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.

Finding Files

If you have ever used a shell script, you have most likely used pattern matching to locate files. In fact, you have probably used it extensively. If you haven't used it, pattern matching uses special characters to create a pattern and then file names can be compared against that pattern. For example, in most shell scripts, the asterisk, `*`, matches any number of characters. For example, the following command lists all the files in the current directory that end in `.html`:

```
% ls *.html
```

The `java.nio.file` package provides programmatic support for this useful feature. Each file system implementation provides a `PathMatcher`. You can retrieve a file system's `PathMatcher` by using the `getPathMatcher(String)` method in the `FileSystem` class. The following code snippet fetches the path matcher for the default file system:

```
String pattern = ...;
PathMatcher matcher =
    FileSystems.getDefault().getPathMatcher("glob:" + pattern);
```

The string argument passed to `getPathMatcher` specifies the syntax flavor and the pattern to be matched. This example specifies *glob* syntax. If you are unfamiliar with glob syntax, see What is a Glob.

Glob syntax is easy to use and flexible but, if you prefer, you can also use regular expressions, or *regex*, syntax. For further information about regex, see the Regular Expressions lesson. Some file system implementations might support other syntaxes.

If you want to use some other form of string-based pattern matching, you can create your own `PathMatcher` class. The examples in this page use glob syntax.

Once you have created your `PathMatcher` instance, you are ready to match files against it. The `PathMatcher` interface has a single method, `matches`, that takes a `Path` argument and returns a boolean: It either matches the pattern, or it does not. The following code snippet looks for files that end in `.java` or `.class` and prints those files to standard output:

```
PathMatcher matcher =
    FileSystems.getDefault().getPathMatcher("glob:*.{java,class}");

Path filename = ...;
if (matcher.matches(filename)) {
    System.out.println(filename);
}
```

Recursive Pattern Matching

Searching for files that match a particular pattern goes hand-in-hand with walking a file tree. How many times do you know a file is *somewhere* on the file system, but where? Or perhaps you need to find all files in a file tree that have a particular file extension.

The `Find` example does precisely that. `Find` is similar to the UNIX `find` utility, but has pared down functionally. You can extend this example to include other functionality. For example, the `find` utility supports the `-prune` flag to exclude an entire subtree from the search. You could implement that functionality by returning `SKIP_SUBTREE` in the `preVisitDirectory` method. To implement the `-L` option, which follows symbolic links, you could use the four-argument `walkFileTree` method and pass in the `FOLLOW_LINKS` enum (but make sure that you test for circular links in the `visitFile` method).

To run the Find application, use the following format:

```
% java Find <path> -name "<glob_pattern>"
```

The pattern is placed inside quotation marks so any wildcards are not interpreted by the shell. For example:

```
% java Find . -name "*.html"
```

Here is the source code for the `Find` example:

```
/**
 * Sample code that finds files that match the specified glob pattern.
 * For more information on what constitutes a glob pattern, see
 * https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#glob
 *
 * The file or directories that match the pattern are printed to
 * standard out.  The number of matches is also printed.
 *
 * When executing this application, you must put the glob pattern
 * in quotes, so the shell will not expand any wild cards:
 *              java Find . -name "*.java"
 */

import java.io.*;
import java.nio.file.*;
import java.nio.file.attribute.*;
import static java.nio.file.FileVisitResult.*;
import static java.nio.file.FileVisitOption.*;
import java.util.*;


public class Find {

    public static class Finder
        extends SimpleFileVisitor<Path> {

        private final PathMatcher matcher;
        private int numMatches = 0;

        Finder(String pattern) {
            matcher = FileSystems.getDefault()
                    .getPathMatcher("glob:" + pattern);
        }

        // Compares the glob pattern against
        // the file or directory name.
        void find(Path file) {
            Path name = file.getFileName();
            if (name != null && matcher.matches(name)) {
                numMatches++;
                System.out.println(file);
            }
        }

        // Prints the total number of
        // matches to standard out.
        void done() {
            System.out.println("Matched: "
                + numMatches);
        }

        // Invoke the pattern matching
        // method on each file.
        @Override
        public FileVisitResult visitFile(Path file,
                BasicFileAttributes attrs) {
            find(file);
            return CONTINUE;
        }

        // Invoke the pattern matching
        // method on each directory.
        @Override
        public FileVisitResult preVisitDirectory(Path dir,
                BasicFileAttributes attrs) {
            find(dir);
            return CONTINUE;
        }

        @Override
        public FileVisitResult visitFileFailed(Path file,
```

```
                IOException exc) {
            System.err.println(exc);
            return CONTINUE;
        }
    }

    static void usage() {
        System.err.println("java Find <path>" +
            " -name \"<glob_pattern>\"");
        System.exit(-1);
    }

    public static void main(String[] args)
        throws IOException {

        if (args.length < 3 || !args[1].equals("-name"))
            usage();

        Path startingDir = Paths.get(args[0]);
        String pattern = args[2];

        Finder finder = new Finder(pattern);
        Files.walkFileTree(startingDir, finder);
        finder.done();
    }
}
```

Recursively walking a file tree is covered in Walking the File Tree.

Previous page: Walking the File Tree
Next page: Watching a Directory for Changes