Listing the Content of a Directory

Listing a the Content of a Directory

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, do not forget to close the stream in the finally block. The try-with-resources statement takes care of this for you. You can learn more about stream in the Stream section.*

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);
}
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

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 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 the What Is a Glob section, at the end of this page.

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);
}
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

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` 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)` 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.

Explore our developer-friendly HTML to PDF API   Printed using PDFCrowd   HTML to PDF

What is a Glob

You can use glob syntax to specify pattern-matching behavior.

A glob pattern is specified as a string and is matched against other strings, such as directory or file names. Glob syntax follows several simple rules:

- An asterisk, `*`, matches any number of characters (including none).
- Two asterisks, `**`, works like `*` but crosses directory boundaries. This syntax is generally used for matching complete paths.
- A question mark, `?`, matches exactly one character.
- Braces specify a collection of subpatterns. For example:
  - `{sun,moon,stars}` matches "sun", "moon", or "stars".
  - `{temp*,tmp*}` matches all strings beginning with "temp" or "tmp".
- Square brackets convey a set of single characters or, when the hyphen character (`-`) is used, a range of characters. For example:
  - `[aeiou]` matches any lowercase vowel.
  - `[0-9]` matches any digit.
  - `[A-Z]` matches any uppercase letter.
  - `[a-z,A-Z]` matches any uppercase or lowercase letter. Within the square brackets, `*`, `?`, and `\` match themselves.
- All other characters match themselves.
- To match *, ?, or the other special characters, you can escape them by using the backslash character, . For example: \ matches a single backslash, and ? matches the question mark.

Here are some examples of glob syntax:

- `*.html` – Matches all strings that end in .html
- `???` – Matches all strings with exactly three letters or digits

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

- `*[0-9]*` – Matches all strings containing a numeric value
- `*.{htm,html,pdf}` – Matches any string ending with .htm, .html or .pdf
- `a?*.java` – Matches any string beginning with a, followed by at least one letter or digit, and ending with .java
- `{foo*,*[0-9]*}` – Matches any string beginning with foo or any string containing a numeric value

> *Note: If you are typing the glob pattern at the keyboard and it contains one of the special characters, you must put the pattern in quotes (`"*"`), use the backslash (`\*`), or use whatever escape mechanism is supported at the command line.*

The glob syntax is powerful and easy to use. However, if it is not sufficient for your needs, you can also use a regular expression. For more information, see the section on Regular Expressions.

For more information about the glob syntax, see the API specification for the `getPathMatcher(String)` method in the `FileSystem` class.

Last update: *January 25, 2023*

Creating and Reading Directories ← **Listing the Content of a Directory** → Walking the File Tree

Home > Tutorials > The Java I/O API > File System Basics > Listing the Content of a Directory

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF