xlog:class+load=info` option. The following commands uses this option once when CDS is enabled and once when CDS is disabled:

```
C:\Java10NewFeatures>java -Xshare:off -Xlog:class+load=info -cp dist\* com.jdojo.java10.
newfeatures.CDSTest
```

```
[0.006s][info][class,load] opened: C:\java10\lib\modules
[0.014s][info][class,load] java.lang.Object source: jrt:/java.base
[0.015s][info][class,load] java.io.Serializable source: jrt:/java.base
[0.015s][info][class,load] java.lang.Comparable source: jrt:/java.base
[0.016s][info][class,load] java.lang.CharSequence source: jrt:/java.base
[0.016s][info][class,load] java.lang.String source: jrt:/java.base
...
```

```
[0.355s][info][class,load] com.jdojo.java10.newfeatures.CDSTest source: file:/C:/
Java10NewFeatures/dist/jdojo.java10.newfeatures.jar
[0.355s][info][class,load] java.lang.PublicMethods$MethodList source: jrt:/java.base
Hello CDS and AppCDS
...
```

```
C:\Java10NewFeatures>java -Xshare:on -Xlog:class+load=info -cp dist\* com.jdojo.java10.
newfeatures.CDSTest
```

```
[0.006s][info][class,load] opened: C:\java10\lib\modules
[0.017s][info][class,load] java.lang.Object source: shared objects file
[0.017s][info][class,load] java.io.Serializable source: shared objects file
[0.017s][info][class,load] java.lang.Comparable source: shared objects file
[0.017s][info][class,load] java.lang.CharSequence source: shared objects file
[0.017s][info][class,load] java.lang.String source: shared objects file
...
[0.361s][info][class,load] com.jdojo.java10.newfeatures.CDSTest source: file:/C:/
Java10NewFeatures/dist/jdojo.java10.newfeatures.jar
[0.362s][info][class,load] java.lang.PublicMethods$MethodList source: shared objects file
Hello CDS and AppCDS
...
```

When CDS was disabled, the sources for system classes, such as `Object` and `String`, are listed as `jrt:/java.base`. This means that these classes were loaded from the disk from `C:\java10\lib\modules`, which is the JRE runtime image. When CDS was enabled, the same classes list the source as a "shared objects file," which means they were loaded from the shared archive. Note that the application class `CDSTest` was loaded from the disk in both cases because CDS allows for sharing only system class data.

Application Class Data Sharing (AppCDS)

Until Java 10, sharing of class data was possible only for core Java classes, which were loaded by the bootstrap class loader. Java 10 extends the CDS feature to share application classes. In Java 10, the platform class loader, application class loader, and custom class loaders can also load classes from the shared archive.

To use AppCDS, you need to use the `-XX:+UseAppCDS` option with the `java` command. To share application class data, you need to create your own shared archive that will include the application classes as well as the core Java classes. When you run your application, you need to specify your shared archive file path using the `-XX:SharedArchiveFile` option. I walk you through the steps of using AppCDS. Your shared archive will contain all classes in the `jdojo.java10.newfeatures.jar` file, which is supplied in the `dist` directory for the source code of this book.

■ **Tip** Java 10 supports AppCDS only for classes loaded from a class path. It does not support AppCDS for classes loaded from a module path, such as using the `--module-path` option. Support for using AppCDS from a module path may be added in the future.

First, you need to create the list of classes you want to include in your shared archive for AppCDS. You can use the `-XX:DumpLoadedClassList` option to dump the list of loaded system classes into the JVM to a file. The `-XX:DumpLoadedClassList` option includes only the classes loaded by the bootstrap class loader. Make sure to use the `-XX:+UseAppCDS` option to include the classes loaded by the application and platform class loaders. You need to run the `java` command with the `-Xshare:off` option to generate the list of classes. The following command creates a `jdojoclasses.lst` file, which contains the list of all classes loaded into the JVM. It is a text file. You can open and edit it in a text editor of your choice.

```
C:\Java10NewFeatures>java -Xshare:off -XX:+UseAppCDS -XX:DumpLoadedClassList=jdojoclasses.lst
-cp dist\* com.jdojo.java10.newfeatures.CDSTest
```

```
Hello CDS and AppCDS
```

At this point, if you are satisfied with the list of classes in the `jdojoclasses.lst` file, which you want to include in the shared archive for AppCDS, use the `-Xshare:dump` and `-XX:SharedArchiveFile` options to create a shared archive. The following command creates a file named `jdojo.jsa`, which is the shared archive for AppCDS:

```
C:\Java10NewFeatures>java -Xshare:dump -XX:+UseAppCDS -XX:SharedClassListFile=jdojoclasses.lst
-XX:SharedArchiveFile=jdojo.jsa -cp dist\*
```

```
narrow_klass_base = 0x0000000800000000, narrow_klass_shift = 3
Allocated temporary class space: 1073741824 bytes at 0x00000008c0000000
Allocated shared space: 3221225472 bytes at 0x0000000800000000
Loading classes to share ...
Loading classes to share: done.
Rewriting and linking classes ...
Rewriting and linking classes: done
Number of classes 672
    instance classes   =   593
    obj array classes  =    71
    type array classes =     8
Updating ConstMethods ... done.
Removing unshareable information ... done.
Scanning all metaspace objects ...
Allocating RW objects ...
Allocating RO objects ...
Relocating embedded pointers ...
Relocating external roots ...
Dumping symbol table ...
Relocating SystemDictionary::_well_known_klasses[] ...
Removing java_mirror ... done.
mc  space:      5512 [  0.1% of total] out of     65536 bytes [  8.4% used] at
0x0000000800000000
rw  space:   2035392 [ 22.0% of total] out of   2097152 bytes [ 97.1% used] at
0x0000000800010000
ro  space:   3743728 [ 40.5% of total] out of   3801088 bytes [ 98.5% used] at
0x0000000800210000
```

```
md  space:      6160 [  0.1% of total] out of     65536 bytes [  9.4% used] at
0x00000008005b0000
od  space:   3190832 [ 34.5% of total] out of   3211264 bytes [ 99.4% used] at
0x00000008005c0000
total    :   8981624 [100.0% of total] out of   9240576 bytes [ 97.2% used]
```

It is time to see your shared archive with AppCDS in action. The following command does this:

```
C:\Java10NewFeatures>java -Xshare:on -XX:+UseAppCDS -XX:SharedArchiveFile=jdojo.jsa -cp
dist\* com.jdojo.java10.newfeatures.CDSTest
```

```
Hello CDS and AppCDS
```

Note the use of the following three options in the previous command. The `-Xshare:on` option enabled CDS, The `-XX:+UseAppCDS` option enabled AppCDS, and the `-XX:SharedArchiveFile=jdojo.jsa` option specifies the shared archive to use. There is nothing extraordinary in the output. The startup time was less and multiple JVMs will also share application classes at runtime. Use the `-Xlog:class+load=info` option to verify that your application class, `CDSTest`, was loaded from shared archive. Notice that the source for the `com.jdojo.java10.newfeatures.CDSTest` class is "shared objects file," which indicates that this application class was loaded from the shared archive.

```
C:\Java10NewFeatures>java -Xshare:on -XX:+UseAppCDS -XX:SharedArchiveFile=jdojo.jsa
-Xlog:class+load=info -cp dist\* com.jdojo.java10.newfeatures.CDSTest
```

```
[0.006s][info][class,load] opened: C:\java10\lib\modules
[0.025s][info][class,load] java.lang.Object source: shared objects file
[0.026s][info][class,load] java.io.Serializable source: shared objects file
[0.026s][info][class,load] java.lang.Comparable source: shared objects file
[0.027s][info][class,load] java.lang.CharSequence source: shared objects file
[0.027s][info][class,load] java.lang.String source: shared objects file
...
[0.290s][info][class,load] com.jdojo.java10.newfeatures.CDSTest source: shared objects file
[0.291s][info][class,load] java.lang.PublicMethods$MethodList source: shared objects file
Hello CDS and AppCDS
...
```

Let's rerun the `CDSTest` class with AppCDS enabled and using a different class path, as follows:

```
C:\Java10NewFeatures>java -Xshare:on -XX:+UseAppCDS -XX:SharedArchiveFile=jdojo.jsa -cp
build\modules\jdojo.java10.newfeatures\;dist\* com.jdojo.java10.newfeatures.CDSTest
```

```
An error has occurred while processing the shared archive file.
shared class paths mismatch (hint: enable -Xlog:class+path=info to diagnose the failure)
Error occurred during initialization of VM
Unable to use shared archive.
```

```
C:\Java10NewFeatures>java -Xshare:on -XX:+UseAppCDS -XX:SharedArchiveFile=jdojo.jsa -cp
dist\*;build\modules\jdojo.java10.newfeatures\ com.jdojo.java10.newfeatures.CDSTest
```

```
Hello CDS and AppCDS.
```

Remember that when you had created the `jdojo.jsa` shared archive, you had specified a class path, which was `dist\*`. When you specified a class path as `build\modules\jdojo.java10.newfeatures\;dist\*`, you received an error stating that there is a mismatch in the shared class path.

A shared archive makes a note of the class path used at the time it was created. When you enable the shared archive in the JVM, one of the following must be true; otherwise you get an error:

- The class path used to launch the JVM is the same as the class path used to create the shared archive.
- The class path noted in the shared archive is a prefix of the class path used to launch the application.

In our examples, the shared archive was created with the class path `dist\*`. When you use `dist\*` as the only part or the first part in the class path, your application runs with AppCDS. In the previous example when you got an error, you had specified `dist\*` as the last part in the class path. To see the details of what JVM was looking for when you got the error, run the command with a `-Xlog:class+path=info` option as follows:

```
C:\Java10NewFeatures>java -Xshare:on -XX:+UseAppCDS -Xlog:class+path=info
-XX:SharedArchiveFile=jdojo.jsa -cp build\modules\jdojo.java10.newfeatures\;dist\* com.
jdojo.java10.newfeatures.CDSTest
```

```
[0.008s][info][class,path] bootstrap loader class path=C:\java10\lib\modules
[0.010s][info][class,path] opened: C:\java10\lib\modules
[0.016s][info][class,path] type=BOOT
[0.016s][info][class,path] Expecting BOOT path=C:\java10\lib\modules
[0.017s][info][class,path] ok
[0.018s][info][class,path] type=APP
[0.018s][info][class,path] Expecting -Djava.class.path=dist\jdojo.java10.newfeatures.jar
[0.019s][info][class,path]
[0.019s][info][class,path] [APP classpath mismatch, actual: -Djava.class.path=build\modules\
jdojo.java10.newfeatures\;dist\jdojo.java10.newfeatures.jar
An error has occurred while processing the shared archive file.
shared class paths mismatch (hint: enable -Xlog:class+path=info to diagnose the failure)
Error occurred during initialization of VM
Unable to use shared archive.
```

Summary

The `javah` tool was deprecated in JDK9 for future removal. In JDK10, the tool has been removed. JDK8 added a `-h` option to the Java compiler (`javac`) to generate the C/C++ header files. The `-h` option accepts the directory name where it will place the generated header files. From JDK10, you need to use the `javac` command with the `-h` option to generate C/C++ header files.

Class Data Sharing (CDS), introduced in JDK5, is a JVM feature that allows sharing of classes across JVMs-thus reducing the startup time and footprints of JVMs. CDS allows sharing of Java core classes loaded by the bootstrap class loader.

Application Class Data Sharing (AppCDS), introduced in JDK10, extended the CDS feature. It allows for sharing application classes on the class path across JVMs. From JDK10, all types of class loaders (bootstrap, platform, application, and custom) can load classes from AppCDS. AppCDS was available in Oracle JDK8 and JDK9, which needed commercial licenses. In JDK10, AppCDS is available in the OpenJDK.

When you use shared archives with AppCDS, the class path used to create the shared archive must have the same prefix as the class path that is used to launch the application with AppCDS enabled. Otherwise, a class path mismatch error occurs and the application fails to start.