[如何理解和使用Java package包](http://blog.csdn.net/zhenyusoso/article/details/6174834)

标签： [java](http://www.csdn.net/tag/java)[apple](http://www.csdn.net/tag/apple)[import](http://www.csdn.net/tag/import)[class](http://www.csdn.net/tag/class)[internet](http://www.csdn.net/tag/internet)[jar](http://www.csdn.net/tag/jar)

2011-02-08 20:07 42031人阅读 [评论](http://blog.csdn.net/zhenyusoso/article/details/6174834#comments)(9) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/zhenyusoso/article/details/6174834#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

读书笔记（5） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

[**Java**](http://lib.csdn.net/base/java)中的一个包就是一个类库单元，包内包含有一组类，它们在单一的名称空间之下被组织在了一起。这个名称空间就是包名。可以使用import关键字来导入一个包。例如使用import java.util.\*就可以导入名称空间java.util包里面的所有类。所谓导入这个包里面的所有类，就是在import声明这个包名以后，在接下来的程序中可以直接使用该包中的类。例如：

**[java]** [view plain](http://blog.csdn.net/zhenyusoso/article/details/6174834) [copy](http://blog.csdn.net/zhenyusoso/article/details/6174834)

1. **import** java.util.\*
3. **public** **class** SingleImport
4. {
5. **public** **static** **void** main(Strin[] args)
6. {
7. ArrayList list=nwe ArrayList();
8. }
9. }

    这里ArrayList就是java.util包中的一个类，但是由于对程序使用了import关键字加载了java.util包，所以这里并没有见到对ArrayList类的定义和声明，也没有见到该类前面有什么限定名，就可以直接使用这个类。

       我们之所以要导入包名，就是要提供一个管理名称空间的机制。我们知道，如果有两个类A类和B类都含有一个具有相同特征标记（参数列表）的方法f()，即便在同一段代码中同时使用这两个方法f()，也不会发生冲突，原因就在于有两个不同的类名罩在前面作为限定名，所以两个方法即便同名也不回发生冲突。但是如果类名称相互冲突又该怎么办呢？假设你编写了一个Apple类并安装到了一台机器上，而该机器上已经有一个其他人编写的Apple类，我们该如何解决呢？因为你如果想弄清楚一台机器上到底已经安装了那些类，并不是一件很容易的事情，所以名字之间总是有存在潜在的冲突的可能。在Java中对名称空间进行完全控制并为每个类创建唯一的标识符组合就成为了非常重要的事情。如果你要编写对于同一台机器上共存的其他Java程序友好的类库或程序的话，就需要考虑如何防止类名称之间的冲突问题。

    当编写一个Java源代码文件时，此文件通常被称为编译单元。每个编译单元都必须有一个后缀名.java，而在编译单元内有且仅有一个public类，否则编译器就不会接受。该public类的名称必须与文件的名称相同（包括大小写，但不包括后缀名.java）。如果在该编译单元之中还有额外的类的话，那么在包之外的世界是无法看见这些类的，因为它们不是public类，而且它们主要用来为主public类提供支持。

    当编译一个.java文件（即一个编译单元）时，在.java文件中的每个类都会有一个输出文件，而该输出文件的名称与.java文件中每个类的名称相同，只是多了一个后缀名.class。因此在编译少量.java文件之后，会得到大量的.class文件。每一个.java文件编译以后都会有一个public类，以及任意数量的非public类。因此每个.java文件都是一个构件，如果希望许许多多的这样的构件从属于同一个群组，就可以在每一个.java文件中使用关键字package。而这个群组就是一个类库。

    如果使用package语句，它必须是.java文件中除注释以外的第一句程序代码。如果在文件的起始处写：

    package fruit;

就表示你在声明该编译单元是名为fruit的类库的一部分，或者换句话说，你正在声明该编译单元中的public类名称是位于fruit名称的保护伞下，由fruit名称罩着。任何想要使用该public类名称的人都必须指定全名或者与fruit结合使用关键字import。

    例如，假设文件的名称是Apple.java，这就意味着在该文件中有且仅有一个public类，该类的名称必须是Apple（注意大小写）：

**[java]** [view plain](http://blog.csdn.net/zhenyusoso/article/details/6174834) [copy](http://blog.csdn.net/zhenyusoso/article/details/6174834)

1. **package** fruit;
3. **public** **class** Apple
4. {
5. //...
6. }

    上面的代码已经将Apple类包含在了fruit包中，现在如果有人想使用Apple或者是fruit中的任何其他public类，就必须使用关键字import来使fruit中的名称可用。

**[java]** [view plain](http://blog.csdn.net/zhenyusoso/article/details/6174834) [copy](http://blog.csdn.net/zhenyusoso/article/details/6174834)

1. **import** fruit.\*;
3. **public** **class** ImportApple
4. {
5. **public** **static** **void** main(String[] args)
6. {
7. Apple a=**new** Apple();
8. }
9. }

或者使用完整限定名称：

**[java]** [view plain](http://blog.csdn.net/zhenyusoso/article/details/6174834) [copy](http://blog.csdn.net/zhenyusoso/article/details/6174834)

1. **public** **class** QualifiedApple
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. fruit.Apple a=**new** fruit.Apple();
6. }
7. }

显然使用关键字import使代码更加简洁。

      作为一名程序员，我们应该牢记：package和import关键字允许做的是将单一的全局名称空间分割成各自独立封闭的名称空间，使得无论多少人使用Internet以及Java开始编写类，都不会出现与我们的类名称相冲突的问题，因为我们的类是被封闭在我们自己定义的独立的名称空间里面的，而非在公共的全局名称空间里面。

      到这里也许你会发现，其实所谓关键字package打包从未将被打包的东西包装成一个单一的文件，并且一个包可以由许多.class文件构成，这就存在将两个名称相同的类打进一个包中的可能。为了避免这种情况的发生，一种合乎逻辑的做法就是将特定的所有.class文件都置于一个目录下。也就是说利用[**操作系统**](http://lib.csdn.net/base/operatingsystem)的层次化的文件结构来解决这一问题。这是Java解决混乱问题的一种方式（这里暂且先不讨论JAR包工具）。

      将所有的文件收入一个子目录还可以解决另外两个问题：一、怎样创建独一无二的名称；二、怎样查找有可能隐藏于目录结构中某处的类。

       这些任务是通过将.class文件所在的路径位置编码称package的名称来实现的。

      按照惯例，package名称的第一部分是类的创建者的反顺序的Internet域名。为什么要用Internet域名呢？因为如果你遵照惯例，Internet域名应该是独一无二的，因此你的package名称也将是独一无二的，也就是前面提到的我们自定义的独立封闭的名称空间将是独一无二的，这样就不会出现名称冲突的问题了。当然，如果你没有自己的域名，你就得构造一组不大可能与他人重复的组合（例如你的姓名），来创立独一无二的package名称。如果你打算发布你的Java程序代码，稍微花费些代价去取得一个域名还是很有必要的。

      另外，如果你的Java程序代码只是在本地计算机上运行，你还可以把package名称分解为你机器上的一个目录。所以当Java程序运行并且需要加载.class文件的时候，它就可以根据package名称确定.class文件在目录上的所处位置。

      程序在运行的时候具体是如何确定.class文件位置的呢？

      来看看Java解释器的运行过程吧：首先，找出环境变量CLASSPATH（可以通过操作系统来设置）。CLASSPATH包含一个或多个目录，用作查找.class文件的根目录。从根目录开始，解释器获取包名称并将每个句点替换成反斜杠，以从CLASSPATH根中产生一个路径（例如，package fruit.Apple就变成为fruit/Apple或fruit/Apple或其他，这将取决于操作系统）。得到的路径会与CLASSPATH中的各个不同的根目录路径相连接以获得一个完整的目录路径，解释器就在这些目录中查找与你所需要的类名称相同的.class文件。（此外，解释器还会去查找某些涉及Java解释器所在位置的标准目录。）

      为了理解这一点，以域名Food[**.NET**](http://lib.csdn.net/base/dotnet)为例。把它的顺序倒过来，并且全部转换为小写，net.food就成了我们创建类的一个独一无二的名称空间。如果我们决定再创建一个名为fruit的类库，我们可以将该名称进一步细分，于是得到一个包名如下：

    package net.food.fruit;

    现在，这个包名称就可以用作下面Apple这个文件的名称空间保护伞了：

**[java]** [view plain](http://blog.csdn.net/zhenyusoso/article/details/6174834) [copy](http://blog.csdn.net/zhenyusoso/article/details/6174834)

1. **package** net.food.fruit;
3. **public** **class** Apple
4. {
5. **public** Apple()
6. {
7. System.out.println("net.food.fruit.Apple");
8. }
9. }

这个文件可能被置于计算机系统中的如下目录中：

    C:/DOC/JavaT/net/food/fruit

    之所以要放在这个目录下面是因为前面提到的，便于系统通过CLASSPATH环境变量来找到这个文件。沿着此路径往回看就能看到包名net.food.fruit，但是路径的前半部分怎么办呢？交给环境变量CLASSPATH吧，我们可以在计算机中将环境变量CLASSPATH设置如下：

    CHASSPATH=.;D:/JAVA/LIB;C:/DOC/JavaT

    CLASSPATH可以包含多个可供选择的查询路径。每个路径都用分号隔开，可以看到，上面这个CLASSPATH环境值的第三个路径就是我们前面文件的根目录。如前所述，Java解释器将首先找到这个根目录C:/DOC/JavaT，然后将其与包名net.food.fruit相连接，连接的时候将包名中的句点转换成斜杠，就得到完整的class文件路径C:/DOC/JavaT/net/food/fruit。

    需要补充说明的一点，这里CLASSPATH环境变量关照的是package中的class文件，如果关照的是JAR包中的class文件，则会有一点变化，即，必须在CLASSPATH环境变量路径中将JAR文件的实际名称写清楚，而不仅仅是指明JAR包所在位置目录。可以想象，因为JAR包所在目录位置上可能存在很多别的JAR包，而我们需要使用的那个class文件只会存在于其中一个JAR包里面，因此可以这样理解，这里JAR包实际上也充当了一级文件目录的角色，因此要在CLASSPATH环境变量中写清楚JAR包文件名。例如如果Apple文件存在于名为fruit.jar的JAR文件中，则CLASSPATH应写作：

    CLASSPATH=.;D:/JAVA/LIB;C:/DOC/JavaT/net/food/fruit.jar

    一旦路径得以正确建立，下面的文件就可以放于任何目录之下：

**[java]** [view plain](http://blog.csdn.net/zhenyusoso/article/details/6174834) [copy](http://blog.csdn.net/zhenyusoso/article/details/6174834)

1. **import** net.food.fruit.\*;
3. **public** **class** LibTest
4. {
5. **public** **static** **void** main(String[] args)
6. {
7. Apple a=**new** Apple();
8. }
9. }

    当编译器碰到fruit库的import语句时，就开始在CLASSPATH所指定的目录中查找，查找过程中分别将CLASSPATH中设定的各项根目录与包名转换来的子目录net/food/fruit相连接，在连接后的完整目录中查找已编译的文件（即class文件）找出名称相符者（对Apple而言就是Apple.class）。找到了这个文件即匹配到了Apple类。