The Java™ Tutorials

Trail: Essential Classes
Lesson: Basic I/O
Section: File I/O (Featuring NIO.2)

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Creating and Reading Directories

Some of the methods previously discussed, such as `delete`, work on files, links *and* directories. But how do you list all the directories at the top of a file system? How do you list the contents of a directory or create a directory?

This section covers the following functionality specific to directories:

Listing a File System's Root Directories

You can list all the root directories for a file system by using the `FileSystem.getRootDirectories` method. This method returns an `Iterable`, which enables you to use the enhanced for statement to iterate over all the root directories.

The following code snippet prints the root directories for the default file system:

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name: dirs) {
    System.err.println(name);
}
```

Creating a Directory

You can create a new directory by using the `createDirectory(Path, FileAttribute<?>)` method. If you don't specify any `FileAttributes`, the new directory will have default attributes. For example:

```
Path dir = ...;
Files.createDirectory(path);
```

The following code snippet creates a new directory on a POSIX file system that has specific permissions:

```
Set<PosixFilePermission> perms =
    PosixFilePermissions.fromString("rwxr-x---");
FileAttribute<Set<PosixFilePermission>> attr =
    PosixFilePermissions.asFileAttribute(perms);
Files.createDirectory(file, attr);
```

To create a directory several levels deep when one or more of the parent directories might not yet exist, you can use the convenience method, `createDirectories(Path, FileAttribute<?>)`. As with the `createDirectory(Path, FileAttribute<?>)` method, you can specify an optional set of initial file attributes. The following code snippet uses default attributes:

```
Files.createDirectories(Paths.get("foo/bar/test"));
```

The directories are created, as needed, from the top down. In the `foo/bar/test` example, if the `foo` directory does not exist, it is created. Next, the `bar` directory is created, if needed, and, finally, the `test` directory is created.

It is possible for this method to fail after creating some, but not all, of the parent directories.

Creating a Temporary Directory

You can create a temporary directory using one of `createTempDirectory` methods:

- `createTempDirectory(Path, String, FileAttribute<?>...)`
- `createTempDirectory(String, FileAttribute<?>...)`

The first method allows the code to specify a location for the temporary directory and the second method creates a new directory in the default temporary-fle directory.

Listing a Directory's Contents

You can list all the contents of a directory by using the `newDirectoryStream(Path)` method. This method returns an object that implements the `DirectoryStream` interface. The class that implements the `DirectoryStream` interface also implements `Iterable`, so you can iterate through the directory stream, reading all of the objects. This approach scales well to very large directories.

Remember: The returned `DirectoryStream` is a *stream*. If you are not using a `try`-with-resources statement, don't forget to close the stream in the `finally` block. The `try`-with-resources statement takes care of this for you.

The following code snippet shows how to print the contents of a directory:

```
Path dir = ...;
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path file: stream) {
        System.out.println(file.getFileName());
    }
} catch (IOException | DirectoryIteratorException x) {
    // IOException can never be thrown by the iteration.
    // In this snippet, it can only be thrown by newDirectoryStream.
    System.err.println(x);
}
```

The `Path` objects returned by the iterator are the names of the entries resolved against the directory. So, if you are listing the contents of the `/tmp` directory, the entries are returned with the form `/tmp/a`, `/tmp/b`, and so on.

This method returns the entire contents of a directory: files, links, subdirectories, and hidden files. If you want to be more selective about the contents that are retrieved, you can use one of the other `newDirectoryStream` methods, as described later in this page.

Note that if there is an exception during directory iteration then `DirectoryIteratorException` is thrown with the `IOException` as the cause. Iterator methods cannot throw exception exceptions.

Filtering a Directory Listing By Using Globbing

If you want to fetch only files and subdirectories where each name matches a particular pattern, you can do so by using the `newDirectoryStream(Path, String)` method, which provides a built-in glob filter. If you are not familiar with glob syntax, see What Is a Glob?

For example, the following code snippet lists files relating to Java: *.class*, *.java*, and *.jar* files.:

```
Path dir = ...;
try (DirectoryStream<Path> stream =
     Files.newDirectoryStream(dir, "*.{java,class,jar}")) {
    for (Path entry: stream) {
        System.out.println(entry.getFileName());
    }
} catch (IOException x) {
    // IOException can never be thrown by the iteration.
    // In this snippet, it can // only be thrown by newDirectoryStream.
    System.err.println(x);
}
```

Writing Your Own Directory Filter

Perhaps you want to filter the contents of a directory based on some condition other than pattern matching. You can create your own filter by implementing the `DirectoryStream.Filter<T>` interface. This interface consists of one method, `accept`, which determines whether a file fulfills the search requirement.

For example, the following code snippet implements a filter that retrieves only directories:

```
DirectoryStream.Filter<Path> filter =
    newDirectoryStream.Filter<Path>() {
    public boolean accept(Path file) throws IOException {
        try {
            return (Files.isDirectory(path));
        } catch (IOException x) {
            // Failed to determine if it's a directory.
            System.err.println(x);
            return false;
        }
```

```
    }
};
```

Once the filter has been created, it can be invoked by using the `newDirectoryStream(Path, DirectoryStream.Filter<? super Path>)` method. The following code snippet uses the `isDirectory` filter to print only the directory's subdirectories to standard output:

```
Path dir = ...;
try (DirectoryStream<Path>
                     stream = Files.newDirectoryStream(dir, filter)) {
    for (Path entry: stream) {
        System.out.println(entry.getFileName());
    }
} catch (IOException x) {
    System.err.println(x);
}
```

This method is used to filter a single directory only. However, if you want to find all the subdirectories in a file tree, you would use the mechanism for Walking the File Tree.

Previous page: Random Access Files
Next page: Links, Symbolic or Otherwise