HZ BOOKS

PROGRAMMER TO PROGRAMMER™

Professional Java Server Programming
J2EE Edition

Wrox程序员参考系列

J2EE服务器端高级编程

(美) Subrahmanyam Allamaraju 等著

闻道工作室 译

机械工业出版社
China Machine Press

乐思 wrox PROGRAMMER TO PROGRAMMER

p2p.wrox.com 提供在线支持

153

Wrox程序员参考系列

J2EE服务器端高级编程

（美） Subrahmanyam Allamaraju 等著

闻道工作室 译

机械工业出版社
China Machine Press

本书主要讲解了开发者如何使用J2EE体系结构为各事务组件编写关键逻辑。通过对Java servlet、JavaServer页面及EJB技术的理解，本书示例了如何设计和构建安全且可扩展的Java n层应用。内容包括：如何使用JSP页面分割事务和表示逻辑；如何使用EJB创建可扩展的应用；及如何使用CORBA、XML和Java Message Service与其他系统集成。

本书需要读者对Java编程语言的知识及分布式开发基础比较熟悉，主要面向用J2EE开发分布式企业级应用的专业Java开发者。

本书版权登记号：图字：01-2000-4097

图书在版编目（CIP）数据

J2EE服务器端高级编程 /（美）阿拉玛拉尤（Allamaraju, S.）等著；闻道工作室译. -北京：机械工业出版社，2001.9
（Wrox程序员参考系列）
书名原文：Professional Java Server Programming J2EE Edition
ISBN 7-111-08944-8

Ⅰ.J… Ⅱ.①阿…②闻… Ⅲ.JAVA语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2001）第026524号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：瞿静华 温丹丹
北京忠信诚胶印厂印刷 · 新华书店北京发行所发行
2001年9月第1版第1次印刷
787mm × 1092mm 1/16 · 85印张
印数：0 001-5 000册
定价：128.00元

前　言

欢迎阅读本书。对于那些阅读过本书第一版的读者来说，他们会发现许多内容都有了变化，而本书距第一版出版时间仅仅一年。

J2EE版

在Java服务器端技术发展中，最显著的变化也许就是Java 2平台企业版的发布（Java 2 Platform，Enterprise Edition，J2EE）。

J2EE代表了一系列的发展过程，在这个过程中，Sun公司力图使Java不仅仅是作为一种有生命力的语言来发展，更重要的是，要作为一种有生命力的企业开发平台来发展。通过本书，可以看到，J2EE并不是一个简单的API的集合，而是定义了一个运行时的体系结构，并用在开发中。这种体系结构对于企业级Java应用开发者而言，它的美妙之处在于，J2EE的主要目标是：通过把在企业级开发中最基本的而又显得零散的、低层次的语义（如连接池和各种事务）封装到一个基于包容器的体系结构中。从而，开发者要去做的工作就是继续编写业务逻辑部分。

因此，这一版不是对第一版的简单更新，而是使得其中的内容完全可以反映出目前有关Java服务器端开发技术的发展变化的状况。

本书面向的读者

本书是面向专业级的Java程序员，读者不必具备丰富的实践经验，但至少要熟悉网络和Web程序设计的基本概念。另外，也应当熟悉Java语言和核心API——关于这方面的内容请阅读Ivor Horton的《Beginning Java 2》，或者其他相似内容的初级书籍。但无论如何，本书中所有与Java服务器端程序设计有关的概念都会被阐述，并不需要预备知识。

另外要说的是：读者最好熟悉一些基本的Java服务器端技术。因为，本书包含了许多这方面的内容，不能保证其中所涉及的所有的概念都能被完全阐述。

本书包含了什么

在本书中，讨论了以下三个方面的内容：

- 开发者在编写企业级组件时要遵循的技术规范中的规则。
- 在现实世界中，典型厂商在实现J2EE时所做的补充和限制。
- 通过使用J2EE技术，现实中的设计所带来的实践结果。

本书的基本结构如下：

- 本书从对Java企业级开发者的最新的要求和Java（具体说是J2EE）是如何发展来满足这些方面的挑战开始的。从中可以了解到J2EE包容器体系结构的初始概貌。
- 在讨论了J2EE体系结构后，阐述了一些在企业级开发中采用的基本技术：RMI、JDBC、

JNDI和XML。

- 然后，通过使用Java Servlet开发Web组件来更具体地讲述J2EE。
- 理解掌握了Servlet技术后，就可以学习JSP（JavaServer Page）是如何应用该技术、进而能够提供更为灵活的创建动态Web内容的方式。
- 接下来，就是通过讲述有关EJB的复杂组件技术来阐述企业级开发。
- 讨论了在J2EE中的三种组件技术之后，本书继续阐述在企业领域开发中要涉及的一些问题，如安全性、性能和可扩展性。
- 最后，讨论了如何把Java应用与其他非Java技术，如COM和CORBA集成起来，如何运用Java消息服务（Java Message Service）来进行基于消息的集成。

在本书的附录部分，包含了一些使用的内容，其中有介绍如何配置和安装所需要用的软件，另外，还涉及到了软件国际化等一些主题。

本书中绝大部分代码在Java 2平台、标准版SDK(JDK 1.3)和Java 2 Platform、Enterprise Edition SDK1.2.1 Reference Zmplementation环境中经过测试。但是，在本书许多章节，这两个引用环境还不够，需要其他的软件，如：

Web包容器

为了运行本书中的Web组件，需要一个支持Servlet 2.2和JSP 1.1规范的Web包容器，可以采用：

- 包含Tomcat3.0的J2EE RI，书中用到了Tomcat 3.1中的一些特征而在3.0中不支持。http://www.apache.org/。
- Allaire的JRUN3.0，http://commerce.allaire.com/download/。

参照附录A中的安装和配置指南。

EJB包容器

在EJB章节里面，需要一个支持EJB 1.1规范的EJB包容器，采用：

- jBoss 2.0 Application Server，http://www.jboss.org。
- Orion Application Server，http://www.orionserver.com。

本书中还有一章涉及到了EJB 2.0规范。但是，该规范还处在公开的起草阶段，还没有被广泛实现。少数的应用服务厂商，如BEA，已经在他们的应用服务器上实现了一个支持目前草案规范的beta版patch，但请注意，在规范最终形成前，什么事情都有可能改变。

数据库

有几部分章节需要访问数据库。在这些章节中，用到了下面一些数据库：

- Cloudscape（随J2EE RI带有一个进程内版），http://www.cloudscape.com。
- Microsoft Access 2000。
- Microsoft SQL Server。
- HypersonicSQL，http://hsql.oron.ch/。

其他的软件

最后，一些章节还需要的几个软件有：

- Sun公司的JNDI SDK，在JDK1.3中包含有。
- XML分析器和XSLT引擎——James Clark的xp和xt，http://www.jclark.com/xml。
- 用于Java API for XML（JAXP），http://java.sun.com/xml。
- IBM的XML分析器和Lotus的XSL处理器，http://alphaworks.ibm.com。
- Xalan的XSLT处理器和Xerces的XML Parser，http://xml.apache.org。
- Java Secure Sockets Extension(JSSE)，1.0.1，http://java.sun.com/products/jsse/。
- Allaire Tag Library，http://commerce.allaire.com/download/。
- LDAP服务器——Netscape的iPlanet Directory Server version 4.11，http://www.iplanet.com。
- 消息服务器——SonicMQ，http://www.sonicmq.com。
- 单元测试框架——JUnit，http://www.junit.org。
- SMTP和/或POP3服务。

本书中的代码可以在单机上运行，要具备网络环境（也就是，可以通过本地浏览器访问http://localhost）。

书中的所有源代码可以从下面的站点下载：

http://www.wrox.com

英文原书ISBN：1-861004-65-6

第1章 Java和J2EE平台

Java作为一种程序设计语言仍在起步之中。但是，Java从作为一种开发在浏览器中运行的applet（Java小程序）的方式到作为一种来驱动目前的企业级应用的程序设计模型过程是令人瞩目的。Java从产生到现在仅有几年的时间，但是在技术和商业应用领域中有着很大的吸引力，目前已经有了广泛的应用。

从一开始，Java就在不同的领域中——设备、电话应用、企业应用，引发了新型的程序设计模型和技术。同时，Java在使某种技术领域更为健壮和更为安全方面扮演着一种催化剂的角色。Java企业级计算平台，Java 2平台企业版（Java 2 Platform, Enterprise Edition, J2EE）就是这样的一种技术领域。

在Java的发展过程中，出现了很多争议，主要是在媒体和技术领域中Java是一种程序设计语言还是作为一种平台。但不管争论结果如何，J2EE是Sun和其协作伙伴的最成功的尝试之一，在企业级计算方面，J2EE使得Java完全可以胜任开发工具平台这一角色。

那么什么是J2EE呢？相关的东西是什么呢？为什么应当采用这种技术来构建企业级应用——从客户/服务器模式到Internet到移动计算模式呢？在本章，将给出J2EE的概貌，并将回答这些问题。希望本书的其余部分能够对成功创建和管理这些应用有所帮助。

本章的介绍主要集中在：

- J2EE技术的体系结构。
- 什么使得J2EE作为一种平台是可信的。
- J2EE带来的挑战。
- 哪些技术构成了J2EE平台。

首先，从当前企业级应用开发所面临的挑战开始。

1.1 企业的今天

随着Internet的不断发展，许多企业认识到整个新的市场是开放的，有待开发。但是，是否许多企业意识到Internet对经济上本质的影响。通过Internet和电子商务的近期发展来看，公司的信息资本越来越有价值。这种向信息经济的转移趋势迫使许多企业不得不重新考虑他们最基本的商业实践活动。为了能够在竞争中处于有利的地位，采用新技术就成为企业能否有效地利用各自的信息资源的关键因素。更为重要的是，新技术的运用要与现有的、以前遗留的系统有机结合成为企业最重要的需求。

在商业实践的转移中，最迫切的一点在应用开发这一层次。用于应用开发的物力和时间正在减少，而对其复杂性的要求却增加了。虽然对于开发者来说对信息的重视不是什么大问题，但由于迅速变化的技术和经济形势所带来的改变，对于目前的企业级应用开发者来说，却要面临新的挑战。

- 反应性 虽然时间线总是很重要，但高速的、迅速变化的信息驱动经济发展意味着企业要对新方向和信息做出快速的反应，这对于企业要在竞争中处于有利的地位来说是很重要的。
- 程序设计的生产率 直接采用新技术还不够，除非正确利用新技术，充分发挥其潜力，尽快有效地与其他相关技术集成。因此，尽可能快速、有效地开发和部署应用程序的能力也是很重要的。随着时间的推移，技术和标准发展变化使得企业要做到这一点并不容易，需要具备较高的开发技巧和经验，而要获得和保持这些本身就是一个问题。更进一步讲，要保证能够有效融合各种技术，加快改变标准的步伐就会带来很大的挑战。
- 可靠性和可用性 在今天，离开Internet，商业就很难获得成功。基于Web应用的操作和运转，并保持其正常运行的能力对于企业的成功来说是至关重要的。但仅仅做到这一点似乎还不够，还必须具备一种能力来保证商业事务的可靠性，从而使得各种商业事务能够被完全地、正确地处理。
- 安全性 不仅Internet的潜在用户呈指数增长，而且其中企业信息的价值也在增长。从而，信息安全就成为一个主要关心的话题。在技术更为先进、应用更为复杂、企业更为复杂的同时，还存在其他的问题，从而使得要实现一个有效的安全模型变得越来越困难。
- 可扩展性 在Internet上，一个应用的潜在用户群可能有上百万个个体，在这种情况下，能够扩充应用来满足可操作性和用户群方面的需求的能力很重要。能够有效地增加规模不仅需要考虑日益增长的大量的客户，而且也要考虑能够更加有效地利用系统的资源。
- 集成化 信息是商业资产中的关键部分，但多数的信息作为数据存在于老的或者已经过时的系统中。为了最大限度地利用这些信息，新应用必须能够与目前存在的系统相互集成——不要认为这很容易办到，因为，当前的技术与遗留系统所采用的技术相比，其先进性使得两者相差太远。因此，企业要成功发展，把新旧技术结合起来的能力也是很关键的。

对于企业级开发者来说，上面提到的问题域并不陌生。但是要用一种广泛的、经济的方式来解决这些问题仍然显得很棘手。也许读者知道已有的技术强调了上述需求的一个或几个方面。但无论如何，所缺乏的是一种广泛采用的充满丰富的基础构架（infrastructure）和大量结构的平台，该平台本身就提供了一种快速应用开发环境。

1.1.1 系统结构

每当谈到企业级应用开发的时候，都会提到n层体系结构。典型的客户/服务器系统是基于2层结构的，其中在数据层和表示层/业务逻辑层之间有着清晰的界线。这类应用一般都是数据驱动的，应用在客户机上，并且在企业中会配置一个数据库服务器。虽然在这种结构下，企业内部可以共享数据，但是，它有许多缺点。

1.2层体系结构

在传统的2级结构中（如图1-1），处理负载都位于客户PC机上，而服务器仅充当了应用程序和数据之间的传输控制器。结果，不仅应用程序的性能受限于PC机资源，而且网络流量也增加了。当整个应用程序都在PC机上运行时，在把结果呈现给用户之前，应用程序不得不多次访问数据，这种多次访问数据库的方式浪费了网络带宽。

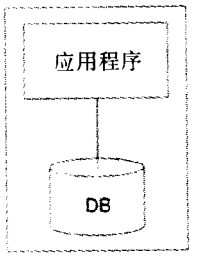

图 1-1

2层体系结构的另外一个问题是可维护性。哪怕是对应用程序的一个小小的修改都会涉及到把整个应用重新分发给用户。即使这个过程是自动化的，仍然要在每一个客户机上更新应用的问题。另外，一些用户可能还来不及更新整个应用程序，而其他一些用户已经及时作了更新，就会造成不同的客户使用的应用程序版本不一致。

2. 3层体系结构

为了解决这些问题，软件界提出了一种3层体系结构的概念（如图1-2）。应用层被划分为几个相互隔离的逻辑层。每一逻辑层都定义好一套接口集。第1层，也就是表示层，主要由类似于图形用户界面的部分组成；中间层，即业务层，由应用逻辑和业务逻辑构成；而第3层——数据层——包括了应用程序所需要的数据。

中间层（应用逻辑）代码由用户调用（通过表示层）来获取所需要的数据。表示层接收数据并且按照适当的格式显示出来。从用户界面中分离出应用逻辑，极大地增强了应用程序设计的灵活性。在应用逻辑对表示层提供了一套定义清晰的接口的情况下，甚至不用改变应用逻辑部分，就可以建立和分发多种用户界面。

第3层包含了应用程序所需要的数据。这些数据可以由不同的信息源组成。例如Oracle和Sybase那样的数据库，也可以是XML文档（按照XML规范存放的数据）集，还可以是如LDAP服务器的目录服务。除了传统的关系数据库存储机制外，还有许多种应用程序可以访问的不同的企业数据源。

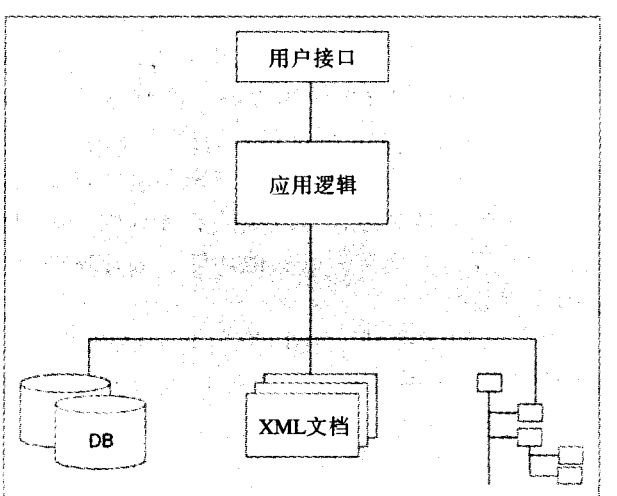

图 1-2

然而，对应用的层次划分还没有终止，最终的目的是创建n层体系结构的应用。

3. n层体系结构

在n层系统中（如图1-3），要较快地定义各个应用层并不容易。实际上，一个n层结构系统能够支持多种不同的配置，在n层结构中，应用逻辑的划分是根据功能而不是根据物理方面来进行的。

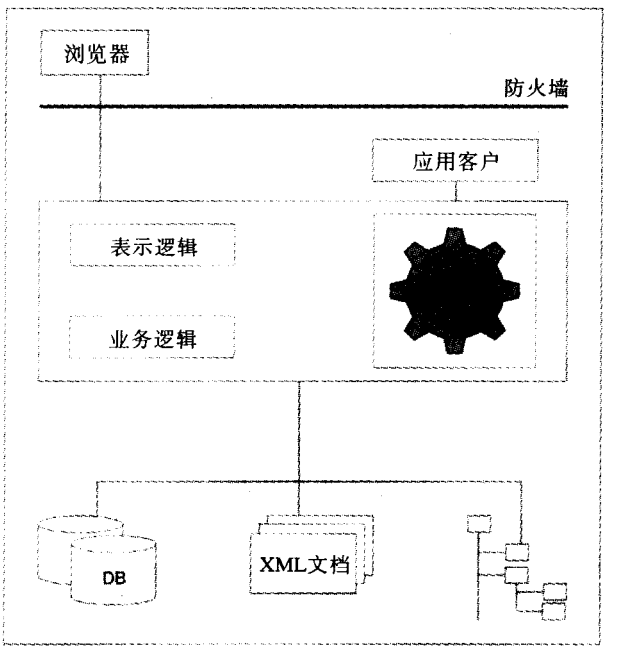

图 1-3

n层结构的划分方式如下：

- 用户接口层，负责处理用户与应用程序之间的交互过程；它可以是一个通过防火墙运行的Web浏览器，也可以是一般的桌面应用程序，甚至还可以是无线设备。
- 表示逻辑层，定义了用户界面要显示的内容和如何处理用户的请求——根据所支持的是什么样的用户接口。对于相应的客户，表示逻辑层版本会有相应的不同。
- 业务逻辑层，通过与应用数据打交道，对应用的业务规则实施建模。
- 基础框架服务层，提供了应用系统需要的其他功能，如传输消息、事务支持等等。
- 数据层，存放企业的应用数据。

基于这种结构的应用程序采用了模型-视图-控制器（Model-View-Controller，MVC）模式。采用这种模式的目的是要把数据（模型）与如何表示信息的部分（视图）分离出来。它们两者之间是由应用/业务逻辑（控制器）来控制信息的流向。通过这种模式，就能设计出基于三种相互交互的功能组件（模型、视图、控制器）的应用程序。

4. 企业结构

到目前为止，所有的讨论只集中在单个应用程序的结构上，而实际上，把这些应用程序

看作“烟囱”（独立而封闭的）是不行的。也就是说，不能仅仅把这些应用看成是不同的——甚至可能是不同体系结构的——它们之间不相互通信。在企业中，应当寻找一种内在统一的、整体的方式来对待这些应用。

不需要在结构上作变动——企业级结构本质上是n层的——需要在对应用的认识上作一些变化。要把n层结构应用到企业级系统中，只需要简单地考虑用多个应用对象而不是单个简单应用来扩展中间层。这些应用对象都必须有各自的接口，从而允许它们之间相互协调工作。

接口就像契约。每个对象通过接口来接受一定的参数，并返回相应的结果集。应用对象通过使用接口来进行相互之间的通信（如图1-4）。

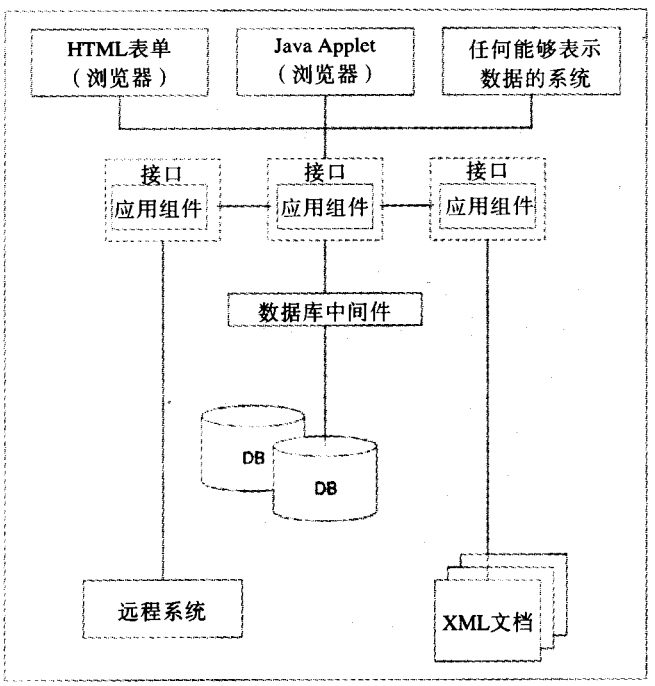

图 1-4

就企业级结构来说，在各个部门之间采用一组通用的组件就可以实现多种应用。从而，通过创建一套具有业务功能的模块集合使整个企业访问，就可以使各种商业实践标准化。如果商业规则发生了变化，那么只需要对商业对象做修改就行了；若有必要的话，也可以修改接口，那就需要修改访问该接口的任何对象。

要注意的重要的一点是：当在设计应用对象及其接口时，最好尽可能使得接口通用化，以免以后修改。因为其他对象通过应用对象的接口而不是其本身进行通信的，所以修改对象本身与修改接口相比，前者要相对简单和快捷得多。

1.1.2 答案是Java吗

到现在为止，我们还未从实现的角度来讨论系统体系结构。而实际上，目前存在许多技

术途径用来实现企业级应用。Microsoft有Windows DNA（和新的.net结构），Sun公司有J2EE，Oracle公司有Oracle8i Internet Platform等等。在众多的服务器端开发途径中做出选择，那么Java的巨大吸引力在于什么地方。

1. 平台独立性

考虑到企业的信息分布在不同的平台和应用中，在程序设计语言方面就需要做出权衡。所选择的语言能够在企业中方便地应用，而不显得笨拙，事务处理也要高效。当集成各种不同的技术进而形成具体的某种平台和应用时，采用统一的程序设计模型就能够减少许多困难。

2. 可重用性

代码重用是所有程序设计所追求的目的之一。要达到重用目的，方法之一就是把应用的商业需求划分为不同的组件；而采用面向对象方式封装公用功能是另外一种方法。Java采用了这两种方法。Java是一种面向对象的程序设计语言，提供了重用机制。但是，不像对象那样，创建和管理组件需要一种复杂的基础框架机制。基本的面向对象概念并未提供这样的机制；然而，Java的Enterprise Edition提供了一种相当强大的机制来重用组件。

3. 模块化

当开发一个完整的服务器端应用程序时，程序会迅速变大、复杂化。应该是把应用划分为相对独立的模块，每个模块负责处理一个具体的任务。按照这种方式，理解和维护应用程序就会变得容易。模块化应用的手段有Java Servlet（小服务程序）、Java Server Page和EnterPrise JavaBeans（EJB）——它们可以把应用程序划分为不同的层和任务。

尽管Java具有上述的有利的特点，但是，直到2000年初，Java才提供了一种在企业级规模下设计应用的统一的程序设计模型——不是Java本身缺乏这种能力，而是没有把Java技术有效的组织起来。Sun公司认识到这个不足并针对它而发布了Java 2 Platform，EnterPrise Edition（J2EE）。

J2EE平台的设计思想是，对按照基于组件的应用模型设计的分布式应用提供一个简单而统一的标准。

在本章的剩余部分，严格来说是本书的其余部分，将讨论J2EE是如何采用Java来进行服务器端应用开发的。

Java对于成千上万的开发者来说，是一项在实际开发的每个方面都能带来巨大潜力的非凡的技术。这项新技术同时也带来了新的不可避免的问题。一些学者认为Java是一项完美的技术，或认为Java对任何应用来说都是适用的技术，这些都是不恰当的。开发者无论是老手还是新手，在知道Java的优点的同时，也要对Java带来的负面影响有清醒的认识。Java作为一种语言还处于发展成熟过程中，在许多方面，这项技术仍旧显得年轻。为了解决早期的不足，有关的规范正在较快地发展变化，与此同时，也会带来（暴露）新的不足。那么，就可以不采用Java了吗？当然不是，相反，应该为对Java的需要和成功的现状感到庆贺；同时要提醒读者的是，Java技术在带来方便的同时，也有危险。在实现Java不成熟的技术的同时也会带来显而易见的危险，但却有着潜在的巨大的回报。

1.2 J2EE平台

正如本书其余部分讨论的那样，为满足当前企业级应用开发的需要，J2EE是所提出的最好的途径之一。J2EE定义了管理应用程序的基础框架结构，也定义了创建应用要使用的服务API。

J2EE平台本质上是一个分布式的服务器应用程序设计环境——一个Java环境，它提供了：

- 宿主应用的一个运行基础框架环境。
- 一套用来创建应用的Java扩展API。

用上述环境建立的应用可以是开发Web页面的应用，或者是实现复杂数据库事务处理的组件，或者甚至可以是Java applet——所有的这些应用均通过网络来进行发布。

1.2.1 J2EE运行环境

J2EE包装了许多API，这些API早就以一种形式或其他形式存在。而J2EE最重要的一个方面是对运行环境基础框架的抽象。要注意的是，J2EE规范并没有对运行环境应当/怎样创建做出定义，相反，它规定了应用程序的接口和角色，和应用程序要被执行的运行环境。结果就是在应用和运行环境基础框架之间划清了界线。这种划分使得运行环境抽象出绝大多数的基础框架服务，而这些服务在以前是由开发者试图亲自去创建的。从而，J2EE应用程序开发者的工作只集中在应用逻辑和相关服务上，而不用考虑与框架环境相关的服务。

有意思的是在快速开发环境中，如Internet，应用开发往往会导致种种生命力短的设计。而为生命力短的应用而进行的设计决策缺乏长远的考虑。

另一方面，当在考虑长远的设计时，就会发现处理好短期的需求并不是件容易的事。采用一些技术能够进行快速开发，而采用另外的一些技术创建应用时，能够很好地进行长远的考虑，如可重用性、维护代价等等。通常，这两方面的优势不可兼得；但利用J2EE，既能够足够灵活地创建应用，又具有上述两方面的优点。这是因为，在J2EE中可以创建应用的每一层，各层之间是松耦合的。每一层可以单独地进行修改扩展，来满足相关方面的改变需求。

除了定义了一套标准的API以外，J2EE结构还提供了一个通过运行环境访问这些API服务的统一的方式。在阐述J2EE技术的细节之前，先回顾一下传统的分布式计算。

在J2EE之前，在一般情况下，把分布计算看做是客户/服务器设计模式。编写服务器应用程序来实现各种接口，客户应用程序连接到这些接口来访问服务器，然后启动服务器和客户程序。虽然这个过程看起来简单，但实际上，依赖于所采用的技术，实现这个过程会遇到一些棘手的障碍。例如，考虑用CORBA对象请求代理来创建分布式应用程序。一般步骤是，首先，通过为每个对象创建接口，采用接口定义语言（Interface Definition Language，IDL）来创建服务器端CORBA对象；接下来的步骤是，在所选择的包含这个IDL的开发语言中进行编译产生占位程序（stub）和框架（skeleton），根据框架来实现对象，然后再编写客户程序、准备运行环境等等。在这里，stub是用来在客户端代表CORBA对象

的类，而skeleton是服务器端逻辑实现的地方。这个过程本身并不复杂，并且能够很好地自动化。

再关注服务器和客户程序。客户程序请求访问服务，如分布式事务、消息传送等等。要使用这些服务，需要在应用程序中加入大量的常规的代码。经常地，需要建立和配置不同的中间件解决途径，编写API调用具体厂商的API来访问相应的服务。除了对关系型数据库访问这样的服务以外，其他服务要么是专有的，要么是非标准的。结果就会导致应用程序越来越复杂，从而会消耗和浪费大量的时间去开发、管理、维护。

除了必须管理好这些不同的API以外，在客户端应用程序还有另外的一个重要的需求。在服务器端，资源是宝贵的。比如，不能够像在客户端应用程序创建对象那样在服务器端创建同样多的对象。其他需要特别考虑的服务器资源包括线程、数据库连接、安全性、事务等等。要定制一个基础框架来处理这些资源往往是一个挑战。在Internet中，这种任务根本就不可能完成。当开发周期只有3个月的情况下，你愿意为数据库访问去考虑创建连接池、对象缓冲或“优雅”的对象层吗？

由于这种服务器端需求对广大的应用来说是常常要求的，所以考虑采用一个内嵌相关解决途径的平台是更可取的。这样，就可以把基础框架级的内容分离出来，而更多地去关注应用本身的需求。J2EE运行环境，就解决了这方面的问题。J2EE服务器厂商按照一种标准化的、兼容的方式来实现上述的特点。

正如上面提到的那样，J2EE并没有对运行环境的结构和本质做出具体的规定。相反，它引入了一种称为包容器（Container）的概念。通过J2EE API，规定了应用程序和包容器之间的规范。在本书后面的内容，可以看到这些规范的更多的细节。

在深入讨论J2EE包容器细节之前，先简单看一下J2EE API。

1.2.2 J2EE API

分布式应用程序要访问一整套企业服务。典型的服务包括事务处理、数据库访问、消息收发等等。J2EE体系结构通过企业级服务API统一了访问这些服务的方式。但无论如何，在J2EE中，不是非要通过专有的或非标准的方式来访问这些服务，而是通过包容器来访问这些API。

一个典型的商用J2EE平台（或J2EE应用服务器）包含了一个或多个包容器，由它们来访问由J2EE定义的企业级API。

> 注意J2EE应用服务器不必实现这些服务；包容器只需要提供通过J2EE API来访问每个已经实现的服务的途径。例如，对Java Messaging Server API的调用，在J2EE实现中就可能是对面向商用的中间件解决途径的调用。而另外的实现就可能是在包容器中包括面向消息的中间件途径。

J2EE平台规范定义了一组Java标准扩展集，每个具体的J2EE平台必须支持这些扩展集：

- Java数据库连接（Java DataBase Connectivity，JDBC）2.0扩展　也叫JDBC 2.0可选包（JDBC 2.0 Optional Package）。这些API通过在获取连接、连接池、分布事务等方面提

供更好的操作方式来完善标准的JDBC 2.0。

- 在Internet对象需求中介协议上的远程方法调用（Remote Method Invocation Over the Internet Inter-ORB Protocol，RMI-IIOP）1.0 在IIOP上实现了通常的Java RMI API。这样就在RMI和CORBA应用程序之间建立起桥梁。
- 企业级JavaBean（Enterprise Java Beans，EJB）1.1 为多层分布式应用程序规定了一个组件框架（framework）。它既定义了编写服务器端组件的标准方式，也定义了服务器端宿主组件的良好的基础框架运行时刻。
- Java Servlets 2.2 Java Servlets API为建立动态的Web应用程序提供了面向对象的方式。
- Java Server Pages（JSP）1.1 这个扩展通过提供基于摸板的Web应用开发技术而进一步增强了J2EE应用。
- Java消息服务（Java Message Service，JMS）1.0 JMS为消息队列、消息发布和订阅面向消息类型的中间件的服务提供了API支持。
- Java命名和目录服务接口（Java Naming and Directory Interface，JNDI）1.2 JNDI API用来标准化访问目前已有的各种类型的命名和目录服务。这种API的设计是独立于任何的命名和目录服务的实现。J2EE也定义了JNDI服务提供接口（Service Provider Interface，SPI），用于命名和目录服务的实现。
- Java事务API 1.0 这些API是用来实现分布式事务处理的应用程序。
- Java Mail 1.1 为建立基于Java的邮件应用程序，这套API提供了平台独立和协议独立的框架机制。

在本章后面将进一步讨论这些API。

上述所有的API都只是规范，它们独立于实现。也就是说，应该能够以一种标准的方式来访问这些通过API所提供的服务，而不管这些服务是怎样实现的。

1.3 J2EE体系结构——包容器

正如前面部分讨论的那样，一个典型的商用J2EE平台包括了一个或多个包容器。那么什么是包容器呢？J2EE包容器是一个用来管理应用程序组件，提供访问J2EE API的运行环境。除了具有运行环境相关的特点外，J2EE并没有定义包容器的其他属性，从而为在包容器运行环境中实现不同特点属性的应用提供了强大而灵活的机制。

图1-5展示了J2EE的体系结构。

在这个结构里面有两个包容器：

- 一个Web包容器，其中包含宿主Java Servlet和JSP页面。
- 一个EJB包容器，其中包括各种企业级JavaBean组件。

除了这两个包容器外，J2EE也定义了另外的两个包容器——一个是applet包容器，用于运行applet；另外一个是客户应用程序包容器，用于运行标准的Java客户应用程序。在本书里面，所有的讨论只集中在Web和EJB两个包容器上。

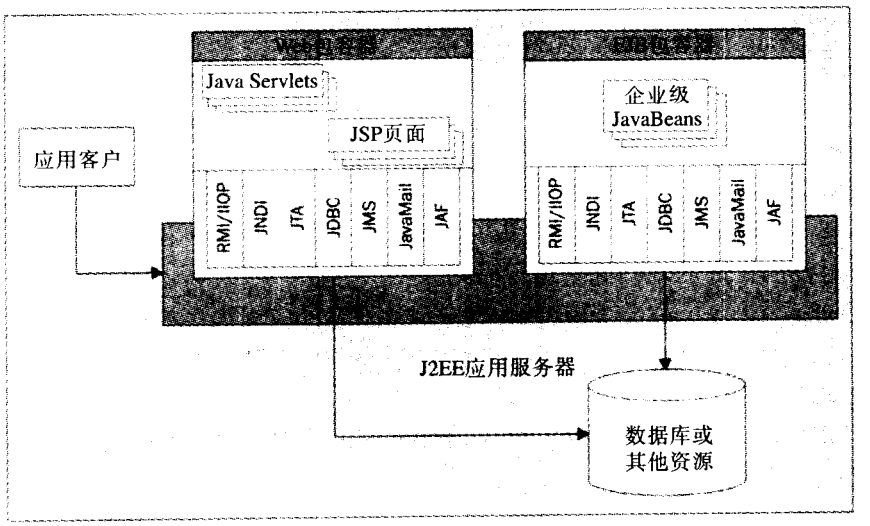

图　1-5

在图1-5中，在每个包容器底部的垂直块代表J2EE API。除了要被访问的基础框架级的API以外，每个包容器还定义了一些相关的具体于包容器的API（Web包容器的Java Servlet API，EJB包容器的EJB API）。

因此，可以说包容器是一个用于应用组件的Java 1.2（Java 2标准版1.2）运行环境。

图中重叠的矩形块（Servlets、JSP Pages和EJB）代表在包容器中开发和放置的程序。在J2EE技术中，把这类程序称为应用组件。

在这个结构中，主要有两种类型的客户。

- Web客户，通常在Web浏览器中运行。对于这类客户程序，用户界面是在服务器端作为HTML或XML来产生的，并被浏览器下载和显示。这种客户使用HTTP协议与Web包容器进行通信。在Web包容器中包括的应用组件有Java Servlet和JSP页面，这些组件实现了Web客户需要的功能。Web包容器负责从Web客户接收请求，并借助于应用组件来产生应答。
- EJB客户，访问EJB包容器中EJB应用组件的应用程序。EJB客户的类型可能有两类。第一类是应用客户，它们是使用RMI-IIOP协议访问EJB组件的单独的应用程序。第二类客户是Web包容器中的组件。也就是说，Java Servlet和JSP页面也可以像应用客户那样，采用RMI-IIOP协议来访问EJB组件。

在两种情况下，客户都是通过相应的包容器访问应用组件。Web客户通过Web包容器访问Java Servlet和JSP页面，而EJB客户通过EJB包容器访问EJB组件。

包容器体系结构

为了更好地理解J2EE的核心内容，先回顾一下前面有关体系结构的讨论，然后讨论J2EE包容器体系结构（如图1-6）。

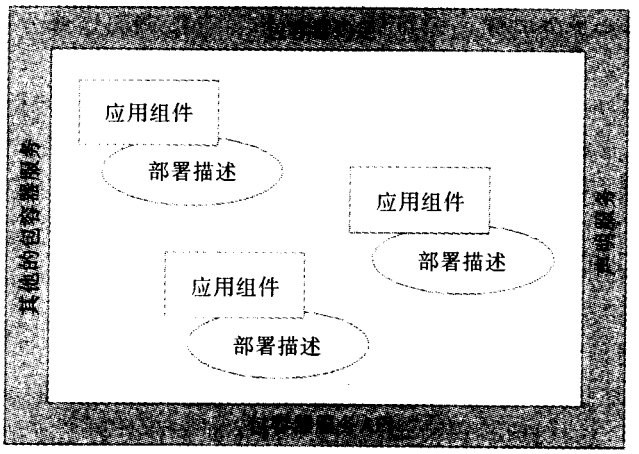

图 1-6

在这个结构中，对于开发者来说，要提供如下的内容：

- 应用组件 正如前面部分讨论的那样，应用组件包括Servlet、JSP和EJB等等。在J2EE中，也可以把应用组件打包形成档案文件。
- 部署 一个部署就是描述应用组件的XML文件，其中也包括了包容器需求的其他信息，包容器用来有效地管理应用组件。

图中其余部分形成了包容器。包容器体系结构可以分为四个部分：

- 组件约定 包容器定义的一组API，由应用程序来进行实现或扩展。
- 包容器服务API 包容器提供的另外的服务，这些服务对于包容器中所有的应用来说是共有的。
- 声明的服务 包容器施加于客户程序的服务，它基于提供给每个应用组件的部署。
- 其他的包容器服务 其他的运行时服务于与组件的生命周期、资源存放、垃圾回收等有关。

下面讨论每一个方面的细节。

1. 组件约定

正如前面提到的那样，J2EE体系结构中的包容器的基本任务就是为应用组件提供一个运行环境。也就是说，在包容器的JVM中创建和调用应用组件的实例。从而包容器就负责管理应用组件的生命周期。但无论如何，为了使得应用组件能够在包容器运行环境被管理，应用组件必须遵守由包容器定义的某种约定。

为了更好地理解这一点，以Java applet为例。通常，浏览器下载Java applet并且在浏览器JVM中进行实例化和初始化。也就是说，applet存在于浏览器JVM提供的运行环境。

但是，为了让包容器能够创建和初始化应用组件，并且调用它们的方法，要求应用组件实现或扩展某些Java接口或类。例如，再以applet为例，要求Java applet扩展由JDK定义的java.applet.Applet类，浏览器JVM假定applet扩展了该类，从而浏览器的JVM可以调用applet的init()、start()、stop()和destroy()方法。这些方法控制着applet的生命周期——如果applet没