java.io

Class FilePermission

java.lang.Object
  java.security.Permission
    java.io.FilePermission

All Implemented Interfaces:

Serializable, Guard

```
public final class FilePermission
extends Permission
implements Serializable
```

This class represents access to a file or directory. A FilePermission consists of a pathname and a set of actions valid for that pathname.

Pathname is the pathname of the file or directory granted the specified actions. A pathname that ends in "/*" (where "/" is the file separator character, `File.separatorChar`) indicates all the files and directories contained in that directory. A pathname that ends with "/-" indicates (recursively) all files and subdirectories contained in that directory. A pathname consisting of the special token "<<ALL FILES>>" matches **any** file.

Note: A pathname consisting of a single "*" indicates all the files in the current directory, while a pathname consisting of a single "-" indicates all the files in the current directory and (recursively) all files and subdirectories contained in the current directory.

The actions to be granted are passed to the constructor in a string containing a list of one or more comma-separated keywords. The possible keywords are "read", "write", "execute", "delete", and "readlink". Their meaning is defined as follows:

read

read permission

write

write permission

execute

execute permission. Allows `Runtime.exec` to be called. Corresponds to `SecurityManager.checkExec`.

delete

delete permission. Allows `File.delete` to be called. Corresponds to `SecurityManager.checkDelete`.

readlink

read link permission. Allows the target of a symbolic link to be read by invoking the `readSymbolicLink` method.

The actions string is converted to lowercase before processing.

Be careful when granting FilePermissions. Think about the implications of granting read and especially write access to various files and directories. The "<<ALL FILES>>" permission with write action is especially dangerous. This grants permission to write to the entire file system. One thing this effectively allows is replacement of the system binary, including the JVM runtime environment.

Please note: Code can always read a file from the same directory it's in (or a subdirectory of that directory); it does not need explicit permission to do so.

Since:

1.2

See Also:

Permission, Permissions, PermissionCollection

Constructor Summary

Constructors

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

Method Summary

Methods

Modifier and Type	Method and Description
`boolean`	`equals(Object obj)` Checks two FilePermission objects for equality.
`String`	`getActions()` Returns the "canonical string representation" of the actions.
`int`	`hashCode()` Returns the hash code value for this object.
`boolean`	`implies(Permission p)` Checks if this FilePermission object "implies" the specified permission.
`PermissionCollection`	`newPermissionCollection()` Returns a new PermissionCollection object for storing FilePermission objects.

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

FilePermission

```
public FilePermission(String path,
              String actions)
```

Creates a new FilePermission object with the specified actions. *path* is the pathname of a file or directory, and *actions* contains a comma-separated list of the desired actions granted on the file or directory. Possible actions are "read", "write", "execute", "delete", and "readlink".

A pathname that ends in "/*" (where "/" is the file separator character, `File.separatorChar`) indicates all the files and directories contained in that directory. A pathname that ends with "/-" indicates (recursively) all files and subdirectories contained in that directory. The special pathname "<<ALL FILES>>" matches any file.

A pathname consisting of a single "*" indicates all the files in the current directory, while a pathname consisting of a single "-" indicates all the files in the current directory and (recursively) all files and subdirectories contained in the current directory.

A pathname containing an empty string represents an empty path.

Parameters:

`path` - the pathname of the file/directory.

`actions` - the action string.

Throws:

`IllegalArgumentException` - If actions is `null`, empty or contains an action other than the specified possible actions.

Method Detail

implies

`public boolean implies(Permission p)`

Checks if this FilePermission object "implies" the specified permission.

More specifically, this method returns true if:

- *p* is an instanceof FilePermission,
- *p*'s actions are a proper subset of this object's actions, and
- *p*'s pathname is implied by this object's pathname. For example, "/tmp/*" implies "/tmp/foo", since "/tmp/*" encompasses all files in the "/tmp" directory, including the one named "foo".

Specified by:

`implies` in class `Permission`

Parameters:

`p` - the permission to check against.

Returns:

`true` if the specified permission is not `null` and is implied by this object, `false` otherwise.

equals

`public boolean equals(Object obj)`

Checks two FilePermission objects for equality. Checks that *obj* is a FilePermission, and has the same pathname and actions as this object.

Specified by:

`equals` in class `Permission`

Parameters:

`obj` - the object we are testing for equality with this object.

Returns:

`true` if obj is a FilePermission, and has the same pathname and actions as this FilePermission object, `false` otherwise.

See Also:

`Object.hashCode()`, `HashMap`

hashCode

```
public int hashCode()
```

Returns the hash code value for this object.

Specified by:

`hashCode` in class `Permission`

Returns:

a hash code value for this object.

See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

getActions

```
public String getActions()
```

Returns the "canonical string representation" of the actions. That is, this method always returns present actions in the following order: read, write, execute, delete, readlink. For example, if this FilePermission object allows both write and read actions, a call to `getActions` will return the string "read,write".

Specified by:

`getActions` in class `Permission`

Returns:

the canonical string representation of the actions.

newPermissionCollection

```
public PermissionCollection newPermissionCollection()
```

Returns a new PermissionCollection object for storing FilePermission objects.

FilePermission objects must be stored in a manner that allows them to be inserted into the collection in any order, but that also enables the PermissionCollection `implies` method to be implemented in an efficient (and consistent) manner.

For example, if you have two FilePermissions:

1. `"/tmp/-", "read"`
2. `"/tmp/scratch/foo", "write"`

and you are calling the `implies` method with the FilePermission:

```
"/tmp/scratch/foo", "read,write",
```

then the `implies` function must take into account both the "/tmp/-" and "/tmp/scratch/foo" permissions, so the effective permission is "read,write", and `implies` returns true. The "implies" semantics for FilePermissions are handled properly by the PermissionCollection object returned by this `newPermissionCollection` method.

Overrides:

`newPermissionCollection` in class `Permission`

Returns:

a new PermissionCollection object suitable for storing FilePermissions.