JDK 9 Outreach

JDK 9 Outreach

Introduction

With JDK 9 development underway in the OpenJDK Community, and based on our experiences with open source projects testing against JDK 9 Early Access builds in the Quality Outreach effort, it seems like a good idea to compile the accumulated wisdom to make it easier for new projects to start testing against JDK 9.

Caveat Lector

JDK 9 has not been released yet. That means the tips and suggestions accumulated here should not be taken as the final word on how to adjust your code for JDK 9 as some things may (and probably will) change as the set of JEPs targeted for JDK 9 changes. If in doubt, ask - preferably on the adoption-discuss mailing list.

JDK 9 Features

This page does **not** provide a comprehensive list of planned or targeted JDK 9 features. For an up to date list of JEPs targeted for JDK 9, please consult the JDK 9 Project page.

The Little Things

Before you get started testing your code against JDK 9, there are a few little things to consider.

JDK 9 Early Access Builds

You can build JDK 9 yourself by following the build instructions to build the OpenJDK source code in the jdk9/jdk9 forest. Please check the list of supported build platforms to verify that your build platform is supported before you attempt building JDK 9 from source yourself.

JDK 9 Early Access builds may also be provided by third parties. Oracle publishes regular JDK 9 builds at http://jdk.java.net/9, for example. In the same location, Oracle also publishes regular JDK 9 builds based on the latest Project Jigsaw source code at http://jdk.java.net/jigsaw/.

Look for unrecognized VM options

If your application's startup script launches the JDK 9 JVM with an unrecognized VM option, it will exit and let you know that it couldn't create a VM because of that unrecognized VM option. Some of the VM flags that were deprecated in JDK 8, have been removed in JDK 9.

Run jdeps on your code

In JDK 8 a new Java Dependency Analysis Tool (jdeps) was added to help developers understand the static dependencies of their applications and libraries. It can help you find dependencies on any internal, unsupported or private APIs that your application or its libraries use. A program using such APIs is not guaranteed to work in future versions of JDK 9 or even the same platform. For more information, please see this document.

A jdeps plugin for Apache Maven exists.

For best results, please make sure to run jdeps from the most recent JDK 8 update release or JDK 9 Early Access build.

Update your dependencies

Third-party libraries that encounter issues on JDK 9 may have had those issues fixed meanwhile as their developers begun to test with JDK 9 and address problems found. If you encounter issues with third party libraries, please check if there is a newer version that addresses them, and update your code to depend on the newer version.

For example, if your application uses Eclipse RCP 4.4 or below, it may exit with an error message on JDK 9:

```
java.lang.NoClassDefFoundError: org/w3c/dom/stylesheets/StyleSheet
```

Eclipse RCP 4.5 contains a fix for that issue.

Cross compilation for older platform versions

In order to use `javac` to cross-compile to an older release of the platform it is not sufficient to just set the `-source` and `-target` options to the older value; the bootclasspath must also be set to correspond to the older release too. Setting the bootclasspath is sometimes forgotten, potentially leading to obscure errors at runtime.

In JDK 9, the new `-release` flag in `javac` addresses both of these shortcomings. Now only a single flag (`-release`) needs to be set to cross compile compared to three flags (`-source, -target, -bootclasspath`) and the needed information is included in the JDK.

Testing Your Code

In general, it's simpler to start by building your code in your familiar build environment, and test it by running it on JDK 9, than to start by building it on JDK 9. The tools and libraries used in your build process might themselves not yet have been tested with JDK 9 by their developers.

JDK 9 changes that may affect your code

Some of the changes targeted for JDK 9 may affect code that relies on default, deprecated, removed, unsupported, internal or unspecified functionality. The list below has been compiled from the list of JEPs targeted for JDK 9 and other resources. It is not an exhaustive list of changes - there may be other changes you run into that have not been listed here.

Where available, the list provides links to further resources, like JEPs, mailing list threads or JBS issues.

Added

OCSP Stapling for TLS

JEP 249 may affect code connecting to TLS servers that cannot accept the `status_request` or `status_request_v2` TLS extensions. It implements OCSP stapling via the TLS Certificate Status Request extension (section 8 of RFC 6066) and the Multiple Certificate Status Request Extension (RFC 6961). The OCSP stapling feature will be enabled by default in the JDK with this implementation.

Multi-Release JAR Files

JEP 238 may affect code implementing runtimes with class loaders using `ZipFile` to load classes, instead of using the URL class loader or a custom class loader leveraging `JarFile` to obtain platform-specific class files. Such code will not be multi-release JAR file aware.

Parser API for Nashorn

JEP 236 may affect code that uses internal classes in the `jdk.nashorn.internal.ir` package and its sub-packages.

Prepare for v53 class files

JDK-8148651 may affect code that needs to process class files. Anyone writing tools and libraries for use on JDK 9 that process class files should update their code to handle the new version number. Code using such tools and libraries should be updated to a version that supports v53 class files.

Prepare JavaFX UI Controls & CSS APIs for Modularization

JEP 253 may affect code that uses internal classes in the `com.sun` package and its sub-packages.

Validate JVM Command-Line Flag Arguments

JEP 245 may affect code that passes invalid flag arguments to the JVM. It validates the arguments to all JVM command-line flags so as to avoid crashes, and ensured that appropriate error messages are displayed when they are invalid.

XML Catalogs

JEP 268 may affect code using the existing internal catalog implementation. Existing libraries or applications that use the internal API will need to migrate to the new API in order to take advantage of the new features.

Platform-Specific Desktop Features

JEP 272 may affect code that uses internal classes in the `apple & com.apple` packages and their sub-packages. It is not intended to provide direct replacements for all of the internal OS X APIs present in JDK 8. Existing libraries or applications that use the internal API will need to migrate to the new API. Specifically, the JEP will not provide a replacement for `com.apple.concurrent` and `apple.applescript` packages.

Changed

Arrays.asList(x).toArray() returns Object[]

JDK-6260652 may affect code that relies on the Arrays.asList(x).toArray() implementation to return a clone of the backing array, which might be a String[], for example, instead of returning an Object[], as specified.

Create PKCS12 Keystores by Default

JEP 229 may affect code that uses keystores. It changes the default keystore type from JKS to PKCS12. By default, new keystores will be created in the PKCS12 keystore format. Existing keystores will not change and keystore applications can continue to explicitly specify the keystore type they require.

Disable SHA-1 Certificates

JEP 288 may affect code that uses X.509 certificate chains with SHA-1 based signatures.

Enable GTK 3 on Linux

JEP 283 may affect graphical applications on Linux. In cases where GTK 3 is required for interoperability, and this requirement can be detected sufficiently early, GTK 3 will be enabled automatically.

Encapsulate Most Internal APIs

JEP 260 may affect code that uses JDK internal APIs. It makes most of the JDK's internal APIs inaccessible by default, leaving a few critical, widely-used internal APIs accessible, until supported replacements exist for all or most of their functionality.

In addition, due to changes necessary to implement this feature, the stack trace of reflective calls will appear somewhat different. Any code analysing, or filtering, based on the stack trace element's class name should be updated appropriately, to handle this.

HarfBuzz Font-Layout Engine

JEP 258 may affect code that uses the font-layout engine. It replaces the existing ICU OpenType font-layout engine with HarfBuzz.There may be some minor rendering differences between the libraries, as a result of different implementations and a use of more up-to-date OpenType specifications.

Indify String Concatenation

JEP 280 may affect code that used to instrument the `StringBuilder::append` chains generated by `javac`, such as byte code manipulation/weaver tools.

Make G1 the Default Garbage Collector

JEP 248 may affect code that does not explicitly specify the Garbage Collector to use. It makes the low-pause collector G1 the default garbage collector on 32- and 64-bit server configurations.The resource usage of G1 is different from Parallel GC, which is currently the default.

When resource usage overhead needs to be minimized a collector other than G1 should be used, and after this change the alternate collector will have to be specified explicitly.

Marlin Graphics Renderer

JEP 265 may affect code using the Java 2D graphics rasterizer. This JEP updatse Java 2D to use the Marlin Renderer as the default graphics rasterizer.

Modular Run-Time Images

JEP 220 may affect code that relies on the removed endorsed-standards override mechanism, the removed extension mechanism, and the existence of rt.jar and tools.jar files as well as the old directory layout of files in the JDK & JRE installation image.

New Version-String Scheme

JEP 223 may affect code that relies on the old version string scheme to distinguish major, minor, and security-update releases.

Unified GC Logging

JEP 271 may change the GC log message format, without necessarily reproducing all log entries in the new logging format or ensuring that current GC log parsers work without change on the new GC logs.

Unified JVM Logging

JEP 158 may change log message formats and possibly the meaning of some JVM options.

Use CLDR Locale Data by Default

JEP 252 may affect code that uses locale-sensitive services such as date, time, and number formatting which may behave differently for locales not supported in JDK 8.

UTF-8 Property Files

JEP 226 may affect code that loads property files. It defines a means for applications to specify property files encoded in UTF-8, and extends the `ResourceBundle` API to load them by default.

Removed

Remove apple script engine code in jdk repository

JDK-8143404 may affect code that relies on the removed AppleScript engine.

Remove GC Combinations Deprecated in JDK 8

JEP 214 may affect code that relies on the removed GC combinations.

Users who are using any of the flags that are being removed will have to update their JVM-startup command lines. If they are moving from JDK 8 to JDK 9 then they will already have seen warning messages and thus should not be surprised. For a detailed list of the flags and flag combinations that have stopped working in JDK 9, please consult the JEP text.

Remove HTTP Proxy implementation from RMI

JDK-8066750 may affect code that relies on the removed HTTP Proxy implementation in RMI.

Remove Launch-Time JRE Version Selection

JEP 231 may affect code that relies on the removed ability to request, at JRE launch time, a version of the JRE that is not the JRE being launched.

Remove java-rmi.exe and java-rmi.cgi

JDK-6512052 may affect code that relies on the removed java-rmi.exe launcher.

Remove the JVM TI hprof Agent

JEP 240 may affect code that relies on the removed `hprof` agent library as part of the JDK.

Remove the jhat Tool

JEP 241 may affect code that relies on the removed experimental, unsupported, and out-of-date jhat tool.

Removed API references to java.awt.peer and java.awt.dnd.peer packages

JDK-8037739 may affect code that uses types from **java.awt.peer** and **java.awt.dnd.peer** packages.

All methods that refer to types defined in the **java.awt.peer** and **java.awt.dnd.peer** packages (the "peer types") will be removed from the Java API in Java SE 9. Application code which calls any such method which accepts or returns a type defined in these packages will no longer link. This is a **BINARY** incompatible change.

Removed Packer/Unpacker addPropertyChangeListener and removePropertyChangeListener methods

JDK-8029806 may affect code that uses the **Packer/Unpacker addPropertyChangeListener** and **removePropertyChangeListener** method**s**, which were deprecated in JDK 8, and removed in JDK 9.

Removed LogManager addPropertyChangeListener and removePropertyChangeListener methods

JDK-8029805 may affect code that uses the **LogManager addPropertyChangeListener** and **removePropertyChangeListener** method**s**, which were deprecated in JDK 8, and removed in JDK 9.

Removed com.sun.security.auth.callback.DialogCallbackHandler

JDK-8029904 may affect code that uses the **com.sun.security.auth.callback.DialogCallbackHandler**, which was deprecated in JDK 8, and removed in JDK 9.

Remove stopThread RuntimePermission from the default java.policy

JDK-7067728 may affect code that relies on the removed stopThread RuntimePermission from the default java.policy.

Remove support for serialized applets from java.desktop

JDK-8134808 may affect code that relies on support for serialized applets.

Build

Annotations Pipeline 2.0

JEP 217 may affect source code that uses annotations due to the refactoring of the `javac` annotation pipeline, which should not be externally noticeable except where bugs were fixed and correctness improved.

Deprecate the Applet API

JEP 217 may affect source code that uses the applet API. The deprecation annotations will cause deprecation warnings to be emitted by the Java compiler for all code that uses this API. If warnings are treated as errors, they will result in build failure.

Enhanced Deprecation

JEP 277 may affect source code that uses a deprecated Java SE API. Several Java SE APIs will have a `@Deprecated` annotation added, updated, or removed. Deprecating APIs will increase the number of mandatory warnings that projects encounter when building against newer versions of Java SE. For more information about these planned changes, please consult the JEP text.

HTML5 Javadoc

JEP 224 may affect source code with Javadoc comments since the `-Xdoclint` feature of `javadoc` will be modified to validate input comments based upon the requested markup.

Milling Project Coin

JEP 213 may affect source code that uses underscore ("_") as an identifier. Its use generated a warning as of Java SE 8, and has been turned into an error in JDK 9.

Modular Java Application Packaging

JEP 275 may affect code that uses the `javapackager` tool. The packager will only create applications that use the JDK 9 runtime.

Module System

JEP 261 may affect source code that is not compiled with the `javac` compiler using the `-source`, `-target` or `-release` options to select the use of *legacy mode*.

The stack traces generated for exceptions at run time have been extended to include, when present, the names and version strings of relevant modules. The detail strings of exceptions such as `ClassCastException`,`IllegalAccessException`, and `IllegalAccessError` have also been updated to include module information. Any code analysing, or filtering, based on the stack trace element's detail strings should be updated appropriately, to handle this.

In addition, this feature may affect code that assumes that the application class loader or the extension class loader is an instance of `URLClassLoader`, rather than of an internal class.

This feature may also affect code that attempts to set or read the bootclasspath using the `-Xbootclasspath` or the `-Xbootclasspath/p` options, or the removed JDK-specific system property `sun.boot.class.path`.

Changes due primarily to the introduction of the Java Platform Module System may affect code which

- expects that applying the `public` modifier to an API element guarantees that the element will be everywhere accessible, or
- expects to use `ClassLoader::getResource*` and `Class::getResource*` methods to read JDK-internal resources, or
- expects to use the `java.lang.reflect.AccessibleObject::setAccessible` method to gain access to members of packages that are not exported by their defining modules, or
- as JVM TI agents expect to be able to instrument Java code that runs early in the startup of the run-time environment.

Existing code that invokes `ClassLoader::getSystemClassLoader` and blindly casts the result to `URLClassLoader`, or does the same thing with the parent of that class loader, might not work correctly.

Existing custom class loaders that delegate directly to the bootstrap class loader might not work correctly; they should be updated to delegate to the extension class loader.

The five-parameter `transform` method declared in the`java.lang.instrument.ClassFileTransformer` interface is now a default method.

Existing code that scans for `META-INF/services` resource files previously found in `rt.jar` and other internal artifacts which are not present in the corresponding system modules files might not work correctly.

The JDK-specific system property `file.encoding` can be set on the command line via the `-D` option, as before, but it will only work when it specifies a charset defined in the base module. Existing launch scripts that specify other charsets might not work correctly.

Removed support for 1.5 and earlier source and target options

JDK-8011044 may affect source code that uses javac -source or -target options below 6/1.6 to compile.

Simplified Doclet API

JEP 221 may affect code that uses the Doclet API. It replaces the Doclet API with a simpler design that leverages newer alternative APIs with improved functionality, and updates the standard doclet to use it.

Feedback on new JDK 9 features

If you would like to provide feedback on JDK 9, but don't know yet where to start, please begin by subscribing to the adoption-discuss mailing list, and initiating a conversation on that mailing list about your particular item of feedback. Depending on the content, you may be asked to file an issue in in JBS, to provide further information or to continue the conversation on a specific OpenJDK Community mailing list more suitable for further discussion.

Initial module-system design, API, and early-access builds

JEP 261 implements the Java Platform Module System, as specified by JSR 376, together with related JDK-specific changes and enhancements. As announced on the jigsaw-dev mailing list, experience reports from early testers and adopters are especially welcome. Please try out the EA builds and let the developers know what you learn on the jigsaw-dev mailing list.

Other potential changes to consider

There may be other changes worth considering to add to this page. If you have suggestions for items to add to this list, please send an e-mail to the adoption-discuss mailing list.