HowToDoInJava

Listing All Files in a Directory in Java

Last Updated: March 14, 2022 By: Lokesh Gupta Java 8 Java IO Basics, Java Stream Basics

Learn to use various Java APIs such as `Files.list()` and `DirectoryStream` to list all files present in a directory, including hidden files, recursively.

- For using external iteration (for loop) use `DirectoryStream`.
- For using Stream API operations, use `Files.list()` instead.

Table Of Contents

1. Listing Files Only in a Given Directory

1.1. Sream of Files with *Files.list()*

If we are interested in **non-recursively listing the files** and excluding all sub-directories and files in sub-directories, then we can use this approach.

- Read all files and directories entries using *Files.list()*.
- Check if a given entry is a file using Predicate *File::isFile*.
- Collect all filtered entries into a *List*.

```
//The source directory
String directory = "C:/temp";

// Reading only files in the directory
try {
  List<File> files = Files.list(Paths.get(directory))
    .map(Path::toFile)
    .filter(File::isFile)
    .collect(Collectors.toList());

  files.forEach(System.out::println);
} catch (IOException e) {
  e.printStackTrace();
}
```

1.2. DirectoryStream to Loop through Files

DirectoryStream is part of Java 7 and is used to iterate over the entries in a directory in for-each loop style.

Closing a directory stream releases any resources associated with the stream. Failure to close the stream may result in a resource leak. The try-with-resources statement provides a useful construct to ensure that the stream is closed.

```
List<File> fileList = new ArrayList<>();

try (DirectoryStream<Path> stream = Files
  .newDirectoryStream(Paths.get(directory))) {
  for (Path path : stream) {
    if (!Files.isDirectory(path)) {
      fileList.add(path.toFile());
    }
  }
}

fileList.forEach(System.out::println);
```

2. Listing All Files in Given Directory and Sub-directories

2.1. *Files.walk()* for Stream of Paths

The *walk()* method returns a *Stream* by walking the file tree beginning with a given starting file/directory in a **depth-first** manner.

Note that this method **visits all levels of the file tree**.

```
  String directory = "C:/temp";
  List<Path> pathList = new ArrayList<>();

  try (Stream<Path> stream = Files.walk(Paths.get(directory))) {
    pathList = stream.map(Path::normalize)
          .filter(Files::isRegularFile)
          .collect(Collectors.toList());
  }

  pathList.forEach(System.out::println);
```

If you wish to include the list of *Path* instances for directories as well, then remove the filter condition *Files::isRegularFile*.

2.2. Simple Recursion

We can also write the file tree walking logic using the recursion. It gives a little more flexibility if we want to perform some intermediate steps/checks before adding the entry to list of the files.

```
String directory = "C:/temp";

//Recursively list all files
List<File> fileList = listFiles(directory);

fileList.forEach(System.out::println);

private static List<File> listFiles(final String directory) {
    if (directory == null) {
      return Collections.EMPTY_LIST;
    }
    List<File> fileList = new ArrayList<>();
    File[] files = new File(directory).listFiles();
    for (File element : files) {
      if (element.isDirectory()) {
        fileList.addAll(listFiles(element.getPath()));
      } else {
        fileList.add(element);
      }
    }
    return fileList;
}
```

Please note that if we're working with a large directory, then using `DirectoryStream` performs better.

3. Listing All Files of a Certain Extention

To get the list of all files of certain extensions only, use two predicates `Files::isRegularFile` and `filename.endsWith(".extension")` together.

In given example, we are listing all `.java` files in a given directory and all of its sub-directories.

```
String directory = "C:/temp";

//Recursively list all files
List<Path> pathList = new ArrayList<>();

try (Stream<Path> stream = Files.walk(Paths.get(directory))) {
      // Do something with the stream.
      pathList = stream.map(Path::normalize)
        .filter(Files::isRegularFile)
        .filter(path -> path.getFileName().toString().endsWith(".java
        .collect(Collectors.toList());
    }
    pathList.forEach(System.out::println);
}
```

4. Listing All Hidden Files

To find all the hidden files, we can use filter expression `file -> file.isHidden()` in any of the above examples.

```
List<File> files = Files.list(Paths.get(dirLocation))
      .filter(path -> path.toFile().isHidden())
      .map(Path::toFile)
      .collect(Collectors.toList());
```

In the above examples, we learn to use the java 8 APIs loop through the files in a directory recursively using various search methods. Feel free to modify the code and play with it.

Happy Learning !!

Sourcecode on Github

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Tomcat – Enable/disable directory listing
2. Manage system log files not to exceed N GB in linux using java
3. Java FilenameFilter to Find Files Matching Pattern
4. Java 11 – Files writeString() API
5. 13 Spring Best Practices for Writing Configuration Files
6. Android Tutorial : Android Project Structure, Files and Resources
7. Python read and write csv files
8. Lucene – Index and Search Text Files
9. Spring Batch MultiResourceItemReader – Read Multiple CSV Files Example
0. Spring boot multiple log files example

10 thoughts on “Listing All Files in a Directory in Java”

Max Reimer

June 22, 2022 at 1:24 pm

List files = Files.list(Paths.get(directory))
.map(Path::toFile)
.filter(File::isFile)
.collect(Collectors.toList());

I have the suspicion that the stream is not closed after this use and handlers get stuck in the system.
Is that correct?

Reply

ReddiSeharReddy

November 27, 2019 at 12:35 pm

Hi,
how to apply filter to get the list of files created between two dates.

Could you please suggest on this.

Reply

ReddiSeharReddy
November 27, 2019 at 1:48 pm

Files.newDirectoryStream(Paths.get(directory),
path -> path.toFile().lastModified() > sd.getTime() && path.toFile().lastModified() < ed.getTime());

Reply

Rich K
October 27, 2018 at 5:42 am

Alex B, that code does not demonstrate what list the filenames go to. Where is the list variable ?

Reply

Anubhav Gupta
February 16, 2018 at 11:36 pm

Here is the code for reading filename and store them in list.

```
List fileNamesList = new ArrayList();
      Files.newDirectoryStream(Paths.get(dir),
      path -> path.toString().endsWith(".java")).forEach(filePath -> fileN
```

Reply

jack

February 6, 2018 at 8:25 pm

I appreciate the post.

But the code in “Find all hidden files in directory” section gives syntax error.

And, its sad that you didn't tell how to capture the file names in a list or something else (will be helpful for the people who are learning java 8). In real world, we don't need to read the file names just to sysout.

Reply

Anubhav Gupta

February 16, 2018 at 11:34 pm

Here is the code for reading file names and store them in list.

```
List fileNamesList = new ArrayList();
    Files.newDirectoryStream(Paths.get(dir), path -> path.toString().ends
```

Reply

Alex B

March 12, 2018 at 9:02 pm

You're actually better off doing something like this:

```
Files.list(Paths.get(dir))
        .map(Path::toFile)
        .map(File::getAbsolutePath)
        .collect(Collectors.toList())
```

Which allows you to use the streams api to transform/filter the list before you collect it.

Reply

sekaijin

September 18, 2019 at 9:53 pm

```
Files.list(Paths.get(dir))
        .map(Path::toAbsolutePath)
        .collect(Collectors.toList())
```

Reply

Dhiraj Kumar

March 21, 2020 at 8:57 pm

nice code

Reply

Leave a Comment

Name *

Email *

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial