java.lang

Class RuntimePermission

java.lang.Object
  java.security.Permission
    java.security.BasicPermission
      java.lang.RuntimePermission

All Implemented Interfaces:

Serializable, Guard

```
public final class RuntimePermission
extends BasicPermission
```

This class is for runtime permissions. A RuntimePermission contains a name (also referred to as a "target name") but no actions list; you either have the named permission or you don't.

The target name is the name of the runtime permission (see below). The naming convention follows the hierarchical property naming convention. Also, an asterisk may appear at the end of the name, following a ".", or by itself, to signify a wildcard match. For example: "loadLibrary.*" or "*" is valid, "*loadLibrary" or "a*b" is not valid.

The following table lists all the possible RuntimePermission target names, and for each provides a description of what the permission allows and a discussion of the risks of granting code the permission.

Permission Target Name	What the Permission Allows	Risks of Allowing this Permission
createClassLoader	Creation of a class loader	This is an extremely dangerous permission to grant. Malicious applications that can instantiate their own class loaders could then load their own rogue classes into the system. These newly loaded classes could be placed into any protection domain by the class loader, thereby automatically granting the classes the permissions for that domain.
getClassLoader	Retrieval of a class loader (e.g., the class loader for the calling class)	This would grant an attacker permission to get the class loader for a particular class. This is dangerous because having access to a class's class loader allows the attacker to load other classes available to that class loader. The attacker would typically otherwise not have access to those classes.
setContextClassLoader	Setting of the context class loader used by a thread	The context class loader is used by system code and extensions when they need to lookup resources that might not exist in the system class loader. Granting setContextClassLoader permission would allow code to change which context class loader is used for a particular thread, including system threads.
enableContextClassLoaderOverride	Subclass implementation of the thread context class loader methods	The context class loader is used by system code and extensions when they need to lookup resources that might not exist in the system class loader. Granting enableContextClassLoaderOverride permission would allow a subclass of Thread to override the methods that are used to get or set the context class loader for a particular thread.

closeClassLoader	Closing of a ClassLoader	Granting this permission allows code to close any URLClassLoader that it has a reference to.
setSecurityManager	Setting of the security manager (possibly replacing an existing one)	The security manager is a class that allows applications to implement a security policy. Granting the setSecurityManager permission would allow code to change which security manager is used by installing a different, possibly less restrictive security manager, thereby bypassing checks that would have been enforced by the original security manager.
createSecurityManager	Creation of a new security manager	This gives code access to protected, sensitive methods that may disclose information about other classes or the execution stack.
getenv.{variable name}	Reading of the value of the specified environment variable	This would allow code to read the value, or determine the existence, of a particular environment variable. This is dangerous if the variable contains confidential data.
exitVM.{exit status}	Halting of the Java Virtual Machine with the specified exit status	This allows an attacker to mount a denial-of-service attack by automatically forcing the virtual machine to halt. Note: The "exitVM.*" permission is automatically granted to all code loaded from the application class path, thus enabling applications to terminate themselves. Also, the "exitVM" permission is equivalent to "exitVM.*".
shutdownHooks	Registration and cancellation of virtual-machine shutdown hooks	This allows an attacker to register a malicious shutdown hook that interferes with the clean shutdown of the virtual machine.
setFactory	Setting of the socket factory used by ServerSocket or Socket, or of the stream handler factory used by URL	This allows code to set the actual implementation for the socket, server socket, stream handler, or RMI socket factory. An attacker may set a faulty implementation which mangles the data stream.
setIO	Setting of System.out, System.in, and System.err	This allows changing the value of the standard system streams. An attacker may change System.in to monitor and steal user input, or may set System.err to a "null" OutputStream, which would hide any error messages sent to System.err.
modifyThread	Modification of threads, e.g., via calls to Thread `interrupt`, `stop`, `suspend`, `resume`, `setDaemon`, `setPriority`, `setName` and `setUncaughtExceptionHandler` methods	This allows an attacker to modify the behaviour of any thread in the system.
stopThread	Stopping of threads via calls to the Thread `stop` method	This allows code to stop any thread in the system provided that it is already granted permission to access that thread. This poses as a threat, because that code may corrupt the system by killing existing threads.
modifyThreadGroup	modification of thread groups, e.g., via calls to ThreadGroup `destroy`, `getParent`, `resume`, `setDaemon`, `setMaxPriority`, `stop`, and `suspend` methods	This allows an attacker to create thread groups and set their run priority.
getProtectionDomain	Retrieval of the ProtectionDomain for a class	This allows code to obtain policy information for a particular code source. While obtaining policy information does not compromise the security of the system, it does give attackers additional information, such as local file names for example, to better aim an attack.
getFileSystemAttributes	Retrieval of file system attributes	This allows code to obtain file system information such as disk usage or disk space available to the

		caller. This is potentially dangerous because it discloses information about the system hardware configuration and some information about the caller's privilege to write files.
readFileDescriptor	Reading of file descriptors	This would allow code to read the particular file associated with the file descriptor read. This is dangerous if the file contains confidential data.
writeFileDescriptor	Writing to file descriptors	This allows code to write to a particular file associated with the descriptor. This is dangerous because it may allow malicious code to plant viruses or at the very least, fill up your entire disk.
loadLibrary.{library name}	Dynamic linking of the specified library	It is dangerous to allow an applet permission to load native code libraries, because the Java security architecture is not designed to and does not prevent malicious behavior at the level of native code.
accessClassInPackage.{package name}	Access to the specified package via a class loader's `loadClass` method when that class loader calls the SecurityManager `checkPackageAccess` method	This gives code access to classes in packages to which it normally does not have access. Malicious code may use these classes to help in its attempt to compromise security in the system.
defineClassInPackage.{package name}	Definition of classes in the specified package, via a class loader's `defineClass` method when that class loader calls the SecurityManager `checkPackageDefinition` method.	This grants code permission to define a class in a particular package. This is dangerous because malicious code with this permission may define rogue classes in trusted packages like `java.security` or `java.lang`, for example.
accessDeclaredMembers	Access to the declared members of a class	This grants code permission to query a class for its public, protected, default (package) access, and private fields and/or methods. Although the code would have access to the private and protected field and method names, it would not have access to the private/protected field data and would not be able to invoke any private methods. Nevertheless, malicious code may use this information to better aim an attack. Additionally, it may invoke any public methods and/or access public fields in the class. This could be dangerous if the code would normally not be able to invoke those methods and/or access the fields because it can't cast the object to the class/interface with those methods and fields.
queuePrintJob	Initiation of a print job request	This could print sensitive information to a printer, or simply waste paper.
getStackTrace	Retrieval of the stack trace information of another thread.	This allows retrieval of the stack trace information of another thread. This might allow malicious code to monitor the execution of threads and discover vulnerabilities in applications.
setDefaultUncaughtExceptionHandler	Setting the default handler to be used when a thread terminates abruptly due to an uncaught exception	This allows an attacker to register a malicious uncaught exception handler that could interfere with termination of a thread
preferences	Represents the permission required to get access to the java.util.prefs.Preferences implementations user or system root which in turn allows retrieval or update operations within the Preferences persistent backing store.)	This permission allows the user to read from or write to the preferences backing store if the user running the code has sufficient OS privileges to read/write to that backing store. The actual backing store may reside within a traditional filesystem directory or within a registry depending on the platform OS
usePolicy	Granting this permission	For more information, refer to Java Plug-In's

	disables the Java Plug-In's default security prompting behavior.	guides, Applet Security Basics and usePolicy Permission.

See Also:

BasicPermission, Permission, Permissions, PermissionCollection, SecurityManager, Serialized Form

Constructor Summary

Constructors

Constructor and Description
`RuntimePermission(String name)` Creates a new RuntimePermission with the specified name.
`RuntimePermission(String name, String actions)` Creates a new RuntimePermission object with the specified name.

Method Summary

Methods inherited from class java.security.BasicPermission

equals, getActions, hashCode, implies, newPermissionCollection

Methods inherited from class java.security.Permission

checkGuard, getName, toString

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

RuntimePermission

```
public RuntimePermission(String name)
```

Creates a new RuntimePermission with the specified name. The name is the symbolic name of the RuntimePermission, such as "exit", "setFactory", etc. An asterisk may appear at the end of the name, following a ".", or by itself, to signify a wildcard match.

Parameters:

`name` - the name of the RuntimePermission.

Throws:

`NullPointerException` - if `name` is `null`.

`IllegalArgumentException` - if `name` is empty.

RuntimePermission

```
public RuntimePermission(String name,
                         String actions)
```

Creates a new RuntimePermission object with the specified name. The name is the symbolic name of the RuntimePermission, and the actions String is currently unused and should be null.

Parameters:

`name` - the name of the RuntimePermission.

`actions` - should be null.

Throws:

`NullPointerException` - if `name` is `null`.

`IllegalArgumentException` - if `name` is empty.