i++与++i有什么不同，重写与隐藏的本质区别是什么
你是否留意过这些细节

本书带你走近这些不被人注意的细节中
一起来感受“细节决定成败”的力量

细节Java

梁勇 编著

人民邮电出版社
POSTS & TELECOM PRESS

细说Java

梁勇 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

细说Java / 梁勇编著. -- 北京 : 人民邮电出版社, 2010.7
ISBN 978-7-115-22694-5

Ⅰ. ①细… Ⅱ. ①梁… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2010)第060906号

内 容 简 介

本书详细讲解了 Java 开发的近百个细节，主要内容包括 path 与 classpath、main、基本数据类型、注释、运算符危机、循环、数组、参数的传递、访问修饰符、私有的意义、构造器、变量、方法重载、垃圾回收、自动封箱与拆箱、继承、重写与隐藏、接口、嵌套类型、遮蔽、泛型、System 类、Arrays 类、Object 类、String 类等。

本书用生动、形象的语言讲解 Java 开发的关键点，使读者透彻理解开发细节，加深对 Java 的认识。

本书适合对 Java 开发感兴趣的读者阅读，也可作为 Java 程序员的参考用书。

细说 Java

◆ 编　　著　梁　勇
责任编辑　黄　焱

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：22.25
字数：528 千字　　2010 年 7 月第 1 版
印数：1－4 000 册　　2010 年 7 月北京第 1 次印刷

ISBN 978-7-115-22694-5

定价：39.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

Forward 前言

不可否认，从 1995 年 Java 语言诞生至今，在短短十几年的时间里，Java 语言已经成为最流行的语言之一，它的简单、安全、多线程以及跨平台等特点吸引着越来越多的用户。然而，在众多 Java 书籍中，介绍 Java 使用细节的图书并不多，或者只是说明了“应该如此”，却没有阐明“为什么如此”。很多读者在学习 Java 后，表面上已经理解并掌握了 Java 的语法和应用，但实际使用起来对某些概念的细节仍是一头雾水，如在数组定义时，[]的位置（在前或者在后）有什么不同？5 % 0 与 5 / 0 两个表达式运行的结果相同吗（或是抛出相同的异常吗）？0.0 与-0.0 相等吗？什么时候相等，什么时候不相等呢？

类似上述的细节不胜枚举，不知你是否留意过。如果你没有注意过也不要紧，本书会带你走入那些不被人注意却容易犯错误的细节中。不要小瞧了这一个个小知识，所谓“不积跬步，无以致千里，不积小流，无以成江海”，正是今天一点一滴的积累，才能有明天惊人的成就。

在本书中，笔者凭借多年从事 Java 开发的经验，为你指出各种容易被忽视且容易出错的概念以及语法，并指出“为什么如此”。我们不仅要“知其然”，更要“知其所以然”。

本书分为 4 篇，具体内容如下。

第 1 篇　Java 语言基础，包括第 1 章～第 8 章。主要介绍了 path 与 classpath、main、基本数据类型、注释、运算符危机、循环、数组、参数的传递等内容。

第 2 篇　类与对象，包括第 8 章～第 15 章。主要介绍了访问修饰符、私有的意义、构造器、变量、方法重载、垃圾回收、自动封箱与拆箱等内容。

第 3 篇　继承与应用，包括第 16 章～第 21 章。主要介绍了继承、重写与隐藏、接口、嵌套类型、遮蔽、泛型等内容。

第 4 篇　重要类，包括第 22 章～第 25 章。主要介绍了 System 类、Arrays 类、Object 类、String 类等内容。

由于时间仓促，加之笔者的水平有限，书中疏漏之处在所难免，希望广大读者批评指证，笔者的邮箱为：floatview@hotmail.com。

注意：如无特殊说明，本书的程序均在 JDK 1.6 上调试运行。

编者

2010 年 3 月

Content

目录

第 1 篇　Java 语言基础

第 3 篇 继承与应用

第 16 章 继承

第 17 章 重写与隐藏

第 18 章 接口

第 19 章 嵌套类型

第 20 章 遮蔽

第 21 章 泛型

第 4 篇 重 要 类

第 22 章 System

第 23 章 Arrays

第 24 章 Object

第 25 章 String

第 1 篇

Java 语言基础

Chapter

1

第1章

path 与 classpath

引言：

关于 path 与 classpath，可以说是学习 Java 程序设计的第一课，如果这两个环境变量配置不当，就会影响 Java 程序的编译与运行。也许有的读者认为这非常简单，不过是一个路径的复制、粘贴而已，说的没错，但是在复制、粘贴的背后，你是否注意到以下的细节……

本章要点：

- 通过“环境变量”对话框配置环境变量
- 通过“命令提示符”配置环境变量
- path 与 classpath 的作用
- path 与 classpath 路径的搜索顺序
- classpath 环境变量与-cp(-classpath)参数的优先级

1.1　细节 1　异曲同工——两种配置方式

可以通过两种方式来配置环境变量，不同的方式产生的效果也有所不同。这里我们以 Windows Vista 系统来举例说明。

1. 通过“环境变量”对话框配置

（1）在“我的电脑”上单击鼠标右键，在弹出的快捷菜单中选择“属性”，打开“系统属性”对话框，选择“高级”选项卡，然后单击页面中的“环境变量”按钮，弹出如图 1-1 所示对话框。其中“用户变量”只对当前用户有效，而“系统变量”对所有用户都有效（这里我们设置系统变量）。

（2）在“系统变量”框中选中 Path（如果没有就新建一个），单击【编辑】按钮，弹出如图 1-2 所示“编辑系统变量”对话框。将<JDK 安装目录>\bin（笔者的 JDK 安装目录为 C:\jdk1.6）目录的完整路径名复制到变量值的最前面，以“;”作为间隔，单击【确定】按钮即可。

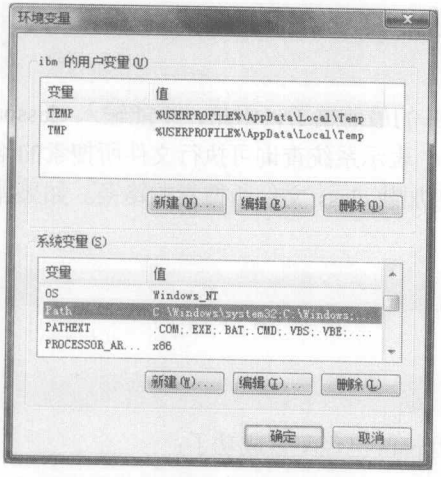

图 1-1　“环境变量”对话框

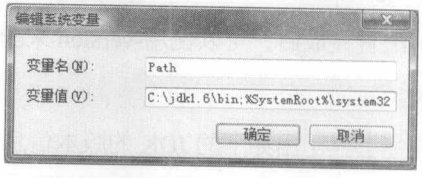

图 1-2　“编辑系统变量”对话框

注意:

从 JDK 1.2 以后，classpath 默认为当前目录，如果只是测试一些简单的小程序，就不必配置 classpath 了。如果在含有 jar 包等复杂的情况下，只需单击【新建】按钮，在弹出的对话框的“变量名”文本框中输入“classpath”，在“变量值”文本框中输入需要的路径就可以了。

2. 通过“命令提示符”配置

（1）打开命令提示符窗口，输入设置 path 的命令：

```
set path=C:\jdk1.6\bin;%path%
```

（2）输入设置 classpath 的命令：

```
set classpath=C:\jdk1.6\example;%classpath%
```

注意：

“=”两端不能有空格。%path%（%classpath%）是以前 path（classpath）的值，如果不加%path%（%classpath%），那么我们设置的路径就会覆盖以前 path（classpath）的值。

（3）设置好之后，输入：

```
path
```

结果如下：

```
PATH=C:\jdk1.6\bin;C:\WINDOWS\system32;C:\WINDOWS;
……
```

可以看出，C:\jdk1.6\bin 已经成功地加入到 path 的最前端了，但是，不能输入 classpath 来查看 classpath 的值，因为 path 为 DOS 的内部命令，表示系统查询可执行文件所搜索的路径，而 classpath 并非 DOS 的内部命令，表示 Java 虚拟机加载.class 文件所搜索的路径。如果需要显示 classpath 的值，可以使用“echo”命令：

```
echo %classpath%
```

（4）配置完成后，可以使用-version 来验证：

```
java –version
```

如果能够显示出如下的 JDK 的版本信息，证明 path 已经配置成功了：

```
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Client VM (build 11.0-b16, mixed mode, sharing)
```

注意：

第一种配置方式是永久的，就算系统重新启动后配置依然生效。第二种是临时的，只对当前的 DOS 窗口生效。如果将当前的窗口关闭，那么配置将丢失。

1.2　细节 2　仙人指路——path 的作用

1. 通过 path 路径寻找 javac 与 java 命令

大家知道，在配置 path 之前，如果我们输入如下命令：

```
javac
```

那么系统会给出如下错误信息，原因是系统无法识别 javac 是什么：

```
'javac' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

为什么要配置 path 后才能使用 javac、java 等命令呢？既然我们在 path 中配置的是<JDK 安装目录>\bin 目录，那就打开看一看里面的文件内容。从图 1-3 所示可以发现，javac.exe、java.exe 等文件都在这个文件夹中，原来这些命令都对应着同名的可执行文件。

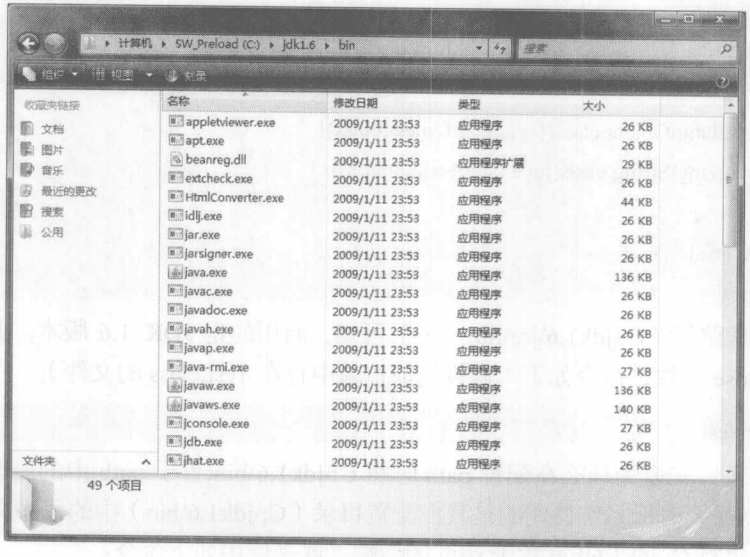

图 1-3　bin 目录中的文件

所以，我们要在 path 中指出 Java 编译运行时需要用到的可执行文件（javac.exe，java.exe 等）所在的路径。例如，当使用 javac 命令时，系统便从 path 指定的路径中搜索 javac.exe。记住：path 变量是系统寻找可执行文件的搜索路径。

2. 多个 JDK 并存时的选择

如果一个系统中安装多个版本的 JDK，比如同时安装了 JDK 1.6 与 JDK 1.5 两个版本，而在

path 中又同时配置了这两个 JDK 安装目录下 bin 目录的路径，如在命令提示符中输入（或者在环境变量窗口修改）：

```
set path=C:\jdk1.6\bin;C:\jdk1.5\bin;%path%
```

那么当程序运行时，会启用哪个版本的 JDK 呢？

实际上，系统在搜索 path 路径时，会按照从前至后的顺序，如果找到了想要的文件，就停止搜索。所以，如果按照上面的方式来配置 path，那么将会启用 C:\jdk1.6\bin 下的相关程序。可以输入如下命令来查看究竟调用的是哪个版本（假定当前目录下存在一个名为 Test.java 的源文件）：

```
javac –verbose Test.java
```

当 Java 虚拟机（JVM）启动时，就会显示其详细的加载过程信息。大致情况如下：

```
[解析开始时间 Test.java]
[解析已完成时间 47ms]
[源文件的搜索路径： .]
[类文件的搜索路径：
C:\jdk1.6\jre\lib\resources.jar,C:\jdk1.6\jre\lib\rt.jar,C:\
……]
[正在装入 java\lang\Object.class(java\lang:Object.class)]
[正在装入 java\lang\String.class(java\lang:String.class)]
……
[已写入 Test.class]
[总时间 844ms]
```

从显示的搜索路径（C:\jdk1.6\jre\lib\……）可知，启用的是 JDK 1.6 版本。此外，也可以在运行时使用-verbose 参数，命令如下（假设当前目录中存在 Test.class 的文件）：

```
java –verbose Test
```

现在回想一下，知道为什么在配置 path 时将 C:\jdk1.6\bin;置于 path 中的最前端了吧？（见图 1-2）对了，就是要确保系统使用的是我们配置目录（C:\jdk1.6\bin）中的程序文件（javac.exe、java.exe 等），如果将 C:\jdk1.6\bin 置于 path 的后端，或是使用如下命令：

```
set path=%path%;C:\jdk1.6\bin
```

那么就不能保证系统选择的一定是 C:\jdk1.6\bin 目录下的程序文件了，因为 path 中之前已经存在指向其他版本的 JDK 路径了。例如，以前的 path 中含有指向 JDK 1.5 版本的路径（C:\JDK1.5\bin），那么，如果单纯地在 path 后面追加 JDK 1.6 版本的路径是起不到任何作用的。因为 path 的搜索路径是从前向后，所以必然是 JDK 1.5 版本的路径（C:\JDK1.5\bin）优先满足条件而返回。

1.3　细节 3　寻觅 class——classpath 的作用

1. 通过 classpath 路径寻找 class 文件

顾名思义，classpath 就是为 Java 虚拟机寻找.class 文件（也称为类文件）所在的路径。javac 命令默认是在当前目录下生成编译后的.class 文件（也可以使用-d 选项来指定生成.class 文件的目录），而 classpath 默认的又是当前的目录，也就是意味着 classpath 总是引导 Java 虚拟机在当前目录中寻找并加载生成在当前目录下的.class 文件，所以运行没有指定包（package）的程序（或是运行时用到了第三方的 jar 包等复杂情况）时，就不需设置 classpath 了。

Java 虚拟机只会搜索 classpath 指定路径下的.class 文件，如果是.jar 文件，那么需要配置包含.jar 文件名称的完整路径才行。例如，如果程序既用到了 D:\classes\A.class 文件，又用到了 D:\classes\B.jar 文件中的.class 文件，那么就需要将 classpath 配置为：

```
D:\classes; D:\classes\B.jar
```

绝不要指望只配置 D:\classes 虚拟机就可以找到 B.jar 中的*.class 文件。

classpath 与 path 类似，也是按照从前向后的顺序进行搜索，如果找到指定的类文件（.class 文件），就停止搜索。

另外，除了修改环境变量 classpath 外，也可以在编译（或运行）时使用-cp 或-classpath 命令来指定.class 文件所在的路径。下面以一个小程序来说明。

【例 1.1】　classpath 测试程序

```
1.  public class ClasspathTest {
2.      public static void main(String[] args) {
3.          System.out.println("classpath test");
4.      }
5.  }
```

我们将文件存放在 D:\example 目录下，编译并运行这个程序，输出结果如下：

```
classpath test
```

现在，我们将当前目录切换到 D:\下，输入：

```
cd ..
```

再次运行程序，很不幸，运行失败了，错误信息如下：

```
Exception in thread "main" java.lang.NoClassDefFoundError: ClasspathTest
Caused by: java.lang.ClassNotFoundException: ClasspathTest
```

```
        at java.net.URLClassLoader$1.run(URLClassLoader.java:200)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:252)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:320)
Could not find the main class: ClasspathTest.  Program will exit.
```

系统提示找不到指定的类文件“ClasspathTest”，这是因为我们没有设置 classpath 环境变量，其默认路径为当前目录，而当前目录（D:\）中根本找不到名为 ClasspathTest.class 的文件（因为其真正路径为 D:\example），所以会出现错误。

如果.class 文件不在当前目录下，而又不想将当前目录切换到.class 文件所在的目录来运行该文件时，就需要 classpath 来辅助了。输入如下的命令：

```
set classpath=D:\example
java ClassPathTest
```

可以看出，这次正常了，也可以不设置环境变量 classpath，而是使用-cp 或-classpath 来指定.class 文件所在的路径，效果是相同的。如：

```
java –cp D:\example ClasspathTest
java ClasspathTest
```

2. classpath 环境变量与-cp(-classpath)的优先级

现在问题来了，既然修改 classpath 与使用-cp 参数都能用来指定.class 文件的路径，那么当二者设置的路径不同时，以哪个为准呢？我们输入如下命令来测试（当前目录为 D:\）：

```
set classpath=C:\
java –cp D:\example ClasspathTest
```

程序运行正常，接着再输入：

```
set classpath=D:\example
java –cp C:\ ClasspathTest
```

运行出错，系统提示找不到.class 文件。

这里 C:\只是一个干扰路径而已，事实上，你可以指定任何一个不相关的路径。从运行结果可以看出，将 classpath 设置为错误的.class 文件所在路径而将-cp 参数设置正确时，可以正常运行，而反之则产生错误，这表明，-cp 参数所指定的路径会覆盖 classpath 环境变量的路径，当二者同时存在时，以-cp 指定的路径为准。

1.4 本章小结

path 为系统指定相关程序（如 javac.exe、java.exe 等）所在的路径，classpath 为 Java 虚拟机指定加载的.class 文件所在的路径。二者既可以在“环境变量”中配置，也可以通过“命令提示符”配置，两种配置有所不同，前者所做的配置可以长期保存，而后者所做的配置只对当前的窗口生效。在搜索路径时二者都遵从由前向后的原则，一旦找到了符合条件的路径就不再继续。如果既配置了 classpath 环境变量，又使用了-cp（-classpath）参数，那么以-cp（-classpath）参数为准。