Path

Интерфейс: `java.nio.file.Path`

Описание

Представляет собой абстракцию пути к элементу файловой системы (файлу, каталогу или чему-то ещё). Понятие "файловой системы" здесь можно трактовать достаточно широко, и это не обязательно именно дисковая файловая система. Например, в стандартной библиотеке есть реализация, представляющая ZIP-архив как файловую систему.

Класс `Path` предназначен только для манипуляции путями как синтаксическими конструкциями, и вовсе не обязательно, что файл (или другой элемент ФС), на который указывает путь, действительно существует. Объект `Path` проще всего рассматривать как *ссылку* на элемент ФС, которая сама по себе не умеет работать с тем, на что ссылается. Для операций с самой ФС используется класс `Files`, принимающий параметры типа `Path`.

Здесь мы рассмотрим только операции со стандартной ФС.

Создание путей

Чтобы создать объект типа `Path` с нуля, используется статический метод `get` в классе `Paths`:

```
static Path get(String first, String... more)
```

Этот метод принимает один и более компонентов пути, разбивает их на отдельные подкомпоненты по платформозависимым разделителям (`/` для Unix, `\` для Windows), и из полученных компонентов создаёт объект типа `Path`.

Например, на платформе Windows все приведённые примеры эквивалентны:

```
Path p1 = Paths.get("C:/windows/explorer.exe");
Path p2 = Paths.get("C:\\windows\\explorer.exe");
Path p3 = Paths.get("C:\\windows", "explorer.exe");
Path p4 = Paths.get("C:", "windows", "explorer.exe");
```

Важно! Обратите внимание, что обратная косая черта должна экранироваться. К сожалению, в Java нет способа отменить экранирование для более короткой записи Windows-путей.

Объекты класса `Path` являются неизменяемыми. Все операции с путями возвращают новые

объекты `Path`.

Строковое представление

У класса `Path` есть вполне привычный метод `toString()`, возвращающий представление пути в виде строки, используя системный разделитель компонентов пути.

```
Path path = Paths.get("C:\\windows\\system32");
System.out.println(path);
// Или, что то же самое:
// System.out.println(path.toString())
```

Результат:

```
C:\windows\system32
```

Абсолютные и относительные пути

Приведённый выше пример пути является *абсолютным путём*, потому что он начинается с корня ФС. Абсолютный путь всегда однозначно идентифицирует элемент ФС, независимо от того, какой каталог является текущим каталогом программы.

В Unix абсолютные пути начинаются с косой черты, обозначающей корень файловой системы, а в Windows — с буквы диска. Например, следующие два пути являются абсолютными:

```
Path unixAbsPath = Paths.get("/usr/bin/firefox");
Path winAbsPath = Paths.get("C:\\Program Files\\Mozilla Firefox\\firefox.exe");
```

Относительные пути — это пути, ведущие отсчёт от какого-то каталога. По умолчанию при выполнении операций ФС над относительными путями они разрешаются относительно *текущего каталога* (current directory), также называемого *рабочим каталогом* (working directory). Метод `toAbsolutePath()` позволяет преобразовать относительный путь в абсолютный, при этом он разрешается относительно текущего каталога.

Пусть в каталоге `/home/user/java` лежит файл `AbsolutePath.class`, откомпилированный из такого файла:

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

public class AbsolutePath {
    public static void main(String[] args) {
        Path relPath = Paths.get("lib/opencsv.jar");
        Path absPath = relPath.toAbsolutePath();
        System.out.println(absPath);
    }
}
```

Тогда мы получим такой результат при запуске этой программы из командной строки (напомним, что команда `cd` устанавливает текущий каталог):

```
$ cd /home/user/java
$ java AbsolutePath
/home/user/java/lib/opencsv.jar
$ cd /tmp
$ java -cp /home/user/java AbsolutePath
/tmp/lib/opencsv.jar
```

Важно! При задании относительных путей из соображений переносимости нежелательно использовать разделитель путей Windows `\`, если только не известно точно, что программа будет запускаться только под Windows. Windows понимает разделитель путей Unix `/`, а вот обратное неверно. Ещё правильнее использовать статический метод `Paths.get` с несколькими параметрами, чтобы не завязываться на конкретный платформозависимый разделитель, а в случае необходимости всё-таки получить этот разделитель в виде строки можно воспользоваться методом `FileSystem.getSeparator`:

```
System.out.println(FileSystem.getDefault().getSeparator());
// Печатает / под Unix и \ под Windows
```

Для преобразования относительного пути в абсолютный относительно не текущего каталога, а какого-то другого, можно воспользоваться методом `resolve`. Его нужно вызвать у того пути, относительно которого мы разрешаем относительный путь. У метода `resolve` есть две версии: одна принимает `Path`, а вторая принимает `String` и рассматривает эту строку как путь, который предстоит разрешить.

`Path resolve(Path other)`

`Path resolve(String other)`

Например:

```
Path configDir = Paths.get("/etc");
System.out.println(configDir.resolve("passwd"));
// /etc/passwd
Path apacheConf = Paths.get("apache2", "apache2.conf");
System.out.println(configDir.resolve(apacheConf));
// /etc/apache2/apache2.conf
```

Обратную задачу решает метод `relativize`, превращающий переданный параметром абсолютный путь в относительный относительно того пути, у которого этот метод вызывается.

`Path relativize(Path other)`

Например:

```
Path homeDir = Paths.get("/home/user");
Path movie = Paths.get("/home/user/Videos/JavaLesson.mkv");
System.out.println(homeDir.relativize(movie));
// Videos/JavaLesson.mkv
```

Переход вверх и вниз по иерархии ФС

Метод `getParent` возвращает родительский элемент ФС (как правило, каталог), или `null`, если такового не существует.

`Path getParent()`

Например:

```
System.out.println(Paths.get("C:\\windows").getParent());
// C:
System.out.println(Paths.get("C:").getParent());
// null
```

А метод `resolve` можно использовать не только для абсолютизации путей, но и для получения пути к какому-либо элементу каталога или какого-то его подкаталога ещё ниже по иерархии ФС:

```
Paths.get("C:\\windows").resolve("explorer.exe")
// C:\windows\explorer.exe
Paths.get("C:\\windows").resolve("system32\\user32.dll")
// C:\windows\system32\user32.dll
```

Есть также метод `resolveSibling`, позволяющий получить "сестринский" элемент ФС, то есть разрешающий переданный путь относительно родительского элемента ФС.

`Path resolveSibling(Path other)`

`Path resolveSibling(String other)`

Например, он позволяет получить путь к другому файлу в том же каталоге:

```
Path dll = Paths.get("C:\\windows\system32\\user32.dll")
System.out.println(dll.resolveSibling("kernel32.dll")
// C:\windows\system32\kernel32.dll
```

Доступ к компонентам пути

Методы `getNameCount` и `getName` позволяют пройтись по всем компонентам пути, представленного объектом `Path`. Эти компоненты сами имеют тип `Path`.

`int getNameCount()`

`Path getName(int index)`

Например:

```
Path path = Paths.get("C:\\windows\explorer.exe");

for (int i = 0; i < path.getNameCount(); i++) {
    System.out.println(path.getName(i));
}
```

Результат:

```
C:
windows
explorer.exe
```

Кроме того, класс `Path` реализует интерфейс `Iterable<Path>` и, следовательно, поддержку цикла for-each, поэтому то же самое делает и такой код:

```
for (Path component: path) {
    System.out.println(component);
}
```

и даже

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

Получение имени файла и каталога

Метод `getFileName` возвращает путь из одного компонента, соответствующего последнему компоненту исходного пути. Обычно это имя файла или другого элемента ФС, на который указывает объект `Path`. Обратите внимание, что этот метод возвращает `Path`, а не `String`.

`Path getFileName()`

Например:

```
Paths.get("/home/user/Pictures/kitty.jpg").getFileName()
// kitty.jpg
Paths.get("C:\\games\World of Warcraft").getFileName()
// World of Warcraft
```

Как мы знаем, получить путь к каталогу, в котором находится файл, можно с помощью `getParent`:

```
Paths.get("/home/user/Pictures/kitty.jpg").getParent()
// /home/user
Paths.get("C:\\games\World of Warcraft").getParent()
// C:\games
```

Работа с расширениями

К сожалению, класс `Path` сам по себе не предоставляет удобных способов работы с расширениями. В некотором смысле это логично, ведь понятие расширения существует только для приложений, работающих с файлами, а для самой ФС это всего лишь часть имени файла. Чтобы отделить расширение, придётся работать с именем файла как со строкой.

Прежде всего эту строку нужно получить:

```
String fileNameStr = path.getFileName().toString();
```

После чего работать с расширениями можно обычными строковыми методами:

```
boolean isPng = fileNameStr.toLowerCase().endsWith(".png");
```

Важно! Обратите внимание на вызов `toLowerCase`. Это гарантирует, что мы проверим имя файла на нужное расширение в любом регистре (`.png`, `.PNG`, `.Png` и т.д.).

Важно! У интерфейса `Path` тоже есть метод `endsWith`, но он проверяет, заканчивается ли *путь* данным компонентом пути, а не заканчивается ли *строка имени файла* данной строкой. Не путайте!

```
Paths.get("/home/user/report.txt").endsWith("report.txt")  // true
Paths.get("report.txt").endsWith(".txt")  // false
Paths.get("report.txt").toString().endsWith(".txt")  // true
```

Наконец, можно написать свой метод, который возвращает расширение имени файла или пустую строку, если у имени файла нет расширения:

```
public String getExtension(Path path) {
    String fileNameStr = path.getFileName().toString();
    int lastDot = fileNameStr.lastIndexOf('.');

    if (lastDot == -1) {
        return "";
    } else {
        return fileNameStr.substring(lastDot + 1);
    }
}
```

Взаимодействие со старыми API

Интерфейс `Path` появился в Java 7. Старые API, спроектированные для Java 6 и ниже, обычно используют вместо него более старый класс `File`, существовавший аж в Java 1.0. Он менее удобен, чем `Path`, потому что поддерживает только стандартную ФС и нарушает принцип единственности ответственности, сочетая в себе методы для синтаксической манипуляции путями и методы доступа к ФС.

Если нужно передать объект `Path` в старое API, используйте метод `toFile`:

`File toFile()`

И, получая объекты `File` из старого API, преобразуйте их в `Path`:

`Path toPath()`

К примеру, класс `ImageIO`, отвечающий за загрузку, сохранение и преобразование изображений, к сожалению, так и не был обновлён для поддержки параметров типа `Path`. Вот так с помощью `ImageIO` можно преобразовать любой поддерживаемый формат изображений в формат PNG:

```
Path source = Paths.get("/home/user/Pictures/diagram.bmp");
BufferedImage image = ImageIO.read(source.toFile());

Path dest = Paths.get("/home/user/Pictures/diagram.png");
ImageIO.write(image, "bmp", dest.toFile());
```