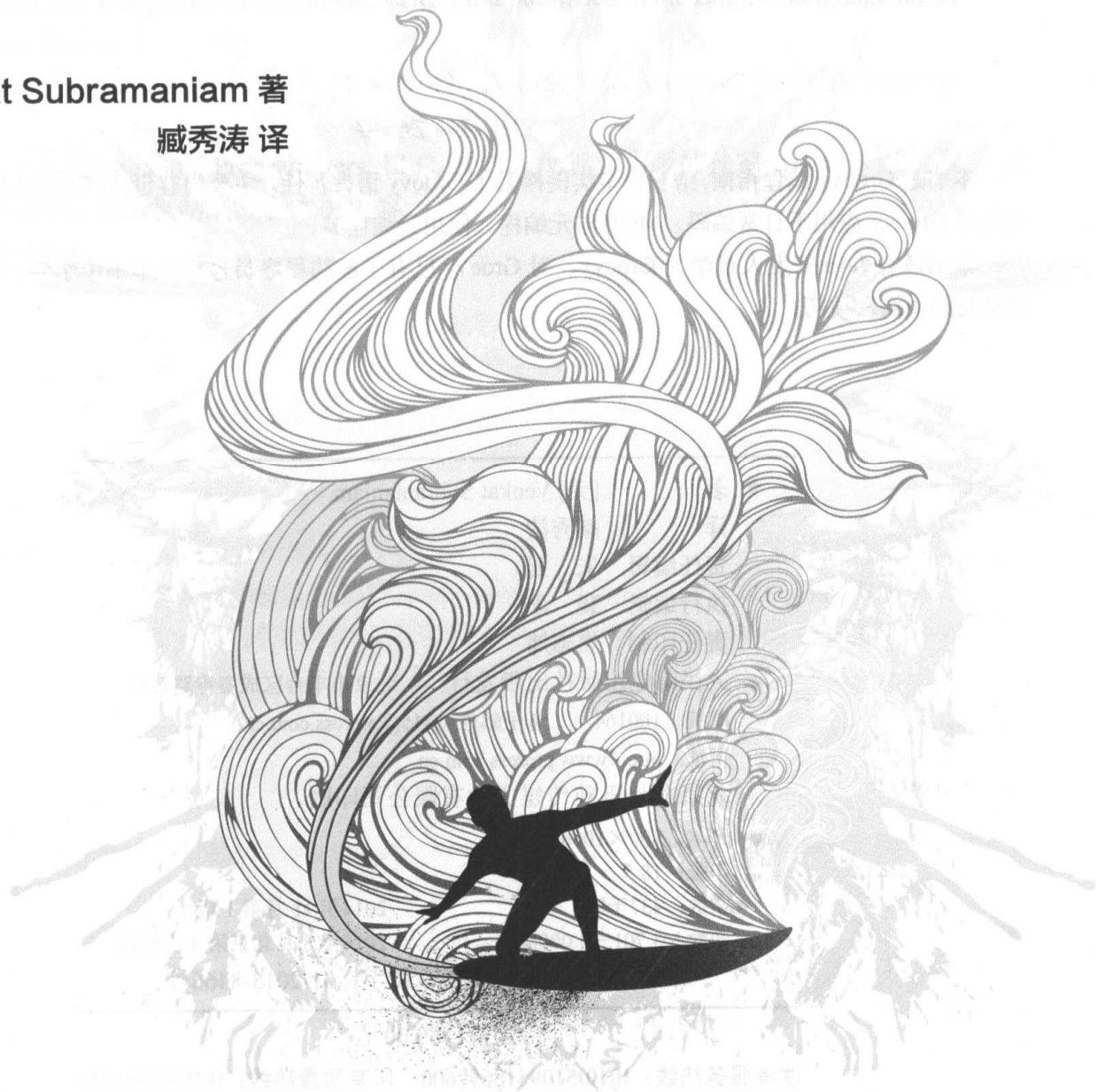


TURING

图灵程序设计丛书

Groovy程序设计

【美】Venkat Subramaniam 著
臧秀涛 译



人民邮电出版社
北京

序

Venkat 曾著书引导读者学习 Groovy 1.5 的所有功能特性，助其成为娴熟的 Groovy 开发者。俗话说，光阴似箭。现在是时候探索一下 Groovy 2 都有哪些功能特性了。当然，Venkat 这位深受读者喜爱的作家都为我们考虑到了。

对于 Groovy 的 2.0 版本，我们 Groovy 团队主要把精力投放在了以下三个方面。首先，使 Groovy 与 JDK 7 接轨：添加了 Java 7 “Project Coin” 所带来的语法增强；用 `invokedynamic` 字节码指令和内部的 API 来支撑 Groovy 的运行时。这样一来，即使用的是比较老的 JDK，也可以使用最新添加的语法。当然，如果运行 JDK 7 的话，还可以获得更好的性能体验。

其次，我们将 Groovy 分解成较小型的模块，包括一个核心模块和一些 API 相关的模块，所以你可以选择感兴趣的部分来组织自己的应用。我们还扩展了 Groovy 开发包（Groovy Development Kit），支持开发者创建自己的扩展方法，就像 Groovy 用著名的 `DefaultGroovyMethods` 类对 JDK 所做的增强那样。

最后，还有一点同样重要，我们引入了一个“静态”（`static`）主题，它包括两个比较新奇的地方：静态类型检查和静态编译。借助前者，我们可以在编译时轻松地捕获输入拼写错误及其他错误，甚至还支持对领域特定语言（Domain-Specific Language）进行类型检查；借助后者，对于应用中要求最高性能的关键部分，我们可以获得与 Java 同样的性能。

有了这些对语言和 API 的增强，Groovy 如美酒佳酿般继续趋向成熟；而 Venkat 就像乐于分享专长的调酒师，将他所知道的 Groovy 的所有强大特性，通过我们正要阅读的这本结构合理的书分享出来，帮助读者紧跟语言发展的步伐，同时更上一层楼。

Guillaume Laforge
Groovy 项目管理者
2013 年 6 月

引言

Java平台可以说是当下功能最为强大、应用最为广泛的生态系统之一。它有3个重要的组成部分。

- Java 虚拟机 (Java Virtual Machine, JVM)。这些年来，JVM已经变得越来越强大，性能也越来越好。
- Java 开发包 (Java Development Kit, JDK)。包括丰富的第三方类库和框架，可以帮助我们有效地利用 Java 平台。
- 基于 JVM 的语言集合。Java 语言当然是第一位的，这些语言集合可以帮助我们在 Java 平台上编写程序。

语言就像能使我们在平台上航行的交通工具，通过这些交通工具我们可以轻松抵达该平台的各个部分。截至目前，Java语言已经有了长足的进步，其类库也被重构和扩充过。尽管Java语言带给我们诸多好处，我们还是需要超越Java，寻找更为轻量级且高效的语言。如果使用得当，动态语言、函数式编程风格和元编程功能可以帮助我们更快速地航行。还以交通工具作比，这些可不是更快的汽车，而是飞行器，一种能将开发效率提高几个数量级的飞行器。

Java语言一直想引入元编程和函数式编程风格，但却总是摇摆不定。未来的版本将对其中的某些特性提供不同程度的支持^①。然而，我们不必等到那一天。现在，就在此时此刻，使用Groovy就可以利用所有的动态功能构建高性能的JVM应用。

Groovy 是什么

韦氏词典对groovy一词的定义是marvelous、wonderful、excellent、hip、trendy，有“非凡、绝妙、优秀和时髦”等意义。Groovy语言集合了上面这一切优点，它是轻量级的，限制较少，而且还是动态、面向对象的，并且运行在JVM上。Groovy基于Apache 2.0许可协议开源。它博采诸如Smalltalk、Python和Ruby等众家语言之长，同时保留了Java程序员熟悉的语法。Groovy编译为Java字节码，它还扩充了Java API和类库。Groovy基于Java 1.5及更高版本运行。要部署的话，除了常规的Java及其组件外，我们需要的就是一个Groovy的JAR文件，而Java的东西我们都已准备好了。

^① Java 8已于2014年3月发布，带来了Lambda表达式，支持一定程度的函数式编程。——译者注

Groovy是一门“几经重生”的语言。^①该语言由James Strachan和Bob McWhirter于2003年启动开发，之后于2004年3月成为JSR 241（Java Specification Request, JSR，即Java规范请求）。不久，因为存在一些困难和问题，该语言几近被放弃。Guillaume Laforge和Jeremy Rayner决定再次努力，并使Groovy重获新生。他们首先修复了一些bug，同时将语言特性稳定下来。前途未卜这种状态持续了一段时间，包括提交者和用户在内的很多人干脆放弃。最后，一群聪明热情的开发者加入到Guillaume和Jeremy的行列，一个充满生气的开发者社区形成了。

Groovy的1.0版本发布于2007年1月2日。令人鼓舞的是，在1.0版本发布之前，美国和欧洲的很多组织已经将Groovy应用于商业项目。一些组织和开发者开始在其项目的各个层次上使用Groovy，在产业中大量应用该语言的时机已然成熟。2012年的年中，Groovy的2.0版本发布了。

像Grails、CodeNarc、easyb、Gradle和Spock这样的框架和工具都是Groovy的闪光点。其中，Grails^②是一种基于“规约编程”（coding by convention）的动态Web开发框架，用到了Groovy的元编程功能。通过Grails，我们可以使用Groovy、Spring、Hibernate以及其他Java框架快速构建JVM上的Web应用。

为什么要使用动态语言

动态语言能够在运行时扩展程序，包括修改类型、行为和对象结构。利用动态语言，静态语言在编译时做的一些事情，我们可以在运行时做，甚至可以执行在运行时即时创建的程序语句。

例如，要计算8万美元的薪水提高5%是多少，只要这样写就行了：

```
5.percentRaise(80000)
```

没错，这就是java.lang.Integer对动态方法的友好响应。动态方法很容易添加，像这样即可：

```
Integer.metaClass.percentRaise = { amount -> amount * (1 + delegate / 100.0) }
```

可见，在Groovy中向类中添加动态方法非常容易。我们向使用delegate变量引用的Integer实例中添加了一个动态方法，负责返回增加相应百分比之后的美元数。

动态语言的灵活性给我们带来了在应用执行时演进程序的优势。这远远超越了代码生成。代码生成是20世纪才会考虑的技术。实际上，生成的代码就像持续的瘙痒，如果一直挠，就会转变为伤痛。而动态语言有更好的方式。代码合成（code synthesis）是运行时在内存中创建代码，动态语言使得代码合成更容易被人接受了。代码基于应用的逻辑流程合成，并即时（just in time）

^① 参见Guillaume Laforge的博客文章“A bit of Groovy history”（Groovy的一点历史）<http://glaforge.free.fr/weblog/index.php?itemid=99>。

^② <http://grails.org>

变为活跃的。

作为应用开发者，通过仔细应用动态语言的功能，我们可以更具开发效率。而更高的开发效率意味着可以在更短的时间内轻松创建更高层的抽象，也可以利用一组人数较少但更能干的开发者来创建应用。此外，更高的开发效率还意味着可以快速创建应用的某些部分，然后得到开发人员、测试人员、领域专家和客户代表等同仁的反馈，而这一切又会使我们更为敏捷。关于开发Web应用，Tim O'Reilly观察到：“不同于完成的画作，它们（即Web应用）只是轮廓，作为对新数据的响应而不断重绘。”在“Why Scripting Languages Matter”（为什么脚本语言至关重要）一文中^①，他也表达了动态语言更适合Web开发的观点。

动态语言已经存在了很长时间，那为什么现在让人倍感兴奋了呢？原因至少有四点：

- 机器速度
- 可用性
- 对单元测试的意识
- 杀手级应用

先从机器速度开始看。将其他语言在编译时做的事情拿到运行时做，这会引发人们对动态语言速度的担忧。在运行时解释代码，而不是简单地执行编译好的代码，也加剧了这种担忧。好在这些年来机器的速度一直在提升，今天手持设备的计算能力和内存都超过了几十年前的大型机。有些任务，使用20世纪80年代的处理器可能是难以想象的，但现在却可以轻而易举地实现。得益于处理器速度及本领域中其他方面的提升，包括更好的即时编译技术和JVM对动态语言的支持等，我们对动态语言性能的担忧已经大大缓解。

再来谈一下可用性。互联网和活跃的基于社区的“开放”开发方式，使较新的动态语言易于获得和使用。开发者可以轻松地下载到这些语言和工具，并加以研究及利用，他们甚至可以参与到社区论坛中来影响语言的演进。Groovy用户邮件列表非常活跃，经常有热心用户参与讨论，表达他们对当前和未来特性的意见、想法和批评。^②这使我们能够比以往更好地实验、学习并调整语言。

下面再来看一下单元测试意识。大部分动态语言是动态类型的。^③类型往往基于对上下文的推断。没有编译器在编译时标记类型强制转换违例。由于很多代码可能是在运行时合成的，而且程序可以在运行时扩展，所以不能单独依赖编写代码时的验证。从测试的角度看，相对于使用静态类型语言编写代码，使用动态语言需要更严格的自律。在过去的几年里，我们看到程序员对测试，特别是单元测试的意识在逐渐增强（尽管采用广度还远远不够）。大部分将这些动态语言应

① 文章链接：<http://www.oreillynet.com/pub/wlg/3190>。Tim O'Reilly探讨了应用的本质及脚本语言扮演的角色。

② 可以参阅<http://groovy.codehaus.org/Mailing+Lists>和<http://groovy.markmail.org>。

③ 这里需要明确，“动态语言”中的“动态”指的是前面提到的“将其他语言在编译时做的事情拿到运行时做”，而非“动态类型”中的“动态”。参见http://en.wikipedia.org/wiki/Dynamic_programming_language。——译者注

用于商业应用的程序员，已经采用了测试和单元测试。

最后，很多开发者实际上已经使用了几十年动态语言了。然而，要唤起业内大多数人对动态语言的兴趣，必须有可以与开发者和管理人员分享的杀手级应用，也就是有令人信服的应用案例。这一引爆点——往小了说是Ruby的，往大了说是动态语言的——以Rails^①的形式出现了。Rails显示出，使用Ruby的动态功能，苦苦挣扎的Web开发者可以如何快速地开发应用。同样，我们有Grails^②这种用Groovy和Java编写的Web框架，它提供了同样的开发效率和便利性。

这些框架在开发社区引起了足够的轰动，使得在整个业界应用动态语言有了极大的可能性。

动态语言，连同元编程功能，使简单的事情更简单，复杂的事情也可以掌控。当然我们仍然需要处理应用的内部复杂性，但动态语言让我们有可能把力气用在刀刃上。在使用了多年C++之后，当我接触到Java时，像反射、良好的类库以及不断演化的框架支持等特性带给我非常高的开发效率。JVM在一定程度上为我提供了使用元编程的能力。然而，除了Java，我还不得不想办法利用一些可能较为重量级的工具，比如AspectJ。和其他一些开发效率很高的程序员一样，我发现自己有两条路可走：使用极为复杂而且不那么灵活的Java，结合多种重量级工具；或者转而使用面向对象的、内建元编程功能的动态语言，比如Ruby。（举个例子，在Ruby和Groovy中实现面向方面编程只需要几行代码。）在几年之前，保持高开发效率的同时利用动态功能和元编程，就意味着要离开Java平台。（毕竟，这些特性的使用是为了提高效率，不能让它们拖慢我们的脚步，对不对？）世易时移，情况变了。现在有了诸如Groovy、JRuby和Clojure这样的动态且运行在JVM上的语言。使用这些语言，我们可以充分利用Java平台的丰富特性和动态语言功能。

为何选择 Groovy

作为Java程序员，我们不必完全切换到一门不同的语言。Groovy感觉就像我们已经熟知的Java外加一些扩展。

很多脚本语言都能在JVM上运行，如Groovy、JRuby、BeanShell、Scheme、Jaskell、Jython和JavaScript等，举不胜举。我们应该基于一系列标准来选择语言：需求、偏好和背景，开发的项目，以及公司的技术背景等。本节探讨一下Groovy何时是正确的选择。

因为下面一些原因，Groovy很有吸引力：

- 易于掌握
- 遵循 Java 语义
- 满足了我们对动态语言的热爱
- 扩展了 JDK

① <http://rubyonrails.org>

② <http://grails.org>

我们来详细研究一下。首先，几乎可以把任何Java代码当作Groovy代码来运行（有一些已知的存在问题的地方，参见2.11节），这意味着学习起来非常易于掌握。现在就可以开始在Groovy中编写代码了，如果卡壳，只需要换个思路，直接编写我们熟悉的Java代码。可以以后再重构那些代码，使其更符合Groovy风格。

例如，Groovy理解传统的for循环，因此可以这样写代码：

```
// Java风格
for(int i = 0; i < 10; i++) {
    //...
}
```

学习了Groovy后，可以将上面的代码修改为以下形式，或者修改为Groovy中其他风格的循环形式（现在不用担心语法，毕竟我们才刚刚起步，你很快就会成为这方面的专家）。

```
10.times {
    //...
}
```

其次，在使用Groovy编程时，Java有的Groovy几乎都有。Groovy类同样也扩展了古老的java.lang.Object类，Groovy类就是Java类。面向对象范型和Java语义也都保留了下来，所以在使用Groovy编写表达式和语句时，对于我们Java程序员而言，其实已经知道其意义。

这里有一个用以演示Groovy类就是Java类的小例子：

Introduction/UseGroovyClass.groovy

```
println XmlParser.class
println XmlParser.class.superclass
```

运行groovy UseGroovyClass，会得到如下输出：

```
class groovy.util.XmlParser
class java.lang.Object
```

爱上Groovy的第三个原因——Groovy是动态的，类型也是可选的。也许你已经在诸如Smalltalk、Python、JavaScript和Ruby等动态类型语言中体会过这些特性的好处，现在在Groovy中你也可以体会到。例如，要向String类添加isPalindrome()方法，以判断一个单词是否为回文结构，即正向拼写和逆向拼写是否一致，那非常容易，只需要几行代码（再次说明，现在不必尝试理解其工作原理的所有细节，本书其余部分会予以解决）：

Introduction/Palindrome.groovy

```
String.metaClass.isPalindrome = {->
    delegate == delegate.reverse()
}
word = 'tattarrattat'
```

```
println "$word is a palindrome? ${word.isPalindrome()}"  
word = 'Groovy'  
println "$word is a palindrome? ${word.isPalindrome()}"
```

通过输出来看看这段代码的效果：

```
tattarrattat is a palindrome? true  
Groovy is a palindrome? false
```

这说明不需要侵入一个类的源代码，即可使用便捷的方法轻松扩展该类，即便是神圣不可侵犯的java.lang.String类。

最后，Java程序员在工作时严重依赖JDK和API，而这些在Groovy中仍然可以使用。此外，通过Groovy JDK（GDK），Groovy使用便捷方法和闭包支持扩展了JDK。下面是一个简单的例子，说明了GDK对java.util.ArrayList的扩展：

```
lst = ['Groovy', 'is', 'hip']  
println lst.join(' ')  
println lst.getClass()
```

从这段代码的输出可以确认用到了JDK，但是除此之外，我们还能使用Groovy添加的join()方法把ArrayList中的元素连接起来：

```
Groovy is hip  
class java.util.ArrayList
```

Groovy扩充了我们所熟知的Java。如果项目团队熟悉Java，该组织的大部分项目中也使用了Java，而且有很多Java代码要集成和使用，那Groovy就是提高开发效率的极佳途径。

本书内容

本书是关于Groovy编程的，面向对JDK已经有所了解且有意学习Groovy语言及其动态能力的Java程序员，书中将结合很多实际的例子来探索Groovy语言的特性。希望本书能帮助程序员利用这门有趣且强大的语言快速提高开发效率。

本书主要内容组织为以下四个部分。

第一部分包括本书的前6章，这些章节关注Groovy相关的“为什么”，以及“是什么”，帮助我们适应Groovy常规编程的基础。本书是为有经验的Java程序员编写的，所以不会在诸如if语句是什么、如何编写if语句这种编程基础知识上浪费任何时间。相反，会直接深入介绍Groovy与Java的相似之处，以及Groovy的特性等主题。

第二部分包括第7章到第10章，我们将看到如何将Groovy应用于日常编码，包括使用XML、访问数据库以及使用多个Java/Groovy类和脚本，因此可以立即把Groovy应用于日常任务。我们还将探讨Groovy对JDK的扩展与补充，这样就可以兼顾Groovy和JDK的优势了。

第三部分包括第11章到第16章，将深入研究Groovy的元编程能力，Groovy真正的特色和优势就在于此，你还会学到如何利用其动态特性。本部分将从元对象协议（MetaObject Protocol, MOP）的基础入手，介绍如何在Groovy中完成类AOP操作，并探讨方法/属性的动态发现与分派。我们还将探索编译时元编程能力，同时看一下这种能力对于在编译阶段扩展与变换代码有何帮助。

第四部分包括最后3章，我们将应用Groovy元编程来创建和使用生成器及领域特定语言（DSL）。在Groovy中，因为其动态特性，单元测试是必要的。不过通过本部分的学习，你会发现单元测试也很容易，我们可以使用Groovy对Java和Groovy代码进行单元测试。

你正在阅读的是引言部分，下面分别介绍一下每章的内容。

在第1章中，我们将下载并安装Groovy，通过试用groovysh和groovyConsole来快速熟悉Groovy。我们还会看到Groovy的其他运行方式，包括从命令行运行和在IDE内运行。

在第2章中，我们将从熟悉的Java代码入手，然后将其重构为Groovy风格的代码。在快速浏览过可以改进日常Java编码的Groovy特性之后，我们再来谈一下Groovy对Java 5特性的支持。Groovy遵循Java语义，仅在少数情况下例外，我们将会探讨这些陷阱，以防遭遇时措手不及。

在第3章中，我们将看到Groovy的类型与Java的类型的异同，Groovy如何处理我们提供的类型信息，以及何时可以利用动态类型或可选类型。本章会介绍如何利用Groovy的动态类型、能力式设计（Design by Capability）以及多方法（multimethods）。对于可以应用动态类型但性能要求无法满足的任务，我们会看到如何通知Groovy对部分代码进行静态类型化处理。

在第4章中，我们将学到关于闭包这一激动人心的Groovy特性的一切，包括闭包是什么、如何工作、何时使用，以及如何使用等。Groovy闭包比简单的Lambda表达式更强大；它还使尾递归优化和记忆化（Memoization）更方便使用了。

在第5章中，我们将探讨Groovy的字符串、多行字符串的运用，以及Groovy对正则表达式的支持。

在第6章中，我们将探索Groovy对Java集合类——List和Map的支持，包括集合类上的各种便捷方法。看完这些之后，我们就再也不想以原来的方式使用集合类了。

Groovy包含并扩展了JDK。在第7章中，我们将探索GDK及其特性，同时看看Groovy对Object类及其他Java类的扩展。

Groovy对处理XML——包括解析和创建XML文档——也提供了良好的支持，在第8章中我们会看到相关介绍。

第9章介绍了Groovy对SQL的支持，这些支持会让数据库相关的编程变得轻松有趣。这一章将介绍迭代器、数据集，以及如何使用更简单的语法和闭包执行常规的数据库操作，甚至如何从Microsoft Excel文档中获取数据。

与Java的集成是Groovy的主要优势之一。在第10章中，我们将研究在Groovy和Java代码内与多个Groovy脚本、Groovy类和Java类紧密交互的不同方式。

一般而言，元编程是动态语言的最大优势之一，在Groovy中该优势尤为突出；借助该特性，我们可以在运行时检查类，以及动态分派方法调用。在第11章中，我们将从Groovy如何处理在Groovy对象和Java对象上的方法调用这些基础入手，探索Groovy对元编程的支持。

利用Groovy，可以使用**GroovyInterceptable**和**ExpandoMetaClass**类执行类AOP的方法拦截，第12章会予以介绍。

在第13章中，我们将深挖Groovy的元编程能力，学习如何在运行时注入方法。

在第14章中，我们将学习如何在运行时合成或生成动态方法。

第15章介绍了如何动态合成类、如何使用元编程委托方法调用，以及如何在前面3章介绍的元编程技术中做出选择。

Groovy的优点不止于运行时元编程，利用抽象语法树（Abstract Syntax Tree，AST）变换技术，Groovy在静态编译时也具有同样的优势，在第16章中我们会看到。

Groovy生成器是专门辅助为嵌套层次结构创建灵活接口的类。在第17章中，我们将探讨如何使用生成器，以及如何创建自己的生成器。

在Groovy中，单元测试并非奢侈品，也不是“有时间才做”的练习。Groovy的动态特性需要单元测试。幸运的是，Groovy让编写测试和创建模拟对象变得很容易，第18章中将详细介绍。我们还会尝试有助于我们使用Groovy对Java代码和Groovy代码进行单元测试的技巧。

利用第19章中介绍的技术，我们可以应用Groovy的元编程能力来构建内部的DSL。我们将从DSL的基础（包括其特点）入手，然后快速转入在Groovy中构建DSL的学习。

最后，在附录中，列出了本书所引用的Web文章和书籍。

Groovy 的变化

现今Groovy又有了长足的进步，本书将介绍的是最新的Groovy 2.1。以下是本书中可能对你有所帮助的更新。

- 你将学到 Groovy 2.x 的特性。
- 你将学到 @Delegate、@Immutable 等 Groovy 代码的生成变换。
- 你将学到新的 Groovy 2.x 静态类型检查和静态编译工具的优点。
- 你将学到利用 Groovy 2.x 中对扩展模块的新支持来创建自己的扩展方法的一些技巧。
- Groovy 中的闭包非常优秀，你将学到它们对尾递归优化和记忆化的新支持。
- 你将学到如何有效地集成 Java 和 Groovy，如何从 Java 中传递 Groovy 闭包，甚至从 Java

中调用动态的 Groovy 方法。

- 你会看到为学习元编程 API 的增强而引入的例子。
- 你将学到如何使用 **Mixin**, 以及如何使用它们实现一些优雅的模式。
- 除了运行时元编程, 你还可以掌握编译时元编程和抽象语法树 (AST) 变换。
- 你将看到构建与读取 JSON 数据的具体细节。
- 再有, 你将学到有助于流畅地创建 DSL 的 Groovy 语法。

目标读者

本书是为在Java平台上工作的开发者编写的。它最适合那些对Java语言有较深了解的开发人员和测试人员。懂得使用其他语言编程的开发者也可以使用本书, 但是他们需要利用有助于深入理解Java和JDK的书籍来补补课, *Effective Java*[Blo08]和*Thinking in Java*[Eck06]就是非常不错的学习资源。

对Groovy有所了解的程序员可以使用本书来学习一些在其他地方无从得见的诀窍和技巧。另外, 已经非常熟悉Groovy的程序员可能会发现本书能够用于培训和辅导组织中的开发者同仁。

在线资源

本书引用的网络资源都汇集在了附录A中。下面两个资源可以帮助你入门。

- Groovy 网站, 可以下载本书使用的 Groovy 版本: <http://groovy.codehaus.org>。
- 本书在 Pragmatic Bookshelf 网站的官方主页: <http://www.pragprog.com/titles/vslg2>。你可以从这里下载所有的示例源代码文件, 也可以通过在本书的论坛中提交勘误或发表意见来反馈问题。

如果你正在阅读的是本书的电子版, 可以点击代码清单上的链接来查看或下载具体的例子。

致谢

过去的四年中, 看着Groovy生态系统成长起来真是非常快乐。感谢Groovy团队开发了这门帮助程序员提高开发效率的同时还能享受快乐的语言及其相关工具。

我要感谢本书第1版的所有读者。特别感谢Norbert Beckers、Giacomo Cosenza、Jeremy Flowers、Ioan Le Gué、Fred Janon、Christopher M. Judd、Will Krespan、Jorge Lee、Rick Manocchi、Andy O'Brien、Tim Orr、Enio Pereira、David Potts、Srivaths Sankaran、Justin Spradlin、Fabian Topfstedt、Bryan Young和Steve Zhang, 感谢他们花时间在本书勘误页面上报告错误。

衷心感谢和感激本书的技术审校人员。他们非常友好地奉献出自己的时间和精力, 通读了书

中的概念，试验了书中的例子，并向我提供了非常有价值的反馈、修正和鼓励。感谢你们，Tim Berglund、Mike Brady、Hamlet D’arcy、Scott Davis、Jeff Holland、Michael Kimsal、Scott Leberknight、Joe McTee、Al Scherer和Eitan Suez。

还要感谢Guillaume Laforge的鼓励，感谢他拨冗撰写序言。Cédric Champeau和Chris Reigrut慷慨地快速通读了本书的beta版本，并且提供了有价值的反馈。非常感谢你们，谢谢。同时感谢Thilo Maier在本书的勘误页面上报告了一些错误。

特别感谢本书的编辑Brian Hogan，感谢他的复核、点评、建议和鼓励。在这一版的创作过程中，他提供了必要的指导。

感谢整个Pragmatic Programmers团队，感谢他们开发Groovy的这个版本，还要感谢他们在出版过程中提供的支持。

目 录

第一部分 Groovy 起步

第1章 起步	2
1.1 安装 Groovy	2
1.1.1 在 Windows 系统环境安装 Groovy	2
1.1.2 在类 Unix 系统环境安装 Groovy	3
1.2 管理多个版本的 Groovy	3
1.3 使用 groovysh	4
1.4 使用 groovyConsole	5
1.5 在命令行中运行 Groovy	5
1.6 使用 IDE	6
1.6.1 IntelliJ IDEA	6
1.6.2 Eclipse Groovy 插件	6
1.6.3 TextMate Groovy Bundle	6
第2章 面向 Java 开发者的 Groovy	8
2.1 从 Java 到 Groovy	8
2.1.1 Hello, Groovy	8
2.1.2 实现循环的方式	9
2.1.3 GDK 一瞥	11
2.1.4 安全导航操作符	13
2.1.5 异常处理	13
2.1.6 Groovy 是轻量级的 Java	15
2.2 JavaBean	15
2.3 灵活初始化与具名参数	19
2.4 可选形参	20
2.5 使用多赋值	21
2.6 实现接口	22
2.7 布尔求值	25
2.8 操作符重载	27

2.9 对 Java 5 特性的支持	28
2.9.1 自动装箱	29
2.9.2 for-each	29
2.9.3 enum	30
2.9.4 变长参数	31
2.9.5 注解	32
2.9.6 静态导入	33
2.9.7 泛型	33
2.10 使用 Groovy 代码生成变换	35
2.10.1 使用@Canonical	35
2.10.2 使用@Delegate	35
2.10.3 使用@Immutable	36
2.10.4 使用@Lazy	37
2.10.5 使用@Newify	38
2.10.6 使用@Singleton	38
2.11 陷阱	40
2.11.1 Groovy 的==等价于 Java 的 equals()	40
2.11.2 编译时类型检查默认为关闭	42
2.11.3 小心新的关键字	43
2.11.4 别用这样的代码块	43
2.11.5 闭包与匿名内部类的冲突	43
2.11.6 分号总是可选的	45
2.11.7 创建基本类型数组的不同语法	45
第3章 动态类型	47
3.1 Java 中的类型	47
3.2 动态类型	48
3.3 动态类型不等于弱类型	49
3.4 能力式设计	50
3.4.1 使用静态类型	50

3.4.2 使用动态类型	51	6.7 Map 上的其他便捷方法	110
3.4.3 使用动态类型需要自律	53		
3.5 可选类型	54	第二部分 使用 Groovy	
3.6 多方法	55	第 7 章 探索 GDK 114	
3.7 动态还是非动态	58	7.1 使用 Object 类的扩展	114
3.8 关闭动态类型	58	7.1.1 使用 dump 和 inspect 方法	115
3.8.1 静态类型检查	59	7.1.2 使用上下文 with() 方法	115
3.8.2 静态编译	62	7.1.3 使用 sleep	116
第 4 章 使用闭包	64	7.1.4 间接访问属性	118
4.1 闭包的便利性	64	7.1.5 间接调用方法	119
4.1.1 传统方式	64	7.2 其他扩展	119
4.1.2 Groovy 方式	65	7.2.1 数组的扩展	120
4.2 闭包的应用	67	7.2.2 使用 java.lang 的扩展	120
4.3 闭包的使用方式	68	7.2.3 使用 java.io 的扩展	122
4.4 向闭包传递参数	69	7.2.4 使用 java.util 的扩展	124
4.5 使用闭包进行资源清理	70	7.3 使用扩展模块定制方法	125
4.6 闭包与协程	72		
4.7 科里化闭包	74		
4.8 动态闭包	75		
4.9 闭包委托	77		
4.10 使用尾递归编写程序	80		
4.11 使用记忆化改进性能	82		
第 5 章 使用字符串	87		
5.1 字面常量与表达式	87		
5.2 GString 的惰性求值问题	90		
5.3 多行字符串	93		
5.4 字符串便捷方法	95		
5.5 正则表达式	96		
第 6 章 使用集合类	98		
6.1 使用 List	98		
6.2 迭代 ArrayList	100		
6.2.1 使用 List 的 each 方法	100		
6.2.2 使用 List 的 collect 方法	102		
6.3 使用查找方法	102		
6.4 List 上的其他便捷方法	103		
6.5 使用 Map 类	106		
6.6 在 Map 上迭代	108		
6.6.1 Map 的 each 方法	108		
6.6.2 Map 的 collect 方法	109		
6.6.3 Map 的 find 和 findAll 方法	109		
第 7 章 探索 GDK 114			
7.1 使用 Object 类的扩展	114		
7.1.1 使用 dump 和 inspect 方法	115		
7.1.2 使用上下文 with() 方法	115		
7.1.3 使用 sleep	116		
7.1.4 间接访问属性	118		
7.1.5 间接调用方法	119		
7.2 其他扩展	119		
7.2.1 数组的扩展	120		
7.2.2 使用 java.lang 的扩展	120		
7.2.3 使用 java.io 的扩展	122		
7.2.4 使用 java.util 的扩展	124		
7.3 使用扩展模块定制方法	125		
第 8 章 处理 XML 128			
8.1 解析 XML	128		
8.1.1 使用 DOMCategory	129		
8.1.2 使用 XMLParser	131		
8.1.3 使用 XMLSlurper	131		
8.2 创建 XML	133		
第 9 章 使用数据库 136			
9.1 创建数据库	136		
9.2 连接到数据库	137		
9.3 数据库的 Select 操作	137		
9.4 将数据转为 XML 表示	139		
9.5 使用 DataSet	140		
9.6 插入与更新	140		
9.7 访问 Microsoft Excel	141		
第 10 章 使用脚本和类 143			
10.1 Java 和 Groovy 的混合	143		
10.2 运行 Groovy 代码	144		
10.3 在 Groovy 中使用 Groovy 类	145		
10.4 利用联合编译混合使用 Groovy 和 Java	145		
10.5 在 Java 中创建与传递 Groovy 闭包	146		
10.6 在 Java 中调用 Groovy 动态方法	148		
10.7 在 Groovy 中使用 Java 类	150		

10.8 从 Groovy 中使用 Groovy 脚本	151
10.9 从 Java 中使用 Groovy 脚本	153
第三部分 MOP 与元编程	
第 11 章 探索元对象协议	158
11.1 Groovy 对象	159
11.2 查询方法与属性	162
11.3 动态访问对象	164
第 12 章 使用 MOP 拦截方法	166
12.1 使用 GroovyInterceptable 拦截方法	166
12.2 使用 MetaClass 拦截方法	168
第 13 章 MOP 方法注入	173
13.1 使用分类注入方法	173
13.2 使用 ExpandoMetaClass 注入方法	178
13.3 向具体的实例中注入方法	182
13.4 使用 Mixin 注入方法	184
13.5 在类中使用多个 Mixin	187
第 14 章 MOP 方法合成	192
14.1 使用 methodMissing 合成方法	192
14.2 使用 ExpandoMetaClass 合成方法	196
14.3 为具体的实例合成方法	199
第 15 章 MOP 技术汇总	201
15.1 使用 Expando 创建动态类	201
15.2 方法委托：汇总练习	203
15.3 MOP 技术回顾	207
15.3.1 用于方法拦截的选项	207
15.3.2 用于方法注入的选项	207
15.3.3 用于方法合成的选项	208
第 16 章 应用编译时元编程	209
16.1 在编译时分析代码	209
16.1.1 理解代码结构	210
16.1.2 在代码结构中导航	211
16.2 使用 AST 变换拦截方法调用	214
16.3 使用 AST 变换注入方法	218

第四部分 使用元编程

第 17 章 Groovy 生成器	224
17.1 构建 XML	224
17.2 构建 JSON	227
17.3 构建 Swing 应用	229
17.4 使用元编程定制生成器	230
17.5 使用 BuilderSupport	233
17.6 使用 FactoryBuilderSupport	236
第 18 章 单元测试与模拟	240
18.1 本书代码与自动化单元测试	240
18.2 对 Java 和 Groovy 代码执行单元测试	241
18.3 测试异常	245
18.4 模拟	245
18.5 使用覆盖实现模拟	247
18.6 使用分类实现模拟	250
18.7 使用 ExpandoMetaClass 实现模拟	251
18.8 使用 Expando 实现模拟	253
18.9 使用 Map 实现模拟	255
18.10 使用 Groovy Mock Library 实现模拟	255
18.10.1 使用 StubFor	256
18.10.2 使用 MockFor	257
第 19 章 在 Groovy 中创建 DSL	261
19.1 上下文	261
19.2 流畅	262
19.3 DSL 的分类	263
19.4 设计内部的 DSL	264
19.5 Groovy 与 DSL	264
19.6 使用命令链接特性改进流畅性	265
19.7 闭包与 DSL	266
19.8 方法拦截与 DSL	267
19.9 括号的限制与变通方案	268
19.10 分类与 DSL	270
19.11 ExpandoMetaClass 与 DSL	271
附录 A Web 资源	273
附录 B 参考书目	277