

Beyond Java



超越 Java™

探讨程序语言的未来



Bruce A. Tate 著

O'Reilly Taiwan公司 编译

O'REILLY®

東南大學出版社

超越 Java



Java 的存在时间已经超过 10 年了，在这段时间里它的成就非凡，彻底改变了我们写软件的方法和想法。但是 Java 已显老态，该是时候让我们想想未来接班人是谁了。

在《超越 Java》一书中，Bruce Tate 公正而诚实地分析 Java 的成就，以及为何 Java 是如此强大的工具。他告诉我们 Java 如何带领计算机编程领域向前迈进，同时也讲述在哪些方面 Java 会阻碍我们，在哪些方面 Java 对我们要做的事而言并不够灵活，在哪些地方因为政治力量而使用 Java 只会导致复杂度的提升。

作者研究了其他的语言和框架，提出了一些很难的问题：这些语言擅长什么？不擅长什么？其中有没有 Java 的接班人？它们能够像 Java 一样主导软件领域吗？未来的工具应该是什么样子？会像 J2EE 或者 Ruby on Rails？说不定更激进，就像 continuation server？Bruce Tate 除了与我们分享他自己的观察结果，也访问并概括了许多开放源代码 Java 和 Ruby 社区的领导人的看法。

不管你是否同意 Tate 的结论，你都会发现这本书相当激励人心。不管你继续使用 Java 还是改用其他技术，这本书都可以帮助你思考写程序的方式、怎样提高生产力、什么东西会阻挠你。这本书可以帮助你开始认清 Java 以外的世界，接班人就要出现了。

Bruce Tate 是 Jolt Award 得奖书籍《轻快的 Java》（《Better, Faster, Lighter Java》）的作者。

“就像 Java 在 10 年前那样，接班人已经蓄势待发。这本书可以带你透视未来。”

——Dave Thomas, The Pragmatic Programmers LLC

“Bruce Tate 说出许多人的想法：Java 之后，还有别的技术会兴起。本书将许多技术拿来聚焦评论，分析各个的优缺点。这本书打开大门，让我们开始讨论 Java 过后还会有什么新技术会出现。”

——David Heinemeier Hansson, Ruby on Rails 的发明者

ISBN 978-7-5641-0639-3



9 787564 106393 >

定价：29.00 元

O'Reilly Media, Inc. 授权东南大学出版社出版

超越 JavaTM

Bruce A. Tate 著
O'Reilly Taiwan 公司 编译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目 (CIP) 数据

超越 Java™/ (美) 泰特 (Tate, B. A.) 著; O'Reilly Taiwan 公司编译. —南京: 东南大学出版社, 2007.1

书名原文: Beyond Java™

ISBN 978-7-5641-0639-3

I. 超... II. ①泰... ②O... III. Java 语言—程序设计
IV. TP312

中国版本图书馆 CIP 数据核字 (2006) 第 155200 号

江苏省版权局著作权合同登记

图字: 10-2006-218 号

©2005 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2006. Authorized translation of the English edition, 2005 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2005。

简体中文版由东南大学出版社出版 2006。英文原版的翻译得到 O'Reilly Media, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有, 未得书面许可, 本书的任何部分和全部不得以任何形式重制。

书 名 / 超越 Java™

责任编辑 / 张烨

封面设计 / Ellie Volckhausen, 张健

出版发行 / 东南大学出版社

地 址 / 南京四牌楼 2 号 (邮编 210096)

印 刷 / 扬中市印刷有限公司

开 本 / 787 毫米 × 980 毫米 16 开本 13.25 印张 223 千字

版 次 / 2007 年 1 月第 1 版 2007 年 1 月第 1 次印刷

印 数 / 0001-4000 册

书 号 / ISBN 978-7-5641-0639-3/TP · 104

定 价 / 29.00 元 (册)



O'Reilly Media, Inc. 介绍

为了满足读者对网络 and 软件技术知识的迫切需求,世界著名计算机图书出版机构 O'Reilly Media, Inc. 授权东南大学出版社, 翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司, 同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》(被纽约公共图书馆评为二十世纪最重要的 50 本书之一) 到 GNN (最早的 Internet 门户和商业网站), 再到 WebSite (第一个桌面 PC 的 Web 服务器软件), O'Reilly Media, Inc. 一直处于 Internet 发展的最前沿。

许多书店的反馈表明, O'Reilly Media, Inc. 是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比, O'Reilly Media, Inc. 具有深厚的计算机专业背景, 这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员, 或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家, 而现在编写著作, O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着, 所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。



作者简介

Bruce A. Tate 是一位泛舟者、越野车骑士、父亲、作家及 Java 程序员，住在德克萨斯州的奥斯汀。他写了 5 本书，包括获得 Jolt 奖的《轻快的 Java》(O'Reilly 出版) 以及热卖的《Bitter Java》(Manning 出版)。他有 17 年的工作经验，包括在 IBM 工作、两次失败的创业以及开办了自己的叫做 J2Life, LLC 的独立顾问公司。

封面介绍

《超越 Java》的封面动物是一只猫鼬 (bassaris)。猫鼬是北美肉食动物，出现在墨西哥、德克萨斯州、加州。体积为家猫大小，外形接近浣熊和狐狸。

这种棕褐毛的动物有着黑白相间的尾巴，而尾巴和身体一样长。尾巴的长度可以让它在通过窄岩壁和大树枝时保持平衡，甚至可以横翻筋斗转 180 度。它的后脚可以转动 180 度，使它相当擅长在峭壁和树木中行走。

猫鼬是夜行动物，不具有侵略性。它居住在洞穴、岩缝以及树洞中，也曾废弃的房子内、有人居住的房子的阁楼中被发现过。它会去露营地乱翻东西，有时候还会把东西带走，特别是亮晶晶的东西。它是敏捷的攀爬者，可以轻易地穿越树木和岩石。

猫鼬猎食的时间主要是晚上，对象主要是小型鸟类、啮齿类、蜥蜴、蛇、无脊椎动物以及水果，偶尔也食腐尸。水果是主食，这可以减少对水分的摄取。

某些地区会为了皮毛而猎捕猫鼬。它也常被驯为宠物，特别是在墨西哥的某些地区。它有一些不同的俗名，包括山猫、civit cat，墨西哥称其为 cacomixl。它的学名 bassaris 起源于希腊的狐狸。在希腊神话中，Dionysus (酒神) 穿着一件猫鼬，象征着新生命。

目录

前言	1
第一章 猫头鹰与鸵鸟	7
忽略是一种美德	8
水煮青蛙	10
新的水平线	16
预告	19
第二章 完美的风暴	20
风暴警告	20
C++ 的经验	24
拨云见日	29
暴怒的释放	32
事过境迁	38
向前走	38
第三章 皇冠上的宝石	40
语言和 JVM 的设计	42
因特网	46
企业集成	49

社区	51
打破迷思	53
第四章 打破玻璃	57
Java 的新工作描述	58
基本的 Java 限制	62
类型	64
基本类型	73
分手时的刻薄话	75
为什么不直接修改 Java?	78
第五章 游戏规则	80
Java 提高竞争门槛	81
企业集成	85
产生话题	87
语言特色	92
几个有潜力的语言	97
第六章 Ruby 简介	102
关于 Ruby	103
应用某些结构	111
Breaking It Down	125
第七章 Ruby on Rails	127
数字游戏	127
Rails 范例	133
查看内部	142
精华	145
第八章 延续服务器	149
问题	149
延续	152

延续服务器	156
Seaside	161
一个 Seaside 范例	164
那又怎样?	168
第九章 竞争者	170
主要竞争者	171
较小的竞争者	182
“下一个大东西”	188
索引	191



光盘索引

实例名称：计算器

路 径：光盘\赠送代码\计算器

视频时间：3 分 17 秒

Windows 中自带了一个计算器，在此以开发一个计算器为例，向读者介绍了如何编写一个窗口程序，如何分布对象，并使用方法，让它能够运算，给出准确的值。

实例名称：吃豆子游戏

路 径：光盘\赠送代码\吃豆子游戏

视频时间：3 分 31 秒

介绍了小游戏的基本结构和功能模块，并详细介绍了小游戏的制作流程，在最后还是向读者介绍了如何美化优秀界面。

实例名称：俄罗斯方块

路 径：光盘\赠送代码\俄罗斯方块

视频时间：3 分 31 秒

通过一个简单的俄罗斯方块游戏实例来说明 Java 语言编写游戏项目的基本方法和技巧，可以拓展读者的开发思路，提高开发技巧。

实例名称：超市管理

路 径：光盘\赠送代码\超市管理

视频时间：4 分 30 秒

介绍了一个基于窗口与数据库信息管理的系统，在这个系统中，介绍了如何加载驱动、如何与数据库连接和如何操作数据等一系列开发流程，并详细批注了各个代码。

前言

2005年3月，我惊喜地收到通知，我写的一本《Better, Faster, Lighter Java》获得 Jolt Award 震撼奖。我在那本书中告诉 Java 开发者如何建立更好、比以往更快的程序。那本书在我心中的特殊地位将永远不会被取代。但是，这一路下来，有些新东西出现了，我必须动手去接触、去了解。

在这期间，我的一个顾客正在建立一个应用程序，包含复杂数据库模式(schema)以及基于Web的用户界面。我们用了Spring和Hibernate搭配Web Work，对于轻量级的Java开发来说，这是很常见的组合，我们一般也算满意。但是有些事情困扰着我们：重复的事情太多、XML配置(configuration)激增、改变的步调太快。一时兴起，我们试了Ruby on Rails，这是一个高生产力、创新的框架，它正在席卷非Java的社区，而且也为Java的架构带来一些争论。使用Rails所获得的高生产力让我们感到震惊，所以后来我们把整个计划都搬过去了。

有些东西和我一拍即合。对于这种应用程序来说，Java是个碍手碍脚的东西。将Java排除，我可以把程序代码减少到原来的1/4，把XML减少到只剩十分之一，达到令人惊讶的生产力以及不错的性能。更棒的是，《Better, Faster, Lighter Java》(简称BFLJ)一书中所提到的概念依然可以在此采用。对于其他项目，如果需要Java所提供的社区和工具，我会用它来取代；如果不需要Java，我可以把BFLJ的原则发挥到极致。我内心的水坝已经溃堤，思潮通过这本书倾泻而出，我要告诉大家我的想法。

几个月后，我找到了听众，在数千里外，离我家有19个小时的路程。在Java用户社区的面前，我感到局促不安，我在更大的场面中演讲过，但是这趟旅程却很不一样。这次挪威Java用户社区帮我支付奥斯陆（挪威首都）之行的花费，他们却无法在会场中听到我歌颂Spring、赞扬Hibernate、褒奖敏捷开发（Agile Development），而是会听到我说出“他们挚爱的宝贝很丑”，我是否要实话实说，我陷入了困境。Java为我带来许多好处：写了畅销书、得到Jolt奖、在不景气的经济形势下还能有相当不错的咨询业务，我当然希望Java列车继续开下去、不要停歇、永远根基稳固，我希望Java让我的生产力直上云霄，希望无尽的群众和脑力能解决Java今日所面对的所有棘手的问题，但是世间没有任何东西可以永恒维持。

在我的演说中，我先陈述Java成功的理由，然后论及它的严重问题，展示了一些可供选择的语言和框架。在整个演说中，我指出时机已经成熟，该是另一个技术出现的时候了。当我对这群好客的人演说时，我回答问题，也观察人的表情，一些人看起来具有敌意，有的人则是很受伤，但是大部分的人则是很能理解，或者有一些恐惧。他们了解我的中心想法。对于我们惯用Java来做的事，有些其他语言的框架可以做得更好。在某些情况下，生产力的差异实在太太，值得我们好好地研究新的技术。

这场演说以及问答超时很长时间，但是没有人离席，令我惊讶的是，他们都能接受。演说完毕后，我们出去参观奥斯陆，一个具有敌意的听众几乎整晚都围绕着我，抛出一个又一个困难的问题：

- 为何我们不能改变Java以修正这些缺失？
- 你所展示的其他语言和框架有没有足够的商业支持？
- 它们支持分布式事务、Web Service、XML吗？
- 怎样找到有这种技术的程序员？又该如何培训自己的程序员？

这些问题都是真正需要注意的问题，这些问题也显示出进入新语言时必须跨越的重重障碍。问问题的人是位绅士，但是他无法完全隐藏自己烦躁不安的情绪以及根深蒂固的思想，他认为进入下一个成功的语言需要付出相当高的代价，并认为在可见的将来我们还是会继续使用Java写程序。他可能没错，但是我已经开始发现Java语言的局限性和许多框架的缺点。在某些问题的解决上，Java已经不再具有高生产力了，我改用一些其他的技术，且获得了成

功。虽然语言可以存活大半世纪以支持旧系统，但是我知道没有语言可以永远保持领先地位，Java 统治的时代将会结束，这不是“会不会”的问题，而是“什么时候”的问题。

谁需要读这本书？

当 C++ 褪色不再让人注意，我的许多好友都伤得很重。他们没有注意到，空气中已经有了不同的气息，巨大的改变瞬间来到。虽然我写这本书会失去许多，但是我写这本书的目的是希望这样的事情不要再发生。如果你不希望在一觉醒来之后忽然发现自己已经被淘汰了，那么你需要读这本书。

如果你认同我的说法，你可以开始试着建立你自己的技能。你可以下载一些我讨论到的框架，学习一些语言，这本书会告诉你成功的语言所具备的要点。如果我很幸运，找到了些未来可能的赢家，那么以后改朝换代时你就不会措手不及了。

如果你不认同我的说法，你可以用最好的语言、最好的框架、最好的技术来改进你今日用 Java 在做的事。新的框架像 PHP、C Omega for .NET、Ruby on Rails 将会偶尔出现在你的周围，你需要知道它们，了解如何评估它们。

所以，不管你是否认同我的说法，你都是赢家。该是注意的时候了，该是看着水平线的时候了，要看得比 Java 更远，超越 Java。

排版约定

由于本书保留了部分术语的原貌，为了避免读者的混淆，我们运用了一些字体变化，让读者能轻易辨别词汇本身的含义。

等宽字 (*Constant width*)

用于范例代码、类名、方法名及对象。

等宽斜体字 (*Constant width italic*)

用于指示程序中应以实际值替换的项目。

斜体字 (*Italic*)

用于文件名、目录、强调重点及第一次使用到的术语。

等宽黑体字 (**Constant width bold**)

用于文本中的用户输入以及范例中显示的输入及输出。也用于在代码中强调重点以及指示有注释的文本块。

某些有缩写的术语在后文再次出现时，会视情况以缩写出现，不一定会使用较长的中文译词。此外，若是在内文中列举多个项目时，为了醒目起见，会以粗体字来表示。

使用程序代码范例

帮助你完成工作是本书的目的。一般来说，你可以在你自己的程序和文档中使用本书的程序代码，你不需要先联系 O'Reilly 公司，除非是明显进行重制的行为。例如，写程序的时候使用到本书的片段并不需要我们的许可，销售本书的范例光盘就需要得到我们的许可。回答问题时引用书中的范例也不需要我们的许可，但是在你的文档中大量使用本书的程序代码就需要我们的许可。

在线资源

与本书相关的信息，包括范例程序代码、相关资源链接、勘误表，都可在本书的中文网页取得：

<http://www.oreilly.com.cn/book.php?bn=978-7-5641-0639-3>

批评和建议

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者对将来的版本有什么建议，请通过下面的地址告诉我们：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

询问技术问题或对本书进行评论，请发电子邮件到：

info@mail.oreilly.com.cn

最后，您可以在以下站点找到我们：

http://www.oreilly.com

http://www.oreilly.com.cn

猫头鹰与鸵鸟

我所认识的某些泛舟者都有死前的愿望，他们奋不顾身地投入 Class V（最高级）的泛舟活动。他们被水中的障碍物弄得翻船似乎只是早晚的事，障碍物会把船困住，而河水的力量会把船牢牢吸在水中。他们就像鸵鸟，把头埋进沙子中就以为危险不见了。

还有另一类泛舟者。当我开始泛舟时，我会侦察周围一切。我会待在 Class II+（初学者）的小波浪中仔细观察，并在开始前设置好安全绳索。我在河中的时间经常会不够，被迫跳进后面的区域，以在落日完成这一切。现在，我很少离开船去侦察小急流。在某些地方，这并不实际，于是我改用“追击”的泛舟技巧（这是在美国东南部陡窄河流中发明出来的技巧），来增加我的机会。我不会用这种方式泛舟，因为我喜欢危险，事实上，我的直觉很准，可以感受到危险最可能出现的地方。我用这种方式泛舟，因为这让我把焦点集中在最需要的侦察上。这类泛舟者是猫头鹰。

结论是这样的：我常常忽略后果不严重的危险，或者发生频率很低的危险，因为把注意力放在这方面并不明智。适当地管理风险可能需要花相当多的精力、金钱、时间，且让我面临额外的风险。让我们回到猫头鹰和鸵鸟的问题上，正常情况下，这两者有很大的差别，但是偶尔猫头鹰会过度自信，或者对风险评估有了小错误，且说服他们自己没有先侦察就进行危险的事，这样的情况就曾经发生在我身上。我已经在这条小溪泛舟几百次了，但有一些状况会改变，像下雨后河面涨高。猫头鹰和鸵鸟只有一线之隔，有时候，难以分辨究竟是鸵鸟还是猫头鹰。身为一个泛舟者，即使我决定忽略某条河在某

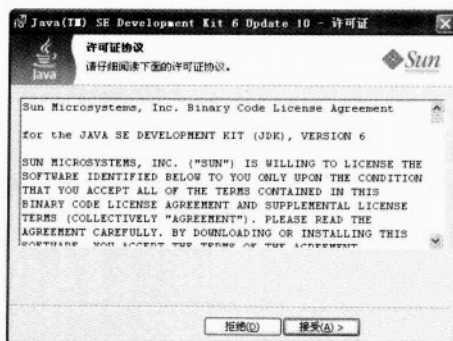


图 1-2 “许可证协议”对话框

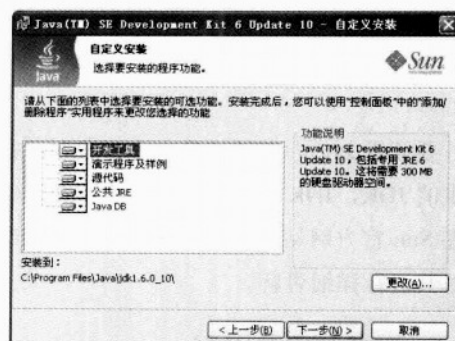


图 1-3 “自定义安装”对话框

(4) 打开“正在安装”窗口，程序将会自动安装，如图 1-4 所示，安装完成后，将会打开“目标文件夹”对话框，单击 **更改(A)...** 按钮，来改变 JRE 的安装路径，在这里选择默认设置，单击 **下一步(N) >** 按钮，如图 1-5 所示。

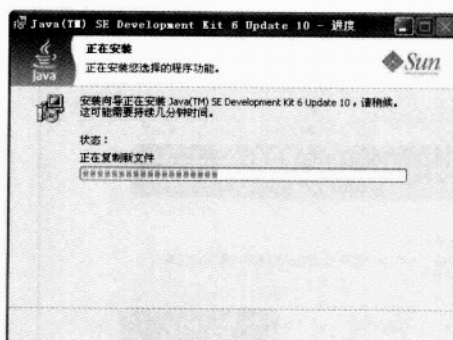


图 1-4 “正在安装”窗口

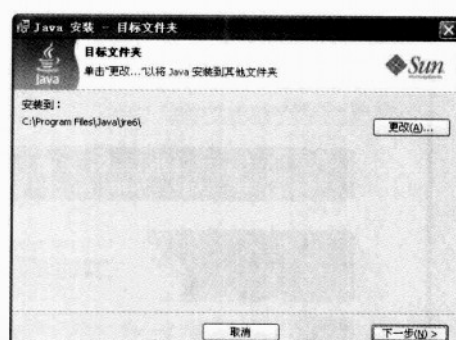


图 1-5 “目标文件夹”对话框

(5) 打开“正在安装 Java”窗口，显示着安装的进程，如图 1-6 所示，安装完成后，单击 **完成** 按钮将会打开如图 1-7 所示。



图 1-6 “正在安装 Java”窗口



图 1-7 “已成功安装”对话框

1.3.2 设置环境变量

在安装完成 JDK 后，并不能使用，因为还需要对其进行配置、设置环境变量，设置环境变量的目的是让系统自动寻找需要的命令，其具体操作步骤如下。

(1) 选择“我的电脑”图标，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，然后单击“高级”选项卡，在该选项卡中单击 **环境变量** 按钮，在“环境变量”对话框的“系统变量”栏中，单击 **新建** 按钮，打开“新建系统变量”对话框，在“变量名”文本框中输入“JAVA_HOME”，在“变量值”文本框中输入 JDK 的路径，如“C:\Program Files\Java\jdk1.6.0_10”，单击 **确定** 按钮，如图 1-8 所示。

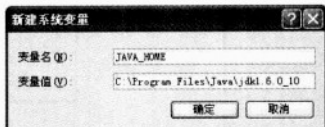


图 1-8 “新建系统变量”对话框

(2) 返回到“环境变量”对话框，在“系统变量”栏中，单击 **新建** 按钮，如图 1-9 所示。

(3) 再次打开“新建系统变量”对话，在“变量名”文本框中输入“PATH”，在“变量值”文本框中输入“%JAVA_HOME%\bin;%PATH%”，设置完成后，单击 **确定** 按钮，如图 1-10 所示。



图 1-9 “环境变量”对话框

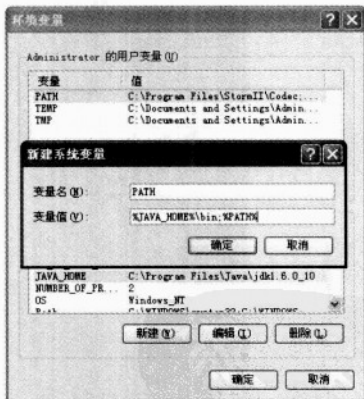


图 1-10 设置 PATH

(4) 用相同的方法，打开“新建系统变量”对话框，在“变量名”文本框中输入“CLASSPATH”，在“变量值”文本框中输入“.;%JAVA_HOME%\lib\tools.jar”，单击 **确定** 按钮，如图 1-11 所示。

(5) 在“环境变量”对话框中,单击 按钮,然后再在“系统属性”对话框中,单击 按钮,设置完成后,如图 1-12 所示。

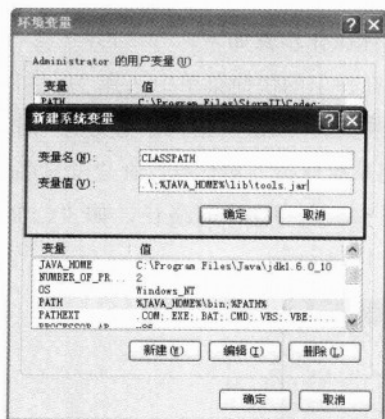


图 1-11 设置“CLASSPATH”

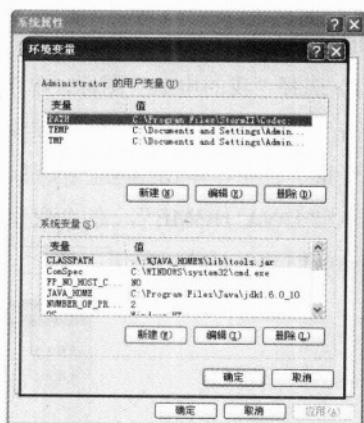


图 1-12 设置完成

(6) 选择“开始”→“运行”命令,打开“运行”对话框,在“打开”文本框中输入“cmd”,然后单击 按钮。

(7) 输入“javac”命令,打开如图 1-13 所示的信息。



图 1-13 输入“javac”命令

(8) 然后在窗口中输入“java”命令,如图 1-14 所示。



图 1-14 输入“java”命令

1.3.3 升级 JDK

当 JDK 安装一定的时间后, Sun 公司有可能对 JDK 进行更新, 在任务栏处出现一个咖啡的形状, 用户单击它, 可以进行升级 JDK, 其具体操作步骤如下。

(1) 打开“有可用的 Java 更新程序”窗口, 单击 **安装** 按钮, 如图 1-15 所示。

(2) 打开“欢迎使用 Java (TM) 和 JavaFX (TM)”对话框, 单击 **接受(A) >** 按钮, 如图 1-16 所示。

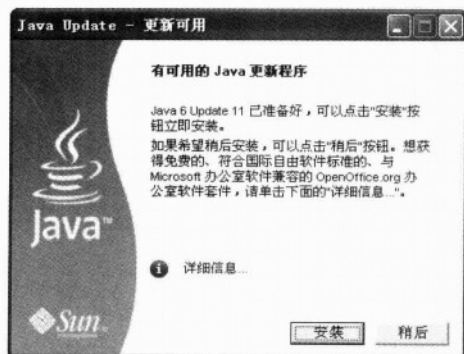


图 1-15 更新 JDK

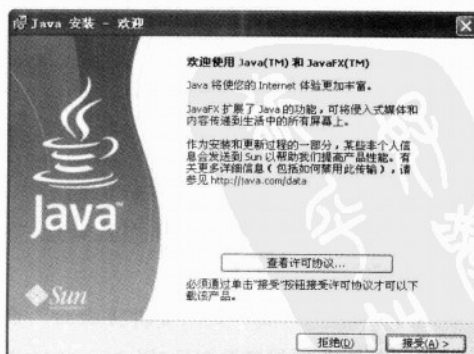


图 1-16 接受许可协议

(3) 打开“正在安装 Java”窗口, JDK 正在安装需要的重要的组件, 如图 1-17 所示。

(4) 打开“您已成功安装 Java (TM)”对话框, 单击 **完成** 按钮, 如图 1-18 所示。



图 1-17 “正在安装 Java”窗口

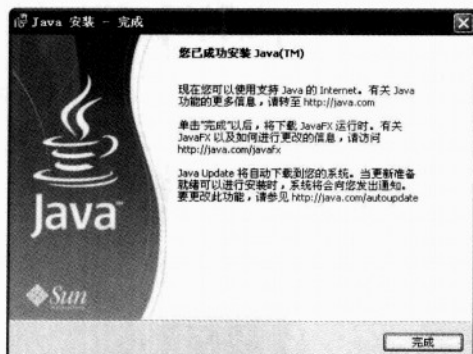


图 1-18 “您已成功安装 Java (TM)”对话框

1.4 第一个 Java 程序

通过第一个 Java 程序, 用户将会明白 Java 程序是如何执行的。Java 程序的结构, Java 程序的开发步骤, 下面将进行详细讲解。

1.4.1 开发 Java 程序的步骤

开发一个 Java 程序, 一般有以下几个步骤。

(1) 创建 Java 源程序: Java 源程序一般用 Java 作为扩展名, 是一个文本文件, 用 Java 语言写成的, 可以用任何文本编辑器创建与编辑。

(2) 编译源程序: Java 编译器, “Javac” 读取你的 Java 源程序并翻译成 Java 虚拟机能够明白的指令集合, 且以字节码的形式保存在文件中。通常, 字节码文件以 class 作为扩展名。

(3) 运行 class (字节码) 文件: Java 解释器读取字节码, 取出指令并且翻译成计算机能执行的代码, 完成运行过程。

1.4.2 编写 Java 程序

新建一个文件夹, 如在 D 盘里新建 “javaone”, 然后在文件夹里新建一个记事本, 其具体操作步骤如下。

(1) 选择“工具”→“文件夹命令”命令, 打开“文件夹选项”对话框, 单击“查看”选项卡, 在“高级设置”列表栏中取消勾选 ☒ 隐藏已知文件类型的扩展名 复选框, 单击 **确定** 按钮, 如图 1-19 所示。

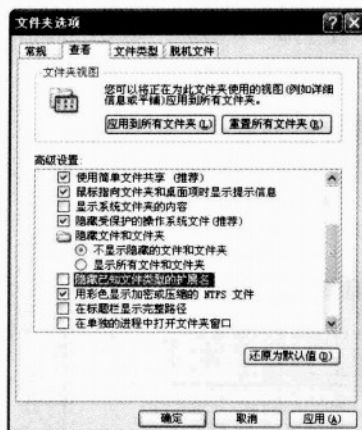


图 1-19 “文件夹选项”对话框

(2) 打开记事本，输入以下程序。

```
/**
 * 此类用于在屏幕上显示消息。
 *
 * @ version 1.0, 2009 年 4 月 20 日
 * @author Michael
 */
class Message{
    /**
     * 这是一个 main 方法
     */
    public static void main(String [] args){
        /* 输出此消息 */
        System.out.println("欢迎来到 Java 世界!");
    }
}
```

输入完成后，选择“文件”→“保存”命令，如图 1-20 所示。

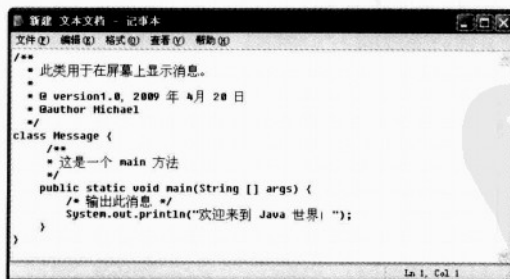



图 1-20 输入 Java 程序

(3) 将这个文件命名为“Message.java”程序，打开“重命名”对话框，单击  按钮，如图 1-21 所示。


(4) 选择“开始”→“运行”命令,打开“运行”对话框,在“打开”文本框中输入“cmd”,单击  按钮,如图 1-22 所示。



图 1-21 “重命名”对话框

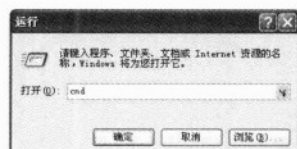


图 1-22 “运行”对话框

(5) 在“CMD”窗口中输入“D:”,然后再输入“cd javaone”打开文件夹,最后输入“javac message.java”,执行命令后,在文件夹中可以看到多了一个“Message.class”的文件,如图 1-23 所示。

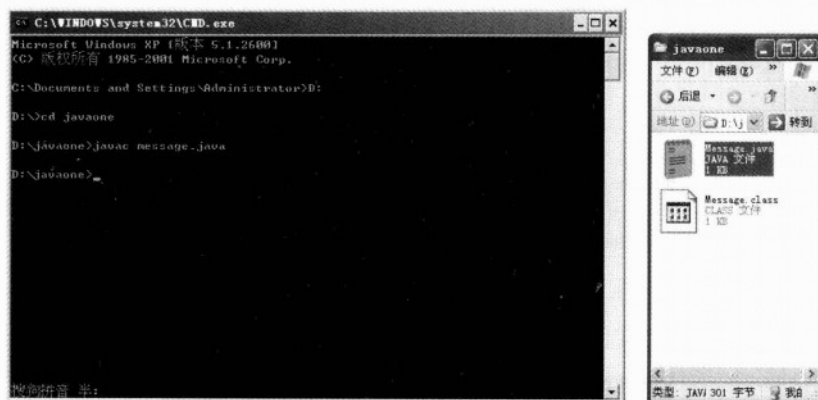


图 1-23 编译 Java 程序

(6) 在“CMD”窗口中输入“javac message java”,按回车键,可以看到运行结果,如图 1-24 所示。

这个程序只是输出一句话“欢迎进入 Java 世界!”,至于程序是什么意思,将在后面进行讲解,通过这个程序,用户将会明白 Java 程序的编译和运行的原理,如图 1-25 所示,用户也将明白 Java 是通过 Java 虚拟机进行跨操作系统平台的,用户只需一次编译,就可以多次运行。

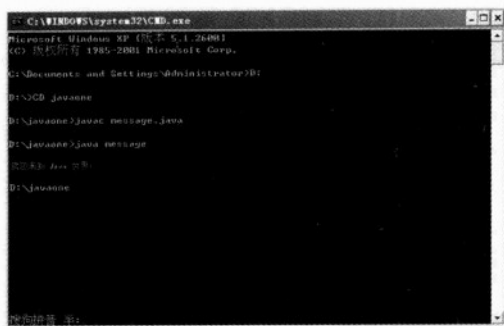


图 1-24 执行结果

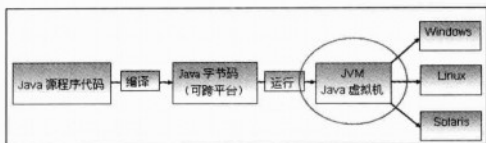


图 1-25 Java 执行的过程

TIPS

Java 源程序里面一般只有一个类 (class)，但允许有多个类 (class)，只能有一个带 “public” 修饰符的类。Java 源程序的文件名必须与 “public” 修饰符类名相同，编译后，将形成对应的 (class) 文件，这些内容将在后面进行详细讲解，这里只作了解。

1.5 本课回顾和网络关键词

在本课的内容中，对 Java 知识进行了介绍，讲解了 Java 语言的特点和 Java 语言的应用前景，然后讲解了 JDK 的下载与安装、升级与设置环境变量，最后通过一个简单的 Java 程序，在实践中给读者讲解了 Java 编译与运行的全过程，本节将对本课中的主要知识点进行简要回顾，并总结出本课知识点的网络关键词。

1. 本课回顾

在本课的内容中，主要讲解了 Java 的基本知识，回顾本课内容，主要的知识点概括如下。

(1) Java 概述。

Java 语言是美国 Sun Microsystems 公司 1995 年推出的面向对象的程序设计语言，该语言充分考虑到互联网时代的特点，非常适合互联网的开发，在设计上它具有跨平台、安全、面向对象等特点，因此深受广大程序爱好者的青睐，对象的主要知识点如下。

- Java 简介。
- Java 特点。
- Java 语言的应用前景。

(2) Java 开发环境。

需要开发 Java，必须要搭建一个开发和运行环境，需要安装 JDK，用户可以在 Sun 官方网站下载，然后将其安装，再进行配置，即可完成开发环境的配置，Java 的开发环境主

要知识点如下。

- 下载和安装 JDK。
- 设置环境变量。
- 升级 JDK。

(3) Java 编译和执行。

讲解 Java 程序的步骤和编写 Java 程序。

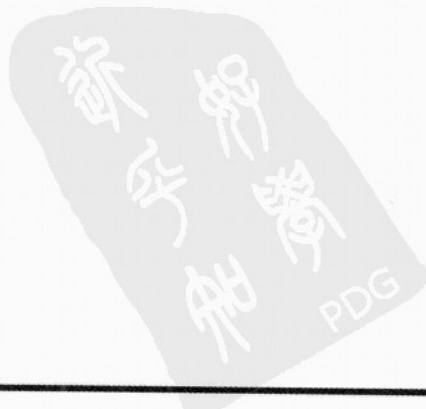
2. 本课网络关键词

经上述对本课内容回顾后，在下面的内容中，将对本课中的主要知识点进行收集整理，总结出本课知识的网络关键词。

本课知识的网络关键词有：“HotJava”、“J2SE (Java2 Standard Edition)”、“J2EE (Java 2 Platform,Enterprise Edition)”、“J2ME (Java 2 Micro Edition)”、“JDK”。

通过整理上述网络关键词，读者可以在百度、Google 或 Yahoo 等网络中获取上述关键词的基本知识，并且可以获取每个知识点的对应使用实例，读者可以通过获取的实例来加深对知识的理解。

希望通过本课内容的学习，读者能够掌握 Java 的基本知识有一个初步的认识，为进入本书后面知识的学习打下坚实的基础。



第 2 课 Java 开发利器

Java 程序和其他程序一样，可以通过记事本编写程序，但是这种编写对于一个大型项目来说需要耗费大量的时间和精力，如果想要快速方便地开发，用记事本开发是不现实的，所以，在本课中，将带领读者认识 Java 当前最流行的开发利器 Eclipse 和 NetBeans 的下载、安装与简单使用。

2.1 本课学习目标

在本课的内容中，将详细讲解 Java 开发利器，引导读者进一步快速开发 Java，为步入本书后面知识的学习打下坚实的基础，本课的具体学习目标如下：

- 下载与安装 Eclipse。
- 安装 Eclipse 汉语语言包。
- 用 Eclipse 新建一个项目、一个程序。
- 下载与安装 NetBeans。
- 用 NetBeans 新建一个项目、一个程序。

2.2 下载与安装 Eclipse

要用 Eclipse 开发程序，必须要下载和安装 Eclipse。

2.2.1 Eclipse 简介

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境，幸运的是，Eclipse 附带了一个标准的插件集，包括 Java 开发工具。Eclipse 最初由 OTI 和 IBM 两家公司的 IDE 产品开发组

创建，起始于 1999 年 4 月。IBM 提供了最初的 Eclipse 代码基础，包括 Platform、JDT 和 PDE，2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会（Eclipse Foundation）管理，2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构，2007 年 6 月，稳定版 3.3 发布，2008 年 6 月发布代号为 Ganymede 的 3.4 版。

Eclipse 是著名的跨平台的自由集成开发环境（IDE）。最初主要用来 Java 语言开发，但是目前也有人通过插件使其作为其他计算机语言比如 C++ 和 Python 的开发工具。Eclipse 的本身只是一个框架平台，但是众多插件的支持使得 Eclipse 拥有其他功能相对固定的 IDE 软件很难具有的灵活性，所以许多软件开发商以 Eclipse 为框架开发自己的 IDE。

2.2.2 Eclipse 的下载

用户可以打开“Eclipse”官方网站，即可下载“Eclipse”程序，其具体操作步骤如下。

(1) 打开 IE 浏览，在浏览器中输入网址“http://www.eclipse.org/”，然后单击“Download Eclipse”超级链接，如图 2-1 所示。

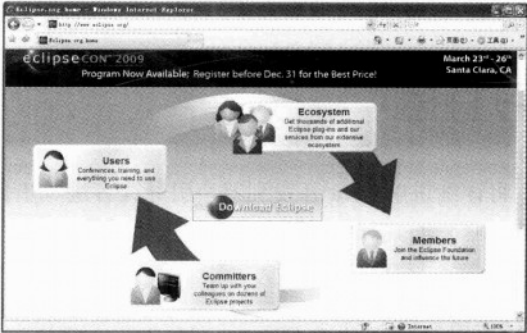



图 2-1 Eclipse 官方首页

(2) 进入 Eclipse 下载页面，在“Eclipse IDE for Java EE Developers (163 MB)”栏中，用户根据操作系统选择 Eclipse 版本，这里单击“Windows”超级链接，如图 2-2 所示。



图 2-2 单击需要的版本

(3) 单击图标, 即可下载 Eclipse, 如图 2-3 所示, 系统将自动完成下载工具, 这里启动了“迅雷”进行下载, 如图 2-4 所示。

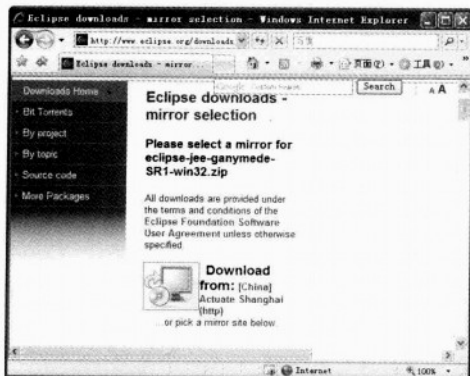


图 2-3 单击图片超级链接

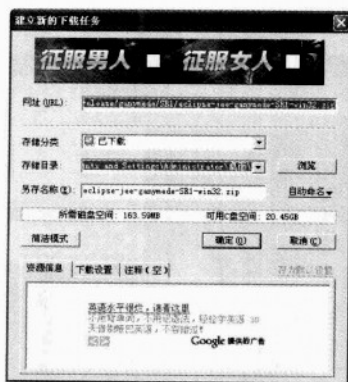


图 2-4 下载 Eclipse

2.2.3 Eclipse 的安装与汉化

Eclipse 的安装十分简单, 但汉化稍微有些复杂, 下面对两者进行详细讲解, 其具体操作步骤如下。




(1) 将下载的文件进行解压, 然后打开的文件, 双击“Eclipse”图标, 将启动 Eclipse 程序, 如图 2-5 所示。



图 2-5 Eclipse 程序

(2) 打开“Workspace Launcher”对话框, 单击按钮可选择项目的路径, 这里设置在 D 盘下的 JAVA 文件, 然后单击按钮, 如图 2-6 所示。

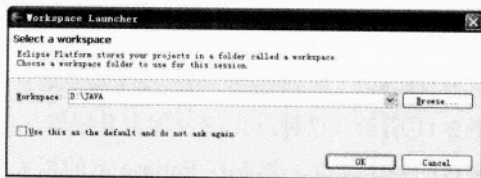


图 2-6 “Workspace Launcher”对话框

(3) 将进入欢迎界面, 选择“Help”→“Software Updates”命令, 如图 2-7 所示。

(4) 打开“Software Updates and Add-ons”窗口, 单击“Available Software”选项卡, 在该选项卡中单击 **Add Site...** 按钮, 如图 2-8 所示。

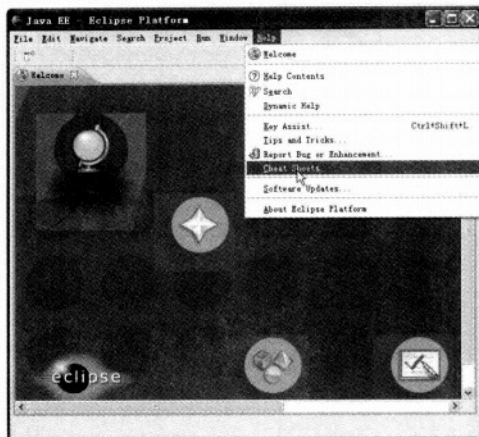


图 2-7 选择命令



图 2-8 “Software Updates and Add-ons” 窗口

(5) 打开“Add Site”对话框, 在“Location”文本框中输入地址“http://download.eclipse.org/technology/babel/update-site/”, 单击 **OK** 按钮, 如图 2-9 所示。

(6) 返回到“Software Updates and Add-ons”窗口, 单击 **Refresh** 按钮, 软件将自动下载, 这需要很长时间, 用户需要耐心等待, 如图 2-10 所示。

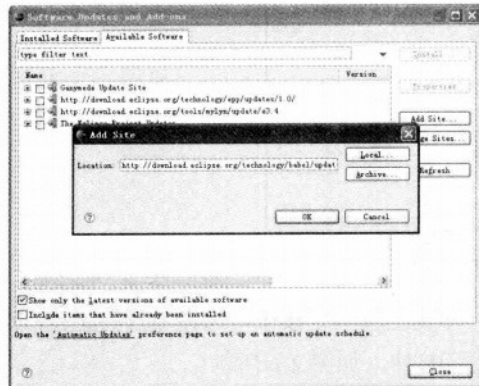


图 2-9 “Add Site” 对话框

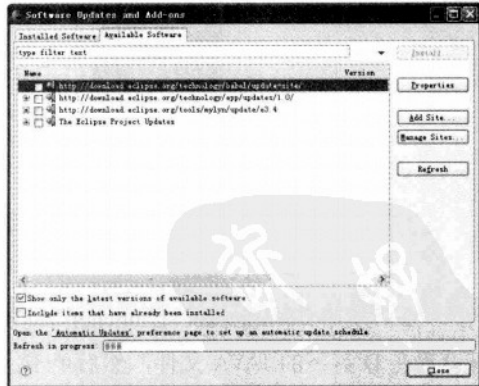


图 2-10 下载语言包

(7) 然后在列表中选择 ☒ Eclipse Language Pack for Simplified Chinese 复选框, 单击 **Install...** 按钮, 重新启动软件, 如图 2-11 所示 (这种汉化的过程十分缓慢, 用户也可以通过搜索引擎下载汉化包, 然后将汉化包的各个文件, 复制在 Eclipse 下的相关文件夹汉化)。

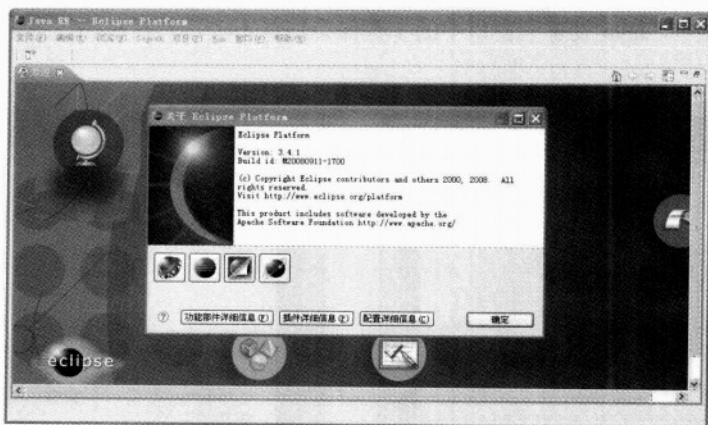


图 2-11 汉化过后的 Eclipse

2.3 新建一个 Java 项目

启动软件后，即可建立一个 Java 项目，下面讲解用 Eclipse 新建一个项目，再建一个 Java 程序，其具体操作如下。

(1) 启动软件，选择“文件”→“新建”→“项目”命令，如图 2-12 所示。

(2) 打开“新建项目”窗口，然后在“Java”树形列表中选择“Java 项目”，单击 **下一步(N) >** 按钮，如图 2-13 所示。

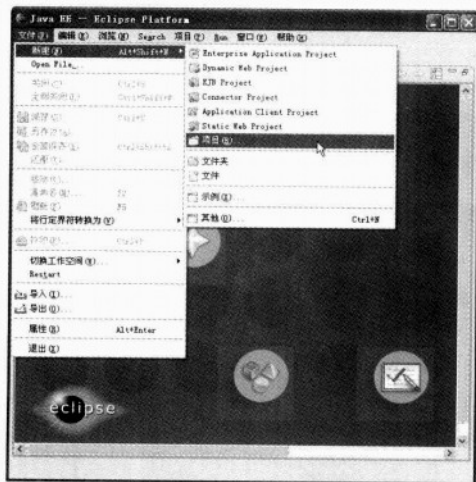



图 2-12 选择“项目”命令



图 2-13 “新建项目”窗口

(3) 在“Project name”文本框中输入项目名，这里假设“two”，单击 **下一步(N) >** 按钮，如图 2-14 所示。

(4) 在打开“Java 设置”窗口中的参数保持默认不变, 然后单击  按钮, 如图 2-15 所示。

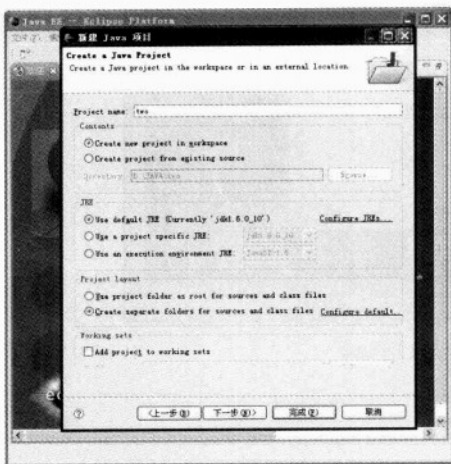


图 2-14 输入项目名

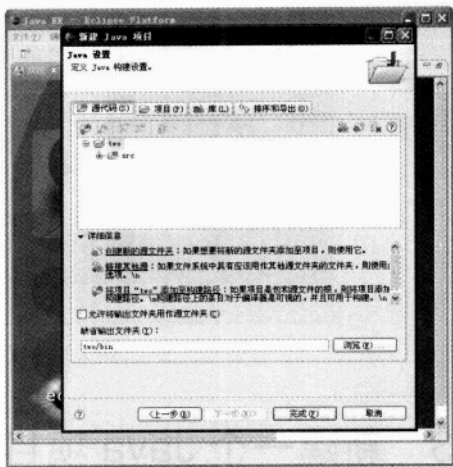


图 2-15 “Java 设置”窗口

(5) 打开“要打开相关联的透视图吗？”对话框，单击 按钮，如图 2-16 所示。

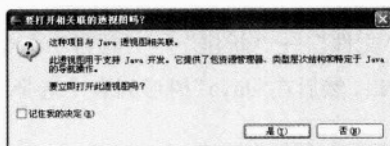


图 2-16 “要打开相关联的透视图吗”对话框

(6) 选择“文件”→“新建”→“类”命令,如图2-17所示。

(7) 打开“新建 Java 类”窗口，在“名称”文本框中输入类名，这里输入“Helloworld”，用户可以根据自己需要勾选各种参数，如图 2-18 所示。



图 2-17 选择“类”命令



图 2-18 “新建 Java 类”窗口

挺进，他们试了许多次，但总是未能完全成功。第一次是 Smalltalk 发动的，这是一个高生产力的环境，但是当经验不足的开发者的开发者试着将它推向自然边界以外的時候，他们就遇到困难了。一开始，对于 OOP 过早、过度地吹捧反而招致不好的后果。OO 语言被定位成工具，可以到达程序代码时再次被使用，并认定没有经验的 OOP 团队的生产力可以比传统的面向过程的开发提升好几倍。

面向对象软件或许比过程化程序设计开发来得简单，但是需要一些时间积累 OOP 的经验，如识别模式、为 OO 软件正确地切割出层次。整个软件界也需要一段时间，才能提供经过培训的开发者的。虽然现在看起来 OOP 好像是一夜之间蓬勃发展起来的，但其实根本就不是如此。经历过早期的一些失败（例如 Smalltalk），系统程序员回到规划阶段，设计出较不易混淆的 OOP 语言，并用更受限的方法传递 OOP 的概念，如图 2-1 所示。

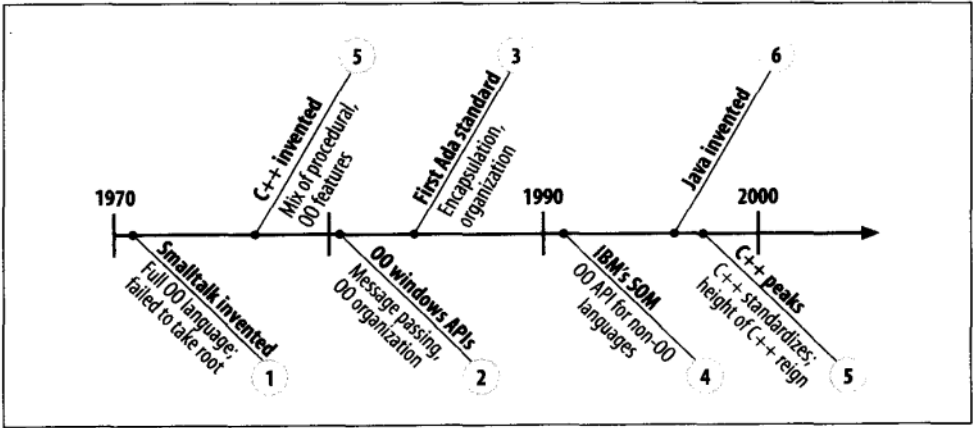


图 2-1：此时间轴展示 OOP 在商业上如何一步一步地获得采用

1. Smalltalk，发明于 1971 年，是成功的研究项目，但商业上并不成功。
2. 20 世纪 70 年代末 80 年代初期，各种 API（尤其是与视觉相关者）开始将接口组织成逻辑上的行动（称为事件）以及形成对象（例如窗口和控件）的概念。
3. 20 世纪 80 年代，美国国防部开发出 Ada 编程语言，其具有一些 OOP 的特色，像封装和继承。
4. 像 IBM 与微软这样的公司设计一些工具包，让用户可以在过程语言中

表达面向对象的想法。最值得注意的是 IBM 的系统对象模型 System Object Model 以及微软的 Component Object Model。

5. C++ 让 C 开发者可以把它当作过程语言来用，同时也可以当作面向对象语言来用。

6. Java 被发明了，结合了一路上其他软件的许多优点。

不幸的是，C++ 自身的问题也不少。

C++ 的经验

程序员不只要和 OOP 进行斗争，他们选择的语言所具备的各种议题也会影响到他们。Visual Basic 开发者开始了解到语言和环境可以很简单，但是性能和设计就比不上其他软件，这使得客户使用软件时必须忍受速度很慢，且未来系统不容易扩展。

对 C++ 来说，服务器端的开发者发现效率很高，但却有别的问题：他们用“系统编程语言”在开发“应用软件”。一些新的术语，像“内存绊跤”以及“DLL 地狱”都是用来描述这种混乱，简单的问题却纠缠不休。

指针运算

在 C++ 中，指针可以指向任何内存，不管是不是正确的地方。例如，在例 2-1 中，将内存从甲地搬到乙地，并重新返回。不幸的是，此范例偏差了一个位置，此程序代码接触到 from 内存边界外面的地方，你可能不会立刻见到错误，但是如果你稍后试着管理此块内存，或者另一个内存的时候，你就会看到错误发生了。C 和 C++ 编译器常常用链接列表（linked list）来管理内存，而此时指向列表下一个块的指针却落于配置的块之外！这种错误让系统开发者焦头烂额，让应用程序开发者死不瞑目，因为应用程序开发者没有足够的能力找出这类错误，其可信度也会受到很大的影响。

例 2-1：移动且倒转内存

// 移动且倒转内存，从 from_block 到 to_block，大小为 size

```
int i;
for(i=0; i<size; i++) {
    to_block[size-i] = from_block[i]; // off by one!
}
```

嵌套引用

我对内存的强烈沮丧经验来自在 IBM 移植一个 C++ 应用程序的过程，此应用程序嵌套了 37 层引用 (include)，这导致难以维护的问题，对没有经验的开发者来说，尤其如此。

问题就像这样。在 C++ 中，你在头文件 (header file, .h 文件) 中指定方法的接口，并附带其他支持的信息。例如，在 MySQL 中，你有一个主要的引用文件 (include file)，里面写了这些引用 (为了简单，我省略了大部分的程序代码)：

```
#ifndef _global_h                /* 如果不是标准的标头 */
#include <sys/types.h>
-
#include <custom_conf.h>
-
#ifdef _LCC_
#include <winsock.h>             /* 给 windows 使用 */
#endif
-
#include "mysql_com.h"
#include "mysql_version.h"
```

这看起来并不糟糕，但是想想其中有些引用是有条件的编译，所以，在你能够确定是否有某些东西被包括进来之前，你必须知道哪些编译器的伪指令 (directive) 被设定过。还有，你的引用文件还可以含括别的引用文件，就像 *mysql_version.h* 引用文件中的下面这行一样：

```
#include <custom_conf.h>
```

事实上，此 MySQL 树只有三层，这是绝佳的范例，展示用 C++ 写企业软件的正确做法。但事情通常不会这么简单，任何依赖都需要一个引用文件，而如果程序代码依赖其他程序，你还必须一次又一次地确定这个引用文件以及它们的关联链接库都安装在正确的地方。

Java 完全没有这个问题，你只需要一种类型的源文件，带着导入 (import)，而且没有条件编译。

字符串

许多大公司使用C++进行企业应用程序的开发,即使如此,对于管理字符串的支持依然相当有限。C程序就只是使用字符数组当字符串,像这样:

```
char str [] = "Hello";
```

这会造成固定长度的字符串被分配给str。这只是一个字符数组,而且所能持有的字符串长度还不能超过6个字符。你可以决定改用C++字符串链接库。

C++支持C风格的字符串链接库作为某种“类字符串”的特色。例如,分配内存后,把一个字符串赋值给另一个字符串时,你需要用复制byte的方式:

```
strcpy (string1, string2);
```

C风格的字符串很丑、危险、乏味。利用这种指针处理时,你可能会在走过块的时候,不小心埋下错误的种子,可能数小时或数月都不会被发现。C++字符串和其他语言(包括Java)比起来也是相当乏味的。

从1997年开始,ANSI的C++标准导入一种更正式的字符串,你可以用更自然的方式使用它,像这样:

```
String str = "Hello, I'm feeling a little better.";
```

许多C++链接库都已经有了专属的字符串链接库,但是伤害已经造成,许多程序员都已经会C,所以不会去用C++风格的字符串。

DLL 地狱

在微软操作系统和OS/2上,你可以编译出依赖别的链接库的链接库,操作系统会将这一切都连接在一起,这种做法就称为动态链接库(Dynamic Linking Library, DLL),但是OS没有做任何依赖检查。许多应用程序都会共享相同编程链接库的各种版本,当安装你的应用程序时,你可能会将某个链接库替换成你的版本,造成别人的应用程序不能与你的链接库兼容,所以无法顺利执行。今天,微软的操作系统依然受到DLL地狱之苦。

CORBA

随着C++社区的成长,他们开始将脚步走向客户机/服务器结构(client/

server)。Common Object Request Broker Architecture (CORBA) 快速出现。利用 CORBA, 你可以从接口定义严谨的对象建立应用程序, 你可以用一个对象, 不需要增加遥控 (remoting) 逻辑, 在因特网上使用。像 IBM 这样的公司试着将 CORBA 模型推广到每个对象上, 像 Iona 这样的公司将焦点放在远程对象的分布式接口上。CORBA 周围的木材日渐升温, 但是依然没点着火。人们建立应用程序, 但此类应用程序依赖网络上细微的沟通, 太多来回的沟通造成性能不佳, 以至 CORBA 的声誉不佳。

继承问题

C++ 将业界向 OOP 推进了一步, 但是这一步实在不灵活, 甚至有不良的后果。C++ 有三个主要的问题:

- C++ 并未强制使用面向对象的方式, 你可以写出独立于类之外的函数, 结果造成许多 C++ 写出来的程序并非面向对象的, 面向对象的 C 反倒变成 (C++)--。
- C++ 没有强制只能有一个根对象, 这造成对象树有许多不同的根源, 这已经被证实会困扰面向对象的开发者。
- C++ 支持多重继承, 程序员没有从经验中累积使用多重继承的智慧。许多语言从这一点得到教训, 利用较清楚的 *maxin* 方式实现多重继承。

多重继承是威力强大的工具, 但是可能会对初学者造成很大的问题。例 2-2 展示了一个多重继承的例子, Werewolf 是半人 (Man) 半狼 (Wolf), 当 Man 和 Wolf 都继承自一个共同的哺乳动物类 (Mammal), 问题就来了。如果 Werewolf 继承一个方法, 此方法是在 Mammal 中导入的, 这时候究竟 Werewolf 是通过 Man 还是通过 Wolf 来继承此方法, 就不明确了, 如图 2-2 所示。这个问题, 称为菱形继承问题 (diamond inheritance problem), 表示此问题是和多重继承相关。

例 2-2: C++ 的多重继承

```
class Werewolf: public Man, public Wolf
```

多重继承就像任何威力强大的工具一样, 让你享受到很棒的功能, 但是如果你的经验不足, 使用起来却不安全, 会伤害到自己。大多数使用 C++ 语言的开发者不会去用多重继承。

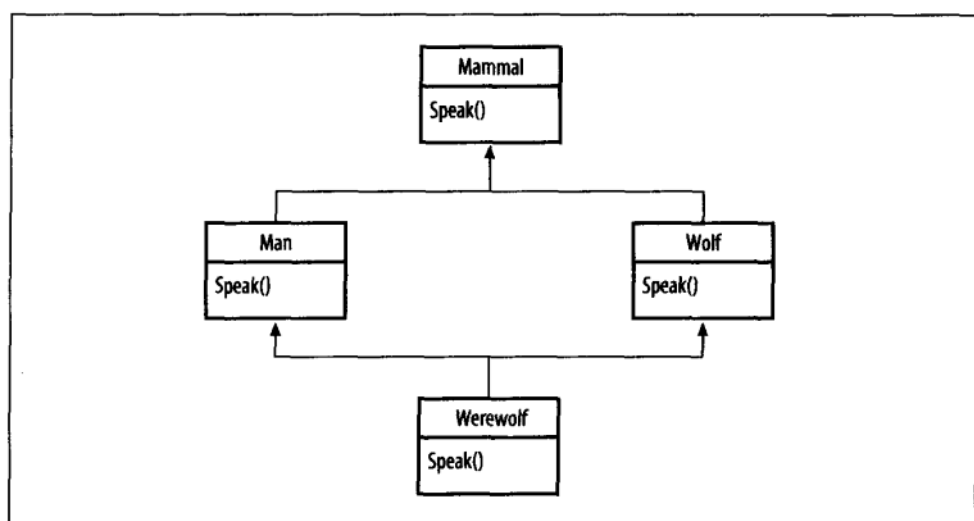


图 2-2: 菱形继承问题, 只是多重继承可能导致的麻烦之一

一致性

和 Perl 一样, C++ 是能够很好表达的语言, 但是这种灵活性需要的成本相当高。对于经验丰富的开发者来说, C++ 是一个充满特色的语言, 但是这些特色在运行时可能会引来灾难。比如, `=` 常常用来作为赋值和测试两种用途, 大多数新的 C++ 开发者, 会因此受到很多磨难。想要熟练运用 C++, 得花好多年的时间进行磨炼。对于系统开发者来说, 这还有一点道理, 毕竟你需要精确地控制每个字节。但是对于应用程序开发者来说, 根本不需要处理这些底层的细节。

可移植性

大多数开发者都希望 C++ 能够具有更好的可移植性, 但结局并非如此。我们被一堆如山的不兼容链接库所埋住, 不同平台上的链接库都不一样。C++ 留下如此多的包袱, 各种实践版本, 以至于 C++ 是有史以来最不具移植性的语言。近几年来, 问题更加严重, 甚至连相同编译器的不同版本链接库都常常无法链接成功, 更别说不同操作系统了。

就像是靴子上的泥土累积, 曾经看起来很酷的语言开始让最聪明的开发者觉得脚步变重变慢, 而一般的开发者更是难以行走。不愿改用受限的语言, 例如 Visual Basic 或 Power Builder, 大家开始等待, 暴风雨的乌云依然集结。

妥协

没有这些状况，完美的风暴就无法成形。Java一开始的成功，正是由于C++族群的异常迁徙，为了要这样，Java必须用绝佳的平衡感在绳索上走路。C++有一些明显的问题，像糟糕的语法、多重继承、用基本类型而非对象、类型（typing）模型、困窘的字符串、笨拙的链接库。在某些状况下，Sun决定设计出简单且清楚的应用程序语言。Java的研究源自嵌入式语言，所以相当简洁。另一方面，Sun也决定保留许多C++的语法，以吸引C++社区。

现在任何人都可以轻易地回过头来看Java并批评这些决策，但是对我来说，他们当时取得相当好的平衡点，这是毋庸置疑的。各种大肆宣传迅速地在Java周围蔓延开来，这使得它的成功完全超出大家的预料。就在微软和IBM的战火中，这一切发生了！如果Java当时就停下来，它就成功了，但是它没有停止，长期来看没有停止。

拨云见日

我们还没有防范，Java风暴已经袭来。为何不呢？它的来源不同，载体不同，且漠视大家对于解释性语言性能上的意见。和其他语言不同，Java的一切都不同常规，连形成风靡的程度也是。回顾以前，你可以看出Java是如何完美地填补起这个空隙的。图 2-3 显示许多因素在一起形成了完美的风暴。

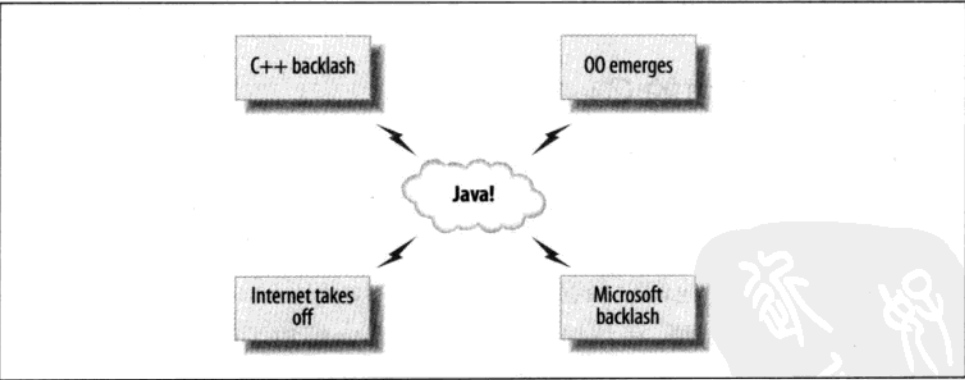


图 2-3：许多力量结合在一起形成完美的风暴

新经济

喷射流 (jet stream) 让这个风暴具有威力，这来自于一系列的标准：TCP/IP、HTTP、URI 以及 HTML。因特网气势高涨，而 Sun 则利用 Java 的优点。因特网无所不在，Java 很酷。Java 开发者很快地建立了 API 集，让开发者可以为因特网写程序，包括用 TCP/IP API 进行沟通，applet 用来建立嵌入网页的用户界面，JDBC 用来访问数据库。

Netscape Navigator 和 Java 完美的结合将两家公司都推向了新的高峰。通过 Netscape，Sun 立刻让 Java 呈现在大量开发者的面前；通过 Java，Netscape 可以展示聪明有趣又实用的应用程序。Navigator 和 Java 的结合，几乎可以解决 client-server 运算中最严重的问题：管理与发布。可以安装浏览器的地方，就可以通过浏览器发布应用程序。Java 有完美的经济局势，可以导致成功。在大型计算机和绿色终端机的方便管理，与胖客户端程序的丰富和生产力的之间，Java 取得绝佳的平衡。

客户想要解决方案，Sun 意识到 Java 可以给客户带来他们需要的一切。Sun 立刻看到眼前的机会，Java 运用因特网的开放标准，Sun 服务器上的 Solaris 变成受到关注的操作系统。总而言之，Java 的出现使 Sun 安全了。因为 Java 虚拟机是在浏览器和许多不同的 OS 上运行，许多不容易做的决定也就变得容易多了。你可以试试一套部署情节，如果你不喜欢，可以换一套做法。

新的喷射流正在增加风暴威力。

服用抗忧郁药物的 C++

当 Lucene 创办人 Doug Cutting 称 Java 为“服用抗忧郁药物的 C++”，我马上喜欢上这个比喻，因为 Java 接近 C++ 语法，所以立刻吸引大量程序员的注意，他们改用 Java，以为自己的程序加入因特网的功能，然后他们就不打算回去用 C++，因为他们已经喜欢上 Java 了。Java 比 C++ 具有更多的优点，但是却没有以往的问题，语法接近所以好学。Java 解放了他们，理由相当多：

- Java 在需要的地方提供了更多的结构，例如，提供接口替换继承。
- Java 减少了指针的必要性，让软件更稳定、更好阅读。

- 垃圾收集更容易，因为JVM自动处理不用的内存。
- Java允许更好的封装机制，简化了程序代码的使用。
- Java消除了一些问题，像嵌套的引用文件（nested include file）以及宏（macro）。

架构

Java的优点超出经济和C++。我依稀记得我第一次看到描述Java的句子，Sun说Java是一个可移植、安全、可靠、面向对象、分布式的因特网编程语言，当时这些可都是热门的词。对于C++开发者来说，Java的根基跨出了巨大的一步：

- JVM具有空前的可移植性。许多专家认为Java最重要的特色在于JVM而非语言。Sun很聪明地将可移植性营销为缩写WORA，Java的开发者都知道这就是Write Once, Run Anywhere（写一次，到处都可以运行）的意思。
- Java为JVM出版了byte code规范，人们可以依据这个规范来建立自己的JVM或自己的语言，甚至修改现有程序的byte code。像JDO这类的框架的确修改了byte code，而且相当成功。
- C++允许对内存进行不受限制的访问，Java则是限制对JVM的访问，这称为沙箱（sandbox）。即使在今天，Java在安全上的问题还是少之又少。
- Java的元模型（metamodel），通过类对象来描述Java类型，允许我们使用相当精巧的反射编程（reflective programming），虽然使用起来有一点笨拙，但是Java这方面的能力远比C++先进。Java的元模型让框架增加了透明度，像Hibernate（保存）以及Spring（遥控与交易等服务）。
- Java之父看到了安全的重要，并将它集成进语言中。Java将程序员世代带进沙箱中，用沙箱来限制范围以及应用程序的破坏力。
- Java已经改进封装和扩展的能力。你可以有效地将扩展渗入Java中，就可以透明地为语言增加能力。你可以用不同类型的文档来封装以及发布程序代码。

不管是低级的新手或高级的架构师，大家都有喜爱Java的地方，商人有动机去推动。在这个时候，如果其他一切都失败，Java会变成成功的语言。但是没有失败，风速正在加快，风暴自己越来越强。

暴怒的释放

applet抓住许多程序员的想象力，这解决了部署的问题，很酷，也很容易建立。我们现在发现到一组技术，用丑陋且常被反对的JavaScript语言，能够建立丰富的Web内容，就和Java一样。applet开始衰退了。

即使在今天，我还是认为applet代表一种很好的构想，但是它会被冷落不是没有原因的：Netscape浏览器的bug很多，往往无法预料。更何况，Java本身变化这么快，applet本身会遇到的问题和client-server运算会遇到的问题如出一辙。你或许不必维护应用程序，但是你还是得维护浏览器。在你配置几个Java applet之后，你就会开始担心浏览器的版本问题。随着JVM的体积变大，要远程安装JVM也就越来越不可能。就算你能这么做，Java版本更新的速度实在太快，差异也不算太小，新的应用程序常常需要具体指出使用哪个版本。但是Sun的一些疯狂科学家准备好接受挑战了。



James Duncan Davidson：Java为何赢

《The Rise of Java》的作者

James Duncan Davidson是一个自由程序员、摄影师、作者、演讲者。他发明了Ant和Tomcat，这是两个Java界最成功的开放源码项目。他的坚持不懈让两个项目走向开放。他现在是Apple操作系统的畅销书作家。

你最喜欢Java什么地方？

JDD: 那个时候，Java看起来是很棒的东西。特别是，Java被设计的目的，他们的方向对了。当然，Java是一个强类型（strongly typing）的语言，这在某些方面很好，但在另一些地方反倒是缺点。

你认为它为何会成功？

JDD: 我认为，用Perl和类似的技术进行服务器端的编程，是很没效率的，而用C和C++则是很难。Java，特别是servlet，真正找到了可以好好发挥的施力点。

我可能会有一点偏心，毕竟我涉入 servlet 相当深。但是如果 Java 没有 server 端的部分，一切就不会这么有趣了。过去如此，现在还是如此。没错，J2ME 出现在许多手机上，但是实际的应用程序还不多，而且 API 的限制很多。

你不喜欢什么？

JDD: 强类型 (strongly typing)，对 API 的信心而不是对框架的信心。这是很微妙且重要的区别，即使简单的 API 也增加了复杂度。例如，你再也不可以只写一个 servlet，你必须写一个 servlet，然后编辑一个 XML 文件。他们磨灭了 servlet 简单易用的特质，这使得 server 上的其他技术开始越来越难以使用。而且对工具的信赖更是让这一切成为一个借口。

还有，我不喜欢用厚重的方式去“编辑”。大部分的人不需要 J2EE，他们只需要一个 Web 容器，就这样。

我也不喜欢唐突地进入很复杂的世界，但是使用 Java 的结果似乎就是如此。没有其他平台能像 Java 这样，这么快就长成一头大象。其他的平台能力一样，但是体积只有 Java 的 5%，所以他们实在需要好好地进行架构。其实，Java 语言没有什么大问题，问题是在 Java 类库上，类库采用大而无用的设计。你看过来自 JSP/servlet/J2EE 容器的堆栈追踪吗？让我暗示一下，将堆栈内的调用一个一个地输出来，差不多需要 44 页（数千行）。

Java 怎样阻碍你？

JDD: 它不会阻碍我。我现在不太用 Java 了，我使用其他的语言，像 Python、Ruby、ObjC。

我希望更多的工具可以记得 Unix 为我们带来的启示：用小小的碎片组成大大的世界。没有一个东西可以包山包海。只要你拿一个管子将两个工具串起来，你所得到的威力和灵活性比任何一个巨大怪兽都更好，这样的威力之大超乎你的想象。

你觉得有没有什么技术可以用来代替Java?

JDD: 对于服务器端的应用程序来说, Ruby on Rails 相当有吸引力。如果你需要涉及大量的内容, 有的内容数据库(像MarkLogic)可以使用XQuery来处理内部东西。在GUI的前端, Java就没有竞争力了, 所以任何东西都可以当Java的替代品。

什么提醒你不可以远离Java或.NET了?

JDD: 我把焦点放到GUI应用程序上, Java和GUI无法很好地交融, Swing是行不通的。

Servlet

当applet在client端松懈下来的时候, server端却继续进行。servlet让Java开发者可以写应用程序, 搭配浏览器使用。应用程序可以接受HTTP请求, 建立单纯的网页(不含Java applet), 然后返回给客户。既然网页是在服务器端建立的, 所以可以产生动态的内容(像数据库查询结果)交给客户端。所谓的Web-based应用程序, 最后不负众望: 现在, 你可以在客户端使用企业应用程序, 你只需要将系统部署到服务器上即可。

客户可以在公司的防火墙内, 但是他们也可以不必这样。现在任何人都可以访问因特网, 这就有机会销售一种新产品: 信息。新经济已经诞生, 至少有一部分要归功于Java, 以及建立这些服务器、软件、数据库的公司。新创的公司想要抓住机会, 创造了巨大的“纸上财富”。风险投资家对于好的想法和坏的想法一律投资, 大家都在抢顾客导致了狂怒的暴风雨。规则很简单: 有最多顾客者就是赢家。新创公司对于顾客的渴望, 让他们愿意赔本找客户。

真实的财富也被创造了。像eBay和Amazon这样的公司, 不用实体的建设就可以提供新经济动力, 这种全新而复杂敏感的经济, 驱动对于新工具的需要。Sun、Oracle、BEA、IBM都致力于新的标准, 以让企业能够跟得上Web的脚步。IBM喊出e-business, 用来称呼这种服务客户的新的、有威力的方法。

J2EE

J2EE，也就是 Java 企业版，包含许多连接到企业的新方法。在众人的期待中，Enterprise JavaBeans (EJB) 规范出现了，用来增加一组工具，帮助编写分布式、事务型、安全以及稳固的应用程序，而不需要自己写入程序。群集 (clustering) 技术带来了很好的可测量性 (scalability) 以及稳定性。这些特色，让许多大公司毫无保留地进入 Java 的世界。

尽管 EJB 没有达到它自己的承诺，规范是一种特别的范例，让想法可以活跃一个社区。EJB 背后的规范相当重要，最重要的是，此规范规划得相当好。Java 在 Server 上面相当成功，而且再度加入竞赛。

行业标准

通过共同的利益来结合并不容易。如果只有 Sun 在背后撑腰，Java 是不会繁荣的，某些力量需要团结起来。大家对于微软的敌视心理，正是一个完美的催化剂。

软件行业比其他行业更容易形成独占局面，因为软件变动很快速，老旧不堪的软件可能会摧毁一家公司。因此，市场占有率偏向于拥护市场领导者。所以，市场领导者喜欢专属技术，他们可以通过领导地位来增加市场占有率，将他们的客户锁定起来，无法离去。的确，微软不是第一家这么做的公司，IBM 过去也是这样。

专属技术对于市场领导者有效，市场上未领先者就会需要开放标准，让游乐场平坦。如果你在市场上并非主导者，你可以通过建立伙伴关系以及拥抱共通标准让你的客户安心。用这种方法，你的顾客就不用担心软件会变成废物。

在 Intel 和微软主导的情况下，Unix 操作系统帮助许多小的专属服务器厂商存活了好多年。在积极支持专属系统数十年后，IBM 开始在许多领域拥抱标准，包括关系型数据库 (IBM 在数据库方面落后于 Oracle)、操作系统 (运用开放源码的 Linux 环境，IBM 让大型计算机更安全)、Java。

IBM 现在是最全面的 Java 开发者，它的 Java 开发者比其他任何公司都多，甚至连 Sun 都比不上 IBM。我相信 IBM 的能耐，它多年来紧咬着 BEA 的 Web

Logic 应用程序,现在已经超越了 BEA。我期待 IBM 用它现在具有主导性的力量建立符合客户利益的专属特色,我也期待 IBM 会对 Java Community Process (JCP) 采取强硬路线,让 JCP 能符合 IBM 的利益。如果不这样做的话,它可能会直接离开 JCP,另立自己的标准。如果它这么做,这个“专属化”的策略也就不会让人意外了。这是市场领导者的特权,事实就是这样。

开放源码

许多开放源码社区轻视 Java,这很讽刺,因为 Java 具有很多开放源码软件,这是其他技术所不能及的。当你建立既时髦又受欢迎的东西时,许多人想玩玩看,并分享他们的创造结果。用户人数一多,语言会被拉向无法预料的方向,你只需要静静欣赏,就会看到许多有趣的事情。噢!通过 Java,开放源码发生了。

一开始, Sun 和开放源码群体保持距离。Sun 开发者, James Duncan Davidson, 致力于改变当时的状况。他建立两个有史以来最重要的 Java 应用程序, Tomcat (用来展示 servlet) 和 Ant (今日所有 Java 应用程序几乎都是用 Ant 建立的)。然后他将这些产品送给了开放源码社区。

典型的开放源码开发周期如下 (如图 2-4 所示):

1. **建立。**一旦 Java 技客 (geek) 解决了一个问题,他们常常会用自己的资源建立解决方案。有时候,他们解决业务问题,有时候,他们只是从中找乐趣。
2. **使用。**用户使用此解决方案,那些没人使用的解决方案就会枯萎、死亡。
3. **修改。**用户接着修改此方案,以符合他们的需求。
4. **贡献。**用户接着对项目做出贡献,可以是给意见,也可以反馈修改后的程序代码。他们愿意这么做,因为他们可以不必维护修改后的程序代码。

通过这样的做法,某些想象中的框架就演化成为 Java Web 应用程序开发的基础。今天,大部分的大公司都会运用开放源码的软件,这样的解决方案在 Java 社区中是相当普遍的:

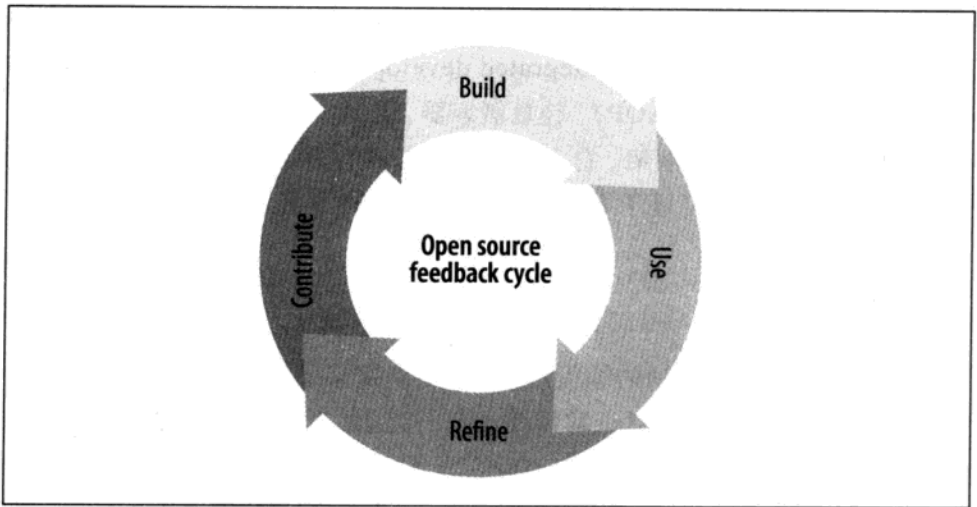


图 2-4：开放源码反馈周期对于 Java 非常重要

- 开发者使用 JUnit 建立机械化的测试案例，每次 build 软件的时候，都会顺便做一次测试。
- Apache Web Server 成为 IT 商店对于 Web server 的首选。
- 顾客部署许多轻量级的应用程序，利用 Tomcat 为 servlet 容器。
- 开发者使用 Hibernate 进行数据保存。
- Web 开发者使用 Struts 来切割系统的 MVC。
- 全世界的程序员都使用 Ant 建立程序。
- 其他的框架，像 Lucene（搜索）、Spring（底层）、Tapestry（Web 组件设计）、JBoss（J2EE）以及许多其他公司的产品开始受到欢迎。

你或许会认为，开放源码会威胁建立软件的软件公司，但是事实相反。开放源码对 Java 的帮助很大。开放源码社区的创新让软件公司有压力，使其必须持续进步，这是很健康的。如果你提供真正的价值，你就成功了。如果你试着远离大家都在使用的技术，你就完蛋了。开放源码软件让赚钱的门坎变高，IBM 对于这样的压力适应得很好，BEA 却水土不服。上面有 IBM，下面有 JBoss，将来如果你没有看到 BEA 的改革，你就会看到像 JBoss、Geronimo、Spring 这一类开放源码的框架将它吞噬。不管怎样，你都是赢家。

你可以说，开放源码软件正驱动着Java领域最重要的创新。开放源码造成下面技术的采用与实现：IDE (integrated development environment)、aspect-oriented programming (AOP)、轻量级容器、持续、单元测试 (unit testing) 以及最好的Web MVC框架。它正驱动最大与最有力量Java厂商的行为。这一切，很不可思议地见证了Java开放源码社区的力量。

事过境迁

我相信Java是目前最成功的编程语言，它重新定义了我们封装软件、递交软件的方式，它改变了我们对于解释性语言的想法，以及我们建立因特网应用程序的方法。Java将部署和管理一起揽下，让我们节省部署应用软件所花的精力。它大量使用链接库，对Web强力支持。Java带来许多重要的标准，形成企业软件开发的标准。Java已经改变了游戏规则，Java重新定义了什么是商业上的成功语言。

在某些方面，Java所订下的规则相当适用于我们。想要达到类似的成功，新的语言需要具有可移植性，并鼓励开放源码社区的积极参与。它需要多方面吸引人，不管是对低水平的程序员或高级架构师来说，都要能感受到它的魅力。它还需要拥抱许多标准。

但是技术只是一部分的问题，一个新的语言想要成功，需要具备业务上的强力理由，才会让人转换过去。某些方面，Java会阻挠我们，因为它不鼓励竞争。你可能会在许多不适当的地方依然使用Java，这使得你花费更多的力气，因为你不知道有更好用的工具存在。这种状况可能会引诱我们产生错误的安全感，所以许多Java程序员处于Java的茧中，感到怡然自得，完全没有接触外面的世界。

向前走

将我们带往Java的完美风暴已逝，或许我们再也不会有机会看到这种完美风暴了。你也不应该试图找一个风暴。相反，你应该从Java的成功中学到经验，学到什么是成功的因素。我相信，下一个成功的编程语言必须满足下面主要的4个准则：

- 需要建立一个明显的社区。只有让采用者安心，他才会去使用此技术。
- 需要具备可移植性，Java 虚拟机已经提高了后继语言的门槛。
- 需要提供经济上的动机。目前，生产力对我来说，看起来就像是经济上的动机，但还是有一些其他的动机在引诱着我，像无线运算以及数据搜索。
- 它需要展示技术优点，这是最重要的一点。

我不认为我们能够彻底分析Java成功的原因。我们很容易高估语言的角色，而低估了JVM和社区力量的重要性。在下一章中，我们将会继续更仔细地寻找Java皇冠上的宝石，并找出这个历来最重要的编程语言的根基。

第三章

皇冠上的宝石

在40分钟内经历了6次下坠，我回头看看河流，并自我反省。我觉得好冷好冷，6个小时没有进食了，我的头和背很痛，我觉得很害怕，担心是否会蒙主宠召。长时间扛着船行走，我的肩膀因为压迫而疼痛，这还不算什么，我面对着几乎无法覆盖住石头的一道水墙，有人可能曾经在这里受伤，甚至丧命，晚上空气中弥漫着无所不在的潮湿橡胶味。我却乐此不疲，泛舟将我带到无人能及的境地，我的方法都会立即反映在结果上。征服一小片流域的感觉美妙极了。

对我来说，Java一度就是如此。我通过Java绝佳的社区以及许多开放源码项目得到了很好的生产力。开放标准以及JVM意味着我的知识以及我的应用程序可以移动到不同的地方。Java无与伦比地成功，为何Java会如此受欢迎，我已经告诉过你我的观点。如果你了解Java之后会出现怎样的接班技术，你需要先了解Java一连串的成功因素：

- 什么东西让Java流行，吸引来这么多人？
- 不管Sun以及其他强大厂商做了些什么，这段时间开放源码厂商是如何兴起的？
- 是哪些不可或缺的技术让Java成功的？
- 什么原因让Java的适用面这么广，从网站到数据库？

回答这些问题，需要许多人的智慧。为了提供较好的答案，我访问了数十个顶尖的 Java 开发者，问他们 Java 如此成功的原因。表 3-1 整理了一些有趣的回答。

表 3-1：根据顶尖 Java 顾问的回答，整理出的 Java 成功的原因

顾问	为何 Java 如此成功？
James Duncan Davidson	我认为原因在于服务器端的 Perl 编程是很没有效率的，服务器端的 C 和 C++ 编程又太难。Java 和 servlet 找到机会，并把握住这个机会
Jason Hunter	Java 让你可以做一些事，而这些事是别的工具做不到的，这就是 applet。applet 最后并未成为重要的技术，但是它可以保护桥头堡，让 Java 一开始就建立了地盘，而且没有具威胁的竞争者
Dennis Sosnoski	Java 具有好语言和好虚拟机。在 1.5 版本之前，它的优点还包括清楚易学
Stuart Halloway	Java 比 C++ 好
Richard Monson-Haefel	Java 是一个很棒的静态面向对象语言，具有可移植性及大量的 API、产品、开放源码项目，也是一个设计良好的语言和虚拟机。一开始，它具有相当先进且合时宜的语言设计，而且可移植性也相当好。今天，它已经无所不在，这就是为什么它依然受到欢迎的原因
Ramnivas Laddad	Java 让垃圾收集以及反射（reflection）广泛地被接受，且成为主流。虽然这些概念已经存在很久，但是主流的开发者的直到 Java 出现才开始使用这些概念。另外，Java 让平台的独立性达到合理的阶段

现在，你的观念越来越清晰了。从表 3-1 的这些答案中浮现出数个重点：

- 成功所需要具备的技术门槛不是很高，因为这么多人用 C++ 开发业务应用程序，但是 C++ 又是系统语言，不太适合业务应用程序。Java 只需要简化这种开发体验，就可能成功。

- 开发企业应用程序的能力很重要, James Duncan Davidson 认为企业的中心问题就在于将自己因特网化。
- 底层的技术 (语言, 特别是 JVM), 往前跨出了很大的一步。
- 社区相当重要, 这是 Java 一项很大的成就。
- Applet 可能是让 Java 红起来的第一个功臣。

如果你将 2005 年的评论和 1997 年的评论比较一下, 你会发现有一些显著的差异: Java 的异常策略和静态类型现在已被视为一种妨碍, 而非一种优点; Java 的生产力不像它以前那样好; Java 对于服务器的影响比对客户端的影响大; Java 不像以前那么简单。但 Java 专家对于下面的看法依然一致: JVM 很重要、Java 适合开发因特网应用程序以及 Java 比 C++ 好。

语言和 JVM 的设计

1996 年, JVM 的出现象征了一个很重要的分界点, 它不同于传统的思维。组织无法抗拒在服务器端使用高性能的编译语言, 开发者只需要对安全进行修补, 不必一切从头来过。厂商通过在很高的层次上建立扩展的链接库, 试图达到可移植。为了不走这条老路, 他们试图用两手操控方向盘, 将所有的动力都置于一旁, 但却突然地进入一个遍布石头、地图上没有标示的未知区域。

可移植性

在 20 世纪 90 年代早期和中期, 业界开始思考可移植性, 特别是, 我印象很深刻, 当时我还在进行面向对象技术。这个计划称为 System Object Model (SOM), 来自一个研究项目, 它是 OS/2 划时代的面向对象桌面的基础, 也衍生出一些未曾离开实验室的研究性质的技术。SOM 的目标很大: 希望建立一个共通的对象模型, 尽可能多地包容对象模型。然后, 我们就可以开发一群共通的链接库, 用户可以使用这些链接库, 不管语言和平台。一段时间以后, 我们发现, 要将一个技术移植到其他不同的操作系统和语言上, 实在太困难了。我们很快丢弃了“类 Smalltalk”的集成开发机器以及虚拟机, 概念来自 Smalltalk 和 Lisp, 因为虚拟机可能会不够快。并非只有我们这么做, 许多 C++ 的公司试着建立跨语言的编程环境, 但很少有人成功。

Java的做法，如图3-1所示，根本不一样。Java的虚拟机就是一台新定义的机器，提供一些低级、基础的工具，以支持可移植性。Java设计者认为他们可以解决性能的问题，这并不是新的想法，也不是很流行的想法。一段时间以后，他们提供了JIT编译器，提高了性能，这使得JVM所带来的额外性能损失变小，可以被人们接受了，这甚至可以和解释语言分庭抗礼。虚拟机被内置到 Netscape Navigator 浏览器中，成了 Java 一飞冲天的跳板。这使得Java能够扩展进入移动领域、应用服务器以及无数个软件产品中。结果，虚拟机的想法就开始受到欢迎，这或许是 Java 最大的技术贡献。

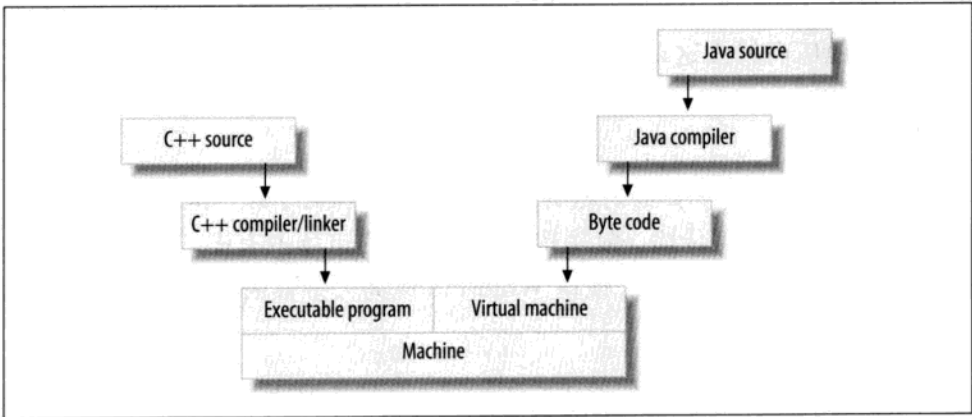


图3-1：JVM对于效能、安全性、可移植性的做法不同以往；大多数的编程语言使用编译器将程序编译成个别机器的版本，但是Java就是重定义出自己的机器的版本

Java提供了丰富的界面，可以进入操作系统的领土。在表面的背后，Java如果不是重新实现自己的功能，就是请求操作系统提供支持。不管是哪一种做法，Java的开发者都可以依赖这一套丰富且一致的链接库，不管是什么平台。在几乎10年的软件开发之后，我将程序移植到不同的操作系统上运行时，未曾遇到什么大问题（但一些小问题是有的）。最后，虚拟机不是Sun发明的，但是却是靠Sun才流行起来的。

Java的可移植性并非毫无问题，图形化的用户界面尤其让人困惑：想要维持可移植性，究竟是使用一致的界面比较好，还是运用操作系统的界面比较好。例如，Swing组件，并非操作系统的组件，实现了Java GUI的菜单(menu)，这会造成苹果机平台上的困扰，因为苹果机传统是将应用程序的菜单放在桌面的单一栏上，而不是每个应用程序都有自己的菜单（像Unix和

Windows 这样)。其他值得注意的差异包括了安全和线程，也显示出在不同操作系统上的差别。即使是文件名称，都可能是一个问题，例如：两个文件分别叫做 *Account.txt* 和 *account.txt*，这在 Unix 上是合法的，但是在 Windows 上却不行。但是整体来说，Java 在处理可移植性问题上，算是相当不错的。

安全

在因特网的时代，安全 (security) 的重要性变得比以往高出许多，任何技术都不能忽视安全，都要好好地处理安全问题。Sun 的工程师知道这一点，并一开始就考虑到安全，还好他们有虚拟机，这可以让问题简化许多。不幸的是，其他厂商就没这么幸运了。

改变威胁

微软 Windows 被许多安全问题所困扰。因特网和电子邮件成为病毒的绝佳媒介，以恐怖的速度传播。微软具有绝对优势的市场占有率，结合了操作系统和浏览器内许多大漏洞，使得 Windows 成为大多数因特网病毒滋生的温床。

事实上，Windows 内喧闹的安全漏洞已经造成一个新的安全威胁，称为广告插件 (adware)，5 年前还没有这种东西，今天我却已经在 Google 上面找到 600 万个相关项目。你大概也知道，广告插件会把 Internet Explorer 绑住，前往某些特定的网站，也会记录并学习用户的行为。大多数的分析师相信，广告插件已经取代病毒，成为最新的安全威胁。因为这些恶意软件传播得又广又快，常常会将安全设定降低，造成更严重的攻击事件。

C 和 C++ 也有很大的安全隐患，C++ 应用程序可以访问操作系统 API，不受限制地访问内存空间的任何位置。许多版本的 Windows 操作系统无法保护应用程序免于被其他程序入侵。今日因特网已经是最终的传递工具，使用 ActiveX 这样的技术，你可能很快地陷入巨大的危机中。我们的机器中有许多敏感的数据，不可以外流，所以我们不能忽视这样的威胁。

Java 疗法

虚拟机的存在，让 Java 的设计者有机会在不安全的平台上建立一个安全的基础。这个优点主要是来自于受限的沙箱 (sandbox)：

- 既然 Java 从头到尾都是 Sun 设计的，所以根本不需要担心修补旧系统的安全漏洞，就像你会在 Unix 和 Windows 上面看到的漏洞。（这些虽然是操作系统，但毕竟操作系统是应用程序的平台。）
- Java（也就是 JVM 的语言）是和因特网一起成长的，所以 Java 创造者可以知道什么攻击可能会发生，而提前因应。
- Java 在很底层的地方具有安全管理员，用来执行安全政策，控制对于底层系统资源的访问。
- JVM 提供受限的沙箱，让 Java 应用程序能在沙箱中执行，恶意或有错误的应用程序在沙箱中是无法造成伤害的，不像 C++ 程序那样。
- 因为没有指针运算，且 Java 有运行时的强类型检查，所以 JVM 就能准确掌握引用（reference）所指的地方。JVM 可以限制应用程序对于内存的访问，大多数的 Java 安全攻击都是试图先打倒类型安全。

相对来说，Java 安全很少被破坏，这对 Java 的创立者来说是很大的恭维。对于病毒、广告软件或安全攻击来说，Java 都是一个很难以生存的环境，倒是下面的操作系统就很容易了。

向前走

虚拟机的想法已经被大家接受。中间的虚拟机将一些基本问题（像可移植性、安全与部署）从难题变成日常事务。如果 JVM 接受动态语言，在可见的未来它将会成为部署平台的选择，如果 JVM 不接受动态语言，新的虚拟机将会出现。

但是可移植的难度相当高，这已经被证实了。Jython，是一个源自 Python 的动态语言，在 JVM 上运行，但是并未如同预期般流行，主要是因为它速度不够快，另一方面也因为 Python 社区没有拥抱它。另一个计划要在 JVM 上面实现 Ruby，称为 JRuby，目前也遇到和 Jython 相同的问题。但许多分析师预测，即使在最后一个 Java 开发者落寞地写下最后一行程序之后，JVM 还会继续在这个世界上存活很久。

我相信下一个主要的编程语言将会更动态，很明显地，新的动态语言也将需要虚拟机的优点，如果安全书籍和 Windows 替代品的销售不佳，这就表示安全不像我们所认定的那么重要，尽管如此，替代品还是可以有 Java 虚拟

机的这些优点的。下一个主要的接班者或许不像Java的安全功夫这么好，因为大部分语言的设计者不会一开始就考虑到安全。在我们修改好我们程序、思想、操作系统中的基本漏洞之前，建立在语言上的安全也就没有太大的关系。

因特网

C是从系统语言演化而来的，主要目的是用来建立操作系统。C是系统编程语言。C和后继的C++直到最近才开始进入企业领域。Java和C++不同，移动运算（mobile computing）是Java最早的目标之一，而且很快就发展成适用于企业的因特网应用程序。你可以在四个地方很轻易地看到Sun的意图：

- Java具有方便的特点，可以让应用编程更容易。Java具有垃圾收集以及内存管理功能，这使得应用程序开发者不需要担心这些问题。Java很重视字符串，所以程序员不需要担心移动字节（byte）的问题。如果是系统语言，可能会有比较多的控制。
- Java的企业运算版本是以因特网为核心的。Java建立数个链接库，大大简化了企业计算，此成长过程中的语言总是以因特网为发展焦点。早期的API就已经可以做许多因特网的事，包括TCP/IP socket（网络通信协议）以及applet框架（将浏览器当成应用程序的平台）。
- Java之父致力于简化，付出的代价是丧失低层的灵活性。例如，虽然C++可以接触到系统内的每一个byte，却因此造成C++应用族群必须和指针运算纠缠。
- 很早的时候，Java的目标是在移动应用程序，但是Sun看到可以颠覆微软的机会，Sun把握机会，将Java的焦点扩展，进入到因特网领域。

请记得：client/server运算让应用程序的部署很困难，由数千个Windows客户端和数百个服务器所组成的分布式网络，虽然比大型计算机便宜，但是管理成本很高。在20世纪90年代末期，公司幻想将client/server运算改成以因特网应用为标准的运算，称为intranet（内部网络），intranet只存在于公司内部。当Sun开始将Java嵌入到Netscape Navigator中时，这样的愿望似乎就有可能成真了。

前后一致的演变愿望

因特网的终极目标是：给所有的用户一个单一的应用平台（也就是浏览器），让他们可以在此平台上运行应用程序。一开始，这些应用程序以 applet 的形式存在，这其实是一个简单的想法：把 JVM 嵌入浏览器中，让用户下载 Java applet bytecode，把 applet 当成另一种新的数据类型（MIME）。浏览器就只是把程序交给 JVM 执行。一开始，许多公司很成功地部署他们的 applet，后来，applet 就不再受欢迎了。在我写本书过程中所进行的会谈，许多人对 applet 失败的原因提出如下看法：

- 部署太难了，applet 开发者发现他们用一个麻烦换来另一个麻烦：部署操作系统升级以及客户应用程序的麻烦没了，但是部署浏览器升级以及虚拟机同步化的问题却来了。
- 编程很难，applet 开发者不容易了解外面的编程模型，以及如何让 applet 和网页整合得天衣无缝。Applet 如果能做得好，效果会很壮观，但是真正做得好的 applet 少之又少。
- Netscape JVM 的 bug 很多。有人说，正是因为 Netscape JVM 的 bug 这么多，所以才会让 applet 一败涂地。如果 Netscape 对于可插入的虚拟机支持得好一点，applet 成功的机会可能会大一点。

不管原因为何，applet 已经失败了，但是在这个时候 Java 却长成一头让人吃惊的猛兽。在 Netscape 的大厅中，服务器端的 Java 出现了。Servlet（这个名词其实还是来自 O'Reilly）让服务器端驱动的因特网应用程序开始被许多开发者所采用。Sun 在这个时候很快地通过标准和开放源码的实现引擎（Tomcat）将 Servlet 推广出去。Servlet 解决了许多 CGI 应用程序的问题，也让企业开发者有全新的方式让桌面可以运行应用程序。这种应用程序，在视觉上还是由浏览器所进行，但是逻辑上则是搬到服务器上进行的。

服务器可以建立动态的内容，并将内容递送给客户端。讽刺的是，这种“新”模型其实是大型计算机时代绿屏的翻版模型。但是的确有一些微妙改进后的优点：

- 虽然绿屏单调又陈旧，但是因特网却很酷很新。用户知道如何使用它们，因为家家户户都有计算机和浏览器，他们喜欢使用新系统，就像开发者喜欢建立新系统一样。

- 浏览器缺乏键盘驱动界面的生产力,但是比较方便学会使用。用户界面提供了一些细节上的加强,像通过链接(link)来进行向导,而不是通过菜单(menu)。
- 服务器端的开发环境,比大型计算机时代的更方便。开发环境(通常是Windows)更便宜。

Java 客户端开发慢慢停滞下来。Swing 长期以来一直被批评不方便用户使用,但其实真正的限制是学习曲线和开发者的生产力,即使是做一件小事情,都会把开发者整得死去活来。

但是 Java 的重心很快地就完全移到服务器端了,直到今天还是如此。Java ServerPages(JSP)继续了这样的演变,让传统的设计师能够在 Web 应用程序的开发中扮演适当的角色。更多模块的设计,加上了 JSP 标签库(tag library)、端口组件(称为 *portlet*)以及 MVC 框架,继续了这场演变。Java 用户界面技术中,没有任何一者的成功可以和 servlet 在 Web 应用程序上的成功相提并论。

向前走

因特网应用程序的愿望尚未完全完成。Google 现在正利用 Ajax 在做实验,这可以让用户更好地使用,因为 JavaScript 和 XML 可以和后端的服务器沟通而做一些事。像 Google Map 这样的应用程序,让我们知道利用 JavaScript 建立丰富的应用程序,让 JavaScript 主动作为 client 和 server 之间的沟通桥梁,这是可行的。我们很渴望新的用户界面技术,让我们可以同时具有部署 servlet 的便利,以及 applet 画面上的绚丽。所有的浏览器都支持 JavaScript,但是它并未精心规划,脚本编程语言(scripting language)容易导致一些问题,而且在不同浏览器上的执行结果可能会不一样。

我的直觉告诉我,浏览器不是最终的答案,但是具有很多相同的特征。你可以想象浏览器和动态语言之间具有更好的结合,让用户具有更丰富的使用体验。对我来说,有一件事很清楚,Java 社区在丰富的客户端方面并不成功,主流的丰富客户端技术,像 Swing 和 Standard Widget Toolkit (SWT),都让程序员停留在很低的层面上,这方面,微软和苹果具有更好的框架。虽然

Java 开发 Web 应用程序时很好，但是越来越多的用户希望有更丰富的使用经验，就像他们希望网络速度更快一样。

企业集成

随着Java将重点放到客户端和服务端（图 3-2），企业集成变得更重要了。在这里，IBM、Oracle、BEA、Borland、Sun 以及其他的公司都竞相耕耘，他们让Java连接到数据库、事务引擎、信息系统以及其他的企业系统。厂商的结合与支持让合作进入标准化，更多有用的结合激增，前所未见。Java 提供了一个绝佳的整合平台。因为背后有如此多重量级厂商，Java变成了非常安全的平台。

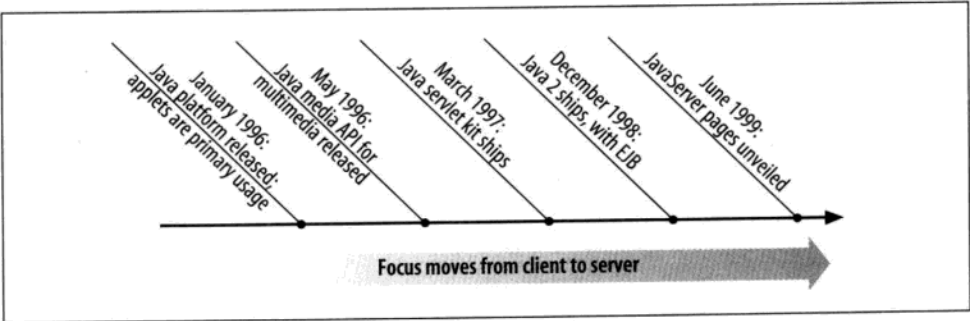


图 3-2：一段时间以后，Java 的焦点从客户端转移到服务器上

Java在企业集成项目上，是一个很好的语言，因为它具有很多框架，可以解决许多重要的问题，像分布式事务处理。静态类型对于大型数据量问题比较重要，因为许多问题都难以测试，bug 的成本就会提高。相对于 C++ 而言，在企业领域，软件开发速度比运行速度重要，因为大部分的运行时间都是在多个企业事务、数据库、网络链接库内进行的。

向前走

Java可以和任何你觉得重要的企业系统连接。除了整合，Java也提供了绝佳的便利，可以让面向对象模型对应到关系型数据库。你可以让事务进行分布式合作，管理有着规则引擎（rule engine）和 workflow（workflow）的大系统。你可以利用 JNI（Java Native Interface）让 Java 延伸到 C++ 中，也可

以让 Java 和 Web service 整合得很好。我们已经有过许多遥控 (remoting) 方案, 从 20 世纪 90 年代的 CORBA 标准, 到 Java 专属的 RMI, 或许你可能决定使用许多轻量级的 HTTP 遥控或 Web service。不同的标准以及免费的框架可以帮助你为你的业务对象 (business object) 管理服务、进行文字搜索、写游戏, 甚至写移动应用程序。

Java 的确征服了许多问题, 但是 Java 也有缺陷。容易的企业问题已经被解决了, 所以主要的厂商开始将时间花在困难的问题上。随着 Java 渐渐往复杂的问题移动, 简单的问题也就越来越不容易解决了。EJB、XML 的急速膨胀以及大量的 Web service 堆栈是三个例子, 它们见证了复杂度的增加。

最后, Java 牺牲了它的主要根基, 生产力和趣味不见了, 换来的是乏味和慢速, 但威力强大。许多应用程序不需要这些多余的企业功能。我想大概有一半的商业应用程序牵涉到 Web 前端和古老的关系型数据库。在这个空间内的挑战者不需要发动正面攻击, 只需要针对某些问题提供比 Java 更有生产力的解决方案即可。看看 Ruby on Rails 这个框架, 它让 Web 用户界面和数据库的开发都相当容易。今天, 这可是很重大的问题, 这可以让你抓住并共享信息, 不管使用在什么领域。



Erik Hatcher: Java 的成功

《Java Development with Ant》的作者之一

Erik 是《Java Development with Ant》和《Lucene in Action》(<http://lucenebook.com/>) 的作者, 他投身在数个开放源码的项目上, 主要是 Apache 软件基金会, 他也是 Apache 的会员。Erik 曾经和 Bruce 一起泛舟, 差点就没能活下来告诉我们精彩的故事。

你最喜欢 Java 的什么地方?

EH: 它有许多内置的能力, 有许多外部 (对我来说就是开放源码) 的链接库。

不喜欢什么地方?

EH: 我很同情新进入 Java “平台” 的人。我们都知道 Java 语言相当容易学习, 这许多人觉得 Java 不难, 但是光靠语言和虚拟机是什么都做不到的, 事实上, 后面困难的学习路程还很长。

举例来说，CLASSPATH让我们都感到麻烦，甚至“专家”都觉得麻烦。想要真正做一些有用的事，要学会相当多的东西：Ant、servlet容器、JMS、JDBC等。光是想到有这么东西要学，就让我觉得害怕，我过去5年的生命就耗在这里了。

Java 如何阻碍你？

EH: 我不觉得受到阻碍，但我常常觉得我得花更多时间才能完成一件事。

有没有什么因素会让你想远离Java或.NET？

EH: 如果Ruby有组件向导的Web框架，具有Ajax兼容的组件，而且有人把Lucene移植到Ruby上，我就可以把我目前的项目完全用Ruby来做，我希望这些在今年就会发生！

社区

社区力量正是Java皇冠上的珠宝。换句话说，Java的市场占有率，让它成为500磅（1磅约等于0.45公斤）的猩猩，想睡哪儿，就睡哪儿。Java的社区很大，也很不一样：

- 各家厂商一致支持Java。虽然Sun是发明者，但IBM大概是支撑Java的最大力量。
- 企业开发者使用Java做所有的事。移动计算平台、Web应用程序语言、中间件（middleware）的系统语言，以及彼此混杂的应用。
- 编程爱好者一起投入开放源码项目。Java曾经是开放源码社区的害群之马，现在却变成开放源码社区主要的玩家。

对于企业计算，标准也是相当重要的角色。从一开始，核心的Java厂商们就合作建立标准。Servlet、EJB、JSP是这10年内三个最重要的标准。为了不让让人觉得Java越来越多专属化技术，他们建立了一个社区参与标准制定的过程JCP。

Java有许多特性，我们许多人都视为理所当然。Java开发者很容易找到，没有人会因为选择Java当方案而造成日后被责备。Java相当成熟，适合项目委外进行，到处都可以学Java，到处都是Java组件，你有许多不同的实践

版本可用 (IBM、Sun 等), 你可以免费做许多事……我还可以继续说下去, 但总而言之, 正是因为 Java 社区的力量, 企业开发变得安全了。

开放源码的重要性

大家都希望建立垄断地位, 因为可以主导市场, 且利益丰厚, 但是 Java 社区背后的力量却反其道而行。社区中的开放源码软件, 逐渐说明了 Java 的体验。

一开始, 开放源码通过 Tomcat 让 servlet 发生革命。然后, 我们学会用 Ant 来建立项目, 用 JUnit 来测试, 用 Cruise Control 这类产品来进行不断的整合。之后, Struts 软件改变了我们组织 Web 用户界面的方式, 而 Hibernate 让我们轻易进行透明的持久性 (persistence)。你可以说, 最具有创新的产品都是来自开放源码项目, 在许多领域:

- *Lucene* 让我们可以进行有效的文字搜索。
- *Tapestry* 是最具潜力的 Struts 的接班者。
- *Spring* (而非 EJB) 让服务被透明地采用。利用 Spring, 你可以用声明的方式为 POJO 加上服务 (例如安全、事务、遥控)。
- *Hibernate* 是使用人数最多的透明持久机制。

你甚至可以看到开放源码软件对于业界产生的冲击。EJB 3.0 规范强制厂商提供较简单的 API 给 POJO, 而不是毫无作为, 继续从现有的 EJB 2.x 服务中收进大笔金钱。Ant 和 JUnit 改变了开发环境的演变, JBoss 建立了一个完全开放源码的应用服务器, 并正在改变软件公司的模型。

现在, 数家公司使用开放源码社区以控制某些重要的技术。例如, 在 IDE 市场多年的惨败之后, IBM 把 IDE 技术开放出来, 变成 Eclipse, 看看现在的差别有多大:

- 虽然 IBM 在营销上比以前花更少的钱, 但是市场占有率却节节攀升。
- 开放源码族群善变的心, 现在已经被 IBM 虏获了。
- 开放源码开发者积极投入贡献 Eclipse 项目, 并免费提供许多 plug-in (插件)。

- IBM 仍然对 IDE 维护某些控制，更重要的，这让它的竞争对手无法通过 IDE 来控制 Java 的任何部分。

我不是说开放源码族群很容易被操作或控制。它本身就是一股力量，如果你正开始经营一家新的软件公司，或者正在管理一家成熟的软件公司，你绝不可以忽视开放源码的冲击。

向前走

在 Java 出现的时候，社区扮演关键的角色。如果没有成功地引导 C++ 社区投入，Java 的步伐会走得更蹒跚，甚至可能不会吸引核心厂商的加入。没有开放源码社区，许多 Java 重要的创新也不会发生，下一个语言也就不会有强劲的对手可以挑战。

如果有新语言想挑战 Java，就必须能迅速地拥抱群众，需要有一种催化剂，像 Netscape 浏览器内的 applet。下一个成功的语言可能也需要结合开放源码社区的力量，才能像 Java 一样享受丰富多变的长寿人生。最后，下一个语言需要具备政治安全（Ruby 是政治安全的语言，C# 不是），标准可以很快出现，不需要经过持续的争吵而阻碍进步。

打破迷思

这些技术很快兴起，变得重要，让人相当崇拜 Java。事实上，许多媒体的 Java 拥护者使用 Java 锐不可当的成功来护卫 Java 的缺失，从 EJB 到静态类型。他们完全相信 Java 一定是百分之百的完美，才能如此成功。这种信念是危险的，事实上，下列许多的迷思都可能会导致 Java 的败亡。

迷思 1：Java 的领导牢不可破

Java 的确在营销上占有相当大的优势，但是暴风雨可能很快就会来到，可能会摧毁现有的风景，只剩下一些残存的东西。技术新陈代谢的速度可能比你想象的更快：

- 看看录音工业，黑胶唱片已经结束了，看起来 CD 也快步上后尘了。随身听很快流行起来，被淘汰的速度也很惊人。结合 iPod 和收音机的机器会很快走进家庭。

- 某些正在兴起的第三世界国家,展现跳跃性的技术采用方式,跳过传统的电话直接使用手机。
- 数码照片将胶卷逼至市场的墙角。
- 你再也看不到 5.25 寸的软盘,而 3.5 寸的软盘也越来越难看到了。
- 让我们回到我们的领域,Visual Basic 可能已经到了生命的尽头。事实证明,迁移到 .NET 之举,对于微软和 Visual Basic 社区来说,都是很痛苦的事。

事实上,微软的 .NET 环境现在正在威胁 Java。某些新出现的编程语言已经让某些聪明的 Java 独立顾问心动且投入了,Java 的一些限制也让一些人主动离开。许多语言都曾有过辉煌的日子,然后回归黯淡,到最后,Java 也会如此。

迷思 2: Java 是绝佳的应用程序语言

Java 的成功,并非因为它是最佳的应用程序编程语言,它甚至算不上是特别好的应用程序编程语言,Smalltalk 和 Python 的生产力更高,Visual Basic 更简单。Java 的成功是因为它能够抓住现有的 C++ 族群,让他们能够写出因特网程序。这个社区,而非这个语言,正是 Java 最重要的资产。某些领导 Java 革命的力量,可能最终会将 Java 带向死亡。Java 通过 C++ 的遗产吸引 C++ 族群,让 Java 社区迅速成长,但这在某些方面也限制了 Java,这方面在第四章会详细探讨。

Java 爆发式的成功,让 Sun 在语言层次上变得相当保守。这让人感到疑惑。例如,我们将会看到支持 aspect-oriented programming (AOP) 的语言,许多人也认为应该这么做。“维护向下兼容”的决定,让 Java 不能快速演变,以追上竞争者的脚步。也就是说,和竞争者比起来,Java 的演变很慢。

迷思 3: Java 是最具有生产力的语言

当你把 Java 和 C++ 比较时,Java 的确具有很高的生产力,那是通过阴雨天的窗户来看 Java。但是 Java 不是应用程序语言,像 C++ 以前那样。任何用过 Basic 或 Smalltalk 的人都能够告诉你“快速完成,快速响应”的重要性。Java 需要编译,使用静态类型,因此丧失实时解释的能力以及快速响应的能

力。静态类型对于减少运行时错误会有帮助，但是生产力不佳。Java的字符串处理能力很受限制。Java的语法缺乏 *closure* 和程序代码块 (code block) 的特色，不能把程序当参数传递。我再说一次，Java赢了，因为它比我们大部分人之前使用的语言具有更高的生产力，它当时有足够的生产力，但不见得会一直这样下去。

推论 3：所有的语言都一样

Java能够让人离开C++，是因为它提供了明显的改进，像垃圾收集、虚拟机以及更好的 OOP。用Java写出来的程序总是比用C++写出来的程序短。但是当你用Java和其他语言比较时，你会发现历史又重演了。Lisp和Haskell这类语言提供了较高的抽象以及完全不同的思维方式。像Ruby这样的语言则是相当动态，且通过元编程 (metaprogramming) 提供较好的访问语言建立块的方式。像code block以及延续 (continuation) 都会影响到你组织以及运用链接库的方法，而Java是两种都不支持。用Java，你常常得做更多事才能得到相同的结果。

迷思 4：商业利益驱动大部分 Java 的创新

虽然业界有一些显著的创新，但是许多创新，像轻量级容器 (Spring)、Web应用模型 (Struts与Tapestry) 以及透明持久性 (Hibernate) 都是开放源码社区的功劳，正是这些想法将Java的边界往外扩展。事实上，业界的目标反倒常常会束缚快速的革新：

- 将许多产品整合起来需要花许多时间，那就是为什么每次 WebSphere 推出新版本都要间隔很久。
- 如果软件是用来赚钱的话，建立与测试该软件就需要花许多时间，因此和开放源码竞争居于劣势。
- 建立标准需要时间，采用标准需要更多时间。
- JCP试图使用专家的知识来创立全新的标准，而不是将过去成功的实践经验予以标准化。

越来越多客户寻求开放源码软件来帮他们解决重要的问题，因为他们也是很成功的创新。你已经见证开放源码框架兴起成为主要力量，下一个受欢迎的编程语言可能是从开放源码社区中出现，也说不定。

迷思 5：接班者的来源通常相近

最后几个主要的编程语言来源很相近，最后两个甚至来自非主流的公司。C 来自贝尔实验室 (Bell Labs)，这是一家通信公司。Java 则是来自 Sun，这是一家硬件公司。下一个流行的语言可能也将会来自不一样的公司。C# 不算，它基本上是 Java 的复制品。Visual Basic 的公司本来也是很小的，这家小公司成立在 Pacific Northwest 的车库内，经费拮据，叫做微软。

Java 只是一个编程语言，就和所有的语言一样，它会有辉煌的时刻、领先的时刻，但也会开始走下坡，而被取代。问题不在是否会发生，而是何时发生。

往前看

到目前为止，我试着描绘出 Java 成功的全部原因。我的事业都是建立在 Java 之上的，而且还很不可思议地成功了，为此我感谢 Java 之父。尽管如此，我还是相信，Java 不是牢不可破的力量，虽然许多人的意见和我相左。我认为 Java 正在失去那些将它推往成功的开发者，那些人喜欢简单技术、讨厌复杂技术。而且，有些吸引 C++ 族群改用 Java 的因素（像基本类型、静态类型以及类似 C++ 的语法）都已经开始呈现负面作用了。反正，Java 已经从妥协中有所收获，下一章，我们会谈谈妥协的代价。

打破玻璃

我不确定我何时感觉到后面的泛舟者有麻烦，我们的心思被周围的混乱所占据了。Barton Creek（河流名）正处于洪流状态，遭受到猛烈的攻击，我们共 10 个泛舟者，需要保持安全距离。我们每三个人编成一个小队，好让每个群体都能留意彼此。那天一开始的场面就不太好，一个别的社团的消防员（它是一个划船专家）愚昧地独自划船，死在相同的溪流里。现在，我们也有自己的问题了。

在我们划了一个小时后，为了要让小组在一起，并规划下一个危险要如何度过时，我们被牵引到巨大的漩涡中。事实上，河岸很危险，树木可以让你翻船，不可预期的水流堆积与倾泻，而河流主要动线却是相当直观。我翻过一次，但是很容易就翻回来了。但是我们穿过几个危险的地方，如果你够倒霉，或者自大而不愿意绕过它们，你就会栽在这些地方。另一个团的三个人失踪了，而我们却无法在河流中往回走。我们等了两个小时，但他们还是杳无音讯。我们等了又等，直到天黑，然后我们重回到河流中。事后，我们才知道后面小队的成员之一，竟然挑战一个洞，我们没有人会勇敢地或愚笨地挑战的一个洞，导致必须用直升机来救他。我再也不会去 Barton Creek 涨水的时候去泛舟了。

我已经发展出对于麻烦事的第六感，不管是对河流或工作。在工作上，当技术变得不对劲或者危险时，我可以感觉得出来，然后我就会把我的顾客带离危险。我现在就意识到 Java 的危险，它已经变得太困难而难以驾驭了，不管

是演变还是革命都要开始解决这些麻烦。本章中，我会介绍一些基本的问题。

Java 的新工作描述

到目前为止，我试着告诉大家Java一直都是通用的编程语言，其语法和核心社区都是来自 C++ 系统语言。还有，我也认为大多数 Java 早期的应用将焦点放在用户界面上。你可以很快地下载 Java 程序并运行。

后来 Java 进入服务器端，成为核心的服务器端开发语言。Java 在企业开发上的负载逐渐增加，从具有分布式事务的对象关系映射，到具有 XML 绑定的 SOA 信息递送。我们用 Java 来做的事情一直在改变，这个语言相当灵活，所以到目前都可以应对这些挑战。

但是这些额外的力量却提高了复杂度。想要很快学会语言的人、想要解决一个简单问题的人、想有很高创造力的新创公司，他们又该如何？当竞争的压力让我们不断地将时程缩短，一个通用的 Java 已经不够使用。在某些地方，相当不适合使用 Java。让我们详细看看我们要求 Java 做什么。

典型的需求

如果 Java 过时了，我想它将会在某些地方被取代。在某些地方 Java 很好用，在某些地方却不行。

- Java 变成撰写中间件 (middleware) 不可或缺的技术，中间件是一种位于应用程序和操作系统之间的系统软件。Java 的主要链接库、性能、可移植性以及普遍性，使得它相当适合于中间件，而且也会继续下去。
- 对 servlet 和一般的 Web 编程来说，Java 需要更快的响应周期，需要在字符串管理上做得更好一点，PHP 在这方面更有生产力。Web 编程一团混乱，原因有许多，Java 不是唯一的原因，但是 Java 的确是在“最简单的”和“一般的”Web 应用程序上没有处理得很好。
- 对于 XML 处理来说，有比 Java 更好的方法。我认为 Java 过度依赖 XML 是问题的一部分，而且 Java 语言本身并不是很适合用来处理 XML。

XML 需要字符串的解析和处理，而 Java 在这方面实在太冗长。例如，Ruby 的 XML 链接库就比 Java 更好用，而且处理大部分的工作都更快。有些其他的语言对 XML 有更好的支持，而且只会越来越好。

- 对于大型企业项目来说，需要一些功能，例如：横跨多个资源的分布式事务、对于旧系统的整合，以及依赖利基（niche）链接库的程序代码。Java 一应俱全的链接库以及随处可得的 Java 开发者让企业项目相当适合用 Java 来做。它会持续在这里扮演一段时间的角色。但是，小心，大多数的企业项目都是比较小的项目，如果语言能更具有生产力会更好。

不要看整个 Java，让我们将焦点集中一些，看看最典型的 Java 工作需求。Java 最常见的工作是，用 Web 用户界面来访问一个庞大的关系型数据库。身为一个顾问，这类工作是最常见的，没有别的工作可以相比较。

我发现我正在牵引 Java 进入一个较小的利基市场，也是它目前所占据的市场。我的确认为这是有原因的。一开始，Java 就是由系统语言转变而来。那一串长长的链接库列表扩展了这一切，程序员相当多，这使得它变成大型企业应用程序的必要选择。但是 Java 从来就不是真正被广泛使用的应用语言，虽然我们多数人今天都是这么使用它。

学习曲线

如果你专注于开发在关系型数据库上的基于 Web 前端，Java 框架的设计者已经花了 8 年的时间不断地解决此问题。我必须承认，从 JSP 以后，Java 在这方面一直都没有越来越好，看看最早的 servlet API：

```
public class HiMom extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        response.getWriter().println("<HTML>\nHi, Mom\n</HTML>");
    }
}
```

真的，这种编程风格造成很丑的程序代码，几乎无法维护。这让外观逻辑和业务逻辑纠葛不清。但是，这很容易理解。在 Tomcat 第一版发布之后，只需要几分钟的设定和不到 10 行程序代码，你就可以写出“Hello, World” servlet。

现在相同的应用程序牵涉到更多的作用，在最新版的 Tomcat 上，你不能只是写一个 servlet，你必须写一个部署描述符，并将其封装成一个标准的 WAR 文件。这意味着你必须花更多时间学习 Tomcat、servlet 规范和 XML。结果是，Tomcat 的“快速指南”文档从几页变成几十页。

你可能不认为 Tomcat 学习曲线的大量增加有什么严重的，你可能愿意在 Tomcat 上投资，因为它是核心技术。问题是，脚步没有随着 servlet API 而停歇，想要建立一个典型的 Java 程序，你必须做的事比 5 年前多许多：

- 你必须了解 Ant，我们用它来建立与部署 Web 应用程序。
- 然后，你需要了解 Tapestry 或者 Struts，或者其他的 Web MVC 框架，来帮助你组织程序代码。
- 我们其中的大多数人，都会学习对象关系映射，例如 Hibernate。虽然这可以降低你的持久性 (persistence) 负担，但是学习曲线并不算太容易。
- 你大概需要像 Spring 这样的框架帮助你组织应用程序的资源，让一切都可以测试。
- 你将会需要一些教育，学习如何使用这些工具，有效地整合这些工具。

我的客户从别的语言移到 Java，当他们看到我建议五周的教育训练时，他们吓坏了，而五周还只是学个皮毛罢了。对他们来说，Java 不再是一个可亲近的语言。

用 Java 开发典型的应用程序

Java 的确改进了典型应用程序的某些方面。如果你有相当常态化的关系型数据库，相当适合对象模型，你就可以好好地对应它们，这比以前方便，因为现在有了对象关系映射技术（像 Hibernate 和 JDO）。利用 Struts 以及更好的 Tapestry，你可以更方便地将业务逻辑从外观逻辑中切割开来。通过像 Spring 这样的框架，你可以将服务（像安全和分布式事务）加入到任何 Java 对象中。

但是如果你真的回归到核心问题，关系型数据库上的基于 Web 的用户界面，你要学的东西比 5 年前多出许多。而且你必须更努力，才成达到相同的效果。

大部分附加的价值都是旁枝末节，无关紧要。我要说的是，当这些高级框架将 Java 语言带离当初让 Java 受欢迎的基础。当这样的事情发生时，就表示 Java 的利基市场是大型企业开发。

敏捷过程

当典型 Java 应用程序的需求维持相对静态的时候，典型的开发过程正在发生根本的改变。虽然许多 Java 程序员不会说他们正在用敏捷方法 (agile method)，像 SCRUM 或极限编程 (Extreme Programming, XP)，但是越来越多的人正开始采用敏捷编程的观念：

简单

敏捷方法建议你使用最简单的可行技术。像 Spring 这样简单的框架取代了像 EJB 这样复杂的框架，这样的变化越来越多。

自动化的单元测试

我们正在“测试复兴”时代，JUnit 框架以及敏捷过程带头照亮这个时代。在我出席过的研讨会上，“测试为先的开发”课程越来越受到欢迎，这显示出现在测试已经变得相当重要、相当普遍了。

缩短的周期

缩短的进度表和较好的顾客响应整合，使得产品开发周期缩短，且每个周期内的反复时间也缩短。

Java 的开发过程

Java 的社区和工具提供敏捷开发的绝佳支持，但是有一个问题，Java 语言不是很适合敏捷开发。Java 不是最简单的语言，对于极短周期的开发也不适合。如果你不清楚这两点，当你读完本书之后，你就会了解。其他的语言可以让你很快改完程序，很快进入下一阶段，不需要经历复杂的编译部署周期。其他的语言，有更具表达力的语法以及其他的框架，可以让你的生产力更高，抽象程度更高。

即使我们了解 Java 不是最敏捷的语言，那些使用其他动态语言的人，正使用敏捷技术（像自动测试）以隔离他们免于受到一些问题的影响（例如编程友

善程度或者异常策略)。Java的创立者相信在编译时期抓住潜在的bug是比较好的。他们没有想到“静态类型”以及“强调检查异常”是要付出代价的。

如果我们要根据当今受重视的开发方法来选择语言,Java不太可能会成为我们的选择,因为敏捷开发者所倡导的守则变得相当突出,Java语言将会受到日渐增加的压力。

基本的 Java 限制

我对于一般项目的描述是这样的。一般团队建立或移植应用程序,此应用程序前端是Web应用,后端是关系型数据库,可能协同一些其他服务。此团队可能渐渐采用敏捷守则,也想进行单元测试。此团队通常的工作期限很短,压力很大。还有其他比Java更动态的技术可以用,对于这一类的项目,在这样的环境下,Java不是我首选的语言:

- 许多框架设计用来简化Java项目开发,的确让Java开发者更具生产力,但是也让学习曲线增加,初学Java者难以负荷。
- 编译期异常与类型检查可以提高安全性,但是让时间和语法的成本增加。
- Java不适合表现结构性的数据,这使得Java过度依赖XML,因此造成额外的复杂度和体积暴增。
- Java的许多折中,像基本类型(primitive),让Java更不好学习,更不好写。
- Java比C++动态,但是远比不上Smalltalk以及Ruby的动态。Java开发者正发现元编程(metaprogramming),但是他们无法用够快的速度执行这样的想法。
- Java比其他直译式的语言、动态的语言,编译和部署周期更长。

上面的每一点都已经很严重了,更何况还有这么多点在一起。对许多开发者来说,Java的生产力变得很差。



Steve Yegge: Java 的限制

语言专家及 Wyvern 创立者

Steve Yegge 是华盛顿大学的研究生,他花了5年的时间在汇编语言编程上(Geoworks公司),当了6年以上的Amazon.com 软件开发经理。Steve 总是能够找时间设计、实践、维护一个巨大的多人游戏,称为 Wyvern (<http://www.cabochon.com/>),他可是为此游戏写了 50 万行的 Java 和 Python 程序代码。

你的 Java 经历如何?

SY: 从 1996 年底到 2003 年中,我是 Java 社区的正式成员。我使用 Java 建立一个很酷的、多人的、用户可扩展的在线游戏。Java 让我走得很远,我爱它爱了 7 年。

你何时开始寻找其他语言的?

SY: 我遇到生产力的瓶颈时。当我的程序代码越来越多,我的创新脚步却慢了下来,直到最后,做一件事花的时间是我认为合理的时间的数倍,于是我停止开发 6 个月,并开始深入探讨问题出在哪里。出乎我意料之外的,问题就在 Java 上,我很不乐意见到这样的结果,毕竟我已经在 Java 上面投资了这么多。AOP 可以帮助一些(尽管门槛颇高),但是根本就不够,没有其他的東西可以帮得上忙,我需要的是一个新的语言。

Java 是如何让你生产力不佳的?

SY: 首先,Java 的抽象工具相当贫瘠,抽象并非 Java 第一类的功能。没有参考参数(reference parameter);没有关键字;没有默认参数;没有析构绑定(destructuring bind)或者平行赋值(parallel assignment);没有办法有效率地返回多个值;没有延续(continuation);没有用户定义的运算符(operator);没有生成器;没有程序代码子句(closure);没有数组(tuple)……我还可以说更多。Java 就像一个嘴巴掉了 25 颗牙齿。

再者,Java 完全是无法扩展的,它不能成长。没有元编程(metaprogramming)、没有宏、没有模板、没有任何东西可以给你语法的抽象。所以,Java 是不能压缩的。Java 的程序代码总是填满一些重复的程序代码,看起来就像剪贴

的，而且还不能重构这些部分。Java 程序代码和 API 总是一大堆（而且看起来都很怪）。

第三，Java 可以表达程序，但是不能表达数据。你必须使用 property 文件、XML 文件以及其他的方法来定义数据。但是程序和数据的分界相当模糊，例如配置设定。所以，Java 族群只能一个框架上面叠放另一个框架，建立一个巨大的转换串联管道，来弥补 Java 的不足。

第四，Java 的静态类型系统很烂。实际上，所有的静态类型系统都很烂，但是 Java 几乎是最烂的。它将你的通道窄化，限制你的思考能力。Java 程序员必须花好多精力学习在方形洞内插进星状木桩；这就是大部分的设计模式。

第五，Java 具有太多非必要的复杂度。比方说，它现在有四种类型：基本类型 (primitive)、类 (classe)、数组 (array) 以及枚举 (enum)。每种类型都有自己的语法和语意，你必须要学会这些。不只是类型，连 Java 的整个语法都大而无当。这些语法的复杂度没有存在的好理由。

类型

编程语言常常会被争议，强类型（也就是静态类型）的策略是否好处比较多。Java 的策略倾向于尽量使用编译期的检查。让我们用世俗的说法来看编程语言设计的概念，然后你再把 Java 拿来对照一下。当建立一个语言的时候，语言设计者需要很早就回答两个关于类型的问题。

强类型 vs. 弱类型

强类型 (strong typing) 和弱类型 (weak typing) 决定了类型如何被实施或解读。弱类型的语言（像 C），变量可以被轻易地强制或者解读成别的东西。而强类型的语言严格限制兼容的类型以及操作。你大概也知道，Java 是属于强类型的语言。

Ruby、Smalltalk、Python 也都实行强类型，你可能会感到惊讶。许多开发者相信 Smalltalk、Python、Ruby 之所以具有很高的生产力，是因为它们是弱类型的，但这其实是误解，看看下面的 Ruby 例子：

```

irb(main):003:0> i=1
=> 1
irb(main):004:0> puts "Value of i:" + i
TypeError: cannot convert Fixnum into String
    from (irb):4:in `+'
    from (irb):4

```

第一行，未声明的变量*i*被赋值为1，此时，Ruby决定让*i*当作Fixnum类型。当Ruby解译到第三行时，它看到+运算符出现在字符串后面，试着将*i*连接到字符串后面。当然，Ruby不知道如何把整数和字符串连接在一起，所以就抛出一个错误。这是强类型的一个例子。（其实，我有点太过简化这件事，你可以在运行时动态地改变Ruby类和对象的定义，此举造成弱化类型的效果。尽管如此，在强到弱类型的光谱中，Ruby比较靠近强类型。）

类似的状况，有着弱类型特质的语言，可能会将类型替换成兼容的格式，例如C。看下面的例子：

```

int a = 5;
float b = a;

```

在第二行中，C将整型转成浮点型，其他的例子甚至更糟糕。在C++中，()转换运算符没有产生类型安全，所以你可以这么写：

```

Cat *cat;
Dog *dog = (Dog *)cat;

```

这是合法的C++语句，不会报告错误。有着弱类型特质的语言，是不会捕获类型错误的，对某些特定操作，后果会如何，没人能预料。弱类型有时候比较方便，也比较不可预料。如你所见，类型不总是黑与白，它是一个具有高度争议性的议题。对语言专家来说，强类型和弱类型是一种连续状态，某些强类型的语言（像Java），允许用户将对象用cast转成别的类型，但是最强类型的语言则是连cast都不行。较弱的类型则允许（甚至需要）强制（coercion）。最弱的则完全不进行类型检查，不管是在编译期或运行期，像汇编语言就是这一类。

静态与动态类型

至今最有趣的问题是：何时进行强制类型。静态类型将类型绑定到对象以及变量和参数等语言构词上。动态类型是在运行时将类型绑定到对象上的。动态类型对于变量的容器没有任何影响。类型是绑定在对象上，而不是变量

上，因此，容器的类型可以改变。一个不完美的大原则是，静态语言强迫你必须声明变量，但是动态语言不要求你这么 做。

讽刺的是，大多数动态语言也倾向于是强类型的语言。大多数的弱类型语言都是静态的。换句话说，强类型可以是动态或静态，但是弱类型通常是静态的。除了汇编语言，你不容易看到太多弱而动态类型的语言。图 4-1 将编程语言放在两个轴上，Java 有强和静态的类型，我相信这是你知道的，因为你一定有过获得“类型不匹配”错误的经验，如下面的例子：

```
class TypeTest {
    public static void main(String args[]) {

        i = 4;                // Nope!!! Static typing

        int j;
        j = 4.2;              // Nyet!!! Strong typing
    }
}
```

……你得到这样的编译错误信息：

```
TypeTest.java:3: cannot resolve symbol
symbol   : variable i
location: class TypeTest
    i = 4;

TypeTest.java:5: possible loss of precision
found    : double
required: int
    j = 4.2;
```

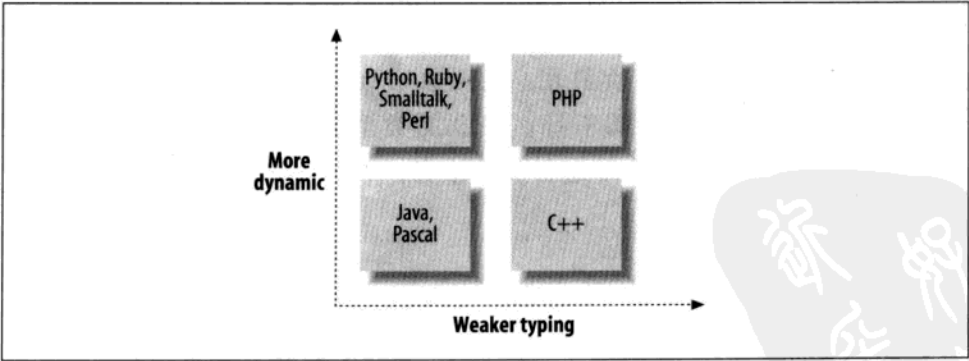


图 4-1：Java 是强类型及静态类型的语言

有时候，这很好。毕竟，在编译期找到错误的成本比在运行时找到错误的成本更低。但是，一般而言，我所认识的动态程序员会说，静态类型会打击生产力。

Syntax

一开始，你能立刻发现Java的语法强制你做更多的事。你必须为所有的变量和参数声明类型，你也需要将兼容但不同类型的对象进行cast转换类型。额外的语法提供了一些价值：让编译器有更多的信息，能够早点找出bug。但也为此付出一些代价，静态类型让你必须工作得更努力，才能达到和动态语言一样的效果，但是你也因为程序代码更长，需要花更多时间理解、维护或加强。很难用使用键盘的时间来证明静态类型让你更有生产力或相反，但是你可以确信静态类型的确会让字符变多、程序代码变长、阅读与维护的负担加剧。

粗糙的程序代码行数并没有绝对的意义，否则，Perl语言使用那些两个或三个字符的快捷方式，不就变成最有生产力的语言了。但是，这还是可以作为参考的指标之一。如果你可以在程序头或尾限制Java额外的程序代码到几行，那么Java的语法就不会是一个大问题，但是你不能这么做。你需要声明参数的类型。每次从集合（collection）中取出对象，你都需要转换类型。而泛型语法的加入，只会让事情更糟糕。

思考过程

某些和类型相关的成本是隐藏的。我相信这样的成本和高级的概念性的工作有关，而和完成的工作无关。我们喜欢先做概念性的工作，再做完成的工作，因为许多程序代码会被丢弃，特别是在早期。在你的程序成形之际，你能够做越来越多的详细工作。你先让预期的使用案例如愿，然后再去处理无关紧要的小事。

Java强制你做事情的顺序相反：让一切先能编译，你必须先彻底处理类型。另外，许多Java的编译器错误，在动态语言中甚至根本不是个问题。

写程序 / 编译周期

动态类型可以派上用场，特别是在你需要实验的时候。请牢记，用静态类型语言，你必须先声明变量。在Java中，这意味着每个应用程序都需要从定义

一个类开始，接下来就如同滚雪球一般。你不能就这样跳进去，执行单行程序，因为编译器的信息不够，所以不能执行。你不能单纯地执行语句，而需要创建一个类，定义类型，然后编译、执行。在 Smalltalk、Lisp、Basic 以及 Ruby 中，你可以直接键入程序。比方说，简单地写个 Fibonacci 序列程序，下面是 Java 版本：

```
class Fib {
    public static void main (String args[]) {
        int x1 = 0;
        int x2 = 1;
        int total = 1;
        for (int i=0; i<10; i++) {
            System.out.println(total);
            total = x1+x2;
            x1 = x2;
            x2 = total;
        }
    }
}
```

共 13 行，41 个词，226 个字符。注意 Java 强迫你声明类，这会让程序代码爆炸。这里还没有算进来。在命令行中，你需要下指令进行保存、编译、执行。Ruby 的版本则如下所示：

```
x1 = 0
x2 = 1
100.times do
    puts x2
    x1, x2 = x2, x1+x2
end
```

共 6 行，16 个词，57 个字符。请注意，这个程序用英文读起来更流畅，但最大的差别在于实验性，你直接输入就可以直接执行。你可以把这一段文字剪贴到 console 窗口。你可以使用命令来重复你所需要的行为。这些优点在 IDE 中也同样可用，而且，如果你需要 100 次的反复，Java 版本就会出问题，因为 int 不够大，但是 Ruby 版本会完全正常运作。

这是一个琐碎的范例，甚至可能不是很公平，毕竟，Java 版本封装起一个完整的类，但是 Ruby 版本则不需要。但是你会发现，随着我们继续下去，更多具有说服力的范例会逐渐出现，特别是在动态、反射风格的编程方面（这是 Java 编程中很重要的部分）。

对于Web和其他的部署步骤,动态语言的范例更具有竞争力,因为你能够在改变之后马上看到结果,不用经过编译、部署,甚至启动 servlet 引擎。Web 编程变得相当容易,改变之后,重新加载,就好了。

从我认识Basic (我高中时候是靠写Basic游戏赚零花钱的) 和Smalltalk (我用来做市场展示) 以来,我就喜欢上快速的开发周期,这真的是动态类型语言的优点。

适用性

如果你大部分的工作都是用Java在写程序,你可能不需要知道你必须跳过这么多火圈,只是为了支持静态类型,但是你确实如此。类型最大的成本来自重构(refactor)。想想一个简单的成员变量类型改变会造成多大的影响,你必须改变 property、getter、setter、每一个类型转换的地方、每一个用它当参数的地方。动态类型语言,将变量或参数类型的绑定延迟,所以你常常不需要进行任何改变,就可以应付一个简单的类型改变。例如,对于Smalltalk而言,你可以很容易地改变类型,如果此新类型支持所有旧类型的信息,那么你就可能可以完全不用改变任何其他地方。

泛型

Java架构师传统上对于确定类型安全方面做得很好,但是有一件事一直困扰他们,当你将一个对象从集合中取出时,你需要对此对象进行类型转换:

```
ArrayList animals = new ArrayList();
animals.add("elephant");
String cat = (String)animals.get(0);
```

编译器没有能力提供编译期的类型安全。你可以把此元素当作任何你需要的类型。为了修改这一点,Java开始支持泛型(generics),用很丑的方式实现泛型。

```
ArrayList<String> animals = new ArrayList<String>();
animals.add("elephant");
String elephant=animals.get(0);
```

将有泛型和没泛型版本的程序相比较,你可能会认为泛型让你避开了转型的必要性,但是实际上不是如此。Java对于泛型的实现方式相当丑陋,叫作类

型消除法 (type erasure)。内部的做法还是让一个 `ArrayList` 负责管理 `Object`, 而不是管理 `String`。当然, 任何需要使用自定义类型来强制分类的链接库, 都必须支持泛型。支持泛型, 看起来有一点丑, 下面是 `List` 的声明, 取自 Java 的 `collection` 包:

```
public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
}
public interface Iterator<E> {
    E next();
    boolean hasNext();
}
```

如果你不是一个静态类型语言的拥护者, 你不会喜欢这些额外的类型检查, 这会让你的负担加重。即使你喜欢泛型, 你大概也不会喜欢Java实现泛型的方式。泛型只是对语法有好处 (syntactic sugar), 而不是真正运行时的保护, 因为JVM对于泛型完全没有概念。在 agiledeveloper.com 一系列的文章中, Venkat Subramaniam 很详细地列举出了许多问题:

- 当你混合进非泛型, 你就失去了安全性。例如, `List notGeneric = genericList;` 的类型安全不会进入 `notGeneric` 中, 即使它被绑定和内存中 `genericList` 相同的 `List`。
- 你不能使用基本类型当作泛型的类型参数或静态字段。
- 不同参数化类型 (像 `ArrayList<String>` 以及 `ArrayList<Book>`) 的实例, 其实属于相同的类型, 都是 `ArrayList`。
- 因为JVM完全不懂泛型, 所以其他类无法利用反射API来得知泛型信息。

所以, 如果你只是表面上受保护, 而且如果新的语言不能够参与这样的解决方案, 这个语法只是徒增用户的负担罢了。泛型的细节不一致、让人疑惑, 这简直是个自讨苦吃的解决方案。当我问我的学生, 他们从集合中得到过多少类转换异常时, 很少有人认为这是一个值得注意的问题。



Ted Neward: 泛型

《Effective Enterprise Java》的作者

Ted Neward 是一位独立的顾问，专长在大型企业系统。他还是一位作家、教师，专攻 Java 和 .NET 的互通操作 (interoperability)。他写了数本广受认可的书籍，聚焦于 Java 和 .NET 领域，包括了最近出版的《Effective Enterprise Java》(Addison Wesley)。他和妻子住在西雅图地区，他们有两个儿子，两只猫和 8 台 PC。

一般来说, Java 有什么问题?

TN: 许多开发者写程序，无法匹配他们正在使用的工具和技术，就大声嚷嚷工具和技术“既丑又无法维护”，然后就重新发明轮子。

Java 1.5 有什么问题?

TN: Java 1.5 展现出反对进步的常见态度，而 Sun 坚决地拒绝让 JVM 进化，宁可继续幻想 Java 语言和 JVM 技术依然紧密地结合。

你喜欢泛型的实现方式吗?

TN: 不喜欢。事实上，他们实现泛型的方式是在语言层，而不是在 JVM 层。也就是说，内部仍然使用 Object 引用，所以：

- 其他语言仍然不具有泛型的概念。
- 我们无法从泛型中提高效率。
- 我们必须保持一些古怪的向下兼容性，依然允许使用对象引用（你可能会和我争论这一点，我的看法是，对象引用版本应该在 1.5 中废弃，在 1.6 中移除）。

Overloading

某些方面，Java 的类型问题恶化是受到另一个限制（但被误认为是特色）的影响：method overloading。overloading 本身不是一个大问题，但是 Java 开发者使用 overloading 让 API 可以支持多重类型，这会酿成问题。想让 API 数量暴增吗？用 overloading 准没错。

想看看例子吗？就看看 `java.util.Array` 界面吧！为了方便起见，你有超过 70 个方法可用。一层一层地剥完洋葱之后，你会发现他们只有差不多 10

个不同的实际功能。利用较聪明的方法声明，你可以用关键字指定参数，将未使用的参数默认为适当的值，像 0 或 null。

其他成本

当你决定为一切分类的时候，这是一个易滑的斜坡。当你需要从 Java 的类型系统中抽身时，并非总能如愿。你会开始看到许多 Java 链接库用不寻常的方式绕着类型打转。以 JMX 界面为例，这是绝佳的例子。它用强类型吗？乍看之下是的。然后你深入一点点，就会发现，它只能被概念性地描述为嵌入式类型系统，也就是一个迷你语言，嵌入在一个名称为 `ObjectID` 的 `String` 参数，完整的语言被定义在 `JavaDoc` 中，其语法对于编译器、界面生成器和处理器来说是完全不存在的。Java 的类型系统在这里不适用。JMX 建构师略过此类型系统，直接在字符串和其他对象内建立元数据 (metadata)。如果你看看周围，你还会发现其他更多类似 JMX 的例子。常常，Java 用字符串隐藏住弱类型，或者动态类型。

静态类型的优点

读了这么多负面的信息，你大概会怀疑为何还有人使用强类型、静态类型。至少有两个好处值得这么做，静态类型减少某些种类的错误（例如打错变量名称），并提供更多信息给你的 IDE 和其他工具（大多数和安全相关的类型争议焦点都在弱类型，而不是在动态类型上）。

看看下面的应用，Java 会在编译器抓出并处理这样的错误：

```
int consumer;
if (conusmer == 0) return consumer; // 拼写错误
```

难以想象一个动态语言，有着严厉的单元测试 (unit testing)，居然会让这样的错误潜藏其中。IDE 问题有一点更难理解。许多 Java 开发者习惯使用的特色，像自动列出方法让我们选用，需要依赖变量类型的信息才做得到。使用 Ruby 或 Smalltalk IDE 的时候，你却不能拥有这样方便的功能。

有漏洞的安全网络

Java 创立者常拿“可以在编译期而非运行时发现类型匹配错误”当成优点大肆宣传，这让我觉得很有趣，因为我所见过的 Smalltalk 和 Ruby 开发者，很

少有人会受到类型匹配错误的困扰。当然，大多数的人相当依赖自动单元测试，而我们确实该这样。不管你是否使用动态类型，你都需要进行单元测试。没有编译器可以完美地得知你的意图，即使你喜欢泛型，你还是会担心这种只是语法甜头的实现方式，而没有 JVM 的实现。

大量使用测试向导的开发的确实可以减少bug数量。事实上，Java的类型安全并非如你所想的面面俱到，不像Java语言设计者所宣称的那样。任何时候，大多数的Java对象都是放在集合中。任何时候你从集合中移除一个对象，你都需要把它转型，你等于是在给对象重新定义类型。如果转换不正确，你会在运行时以“类型转换异常”的方式打破玻璃。同时，运用改进的工具和强调自动单元测试，可以更容易抓住动态语言的类型问题。我的经验告诉我，Java的类型安全并不完整，也不彻底，并不像大部分的程序员所想的那样。在更动态的语言中，有了单元测试，类型就不会变成一种限制。

动态类型语言IDE的“程序代码自动完成”(code completion)问题，或许会随着更好的浏览器和更聪明的环境结合而解决。从程序的正确性立场来看，单元测试可以让类型安全变得不起作用。到最后，对于应用编程来说，更动态的类型将会胜出。因为动态类型而造成的生产力增加相当吸引人，让人无法忽视。

基本类型

从一开始，Java的设计者就决定故意吸引C++社区，效率是最重要的考量。最大的妥协是：包含了基本类型(primitive type)。这使得Java并非是完全的面向对象的语言，而且会有数个挑战出现。那些来自C++社区的人并不见得认为这是一个问题，但是来自其他编程语言的人认为基本类型实在很丑。基本类型并非继承自Object，所以Java是混种语言，而非纯面向对象语言。但这都只是学术理论，这个理论有相关的实际成本。

基本类型是受限的

Java基本类型会限制住你，因为它们不是继承自共同的Java对象。面向对象的好处之一是多态(polymorphism)：你可以用一般的方法处理特定的对象。但是在Java中，就不是完全正确了，因为基本类型不是继承自Object，所以，你不能说`6.clone()`或者`6.getClass()`。

如果你曾建立过 XML 输出程序 (emitter) 或者对象关系映射程序, 就会知道支持基本类型有多么麻烦。在 Java 中, 你不能够同样对待所有的类型, 你不能在基本类型上使用方法 (method)。你的程序中必须个别处理对于对象、基本类型、数组的支持。

因为大多数的人都没有建立过 XML 输出或者持久框架 (persistence framework), 所以这样的成本与我们无关, 对吧? 没这么单纯。你还是得面对语言的复杂, 例如不一致的 API 和各式各样的框架。反射 (reflection) 大概是最糟糕的。为了要取得字段的值, 你必须先知道其类型, 然后用下面的 get 方法之一取出这个值: getBoolean、getByte、getChar、getDouble、getFloat、getInt、getLong、getShort。当然, 如果是数组, 又是另外一回事了, 数组可以包含基本类型或对象, 所以甚至不能用一种方法来应付所有的内容, 因此, 上面的过程你还得重来一次。

在纯面向对象语言中, 反射 (reflection) 的设计更简单了。为了要取得字段的值, 你使用一个单一的 API 询问字段, 然后得到一个对象。你可以询问对象得出定义的类。如果你想要以顶层对象来对待它, 你甚至不必这么做。

基本类型不自然地冗长

当然, 你有时候需要对基本类型做一些事, 基本类型本身做不到的事, Java 利用类型封装 (wrapper) 来解决这个问题。基本类型是如此的笨拙, 因为有时候你使用基本类型, 有时候使用封装, 使用上无法达到一致。

当你加入额外的封装和类型转换, 你会发现基本类型的存在无法让 Java 更干净, 而性能也只提升一点点。既然同时有基本类型和封装, 你必须在这两者之间转换, 这就产生不必要的语法, 常常会导致不可预期的行为, 例如 Java 1.5 的 autoboxing (自动将基本类型转成封装) 就有一些奇怪的行为。

最佳平衡点

这一切的一切, 之所以支持基本类型是为了: 吸引 C++ 开发者前来使用 Java, 因为两个语言的想法和语法都很像。回顾起来, 这却造成一些明显的问题, 包括语言的清晰性、生产力和可读性。

为了要吸引 C++ 社区投入 Java, 我们付出了一些代价, 我认为这很公平。但是别低估这个代价。基本类型让程序代码变复杂, 造成不一致, 让语言变得

肥胖。下一个受欢迎的语言，大概不会是基本类型和对象混合的语言。C++ 开始了面向对象的转变，而Java接手完成。现在，我们不再需要依赖拐杖了。

分手时的刻薄话

当然，你可以写一整本书，好好地讨论Java的优缺点，但我不觉得这会有帮助。我不会重讲一遍“EJB的冲天臭气”的信息，这些在我的最近三本书中已经说太多了，我也不打算投入空白（whitespace）、注释风格、bytecode加强（enhancement）优缺点的辩论。但，还是有一些事需要提及，在大部分Java应用中，异常和字符串扮演着很重要的角色。

Sun

Sun已经不再是以前的那家公司了，他们让Java的前途堪忧。我不是说Java会消失，但是Sun可能会消失。Sun在银行有许多现金，用来赚利息吗？它在低端受到Intel、Dell、AMD等公司的威胁，IBM则是从上面压缩Sun的生存空间。Sun的软件与服务业务从未真正成功，我想Sun是一个相当适合被并购的对象。

如果Sun真的出了什么大问题，Java何去何从？我害怕如果IBM收购Sun之后，会把发展重心放在最困难的企业问题上，让Java偏离它的基础。开放源码Java会造成语言的分裂。其他潜在的并购者，像Oracle和BEA都可能会造成利益冲突，妨碍新的标准。

IBM可能紧张了，它已经在许多方面构筑起它的Java城墙：

- IBM和BEA紧密地联盟，进行SDO标准的制订，这和JCP不一致。以IBM的地位，应该有能力挑战JCP，或者建立JCP之外的标准。
- IBM看起来似乎会越来越密切地拥抱PHP，利用这块渐渐变大的市场，不放弃中小企业的市场。
- IBM继续协同微软在XML技术上投资。

最后，不管Sun是健康或失去健康，都会让人怀疑Java未来的走向。如果你想要维持市场主导地位，你最好不要选择“不确定未来”的东西。

异常

就像静态类型一样，Java 强调受检异常（checked exception），就像是不可摇动的根基一样。论点是这样的：如果一般的开发者不能明确地处理异常，他大概根本就不去处理。对我以及我许多的顾客来说，受检异常带来的缺点比优点大，理由如下：

- 异常语法的侵入性很高。异常轻易地主宰一般的方法，甚至在很低的层次上也是如此，你常常没办法处理。
- 大部分的时候，你不能处理异常，所以只能丢弃它，让它继续发出警告。你不应该做这种编译器能帮你做的事。
- 这么多异常处理的语法，让你忽略真正重要的程序代码。换句话说，在泥巴内很难识别出钻石。

最近，像 Spring 以及 AspectJ 这一类的 Java 框架开始发现未受检异常（unchecked exception）的威力。Hibernate 的创立者 Gavin King 常常说，如果有机会再设计一遍 Hibernate 的话，希望将 Hibernate 建立在未受检异常模型上。Hibernate 在第三版的时候，已经改用未受检异常了。

表示数据

编程和数据携手合作，在大多数的语言中，结构化数据变成应用程序很自然的一部分。Java 之所以过度依赖 XML，部分的原因是 Java 本身不适合表示结构化的数据。而 Ruby，比方说，我可以很快地声明一个数组的散列表（hash map），这样的结构动态地简化配置（configuration），并允许自然的元编程。

字符串

如果你看到 Perl，你可以很快地了解它被设计来做什么，它是一个用来处理文本的引擎。虽然 Perl 许多方面很复杂，但很受欢迎，因为它在文本处理方面做得相当好。

相反，如果你看看 Java，没有同样方便、高效率的文本处理功能。这让人感到很惊讶，特别是当你看到我们用 Java 来做许多核心工作，更是惊讶得不知所措。这些核心工作包括 servlet、XML、JSP、HTML，都是和字符串处理

相关的。事实上，我花在字符串上的工作时间大概比其他事多。我觉得很惊讶，Java在处理字符串方面居然没有比较好的表现。他的模式匹配支持是次级的，而且主要的字符串 API 水平都很低。

简单性

对于大企业的编程项目来说，Java已经是一个好的语言，在解决这方面的问题时，它越来越好。但Java在小应用程序上就很不好用，你可能会想用Visual Basic写小企业的系统。在这极大和极小两个问题之间，还有相当大的区域。Java进入这块区域，带着复仇的心态，并狠狠地扯动微软企业编程社区的心。但是图 4-2 显示 Java 很快地离开了这个社区。

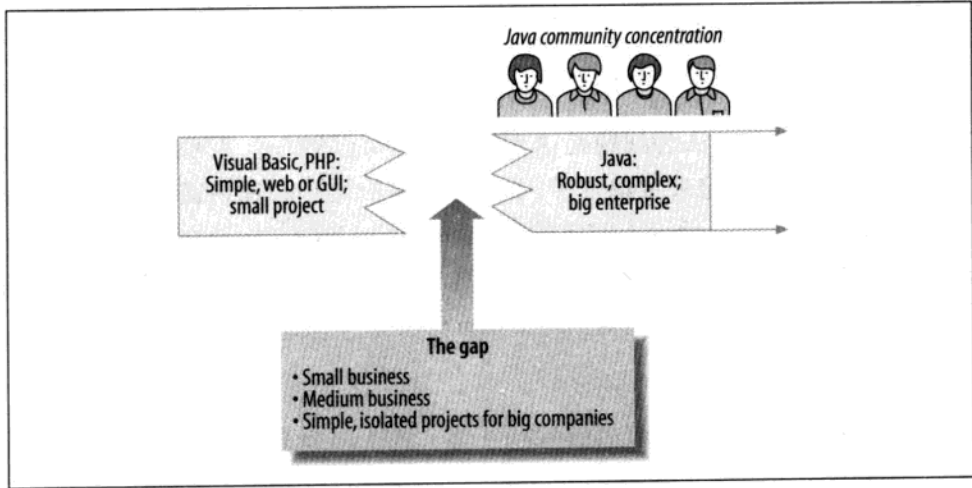


图 4-2：Java 已经控制住大型和小型项目之间的缝隙，但是正在离开这个社区

在我过去的三本书中，已经举过例子，Java想要继续成功，就必须致力于简化。但这样的事情并没有发生，Java的威力架构牢牢地企业领域生根。在过去的三届 Java One 研讨会上，Sun 只会嘴上说说需要简化 Java API，但是没有采取行动，我们只看到限制焦点在更丰富的用户界面上。大厂商宣称致力于简化，结果出来的答案是EJB 3.0、泛型以及Java Server Faces (JSF)。

事实上，Java正远离它的基础。别忘了，我们中许多人都在等着使用更好、更简单的方法建立拥有Web前端的关系数据库的系统。但是，我们看到越来越多的XML、配置、抽象层，Java以稳定的速度漂离用户界面和最终用户，学习Java要花的时间越来越长，让人越来越不想接触Java。

工具

问题的症状之一是：Java 过度依赖工具。Java 开发者喜欢 IDE，事实是离不开 IDE。在不久的过去，我帮一家主要的应用服务器厂商做了一些研究，我发现最具有生产力的开发者喜欢使用命令行工具。命令行工具和编辑器到处都有，如果你熟悉这些工具，到哪里都能如鱼得水。

但是，在过去三年中，我们勉强到达了顶点。最聪明的开发者正纷纷移往 IDE，因为 Java 语言已经变得太复杂，如果不使用 IDE，就会完全失控。你就是需要 IDE 来帮你做各式各样的重构。其他的语言也有 IDE，也有好的程序员，他们就算没用 IDE 依然怡然自得。

为什么不直接修改 Java?

你可能认为我们可以修改 Java，而不是扔掉它，如果你可以指出问题的话，这会很容易。如果你认为问题是语言的话，你可以做一些手术，提供新版的 Java。说的容易，做起来可就难了。Sun 不计代价地小心保持回溯兼容性。如果你看到全盘翻新之后少人问津的 Visual Basic .NET，你就会更能尊重 Sun 的观点。微软对 Visual Basic 语言和链接库做了重大的改变，但是却不被大家接受。不管这是不是一个好点子，Sun 将会继续保持保守的作风，以保留客户。

但是，即使你看着稍微具有侵略性的改变，我认识的大多数专家都认为 Sun 根本走错方向了。Java 所做出的一些改变，并没有让 Java 在基础上变得更敏捷与动态，大部分的时候，只是加入一些语法上的甜头，却往往导致 JVM 行为不一致。

链接库和社区

很清楚地，链接库也是个麻烦。Sun 已经展开一些简化的动作，但为时已晚。所以，如果问题出在 Java 链接库，而不是语言本身，那么我们何不丢弃一些链接库，重新设计出简化的版本？我在《Better, Faster, Lighter Java》一书中就这么建议过了。对于 Java 最重要的和最基本的工作，也就是关系数据库上的 Web 用户界面，我一点都不认为 Java 的框架现在的方向是对的。大多数框架都一直在快速地增加新的特色，而不是简化已经乱成一团的现况。

一个很差的链接库，可能会导致一些区域的错误。Java的问题则是全面受影响，它们面对“非常复杂的问题”，所以只好牺牲“简单的问题”作为代价，但是这些简单的问题又是大部分 Java 开发者会遇到的问题。这一切很清楚了，Java的领导地位让它自愿快速地放弃根基。这是文化的问题，问题来自 Java 社区、厂商、程序员以及 Java 的领导地位。Java 变成只适合大型企业系统开发的语言和平台。

别的选择

之后的5年中，会存在这样的问题：我们真的值得为 Java 社区和膨胀的链接库而牺牲使用其他语言所能带来的高生产力吗？到目前为止，答案都是响亮的“是的”。但是我们现在已经到了一个不归路的路口，如果我们要继续走下去，Java 需要大改，但是 Java 的社区、文化以及领导地位却无法做出我们所需要的改变。Sun 对待 Java 总是相当保守，JCP 总是会建立那些软件，我们都可以想象得到的软件：膨胀的折中方案，到最后，完全没人喜欢那些软件。Java 社区长久以来都在忍受太多的架构、XML、层次。

在本书的后半部，我会举出一个清晰、动态的语言可以在 Visual Basic 和企业级 Java 之间取得稳定的位置。一旦根深蒂固之后，它当然也可能和 Java 一样，进入企业运算领域。毕竟，Java 开发的大部分应用（即使是在企业端）并非充满着分布式事务和超大的负载。5 年以前，和我交谈过的大部分开发者，都只想要一个好方式可以应付庞大的关系数据库上的 Web 用户界面。而 5 年后的今天，他们还是有一样的希望。

到目前为止，我已经让你看到 Java 如何漂离它的基础，在下一章，你将会看到下一个语言盟主擂台赛的游戏规则。在本书的后半部，我会探讨成功语言必须具备的因素，以及是否会足以让你超越 Java。

第五章

游戏规则

10 年泛舟的经验，有几个惊骇的急流值得一提。Chatooga 河有许多急流，Chatooga 上面的 Bull 水闸有一个瀑布，瀑布穿过河床的洞，倾泻而下。它的一端很大，可以容纳一个泛舟者，但是却不够使其退出。Cork Screw 是一个猛烈的方法，也是一个训练水力的方法。Woodall Shoals 则是表面沉着地落下，接近完美的水流，我认为即使在我泛舟的顶峰那几年，依然无法办得到。在这些急流中，容许的误差幅度相当小。不是到达你预设的终点线，就是受伤或死亡。游戏规则就是这样。

让我们假设你认同我在本书一开始说的前提：时机已经成熟，该是另一个应用编程语言出现的时候了，因为 Java 正放弃它的基础。我不打算假装我知道 Java 接班者是谁，我只能够保证展示一些有趣的语言和框架。我也会根据游戏规则立刻排除一些语言。

如果你仔细想过，你本能的就知道某些编程语言根本不适合成为下一个主流的候选人。Lisp 具有很高的生产力，但是一般人不容易理解这个语言。Perl 既丰富又具威力，但是相当不一致，而且容易导致无法维护的程序代码。通过一些经验，你可以想象出 Java 之后的语言应该是怎样的。我建议用比较松散的方式定义成功：语言应该被现在的 Java 开发者所广为认识与采用。本章，我会建议一些特质，语言应该具备这些特质以获得商业上的成功。

Java 提高竞争门槛

每个新的语言都符合当时的主流规则。如果你想想音乐世界的新发明，你会看到同样的原则。在早期的唱片行业，一个唱片品牌会签下一个艺人，详细定义合约、制作唱片、在电台打广告、配销到商店。今天你会在产品周期和广告看到许多人扮演许多角色。标准的改变强迫业界重新设计工具，相对之下较小的改变（例如速度）是让制造商在唱片播放器上增加一些功能，有些则是根本的大改变，这些改变需要临界质量才能发生。CD达到了临界质量，但是八轨录音则没有。有时候，分裂性的改变会完全重新定义组织，并影响业界的每个部分。我们的孩子通过Napster和iTunes等服务重新定义音乐被传播的方式，某些艺人则是完全通过因特网传播音乐，而他们让出版工业有了全新的面貌。

新的编程语言也和唱片工业一样。每个语言都留下了遗产，有时候，语言会改变，以拥抱这些遗产。比方说，你将C程序编译成DLL或者可执行文件，你可以在换成C++编译器之后，依然利用你的C程序代码。你甚至可以使用C++以编写传统程序（procedural）代码，或者面向对象的程序。C++改变了我们思考的方法，但是没改变机器。C编程语言许多方面也是具有破坏性，Java也是。破坏性重新定义了游戏规则。

孩子们喜欢立即下载歌曲，像“Macarena”，所以老旧的唱片行已经不再吸引人了，而他们已经关门，不再试图开门，除非你计划用你自己的钱资助他们。同样，我们喜欢JVM的便利性、人数众多的开放源码社区以及以因特网为焦点，这一切为下一个主流的应用语言订立了很高的门槛。

可移植性

请记得我们技术皇冠上的珠宝。Java将虚拟机的概念推广进入商业圈，重新定义了环境。你将Java程序编译成中间的byte code，可以在虚拟机上运行。我们现在已经知道虚拟机的优点。下一个主要的应用语言一定会支持虚拟机。你不能忽略虚拟机的优点：

安全保护

如果你能够在虚拟机中进行安全保护，语言的保护将会容易得多。

移植

虚拟机让语言能够有一个共同而清楚的基础。

扩展

如果你的语言不适合做一些事,你还可以在byte code中进行改变。Java的一些扩展技术(例如JDO和AspectJ)就是使用这种称之为byte code增强(enhancement)的机制来扩展Java的语言能力。

相互合作

低端的byte code让一个语言可以使用相同的部署层,甚至可以和用其他语言写出来的程序协同运行。

所以,虚拟机很重要,我相当有信心。下一个商业成功的语言应该有JVM的这种愿望,这可以帮助语言减少许多障碍,不管是政治上的,还是技术上的障碍。



Dion Almaer: 为何Java难以被取代

Dion Almaer是Adigio公司的CTO和创立者。他是一个架构师、导师、实际的人以及技术传教士,涉及的技术包括J2EE、JDO、AOP、Groovy。他是TheServerSide.com J2EE社区网站的前任总编辑,也是Java Community Process (JCP) 成员,他参加过许多JCP专家小组。

你已经是Ruby on Rails的拥护者。你认为Ruby有潜力取代Java吗?

DA: 我很喜欢像Ruby这样的技术,但是我不知道如何在主流领域使用它。毕竟这两个大虚拟机(JVM/CLR)的威力实在太强大了。

“你是说,我应该把财富500强企业的赌注放在一个名为Matz的日本摩门教徒身上吗?”(译注1)

DA: 惯性是大公司考虑的重要因素。Ruby的道路会怎么走?标准在哪里?这么多模块的质量如何?Ruby用在手机上会怎样?

要如何排除这些障碍?

DA: Ruby是一个顶尖的语言,具有一些令人惊讶的框架,但是要进入下一个层次需要更多条件。我希望看到

译注1: Ruby语言是日本软件工程师Matz所设计的。

JRuby 以及 Ruby.NET 的起飞，对此会有许多政治上的争辩，但实际上会很有帮助。

在 JVM 上面，已经有人写了许多好东西。Ruby 对我来说感觉是一个语言（在某些工作上），但说它是平台就比较不具有卖点。如果我能够结合这两点，我就准备起跑了，这就是为什么 Groovy 能够有施力点，且 Java 开发者会这么兴奋的原因。对老板来说，语言是“Java”，但是他们可以同时用 Groovy 来写 script 程序。

外面有许多过去遗留下来的程序代码，所以要转移平台并不容易，除非真的有严格的迁移计划。

像 Ruby 这样的东西需要“杀手级应用”，目前这方面要感谢 Rails，但是这就够了吗？哪一类的程序可以在 Rails 上面执行？我们等一等就会看到了。不要误会我，我大部分的想法之所以如此，是因为我希望业界能够转移到更动态的语言上，我认为我们要这样，但是我有些怀疑。

因特网的焦点

Java 为因特网集成设定了一个新的门槛，Java 用户充分运用这样的优点。公司内部使用因特网，以区分信息并控制应用的部署成本。业务部门对外使用因特网，以接触客户和伙伴。让应用程序具有因特网功能，变成了业务上必须解决的当务之急。Java 集成新一代的因特网应用，这方面是通过 servlet 编程模型、以 JSP 当编译模板语言以及一整套的企业链接库。下一个成功的语言必须处理因特网的需求，且要比 Java 做得更好。

因特网至少有两个尺度：计算机的接口以及人的接口。对人来说，下个时代的语言应该建立更有威力的接口，有比 Java 更快的速度。只能建立简单的 HTML 是不够的。你需要建立网页，能够保持企业内通用的、层次化的外观感受（look and feel），所以下一个语言需要支持某些组件模型。还有，用户已经开始了解到 HTML 是不够的。像 Google Maps 以及 Google Mail 这样的应用程序，扩展了 HTML 与 JavaScript，到了全新的境界。对于下一个成功的语言来说，这相当重要。

事实上，本书的许多顾问都相信HTML基本上已经不能用了。一个广泛成功的语言要呈现较高的抽象，让它更容易在业界慢慢立足。Ruby on Rails以及 Ajax 技术似乎都是朝这个方向在发展。

相互合作

在Java和新语言之间建立桥梁也很重要。当然，如果新的语言拥抱JVM，在低端的地方互相合作将不会有问题。在因特网上的相互合作将会扮演相当重要的角色，我认为这会造成三个重要的能力：XML、Web service以及SOA。

XML 与数据结构

编程总是要和数据为伍，但是Java无法让你很容易地声明嵌套（nested）结构的数据。在Java中，你会看到XML的暴增，即使价值不高的地方，大家都还是用XML。比方说，元编程（metaprogramming）和各种配置（configuration）需要你表达出结构数据。下一个语言应该让你可以声明并表达结构化的数据，用清楚而固有（native）的方式。

但是，结构化的数据和用来描述它的语言都很重要。如果你是在因特网上处理结构化的数据，你大概会处理XML。下一个成功的语言应该让你以有生产力的方式处理XML，效率也必须很好。在Java中，我们使用解析纲要（scheme）、查询语言、绑定框架来处理这样的问题。当解析器一打开XML，让你进入组成部分的时候，绑定框架让你可以将对象模型直接转成XML，或者处理XML，仿佛它是固有的对象模型。XML查询语言（像XQuery）可以进入一个复杂的XML文档，取出一个有名称的数据。期待下一个流行的语言支持这三种XML技术是很合理的。事实上，大部分的新语言都支持了XML，但程度不一。

面向服务架构（SOA）

一个共同的结构化数据格式不足以连接起两种语言，你还需要一个共同的机制。现在的语言趋势（像Java这样）是建立松散耦合的服务，提供网络服务，通过XML的有效负载（payload），让它们之间利用简单的信息来沟通。这对相互合作来说是很好的策略，因为：

- SOA 最好搭配粗糙纹理型的架构，或者调用一块程序代码。语言之间的相互合作是一个大方向的问题。

- SOA 很流行，因为政治上受欢迎，支持和热衷程度可能会维持在高峰状态。
- SOA 使用因特网标准，这意味着你可以运用现有的底层建设，像安全和现有的信息协议。

如果 Web service 像微软和 IBM 定义的那样，我不确定 Web service 会具有持久力。我相信一个名为 REST 的轻量级 Web service 可能会更持久。REST 是 Representational State Transfer 的缩写，它利用服务来提升因特网多年来的做法。就像因特网一样，REST 将网络视为是资源的集合，而不是方法 (method) 的集合 (像 CORBA 或传统 Web service 那样)。

REST 为基础的资源返回一个“自己的表现”，通常是用 XML 格式。REST 允许甚至鼓励链接 (link)。基于 REST 的服务建立在被理解透彻，且成熟的 API 上。所以不像易碎的传统 Web service 堆栈，它们可以和其他技术集成得很好。它们也依赖现有的架构，能够缓存内容、建立链接或者安全沟通。这是一个威力强大的思维转换 (paradigm shift)。

所以 Java 提供了第一组规则，如表 5-1 所示。如果你想征服这条河流，你需要超越 Java 所立下的标准，达不到这个标准免谈。

表 5-1: Java 所设立的旧标准

规则	描述
JVM 和 / 或微软的 .NET CLR	在 JVM 内运行，在自己的虚拟机内运行
焦点在因特网上	支持因特网应用程序
因特网用户界面	允许丰富的因特网用户界面
服务层	提供和 Java 整合的 SOA 风格
Web service	允许某些种类的 Web service，不管是全功能的 Web service 堆栈，还是 REST 的 Web service
XML	提供丰富且具有生产力的 XML API

企业集成

在某些方面，通过允许从开放式 API (对微软应用程序来说，就是 ODBC)

提供事务监控（像 Tuxedo 和 Encina），C 重新定义了企业集成。C 是具有析构力的：它让企业编程走出大型计算机。Java 接下这个遗产，有着事务框架、丰富的数据库整合、信息传送以及许多下层功能的中间件（middleware）。

我不认为下一个主要的应用语言会从一开始就具备 C 和 Java 在企业应用方面的能力。Visual Basic 没有这些能力，依然获得相当大的成功，VB 可以运用其他低级语言（例如 C）所写出来的服务，来达到目的。我们已经决定下一个语言应该和 Java 程序相互合作，最好还是在同一个虚拟机上，也应该在大方向的服务层上一同合作。也就是说，某些企业能力将会相当重要。

数据库的整合

最低限度，一个新的语言应该以自然且具生产力的方式访问关系数据库。我不认为任何特定的应用风格很重要，你可以看到许多成功的环境其实是使用不同的策略：

- 微软建立了一个框架，可以运用 SQL、row set（查询结果的集合）以及关系型数据库（relational database）。从数据库的观点来看，微软宇宙的重心，就是关系型数据库。此策略可以适合各种大小的应用，而且相当具有生产力。
- 相反，Java 似乎走向 ORM。Java 的数据宇宙重心是面向对象模型以及持久模型（persistent model）。有一些 Java 应用程序运用 JDBC 搭配辅助框架，也相当成功。
- Ruby on Rails 利用中间的方法。Rails 将数据库表格封装成对象，对象会自动、动态地发现数据库的结构。

所有的策略都有优点和缺点，而且每一个都可以作为新语言的基础。我认为，下一个热门的语言以及它所需要的核心框架，应该遵守下列规则：

拥抱关系型数据库

当一个语言可能和其他语言整合时，关系型数据库应该是第一等公民。将太多焦点放在面向对象数据库上，对于采用像 Smalltalk 这样的框架来说会有问题。面向对象数据库是优雅的解决方案，会忽略目前的政治现实。

不要在关系型数据库上面强制结构

在某些层次上，新语言必须容易使用既有的关系纲要 (schema)。强迫使用代理人独特的 ID，而不使用合成的主键 (primary key)，这等于忽略了现实。

性能和规模

评估一个应用程序的性能如何，数据库的性能是最重要的单一指标。

事务和安全

企业开发者需要具有定义商业事务的能力。不管你能用多快的速度建立不正确的应用程序，如果 Joe Bob 在因特网交易时损失了 \$50，只是因为 Sally Sue 绊到电源线而使得服务器关机，那么这么差的框架就不能被用在企业内。

安全也很重要，虽然你可能会不以为然，毕竟底层的操作系统都已经充满让人目瞪口呆的安全漏洞了。Java 在这个领域已经立下标杆，相当难以超越。下一个语言需要具备整合现有企业安全框架的能力，至少要符合因特网方案和标准 (例如 LDAP)。表 5-2 汇总新语言必须具备的企业特色。

表 5-2：新语言必须具备的企业特色

规则	描述
数据库访问 API	提供简洁、具有生产力的 API 以进行数据库访问
关系数据库	先把焦点放在关系数据库上
数据库性能	和数据库的交互必须很快
事务	能够进行应用事务的划分
语言安全	提供简洁的基础，以进行语言的安全保护
应用程序安全	让开发者保护他们自己应用程序的安全
安全整合	让开发者整合企业安全，特别是针对因特网应用程序

就像 Java 的特色一样，新的语言拥有这些基本的企业特色并不能保证一定会成功，只能拥有参赛权罢了。

产生话题

许多语言的技术都超越了 Java，但还是赢不了 Java。Betamax 技术上比 VHS

技术好，但还是输了。最大的关键因素是社交。如果没有可靠的社区，就不会成功。对一个程序员来说，语言是未来的投资，语言甚至就可以代表个人。你可以说这是营销、蜂鸣（buzz）甚至是夸大宣传。如果它红了，或者至少大家有兴趣了，那么这个语言就有了登台一搏的机会；否则，连一点机会都没有。在某些方面，Java 为下一个语言铺好路了：

- 像 TheServerSide 以及 Slashdot 这样的社区提供论坛让大家发表想法，快速传播到 Java 和非 Java 的编程社区。较小的项目会比较大的项目更容易产生话题。
- 开放源码日益受到重视，部分是由 Java 所推动的，这使得技术可以摆脱大公司的控制。还有，同样的公司也会发现开放源码技术比较容易，且使用上有较少的威胁。
- 许多 Java 标准，像 Web Service（以及轻量级的 HTTP 替代技术），使得不同语言之间的合作变得容易许多。
- JVM 将会运行其他的语言。新语言建立在 JVM 之上，将会比在其他环境上更具有卖点。

但是，建立一个社区的挑战很大，让人为之却步。微软为了营销 .NET，已经花了数百万美元，但是服务器端的使用率还是远不能和 Java 相比，虽然 .NET 的确有许多特色和能力都和 Java 一样，甚至超越了 Java。而 Sun 在 Java 平台成功之后，不能够将它利用到自己的软件领域，采用 Sun 的应用服务器以及管理软件可能问题多多。IBM 失去了操作系统的战争，因为它不能通过营销告诉大家 OS/2 技术相当卓越。

程序员都要精神分裂了。一会儿，我们是极端的怀疑论者，抛弃微软窗口的安全性，而在桌面上使用不可预期的 Linux 系统。接下来，我们没有任何好的例证就使用像 EJB 这样丑不可档的架构。在编程社区中，你也有许多不同的利基。Java 对于企业开发者来说是成功的，但是 Perl 和 Python 社区的黑客高手却鄙视 Java。而 Microsoft 开发者则有自己的文化，内部还分成 C++ 与 Visual Basic 的次文化。

这代表着赢家的公式也会改变。一方面，像 Steve Jobs 这种有统治权的人物会走出不一样的路，但这是不够的，就像运气欠佳的 NextStep 平台一样。也就是说，产生蜂鸣是相当高的艺术，比科学还高，而且比艺术还靠运气。但是，某些主题和趋势则是环环相扣的。

开放源码

除非是具有分裂性的技术,否则下一个主要的编程语言不会来自较大的商业厂商。微软、IBM、Sun等大厂商之间有太多的猜忌和恐惧。我认为下一个广受欢迎的技术比较可能来自开放源码社区。开放源码模型提供了一个舞台,让数千个项目在此上演,它们可能成功,可能失败,这就要看它们的表现了。开放源码项目需要提供有效的技术,并对那些愤世嫉俗的、关键的群众营销,才能获得成功。在开放源码社区中,现在有数个有趣的测试案例:Perl、PHP、Python、Ruby等。你很少看到商业语言有这么大的动力,不要提C#,因为它只是Java的翻版。

开放源码软件会伴随着一些问题而来。因为开放源码项目通常没有正式的支持,社区必须支持此语言,这样的环境直接测试社区对于此语言和技术的动态性。社区有自己的个性,像势利的、急躁的、有教养的、争吵的。像Java这种较大的语言可能会有次社区,具有自己的个性。当一个语言忽然间受到注意,此社区的个性就会变得具有吸引力,或让新的用户厌恶。微调社区的动态是一个困难的提议,因为这种个性可能不容易被社区内部所评断。新的语言将需要一个具有吸引力的社区才能获得成功,而开放源码社区似乎自然地变成塑造这种个性的摇篮。

经济

虽然开放源码框架通常将某种智慧的正直融入项目中,但是商业力量才具有决定性的一票。新的语言需要一个支撑它的“生态系统”才能兴旺起来,这意味着某人最终必须开一张支票。简而言之,你不可以没有经济支撑的情况下远离Java。对我来说,首要的经济催化剂是很清楚的:就是超高的生产力。

当我第一次离开Java时,我需要“经济的母鸟”来喂食巢内的我。Java的环境是相当舒适安全。写这本书时,我回想起以前,而我现在正在为一件事情的开端发号施令。当时我们正写一个Java应用程序,以帮助工程师为设备做好配置设定,该设备用来进行工厂安全的测量。我建议该应用程序不要用Java,而是改用Ruby来做,因为我发现用Ruby的生产力比用Java提高了很多,已经到了不可以忽略的地步。优点还不只这样,当应用程序开发完毕之后,此新的应用程序具有较少的程序代码,比较容易维护,比较快,比较容易微调,也比较容易扩展安全机制。我估计用Ruby的生产力比用Java快

3到5倍。的确，Ruby不是唯一一个比Java更具有生产力的语言，但这是一个有趣的实际案例。你将会在第七章看到更多这方面的资料。生产力将会是经济上的催化剂，开始侵蚀Java的根基，它将会推动下一个伟大语言的出现。

可亲性

所有成功语言早期的使用者都会告诉你关键在于可亲性（approachability），新语言需要很快地抓住新的用户。你应该能很快上手，并立刻解决手边的问题。C是可亲的语言，因为它让系统编程的中间人能用高级语言解决低端问题，同时比其他高级语言具有更好的性能和灵活性。C++是可亲的语言，因为可以直接兼容于C而不需要修改，你也可以让C程序升级到具有C++的特色。Java是可亲的，因为它具有类似的语法、较友善的内存模型以及一个清楚而一致的路径，可以作为因特网的解决方案。Smalltalk不是可亲的语言，因为厂商要价太高，太贵，所以大家用不起。

虽然“类C++”的语法并不简单，我仍然认为许多语言会因为具有友善的、具生产力的语法以及熟悉的对象模型而变成可亲的。Python相比较于Ruby，就是个好例子。Ruby有最具生产力的Web环境以及一个社区和让程序员无痛苦学习的哲学。你可以很容易地安装组件，只需要用到两三个字，这是因为Gems会帮你做许多事，让你很快就可以拥有因特网连接。而Python具有简单的语言和语法，但是Web链接库则一点都不具有可亲性。当你开始学习Python的Web链接库时，你找不到自学手册，也没有社区可以帮你。Ruby on Rails的人们了解如何让Rails可亲。

杀手级应用

如果没有某种催化剂，将难以想象成功的社区要如何开始。Applet让Java很快地传播到许多桌面上，只要几个小时，开发者就可以将动态内容嵌入到他们的网页中。

表面上，似乎语言必须要有一个活跃的社区，好让语言变得丰富，可以应用于任何规模。但是如果语言不够丰富，社区通常就不会集结壮大。对开发者来说，杀手级的应用是一种解决方案，相当具有吸引力，已超越了语言本身


```

int o = (n/2);
int t= 1;
int step = 2;
for (int i=0; i<n; i++)
{
    for (int j=0; j<Math.abs(o); j++)
    {
        System.out.print((char)32);
    }
    o--;
    for (int k=1; k<=t; k++)
    {
        System.out.print("*");
    }
    t = t + step;
    if (t == n)
    {
        step = -step;
    }
    System.out.println();
}
}
}

```

在多个 for 循环嵌套语句中，使用了多个变量，然后用“*”绘制了一个菱形的图案，用户只需要记住判断出什么时候有一个“*”，在这个程序里出现了一个 Math.abs(o)，初学者也许不太理解，只需记住这个语句是取“0”的绝对值即可，下面运行这个程序，得到如图 6-5 所示的结果。



图 6-5 多重 for 循环的结果

实例探索和读者练习：

在上面的一个实例中，进行 for 循环的多次嵌套，运行结果是一个用“*”组成的菱形，下面将展示一段代码，读者将代码从光盘复制到计算机中运行，观察运行结果，如图 6-6 所示。

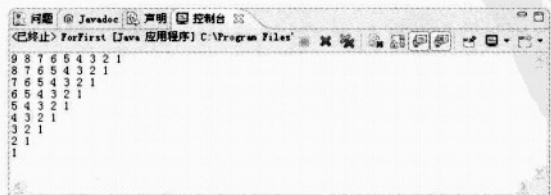


图 6-6 for 嵌套

看了这个运行结果，下面将展示这段代码，其代码如下：

```
public class ForFirst
{
    public static void main(String[] args)
    {
        for(int i=9; i>=1; i--)
        {
            for(int j=i; j>=1; j--)
            {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

6.4 while 语句

while 语句是 Java 程序中的另一种循环语句，当不知道语句块或者语句需要重复多少次时，使用 while 语句无疑是最好的选择。当它的表达式为真时，while 语句重复执行一条语句或者语句块，它的基本格式如下：

```
While (condition)
{
}
```

while 循环控制语句的流程图如图 6-7 所示。

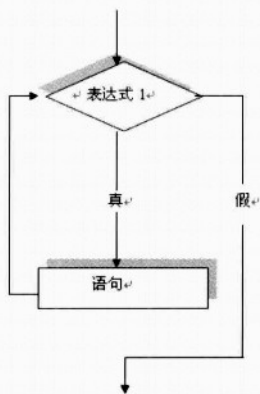


图 6-7 while 循环控制语句的流程图

在这个流程图中，值得注意的是表达式 1（条件表达式）是布尔表达式；只要表达式 1 为真，语句块将被执行；当表达式为假时，它将跳出循环。下面通过一段简单的代码认识 while 语句，其代码如下：

```

public class WhileOne
{
    public static void main(String args[])
    {
        int a=0;
        while(a<10)
        {
            System.out.println(a);
            a++;
        }
    }
}

```

这段代码和 for 循环的第一个代码的结果完全相同，它们起到同样的作用，执行结果如图 6-8 所示。

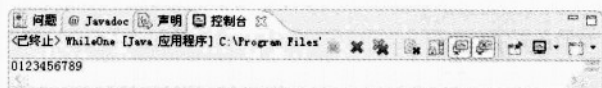


图 6-8 while 语句执行的结果

当条件表达式从一开始都不成立时，这个时候 while 的语句块将一次也不执行，如下面代码：

```

public class WhileTwo
{
    public static void main(String args[])
    {
        int a=0;
        while(a<=-5)
        {
            System.out.println(a);
            a++;
        }
    }
}

```

这个程序执行到第 6 行时，将直接跳到第 10 行执行，7~9 行将不执行，因为 $0 \leq -5$ 不成立为假，所以这个程序将直接跳到第 10 行，不会产生任何效果，执行这个程序后，得到如图 6-9 所示结果。

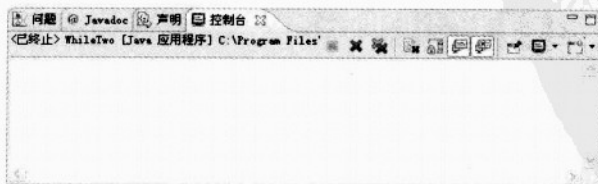


图 6-9 while 语句执行的结果

学习了 while 循环的基础后，下面将讲解通过一个实例来巩固 while 语句。

实例 15: 这是一个 while 的循环语句, 进行判断累加和不大于 10 的所有自然数, 其代码如下:

```
public class WhileSum
{
    public static void main(String[] args) {
        int sum=0;
        int i=1;    //由于是计算自然数, 所以 i 的初始值设置为 1
        System.out.println("累加和不大于 10 的所有自然数如下: ");
        while(sum<10){
            sum=sum+i;
            System.out.println(i);
            i++;    //该语句一定不能少
        }
    }
}
```

运行结果如图 6-10 所示。

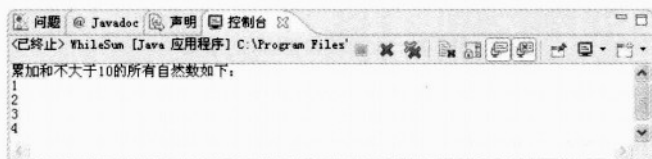


图 6-10 while 实例运行结果

实例探索和读者练习：

在上面的一个实例中, 进行条件语句的判断, 对条件 while 语句进行深层次的学习。下面将展示一段代码, 读者看完代码后, 给出运行结果, 然后再将程序输入到计算机中进行编译, 看结果是不是与自己给出的结果相同, 程序如下:

```
public class whileSum2
{
    public static void main(String[] args)
    {
        int a=0;
        while(++a<=100)
        if ((a%9)==0)
            System.out.print(a+"t");
        System.out.println();
    }
}
```

6.5 do...while 语句

从上面一节中, 可以看到 while 语句执行到条件表达式为假时, 循环体中的语句将不再执行, 直接跳出循环体语句, 执行下面的语句, 但在某种情况下, 即使条件为假时, 也

需要执行语句一次。初学者可以这么理解，在执行一次循环后再测试表达式，在 Java 中，当然为用户提供了这样一种循环的语句，那就是 do...while 循环语句；do...while 语句的特点至少会执行一次循环体，因为它的条件表达式在循环体的最后，do...while 的格式如下：

```
do
{
}
While (condition)
```

do...while 循环控制语句的流程图如图 6-11 所示。

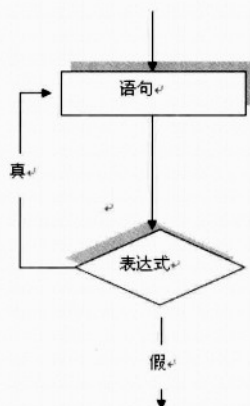


图 6-11 do...while 循环控制语句的流程图

do...while 是先执行一次再判断表达式，如果表达式为真，则循环继续；如果表达式为假，循环到此结束，如下面代码：

```
public class DowhileOne
{
    public static void main(String args[])
    {
        int a=0;
        do
        {
            System.out.println(a);
            a++;
        } while(a<10);
    }
}
```

这段代码是重写的上面两个循环语句中的例子，执行结果如图 6-12 所示。

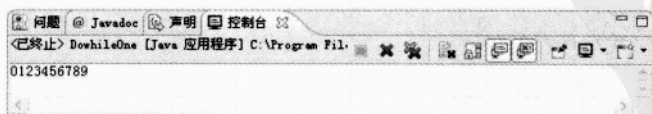


图 6-12 do...while 语句

这段代码和前面所学的 for 和 while 循环的第一个程序结果一样,不同的地方是,这段程序即使条件不满足,它也要执行一次,如下面代码:

```
public class DowhileTwo
{
    public static void main(String args[])
    {
        int i=0;
        do
        {
            System.out.println(i);
            i++;
        }while(i<=5);
    }
}
```

这段程序的条件依然不成立。在上面的 while 循环中也出现过类似的情况,结果是空白无内容;但是这段代码依然会有结果,结果是“0”,执行程序,得到如图 6-13 所示的结果。

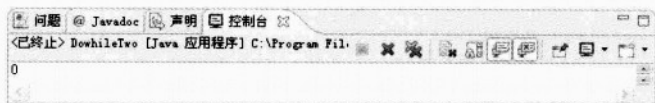


图 6-13 do...while 执行的结果

TIPS

在书写 do...while 程序时,千万不要忘记 while()语句后面有一个“;”,初学者很容易漏掉这个分号,这样会造成编译和运行时报错。

实例 16: 这是一个 while 的循环语句,进行判断累加和不大于 100 的所有自然数,其代码如下:

```
public class DowhileShi
{
    public static void main(String args[])
    {
        int i = 1;
        int sum = 0;
        do
        {
            sum += i++;
        }
        while(i<=100);
        System.out.println(sum);
    }
}
```

从 1 加到 100 执行结果是 5050,下面执行程序,得到结果如图 6-14 所示。

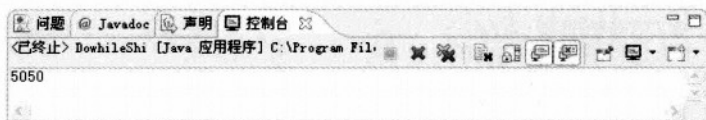


图 6-14 do...while 循环语句

实例探索和读者练习：

在上面的一个实例中，进行条件语句的判断，对条件 do...while 语句进行深层次的学习。下面将展示一段代码，读者看完代码后，给出运行结果，然后再将程序输入到计算机中进行编译，看结果是不是与自己给出的相同，程序如下：

```
public class DowhileTao
{
    public static void main(String[] args)
    {
        int tao=2;
        do
        {
            tao=tao-5;
            System.out.println(tao);
        }
        while (tao>=5);
    }
}
```

6.6 跳转控制语句

在上一课中学习 switch 语句时候，已经接触到跳转语句 break；在这一节中，将会为初学者详细讲解 break 语句以及其他的跳转控制语句。读者学过本节后，将会一一掌握所有的跳转控制语句。

6.6.1 break 跳转语句

break 除了讲解可以在 switch 语句中终止一个语句外，它还有其他的作用，如用来退出一个循环等。break 语句根据用户可分为无标号退出循环和有标号退出循环，下面对它进行一一讲解。

1. 无标号退出循环

无标号退出循环就是直接退出循环，在循环语句中遇到 break 语句后，循环就会立即终止，程序在循环体外面的语句将会重新开始。下面展示一个简单的代码，其代码如下：

```
public class BrankOne {
    public static void main(String args[])
    {
```

```

        for(int d=0;d<9;d++)
        {
            if(d==2)
            {
                break;
            }
            System.out.println(d);
        }
    }
}

```

程序输出的结果是 0, 1, for 循环语句则是从 0 到 9, 当 d=2 时候终止了程序, 不管 for 循环有多少次循环, 它都会在 d=2 时候终止程序, 得到如图 6-15 所示的结果。

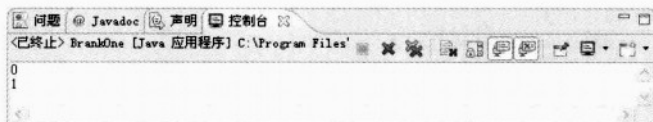


图 6-15 break 语句

初学者也许会有疑问, break 是不是只能用于 for 循环控制语句中? 其他的循环控制语句就不能使用 break 语句了? 答案是其他循环控制语句依然可以使用 break 语句。下面将一一展示在 while 语句中出现 break 语句和在 do...while 语句中出现 break 语句的程序, 先展示在 while 语句中出现 break 语句, 其代码如下:

```

public class BreakTwo {
    public static void main(String args[])
    {
        int d=0;
        while(d<9)
        {
            if(d==2)
            {
                break;
            }
            System.out.println(d);
            d++;
        }
    }
}

```

运行后, 结果如图 6-16 所示。

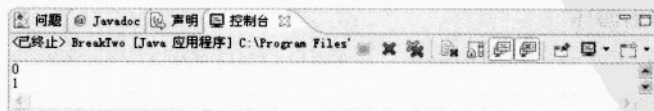


图 6-16 在 while 语句中出现 break 语句的结果

观察了 for 循环和 while 循环控制语句中的 break, 下面将展示在 do...while 语句中出现

break 语句，其代码如下：

```
public class BreakThree
{
    public static void main(String args[])
    {
        int d=0;
        do
        {
            if(d==2)
            {
                break;
            }
            System.out.println(d);
            d++;
        }
        while(d<9);
    }
}
```

执行后，结果如图 6-17 所示。

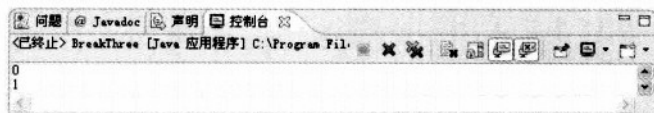


图 6-17 运行后的结果

break 语句除了上面的用法外，还可以用在嵌套语句中，它将终止它所在的循环。先让读者看一下终止内层循环的程序，其代码如下：

```
public class BreakQian
{
    public static void main(String args[])
    {
        for(int i=1;i<10;i++)
        {
            for(int j=i;j<10;j++)
            {
                if(j==5)
                {
                    break;
                }
                System.out.print("☆");
            }
            System.out.println();
        }
    }
}
```

执行后，结果如图 6-18 所示。

PHP

PHP 是一个脚本编程语言。有了 PHP，你可以从 HTML 开始，然后加上标记 (tag) 让你的应用程序连接到数据库或者其他后端系统。标记会被服务器解释并运行，传给客户的时候已经是纯粹的 HTML 了，其作用等同于 JSP。下面是一个 PHP 版本的“Hello, World”的例子：

```
<html>
  <head>
    <title>Hello, world</title>
  </head>
  <body>
    <?php echo '<p>Hello world</p>'; ?>
  </body>
</html>
```

我喜欢

PHP 的成功相当快，几乎都是从 Visual Basic 程序员那里转换而来。它相当适合让网页访问数据库，容易理解，容易学习。PHP 成功地吸引了因为微软进入 .NET 时代而受到遗弃的 Visual Basic 程序员，它对这方面族群的吸引力超越了其他的语言。

我不喜欢

从理论上来看，PHP 很糟糕。模型和用户界面以及数据库紧密地结合，而这通常是很糟糕的，因为任何一者的改变都会影响到其他不相干的部分。因为 PHP 成长得很快速，而且很杂乱，受到了 Perl 很大的影响，所以方法名称常常都不一致，有些选择用下划线当作单词的分隔符 (stream_get_line)，有些直接连接两个单词 (readline)。PHP 的高生产力以及快速创新，付出的代价是成长为一个不一致的语言。身为一个 Java 程序员，大概也看过试着做太多事的 JSP 网页，这种网页写起来很快，但很快就会陷入泥淖而动弹不得。

C# 和 Visual Basic

C# 其实是 Java 的复制品，两者的优缺点都差不多。Visual Basic.NET 似乎失去动力，因为旧的 Visual Basic 开发者对于 VB.NET 失去了以往的热诚。微软还有其他的语言，到最后，微软总是会有一群开发者的，但还是一个封闭的生态系统。这只会受到窗口平台成功的限制，也就是只会在客户端较为

成功，但在服务器端较不成功。我不打算预测成功或失败，我只是说，微软的语言是否成功是由微软平台是否成功来决定的，而不是由语言本身的优缺点来决定的。

Smalltalk

Smalltalk发明在20世纪70年代早期，是一个建立得很好但运气很背的面向对象语言。许多人将 Smalltalk 视为第一个面向对象语言，但是却没有真正获得商业上的成功，尽管直到1995年的时候IBM曾做过一些尝试。Smalltalk 的生产力极高，有一点笨拙且极度奇怪。Smalltalk 的社区活跃，但不大。它将大部分的重心都放在高生产力上。我在第八章会介绍一个名为 Seaside 的延续 (continuation) 应用程序开发框架。

我喜欢

Smalltalk具有相当简洁的对象模型、不可思议的表达力以及一个聪明的设计与社区。它有一些坚固的免费实现版本可以用，也具有一个潜力十足的催化剂 Seaside。Glenn Vanderburg 喜欢说：“一切都将可能回归到 Smalltalk，只是到时候并不叫做 Smalltalk 了。”当你看到像 Ruby 这样的语言深受 Smalltalk 的影响时，你会发现他说的还挺有道理的。

我不喜欢

Smalltalk 不像是一个接班者，因为它不够亲切。如果没有 Java，Smalltalk 本来可以成为 C++ 的接班者。但是 Smalltalk 一直都太贵、与众不同或太不起眼。

没有银弹

你可能注意到，没有一个语言同时具有我们所寻觅的全部特色。这不让人惊讶，如果真有这样的语言，我们早就在用它了。尽管如此，你可以看到这些语言的确各有强项，在后面的章节中，我会更深入地介绍 Ruby。既然一个较好的语言是不够的，我们会随后继续介绍某些具有潜力的杀手级应用。

第六章

Ruby 简介

我站在Watauga河的河岸上，看着16英尺（约4.9m）的州界瀑布（State Line Falls），真是一只Class V（泛舟最高级）怪兽。它的水流中有五块石头，四个急流从石间穿过，其中有三个石缝是不可能穿越的（特别是当水位如此高的时候），第四个尤其湍急。然而，方法相当简单，我认为我应该可以办得到，奋力征服它或者绕过它。我必须作出抉择。

这些年来，我也经历过一些这样的时刻。有时候，我会把我的小舟扛在肩上，绕过它；有些时候，我觉得这条线路不错，我的技巧足以应付这个挑战，所以就奋力一搏。但是这次，我只是站着，不知道该如何是好，任凭瀑布四溅的水花洗涤我的全身。

现在，我正面临相同的状况，我的确认为Java的领导地位已经走到了尽头，至少对应用程序来说是如此。但是万一我真的决定继续往前走，赌注又高得不可思议。我怎样才能知道现在就是正确的时机？我能够挑选到正确的语言吗？我冒了怎样的风险？

我不希望这本书变成各种编程语言的完整浏览。我希望指出一个语言和两个框架（一个用Ruby，一个用Smalltalk）的特殊之处。在本章，我会介绍一个可能成为另一种选择的语言：Ruby。我想要让你看看，某些语言可以超过Java。但是这并不表示Ruby一定会成功，也不表示Ruby是最好的选择。我目前能做的事，就是向你展示一个可能的选择，好让你自己判断这是否合理。

关于 Ruby

Ruby 是一个动态的、纯面向对象的语言，常常被归为脚本编程语言 (scripting language)。对于 Ruby、Smalltalk 以及 Python 来说，用脚本编程 (scripting) 这样的词来描述，实在很不够，所以我要改用应用程序语言 (applications language) 来称呼它。如果你只用过编译式语言 (compiled language)，像 Java 和 C，那么你现在可以开始准备见识不一样的乐趣了，Ruby 会让你放松。我建议你安装它 (到 <http://ruby-lang.org>)，然后开始输入，你可以用它内置的原始 IDE 或使用命令行 (command line)。只要输入 `irb` 就可以启动 Ruby，你会看到下面的提示信息：

```
irb(main):001:0>
```

Ruby 是纯面向对象的

现在，你可以开始动手尝试 Ruby 语句。你会常常利用 `irb` 来研究编写过程中遇到的小问题。在 Ruby 语言中，一切都是对象，如果你键入一个对象，Ruby 会直接将对象返回给你。键入 `4`，然后按下 Enter：

```
irb(main):001:0> 4
=> 4
```

和 Java 不同的是，在 Ruby 中，数字是对象，而不是基本类型 (primitive)。例如，你可以这么做：

```
irb(main):008:0> 4.4765.round
=> 4
```

甚至连 `nil` (无) 都是一个类，代表空无一物。

```
irb(main):009:0> nil.class
=> NilClass
```

你完全不需要担心基本类型、封装 (wrapper) 类这些事情。更重要的，你不需要在 API 内处理这些。因为你不需要处理这些基本类型和基本类型的数组转换的琐事，Ruby 的反射 (reflection)、持久化引擎 (persistence engine)、XML 框架都比 Java 简单多了。

类型

试着不要声明，直接赋值：

```
irb(main):011:0> n=1
=> 1
irb(main):012:0> n.class
=> Fixnum
```

n 有一个 Fixnum 类型的对象，你不需要先声明 n，这强烈暗示你 Ruby 是一个动态的语言。现在，给 n 赋上其他值：

```
irb(main):013:0> n="fish"
=> "fish"
irb(main):014:0> n.class
=> String
```

现在 n 有一个字符串，我们将 n 的变量类型改变了。更准确地说，Ruby 的类型是跟随着对象，而不是跟随着包含者（变量），所以 Ruby 是一种动态类型（dynamically-typed）的语言。让我们试着做一件奇怪的事：

```
irb(main):015:0> n+4
TypeError: cannot convert Fixnum into String
    from (irb):15:in '+'
    from (irb):15
```

Ruby 不会把字符串强迫成 Fixnum，造成类型规则被打破。也就是说，Ruby 是强类型（strongly typed）语言（注 1）。你可以对 n 调用 size 方法，得到字符串长度：

```
irb(main):016:0> n.size
=> 4
```

你如何知道字符串支持哪些方法？问了就知道：

```
irb(main):017:0> n.methods
=> ["send", "%", "rindex", "between?", "reject", "[[]=", "split", "<<",
"object_id", "strip", "size", "singleton_methods", "downcase", "gsub!",
...and so on...
```

看得出来，字符串支持许多方法。你可以用 size 方法来计算它们的个数。如果一直以来你都是使用静态类型（statically typed）语言，你或许会无法体会这里的优点？你听说过动态类型（dynamic typing）可以让你在正确的

注 1：事实上，强类型的说法太过简化。因为你可以随意改变 Ruby 的类型，所以有人认为 Ruby 不算是强类型的。关于这一点，我一直使用本章简化的定义方式。

时候将注意力集中在正确的问题上，减少重构的负担，减少程序代码长度，使代码写起来更快，也更好维护。

条件

Ruby的条件 (conditional) 比较像C，而不是Java。在Ruby语言中，`nil`和`false`计算结果是`false`，其他 (包括`true`) 都是`true`。再一次阅读上面的句子。和C不一样，Ruby的`0`也是`true`。你应该注意到，`false`和`"false"`是不同的，一个是布尔 (Boolean) 常量，另一个是字符串。例如，`puts "It's false." unless "false"`返回`nil`，但是`puts "It's false." unless false`将显示出`"It's false."`。

Ruby还有一些惯例，你应该要知道。`?`和`!`是合法的方法 (method) 名字。习惯上，以`?`结尾的方法，用来做测试。例如，`nil?`用来测试某值是否为`Nil`。以`!`结尾的方法，具有潜在的危险，因为这些方法有副作用 (side effect)。例如，名为`replace(in_string, sub_string, replacement)`的方法可能会返回一个新的字符串，其中`sub_string`被替换了，而`replace!(in_string, sub_string, replacement)`则会直接修改原始字符串。

和Java一样，Ruby有`if`语句，Ruby也支持`unless`语句，作用一样。你可以在使用`if`和`unless`的时候，使用块格式，就和Java一样。你也可以让它们直接作用到行尾，有条件地执行单一行程序代码。所以，你可以这么做：

```
irb(main):099:0> def silence?(b)
irb(main):100:1>   puts  "SCREAM!" unless b
irb(main):101:1> end
=> nil
irb(main):106:0> silence? "False"
=> nil
irb(main):107:0> silence? "false"
=> nil
irb(main):108:0> silence? 0
=> nil
irb(main):109:0> silence? "quit kicking the cat"
=> nil
irb(main):110:0> silence? false
SCREAM!
=> nil
irb(main):111:0> silence? nil
```

```
SCREAM!  
=> nil
```

看看 `silence?` 方法，Ruby 会返回最后一条语句的值，除非方法清楚地指定返回值。在这个例子中，这条语句 `puts "SCREAM!" unless b` 总是返回 `nil`。更重要的，除非你传进 `true`，否则此方法会输出 `SCREAM`。

循环

Ruby 有两种条件循环。你会注意到，许多 Ruby 链接库做完事情之后，都会返回 `nil`。如果你想从标准输入读取数据，你可以这么做：

```
irb(main):010:0> puts line while line=gets  
one  
one  
two  
two  
^Z  
=> nil
```

此循环会持续下去，直到我输入文件尾字符。当然，你也可以把输入流导向文件。而且，你可以在开始的地方使用 `while`，但是要记得在最后放一个 `end`：

```
irb(main):013:0> while line=gets  
irb(main):014:1>   puts line  
irb(main):015:1> end
```

你已经见过 `until`，这是另一种循环的结构 (construct)，运作的方法完全一样，但是 `until` 是在表达式 (expression) 为 `false` 的时候才进行循环。稍后你也会看到 `for` 循环，但这只是语法甜头 (syntactic sugar)。

范围

Java 程序员通常会利用数学表达式来指定范围 (range)，像这样：

```
class Range {  
  public static void main (String args[ ]) {  
    int i = 4;  
    if (2 < i && i < 8) System.out.println("true");  
  }  
}
```


用Ruby也可以做类似的事，但是你还有别的选择：使用Ruby内置的范围支持。`x..y`表示从`x`到`y`的值（包含`x`与`y`）。比方说，`1..3`代表1，2，3。如你所能想象的，在Ruby中，范围（range）也是对象：

```
irb(main):004:0> range=1..3
=> 1..3
irb(main):005:0> range.class
=> Range
```

你可以利用`===`运算符检查是否在某范围内：

```
irb(main):010:0> ('a'..'z') === 'h'
=> true
irb(main):011:0> ('a'..'z') === 'H'
=> false
irb(main):012:0> (1..10) === 5
=> true
```

你也可以利用方便的语法甜头，`for`循环：

```
irb(main):021:0> for c in 'g'..'k'
irb(main):022:1>   puts c
irb(main):023:1> end
g
h
i
j
k
```

`for/in`循环也可以用在Array以及Hash中。`===`运算符是另一种比较方式，接下来，你将会看到第三种比较，称为匹配（match），一般与正则表达式（regular expression）一起使用。

正则表达式

Java具有支持正则表达式的API，Ruby则是将正则表达式建立到语法中，某些人喜欢正则表达式，但也有人不喜欢。对我来说，正则表达式是处理字符串时很重要的技巧。和任何其他类型的编程一样，你也可能会使用过度，如果你用了16个连续的反斜线，差不多是该重构的时候了。但是需要识别字符串模式的时候，使用正则表达式还是比使用手写的程序代码更方便。

在 Ruby 中，你会把正则表达式定义在两个正斜线之间，会用类似下面的方法：

```
irb(main):027:0> regex = /better/
=> /better/
irb(main):028:0> regex.class
=> Regexp
irb(main):029:0> "Mine is bigger" =~ regex
=> nil
irb(main):030:0> "Mine is better" =~ regex
=> 8
```

Ruby 将符合的字符索引值返回。Ruby 的正则表达式威力强大，不是这样一个简单的例子能轻易表明的。我只能说，Java 开发者至少花了一半的时间处理字符串。你想想，servlet、XML 字符串、配置文件（configuration）、部署描述符、应用数据……字符串无所不在。想要有效地解析字符串，你需要内置的模式匹配（pattern matching），例如 Ruby 的正则表达式以及范围。Java 1.5 已经填补了一部分的缺陷，但还不完整。

容器

Ruby 容器（container）就和 Java 的集合（collection）一样，你只看到一个数组。和 Java 一样，数组是对象，`[1, 2, 3].class` 返回 `Array`。和 Java 不同的是，数组内的每个元素都是对象。Ruby 也具有 Hash，类似 Java 的 `HashMap`。Ruby 的 Hash 是对象。和 Java 的 `HashMap` 不同，Ruby 的 Hash 也具有语法甜头。使用花括号取代中括号，使用 `key=>value` 来定义一组 key-value，就像这样：

```
irb(main):011:0> numbers={0=>"zero", 1=>"one", 2=>"two", 3=>"three"}
=> {0=>"zero", 1=>"one", 2=>"two", 3=>"three"}
irb(main):012:0> 4.times {|i| puts numbers[i]}
zero
one
two
three
```

和 Java 集合（collection）一样，Ruby 容器持有对象，而且不需要是同一类型的对象。在 Java 1.5 中，泛型（generics）让你可以建立类型一致的集合。如果要在 Ruby 中做到这一点，你可以修改 `Array` 或 `Hash` 来实现。（记住，

Ruby的任何类都可以被直接修改，Ruby是一个动态的语言。)虽然Ruby不像Java一样有各式各样的容器，但是你会注意到这反而有一些直接的好处：

- 因为不区分基本类型和对象，所以你可以在任何容器内放任何东西，甚至容器内放容器也可以很容易做到。
- 因为object是一切事物继承的根源，所以任何事物都具有散列码(hash code)。
- 语言让你可以在hash和array上使用相同的语法甜头。
- “程序代码块”使得迭代(iteration)较紧凑、容易。

如果你常常使用Java集合，且用过动态语言，你或许会注意到Java的集合常常感觉出错。你必须避免静态类型检查，因为你将某些事物当成对象加入到集合中，而且在从集合取出对象之后，你必须强迫它转换类型。这些事情重复进行既影响性能又笨拙。集合感觉不像是标准的数组，数组可以容纳基本类型。

Ruby容器感觉就不同了。你不需要使用让人发狂的转换类型或者泛型的语法。程序代码块简化了反复的动作。你虽然没有看到许多集合类型，但可不要被这一点给愚弄了，通过各种丰富的方法，数组可以当作list、queue、stack或者任何其他有序集合(ordered collection)来使用。比方说，下面的程序将数组当作堆栈来用：

```
irb(main):001:0> stack=[1,2,3]
=> [1, 2, 3]
irb(main):002:0> stack.push "cat"
=> [1, 2, 3, "cat"]
irb(main):003:0> stack.pop
=> "cat"
irb(main):004:0> stack
=> [1, 2, 3]
```

同样的，你可以在任何需要set、dictionary或任何无序集合(unordered collection)的地方使用Hash。你会发现你花了比较多时间在集合上，而花比较少的时间在自定义的迭代上。

文件

在一个文件中迭代，感觉就像在一个集合内迭代一样。你将会创建一个新的文件，并将它传递给一个程序代码块，比方说，下面是一个简单的 GREG：

```
File.open(ARGV[0]) do |file|
  rx = Regexp.new(ARGV[1])
  while line=file.gets
    puts line if line =~ rx
  end
end
```

将它键入并保存为 *grep.rb* 文件，然后你就可以调用它（在 *irb* 之外），像这样：

```
ruby grep.rb filename regex
```

请注意你没有看到的东西，你不需要关闭文件或管理异常。这个实现确保如果异常发生文件会自动关闭。你有效地使用链接库，此链接库让一切都在“迭代文件的控制循环”之外，Ruby帮你把迭代时的一些麻烦事都处理好了，你只需要用程序代码块来自定义控制循环的内部内容。

你为何要在乎？

现在，你应该已经能感受到 Ruby 的威力与简洁。你或许能够体会程序代码行数的减少，且抽象程度提高。你也许觉得差别不大，大可更依赖开发环境以及程序代码生成工具，像 XDoclet，就可以将自己隔离于问题之外。但是，让我告诉你：程序代码行数才是关键！

- 你仍然必须了解你的工具所产生出来的程序代码。我每年和数十人一起工作，他们不了解 Hibernate 产生的 SQL，有些人则必须维护产生出来的程序代码，将这些程序代码改成符合他们自己的需求。
- 程序代码越多，可能隐藏的 bug 就越多。单元测试能做的只有这么多。你仍然需要深入程序代码，加强它，维护它。
- 写程序不是唯一的成本。培训的成本、维护的成本、扩展的成本都要算进去。
- 不管你用什么程序代码生成器，它都会限制你的灵活性。大多数的 Java 开发者越来越依赖工具帮他们做事，你所采用的每个工具都会提高你的

成本。我习惯使用 IntelliJ IDEA，我的一些客户使用 Eclipse。我对于 Eclipse 不熟悉，所以当我被迫使用 Eclipse 时，我的客户就会因此失去一些优势。XDoclet 让反馈（feedback）过程变长变慢。

- **Java 开发者相当依赖 XML 作为配置（configure）工具。**不要忘了，配置文件也是程序代码的一部分。其他语言的开发者常常发现 Java 对于 XML 配置文件的过度依赖已经造成了困扰。我们使用这么多外部的配置文件是因为用 Java 来配置是很痛苦且琐碎的事。我们使用 XML 进行配置，而不用 Java 自己的 property，是因为……这个嘛，因为在 Java 内使用 XML 很流行。当 Java 开发者们还陷在 XML 配置文件的同时，Ruby 的配置文件简洁又舒服。

你可能愿意为程序代码的行数付出成本，但是你还得考虑到更高的抽象。在 Java 中，你必须使用难看的 iterator（迭代器）。有了 Ruby，你可以将迭代策略建立在容器中，并重复使用此程序逻辑。

换句话说，Java 的用户化通常由外而内发生。你建立一大块可多次使用的程序来填补你的应用程序，但这只是一种用户化策略。对许多工作来说，你会希望工作具有广泛性，只对方法内几行程序用户化就可以了。通过 JDBC 循环、处理文件、处理集合，这些迭代只是这种用户化策略的几个例子。某些 Java 开发者称此策略为控制反转（inversion of control）。

Ruby 让你能够用两种方式写程序，如图 6-1 所示。用此策略所写的程序代码维护起来很容易，而且为你隐藏重复的部分。为了公平，有些 Java 框架（例如 Spring）也为你做一些这样的事。但是要用 Java 语言这么做并不容易，而且这种编程风格很少见，因为这必须使用大量的内部匿名类才办得到。对于动态的语言（像 Ruby 和 Smalltalk）来说，这种编程策略给你很大的自由，不管是在你使用的框架，或者你所建立的框架上，都是如此。

应用某些结构

Ruby 和 Java 都是面向对象语言，都支持单一继承的对象模型。尽管如此，你还是会看到 Ruby 和 Java 之间的许多差异。

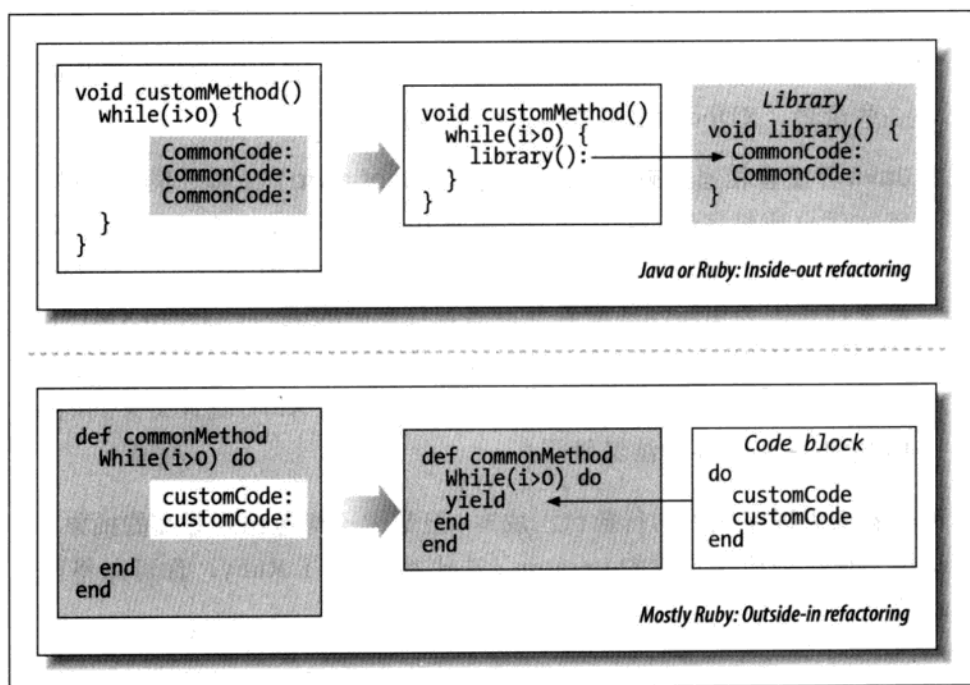


图 6-1: Java 程序员重构循环内部, 程序代码块让 Ruby 开发者也可以重构循环外部

- 在 Java 中, 最小的应用是一个类; 在 Ruby 中, 任何东西都是对象, 你可以估算基本类型、表达式、程序代码块或脚本, 这些通通都是对象, 都是有效的 Ruby 对象。
- 在 Java 中, 类定义是静态的; 在 Ruby 中, 你可以在运行的时候修改类。当你看到一个类的定义, 如果此类已经存在, 新的定义将会修改已经存在的类。
- Ruby 支持 mixin, Java 则没有。把 mixin 想成是一个界面, 加以实现, 你可以将其添加到类中。
- 在 Ruby 中, 一切的东西都会返回一个值, 而此值是动态类型的, 所以你不会看到方法的定义内会出现 return。
- 在 Ruby 中, 方法参数和实例变量都没有类型, 但是实例本身有类型。

在 Ruby 大部分的地方, 你都可以使用你原来在 Java 中的 OO 设计技巧, 你也会见到一些常见的设计规则, 像 MVC。



David Heinemeier Hansson: Ruby

Ruby on Rails 的设计者

David Heinemeier Hansson 是 Basecamp、Backpack 以及 Tada List 的程序员，开发过许多商业产品。但是借由贡献 Rails Web 开发框架以及 Instiki（最受欢迎的 Ruby 应用之一），他也以具体行动支持开放源码。他致力于提升程序员生产力方面的研究，不管是通过软件（像 Rails），还是通过实践（例如 Less Software）。

为什么 Rails 比 Java 更具生产力，要知道两者的架构其实差不多？

DHH: Ruby 让 Rails 能够在运行时实现惯例 (convention)，不仅可以移除不必要的重复，还可以减缓编程周期，免于编译、程序代码产生、部署而陷入泥淖。它具有“修改—重载”的立即性，像 PHP 语言一样，伴随着现代软件工程的技巧（像 domain-driven、test-driven 以及设计模式）。它快速但不会弄脏数据；可升级却不会变重。

对 Rails 而言，Ruby 的哪三个特色是最重要的？

DHH: 首先是 metaprogramming（元编程）。你可以在一个类正在被定义时操控它。你可以创建特定领域的语言，因为你可以类在对象生命周期的任何地方插入程序。这是框架建立者的梦想。

其二，开放的类（open class）。活跃记录（Active Record）包含约 10 层，这些层全都被基类所采用，这使得 API 保持简洁。你不需要使用 10 个不同的类，而 Rails 仍然可以满足维护程序代码的需求。这也会有帮助，能够扩展基类，并在标准的链接库新版本推出之前修正其错误。

其三，任何事物都是对象，但有例外。你可以按顺序在面向对象上工作，但这是一种秩序。它具有难以置信的一致经验，真正达到“最少意外规则”的境界。你可以猜测 Ruby 类行为的名称，猜中的机会很高。

你觉得 Java 有何限制？

DHH: 在“每个语言都能做任何事”的层次上，Java 没有什么天生的限制，但是的确有一些地方会对不同的语言和不同的人有舒适度上的感受差异。我不能够忍受一直重复；我不能够忍受很长的响应周期；我不能够忍受那些计算机应该可以主动理清我的意图的地方，我却得先在头脑中盘算，或者用手写下。

Java 没有让我成为一个快乐的程序员，Ruby 却可以。我不想带着不能让我快乐的工具一起工作。所以，如果 Java 是唯一的选择，我宁可换别的工作，好让我能够使用让我快乐的工具来工作。

Ruby 以及 Rails 已经准备好担负 Web 应用的大任了吗？

DHH: 不仅准备好了，而且许多人都已经这么做了。Basecamp（孕育 Rails 的应用）已经运行一年多了，而且相当成功。Upstarts 选择使用 Ruby on Rails 进行 Web 2.0 的探索，43things.com 以及 Odeo.com 正是其中的两个例子。

类

Ruby 是面向对象的，我已经向你展示过如何使用 Ruby 对象，但是还没让你看到如何创建对象。现在就创建一个名为 Calculator（计算器）的类。创建一个文件名为 *calculator.rb*，内容如下：

```
class Calculator

  def initialize
    @total=0
  end

  def add(x)
    @total += x
  end

  def subtract(x)
    @total -= x
  end

end
```


以上声明了三个方法。当创建新对象时，Ruby会调用initialize进行初始化。请注意，这里的initialize定义了一个名为@total的实例变量。在Ruby语言中，以@开始的是实例变量，以@@开始的是类变量，以\$开始的是全局变量。现在，你可以在irb中加载这个文件，且使用此计算器。

```
irb(main):005:0> require 'Calculator'
=> true
irb(main):006:0> c=Calculator.new
=> #<Calculator:0x28b4a98 @total=0>
irb(main):007:0> c.add 100
=> 100
irb(main):008:0> c.subtract 40
=> 60
```

它奏效了，一切就和你所期望的一样。Ruby开发者运用开放类（open class）的优点。我要改变Calculator的定义，但是别忘了我们还有一个Calculator的实例c。我再次开放了类的定义，方法像下面这样：

```
irb(main):009:0> class Calculator
irb(main):010:1>   def reset
irb(main):011:2>     @total = 0
irb(main):012:2>   end
irb(main):013:1> end
```

这么做就可以加入一个名为reset的方法。这也会影响到c：

```
irb(main):014:0> c.reset
=> 0
```

这真地让人惊讶，不是吗？我把一个现有类的定义改变了，这对于调试、迭代式编程（iterative programming）以及元编程（metaprogramming）都很有帮助。Ruby也让你产生子类，利用<运算符就可以产生子类：

```
irb(main):015:0> class IrsCalculator < Calculator
irb(main):016:1>   def add(x)
irb(main):017:2>     x = x / 2 if x>0
irb(main):018:2>     super
irb(main):019:2>   end
irb(main):020:1> end
=> nil
```

你可以使用add，而IrsCalculator会帮你将增加值减掉一半：

```
irb(main):027:0> c=IrsCalculator.new
=> #<IrsCalculator:0x28b6b80 @total=0>
```

```
irb(main):028:0> c.add 100
=> 50
```

这样的概念对你来说应该不陌生。类将实例数据与方法封装在一起。类的实例就是对象，每个类都有一个父（parent）类（Object 类是唯一的例外），所有的类追溯源头都继承自 Object。

```
irb(main):031:0> Class.superclass
=> Module
irb(main):032:0> Module.superclass
=> Object
irb(main):033:0> Object.superclass
=> nil
```

使用 Mixin

为了要实现混合（mixin），Ruby 使用一个称为模块（module）的概念。模块让你将方法（method）和类聚合在一起。你不能够实例化一个模块，模块不能独自存在。模块不是类，但是有自己的名称空间。模块构成类和混合器的基础。

混合器并非新的概念，早在 1971 年 Smalltalk 就支持混合器。混合器是实现的接口，这意味着你可以将许多类可能会用到的方法集合在一起。

看看这个小例子。为了要建立一个最友善的范例，你可能想建立一个混合器，根据名字来向任何对象打招呼。程序是这样的：

```
irb(main):021:0> module Greetable
irb(main):022:1>   def greet
irb(main):023:2>     puts "Hello, " + self.name
irb(main):024:2>   end
irb(main):025:1> end
=> nil
```

然后，你可以把这个程序代码引入（include）一个名为 Person 的类中：

```
irb(main):011:0> class Person
irb(main):012:1>   include Greetable
irb(main):013:1>   def initialize(name, age)
irb(main):014:2>     @name=name
irb(main):015:2>     @age=age
irb(main):016:2>   end
irb(main):017:1>   attr_reader :name
```

```
irb(main):018:1> end
=> nil
```

你可以在 Person 内使用这个程序代码：

```
irb(main):039:0> person=Person.new("Bruce",40)
=> #<Person:0x2a970a0 @age=40, @name="Bruce">
irb(main):040:0> person.greet
Hello, Bruce
=> nil
```

虽然混合器似乎很有趣，但是你可能会觉得这个程序代码有点不对劲。除非你可以更好地整合混合器内的 Person 方法，否则这只会带来不好的设计决策：你可能会将和 Person 完全不相关的东西塞进 Person 中。但其实这样的威力相当大，你可以将一个 aspect 或者一种能力，分离地进入一个混合器中。混合器之所以威力很大，是因为：你也可以在你的模块中访问 Person 的类方法。事实上，我们在模块中用到 Person.name，而这还是在我们定义 Person 之前。如果这让你觉得困惑，看看下面的例子。inspect 是一个类方法，用来将对象组织成字符串格式：

```
irb(main):147:0> module Reversible
irb(main):148:1>   def inspect
irb(main):149:2>     super.reverse
irb(main):150:2>   end
irb(main):151:1> end
=> nil
```

请注意，你还没定义类，但已经在用 inspect 类方法了。这可能让你觉得有点奇怪，但是将此模块包含进我们稍早定义的 Calculator 类中，一切就很自然了：

```
irb(main):152:0> class Person
irb(main):153:1>   include Reversible
irb(main):154:1> end
=> Person
```

你已经将模块包含进来，它有一个类，现在是一个混合器。你可以调用它所定义的任何实例变量，它会假设类是你所加进去的那个类。用到具体实例中，看看会发生什么：

```
irb(main):155:0> p=Person.new("Bruce", 40)
=> ">"e curB"=eman@ ,04=ega@ 0711c82x0:nosreP<#
```

当你实例化对象时，`irb`会调用`inspect`。你看到下面混乱的那一行吗？其实它是“`Person:0x28c1170 @age=40, @name=\"Bruce\"`”的反序，让人印象深刻。现在你可以加入一个能够检查（`inspect`）类的混合器，且将类内最隐私的细节交给混合器。这样的整合甚至可以在类尚未存在前就进行。我能够用 `mixin` 来处理安全性（`security`）或持久性（`persistence`）。Java 程序员常常使用 AOP 以达到混合器的功效。

拦截器

我说过，这些日子以来Java框架开发者在“改变现有类的行为，却不需要改变程序代码”的技术上，持续有所收获。其中一个技术是拦截（`interception`）。JBoss 与 Spring 使用方法拦截（`method interception`）来获取隶属于POJO的任意服务。利用 Ruby，拦截轻而易举，就是把一个方法改名，将另一个方法放在它的位置上（请见图 6-2）。

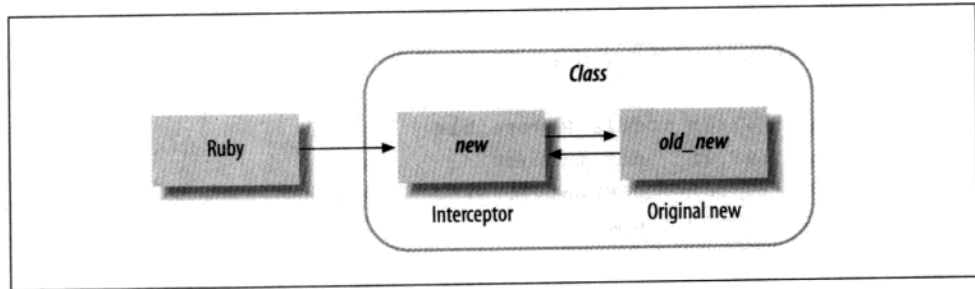


图 6-2：在 Ruby 中想做方法拦截，就是把方法改名并用新的方法取代之，再于新的方法中调用旧的方法

比方说，我的朋友 Dave Thomas 要求我在他的 Ruby 盛大演讲开始之前，先帮他看着计算机几分钟。我可以使用他的 Ruby 外壳（`shell`），然后输入我从他的《`Programming Ruby`》学来的这一小段程序代码。这个版本的程序代码拦截 `new` 方法，如图 6-2 所示，我只是将原来的 `new` 改名，并让新的 `new` 调用旧的 `new`。拦截器用来输出信息，告知我们 Ruby 建立了新的对象。就是这么简单：

```
class Class
  alias_method :original_new, :new
  def new(*args)
    result = original_new(*args)
    print "Unattended laptop error. "
```

```
    return result
  end
end
```

当 Dave 回到他的课堂时，一旦他创建对象（对 Ruby 来说，这等于是不管建立任何东西），他将会感受到这样的惊喜：

```
irb(main):009:0> i=[1,2,3]
Unattended laptop error. Unattended laptop error. Unattended laptop error.
Unattended laptop error. Unattended laptop error. Unattended laptop error.
Unattended laptop error. Unattended laptop error. Unattended laptop error.
Unattended laptop error. Irb(main):010:0>
```

这是一个 8 行程序代码的中断器，如果你知道哪 10 个对象被创建，你就太厉害了。你没有用到任何的 Java 代理器（proxy）、程序代码生成器（code generation）或者 AOP（aspect-oriented programming）。当然，你不会想真的尝试这么做，这就像在爱因斯坦的车内丢进一个鞭炮。就和 Albert（爱因斯坦的名字）与原子一样，你不会在不知道能量会往何处散发的情况下，释放出这样的威力。

AOP

Java 开发者越来越依赖 AOP。AOP 让你可以在 POJO 中加入服务，不需要修改任何程序代码。AOP 帮助你控制应用程序的流程，在感兴趣的点（例如，方法执行前后）加入自定义的方法。特别是，你将会常常看到 AOP 用来：

调试或日志

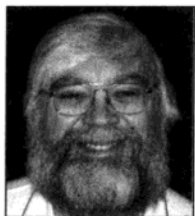
AOP 让你可以用很少的语法，在任何你需要的地方加入调试或日志的能力。

声明服务

EJB 使用容器（container）来提供服务，你将会利用配置来指定服务，而不是利用写程序的方式。轻量级的容器，做的事情和 AOP 一样。你将会常常看到拦截器管理事务、安全、遥控（remoting）。

混合器

Java 没有提供混合器（mixin），但是你可以用 AOP 来模拟。



David Heinemeier Hansson 与 Jim Weirich, 两位 Ruby 专家: Ruby 的 AOP

Jim Weirich 是 Compuware 的软件顾问。他设计过用来测试喷射机引擎的实时数据系统、信息系统的网络软件、金融工程的图像处理软件。Jim 在 Ruby 社团中相当活跃, 贡献了数个 Ruby 计划, 包括 Rake 与 RubyGems。

为何 AOP 在 Ruby 中没有被采用?

DHH: 标准化的 AOP 框架未曾真正地在 Ruby 中被采用, 因为语言本身已经支持许多 AOP 所能实现的功能。下面是一个 Action Pack 的范例, Rails 的 controller/view 部分。后面跟着程序代码块, 将版面配置 (layout) 的功能注入到原始的交付 (render) 方法中:

```
base.class_eval do
  alias_method :render_without_layout, :render
  alias_method :render, :render_with_layout
end
```

所以, 我们将原始的 `render` 方法改名为 `render_without_layout`, 然后我们能够从加强版的 `render_with_layout` 中调用此方法。最后, 我们让这个改进的 `render_with_layout` 方法取代 `render` 的位置。所以, 我们热衷于交换 (hot-swapping) 基类的行为, 使其具有改进的功能, 同时不改变公共的接口, 且不会把基类搞乱。下一个版本的 Ruby 将会让这些更进一步, 在语言中加入类似 AOP 的构造函数 (construct), 有着 pre (前)、post (后) 以及 wrap (封装) 条件。

JW: Ruby 元编程 (metaprogramming) 的能力如此接近于表面化, 且对于一般的 Ruby 程序员来说, 容易取用。我怀疑 AOP 试图解决的大多数问题在 Ruby 中都是利用元编程来解决的。

这是标准库中的一个范例。Date 对象不可以改变, 一旦你计算某个 Date 对象是星期几, 可以将结果保存起来, 以

后调用时就直接返回，而不需要计算。检查之前计算过的值的程序代码写起来很简单，但是在Date内的13个方法中都要实现这样的做法的话也很繁琐。

Date类的作者采用这样的方法：他将每个方法写得仿佛每次被调用时都会计算（也就是说，不会特别检查之前的值）。然后他写了一个类方法，名为once，可以获取方法名称列表。once方法这么做：为有名称的方法创建别名，将此别名定义为私有的(private)，然后用原始名称建立一个新的方法，此新方法在调用旧方法（利用别名）之前会先检查之前的计算值。换句话说，once方法重写现有的方法以计算一次其返回值，然后将结果保存起来。

进一步查看如何用Ruby的元编程方式来很容易地做到“类AOP”的解决方案。AspectR链接库在Ruby内加入某些简单的AOP操作，虽然它不如Java世界的AspectJ完整，但是链接库本身的大小就差了许多倍，AspectR只用了210行左右的程序代码。

我不是AOP专家，但是我看到AOP所定位的问题的范围比较窄，Ruby使用的元编程解决方案反倒比较宽。或许不是Ruby需要AOP，而是Java需要元编程！

当然，AOP是一个相当广为使用的工具，而且如果它成功的话，很明显地将会有典型的使用案例出现，不管是在范围还是威力方面。虽然，目前Java开发者常常通过框架（例如Spring）来发挥AOP的威力，你可以把拦截器当成用来完成同类事情的一个更基本的工具。JBoss框架与HiveMind容器使用拦截器以提供各式各样的服务。对于Ruby开发者来说，AOP并不急需，因为你已经具有稳定的工具，可以处理这些问题：

- 你可以使用拦截器于任何时间在任何对象内加入服务。这很容易就可以办到，差不多就是“把方法改名并写一个新的方法”这样。
- 你可以使用混合器(mixin)，甚至在运行时把它们加入。例如，你可以轻易地让所有的方法在一个领域模型安全中。

- 你可以使用挂钩 (hook)。Ruby 提供了挂钩, 让你可以在特定位置注入 (inject) 你的程序代码。下一个版本的 Ruby 将会支持名为 `__before`、`__after`、`__wrap` 的挂钩。

简单来说, Ruby 能够用非 AOP 的手段解决许多“类 AOP”的问题, 且会在不久的将来加入“类 AOP”的功能。某些 Ruby 程序员可能会担心 AOP 程序代码可能不容易维护。AOP 的核心价值, 即 Ruby 尚未支持的那部分, 就是有效且快速地指定切入点 (point cut), 只要你有需要, 你就可以使用正则表达式来定义拦截器。Ruby 已经有一个核心功能, 可以让切入点容易实现:

- 你可以快速地询问对象支持哪些方法。
- 你可以匹配正则表达式。
- 你可以用字符串来调用方法。
- 很快地 (在 Ruby 2.0), 你就可以用 `before`、`after`、`wrap` 来挂钩 Ruby 方法。
- Ruby 很方便被配置设定 (configure)。你可以轻易地在 Ruby 中指定切入点, 不需要 XML 或者全新的语法, 像 AspectJ。

Ruby 具备这些能力, 使得 AOP 相形之下变成非常轻量级。现在, Ruby 开发者倾向于用最符合 Ruby 架构的方式, 用片段组合出“类 AOP”的功能。

依赖注入

Java 和 Ruby 之间不同的依赖注入 (dependency injection) 让 Java 开发者难以理解。在 Java 中, 依赖注入正快速地改变我们建立应用程序的方式。这是相对简单的概念:

```
class Speaker {
    void speak(String words) {
        System.out.println(words);
    }
}

class Consumer {
    Speaker mySpeaker;
    void saySomething() {
        mySpeaker.speak("something");
    }
}
```



```
    }  
}
```

注意Consumer。它没有将Speaker予以实例化,这个工作是由第三方(third party)所进行的,我们将它称为Container(容器):

```
class Container {  
    public static void main(String[] args) {  
        Speaker speaker=new Speaker();  
        Consumer consumer=new Consumer();  
        consumer.mySpeaker = speaker;  
        consumer.saySomething();  
    }  
}
```

你可以做出一些简单的改进,可以用getter和setter方法将mySpeaker封装起来,然后可以选取(extract)出一个名为Speaker的接口并实现FrenchSpeaker、EnglishSpeaker以及SpanishSpeaker。你也能够用Java或XML做出配置文件,描述所有你想要以这种方式对待的对象。

对于基本依赖注入容器(dependency injection container)来说,你几乎已经有了所需要的一切:配置、第三方的生命周期控制以及解开Speaker和Consumer之间结合的能力。通过依赖注入容器,你也可以改变Speaker的实现,而无需改变任何consumer的程序代码。你可以注入Speaker的一个测试实现,而不会影响到基础程序代码,这是进行Java测试优先开发(test-first development)的一个相当重要的技巧。对于配置,你也有一致的策略可用。

当你看到Ruby的依赖注入时,需要面对几件事。首先,Java在配置方面的能力并不好,但是Ruby却可以让你在结构数据上表现得相当好,且不像XML的语法那么具有侵略性。你也可以通过瞬间改变类定义来解决许多结合(coupling)的问题。比方说,注入这些模仿的(mock)对象到难以到达的地方会比较容易。

某些Ruby开发者似乎认为依赖注入很重要,也认为一段时间后这将会成为Ruby的主流之一。Ruby有一个出色的依赖注入框架,称为Needles,这件事应该不会让你感到惊讶。

其他人则认为依赖注入应该在某些地方发生,而不是在一个单一的、一般目的的框架中。因为动态地改变类定义很容易,你可以轻易地注入你所需要的

行为，而不会增加另一层跨越应用的复杂度。大多数的 Ruby 程序员似乎趋向于这个想法，Ruby 的整体动态设计使得依赖注入变得完全没有必要，最复杂的应用除外。（请见花絮“Jim Weirich、Jamis Buck、David Heinemeier Hansson，三个 Ruby 专家谈论：Ruby 需要依赖注入吗？”）。



Jim Weirich、Jamis Buck、David Heinemeier Hansson，三位 Ruby 专家谈论：Ruby 需要依赖注入吗？

Jamis Buck 从 2001 年年底开始写 Ruby 程序，写过许多东西，其中包括两个 Ruby 依赖注入容器、Ruby 对 SQLite 和 SQLite 的绑定以及 Ruby 的 SSH 客户端链接库。在这些日子中（以及无数个夜晚），他用 Ruby on Rails 写程序。Jamis 和他的妻子及两个小孩住在犹他州的 Provo。

Ruby 需要依赖注入吗？

JW: 问题不应该是“Ruby 是否需要依赖注入”，而是“是否特定应用与框架需要依赖注入”。DI（依赖注入）是一个工具，用来帮助你建立分离的程序，但是这会伴随着某种概念性的负担而来。当软件结合让你无法负荷额外的复杂度时，你会开始使用 DI。使用某些语言的话，这种痛苦会很快就来，但是因为 Ruby 是动态类型的语言，这种痛苦会减轻点，所以依赖注入的优点会在价值曲线的后面才发生。除此之外，有一些很简单的技术可以用来降低结合的成本，而不需要使用正式的依赖注入框架。

最后，当你的 Ruby 应用或框架已经到达对依赖注入感兴趣的阶段，你会发现这很简单，不到 30 行的代码就可以满足大部分人的需求。

JB: 一两年，我为 Ruby 写了两个非常不同的 DI 容器。这两个容器都被我发展到极致，我也很努力地在 Ruby 社区中耕耘 DI。但是后来，我真正了解到一点：Ruby 的动态天性，让 DI 被 Java 所需要的大部分因素都消失无踪了。

DHH: 我们真地研究过在 Rails 和 Needles 中建立配置服务,但是我所想用 DI 解决的问题,其实可以用更简单的方式解决。比方说,运用依赖注入技术,使得注入这些模仿的 (mock) 对象到难以到达的地方会比较容易。如果有一个付款 (Payment) 类,初始化一个付款网关 (Payment Gateway) 以对信用卡进行授权与收费。如果没有 DI,看起来会是这样的:

```
class Payment < ActiveRecord::Base
  belongs_to :credit_card
  def capture
    PaymentGateway.capture(amount, credit_card)
  end
end
```

现在,在一个语言中(像 Java)直接使用 PaymentGateway 是一个比较不好的依赖于实例类的方法,因为这不容易模仿 (mock) 和测试。但 Ruby 就不一样了, Rails 尤其是如此,因为它对于此类型的模仿 (mock) 具有完美的支持。我们能够消灭真正需要被消灭的方法,而不会误删其他的。对于此付款类来说,我们这么做:

```
require "original/payment_gateway"

class PaymentGateway
  attr_accessor :desired_result

  def self.capture(amount, credit_card)
    Response.new(desired_result)
  end
end
```

我们现在能够指定 PaymentGateway.desired_result = :success,且局部模仿的 PaymentGateway 将会按照我们的意思,使用所有剩下的基础设施,不会实际调用远程的系统。

Breaking It Down

这是 30 分钟的 Ruby 向导。我没有说 Ruby 是下一个成功的语言,但是我要

说的是，Ruby 让一些 Java 中棘手的问题变得很容易。越来越多顶尖的独立顾问正在想办法利用 Ruby 以及其他动态的语言来赚钱。Java 社区花了相当多的金钱和精力要让 Java 更动态。依赖注入以及 aspect-oriented programming (AOP) 对 Java 来说都是具有相当开创性的，而且也开始具有商业上的吸引力。对于 Java 开发者来说，这些想法代表的是更好的透明程度以及应用程序的开发变得较简单。

崩溃，是因为抽象的重量？

我在研究 Ruby 的这段时间里，产生了一个很有威力的念头，它越来越清晰。当我们把 Java 扩展到令人吃惊的不自然方向时，我们也会付出代价。AOP 以及依赖注入对 Ruby 来说简直是无关痛痒，但是对 Java 开发者来说，却得为此学习新的编程模型、处理 XML、引入复杂的语法。每个新的元编程概念被导入 Java 中，就会让人觉得这些复杂度不知要将我们带往何方。这样的净效益把 Java 推到企业利基市场上，离我们一般人越来越远。而在 Ruby 中，依赖注入和 AOP 却不会是我们的焦点；你需要这些概念的时候也可以采用，没有问题。

我的确相信 Ruby 加上 Rails 的框架简直是我们所追求的理想解决方案：Web 前端，加上关系型数据库后端。我也说过，在商业应用上使用 Ruby，生产力和价钱都能达到客户的要求，我相信这是用别的技术达不到的。我还是建议许多客户使用 Java，因为他们所需要的复杂框架是 Ruby 目前没有的，或者因为他们的核心开发者都已经熟悉 Java，或者因为他们已经有太多 Java 的程序代码，用 Ruby 改写并不合乎效益。

在后面的章节，我会把这些论点一一展现在你的面前，从无到有地建立一个 Web 应用程序，以用来访问关系型数据库。然后，我会展示对另一个语言来说，杀手级的应用程序可能是什么。

Ruby on Rails

当我放声大叫沿着三英尺的岩架上山的时候，我脑海中的一个声音告诉我“别抗拒，去征服它吧！”知识渊博的登山骑士称此举为冲刺（lunge），但是我尚未为其命名，也不习惯用这样的词汇。有人说“靠近岩架时，骑士应该相信他的车子可以无所不能，绝对不会坠毁”，我的脑袋却抗拒这种愚笨至极的想法，但是我看过成功的例子。我高速进行到岩架，我相信车子可以办得到，把手中的油门一加，车子腾空飞起。在某种程度上，我不相信有机会成功。虽然我还是安全地登顶、下车，但心里仍肯定我会失败。这个想法就像是想通过用力向上拉扯鞋带，而让自己飞起来一样。学会这种神秘的冲刺需要花上一点时间。

就和冲刺一样，元编程（metaprogramming）对我来说也有一点不自然。不过，毕竟我的职业生涯中的大部分时间都在写Java和C的程序。如果你想要体验元编程框架的威力，Rails是最好的选择。

数字游戏

身为一个很有自信的Java程序员，我并没有在寻找其他替代品，但是Rails在冥冥之中却找到我了。Dave Thomas和我在同一个研讨会中发表演讲，我讲了数场，主题都是Spring和Hibernate，而我对自己的生产力很满意。当然，和EJB相较之下，我是很有生产力。但是Dave指出就算用Hibernate和Spring，你还是在许多地方一再重复地做相同的事。

我想了想 David 的评论，为了要做出持久 (persistent) 领域模型，需要以字段和索引来指定数据库表，以类 (重复了) 和字段当作属性 (重复了) 来指定对象领域模型，并为此字段添加访问器 (一再重复)。然后，你需要为数据库表 (重复了) 和类名 (重复了) 建立一个映射 (mapping)。最后，你的映射必须指定每个数据库栏 (column) 以及对应的数据库字段 (每个字段再重复两次)。当然，大多数神智清楚的 Java 程序员不会做所有这些重复的工作，他们会用工具来帮他们做事。但是现在你的编程模型影响了你的工具选择、你的开发经验以及产生了更多行的程序代码必须维护。我下了一个结论，当领域模型以及对象模型差异很大时，ORM 就显得很有道理，所以我当时认为生产力有些许影响是没关系的，因为可以换来较好的执行性能，甚至较好的映射。

被蒙蔽的视野

我说过，我在一家公司上班，这家公司为一家制造厂建立安全软件。我们有效地建立一个 Web 用户界面，以管理复杂的领域模型。我们决定使用 Spring、Hibernate、Web Work 的轻量级 Java 组合建立这个系统。我们进展得很快，我们为进度感到高兴，为成就感到满足，因为我们只用 4 个星期的时间，就用 Java 改写了一个原本花一年时间用微软技术写出来的系统。自然地，当我们累积到一个较大的程序代码基础时，写程序的速度就开始变慢。

一段时间之后，Dave 的论调不断在我的脑海中浮现，Dave 和我的业务伙伴也说过一样的话。幸运地，在同一个星期，我们试着用 Rails 建立部分的应用程序，我们花了两三天的时间将一切准备好。靠着扎实的 HTML 经验，Justin 的进度比我还快，他只花了 4 个晚上的时间就把整个系统用 Rails 做出来了。Rails 版本让我们大吃一惊的还不只这些，它的执行速度快得很！



Justin Gehtland: Ruby on Rails 案例探讨 《轻快的 Java》作者之一

Justin Gehtland 是 Relevance 的共同创办人之一。这是一个顾问与训练组织，地点在北加州的 Durham。他是 Jolt 得奖书籍《轻快的 Java》的共同作者，从 20 世纪 90 年代早期开始至今已经开发过各种大小的应用程序。过去的 6 年中，他用 Java、.NET、LAMP 开发系统，现在用 Ruby on Rails。

你最近把一个 Java 项目移到 Rails 上, 这个系统原本使用什么 Java 框架?

这次的经验, 你最吃惊的是什么事?

Rails 已经准备好向 Java 争夺地盘了吗?

让 Ruby 如此高生产力的三个主要原因是什么?

JG: 原本使用一些轻量级的产品: Spring 以及 Hibernate, 加上一些前端的 JSTL (为了让客户更容易修改界面)。我使用 ACEGI 安全框架来处理身份确认和授权, 但是只进行本地数据库账号内的身份确认。

JG: 在将系统移植完后, 讲到经验, 我真正吃惊的是大家对此讨论不休。关于破坏性的技术, 我们总是可以回头去用锯子; 如果此讨论有什么结论的话, 那就是 Rail 很清楚地是一项我们可以采用的技术。

从技术的观点而言, 我很惊讶我竟然如此高效率地达到这样的成果。Rails 版本的系统很快, 甚至比原先的 Java 版本更快。一方面是因为在做 Java 版本的过程中累积了对问题领域的了解, 另一方面是因为不需要进行各种软件之间的性能调节, 最大的因素应该就是用 Rails 本来就很容易达到性能提升。网页和活动缓存策略表面上正确, 容易管理它们的生命周期。我能够让 Web 服务器全力服务网页, 这是我所能达到速度上的极限。

JG: 在笔直的道路上, Ruby 不会跑得比 Java 快, 毕竟 JVM 是一再微调过的, 对于 bytecode 的执行已经达到最优化。但是 Ruby on Rails 却是相当擅长转弯的, 它整合了许多框架, 是动态语言且不需要编写/编译/部署/测试这样的冗长流程。对于我们的系统来说, 这就像是越野车和跑车, 跑车虽然马力十足, 跑得很快, 但在颠颇的山路上完全不适用。

JG: 我必须说, Rails 致力于聪明的默认做法, 是 Ruby on Rails 的主要优点。Ben Galbraith 这么说过好几次, 我也同意他的话。因为 Rails 总是让你可以覆盖掉它的默认做法, 所以你不会陷入动弹不得的危机中。但最棒的是, 你可以建立一个系统, 将配置文件会遇到的问题在一瞬间用另一个框架来解决掉 90%。

第二, 我真的感到很惊讶, 没想到缺乏配置/编译/部署/测试周期, 会有这么大的差异。保存修改, 执行测试, 用

浏览器重新加载，这简直一气呵成，我不知道这是否真的让我更有生产力，但是这的确让我感觉到更有生产力。

最近，Ruby 动态的特质真的为我带来许多好处。我的应用系统需要一些常用的功能，这些功能应该归属于框架层次。不需要深入源代码就能加入这些新功能，我就只是在运行时动态地扩展我需要的类。这种扩展性是许多静态类型语言所诅咒的（且不会这么做的）。

作出承诺

当然，向顾客展示原型并试图让顾客从验证过的Java技术转换到一个不知名的框架和一个不知名的语言，这大概不是一件简单的工作。我和新创公司的老板有过对话，结果让我大为惊讶。他一口答应，准备采用。我想，或许我不应该这么惊讶，我们不能低估高生产力的诱惑。

在全新的系统开发上，生产力是大家关心的要点。因为你的资源常常比不上你的竞争者，所以你需要快速。你可以花更多时间工作并减少官僚作风，让你的系统开发速度加快。但是如果你可以用技术产生优势，你一定要掌握这样的机会。在一个全球化竞争的时代，我们需要更像是新创公司。如果一个框架可以让你仅变快 20%，或许你应该继续使用比较保险的语言，像 Java。但是如果可以变快 300%，甚至更高，那么其他的差异就都变得不重要了。

请注意，我的前提是Java正在远离原来的基地。我们大多数的人需要在关系型数据库上建立 Web 应用系统。语言主题很重要，但是 Java 的设计者太偏重于企业问题的骨干，导致核心问题变得复杂。即便 Rails 没有填补这个问题，其他的也会。

一些数字

我要让你看一些关于性能和生产力的数字，我承认这些数字并不完善，这有许多原因。有些方面，Rails 比较占上风：

- Ruby 应用系统实现了更多自定义的需求。当 Justin 发现他的经验很重要之前，他已经实现出某些功能——Java 版本无法拥有的功能。

- Justin 被认定是 Java 专家，但从未在项目中用过 Ruby，也从未用过 Rails。他写过一本关于 Spring 的书，他每年教为期两周的 Hibernate 课程 16 次。
- Rails 框架有一些设计上的哲学，对于 Java 开发者来说，是相当不一样的做法。

更重要的，有些因素在实现上和 Java 相反：

- Java 程序代码没有被完全调整。Java 的调整相当不容易，进展不多。我们才开始看性能表现。(Ruby 的程序代码也没有被完全调整，但是默认的实现性能就已相当不错，只有一些小困扰。)
- 我们已经实现过一次这个应用程序，所以 Ruby 版本的实现有经验上的优势。在应用结构上面巨大的差异会影响到结果，但是用户界面几乎是不变的。
- Justin 没有机会在所有的环境实现所有可能的调整情况，此 Java 版本在 Apple iBook 上面执行 Tomcat，而不是 Resin 或更快的服务器软件。Justin 仅进行了几个测试。
- 缓存模型相当不同，在 Rails 中进行微调要容易得多。

结果，用 Ruby 还是比较快，生产力大概是原来的四倍到五倍。表 7-1 展示了生产力的量度值。我们写的程序代码更少，需要维护的程序代码也就更少，时间大幅缩短，客户更加满意，我们有更充裕的时间处理客户最后加入的改变需求，我们的测试程序代码功能不逊以往，甚至可能超越以往。

表 7-1：生产力量度

量度	Java Spring/Hibernate	Ruby Rails
交付时间	4 个月，每周近 20 小时	4 个晚上（每晚 5 小时）
程序代码行数	3293	1164
配置行数	1161	113
类数量 / 方法数量	62/549	55/126

表 7-2 展示了性能上的数字，这可能会更有争议。我不会告诉你 Ruby 应用程序一定比 Java 应用程序执行速度快，我只是要展示，在这个案例中，Ruby 已经够快了，而且要达到这样的成果，不需要太多时间和经验，一般程序员都能办到。

表 7-2: 将未经调整的 Java 应用程序移植到 Ruby on Rails 之后所展现出来的性能差异

量度 (每秒请求数)	Java Spring/Hibernate	Ruby Rails
用户情况 1 (100 次) (没有预先存在的缓存)	71.89	75.59
用户情况 1 (100 次) (有预先存在的缓存)	80.86	174.39
用户情况 2 (100 次) (没有预先存在的缓存)	80.86	62.50
用户情况 2 (100 次) (有预先存在的缓存)	88.97	1785.15

我强调过，Justin 绝对没有说他已经把 Java 应用程序调整到尽善尽美了，这些数据的重点是让 Rails 到这样高效率的地步，几乎不需要花力气调节。但是调整 Java 却需要许多技巧、时间、努力。Ruby 版本执行得很快，只要一点点额外的努力，就可以满足需求。

社区响应

Justin 将他的遭遇发表在两个网络日志中 (注 1)，后来又公布了支持他的论调的数据。这个时候，Java 社区火力旺盛，群起攻击。这实在太讽刺了，因为 Justin 对于数据完全诚实，而且他只有在被社区质疑时，才被动地公布数据。你应该可以想象，这种反作用力相当大。在这个例子中，反作用力可以好好地接受公评，因为在许多利基点上，Rails 是 Java 相当可怕的竞争对手，

注 1: Justin Gehtland, Weblogs for Relevance, LLC (April 2005); http://www.relevancellc.com/blogs.1*heart*rails;SomeNumbersatLast.

而且会越来越强。如果 Rails 真的发生了，许多知识将会迅速地被排挤出主流。

听好，我没说这份数据相当科学、完整、广泛可行的。这只是反映出我们的经验，也因此，你可以相信这样的数据。这份数据告诉我 Rails 是很有生产力的，可以让我们很快地把事情做完，产生较少的程序代码，也比较容易调整。此数据没有证明，但是提供了一些强有力的暗示。对于范围相当大的应用来说，Rails 比 Java 更有生产力。Rails 可以用继承和关系来处理复杂领域，Rails 已经足以完成工作。

保持开放的心，然后自行判断。

Rails 范例

了解 Rails 最好的方法是看看它是怎么运作的。到 <http://rubyforge.org> 下载 Ruby 以及 RubyGems。(如果你使用 Windows 版的安装程序，里面直接就包含了 RubyGems)。如果你还没有关系型数据库管理系统，就去下载一套回来，我使用 MySQL。然后你就可以马上体验到 Rails。RubyGems 让你可以安装 Ruby 应用程序以及它的相关程序。在命令行 (shell) 下输入：

```
gem install rails -v 0.12.1
```

Ruby 将会开始安装程序，连接到 RubyForge (rubyforge.org) 并取回适当版本的 Rails 索引，如果你省略版本号码，Ruby 将会给你最新的稳定版本。针对每种依赖性，RubyGems 都会先问你，你可以回答“Y”或者“a”表示对所有的依赖都是这个答案：

```
Attempting remote installation of 'rails'
Updating Gem source index for: http://gems.rubyforge.org
Install required dependency rake? [Yn] Y
Install required dependency activesupport? [Yn] Y
Install required dependency activerecord? [Yn] Y
Install required dependency actionpack? [Yn] Y
Install required dependency actionmailer? [Yn] Y
Install required dependency actionwebservice? [Yn] Y
Successfully installed rails, version 0.12.1
```

你会注意到 RubyGems 随即试着为每个子组件和 Rails 建立文档，一切就是这样简单，Rails 安装完毕。你已经开始体会到 Rails 亲切好用的一面了！

生成一个基本的应用程序

现在你可以生成一个 Rails 项目。进入你的工作目录，要求 Rails 生成一个项目，名为 *trails*：

```
rails trails
```

Ruby 创建了一个完整的目录结构以容纳你的应用程序。不会有不明朗而需要猜测的地方，因为所有的 Rails 项目都有一致的格式。我会指出一些重要的目录：

app

这个目录放置你的应用程序代码。你将会看到 MVC 的每个组件以及其他组件都各有一个目录。

config

此目录内东西会很少。你会放进任何需要特别配置的东西，像数据库的连接参数。因为 Ruby 善于使用默认配置，所以你的 *config* 目录下将会很空。

script

你的 *trails* 应用程序伴随着许多脚本而来。这些脚本能帮助你产生程序代码，并开启你的应用服务器。

你会注意到一些其他的好东西，但是现在，让我们就用一个脚本启动 Ruby 应用服务器。切换到 *trails* 目录并输入：

```
ruby script/server
```

如果顺利的话，你会看到服务器在 3000 端口 (port) 的位置开启服务器。你可以用浏览器连接到 <http://127.0.0.1:3000/>。确定此服务器运作正常后，你将会看到 Rails 的欢迎信息。你刚刚已经开始了一个 Ruby 的 Web 服务器的开发以及 Rails 的配置设定。如果你需要改变服务器的某些属性，改变 *script/server* 脚本就行了。请注意，Ruby 程序员通常会用 Ruby 脚本来进行配置设定，就像此服务器脚本一样。你已经知道 Ruby 很适合处理有结构的数据，不需要用 XML。比方说，下面是服务器脚本的一部分，具有配置选项 (option)：

```
OPTIONS = {  
  :port      => 3000,
```

```

:ip => "0.0.0.0",
:environment => "development",
:server_root => File.expand_path(File.dirname(__FILE__) + "../public/"),
:server_type => WEBrick::SimpleServer
}

```

此程序代码定义了一个名为 `OPTIONS` 的 hash 映射。`=>` 运算符 (operator) 将左边的值对应到右边。虽然还没发生什么事,但是你应该知道,其实你在很短的时间之内,就已经将许多底层的配置都设定好了。

我们的 *trails* 项目将会收集 mountain bike trails 的描述,我们先从简单的开始,收集 ID 以及 trail 名称、描述、困难度。你将会输入一次字段名称, Rails 的元编程特色将会从数据库读取 column, 且动态地在你的模型对象中添加属性。如果你使用 MySQL, 你可以启用 `mysql` 命令处理器。创建一个数据库, 将其命名为 *trails*, 然后进入此数据库。创建一个名为 `trails` 的表:

```

mysql> CREATE TABLE trails (
->   id int(6) NOT NULL auto_increment,
->   name varchar(20),
->   description text,
->   difficulty varchar(20),
->   primary key (id));
Query OK, 0 rows affected (0.36 sec)

```

请注意名称, 名称很重要。习惯上, 如果超过一行 (像在表中或者在 list 中), 名称就应该用复数名词。字段 (column) 或类如果代表单一对象, 就应该用单数名词。Rails 很聪明, 懂得翻译英文复数, 所以它会为名为 *People* 的表创建一个名为 *Person* 的模型。请看看这些范例的大小写, 这也很重要。如果你按照 Rails 惯例来命名, 你就可以享受 Rails 默认的一切, 你的程序代码就可以更简洁。

你需要告诉 Rails 到哪里去找你的数据库。编辑 *config/database.yml*, 像下面这样:

```

development:
  adapter: mysql
  database: trails
  host: localhost
  username: root
  password: password

```

停止服务器并重新启动（只有在改变数据库配置的情况下需要这么做）。让我们生成一个简单的模型。在 *trails* 目录下输入：

```
ruby script/generate model trail
```

Rails 生成模型、一些辅助文件、测试以及固定设备（*fixture*）。比方说，你可以看看此模型。编辑 *app/models/trail.rb* 文件：

```
class Trail < ActiveRecord::Base
end
```

这的确很虎头蛇尾，看起来就像是你就是在此输入的自定义的程序代码，希望 Rails 会在其他地方生成剩下的程序代码。但是事情不是这样的。运行时，Rails 会加载 *ActiveRecord* 基类，Rails 会看类的名称并加载名为 *trails* 的表的定义。然后，它会动态地在此 *Trail* 基类中加入属性、*getter*、*setter* 以及数据库访问方法。实际比我们看到的还多。

Rails 所生成的脚本之一让你通过 *irb* 来操作你的模型。输入：

```
ruby script/console
```

你现在就可以轻易地处理你的模型。例如，你可以说：

```
Trail.new do |trail|
  trail.name="Walnut Creek"
  trail.description="Meandering trail in Austin park"
  trail.difficulty="hard"
  trail.save
end
```

现在，你需要一个控制器（*controller*）。你也可以生成这部分：

```
ruby script/generate controller trails
ruby script/generate model trails
```

你会为一个 *trails* 的集合页（*collective page*）创建一个模型（*model*）和一个默认的控制器（在 *app/controllers/trails_controller.rb* 中）。控制器原本是空的，把它编辑成这样：

```
class TrailsController < ApplicationController
  def index
    render_text "This will be a trail someday..."
  end
end
```

用浏览器去看此 URL: <http://localhost:3000/trail>, 你会看到信息被显示出来。当然, 用 Rails 来输出信息这一点小伎俩是不会让你满足的, 接下来你可以将控制器改成这样:

```
class TrailsController < ApplicationController
  scaffold :trails
end
```

scaffold是一个方法, 第一个参数是:trails, 这是一个直接量 (literal), 指向Trails类。将它保存下来, 并用浏览器加载相同的URL。现在更有趣了。你会看到一系列的trail, 点击底部的新的trail链接就更有兴趣了, 你会看到一个表, 就和图 7-1 一样。你会看到此元编程框架正在“加班”。此scaffold方法推测数据库的性质 (property) 并通过此模型来传播它们, 传到上面的用户界面。稍后你就会看到 scaffold 控制器到底是怎么办到的, 现在就姑且将这一切当成神奇的魔术吧!

管理关系并更新视图

一系列的trail并无法让你多做多少事情。在对象之间的交互让事情变得相当困难。假设你想要通过其城市的方式访问trail, 你要做的第一件事就是为地点 (location) 生成模型。首先你需要一个数据库表:

```
mysql> CREATE TABLE locations (
->   id int(6) NOT NULL auto_increment,
->   city varchar(20),
->   state varchar(20),
->   primary key (id));
Query OK, 0 rows affected (0.35 sec)
```

不光是动态地构建 scaffold, 你可以同时生成控制器 (controller) 和视图 (view) 的源代码, 搭配建立 scaffold (用 `ruby script/generate scaffold locations`), 用 `ruby script/generate model location` 为单一地点建立模型。当你在内部时, 不妨详细看看屏幕后面的情况, 用 `ruby script/generate scaffold trails` 来为Trail做相同的事。将浏览器指向 <http://localhost:3000/locations>, 确定一切都顺利, 然后再加几个地点。我自己加入了德克萨斯州的 Austin 和科罗拉多州的 Durango。

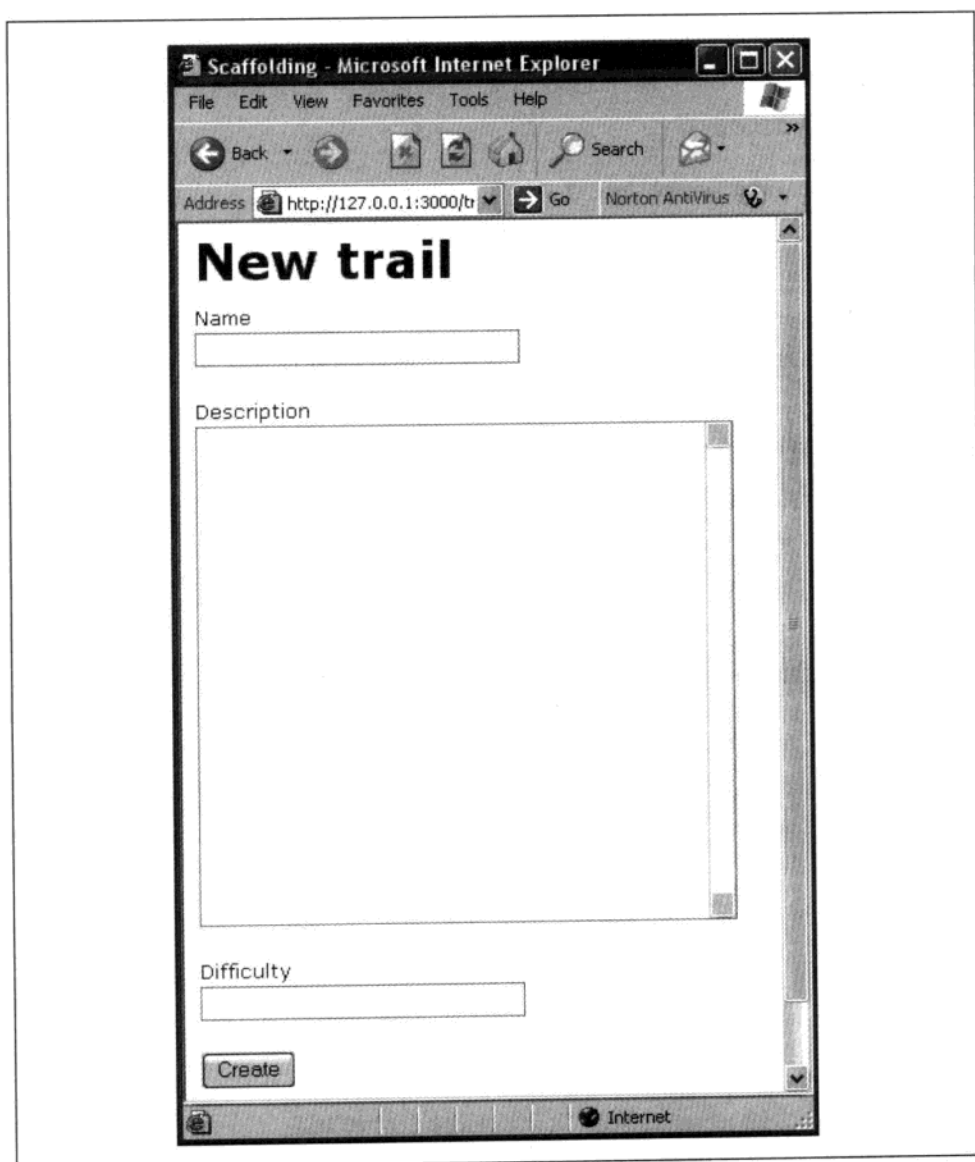


图 7-1：此应用程序的代码和配置加起来不到 10 行，因为 Rails 从数据库主动推测出结构

Rails 已经帮我们做很多事了，现在我们自己来写一些程序吧！你需要更新你的 Trails 表来指向新 locations 表的正确 row，方法是加入新的数据库 column，此 column 指向 location_id。如下所示：

```
alter table trails add location_id int(6);
```


你也需要告诉Rails这个关系。修改此Trails模型以及Location模型，以反映此新关系。如下所示：

```
class Trails < ActiveRecord::Base
  belongs_to :location
end

class Locations < ActiveRecord::Base
  has_many :trails
end
```

有个小描述很有趣。你已经创建Base模块中的ActiveRecord类的子类，然后你触发（fire）一个名为belongs_to的方法并传给它一个Locations类的符号。此方法将会触发更多的元编程程序代码，将诸多性质（property）和方法（method）加入到你的程序中以帮你管理关系。

接着，你需要去编辑Trails视图（view）和控制器（controller）以编辑地点（location）。建立scaffold的时候，分别在trails和locations中创建了新的控制器和视图。

我们现在来修改某些视图程序代码。视图程序代码包含了HTML，<%和%>里面混合了Ruby程序。首先，你必须确定view具有你需要的一切信息。你将在控制器的edit方法中做这件事。改变trails_controller.rb中的edit方法，创建一个名为@locations的性质（property），用来持有所有的地点（location）：

```
class TrailsController < ApplicationController
  ...
  def edit
    @trail = Trail.find(@params[:id])
    @locations = Location.find_all
  end
  ...
end
```

现在让我们来看看用来编辑路径的视图的部分。你希望用户从所有可能地点的“挑选列表”中挑选一个位置。把app/views/trails/edit.rhtml修改成这样：

```
<html>
  <head><title>Edit a Trail</title></head>
  <body>
    <h1>Edit Trail</h1>
```

```

<form action="../update" method="POST">
  <input id="trial_id" name="trail[id]" size="20"
    type="hidden" value="<%= @recipe.id %>" />
  <p><b>Name</b><br>
  <input id="trail_name" name="trail[name]" size="20"
    type="text" value="<%= @trail.name %>" />
  </p>
  <p><b>Location:</b><br>

  <%= collection_select("trail", "location_id", @locations,
    "id","city") %>

  <p><b>Description</b><br>
  <textarea cols="40" id="trail_description"
    name="trail[description]"
    rows="20" wrap="virtual">
    <%= @trail.description %>
  </textarea> </p>
  <input type="submit" value="Update" />
</form>

<a href="/trail/show/<%= @trail.id %>">
  Show
</a> |
  <a href="/trail/list">
    Back
  </a>

</body>
</html>

```

就和大多数的应用程序一样，你的scaffold不会一直都架在那里。你常常会将大部分的view程序代码改掉。Rails让你可以继续往你的目标构筑，而不是将所有模型（model）、视图（view）以及控制器（controller）都固定不变。

注意用黑体字印刷的程序代码，这里加入了所有地点的选项值（这是你在控制器的edit方法中指定的）并选择一个能够符合反射模型的，其中模型显示在trails.location.city变量中。

最后，你需要显示trail列表中和show方法中的新数据，方法完全都一样。在show.rhtml视图中加入一行，就在页面底部的链接上面：

```

<p>
  <b>Location:</b> <%=h @trail.location.city %>
</p>

```

很简单吧！控制器传入模型给你，然后你从模型中取得地点。此列表视图使用相同的技巧。你可以从 `app/views/trails/list` 视图中编辑表：

```
<table>
  <tr>
    <th>Name</th>
    <th>Location</th>
  </tr>

  <% for trail in @trails %>
    <tr>
      <td><%= trail.name %></td>
      <td><%= trail.location.city %></td>
      <td><%= link_to 'Show', :action => 'show', :id => trail %></td>
      <td><%= link_to 'Edit', :action => 'edit', :id => trail %></td>
      <td><%= link_to 'Destroy', :action => 'destroy', :id => trail %>, :
confirm => "Are you sure?" %></td>
    </tr>
  <% end %>
</table>
```

图 7-2 显示了执行结果，有着主要列表中每个 trail 的地点。别忘了，所有的 trails 都必须有地点。如果没有的话，你就会看到错误信息。

Listing trails		
Name	Location	
Barton Creek	Austin	Show Edit Destroy
Emma Long	Austin	Show Edit Destroy
Headlands	Austin	Show Edit Destroy
Hermosa Creek	Durango	Show Edit Destroy
New trail		

图 7-2：此列表来自一个应用程序（此应用程序可以让你观看并更新数据库），其中 trail 和 location 分属两个不同的表格

此教学课程已经够长了，我希望你能够欣赏到 Rails 开发的威力与流畅。你可以很快地让你的应用程序动起来，因为 Rails 会主动地从数据库设计中发现你的应用结构。然后你可以很快地动手修改，因为 Rails 的响应周期（feedback cycle）很短，只需要“写程序/重载”。你所建立的应用程序的质量超越 PHP 所能给你的，因为你使用的是经过证实的好东西——model/view/controller（MVC）设计模式，还有内置的特色进行日志记录、缓存以

及自动测试。现在你已经看到 Rails 的能耐了，你可以打开引擎盖，查看一下内部的玄妙之处。

查看内部

你已经看过，Rails 框架也是由数个现有的框架所组成的，包括 Active Record、Action Pack 及其他。Active Record 负责处理关系型数据库；Action Pack 处理请求以及管理 MVC 的分配。Rails 集成了这些功能并提供其他的功能。

Active Record

Active Record 实现了 Martin Fowler 在《Patterns of Enterprise Application Architecture》(Addison Wesley 出版)一书中提到的 Active Record 设计模式。这其实是一个将数据库表封装起来的工具，同时将域逻辑 (domain logic) 封装进来。Rails 实现加入了两个重要的创新：你可以继承并管理关系，当然还有一些其他的特色。

自动性质

Active Record 利用访问器 (accessor) 自动加入性质 (property) 到塑模对象。它也为 CRUD 数据库方法自动地加入一些方法。比方说，在你刚刚写过的视图 (view) 中，尽管根模型是空的，view 还是会去访问 Trail 中的 name 属性：

```
class Trail < ActiveRecord::Base
end
```

关联管理

Rails 自动使用方法来增加管理关联的方法。看看下面的例子，一个 location 有许多 trail：

```
class Location < ActiveRecord::Base
  has_many :trails
end
```

如你所见，has_many 是一个方法，而 :trails 是一个符号 (symbol)，在这个例子中，就是代表 Ruby 类 Trails。

构建

你可以使用 Active Record 从多个表中构建对象，就像这样：

```
class Location < ActiveRecord::Base
  composed_of :street, :class_name => "Street",
             :mapping => %w(street name)
end
```

继承

可以用继承将所有的子类和父类一起放在一个单一表中：

```
class Product < ActiveRecord::Base
end

class Bike < Product
end
```

其他特色

当然，完整的 Active Record 讨论超出本书的范围，但 Active Record 的确是很好用的技术。你可以建立递归关系，例如树状结构。你可以使用 Active Record 来验证某些规则（例如：对新的山径来说，地点必须已经存在）。重大事件发生时，Active Record 能够通过电子邮件进行通知。

Active Record 也具有相当好的内部设计，它支持事务（transaction）以及错误日志（logging），可以灵活运用表字段的元数据（metadata），也支持多种数据库类型。它也提供让你容易建立测试设备（test fixtures）的支持。Active Record 是一个威力强大的框架，是 Java ORM 框架的竞争对手。

Action Pack

Action Pack 把请求分成两部分：控制器和视图。请求被分配器送进 Action Pack。分配器将请求送到控制器，而控制器会调用（invoke）模型的逻辑，并将请求送给以模板驱动的（template-driven）视图系统。此模板引擎会触发 Ruby 模板，可能会执行 Ruby 程序代码并将得到的 HTML 结果交给浏览器。这样的流程显示在图 7-3 中，不禁让人想起 Struts。但是它和 Struts 之间是有差异的，比方说，控制器有一群动作（action），而不是将每个动作封装进入不同的类中。如果你想重构，你必须让动作共享方法。

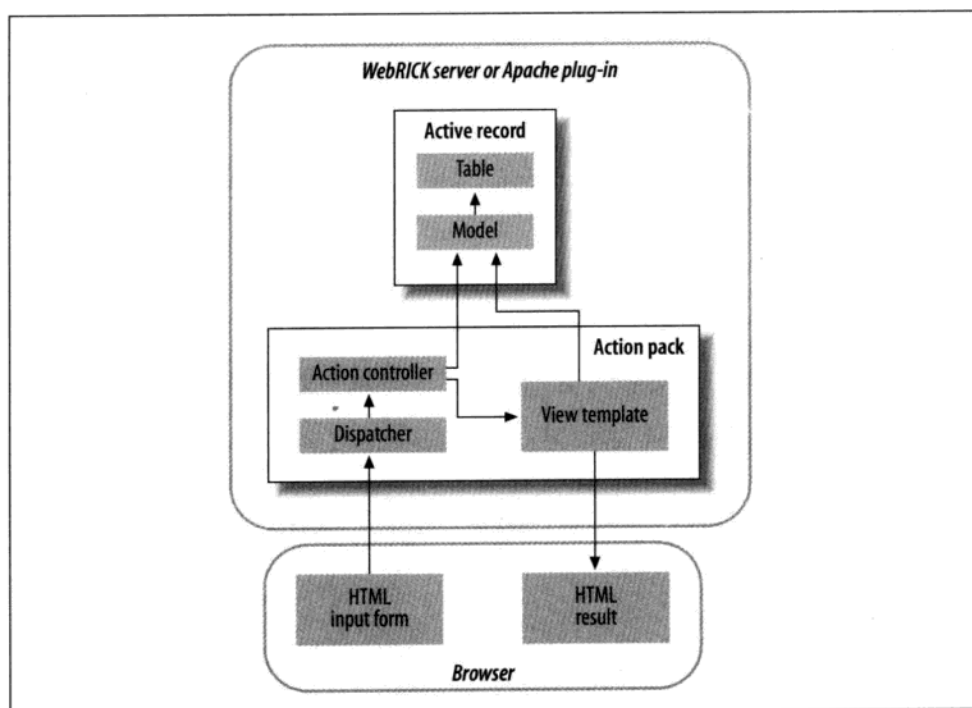


图 7-3: Ruby on Rails 是由数个现有的框架所构架出来的, 主要是 Active Record 以及 Action Pack

Action Pack 将请求分离成控制器的部分或者视图的部分。利用 Rails, 许多事情会自动发生。在某些方面, 这不见得好, 你无法在你的类中看到所有的方法和属性, 除非你先查看了数据库, 否则你甚至不知道这些方法和属性是什么。另一方面, 这对生产力的帮助很大, 你可以很轻易地改变你的模型、模式 (schema) 以及视图, 只要改变数据库就好了。让我们详细看看 Action Pack 能够做哪些事。

功能

Action Pack 不只是简单的请求处理, 它甚至具有让 Web 应用程序开发变得很容易的功能。我在这里会接触到一些主要的功能。

你已经看过 Action Pack 使用 Ruby 作为脚本编程语言。Java 开发者认为将 Java 程序嵌入到 JSP 网页中是不好的做法, 但是我认为, 不管是不是用 Ruby, 程序代码都会在视图 (view) 内。早些年, 某些狂热者对于 Java scriptlet 早期的繁殖反应过度, 因此马上颁布 MVC, 并言 MVC 就是“页面内完全没有

程序代码”。许多Ruby开发者却相信，处理视图（且不涉及其他）的程序代码当然是属于页面的一部分。将Java程序代码埋在自定义的标记（tag）内，只会让事情变得复杂、混乱。

举例来说，Ruby的脚本编程语言就比JSTL标记容易使用。就像servlet，Action Pack让你可以附带过滤器（例如身份确认）。Action Pack也处理某些方便的设计元素，像为结果自动标上页码、提供翻页的超链接。

Action Pack还具有一些特色，使得组件的建立更容易，像辅助类（比方说，用来显示日期）、版面配置共享的特色（类似Struts的Tiles）、组件之间的沟通以及和Ajax相当好的集成。就像Struts和Spring一样，Action Pack提供很良好的支持，让我们可以建立表单、验证表单。

你需要管理你的方案，而Action Pack内置的一些功能可以帮你。它可以进行日志记录、在三层次结构中缓存（页面、动作、片段）以及基准测试支持。开发者可以对单元测试和调试使用集成支持。在某些方面，它不像Struts那么威力强大，但是很简单，也相当容易自定义。

精华

也就是说，Rails不是玩具，也不是华而不实的东西。我认为，Rails代表向前跨越很大的进步，相当先进。你大概看过一些这类的框架，用来解决数据库和用户界面的一些问题，但做法不一样：

- 面向对象开发框架，灵活、稳定，它们通常在较低的抽象层，所以生产力可能不是很好。你可以用它们来建立灵活、稳定、强大的应用程序，但是必须付出低生产力的代价。
- “快速妥协框架”牺牲一些智能和稳定的设计来换取生产力的提升。比方说，PHP和Visual Basic就牺牲了设计上的智能（框架应该要鼓励模型和视图逻辑的分割）而迁就开发速度。
- “程序代码生成框架”在编译的时候生成大部分的程序代码。它们牺牲响应周期（feedback cycle）以及维护的容易性，甚至包括自定义的能力来换取速度。

- “自定义点框架”需要一些参数，像数据库表或模型，并建立附带一些定义明确的预期自定义点 (customization point) 的默认实现。当框架设计者没有预料到重要的挂钩点 (hook point) 时，用户完全没有办法处理。

Rails 不属于上述的任何一者，它会根据数据库的结构以及一些正确放置的宏，使用宏来帮你很快地生成程序代码。因为是在运行时生成程序代码，所以你不会看到一复杂的源代码，当然也无需维护这些额外的程序代码。Rails 利用 Ruby 的扩展挂钩点 (extensive hook point) 避免陷入自定义点的陷阱。你一开始会看到很清楚的设计，然后会用继承的方法（或者任何我们讨论过的其他元编程技巧）去扩展它，改变类定义。你甚至能够将主要的 Rails 组件取代，像 Active Record。

Rails 通过有意义的惯例和默认情况来加速你的开发。通过将你的命名策略放进数据库，Rails 可以帮你省下许多打字的时间，并从你所提供的名称推断出你的意图。

Rails 提供脚本和调试工具，让开发更方便。你可以用脚本做许多事，像运行服务器、管理活动记录 (active record) 类和 console 数据库表、使用生成的测试设施 (test fixture) 或者从生成的脚本运行性能测试。

在《Hackers and paints》(O'Reilly 出版) 一书中，Paul Graham 认为给好程序员用的好工具必须是由程序员设计的，因为只有程序员才知道程序员每天都在解决哪些问题。我觉得他说得不无道理。或许 Rails 之所以这么好，一开始是因为作者建立它以解决他们自己的问题。如你所见，建立 Rails 的目的是要帮助建立受欢迎的 Base Camp 以及 Back Pack 项目。

Rails 是杀手级的应用吗？

Rails 是帮我们超越 Java 的催化剂吗？我不确定。Ruby 在商业支持方面并不强。目前还没有 JVM 版本可以运行 Rails，而现有的项目已经有了某些不正确的开端。Ruby 没有丰富的框架，也没有 Python 以及 Java 的名称识别机制，但是 Ruby 在生产方面的确相当进步，这是相当大的利基。而且不像 Python、Groovy、Lisp 那样，Rails 已经吸引了 Java 社区。像 Rails 这样的东西，可能最终会在 Web 开发的利基市场上取代 Java。

或者元编程才是杀手级的技术？

另一个层面，Rails 可能用了一个杀手级的技术。Rails 展示出 Ruby 元编程（结合有意义的预定机制）在商业上的成功应用。让我们深入一点探讨元编程。

在某些方面，此编程技术让我想起另一个热门词：*domain specific language (DSL)*。一个 DSL 解决某个领域的问题，使用适合该领域的语法和关键字。再看一次 Active Record，这个框架让你表达出你对于数据库和模型之间的关系的概念，你可以使用一些特殊关键字和语法，像表达继承、关系、名称映射等概念。

Rails 可能是打破水坝的应用，我的一些技术导师，像 Stuart Halloway 以及 Glenn Vanderburg，常常会提到这些技术的重要性。通过展示 Ruby，Rails 可能会释放出元编程框架的巨大能量波，使得大家纷纷建立用于各个不同领域的框架。如果我们真的看到这样的风潮，这不会发生在 Java 上，因为 Java 的反射 (reflective) 机制用起来太痛苦了，而且加上基本类型和对象的混合，这一切都太繁琐了。

对于 Ruby and Rails 最后的一些想法

对我来说，Ruby 感觉上像一个好语言，而 Rails 感觉则很特别，这并不足以让它成功。在这个行业中，个体常常会有不同，而两个 David（也就是 Thomas 和 Hansson）或许够特别，可以将这个语言和框架带进主流。Dave Thomas 孜孜不倦地提倡各种务实的事物，而且他似乎将出版的重心都放在 Ruby 上。他已经锁定许多顶尖的 Ruby 作者，对他们很好，分享他的出版经验，这是大出版商所不愿意的。大量印刷书籍的问世，是技术成功的必要关键。David Heinemeier Hansson 具有独到的全面技术，他不但了解终端用户，也具有营销的热情（这不多见）。Rails 对许多 Web 开发者来说是个一看就觉得理论可行的技术，相当具有吸引力，而且也简单好用，足以让大家都为之着迷。

领导常常是技术成功或失败的关键。技术热情和市场愿望被融合在一个单一头脑中，这不多见，但是一旦你能做到这样，好事就会到来。Bill Gates 靠着车库内的 BASIC 创立微软，然后变成一家大公司。Steve Jobs 让苹果很酷，离开苹果，然后再回来修复苹果的形象，将苹果再度带回市场。Java 也

是，其中充满了许多技术幻想人士。James Duncan Davidson 对抗 Sun 的官僚作风，通过开放源码，Tomcat 将 servlet 变成主流，后来在 Ant 上又发生一次相同的事。

Java 似乎正在失去这些幻想人士以及我最敬重的技术专家。Glenn Vanderburg 或许靠着 Java 可以帮他付一些账单，但是其实他现在公开在 Smalltalk (Seaside) 和 Ruby 社区上面花大部分的时间，因为他对元编程感兴趣。James Duncan Davidson 几年前离开 Java 社区，将焦点放在苹果的 MacOS X，主要是 Objective C 语言。许多人之所以还会和 Java 保持关系，是因为 Java 可以让他们赚钱付账单。

Ruby and Rails 似乎要往另一个方向走，渐渐地 Rails 会来到争议的十字路口，你大概已经听到有人质疑：

- 它能应付大型的需求吗？
- 它适合企业使用吗？
- 没有这些 Java 程序员和链接库，你该怎么办？
- Rails 只是个玩具吧？
- 整个业务系统用脚本编程语言来写，你没搞错吧？

在 2005 年的上半年，我看到有超过 20 个网络日志上在攻击 Rails。有些论调有凭有据，Java 的确是能够做一些 Ruby 做不到的事；有些论调则是愚昧或错误的。我感到很好奇，因为越来越多的 Java 社区都开始注意到 Rails。两个 David 的确是把 Rails 的讯息传递到 Java 社区了，我们接着就看看舞台聚光灯是否会提供足够的成长能量，或者是会让它闷热、枯萎、凋零。

延续服务器

我很少会因为盲目的信念而去泛舟。如果有危险的话，我会想知道水和石头会对我造成什么影响，而且我需要规划处理任何潜在的难题。这一天，尽管失败的代价很高，但是一切做起来都很容易。我仍然不知道为何会成功，但是我看到划船的人经历推、撑而到达 Elbow 的动荡水域，这是一个有 20 英尺落差的沟槽，导游手册说这是一个死亡诱捕点。当然，我大可告诉你这样的举动是一个过沟行动，我需要采用支撑的手法，在正确的时间点，用正确的角度，避免在下落的时候撞到下方的墙。我知道正确的时间点，因为有人告诉过我，至于“为什么”这样做就不得而知了。专家试着告诉我为什么这样可行，但其实大多数的人不知道，没有人可以真正告诉我底下的石头是如何分布的。他们只知道在这条河流的层次，这么做就对了，而且也真的是这样。

在我编程生涯中的一些不同时间点，几个花招对我来说也是相当神秘的：当我还是大学生的时候，觉得递归（recursion）很神秘；第一次看到反射（reflection）的时候；还有现在遇到延续服务器（continuation server）的时候。在本章，你会看到延续服务器以及如何使用它们。

问题

尽管 Web 很有用，但是 Web 开发却常常是不优雅的。就好像制造香肠一样，我喜欢结果，但是我不想看到香肠的制作过程。Java 的 Web 编程比其他语言的 Web 编程好，具有更多的结构，更容易维护，而且常常比 Visual Basic

或 Perl 更适合于大型应用程序且比 C++ 更容易。尽管优点如此多，但特定的一些问题却让它沉重而笨拙。

你想要怎样

目前的 Web 应用服务器或许威力强大，但是它们不方便也不自然。所以，到底自然和方便是什么？这不会很难说明。万一你的控制器看起来这样：

```
if (logon.show() == true) {  
    mainPage.show();  
}
```

或这样：

```
if (shoppingCart.verify()) checkout.show();
```

这就好多了。你真的想要做的是将一个或多个 Web 画面的表现方式封装在一个方法 (method) 中。接下来，更多复杂的页面流程可能会遇到困难。你可以将越来越多页封装成较高级的组件。比方说，下面的程序代码：

```
checkoutAddress.showForm();  
if (checkoutAddress.getSeparateBilling)  
    checkoutBilling.showForm();  
creditCardNumber.showForm();
```

将它包成一个方法：

```
public static void showCheckoutWizard() {  
    checkoutAddress.showForm();  
    if (checkoutAddress.getSeparateBilling) checkoutBilling.showForm();  
    creditCardNumber.showForm();  
}
```

所以使用方式就变成：

```
cart.showCheckoutWizard();
```

这种写法既简洁而且是重构的。但是你不能这么写，因为你的 Web 服务器不让你这么做。对于大部分 Web 应用服务器的建立者来说，他们会愿意出卖灵魂来保持事情无状态 (stateless) 且可调整规模 (scalable)，这是他们相当坚持的。

无状态

试想，如果我们生活中完全没有短期记忆，会怎样？正常的日常生活对话几乎变成不可能，看看下面的做法：

- 你必须一边对话一边写下对话中每一件重要的事。
- 然后，当有人问你一个问题时，你必须看看这张对话的历史记录，然后才可以回答。
- 为了要优化事情，你必须决定哪些信息要放在比较近的地方（能比较快取得）。比方说，放在公文包内，而不是放在家中的文件柜。
- 当信息变得太旧时，你就需要丢弃该信息。
- 你必须维护整个系统并在它不合用时重新修改它。

这就是 Web 开发者的现况。你的公文包就是 HTTP session，而你家里的文件柜就是关系型数据库。这真是疯狂，但是因为 Web 很重要，而且无状态的解决方案比较容易调整规模，所以我们只好自己来处理这些冗长繁琐的过程。所以，你心甘情愿地踏出后退的一大步，离理想的解决方案更远了。但每次在你和这些笨拙的用法奋战时，你都会问自己是否有更好的方法，某种抽象可以让这个问题更清爽。

“上一页”按钮

将简单会话的内部状态存储起来并不会涵盖全部的问题。在 Web 中，会话不是线性的。用户可以改变心意，按下“上一页”按钮。如果你假设数据是单纯地累积增加，这个时候就行不通了。

有时候，你想要让用户不能回到上一页，例如，当她已经买东西了，或者做某件事造成数据库内的数据已经改变了。在这些例子中，你可能直接禁止按下“上一页”按钮。通常，你需要对“上一页”按钮建立特殊的支持，你可能甚至得从数据库移除用户从来没看过的数据。更糟的是，许多 Web 设计者就是不去解决这个问题，而是告诉用户去做一些不直观的动作。你退回一步，离理想的做法更遥远一些。再说一次，这个恐怖的“上一页”按钮让我们回到了每个案例都要个别处理的情况，这实在是不对的。

导航

关于Java的Web开发,数量相当惊人的脑力将焦点放在导航(navigation)和流程(flow)。你认为从服务器端控制流程是很自然的,但是服务器不能更新客户端,服务器只能被动地响应请求。用这么简单的方式,服务器得处理数百个小小的请求,而不是数十个应用程序流程。用户界面和模型(数据)之间的同步也变得相当困难。你想使用简单的方法调用来控制用户界面和模型,但是你就是办不到。Web服务器就是不这么做。你又往回退一步,而你已经开始怀疑,悬崖是否就在你背后的不远处。

延续服务器来解救你

一个称为延续服务器(continuation server)的新品种Web服务器已经出现了,人们开始注意到它。延续服务器使用一种名为延续(continuation)的编程构造函数(construct),帮助提供关于请求的足够信息,让我们能完成重新构建环境(context)。从技术上来看,一个“延续”可以存储执行环境,包括call stack。实际上来看,在Web服务器上使用“延续”技术,可以让服务器帮你维持环境,让你获得自由,可以用比较自然的方式写程序。

延续

你大概玩过计算机游戏,可以把“延续”想成是“存储游戏进度”的功能。当你玩游戏时,你可以把当前的游戏存储起来,你可以在以后和大怪兽再来一次激战。如果你“死”了,可以回到以前存储的进度。换句话说,一个延续是某个时间点的状态保留。延续可以让你将系统的状态(系统的执行堆栈)记录在一个地方,然后下命令就可以回到该状态。

既然我已经介绍过Ruby的语法,在本章我会先介绍Ruby的延续,Ruby的延续语法既简单又精确。然后我会向你展示Seaside,这是最普及的延续服务器,是用Smalltalk开发的。

在Ruby中,程序代码块定义了延续的一切。你使用“延续对象”以持有由执行堆栈(execution stack)所组成的执行状态。你在稍后调用延续对象的方法,就可以恢复系统状态,取代当前的执行状态,包括调用堆栈(call stack)会被延续对象内的调用堆栈所取代。此调用将返回执行到此程序代码

块的后面。从Ruby的观点来看，在概念上这等于是让你的执行状态回到过去。

语法

在Ruby中，你可以通过调用Kernel中的callcc方法并传给它一个程序代码块，从而取得一个延续。下面的程序代码块不会对延续做任何事，但是会输出对象的ID：

```
irb(main):001:0> callcc {|continuation| puts continuation}
#<Continuation:0x28c2dd8>
```

这个被动的小程序做的事比你认为的多。名为continuation的参数是威力强大的一个小宝石，在你调用callcc的时候，continuation持有全部的执行内容，包括“变量值”和整个“调用堆栈”。把它视为一个被存储的游戏进度或者是时间上冻结的一刻，你就可以随时回到那一刻。特别是，通过调用延续，Ruby将会回到延续块后面的位置以执行下一个语句。下面是一个有趣的延续范例：

```
callcc do |continuation|
  for i in 1..10 do
    continuation.call if (i == 7)
    puts i
  end
  puts 'This never happens.'
end
puts 'Good bye.'
```

输出为：

```
>ruby forloop.rb
1
2
3
4
5
6
Good bye.
>
```

再一次说明，整个callcc语句是一个时间点。当i是7的时候，Ruby执行continuation.call。这会将执行点跳到延续块的后面，所以最后两个数字不会被输出来，而puts 'This never happens.'没有被执行到。此

`callcc`方法将加载延续中的应用程序堆栈，突然将执行的位置跳到延续程序代码块的后面，也就是 `puts 'Good bye.'`。这种移动执行的方式有点像 `goto`。

当然，你通常不会使用延续来做跳出 `for` 循环之类的事。当你将延续传递到程序代码块之外（例如方法调用），更能展示延续的强大威力。

更具有威力的范例

千万要记得，延续会将“调用堆栈”和“块内的局部变量”交还到它们原来的地方，也就是你调用延续时的地方。所以下面的程序：

```
1 def loop
2   for i in 1..5 do
3     puts i
4     callcc {|continuation| return continuation} if i==2
5   end           # cont.call returns here
6   return nil
7 end
8
9 puts "Before loop call"
10 cont=loop()
11 puts "After loop call"
12 cont.call if cont
13 puts "After continuation call"
```

会带来下面的结果：

```
>ruby continuation.rb
Before loop call
1
2
After loop call
3
4
5
After loop call
After continuation call
```

所以，我们能够在有事情发生时离开循环，需要的时候再回到循环。因为延续是如此与众不同，让我们更详细地看看这个范例，一旦你知道这是怎么回事，读起来就并不难理解。第4行将游戏存起来并将它放进一个容器中。

第12行回到以前的游戏进度。让我们进一步地分解它,想象你自己就是Ruby解释器:

- 从第9行开始,方法声明之后。
- 执行第9行,输出字符串 `Before loop call`。
- 执行第10行,调用一个名为 `loop` 的方法。将第10行记录在调用堆栈中,当调用结束的时候,知道该回到何处。
- 进入 `loop` 方法,也就是第1行。
- 第一次运行循环,执行第2行到第5行。`i` 的值是1,输出1。
- 第二次运行循环,`i` 的值是2,输出2。
- 在第4行,`i` 是2,所以用三个步骤调用 `callcc`。首先,复制一份调用堆栈;然后,复制一份实例变量 (`i` 是2);最后,将延续块后面的行号(第5行)放进复制版的调用堆栈中。所以现在延续的复制版堆栈里面就记录着(第5行,第10行),而原始版调用堆栈内是(第10行)。
- 在第4行,执行 `return` 语句。你将 `continuation` 的值返回给调用堆栈顶端的那一行。调用堆栈内是第10行,所以你将 `continuation` 的值返回给第10行。将 `cont` 设定成被返回的延续。记得此延续具有当前的执行环境:调用堆栈内有(第5行,第10行),变量 `i` 的值是2。
- 执行第11行,在屏幕上输出 `After call loop`。
- 执行第12行,调用延续,回到以前的执行状态,将 `i` 的值设定为2。回到调用堆栈顶端的行号,然后你就可以把该行号从调用堆栈上移出。现在调用堆栈内只有第10行。
- 执行 `for` 循环剩下的部分, `i=3、4、5`。
- 返回 `nil`。此调用堆栈内有10,所以你会返回第10行并将 `cont` 设定成 `nil`。
- 执行第13和15行,略过第14行,因为 `cont` 是 `nil`。

此延续范例展示了几个不错的能力。你可以将某个时间点的执行状态记录下来,像我们在 `for` 循环内做的那样。你可以将执行状态记录在一个对象中,像我们在 `cont` 对象中做的那样。然后你可以返回之前的执行状态。

你为何使用它们？

你可能一开始会认为延续最大的用途是可以打破逻辑控制结构，就像在我们的 `for` 循环内实现一个 `break`。虽然大部分的状况下你想把它想成是“暂停—继续”，但在这种使用状况下，延续是不可思议的。合作式多任务 (cooperative multitasking) 让一个程序自愿放弃控制权，让另一个程序取得控制权，以后再取回控制权。这样的问题相当容易利用延续来解决。较微妙的使用牵涉到沟通。当你的应用程序牵涉到多台计算机之间同步的“请求—响应”沟通时，你常常想要暂停控制权，直到远程的系统响应之后再继续。当你需要这样的大型解决方案时，在你等待的时候暂停控制权可以让系统有时间处理其他的请求。当远程系统响应时，只要调用一个延续，系统就可以很轻易地恢复你的应用程序。

延续服务器

你或许开始理解，为何对 Web 服务器来说，延续机制是一个有趣的技术。如果你想将一个 Web 应用程序当成是一个延续应用程序，在和用户沟通的过程中有着“暂停—继续”的中断，这会更有道理。在等待用户输入（以 HTTP 请求的形式）的时候，Web 服务器可以简单地存储一个状态、将延续对象存储在 HTTP session 中，且在该处理另一个请求的时候立刻返回之前的时间冻结点 (frozen point)。请注意，在图 8-1 中，我可以很方便地转换控制。不把 Web 应用程序当成一系列由用户触发的“请求—响应”，我可以把 Web 应用程序当成一系列由服务器控制的“响应—请求”。我的服务器程序代码变得简单许多。

你的 Web 应用服务器不再是由许多不同的独立请求所组成。服务器可以方便地看着这个世界，将其视为一群简单的端到端应用。当需要处理另一个请求时，它加载每个用户的状态；当需要和用户再次沟通时，它暂停该用户的应用。就是这样处理个别的请求，你瞧！你的应用程序可以维护状态，并使用它来无缝隙地控制应用流程。

在低层次，延续服务器变成 Web 应用程序的集合，每个 Web 应用程序都有着在某时间冻结的状态，冻结的状态以“延续”的形式存在。每个用户都有一个 session。延续服务器为每个 session 指定一个 ID 并为每个 session 组织其延续对象。在每次请求之后，延续服务器会将当时的执行状态记录在一个

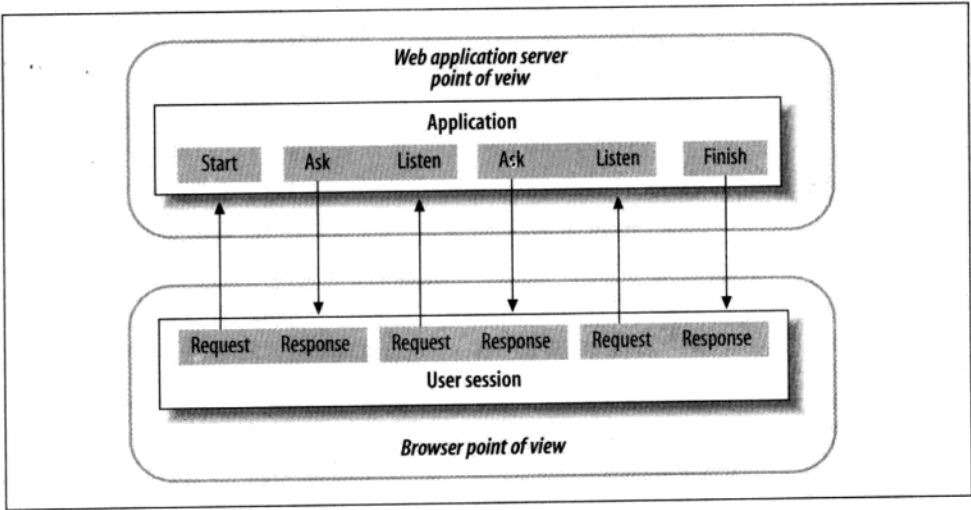


图 8-1：延续服务器将控制权从客户端转到服务器端，简化了服务器的世界观与程序代码

延续对象内并在延续对象和 session 之间建立关联。所以，一个服务器有多个 session，每个 session 有一个或多个延续对象（代表某个冻结的时刻），如图 8-2 所示。你可以再也不要看到个别的 HTTP 请求，因为它们都被隐藏在应用程序的流程中。本来就该这样！

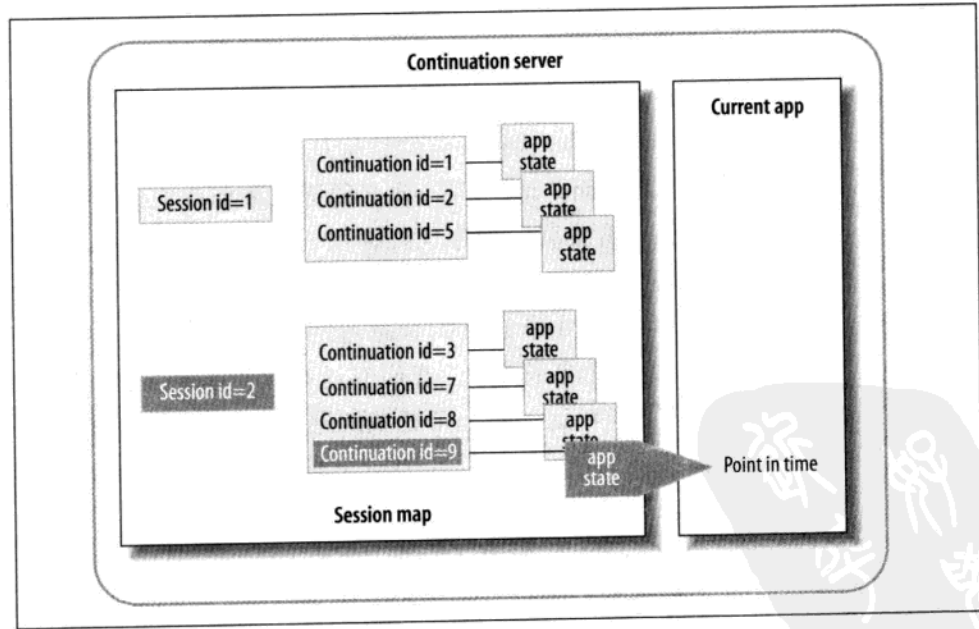


图 8-2：延续服务器存储 Web 应用程序执行过程中的许多时间点的状态



Glenn Vanderburg: 延续服务器

《Maximum Java 1.1》一书作者

Glenn Vanderburg 是一位来自达拉斯 (Dallas) 的顾问, 在 Java 还不叫做 Java 的时候就已经开始写 Java 程序了。他是最早的 Java 高端书籍作者。Glenn 有 19 年的软件开发经验, 用过许多语言、平台及领域。

目前的 Web 编程模型有什么问题? 比方说 servlet 模型。

GV: 有两大问题, 先从明显的讲起。当我做大型计算机编程时, 我会建立一个有许多表单字段的信息屏幕, 将这样的信息推送到 3270 部终端机。直到用户按下 Enter 之前, 这个程序都不会再从终端机收到信息。这听起来熟悉吗?

在大型计算机的日子, 程序必须暂停, 等待用户再次送出信息。Web 编程实际上更糟了, 因为在数千个 (而不是数百个) 用户同时上线的情况下, 程序还被要求在每次交互之间都尽量遗忘, 好让每次客户递送过来的数据都能独立。Web 编程模型的这种“无状态”本质, 强迫程序员每次都得手动处理、存储、取出程序状态。Web 框架可以帮助一些人, 但是程序员还得小心考虑如何处理每个状态。一个错误就会让 Web 应用程序变得相当难以使用。

Web 开发模型的其他不足之处在于我们的程序都是用字符串衔接起来的。这种导航的结构是通过我们所点击的 URL 所定义的, 而其他的 URL 也必须进入到配置文件中, 将自己和某些有关联的程序代码绑在一起。用户的输入以表单 (form) 方式传来, 里面有许多字段, 字段有名字和值。我们在 session 中所存储的状态通常是通过 key 的方式取得 (key 是一个字符串)。我们有这些强类型的编程语言和 IDE, 可以在这一路上帮助我们避免犯下愚笨的错误, 像拼错变量名称。但是对 Web 应用程序来说这是不够的, 因为这些工具不会帮助我们确认对下面东西的使用: URL 片段 (fragment)、表单字段等。还有, 这些字符串为黑客打开了攻击我们的应用程序的方便之门。再说一次, 某些框

架帮助我们处理混乱的字符串，但是大多数的框架只是减少问题，并没有解决问题。

但是这些问题基本上都是来自 HTTP 和 HTML，而不是 Java，对吧？

GV: 没错，但是我们应该尽量减少它们对我们的生产力的影响。这两个东西在一起，使得 Web 应用程序更明显的比传统的应用程序更复杂，也让我们付出的代价更高，不管是在时间还是品质上。管理系统的复杂度是软件开发的基本问题。

什么是延续服务器？

GV: 首先，我要表明，我不喜欢延续服务器这个称呼方式，有两个原因。第一个原因，这个词无法清楚地表达出这种服务器和框架是干什么用的。它们是用来服务 Web 应用程序的。像 Seaside 和 Iowa 这样的框架采用延续帮助程序员处理 Web 应用程序的无状态、来来回回沟通的本质。延续的机制在此框架中涉入相当深，开发者不会直接处理延续。第二个理由，延续并非这种框架（例如 Seaside）的全部，这些框架其实用到更多技术，以帮助 Web 开发者更容易进行 Web 应用程序的开发。

这些服务器做的事，是利用延续（以及调用时用的程序子句，加上 session 状态自动追踪与回溯信息的缓存）以建立高级抽象来帮助 Web 开发，透明地处理许多混乱的细节——那些 Web 开发者一直都在奋力挣脱的细节。恭喜，有了子句以及动态语言的诸多特色，我们可以拥有比 Java 更具威力的工具。

它们将哪些带到台面上？

GV: 让 Web 开发变得比较容易。而且它相当简单：Web 开发中许多最困难的议题，几乎是所有应用程序都会面对的事情，都在延续服务器中获得自动而透明地解决，被默认建立到 Web 应用程序中。比方说，Seaside 让 Web 应用程序的开发变得很容易，以用户所期望的方式运作：适当地处理“上一页”按钮、如果用户打开多个窗口或 tab 页就适当地进行 session 分支、如果再回到结果页不会“意外地买了两次”。

在 Seaside 中，Web 应用程序代码看起来像一般桌面应用程序的代码。有问题需要问用户吗？调用一个对话框 (dialog)，等待它返回，就行了。当然，在对话框调用的范围内发生许多事：一个延续对象被存储、一个对话框页面被送给浏览器、浏览器思考此问题（或许很久）并回答，而当接收到此回答时，它会找寻之前存储的延续并调用它，然后此对话框就返回了，就仿佛此线程一直在等待此响应一样。而从某个角度来看，的确也是如此。

优点和缺点

你已经看到了主要的优点：可以将一个 Web 应用程序视为一大块，而不是许许多多的小请求之间相互协调。这样的威力相当惊人。延续服务器还有一些其他的能力。“上一页”按钮的问题变得相当容易解决。因为如果“上一页”的按钮没有被禁用 (disabled)，你可以直接把应用程序的状态返回到前一个延续或者前面的任何一个延续。要禁用“上一页”按钮，就告诉浏览器并删除过去的延续。线程也变得相当不重要，因为每个线程都可以处理某一个私有的延续，每个都有某个应用程序自己的资源。你不必担心对一个共享 session 的序列化 (serializing) 访问会遇到什么问题。

对于有着复杂状态管理主题以及复杂页面控制流程的应用程序，延续服务器最能发挥效用。延续服务器动态地简化导航 (navigation)，让你维护网页之间的应用状态。

延续服务器有下面的问题：

- 服务器通常将 ID 添加到 URL 上，但是有些人不喜欢丑陋的 URL（不过像 Amazon.com 这样的网站还是这么做）。
- 你必须保证 session 的本性，也就是说在一个用户的 session 中的初始化请求之后，必须由相同的机器为每个后续请求服务用户。你可以利用一个分布式的延续缓存克服此问题，但是正如同分布式的 HTTP session 本身就是一个问题一样，分布式的延续缓存可能不见得完全实用。
- 延续比起其他的管理机制来说更昂贵。我已经看到一点点实际的证据，显示这已经造成部署上的问题。但尽管如此，有些人还是相信这个做法

不会像传统的做法一样适合应用在大型系统上。“部分延续”(partial continuation)的研究或许最终能够解决这样的问题。

对我来说,威力强大的优点可以让我忽略潜在的缺点。某种语言的延续服务器将会累积足够的欢迎程度而成为催化剂,这是很有可能发生的。我敬重的程序员 Dave Thomas、Glenn Vanderburg、David Heinemeier Hansson 都表示延续服务器是我们应该要注意的技术。《Hacks and painters》的作者 Paul Graham 在 Web 应用程序上使用延续机制,在 Viaweb 上产生巨大的破坏力,这个应用程序后来变成 Yahoo! Store。他也是延续服务器的提倡者。让我们看一个范例,支持延续机制的最受欢迎的 Web 框架: Seaside。

Seaside

Seaside 是一个具有高生产力的 Web 编程框架,用 Smalltalk 语言所开发的。Avi Bryant 一开始是用 Ruby 开发 Seaside,在名为 Iowa 的框架中。早期的 Ruby 延续有一些问题,所以 Seaside 的原始作者就将该项目移到 Smalltalk 上。从那时起,他就开始改进该框架,并使用它来处理商业应用程序。Seaside 有一些特征:

- Seaside 用程序设计的方式展现 HTML,而大多数的 Java 框架利用模板(template)来呈现 HTML。我知道得不够多,不足以告诉大家这个好、那个不好,但是这样的方法的确相当不同,在 Seaside 的模型中也运作得相当妥贴。
- Seaside 有一个组件模型。一个 Seaside 组件管理员使用界面状态并将自己用 HTML 呈现。Seaside 组件具有可重用性,并让你用比一个网页更小的增量单位来思考。
- Seaside 让链接的管理变得容易。你可以用一个程序代码块来指定一个链接,所以链接就不会失去同步。此框架可以帮你管理它们。
- Seaside 是有情境关联的(modal)。这是作者对于延续服务器方法的说法。Seaside 让你可以用一个方法表达一个网页或多个 Web 流程。
- Seaside 的调试方式是我见过最好的。从浏览器内,你可以打开一个基于 Web 的 Smalltalk 浏览器,将它写完。你也可以在应用程序内查看(inspect)所有对象的值。

当然，你也因此得到使用此高级动态语言的优点：你有很快的响应循环、交互式的解释工具、可以使用Smalltalk绝佳的环境。在本章中，我使用Squeak IDE。Squeak是Smalltalk的一个方言，是由迪斯尼所提倡的Smalltalk版本。

一些 Smalltalk 的语法

在我们走太远之前，你应该先了解一下Smalltalk的语法。别担心，我不是说Smalltalk接下来就会变成热门语言，我只是想让你看到最佳的延续服务器威力。如果你想跟随我的步伐，你可以先去下载Squeak IDE并安装，网址在<http://www.squeak.org/download/index.html>。启动Squeak，点击Tools，然后拖动一个workspace（工作空间）和transcript（报告）的窗口到你的桌面。使用workspace窗口作为输入，拿transcript窗口当输出。

Smalltalk语言相当简单。先输入一个对象名称，然后是方法名称，最后是参数。让我们来执行一些语句。在你的workspace内输入：

```
Transcript show: 'Hello'
```

将它选成高亮度，单击鼠标右键，然后从菜单中选择do it（你也可以在按下Enter之前使用Alt+D，如此可以直接计算这一行）。你应该看到“Hello”字样出现在你的Transcript窗口。Transcript是一个对象，show:是方法（Smalltalk将方法称为“信息”），而'Hello'是参数。

就和Ruby一样，Smalltalk也支持程序代码块，虽然这样的语法有一些不同。执行下面的程序：

```
1 to: 5 do: [:i | Transcript show: i]
```

首先，你看到[cmd]用来表示程序代码块的开始与结束。对程序代码块来说，i是一个参数。在这个声明中，你在它前面放上一个冒号。

让我们尝试多条语句。Smalltalk用英文句号表示一个语句的结束。逻辑使用信息和程序代码块：

```
age := 4.  
(age > 16)  
  ifFalse: [Transcript show: 'Youngster.']  
  ifTrue: [Transcript show: 'Old timer.']
```

这里通过:=信息将age设定为4。然后，它送出ifFalse:方法给(age >

16) 表达式。第一个程序代码块是给 `ifFalse` 的参数，如果表达式的计算结果是 `false`，就会被调用。

你可以看到 Smalltalk 的优雅影响许多语言，包括 Java 和其他语言。Java 的垃圾收集、设计模式以及集合都受到 Smalltalk 的影响。想想 Hibernate 对于信息链 (message chaining) 的使用。如果一个方法不必返回一个值，它就是返回自己，这可以让一个程序变得紧凑，像这样：

```
cfg.add("pet.hbm")
    .add("vet.hbm")
    .add("pet.hbm");
```

许多来自 Eclipse 的想法都起源于 IBM 的 VisualAge for Java，这个产品最早是和 Smalltalk 产品一起共享 IDE 程序以及虚拟机。Smalltalk 语法相当一致。

Seaside 概述

Seaside 是一个 Smalltalk 框架以及一个服务器。请记住，一个延续服务器是和其他 Web 服务器相当不同的，所以 Seaside 必须运行它自己的环境。在 Squeak 中，单击鼠标左键后会出现一个菜单，称为 “world menu” (世界菜单)。然后，你会选择 Open → SqueakMap Package Loader。使用它来安装四个包：DynamicBindings、KomServices、KomHttpServer 以及 Seaside，按照这样的顺序来安装。现在，你的 Smalltalk 映像 (image，指的是内存中的二元分布方式) 中已经具备 Seaside。要看看它的样子，可以启动服务器。在 Squeak 中，你会打开一个 workspace 并执行：

```
WAKom startOn: 9090
```

WAKom 是服务器的名字。startOn: 用来告诉服务器，在某个端口 (port) 开启网络，本例中的端口是 9090。在某些例子中，WAKom 就和 Tomcat 一样，和其他的 Web 应用服务器一样，你可以利用浏览器来调整它的状态。将浏览器连接到：

```
http://localhost:9090/seaside/config
```

你会看到 Seaside 的配置画面。对你来说，某些项目看起来应该有一点熟悉。你会看到一串注册过的应用程序以及某些配置选项。稍后，这一切会变得更清楚，你会知道 Seaside 比 Java 的 Tomcat 多出很多特色。

一个 Seaside 范例

在 /seaside 开头的 URL 下，你可以看到一串应用程序列表，其中有一个 store（商店）。点击它，你会看到 SushiNet（寿司网），这是一个有点奇特的 Web 框架范例。在搜索窗口中输入 Tuna（鲔鱼），点击两条不同的鲔鱼，把它们加入你的购物车（cart）。现在请点击“上一页”按钮并注意你回到前一页了，就和往常一样。再加入另一条鲔鱼，你会看到之前的两条鲔鱼还在购物车内。所以，如果你需要的话，你可以覆盖（override）“上一页”按钮。

组件

请注意在图 8-3 中跨越屏幕上方的三个框子。Seaside 是一个基于组件的架构。每个组件都有独立的呈现方式，而且每个组件背后都有一个模型。

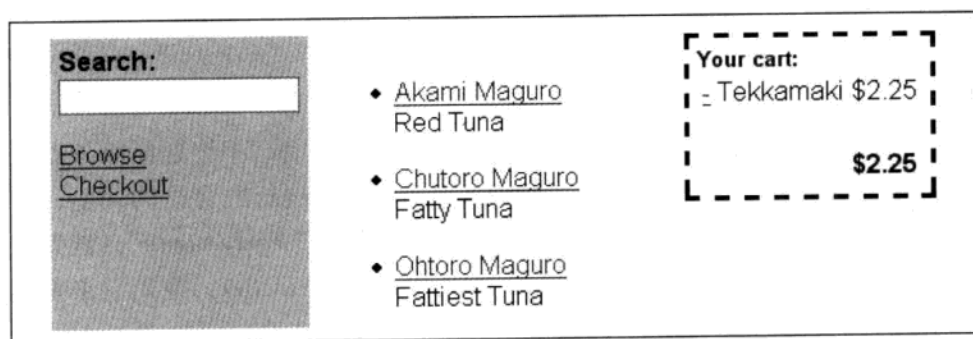


图 8-3: Seaside 应用程序有三个主要的组件，每个都有独立的呈现方式和业务逻辑

这个面向组件的方法常常会让设计和重构复杂的 Web 画面更容易。比方说，下面是购物车的展现方式：

```
html divNamed: 'cart' with: [
  html small: [html bold: 'Your cart:'].
  html table: [
    cart countsAndItems do:
      [:assoc | self renderRowForCount:
        assoc key of: assoc value on: html ].
    html spacerRow.
    html
      tableRowWith: ''
      with: ''
      with: [html bold: cart totalPrice printStringAsCents].
  ]
]
```

请注意 Seaside 组件内有生成 HTML 的程序代码, Java 用户不会喜欢这样的做法, 但是在 Seaside 中这样却很有生产力。黑体的程序代码会生成表。首先, 你看到表信息被送到此 `html` 对象。这将会在程序代码块内生成 `table` 标记。接下来, 你将会看到一个循环 (处理购物车内的商品)、一个空白行以及一个总金额行。

复杂控制流

对于这个应用程序来说, 最复杂的一系列窗口就是 checkout (结账)。想一想传统的应用程序如何管理控制流。试试此应用程序的结账, 看它是如何运作的。在你的购物车内加入几片寿司 (sushi) 并按下 Checkout。SushiNet 将会带你走过几个步骤:

- 你将会确认购物车内的商品。如果你喜欢你的订单, 可以按下 “Proceed with checkout” (继续结账); 否则, 你可以按下 “Modify my order” (修改订单)。用户在这里可以做决定, 流程会根据用户的输入而改变。
- 你将会指定送货地址 (shipping address), 然后你可以选择是否使用此地址当作账单地址 (billing address)。再一次说明, 这样的决策会影响应用程序的流程。如果你不想使用相同的地址作为送货和账单地址, SushiNet 将会重复利用出货地址的组件来询问你的账单地址, 不错吧!
- 你将会输入你的信用卡信息。如果你不确认, 将会回到相同的画面; 如果你确认, 就会得到成功的画面。
- 任何时候, 用户都可以点击 “上一页” 按钮。如果用户在订单递交之后才按下 “上一页”, 他将会得到一个信息, 告知该网页已经过期 (expired)。

所以, 此流程看起来就像图 8-4 那样并不复杂。你必须做四个决定, 根据这些决定, 你的路径 (route) 会被引导到适合的地方。

如果你用 Java servlet 实现这样的流程, 则需要处理四个或更多个 “独立的” 请求, 如图 8-5 所示。每个都必须在请求一开始先加载当前的状态, 然后在请求结束前存储当前的状态。这个 Web 流程可以由用户的决策所决定, 所以你可以有多个 “回到前一页”。流程的改变会造成潜在的主重构。

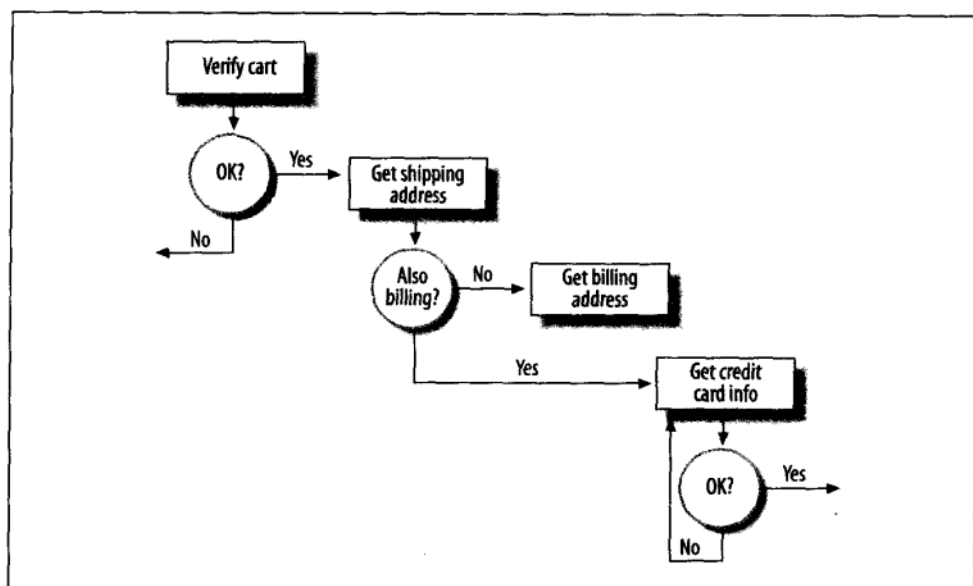


图 8-4：此流程有三个不同的用户决策，而且会将传统 Web 应用程序变复杂

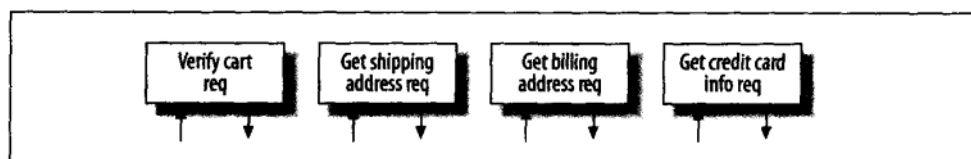


图 8-5：Java servlet 将 checkout 的问题当作是四个（或更多个，如果回到前一页的话）独立的请求

利用延续的做法，此逻辑变得几乎微不足道，如图 8-6 所示。你可以把这整个流程看成是一个简单的组件，称为 Checkout。这个组件可以处理牵涉一个以上组件以及一个以上页面的流程！这个程序代码看起来相当简单，而且吸引人。

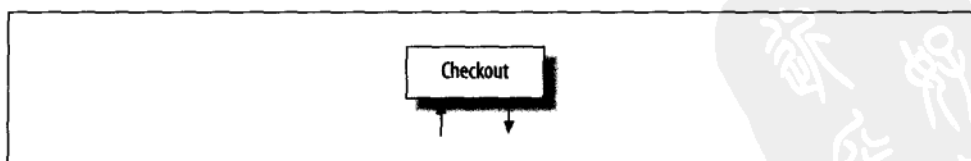


图 8-6：有了 Seaside 和其他延续服务器，此流程变成单一、整合的方法

调试和浏览

自从你有了一个冻结的延续，对于 Seaside 来说，提供一个完整的执行状态记录就变得很容易。Seaside 更进一步，让你具有访问 Web 浏览器的能力。在屏幕下方，你可以看到几个链接 (link)。不管是什么应用程序，Seaside 默认就会创建这些链接。请注意，你可以分析或研究内存的使用状况，但是在我心中已有想法。按下这个名为 Toggle Halos 的链接。

你应该看到一个框架，有着三个图标 (icon) 出现在每个组件周围。这些图标可以给你一个完整的程序代码浏览器、一个检查器 (inspector) 以及一个 CSS (cascading style sheet) 编辑器。点击此浏览器图标 (第一个)，注意你可以看到执行状态被冻结在何处。接下来，由左而右点击 Seaside-Examples-Store、WASoreTask、Go。你可以看到用来存储任务 (task) 的程序代码。

在图 8-4 中，你将会看到实现 cart 的程序代码：

```
go
| shipping billing creditCard |
cart _ WASoreCart new.
self isolate:
    [[self fillCart.
     self confirmContentsOfCart]
     whileFalse].

self isolate:
    [shipping <- self getShippingAddress.
     billing <- (self useAsBillingAddress: shipping)
                 ifFalse: [self getBillingAddress]
                 ifTrue: [shipping].
     creditCard <- self getPaymentInfo.
     self shipTo: shipping billTo: billing payWith: creditCard].

self displayConfirmation.
```

任务

在 Seaside 中，任务 (task) 处理业务逻辑 (business logic)。让我们将注意力放在黑体字的程序代码部分。在购物车确认之后它会处理一切。此 self isolate 方法取出一个程序代码块，并确定块内的一切都是自动运作或是一个事务 (transaction)。下一行程序代码相当有趣：

```
[shipping <- self getShippingAddress.
```

这个语句向用户呈现 getShippingAddress 网页，并将结果地址放在购物车地址内。你可以看到此框架如何转换控制。现在，除了浏览器是在控制中，Seaside 还让你将交通导向服务器。下三行代码展示了一个决策：

```
billing <- (self useAsBillingAddress: shipping)
           ifFalse: [self getBillingAddress]
           ifTrue: [shipping].
```

此 useAsBillingAddress 方法呈现决策画面。此表达式 (self useAsBillingAddress: shipping) 返回一个布尔值且将触发 ifFalse: 或者 ifTrue: 方法。ifFalse: 实际触发此程序代码块 [self getBillingAddress]，把另一个网页送给用户。

虽然对于 Smalltalk 的语法可能有一点不习惯，但如果你是一个 Struts 或 Servlet 开发者，你现在可能已经在微笑了。这个方法释放了你，让你可以在更高级的抽象中做事。你可以卷起数个组件或网页而成为一个单一的任务 (task)，且此延续服务器会让管理保持简单。“状态”和“导航”的议题将会烟消云散。

那又怎样？

我相当确定，延续服务器有朝一日会被证明是很重要的技术。我也同样地确定 Seaside 不是一个杀手级的应用，不会让 Smalltalk 一夕之间成为主流的热门技术。Smalltalk 有 30 年的历史声誉有待改变。此刻，Smalltalk 差不多只能算是一个学术的语言，商业应用相当少。Smalltalk 社区很聪明也有技术眼光，但是我还是没看到可以将 Smalltalk 带进主流的营销活动。30 年之后，这大概也不会改变了。

延续服务器还是有一些小小的栅栏必须跨越：

- 到目前为止，此服务器需要丑陋、暂时的 URL，因为每个延续都需要一个唯一的 ID。用户不喜欢丑的 URL。就像 Amazon 网站一样，Seaside 目前还有这样的限制，必须在有意义的 URL 之后加上延续的 ID。
- 延续服务器无法适合大型应用，因为存储延续相当昂贵，虽然你想过，但此问题不见得很糟。大多数的延续服务器在服务器端会有此框架共同

的程序代码，应该只有“调用堆栈”的最后一部分在不同的延续之间是不同的。采用“部分的延续”应该可以让性能提高。

- 到目前为止，最好的服务器是在学院的语言上。Lisp、Smalltalk、Ruby或许会阻碍一般的商业应用。当然，延续服务器可以帮助这些语言更接近于主流市场。

但是，最终，延续服务器有机会扮演一个角色，因为它们是比较自然且威力强大的抽象，写起程序来更方便。系统在计算能力方面持续增加，各种内存和存储设备变得更便宜，生产力最终会成为最重要的考虑项目。毕竟到最后，延续服务器会变得够快。较高的抽象会让我们更具有生产力，如果你拿枪逼我预言的话，我会说延续服务器将会演化并成为主流。但是不是在Java身上或者Java的衍生者（例如C#）身上，因为这类的语言必须很不自然地模拟延续。延续机制会被实现在一个更动态、更高级的语言上。我想，用Python或Ruby所实现的延续服务器将会在不久的将来证明这一切。

第九章

竞争者

这是我第一次到达 Class IV 河流，而我即将到达不会很有名的 Five Falls 瀑布。在典型的和缓 Ouachita 山脉，“Cassatot”（印地安语，压碎头盖骨的东西）相当危急。说实话，我其实还没准备好面对这条河。看不到的小精灵送来厚实的喷射和水波，通过瀑布射出，戏弄着我，让我的船撞击岩石，将我转了一圈，然后随心所欲地将我翻动。我的导游却似乎和河流的每个分子和睦相处，他运用急流丢向他的全部力量，让他的船跳舞似的跨过许多沟槽、波浪，甚至面对人称洗衣机（Washing Machine）的瀑布。

整个过程，我身体的每一处都感到酸痛，我学习推出我的脚，用到每一块肌肉，甚至是极少使用到的肌肉。这个河流是如此狂野，我必须用尽全力。结束时我筋疲力尽，滑出船外。我的导游蹒跚地离开他的船，而我说不出话来。他的两腿在膝盖以上就截肢了，我简直不敢相信，没有两腿所能带来的平衡和力量，他照样能够在河流上做出一切的动作。这几秒完全改变了我对于泛舟的观感，原来泛舟有无限的可能，这几秒比起任何时刻都更能够让我认识桨的妙用。既然我知道我能够走多远，我总是会找方法来用船、桨、身体以及河流，用更少的力气来做更多的事。

这本书是关于改变感知的主题。当然，Java 链接库有“腿”，也就是链接库和社区。但是此社区有时候可能会失常，而文化正造成复杂度日渐增加的链接库。JCP 似乎很碍事，专门评估政策和开会，而不在于根据实际经验设计出来的好链接库。


```

    {
        System.out.println(c+d+f);
    }
}

class text
{
    public static void main(String args[])
    {
        JieOne1 a1=new JieOne1();
        a1.print();
        a1.print1();
    }
}

```

在这段代码中定义的常量具备 public、static 和 final 等属性，所以它只能赋值一次，一旦赋值就不能更改，执行代码，得到如图 9-11 所示的结果。

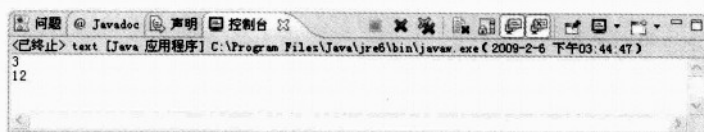


图 9-11 接口的量是常量

9.6.3 实现接口

一个类可以实现多个接口，但是实现多接口的类必须符合 3 个原则，第一是能为所有的接口提供实现的功能，能遵循重写的所有的规则，能保持相同的返回的数据类型。实现接口的方法十分简单，其格式如下：

```

[<修饰符>] class<类名> implements <接口名>
{
    .....
}

```

实例 32：下面定义一个类实现多个接口，其代码如下：

```

interface JieOne
{
    int add(int a,int b);
}

interface JieTwo
{
    int sub(int a,int b);
}

interface JieThree
{
    int mul(int a,int b);
}

```

```

interface JieFour
{
    int umul(int a,int b);
}

class JieDuo implements JieOne,JieTwo,JieThree,JieFour
{
    public int add(int a,int b)
    {
        return a+b;
    }

    public int sub(int a,int b)
    {
        return a-b;
    }

    public int mul(int a,int b)
    {
        return a*b;
    }

    public int umul(int a,int b)
    {
        return a/b;
    }
}

class text
{
    public static void main(String args[])
    {
        JieDuo aa=new JieDuo();
        System.out.println("a+b="+aa.add(20,100));//提供具体实现方法
        System.out.println("a-b="+aa.sub(20,100)); //提供具体实现方法
        System.out.println("a*b="+aa.mul(20,100)); //提供具体实现方法
        System.out.println("a/b="+aa.umul(20,100)); //提供具体实现方法
    }
}

```

执行代码，得到如图 9-12 所示的方法。

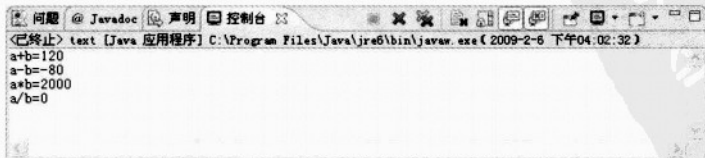


图 9-12 接口的实现

实例探索和读者练习：

在上面的一个实例中讲解了多个接口的实现，下面展示一个接口实现的程序，读者领会了上面一个代码，不妨再看这段代码，读完程序，给出结果，将代码复制到计算机中进行编译，得到结果，对比自己的答案与编译结果是否一致，其代码如下：

```

interface Aone
{
    void add(int a,int b);
}

interface Atwo
{
    void sub(int a,int b);
}

interface Athree extends Aone,Atwo//创建 Athree 接口继承了 Aone、Atwo 接口
{
    void mul(int a,int b);
}

class JieXian implements Athree
{
    public void add(int a,int b)
    {
        System.out.println(a+b);
    }

    public void sub(int a,int b)
    {
        System.out.println(a-b);
    }

    public void mul(int a,int b)
    {
        System.out.println(a*b);
    }
}

class text
{
    public static void main(String args[])
    {
        JieXian aa=new JieXian();
        aa.add(12,22);
        aa.sub(12,22);
        aa.mul(12,22);
    }
}

```

9.7 内部类

内部类也是一个类，它是类的一个成员，内部类初学者可以把它理解为类的嵌套，下面讲内部类的具体知识。

9.7.1 如何定义内部类

在 Java 中，一个类被嵌套定义在另一类中，那么这个类被称为内部类，而内部类的被称为外部类，对于外部类而言，内部类相当于变量和方法，是外部类的一个成员，但是内部类也是属于一个类，下面将展示一个内部类的代码，其代码如下：

```
class WaiBuLei
{
    int a;
    void print()
    {
        System.out.println("a="+a);
    }

    class NeiBuLei //定义了一个内部类
    {
        int b; //内部类的变量
        void print1()//内部类的方法
        {
            System.out.println("b="+b);
        }
    }
}
```

从上面的代码中可以看出，首先定义了一个类 WaiBulei，然后在这个类中再定义一个类 NeiBulei，从程序中的注释可以清楚地看到这个类中有变量和方法，只能 NeiBulei 的内容，它和其他的普通类没有什么区别，下面将一个实例来讲解。

实例 33：下面这段代码定义了一个内部类，然后打印出内容，其代码如下：

```
class Neilei
{
    private String a="这是一个变量";

    void print()
    {
        final String b="这是一个方法";
        class Neilei1//定义了一个内部类
        {
            public void kk()
            {
                System.out.println(a);
                System.out.println(b);
            }
        }
    }
}
```



```

    }

    }

    Neilei aa=new Neilei();
    aa.kk();
}

}

class textone
{
    public static void main(String args[])
    {
        Neilei Neilei=new Neilei();
        Neilei.print();
    }
}

```

运行程序，得到如图 9-13 所示的结果。

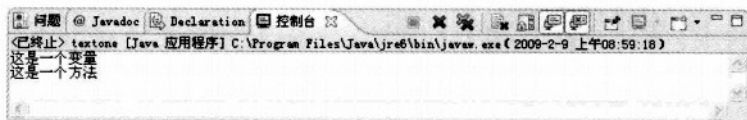


图 9-13 方法内部类

实例探索和读者练习：

在上面的一个实例中讲解了内部类，下面再展示一个内部类的程序，读者领会了上面一个代码，不妨再看这段代码，读完程序，给出结果，将代码复制到计算机中进行编译，得到结果，对比自己的答案与编译结果是否一致，其代码如下：

```

interface Neiuse//接口里的内部类
{
    public void kk();
}

class Usein
{
    Neiuse a=new Neiuse(){
        public void kk()
        {
            System.out.println("这是一个接口类型的内部类");
        }
    };

    void print()
    {
        a.kk();
    }
}

```

```

    }

    class text
    {
        public static void main(String args[])
        {
            Usein b=new Usein();
            b.print();
        }
    }

```

9.7.2 如何正确理解内部类

从上面一节中,读者已经理解了内部类,但是如何正确理解内部类和使用内部类,下面通过一段代码进行讲解,其代码如下:

```

public class Nei
{
    public static void main(String args[])
    {
        class foo{
            public String name;
            public foo(String x)
            {
                name=x;
            }
        }
        Object obj=new foo("hi");
        foo p=(foo)obj;
        System.out.println(p.name);
    }
}

```

内部类也是一个类,在前面所讲解的类的所有知识,例如 final 关键字、重载、重写等,这些内容对内部类来说同样适用,执行上面代码,得到如图 9-14 所示的结果。

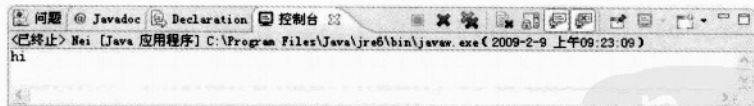


图 9-14 正确使用内部类

9.8 本课回顾和网络关键词

在本课的内容中,首先,简要讲解了类的继承、对父类进行访问、重写和重载等知识。在本节的内容中,将对本课中的主要知识点进行简要回顾,并总结出本课知识点的网络关键词。

1. 本课回顾

在本课的内容中，主要讲解了面向对象的基础知识。回顾本课内容，主要的知识点概括如下。

(1) 继承。

- 类的继承。
- 对象的使用。

(2) 对父类进行访问。

- 调用父类的构造方法。
- 访问父类的变量和方法。
- 多重次继承。

(3) 重写。

(4) 重载。

(5) 接口。

(6) 内部类。

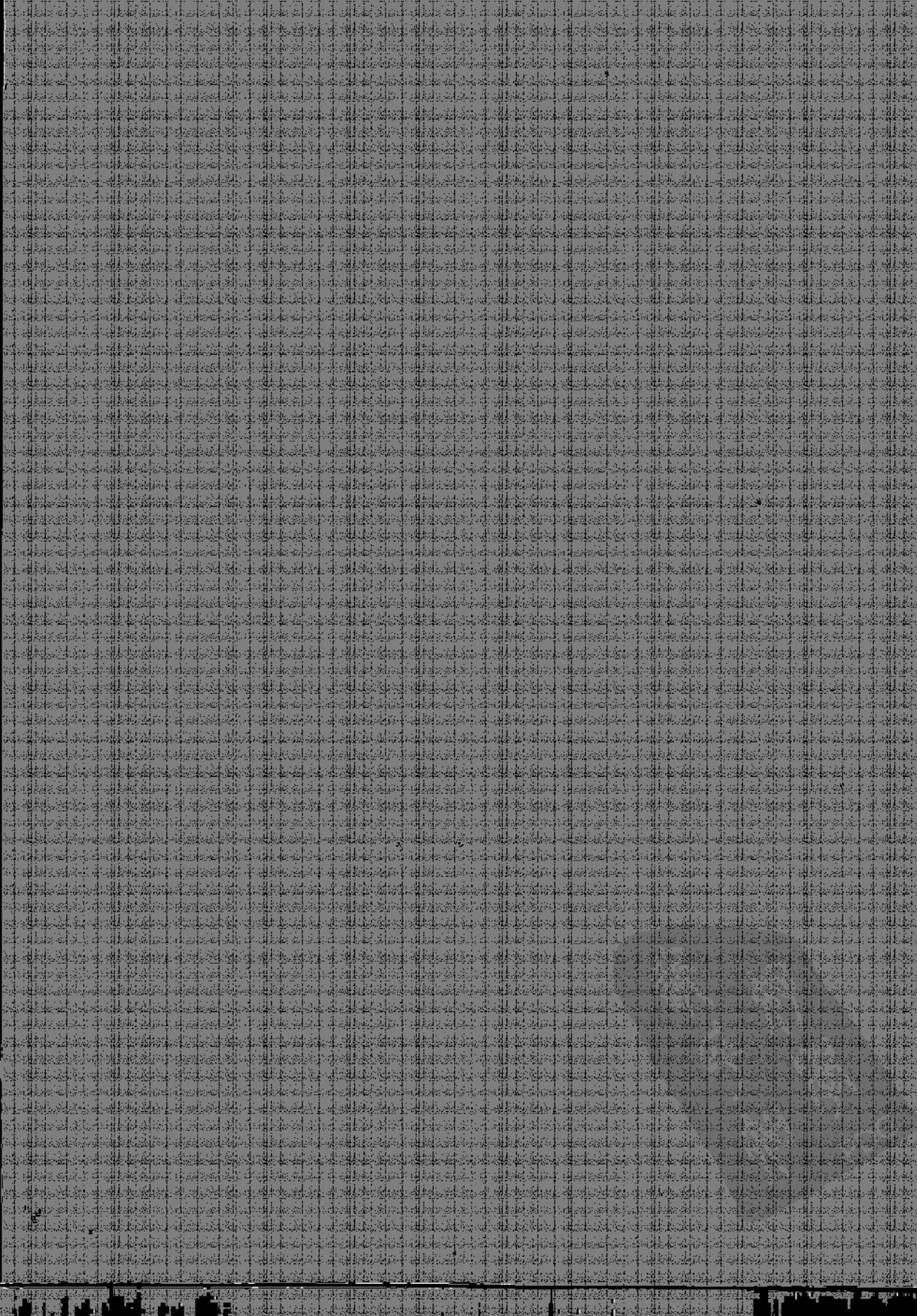
2. 本课网络关键词

在下面的内容中，将对本课中的主要知识点进行收集整理，总结出本课知识的网络关键词。

本课知识的网络关键词有：“继承”、“多重次继承”、“重载”、“重写”、“接口”、“内部类”。

通过整理上述网络关键词，读者可以在百度、Google 或 Yahoo 中获取上述关键词的基本知识。有的关键词还可以继续细分并且可以获取每个知识点的对应使用实例，读者可以通过获取的实例来加深对知识的理解。

希望通过本章内容的学习，读者可以对 Java 面向对象编程有了进一步认识，为进入本书后面知识的学习打下坚实的基础。



第 10 课 初步图形编程——AWT

图形编程对一个软件来说十分重要，虽然它不能影响软件的功能，但是一个赏心悦目的界面可以吸引更多的使用者，AWT 是 Java 软件图形编程的工具之一，使用者较多，任何学习 Java 程序的人都必须掌握这个工具，本课将由浅到深地讲解 AWT。

10.1 本课学习目标

在本课的内容中，将深入讲解 AWT 编程软件界面的知识，本课的具体学习目标如下：

- AWT 简介。
- 创建简易图形用户界面。
- 布局管理器。
- 组件和监听接口。

10.2 AWT 简介

AWT (Abstract Windowing Toolkit)，中文译为抽象窗口工具包，是 Java 提供的用来建立和设置 Java 的图形用户界面的基本工具。AWT 由 Java 中的 java.awt 包提供，里面包含了许多可用来建立与平台无关的图形用户界面 (GUI) 的类，这些类又被称为组件 (components)。

AWT 是 Java 的平台独立的窗口系统、图形和用户界面器件工具包。AWT 是 Java 基础类 (JFC) 的一部分，为 Java 程序提供图形用户界面 (GUI) 的标准 API，下面通过一段代码来认识 AWT，其代码如下：

```
import java.awt.*; //引入 AWT 包
public class MyFrame extends Frame
{
```

只能算是一个实验性的产品。我希望它成功，但是我不认为它会成功，它要走的路还很长。

.NET

.NET是唯一一个我会提及的可以担任Java的后继者的非编程语言。.NET是微软最新的开发平台，值得特别被提及，因为它有一个巨大的链接库以及一个和语言无关的引擎，称为 Common Language Runtime (CLR)。如果微软让 .NET 成功了，真的做到语言中立，它将会成为各种语言的发射台。到目前为止，就和 JVM 一样，CLR 具有一些技术上的问题有待解决，才有可能完整地支持像 Ruby 这样的动态语言，而微软确实在往这方面进行着。

语言的选择

在某种程度上，.NET 的编程链接库比语言本身更重要。他们的使用模型常常会影响应用程序的架构，这方面的影响力甚至超过编程语言的选择。尽管如此，微软还是提供数个编程语言，各自瞄准不同的社区。

Visual Basic.NET

对于 Visual Basic 程序员的处置，微软相当棘手。似乎大部分的 Visual Basic 程序员都不喜欢 .NET，他们都各自寻找其他的解决方案，因为 .NET 框架改变了 Visual Basic 的编程模型。到目前为止，大多数的 Visual Basic 程序员如果不是积极地决定好追随的对象，就是被动地等待升级。不管怎样，微软都输了。结果，看起来 Visual Basic 身陷麻烦中。

在公开的场合，Java 和 .NET 开发者彼此不合，但是两方面常常勉为其难地承认对方有一些优点。虽然和平台结合得很紧密，但是 Java 开发者常常偷偷地表示羡慕 Visual Basic 的高生产力以及用户界面开发框架；Visual Basic 用户也秘密地响应这样的羡慕，赞美 Java 的结构。我合理地猜测，微软认为它可以偷偷地加进一些结构化，相信以 BASIC 的语法可以战胜对于底层框架不熟悉的挫折感。他们的如意算盘打错了。

微软正做出一些举动以满足 Visual Basic 社区。某些项目似乎是持续维护 Visual Basic 的经典版本，也就是旧版的 Visual Basic。对我来说，这样的举动就像是新口味的可口可乐和经典口味的可口可乐同时存在一样。对公关人员来说，这简直是无法自圆其说。

C#

C#（发音是 see sharp）是一个编程语言，扮演着 .NET 平台上的 Java 语言角色。在这本名为《超越 Java》的书中，对于 C# 实在没什么好说的，因为 C# 和 Java 很类似你将会看到几个小差异，像“依赖未受检的异常”而非“依赖受检的异常”以及一些语法甜头（syntactic sugar）。Java 最近所做的许多改变，像注释（annotation）以及自动装箱（autoboxing），都是从 .NET 中学来的。虽然大多数情况下，C# 的这些看起来像从 Java 中学到的好特色以及避开的将会在未来出现的新问题。C# 只是 Java 的邪恶双胞胎。

但是，微软似乎愿意将旧版本的 C# 带出新的语言，正在开发中，叫做 C Omega。这个语言将会具有可以大步向前迈进的潜力，甚至可能会打破和 C# 的兼容性。这样的语言有潜力可以提供更动态的语言的特色，加上微软的商业支持以及 CLR 作为便携的虚拟机。它正在受到大家的注视。但它是专属的语言，而且可能许多人只凭这个原因就直接打退堂鼓了。

CLR 上面的其他语言

.NET 不是微软的语言，它是 CLR 平台上开放源码语言的保证。现在，既然微软大部分的精力都毫无疑问地放在 Visual Basic、C++ 以及 C# 上，你不会看到链接库开始利用像程序代码块以及延续等概念。但是，微软积极地向 Ruby 和 Python 社区的关键人物招手，所以你可能不久后就会看到 CLR 上面的这些语言（译注 1）。

缺点与优点

.NET 和 CLR 还有一个问题，那就是微软。有时候这是一股力量，让你的肌肉更加结实，有时候则不然。CLR 会在其他操作系统上（比方说 Linux）运行得很好吗？应该不会这样。微软对于财产的保护严密以及技术上缺乏可移植性，所以你不容易看到 Java 社区拥抱 .NET。你或许会对我的看法感到很惊讶，我不认为微软的姿态将会维持在全面性的保护措施上，特别是在服务器端。

译注 1：在微软的支持下，IronPython 已经推出 CLR 平台上的 beta 1 版本了。

我以前说过，市场领导者想要垄断，非领导者则需要开放才能竞争。微软是客户端操作系统的领导者，微软在客户端采用专属框架是有道理的，毕竟微软长久以来几乎独占市场。但是在客户端这么做是行不通的，这会让他们打不进中大型企业的市场。一段时间之后，我相信微软将会认识到这样的事实并跳进开放源码软件的队伍。我不是唯一这么想的人。NoFluffJustStuff（没有废话，只有实话）研讨会是JavaOne之外最成功且影响深远的Java研讨会之一，Stuart Halloway 是很受尊重的Java顾问，特别是在元编程和反射方面。他在NoFluffJustStuff研讨会上说，他强烈觉得微软将会是世界上最大的开放源码厂商，而Dave Thomas似乎也同意这样的看法。

如果微软真地向开放源码软件转移，而Java社区承认这一点，微软将会对Java敞开CLR的大门，更重要的是，对以后所有的语言敞开大门。

较小的竞争者

现在，我已陷入天人交战，是否要讨论一下Perl、Lisp、PHP或者Smalltalk。这些都是很棒的语言，但是我不认为它们会流行起来。

如果你深深地信仰这些语言，你可以只读一下这里的简短总结，然后就跳到下一节：Perl太松散、太混乱；PHP和HTML绑定得太紧；Lisp太不好学；而Smalltalk不是Java。

如果你觉得太轻描淡写，就必须继续读下去。如果你是一个语言狂热者，而我对你挚爱的语言太轻视，或者更糟，我根本没提到，那么你可以打开你的Gmail发来不同的意见。Ted Neward审阅过这本书，所以我可以承受得了“烂书”这一类的批评。但是请你务必牢记：我没有说你的语言不好或不受欢迎，我只是说从现在开始的10年里，我们大概不会把这些语言当成Java杀手。

Steve Yegge: Perl、Lisp、PHP与Smalltalk

为何Perl不能取代Java? **SY:** 这个嘛！我认为Perl以前真的很成功，至今依然是相当重要的语言。Perl具有世界级的营销：Larry Wall了解

程序员，他相当风趣且善于表达。Perl 在 Unix 的脚本编程世界具有利基，另外在 CGI 上也是。Perl 过去相当成功，是因为它用处很广，就和 Java 一样。

我相信它正日益衰落，尽管 Perl 曾经比其他技术更具有生产力。所以你可以说它相当丑，但是人们还是可以很快地把事情做完。在较新的语言，特别是 Ruby 出现之后，局势已经有所改变。

Perl 使用无意义的抽象 (pointless abstraction)，像引用 (reference)、typeglobs、one-off 快捷方式以及 plain old gross hacks，加上额外的语法以掩盖缺点。它的生产力很高，但是程序员必须想办法走阻力最小的路径，而 Ruby 所提供的摩擦力相当小。

那么 Lisp 呢？

SY: 这是一个不简单的家伙。Lisp 是世界级仅存的古老技术，人们不断地重新发明或重新发现它，但是 Lisp 也是一个大家族，有许多彼此不兼容的设计和实现，没有任何一个看起来像是我们的赢家。当然，Common Lisp 需要大修，但是由委员会来重新设计 Common Lisp 在此刻来说相当不恰当。Lisp 需要一个仁慈的独裁者，具有好的伟大的执行力以及伟大的营销，才可能成功。

那么 PHP 呢？

SY: PHP 相当受欢迎，越来越受欢迎，因为它让 Web 编程变得很容易，比其他技术都容易，但是 Ruby on Rails 将会改变这样的局面。在某个时间点，将会有有一个简化的 Web 编程通行证。PHP 没有 Web 的简化通行证，它只是试着让你复制现有的复杂度。此语言被它的 Perl 遗产大大地削弱分量，它具有许多混乱且让人遗憾的设计决策，而且它也远不是 Ruby、Python、Lisp 等威力强大语言的对手。

那么 Smalltalk 有机会吗？

SY: 我相当怀疑。到最后，语言必须有话题和动力，而我没看到 Smalltalk 有任何的营销活动。此社区受到 Java 的重创，至今尚未恢复元气。

PHP

PHP是一个开放源码的脚本编程语言，从2000年早期就开始汇聚动力，它是一个标记语言（markup language），被设计嵌入在HTML内。用PHP开发简单的Web应用程序相当容易，但是这些应用程序通常只有小型的后端结构。因此，它和Java所适合的利基市场是不同的。下面是PHP版本的“Hello, World”：

```
<html>
  <head>
    <title>Hello, World</title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Web程序员都知道这就是HTML脚本编程语言。此程序代码是在服务器端先被处理过，得到纯HTML才送给客户端。PHP实际上把脚本编程处理得相当好，但是它只是一个标记语言。PHP的问题在于它的结构和外观都混在一起了。可以在PHP中将外观切割出来，但是这实在太不方便、太繁琐。

尽管如此，作为一个纯Web脚本编程语言，PHP的声势越来越浩大。2005年最不可思议的事件之一是IBM宣称要支持PHP，此举无疑是针对意图拥抱PHP的中小企业而来。IBM现在可以在理论上将软件和服务销售给客户以满足客户的需求。对于Visual Basic的用户来说，如果不想进入门槛很高的.NET，那么PHP似乎是一个相当自然的选择。就和Visual Basic一样，随着开发者在错误的地方寻找简单性，PHP将会被利用在某些不适合的地方。

使用最基本的Google技巧，你可以找到几十个试图比较Java和PHP文件。你将会看到两种结论：PHP阵营说Java的生产力不够高，而Java阵营说PHP的结构不够好。我则是同时相信这两个结论。对于小型应用程序来说，使用PHP主要的危险是，很快就会长成很大的PHP应用程序，到时候，没有结构可以帮你轻易地维护和扩展应用程序，你会留下这团混乱，掉头离去。

Perl

对于想要有原始效率的程序员来说，Perl是一个非常受欢迎的语言。对于shell脚本来说，Perl相当受欢迎，直到后来有许多较简单的替代品出现。就

生产力来说, Perl 具有其他高级语言的许多特征。它相当具有表达力、精炼而且是动态类型的。它让你自由地想做什么就做什么, 且有快速的响应循环 (feedback loop)。Paul Graham 说 Perl 是一个很棒的语言, 适合用来进行 “hacking” 或者快速地实验性编程。因特网有很大的一部分是利用 CGI 的 Perl 脚本提供服务的。

Perl 也有它的缺点, 当你评估一个语言的全面生产力时, 你应该要把维护和可读性也估量进去。许多专家都把 Perl 评为可读性很差的语言。就和 Java 一样, Perl 的许多问题是文化造成的。某些 Perl 程序员宁愿断掉手指也不愿意多打四个字符, 他们才不管这些字符是否可以提升可读性, 毕竟难写的程序应该也会难读。其他的 Perl 问题就和语言本身有关了, Perl 的面向对象相当明显是急着加进来的, 而 Perl 具有相当多火星文般的语法快捷方式, 简直是神秘的沟通方式 (secret handshake), 只有 Perl 的妈妈不会觉得自己的宝贝很丑。我们许多人在某些时刻都和 Perl 有过爱恨交织的关系。说来有趣, 但是对照起 Java 则是截然不同, 还是别去用 Perl 吧!

Smalltalk

Smalltalk 是一个美丽的语言, 相当先进, 超越它当时的时代。Smalltalk 和 Lisp 大概是两个和 Ruby 有最多共同概念的语言。早在 Java 尚未在 James Gosling (Java 之父) 的眼中闪烁时, 聪明的开发者就已经使用 Smalltalk 来建立成功的面向对象应用程序了, 而 “不是很聪明” 的开发者使用 Smalltalk 建立了某些有史以来最丑的面向对象程序。事实上, 在 20 世纪 70 年代中期和晚期, 我们还不具有智慧或处理能力可以应付 OOP, 而且我们当时也没有像 just-in-time 编译器这样的技术。

在第八章, 你看到了 Smalltalk 语言的优雅, 它彻底地面向对象, 语法相当一致。对许多 COBOL、BASIC、Pascal、C 或者 C++ 的程序员来说, Smalltalk 的语法似乎很奇怪。我所知道真的去尝试过 Smalltalk 的例子, 大部分都能够按时交出他们的应用程序, 只是他们从来不能将那些应用程序和其他的世界整合得很好。

Smalltalk 一直都无法吸引 C 和 C++ 的开发者, 因为它太特殊, 也让人觉得太慢。当小小的 Smalltalk 社区正静静等待走红的时机时, Java 的设计者已经积极地掐住 C++ 社区的喉咙, 把他们拉到 Java 阵营。Java 采用 C++ 的语

法和使用模型，提供解决方案以消除C开发者遇到的最迫切的问题。我们已经看到过，Java是完美的OO和C++社区之间的一个妥协。稍后，IBM买下OTI，这是一家Smalltalk虚拟机厂商。在Smalltalk的最后一次出击，IBM在Visual Age IDE内建立了一个通用的虚拟机，并希望通用的JVM可以为Smalltalk带来加分的效果，但是效果太小而且太迟了。我们对于Java的自由、安全以及接近C++感到相当满意，不会想去正眼瞧Smalltalk。

难以想象，Smalltalk已经被压抑了30年，无法出头，而且大概以后也不会有机会了。但是，你还是可以找到一个小而活跃的Smalltalk开发者社区。迪斯尼建立了Squeak，这是一个Smalltalk的方言和实现，焦点在多媒体应用。许多其他的方言依然存在于这个世界上。

终究，Smalltalk或许会对开发造成影响，但会是通过延续服务器这样的概念。你在许多语言中都可以看到袭自Smalltalk的对象模型和语法。最值得注意的是，Ruby大方地从Smalltalk借来程序代码块（code block）以及一些惯用语（像把自己作为返回值）。我想延续服务器终究将会在Web开发中扮演一个很重要的角色，因为延续服务器实在太有道理、太自然、太有吸引力了。Smalltalk正是目前大家研究“延续”的工具。

Lisp

Lisp是一个极具有威力的语言，以它奇特而纯粹的语法、抽象塑模、原始效益而著称。在Lisp中，一切东西都是一个列表（list），包括Lisp程序也是列表。在Lisp中，元编程相当自然，也相当普遍。重要的想法像aspect-oriented programming（AOP）以及延续服务器都是从Lisp开始的。Lisp的方言会周期性地出现，包括Dylan和Scheme（译者注：还有我最喜欢的REBOL），但是没有任何Lisp家族成员可以在商业上成为主流，最好的成就就是当作Emacs编辑器的宏语言。但是，新公司常常会用Lisp，因为一旦你学会它，你就具有不可思议的生产力（译者注：这是真的！我建议读者找机会学会一个Lisp家族的语言，绝对值得。我推荐读者阅读Peter Seibel所写的《Practical Common Lisp》一书）。某些相当成功的程序员，像Paul Graham（O'Reilly出版的《Hacks and painters》一书作者）就相信Lisp是最具有表达力的编程语言。他可能是对的。

Lisp的社区总是由一群天资聪颖的程序员所组成，而且Lisp在校园内依然

相当流行。事实上，一些具有最好信息系的大学，像是麻省理工学院，很早就强调学习Lisp，这样可以让学生很快地用抽象的方式思考并使用函数的技巧。

或许所有的语言都将会再度回归到Lisp，但是我不认为Lisp本身会是一个最终的解答。它太与众不同，而且需要花相当多的时间和精力去学习。

函数性语言

现在谈论函数性语言 (functional programming)，可能有一点不合时宜，因为我们似乎正在进入面向对象的世界。但我还是要说，函数性语言提供了较高的抽象以及非常高的生产力。如果有杀手级应用出现的话，函数性语言大概会立刻受到欢迎。

Haskell以及Erlang是函数性语言的两个家族。函数 (function) 是函数性语言的重点所在。我使用数学上对于函数的解释：

- 函数不可以有副作用 (side effect)。对于大多数过程式 (procedural) 程序员来说，这一点有些奇怪，但是它也有很大的优点。
- 函数必须返回值。
- 你能够将函数的返回值用在任何类型正确的地方。

你可以在Ruby和Lisp语言中进行函数式编程，但是为了要研究或为了纯粹性，则使用较纯的语言比较好。下面是一个Haskell的范例，计算一个数的阶乘：

```
fact 0 = 1
fact n = n * fact (n - 1)
```

然后，如你所预期的，你可以使用下面的方式来计算阶乘：

```
fact 10
```

下面是Fibonacci序列（每个数字是前两个数字的和）：

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

函数性语言让你可以在比较高的抽象层做事。Haskell 在研究机构和校园中有很好的动力,而且似乎具有一个小而活跃的商业社区。它很容易教,因此,它可以作为函数式编程的敲门砖,就像Pascal适合作为过程语言的敲门砖一样。

你可以在Erlang语言中看到函数式编程的威力。Erlang由Ericsson所开发,它的主要焦点是在并发(concurrency)。Erlang让你可以轻易地建立与使用线程并在线程之间沟通。Erlang也可以改进分布式运算,因为线程的位置是透明的:线程可以在相同的处理器中,也可以在另一个处理器中,或者在不同的机器上。它具有高生产力、动态类型、垃圾收集而且很小。最近对于Erlang感兴趣的人忽然增加,因为他们需要建立支持并发与分散的应用程序。此刻,作为一个一般目的的语言来说,Erlang仍在婴儿期。Erlang用户倾向于将它搭配C语言一起使用(考虑效率),而且它也没有真正的用户界面链接库。但是,Erlang在它的利基市场中威力相当大,这可能会直接或间接地影响到未来。

“下一个大东西”

当然,这本书的整个前提本身就相当自大。我做了相当多的假设,而且还根据了几十个访谈、我敏锐的直觉、一些环境的迹象而归纳出一些具有冒犯性的结论。

Java或许需要的只是进厂大修,或许问题在于又大又复杂的链接库,而对于语言的一些扭曲或许可以让Java再维持10年的领导地位。或许社区的文化无法帮助定义我们的链接库,但厂商可以做一些表面的事并简化80%的路径,而不是又建立一个新的“被XML困扰”的框架,JCP或许会开始支持现有的好框架,支持那些根据经验做出来的实用框架,而不是通过开会和标准化而设计出来的框架。

或许Dion Almaer是对的,业界的大公司并不想离开Java,而在可见的未来,我们将会依然会与Java为伍。

或许Jason Hunter是对的,下一个大东西不会是编程语言,或许Java正是我们所需要的一切,而我们会用这样的基础往上攀爬抽象的阶梯。或许Glenn与David两个人都是对的,根本不会有所谓的下一个大东西,将会有许多个小东西,而元编程与延续都将会扮演重要的角色。

我不知道最终的答案为何，所以我从我的导师和伙伴那里学习他们的看法。本书的访谈对象都是我最尊敬的人，很荣幸能够在本书中和他们进行这些篇幅的交流。我没说Java已死，或者Ruby即将接班，或者延续服务器将会支配一切。我只知道：

- 我现在很受伤，我的顾客也是。我越来越难让我的顾客满意Java的一切。
- 某些东西，像用基于Web的用户界面来处理关系型数据库，应该用Java会很容易才对，毕竟已经经过了10年的努力。但是事实上却相当繁琐。
- 两年前，在研讨会听到其他语言就会打瞌睡的同一群人，现在似乎开始注意其他语言了。我在Java研讨会上的“超越Java”演讲，每次都是听众爆满。

对我来说，我的眼睛睁得大大的，我已经看到替代技术能够做什么事，特别是Ruby on Rails让我可以很快地建立可靠的程序代码，并更自信、更频繁地将系统交到客户面前。我没有积极去寻找替代技术，相反，我写过四本关于Java的书籍，并且在Java圈里有了声誉，我有足够的理由维持现状。我的确发现有一些替代技术相当吸引人，而且可以平稳地转移到该技术。

你的代价

如果你是一个Java开发者且此信息让你感到困扰，这是很自然的反应。你有绝对的理由感觉受到威胁，你的世界观受到挑战，你甚至可能觉得你的生存基础受到破坏。如果是这样，我鼓励你放下这本书，自己去研究了解实际的状况。

环顾四周吧！当James Duncan Davidson环顾四周的时候，他发现一个语言符合他的需求，适合低级用户界面的开发；Stuart Halloway找到一个语言，让他的新创公司以他理想的速度成长；Dave Thomas找到一个机会，出版一系列相当重要的书籍；Glenn Vanderburg找到几个语言，相当适合用来发展他所钟爱的元编程技术。

如果你决定扩展你的视野，超越Java的水平线，你将会发现我是对的。我在这本书中所提到的某些技术，甚至我没提到的技术，都会让你获得解放。你将会在下一个将我们往前推的波浪中冲浪。

如果我错了，至少Java还是在那里等你使用，嗯，甚至COBOL也还是在那里等你使用。但是对你来说，这个时候Java将会不同于以往。其他的语言将会扩展你的视野，你会看到其他的做法，就如同Java开发者的波浪将为我们带来独特的世界观一样。如果你花时间在Smalltalk上，你大概会更常使用Java的反射，你将会找机会通过模拟程序代码块以反转控制权，你将会适当地减少使用XML（好吧！对于这一点，我可能对你的要求太严格了）。如果你探索过延续服务器，你或许会找机会在Java中模拟这样的编程方式。如果你尝试过Rails，你将学会付出更多注意力在默认状态（default）和惯例（convention）上。Hibernate、Spring、Struts、servlet、collection以及JDO都可以使用这些技巧。

通过学习一种语言，可以让你开拓眼界，将心智扩展到更有威力、更动态的境界。通过函数式编程或延续，你的视线将会截然不同。用不一样的想法去干扰你的朋友，告诉他们你不认为世界是平的，超越Java的地平线，外面还有更广的宇宙。

索引

本篇索引让读者可以方便地查找特定词汇在本书里的页码,以快速找到所需的内容。传统上,中文书没有编制索引的惯例,因为中文不像英文那样,有一套公认而且大家都知道的排序规则(字母顺序)。然而,为了方便读者查阅,索引是不可或缺的。为了兼顾“查阅”与“中文化”,我们决定沿袭英文版的编排格式,也就是依照英文字母顺序排列所有项目,所以,我们保留大部分词汇的原文,如此读者才能快速找到想找的项目,而中文是以辅助说明的方式出现。

然而,使用英文字母顺序编排的结果,表示读者必须先知道英文词汇的原文,才能顺利找到其所在的页码。例如,假设读者想知道某本书哪几页提到了“网域名称服务器”,那么,你必须先知道其原文是“Domain Name Server”,或是知道其缩写是“DNS”,然后才能推断此项目应该是编在“D”小节。如果读者觉得这样不方便,我们为此感到抱歉,因为我们实在找不到一套大家都公认的中文排序规则。如果我们像编字典那样,使用首字笔划顺序来排列,那么除了不方便查找之外,还必须面临一词多译的问题。例如,有人习惯将“serial port”翻译成“串行埠”,但也有人将它翻译成“串行端口”,如果你不知道本书采用哪一种译词,要第一次就顺利找到“serial port”的页码,唯一的办法是碰运气。如果运气不好,那么你必须同时知道 serial port 的每一种可能的译词,才有机会找到。

格式说明

所有项目都是依照英文字母顺序排列。举一个实例说明如何使用本篇索引以及

索引项目的编排格式。如果你想知道某本书哪几页提到了“workbench”，则必须先翻到“W”小节，然后你会看到：

workbench (工作台)

Eclipse 3.0, 313

编辑器窗口, 10

概况, 8

视图, 10

(参见“Run-time Workbench”)

如你所见，我们以缩排格式来表示各项信息。通常，从最左侧逐层往右读，可以得到一个符合文法的完整句子或词汇。例如：你在第8页可以看到“概况”，可得到特定的概念还有其他和“工作台”相关的信息；可以发现第313页讨论和Eclipse 3.0相关的工作台。另外，后面的括号中的“参见‘Run-time Workbench’”，指示你也可参考“R”小节的“Run-time Workbench”，272-274、283、299这几页也都有相关信息。

在上例中，可以看到对于第一层的词汇我们保留了原文，并在其后加上中文译词，这是为了维持本索引的可查阅性，让读者有个规则可循。然而，在不妨碍查阅的前提下，对于第二层与第三层的项目我们会予以中文化，但若有模棱两可的情况，则在中文后面仍会保留原文。

希望本节的说明有助于读者使用本篇索引。如果读者对O'Reilly公司书籍的索引方式有任何意见，请用E-mail告诉我们，好让我们知道如何改进。

A

Action Pack framework (动作封装框架), 143

Active Record framework (动作记录框架), 142

adware (广告插件), 44

agile process (敏捷过程), 61

Ajax, 48, 84

Almaer, Dion, 188

访谈, 82

alpha geeks (阿法技客), 12

annotation (注释), 15

anonymous inner class (匿名的内部类), 93

Ant, 52, 60

ANTLR parser generator (ANTLR 解析生成器), 179

欢迎提出改进索引方面的建议。请发邮件至 info@mail.oreilly.com.cn。

applet, 32
……的缺点, 47
杀手级应用, 91
application language (应用程序语言),
103
approachability (可接近性), 12
新语言的, 90
array (数组), 95
AspectJ, 76
aspect-oriented programming (AOP),
10, 119

B

back button (“上一页”按钮), 151
boiling frogs (水煮青蛙), 10
browser as single application platform (浏览器作为单一应用程序平台), 46
Buck, Jamis 访谈, 124

C

C#

Java 潜在的取代者, 100
优缺点, 181

C++

……的可接近性, 90
语言, 24
安全与……, 44

C Omega

C programming language (C 编程语言),
81

client/server 主从运算, 46

……的经济, 21

code block (程序代码块), 55, 93

Common Gateway Interface (CGI), 22

Common Language Runtime (CLR),
180

Common Object Request Broker
Architecture (CORBA), 27

community and Java development (社区
与 Java 开发), 52

community-centric role (以社区为中心
的角色), 91

concurrency support (并发支持), 172

consistency (一致性), 28

新语言的需求, 97

continuation servers (延续服务器), 16,
149-169

优点和缺点, 160, 168

session 的本性, 160

continuation (延续), 19, 93

defined (定义), 152

D

database (数据库)

整合, 86

性能, 87

Davidson, James Duncan, 12, 18, 32

dependency injection (依赖注入),
122-125

development process (开发过程), 61

diamond inheritance problem (菱形继承
问题), 27

distributed transaction (分布式事务),
59

DLL Hell (DLL 地狱), 24, 26

domain specific language (DSL, 特定领
域语言), 147

dynamic class model (动态类模型), 94

dynamic language (动态语言), 15, 17

Dynamic Linking Libraries (DLL, 动态
链接库), 26

dynamic typing (动态类型), 92

E

economic justification for replacing Java
(取代 Java 的经济上的理
由), 89

EJB 3.0, 77

Specification (规范), 52

enterprise application development (企业应用程序的开发), 18

enterprise computing (企业运算), 46
standards within (标准), 51

enterprise computing versus the Internet
(企业运算与因特网), 22

enterprise integration (企业整合), 49,
86

Enterprise JavaBeans (EJB), 35

enterprise requirement (必须具备的企业特色), 87

Erlang, functional language (函数性语言), 187

exception (异常), 76

expressing data (表示数据), 76

extensibility (扩展), 82

Extreme Programming (极限编程), 61

F

feedback loop, rapid (响应循环, 快速),
94

functional language (函数性语言), 187

G

Geary, Stuart, 17

Gehtland, Justin, 17
访谈, 128

Generics (泛型), 69, 77

Google Map, 48

Graham, Paul, 161, 186

Groovy, 171

优点与缺点, 178

H

Halloway, Stuart, 147

Hansson, David Heinemeier, 147
访谈, 113, 120, 124

Haskell, functional language (函数性语言), 187

Hatcher, Erik 访谈, 50

Hibernate, 52, 60

message chaining (信息链), 163

HTML, 84

Hunter, Jason, 11, 188

I

IBM

Java 的未来, 75

Java 与……, 35

industry standard (行业标准), 35

integrated development environment
(IDE, 集成开发环境), 78

Internet (因特网)

应用, 46

对于专属产品和框架的影响, 22

集成, 83

interoperability (相互合作), 82, 84

inversion of control (控制反转), 111

J

J2EE, 35

Java

优点, 15

架构, 31

服用抗忧郁药物的 C++, 30

社区是很有价值的资源, 51

妥协, 16

竞争者, 170-190

打破迷思, 53-56

成功因素, 40-56

修改, 78

……的历史, 20-39

IBM 与……, 35

链接库, 57-79

受限制的演进, 54

Netscape Navigator 与……, 30

生产力与……, 54

最近的创新, 13
 Ruby 与……, 111, 126
 缺点, 2
 用作应用程序语言时的缺点, 54
 不自然的伸展, 14
 厂商的支持, 51
 Java 1.5, 71
 Java, 11, 10, 14
 Java Server Faces (JSF), 77
 Java Server Pages (JSP), 48
 Java Specification Request (JSR), 14
 Java virtual machine (JVM, Java 虚拟机), 14, 42, 88
 可移植性与……, 42
 JavaScript, 32, 48, 83
 JBoss, 52, 117
 JDBC, 86
 JDO, 60
 JMX, 72
 JRuby, 45, 83
 JUnit, 52, 61
 Jython, 45
K
 kayaking, as a metaphor for programming
 (泛舟, 一种编程哲学), 7
 killer app (杀手级应用)
 新语言的催化剂, 91
 PHP 的应用, 91
 Ruby 的元编程环境, 91
 Smalltalk 的延续服务器, 91
 King, Gavin, 76
L
 language, functional (语言, 功能性),
 187
 languages that could supplant Java (可能
 取代 Java 的语言), 80-101
 learning curve (学习曲线), 59
 legacy requirement (旧标准), 85

library (链接库), 59, 78
 limitation (限制), 62
 Java 的, 10
 Lisp, 80, 175, 182
 优点与缺点, 186
 Lotus 1-2-3 电子数据表, as killer app
 (作为杀手级应用), 91
 Lucene, 52

M

marketing the new language (营销新的语言), 88
 Matsumoto, Yukihiro (Matz), 98
 memory-stomper (内存绊跤), 24
 metaprogramming (元编程), 16, 17,
 173, 187
 以及杀手级应用问题, 147
 method interception (方法拦截), 118
 Microsoft (微软)
 影响应用程序开发, 22
 Windows, 安全与……, 44
 Middleware (中间软件), 51, 58
 mixin (混合器), 27, 116
 mobile application (移动应用程序), 46
 model-view-controller (MVC) framework
 (模型-视图-控制器框架), 13, 48
 module (模块), 116
 multiple inheritance (多重继承), 27

N

navigation and flow (导航和流程), 152
 nested include (嵌套引用), 25
 .NET
 Java 与……, 54
 优点与缺点, 180
 Netscape Navigator, 30
 Neward, Ted
 访谈, 71

O

- object orientation requirement (面向对象需求), 96
- Object Relational Mapping (ORM, 对象关系映射), 10, 60, 86
- object-oriented programming (OOP, 面向对象编程), 23
- open source community (开放源码社区), 36
- open source software (开放源码软件), 89
 - Java 的未来与……, 75
 - IBM 与……, 52
 - ……的重要性, 52
- open source tools and Java (开放源码工具与 Java), 12
- overloading, 71
- owls and ostriches (猫头鹰与鸵鸟), 7-19

P

- perception, changing (感知, 改变), 170
- performance (性能), 131
- Perl, 22, 76, 80, 89, 182
 - 对 Java 潜在的取代, 98
 - 优点与缺点, 184
- persistence framework (持久框架), 13
- PHP, 58, 75, 89, 184
 - 对 Java 潜在的取代, 100
 - 优点与缺点, 184
- plain old Java object (POJO), 12
- pointer arithmetic (指针运算), 24
- portability (可移植性), 28, 42, 81
- portlet (端口组件), 48
- primitive (基本类型), 73, 94
- productivity (生产力), 19, 63, 92, 130
 - 换掉 Java 的动机, 91

programming language feature (编程语言的特色), 92

- Python, 45, 89, 171
 - 对 Java 潜在的取代, 98
 - 优点与缺点, 176
 - 强类型与……, 64

R

- reflection (反射), 94
- relational database integration (关系数据库的整合), 86
- Representational State Transfer (REST), 85
- requirement (需求), 58
- risk, managing (风险, 管理), 7
- Ruby, 83, 89, 102-126, 171, 172
 - AOP, 120
 - 数组, 107
 - 程序代码块, 109, 152
 - 条件, 105
 - 容器, 108
 - 在……中延续, 152
 - 依赖注入, 122-125
 - 文件, 110
 - 挂钩, 122
 - Java 与……, 111
 - 循环, 105
 - 方法拦截, 118
 - 混合器, 116
 - 模块, 116
 - 对象, 103, 114
 - 开放类, 115
 - 对 Java 潜在的取代, 99
 - 基本类型, 109
 - 优点与缺点, 172
 - 范围, 106
 - 减少程序代码数的量以及原因, 110
 - 反射与……, 96
 - 正则表达式, 107

- ruby-lang.org, 103
- 强类型与……, 64
- 类型, 103
- Ruby on Rails, 17, 34, 84, 86, 113,
128-148, 189
 - Action Pack 框架, 143
 - Active Record 框架, 142
 - 优点, 9
 - ……的可亲性, 90
 - Base Camp 以及 Back Pack, 146
 - 案例探讨, 128
 - 框架, 50
 - 生成一个基本的应用程序, 134-142
 - 安装, 133
 - Java 社区的响应, 132
 - Java 与……, 172
 - 杀手级应用的问题, 147
 - 性能, 131
 - 生产力, 130
 - rubyforge.org, 133
 - RubyGems, 133
- Ruby.NET, 83

S

- sandbox (沙箱), 31
 - 安全与……, 44
- SCRUM, 61
- Seaside, 162-168
 - 组件式架构, 164
 - 控制流, 165
 - 概述, 163
 - 任务, 167
- security (安全), 44, 82, 87
- service-oriented architecture (SOA, 面向服务架构), 84
- servlet, 34, 47, 58
- session affinity (session 的本性), 160
- shortcomings of Java (Java 的缺点), 2
- simplicity (简单性), 77
- Slashdot, 88
- Smalltalk, 23
 - 程序代码块, 163
 - 对 Java 潜在的取代, 101
 - 生产力, 54
 - 优点与缺点, 185
 - Seaside 与……, 162
 - Squeak IDE, 162
 - 强类型与……, 64
- Spille, Mike, 179
- Spring, 60, 76
 - EJB 与……, 52
 - 方法拦截与……, 118
- SQL, 86
- Squeak IDE, 162
- Standard Widget Toolkit (SWT), 48
- statelessness (无状态), 150
- state, saving (状态, 存储), 151
- static typing (静态类型), 92
 - 优点, 72
- Strachan, James, 178
- string (字符串), 26, 76
- structured data (结构数据), 84
- Struts, 60
- Subramaniam, Venkat, 70
- Subway, Rails clone (Subway, 仿 Rails 的产品), 176
- Sun, future of Java and (Sun, Java 的未来), 75
- SushiNet (寿司网), 164
- Swing, 48
 - 用户界面开发与……, 94
- syntax (语法), 67
- System Object Model (SOM, 系统对象模型), 42

T

- Tapestry, 52, 60
- TheServerSide, 88
- Thomas Dave, 16, 147
- Tomcat, 47, 60

transaction (事务), 86
type erasure (类型消除法), 69
typing (类型), 64-73
 适用性, 69
 写程序/编译周期, 67
 动态, 92
 泛型, 69
 隐藏成本, 67
 overloading, 71
 静态, 92
 静态, 优点, 72
 静态与动态, 66
 强 vs. 弱, 65
 syntax (语法), 67

U

user interface focus (用户界面焦点), 94

V

Venderburg, Glenn, 101, 147
 访谈, 158

verbosity (冗长), 59, 74
virtual machine (虚拟机), 81
Visual Basic, 77, 86
VisualAge for Java, 163

W

Web Services, 88
Weirich, Jim
 访谈, 120, 124

X

XDoclet, 110
XML, 58, 84
 配置与……, 111
 XML-binding 框架, 13

Y

YAGNI, 8
Yegge, Steve
 访谈, 63, 171, 182

13.2.1 异常处理基础

异常处理就是处理程序在不正常情况下的一些问题，比如用户要一个程序连接网络，当网络没有连接，这时候就会报异常，提示给使用者。异常的处理格式如下：

```
public class Yichang
{
    public static void main()
    {
        try
        {
            //要检查的程序代码
        }
        catch(Exception y1)
        {
            //发生异常 y1，运行的代码
        }
        catch(Exception y2)
        {
            //发生异常 y2，运行的代码
        }

        finally
        {
            //绝对执行的代码
        }
    }
}
```

13.2.2 异常类

在 Java 的 lang 里有一个 Throwable 类，它是所有异常的父类或简洁父类，每个异常类都是它的子类，在这里面子类 Error 和 Exception，这两个类十分重要，前者是用来定义那些通常情况下不希望被捕获的异常，而后者是程序能够捕获的异常情况，下面将常见异常展示出来，如表 13-1 所示。

表 13-1 常见异常

异常类名称	异常类含义
ArithmeticException	算术异常类
ArratIndexOutOfBoundsException	数组小标越界异常类
ArrayStroeException	将与数组类型不兼容的值赋值给数组元素时抛出的异常
ClassCastException	类型强制转换异常类
ClassNotFoundException	为找到相应大类异常
EOFException	文件已结束异常类
FileNotFoundException	文件未找到异常类

(续表)

异常类名称	异常类含义
IllegalArgumentException	访问某类被拒绝时抛出的异常类
InstantiationException	试图通过 newInstance()方法创建一个抽象类或抽象接口的实例时抛出异常类
IOException	输入/输出抛出异常类
NegativeArraySizeException	建立元素个数为负数的异常类
NullPointerException	空指针异常
NumberFormatException	字符串转换为数字异常类
NoSuchFieldException	字段未找到异常类
NoSuchMethodException	方法未找到异常类
SecurityException	小应用程序执行浏览器的安全设置禁止动作时抛出的异常类
SQLException	操作数据库异常类
StringIndexOutOfBoundsException	字符串索引超出范围异常类

13.3 捕获异常

在编写程序中，程序员会设想这个程序在运行时，会出现什么意外和出现什么错误，用户需要通过捕获异常的方法将它寻找出来，本章将详细讲解如何捕获异常。

13.3.1 try...catch 捕获异常

在 Java 程序中，进行异常处理通常是将要处理异常程序块放在 try 里，然后创建 catch 语句块，下面通过一个例子进行讲解。

实例 44：下面将处理一个算术异常，其代码如下：

```
public class Yichang1
{
    public static void main(String args[])
    {
        int a,b;
        try
        {
            a=0;
            b=5/a;
            System.out.println("需要检验的程序");
        }
        catch(ArithmeticException e)
        {
            System.out.println("发生 ArithmeticException 异常");
        }
        System.out.println("结束");
    }
}
```

看上述程序, 有一个很明显的错误, 那就是 $b=5/a$, a 等于 0, 发生错误, 编写的程序者要将这个错误抛出来并进行处理, 运行程序, 得到如图 13-1 所示的答案。



13-1 抛出异常

实例探索和读者练习：

在上面的一个实例中, 讲解了 try...catch 的异常处理, 下面展示一段代码, 巩固这一个知识点, 读者读完程序, 给出结果, 将代码复制到计算机中进行编译, 得到结果, 对比自己的答案与编译结果是否一致, 其代码如下:

```
public class Yichang1
{
    public static void main(String args[])
    {
        int a,b;
        try
        {
            a=0;
            b=5/a;
            System.out.println("需要检验的程序");
        }
        catch(ArithmeticException e)
        {
            System.out.println("异常:"+e);
        }
        System.out.println("结束");
    }
}
```

13.3.2 多异常捕获

在 Java 程序中, 很可能出现多种错误的情况, 这时候就需要设计处理多种异常的情况, 下面通过一个代码来讲解多异常的捕获。

实例 45: 下面将这段程序, 出现了多个 catch 语句, 有一个 catch 适合后, 将执行其中的代码, 且跳过其他 catch 语句块的执行, 其代码如下:

```
public class Yichang3
{
    public static void main(String args[])
    {
```



```

int [] a=new int[3];
try
{
    a[3]=123;
    System.out.println("需要检验的程序");
}
catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("发生 ArrayIndexOutOfBoundsException 了异常");
}
catch(ArithmeticException e)
{
    System.out.println("发生了 ArithmeticException 异常");
}
catch(Exception e)
{
    System.out.println("发生了 Exception 异常");
}
System.out.println("结束");
}
}

```

执行程序，得到如图 13-2 所示的结果。

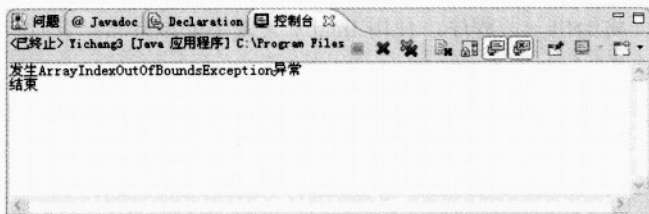


图 13-2 多异常捕获

实例探索和读者练习：

在上面的一个实例中，讲解了 try...catch 的多异常处理，下面展示一段代码，巩固这一个知识点，读者读完程序，给出结果，将代码复制到计算机中进行编译，得到结果，对比自己的答案与编译结果是否一致，其代码如下：

```

public class Yichang4
{
    public static void main(String args[]){
        int [] a=new int[3];
        int b;
        try
        {
            a[1]=123;
            b=a[1]/0;
            System.out.println("需要检验的程序");
        }
    }
}

```



```

        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("发生 ArrayIndexOutOfBoundsException 了异常");
        }
        catch(ArithmeticException e)
        {
            System.out.println("发生了 ArithmeticException 异常");
        }
        catch(Exception e)
        {
            System.out.println("发生了 Exception 异常");
        }
        System.out.println("结束");
    }
}

```

13.3.3 try...catch...finally

使用 try...catch 很多时候需要用到关键字 finally, 这个关键字的意思是不管程序有无异常发生, 都将执行 finally 语句块的内容, 这使得一些不管在任何情况下都必须执行的步骤被执行, 从而保证了程序的健壮性, 下面通过一个实例进行讲解。

实例 46: 下面将给出一段程序, 使用 finally 关键字处理异常, 其代码如下:

```

public class Yichang5
{
    public static void main(String args[])
    {
        try
        {
            int age=Integer.parseInt("25L");//抛出异常
            System.out.println("我要打印 1");
        }
        catch(NumberFormatException e)
        {
            int b=8/0;
            System.out.println("年龄请输入整数");
            System.out.println("错误"+e.getMessage());
        }
        finally {
            System.out.println("我要打印 2");
        }
        System.out.println("我要打印 3");
    }
}

```

执行程序, 得到如图 13-3 所示的结果。



图 13-3 finally 关键字的使用

实例探索和读者练习：

在上面的一个实例中，讲解了 finally 在 try...catch 的异常处理出现情况，下面展示一段代码，巩固这一个知识点，读者读完程序，给出结果，将代码复制到计算机中进行编译，得到结果，对比自己的答案与编译结果是否一致，其代码如下：

```
public class Yichang6
{
    public static void main(String args[])
    {
        int [] a=new int[3];
        try
        {
            a[3]=5;
            System.out.println("需要检验的程序");
        }
        catch(Exception e)
        {
            System.out.println("发生 Exception 异常");
        }
        finally{
            System.out.println("结束");
        }
    }
}
```

13.4 抛出异常

就是遇到异常了，对它不进行处理交给父类，这在 Java 中是比较常用的，下面进行详细讲解。

13.4.1 throws 抛出异常

声明异常是指一个方法不处理异常，调用层次向上传递，谁调用这个方法，这个异常就由谁处理，throws 是其中的一个，它的声明格式如下：

```
Void methodName (int a) throws Exception{
}
```

实例 47：下面将给出一段程序，使用 throws 关键字抛出异常，其代码如下：

```
public class YiThrows
{
    public void methodName(int x) throws
        ArrayIndexOutOfBoundsException, ArithmeticException
    {
        System.out.println(x);
        if(x==0)
        {
            System.out.println("没有异常");
            return;
        }
        else if(x==1)
        {
            int [] a=new int[3];
            a[3]=5;
        }
        else if(x==2)
        {
            int i=0;
            int j=5/i;
        }
    }
}

public static void main(String args[])
{
    YiThrows yc=new YiThrows();
    try
    {
        yc.methodName(0);
    }
    catch(Exception e)
    {
        System.out.println("异常:"+e);
    }
    try
    {
        yc.methodName(1);
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
        System.out.println("异常:"+e);
    }
    try
    {
        yc.methodName(2);
    }
    catch(ArithmeticException e)
    {
        System.out.println("异常:"+e);
    }
}
```

在上面这段程序中，需要注意的是在方法中必须要声明可能发生的所有异常，同时在调用该方法的 `main()` 方法中定义 `try...catch` 语句来捕获异常，执行程序，得到如图 13-4 所示的结果。

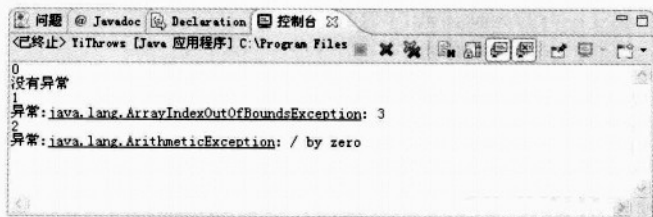


图 13-4 使用 `throws` 关键字抛出异常

实例探索和读者练习：

在上面的一个实例中，讲解了声明异常 `throws` 处理方法，下面将展示一段代码，巩固这一个知识点，读者读完程序，给出结果，将代码复制到计算机中进行编译，得到结果，对比自己的答案与编译结果是否一致。

先创建代码 `MyException.java`，其代码如下：

```
public class MyException extends Exception
//自定义异常，这里不要求掌握，后面将会讲解
{
    private String name;
    public MyException(String name)
    {
        this.name=name;
    }
    public String getMessage()
    {
        return this.name;
    }
}
```

然后编写 `Ren.java` 代码，其代码如下：

```
public class Ren
{
    public static int check(String strage) throws MyException
    {
        int age=Integer.parseInt(strage);
        if(age<0)
            throw new MyException("年龄不能为负数！");
        return age;
    }
    public static void main(String[] args)
    {
        try{
            int myage=check("-101");
```



```

        System.out.println(myage);
    }catch(NumberFormatException e){
        System.out.println("数据格式错误! ");
        System.out.println("原因: "+e.getMessage());
    }catch(MyException e){
        System.out.println("数据逻辑错误! ");
        System.out.println("原因: "+e.getMessage());
    }
}
}

```

13.4.2 throw 抛出异常

除了上面 throws 关键可以抛出异常, 还有一个关键字可以抛出异常, 那就是 throw 抛出异常, 把它抛给上一级调用的异常, 抛出的异常可以是异常引用, 也可以是异常对象, 下面通过一个例子进行讲解。

实例 48: 下面将给出一段程序, 使用 throw 关键字抛出异常, 其代码如下:

```

public class YiPao
{
    void methodName()
    {
        try
        {
            throw new ArrayIndexOutOfBoundsException();
        }
        catch(ArrayIndexOutOfBoundsException aoe)
        {
            throw aoe;
        }
    }
    public static void main(String args[])
    {
        YiPao yc=new YiPao();
        try
        {
            yc.methodName();
        }
        catch(ArrayIndexOutOfBoundsException aoe)
        {
            System.out.println("异常:"+aoe);
        }
    }
}

```

执行程序, 得到如图 13-5 所示的结果。

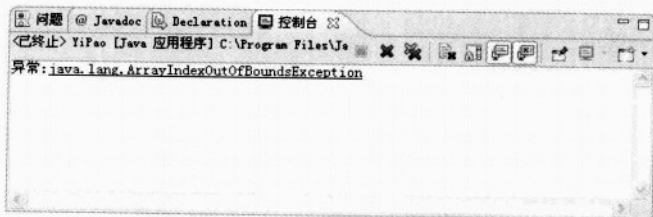


图 13-5 使用 throw 关键字抛出异常

实例探索和读者练习：

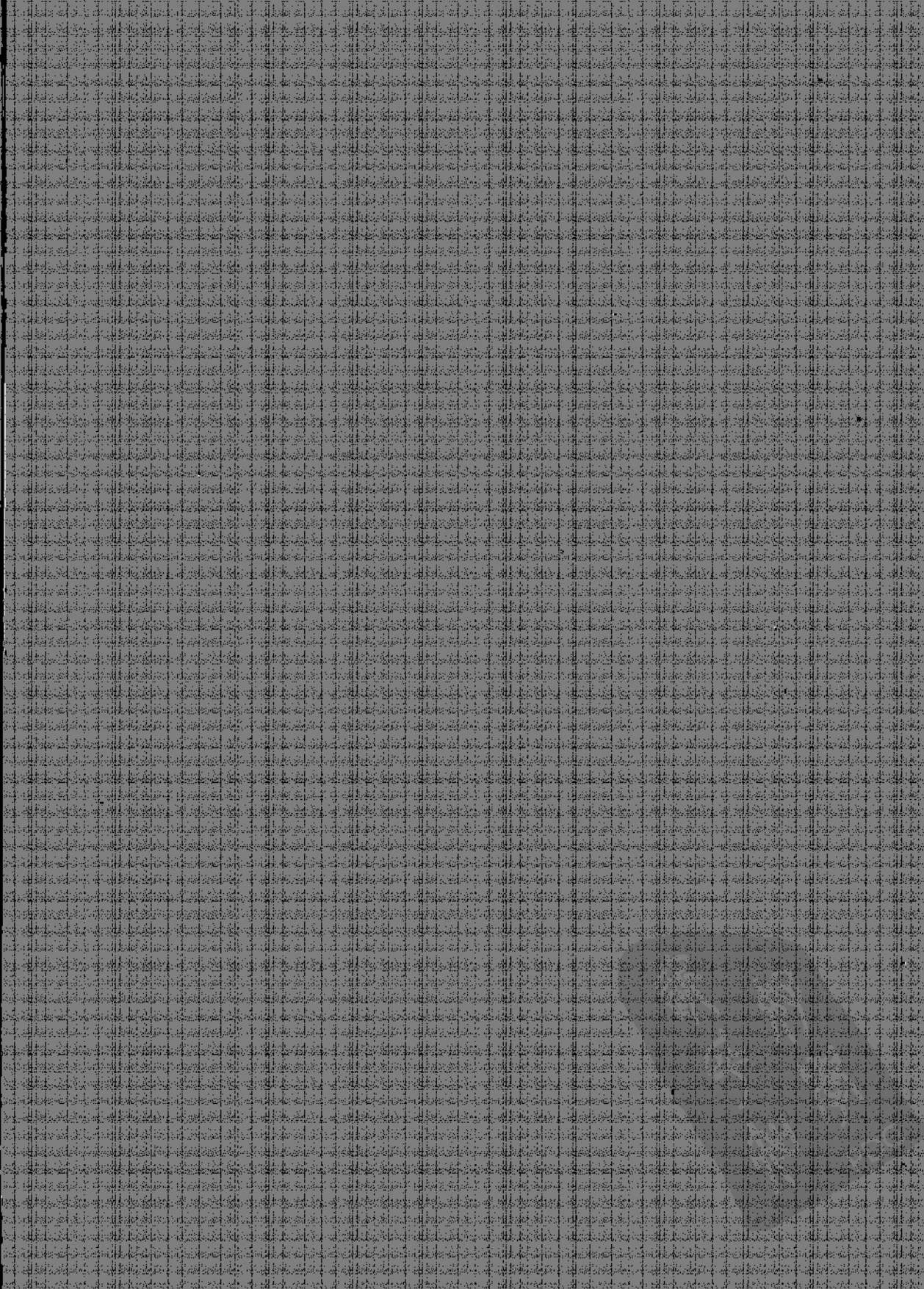
在上面的一个实例中，讲解了声明异常 throw 处理方法，下面将展示一段代码，巩固这一个知识点，读者读完程序，给出结果，将代码复制到计算机中进行编译，得到结果，对比自己的答案与编译结果是否一致，其代码如下：

```
public class YiThrow
{
    public static void main(String args[])
    {
        try
        {
            throw new ArrayIndexOutOfBoundsException();
        }
        catch(ArrayIndexOutOfBoundsException aoe){
            System.out.println("异常:"+aoe);
        }
        try
        {
            throw new ArithmeticException();
        }
        catch(ArithmeticException ae)
        {
            System.out.println("异常:"+ae);
        }
    }
}
```

13.4.3 创建自定义异常

前面讲了几种异常，又讲解了抛出异常的两种方式，不过在一些特殊情况下，往往需要自己创建异常，创建自定义异常只需要继承 Throwable 类或者它的子类 Exception 就可以了，自定义异常知识让系统把它看成一种异常来对待，由于自定义异常继承 Throwable 类，所以也继承它的方法，Throwable 类的方法如下。

- fillInStack Trace()。
- getCause()。
- getLocalizedMessage()。



第 17 课 HTML 和 XML

Java 是一种重要的网络编程语言，它也是 J2EE 的重要组成部分，在现实生活中，读者可以知道许多系统都是 Java 通过 Web 开发的，如中国移动的网上客户中心、中国农业银行的网上银行等。前面虽然学习了许多的类和接口，用户已经掌握了 Java 的基本功能，但是读者仍然不会使用 Java 开发 Web 程序，本课为了让读者进入 Java 的网络编程的知识，故讲解了网页方面的基础知识 HTML 和 XML。

17.1 本课学习目标

在本课的内容中，将讲解 HTML 和 XML 等知识点，其具体学习目标如下：

- HTML 简介。
- 字体属性标记和标识标记。
- 区域和段落标记。
- 表格标记。
- 表单标记。
- XML。

17.2 HTML 概述

HTML 是一种网页标记语言，它都是由标记组成的，在本章后面知识里，将会为大家显示它的各种标记。

17.2.1 HTML 的基本结构

HTML 是一种网页标记语言，它的所有部分都是标记“< >”和“</>”括起来，来看

下面的代码,其代码如下:

```
<html>
<head>
<title>这是网页的标题标签</title>
</head>
<body>
这是网页内容
</body></html>
```

上面展示的代码,其实就是一个很简单的网页,网页就是通过这种方式展现给浏览者的,各个参数介绍如下。

- `<html>.....</html>`: 这是 HTML 标签,所有标记都要放在这里,`<html>`是开始标签,`</html>`是标签的结束。
- `<head>.....</head>`: 表示网页的头部。
- `<title>.....</title>`: 表示网页的标题。
- `<body>.....</body>`: 表示网页的内容。

17.2.2 HTML 的标记特性

HTML 必须以`<html>`开始,以`</html>`结束,文件头包含在`<head>.....</head>`里面,文件体包含在`<body>.....</body>`里面,在文件头部,用户可以用`<title>.....</title>`标记来声明文件标题,在 HTML 文档中,值得提醒读者的是 HTML 也有注释,它和 Java 是完全不同的,HTML 采用“`<!--注释-->`”这个标记注释,在 HTML 中,每一个标记都是成对出现的,下面展示一段代码:

```
<html>
<head>
<title>欢迎进 Java 网络世界</title>
</head>
<body>
这里是 Java 网络世界!
</body>
</html>
```

将文件保存为 HTML 文件,双击打开,会得到如图 17-1 所示的效果



图 17-1 Java 网络世界的第一个页面

17.3 字体格式设置

字体是网页中经常出现的内容，不同的网页字体也不同，它们是如何实现这一个目标的呢，下面分别进行讲解。

17.3.1 标题的设置

在 HTML 中，用户可以通过 `<Hn>.....</Hn>` 来设置标题的大小，`n` 的值可以取 1~6 中的任意一个整数，下面通过一个 HTML 代码讲解一个问题，其代码如下：

```
<html>
<head>
<title>标题标记</title>
</head>
<body>
<h1>相信标题标记的力量</h1>
<h2>相信标题标记的力量</h2>
<h3>相信标题标记的力量</h3>
<h4>相信标题标记的力量</h4>
<h5>相信标题标记的力量</h5>
<body>
</html>
```

保存为 HTML，双击打开文件，得到如图 17-2 所示的结果。



图 17-2 标题标记

17.3.2 将字体加粗、倾斜和加底线

在创建网页的时候，将字体加粗、倾斜和加底线是避免不了的，它们到底是通过什么样的标记语言实现的呢？下面通过一段 HTML 语言进行讲解，其代码如下：

```
<html>
<head>
<title>加粗 倾斜 加底线</title>
</head>
```

```

<body>
相信标题标记的力量<br></br>
<b>相信标题标记的力量</b><br></br>
<i>相信标题标记的力量</i><br></br>
<u>相信标题标记的力量</u><br></br>
</body>
</html>

```

在代码中出现了几个新的标记，介绍如下。

- `.....`：将文字加粗。
- `
.....</br>`：用来换行。
- `<i>.....</i>`：将文字倾斜。
- `<u>.....</u>`：给文字加上底线。

执行代码，得到如图 17-3 所示的结果



图 17-3 将文字加粗、倾斜和加底线

17.3.3 将字体加上删除线、打字体和上标标记

在创建网页的时候，将加上删除线、打字体和上标标记是避免不了的，它们到底是通过什么样的标记语言实现的呢？下面通过一段 HTML 语言进行讲解，其代码如下：

```

<html>
<head>
<title>神奇的 HTML</title>
</head>
<body>
神奇的 HTML
<br></br>
<DEL>神奇的 HTML</DEL><br></br>
<TT>神奇的 HTML</TT><br></br>
神奇的 HTML
<SUP>神奇的 HTML</SUP>
</body>
</html>

```

在代码中出现了新的标记，介绍如下。

- `.....`：将文字加上删除线。
- `<TT>.....</TT>`：将文字设置成打字体。
- `<TT>.....</TT>`：将文字设置成上标。

执行代码，得到如图 17-4 所示的结果。



图 17-4 为文字加上删除标记、打字体和上标样式

17.3.4 设定字体大小、颜色、字形标记

这 3 种字体的属性是字体的常用格式，几乎所有网页都会设置这 3 种属性，它和前面有所不同，下面通过一段 HTML 代码（17-1.html）进行讲解。

```
<html>
<head>
<title>设置文字的格式</title>
</head>
<body>
<font color="#CC200" size="5" face="隶书">还好吗？现在过的无忧无虑还是仍然那样多愁善感？我好几次都在梦中梦到过你，你有的时候是哭着的，有的时候却又笑得毫无遮掩，弄得我不知所措，搞不清是该安慰还是该保持沉默，可等我醒了以后，却发现你好像在梦里什么都没有说过，只是哭或者笑，于是我猜，你肯定是有说不出的悲伤和快乐。</font>
<br>
<font color="#ee00FF" size="4" face="宋体">弄得我不知所措，搞不清是该安慰还是该保持沉默，可等我醒了以后，却发现你好像在梦里什么都没有说过，只是哭或者笑，于是我猜，你肯定是有说不出的悲伤和快乐。</font>
</body>
</html>
```

如果要设置字体大小、颜色和字形用户可以在这个首标签里设置，其参数介绍如下。

- `color=" "`：设置颜色。
- `size=" "`：设置字号。
- `face=" "`：设置字体。

执行代码，得到如图 17-5 所示的结果。

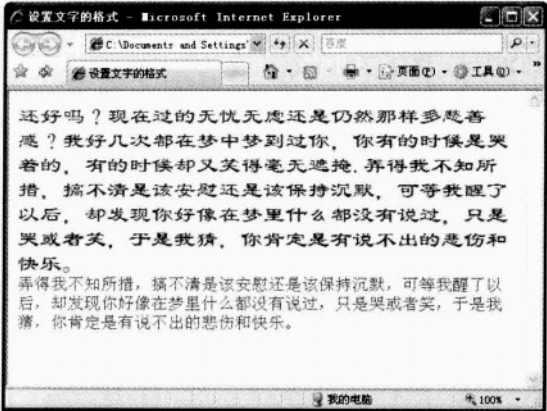


图 17-5 设置字体

17.4 标识标记的使用

在 HTML 语言中，为了使用显示的文字更加工整，条理顺序更加明朗，就要用到标识标记，下面通过一段 HTML 代码进行讲解，其代码（17-2）如下：

```
<html>
<head>
  <title> 标识标记</title>
</head>
<body>
  <i>中国人
  <i>英国人
  <i>德国人
  <ol type=i>
    <i>打开冰箱门
    <i>把它装进去
    <i>关上冰箱门
  </ol>
  <DL>
    <DT>性别: <DD>男、女
    <DT>职业 :<DD>工程师、教师、程序员
  </DL>
</body>
</html>
```

参数介绍如下。

- <i>: 是设置项目。
- ……: 它和<i>组合将形成带编号的项目，编号采取什么字体，取决于 type。
- <dt>: 用于定义项目。

- <dd>: 定义资料。
- <dl>.....</dl>: 定义标识。

执行代码, 得到如图 17-6 所示的结果。

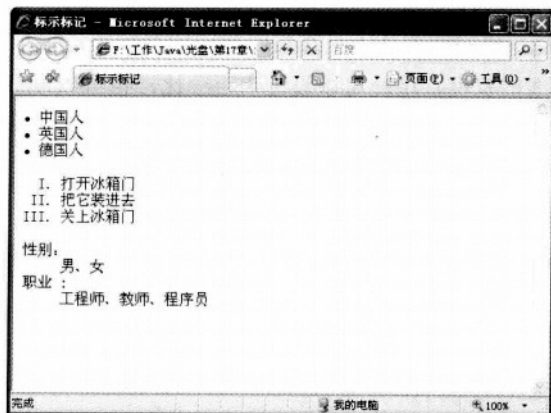


图 17-6 标识标记

17.5 区域和段落标记的使用

区域和段落在 HTML 是必不可少的, 在前面其实读者已经看到通过
.....</br>换行, 这里就不多讲, 这里讲解几个重要的区域标记和段落标记。

17.5.1 <hr>水平线

在许多页面中, 为了文字的美观性, 常插入水平线标记, 下面将通过一段代码进行讲解几种绘制分隔线的方法, 其代码 (17-3) 如下:

```
<html>
<head>
  <title>水平线的插入</title>
</head>
<body>
  绘制水平线
  <hr>
  绘制水平线
  <hr width="120%">
  绘制分隔字符串的水平线
  <hr width="30%" size="4">
  绘制分隔字符串的水平线
  <hr width="400" size="30" noshade>
  水平线的不同对齐方式
  <hr align="left" width="400" size="10">
```

```

<hr align="center" width="400" size="10">
<hr align="right" width="400" size="10">
</body>
</html>

```

参数介绍如下。

- `<hr>.....</hr>`：水平线的插入，在前面标记的参数那是水平线的属性。
- **Width**：水平线的宽度，用户可以设置为百分比，也可以设置为像素。
- **Align**：水平线位置的设置，用户可以设置 `left`，表示居左边对齐，设置 `center`，表示居中对齐，设置 `right`，表示居右边对齐。

双击打开网页，看到如图 17-7 所示的结果。

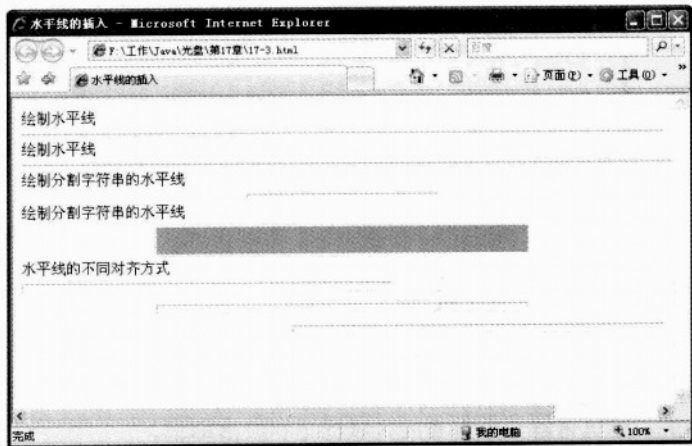


图 17-7 水平线的插入

17.5.2 `<p>.....</p>`段落标记

在段落间，用户可以通过`<p>.....</p>`让网页之间形成一行空白，需要注意的是用户可以不写`</p>`，下面通过代码进行讲解，其代码（17-4）如下：

```

<html>
<head>
<title>我的心跟着希望在动</title>
</head>
<body>
<p>
我的未来不是梦
</p>
<p>
我的心跟着希望在动
</body>
</html>

```

执行代码，得到如图 17-8 所示的结果。



图 17-8 段落标记

17.6 表格标记的使用

Java 是动态优秀的设计语言，在许多时候肯定会为浏览者表现一些数据，而表格是表现数据的最好工具，无论如何，Java 优秀的编程设计者是离不开表格标记的语言的，下面将详细表格标记的使用。

17.6.1 <table>容器标记

表格实际上是一个容器，理解它十分简单，用它来装各种东西，下面通过一段代码进行讲解，其代码（17-5）如下：

```
<html>
<head>
<title>表格</title>
</head>

<body>
<table width="200" border="1">
  <tr>
    <td width="63">姓名</td>
    <td width="71">语文</td>
    <td width="44">数学</td>
  </tr>
  <tr>
    <td>张三</td>
    <td>78</td>
    <td>65</td>
  </tr>
  <tr>
    <td height="23">李四</td>
    <td>45</td>
    <td>67</td>
  </tr>
</table>
</body>
</html>
```


参数介绍如下。

- `<table>.....</table>`：表格区域，开始标签可以定义表格的属性，这里定义了表格的宽度和表格边框线的粗细。
- `<td>.....</td>`：单元格。
- `<tr>.....</tr>`：表格中的行。

执行代码，得到如图 17-9 所示的结果。

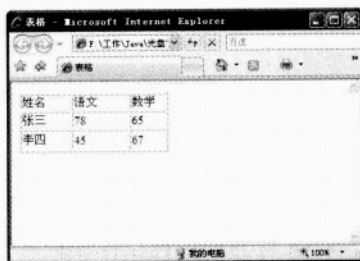


图 17-9 表格

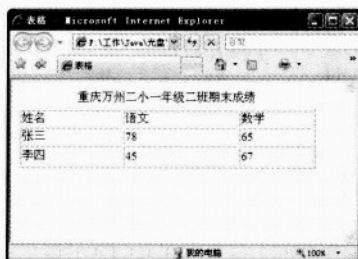
17.6.2 表格标题

用户可以通过一个`<caption>.....</caption>`标记为表格设置标题，设置十分简单，下面将一段 HTML 进行讲解，其代码（17-6.html）如下：

```
<html>
<head>
<title>表格</title>
</head>
<body>
<table width="400" border="1">
<caption align="center">重庆万州二小一年级二班期末成绩</caption>
<tr>
<td width="63">姓名</td>
<td width="71">语文</td>
<td width="44">数学</td>
</tr>
<tr>
<td>张三</td>
<td>78</td>
<td>65</td>
</tr>
<tr>
<td height="23">李四</td>
<td>45</td>
<td>67</td>
</tr>
</table>
</body>
</html>
```

其中,参数 Align 为水平线位置的设置,用户可以设置 left,表示居左边对齐,设置 center,表示居中对齐,设置 right,表示居右边对齐。

执行代码,得到如图 17-10 所示的结果。



The screenshot shows a web browser window titled '表格' (Table) with the address bar displaying 'http://www.163.com/'. The main content area displays a table with the caption '重庆万州二小一年级二班期末成绩' (Final Exam Results of Class 2, Grade 1, Wanzhou No. 2 Primary School). The table has three columns: '姓名' (Name), '语文' (Chinese), and '数学' (Mathematics). The data rows are as follows:

姓名	语文	数学
张三	78	65
李四	45	67

图 17-10 表格标题

17.6.3 表格中的标题栏

在前面的学习成绩表里,虽然有标题栏,但是它和普通的一样没有什么区别,在表格里,有专门的标题栏标记<th>……</th>,下面通过一段 HTML 代码进行讲解,其代码(17-7.html)如下:

```
<html>
<head>
<title>表格</title>
</head>

<body>
<table width="400" border="1">
<caption align="center">重庆万州二小一年级二班期末成绩</caption>
<tr>
<tr><th colspan="3">语文和数学成绩</th></tr>
<th>姓名</th>
<th>语文</th>
<th>数学</th>
</tr>
<tr>
<td>张三</td>
<td>78</td>
<td>65</td>
</tr>
<tr>
<td height="23">李四</td>
<td>45</td>
<td>67</td>
</tr>
</table>
</body>
</html>
```

执行代码, 得到如图 17-11 所示的结果。

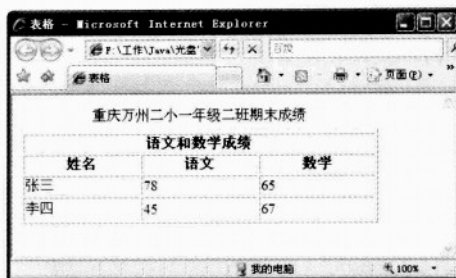


图 17-11 表格中的标题栏

17.7 表单标记的使用

在 HTML 中, 表单的重要性不言而喻, 它是服务器和浏览者交换的窗口, 本节将讲解表单的控件和表单的组件。

17.7.1 表单容器

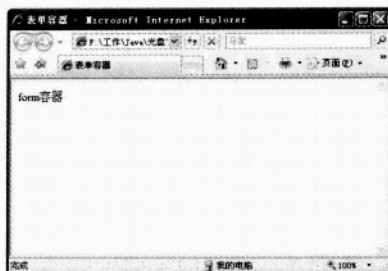
在 HTML 中, `<form>.....</form>` 表示表单的容器, 它建立后, 才能建立各个组件, 下面通过一段 HTML 代码进行讲解, 其代码 (17-8.html) 如下:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>表单容器</title>
</head>
<body>
  表单容器
  <form id="form1" name="form1" method="post" action="">
  </form>
</body>
</html>
```

参数介绍如下。

- `<form>`: 表单容器的标记。
- `id="form1"`: 表单的 ID 名称, 名称是 form1。
- `name="form1"`: 表单名称。
- `method="post"`: 数据的传送方式。
- `action=""`: 传送页面的设置, 用户可以设置一个 Java 的 Web 页面用来处理这个信息。

执行代码, 得到如图 17-12 所示的结果。



17-12 表单容器

17.7.2 单行文本框

单行文本框是一种常用的组件，下面创建一个单行文本框，其代码（17-9.html）如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>文本框</title>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
  请输入你的名字：
  <input type="text" name="textname" id="textname" />
</form>
</body>
</html>
```

文本框的属性参数有很多，除了上面的，还有 size、value 等，用户不必记住，在后面会讲解如何通过可视化操作处详细说明，执行代码，得到如图 17-13 所示的结果。



图 17-13 单行文本框

17.7.3 密码文本框

密码文本框也是比较常见的表单元素，下面通过一段代码进行讲解，其代码（17-10.html）如下：

```
<html>
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>密码文本框</title>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
请输入你的名字:
    <input type="text" name="textname" id="textname" />
请输入密码:
    <input type="password" name="password" id="password" />
</form>
</body>
</html>

```

执行代码, 得到如图 17-14 所示的效果。



图 17-14 密码文本框

17.7.4 单选按钮

单选按钮只能选择一个, 单选按钮是如何实现的呢? 下面进行讲解, 其代码(17-11.html)如下:

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>单选按钮</title>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
<p>
    <input type="radio" name="radio" id="D1" value="D1" /> 橘子
    <br />
    <input type="radio" name="radio" id="D2" value="D2" /> 苹果
    <br />
    <input type="radio" name="radio" id="D3" value="D3" />
    栗子
</form>
</body>
</html>

```

执行代码, 得到如图 17-15 所示的结果。



图 20-1 复制 servlet-api.jar 文件



图 20-2 粘贴文件

(3) 然后将*.java 进行编译，这里编译 TextServlet.java，编译后如图 20-3 所示。

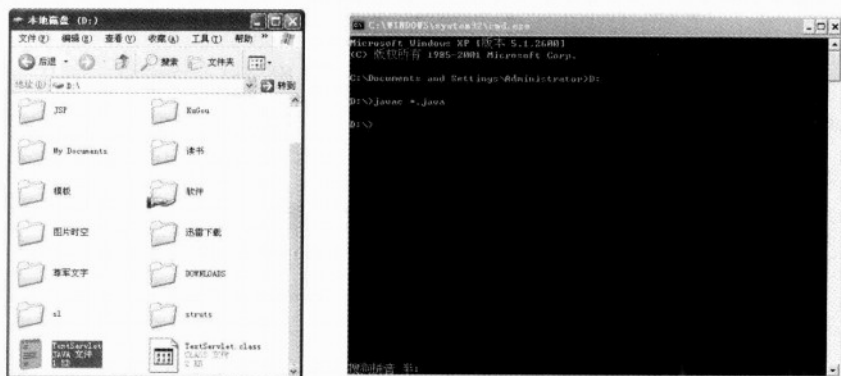


图 20-3 编译文件

(4) 打开 Tomcat 6.0\webapps，打开地址是“C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps”，然后新建一个文件夹：20-1，如图 20-4 所示。

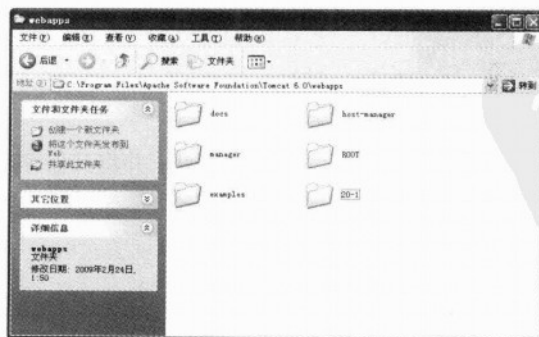


图 20-4 新建 20-1 文件夹

(5) 用户新建几个文件夹, 将 TextServlet.java 复制到 20-1 文件夹(可以不复制, 这和运行没有关系)中, 然后新建两个文件夹, 如图 20-5 所示。

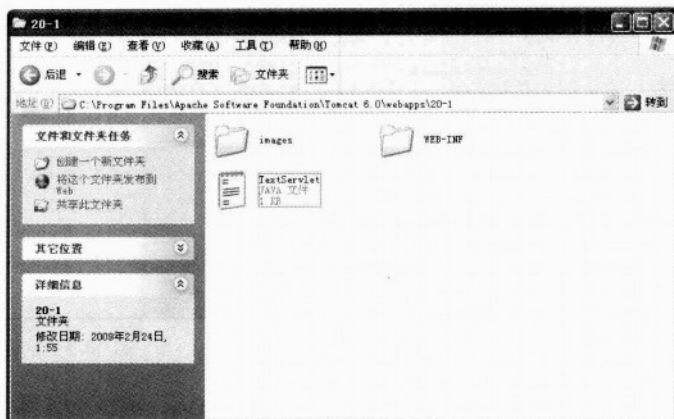


图 20-5 新建文件夹

(6) 用户将文件 “b01.jpg” 复制到 images 文件夹(用来显示背景图, 与上一节的程序代码 “out.println("<BODY background=images/b01.jpg>")” 一一对应)中, 然后打开 WEB-INF 文件夹, 在其中新建两个文件夹: classes 和 lib, 再新建一个 Web.xml, 如图 20-6 所示。



图 20-6 WEB-INF 文件夹

(7) 用户用记事本打开 Web.xml, 然后写入代码, 并将其保存, 代码如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
```

```

<description>This is the description of my J2EE component</description>
<display-name>This is the display name of my J2EE component</display-name>
<servlet-name>TextServlet</servlet-name>
<servlet-class>com.wy.TextServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TextServlet</servlet-name>
  <url-pattern>/TextServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

(8) 按照包的路径 (package com.wy;) 在 classes 中新建文件夹, 然后将编译的后缀名为.class 文件复制到下面, 如图 20-7 所示。

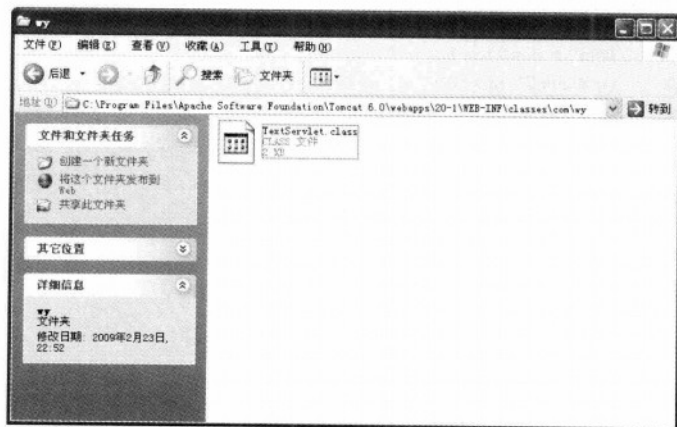


图 20-7 复制编译的文件

(9) 打开浏览器, 在浏览器的地址栏中输入 “http://localhost/20-1/testervlet”, 如图 20-8 所示。

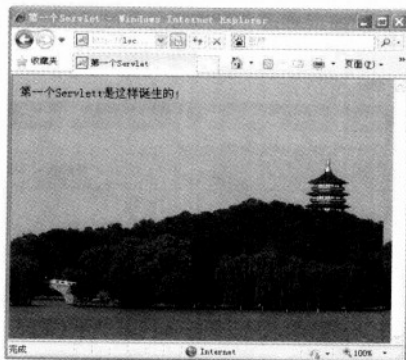


图 20-8 最终效果

实例探索和读者练习:

在上面的实例中, 创建了一个 Servlet 类, 然后进行编译, 下面展示一个 Servlet 源代码, 其代码如下:

```
package cn.dt.web;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class Welcome extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException
    {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>welcome</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h3>");
        out.println("你好! ");
        out.println("欢迎你访问该网站! <br>");
        out.println("</body>");
        out.println("</html>");
    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException
    {
        this.doGet(request,response);
    }
}
```

将这个源文件进行编译, 然后将其部署并发布得到如图 20-9 所示的效果(参考答案在光盘下 20-2 中)。



图 20-9 Servlet 效果

20.4 Servlet 的常用接口和类

在 Java 中，有许多的 Servlet 接口和类，让 Servlet 的功能变得十分强大，下面将详细介绍使用频率较高的常用接口和类。

20.4.1 与 Servlet 配置相关的接口

与 Servlet 配置有关系的接口是 `java.servlet.ServletCongfig` 接口，下面就是一个简单的 Servlet 的配置，其配置如下：

```
<servlet-name>TextServlet</servlet-name>//
  <servlet-class>com.wy.TextServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TextServlet</servlet-name>
  <url-pattern>/TextServlet</url-pattern>
</servlet-mapping>
```

在 Java API 手册中有许多方法，几个比较常用的方法介绍如下。

- `getinitParmeter(String name)`: 返回指定名称的参数的字符串。
- `getinitparameterName()`: 返回 Servlet 所有初始化的值。
- `getServletcontext()`: 返回当前 Servlet 正在执行的上下文的引用。
- `getServletName()`: 返回当前 Servlet 实例的名称。
- `getContext(String uripath)`: 返回由参数指定的一个对象。
- `long(String msg)`: 向 Servlet 的日志写入信息。
- `getServletinfo()`: 返回当前运行的 Servlet 容器名称和版本代码。
- `getAttribute(String name)`: 返回由参数指定的名称。
- `getAttributeNames()`: 返回由容器中所有属性指定的名称。
- `getServletContextName`: 返回当前 Web 应用程序的 `ServletContext` 的名称。

20.4.2 Servlet 编程接口

Servlet 编程需要引用 `javax.servlet` 包和接口，Servlet 的常用方法如下。

1. Servlet 初始化

在 Web 容器加载和实例化 Servlet 类之后、Servlet 实例传递来自客户端的请求之前，Web 容器对 Servlet 进行初始化。用户可以自定义这个初始化过程，以允许 Servlet 读持久的配置数据、初始化资源，其初始化的格式如下：

```
Public void init (ServletCongfig arg0) throws servletException
```

2. Servlet 业务实现

在编程的过程中，Servlet 的业务实现是平常的，Servlet 的业务实现格式如下：

```
Public void service (ServletRequest arg0,ServletResponse arg) throws
    ServletException,IOException
```

3. Servlet 销毁

执行完毕时，不再需要一些内容，这时候就需要执行销毁这个动作，销毁的格式如下：

```
Public void destroy()
```

4. Servlet 和 Web 容器进行通信

与 Web 通信十分简单，该方法的格式如下：

```
Public servletconfig getServletConfig()
```

5. 获取 Servlet 的基本信息

获取 Servlet 格式如下：

```
Public servlet getServletInfo()
```

6. Servlet 接口应用

上面讲解了这么多的方法，下面通过一段代码来巩固这个方法，其代码如下：

```
package com.wy;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TextServlet implements Servlet
{
    public void init(ServletConfig arg0) throws ServletException {
        System.out.println("执行 init()方法");
    }

    public void service(ServletRequest arg0, ServletResponse arg1)
        throws ServletException, IOException {
        System.out.println("执行 service()方法");
    }

    public void destroy() {
        System.out.println("destroy()方法");
    }

    public ServletConfig getServletConfig() {
        return null;
    }

    public String getServletInfo() {
        return null;
    }
}
```

运行 Servlet，必须要配置 Web.xml 文件，其配置如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <description>This is the description of my J2EE component</description>
    <display-name>This is the display name of my J2EE component</display-name>
    <servlet-name>TextServlet</servlet-name>
    <servlet-class>com.wy.TextServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>TextServlet</servlet-name>
    <url-pattern>/TextServlet</url-pattern>
  </servlet-mapping>
</web-app>

```

上面这个程序为空，在浏览器中输入浏览地址“http://localhost/20-3/TextServlet”，得到的结果也是一个空白页面，如图 20-10 所示。

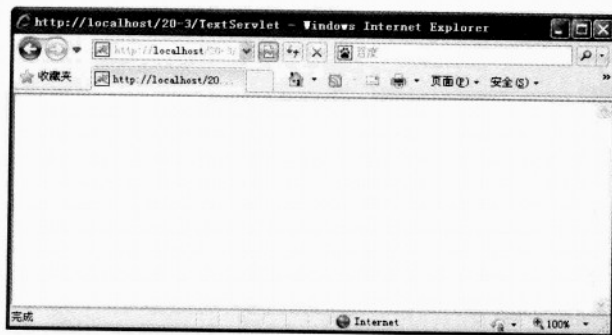


图 20-10 浏览结果

倘若用控制器查看的话，第一次会看到如下的结果。

- 执行 init()方法。
- 执行 service()方法。

重新访问网页一次，控制器会出现如下的结果。

- 执行 init()方法。
- 执行 service()方法。
- 执行 service()方法。

20.4.3 HttpServlet 类

这个类是针对 HTTP 协议的 Web 服务器的 Servlet 类的，它是对 GenericServlet 类的一

个扩展,同时它又是一个抽象的类,一个 `HttpServlet` 的子类必须至少重载以下方法中的一个,其方法如下。

- `doGet()`方法,适用于 HTTP Get 请求。
- `doPost()`方法,适用于 HTTP Post 请求。
- `doPut()`方法,适用于 HTTP Put 请求。
- `doDelete()`方法,适用于 HTTP Delete 请求。
- `init()`和 `destroy()`方法,管理 Servlet 生命周期中的资源。
- `getServletInfo()`方法,提供 Servlet 本身的信息

`HttpServlet` 类中的 `getServletConfig()`、`getServletContext()`、`getServletInfo()`、`getServletName()`、`log()`等方法与 `GenericServlet` 类中的同名方法功能相同,本节不再赘述。

HTTP 的构造方法如下:

```
public HttpServlet ()
```

TIPS

该构造方法什么也不做,因为该类是一个抽象类。

下面将通过一段代码来讲解 `HttpServlet` 的作用,其代码(20-05)如下:

```
package com.wy;
import java.io.*;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class InitServlet extends HttpServlet
{
    private String driver = "";
    private String URL = "";
    private String username = "";
    private String password = "";
    public void init() throws ServletException
    {
        driver = getInitParameter("driver");
        URL = getInitParameter("URL");
        username = getInitParameter("username");
        password = getInitParameter("password");
    }
    //获得数据库连接的方法
    public Connection getConnection()
    {
        Connection con = null;
        try {
            Class.forName(driver);
            con = DriverManager.getConnection(URL, username,
```



```

        This is the description of my J2EE component
    </description>
    <display-name>
        This is the display name of my J2EE component
    </display-name>
    <servlet-name>InitServlet</servlet-name>
    <servlet-class>com.wy.InitServlet</servlet-class>
    <init-param>
        <param-name>driver</param-name>
        <param-value>
            com.microsoft.jdbc.sqlserver.SQLServerDriver
        </param-value>
    </init-param>
    <init-param>
        <param-name>URL</param-name>
        <param-value>
            jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_database19
        </param-value>
    </init-param>
    <init-param>
        <param-name>username</param-name>
        <param-value>sa</param-value>
    </init-param>
    <init-param>
        <param-name>password</param-name>
        <param-value></param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>InitServlet</servlet-name>
    <url-pattern>/InitServlet</url-pattern>
</servlet-mapping>
</web-app>

```

在这里，用户可以进行编译与配置，但是会出错，因为这里没有建立数据库，数据库的知识，将在下一课讲解，这里只让读者对 HttpServlet 知识点进行理解。

20.4.4 请求与响应

接口 `HttpServletRequest`、`HttpServletResponse` 是用来请求与响应的，它提供了很多方法，用户可以去查看 Java API，这里不再赘述。

下面通过一个实例讲解这个 Web 请求与响应的实例。

实例 68：新建一个请求与响应的实例，其代码（20-4）如下：

```

package co.cm.ino;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;

```

```

public class studentInfo extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException
    {
        request.setCharacterEncoding("gb2312");
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>基本信息</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>学生的基本信息</h2>");
        out.println("<h3>姓名:"+request.getParameter("sname")+"</h3>");
        out.println("<h3>学号:"+request.getParameter("snumber")+"</h3>");
        out.println("<h3>班级:"+request.getParameter("sclass")+"</h3>");
        out.println("</body>");
        out.println("</html>");
    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException
    {
        this.doGet(request,response);
    }
}

```

然后编写 Web.xml，编写后如下：

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <servlet>
        <servlet-name>studentInfo</servlet-name>
        <servlet-class>co.cm.ino.studentInfo</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>studentInfo</servlet-name>
        <url-pattern>/studentInfo</url-pattern>
    </servlet-mapping>
</web-app>

```

最后编写一个名为 Info.html 的网页，其网页代码如下：

```

<html>
<head>
    <title></title>

```



```

</head>
<body>
    <form action="studentInfo" name="myform" method="post">
        姓名:<input type="text" name="sname"><br>
        学号:<input type="text" name="snumber"><br>
        班级:<input type="text" name="sclass"><br>
        <input type="submit" value="提交">
    </form>
</body>
</html>

```

然后将 Java 文件编译, 在浏览器中输入“http://localhost/20-4/Info.html”, 得到如图 20-11 所示的结果。



图 20-11 运行结果

实例探索和读者练习：

在上面的实例中, 创建了一个 HttpServlet 类, 然后进行编译, 下面展示一段源代码, 读者请认真阅读代码, 将代码阅读懂后, 进行配置和编译, 倘若配置不出来, 查看一下光盘中的 (20-5) 文件夹, 其代码如下:

```

package mypack;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloServlet extends HttpServlet//第一步: 扩展 HttpServlet 抽象类
{
    //第二步: 覆盖 doGet()方法
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)throws IOException,ServletException{
        //第三步: 获取 HTTP 请求中的参数信息
        String clientName=request.getParameter("clientName");
        if(clientName!=null)
            clientName=new String(clientName.getBytes("ISO-8859-1"),"GB2312");
        else
            clientName="我的朋友";
        //第四步: 生成 HTTP 响应结果

```

```

    }
    else if(e.getSource()==jMenuItem[0][3])
    {
        //另存新档
        save();
        try{
            int dotpos = drawPanel.filename.lastIndexOf('.');
            ImageIO.write(drawPanel.bufImg, drawPanel.filename.substring(dotpos + 1), new File
(drawPanel.filename));
        }
        catch(IOException even)
        {
            JOptionPane.showMessageDialog(null, even.toString(), "无法保存", JOptionPane.ERROR_
MESSAGE);
        }
    }
    else if(e.getSource()==jMenuItem[0][4])
    { //离开
        System.exit(0);
    }
    else if(e.getSource()==jMenuItem[3][0])
    ){ //关于
        JOptionPane.showMessageDialog(null, "程序名称: 七喜猫猫(2009/2/28)\n 作者: 七喜猫猫\n 信箱:
qiximm@263.com\n\n 版本特点: 七喜猫猫稳定版
\n", "于 七喜笨猫", 1, new ImageIcon("img/paint.gif"));
    }
    for(i=0;i<2;i++){
        if(jCheckBoxMenuItem[i].isSelected())
            jPanel[i+2].setVisible( true );
        else
            jPanel[i+2].setVisible( false );
    }
    if(jCheckBoxMenuItem[3].isSelected()){
        setPanel.setVisible( true );
        jPanel[4].setVisible( true );
    }
    else{
        setPanel.setVisible( false );
        jPanel[4].setVisible( false );
    }
    if(jCheckBoxMenuItem[2].isSelected())
        jLabel[0].setVisible( true );
    else
        jLabel[0].setVisible( false );
}

```

23.3.4 界面的实现

新建一个类, 在这个类中写入版面的布局, 然后写入一定的方法让软件实现其大部分功能, 其代码如下:

```

public class UnderDrawPanel extends JPanel
implements MouseListener, MouseMotionListener
{
    public int x,y;
    float data[]={2};
    public JPanel ctrl_area=new JPanel(),
    ctrl_area2=new JPanel(),ctrl_area3=new JPanel();
    public UnderDrawPanel()
    {
        this.setLayout(null);
        this.add(ctrl_area);
        this.add(ctrl_area2);
        this.add(ctrl_area3);
        ctrl_area.setBounds(new Rectangle(draw_panel_width+3,
draw_panel_height+3, 5, 5));
        ctrl_area.setBackground(new Color(0,0,0));
        ctrl_area2.setBounds(new Rectangle(draw_panel_width+3,
draw_panel_height/2, 5, 5));
        ctrl_area2.setBackground(new Color(0,0,0));
        ctrl_area3.setBounds(new Rectangle(draw_panel_width/2,
draw_panel_height+3, 5, 5));
        ctrl_area3.setBackground(new Color(0,0,0));
        ctrl_area.addMouseListener(this);
        ctrl_area.addMouseMotionListener(this);
        ctrl_area2.addMouseListener(this);
        ctrl_area2.addMouseMotionListener(this);
        ctrl_area3.addMouseListener(this);
        ctrl_area3.addMouseMotionListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
    }

    public void mousePressed(MouseEvent e)
    {
    }

    public void mouseReleased(MouseEvent e)
    {
        draw_panel_width=x;
        draw_panel_height=y;
        ctrl_area.setLocation(draw_panel_width+3,draw_panel_height+3);
        ctrl_area2.setLocation(draw_panel_width+3,draw_panel_height/2+3);
        ctrl_area3.setLocation(draw_panel_width/2+3,draw_panel_height+3);
        drawPanel.setSize(x,y);
        drawPanel.resize();
        repaint();
    }

    public void mouseEntered(MouseEvent e)
    {
    }
}

```

```

        public void mouseExited(MouseEvent e)
        {
        }

        public void mouseDragged(MouseEvent e)
        {
            if(e.getSource()==ctrl_area2)
            {
                x = e.getX()+draw_panel_width;
                y = draw_panel_height;
            }
            else if(e.getSource()==ctrl_area3)
            {
                x = draw_panel_width;
                y = e.getY()+draw_panel_height;
            }
            Else
            {
                x = e.getX()+draw_panel_width;
                y = e.getY()+draw_panel_height;
            }
            repaint();
            jLabel[0].setText(x+" "+y);
        }
        public void mouseMoved(MouseEvent e)
        {
        }

        public void paint(Graphics g)
        {
            Graphics2D g2d = (Graphics2D) g;
            super.paint(g2d);
            g2d.setPaint( new Color(128,128,128) );
            g2d.setStroke( new BasicStroke( 1, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER, 10, data,
0));
            g2d.draw( new Rectangle2D.Double( -1, -1, x+3, y+3 ) );
        }

        public class SetPanel extends JPanel
        implements ItemListener, ChangeListener, ActionListener{
            private JPanel jPanel_set1=new JPanel();
            private JPanel jPanel_set2=new JPanel();
            private JPanel temp0=new JPanel(new GridLayout(4,1)), temp1=new JPanel(new FlowLayout
(FlowLayout.LEFT)), temp2=new JPanel(new FlowLayout
(FlowLayout.LEFT)), temp3=new JPanel(new FlowLayout(FlowLayout.LEFT)), temp4=new JPanel(new FlowLayout
(FlowLayout.LEFT)), temp5=new JPanel(new FlowLayout
(FlowLayout.LEFT)), temp6=new JPanel(new FlowLayout(FlowLayout.LEFT)), temp7=new JPanel(new FlowLayout
(FlowLayout.LEFT)), temp8=new JPanel(new GridLayout
(3,1));
            public JCheckBox jCheckBox = new JCheckBox();
            private BufferedImage bufImg =
            new BufferedImage(50,50,BufferedImage.TYPE_3BYTE_BGR);

```



```

private JLabel jlbImg=new JLabel();
float data[]={20};
JLabel pie[]=new JLabel[3];
public int number=5;
JSpinner lineWidthSelect = new JSpinner();
JRadioButton style[] = new JRadioButton[ 5 ];
ButtonGroup styleGroup =
new ButtonGroup() .pieGroup = new ButtonGroup();

public SetPanel(){//生成版面
    this.setLayout(null);
    this.add(jPanel_set1);

    jlbImg.setIcon(new ImageIcon(bufImg));
    jPanel_set1.setLayout(new FlowLayout());
    jPanel_set1.setBounds(new Rectangle(0, 0, 100, 160));
    jPanel_set1.setBorder( new TitledBorder(null,
"边框",TitledBorder.LEFT, TitledBorder.TOP) );
    lineWidthSelect.setValue(new Integer(5));
    for(i=0;i<=1;i++)
    {
        style[i] = new JRadioButton();
        styleGroup.add(style[i]);
        style[i].addActionListener(this);
    }
    style[0].setSelected( true );
    temp1.add(new JLabel("大小:"));
    temp1.add(lineWidthSelect);
    temp2.add(new JLabel("虚线:"));
    temp2.add(jCheckBox);

    temp3.add(new JLabel("圆角:"));
    temp3.add(style[0]);

    temp4.add(new JLabel("尖角:"));
    temp4.add(style[1]);

    temp0.add(temp1);
    temp0.add(temp2);
    temp0.add(temp3);
    temp0.add(temp4);

    jPanel_set1.add(temp0);
    lineWidthSelect.addChangeListener( this );
    jCheckBox.addItemListener( this );

    jPanel_set2.setBounds(new Rectangle(0, 170, 100, 130));
    jPanel_set2.setBorder( new TitledBorder(null, "扇型设定",TitledBorder.LEFT, TitledBorder.TOP) );
    for(i=2;i<=4;i++)

```

```

{
    style[i] = new JRadioButton();
    pieGroup.add(style[i]);
    style[i].addActionListener(this);
}
style[4].setSelected( true );
pie[0] = new JLabel("弦状:");
temp5.add(pie[0]);
temp5.add(style[2]);

pie[1] = new JLabel("开放:");
temp6.add(pie[1]);
temp6.add(style[3]);

pie[2] = new JLabel("派状:");
temp7.add(pie[2]);
temp7.add(style[4]);

temp8.add(temp5);
temp8.add(temp6);
temp8.add(temp7);

temp8.setPreferredSize(new Dimension( 71 , 95 ));

jPanel_set2.add(temp8);
this.add(jPanel_set2);

pie_remove_ctrl();
stroke = new BasicStroke(5,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
}

public void pencil_add_ctrl()
{
    style[0].setSelected(true);
    style[1].setEnabled(false);
    jCheckBox.setSelected(false);
    jCheckBox.setEnabled(false);
    BasicStroke stroke2 = (BasicStroke) stroke;
    stroke = new BasicStroke(stroke2.getLineWidth(),
BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
}

public void pencil_remove_ctrl()
{
    style[1].setEnabled(true);
    jCheckBox.setEnabled(true);
}

public void pie_add_ctrl(){

```

```

        pie[0].setEnabled(true);
        pie[1].setEnabled(true);
        pie[2].setEnabled(true);
        style[2].setEnabled(true);
        style[3].setEnabled(true);
        style[4].setEnabled(true);
    }

    public void pie_remove_ctrl()
    {
        pie[0].setEnabled(false);
        pie[1].setEnabled(false);
        pie[2].setEnabled(false);
        style[2].setEnabled(false);
        style[3].setEnabled(false);
        style[4].setEnabled(false);
    }

    public void actionPerformed( ActionEvent e )
    {
        BasicStroke stroke2 = (BasicStroke) stroke;
        if ( e.getSource() == style[0] )
            stroke = new BasicStroke( stroke2.getLineWidth(), BasicStroke.CAP_ROUND,
stroke2.getLineJoin(), stroke2.getMiterLimit(),
stroke2.getDashArray(), stroke2.getDashPhase() );
        else if ( e.getSource() == style[1] )
            stroke = new BasicStroke( stroke2.getLineWidth(), BasicStroke.CAP_BUTT,
stroke2.getLineJoin(), stroke2.getMiterLimit(),
stroke2.getDashArray(), stroke2.getDashPhase() );
        else if ( e.getSource() == style[2] )
            drawPanel.pie_shape=Arc2D.CHORD;
        else if ( e.getSource() == style[3] )
            drawPanel.pie_shape=Arc2D.OPEN;
        else if ( e.getSource() == style[4] )
            drawPanel.pie_shape=Arc2D.PIE;
    }

    public void stateChanged(ChangeEvent e)
    {
        number = Integer.parseInt
(lineWidthSelect.getValue().toString());
        if(number <= 0)
        {
            lineWidthSelect.setValue(new Integer(1));
            number = 1;
        }
        BasicStroke stroke2 = (BasicStroke) stroke;
        stroke = new BasicStroke( number, stroke2.getEndCap(), stroke2.getLineJoin(), stroke2.getMiterLimit(),
stroke2.getDashArray(),
stroke2.getDashPhase() );
    }

```



```

    }

    public void itemStateChanged( ItemEvent e )
    {
        BasicStroke stroke2 = (BasicStroke) stroke;
        if ( e.getSource() == jCheckBox )
        {
            if ( e.getStateChange() == ItemEvent.SELECTED )
            stroke = new BasicStroke( stroke2.getLineWidth(), stroke2.getEndCap(),
            stroke2.getLineJoin(), 10, data, 0 );
            else
            stroke = new BasicStroke(stroke2.getLineWidth(), stroke2.getEndCap(),
            stroke2.getLineJoin());
        }
    }

    public Dimension getPreferredSize()
    {
        return new Dimension( 100, 300 );
    }
}

public class Gradient extends JPanel{//渐变预览用
public Color G_color_left = new Color(255,255,255);
public Color G_color_right = new Color(0,0,0);
public Gradient()
{
    repaint();
}

    public void paint(Graphics g)
    {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setPaint( new GradientPaint( 0, 0, G_color_left, 100, 0, G_color_right, true ) );
        g2d.fill( new Rectangle2D.Double( 0, 0, 100, 25 ) );
    }

    public Dimension getPreferredSize()
    {
        return new Dimension( 100, 25 );
    }
}

```

23.3.5 调色盘实现

新建一个类，通过这个类实现底端的调色功能，其代码如下：

```

public class ColorPanel extends JPanel
implements MouseListener, ActionListener
{//调色盘 class
    private JPanel jPanel_color0[]=new JPanel[5];
    private JPanel jPanel_color1[]=new JPanel[32];

```



```

        private JPanel jPanel_color2[]=new JPanel[32];
        private ImageIcon special_color[]= new ImageIcon[4];
        private BufferedImage bufImg =
new BufferedImage(12,12,BufferedImage.TYPE_3BYTE_BGR) ,bufImg2 = new BufferedImage(12
,12,BufferedImage.TYPE_3BYTE_BGR);
        private JLabel jlImg=new JLabel() ,jlImg2=new JLabel();
        private ImageIcon icon;
        private JDialog jDialog;
        private JButton ok, cancel,left,right;
        private Gradient center = new Gradient();

        private int rgb[][]={
                {0,255,128,192,128,255,128,255,0,0,0,0,0,128,255,128,255,0,0,0,128,0,128,128,255,128,255,255,
255,255,255},
                {0,255,128,192,0,0,128,255,128,255,128,255,0,0,0,128,255,64,255,128,255,64,128,0,0,64,128,255,
255,255,255},
                {0,255,128,192,0,0,0,0,0,128,255,128,255,128,255, 64,128, 64,128,255,255,128,255, 55,128,0,64,
255,255,255,255}
        };

        public ColorPanel()

        { //产生版面

                addMouseListener( this );
                jlImg.setIcon(new ImageIcon(bufImg));
                jlImg2.setIcon(new ImageIcon(bufImg2));

                special_color[0] = new ImageIcon( "img/icon1.gif" );
                special_color[1] = new ImageIcon( "img/icon2.gif" );
                special_color[2] = new ImageIcon( "img/icon3.gif" );
                special_color[3] = new ImageIcon( "img/icon4.gif" );

                this.setLayout(null);
                color_border=new Color(0,0,0);
                color_inside=null;

                for(i=0;i<jPanel_color0.length;i++){
                        jPanel_color0[i]=new JPanel();
                        if(i<=2){
                                jPanel_color0[i].setBorder(BorderFactory.createEtchedBorder(BevelBorder.RAISED));
                                jPanel_color0[i].setLayout(null);
                        }
                        else{
                                jPanel_color0[i].setBackground(new Color(rgb[0][i-3],rgb[1][i-3],rgb[2][i-3]));
                                jPanel_color0[i].setLayout(new GridLayout(1,1));
                                jPanel_color0[i-2].add(jPanel_color0[i]);
                        }
                }
                for(i=0;i<jPanel_color2.length;i++){
                        jPanel_color2[i]=new JPanel();

```

```

        jPanel_color2[i].setLayout(new GridLayout(1,1));
        jPanel_color2[i].setBounds(new Rectangle(2, 2, 12, 12));
        jPanel_color2[i].setBackground(new Color(rgb[0][i],rgb[1][i],rgb[2][i]));
        if(i>=28)
            jPanel_color2[i].add(new JLabel(special_color[i-28]));
    }

    for(i=0;i<jPanel_color1.length;i++){
        jPanel_color1[i]=new JPanel();
        jPanel_color1[i].setLayout(null);
        jPanel_color1[i].add(jPanel_color2[i]);
        this.add(jPanel_color1[i]);
        if(i%2==0){jPanel_color1[i].setBounds(new Rectangle(32+i/2*16, 0, 16, 16));}
        else {jPanel_color1[i].setBounds(new Rectangle(32+i/2*16, 16, 16, 16));}
        jPanel_color1[i].setBorder(BorderFactory.createEtchedBorder(BevelBorder.RAISED));
    }

    jPanel_color0[3].add(jlImg);
    jPanel_color0[4].add(jlImg2);

    Graphics2D g2d = buffImg2.createGraphics();
    g2d.setPaint( Color.white );
    g2d.fill( new Rectangle2D.Double( 0, 0, 12, 12 ) );
    g2d.setPaint( Color.red );
    g2d.draw( new Line2D.Double( 0, 0, 12, 12 ) );
    g2d.draw( new Line2D.Double( 11, 0, 0, 11 ) );
    repaint();

    this.add(jPanel_color0[1]);
    this.add(jPanel_color0[2]);
    this.add(jPanel_color0[0]);

    jPanel_color0[0].setBounds(new Rectangle(0, 0, 32, 32));
    jPanel_color0[1].setBounds(new Rectangle(4, 4, 16, 16));
    jPanel_color0[2].setBounds(new Rectangle(12,12,16, 16));
    jPanel_color0[3].setBounds(new Rectangle(2, 2, 12, 12));
    jPanel_color0[4].setBounds(new Rectangle(2, 2, 12, 12));

    jDialog = new JDialog(Painter.this, "请选择两种颜色渐变", true);
    jDialog.getContentPane().setLayout(new FlowLayout());
    jDialog.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    jDialog.setSize(250, 110);
    JPanel temp = new JPanel(new GridLayout(2,1));
    JPanel up = new JPanel(new FlowLayout());
    JPanel down = new JPanel(new FlowLayout());

    ok = new JButton("确定");
    cancel = new JButton("取消");
    left = new JButton(" ");
    right = new JButton(" ");

```

```

        center.setBorder(BorderFactory.createEtchedBorder(BevelBorder.RAISED));
        up.add(left);
        up.add(center);
        up.add(right);
        down.add(ok);
        down.add(cancel);
        temp.add(up);
        temp.add(down);
        jDialog.getContentPane().add(temp);

        ok.addActionListener(this);
        cancel.addActionListener(this);
        left.addActionListener(this);
        right.addActionListener(this);
    }
    public void actionPerformed( ActionEvent e ){
        if(e.getSource() == left){
            center.G_color_left = JColorChooser.showDialog( Painter.this, "请选择边线颜色",
center.G_color_left );
            center.repaint();
        }
        else if(e.getSource() == right){
            center.G_color_right = JColorChooser.showDialog( Painter.this, "请选择边线颜色",
center.G_color_right );
            center.repaint();
        }
        else{
            jDialog.dispose();
        }
    }

    public Dimension getPreferredSize()
    {
        return new Dimension( 300, 32 );
    }
    public void mouseClicked( MouseEvent e )
    {
    }

    public void mousePressed( MouseEvent e ){
        Graphics2D g2d;
        if(e.getX()>=5 && e.getX()<=20 && e.getY()>=5 && e.getY()<=20)
        {
            g2d = buflmg.createGraphics();
            color_border = JColorChooser.showDialog( Painter.this, "请选择边线颜色",
(Color)color_border );
            g2d.setPaint(color_border);
            g2d.fill( new Rectangle2D.Double( 0, 0, 12, 12 ) );
            repaint();
        }
        else if(e.getX()>=13 && e.getX()<=28 && e.getY()>=13 && e.getY()<=28){

```



```

        g2d = bufImg2.createGraphics();
        color_inside = JColorChooser.showDialog( Painter.this, "请选择填充颜色",
(Color)color_inside );

        g2d.setPaint(color_inside);
        g2d.fill( new Rectangle2D.Double( 0, 0, 12, 12 ) );
        repaint();
    }

    if(!(e.getX()>=32 && e.getX()<=288)) return;
    int choose=(e.getX()-32)/16*2+e.getY()/16;

    if(e.getButton()==1)
//判断填充边框或填满内部
        g2d = bufImg.createGraphics();
    else
        g2d = bufImg2.createGraphics();

    if(choose==28){//填充无颜色
        g2d.setPaint( Color.white );
        g2d.fill( new Rectangle2D.Double( 0, 0, 12, 12 ) );
        g2d.setPaint( Color.red );
        g2d.draw( new Line2D.Double( 0, 0, 12, 12 ) );
        g2d.draw( new Line2D.Double( 11, 0, 0, 11 ) );
        repaint();

        if(e.getButton()==1)
            color_border=null;
        else
            color_inside=null;
    }
    else if(choose==29)
//填充渐变
        jDialog.show();

        g2d.setPaint( new GradientPaint( 0, 0, center.G_color_left, 12, 12, center.G_color_right, true ) );
        g2d.fill( new Rectangle2D.Double( 0, 0, 12, 12 ) );
        repaint();

        if(e.getButton()==1)
            color_border=new GradientPaint( 0, 0, center.G_color_left, 12, 12, center.G_color_right,
true );
        else
            color_inside=new GradientPaint( 0, 0, center.G_color_left, 12, 12, center.G_color_right,
true );
    }
    else if(choose==30)
//填充图案
        FileDialog fileDialog = new FileDialog( new Frame() , "选择一个图档", FileDialog.LOAD );//
        利用 FileDialog 抓取文件名
        fileDialog.show();//显示视图窗

```