Java 的 Class Path 和 Package

火龙果软件 发布于 2013-10-12

前言：

由于这两个问题新手问得较多， 且回答比较零散， 很难统一整理， 所以就直接写了一篇， 还请大家见谅.

正文：

一、 类路径 (class path)

当你满怀着希望安装好了 java， 然后兴冲冲地写了个 hello world，然后编译，运行，就等着那两个美好的单词出现在眼前， 可是不幸的是， 只看到了 Can't find class HelloWorld 或者 Exception in thread "main" java.lang.NoSuchMethodError ： maain.为什么呢？ 编译好的 class 明明在呀.

我们一起来看一看 java 程序的运行过程. 我们已经知道 java 是通过 java 虚拟机来解释运行的， 也就是通过 java 命令， javac 编译生成的 .class 文件就是虚拟机要执行的代码， 称之为字节码(bytecode)， 虚拟机通过 classloader 来装载这些字节码， 也就是通常意义上的类. 这里就有一个问题， classloader 从哪里知道 java 本身的类库及用户自己的类在什么地方呢？ 或者有着缺省值(当前路径).或者要有一个用户指定的变量来表明， 这个变量就是类路径(classpath)， 或者在运行的时候传参数给虚拟机. 这也就是指明 classpath 的三个方法. 编译的过程和运行的过程大同小异， 只是一个是找出来编译， 另一个是找出来装载. 实际上 java 虚拟机是由 java luncher 初始化的， 也就是 java (或 java.exe)这个程序来做的. 虚拟机按以下顺序搜索并装载所有需要的类：

1， 引导类： 组成 java 平台的类， 包含 rt.jar 和 i18n.jar 中的类.

2， 扩展类： 使用 java 扩展机制的类， 都是位于扩展目录($JAVA_HOME/jre/lib/ext)中的 .jar 档案包.

3， 用户类： 开发者定义的类或者没有使用 java 扩展机制的第三方产品. 你必须在命令行中使用 -classpath 选项或者使用 CLASSPATH 环境变量来确定这些类的位置. 我们在上面所说的用户自己的类就是特指这些类.这样， 一般来说， 用户只需指定用户类的位置， 引导类和扩展类是"自动"寻找的.

那么到底该怎么做呢？ 用户类路径就是一些包含类文件的目录， .jar， .zip 文件的列表， 至于类具体怎么找， 因为牵扯到 package 的问题， 下面将会说到， 暂时可认为只要包含了这个类就算找到了这个类. 根据平台的不同分隔符略有不同， 类 unix 的系统基本上都是":"， windows 多是 ";". 其可能的来源是：

1.″.″， 即当前目录， 这个是缺省值.

2.CLASSPATH 环境变量， 一旦设置， 将缺省值覆盖.

3.命令行参数 -cp 或者 -classpath， 一旦指定， 将上两者覆盖.

4.由 -jar 参数指定的 .jar 档案包， 就把所有其他的值覆盖， 所有的类都来自这个指定的档案包中. 由于生成可执行的 .jar 文件， 还需要其他一些知识， 比如 package， 还有特定的配置文件， 本文的最后会提到. 可先看看 jdk 自带的一些例子.

我们举个 HelloWorld 的例子来说明. 先做以下假设：

1.当前目录是 /HelloWorld (或 c：\HelloWorld， 以后都使用前一个)

2.jdk 版本为 1.2.2 (linux 下的)

3.PATH 环境变量设置正确. (这样可以在任何目录下都可以使用工具)

4.文件是 HelloWorld.java， 内容是：

```
public class HelloWorld

{

public static void main(String[] args)

{

          System.out.println("Hello World!\n");

          System.exit(0);

}

}
```

首先这个文件一定要写对, 如果对 c 熟悉的话, 很有可能写成这样:

```
public static void main(int argc, String[] argv)

{

....
```

```
}
```

这样是不对的，不信可以试一试. 由于手头没有 java 的规范，所以作如下猜想：java 的 application 程序，必须以 public static void main(String[])开始，其他不一样的都不行.

到现在为止，我们设置方面只设置了 PATH.

1，当前路径就是指你的 .class 文件在当前目录下，

```
[HelloWorld]$ javac HelloWorld.java  //这一步不会有多大问题，

[HelloWorld]$ java HelloWorld        // 这一步可能就会有问题.
```

如果出了象开头那样的问题，首先确定不是由于敲错命令而出错. 如果没有敲错命令,那么接着做：

```
[HelloWorld]$ echo $CLASSPATH 或者 c:\HelloWorld>echo %CLASSPATH%
```

看看 CLASSPATH 环境变量是否设置了，如果设置了，那么用以下命令：

```
[HelloWorld]$ CLASSPATH= 或者 c:\HelloWorld> set CLASSPATH=
```

来使它为空，然后重新运行. 这次用户类路径缺省的是 ″.″，所以应该不会有相同的问题了. 还有一个方法就是把 ″.″ 加入到 CLASSPATH 中.

```
[/]$ CLASSPATH=$CLASSPATH:. 或者 c:\HelloWorld> set
CLASSPATH=%CLASSPATH%;.
```

同样也可以成功. Good Luck.

2，当你的程序需要第三方的类库支持，而且比较常用，就可以采用此种方法.比如常用的数据库驱动程序，写 servlet 需要的 servlet 包等等. 设置方法就是在环境变量中加入 CLASSPATH. 然后就可以直接编译运行了. 还是以 HelloWorld 为例，比如你想在根目录中运行它，那么你直接在根目录下执行

```
$ java HelloWorld 或者 c:\>java HelloWorld
```

这样肯定会出错，如果你的 CLASSPATH 没有改动的话. 我想大家应该知道为什么错了吧，那么怎么改呢？前面说过，用户类路径就是一些包含你所需要的类的目录，.jar 档案包，.zip 包. 现在没有生成包，所以只好把 HelloWorld.class 所在的目录加到 CLASSPAT 了，根据前面的做法，再运行一次，看看，呵呵，成功了，换个路径，又成功了!! 不仅仅可以直接运行其中的类，当你要 import 其中的某些类时，同样处理. 不知道你想到没有，随着你的系统的不断的

扩充, (当然了, 都是一些需要 java 的东西).如果都加到这个环境变量里, 那这个变量会越来越臃肿, 虽然环境变量空间可以开很大, 总觉得有些不舒服. 看看下面一个方法.

3, 在命令行参数中指明 classpath. 还是和上面相同的目标, 在任何目录下执行 HelloWorld, 用这个方法怎么实现呢?

```
[/]$ java -cp /HelloWorld HelloWorld 或者 c:\>java -cp c:\HelloWorld HelloWorld
```

就可以了. 这是这种方法的最简单的应用了. 当你使用了另外的包的时候, 还可以采用用这种方法. 例如:

```
$ javac -classpath aPath/aPackage.jar:. myJava.java

$ java  -cp aPath/aPackage.jar:. myJava

          或者

c:\> javac -classpath aPath\aPackage.jar;. myJava.java

c:\> java  -cp aPath\aPackage.jar;. myJava
```

这种方法也有一个不方便的的地方就是当第三方包所在的路径较长或者需要两个以上包的时候, 每次编译运行都要写很长, 非常不方便, 这时候可以写脚本来解决. 比如一个例子:

```
compile   (文件, 权限改为可执行, 当前目录)

$ cat compile

---------------------------

#!/bin/bash

          javac -classpath
aPath\aPackage.jar:anotherPath\anotherPackage.jar:. m

yJavva.java

---------------------------

run (文件, 权限改为可执行, 当前目录)

$cat run
```

```
---------------------------

#!/bin/bash

java -cp aPath\aPackage.jar:anotherPath\anotherPackage.jar:. myJava

---------------------------

          或者:

compile.bat

c:\HelloWorld> type compile.bat

-------------------------

          javac -classpath
aPath\aPackage.jar:anotherPath\anotherPackage.jar:. m

yJavva.java

-------------------------

run.bat

c:\HelloWorld> type run.bat

-----------------------

java -cp aPath\aPackage.jar:anotherPath\anotherPackage.jar:. myJava

-----------------------
```

就可以了. 试试看.

前面提到了扩展类, 扩展类是什么呢? java 的扩展类就是应用程序开发者用来扩展核心平台功能的 java 类的包(或者是 native code). 虚拟机能像使用系统类一样使用这些扩展类. 有人建议可以把包放入扩展目录里, 这样, CLASSPATH 也不用设了,也不用指定了, 岂不是很方便? 确实可以正确运行, 但是个人认为这样不好, 不能什么东西都往里搁, 一些标准的扩展包可以, 比如, JavaServlet, Java3D 等等. 可以提个建议, 加一个环境变量, 比如叫 JARPATH, 指定一个目录, 专门存放用户的 jar zip 等包, 这个要等 SUN 公司来做了. windows98 下, 我原来安装的时候, 一直装不上, 总是死机, 好不容易装上了, 缺省的是不能运行正确的, 然后把 tool.jar 放入 CLASSPATH 后工作正常. 现在作测试,去掉仍然是正确的. 经过多次测试, 发现

如果原来曾装过 jdk 的都很好，没有装过的，装的时候会死机，多装几次就可以了. 如果你发现正确安装后，不能正常工作，就把 tools.jar 加入 CLASSPATH，试一下.

二、 包 (package)

Java 中的 "包" 是一个比较重要的概念，package 是这样定义的:Definition: A package is a collection of related classes and interfaces that provides access protection and namespace management. 也就是: 一个包就是一些提供访问保护和命名空间管理的相关类与接口的集合. 使用包的目的就是使类容易查找使用，防止命名冲突，以及控制访问. 这里我们不讨论关于包的过多的东西，只讨论和编译，运行，类路径相关的东西.至于包的其他内容，请自己查阅相关文档. 简单一点来说，包就是一个目录，下面的子包就是子目录，这个包里的类就是这个目录下的文件. 我们用一个例子来说明.

首先建目录结构如下: PackageTest/source/，以后根目录指的是 PackageTest 目录，我们的源程序放在 source 目录下. 源程序如下:

```
PackageTest.java

package pktest;

import pktest.subpk.*;

public class PackageTest

{

private String value;

public PackageTest(String s)

{

value = s;

}

public void printValue()

{

          System.out.println("Value of PackageTest is " + value);

}
```

```
public static void main(String[] args)

{

          PackageTest test = new PackageTest("This is a Test
Package");

          test.printValue();

          PackageSecond second = new PackageSecond("I am in
PackageTest");

          second.printValue();

          PackageSub sub = new PackageSub("I am in PackageTest");

          sub.printValue();

          System.exit(0);

}

}
```

PackageSecond.java

```
package pktest;

public class PackageSecond

{

private String value;

public PackageSecond(String s)

{

value = s;

}

public void printValue()
```

```
{
          System.out.println("Value of PackageSecond is " + value);
}
}
```

PackageSub.java

```
package pktest.subpk;
import pktest.*;
public class PackageSub
{
private String value;
public PackageSub(String s)
{
value = s;
}
public void printValue()
{
          PackageSecond second = new PackageSecond("I am in
subpackage.");
          second.printValue();
          System.out.println("Value of PackageSub is " + value);
}
}
```

```
Main.java

import pktest.*;

import pktest.subpk.*;

public class Main()

{

public static void main()

{

          PackageSecond second = new PackageSecond("I am in Main");

          second.printValue();

          PackageSub sub = new PackageSub("I am in Main");

          sub.printValue();

          System.exit(0);

}

}
```

其中,Main.java 是包之外的一个程序,用来测试包外的程序访问包内的类,PackageTest.java 属于 pktest 这个包,也是主程序.PackageSecond.java 也属于 pktest,PackageSub 属于 pktest 下的 subpk 包,也就是 pktest.subpk.详细使用情况,请参看源程序。

好了， 先把源程序都放在 source 目录下， 使 source 成为当前目录， 然后编译一下，呵呵， 出错了，Main.java：1： Package pktest not found in import.import pktest.*;这里涉及到类路径中包是怎么查找的， 前面我们做了一点假设： "只要包含了这个类就算找到了这个类"， 现在就有问题了. 其实 jdk 的 工具 javac javajavadoc 都需要查找类， 看见目录， 就认为是包的名字， 对于 import 语句来说，一个包对应一个目录. 这个例子中，import pktest.*， 我们知道类路径可以包含一个目录， 那么就以那个目录为根， 比如有个目录 /myclass， 那么就会在查找/myclass/pktest 目录及其下的类. 所有的都找遍， 如果没有就会报错. 由于现在的类路径只有当前目录， 而当前目录下没有 pktest 目录， 所以就会出错. 类路径还可以包含 .jar .zip 文件， 这些就是可以带目录的压缩包， 可以把 .jar .zip 文件看做一个虚拟的目录， 然后就和目录一样对待了.

好了， 应该知道怎么做了吧， 修改后的目录结构如下：

```
PackageTest

|

|__source    Main.java

|

|__pktest    PackageTest.java    PackageSecond.java

|

|__subpk   PackageSub.java
```

然后重新编译， 运行， 哈哈， 通过了. 我们再来运行一下 PackageTest.

```
[source]$ java pktest/PackageTest
```

怎么又出错了？

Exception in thread "main" java.lang.NoClassDefFoundError： pktest/PackageTest 是这样的， java 所要运行的是一个类的名字， 它可不管你的类在什么地方， 就象我们前面所讨论的一样来查找这个类， 所以它把 pktest/PackageTest 看成是一个类的名字了， 当然会出错了， 应该这么做，

```
[source]$ java pktest.PackageTest
```

注意 javac 不一样， 是可以指明源文件路径的， javac 只编译， 不运行， 查找类也只有在源文件中碰到 import 时才会做， 与源文件所在的包没有关系. 似乎还又些不好的地方，怎么生成的 .class 文件这么分散呀， 看着真别扭. 别急，javac 有一个 -d 命令行参数， 可以指定一个目录， 把生成的 .class 文件按照包给你好好地搁在这个目录里面.

```
[source]$ mkdir classes

[source]$ javac -d classes pktest/PackageTest.java

[source]$ javac -d classes Main.java
```

那么运行怎么运行呢?

```
[source]$ cd classes

[classes]$ java pktest.PackageTest

[classes]$ java Main
```

就可以了. 其实 jdk 的这一套工具小巧简单, 功能强大, 不会用或者用错其实不关工具的事, 关键是明白工具背后的一些原理和必要的知识. 集成环境是很好,但是它屏蔽了很多底层的知识, 不出错还好, 一旦出错, 如果没有这些必要的知识就很难办, 只好上 bbs 问, 别人只告诉了你解决的具体方法, 下一次遇到稍微变化一点的问题又不懂了. 所以不要拘泥于工具, java 的这一套工具组合起来使用, 中小型工程(五六十个类), 还是应付得下来的.

三、 jar 文件

以下把 .jar .zip 都看做是 .jar 文件.

1, 从前面我们可以看出来 jar 文件在 java 中非常重要, 极大地方便了用户的使用. 我们也可以做自己的 .jar 包.

还是使用前面那个例子, Main.java 是包之外的东西, 用了 pktest 包中的类,我们现在就是要把 pktest 做成一个 .jar 包, 很简单, 刚才我们已经把 pktest 中的 .class 都集中起来了,

```
[classes]$ jar -cvf mypackage.jar pktest
```

就会生成 mypackage.jar 文件, 测试一下, 刚才我们生成的 Main.class 就在 classes 目录下, 所以, 从前面可以知道:

```
[classes]$ java -cp mypackage.jar:.Main
```

就可以运行了.

2, 如果你看过 jdk 所带的例子, 你就会知道, .jar 还可以直接运行,

```
[/demo]$ java -jar aJar.jar
```

那好, 就那我们的试一试,

看来我们的 jar 和它的 jar 还不一样, 有什么不一样呢? 拿它一个例子出来,重新编译, 生成 .jar 文件, 比较后发现, 是 .jar 压缩包中 META-INF/MANIFEST.MF 文件不一样, 多了一行, Main-Class: xxxxx, 再看看出错信息, 原来是没有指定 Main-Class, 看看 jar 命令, 发现有一个参数 -m,-m include manifest information from specified manifest file

和出错信息有点关系，看来它要读一个配制文件. 只好照猫画虎写一个了.

```
[classes]$ cat myManifest

Manifest-Version: 1.0

Main-Class: pktest.PackageTest

Created-By: 1.2.2 (Sun Microsystems Inc.)

[classes]$ jar -cvfm mypackage.jar myManifest pktest

added manifest

adding: pktest/(in = 0) (out= 0)(stored 0%)

adding: pktest/PackageSecond.class(in = 659) (out= 395)(deflated
40%)

adding: pktest/subpk/(in = 0) (out= 0)(stored 0%)

adding: pktest/subpk/PackageSub.class(in = 744) (out= 454)(deflated
38%)

adding: pktest/PackageTest.class(in = 1041) (out= 602)(deflated 42%)

[classes]$ java -jar mypackage.jar

Value of PackageTest is This is a Test Package

Value of PackageSecond is I am in PackageTest

Value of PackageSecond is I am in subpackage.

Value of PackageSub is I am in PackageTest
```

好了，成功了，这样就做好了一个可以直接执行的 .jar 文件. 大家可以自己试一试

做一个以 Main 为主程序的可执行的 jar.
