2.2 使用 Google 图表工具

Chart API 是一种 REST 式的 web 服务，同时也是谷歌公布的众多 API 之一。就是下面那个图的 API。它的文档在 https://developers.google.com/chart/image/。这个图表工具为 JavaScript 使用者提供了大量的 API，但是在输入端最终却是带有查询参数的 URL。

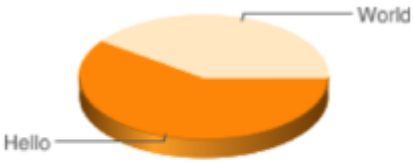

Figure 2.1 The Google Chart API "Hello, World" example

开发者发送请求到 URL https://chart.apis.google.com/chart 同时还得附加一些请求参数，以获得确定的图表种类、型号、数据等。这个 API 也需要一个“Hello World”例子，这有一个饼图的 URL（结果你刚刚已经看到了）

```
https://chart.apis.google.com/chart?cht=p3&chs=250x100&chd=t:60,40
&chl=Hello|World
```

上面那段 URL 是一个(但是一行里放不下)。输入这个 URL 以后一个被分为 60%、40%；250x100 像素的；分别写着“Hello”和“World”的 3D 饼图就呈现在你眼前了。

刚刚展现的 URL 是写死的，为了更有通用性，我将向你展示一种从“string、lists、maps、closures、buiders”中获取 URL 的方式。

目标：用 Groovy 写一个段脚本来生成这个 3D 饼图桌面应用。

接下来，我们将讨论：

字符串的操作

Lists 和 maps

用 closures 获取数据

Groovy 的构建者类

一开始我将会写一个简单的小脚本，一点点将它改成最终版本。

2.2.1 用查询字符串构建 URL

一开始，我们需要一个变量去表达基础 URL。在 Groovy 脚本中实际上你不必定义任何类型。一旦你定义一个类型，这个变量就会变成局部变量。如果没有，这个变量就会是“局部绑定”的（什么是“局部绑定”我们下章再讲）。因为我知道这个 URL 在我改变它之前一定包含一个字符串，所以我将宣称这个变量为 java.lang.String 类型

```
String base = 'http://chart.apis.google.com/chart?'
```

Groovy 的类型是任选的。这意味着你可以确定一个类型，或者在你不知道或不关心变量类型时可以用一个 del 关键字去宣称变量。Dierk Koening Groovy in action 的作者曾说过：

用 def： 如果你知道这个类型，就去定义它吧。话句话讲，如果你知道这个变量是

String 或者 Date，或者是一个 Employee，就去定义这个变量吧。
以我的经验来看，最开始我用 def 很多，但随着时间的推移，我越来越少用它。我十分同意 Dierk，现在在我用 def 之前我都会停下来想一会，这个变量是不是有一个确定的类型。但也有很多跟我意见相左的开发者。可变换类型的语言魅力就在于此：它可以容下任何人。
现在我需要为 URL 构建查询参数了。我用将要用一种这类问题的典型用法来取代将查询参数直接写死在 URL 中，这种方法就是构建一个 map，然后从这个 map 中生成查询字符串。带着这种思像，这有些 map 参数。

```
def params = [cht:'p3',chs:'250x100',chd:'t:60,40',chl:'hello|World']
```

在 Groovy 中，能用一个方括号来建立一个 map，每个键值对的键和值分别在冒号的左右两边。键的类型默认为字符串，值可以为任意类型。默认的，变量'params'为 java.util.LinkedHashMap 类的实例。

容器：Groovy 有自带的 lists 和 maps，maps 的键默认为 string 类型的。
每一个对应的值在单引号之间。在 Groovy 中，在单引号中的字符串是 java.lang.String 的实例。双引号中的字符串是"被修改过的"字符串，叫做 GString。我将给你展现一个例子。

为了将 map 变成一个查询字符串，我首先要将 map 中的键值对变成"key=value"形式的字符串，并且随后我要用分隔符将它们串联在一起。第一步就是用 collect 方法变换（这是一个特别的、在 Groovy 全部容器类都有的方法）。Collection 方法将一个闭包作为参数，这适用于容器中的每一个元素，能返回一个包含结果的新容器。

闭包将会在下一个补充块中被介绍，并且对他的讨论将会贯穿整部书，但目前认为他们的代码块代表一个函数体，这可能需要虚拟参数。当被用作 map 这个闭包可能会有一两个参数，当只有一个参数的时候，它代表着 map 中的键值对，当有两个参数时，一个代表 key 另一个代表 value。

为了将 map 变成 key=value 的 list，我们在 collect 方法中需要两个参数。

```
Params.collect{k,v->"$k=$v"}
```

在 Groovy 中，如果任何一个方法最后一个参数是闭包，你就可以将这个闭包拿到小括号外面。如果集合的唯一参数是闭包，这样的话圆括号可以省略。

什么是闭包？

闭包是一段被括号分隔的代码块，它能被视作一个对象。这个箭头符号用于指定虚参。在当前的例子中，闭包被适用于 map，两个虚参分别是 k 和 v（代表着键值对中的 key 和 value）。在箭头右边的表达式代表，将每对 key 和 value 带入被'='分隔的 GString 中。Collect 方法将每个键值对变成目标形式的字符串，最后产生一个包含结果的 list。

操作结果在这里：

["cht=p3","chs=250x100","chd=t:60,40","chl=Hello|World"]

图 2.2 说明了这一过程。

为了构造查询字符串，用一个 Groovy 加入容器类的方法—---join。这个 join 方法有一个参数，用来加到相邻两组字符串之间。为了构造查询字符串，调用 join 伴着一个分隔符作为参数。

["cht=p3","chs=250x100","chd=t:60,40","chl=Hello|World"].join('&')

我们需要的字符串就出来了：

"cht=p3&chs=250x100&chd=t:60,40&chl=Hello|World"

我们总结一下获得这个 URL 的过程，用基础的 URL，和参数 map 构建：

```
String base = 'http://chart.apis.google.com/chart?'
def params = [cht:'p3',chs:'250x100',
              chd:'t:60,40',chl:'Hello|World']
String qs = params.collect { k,v -> "$k=$v" }.join('&')
```

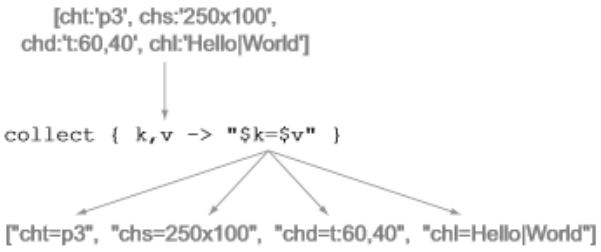

Figure 2.2 Apply `collect` to a map to convert it into a list, where each entry is transformed into a string.

现在我们得到的结果其实是一个字符串，而不是 URL。在将它变成 URL 之前，让我们来验证这个程序能否正常工作。正常情况下我们需要测试，在第 6 章中我们再谈测试。这我们只需要用 Groovy 中 assert 这个关键字，他会用到一个布尔表达式参数。如果最后结果是 true，什么都不会返回。但如果错了，就会在打印错误报告。如果我用 Map 中的 contains 方法，检查每个键值对是否被修改成了合理的形式：

```
params.each{     k,v ->
      assert    qs.contains("$k=$v")
}
```

这个 assert 关键字：Groovy 的断言，是一个简单的判断正确性的方式。如果最后结果是 true，什么都不会返回。但如果错了，就会在打印错误报告。

这个 join 方法好处之一就是你不用担心分隔符被加到了字符串的开始或者结尾。它只会加到字符串中间。

注意：有一种情况括号（在 join 方法中）是必要的。在 Groovy 中，如果你调用一个函数，它没有参数，而且你没加括号，编译器就会假设你正在调用 getter 或 setter 方法。如果你想用 join 方法（而不是调用 getJoin()，它也不存在），你需要这个括号。

2.2.2 发送 URL

这个 Groovy 的 JDK 在 String 类中加入了 toURL()的方法。和你想象的一样，这个方法可以将 java.lang.String 的实例对象转化成 java.net.URL 的实例对象。

Groovy 的 JDK

Groovy 在现有的 java 类库中加了很多有用的方法。一次次，当我发现这些被添加在 String、Date 或 Collection 中的方法是，我多希望它也被加入到了原生 java 中。Groovy 的 JDK 就是这些被添加的方法集合，它们有他们自己的 JavaDoc。Groovy 的 JDK 文档在 Groovy 的官网上可以下载。

Groovy 的 JDK 更多细节将在第 3 章被讨论。

为向这个 URL 发送 HTTP GET 请求，并且获取结果，将字符串变换成 URL。就得调用 Groovy 的 JDK 中领一个方法----------getText()方法（被加入到了

java.net.URL 之中）。换句话说，一个网页的数据能通过 url.toURL().text 方式获取。

在这里我故意去用.text 这种写法，因为在 Groovy 中它会去调用 getText()方法。在这里调用 getText()方法没有任何错误。但是之前那种是更加地道的 Groovy 写法。

我们代码如我们所料的运行了。在本章中我通过一些例子，向大家展示了一些 web 服务的技术。但有些特殊的时候，如果我们通过 Google 图表获取的信息是个图片，那么将它变成文本就是没意义的。

Groovy 的属性

访问一个类的属性时，Groovy 中自动会调用相关的 getter、setter 方法。

接下来我将要构建一个图形用户接口，它将包括在 javax.swing.ImageIcon 的图片。我将要给大家举一个“构建者”的例子，这是一个使你加深对于 Groovy 中元编程理解极好的例子。

2.2.3 用 SwingBuilder 创建一个 UI 界面

在 Groovy 中每一个类都有一个元类。一个元类是实际调用进程的类。如果你调用一个类不存在的方法，这个调用最终会被在元类中叫 methodMissing 的方法拦截。类似的，如果你访问了一个不存在的属性，同样会被一个叫 propertyMissing 的方法拦截。订制 methodMissing 和 propertyMissing 的行为是 Groovy 元编程的核心。

Groovy 元编程是一个很大的课题，但在这我就将展示其中一个有用的方面：创建一个构建者类。在构建者类中，如果调用 methodMissing 将会对这种类型的构建者，做出特别的行为。

在这里我将举例说明一个 Swing 构建者。这个类要将组成和构造 Swing 用户接口的名称拦截在外。这事实上演示要比解释更容易。在我们开始之前在引入几个 Google 图表的脚本：

```
import java.awt.BorderLayout as BL
import javax.swing.WindowConstants as WC
import groovy.swing.SwingBuilder
import javax.swing.ImageIcon
```

自动引入：

你也许注意到了，我刚刚没有进行任何引入操作。Java 会自动引入 java.lang 这个包。Groovy 会自动引入 java.util, java.io, java.net, groovy.lang, groovy.util, java.math.BigInteger 还有 java.math.BigDecimal 以及 java.lang。

这段脚本中，我们将从 java 标准库中引入三个类。前两个引入用了 as 操作给了各自类一个别名。就是当用到 BorderLayout 和 WindowConstants 时，

可以用 BL 和 WC 取代。我也加入了 ImageIcon 类，它将用来捕获 Google 图表传回来的图片。从 Groovy 类库中引入的 SwingBuilder 将要负责 Swing UI 的创建。

as 关键字：这个 as 关键字有几种用法，其中之一就是为引用提供别名。

这个 as 关键字相当于 asType 方法（它会讲 java.lang.Object 作为 Groovy JDK 的一部分加入其中）

在 SwingBuilder 中，有时你会调用本不存在的方法，但是 Swing API 会去调用其他相应的方法。例如当你调用 frame 方法时，实际上你在实例化 JFrame 类。给它一个类似于 map 的参数 visible：true 相当于调用 setVisible 方法并且传了一个 true 参数。

这有一个用了构建者的代码。每一个方法都不在 SwingBuilder 中，而是调用在 Swing 类库中相应的方法:

```
swingBuilder.edt {
  frame(title:'Hello, World!', visible:true, pack: true,
      defaultCloseOperation:WC.EXIT_ON_CLOSE) {
        label(icon:new ImageIcon("$base$qs".toURL()),
            constraints:BL.CENTER)
  }
}
```

这个 edt 方法用分支线程创建了一个 GUI。它有一个闭包作为参数，随后有意思的事就开始了。在闭包中第一句是调用 frame 方法，但事实上在 SwingBuilder 中没有这个方法。这个构建者的元类就拦截了这个调用并且将它作为 javax.swing.JFrame 类的实例的一个请求。这个 frame 方法在这里陈列了一系类 map 键值对，来用于 JFrame 的标题，可视化，关闭操作。这个构建者将它理解为 JFrame 实例 setTitle, setVisible, 和 setDefaultCloseOperation 方法的调用。

在圆括号之后又有一个闭包。它们被理解为 JFrame 实例的组成成分。下一调用是 label 方法，这在这里是不存在的。这个 Swing 构建者知道要创建一个

JLabel 作为结果，调用它的 setIcon 方法用从 Google 图表返回的图片 new 一个 ImageIcon 对象，并将 JLabel 放在 BorderLayout 的中央。

最终，在这个 frame 闭包以后，我调用 JFrame 中的 pack 方法，来返回一个能足够装下图片的 GUI。下面有完整的代码。

Figure 2.3 The "Hello, World" Swing user interface, holding the image returned by Google Chart

代码清单 2.1 用 Google 图表构建一个 3D 饼图

```
import java.awt.BorderLayout as BL
import javax.swing.WindowConstants as WC
import groovy.swing.SwingBuilder
import javax.swing.ImageIcon
def base = 'http://chart.apis.google.com/chart?'
def params = [cht:'p3',chs:'250x100',
              chd:'t:60,40',chl:'Hello|World']
String qs = params.collect { k,v -> "$k=$v" }.join('&')
SwingBuilder.edt {
    frame(title:'Hello, Chart!', pack: true,
        visible:true, defaultCloseOperation:WC.EXIT_ON_CLOSE) {
            label(icon:new ImageIcon("$base$qs".toURL()),
                constraints:BL.CENTER)
    }
}
```

结果如图 2.3 所示。

本节学到了什么？

1. Groovy 的变量是有类型的，但当你不知道或不关心它的类型时 你可用 def 关键字。这中 def 类型也能用在方法的返回值类型和参数类型。
2. Groovy 对于 lists 和 maps 有原生的语法。这里有几个用 Groovy map 的例子；list 的例子还有很多。
3. 闭包像带有参数的方法体。
4. 这个 collect 方法可以通过对每一个元素提供闭包和返回列表，来

生成容器。

5．Groovy 的 JDK 在标准 java API 中添加了很多方法。

6．Groovy 的解释器和构造器可以简单的与众多 API 工作。

下一个例子是展示 Groovy 对于 xml 解释、生成的能力，还有数据库操作，正则表达式，groovlets 等。