内附光盘

Java 阶梯丛书

Java

实用系统开发指南

■ 彭晨阳 编著

本书精选了8个大型的综合实用系统，展示了伸缩的、可扩展的和可重用系统的开发过程，力求启发读者编程思想，培养编程感觉。同时，随书光盘中附带了实例的源代码。

机械工业出版社
China Machine Press

Java 阶梯丛书

Java 实用系统开发指南

彭晨阳　编著

机 械 工 业 出 版 社

本书利用 8 个综合实例介绍了多个 Java 系统的设计和开发，重点描述了 J2EE 实用系统的架构设计和应用。本书的特点是从可复用的面向对象设计的高度总结出解决同类问题的通用规律，探讨了设计模式和框架在可重用性、可扩展性和可伸缩性等方面的实际应用，从而提高软件的开发质量和速度，帮助开发人员解决 Java 实战中常见的主要问题。

本书适合大学计算机专业的学生、研究生、软件设计师以及软件开发人员学习参考。

图书在版编目（CIP）数据

Java 实用系统开发指南/彭晨阳编著.

-北京：机械工业出版社，2004.1

（Java 阶梯丛书）

ISBN 7-111-13535-0

Ⅰ.J… Ⅱ.彭… Ⅲ.Java 语言-程序设计

Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 110814 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：吴宏伟 责任编辑：王金航 版式设计：谭奕丽

北京中加印刷有限公司印刷·新华书店北京发行所发行

2004 年 4 月第 1 版第 1 次印刷

787mm×1092mm 1/16·26 印张·608 千字

0001-5000 册

定价：42.00 元（含 1CD）

前　言

Java 是复杂的，Java 包含的东西太多，它已经不再是一个简单的语言，它代表着一种新的设计和编程体系。Java 虽然是由 Sun 公司发明，但是它的发展已经脱离了某个公司的完全控制，Java 是由整个 Java 社区推动发展，Java 和开源力量的结合为用户提供更先进、更简洁、更多重的选择。

Java 至少从下面几个方面提供了最新的理念实现。

首先是多层结构，Java 在企业服务器端的主要应用体现在 J2EE 技术上，J2EE 是一种多层结构的框架技术。

传统的 C/S 系统结构只有两层，将业务逻辑要么紧紧封装在数据库端，要么耦合在客户端。这样带来的缺陷就是很多业务功能无法复用，维护和扩展起来困难，经常因为修改一个小问题而导致整个系统错误百出，所谓牵一而动百。这种脆弱结构显然无法满足不断变化的客户需求。

世界上再也没有一个技术像软件这样能够帮助人们快速应付变化的环境，因此，动态扩展性和可伸缩性对于软件异常重要。

为了提高软件系统的扩展性和可维护性，J2EE 将整个系统划分成多个层次，如负责界面显示的表现层，负责数据库操作的持久层以及各种框架技术组成的中间层，这样，整个系统就可谓泾渭分明，修改界面或数据表所造成的影响只在一个层面中，不会对其他层面形成影响和冲击。

多层结构同时带来了另外一个好处：最大化的可重用性。由于中间层脱离了具体界面联系，也和数据库具体操作实现了解耦，那么它的功能代码可以重用在多个项目开发中，大大节省了开发成本，提高了开发效率。

一个软件系统可以看成由两个部分组成：“新”和“旧”。所谓“新”就是针对具体应用的新设计和新代码；而所谓“旧”，就是重用了以前类似功能的软件组件或软件设计。

在这两个组成因素中，如果可重用的组件或设计越多，那么所需要的新设计和新代码就会越少，因此，可重用性对于软件设计和开发是极其重要的。

通常情况下，可重用的往往不仅是软件功能，更多的可能是软件设计，后者体现在设计模式和框架等概念上。

J2EE 多层结构还带来了项目开发和管理的革新。J2EE 从技术手段实施上保证了大型项目管理目标的真正实现。过去有不少大中型项目，由于没有可操作的实施手段，多人协作管理只能是一句空话，再有能力的项目经理也不可能做好。

J2EE 使得大型项目开发变得类似一个自动化工厂，J2EE 框架技术本身就类似那套自动化生产流水线，程序员只要做好流水线的某个环节工作就可以了，最后生产的产品通过 ejb-jar.xml 等文件配置组装后就可以运行了。有人觉得 J2EE 开发环节太多，配置太多，而这些恰恰是它为应付大型项目所做的准备。

Java 带来的新理念不只是上面这些，它最重要的贡献是改变了开发设计人员的系统分析和设计思维，分派和包装可能是这个新思维的核心。

坚持完全面向对象的 J2EE 对于大型系统的思维方式是一分再分，将大项目拆成小项目、组件等，再细分到小类，将逻辑或者数据都进行封装、分派。粒度越细，整个系统越加稳固，这就像一个大厦是由很多楼层（组件）组成，楼层又是由砖或水泥浇筑，而砖和水泥都是由沙粒组成，当把大厦细分到沙粒，无疑可以用这些沙粒组合建筑全世界各种大厦或房屋了。同样，当将一个大项目细分到一定粒度，系统的可伸缩性、可扩展性、可维护性、可重用性会大大增强。当然带来的缺点和盖房屋一样，不可能一个人来完成这些，需要有很多人协同完成（可能这不能算缺点吧）。

总之，Java 带给我们的是一个令人激动的新世界，因为其博大精深的丰富内容使得它的使用也变得复杂，目前，Java 世界最大的努力就是简单化，本书的编写和出版也应该说是这种努力的表现。

本书从实战角度介绍了多个 Java 系统的设计、开发全过程，从这些实例中，试图总结出解决类似问题的通用规律，提出了一些可重用的模式或框架，帮助开发人员解决 Java 实战中常见的问题。

以笔者的经验来说，学习 Java 可以先从 JSP 入手，如果有其他编程脚本经验如 PHP 或 ASP 则更佳。通过阅读第 2 章“简单的用户注册系统”可以帮助从纯粹的脚本编程过渡到 Jsp/JavaBeans 的开发模式。

当你逐步感觉到分层结构的优点后，也许需要在开发思维上进行一些面向对象化概念的培训，设计模式则是一道不可逾越的鸿沟，GoF 的 23 种设计模式你至少要知晓几种，第 3 章的“Jive 论坛系统”可以帮助你学会如何使用设计模式。

模式是一种有一定高度的抽象概念，但是要注意不要“走火入魔”，必要的时候，把自己往“下”拉一拉，深入到 Web 底层原理机制中研究一番。现在 Web Services、RMI、RPC 等概念很热门，Robert C. Martin 却说“我宁可使用 Socket”，第 1 章的“高性能聊天系统”是基于 Java 非堵塞 Socket 设计线程级实时系统，研究多线程等概念可以帮助你牢固掌握 Jsp/Servlet。

有了之前的“上下”折腾，总算可以练成一个 Web 开发熟手了，但是想成为 Web 开发高手，还需要再看看第 4 章的“网站内容管理系统”，这一章帮助你进一步深入掌握 J2EE 的 Web 技术，此章将从层的概念来设计和开发应用系统，同时帮助你学习应用 MVC 模式的开源产品 Strut。

第 5 章开始，将进入 J2EE 学习的精彩和核心部分 EJB。EJB 概念和原理非常复杂，但是使用比较简单，可以通过学习 EJB 设计模式，将 EJB 作为普通的 JavaBeans 操练实战，然后开始研究 EJB 性能提升和使用陷阱等问题，这样，可以逐步理解 EJB 各种复杂的概念和原理。

用户注册登录是开发一个新系统首先必须解决的功能，因此如何建立一个统一的用户注册登录系统是很多开发者关心的问题。第 6 章讨论了使用 J2EE 的 JAAS 实现用户注册登录的主要步骤。

针对自己的应用领域设计出一种框架，将复杂的机制隐藏于其中，从而在不丧失可拓

展性、可重用性的基础上进一步简化开发步骤，是一个两全其美的办法。第 7 章的“EJB 方法调用框架”介绍了这样一种框架的产生、设计和实现的全过程，EJB 方法调用框架不仅提供了在瘦客户端/服务器结构下 EJB 服务的调用方式，而且提供了远程肥客户端和 EJB 服务之间交互式调用的框架。

数据的增、删、改功能实现往往要经过表现层、中间层和数据持久层等多个阶段，若能在 EJB 和 Strut 体系下建立一种数据增、删、改、查的通用操作框架，这样在开发大型数据库信息系统时，就可以轻松完成数百个数据对象的开发工作。第 8 章的“网上商店系统”在这方面作出了一定的尝试。

本书的软件系统基本都是建立在开源软件基础上，J2EE 服务器采用的是 Tomcat 和 JBoss，在书中或多或少介绍了这些开源系统的主要使用方法和注意点，相信对于文档相对缺乏的开源软件使用会有一定的帮助。

当然，本书的后面还有一个强大的中文 Java 社区，如果想一直学习和关注 Java 的发展，欢迎访问 J 道网站 htpp://www.jdon.com。由于笔者水平有限，本书难免存在不妥之处，希望广大读者通过该网站和笔者联系，敬请批评指正。

本书的写作和完成是在身边很多朋友的帮助下才完成的，首先感谢吴宏伟编辑给予的鼓励和支持，同时也感谢远在美国的王万春博士的关心，感谢 J 道社区的范凯先生、韩卿先生、冯志强先生、张建军先生以及我的家人，也感谢我两岁的儿子彭莅，因为他很“自觉”，没有经常打扰爸爸的写作。

彭晨阳

目　录

目 录

第 1 章　高性能聊天系统

本章以建立一个聊天系统为例，介绍如何使用 J2SE1.4 非堵塞 I/O（NI/O）为核心开发一个高性能的实时互动服务器系统，这个高性能服务器系统可以拓展为更广阔的应用，如游戏系统、社区系统或者数据实时采集系统。

1.1　系统需求

聊天交流是目前互联网提供的主要内容。聊天系统有多种实现方式，类似 ICQ 属于一种点对点的聊天系统；还有一种是基于 Socket 的集中式聊天系统，这种聊天系统需要登录统一的聊天服务器，每个人的聊天信息其他人都可以看到，类似一种会议室，当然，两个人之间也可以进行保密的私语。

在基于 Socket 的聊天系统中，主要有两种角色：服务器和客户端，不同的客户端登录集中式的服务器，通过服务器将一个客户端发出的信息推送到其他所有客户端。

基于 Socket 的聊天系统最早实现是使用网页刷新方式，通过客户端不断地自动刷新，将服务器端整个页面内容下载到客户端显示，这种方式的聊天速度慢，而且有刷屏现象，很快被更新的聊天技术所替代。

聊天系统在客户端和服务器之间主要传送的是文字信息，服务器端只需要把最新的文字信息推送到客户端，这样减少了网络传输内容，节省了网络传输的时间，无疑提高了聊天速度。这种“推”技术是目前基于 Socket 聊天系统的主要实现技术。

一个基于 Socket 的聊天系统有下列具体功能要求：

（1）客户端和服务器必须保持随时随地的连接。这有别于普通 Web 浏览的连接方式。在使用浏览器访问服务器时，先由客户端发出 HTTP 协议，然后服务器响应处理这个客户端的响应，再返回处理结果；请求（Request）和响应（Response）是一种一对一的前后因果关系。

而在基于 Socket 的聊天系统中，客户端发出聊天信息的同时，客户端也在接受服务器发送过来的其他人的聊天信息，因此，请求和响应不存在那种前后对应关系，是两种分别独立进行的进程。

因为服务器任何时候都可能发送信息到客户端，因此，客户端和服务器一旦建立连接，必须能让服务器在以后发送中寻找定位到这个连接。

（2）在速度性能方面，聊天系统提出了更高的要求。在网络连接的薄弱环节 I/O 通信方面，要求能够实现无堵塞地、流畅地数据读写。在面对几百个甚至更多的客户端同时发出连接信息的情况下，服务器要求能够保持高性能的并发处理机制，迅速地完成这几百个并发请求的处理和发送任务。

（3）在扩展性和伸缩性方面，聊天系统也提出了一定的要求。当一台服务器不能满足

要求时，必须在客户端不知晓的情况下，通过不断增加服务器就能方便地拓展聊天系统的整体处理能力。对于客户端用户来说，这些服务器群都象征一个统一的服务器，不需要他们在进入聊天室之前先选择具体的服务器；也没有单个聊天室最大人数的限制，如果可以，服务器群可以支撑一个巨大容量的聊天室。

1.2 架构设计

本系统的设计核心是 Socket 底层通信，基于快速稳定的 Socket 底层通信架构，不但可以实现聊天系统，还可以实现其他如游戏、数据采集等实时性要求较高的系统，甚至可以建立一个快速的平台服务器系统。相比 J2EE 服务器系统，该平台系统的最大优势就是精简的代码带来的高性能。

当然，如果单纯追求高性能和速度，也许直接使用汇编就可以。使用 Java 设计这样的实时系统，实际还有一种很重要的目的，即追求高度的可扩展性和伸缩性。

因此，本系统设计必须将高性能和高伸缩性两个方面和谐地统一起来，不能盲目追求性能而破坏面向对象的编程风格和模式；也不能因为追求更大的重用性，建立太多复杂的中间层。其实这方面 J2EE 已经做得很好，有关 J2EE 的应用将在以后章节重点讨论。

当然，高性能应该是本系统的主要特色，为了实现本系统高效率的并发处理性能，设计上将采取 Reactor 模式实现自我快速触发；网络通信上，使用非堵塞 I/O 进行流畅的数据读写，在应用逻辑上，通过非堵塞的多线程技术实现并发功能处理。特别是 J2SE 1.4 以后版本推出的非堵塞 I/O，将使得 Java 表现出和 C 语言同样的优越性能。

1.2.1 Java 事件模型

事件只有在被触发时才会发生，它的发生是程序系统无法预料和计划的。例如，火警探测器只有在发生火警时才会触发，但是火警的发生是无法事先预料的，它处于时刻可能发生当中。为了能响应这些无法预料发生的事件，必须建立一套事件处理机制，以预备在发生事件时实现相应的处理。

通常，在一个事件处理框架里有下列 3 种角色。

- 源目标：事件发生者。
- 监视者：将监察侦听事件的发生，事件发生后，它会被通知到。
- 处理者：事件发生后，它将实现一定的行为动作，也就是处理事件。

在 Java 中有几种模式表达这 3 种角色之间的关系。

监视模式是一种最常用的模式，GOF《设计模式》中的观察者模式和监视模式类似，这将在以后章节讨论。

在监视模式中，监视者本身也兼任处理者的角色，3 种角色被实现为两个独立对象，源目标为一个对象，而监视者和处理者两个角色同位于一个对象中。

例如，一个可视化 JavaBean 对象 Button 属于源目标，它通过 addActionListener 绑定一个监视者对象 java.awt.event.ActionListener 的子类来完成事件触发机制，当它被用户点按

时，ActionListener 的子类将完成相应点按事件的处理，如图 1-1 所示。

图 1-1　Java GUI 中的事件处理

以本系统为例，当用户输入服务器端地址和端口，单击连接按钮后，将启动连接远程服务器线程，其中部分重要方法如下：

```
connectButton.addActionListener(new ClientFrame_connectButton_actionAdapter(this));
void connectButton_actionPerformed(ActionEvent e) {
        //启动连接服务器线程
        NonBlockingSocket nonBlockingSocket = new NonBlockingSocket(url, port);
        nonBlockingSocket.setDaemon(true);
        nonBlockingSocket.start();

}
```

连接按钮 connectButton 通过 addActionListener 方法加入了一个监视者对象，监视者 ClientFrame_connectButton_actionAdapter 代码如下：

```
class ClientFrame_connectButton_actionAdapter
implements java.awt.event.ActionListener {
  ClientFrame adaptee;
  ClientFrame_connectButton_actionAdapter(ClientFrame adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.connectButton_actionPerformed(e);
  }
}
```

上述代码中，通过适配器模式将事件处理委托源目标实现，这就产生了第二种事件处理模式——委托模式。

委托模式使源目标、监视者和处理者 3 种角色各自实现为独立的 3 个对象，在这 3 个对象之间传输的是事件，这个模式在系统复杂时会经常使用。例如在 J2EE 的 B/S 架构中，前台 JSP 的表单提交事件后，由 MVC 模式中的 Servlet 获得事件数据，经过简单封装成事件对象后，委托给后台 EJB 层实现进一步处理，如图 1-2 所示。

图 1-2 中，客户端浏览器将表单事件直接提交到服务器端的 JSP/Servlets，JSP/Servlets 作为事件的监视者，接收到事件后，并不对事件立即进行处理，而是委托给 JavaBeans/EJB 进行复杂的逻辑或运算处理。

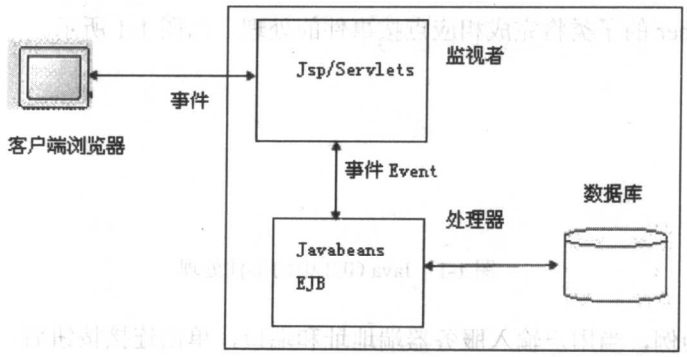

图 1-2　B/S 多层结构委托模式

以上两种模式的特点是至少有两个对象分别代表事件的3个角色，而在Reactor模式中，则是在一个对象中绑定了这 3 种角色，Reactor 的意思是自我触发、自主激活的意思。

在 J2SE 1.4 版本中的新特性——非堵塞 I/O（Nonblocking I/O）提供了基于 Reactor 模式的实现，这大大简化了基于 Socket 的应用和编程，如图 1-3 所示。

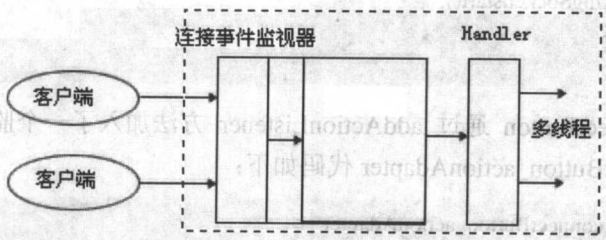

图 1-3　Reactor 模式

在非堵塞 I/O API 中，监视器是其一个重要的类 Selector，被监视的源目标是可以被 Selector 联系的 SelectableChannel（基本也属于 Selector 的相关部分），事件类型有：是否有接受的连接（OP_ACCEPT）、是否可以连接（OP_CONNECT）、是否可以读取（OP_READ）和是否可以写入（OP_WRITE）。

监视器 Selector 主要是监视这些事件，一旦发生，生成 SelectionKey 对象，Selector 是自我触发、自我激活的，因此是典型的 Reactor 模式实现，其原理将在后面章节详细讨论。

但是，在非堵塞 I/O API 中，并不是由 Selector 来实现事件的处理，而是由 Selector 激活出来后，通过其他处理器 Handler 来实现处理。开发者使用非堵塞 I/O API 需要做的工作就是：获取 Selector 激发的事件，然后根据相应事件类型，编制自己的处理器代码来进行具体处理。例如，如果是可读取事件，那么编制代码从 SelectableChannel 读取数据包，然后处理这个数据包。

在非堵塞 I/O API 中，使用 Reactor 模式将事件发生和事件处理两个部分实现分离解耦，事件发生部分只负责事件的激活，而事件处理由专门的处理器实现具体处理。

1.2.2 架构设计图

考虑到系统的可重用性和伸缩性，需要将本系统的网络通信底层和应用系统分离开。这样，基于可重用的网络通信层，可以实现其他各种实时性较高的应用系统，同时，系统还需要提供一些基本功能支持，如网络连接状态管理以及用户状态相关管理，前者为实现一个动态的实时在线系统提供基本连接的管理，后者类似 J2EE 中 Servlet 部分的 Session 管理。

本系统在架构设计上将分 3 个层次，如图 1-4 所示。

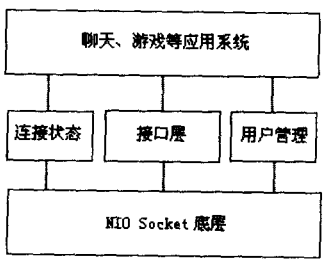

图 1-4 架构层次图

本系统最底层是 Socket 通信层，将负责客户端和服务器之间快速的数据交换，它通过接口层和最上面应用层实现解耦，同时又通过接口层和应用层保持实时数据联系。用户从客户端进入到本系统前，将实现统一的用户登录验证机制。有关用户登录验证机制有多种实现方式，可以参见后面章节的讨论。

用户成功进入系统以后，将会有一个生存周期，生存周期依据不同底层协议有不同的具体实现。不管哪一种实现方式，都必须在内存中保存用户连接的相关状态，如用户的 IP 地址、用户最新连接时间等。

为了保证系统的安全性，用户在登录验证通过后将分配一个随机的 SessionID，用户的每次请求都将包含这个 SessionID，服务器每次接受请求后，将此 SessionID 和保存在内存中的数据实现核对。

这里将着重讨论 Socket 底层以及接口层的设计和实现。Socket 底层设计分两大部分：协议设计和连接处理设计；接口层的目的是提供底层和应用层一个中介媒体作用，但是不能设计得太复杂，以免延误数据传送时间。

1.2.3 协议设计

TCP 是一种面向连接的协议，传输数据比较可靠。TCP 协议中包含了专门的传递保证机制：当接收方收到发送方传来的信息时，会自动向发送方发出确认消息；发送方只有在接收到该确认消息之后才正式传送数据信息，否则将一直等待直到收到确认信息为止。

TCP 在正式收发数据前，首先必须建立可靠的连接。一个 TCP 连接需要经过 3 次对话才能建立起来，其中的过程非常复杂。

基于 TCP 有各种会话应用协议，如 HTTP、FTP 等协议。其中，HTTP 协议是 Internet 最常用的协议，其最大的特点是能够穿透各种防火墙，因此，传送数据包以 HTTP 协议传送是一个实用的选择。