# 第一部分 Groovy起步

## 起步

Groovy下载：<http://www.groovy-lang.org/download.html>

Groovy官网：<http://www.groovy-lang.org/>

安装exe文件即可，安装时会自动设置GROOVY\_HOME，如果不是使用exe安装而是使用解压版本则需要手动设置GROOVY\_HOME，设置为：C:\Program Files (x86)\Groovy\Groovy-2.4.10，Path中增加：%GROOVY\_HOME%\bin; Groovy需要使用到Java，所以也需要配置有JAVA\_HOME。

Groovy语言于2003年启动开发，Groovy 1.0版本发布于2007年1月2日，2.0版本发布于2012年

Groovy是一门动态语言。

为要使用动态语言？因为动态语言可以在运行时扩展程序，包括修改类型、行为和对象结构，可把静态语言（如Java）在编译时做的一些事情放在运行时做，甚至可以执行在运行时即时创建的程序语句。

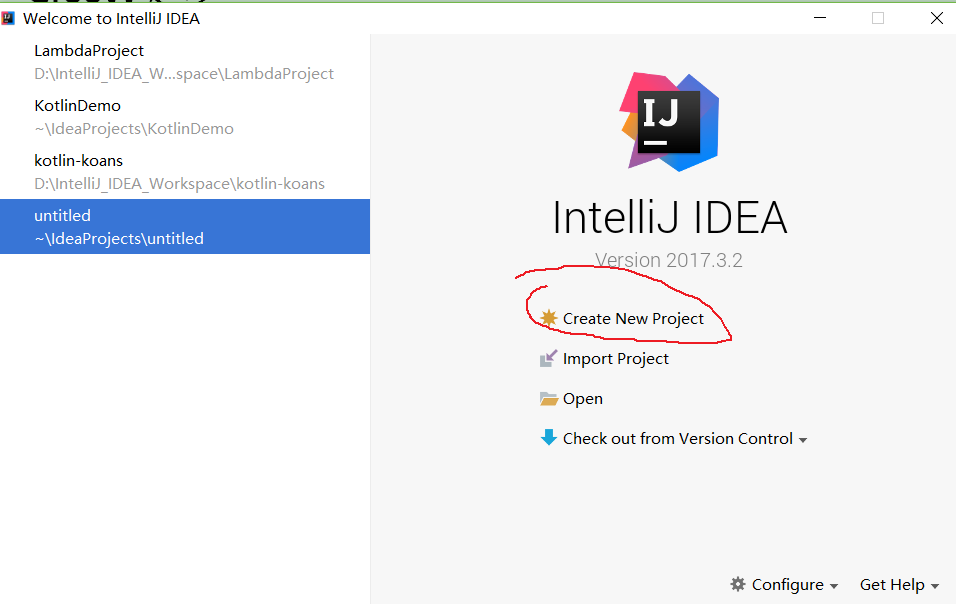
为何选择Groovy：

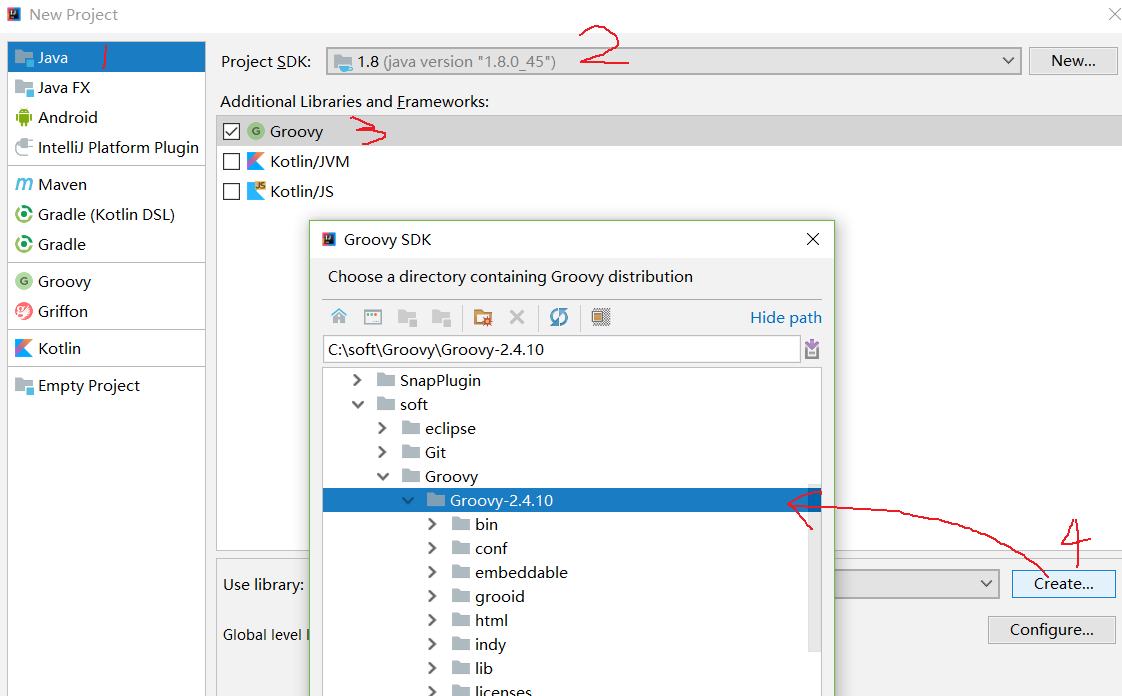
* 易于掌握
* 遵循Java语义 （几乎可以把任何Java代码当作Groovy代码来运行）
* 满足了我们对动态语言的热爱
* 扩展了JDK

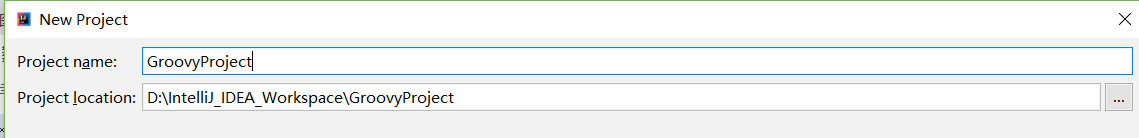
运行Groovy：Ctrl + R或Ctrl + Enter都可以

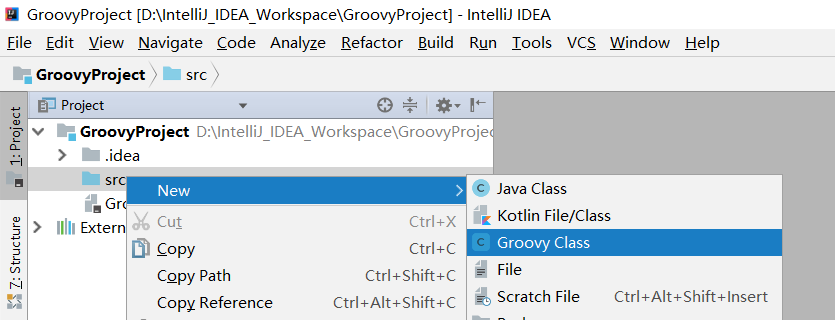
最好使IntelliJ IDEA进行开发：

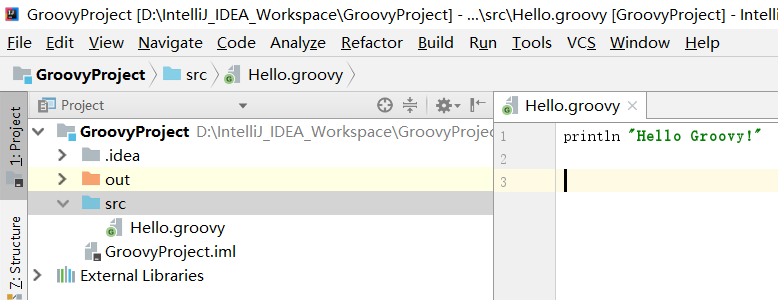
创建项目：











Ctrl + Shift + F10运行

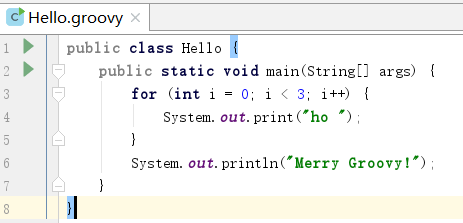
## 第2章面向Java开发者的Groovy

因为Groovy支持java语法，并且保留了Java语义，所以我们可以随心所欲地混用两种语言风格。

### 2.1 从Java到Groovy

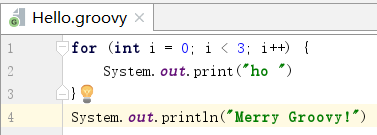
#### 2.1.1 Hello, Groovy

Java风格，创建Hello.groovy文件，并编写代码，如下：



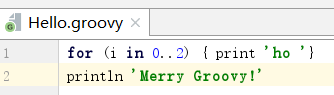
这么简单的任务，代码可真够多的。

Groovy的信噪比比Java要高，所以可以用较少的代码获得更多的结果，上面的代码可以删除一行结束的分号，去掉类和方法定义也可以，如下：



**默认导入：** Groovy默认导入java.lang、java.util、java.io和java.net。它也会导入java.math.BigDecimal和java.math.BigInteger两个类，此外，它还导入了groovy.lang和groovy.util这些Groovy包。

Groovy能够理解 println()，因为该方法已经被添加到java.lang.Object，它还有一种使用Range对象的、更为轻量级的for循环形式，而且Groovy对括号很宽容，因为前面的例子可修改为：



#### 2.1.2 实现循环的方式

前面我们已经使用过了range 0..2。其实Groovy为我们提供了很多优雅的迭代方式。

比如upto()，它是groovy向java.lang.Integer类中添加的一个便于使用的实例方法，可用于迭代，示例如下：

0.upto(2) { print **"**$it **"** }

输出结果为：0 1 2

"$it" 在这个上下文中，它代表进行循环时的索引值。

0是Integer的实例，方法upto实现循环，从0到2的循环，这里输出时只能使用双引号，如果使用单引号，则里面的$it不能识别为循环的索引值。

使用upto方法可以设置循环的上限和下限，如2.upto(4)，3.upto(20)，如果上限是从0开始的，可以使用times，如下：

3.times {print **"**$it **"**}

输出结果为：0 1 2

在循环时要跳过一些值可以使用step方法：

0.step(10, 2) {print **"**$it **"**}

输出结果：0 2 4 6 8

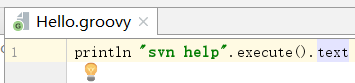
#### 2.1.3 GDK一瞥

Java平台的核心优势之一就是其Java开发包（JDK）。Groovy并没有强迫我们学习一组新的类库。通过向JDK的各种类添加便捷方法，Groovy扩展了强大的JDK，扩展了之后就称为GDK（或Groovy JDK）。

Java中可以使用java.lang.Process与系统级进程交互。假设我们想在代码调用Subversion的help，下面是java的实现代码：

**import** java.io.\*;  
  
**public class** JavaDemo {  
 **public static void** main(String[] args) {  
 **try** {  
 Process process = Runtime.*getRuntime*().exec(**"svn help"**);  
 BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(process.getInputStream()));  
 String line;  
 **while** ((line = reader.readLine()) != **null**) {  
 System.***out***.println(line);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Process类非常有用，但是我们在使用它时，真是大费周章，异常处理代码以及为实现输出所做的努力，所有的这些让我们头晕目眩。通过在java.lang.String类上添加一个execute()方法，GDK使这一切变得非常简单：



可看到，无需要导包，无需处理异常。

当在String实例上调用execute()方法时，Groovy创建了一个扩展了Process类的实例，验证如下：

println **"svn help"**.execute().getClass().name

打印结果为：java.lang.ProcessImpl

当调用text时，我们是在调用Groovy在Process类上添加 的getText()方法，其功能是将该进程的整个标准输出读到一个String对象中。如果只是想等待进程结束，waitFor()或Groovy添加的waitForOrKill()方法（该方法接收一个以毫秒表示的超时值）会有所帮助。

上面的示例，把svn help换成java -verson没有输出，换成groovy -v会挂异常，不知道是不是因为我安装的Groovy是2.x版本，不支持我的JDK 1.8？试验证明不是，我通过创建普通的Java Project，然后用Java的代码去运行也是一样的结果。

在一个Unix系统上，要得到当前目录下内容的列表，可以调用ls，如下：

println **"ls -l"**.execute().text

在Windows上，简单地把ls替换为dir是不起作用的。原因在于dir并不是一个程序，它只是一个shell命令，所以我们需要调用cmd，并让它来执行dir命令：

println **"cmd /C dir"**.execute().text

### 2.1.4 安全导航操作符

Java中的非空判断很常用，在Groovy中非常简化，如下：

**def** foo(str) {  
 str?.reverse()*// 这里的“?.”是安全导航操作符，这句代码等同于：if(str != null) { str.reverse() }*}  
  
println foo(**'evil'**) *// 输出：live*println foo(**null**) *// 输出：null*

### 2.1.5 异常处理

与Java相比，Groovy少了很多繁文缛节，这一点在异常处理上极其明显。Java强制我们处理所有受检查异常（Checked Exception），如下：

**try** {  
 Thread.*sleep*(1000);  
} **catch** (InterruptedException e) {  
 *// 啊？这里该做什么？我都因为这个寝食难安了。*}

Groovy中的异常如果不捕获，会自动传给上一层，如下：

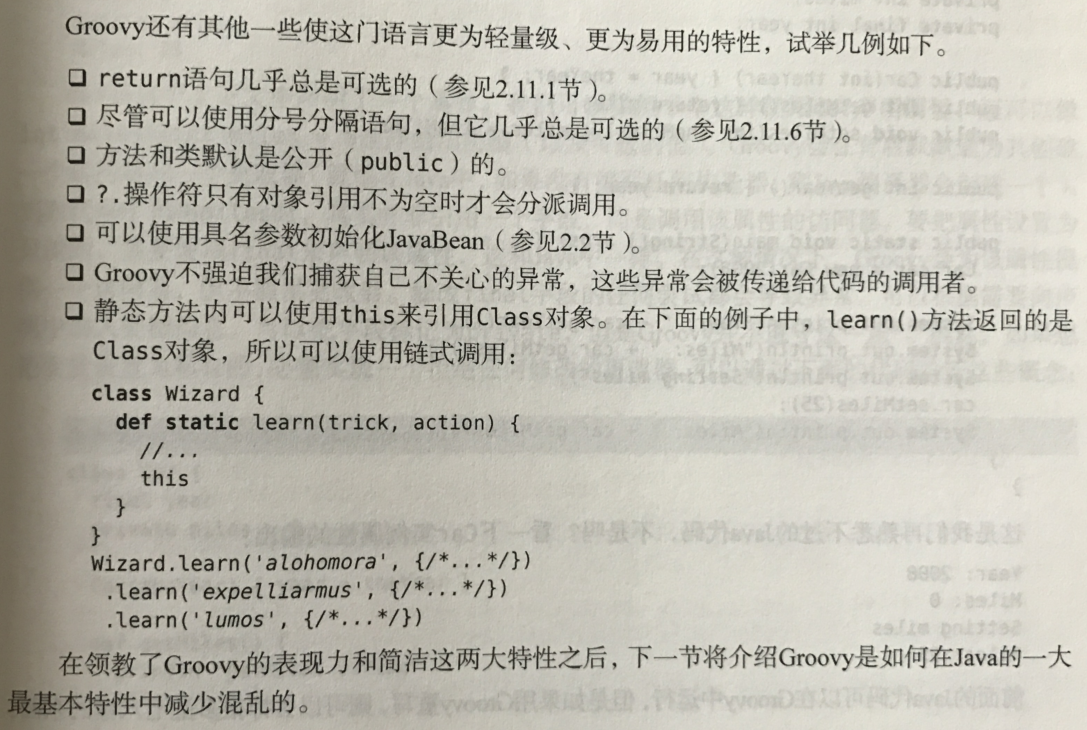
**def** openFile(fileName) {  
 **new** FileInputStream(fileName);  
}  
  
**try** {  
 openFile(**"nonexistentfile"**);  
} **catch**(FileNotFoundException ex) {

println **"Oops: "** + ex  
}

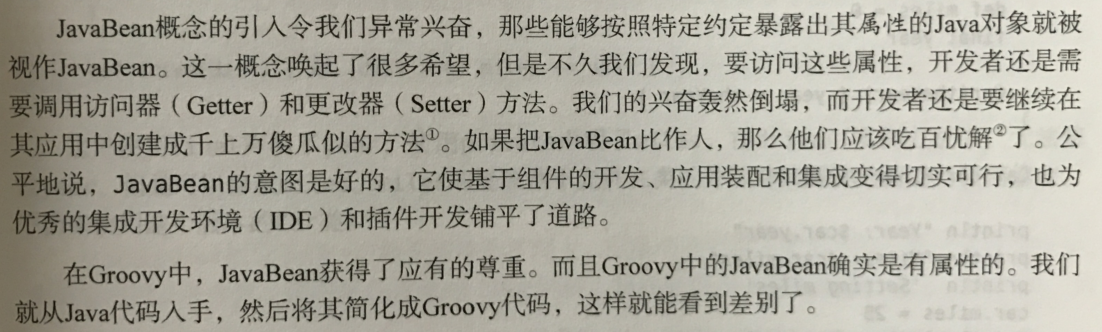
如果想捕获所有的异常，可以省略异常的类型，如下：

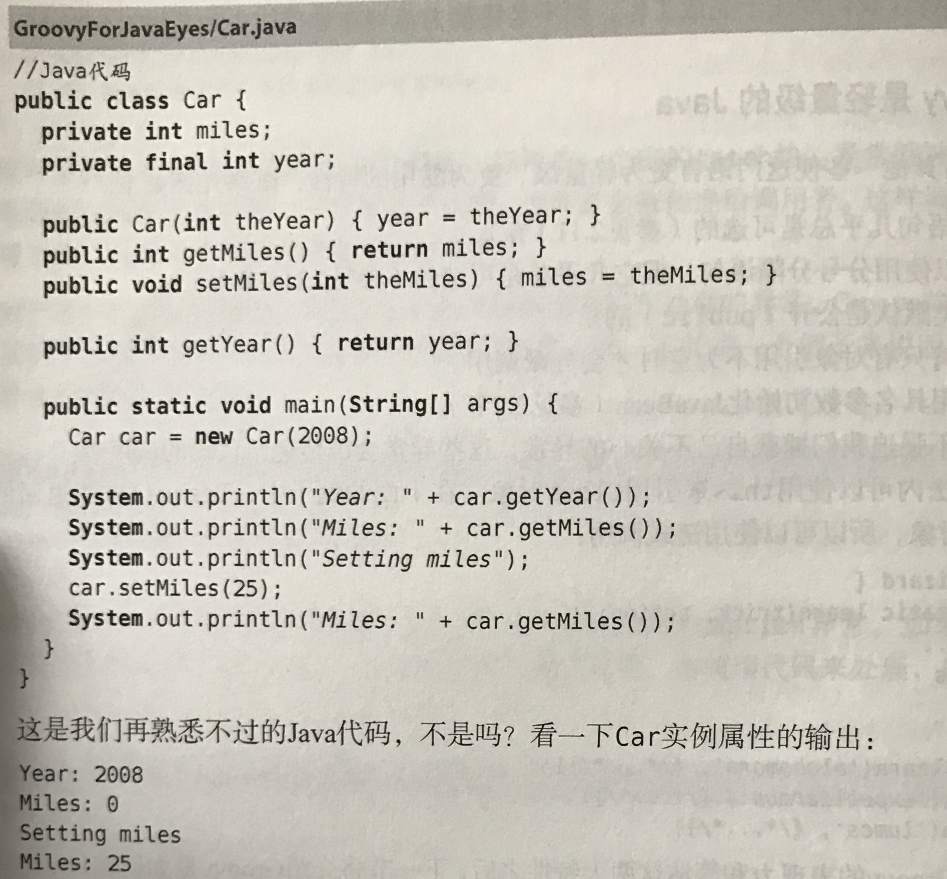
**try** {  
 openFile(**"nonexistentfile"**);  
} **catch**(ex) { *//这里没有声明是什么异常，可以捕获任何的异常，Error或Throwable除外，要捕获这些异常请使用Throwable th* println **"Oops: "** + ex  
}

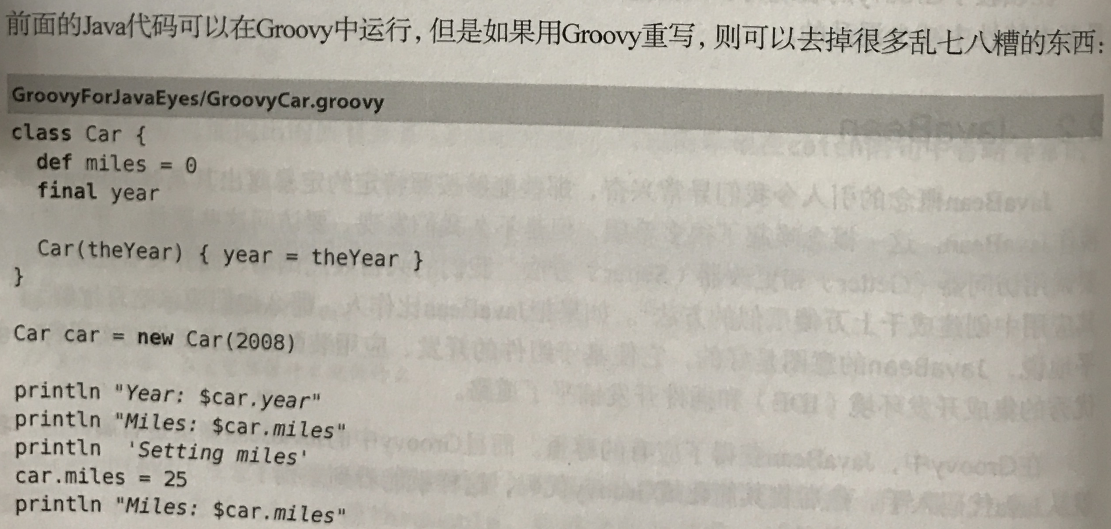
### 2.1.6 Groovy 是轻量级的Java

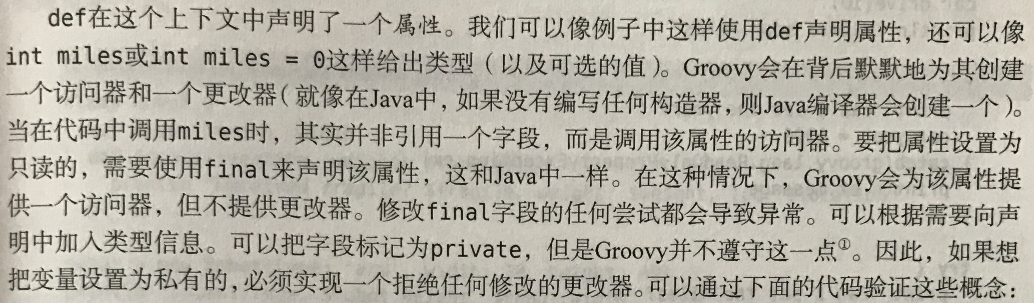


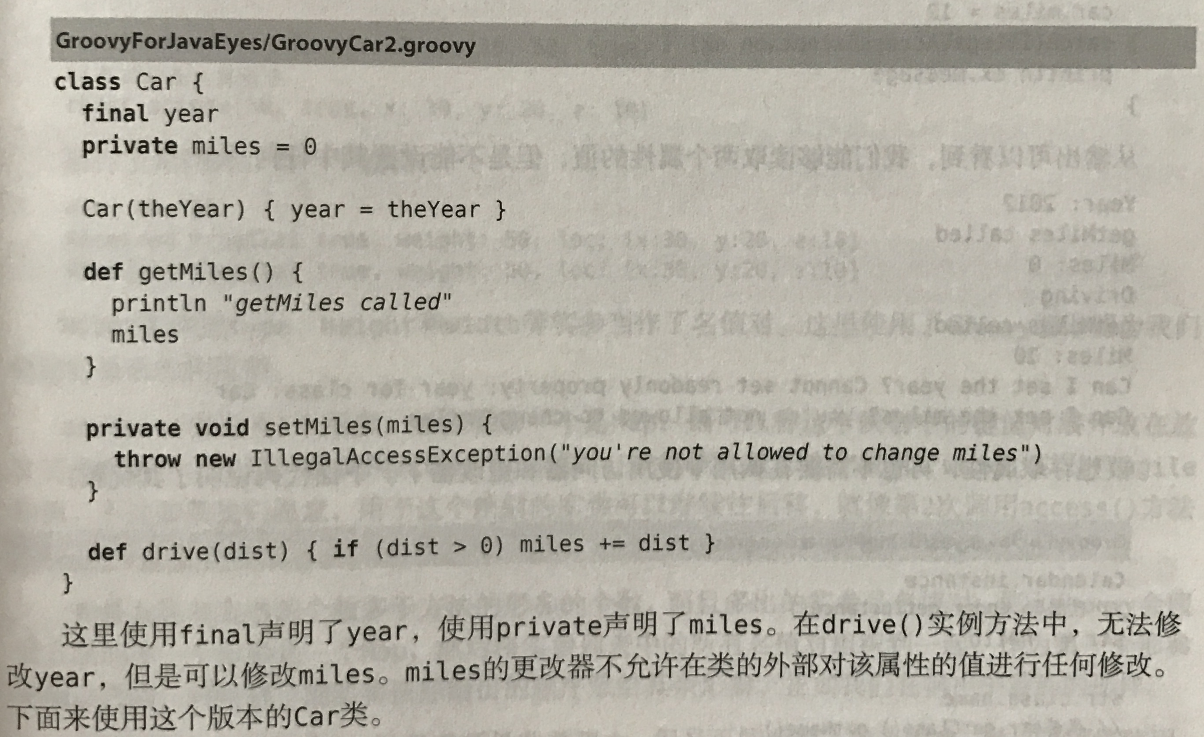
### 2.2 JavaBean



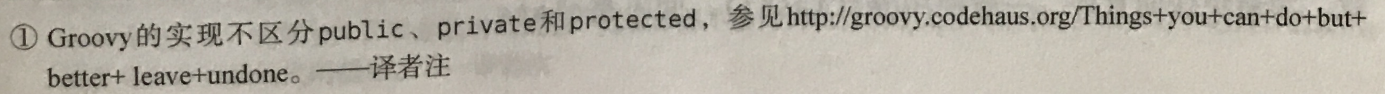


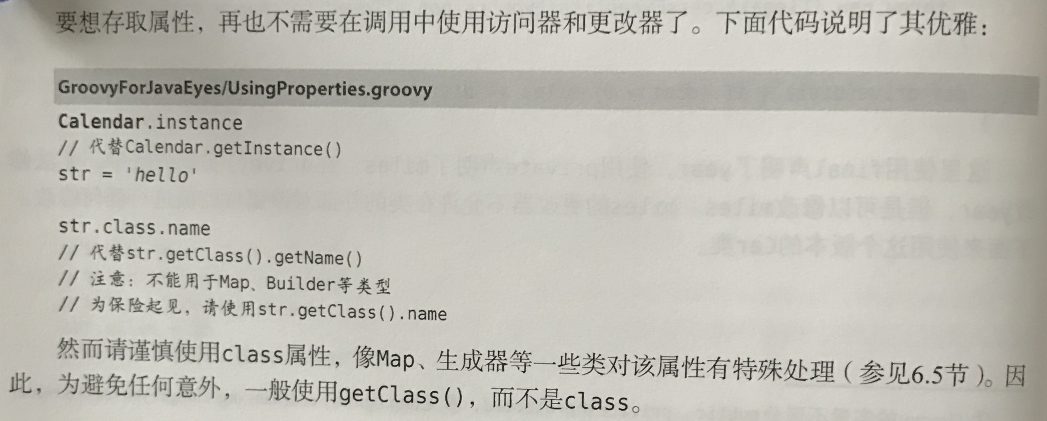






上面代码中setMiles方法就是抛出了一个异常，这样做是因为Groovy会为miles生成访问器，所以我们手动创建这个访问器并抛出异常，这样就相当于禁止了private的访问器的使用。





2.3 灵活初始化与具名化参数

## 第3章动态类型

## 第4章使用闭包

## 第5章使用字符串

## 第6章使用集合类