链滴

# Java 基础之 IO 和 NIO 补完

作者：james

原文链接：https://ld246.com/article/1512039236654

来源网站：链滴

# Java Stream，File，IO

- 关于NIO和IO的比较，参考： Java NIO系列教程（十二） Java NIO与IO

# java包之java.io

- 参考材料： 菜鸟教材

# java包之 java.nio

- 参考： java NIO系列教程

## 15 Java NIO Path

- 翻译自： 15Java NIO Path
- Java的java.nio.file.Path接口是JavaNIO 2 update的一部分，在Java6和Java7中得到更新。Java path接口在java7中被加入到Java NIO中。Path接口位于java.nio.file包中，所以Java Path接口的全名是java.nio.file.Path。
- 一个Java Path接口代表了一个文件系统中的path路径。path路径可以指向文件或目录。一个路径以是绝对路径也可以是相对路径。一个绝对路径是从文件系统根目录指向该文件或路径的全路径。一相对路径是指相对于其他路径一个文件或目录的相对路径。相对路径可能比较费解，不用担心，我会该文中详细介绍。
- 对于操作系统中的系统路径这样的环境变量不要疑惑，java Path接口跟它们没有关系。
- 在很多方面java.nio.file.Path和java.io.File类是类似的，在一些地方你可以使用Path接口来代替File，但是也有一些不同。

### 1. 创建Path实例

- 你可以使用java.nio.file.Paths类的静态方法Paths.get()来创建一个java.nio.file.Path的实例:

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class PathExample {

    public static void main(String[] args) {

        Path path = Paths.get("c:\\data\\myfile.txt");

    }
}
```

- Paths.get()方法是创建Path实例的工厂方法。

### 2. 创建一个绝对路径Path实例

- Windows文件系统下的绝对路径：

```
Path path = Paths.get("c:\\data\\myfile.txt");
```

- 类Unix操作系统下的绝对路径：

```
Path path = Paths.get("/home/jakobjenkov/myfile.txt");
//如果你在windows上这么使用，会表示当前系统分区下的路径，如：C:/home/jakobjenkov/myfile.
xt
```

### 3. 创建一个相对路径Path实例

- 相对路径需要一个basepath和relativepath来创建：

```
Path projects = Paths.get("d:\\data", "projects");

Path file     = Paths.get("d:\\data", "projects\\a-project\\myfile.txt");
```

- 可以使用'.','..'来创建路径实例：

```
Path currentDir = Paths.get(".");
System.out.println(currentDir.toAbsolutePath());
```

### 4. Path.normalize()

```
String originalPath =
        "d:\\data\\projects\\a-project\\..\\another-project";

Path path1 = Paths.get(originalPath);
System.out.println("path1 = " + path1);

Path path2 = path1.normalize();
System.out.println("path2 = " + path2);
path1 = d:\data\projects\a-project\..\another-project
path2 = d:\data\projects\another-project
```

## 16. Java NIO Files

- 翻译自：  16Java NIO Files
- java.nio.file.Files类提供了多种方法来操作文件。

### 1. Files.exists()

- LinkOption.NOFOLLOW_LINKS表示文件存在，但是不能是链接形式的。

```
Path path = Paths.get("data/logging.properties");

boolean pathExists =
        Files.exists(path,
            new LinkOption[]{ LinkOption.NOFOLLOW_LINKS});
```

## 2. Files.createDirectory()

```
Path path = Paths.get("data/subdir");

try {
    Path newDir = Files.createDirectory(path);
} catch(FileAlreadyExistsException e){
    // the directory already exists.
} catch (IOException e) {
    //something else went wrong
    e.printStackTrace();
}
```

## 3. Files.copy()

```
Path sourcePath      = Paths.get("data/logging.properties");
Path destinationPath = Paths.get("data/logging-copy.properties");

try {
    Files.copy(sourcePath, destinationPath);
} catch(FileAlreadyExistsException e) {
    //destination file already exists
} catch (IOException e) {
    //something else went wrong
    e.printStackTrace();
}
```

## 4. Overwriting Existing Files

- 通过复制选项来覆盖已经存在的Files

```
Path sourcePath      = Paths.get("data/logging.properties");
Path destinationPath = Paths.get("data/logging-copy.properties");

try {
    Files.copy(sourcePath, destinationPath,
            StandardCopyOption.REPLACE_EXISTING);
} catch(FileAlreadyExistsException e) {
    //destination file already exists
} catch (IOException e) {
    //something else went wrong
    e.printStackTrace();
}
```

## 5. Files.move()

- java.io.File 有方法 renameTo() ,Files可以Files.move()

```
Path sourcePath      = Paths.get("data/logging-copy.properties");
Path destinationPath = Paths.get("data/subdir/logging-moved.properties");
```

```
try {
    Files.move(sourcePath, destinationPath,
            StandardCopyOption.REPLACE_EXISTING);
} catch (IOException e) {
    //moving file failed.
    e.printStackTrace();
}
```

## 6. Files.delete()

```
Path path = Paths.get("data/subdir/logging-moved.properties");

try {
    Files.delete(path);
} catch (IOException e) {
    //deleting file failed
    e.printStackTrace();
}
```

## 7. Files.walkFileTree()

- Files.walkFileTree()用来递归遍历文件目录，有Path实例和FileVisitor来做参数。
- FileVisitor接口有如下形式：

```
public interface FileVisitor {

    public FileVisitResult preVisitDirectory(
        Path dir, BasicFileAttributes attrs) throws IOException;

    public FileVisitResult visitFile(
        Path file, BasicFileAttributes attrs) throws IOException;

    public FileVisitResult visitFileFailed(
        Path file, IOException exc) throws IOException;

    public FileVisitResult postVisitDirectory(
        Path dir, IOException exc) throws IOException {

}
```

- FileVisitor有一个简单实现SimpleFileVisitor类：

```
Files.walkFileTree(path, new FileVisitor<Path>() {
  @Override
  public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs) throws IOException

    System.out.println("pre visit dir:" + dir);
    return FileVisitResult.CONTINUE;
  }
```

```
  @Override
  public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
    System.out.println("visit file: " + file);
    return FileVisitResult.CONTINUE;
  }

  @Override
  public FileVisitResult visitFileFailed(Path file, IOException exc) throws IOException {
    System.out.println("visit file failed: " + file);
    return FileVisitResult.CONTINUE;
  }

  @Override
  public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws IOException {
    System.out.println("post visit directory: " + dir);
    return FileVisitResult.CONTINUE;
  }
});
```

FileVisitResult的集合有：
CONTINUE 表示继续
TERMINATE 表示停止
SKIP_SIBLINGS 表示继续但是不再便利兄弟目录
SKIP_SUBTREE 表示继续但是不再便利子目录

## 8. Searching For Files

- 下面可以实现SimpleFileVisitor来查找文件：

```
Path rootPath = Paths.get("data");
String fileToFind = File.separator + "README.txt";

try {
  Files.walkFileTree(rootPath, new SimpleFileVisitor<Path>() {

    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
      String fileString = file.toAbsolutePath().toString();
      //System.out.println("pathString = " + fileString);

      if(fileString.endsWith(fileToFind)){
        System.out.println("file found at path: " + file.toAbsolutePath());
        return FileVisitResult.TERMINATE;
      }
      return FileVisitResult.CONTINUE;
    }
  });
} catch(IOException e){
    e.printStackTrace();
}
```

## 9. Deleting Directories Recursively

```
Path rootPath = Paths.get("data/to-delete");

try {
  Files.walkFileTree(rootPath, new SimpleFileVisitor<Path>() {
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
      System.out.println("delete file: " + file.toString());
      Files.delete(file);
      return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws IOException {
      Files.delete(dir);
      System.out.println("delete dir: " + dir.toString());
      return FileVisitResult.CONTINUE;
    }
  });
} catch(IOException e){
  e.printStackTrace();
}
```

## 10. Additional Methods in the Files Class

● Files类有很多其他方法，如 creating symbolic links, determining the file size, setting file perm ssions 等。参考java文档。

# 17. Java NIO AsynchronousFileChannel

● java7中AsynchronousFileChannel加入到了java NIO的包中。AsynchronousFileChannels使得步读写文件实现。

## 1. Creating an AsynchronousFileChannel

```
Path path = Paths.get("data/test.xml");

AsynchronousFileChannel fileChannel =
    AsynchronousFileChannel.open(path, StandardOpenOption.READ);
```

## 2. Reading Data

● Reading Data Via a Future

```
Future<Integer> operation = fileChannel.read(buffer, 0);
AsynchronousFileChannel fileChannel =
    AsynchronousFileChannel.open(path, StandardOpenOption.READ);

ByteBuffer buffer = ByteBuffer.allocate(1024);
long position = 0;

Future<Integer> operation = fileChannel.read(buffer, position);
```

```
// Of course, this is not a very efficient use of the CPU - but somehow you need to wait until t
e read operation has completed.

while(!operation.isDone());

buffer.flip();
byte[] data = new byte[buffer.limit()];
buffer.get(data);
System.out.println(new String(data));
buffer.clear();
```

● Reading Data Via a CompletionHandler

```
fileChannel.read(buffer, position, buffer, new CompletionHandler<Integer, ByteBuffer>() {
//Once the read operation finishes the CompletionHandler's completed() method will be calle
.
    @Override
    public void completed(Integer result, ByteBuffer attachment) {
        System.out.println("result = " + result);

        attachment.flip();
        byte[] data = new byte[attachment.limit()];
        attachment.get(data);
        System.out.println(new String(data));
        attachment.clear();
    }
//If the read operation fails, the failed() method of the CompletionHandler will get called inst
ad.
    @Override
    public void failed(Throwable exc, ByteBuffer attachment) {

    }
});
```

## 3. Writing Data

● Writing Data Via a Future

```
Path path = Paths.get("data/test-write.txt");
//If the file does not exist the write() method will throw a java.nio.file.NoSuchFileException .
if(!Files.exists(path)){
    Files.createFile(path);
}
AsynchronousFileChannel fileChannel =
    AsynchronousFileChannel.open(path, StandardOpenOption.WRITE);

ByteBuffer buffer = ByteBuffer.allocate(1024);
long position = 0;

buffer.put("test data".getBytes());
buffer.flip();
```

```
Future<Integer> operation = fileChannel.write(buffer, position);
buffer.clear();

while(!operation.isDone());

System.out.println("Write done");
```

● Writing Data Via a CompletionHandler

```
Path path = Paths.get("data/test-write.txt");
if(!Files.exists(path)){
    Files.createFile(path);
}
AsynchronousFileChannel fileChannel =
    AsynchronousFileChannel.open(path, StandardOpenOption.WRITE);

ByteBuffer buffer = ByteBuffer.allocate(1024);
long position = 0;

buffer.put("test data".getBytes());
buffer.flip();

fileChannel.write(buffer, position, buffer, new CompletionHandler<Integer, ByteBuffer>() {

    @Override
    public void completed(Integer result, ByteBuffer attachment) {
        System.out.println("bytes written: " + result);
    }

    @Override
    public void failed(Throwable exc, ByteBuffer attachment) {
        System.out.println("Write failed");
        exc.printStackTrace();
    }
});
```