

买车vs不买车 (1750)

在MyEclipse8.0上安装Cl (1683)

评论排行

前缀、中缀、后缀表达式 (39)

Java开发和运行环境的搭 (7)

怎么用HTML5 Canvas制 (4)

Flash中的注册点和中心) (2)

开通博客纪念 (1)

要爱惜自己的身体 (1)

利比亚内战始末大事记 (0)

买车vs不买车 (0)

Java代码变成位图是什么 (0)

C运算符的结合方向 (0)

推荐文章

* CSDN日报20170725——《新的开始，从研究生到入职亚马逊》

* 深入剖析基于并发AQS的重入锁(ReentrantLock)及其Condition实现原理

* Android版本的"Wannacry"文件加密病毒样本分析(附带锁机)

* 工作与生活真的可以平衡吗？

* 《Real-Time Rendering 3rd》提炼总结——高级着色：BRDF及相关技术

* 《三体》读后思考-泰勒展开/维度打击/黑暗森林

最新评论

前缀、中缀、后缀表达式
luanqibaazao: 楼主 可以转载吗

前缀、中缀、后缀表达式
qq_37996354: 如果是-5*8+7*{8.2/(4-2)}呢，博主能否提供一个解决负数的方法

Java开发和运行环境的搭建
淘到星星的小管子: 楼主写的好详细啊，可以出教程了！感谢楼主分享！

前缀、中缀、后缀表达式
JieRiTian: 最近被这几个缀弄乱了，看了楼主的写的很详细。先收藏了，多谢楼主！

前缀、中缀、后缀表达式
hello_taozi: 太棒惹！！

前缀、中缀、后缀表达式
GongchuangSu: 写的好详细，谢谢楼主

前缀、中缀、后缀表达式
GongchuangSu: @caichongjun: 要看清楼主写的哦，后缀表达式跟前缀的不一样（次项元素 op 栈顶元素）

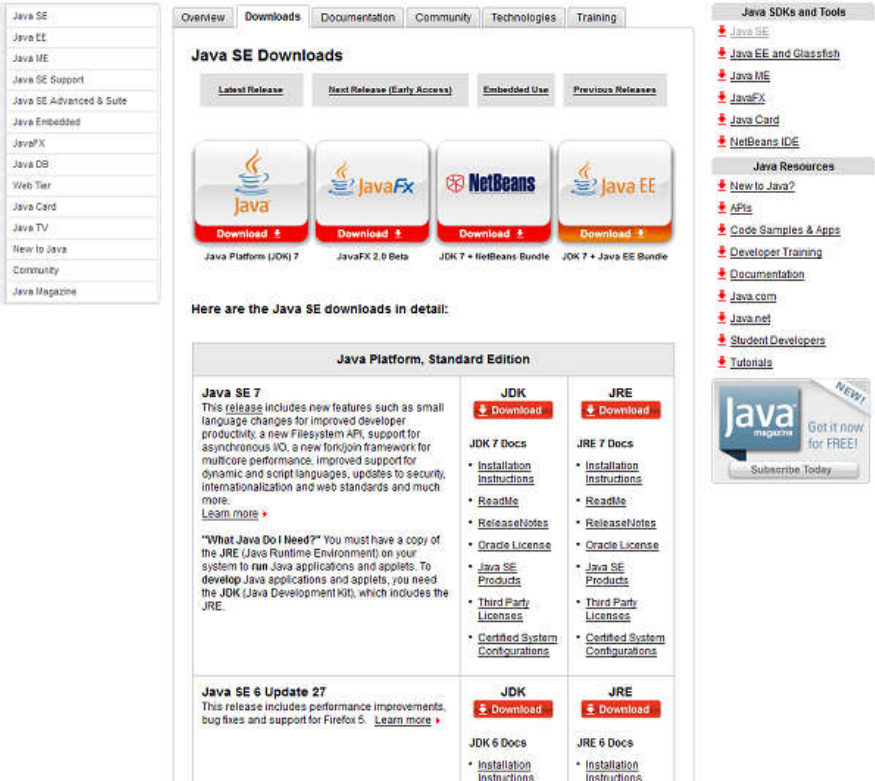
前缀、中缀、后缀表达式
GongchuangSu: @caichongjun: 要看清楼主写的哦，后缀表达式跟前缀的不一样（次项元素 op 栈顶元素）

前缀、中缀、后缀表达式
GongchuangSu: 总结的太好了，感谢感谢！

前缀、中缀、后缀表达式
caichongjun: 谢谢楼主。我还有个疑问，希望你能百忙中抽空解答一下：后缀表达式那块。计算出7*5=35，将35入栈：...

1. 下载JDK/JRE:

首先，访问Oracle公司的Java SE的下载主页 (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)，选择一个版本（目前最新版为Java SE 7），如下图：



此页面包含多个版本的JDK、JRE、帮助文档、源代码等下载内容的链接。如果不是Java程序的开发者，仅仅想在自己的系统中运行Java程序，那么只需要一个JRE就可以了；如果想使用Java开发自己的应用程序，则需要下载JDK，其中已包含JRE，因此下载了JDK后无需再单独下载JRE。

这里以下载Java SE 7的JDK为例，点击相应的Download按钮，转到下载页面：

Java SE Development Kit 7 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java™ platform.

Java SE Development Kit 7

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

☐ Accept License Agreement ☒ Decline License Agreement

Product / File Description	File Size	Download
Linux x86 - RPM Installer	77.28 MB	jdk-7-linux-i586.rpm
Linux x86 - Compressed Binary	92.17 MB	jdk-7-linux-i586.tar.gz
Linux x64 - RPM Installer	77.91 MB	jdk-7-linux-x64.rpm
Linux x64 - Compressed Binary	90.57 MB	jdk-7-linux-x64.tar.gz
Solaris x86 - Compressed Packages	154.74 MB	jdk-7-solaris-i586.tar.Z
Solaris x86 - Compressed Binary	94.75 MB	jdk-7-solaris-i586.tar.gz
Solaris SPARC - Compressed Packages	157.81 MB	jdk-7-solaris-sparc.tar.Z
Solaris SPARC - Compressed Binary	99.48 MB	jdk-7-solaris-sparcv9.tar.gz
Solaris SPARC 64-bit - Compressed Packages	16.28 MB	jdk-7-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit - Compressed Binary	12.38 MB	jdk-7-solaris-sparcv9.tar.gz
Solaris x64 - Compressed Packages	14.66 MB	jdk-7-solaris-x64.tar.Z
Solaris x64 - Compressed Binary	9.39 MB	jdk-7-solaris-x64.tar.gz
Windows x86	79.48 MB	jdk-7-windows-i586.exe
Windows x64	80.25 MB	jdk-7-windows-x64.exe

在此页面中，包含了对应各种操作系统的JDK下载链接，选择自己系统对应的JDK，将其下载到本地硬盘上。注意，在下载之前需要先阅读“Oracle Binary Code License Agreement for Java SE”，必须接受其中的条款才能下载JDK（选中“Accept License Agreement”）。

2. 安装JDK/JRE:

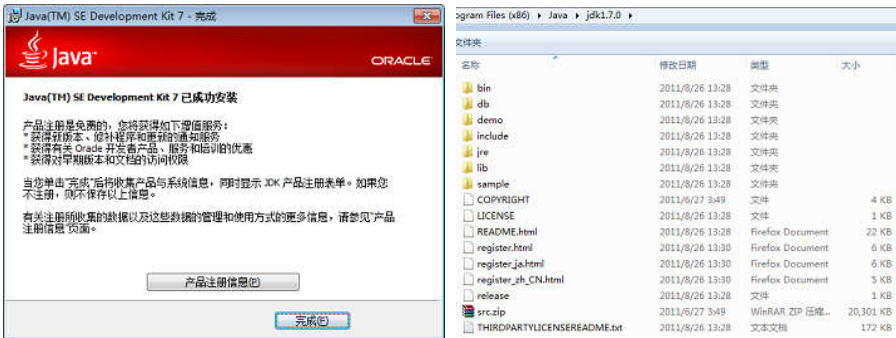
无论是在Windows还是在Linux下安装JDK都很简单，与安装其他程序没什么不同。（因为我没有其他操作系统的环境，也没用过其他系统，因此不清楚在其他操作系统下的安装方法，但想来应该也不是难事——至少不会比安装其他程序难）。

在Windows中，双击刚才下载的“jdk-7-windows-i586.exe”文件，就会打开安装界面。点击“下一步”按钮，可以在此选择需要安装的组件和安装目录，窗口右侧是对所选组件的说明，包括组件功能和所需的磁盘空间；可以点击“更改”按钮来改变安装目录。点击“下一步”即开始正式安装。安装完毕后，将会显示安装已完成的信息，点击“完成”按钮即可完成安装。

来到安装文件夹下，即可以看到已安装的JDK的目录结构。（注意其中包含名为“jre”的文件夹，这就是前面说的JDK包含JRE的原因所在）

整个安装过程如下面几幅图所示：





注意：操作系统分为32位操作系统和64位操作系统，对应地，JDK也分为32位版和64位版（名称中带有“i586”或“x86”的为32位版，带有“x64”则表示该JDK为64位版）。64位版JDK只能安装在64位操作系统上，32位版JDK则既可以安装在32位操作系统上，也可以安装在64位操作系统上。原因是64位的操作系统能够兼容32位的应用程序。换句话说，即使CPU是64位的，但如果安装的操作系统是32位的，那么也无法安装64位版的JDK。

在Linux中安装JDK与安装其他程序相同。下载时可以选择.rpm或.tar.gz格式的安装文件，这里以后者为例进行说明。

首先解压缩下载的文件，输入命令“tar -xvf jdk-7-linux-i586.tar.gz -C /usr”，将文件解压到/usr目录下，这样就完成了安装（如图）：

```
qiaomk@qiaomk-desktop:~/Downloads$ ls
jdk-7-linux-i586.tar.gz
qiaomk@qiaomk-desktop:~/Downloads$ tar -xvf jdk-7-linux-i586.tar.gz -C /usr
qiaomk@qiaomk-desktop:~/Downloads$ cd /usr
qiaomk@qiaomk-desktop:/usr$ ls
bin  games  include  jdk1.7.0  lib  local  sbin  share  src
qiaomk@qiaomk-desktop:/usr$ cd jdk1.7.0/
qiaomk@qiaomk-desktop:/usr/jdk1.7.0$ ls
bin      demo      lib      README.html  src.zip
COPYRIGHT  include  LICENSE  release      THIRDPARTYLICENSEREADME.txt
db       jre       man      sample
qiaomk@qiaomk-desktop:/usr/jdk1.7.0$
```

3. 设置环境变量：

环境变量是指在操作系统中用来指定操作系统运行环境的一些参数，比如临时文件夹位置和系统文件夹位置等。环境变量相当于给系统或应用程序设置的一些参数。

编译或运行Java程序时，都是基于命令行的，因此在此之前必须设置一些环境变量的值。有些Java IDE（集成开发环境）内置了JDK，因此使用这些IDE时可以不指定环境变量。还有些程序需要个性化的环境变量（如Apache Tomcat需要JAVA_HOME环境变量）。

与JDK或JRE的使用有关的是PATH、CLASSPATH等几个环境变量。这里先解释一下这些变量的含义：

PATH变量用来告诉操作系统到哪里去查找一个命令。如果清空PATH变量的值，在Windows中运行一个外部命令时，将提示未知命令错误（当然，在Linux中也是一样）：

```
C:\Users\mkqiao>set path=
C:\Users\mkqiao>java
'java' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

注意：在Windows中，如“dir”、“cd”等命令是内部命令，类似于DOS中的常驻命令。这些命令在命令行窗口启动时会自动加载到内存中，不需要到磁盘上去寻找对应的可执行文件，因此即使清空了PATH变量的值也不会影响这些命令的使用。然而，像“java”这样的外部命令，在执行时必须先由操作系统到指定的目录找到对应的可执行程序，然后才能加载并运行。到哪里去寻找这些程序就是依靠PATH变量来指定的。

Linux也是类似，甚至可以说在Linux中，PATH环境变量更为重要，因为Linux的很多基本命令都属于外部命令，如“ls”、“mkdir”等。当将PATH变量清空后，这些命令都将无法使用（当然，还是有一些内部命令我们仍然可以使用）。

CLASSPATH是编译或运行Java程序时用来告诉Java编译器或虚拟机到哪里查找Java类文件的，后面会对其做详细介绍。

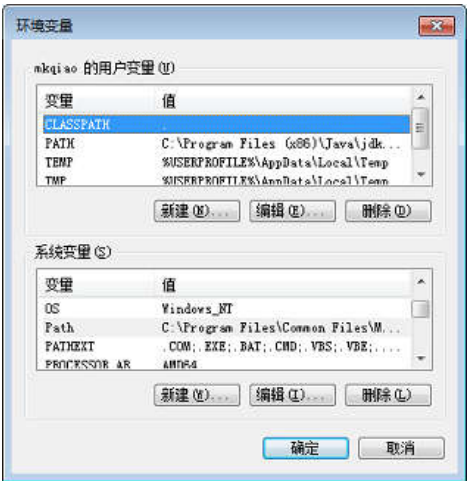
在Windows XP或之前的版本中，依次点击“右键我的电脑”->“属性”->“高级”->“环境变量”；在Windows Vista和Windows 7中则依次点击“右键我的电脑”->“属性”->“高级系统设置”->“高级”->“环境变量”，打开环境变量设置窗口：



新建一个用户变量，名称为PATH，值为“C:\Program Files (x86)\Java\jdk1.7.0\bin”（还记得前面JDK安装到哪个目录吗？），点击“确定”按钮。然后用同样的方法新建一个CLASSPATH变量，暂时将值设置为“.”（英文句号）。为什么说CLASSPATH的值是暂时的，后面会解释。



设置完成后，环境变量设置窗口如下图所示。点击确定按钮，环境变量设置完成。



注意：在Windows中，环境变量分为“用户变量”和“系统变量”，它们的区别是，“用户变量”只对当前的用户起作用，而“系统变量”则对系统中的所有用户起作用。如果希望在多个用户之间共享环境变量的设置，可以将这些环境变量设置为系统变量，否则，应该使用用户变量，避免影响其他用户。在Linux中也有类似的概念，接下来会讲到。

在Linux中，可以通过编辑“~/.bashrc”文件来修改环境变量。在最后添加下面几行脚本，然后保存并退出：

```
JAVA_HOME=/usr/jdk1.7.0
JAVA_BIN=/usr/jdk1.7.0/bin
PATH=$PATH:$JAVA_HOME/bin
CLASSPATH=.
```

```
export JAVA_HOME JAVA_BIN PATH CLASSPATH
```

```
qiaomk@qiaomk-desktop:~$ tail .bashrc
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
  . /etc/bash_completion
fi
JAVA_HOME=/usr/jdk1.7.0
JAVA_BIN=/usr/jdk1.7.0/bin
PATH=$PATH:$JAVA_HOME/bin
CLASSPATH=.
export JAVA_HOME JAVA_BIN PATH CLASSPATH
```

注意：Linux中，每个用户的home目录下都有.bashrc文件，这个文件用来保存用户的个性化设置，如命令别名、路径等，当然也可以用来定义环境变量。此文件是与每个用户相关的，一个用户的设置不会影响到其他用户，在这里设置环境变量相当于前面讲的Windows的用户环境变量。Linux中全局设置通常保存在“/etc/profile”文件中。

另外，Linux中PATH和CLASSPATH的分割符都是“:”（冒号），而Windows中是“;”（分号）。

当环境变量设置完成后，在Windows中打开新的命令行窗口，在Linux中使用“source ~/.bashrc”命令重新加载.bashrc文件，即可使新的环境变量生效。输入“java -version”命令，应该会打印出类似下面两幅图所示的内容：

Windows命令行的输出：

```
C:\Users\nkqiao>java -version
java version "1.7.0"
Java(TM) SE Runtime Environment (build 1.7.0-b147)
Java HotSpot(TM) Client VM (build 21.0-b17, mixed mode, sharing)
```

Linux的输出：

```
qiaomk@qiaomk-desktop:~$ java -version
java version "1.7.0"
Java(TM) SE Runtime Environment (build 1.7.0-b147)
Java HotSpot(TM) Client VM (build 21.0-b17, mixed mode)
```

对以上步骤补充说明几点：

1. 可以在Windows命令行或Linux Shell中使用命令设置环境变量。例如，在Windows中可以使用“set var_name=some value”，在Linux中使用“var_name=some value”，这种方式与上面介绍的方式的区别在于：这种方式的设置是临时性的，当重新启动一个新的命令行窗口（Windows）或重新登录（Linux）后，这些临时变量就会丢失。
2. JDK版本混乱：有时候，使用“java -version”命令可以打印出JDK的版本信息，但却与我们刚刚安装的JDK版本不一致。比如我们明明安装的是JDK 7，但却打印出JDK 6的版本信息，如下图所示：

```
C:\Users\nkqiao>java -version
java version "1.6.0_25"
Java(TM) SE Runtime Environment (build 1.6.0_25-b06)
Java HotSpot(TM) Client VM (build 20.0-b11, mixed mode, sharing)
```

检查PATH变量，发现其中有一个路径为“C:\Program Files (x86)\Java\jdk1.6.0_25\bin”，原来我的系统中安装了两个版本的JDK，JDK 6和JDK 7。由于此JDK 6在系统环境变量PATH中，而Windows查找命令对应的程序时，首先查找的是系统变量，当找到了一个可用的java程序时，Windows将运行这个程序，而不再进一步查找。也就是说，系统PATH环境变量屏蔽了用户PATH环境变量。

不光如此，靠近PATH变量前部的路径中的程序将屏蔽其之后的路径中的同名程序。如同样是在系统PATH变量中，“C:\Program Files (x86)\Java\jdk1.6.0_25\bin;C:\Program Files (x86)\Java\jdk1.7.0\bin”，那么JDK 6仍然将屏蔽JDK 7，如果将它们的顺序交换：“C:\Program Files (x86)\Java\jdk1.7.0\bin;C:\Program Files (x86)\Java\jdk1.6.0_25\bin”，结果则相反。

不只是用户安装了多个版本的JDK时可能导致JDK版本的混乱，而且很多软件产品自身会包含JDK，即使用户只安装了一个JDK，但仍有可能与这些软件中的JDK互相屏蔽（如果这些软件同时也设置了环境变量的话）。例如Oracle数据库、MyEclipse等都包含自己的JDK。

在Windows下我还遇到过一个问题，那就是居然在Windows\system32目录下发现了java.exe、javaw.exe、javaws.exe三个文件，因为系统PATH变量中此目录处于较靠前的位置，因此很容易将用户自己安装的JDK屏蔽掉。

有三种方法来解决这个问题：

第一种方法是使用绝对路径，例如我们运行命令时使用"C:\Program Files (x86)\Java\jdk1.7.0\bin\java.exe"（当然，.exe可以省略）而不是"java"。使用绝对路径时，操作系统会直接根据路径定位到命令所在的目录，不再通过PATH变量来查找。这种方法的优点是绝对不会导致命令的覆盖，但缺点也很明显：必须输入完整的路径来运行命令（通常也很长）。

需要注意的是，当绝对路径中存在空格时（如上面的例子那样），需要将命令用英文双引号引起来。在设置PATH变量时不需要这样做，操作系统会自动完成这件事。

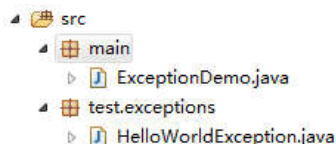
第二种方法是将自己安装的JDK路径设置到系统PATH变量的开头，这样，操作系统查找命令时就会最先查找我们设置的路径。但这种方法的缺点就是可能会影响其他用户（设置在了系统PATH变量中），并且可能会影响其他程序（其他的JDK被我们的屏蔽了）。

第三种方法就是设置一个新的环境变量，例如"JAVA_HOME"，将其值设置为我们安装的JDK的路径，如"C:\Program Files (x86)\Java\jdk1.7.0"，我们运行时，只需输入"%JAVA_HOME%\bin\java"即可（注意当路径中含有空格时要用双引号将命令引起来）。Apache Tomcat就使用这种方法。

4. 编译并运行例子程序：

经过了以上的步骤，JDK的环境就搭建好了，此时，可能需要再编译并运行一个Java例子程序来对刚搭建的环境做最终的检验。在这一节中，也会顺便讲到如何编译和运行一个Java程序，以及CLASSPATH的作用。更详细的，可以参考另一篇文章《JDK下提供的工具详解》。

此程序包含两个.java文件：ExceptionDemo.java和HelloWorldException.java，前者属于main包，而后者位于exceptions包（虽然它位于test\exceptions文件夹，这样安排的目的是为了更好地描述CLASSPATH的作用）：



下面是它们的源代码：

(1) ExceptionDemo.java:

```
[java]
01. package main;
02. import exceptions.HelloWorldException;
03. public class ExceptionDemo {
04.     /**
05.      *
06.      * @param args
07.      * @throws HelloWorldException
08.      */
09.     public static void main(String[] args)
10.         throws HelloWorldException {
11.         throw new HelloWorldException();
12.     }
13. }
```

(2) HelloWorldException.java:

```
[java]
```

```
01. package exceptions;
02. public class HelloWorldException extends Exception {
03.     private static final long serialVersionUID = 8679349130620681877L;
04.     public HelloWorldException() {
05.         super("Hello World!");
06.     }
07. }
```

此程序仍然是一个经典的HelloWorld程序（虽然这次它是以很不友好的方式向世界问好——通过抛出异常）。

要编译这个程序，首先尝试第一种方法（下面的操作是在Windows命令行下进行的，Linux与此类似）：进入src文件夹，输入“javac main\ExceptionDemo.java”，但编译报错：

```
D:\workspaces\workspace_v1.1\my-test\src>javac main\ExceptionDemo.java
main\ExceptionDemo.java:3: 错误: 程序包exceptions不存在
import exceptions.HelloWorldException;
      ^
main\ExceptionDemo.java:12: 错误: 找不到符号
        throws HelloWorldException {
              ^
  符号:   类 HelloWorldException
  位置:   类 ExceptionDemo
main\ExceptionDemo.java:13: 错误: 找不到符号
        throw new HelloWorldException();
              ^
  符号:   类 HelloWorldException
  位置:   类 ExceptionDemo
3 个错误
```

为什么会提示找不到HelloWorldException呢？那是因为该Java文件位于“test\exceptions”目录下，但它的包名却是“exceptions”，从当前的src目录，javac无法找到exceptions目录，因为“src\exceptions”目录是不存在的。

接下来，我们尝试第二种方法：由src目录进入test目录，运行“javac ../main\Exceptiondemo.java”：

```
D:\workspaces\workspace_v1.1\my-test\src>cd test
D:\workspaces\workspace_v1.1\my-test\src\test>javac ../main\ExceptionDemo.java
D:\workspaces\workspace_v1.1\my-test\src\test>dir ../main
驱动器 D 中的卷没有标签。
卷的序列号是 7234-9B9F

D:\workspaces\workspace_v1.1\my-test\src\main 的目录
2011/08/26  18:35    <DIR>          .
2011/08/26  18:35    <DIR>          ..
2011/08/26  18:49                347 ExceptionDemo.class
2011/08/26  18:33                274 ExceptionDemo.java
                2 个文件             621 字节
                2 个目录      218,820,005,888 可用字节
```

编译居然通过了！可以看到没有报错，并且main目录下生成了ExceptionDemo.class文件（Java字节码文件），说明编译确实成功了。但是为什么？我们使用了“../main\Exceptiondemo.java”，这明显不是ExceptionDemo的包路径，为什么编译器却不报错呢？另外我们还注意到，我错误地将“ExceptionDemo.java”写成了“Exceptiondemo.java”，即将字母“D”的大小写弄错了，编译器同样没有报错。

原来，javac只是将“../main\Exceptiondemo.java”当做普通路径来寻找Java源程序文件，找到后即开始编译此文件，而当其在编译过程中发现程序还引用了其他类时（如ExceptionDemo.java中引用了HelloWorldException类），就会暂停对当前文件的编译，开始寻找这个引用的类文件，如果未找到，那么将会报告错误，编译失败。前一种方法就是因为没有找到HelloWorldException类而出错的。

那么javac程序是如何查找程序引用的其他类的呢？答案是按照CLASSPATH指定的路径加上程序所引用类的包名来查找的。CLASSPATH默认为“.”，即当前路径（我们之前也设置了CLASSPATH的值为“.”，但即使不设置，javac也会默认以当前路径为起点来查找所引用的类文件）。

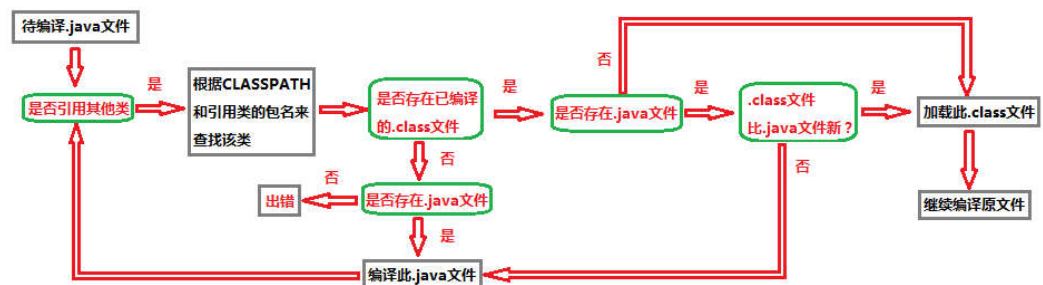
因此在这里javac会检查“src\test\exceptions\”中是否有HelloWorldException.class文件，如果有，则继续检查其中是否有

HelloWorldException.java文件，如果两者都存在，则检查HelloWorldException.class是否比HelloWorldException.java更新，如果答案是肯定的，则加载HelloWorldException.class并继续编译ExceptionDemo.java。而如果比较结果是HelloWorldException.java更新，或者不存在HelloWorldException.class，则说明需要重新编译HelloWorldException.java文件。如果只有.class文件而不存在.java文件，则加载之并继续编译ExceptionDemo.java。

如果没有找到目标文件（`HelloWorldException.class`或`HelloWorldException.java`），那么`javac`将报告错误（如之前那样）。

也就是说，编译是递归进行的：当程序中引用了其他类时，**javac**会判断是否需要编译这些类，如果需要，则**javac**会首先编译它们，如果这些类再次用到了其他的类，**javac**将再次重复此过程，直到完成全部编译。只要在此过程中有任何类没有找到，或者在其中发现了任何错误，那么**javac**将报告错误并中止编译（**javac**可能在中止之前尽可能多地编译，以尽量多地向用户报告程序中的错误）。

可以用下面的图来形象地展示这一过程:



至于我们将大小写弄错了但javac却没有报错的原因，其实前面的说明已经隐含了解释：是因为javac只是将命令中的.java文件当做普通文件，又由于Windows是不区分大小写的，因此不会报错。如果换成Linux系统，将会提示文件无法找到的错误。

默认的CLASSPATH是当前目录(“.”)，我们也可以设置为需要的路径，让java[®]c据此查找类文件（这就是前面所说的为什么只是暂时将CLASSPATH设置为“.”的原因）。在这个例子中，我们设置CLASSPATH为“.;D:\workspaces\workspace_v1.1\my-test\src\test”，注意Linux中分隔符为“:”（冒号）。然后在src目录下就可以使用命令“javac main\ExceptionDemo.java”进行编译：

```
D:\workspaces\workspace_v1.1\my-test\src\test>set classpath=.

D:\workspaces\workspace_v1.1\my-test\src\test>set classpath=%classpath%;D:\workspaces\workspace_v1.1\my-test\src\test

D:\workspaces\workspace_v1.1\my-test\src\test>echo %classpath%
.;D:\workspaces\workspace_v1.1\my-test\src\test

D:\workspaces\workspace_v1.1\my-test\src\test>cd ..

D:\workspaces\workspace_v1.1\my-test\src>javac main\ExceptionDemo.java

D:\workspaces\workspace_v1.1\my-test\src>
```

实际上，此时在任何目录都可以对ExceptionDemo.java进行编译，只是文件的路径要适当更改。例如我们在D盘根目录输入以下命令编译：

```
D:\workspaces\workspace_v1.1\my-test\src>cd \
D:\>javac workspaces\workspace_v1.1\my-test\src\main\ExceptionDemo.java
D:\>
```

这是因为设置了CLASSPATH后，javac总能找到HelloWorldException类。

有时候必须使用CLASSPATH：当涉及到的类很多时，而这些类并不在同一个目录下，此时我们只能使用CLASSPATH来指定这些类的路径——我们无法同时处于多个类的“当前目录”下。

另外一个需要注意的问题是，JDK包含的Java基础类（例如java.lang包中的类）并不需要指定CLASSPATH——Java知道如何

找到它们。

编译完成后，运行我们的例子，例子将抛出一个异常，并向世界问好：

```
D:\>cd workspaces\workspace_v1.1\my-test\src

D:\workspaces\workspace_v1.1\my-test\src>java main.ExceptionDemo
Exception in thread "main" exceptions.HelloWorldException: Hello World!
    at main.ExceptionDemo.main(ExceptionDemo.java:13)

D:\workspaces\workspace_v1.1\my-test\src>_
```

注意，必须输入完整的包名和类名（不需要.class后缀），且大小写不能弄错（因为Java是区分大小写的）。完整的包名+类名在Java中称为类的完全限定名。

至此为止，我们成功地搭建起了Java开发和运行环境。

顶

3

踩

0

上一篇

要爱惜自己的身体

下一篇

前缀、中缀、后缀表达式

相关文章推荐

- 简单java web应用程序搭建与部署
- 在服务器上部署javaweb的总结
- Java开发环境的搭建以及使用eclipse从头一步步创..
- Java我的高效编程之环境搭建
- java开发环境搭建

- Java环境搭建，以win10为例
- java环境搭建系列：JDK从下载安装到简单使用
- 【java项目实战】一步步教你使用MyEclipse搭建ja..
- Java环境搭建一JDK安装下载配置
- Java开发环境搭建

猜你在找

- 【直播】机器学习&数据挖掘7周实训--韦玮

【直播】3小时掌握Docker最佳实战--徐西宁

【直播】计算机视觉原理及实战--屈教授

【直播】机器学习之矩阵--黄博士

【直播】机器学习之凸优化--马博士
- 【套餐】系统集成项目管理工程师顺利通关--徐朋

【套餐】机器学习系列套餐（算法+实战）--唐宇迪

【套餐】微信订阅号+服务号Java版 v2.0--翟东平

【套餐】微信订阅号+服务号Java版 v2.0--翟东平


【套餐】Javascript 设计模式实战--曾亮

查看评论

7楼 淘到星星的小笛子 2017-02-01 16:51发表

 楼主写的好详细啊，可以出教程了！感谢楼主分享！

6楼 泉声 2015-08-08 08:47发表

 查找引用类的时候 应该是先在当前目录寻找 找不到再到CLASSPATH找 再找不到再报错 是吗？

5楼 SYSGIS 2015-06-29 15:05发表

 多谢

4楼 [mengxingxia](#) 2015-05-09 10:22发表



Mark

3楼 [satan_dongdong](#) 2014-11-18 21:10发表



留名备用

2楼 [w上善治水](#) 2013-11-20 10:36发表



感谢楼主，因为这个CLASSPATH困惑了好久。

1楼 [hikefreeman](#) 2013-06-16 13:23发表




写的真详细。。谢谢

发表评论

用户名: qq_36596145

评论内容:



提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#) [杂志客服](#) [微博客服](#) webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved 