Project Jigsaw: Module Class Loading and Bootstrapping

Jigsaw Team

May 4, 2012

Class Loader Relationship

- Java SE mandates two loaders
  - VM bootstrap class loader (from the JVMS)
  - System class loader (from the SE API)
    - Default delegation parent for user-defined loaders
    - Delegation in general may not be hierarchical
- In “classpath mode”, JDK creates three loaders
  - VM bootstrap class loader
  - Extension class loader (implementation-specific)
  - System class loader
    - Typically the loader used to start an application

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Class Loader Relationship (2)

- In “module mode”, JDK creates m+1 loaders
  - VM bootstrap class loader (see later for what it does)
  - One loader per one or more modules
    - A module loader is used to start an application
    - Module loaders load their dependencies, e.g. java.base, that are lazily created when it loads a class
    - A module loader has no parent but instead it does direct class lookup and delegates to the module loader that defines the referenced class
- No need for JDK-specific extension loader
- Application code can still create its own custom class loader (e.g. URLClassLoader to load from the network)

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Module Class Loader

- A class loader for one or more modules
  - Define Module (module-info)
  - Load classes
    - Define the classes if found in the modules
    - Delegate to the module loader exporting a class (i.e. classes visible but not defined by this module loader)
    - No accessibility checking (just like SE 7)
  - Find resources
  - Load native libraries

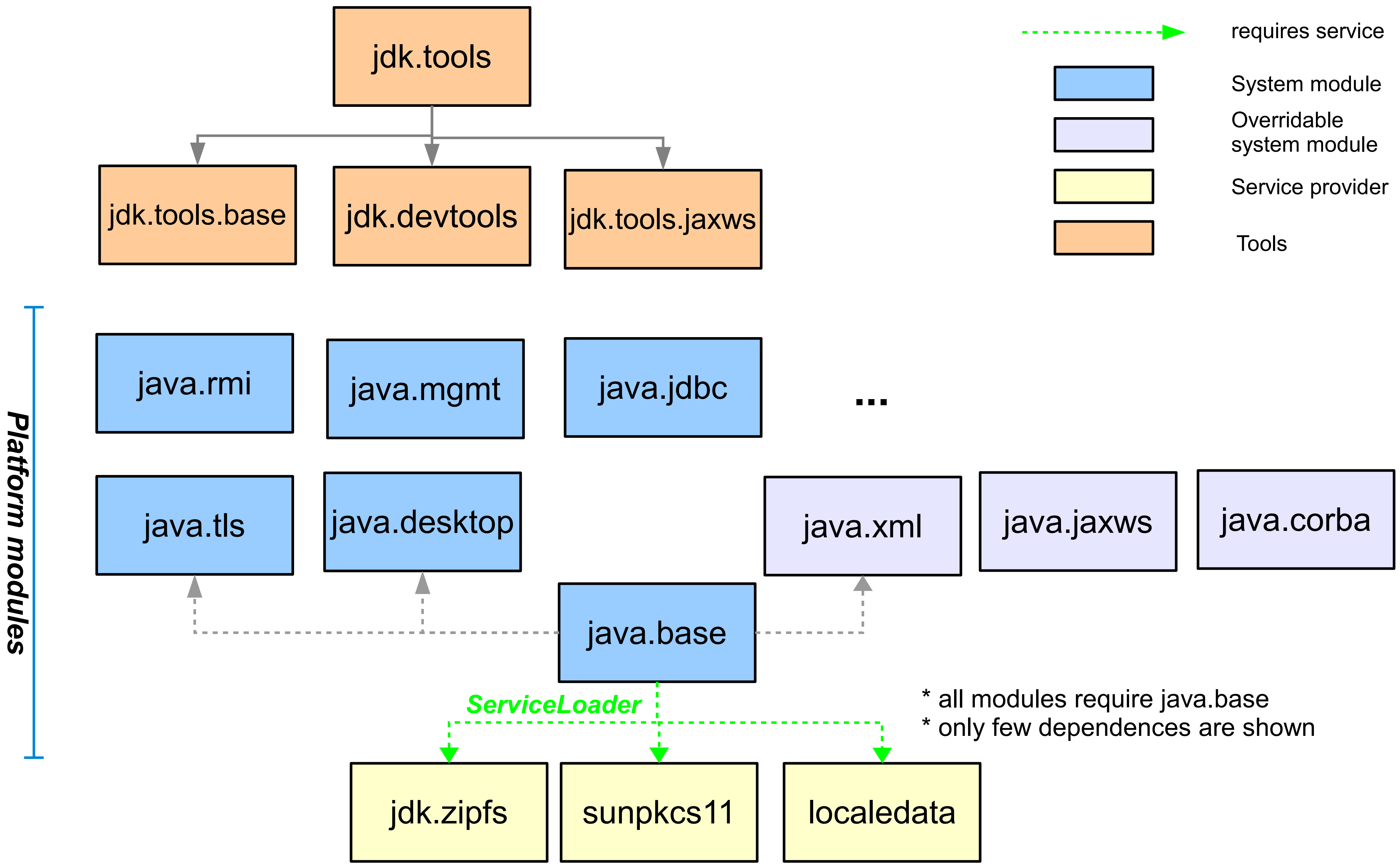
Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Terminology

Traditional terminology	**Traditional location**	**Traditional permissions**	**Traditional loader**	**Modular terminology**	**Module location**
Bootstrap classes	Classes loaded from rt.jar and other bootstrap search locations.	System domain granting all permissions	*Bootstrap class loader*	*System modules* ** includes Java SE API and JDK-specific classes*	*TBD: One version of a given system module per library in JDK 8 to reduce the scope of the release.*
Extension classes	Classes loaded from the "lib/ext" directory of the JRE or the system-wide platform-specific extension directory through the extension mechanism.	Permissions are configurable and the default policy is to grant all permissions.	Extension class loader	Normal modules	Multiple versions of a normal module per library.
Tools classes	Classes loaded from JAR files in the JDK's "lib" directory, notably tools.jar.	Permissions based on user-defined policy.	System class loader	Normal modules	Multiple versions of a normal module per library.
Endorsed Standards & Standalone Technologies	Bootstrap classes that can be overridden by a newer version of a standard defined by the Java SE platform (e.g. CORBA, JAXP, JAXB).	System domain granting all permissions.	Bootstrap class loader	*Overridable system modules*	Multiple versions of an overridable system module per library.

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Modular JDK



* all modules require java.base
* only few dependences are shown

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Implementation: Bootstrapping

- Launcher passes module query + library to VM
- VM uses jigsaw's native library to:
  - Read the configuration of the module matching the module query
  - Find the java.base module required by the config
  - Preload primordial classes (e.g. java.lang.Object + *core* module system classes) with VM bootstrap loader
    - Not configurable by -Xbootclasspath
- VM initializes the module system
  - Create base module loader
  - Load *non-core* module system classes with a loader...

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Which loader?

- Option 1 ("Split bootstrapping")
  - VM bootstrap loader loads primordial classes and core module system classes
  - Base module loader loads non-core module system classes and all other classes in the base module
- Minimizes number of classes loaded by bootstrap loader
- VM needs to maintain a list of primordial + core module system classes
- Prototyped and discarded because it's error-prone and there is no robust way to determine that list
- Hard to detect and diagnose errors when a *core* module system class starts to depend on a *non-core* module system class

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Which loader?

- Option 2 (“Unified bootstrapping”)
  - VM bootstrap loader loads *all* classes from the base module
  - Base module loader still exists, but only used when the VM bootstrap loader delegates to it to load optional dependencies, service providers, and resource bundles
- Less error-prone than Option 1
- Benefits from existing VM optimizations for the base module
  - CDS, null initiating loader, …

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

What should Class.getClassLoader() return for the base module's classes?

- The same value should be returned for *all* classes in the base module, regardless of whether the base module's classes are loaded by split bootstrapping or unified bootstrapping
- Option A
  - Return a module class loader for the base module
  - Simplify access to loaders and resources
    - Class.getClassLoader v. ClassLoader.getSystemClassLoader
    - ClassLoader.getResource v. ClassLoader.getSystemResource
    - ClassLoader.findClass v. ClassLoader.findSystemClass
- Option B
  - Return null for behavioral compatibility

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Bootstrap classes

	Classpath mode	**Module mode**
Classes in the base module loaded by	VM built-in bootstrap loader (null)	VM built-in bootstrap loader (null)
All other bootstrap classes loaded by	VM built-in bootstrap loader (null)	ModuleClassLoader
Protection domain / Permissions	Grant all permissions	Grant all permissions
CodeSource that can be identified as system protection domain	null	module URL (TBD)
Class.getClassLoader()	null	ModuleClassLoader TBD for the base
Visibility of JDK-internal public classes (non-exported types)	Runtime: allowed Compile-time: ct.sym	No visibility of JDK-internal public classes
Visibility of the bootstrap classes	All bootstrap classes on the bootclasspath	Only exported types from the modules specified in the module dependencies

Copyright © 2012 Oracle and/or its affiliates · All rights reserved

Open Issues

- Revisit methods related to system class loader e.g.
  - ClassLoader.getSystemClassLoader,
  - ClassLoader.getSystemResource and relevant methods
- Revisit some ClassLoader methods for modules
  - definePackage and getPackage(s) that are tied with JAR's Manifest
- Permission required for retrieving a ClassLoader
  - getParent, getClassLoader, etc that traditionally assumes the hierarchical delegation model
- JDK areas to be updated with modules such as serialization, RMI, CORBA, JMX, etc
- Testing depending on JDK-internal classes

Copyright © 2012 Oracle and/or its affiliates · All rights reserved