Manipulating Files and Directories ← Links, Symbolics and Otherwise → Managing Files Attributes

Links, Symbolics and Otherwise

As mentioned previously, the `java.nio.file` package, and the `Path` class in particular, is "link aware." Every `Path` method either detects what to do when a symbolic link is encountered, or it provides an option enabling you to configure the behavior when a symbolic link is encountered.

The discussion so far has been about symbolic or soft links, but some file systems also support hard links. Hard links are more restrictive than symbolic links, as follows:

- The target of the link must exist.
- Hard links are generally not allowed on directories.
- Hard links are not allowed to cross partitions or volumes. Therefore, they cannot exist across file systems.
- A hard link looks, and behaves, like a regular file, so they can be hard to find.
- A hard link is, for all intents and purposes, the same entity as the original file. They have the same file permissions, time stamps, and so on. All attributes are identical.

Because of these restrictions, hard links are not used as often as symbolic links, but the `Path` methods work seamlessly with hard links.

Creating a Symbolic Link

If your file system supports it, you can create a symbolic link by using the `createSymbolicLink(Path, Path, FileAttribute)` method. The second `Path` argument represents the target file or directory and might or might not exist. The following code snippet creates a symbolic link with default permissions:

```
Path newLink = ...;
Path target = ...;
try {
    Files.createSymbolicLink(newLink, target);
} catch (IOException x) {
    System.err.println(x);
} catch (UnsupportedOperationException x) {
    // Some file systems do not support symbolic links.
    System.err.println(x);
}
```

The `FileAttribute` vararg enables you to specify initial file attributes that are set atomically when the link is created. However, this argument is intended for future use and is not currently implemented.

Creating a Hard Link

You can create a hard (or regular) link to an existing file by using the `createLink(Path, Path)` method. The second `Path` argument locates the existing file, and it must exist or a `NoSuchFileException` is thrown. The following code snippet shows how to create a link:

```
Path newLink = ...;
Path existingFile = ...;
try {
    Files.createLink(newLink, existingFile);
} catch (IOException x) {
    System.err.println(x);
} catch (UnsupportedOperationException x) {
    // Some file systems do not
    // support adding an existing
    // file to a directory.
    System.err.println(x);
}
```

Detecting a Symbolic Link

To determine whether a `Path` instance is a symbolic link, you can use the `isSymbolicLink(Path)` method. The following code snippet shows how:

```
Path file = ...;
boolean isSymbolicLink =
    Files.isSymbolicLink(file);
```

For more information, see the section Managing Metadata.

Finding the Target of a Link

You can obtain the target of a symbolic link by using the `readSymbolicLink(Path)` method, as follows:

```
Path link = ...;
try {
    System.out.format("Target of link" +
        " '%s' is '%s'%n", link,
        Files.readSymbolicLink(link));
} catch (IOException x) {
    System.err.println(x);
    }
```

If the `Path` is not a symbolic link, this method throws a `NotLinkException`.

Last update: *January 25, 2023*

Manipulating Files and Directories ← **Links, Symbolics and Otherwise** → Managing Files Attributes

Home > Tutorials > The Java I/O API > File System Basics > Links, Symbolics and Otherwise