CS209A - File & NIO

Key Content

- File and Path
- NIO

1. Class `File` basic usage

1.1 File and path

The file path should be passed into the constructor of `File` when new a `File` object.

```
File f = new File("/Users/zhaoyao/Downloads/1.jpeg");
```

When constructing a `File` object, either an absolute path or a relative path can be passed as input parameter. The absolute path is the full path beginning with the root directory, for example:

```
File f = new File("c:\\windows\\1.jpeg");
```

Note that the Windows platform uses '\'as a path separator, while Linux platform uses '/' as a path separator.

Relative path

When passing in a relative path, the current directory appends the relative path forms an absolute path:

```
// Assume current directory is c:\windows
File f1 = new File("1.jpeg"); // absolute path: "c:\\windows\\1.jpeg"
File f2 = new File(".\\1.jpeg");// absolute path: "c:\\windows\\1.jpeg"
File f3 = new File("..\\1.jpeg");// absolute path: "c:\\1.jpeg"
```

Note: "." represents the current directory and ".." represents the parent directory.

Canonical path

What is canonical path? What is the difference between absolute path and canonical path?

Please run the following code and observe the result :

```
File f = new File("..");
System.out.println(f.getPath());
System.out.println(f.getAbsolutePath());
System.out.println(f.getCanonicalPath());
```

An absolute path maybe contains “.” And “..”, canonical path is a standard absolute path which will not contain “.” And “..”.

1.2 File and Directory

`File` Objects can represent files or directories. When a new `File` Object is created successfully, it doesn’t mean the file or directory exists, because creating a file object doesn’t cause any disk operations.
When does the disk operation actually performed ?
The methods of the `File` Object are invoked.
Please run the following code and observe the result :

```
File f1 = new File("C: \\Windows");
File f2 = new File("C: \\Windows\\notepad.exe");
File f3 = new File("C: \\Windows\\null");
System.out.println(f1.isFile());
System.out.println(f1.isDirectory());
System.out.println(f2.isFile());
System.out.println(f2.isDirectory());
System.out.println(f3.isFile());
System.out.println(f3.isDirectory());
```

You can use the following methods to get more information of the file:

- boolean canRead()
- boolean canWrite()
- boolean canExecute()
- long length()

Create and delete files:

- boolean createNewFile()
- boolean delete()

Create and delete directories:

- boolean mkdir():Create the directory represented by the current File object;
- boolean mkdirs():Create the directory represented by the current File object, and if necessary, create the non-existent parent directory;
- boolean delete():Delete the directory represented by the current File object. The current directory must be empty, otherwise it can't be deleted successfully.

Traverse files and directories:

- File[] listFiles(): if the current object is a directory, the method can list all the names of files and subdirectories under the directory

Please run the following code and observe the result :

```
File f = new File("C: \\Windows");
File[] fs = f.listFiles();
if (fs != null) {
    for (File f : files) {
        System.out.println(f);
    }
}
```

More detail: https://docs.oracle.com/javase/tutorial/essential/io/fileio.html

2. NIO

2.1 Class `Path` basic usage

JDK also provides class `Path` to do file operations. The usage of class `Path` is similar with the class `File`, and even simper.

A `Path` instance represents a path in the file system. A path can point to either a file or a directory.

```
Path p1 = Paths.get(".", "project", "study"); // create a Path object
System.out.println(p1);
Path p2 = p1.toAbsolutePath(); // convert to a canonical path
System.out.println(p2);
Path p3 = p2.normalize(); // convert to a canonical path
System.out.println(p3);
File f = p3.toFile(); // convert to a File object
System.out.println(f);
for (Path p : Paths.get("..").toAbsolutePath()) { // trace back the Path
    System.out.println("    " + p);
}
```

2.2 Class `Files` basic usage

The `java.nio.file.Files` class works with `java.nio.file.Path` instances, so you need to understand the `Path` class before you can work with the `Files` class.

Files.exists()method checks if a given Path exists in the file system.

```
Path path = Paths.get("C: \\Windows\\null ");
boolean pathExists = Files.exists(path);
```

The Files.createDirectory() method creates a new directory from a Path instance.

```
Path path = Paths.get("C: \\Windows\\null ");
try {
    Path newDir = Files.createDirectory(path);
} catch(FileAlreadyExistsException e){
    System.out.println("The directory already exists." );
} catch (IOException e) {
    //something else went wrong
    e.printStackTrace();
```

```
}
```

The Files.copy() method copies a file from one path to another.

```
Path sourcePath      = Paths.get("C: \\Windows\\1.txt ");
Path destinationPath = Paths.get("C: \\Windows\\documents\\1_copy.txt ");


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

It is possible to force the Files.copy() to overwrite an existing file, append the copy option when copying.

```
Files.copy(sourcePath, destinationPath,
                    StandardCopyOption.REPLACE_EXISTING);
```

The Files.move() method moves a file from one path to another and can change its name in the same operation

```
Path sourcePath      = Paths.get("C: \\Windows\\1.txt ");
Path destinationPath = Paths.get("C: \\Windows\\documents\\2_move.txt ");
try {
    Files.move(sourcePath, destinationPath,
            StandardCopyOption.REPLACE_EXISTING);
} catch (IOException e) {
    //moving file failed.
    e.printStackTrace();
```

```
}
```

First, the source path and destination path are created. The source path points to the file to move, and the destination path points to where the file should be moved to. Then the Files.move() method is called. This results in the file being moved.

Notice the third parameter passed to Files.move() . This parameter tells the Files.move() method to overwrite any existing file at the destination path. This parameter is actually optional.

The Files.move() method may throw an IOException if moving the file fails. For instance, if a file already exists at the destination path, and you have left out the StandardCopyOption.REPLACE_EXISTING option, or if the file to move does not exist etc.

The Files.delete() method can delete a file or directory.

```
Path path = Paths.get("C: \\Windows\\1.txt ");
try {
    Files.delete(path);
} catch (IOException e) {
    //deleting file failed
    e.printStackTrace();
}
```

The java.nio.file.Files class contains many other useful functions, like functions for creating symbolic links, determining the file size, setting file permissions etc. Check out the JavaDoc for the java.nio.file.Files class for more information about these methods.

3. Sample code

```java
public static boolean createFile(String destFileName) {
    File file = new File(destFileName);
    if (file.exists()) {
        System.out.println("Create single file " + destFileName + " fail,target file already exists! ");
        return false;
    }
    if (destFileName.endsWith(File.separator)) {
        System.out.println("Create single file " + destFileName + " fail, target file cannot be a directory! ");
        return false;
    }
    // Check if the directory where the target file is located exists
    if (!file.getParentFile().exists()) {
        // if the directory where the target file is located doesn't exist, create
        // its' parent directory.
        System.out.println("directory where target file is located doesn't exist, create its' parent directory! ");
        File parentFile = file.getParentFile();
        parentFile.mkdirs();
        if (!file.getParentFile().mkdirs()) {
            System.out.println("Create directory where target file is located fails! ");
            return false;
        }
    }
    // Create target file
    try {
        if (file.createNewFile()) {
            System.out.println("Create single file " + destFileName + " success! ");
            return true;
        } else {
            System.out.println("Create single file " + destFileName + " fail! ");
            return false;
        }
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Create single file " + destFileName + " fail! " + e.getMessage());
        return false;
    }
```

Invoke the above method and observe the result.

How to do the same operations using `Path` and `Files`?

Reference

https://docs.oracle.com/javase/tutorial/essential/io/index.html