第 3 章　商务逻辑层及其技术

在多层的电子商务系统的体系结构中，商务逻辑层处于核心的地位，电子商务中的大部分功能都是在这里实现的。通过使用现成的中间件软件包，借助组件调用技术或 Web Service 技术就可搭建一个具备绝大部分功能的系统。本章对商务逻辑层的实现技术进行全面的阐述，重点说明组件、组件调用、Web Service 等技术与方法。

3.1　商务逻辑层的构成

商务逻辑层可以分为两大部分：一部分是构成商务应用的核心商务逻辑，与具体的企业应用密切相关；另一部分是支持核心商务逻辑运行的软硬件环境。这些软硬件环境主要包括以下平台。

（1）商务服务平台，如客户关系管理、供应链管理、电子市场、网络社区、支付网关接口、认证中心接口等。

（2）商务支持平台，如内容管理、目录管理、搜索引擎等。

（3）基础支持平台，如 VB、C++ 、Java 等之类的应用开发环境与开发工具，负载均衡与错误恢复等高性能与高可靠性的系统环境，主机管理、网络管理、安全管理等一些系统管理工具，JDBC、ODBC、EJB、XML 等一些对象组件与服务集成环境。

（4）Web 服务器平台、数据库平台、操作系统、计算机硬件与网络基础设施，如表 3-1 所示。

表 3-1　商务逻辑层的构成

核心商务逻辑应用（企业宣传、网上销售、网络银行等）
商务服务平台
商务支持平台
基础支持平台
Web 服务器平台、数据库平台
操作系统
计算机硬件及网络基础设施

通常，将 Web 服务器、部分的商务服务平台软件、商务支持平台软件、基础支持平台软件中的部分集成与开发工具集中在一个称为应用服务器的软件包中。此时，商务逻辑层在物理上可以简化为三部分，分别是核心商务应用、应用服务器、其他支持的软硬件。

3.2　应用服务器

应用服务器是从 Web 服务器发展而来。从 1994 年开始，在基于三层或 N 层分布式 Web 计算技术及动态 Web 应用的思想主导下，将核心的应用从 Web 服务器和数据库服务

器中分离出来,这样就出现了应用服务器。应用服务器为处理大量的用户与事务提供了一个更为结构化的解决方案;不但如此,应用服务器还能提供诸如负载均衡、线程池和服务恢复、Web Service 等特性,为分布式的电子商务应用打下了良好基础。

3.2.1 应用服务器的技术演变

应用服务器自出现之后,经过了若干个发展阶段,其所采用的技术由最初的 CGI 发展到最近的 Web Service,发展非常迅速。

1. 基于 CGI 的应用服务器

第一代应用服务器产品,功能比较简单,只是在 Web 服务器的基础上,添加了运行 CGI 程序的功能,早期 Microsoft 公司的 IIS 即是这样的应用服务器。IIS 原本是用来发布静态网页的 Web 服务器产品,通过添加 CGI、ISAPI(Internet Server Application Programming Interface)等应用接口,变成了应用服务器。其基本特征是将 HTML 代码嵌入到相关的程序代码中,当核心的商务功能比较简单时,可以考虑使用这种服务器;但当商务功能比较复杂时,就应另作它选。

公共网关接口(Common Gateway Interface,CGI)是 Web 服务器用来调用外部程序的一个接口。通过 CGI,Web 服务器能将用户从浏览器中送来的数据,交给 CGI 程序进行处理,并能将处理的结果再传给浏览器。Perl、C/C++ 、C Shell、VB 等语言可以用来编制 CGI 程序,程序的扩展名可为.pl、.exe、.bat 等。CGI 程序的工作原理如图 3-1 所示。

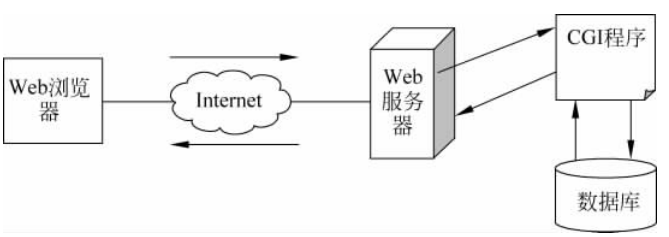

图 3-1 CGI 工作原理

从图 3-1 可以看出,CGI 的工作过程如下。

(1) 用户在客户端浏览器上单击链接或提交表单,利用 HTTP 协议中的 get 或 post 方法将数据发送到 Web 服务器。

(2) Web 服务器激活 CGI 程序,同时将数据表示成一定的格式,利用环境变量或标准输入流的方式将数据传给 CGI 程序。

(3) CGI 程序进行数据处理,必要时还会访问数据库或数据文件。

(4) CGI 程序处理完毕,将结果数据表示成一个 HTML 文件,交给 Web 服务器并通过它发回到浏览器。

由此可见,CGI 第一次使客户端与服务器端实现了信息交互。利用 CGI 可实现常见的访问计数、表格提交等功能。以下是用 C 语言编写的一个简单 CGI 程序,程序代码如下。

```
#include "stdio.h"
#include "stdlib.h"
void main()
```

```
{ char * str1, * str2;
  printf("Content - type: text/html\n\n");
  printf("< html > \n");
  printf("< head >< title > 用 C 编制 CGI 程序 </title ></head > \n");
  printf(" < body > \n");
  printf(" < p >第一个 CGI 程序,这是用 C 编写的 </p > \n");
  str1 = getenv( "QUERY_STRING" );
  str2 = getenv( "ACCEPT" );
  printf("< p > query_string: "); printf(" % s",str1); printf("</p >\n");
  printf("< p > accept: "); printf(" % s",str2); printf("</p >\n");
  printf(" </body ></html > \n");
}
```

将此程序(假如文件名为 test_c. c)编译之后,放到 IIS 站点根目录下(一般为 C:\inetpub\wwwroot),在浏览器中输入"http://127. 0. 0. 1/test_c. exe? name=zhang&age=20"所示的网页地址,即可看到结果。

从程序中可以看出,在用 get 方法提交网页访问请求的时候,CGI 程序利用了环境变量 QUERY_STRING 来获取相关的请求数据。由于 HTTP 协议对用 get 方法传送的请求数据的长度有所限制,一般不能超过 1024B(这也是 URL 的极限长度),当超过此长度时,就只能选用 post 方法来传送。用 post 方法传送时,Web 服务器通过 stdin(标准输入)流向 CGI 程序提供数据。注意,在有的应用服务器中,没有使用 EOF 字符作为 stdin 流的最后结束标志,当 CGI 程序读取 stdin 流中数据的时候,需要使用环境变量 CONTENT_LENGTH 来获知数据的长度,下面的例子就演示了这种情况。

例 3-1　一个使用 Post 方法提交请求数据的 CGI 程序,其程序代码如下(用 C 语言编写,文件名为 research. c,编译后的可执行文件名为 research. exe)。

```
# include "stdio. h"
# include "stdlib. h"
# include "string. h"
void main()
{
  char * strpost,buffer[400];
  FILE * fp;
  char * filename = "research. dat";
  printf("Content - type: text/html\n\n");
  printf("< html >\n");
  printf("< title >化妆品品牌调查系统</title >\n");
  printf("< body >\n");
  strpost = getenv("REQUEST_METHOD");
  if(strcmp(strpost,"POST") == 0)
  {
//这里假设 stdin 中的数据是以 EOF 作为结束标志
    fscanf(stdin," % s",buffer);
    //form 中传过来的数据格式为 name1 = S1&name2 = S2...
    //所以,这里 buffer 的内容为"brand = A"
fp = fopen(filename,"a");
    //这里只是简单地将数据存入 research. dat 文件中
    fprintf(fp," % s\n",buffer);
     fclose(fp);
    switch(buffer[6])
```

```
    {
    case 'A': printf("%s%s","您选择的是:","persume1"); break;
    case 'B': printf("%s%s","您选择的是:","persume2"); break;
    case 'C': printf("%s%s","您选择的是:","persume3"); break;
    case 'D': printf("%s%s","您选择的是:","persume4"); break;
    case 'E': printf("%s%s","您选择的是:","persume5"); break;
    }
  }
  else
    if(strcmp(strpost,"GET") == 0)
    {
      strcpy(buffer,getenv("QUERY_STRING"));
      //buffer 内容为 url 中?后面的参数,这里为"command = view"
      if(strcmp(buffer,"command = view") == 0)
      {
        fp = fopen(filename,"r");
        while(!feof(fp))
        {
            fscanf(fp,"%s",buffer);
            printf("<br>%s",buffer);
        }
        fclose(fp);
        printf("%s",buffer);
      }
    }

  printf("\n</body>\n");
  printf("</html>");
}
```

将 research.exe 文件放到站点的根目录下面,再在根目录下生成一个页面文件,内容如下(文件名为 research.htm)。

```
<HTML>
<HEAD><TITLE>一个化妆品使用品牌调查程序</TITLE></HEAD>
<BODY>
<h1>化妆品使用品牌调查</h1>
<FORM METHOD = "post" ACTION = "/cgi-bin/research.exe">
你喜欢什么品牌?<br>
<input type = "radio" name = "brand" value = "A" checked> persume1 <br>
<input type = "radio" name = "brand" value = "B"> persume2 <br>
<input type = "radio" name = "brand" value = "C"> persume3 <br>
<input type = "radio" name = "brand" value = "D"> persume4 <br>
<input type = "radio" name = "brand" value = "E"> persume4 <br>
<input type = "submit" value = "执行">
<input type = "reset" value = "取消">
</form>
查询<a href = "/cgi-bin/research.exe?command = view">化妆品使用品牌调查结果</a>
</BODY>
</HTML>
```

在浏览器中输入网址“http://127.0.0.1/research.htm”,单击“执行”按钮,就可运行research.exe 文件,并生成结果。

与下一代应用服务器所采用的 ASP、JSP、PHP 技术相比，CGI 程序是将 HTML 标识嵌入在传统的程序设计语言中，而不像 JSP、ASP 那样将控制代码嵌入在 HTML 标识中。所以在 CGI 程序中，如果要改变 HTML 的内容，就需要直接修改 CGI 程序，维护工作变得非常复杂，这是 CGI 应用技术的一大缺点。

另外，CGI 也存在严重的扩展性问题。每个 CGI 程序在服务器上执行都会产生一个进程，进程需要占用系统资源，当多个用户并发地访问 CGI 程序时，产生的多个独立的进程将会耗费服务器上的大量资源，甚至用尽服务器资源，导致系统瘫痪。

为克服以上弊端，下一代应用服务器使用了 ASP 技术。

2. 基于 ASP 的应用服务器

基于 ASP 的应用服务器，不但致力于克服 CGI 的缺点，而且还提供了集成开发的工具和相关的实用组件，通过使用 ActiveX 控件来实现相关的核心商务逻辑功能，使开发和发布动态网页变得十分容易，这样的应用服务器主要有微软公司的 IIS 系统。

在 IIS 系统中，ASP 利用“插件”和 API 简化了 Web 应用程序的开发。与 CGI 相比，其优点是 ASP 代码可以直接放在 HTML 中，程序编制具有灵活性，可以直接存取数据库及使用功能无限扩充的 ActiveX 控件。但其缺点也较为明显，由于它只能运行在 Microsoft 的操作系统平台上，不能或很难实现跨平台的 Web 服务器程序开发；同时 ASP 自身还存在安全问题。如有人曾在 IE 浏览器的地址栏中输入 URL 网址，并在其后跟随一个特殊的字符（例如，“http://www. somehost. com/some. asp::$DATA”，“http://www. somehost. com/some. asp&.2e”，“http://www. somehost. com/some%2e@41sp”，“http://www. somehost. com/some%2e%asp”等），ASP 源代码就会直接在用户的浏览器上显示出来；也有人曾编写如下的 ASP 代码文件 cat. asp。

```
<%
  Response.ContextType = "text/plain"
  file = Request.QueryString("file")
  set fsFilesys = CreateObject("Scripting.FileSystemObject")
  set tsText = fsFilesys.OpenTextFile(file)
  Response.Write tsText.Readall
  tsText.Close
  set fsText = Nothing
  set fsFilesys = Nothing
%>
```

通过在 IE 地址栏中输入“http://www. somehost. com/cat. asp? file＝文件名”，利用 VBScript 脚本程序来调用 FileSystemObject()函数，可以直接查看到站点上文件名所指定的文件中的 ASP 源代码；除此之外，还发生了诸如数据库密码验证、inc 文件安全的问题。尽管 Microsoft 公司事后进行了补救，但有关安全的问题终不能让人放心。

3. 基于 Java 的应用服务器

Java 具有跨平台性，利用 Java 来构筑服务器端的应用，不管是在 Windows NT、UNIX 上，还是在其他的主机系统上都能运行；利用互联网、中间件和分布对象等新功能，还能将 Java 应用配置在多个结点上，实现负载平衡；另外，Java 应用还具有安全的特性。

1) Servlet

正如前面所说，Servlet 是一组运行在 Web 服务器端的 Java 程序，Sun 公司之所以将它

取名为Servlet,可能与运行在客户端的Applet有关。Servlet的工作原理和CGI有很多相似之处。由于现在的Web服务器,大多提供Servlet接口,使用Servlet程序可做到与平台无关;同时由于Servlet内部是以线程的形式提供服务,对每个请求不必都启动一个进程,因此Servlet的执行效率非常高。

2) JSP

如果说,Servlet程序是把HTML内容嵌入在Java程序中,那么JSP则是把Java程序嵌入在HTML文件中,这样做的优势在于,当改变HTML内容时不必重新编译程序,这与ASP有点类似。随着商务应用的进一步复杂,嵌入在HTML中的Java代码也越来越复杂,并且越来越庞大,对这些代码的管理和程序调试越难进行,在这样的情况下,对于商务应用的开发者来说,最好的方法是尽可能地减少JSP中的代码,也就是需要将HTML代码和Java代码分离开来,或者说是需要将页面显示和商业逻辑处理分开,因此JSP常采用如图3-2所示的模式。

图3-2 JSP+Bean服务模式

3) JSP+Servlet+JavaBean

如图3-3所示,JavaBean主要实现业务逻辑处理,JSP主要实现页面的构建,Servlet主要是实现与用户的交互及控制功能,即接受用户的请求,控制JSP来产生响应。

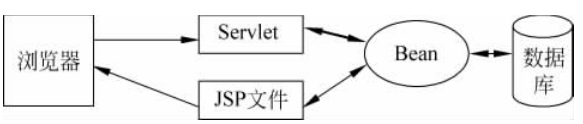

图3-3 JSP+Servlet+JavaBean服务模式

基于Java的应用服务器中的应用,对于数据库和中间件的访问,一般是采用直接调用API来实现,在复杂性和应对未来技术变革的能力上,还有许多问题需要解决。

4. 基于Java组件和应用服务的应用服务器

现在流行的适应Java组件的应用服务器技术,对系统开发产生了划时代的影响。这里所说的Java组件,是以EJB为中心的服务器端的组件技术,是构建分布式应用的关键技术。分布式的电子商务多层体系结构要求应用服务器具备三方面的技术,首先是开发环境,开发人员需要一种创建新组件、并将已有组件加以集成的开发环境;其次是应用程序的集成,由于企业商务计算环境比较复杂,它综合了传统的应用程序和新型应用程序,因此开发人员需要集成各种应用程序,以创建出更强大的应用;第三是应用程序的配置,由于典型的Web应用程序是分布式,其组件运行在不同的服务器上,并且有大量的用户对其进行访问,因此需要配置平台的支持,以便在用户剧增时能有效地扩展,并保持系统的稳定。

目前,应用服务器技术正朝着面向服务的方向发展,朝着集成化、可扩展的方向发展。一方面,应用服务器会集成越来越多的功能,有的功能是应用服务器厂家自己开发的,有的是第三方开发的,它们组成了一个统一的整体;另一方面,应用服务器又向着兼容多种技术

标准(如 CORBA、DCOM、EJB、RMI、XML、Web Service 等)的方向发展,可在多个平台上运行,能连接多种不同的数据库(如 Oracle、Sybase、DB2、SQL Server、Informix、MySQL 等)。

3.2.2　应用服务器的产品类型及开发工具

下面列举了目前市场上常用的一些应用服务器,见表 3-2[①]。

表 3-2　常见的应用服务器

产　　品	产品类型	是否支持 Java Servlets	是否支持 JSP
BEA WebLogic Server 7.0	服务器	2.3	1.2
Borland Enterprise Server, AppServer Edition	服务器	2.3	1.2
Caucho Technology Resin	新增引擎	2.3	1.2
Computer Associates Advantage Joe	服务器	2.3	1.2
EasyThings Web Server	服务器	2.2	1.1
Fujitsu INTERSTAGE	服务器	2.3	1.2
Gefion Software LiteWebServer	服务器	2.3	1.2
Hewlett Packard Total-e-Server	服务器	2.2	1.1
IBM WebSphere Technology for Developers	服务器	2.3	1.2
IBM WebSphere Application Server 4.0	服务器	2.2	1.1
IONA Orbix E2A Application Server Platform, J2EE Edition	服务器	2.3	1.2
Jetty	服务器		
	3.1	2.2	1.1
	4.0	2.3	1.2
Lutris Technologies Enhydra	服务器	2.2	1.1
Macromedia JRun 4	服务器、附加引擎	2.3	1.2
New Atlanta ServletExec	新增引擎		
	3.1	2.2	1.1
	4.0	2.3	1.2
Novocode NetForge	服务器	2.1	No
Oracle 9i Application Server	服务器	2.3	1.2
Orion Application Server	服务器	2.3 PFD2	1.1
Pramati Server 3.0	服务器	2.3	1.2
Secant Technologies ModelMethods Enterprise Server	服务器	2.1	1.0
Servertec Internet Server	服务器	2.2	1.1
Silverstream eXtendApplication Server 4.0 Beta	服务器	2.3	1.2
SITEFORUM Web Server and Interaction Platform	服务器	Yes	No
Sun ONE Web Server (formerly iPlanet Web Server)	服务器	2.3	1.2
Sun ONE Application Server 7, Platform Edition (formerly iPlanet Application Server)	服务器	2.3	1.2
Sybase EAServer	服务器	2.3	1.2

① 参见 http://developers.sun.com。

续表

产　品	产品类型	是否支持 Java Servlets	是否支持 JSP
Tagtraum Industries jo!	服务器	2.1,2.2	1.1
Trifork Application Server 3.1	服务器	2.3	1.2
vqSoft vqServer	服务器	2.0	No
W3C Jigsaw	服务器	2.2	No
Zeus Technology Web Server	服务器	2.2	No

表3-3列出了一些支持JSP和Servlet的开发工具。

表3-3　支持JSP和Servlet的开发工具

产　品	是否支持 Servlets	是否支持 JSP
Adobe GoLive 6	Yes	Yes
Borland JBuilder 7	2.3	1.2
IBM WebSphere Studio 4.0	2.2	1.1
IBM VisualAge 4.0	2.2	1.1
Macromedia Dreamweaver UltraDev 4	No	1.0
Macromedia JRun Studio 4	2.3	1.2
MERANT PVCS Content Management Server	Yes	Yes
NetObjects Fusion MX	Yes	Yes
Oracle JDeveloper	Yes	Yes
Pramati Studio 3.0	2.3	1.2
Sun ONE Studio 4.0 (formerly Forte for Java)	2.3	1.2
WebGain Studio	2.2	1.1

3.2.3　应用服务器的基本功能

应用服务器的产品很多,功能差异较大,可以提供的服务千差万别,但通常都提供如下的功能[1]。

1. 提供高性能的应用程序运行环境

1) 内容缓存

当用户访问Web服务器,存取HTML页面时,一般Web服务器需要从硬盘上读取HTML文件,然后传输给用户。每次用户访问,Web服务器都重复相同的处理过程。内容缓存将用户经常访问的HTML页面或者Web处理结果存储在服务器的内存中,在用户请求某一服务时,服务器首先从内存中检索是否有相应的结果,如果存在,则不必重新处理,而直接将结果返回给用户。由于内存的存取速度远比硬盘的存取速度快,所以服务器如果支持内容缓存功能,那么系统的响应时间可以缩短。

2) 数据库连接缓存

通常用户通过Web服务器访问数据库时,Web上的应用程序和数据库服务器之间需要首先建立连接,然后才能存取数据,在处理结束后,这种连接被关闭,每次用户访问都需要重复同样的步骤。由于数据库连接过程比较消耗系统资源,而且时间也比较长,尤其是利用

公共网关接口 CGI 进行连接时，效率尤其低，对系统响应时间影响很大。数据库连接缓存是指在 Web 服务器和数据库服务器之间建立经常性的连接，当用户需要访问数据库时，直接利用这些已经存在的连接，操作结束后，连接仍然保持而不被关闭，这样一来，用户访问数据库的步骤简化，进而提升系统效率。

3）支持进程的多线程执行

该功能是将一个进程分解成多个可以独立的线程并行执行，提高应用程序的运行效率，缩短运行时间。

4）大量用户访问情况下的负载均衡

该功能根据用户访问量及服务器的处理能力，动态调整每个服务器的负载，使服务器的处理能力和负载之间保持平衡，提高系统的可靠性和整个系统的性能，支持分布式联机事务处理。

2. 提供可扩充性

由于电子商务系统在规划设计阶段难以精确预测未来服务的客户数量，预测其客户增长的速度也比较困难，所以，电子商务应用系统应当具备良好的可扩充性能。应用服务器主要通过三种方式提高系统可扩充性：首先利用服务器集群技术，将系统压力分摊在集群服务器的各个设备上；其次，应用服务器大多支持多 CPU 系统，从而在系统访问压力增加时，通过增加服务器 CPU 来提高系统的处理能力；最后，一些优秀的应用服务器还提供应用的动态负载均衡，使服务器的性能和访问压力之间得到匹配。

3. 提供会话管理

用户一般通过和电子商务系统进行对话完成商务活动。以一次网络购物过程为例，用户经常要经过多次挑选才能够确定购物清单，每次挑选货物并将其放入购物车的过程就是一次会话过程。如果计算机系统无法记录用户购物清单的话，那么必须要求用户一边在线挑选，一边用纸笔记录，否则在付款时可能发现难以回忆起整个购物过程中都买了些什么商品，这是一件时常发生但又令人尴尬的事件。

会话管理的作用是记录和管理客户的每次人机会话过程。应用服务器如果支持会话管理功能，那么如果用户重复上述的购物过程，付款时，电子商务系统会自动地提供其购买清单。因此，如果一次商务活动必须经过多个复杂的步骤才能完成，会话管理是非常重要的。

4. 提供目录及内容管理

应用服务器一般提供目录管理和内容管理工具，利用这些工具可以完成用户访问内容的控制、分层数据组织、目录更新及控制等服务。

5. 提供商务引擎

商务引擎主要是为商务系统提供业务支持，它所提供的服务主要包括个性化服务（如 BEA 的 Weblogic Personal Server）、客户关系管理 CRM（如 BroadVision，Oracle 公司的产品）、供应链管理、电子交易市场（如 Oracle iExchange）等，不同的应用服务器产品在这方面的差异很大。

6. 提供系统管理

目前应用服务器都提供系统管理工具，具体功能包括以下几个方面。

(1) 性能配置管理。性能配置管理主要围绕如何为商务应用配置合理的系统资源,如对服务进程数的调整、结果缓存大小的调整等。

(2) 存取控制管理。又称为访问控制,其目的是对系统资源的访问权限进行限制,保护特定内容的安全,例如控制只有特定权限的用户才能访问系统当中的某些应用或者页面等。

(3) 系统日志管理。这一功能对系统访问、应用运行、存取失败等情况进行记录,从而为系统的故障诊断、分析和性能优化提供依据。

3.2.4 应用服务器的安装

目前,许多企业尤其是中小企业,在建立自己的电子商务系统时,基于成本考虑,往往选择 Tomcat 应用服务器作为核心的商务应用环境;同时由于本书中的许多实例也是基于 Tomcat 运行环境,在运行这些实例之前需要预先安装好 Tomcat 服务器,所以在这里集中了解一下它的安装与测试过程。

1. 安装及配置

Tomcat 作为一个开源的应用服务器,支持 Servlet 和 JSP,受到越来越多的软件公司和开发人员的喜爱,应用前景越来越广。作为 Jakarta 项目中的一个重要子项目,曾被 JavaWorld 杂志选为 2001 年度最具创新的 Java 产品。

安装 Tomcat 之前,在系统上必须先安装好 JDK 1.2 以上版本的 Java。Tomcat 4.0.1 版本用了一个新的 Servlet 容器 Catalina,完整地实现了 Servlet 2.3 和 JSP 1.2 规范。

1) 安装

对于 Windows 平台,从 Tomcat 网站[①]下载 jakarta-tomcat-4.0.1.exe,按照一般的 Windows 程序进行安装即可。对于 Linux 平台,下载 jakarta-tomcat-4.0.1.tar.gz4,将其解压到一个目录。安装好的 Tomcat,其目录结构见表 3-4。

表 3-4 Tomcat 的目录结构

目录名	简 介
bin	存放启动和关闭 tomcat 脚本
conf	包含不同的配置文件,server.xml(Tomcat 的主要配置文件)和 web.xml
work	存放 JSP 编译后产生的.class 文件
webapp	存放应用程序示例,以后要部署的应用程序也要放到此目录
logs	存放日志文件
lib/japser/common	这三个目录主要存放 tomcat 所需的.jar 文件

2) 配置

运行 Tomcat 之前,需要正确地设置 JAVA_HOME、CLASSPATH、CATALINA_HOME 及 PATH 这 4 个环境变量。在 Windows 下其内容应设置如下。(假设 JDK 安装在“C:\jdk1.4”路径下,Tomcat 安装在“C:\tomcat4.1”路径下)。

```
set JAVA_HOME = c:\jdk1.4;
```

① 参见 http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.1/bin/jakarta-tomcat-4.0.1.tar.gz。

```
set CLASSPATH = c:\jdk1.4\lib;
set CATALINA_HOME = c:\tomcat4.1;
set PATH = c:\jdk1.4\bin; %path%;
```

2. 服务测试

为了测试 Tomcat 的安装及配置是否正常，在安装、配置好 Tomcat 后，可以启动 Tomcat 服务来进行测试。

进入 Tomcat 安装路径下的 bin 子路径，运行 startup 程序，启动 Tomcat 服务；然后打开浏览器，在浏览器中输入“http://localhost：8080/”网址进行测试，当看到如图 3-4 所示的画面时，说明一切正常。

图 3-4　测试 Tomcat 服务器

需要说明的是，Tomcat 属于应用服务器，本身就具有 Web Service 的功能，只不过默认的 Web Service 端口号是 8080，而不是 80。当然，也可以在它的配置文件中进行修改，将端口号设置为其他的值。

3. 应用配置

安装并测试好 Tomcat 后，接下来就可以布置电子商务的应用程序了。尽管如此，为了使 Tomcat 符合自己的特定应用，在布置之前，常常需要修改 Tomcat 中的一些配置信息。Tomcat 的配置信息主要放在 conf\server.xml 文件中，核心内容见表 3-5。

表 3-5　Tomcat 中的核心内容

元素名	属性	解　　释
server	port	指定一个端口，这个端口负责监听关闭 Tomcat 的请求
	shutdown	指定向端口发送的命令字符串
service	name	指定 service 的名字

续表

元素名	属性	解　　释
Connector(表示客户端和service之间的连接)	port	指定服务器端要创建的端口号,并在这个断口监听来自客户端的请求
	minProcessors	服务器启动时创建的处理请求的线程数
	maxProcessors	最大可以创建的处理请求的线程数
	enableLookups	如果为true,则可以通过调用request.getRemoteHost()进行DNS查询来得到远程客户端的实际主机名,若为false则不进行DNS查询,而是返回其IP地址
	redirectPort	指定服务器正在处理HTTP请求时收到了一个SSL传输请求后重定向的端口号
	acceptCount	指定当所有可以使用的处理请求的线程数都被使用时,可以放到处理队列中的请求数,超过这个数的请求将不予处理
	connectionTimeout	指定超时的时间数(以毫秒为单位)
Engine(表示指定service中的请求处理机,接收和处理来自Connector的请求)	defaultHost	指定默认的处理请求的主机名,它至少与其中的一个host元素的name属性值是一样的
Context(表示一个Web应用程序,通常为war文件,关于war的具体信息见servlet规范)	docBase	应用程序的路径或者是WAR文件存放的路径
	path	表示此Web应用程序的URL的前缀,这样请求的url为http://localhost:8080/path/* * * *
	reloadable	这个属性非常重要,如果为true,则Tomcat会自动检测应用程序的/WEB-INF/lib和/WEB-INF/classes目录的变化,自动装载新的应用程序,我们可以在不重启Tomcat的情况下改变应用程序
host(表示一个虚拟主机)	name	指定主机名
	appBase	应用程序基本目录,即存放应用程序的目录
	unpackWARs	如果为true,则Tomcat会自动将war文件解压,否则不解压,直接从war文件中运行应用程序
Logger(表示日志、调试和错误信息)	className	指定logger使用的类名,此类必须实现org.apache.catalina.Logger接口
	prefix	指定log文件的前缀
	suffix	指定log文件的后缀
	timestamp	如果为true,则log文件名中要加入时间,如localhost_log.2001-10-04.txt
Realm(表示存放用户名,密码及role的数据库)	className	指定Realm使用的类名,此类必须实现org.apache.catalina.Realm接口
Valve(功能与Logger差不多,其prefix和suffix属性解释和Logger中的一样)	className	指定Valve使用的类名,如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息
	directory	指定log文件存放的位置
	pattern	有两个值,common方式记录远程主机名或IP地址、用户名、日期、第一行请求的字符串、HTTP响应代码、发送的字节数。combined方式比common方式记录的值更多

Tomcat 作为一个 Servlet(JSP 也被编译为 Servlet 来执行)容器,其应用前景是非常好的。如果与 Jboss[①] 软件结合起来,则可以实现 Sun 公司提出的 J2EE 规范(Jboss 用作 EJB 服务器);如果再与另外一个开源的应用软件 Enhydra[②] 结合起来,则功能将会更强大,管理界面将更友好,部署应用程序也将更简单。

3.3　中间件与组件的开发

3.3.1　中间件与组件

在大型的电子商务系统中,通常存在多种硬件系统平台,在这些硬件平台上又存在各种各样的电子商务应用软件以及多种风格各异的用户界面,这些硬件系统平台还可能采用不同的网络协议和网络体系结构来进行连接。如何把这些系统(即所谓的异构系统)集成起来并开发新的应用是一个非常现实而困难的问题。为了解决这些问题,人们提出了中间件(middleware)的概念。中间件是位于平台(硬件和操作系统)和应用之间的通用服务,这些服务具有标准的程序接口和协议。标准的程序接口和协议定义了一个相对稳定的高层应用环境,不管底层的计算机硬件和系统软件怎样更新换代,只要将中间件升级更新,并保持中间件对外的接口定义不变,电子商务的应用软件几乎不需要任何修改,就能实现系统的升级。

中间件所包括的范围十分广泛,针对不同的应用需求涌现出多种各具特色的中间件产品。但至今中间件还没有一个比较精确的定义,因此,在不同的角度或不同的层次上,对中间件的分类也会有所不同。由于中间件需要屏蔽分布环境中异构的操作系统和网络协议,它必须能够提供分布环境下的通信服务。基于目的和实现机制的不同,通信服务可分为以下几类:远程过程调用(Remote Procedure Call,RPC)、面向消息的中间件(message-oriented middleware)、对象请求代理(Object Request Brokers,ORB)、面向服务的中间件(service-oriented middleware)。组件是实现中间件最有效的技术手段。

组件规范描述了开发可重用组件及组件间相互通信的标准。按照组件规范,通过重用已有的组件,电子商务的开发者就可以像搭积木一样快速地构造自己的中间件,不仅节省时间和经费,提高工作效率,而且产生的中间件更加规范、更加可靠。

组件技术已经成为建立应用框架(application framework)和软构件(software component)的核心技术,在开发大型分布式电子商务应用系统中表现出强大的生命力,并形成了几项具有代表性的主流技术,即对象管理组(Object Management Group,OMG)的 CORBA(Common Object Request Broker Architecture)、Microsoft 公司的 ActiveX/DCOM (Distributed Compound Object Model)、Sun 公司的 JavaBean/EJB/RMI 和最近流行的基于 XML/Web Service 的组件技术。

1. CORBA

OMG 是一个非营利性国际组织,致力于使 CORBA 成为“无所不在的中间件”。1989

① 网址为 http://www.jboss.org。

② 网址为 http://www.enhydra.org。

年成立时仅有8家公司参与,而今天已经是拥有900多个机构成员的"议会式"标准化组织。世界上几乎所有最有影响的计算机公司(如IBM、Microsoft和HP等)、著名的工商企业(如Boeing、Citibank和FordMotor等)和大学研究机构都是这个组织的成员。OMG所制定的分布对象计算标准规范包括CORBA/IIOP、对象服务、公共实施和领域接口规范。遵照这些规范开发出的分布处理软件环境可以在几乎所有的主流硬件平台和操作系统上运行。现在,CORBA/IIOP已成为Internet上实现对象互访的技术标准,OMG的IIOP也已成为许多公司(如Oracle、Netscape、Sun和IBM等)进行系统集成的基本协议。1995年以来,基于CORBA软件的企业级应用发展迅猛,大有覆盖RPC(Remote Procedure Call protocol,远程过程调用协议)、DCE(Data Circuit-terminating Equipment,数据通信设备)之势。目前世界上有一定影响的CORBA软件制造商已有10多家。

CORBA是一种规范,它定义了分布式组件如何实现互操作,它由下列四部分组成。

(1) Object Request Broker,CORBA对象互通信的软总线。

(2) Object Services(可选),ORB所能提供的系统级服务,如安全性、命名/目录和事务处理。

(3) Common Facilities(可选),应用程序级服务,如复合文档等。

(4) Application Interface,CORBA对象和应用的外部接口。

CORBA对象可以用任何一种CORBA软件开发商所支持的语言,如C、C++、Java、Ada和Smalltalk来编写;同时,CORBA对象可以运行在任何一种CORBA软件开发商所支持的平台上,如Solaris、Windows 95/NT、OpenVMS、DigitalUnix、HP-UX或AIX等。这意味着,可以在Windows 95下运行Java应用程序,同时动态调入并使用C++对象,而该对象可能存储于一个在Internet上的UNIX Web服务器上。

在CORBA中专门以接口描述语言(Interface Description Language,IDL)来统一描述CORBA对象所提供的所有外部接口,脱离特定语言的约束,使得与语言无关。但同时需要一个从IDL接口到由本地语言(C/C++、Java)所编写的CORBA对象的"桥梁"——对象请求解析器(object request broker)为中介,它可以架构在多种流行通信协议之上(如TCP/IP或是IPX/SPX)实现。在TCP/IP上,来自于不同开发商的ORB用Internet Inter-Orb协议(IIOP)进行通信,这是CORBA 2.0标准(最新的版本)的一部分。

目前,对于较为流行的编程语言(包括C++、Smalltalk、Java和Ada 95),已经有了许多第三方的ORB。随着其他语言的逐渐流行,CORBA开发商毫无疑问地要开发出相应的ORB来支持它们。

2. DCOM/COM+

ActiveX/DCOM是由Microsoft公司推出的对象组件模型,最初用于集成Microsoft的办公软件,目前已发展成为Microsoft世界的应用系统集成标准,并集中反映在其产品ActiveX中。目前,只有OMG的技术能够支持异构环境中大型分布式电子商务应用的开发,而Microsoft的DCOM技术尚不能完全胜任。Microsoft的优势主要表现在应用和市场能力上,从市场策略考虑,Microsoft决定支持OMG提出的OLE/COM与CORBA的互操作标准,从而使COM的对象能够与CORBA的对象进行通信。今后若干年内,OMG和Microsoft的分布对象技术将共存,并在许多方面相互渗透。

3. RMI

按照 Sun 和 Javasoft 公司对 Java 的界定，Java 是一个应用程序开发平台，它提供了可移植、可解释、高性能和面向对象的编程语言及运行环境。远程方法调用(Remote Method Invocation，RMI)是分布在网络中的各类 Java 对象之间进行方法调用的一种 ORB 实现机制。之所以这么讲，是因为 CORBA 技术与 Java 技术存在着天然的联系。Sun 公司是 OMG 的创始成员，CORBA 标准中的许多内容(例如 IDL 标准、IIOP 标准)是以 Sun 公司提交的方案为核心制定的。CORBA 与 JavaBean/EJB/RMI 的主要区别在下列两个方面。

(1) 程序设计语言无关性是 CORBA 的重要设计原则，而 Java/RMI 依赖于 Java 语言和 Java 虚拟机。

(2) JavaBean/EJB/RMI 技术的最大成就是使对象能够作为参数在 Internet 上迁移和执行(对象序列化，serialize)，而 CORBA 2.0 标准中只考虑对象的远程访问，没有对象作为“值”传递的承诺。

正是由于这两个技术的天然联系和各自的优势，CORBA 技术与 Java 技术的融合已成为必然。

RMI 早期使用 Java 远程消息交换协议(Java Remote Messaging Protocol，JRMP)进行通信。JRMP 是专为 Java 的远程对象制定的协议。用 Java RMI 技术开发的应用系统可以部署在任何支持 Java 运行环境(Java Run Environment，JRE)的平台上。但由于 JRMP 是专为 Java 对象定制的，因此 RMI 对于用非 Java 语言开发的应用系统的支持不足，不能与用非 Java 语言书写的对象进行通信。随着 Java 与 CORBA 技术的融合，后期出现了支持与用非 Java 语言书写的对象进行通信的 RMI-IIOP 协议。这两种协议的实现机制不同，但编写 RMI 程序的过程基本相同，它们都是基于存根(stub)/骨架(skeleton)架构结构。RMI-IIOP 和 JNDI(J2EE 中的目录和名字服务，商务支持平台的基本功能)是实现企业 JavaBean 的技术基础。

4. Web Service

Web Service 是最近制定的一组标准，目的是利用成熟的 Web 技术，通过 SOAP (Simple Object Access Protocol，简单对象访问协议)协议、WSDL(Web Services Description Language，服务描述语言)服务描述语言和 UDDI(Universal Description Discovery and Integration，统一描述发现集成协议)统一描述发现集成协议来实现跨语言(RMI 要求处理两端都是 Java 环境)、跨平台(DCOM 要求处理两端为 Windows 平台，CORBA 要求处理两端为同一个 ORB)、跨网络之间的分布处理与组件应用。

3.3.2　JeanBean 与 JSP 指令

尽管可以在 JSP 代码段中放置大量的代码来实现所要的商务逻辑功能，但这是很费力的，有时也是不可能的，通常需要调用系统可重用的、被称为 JavaBean(或 EJB)的组件。在 Sun 公司的 JavaBean 规范定义中，Bean 的正式说法是：“Bean 是一个基于 Sun 公司的 JavaBean 规范的、可在编程工具中被可视化处理的可复用的软件组件”。因此 JavaBean 具有 4 个基本特性：独立性、可重用性、在可视化开发工具中使用和状态可以保存。

JavaBean 分成可视组件和非可视组件。在 JSP 中主要使用非可视组件，对于非可视组

件,不必去设计它的外观,主要关心它的属性和方法。JavaBean 就像 ActiveX 控件,但与 ActiveX 相比,JavaBean 真正实现了代码的可重用性,不过分依赖于处理平台和开发语言。从本质上讲,JavaBean 组件就是 Java 类,只不过 JavaBean 的结构必须满足一定的命名约定,属性必须具有 set 和 get 的方法。一个完整的 JavaBean 在类的命名上需要遵守以下 4 项规定。

(1) 如果类的成员变量的名字是 xxx,那么为了更改或获取成员变量的值,在类中使用方法 getXxx(),来获取属性 xxx; 使用方法 setXxx(),来修改属性 xxx。

(2) 对于 boolean 类型的成员变量,允许使用 is 代替 get 和 set。

(3) 类中方法的访问属性必须是 public 的。

(4) 类中如果有构造方法,那么这个构造方法也是 public 的,并且是无参数的。

JavaBean 的价值体现在它的一系列属性里。用一个人来做类比,如果这个人是一个 JavaBean,那么他的姓名、身份证号码和地址就是这些属性。在 JSP 中可以通过 JSP 指令来访问它的各种属性。JSP 指令是一种特殊标记,用<JSP: >来表示,用于控制 JSP 引擎的动作。在 JSP 中共有以下 6 种 JSP 指令,其中后 3 种专用于 JavaBean。

```
<jsp: include>
<jsp: forward>
<jsp: plugin>
<jsp: setProperty>
<jsp: getProperty>
<jsp: useBean>
```

下面介绍其中几种 JSP 指令。

(1) <jsp: useBean>指令: <jsp: useBean>指令在 JSP 中的语法格式如下。

```
<jsp: useBean id = "beanInstanceName" scope = "page | request | session | application"
{
  class = "package.class" | type = "package.class" | class = "package.class" type = "package.class" |
  BeanName = "{package.class | <% = expression %>}"
} />
```

或者

```
<jsp: useBean …>
    other elements
</jsp: useBean>
```

这里 other elements 主要指的是 setProperty 和 getProperty 动作,以及各种设置或获取属性值的方法等。

例如,"<jsp: useBean id = "localName" class = "company.Person" scope = "application" />",这里用 id 属性来确定实例,用 class 来指定到哪里去找相应的 Java 类(此类一般放在 Web 应用目录 WEB-INF\classes 下),用"scope="page""来确定是否为这个单独的页面保存信息(默认设置),用"scope="request""来确定是否为这一次请求保存信息,用"scope="session""来确定是否为这一次会话过程保存信息,用"scope="application""来确定是否为整个应用程序保存信息。若把 scope 设为"session",则可以在

JSP 页面中轻松地保存诸如购物车之类的内容。

(2) <jsp: getProperty>与<jsp: setProperty>指令：在声明了一个 JavaBean 之后，可以使用<jsp: getProperty>标记来得到一个 JavaBean 属性的值。在使用<jsp: getProperty>标记时，需要指定 useBean 的 id 以及要取值的属性，这样才会取到所需要的实际值。例如：

```
<jsp: getProperty id = "localName" property = "name" />
```

若要改变 JavaBean 的属性，需要使用<jsp: setProperty>标记。使用这个标记时，也需要确定 useBean 的 id 和属性和所提供的新值。例如，“<jsp: setProperty id = "localName" property = "*" />”，这里“property = "*"”表示从一个提交的表单中直接获取同名的属性值，作为所设置的值。若为“<jsp: setProperty id = "localName" property = "address" param = "parameterName" />”，则表示用参数 parameterName 值来设置 id 所指定的属性的值。若为“<jsp: setProperty id = "localName" property = "serialNumber" value = "string" />”，则直接用给出的值来设置 id 所指定的属性的值。若为“<jsp: setProperty id = "localName" property = "serialNumber" value = <%= expression %> />”，则表示从 JSP 表达式中来获取值，以此作为设置 id 所指定的属性的值。

3.3.3　使用 JavaBean 实现中间件

作为一个例子，下面用 Java 语言编写了一个 JavaBean 组件，以此作为一个简单的中间件，可以安装到 Tomcat 应用服务器上，其程序如下(文件名为 userBean.java)。

```
package company;
public class userBean
{ public String useName;
  public int myID;
  public userBean(){ myID = 1001; useName = "zahnghz"; }
  public String getname(int ID)
        {
          if(ID == this.myID) return useName;
          else
            return new String("No such man");
        }
  public int getID() { return myID; }
  public void setID(int InID) {myID = InID; }
}
```

编译此文件后，将会生成一个名为 userBean.class 类文件，将此文件复制到 Tomcat 服务器的“webapps\ROOT\WEB-INF\classes\company\”目录下(这里需要自己在“webapps\ROOT\WEB-INF\classes\”目录下新建一个子目录 company)；然后在“webapps\ROOT\”目录下生成一个名为 userbean.jsp 的文件，内容如下。

```
<html>
<head><title>Using Bean</title></head>
<body>
<jsp: useBean id = "OurBean" scope = "page" class = "company.userBean" />
```

```
<h4>Using Bean</h4>
<%
  String tempStr = request.getParameter("ID");
  int userID = Integer.valueOf(tempStr).intValue();
 %>
<% = userID %>
<% = OurBean.getname(userID) %>
ID: <jsp: getProperty name = "OurBean" property = "ID" />
</body>
</html>
```

从 userbean.jsp 脚本文件中可以看出,它使用了一个名为 userBean.class 的 JavaBean 组件,放在包 company 中。在执行时,该组件的 id 为 OurBean。

最后,再在"webapps\ROOT\"目录下生成一个名为 onejavabean.htm 网页文件,内容如下。

```
<html>
<head><title>Name</title></head>
<body>
<form method = "post" action = "userbean.jsp">
    <input type = "text" size = "10" name = "ID"><p>
    <input type = submit value = "submit">
</form>
</body>
</html>
```

打开浏览器,在地址栏输入"http://localhost:8080/onejavabean.htm"网址,即可进行测试。

3.4 EJB 组件的开发

3.4.1 EJB 组件简介

EJB(Enterprise JavaBean)是 J2EE 中的核心技术,EJB 不是一个产品,而是 Java 服务器端服务框架的规范。软件厂家可根据它来编程实现 EJB 服务器;应用程序开发者可以在此环境下实现应用所需的商业逻辑,而不用关注其他的实现问题。EJB 的组件架构模型如图 3-5 所示。

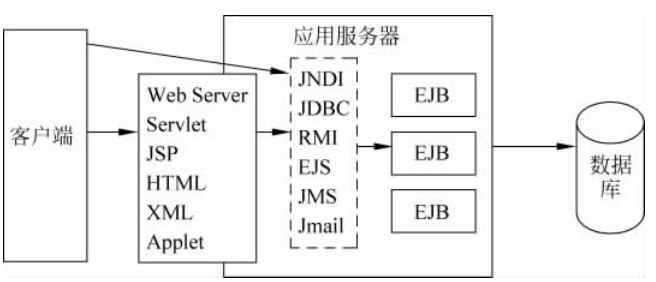

图 3-5 EJB 组件架构模型

Sun 公司在发布 EJB 2.0 中对 EJB 的定义如下。

EJB 是用于开发和部署多层结构的、分布式的、面向对象的 Java 应用系统的跨平台的组件体系结构。采用 EJB 可以使开发商业应用系统变得容易，应用系统可以在一个支持 EJB 的环境中开发，开发完之后部署在其他支持 EJB 规范的服务器平台上的环境中，随着需求的改变，应用系统可以不加修改地迁移到其他功能更强、更复杂的服务器上，并且具有可扩展性、交互性、多用户安全特性。

EJB 是部署在服务器上的可执行组件或商业对象。有一个协议允许对其进行远程访问或在特定服务器上安装或部署它们。有一系列机制允许它们将服务安全性、事务行为、并发性(多个客户机同时访问的能力)和持久性(其状态可以保存多久)的主要方面授权给 EJB 服务器上其所在的容器。当安装在容器中时，它们获得各自的行为，该行为提供不同质量的服务，因此，选择正确的 EJB 服务器至关重要。

EJB 是设计成运行在服务器上，并由客户机调用非可视远程对象。可通过多个非可视 JavaBean 构建 EJB。它们有一个部署描述符，其目的与 JavaBean 属性相同，以后可由工具读取 Bean 的描述。EJB 还独立于平台，一旦编写好，还可以在任何支持 Java 的平台(包括客户机和服务器)上使用。

需要强调的是，JavaBean 不仅可以用于客户端应用程序的开发，也可以用于非图形化服务器端 Java 应用开发的组件模型，但和 EJB 的区别是，如果用 JavaBean 创建服务器端应用程序，需要设计整个的服务框架。在多层结构的分布式应用中，服务框架的实现是非常烦琐的，对于 EJB 来说，服务框架已经提供，因此大大地简化了系统的开发过程。EJB 组件总是分布式的，这是它们与标准 JavaBeans 组件最根本的区别。JavaBean 是 Java 的组件模型。在 JavaBean 的规范中定义了事件和属性等特性。EJB 也定义了一个 Java 组件模型，但是 EJB 组件模型和 JavaBean 组件模型是不同的。JavaBean 提供了基于组件的开发机制，一般 JavaBean 是可视化的组件，也有一些 JavaBean 是非可视化的，在 JavaBean 组件模型中，重点是允许开发人员在开发工具中可视化的开发组件，为此，JavaBean 规范详细地解释了组件间事件登记、传递、识别和属性使用、定制和持久化应用编程接口和语义。而 EJB 没有用户界面，并完全位于服务器端。EJB 的侧重点是详细地定义了可移植的部署 Java 组件的服务框架模型。因此，其中并没提及事件，因为 EJB 通常不发送和接收事件。同样，EJB 也没有提及属性，属性定制并不是在开发时进行，而是在运行时(实际上在部署时)通过一个部署描述符来描述。

3.4.2　EJB 中的角色

EJB 结构中的角色 EJB 组件结构是基于组件的分布式计算结构，是分布式应用系统中的组件。一个完整的基于 EJB 的分布式计算结构由 6 个角色组成，这 6 个角色可以由不同的开发商提供，每个角色所做的工作必须遵循 Sun 公司提供的 EJB 规范，以保证彼此之间的兼容性。这 6 个角色分别是 EJB 组件开发者(Enterprise Bean Provider)、应用组合者(Application Assembler)、部署者(Deployer)、EJB 服务器提供者(EJB Server Provider)、EJB 容器提供者(EJB Container Provider)和系统管理员(System Administrator)。不同的开发角色负责不同的开发任务。

1. EJB 组件开发者

EJB 组件的开发者负责开发执行商业逻辑规则的 EJB 组件,开发出的 EJB 组件打包成 ejb-jar 文件。EJB 组件开发者负责定义 EJB 的 remote 和 home 接口,编写执行商业逻辑功能的 EJB class,提供部署 EJB 的部署文件(deployment descriptor)。部署文件包含 EJB 的名字和 EJB 用到的资源配置,如 JDBC 等。EJB 组件开发者是典型的商业应用开发领域专家。

2. 应用组合者

应用组合者负责利用各种 EJB 组合成一个完整的应用系统。应用组合者有时需要提供一些相关的程序,如在一个电子商务系统里,应用组合者需要提供 JSP 程序。

3. 部署者

部署者负责将包含 EJB 组件的 ejb-jar 文件部署到应用服务器中。系统环境包含某种 EJB Server 和 EJB Container。部署者必须保证所有由 EJB 组件开发者在部署文件中声明的资源可用,例如,部署者必须配置好 EJB 所需的数据库资源。部署过程分两步:部署者首先利用 EJB Container 提供的工具生成一些类和接口,使 EJB Container 能够利用这些类和接口在运行状态管理 EJB;部署者安装 EJB 组件和其他在上一步生成的类到 EJB Container 中。部署者是某个 EJB 运行环境的专家。在某些情况下,部署者在部署时还需要了解 EJB 包含的业务方法,以便在部署完成后,进行简单的程序测试。

4. EJB 服务器提供者

EJB 服务器提供者是系统领域的专家,精通分布式交易管理,分布式对象管理及其他系统级的服务。EJB 服务器提供者一般由操作系统开发商、中间件开发商、数据库开发商担任。在目前的 EJB 规范中,假定 EJB 服务器提供者和 EJB 容器提供者来自同一个开发商,所以没有定义 EJB 服务器提供者和 EJB 容器提供者之间的接口标准。

5. EJB 容器提供者

EJB 容器提供者提供 EJB 部署工具为部署好的 EJB 组件提供运行环境。EJB 容器负责为 EJB 提供交易管理、安全管理等服务。EJB 容器提供者和 EJB 服务器提供者一般由相同的厂商担任,提供的产品叫应用服务器。EJB 容器提供者必须是系统级的编程专家,还要具备一些应用领域的经验。EJB 容器提供者的工作主要集中在开发一个可伸缩的、具有交易管理功能的、集成在 EJB 服务器中的容器。EJB 容器提供者为 EJB 组件开发者提供了一组标准的、易用的 API 访问 EJB 容器,使 EJB 组件开发者不需要了解 EJB 服务器中的各种技术细节。EJB 容器提供者负责提供系统监测工具,用来实时监测 EJB 容器和运行在容器中的 EJB 组件状态。

6. 系统管理员

系统管理员负责为 EJB 服务器和容器提供一个企业级的计算和网络环境。系统管理者负责利用 EJB 服务器和容器提供的监测工具监测 EJB 组件的运行情况。

3.4.3　EJB 的类型

1. 会话 Bean

在企业级应用系统内，会话 Bean 是一种代表客户程序执行操作的 EJB。对于 EJB 客户程序，会话 Bean 常常起着入口点或“前线”EJB 的作用。EJB 客户程序通过与会话 Bean 的交互，从企业应用系统获取他们想要利用的功能或服务。

会话 Bean 分为无状态和有状态两种。无状态会话 Bean 不能够维持一个调用客户的状态。在一个方法调用的过程中，无状态会话 Bean 可以维持调用客户的状态，但是，当方法执行完时，状态不会被保持。在调用完成后，无状态会话 Bean 被立即释放到缓冲池中，所以，无状态会话 Bean 具有很好的伸缩性，可以支持大量用户的调用。

有状态会话 Bean 可以一对一地维持某个调用客户的状态，并且在不同的方法调用中维持这个状态，由于对于每一个并发用户，必须有一个对应的有状态会话 Bean，为了提高系统的效率，有状态会话 Bean 可以在一定的客户空闲时间后被写入二级存储设备（如硬盘），在客户发出新的调用请求后，再从二级存储设备恢复到内存中。

显然，在多用户并发执行条件下，无状态会话 Bean 运行效率要高于有状态会话 Bean，因为在有状态会话 Bean 中要保持会话状态。

2. 实体 Bean

实体 Bean 代表数据库或另外一个企业应用系统中的数据对象，如代表数据库的一行记录。与 Session Bean 不同，实体 Bean 是持久的，允许共享访问。持久性是指实体 Bean 的状态不依赖于应用服务器而存在。因为实体 Bean 是低层数据库记录的影像，会和数据库记录保持同步，所以即使应用服务器崩溃或停止运行，实体 Bean 的状态还会保存在数据库中，不会丢失。

最常用的实体 Bean 是代表关系数据库中的数据。一个简单的实体 Bean 可以代表数据库表中的一个记录，也就是每一个实例代表一个特殊的记录。更复杂的实体 Bean 可以代表数据库表间关联视图。在实体 Bean 中还可以考虑包含厂商的增强功能，如对象-关系影射的集成。

按持久性划分，实体 Bean 分为 Bean 管理持久化和容器管理持久化两种。Bean 管理持久化是指由 Bean 开发者自己管理 Bean 和它所代表的数据库记录的同步；容器管理持久化指由容器自动管理 Bean 和它所代表的数据库记录的同步，不需要开发者编写 SQL 语句。

3. 消息驱动 Bean

消息驱动 Bean 是 EJB 2.0 规范中添加的一种全新的企业级 Bean 类型。消息 Bean 体现出 JMS(Java Message Service)与 EJB 的集成，创造出一种全新的 Bean 类型，它用来处理异步的 JMS 消息。这种全新的 Bean 为 JMS 客户机提供一种组件模型，允许将它们部署到 EJB 容器系统的丰富而强健的环境中去。

消息 Bean 并不直接与客户交互，相反，消息 Bean 是 JMS 消息监听器。一般是客户把消息发布给 JMS 目的地，然后，JMS 提供者和 EJB 容器协作，把消息发送给消息驱动的 Bean。

3.4.4 EJB开发步骤

EJB的开发分为服务器端的开发和客户端的开发。

1. 服务器端的开发

下面是服务器端开发的主要步骤。

(1) 设计特定业务接口。包括创建客户端远程接口和本地接口,并实现EJB组件模型所需要的标准接口。比如,对于会话Bean和实体Bean,需要定义3个类:Bean的远程接口(Remote)、本地接口(Home)和Bean的实现类。

Bean Remote接口定义了EJB的业务方法,EJB所有的业务方法必须在Remote接口中定义才能被客户端访问。Bean Home接口定义了一些方法以供EJB客户端创建、移动和查找EJB对象。Bean实现类用于实现在Bean Remote接口中定义的业务方法。

(2) 配置EJB的部署描述器。在WebLogic Server环境下,需要配置的部署描述器主要是ejb-jar.xml和weblogic-ejb-jar.xml。

(3) 部署应用程序。将EJB模块打包成J2EE JAR包或作为J2EE EAR的子包,发布到EJB应用服务器中。

2. 客户端的开发

客户端可以是Servlet、JSP、应用程序或其他Bean。开发步骤与服务器端相似,但配置、打包与部署是可选的,如客户端是应用程序时就不需要进行。客户端通过JNDI来查找EJB Home接口,然后用Bean Home接口来创建、查找Bean实例,从而完成业务操作。

3. 简单实例

1) EJB服务器端的开发

(1) 定义远程接口。远程接口用于规定客户机与EJB之间的约定,即列出EJB供客户机使用的业务逻辑方法。这种接口通常由EJB提供者设计,而具体的实现则一般由EJB开发者进行。定义远程接口必须强制性地扩展标准的EJBObject接口,EJBObject接口和后面的EJBHome接口实际上都是java.rmi.Remote接口的扩展。

```
package helloejb;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;
import java.math.*;
public interface HelloEJBRemote extends EJBObject
{
  public String HelloWorld() throws RemoteException;
  public BigDecimal dollarToRMB(BigDeciaml dollars) throws RemoteException;
  public BigDecimal poundToRMB(BigDecimal pound) throws RemoteException;
}
```

(2) 定义本地接口。本地接口是操纵EJB的工具,客户机使用本地接口创建、找出和删除EJB实例。定义本地接口必须强制性地扩展标准的EJBHome接口。

```
package helloejb;
import java.io.Serializable;
```

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
public interface HelloEJBHome extends EJBHome
{
  HelloEJBRemote create() throws RemoteException,CreateException;
}
```

(3) EJB 实现类。EJB 实现类实现在远程接口中定义的业务逻辑方法。

```
package helloejb;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.math.*;
public class HelloEJBBean implements SessionBean
{ private SessionContext ctx;
  BigDecimal dollarRate = new BigDeciaml("8.2400");
  BigDecimal poundRate = new BigDeciaml("12.5000");

  public void setSessionContext(SessionContext sc)
    { ctx = sc; //由 EJB 容器自动调用该方法,在命名/目录中进行 RMI 注册
    }
  public String HelloWorld() { return "Hello World,Welcome learn EJB!"; }
  public BigDecimal dollarToRMB(BigDeciaml dollars)
  { BigDecimal result = dollars.multiply(dollarRate);
    return result.setScale(2,BigDecimal.ROUND_UP);
  }
  public BigDecimal poundToRMB(BigDecimal pound)
  { BigDecimal result = pound.multiply(poundRate);
    return result.setScale(2,BigDecimal.ROUND_UP);
  }

  public HelloEJBBean() {}
  public void ejbCreate() {}   //对应 Home 接口中的 create()方法,用于创建 EJB 实例
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void ejbPassivate(){}
}
```

除了构造器 HelloEJBBean()和 ejbCreate()方法之外,其余方法都是从 Javax.ejb.SessionBean 接口继承来的。

(4) 部署 EJB。首先,需要编写部署描述器文件,对于 WebLogic 应用服务器,分别为 ejb-jar.xml 和 weblogic-ejb-jar.xml。ejb-jar.xml 部署描述器告诉 Weblogic EJB 容器,EJB 的本地接口、远程接口和 Bean 类的名字。weblogic-ejb-jar.xml 用于配置 WebLogic 的特定参数。

然后,建立 EJB 档案文件,并进行下列部署。

```
sessionejb\helloejb\HelloEJBRemote.class
sessionejb\helloejb\HelloEJBHome.class
sessionejb\helloejb\HelloEJBBean.class
```

```
sessionejb\META-INF\ejb-jar.xml
sessionejb\META-INF\weblogic-ejb-jar.xml
```

EJB档案文件是通过Weblogic Server控制台上进行部署。

2) EJB客户端开发

对EJB应用系统来说,主要有两种类型的客户端:应用客户端和Web浏览器客户端。开发EJB Web浏览器客户端程序同开发一般的Web客户程序没有什么区别,只是它要实现EJB客户程序所要求的功能。以Servlet程序开发为例,给出一个EJB Servlet客户端程序。

```
import helloejb.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.math.BigDecimal;
import java.rmi.RemoteException;
import java.util.Properties;
import javax.ejb.CreateException;
import javax.ejb.RemoveException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class HelloEJBServlet extends HttpServlet
{ private static final String JNDI_NAME = "HelloEJB";
  private static final String url = "t3://localhost: 7001";
  public void init(ServletConfig config) throws ServletException
    { super.init(config); //set self environment variable }
  public void service(HttpServletRequest req.HttpServletResponse resp) throws IOException
    { resp.setContentType("text/html");
      PrintWriter out = resp.getWriter();
      try {
          //get an InitialContext
          Properties env = new Properties();

env.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
          env.put(Context.PROVIDER_URL,url);

          Context ctx = new InitialContext(env);
          Object h = ctx.lookup(JNDI_NAME);
HelloEJBHome home = (HelloEJBHome)PortableRemoteObject.narrow(h,HelloEJBHome.class);
          HelloEJBRemote currencyHelloWorld = home.create();
          String hello = currencyHelloWorld.HelloWorld();

          out.println("<html><head><title>Hello World!</title></head><body>");
          out.println("<h4>My EJB said: " + hello + "</h4>");
          BigDecimal param = new BigDecimal("100.00");
          BigDecimal amount = currencyHelloWorld.dollarToRMB(param);
          out.println("<h4>100 dollar = " + amount + "RMB</h4>");
```

```
            out.println("</body></html>");
          } catch(Exception ex) {
            System.err.println("Caught an unexpected exception!");
            ex.printStackTrace();
          }
        }
    }
```

最后需要对这个 Servlet 程序进行打包部署，其档案文件结构如下。

```
sessionejb\WEB-INF\web.xml
sessionejb\WEB-INF\weblogic.xml
sessionejb\WEB-INF\ classes\HelloEJBServlet
sessionejb\WEB-INF\classes\helloejb\HelloEJBRemote.class
sessionejb\WEB-INF\classes\helloejb\HelloEJBHome.class
```

3.4.5　EJB 环境和资源

几乎每一种开发环境都有从外部设置或获取环境变量的手段。J2EE 开发环境也不例外。在 J2EE 中，主要是在其部署描述器中设置环境变量，在运行时再从程序中获取。

1. EJB 环境变量

EJB 环境变量的值在 ejb-jar.xml 文件中定义。通过＜env-entry＞标记符进行声明，其中包括描述信息、变量名、数据类型和值。例如：

```
<env-entry>
  <description>stock EastAir's price is 20.0 RMB</description>
  <env-entry-name>EastAir</env-entry-name>
  <env-entry-type>java.lang.float</env-entry-type>
  <env-entry-value>20.0</env-entry-value>
</env-entry>
```

在部署和运行 EJB 中，容器将在“java：comp/env”处创建一个 JNDI 上下文环境，在该环境中，可以查找获得在 ejb-jar.xml 中定义的变量。这是通过在程序中用 JNDI 的 lookup()方法获得，例如：

```
Context ctx = new InitialContext();
float EastAir = (float) ctx.lookup("java: comp/env/EastAir");
```

2. 资源管理器引用

当在 EJB 中使用数据库等资源时，传统的做法是在程序中定义这些资源，这样做的缺点是削弱了程序的可移植性，增加了编码的复杂性。值得庆幸的是，EJB 规范也提供了在部署描述器中定义资源的方法，在程序中对这些资源可以进行引用。EJB 中常用的资源主要有数据库资源、邮件资源、URL 资源。

EJB 可以通过数据库资源引用来访问数据库。EJB 使用数据库资源引用来访问数据库，服务器将负责连接指定的数据库，并管理数据库连接池，应用系统可以通过配置在各种不同的数据库之间进行随意的移植。例如：

```
<resource-ref>
  <res-ref-name>jdbc/orderdbPool</res-ref-name>
  <res-type>javax.sql.Dataource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

这里定义了一个名字为“jadc/orderdbPool”的数据库资源引用(根据EJB规范,所有的数据库资源引用的名称都是以jdbc/为前缀)。定义数据库引用,<res-type>值总是javax.sql.DataSource;<res-auth>的值总是Container。然后,在weblogic-ejb-jar.xml中用下列代码把jdbc/orderdbPool映射到JDBC名字DBPool。

```
<ejb-reference-description>
  <ejb-ref-name>jdbc/orderdbPool</ejb-ref-name>
  <jndi-name>DBPool</jndi-name>
<ejb-reference-description>
```

在程序中,通过JDNI来访问数据库引用,例如:

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java: comp/env/jdbc/orderdbPool");
Connection con = ds.getConnection();
```

最后,在程序运行之前,通常还需在应用服务器的config.xml文件中进行相关的配置。

同样,EJB可以通过对邮件资源的引用来发送或接收电子邮件。

```
<resource-ref>
  <res-ref-name>mail/demo-smtp</res-ref-name>
  <res-type>javax.mail.Session</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

在weblogic-ejb-jar.xml中,用下列代码将“mail/demo-smtp”映射到JDBC名字mailSmtp。

```
<ejb-reference-description>
  <ejb-ref-name>mail/demo-smtp</ejb-ref-name>
  <jndi-name>mailSmtp</jndi-name>
<ejb-reference-description>
```

在程序中,EJB类通过JNDI来访问邮件资源引用,例如:

```
Context ctx = new InitialContext();
Session session = (Session)ctx.lookup("java: comp/env/mail/demo-smtp");
```

3.5 面向服务的系统开发

3.5.1 面向服务的系统架构及其特征

信息系统(包括电子商务系统)的软件开发,从最初的面向过程、面向对象,到后来的面

向组件、面向集成，直到最近的面向服务，走过了一条螺旋上升的曲线。自 20 世纪 70 年代出现“软件危机”以来，它一直按照软件工程的方法，从需求分析开始，经过设计、实现、测试、运行，最后才算完成。即便如此，由于需求不断地变化，甚至在最后运行阶段还会发生，软件开发变得异常复杂；无论采用瀑布型分析方法或原型分析方法，也无论采用面向对象甚至面向组件的设计与实现方法，这种情况没有得到根本好转。为了彻底摆脱这种状况，人们进行了不懈的努力，面向服务架构（Service-Oriented Architecture，SOA）的提出，正是这种努力的反映。

SOA 并不是一个新概念，有人就将 CORBA 和 DCOM 等组件模型看成 SOA 架构的前身。早在 1996 年，Gartner Group 就已经提出了 SOA 的预言。不过那个时候仅仅是一个“预言”，当时的软件发展水平和信息化程度还不足以支撑这样的概念走进实质性应用阶段。只是到了最近，SOA 实现的技术手段才渐渐成熟。关于 SOA，目前尚未有一个统一的、业界广泛接受的定义。一般认为 SOA 是一个组件模型，它将应用程序的不同功能单元即服务（service），通过服务间所定义的接口和契约（contract）联系起来。接口采用中立的方式定义，独立于具体实现服务的硬件平台、操作系统和编程语言，使得构建在这样的系统中的服务可以使用统一和标准的方式进行通信。这种具有中立接口的定义（没有强制绑定到特定的实现上）的特征被称为服务之间的松耦合。从这里可以看出：

（1）它是一种软件系统架构。SOA 不是一种语言，也不是一种具体的技术，更不是一种产品，而是一种软件系统架构。它尝试给出在特定环境下推荐采用的一种框架结构，从这个角度上来说，它其实更像一种架构模式（pattern），是一种理念架构。

（2）服务是整个 SOA 实现的核心。SOA 架构的基本元素是服务，SOA 指定一组实体（服务提供者、服务消费者、服务注册表、服务条款、服务代理和服务契约），这些实体详细说明了如何提供和消费服务。遵循 SOA 观点的系统必须要有服务，这些服务是可互操作的、独立的、模块化的、位置明确的、松耦合的，并且可以通过网络查找其地址。

图 3-6 给出的 SOA 模型中 3 种不同角色之间的关系。其中，服务是一个自包含的、无状态（stateless）[①]的实体，可以由多个组件组成。它通过事先定义的界面响应服务请求，也可以执行诸如编辑和处理事务（transaction）等离散性任务。服务本身并不依赖于其他函数和过程的状态。用什么技术实现服务，其定义中未加限制，可以用各种合适的技术来实现。服务提供者（service provider）提供符合契约的服务，并将它们发布到服务代理。服务请求者（service consumer）也叫服务使用者，它发现并调用服务提供者提供的服务来实现商业解决方案。服务请求者通常为客户端。服务代理者（service broker）作为储存库、电话黄页或票据交换所，产生由服务提供者发布的软件接口。

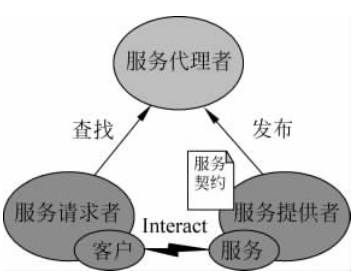

图 3-6　SOA 模型中的 3 种角色

① 所谓服务的无状态，是指服务不依赖于任何事先设定的条件，是状态无关的（state-free）。在 SOA 架构中，一个服务不会依赖于其他服务的状态，它们从客户端接受服务请求。因为服务是无状态的，它们可以被编排（orchestrated）和序列化（sequenced）成多个序列（有时还采用流水线机制），以执行商业逻辑。这里，编排是指序列化服务并提供数据处理逻辑，但不包括数据的展现功能。

服务提供者、服务代理者以及服务请求者通过3个基本操作：发布(publish)、查找(find)、绑定(bind)来相互作用。服务提供者向服务代理者发布服务，服务请求者通过服务代理者查找到所需的服务并绑定到这些服务上，服务提供者和服务请求者之间通过绑定的服务进行交互。

SOA具有如下的一些特征。

(1) 服务的封装。将服务封装成用于业务流程的可重用组件的应用程序函数。它提供信息或简化业务数据从一个有效的、一致的状态向另一个状态的转变。封装隐藏了复杂性。服务的API保持不变，使得用户远离具体实施上的变更。

(2) 服务的重用。服务的可重用性设计显著地降低了成本。为了实现可重用性，服务只工作在特定处理过程的上下文中，独立于底层实现和客户需求的变更。

(3) 服务的互操作。互操作并不是一个新概念，在CORBA、DCOM、Web Service中就已经采用了互操作技术。在SOA中，通过服务之间既定的通信协议进行互操作。主要有同步和异步两种通信机制。SOA提供服务的互操作特性更有利于其在多种场合被重用。

(4) 服务是自治的(autonomous)功能实体。服务是由组件组成的组合模块，是自包含和模块化的。SOA非常强调架构中提供服务的功能实体的完全独立自主的能力。传统的组件技术，如.NET Remoting、EJB、COM或者CORBA，都需要有一个宿主(host或者server)来存放和管理这些功能实体；当这些宿主运行结束时，这些组件的寿命也随之结束，这样当宿主本身或者其他功能部分出现问题的时候，在该宿主上运行的其他应用服务就会受到影响；而SOA架构中非常强调实体自我管理和恢复能力，常见的用来进行自我恢复的技术，比如事务处理(transaction)、消息队列(message queue)、冗余部署(redundant deployment)和集群系统(cluster)等在SOA中都起到至关重要的作用。

(5) 服务之间的松耦合度(loosly coupled)。服务请求者到服务提供者的绑定与服务之间应该是松耦合的。这就意味着，服务请求者不知道提供者实现的技术细节，比如程序设计语言、部署平台等。服务请求者往往通过消息调用操作，请求消息和响应，而不是通过使用API和文件格式。

(6) 服务是位置透明的(location transparency)。服务是针对业务需求设计的，需要反映需求的变化，即所谓敏捷(agility)设计。要想真正实现业务与服务的分离，就必须使得服务的设计和部署对用户来说是完全透明的。也就是说，用户完全不必知道响应自己需求的服务的位置，甚至不必知道具体是哪个服务参与了响应。

3.5.2 Web Service

Web服务(Web Service)是目前用来实现SOA的技术集合。其所涉及的技术主要包括以下几个方面。

1. 可扩展标记语言(Extensible Markup Language，XML)

XML 1.0标准是一个基于文本的W3C(World Wide Web)组织规范的标记语言。与HTML使用标签来描述外观和数据不同，XML严格地定义了可移植的结构化数据。它是一种元语言，可用来作为数据描述语言的语言，如标记语法或词汇、交换格式和通信协议。

2. 简单对象访问协议

SOAP是一个基于XML的、用于在分布式环境下交换信息的轻量级协议。SOAP在

请求者和提供者对象之间定义了一个通信协议，这样，在面向对象编程流行的环境中，该请求对象可以在提供的对象上执行远程方法调用。因为 SOAP 是与平台和厂商无关的标准，对于松耦合系统的互操作，SOAP 是支持服务调用的最好方法。在 W3C SOAP 1.2 规范中，规定服务请求者和服务提供者之间使用 XML 格式的消息来进行通信；也就是说，请求者将应用程序请求（在 XML 中）放入 SOAP 信封中（也是 XML），并通过网络发送给提供者，提供者发回的响应也采用这种形式，例如：

```
POST /test/eb1/Service1.asmx HTTP/1.1
Host: localhost
Content-Length: length
SOAPAction: http://tempuri.org/eb1/Service1/getName

<?xml version = "1.0" encoding = "utf-8"?>
< soap: Envelope xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
  < soap: Body >
    < getName xmlns = "http://tempuri.org/eb1/Service1">
      < id > int </id >
    </getName >
  </soap: Body >
</soap: Envelope >
```

3. Web Service 描述语言

WSDL 定义了一个 XML 词汇表，该词汇表依照请求消息和响应消息，在服务请求者和服务提供者之间定义一种契约。WSDL 是一种 XML 文档，在应用开发环境中使用 WSDL 将集成服务的流程自动处理到请求者应用程序。例如，WebSphere Studio 产生一个 Java 的代理对象，它能够像本地对象一样实现服务，但是实际上代理对象仅仅处理请求的创建和响应消息的解析。不管服务是否用 Java、C＃或者其他的语言实现，生成的 Java 代理对象都能够从 WSDL 描述中调用任何的 Web Service。实际上，WSDL 不能像编程语言那样描述实现细节。

4. 统一描述、发现和集成

UDDI 为发布服务的可用性和发现所需服务定义了一个标准接口（基于 SOAP 消息）。UDDI 实现将发布和发现服务的 SOAP 请求解释为用于基本数据存储的数据管理功能调用。

需要说明的是，尽管 Web Service 可用于实现 SOA，但它们是不同的两个概念。SOA 是独立于实现技术的，是抽象的一个概念，不一定非用 Web Service 来实现，也可用其他技术来实现；反之，Web Service 也并不一定要遵循 SOA 标准。另外，Web Service 与组件也不完全一样，Web Service 和组件尽管都有一个或多个接口，并且服务使用者或组件使用者都得遵守这些接口，但服务是关于模式（schemas）的，组件对象是关于对象类型（object types）的，服务通过像 SOAP 这样的标准消息机制（messages）来实现通信，而组件对象通过方法调用（method calls）来交互。与 CORBA 中的接口定义语言（Interface Definition Language，IDL）相比，Web Service 在 WSDL 中采用 XML，显得更加普遍和通用。服务与组件的不同点见表 3-6。

表 3-6 服务与组件的差异

分布式组件架构	面向服务的架构
面向功能	面向流程
设计目的是为了实现需求	设计目的是为了适应变化
开发周期长	交互式和重用性开发
成本为中心	业务为中心
应用阻塞	服务协调
紧密耦合	敏捷的和松耦合的
同构技术	异构技术
面向对象	面向消息
需深入了解实施细节	独立于实施细节

3.5.3 面向 Web Service 的系统开发方法[①]

利用 Web Service 可以在网络上建立分布式的电子商务系统,系统中的服务可通过任何平台、任何语言和以任何方式来访问。以前很少使用 Java 语言来实现 Web Service,原因是太困难了,但随着新一代 Web Service 引擎 XFire 的发布,这种情况不复存在。使用 XFire,可以直接把 Java 类中的方法发布为 Web Service,而不需要编写额外的代码。XFire 是与 Axis 2 并列的新一代 Web Service 框架,它具有下列特点。

(1) 支持一系列 Web Service 的新标准,如 JSR181、WSDL 2.0、JAXB2、WS-Security 等。

(2) 使用 Stax 来解释 XML,性能有了质的提高。

(3) 容易建立服务。

(4) 易于与 Spring 框架结合。

(5) 灵活的 Binding 机制,可包括默认的 Aegis、xmlbeans、jaxb2 和 castor。下面以 Eclipse 3.2+Tomcat 5.5.9+XFire 1.2.6 开发环境为例,介绍一下如何利用 XFire 开发 Web Service 应用。具体过程如下。

(1) 配置 XFire 1.2.6。

① 下载 XFire 1.2.6。从网址"http://xfire.codehaus.org"下载 xfire-distribution-1.2.6.zip 文件,解压缩后得到的目录结构如图 3-7 所示。

图 3-7 xfire-distribution-1.2.6.zip 解压目录

① 参见 http://tb.blog.csdn.net/TrackBack.aspx?PostId=1778085。

这里，api 目录下主要是 javadoc 文档资料；examples 目录下是 xfire 自带的例子程序；lib 目录下是 xfire 所需的 jar 文件；manual 目录下是 xfire 模块；xfire-all-1.2.6.jar 文件是 xfire 提供的整体 jar 包。

② 在 Tomcat 应用服务器下面配置 Xfire 环境(假设已预先安装好 Tomcat 5.5.9 和 JDK 1.5)。先在 Tomcat 下按照如下的目录结构新建相关的文件或者文件夹。

```
webapp
  |--xfire
    |--WEB-INF
      |--lib
      |--web.xml
      |--classes
        |--META-INF
          |--xfire
            |--services.xml
```

然后将解压缩后 XFire 中的内容按照下面的要求复制到对应的目录下。

将“xfire-1.2.6\lib”下面的内容复制到“jakarta-tomcat-5.5.9\webapps\xfire\WEB-INF\lib”目录中；将“xfire-1.2.6\xfire-all-1.2.6.jar”复制到“jakarta-tomcat-5.5.9\webapps\xfire\WEB-INF\lib”目录中。接下来将 web.xml 文件内容设置如下。

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!-- START SNIPPET: webxml -->
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems,Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>XFireServlet</servlet-name>
    <display-name>XFire Servlet</display-name>
    <servlet-class>
        org.codehaus.xfire.transport.http.XFireConfigurableServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/servlet/XFireServlet/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
</web-app>
<!-- END SNIPPET: webxml -->
```

将 services.xml 文件内容设置如下。

```
<!-- START SNIPPET: services -->
  <beans xmlns = "http://xfire.codehaus.org/config/1.0">
```

```
  </beans>
  <!-- END SNIPPET: services -->
```

③ 配置好上面的内容后,启动 Tomcat 应用服务器;打开 IE 浏览器;访问如下的网址:

```
http://localhost: 8080/xfire/services/
```

如能正确地显示成功页面,说明已成功配置了 XFire。

(2) 基于 xfire 开发 Web Service。

① 在 Eclipse 中新建项目 XFireProject。

建立一个源文件夹 src. main,再新建一个 HelloService. java 文件,在该文件中可以只简单地声明一个 sayHello(String name)方法。HelloService. java 文件的具体内容如下。

```
package com. liuxiang. xfire;
/**
   * 简单例子 HelloWorld 例子
   * HelloService. java
   * com. liuxiang. xfire
   * XFireProject
 */
public class HelloService {
     public String sayHello(String name){
         return name + ",你好!";
     }
}
```

② 在 src. main 目录下新建 META-INF/xfire/services. xml 文件,该文件用于声明一个 service。service. xml 文件的内容如下。

```
<!-- START SNIPPET: services -->
<beans xmlns = "http://xfire.codehaus.org/config/1.0">
  <service>
    <name>HelloService</name>
      <namespace>http://com.liuxiang.xfireDemo/HelloService</namespace>
      <serviceClass>com.liuxiang.xfire.HelloService</serviceClass>
    </service>
</beans>
<!-- END SNIPPET: services -->
```

③ 将编译后的 HelloService. class 文件和 service. xml 文件部署到 Tomcat 中。文件的位置如下。

```
webapps\xfire\WEB-INF\classes\META-INF\xfire\services.xml;
webapps\xfire\WEB-INF\classes\com\liuxiang\xfire\HelloService.class;
```

④ 启动 Tomcat。正确启动 Tomcat 之后,在 IE 地址栏里输入“http://localhost:8080/xfire/services/”,将会出现图 3-8 所示的页面,该页面正常显示了刚才部署的 HelloService。单击该页面的链接[wsdl],IE 地址栏里将会出现“http://localhost:8080/xfire/services/HelloService? wsdl”,生成一个 wsdl 文件,IE 窗口中出现该 wsdl 文件的内容。

图 3-8　部署的 HelloService

⑤ 生成 Web Service 客户端调用的文件。XFire 提供了两种生成客户端代码的测试方式，一种是提供了 ant 脚本，另一种是提供了 XFire 的 Eclipse 插件；下面介绍使用 ant 脚本生成客户端代码的方法。

xFire 提供了一个 ant 任务，内容为“<taskdef name="wsgen" classname="org.codehaus.xfire.gen.WsGenTask" classpathref="myclasspath" />”。在项目 XFireProject 中增加一个 build.xml 文件，build.xml 文件的内容如下。

```
<?xml version="1.0"?>
<project name="XFireProject" default="genfiles" basedir=".">
    <property name="lib" value="lib" />
    <path id="myclasspath">
        <fileset dir="${lib}">
            <include name="*.jar" />
        </fileset>
        <pathelement location="${genfiles}" />
    </path>
    <!--通过 XFire ant 任务生成客户端代码的存放位置-->
    <property name="code_path" value="src.client" />
    <!--需要生成客户端代码的 wsdl 文件-->
    <property name="wsdl_path" value="http://localhost:8080/xfire/services/HelloService?
wsdl" />
    <!--生成客户端代码的包名-->
    <property name="code_package" value="com.liuxiang.xfire.client" />

    <!-- Remove classes directory for clean build -->
    <target name="clean" description="Prepare for clean build">
        <delete dir="${code_path}"/>
        <mkdir dir="${code_path}"/>
    </target>

    <!--<target name="genfiles" depends="clean" description="Generate the files">-->
    <target name="genfiles" description="Generate the files">
    <taskdef name="wsgen" classname="org.codehaus.xfire.gen.WsGenTask" classpathref=
"myclasspath" />
        <!--outputDirectory 属性定义创建的代码所在的文件夹
            wsdl 是 Web Service 的 wsdl 文件
            package 代表创建的代码的 package
        -->
```

```
        <wsgen outputDirectory="${code_path}" wsdl="${wsdl_path}" package="${code_
package}" binding="xmlbeans" />
    </target>
</project>
```

执行 ant 脚本,将会生成客户端代码,共 3 个文件,放在包 com.liuxiang.xfire.client 下面,文件分别是 HelloServiceClient.java、HelloServiceImpl.java 和 HelloServicePortType.java。

⑥ 编写测试代码,通过调用④中生成的代码,编写 TestClient.java 文件。文件内容如下。

```
package com.liuxiang.xfire;
import java.net.MalformedURLException;
import org.codehaus.xfire.XFire;
import org.codehaus.xfire.XFireFactory;
import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;
import com.liuxiang.xfire.client.HelloServiceClient;
import com.liuxiang.xfire.client.HelloServicePortType;
  public class TestClient {
    public static String testClient(String name){
        HelloServiceClient helloSC = new HelloServiceClient();
        HelloServicePortType helloSP = helloSC.getHelloServiceHttpPort();
        String result = helloSP.sayHello(name);
        return result;
    }
    public static void main(String[] args) throws Exception {
        System.out.println(testClient("Liuxiang"));
    }
  }
```

运行该代码,在控制台会输出信息:“Liuxiang,你好!”,表明第一个 Web Service 用例已经运行成功了。

3.5.4 WSDL 内容分析

WSDL 是一个用于精确描述 Web Service 的文档,WSDL 文档是一个遵循 WSDL XML 模式的 XML 文档。WSDL 文档将 Web Service 定义为服务访问点或端口的集合。在 WSDL 中,由于服务访问点和消息的抽象定义已从具体的服务部署或数据格式绑定中分离出来,因此可以对抽象定义进行再次使用:消息,指对交换数据的抽象描述;而端口类型,指操作的抽象集合。用于特定端口类型的具体协议和数据格式规范构成了可以再次使用的绑定。将 Web 访问地址与可再次使用的绑定相关联,可以定义一个端口,而端口的集合则定义为服务。

1. WSDL 文档组成

一个 WSDL 文档通常包含 7 个重要的元素,即 types、import、message、portType、operation、binding 和 service。这些元素嵌套在 definitions 元素中,definitions 是 WSDL 文

档的根元素。types 元素是数据类型定义的容器，它使用某种类型系统（一般地使用 XML Schema 中的类型系统）；message 元素是通信消息的数据结构的抽象类型化定义，使用 types 所定义的类型来定义整个消息的数据结构；operation 元素是对服务中所支持的操作的抽象描述，一般单个 operation 描述了一个访问入口的请求/响应消息对；portType 元素是对某个访问入口点类型所支持的操作的抽象集合，这些操作可以由一个或多个服务访问点来支持；binding 元素是特定端口类型的具体协议和数据格式规范的绑定；port 元素是定义为协议/数据格式绑定与具体 Web 访问地址组合的单个服务访问点；service 元素是相关服务访问点的集合。可以参考图 3-9 来理解一下 WSDL 的文档结构。

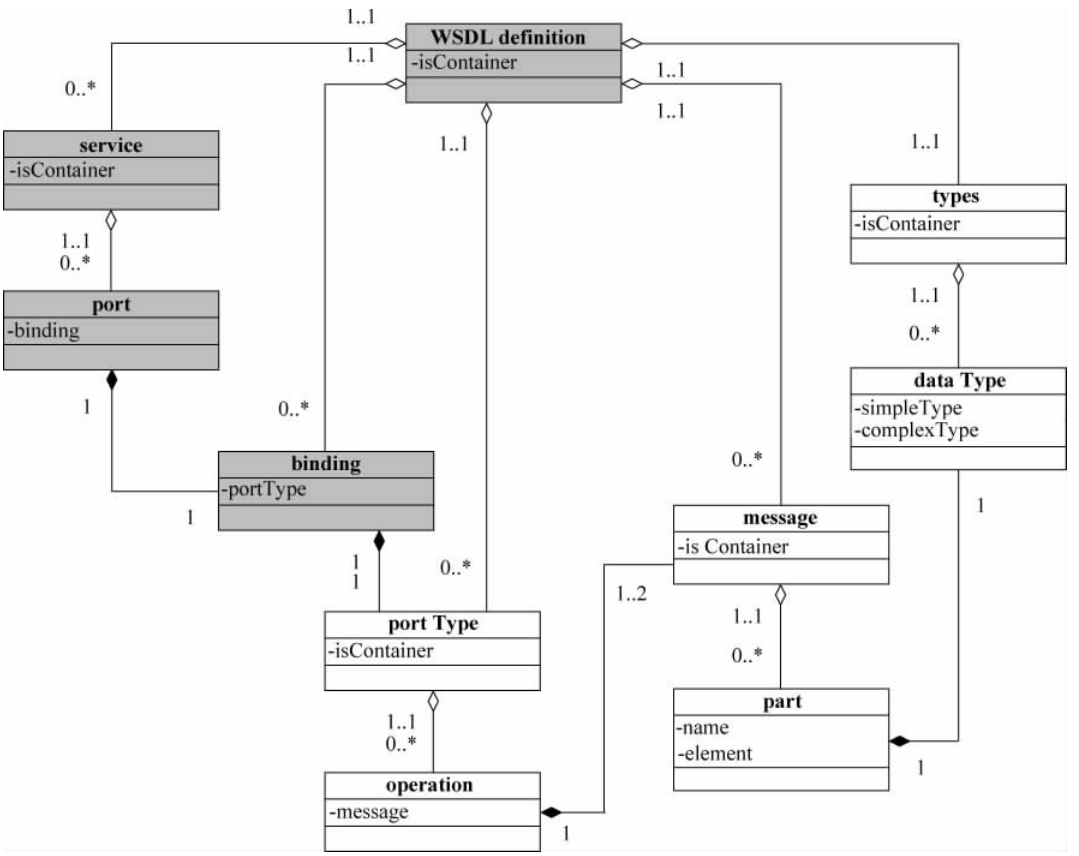

图 3-9　WSDL 的文档结构

有关 WSDL 的 XML Schema，可以参考“http://schemas.xmlsoap.org/wsdl/”。

2. WSDL 的基本结构

下面是前面所产生的一个简单的 WSDL 文档。其服务支持名为 sayHello 的唯一操作，该操作通过在 HTTP 上运行 SOAP 协议来实现的。该请求接受一个字符串 name，经过处理后返回一个简单的字符串。文档内容如下。

```
<?xml version = "1.0" encoding = "UTF-8" ?>
< wsdl: definitions targetNamespace = "http://com.liuxiang.xfireDemo/HelloService"
xmlns: tns = "http://com.liuxiang.xfireDemo/HelloService"
```

```
xmlns: wsdlsoap = "http://schemas.xmlsoap.org/wsdl/soap/"
xmlns: soap12 = "http://www.w3.org/2003/05/soap-envelope"
xmlns: xsd = "http://www.w3.org/2001/XMLSchema"
xmlns: soapenc11 = "http://schemas.xmlsoap.org/soap/encoding/"
xmlns: soapenc12 = "http://www.w3.org/2003/05/soap-encoding"
xmlns: soap11 = "http://schemas.xmlsoap.org/soap/envelope/"
xmlns: wsdl = "http://schemas.xmlsoap.org/wsdl/">
< wsdl: types >
    < xsd: schema xmlns: xsd = http://www.w3.org/2001/XMLSchema
      attributeFormDefault = "qualified" elementFormDefault = "qualified"
      targetNamespace = "http://com.liuxiang.xfireDemo/HelloService">
      < xsd: element name = "sayHello">
        < xsd: complexType >
          < xsd: sequence >
            < xsd: element maxOccurs = "1" minOccurs = "1"
              name = "name" nillable = "true" type = "xsd: string" />
            </xsd: sequence >
          </xsd: complexType >
        </xsd: element >
        < xsd: element name = "sayHelloResponse">
          < xsd: complexType >
            < xsd: sequence >
              < xsd: element maxOccurs = "1" minOccurs = "1"
                  name = "out" nillable = "true" type = "xsd: string" />
                </xsd: sequence >
              </xsd: complexType >
            </xsd: element >
          </xsd: schema >
</wsdl: types >
< wsdl: message name = "sayHelloResponse">
  < wsdl: part name = "parameters" element = "tns: sayHelloResponse" />
</wsdl: message >
< wsdl: message name = "sayHelloRequest">
  < wsdl: part name = "parameters" element = "tns: sayHello" />
</wsdl: message >
< wsdl: portType name = "HelloServicePortType">
  < wsdl: operation name = "sayHello">
    < wsdl: input name = "sayHelloRequest"
          message = "tns: sayHelloRequest" />
    < wsdl: output name = "sayHelloResponse"
          message = "tns: sayHelloResponse" />
    </wsdl: operation >
</wsdl: portType >
< wsdl: binding name = "HelloServiceHttpBinding"
      type = "tns: HelloServicePortType">
      < wsdlsoap: binding style = "document"
          transport = "http://schemas.xmlsoap.org/soap/http" />
       < wsdl: operation name = "sayHello">
          < wsdlsoap: operation soapAction = "" />
```

```
            <wsdl: input name = "sayHelloRequest">
                <wsdlsoap: body use = "literal" />
            </wsdl: input>
            <wsdl: output name = "sayHelloResponse">
                <wsdlsoap: body use = "literal" />
            </wsdl: output>
        </wsdl: operation>
    </wsdl: binding>
    <wsdl: service name = "HelloService">
        <wsdl: port name = "HelloServiceHttpPort"
            binding = "tns: HelloServiceHttpBinding">
            <wsdlsoap: address
              location = "http://localhost: 8080/xfire/services/HelloService" />
        </wsdl: port>
    </wsdl: service>
    </wsdl: definitions>
```

对以上的 WSDL 文档，作一简单说明。

(1) types 元素。使用 XML 模式语言来声明在 WSDL 文档中的其他位置所使用的复杂数据类型与元素。

(2) import 元素。类似于 XML 模式文档中的 import 元素，用于从其他 WSDL 文档中导入 WSDL 定义。

(3) message 元素。使用在 WSDL 文档的 type 元素中定义或在 import 元素引用的外部 WSDL 文档中定义的 XML 模式的内置类型、复杂类型或元素描述了消息的有效负载。

(4) portType 元素和 operation 元素描述了 Web Service 的接口并定义了它的方法。portType 元素和 operation 元素类似于 Java 接口和接口中定义的方法声明。operation 元素使用一个或者多个 message 类型来定义它的输入和输出的有效负载。

(5) binding 元素。将 portType 元素和 operation 元素赋给一个特殊的协议和编码样式。

(6) service 元素。负责将 Internet 地址赋给一个具体的绑定。

(7) definitions 元素。所有的 WSDL 文档的根元素均是 definitions 元素，该元素封装了整个文档，同时通过其 name 提供了一个 WSDL 文档。除了提供一个命名空间外，该元素没有其他作用。

(8) types 元素。WSDL 采用了 W3C XML 模式内置类型作为其基本类型系统。types 元素用作一个容器，用于定义 XML 模式内置类型中没有描述的各种数据类型。当声明消息部分的有效负载时，消息定义使用了在 types 元素中定义的数据类型和元素。在上面的 types 元素部分，定义了两个元素，一个是 sayHello，一个是 sayHelloResponse。sayHello 定义了一个类型，包含一个简单的字符串，用来描述操作的参数传入部分；sayHelloResponse 定义了另一个数据类型，包含一个字符串，用来描述操作的返回值。

(9) import 元素。import 元素使得可以在当前的 WSDL 文档中使用其他 WSDL 文档中指定的命名空间中的定义元素。此例中没有使用 import 元素。

(10) message 元素。message 元素描述了 Web Service 使用消息的有效负载。message 元素可以描述输出或者接受消息的有效负载；还可以描述 SOAP 文件头和错误

detail元素的内容。定义message元素的方式取决于使用RPC样式还是文档消息的传递样式。此例中采用了文档消息的传递样式,定义了两个消息,分别为sayHelloResponse和sayHelloRequest。如果采用RPC样式,只需将文档中的element元素修改为type即可。

(11) portType元素。portType元素定义了Web Service的抽象接口,类似Java的接口。在WSDL中,portType元素由binding和service元素来实现,这两个元素会具体说明Web Service所采用的Internet协议、编码方案以及Internet地址。在一个portType中可以定义多个operation,一个operation可以看作是一个方法。

(12) bindingbinding元素。将一个抽象的portType映射成一组具体的协议(SOAP和HTTP)、消息传递样式、编码样式。

(13) service元素和port元素。service元素包含一个或者多个port元素,每个port元素表示一个不同的Web Service。port元素将URL赋给一个特定的binding,甚至可以使两个或者多个port元素将不同的URL赋值给相同的binding。

3.5.5 Apache开源项目

Apache开源项目的分类列于表3-7。

表3-7 Apache开源项目分类列表

分 类	项目名	说 明	开发语言
服务器(共21个)	Apache	HTTP服务器	C/C++
	Tomcat	Java的Web服务器	Java
	James	邮件服务器	Java
	SpamAssassin	反垃圾邮件	C/C++
	Perl	Apache的Perl编程语言支持	C/C++
	Tcl	TCL脚本语言	C/C++
	Directory Server	超级目录服务器	Java
	Axis	WebService服务器	Java
	XFire	WebService服务器	Java
	Kandula	Axis中WS-Coordination、WS-AtomicTransaction、WS-BusinessActivity协议的实现	Java
	Muse	Axis中WS-ResourceFramework(WSRF)、WS-BaseNotification(WSN)和WS-Distributed-Management(WSDM)标准的实现(该项目Logo是个不认识的古汉字)	Java
	Pubscribe	Web Services Notification(WSN)标准实现	Java
	Sandesha	WS-ReliableMessaging标准实现	Java
	WSS4J	WS-Security标准实现	Java
	WSRF	Web Services Resource Framework标准实现	Java
	Addressing	WebService的WS-Addressing标准(IBM、Microsoft、BEA发布)实现	Java
	XML Security	XML签名与加密标准的Java、C++实现	Java/C++
	jUDDI	UDDI的Java实现	Java

续表

分　类	项目名	说　明	开发语言
服务器(共21个)	XML-RPC	XML-RPC 实现	Java
	Derby	纯 Java 做的关系数据库	Java
	Xindice	XML 数据库	Java
开发工具(共5个)	Ant	自动编译	Java
	Maven	项目管理工具,比 Ant 强大,支持插件开发	Java
	Gump	每日集成工具,支持 Ant、Maven	Java
	JMeter	Web 应用性能测试	Java
	DdlUtils	用 XML 来定义 DDL	Java
Web 开发框架(共19个)	Struts	MVC 的 Web 开发框架	Java
	Cocoon	Web 开发框架,基于可运行的 XML 管道语言	Java
	FOP	XSL-FO 打印与输出解决方案,基于 Java	Java
	AxKit	基于 XML 的 Web 发布	Java
	Tapestry	Web 开发框架	Java
	Turbine	Web 开发框架	Java
	Shale	基于 JSF 的 Web 开发框架	Java
	MyFaces	第一个开源的 JSF 实现	Java
	Beehive	基于 Structs 的 J2EE 框架,简化 J2EE 编程;含 Web 界面、WebService 开发框架	Java
	Velocity	模板引擎	Java
	Portals	门户解决方案	Java
	Cactus	Web 开发测试框架	Java
	Forrest	基于 Cocoon 的 Web 发布解决方案	Java
	Slide	内容管理,支持 WebDAV	Java
	Jackrabbit	内容库,用于内容管理	Java
	Lenya	内容管理,支持版本管理、工作流、所见所得编辑器	Java
	Xang	基于 JavaScript 进行动态 Web 开发	Java
	Xindice	纯 XML 数据库	Java
	JCS	分布式 Cache 系统(Java Caching System)	Java
容器(共7个)	Geronimo	J2EE 容器,类似 JBoss	Java
	iBATIS	简单 OR 映射,有.NET 版本	Java/C#
	Torque	OR 映射	Java
	ORB	ObjectRelationalBridge,OR 映射	Java
	JDO	JDO 标准的一个实现	Java
	HiveMind	类似 Spring 的东西,微内核 DI 容器	Java
	Excalibur	IoC 容器	Java
组件(共82)	APR	不同操作系统间可移植运行时库	C/C++
	Regexp	Java 正则表达式	Java
	ORO	Perl 风格的正则表达式	Java
	Xerces	XML 解析,Java/C 两种版本	Java/C++
	Crimson	XML 解析器	Java
	AXIOM	更高效的 DOM 实现	Java

续表

分 类	项目名	说 明	开发语言
组件(共82)	Lucene	全文检索,有.NET版本	Java/C#
	Logging	不仅Log4j,各个语言的版本都有了	Java/C++ /Perl/C#
	XMLBeans	XML转对象	Java
	JaxMe	Java/XML绑定的实现	Java
	Taglibs	JSP Tag库	Java
	HttpComponents	HTTP访问控件	Java
	ECS	辅助生成标签(Element Construction Set)	Java
	WSIF	Web Service调用(Web Services Invocation Framework)	Java
	SOAP	SOAP标准实现	Java
	Woden	WSDL书写工具	Java
	Tuscany	简化SOA开发	Java
	MIRAE	让手机支持基于XML的服务	Java
	BSF	脚本语言框架(Bean Scripting Framework),支持JavaScript等多种脚本语言	Java
	BCEL	用于直接生成字节码(Byte Code Engineering Library)	Java
	POI	存取Office文档	Java
	Batik	Java的SVG实现	Java
	Attributes	访问Java 1.5语言中定义的meta	这些项目都在jakarta commons中,都是Java的
	BeanUtils	反射支持	属于Java中的jakarta commons项目
	Betwixt	XML/JavaBean转换	属于Java中的jakarta commons项目
	Chain	职责链模式实现	属于Java中的jakarta commons项目
	CLI	命令行参数解析	属于Java中的jakarta commons项目
	Codec	通用加密/加密算法	属于Java中的jakarta commons项目
	Collections	Java容器类完善扩充	属于Java中的jakarta commons项目
	Configuration	各种来源配置文件存取	属于Java中的jakarta commons项目
	Daemon	Java模拟UNIX的Daemon	属于Java中的jakarta commons项目
	DBCP	数据链连接池	属于Java中的jakarta commons项目
	DbUtils	JDBC辅助类	属于Java中的jakarta commons项目

续表

分　类	项目名	说　明	开发语言
组件（共82）	Digester	XML 到 Java 对象映射工具	属于 Java 中的 jakarta commons 项目
	Discovery	根据名称来查找资源	属于 Java 中的 jakarta commons 项目
	EL	JSP 2.0 表达式标准实现	属于 Java 中的 jakarta commons 项目
	E-mail	发送 E-mail 类	属于 Java 中的 jakarta commons 项目
	FileUpload	文件上传辅助类	属于 Java 中的 jakarta commons 项目
	HttpClient	HTTP 客户端	属于 Java 中的 jakarta commons 项目
	IO	IO 操作辅助类	属于 Java 中的 jakarta commons 项目
	Jelly	基于 XML 的脚本引擎	属于 Java 中的 jakarta commons 项目
	Jexl	JSTL 表达式语言扩展	属于 Java 中的 jakarta commons 项目
	JXPath	用 XPath 语言来操作对象的辅助类	属于 Java 中的 jakarta commons 项目
	Lang	java.lang.类扩充	属于 Java 中的 jakarta commons 项目
	Launcher	跨平台 Java 应用启动器	属于 Java 中的 jakarta commons 项目
	Logging	不同 Log 实现的封装	属于 Java 中的 jakarta commons 项目
	Math	数学、统计辅助类	属于 Java 中的 jakarta commons 项目
	Modeler	创建兼容 JMX 标准的 MBeans	属于 Java 中的 jakarta commons 项目
	Net	各种网络协议实现	属于 Java 中的 jakarta commons 项目
	Pool	对象池	属于 Java 中的 jakarta commons 项目
	Primitives	很小的 Java 原始对象类型操作辅助类	属于 Java 中的 jakarta commons 项目
	SCXML	状态图 XML 标准实现	属于 Java 中的 jakarta commons 项目
	Transaction	多层次容器、文件操作事务支持	属于 Java 中的 jakarta commons 项目
	Validator	用 XML 定义校验器和校验规则	属于 Java 中的 jakarta commons 项目

续表

分 类	项目名	说 明	开发语言
组件(共82)	VFS	虚拟文件系统用于操作FTP、SMB、Zip等	属于Java中的jakarta commons项目
	Compress	tar、zip、bzip2压缩格式文件操作	这些项目都在jakarta commons的Sandbox中
	CSV	CSV文件格式支持	属于Java的Sandbox项目
	Exec	外部进程执行和环境设置辅助类	属于Java的Sandbox项目
	Finder	模拟Unix find命令	属于Java的Sandbox项目
	I18n	国际化辅助类	属于Java的Sandbox项目
	Id	生成ID辅助类	属于Java的Sandbox项目
	Javaflow	应用状态管理	属于Java的Sandbox项目
	JCI	Java编译器接口	属于Java的Sandbox项目
	OpenPGP	OpenPGP封装	属于Java的Sandbox项目
	Pipeline	管道辅助类用于并行或者顺序操作数据	属于Java的Sandbox项目
	Proxy	动态代码生成辅助类	属于Java的Sandbox项目
	Cache	对象缓存服务	这些项目都在jakarta commons中的Dormant(睡眠)中
	Clazz	class操作和反射操作	属于Java的Dormant项目
	Contract	契约编程用到Java中	属于Java的Dormant项目
	Convert	Java对象类西文转换辅助类	属于Java的Dormant项目
	Events	事件管理容器	属于Java的Dormant项目
	Feedparser	RSS和Atom实现	属于Java的Dormant项目
	Functor	用对象方式来操作函数	属于Java的Dormant项目
	JJar	Jar操作	属于Java的Dormant项目

续表

分　类	项目名	说　明	开发语言
组件（共82）	Latka	HTTP 功能测试	属于 Java 的 Dormant 项目
	Mapper	简单封装后可以选择不同的对象映射实现	属于 Java 的 Dormant 项目
	Messenger	JMS 用于 Web 开发中的辅助类	属于 Java 的 Dormant 项目
	Resources	国际化资源信息查找	属于 Java 的 Dormant 项目
	Scaffold	Web 应用开发工具	属于 Java 的 Dormant 项目
	ThreadPool	线程池	属于 Java 的 Dormant 项目
	Workflow	工作流管理系统框架	属于 Java 的 Dormant 项目
	XMLIO	XML 配置快速简便导入	属于 Java 的 Dormant 项目

本章小结

本章重点介绍了电子商务核心——商务逻辑层的实现、组件、组件调用、Web Service 等相关技术内容。商务逻辑层可以分为两大部分：一部分是构成商务应用的核心商务逻辑，与具体的企业应用密切相关；另一部分是支持核心商务逻辑运行的软硬件环境。这些软硬件环境主要包括：

（1）商务服务平台，如客户关系管理、供应链管理、电子市场、网络社区、支付网关接口、认证中心接口等。

（2）商务支持平台，如内容管理、目录管理、搜索引擎等。

（3）基础支持平台，如 VB、C++ 、Java 等之类的应用开发环境与开发工具，负载均衡与错误恢复等高性能与高可靠性的系统环境，主机管理、网络管理、安全管理等一些系统管理工具，JDBC、ODBC、EJB、XML 等一些对象组件与服务集成环境。

（4）Web 服务器平台、数据库平台、操作系统、计算机硬件与网络基础设施。

应用服务器的基本功能如下。

（1）提供高性能的应用程序运行环境。

（2）提供可扩充性。

（3）提供会话管理。

（4）提供目录及内容管理。

（5）提供商务引擎。

（6）提供系统管理。

EJB是用于开发和部署多层结构的、分布式的、面向对象的Java应用系统的跨平台的组件体系结构；EJB是部署在服务器上的可执行组件或商业对象；EJB是设计成运行在服务器上并由客户机调用繁荣非可视远程对象；可通过多个非可视JavaBean构建EJB；EJB还独立于平台，一旦编写好，还可以在任何支持Java的平台(包括客户机和服务器)上使用。

Web服务器是最近制定的一组标准，目的是利用成熟的Web技术，通过SOAP协议、WSDL服务描述语言和UDDI统一描述发现集成协议来实现跨语言(RMI要求处理两端都是Java环境)、跨平台(DCOM要求处理两端为Windows平台，CORBA要求处理两端为同一个ORB)、跨网络之间的分布处理与组件应用。

使用XFire，可以直接把Java类中的方法发布为Web Service，而不需要编写额外的代码。XFire是与Axis 2并列的新一代WebService框架，它具有以下特点。

(1) 支持一系列Web Service的新标准，如JSR181、WSDL 2.0、JAXB2、WS-Security等。

(2) 使用Stax来解释XML，性能有了质的提高。

(3) 容易建立服务。

(4) 易于与Spring框架结合。

(5) 灵活的Binding机制，可包括默认的Aegis、xmlbeans、jaxb2、castor。

一个WSDL文档通常包含7个重要的元素，即types、import、message、portType、operation、binding和service元素。这些元素嵌套在definitions元素中，definitions是WSDL文档的根元素。types元素是数据类型定义的容器，它使用某种类型系统(一般地使用XML Schema中的类型系统)；message元素是通信消息的数据结构的抽象类型化定义，使用types所定义的类型来定义整个消息的数据结构；operation元素是对服务中所支持的操作的抽象描述，一般单个operation描述了一个访问入口的请求/响应消息对；portType元素是对某个访问入口点类型所支持的操作的抽象集合，这些操作可以由一个或多个服务访问点来支持；binding元素是特定端口类型的具体协议和数据格式规范的绑定；port元素是定义为协议/数据格式绑定与具体Web访问地址组合的单个服务访问点；service元素是相关服务访问点的集合。

习题与思考

1. 商务逻辑层是如何构成的？
2. 商务逻辑层的主要任务是什么？主要通过哪些技术手段来实现？
3. 什么是应用服务器？有什么功能？市场上主流的应用服务器有哪些技术特征？
4. 试比较各种不同类型的应用服务器的技术特点。
5. 基于Java组件的第四代应用服务器与前几代应用服务器相比有哪些优点？
6. 在多层应用体系结构中，Java平台提供了哪些关键技术？
7. 应用服务器的基本功能是什么？
8. 何为Web Service？有何作用？
9. Java/RMI技术与CORBA技术相比，主要区别是什么？
10. 组件技术的核心思想是什么？试分析几种不同的组件规范。

11. 简述 RMI 的通信机制。
12. EJB 的特点是什么？EJB 与 JavaBeans 有何区别？
13. 开发一个 EJB 应用有哪些步骤？
14. SOA 有什么特征？
15. WSDL 是由什么构成的？
16. 下载并安装 Tomcat 应用服务器，分析如何配置 Tomcat 服务器。
17. 试述 CGI 技术的原理，并用 C 语言或其他语言编写一个简单的 CGI 应用程序。
18. 试用 JSP 编写一个简单的 Web 应用程序，并将它布置到 Tomcat 应用服务中去。
19. 试做一次 EJB 开发。
20. 试利用 XFire 开发 Web Service 应用。