开始探索NIO.2 API的推荐入口点,又被称为"JSR 203":"为Java平台提供了更多的新I/O APIs"(NIO.2),新的抽象类是 java.nio.file.Path.这个类是NIO.2的里程碑(milestone),每个包含I/O操作的应用程序将利用(exploit: 开发、利用)这个类的强大的功能.实际上,它是NIO.2最通用的类,因为许多的I/O操作基于一个Path资源.

Path类支持两种类型的操作: 语法操作(几乎任何包含操纵路径不会访问系统路径的操作;这些是在内存的逻辑操作)和操作的文件路径引用(FIXME: operations over files referenced by paths).这章讨论了第一个类型的操作和向你介绍了Path API.在第4章,将关注于探讨第二个操作类型.这一章所呈现的理念在本书的其余部分也非常有用.

Path类介绍

一个路径驻留在一个文件系统中,以一些媒体的形式存储和组织,通常是一个或多个硬盘驱动器,在这种方式中,它们可以被轻松地检索(*).文件系统可以通过java.nio.file.FileSystems final类访问,用来获取一个我们想要在此上工作的java.nio.file.FileSystem的一个实例.FileSystems包含了以下两个重要的方法,一组newFileSystem()方法集合,(这里分区FileSystem和FileSystems类)用来构造新的文件系统:

- getDefault(): 这个静态方法返回JVM默认的FileSystem-通常的是操作系统的默认文件系统.
- getFileSystem(URI uri): 这个静态方法返回一个匹配给定的URI schema的从一系列可用文件系统集合提供的一个文件系统.Path类维护任意的文件系统(FileSystem)的一个文件,可以使用任意的存储位置(java.nio.file.FileStore;这个类代表了底层存储).默认情况下(通常),Path在默认文件系统(计算机的文件系统)中引用文件.但是NIO.2完全模块化-一个数据的FileSystem的实现在内存中,网络中,或者一个虚拟文件系统中存储数据,这个完全顺从NIO.2.NIO.2为我们提供了所有的文件系统功能,我们可能需要通过一个文件,一个目录或者一个链接执行.

Path类是众所周知的java.io.File类的升级版本,但是File类保留了一些特别的操作,所以它没有被废弃,不能被认为是过时的(obsolete).此外,从Java7开始,两个类都是可用的,这就意味着开发者可用混合它们的功能去获得最好的I/O APIs.Java7提供了一个简单的API来在它们之间转换.还记得你做以下的操作是哪一天吗?

```
import java.io.File;
…
File file = new File("index.html");
```

嗯,那些日子已经过去了,因为Java7可以这样做:

```
import java.nio.file.Path;
import java.nio.file.Paths;
…
Path path = Paths.get("index.html");
```

在仔细地看下,一个Path是一个在文件系统中的路径的编程式表示.路径字符串包含了文件名称,目录列表和OS-依赖文件分隔符(比如: Windows中的反斜杠"\"在和Solaris和Linux中的斜杠"/"在),这就意味着一个Path不是系统依赖的,因为它基于一个系统-依赖字符串路径.因为

Path基本是一个字符串,引用的资源可能并不存在.

---

* Oracle, The Java Tutorials, "What Is a Path? (And Other File System Facts),"
http://download.oracle.com/javase/tutorial/essential/io/path.html.

定义一个路径

一旦你定义了文件系统和文件或者目录的位置,你可以为它创建一个Path对象,绝对路径,相对路径,使用符号"."(表明为当前目录)定义的路径或者".."(表明为上一级目录),路径只包含一个文件/目录的名称,被Path类覆盖(FIXME: are covered by the Path class).定义一个Path的最简单的方式就是调用Paths帮助类的其中一个get()方法.下面的随后的章节展示了几种不同的方式去为相同的文件定义一个路径(Windows)-
C:\rafaelnadal\tournaments\2009\BNP.txt.

定义一个绝对路径

一个绝对路径(也称为一个全路径或者文件路径)是一个包含了根目录的路径,所有的其它的子目录包含一个文件或者目录.在NIO.2中定义一个绝对路径就是一行代码,你可以在下面的示例中看到,指向了名为在C:\rafaelnadal\tournaments\2009目录的BNP.txt的文件(测试这个代码的文件可以不存在):

```
Path path = Paths.get("C:/rafaelnadal/tournaments/2009/BNP.txt");
```

get()方法也允许你将一个路径划分为一组块,NIO将为你重构这个路径,无论有多少块.注意,如果你为路径的每个组件都定义一个块,你可以省略(omit)文件分隔符.之前描述的绝对路径可以如下分块:

```
Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "BNP.txt");
Path path = Paths.get("C:", "rafaelnadal/tournaments/2009", "BNP.txt");
Path path = Paths.get("C:", "rafaelnadal", "tournaments", "2009", "BNP.txt");
```

定义一个相对于文件存储根目录的路径

一个相对路径(也被称为非绝对路径或者局部路径)只是一个全路径的一部分(portion).一个相对路径通常使用在创建一个web网页.相对路径比绝对路径使用更加频繁.定义一个到当前文件存储根目录的相对路径应该以文件分隔符开始.在下面的示例中,如果当前文件存储根目录是C:,绝对路径是C:\rafaelnadal\tournaments\2009\BNP.txt:

```
Path path = Paths.get("/rafaelnadal/tournaments/2009/BNP.txt");
Path path = Paths.get("/rafaelnadal","tournaments/2009/BNP.txt");
```

定义一个相对于工作目录的路径

当你定义一个相对于工作目录的路径,路径不应该以文件分隔符开始.如果当前目录是位于C:根目录下的/ATP,下面代码片段返回的绝对路径是

C:\ATP\rafaelnadal\tournaments\2009\BNP.txt:

```
Path path = Paths.get("rafaelnadal/tournaments/2009/BNP.txt");
Path path = Paths.get("rafaelnadal","tournaments/2009/BNP.txt");
```

使用快捷方式定义一个路径

使用符号"."(表明为当前目录)或者".."(表明为上级目录)定义路径是一个常规做法(a common practice).这种类型的路径可以被NIO.2处理,消除(eliminate)可能的冗余(redundancy)情况如果你调用Path.normalize()方法(这个方法将去除任何冗余的元素,包含任何出现的"."或者"目录/.."):

```
Path path = Paths.get("C:/rafaelnadal/tournaments/2009/dummy/../BNP.txt").normalize();
Path path = Paths.get("C:/rafaelnadal/tournaments/./2009/dummy/../BNP.txt").normalize();
```

如果你想看到normalize()方法的效果,尝试去使用和不使用normalize()定义相同的路径,如下,打印结果到屏幕上:

```
Path noNormalize = Paths.get("C:/rafaelnadal/tournaments/./2009/dummy/../BNP.txt");
Path normalize =
Paths.get("C:/rafaelnadal/tournaments/./2009/dummy/../BNP.txt").normalize();
```

如果你使用System.out.println()去打印之前的路径,你看看到如下的结果,normalize()已经溢出了冗余的元素:

```
C:\rafaelnadal\tournaments\.\2009\dummy\..\BNP.txt
C:\rafaelnadal\tournaments\2009\BNP.txt
```

从一个URI中定义一个路径

在某些情况下,你可能需要去从一个统一资源定位符(URI)中创建一个Path.你可以通过使用URI.create()方法从给定的字符串创建一个URI然后通过使用Paths.get()方法将一个URI对象作为一个参数完成.这是有用的当你需要去封装一个路径字符串,可以在一个web浏览器的地址栏中进入:

```
import java.net.URI;
…
Path path = Paths.get(URI.create("file:///rafaelnadal/tournaments/2009/BNP.txt"));
Path path = Paths.get(URI.create("file:///C:/rafaelnadal/tournaments/2009/BNP.txt"));
```

使用FileSystems.getDefault().getPath()方法定义一个路径

另一个创建一个Path的常用方式是使用FileSystems类.首先,调用getDefault()方法去获得默认的FileSystem-NIO.2将提供一个通用的对象,具有访问默认文件系统的能力.然后,你可以调用getPath()方法,如下(之前的示例中的Paths.get()方法)是这种方式的速写法(shorthand)):

```
import java.nio.file.FileSystems;
…
Path path = FileSystems.getDefault().getPath("/rafaelnadal/tournaments/2009", "BNP.txt");
Path path = FileSystems.getDefault().getPath("/rafaelnadal/tournaments/2009/BNP.txt");
Path path = FileSystems.getDefault().getPath("rafaelnadal/tournaments/2009", "BNP.txt");
Path path = FileSystems.getDefault().
getPath("/rafaelnadal/tournaments/./2009","BNP.txt").normalize();
```

获取根目录的路径

当你需要一个指向根目录的路径,你可以如下面示例所展示的处理(返回的根目录依赖于每种机器或每种操作系统):

```
Path path = Paths.get(System.getProperty("user.home"), "downloads", "game.exe");
```

在我的Windows 7机器上,这个返回 C:\Users\Leo\downloads\game.exe,在我的朋友的CentOS系统(Linux),这个返回 /home/simpa/downloads/game.exe.

获取关于一个路径的信息

在你定义个Path对象之后,你可以访问一组方法,这组方法提供了关于路径元素的有用的信息.这些方法基于实际的,NIO,2分隔路径字符串为一组元素(一个元素是一个子路径代表了一个目录或者文件),分配索引0给最高的元素,索引index-1给最低的元素,n是路径元素的数目;通常,最高元素是根目录,最低元素是一个文件.这节展示了示例,应用这些信息-获取方法到路径C:\rafaelnadal\tournaments\2009\BNP.txt:

```
Path path = Paths.get("C:", "rafaelnadal/tournaments/2009", "BNP.txt");
```

获取路径文件/目录的名称

文件/目录通过一个路径的getFileName()方法表明(FIXME: The file/directory indicated by a path is returned by the getFileName() method),为在目录层次结构中从根目录获取的最远的元素:

```
//output: BNP.txt
System.out.println("The file/directory indicated by path: " + path.getFileName());
```

获取根路径

路径的根路径可以使用getRoot()方法获取(如果路径没有根目录,返回null):

```
//output: C:\
System.out.println("Root of this path: " + path.getRoot());
```

获取上级路径

这个路径的上级路径(路径的根组件)通过getParent()方法返回(如果路径没有上级目录,返回null):

```
//output: C:\rafaelnadal\tournaments\2009
System.out.println("Parent: " + path.getParent());
```

获取路径名称元素

你可以通过getNameCount()方法获取在一个路径中元素的数目,使用getName()方法获得每个元素的名称:

```
//output: 4
System.out.println("Number of name elements in path: " + path.getNameCount());

//output: rafaelnadal  tournaments  2009  BNP.txt
for (int i = 0; i < path.getNameCount(); i++) {
  System.out.println("Name element " + i + " is: " + path.getName(i));
}
```

获取一个路径的子路径

你可以使用subpath()方法去的一个相对路径,这个方法需要两个参数,开始索引和结束索引,代表了后续的元素(FIXME: subsequence of elements):

```
//output: rafaelnadal\tournaments\2009
System.out.println("Subpath (0,3): " + path.subpath(0, 3));
```

转换一个路径

在这节,你将看到如何转换一个Path为一个字符串,一个URI,一个绝对路径,一个真实的路径,一个File对象.Path类为这些转换包含了一个专门的方法,如下面的随后章节展示的.以下是我们将要工作的路径:

```
Path path = Paths.get("/rafaelnadal/tournaments/2009", "BNP.txt");
```

转换一个路径为一个字符串

一个路径的字符串转换可通过toString()方法完成:

```
//output: \rafaelnadal\tournaments\2009\BNP.txt
String path_to_string = path.toString();
System.out.println("Path to String: " + path_to_string);
```

转换一个路径为一个URI

你可以应用toURI()方法转换一个Path为一个web浏览器格式字符串,如下面的示例所展示的.结果是一个URI对象封装了一个路径字符串,可以通过一个web浏览器的地址栏进入:

```
//output: file:///C:/rafaelnadal/tournaments/2009/BNP.txt
URI path_to_uri = path.toUri();
System.out.println("Path to URI: " + path_to_uri);
```

转换一个相对路径为一个绝对路径

从一个相对路径获取绝对路径是一个非常常见的任务.NIO.2可使用toAbsolutePath()方法完成(注意如果你应用这个方法到一个已经是绝对路径的路径上,将返回相同的路径):

```
//output: C:\rafaelnadal\tournaments\2009\BNP.txt
Path path_to_absolute_path = path.toAbsolutePath();
System.out.println("Path to absolute path: " + path_to_absolute_path.toString());
```

转换一个路径为一个真实路径

toRealPath()方法返回一个已经存在的文件的真实路径-这意味着文件必须存在,如果你使用toAbsolutePath()方法的话这是没必要的(文件可以不存在).如果没有参数传递到这个方法,文件系统支持符号链接,这个方法解析在这个路径中的任意的符号链接.如果你想忽略符号链接,传递LinkOption.NOFOLLOW_LINKS枚举常量到这个方法.此外,如果Path是相对的,它返回绝对路径,如果Path包含任何冗余元素,它返回移除这些元素的路径.这个方法抛出IOException,如果文件不存在或者不能被访问.

下面的代码片段返回一个文件的真实路径,没有按照符号链接(FIXME: by not following symbolic links):

```
import java.io.IOException;
…
//output: C:\rafaelnadal\tournaments\2009\BNP.txt
try {
    Path real_path = path.toRealPath(LinkOption.NOFOLLOW_LINKS);
    System.out.println("Path to real path: " + real_path);
} catch (NoSuchFileException e) {
    System.err.println(e);
} catch (IOException e) {
    System.err.println(e);
}
```

转换一个路径为一个文件

一个Path可以使用toFile()方法转换一个File对象,如下.这个在Path和File的一个重大的桥接(FIXME: a great bridge),因此File类也包含了一个用来转换的toPaht()方法.

```
//output: BNP.txt
File path_to_file = path.toFile();
//output: \rafaelnadal\tournaments\2009\BNP.txt
Path file_to_path = path_to_file.toPath();
System.out.println("Path to file name: " + path_to_file.getName());
System.out.println("File to path: " + file_to_path.toString());
```

组合两个路径

组合两个路径是一种允许你定义一个固定的根路径和将一个部分路径追加到它的技术.这个对于基于一个共同的部分定义路径非常有用.NIO.2通过resolve()方法提供了额这个操作.下面是一个它如何工作的示例:

```
//define the fixed path
Path base = Paths.get("C:/rafaelnadal/tournaments/2009");

//resolve BNP.txt file
Path path_1 = base.resolve("BNP.txt");
//output: C:\rafaelnadal\tournaments\2009\BNP.txt
System.out.println(path_1.toString());

//resolve AEGON.txt file
Path path_2 = base.resolve("AEGON.txt");
//output: C:\rafaelnadal\tournaments\2009\AEGON.txt
System.out.println(path_2.toString());
```

也有一个方法专门用于兄弟路径(FIXME: There is also a method dedicated to sibling paths),名为resolveSibling().它针对当前路径的上级路径解析传递的路径.特别是,这个方法使用给定路径的文件名称替换当前路径的文件名称.

下面的示例说明了这个概念:

```
//define the fixed path
Path base = Paths.get("C:/rafaelnadal/tournaments/2009/BNP.txt");

//resolve sibling AEGON.txt file
Path path = base.resolveSibling("AEGON.txt");
//output: C:\rafaelnadal\tournaments\2009\AEGON.txt
System.out.println(path.toString());
```

在两个位置间构造一个路径

当你需要构造一个路径从一个位置到另一个,你可以调用relarivize()方法,这个方法会在这个路径和一个给定的路径之间构造一个相对路径.这个方法构造一个路径起源于原始的路径,终止于指定的传递路径的位置.新的路径相对于原始的路径.为了更好地理解这个强大的功能,思考一个简单的示例.假设你有如下两个相对路径:

```
Path path01 = Paths.get("BNP.txt");
Path path02 = Paths.get("AEGON.txt");
```

在这种情况下,假设BNP.txt和AEGON.txt是兄弟,这意味着你可以通过上级一个级别和下降一个级别从一个导航到另一个.应该relativize()方法输出..\AEGON.txt和..\BNP.txt:

```
//output:  ..\AEGON.txt
Path path01_to_path02 = path01.relativize(path02);
System.out.println(path01_to_path02);

//output:  ..\BNP.txt
Path path02_to_path01 = path02.relativize(path01);
System.out.println(path02_to_path01);
```

另一种典型的解决方案包含两个路径(包含了一个跟元素).思考下面的路径:

```
Path path01 = Paths.get("/tournaments/2009/BNP.txt");
Path path02 = Paths.get("/tournaments/2011");
```

在这种情况下,两个路径包含了相同的根元素:/tournaments.为了从path01导航到path02,你将上升两个级别和下降一个级别(..\..\2011).为了从path02导航到path01,你将上升一个基本和下降两个级别(..\2009\BNP.txt).这个准确的说明了relativize()方法是如何工作的:

```
//output:  ..\..\2011
Path path01_to_path02 = path01.relativize(path02);
System.out.println(path01_to_path02);

//output:  ..\2009\BNP.txt
Path path02_to_path01 = path02.relativize(path01);
System.out.println(path02_to_path01);
```

注意:如果只有一个路径包含了根元素,相对路径不能被构造.两个路径必须都包含一个根元素.即使那样,相对路径的构造是系统依赖的.

比较两个路径

两个Path的等同可以为不同的目的而以不同的方式测试.你可以通过调用Path.equals()方法测试两个路径是否相同.这个方法遵循Object.equals()规范.它不范文文件系统,所以比较的路径不需要存在,它不会检查路径是否为同一个文件.在一些操作系统实现上,路径的比较忽略大小写,同时在其它的实现上,比较是大小写敏感的-实现将制定大小写是否应该被考虑.在这里我展示了一个相对于当前文件存储的路径和一个绝对路径,两个代表了相同的文件,但是不相同.

```
Path path01 = Paths.get("/rafaelnadal/tournaments/2009/BNP.txt");
Path path02 = Paths.get("C:/rafaelnadal/tournaments/2009/BNP.txt");
if(path01.equals(path02)){
    System.out.println("The paths are equal!");
} else {
    System.out.println("The paths are not equal!"); //true
}
```

有时候你想去检查两个路径是否为同一个文件/目录.你可以简单地通过调用java.nio.File.Files.isSameFile()方法完成(accomplish)(如下面的示例所展示的),这个方法返回一个boolean值.在幕后(Behind the scenes),这个方法使用了Path.equals()方法.如果Path.equals()返回true,路径是相同的,因此不需要进一步地2比较.如果它返回false,然后isSameFile()方法进入双-检查.注意,这个方法需要比较的文件存在于文件系统;否则,它抛出IOException.

```
try {
    boolean check = Files.isSameFile(path01, path02);
    if(check){
        System.out.println("The paths locate the same file!"); //true
```

```
    } else {
        System.out.println("The paths does not locate the same file!");
    }
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

因为Path类实现了Comparable接口,你可以通过使用compareTo()方法比较路径,比较两个抽象路径的字母顺序(FIMXE: compares two abstract paths lexicographically).这个对于排序非常有用.这个方法返回0如果参数等于这个路径,小于0如果路径字母顺序小于参数,或者大于0如果路径的字母顺序大于参数.下面的示例使用了compareTo()方法:

```
//output: 24
int compare = path01.compareTo(path02);
System.out.println(compare);
```

部分比较可以通过使用startsWith()和endsWith()方法完成,如下面示例所示.使用这些方法,你可以分别测试是否以当前路径开始或结束.两个放阿飞返回布尔值:

```
boolean sw = path01.startsWith("/rafaelnadal/tournaments");
boolean ew = path01.endsWith("BNP.txt");
System.out.println(sw);  //output:  true
System.out.println(ew);  //output:  true
```

遍历一个路径的名称元素

因为Path类实现了Iterable接口,你可以获取一个对象运行你去在一个路径中遍历元素.你可以通过使用明确的迭代器或者使用一个foreach循环每次遍历返回一个Path对象来遍历.下面是一个示例:

```
Path path = Paths.get("C:", "rafaelnadal/tournaments/2009", "BNP.txt");
for (Path name : path) {
    System.out.println(name);
}
```

这个输出从最靠近根元素的元素开始,如下:

```
rafaelnadal
tournaments
2009
BNP.txt
```

总结

在这章,你迈出你的第一步到NIO.2 API.除了学习关于基本的NIO.2概念,比如文件系统和文件

存储.你概览了Path类,这个知识对于每个想去学习如何使用NIO.2 API是重要的.知道如何去获取默认文件系统和如何定义和管理文件路径是重要的,因为Path类贯穿全书用一些示例证明(FIXME: Path class will sustain the examples throughout the book),它通常是应用程序是入口点.