Java 的 Package 和 Classpath

Package

在 Java 中，Package 是用来包含一系相关实例的集合。这些相关联的实例包括：类、接口、异常、错误以及枚举。

Package 主要有一些的几点作用：

1. Package 可以处理名字冲突，在冲突的名字前加上包的名字，通过使用名字的全限定名来访问名字的时候，可以避免名字冲突。因为在不同的包之间，具有不同的包名，所以可以通过全限定名来区分不同包中同名的名字。Package 的这种机制称为名字空间管理（Namespace Management）。

2. Package 可以实现访问控制，在 Java 中，除了常用的 `public` 和 `private` 这两个访问控制修饰符外，还包含了 `protected` 和 `default` 这两个访问控制修饰符，这两个修饰符都和 Package 相关。通过 `protected` 修饰的实体，它的访问受限于同一个包和它的子类中。如果一个实体没有包含任何的访问控制修饰符，那么默认就是 `default`，它的访问受限于同一个包中。

3. 用于发布可重用的类的集合，通常会将这些类打包成 JAR 包的形式。

Package 的名字约定

一个包的名字可以通过将互联网的域名反向后加上自己的项目名字产生。中间通过 . 进行分隔。Package 的名字采用小写的方式。（i.e. 如果一个人的域名是'abc.com'，那么他可以将自己项目的包名写成 'com.abc.project'）。
我们可能会看到在 Java 官方提供的包中，包含了 `java` 和 `javax` 前缀的包名，这两个前缀分别用于官方提供的 java 包和 java 的扩展包。

包名和目录结构的关系

Java 中的包名和文件系统的目录结构之间是有联系的。同一个包中的实体，都被存储在同一个目录下，确切的说，这些实体被存储在通过包名确定的子目录结构下。i.e.，假设存在一个包名为 `com.abc.project` 的包，那么这个包中的实体被存储在目录 `$BASE_DIR/com/abc/project` 下，其中的`$BASE_DIR` 表示了包的根目录，也可以称为 Java 中的类路径。通过上面的例子，我们可以发现，将包名中的 . 转换为 / 就是相应的子目录结构了。
上面提到的`$BASE_DIR` 可以存在于文件系统的任何位置，所以 Java 的编译器和虚拟机必须知道`$BASE_DIR` 的位置，才可以定位到需要的实体。对这个位置的查找是通过环境变量 `CLASSPATH` 来实现的。`CLASSPATH` 是一个类似于 `PATH` 的环境变量，只是 `CLASSPATH` 是用于查找 Java 的类的位置的。
在 Java 中，没有子包的概念。i.e.，两个包 `java.awt` 和 `java.awt.event`，这两个包具有相同的前缀。其中包 `java.awt` 中的实体存储在路径`$BASE_DIR\java\awt` 下，而包 `java.awt.event` 中的实体存储在路径`$BASE_DIR\java\awt\event` 下，这是两个独立的包，只是具有相同的前缀而已，所以 `java.awt.event` 不是 `java.awt` 的子包。

例子

```
package com.yyy;
```

```
public class Circle {

   private double radius;

   public Circle(double radius) {

      this.radius = radius;

   }

   public double getRadius() {

      return radius;

   }

   public void setRadius(double radius) {

      this.radius = radius;

   }

}
```

我们在./src/java/com/yyy 目录下创建了一个 com.yyy 的包，并且在包中创建了一个 Circle 类。然后我们需要将这个类的编译后的 class 文件存储在目录 classes 目录下。

```
javac -d ./classes ./src/java/com/yyy/Circle.java
```

javac 命令会编译这个 java 文件，然后将编译后的 class 文件存储在 classes/com/yyy 目录下，com/yyy 这个子目录会根据包名 com.yyy 自动生成。在这里，我们通过选项-d 来指定生成的类存放的位置的根目录。

下面，我们使用上面创建的类对象：

```
import com.yyy.Circle;

public class TestCircle {

   public static void main(String[] args) {

      Circle c = new Circle(1.23);

      System.out.println(c.getRadius());

   }

}
```

我们在目录./test 下创建类 TestCircle，但是我们不能直接编译这个类：

```
cd test
```

```
javac TestCircle.java
```

上面的命令会报错，提示找不到类 com.yyy.Circle，所以我们需要告诉编译器该类的位置。通过选项-cp(-classpath)可以指定类路径的位置（这个路径是包所在的根目录，而不是包中类的目录。通过这个根目录，java 会自动查找包中的类）。

```
javac -cp ./classes TestCircle.java
```

通过-cp 指定类路径以后，可以顺利编译通过了，但是如果我们直接执行生成的 class 文件，还是会出现问题：

```
java TestCircle
```

上面的命令会报错，提示找不到类 com.yyy.Circle，所以我们需要告诉命令哪里可以找到这个类，跟上面一样，给出这个类所在的位置，也就是类路径：

```
java -cp ./classses TestCircle
```

上面的命令，看似没什么问题了，但是运行还是会报错，但是这次提示找不到 TestCircle 类。这是因为 java 的 CLASSPATH 如果没有被显式定义指定，那么默认值是当前目录，所以第一次没有报找不到 TestCircle 的错误。但是如果我们显式更改 CLASSPATH 的值（通过上面的-cp 选项），那么就默认不会包含当前的目录，如果需要包含当前的目录，则需要显式指定。在 CLASSPATH 中可以通过:分隔不同的路径。

```
java -cp .:./classes TestCircle
```

如果 TestCircle.java 被定义在包 com.abc 中，那么如果我们在这个包的根目录下引用这个类，则需要使用这个类的全限定名：

```
//TestCircle.java 和 Circle.java 的 class 文件都被放在 classes 目录下

javac -d classes -cp ./classes src/com/abc/TestCircle.java

//我们当前所在的目录是包的根目录，所以引用 TestCircle 的时候，需要使用全限定名

java -cp ./classes com.abc.TestCircle
```

CLASSPATH

CLASSPATH 是一个环境变量，Java 虚拟机和编译器会通过这个环境变量来查找 java 类和包的位置。

Java 虚拟机查找类的方式

在讨论 Java 的 CLASSPATH 环境变量之前，先介绍下 Java 虚拟机是如何查找 Class 文件的。

java 虚拟机，也就是 `java` 命令，通过以下的顺序查找和加载 java 的 class 文件：

1. Bootstrap classes - 构成 java 平台的 class 文件，包括 rt.jar 和一些其他的重要的 jar 文件
2. Extension classes - java 的扩展机制的 class 文件，这些 class 文件以 jar 的方式组织并存在于扩展目录中
3. User classes - 开发者和第三方创建的 class 文件，这些 class 文件的位置通过 `-classpath(-cp)` 选项来指定，或者通过设置环境变量 `CLASSPATH` 来指定。

Java 虚拟机查找 Bootstrap classes 的方式

Bootstrap classes 是一些用于实现 Java 2 平台的 class 文件。Bootstrap classes 包含在 rt.jar 和一些其他的 jar 文件中，这些 jar 文件放在 `jre/lib` 目录下。这些包是通过 bootstrap class 路径指定的，这个路径值存储在 `sun.boot.class.path` 这个系统属性中，这个系统属性应该是只读的，不应该直接修改。
一般情况下，bootstrap classes 路径是不应该被重新修改的。Java 的非标准选项 `-Xbootclasspath`，允许修改这个路径值来进行自定义定制核心 class。
需要指出的是，实现 Java 2 SDK 的工具 class 文件并不和 bootstrap class 文件存放在一起。这些工具 sdk 存放在目录 `/lib/tools.jar` 下。开发工具会在启动 Java 虚拟机的时候将这个 jar 包添加到 user class 路径中。然而，这个增强的 user class 路径只是用于执行这些工具，对于处理源代码的工具，如 `javac` 和 `javadoc`，它们使用的是原始的 class 路径，而不是这个增强版本的 class 路径。

Java 虚拟机查找 Extension classes 的方式

Extension classes 是一些用于扩展 Java 平台的 class 文件。这些用于扩展 Java 平台的 class 文件被组织成 jar 包的形式，存储在 `jre/lib/ext` 目录下。在这个目录下的 jar 文件会通过 Java Extension Framework 进行加载。在这个目录下的松散的 class 文件不会被查找，这些 class 文件必须是以 jar/zip 包的形式打包以后才可以被查找。这个扩展包存放的目录的位置是不能被修改的。
在目录 `jre/lib/ext` 目录下如果存在多个 jar 文件，并且这些 jar 包中包含了同名的 class 文件，如：

```
smart-extension1_0.jar contains class smart.extension.Smart

smart-extension1_1.jar contains class smart.extension.Smart
```

那么，具体加载上面哪个 smart.extension.Smart 类是未定义的。

Java 虚拟机查找 User classes 的方式

Uesr Classes 是一些在 java 平台上创建的 class 文。Java 虚拟机通过引用 *user class path* 来查找这些 class 文件的位置， *user class path* 是一个包含了 class 文件的目录、jar 包和 zip 包的路径列表。

一个 class 文件拥有一个反映类的全限定名的子路径名，如：假设有一个名为 `com.mypackage.MyClass` 的类，并且存放在目录`/myClasses` 下，那么目录 `myClasses` 必须在 *user class path* 中，并且 MyClass 这个类的路径必须是 `/myClasses/com/mypackage/MyClass.class`。如果这个类被存储在 `myclasses.jar` 这个 jar 包中，那么 `myclasses.jar` 这个包必须在 *user class path* 中，并且这个类在 jar 包中的路径必须是 `com/mypackage/MyClass.class`。
用户类路径，也就是 *user class path* ，是以字符串的形式指定的，在 Unix 下通过:进行分隔，而 Win 下使用;进行分隔。Java 虚拟机会将用户类路径中的字符串添加到 `java.class.path` 系统属性中，这个属性的值有几种设置途径：

- 默认值是 .，表示当前目录
- 环境变量 `CLASSPATH` 的值，这个值会覆盖默认值。
- 在命令行中通过`-cp / -classpath` 选项指定的路径值，这个值会覆盖默认值和 `CLASSPATH` 环境变量的值。
- 在命令行中通过`-jar` 选项指定的 jar 包的路径值，这个值会覆盖上面所有的值，如果这个选项被设置，那么所有的用户类必须通过 jar 包的形式指定。

通过上面列出的规则，我们可以知道，`CLASSPATH` 环境变量的值只是 Java 查找类的一中方式。

Java 虚拟机查找 jar 包中的类的方式

一个 jar 包文件通常包含一个称为 `manifest` 的文件，这个文件包含了这个 jar 包中的内容。这个 manifest 文件可以定义一个称为 `JAR-class-path` 的类路径，这个类路径可以用于扩展 Java 的类路径（前提是这个 jar 包被加载进来）。通过 `JAR-class-path` 访问类的顺序定义如下：

- 一般情况下，在一个 jar 包中的所有 class 文件都是通过一个 JAR-class-path 入口引用的，在查找的时候也是以 JAR-class-path 入口在类路径中的先后顺序进行访问的。
- 如果 JAR-class-path 指向的 jar 包文件已经被查找过，那么这个 jar 包文件将不会被再次查找（这种优化可以提高效率，并且避免循环查找）。
- 如果一个 jar 包文件是作为一个扩展包安装的，那么这个 jar 包定义的 JAR-class-path 会被忽略。所有的在这个 jar 包中的类文件会被认为是 SDK 的一部分，或者已经被作为扩展包安装。

引用

http://docs.oracle.com/javase/7/docs/technotes/tools/findingclasses.html
http://www.ntu.edu.sg/home/ehchua/programming/java/J9c_PackageClasspath.html

分类: Programming Language

标签: Java