http://wiki.jikexueyuan.com/project/java-nio-zh/java-nio-path.html

Java的path接口是作为Java NIO 2的一部分是Java6,7中NIO的升级增加部分。Path在Java 7新增的。相关接口位于java.nio.file包下，所以Javaz内Path接口的完整名称是java.nio.file.Path.

一个Path实例代表一个文件系统内的路径。path可以指向文件也可以指向目录。可以使相对路径也可以是绝对路径。绝对路径包含了从根目录到该文件（目录）的完整路径。相对路径包含该文件（目录）相对于其他路径的路径。相对路径听起来可能有点让人头晕。但是别急，稍后我们会详细介绍。

不要把文件系统中路径和环境变量的路径混淆。java.nio.file.Path和环境变量没有任何关系。

在很多情况下java.no.file.Path接口和java.io.File比较相似，但是他们之间存在一些细微的差异。尽管如此，在大多数情况下，我们都可以用File相关类来替换Path接口。

创建Path实例（Creating a Path Instance）

为了使用java.nio.file.Path实例我们必须创建Path对象。创建Path实例可以通过Paths的工厂方法get（）。下面是一个实例:

```
import java.nio.file.Path;
import java.nio.file.Paths;
public classs PathExample {
  public static void mian(String[] args) {
    Path path = Paths.get("c:\\data\\myfile.txt");
  }
}
```

注意上面的两个import声明。需要使用Path和Paths的接口,必须先把他们引入。

其次注意Paths.get("c:\data\myfile.txt")的调用。这个方法会创建一个Path实例，换句话说Paths.get()是Paths的一个工厂方法。

创建绝对路径（Creating an Absolute Path）

创建绝对路径只需要调动Paths.get()这个工厂方法，同时传入绝对文件。这是一个例子:

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

对路径是c:\data\myfile.txt，里面的双斜杠\字符是Java 字符串中必须的，因为\是转义字符，表示后面跟的字符在字符串中的真实含义。双斜杠\表示\自身。

上面的路径是Windows下的文件系统路径表示。在Unixx系统中（Linux, MacOS,FreeBSD等）上述的绝对路径是这样的：

```
Path path = Paths.get("/home/jakobjenkov/myfile.txt");
```

他的绝对路径是/home/jakobjenkov/myfile.txt。 如果在Windows机器上使用用这种路径，那么这个路径会被认为是相对于当前磁盘的。例如：

```
/home/jakobjenkov/myfile.txt
```

这个路径会被理解其C盘上的文件，所以路径又变成了

```
C:/home/jakobjenkov/myfile.txt
```

创建相对路径（Creating a Relative Path）

相对路径是从一个路径（基准路径）指向另一个目录或文件的路径。完整路径实际上等同于相对路径加上基准路径。

Java NIO的Path类可以用于相对路径。创建一个相对路径可以通过调用Path.get(basePath, relativePath),下面是一个示例：

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

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

第一行创建了一个指向d:\data\projects的Path实例。

第二行创建了一个指向d:\data\projects\a-project\myfile.txt的Path实例。

在使用相对路径的时候有两个特殊的符号：

- .
- ..

.表示的是当前目录，例如我们可以这样创建一个相对路径：

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

currentDir的实际路径就是当前代码执行的目录。 如果在路径中间使用了.那么他的含义实际上就是目录位置自身，例如：

```
Path currentDir = Paths.get("d:\\data\\projects\.\a-project");
```

上诉路径等同于：

```
d:\data\projects\a-project
```

..表示父目录或者说是上一级目录：

```
Path parentDir = Paths.get("..");
```

这个Path实例指向的目录是当前程序代码的父目录。 如果在路径中间使用..那么会相

应的改变指定的位置：

```
String path = "d:\\data\\projects\\a-project\\..\\another-project";
Path parentDir2 = Paths.get(path);
```

这个路径等同于：

```
d:\data\projects\another-project
```

.和..也可以结合起来用，这里不过多介绍。

Path.normalize()

Path的normalize()方法可以把路径规范化。也就是把.和..都等价去除：

```
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);
```

这段代码的输出如下：

```
path1 = d:\data\projects\a-project\..\another-project
path2 = d:\data\projects\another-project
```