E/S de arquivos com Java NIO.2

ACH 2003 — Computação Orientada a Objetos

Daniel Cordeiro

27 de abril de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

PATH

```java
// Nenhum dos métodos exige que o arquivo exista de fato
// sintaxe Windows
Path path = Paths.get("C:\\home\\joe\\foo");
// sintaxe Unix
Path path = Paths.get("/home/joe/foo");

System.out.format("toString: %s%n", path.toString());
 // /home/joe/foo ou C:\home\joe\foo
System.out.format("getFileName: %s%n", path.getFileName());
 // foo
System.out.format("getName(0): %s%n", path.getName(0));
 // home
System.out.format("getNameCount: %d%n", path.getNameCount());
 // 3
System.out.format("subpath(0,2): %s%n", path.subpath(0,2));
 // home/joe ou home\joe
System.out.format("getParent: %s%n", path.getParent());
 // /home/joe ou \home\joe
System.out.format("getRoot: %s%n", path.getRoot());
 // / ou C:\
```

Glob

Um argumento *glob* é uma string com um padrão de busca que será comparada com outras strings que representam nomes de arquivos e diretórios. Exemplos:

`*`	casa com qualquer quantidade de caracteres
`**`	funciona como `*`, mas percorre vários diretórios
colchetes	conjunto ou faixa de caracteres. Ex: `[aeiou]`, `[0-9]`, `[a-z,A-Z]`, etc.
chaves	uma coleção de consultas. Ex: `{temp*,tmp*}`

Exemplo de uso:

- `*.java`
- `???`
- `*.{html,htm,pdf}`
- `{foo*,*[0-9]*}`

VERIFICAR UM ARQUIVO OU DIRETÓRIO

Verificação de existência

- `Files.exists(Path, LinkOption...)` e `Files.notExists(Path, LinkOption...)`

Três resultados são possíveis:

- o arquivo existe com certeza
- o arquivo não existe com certeza
- o estado do arquivo é desconhecido (ex: o programa não tem acesso ao arquivo); nesse caso tanto `exists()` como `notExists()` devolvem `false`.

Permissão de acesso

O código a seguir verifica se o arquivo existe e se pode ser executado:

```
Path file = ...;
boolean isRegularExecutableFile = Files.isRegularFile(file) &
         Files.isReadable(file) & Files.isExecutable(file);
```

Permissão de acesso

O código a seguir verifica se o arquivo existe e se pode ser executado:

```
Path file = ...;
boolean isRegularExecutableFile = Files.isRegularFile(file) &
         Files.isReadable(file) & Files.isExecutable(file);
```

Dois caminhos podem apontar pro mesmo arquivo

```
Path p1 = ...;
Path p2 = ...;

if (Files.isSameFile(p1, p2)) {
    // p1 pode ser um link simbólico para p2
}
```

REMOVER UM ARQUIVO OU DIRETÓRIO

- você pode remover arquivos, diretório ou *links*
- quando um *link* é removido, seu alvo não é
- um diretório precisa estar vazio para ser removido

A classe `Files` provê dois métodos para remoção:

delete(Path) remove um arquivo ou lança uma exceção se ele não existir

```
try {
    Files.delete(path);
} catch (NoSuchFileException x) {
    System.err.format("%s: no such" + " file or directory
} catch (DirectoryNotEmptyException x) {
    System.err.format("%s not empty%n", path);
} catch (IOException x) {
    // File permission problems are caught here.
    System.err.println(x);
}
```

deleteIfExists(Path) remove um arquivo, mas não faz nada se ele existir

COPIAR UM ARQUIVO OU DIRETÓRIO

- cópias podem ser realizadas com o método `Files.copy(Path, Path, CopyOption...)`
- a cópia de um arquivo falha se o destino existir, exceto se a opção `REPLACE_EXISTING` for usada
- a cópia de um link produz uma cópia de seu alvo. Para copiar o próprio link, use a opção `NOFOLLOW_LINKS`
- para copiar o arquivo e seus atributos (ex: last-modified-time), use a opção `COPY_ATTRIBUTES`

Cópia de/para fluxos

- `Files.copy(InputStream, Path, CopyOption...)`
- `Files.copy(Path, OutputStream)`

MOVER UM ARQUIVO OU DIRETÓRIO

`Files.move(Path, Path, CopyOption...)`

As seguintes opções são válidas:

REPLACE_EXISTING sobrescreve o destino; se o destino for um link simbólico, o link é sobrescrito mas o arquivo para o qual aponta, não

ATOMIC_MOVE se o sistema de arquivos permitir, a operação é garantida como sendo atômica

Exemplo:

```
import static java.nio.file.StandardCopyOption.*;
...
Files.move(source, target, REPLACE_EXISTING);
```

METADADOS

Métodos de `Files` que devolvem metadados sobre arquivos e diretórios:

- `size(Path)`
- `isDirectory(Path, LinkOption)`
- `isRegularFile(Path, LinkOption...)`
- `isSymbolicLink(Path)`
- `isHidden(Path)`
- `getLastModifiedTime(Path, LinkOption...)`
- `setLastModifiedTime(Path, FileTime)`
- `getOwner(Path, LinkOption...)`
- `setOwner(Path, UserPrincipal)`
- `getPosixFilePermissions(Path, LinkOption...)`
- `setPosixFilePermissions(Path, Set<PosixFilePermission>)`

ABERTURA DE ARQUIVOS

Vários métodos de manipulação de arquivo recebem um parâmetro opcional chamado `OpenOptions`, que pode receber um dos seguintes valores definidos pelo enum `StandardOpenOptions`:

WRITE	abre um arquivo para escrita
APPEND	adiciona novos dados ao final do arquivo
TRUNCATE_EXISTING	trunca o arquivo para zero bytes
CREATE_NEW	cria um novo arquivo e lança exceção se o caminho já existir
CREATE	abre um arquivo se existir ou cria um novo
DELETE_ON_CLOSE	remove o arquivo quando o fluxo for fechado. Útil para arquivos temporários
SPARSE	informa que o arquivo será esparso, alguns sistemas de arquivos podem usar a informação para economizar espaço (ex: NTFS)
SYNC	mantém o arquivo (conteúdo e metadados) sempre sincronizado com o dispositivo de armazenamento
DSYNC	mantém o conteúdo do arquivo sempre sincronizado

MANIPULAÇÃO DE ARQUIVOS PEQUENOS

Ler todos os bytes/linhas do arquivo[1]

```
Path file = ...;
byte[] fileArray;
fileArray = Files.readAllBytes(file);

// ou
String[] linhas = Files.readAllLines(file, StandardCharsets.UTF_8)
```

[1] Se o charset não for especificado, assume-se UTF-8.

MANIPULAÇÃO DE ARQUIVOS PEQUENOS

Ler todos os bytes/linhas do arquivo[1]

```
Path file = ...;
byte[] fileArray;
fileArray = Files.readAllBytes(file);

// ou
String[] linhas = Files.readAllLines(file, StandardCharsets.UTF_8)
```

Escrever todos os bytes/linhas em arquivo

```
Path file = ...;
byte[] buf = ...;
Files.write(file, buf, StandardOpenOption.APPEND);

// ou
Files.write(file, buf, lista_de_strings);
```

[1] Se o charset não for especificado, assume-se UTF-8.

Ler um arquivo usando um fluxo com buffer

```
Charset charset = Charset.forName("US-ASCII");
try (BufferedReader reader = Files.newBufferedReader(file, charset)) {
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```

Ler um arquivo usando um fluxo com buffer

```
Charset charset = Charset.forName("US-ASCII");
try (BufferedReader reader = Files.newBufferedReader(file, charset)) {
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```

Escrever um arquivo usando um fluxo com buffer

```
Charset charset = Charset.forName("UTF-8");
String s = ...;
try (BufferedWriter writer = Files.newBufferedWriter(file, charset)) {
    writer.write(s, 0, s.length());
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```

Ler arquivo usando fluxo de E/S

```
Path file = ...;
try (InputStream in = Files.newInputStream(file);
    BufferedReader reader =
      new BufferedReader(new InputStreamReader(in))) {
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException x) {
    System.err.println(x);
}
```

Criar e escrever em fluxos de E/S

```
import static java.nio.file.StandardOpenOption.*;
import java.nio.file.*;
import java.io.*;

public class LogFileTest {

  public static void main(String[] args) {

    // Convert the string to a
    // byte array.
    String s = "Hello World! ";
    byte data[] = s.getBytes();
    Path p = Paths.get("./logfile.txt");

    try (OutputStream out = new BufferedOutputStream(
      Files.newOutputStream(p, CREATE, APPEND))) {
      out.write(data, 0, data.length);
    } catch (IOException x) {
      System.err.println(x);
    }
  }
}
```

Arquivos regulares

```
Path file = ...;
try {
    // Create the empty file with default permissions, etc.
    Files.createFile(file);
} catch (FileAlreadyExistsException x) {
    System.err.format("file named %s" +
        " already exists%n", file);
} catch (IOException x) {
    // Some other sort of failure, such as permissions.
    System.err.format("createFile error: %s%n", x);
}
```

MÉTODOS PARA CRIAÇÃO DE ARQUIVOS

Arquivos regulares

```
Path file = ...;
try {
    // Create the empty file with default permissions, etc.
    Files.createFile(file);
} catch (FileAlreadyExistsException x) {
    System.err.format("file named %s" +
        " already exists%n", file);
} catch (IOException x) {
    // Some other sort of failure, such as permissions.
    System.err.format("createFile error: %s%n", x);
}
```

Temporários

```
try {
    Path tempFile = Files.createTempFile(null, ".myapp");
    System.out.format("The temporary file" +
        " has been created: %s%n", tempFile)
;
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
// The temporary file has been created: /tmp/5096687029745371184.myapp
```

Listar os diretórios raiz

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name: dirs) {
    System.err.println(name);
}
```

Listar os diretórios raiz

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name: dirs) {
    System.err.println(name);
}
```

Criar diretório

```
Path dir = ...;
Files.createDirectory(path);

// Definição de permissões
Set<PosixFilePermission> perms =
    PosixFilePermissions.fromString("rwxr-x---");
FileAttribute<Set<PosixFilePermission>> attr =
    PosixFilePermissions.asFileAttribute(perms);
Files.createDirectory(file, attr);
```

Listar os diretórios raiz

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
for (Path name: dirs) {
    System.err.println(name);
}
```

Criar diretório

```
Path dir = ...;
Files.createDirectory(path);

// Definição de permissões
Set<PosixFilePermission> perms =
    PosixFilePermissions.fromString("rwxr-x---");
FileAttribute<Set<PosixFilePermission>> attr =
    PosixFilePermissions.asFileAttribute(perms);
Files.createDirectory(file, attr);
```

Diretórios temporários

- `createTempDirectory(Path, String, FileAttribute<?>...)`
- `createTempDirectory(String, FileAttribute<?>...)`

LISTAGEM DE DIRETÓRIOS

```
Path dir = ...;
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path file: stream) {
        System.out.println(file.getFileName());
    }
} catch (IOException | DirectoryIteratorException x) {
    System.err.println(x);
}
```

LISTAGEM DE DIRETÓRIOS

```
Path dir = ...;
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path file: stream) {
        System.out.println(file.getFileName());
    }
} catch (IOException | DirectoryIteratorException x) {
    System.err.println(x);
}
```

Filtragem usando um glob

```
Path dir = ...;
try (DirectoryStream<Path> stream =
     Files.newDirectoryStream(dir, "*.{java,class,jar}")) {
    for (Path entry: stream) {
        System.out.println(entry.getFileName());
    }
} catch (IOException x) {
    System.err.println(x);
}
```

BIBLIOGRAFIA

- The Java™ Tutorials – Basic I/O: `https://docs.oracle.com/javase/tutorial/essential/io/`