Package java.lang.instrument

Provides services that allow Java programming language agents to instrument programs running on the JVM.

See: Description

Interface Summary

Interface	Description
ClassFileTransformer	An agent provides an implementation of this interface in order to transform class files.
Instrumentation	This class provides services needed to instrument Java programming language code.

Class Summary

Class	Description
ClassDefinition	This class serves as a parameter block to the `Instrumentation.redefineClasses` method.

Exception Summary

Exception	Description
IllegalClassFormatException	Thrown by an implementation of `ClassFileTransformer.transform` when its input parameters are invalid.
UnmodifiableClassException	Thrown by an implementation of `Instrumentation.redefineClasses` when one of the specified classes cannot be modified.

Package java.lang.instrument Description

Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.

Package Specification

An agent is deployed as a JAR file. An attribute in the JAR file manifest specifies

the agent class which will be loaded to start the agent. For implementations that support a command-line interface, an agent is started by specifying an option on the command-line. Implementations may also support a mechanism to start agents some time after the VM has started. For example, an implementation may provide a mechanism that allows a tool to *attach* to a running application, and initiate the loading of the tool's agent into the running application. The details as to how the load is initiated, is implementation dependent.

Command-Line Interface

An implementation is not required to provide a way to start agents from the command-line interface. On implementations that do provide a way to start agents from the command-line interface, an agent is started by adding this option to the command-line:

`-javaagent:`*jarpath[`**`=`***options]*

jarpath is the path to the agent JAR file. *options* is the agent options. This switch may be used multiple times on the same command-line, thus creating multiple agents. More than one agent may use the same *jarpath*. An agent JAR file must conform to the JAR file specification.

The manifest of the agent JAR file must contain the attribute `Premain-Class`. The value of this attribute is the name of the *agent class*. The agent class must implement a public static `premain` method similar in principle to the `main` application entry point. After the Java Virtual Machine (JVM) has initialized, each `premain` method will be called in the order the agents were specified, then the real application `main` method will be called. Each `premain` method must return in order for the startup sequence to proceed.

The `premain` method has one of two possible signatures. The JVM first attempts to invoke the following method on the agent class:

```
public static void premain(String agentArgs,
Instrumentation inst);
```

If the agent class does not implement this method then the JVM will attempt to invoke:

```
public static void premain(String agentArgs);
```

The agent class may also have an `agentmain` method for use when the agent is started after VM startup. When the agent is started using a command-line option, the `agentmain` method is not invoked.

The agent class will be loaded by the system class loader (see `ClassLoader.getSystemClassLoader`). This is the class loader which typically loads the class containing the application `main` method. The `premain` methods will be run under the same security and classloader rules as the application `main` method. There are no modeling restrictions on what the agent `premain` method may do. Anything application `main` can do, including creating threads, is legal from `premain`.

Each agent is passed its agent options via the `agentArgs` parameter. The agent options are passed as a single string, any additional parsing should be performed by the agent itself.

If the agent cannot be resolved (for example, because the agent class cannot be loaded, or because the agent class does not have an appropriate `premain` method), the JVM will abort. If a `premain` method throws an uncaught exception, the JVM will abort.

Starting Agents After VM Startup

An implementation may provide a mechanism to start agents sometime after the the VM has started. The details as to how this is initiated are implementation specific but typically the application has already started and its `main` method has already been invoked. In cases where an implementation supports the starting of agents after the VM has started the following applies:

1. The manifest of the agent JAR must contain the attribute `Agent-Class`. The value of this attribute is the name of the *agent class*.
2. The agent class must implement a public static `agentmain` method.
3. The system class loader (`ClassLoader.getSystemClassLoader`) must support a mechanism to add an agent JAR file to the system class path.

The agent JAR is appended to the system class path. This is the class loader that typically loads the class containing the application `main` method. The agent class is loaded and the JVM attempts to invoke the `agentmain` method. The JVM first attempts to invoke the following method on the agent class:

```
public static void agentmain(String agentArgs,
Instrumentation inst);
```

If the agent class does not implement this method then the JVM will attempt to invoke:

```
public static void agentmain(String agentArgs);
```

The agent class may also have an `premain` method for use when the agent is started using a command-line option. When the agent is started after VM startup the `premain` method is not invoked.

The agent is passed its agent options via the `agentArgs` parameter. The agent options are passed as a single string, any additional parsing should be performed by the agent itself.

The `agentmain` method should do any necessary initialization required to start the agent. When startup is complete the method should return. If the agent cannot be started (for example, because the agent class cannot be loaded, or because the agent class does not have a conformant `agentmain` method), the JVM will not abort. If the `agentmain` method throws an uncaught exception it will be ignored.

Manifest Attributes

The following manifest attributes are defined for an agent JAR file:

`Premain-Class`
: When an agent is specified at JVM launch time this attribute specifies the agent class. That is, the class containing the `premain` method. When an agent is specified at JVM launch time this attribute is required. If the attribute is not present the JVM will abort. Note: this is a class name, not a file name or path.

`Agent-Class`
: If an implementation supports a mechanism to start agents sometime after the VM has started then this attribute specifies the agent class. That is, the class containing the `agentmain` method. This attribute is required, if it is not present the agent will not be started. Note: this is a class name, not a file name or path.

`Boot-Class-Path`
: A list of paths to be searched by the bootstrap class loader. Paths represent directories or libraries (commonly referred to as JAR or zip libraries on many platforms). These paths are searched by the bootstrap class loader after the platform specific mechanisms of locating a class have failed. Paths are searched in the order listed. Paths in the list are separated by one or more spaces. A path takes the syntax of the path component of a hierarchical URI. The path is absolute if it begins with a slash character ('/'), otherwise it is relative. A relative path is resolved against the absolute path of the agent JAR file. Malformed and non-existent paths are ignored. When an agent is started sometime after the VM has started then paths that do not represent a JAR file are ignored. This attribute is optional.

`Can-Redefine-Classes`
: Boolean (`true` or `false`, case irrelevant). Is the ability to redefine classes needed by this agent. Values other than `true` are considered `false`. This attribute is optional, the default is `false`.

`Can-Retransform-Classes`
: Boolean (`true` or `false`, case irrelevant). Is the ability to retransform classes needed by this agent. Values other than `true` are considered `false`. This attribute is optional, the default is `false`.

`Can-Set-Native-Method-Prefix`
: Boolean (`true` or `false`, case irrelevant). Is the ability to set native method prefix needed by this agent. Values other than `true` are considered `false`. This attribute is optional, the default is `false`.

An agent JAR file may have both the `Premain-Class` and `Agent-Class` attributes present in the manifest. When the agent is started on the command-line using the `-javaagent` option then the `Premain-Class` attribute specifies the name of the agent class and the `Agent-Class` attribute is ignored. Similarly, if the agent is started sometime after the VM has started, then the `Agent-Class`

attribute specifies the name of the agent class (the value of `Premain-Class` attribute is ignored).

Related Documentation

For tool documentation, please see:

- JDK Tools and Utilities

Since:
JDK1.5