基于构件的中间件技术 J2EE

窦 蕾 袁 臻 刘冬梅

(国防科学技术大学计算机学院 长沙 410073) (总后勤部后勤科学研究所 北京 100071)

摘 要 近年来,随着网络技术及应用的发展,对中间件的需求也日益增多。虽然有很多成熟的分布式中间件技术,例如CORBA,但其过于复杂而且存在缺陷,很难满足企业应用的需求。因此,基于构件的中间件技术 J2EE 应运而生。本文首先从企业对分布式中间件的发展需求说起,介绍 J2EE 产生的背景及其总体框架。其次,本文重点分析 J2EE 的核心技术,服务器端构件技术 EJB,阐述构件技术的基本理念和主要设计思想。最后本文比较了当前几种主流的基于构件的中间件技术:J2EE、CORBA、Microsoft .Net。

关键词 分布式系统,中间件,J2EE,构件,EJB

Component Based Middleware Technology J2EE

DOU-Lei YUAN-Zhen LIU Dong-Mei

(School of Computer Science, National University of Defense Technology, Changsha 410073)

(Logistics Science Research Institute of GLD, Beijing, 100071)

Abstract Currently there is increasing need in the use of middleware with the development of the network technology and application. Although there have been some mature middleware technologies, such as CORBA, yet its complexity cannot meet the need of the enterprises application. So the new middleware technology, J2EE, come to being. The article will introduce the background and the architecture of J2EE firstly. As the most important character of J2EE, component model, especially the server side component model EJB will be introduced in detail secondly. At last we will discuss the flaw of J2EE and make some comparison.

Keywords Distributed system, Middleware, J2EE, Component, EJB

1 引言

随着网络的出现和发展,各种基于网络的技术及应用也不断涌现。在网络环境下,企业应用由集中式企业内部应用,逐渐转为分布式全球应用,典型的是 client-server-database 三层体系结构的出现。客户可从世界任何角落,通过网络访问企业内部资源,浏览产品信息,与企业建立买卖关系;企业的合作伙伴可以通过网络与企业建立资源供销等合作关系;企业可以通过网络建立内部管理及运营系统。无论哪种类型,一个完整应用的逻辑实现对象都是分布在多个地点,多台机器上,这就是分布式应用。

企业级分布式应用的实现,对技术有两方面要求:首先,要求分布在各台机器上的对象能够互相访问、互相操作;其次,由于应用的企业级特性,对事务、安全、高可用、高可靠等性能的要求,在互连的基础上还必须为应用提供事务、安全等服务。目前已经出现了多种中间件技术支持企业分布式应用的开发,如 OMG 组织的 CORBA、微软的 Windows COM/DCOM,不仅实现分布对象的互连,而且提供多种服务支持。

但是,随着企业分布式应用的逐步成熟,激烈的商业竞争又对技术提出了更高的要求:

时间快。不仅开发应用的时间快,而且对于出现的新技术及客户的新要求反应也要快。

低成本:开发容易,所用人力、物力、财力尽可能少。

多功能:开发的应用功能强大,除了实现业务逻辑,还要支持邮件、Web Service、管理等多种功能。

高可靠:需要提供完善的安全、事务等服务。

易扩展:开发的应用应容易扩展以不断增添新功能。

易集成:当今技术发展迅速,大量遗留应用出现。所以需要方便集成遗留应用,以降低企业开发成本,保护投资。

面对新的要求,原有的中间件技术已无法满足需求。CORBA 虽然功能强大,可以跨平台、跨语言,但其编程复杂,由此带来学习周期较长、培训费用高,开发时间长,程序可靠性低等问题。COM/DCOM,则存在操作系统绑定、扩展、集成能力较弱的问题。新需求呼唤新技术,新的中间件技术 J2EE (Java 2 Enterprise Edition)应运而生。

J2EE 中间件的研究历史,可以追溯到 1997 年,1999 年由 SUN 公司发起,多家大公司支持,包括 IBM、Oracle、Sybase、Bea 等,共同提出了 J2EE 中间件。J2EE 中间件技术,以构件化为主要特点,主要目标是简化分布式应用的开发,由此满足开发时间快、成本低、易扩展等要求;J2EE 集成了大量技术,不但为应用提供多种功能,而且也提供了多种完善的服务;J2EE 的互操作模型及内部 Connector 机制实现了强大的集成能力。目前,多家公司已发布了支持 J2EE 规范的中间件产品,例如,BEA 的 Weblogic、Oracle 的 Oracle9i、IBM 的 WebSphere、iPlanet 的 iPlanet Application Server、Macromedia的 JRun、Persistence 的 PowerTier、Brokat 的 Gemstone/J、HP 的 BlueStone、Borland 的 AppServer 以及 INOA 公司的 IportableSuit,该产品已被波音公司采用。可见,J2EE 中间技术,具有良好的实用价值和应用前景。

2 J2EE 体系结构

Sun 公司提出的 J2EE 体系结构[1,2],如图 1 所示。

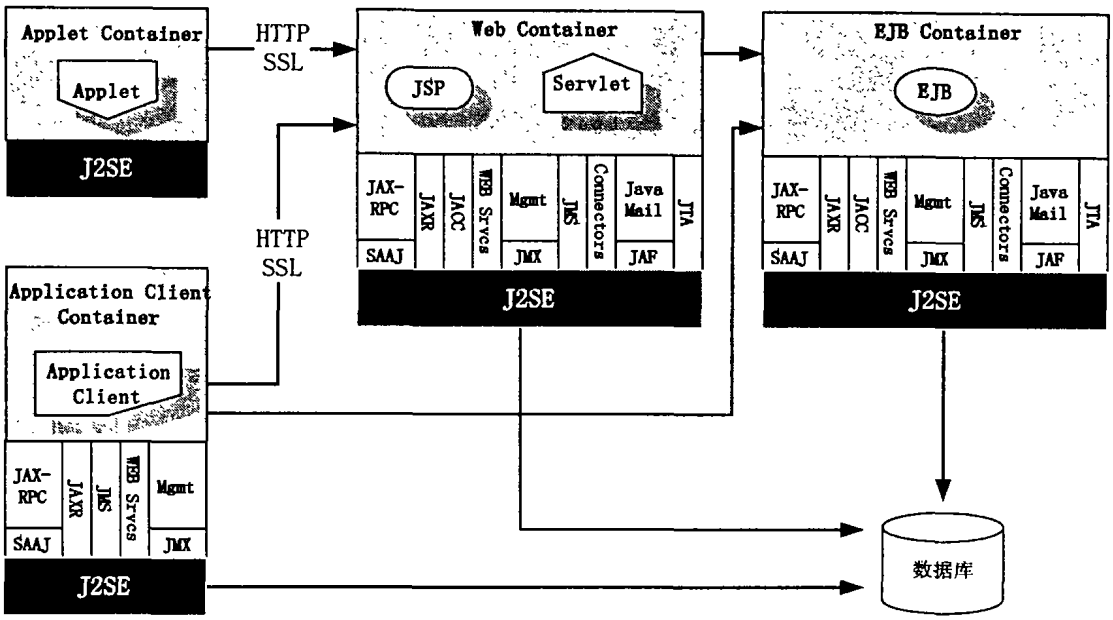

图 1　J2EE 体系结构

通过分析 J2EE 体系结构，不难发现 J2EE 的主要特点：

其一，J2EE 应用具有跨平台的特性。J2EE 应用建立在 Java 2 标准版(J2SE)的基础上，因此继承了 Java 的跨平台特性。

其二，J2EE 应用体系结构扩展三层结构到多层结构，包括：客户端表现层、服务器端表现层(即 WEB 层)、服务器端业务逻辑处理层、数据存储层。将原有的服务器业务逻辑处理层，进一步划分出服务器端表现层，并不是无中生有，是对功能的合理划分，不仅使整个应用结构清晰，而且层与层之间功能、设计相对独立，无疑对应用的扩展、移植和重用带来便利。

其三，构件技术构成了 J2EE 的核心技术。构件概念的提出主要基于软件重用的思想。虽然对象也是一种可重用的软件实体，但进一步分析，不难看出，对象实现的逻辑包括两部分：实现业务功能的逻辑、实现服务支持的逻辑，如安全服务、事务服务等。前者的实现相对简单，而后者则比较复杂，而且后者的实现通常是通过调用底层中间件或应用服务器提供的服务接口而实现，与底层平台结合紧密。因此，一个对象的开发不仅比较复杂，涉及大量底层实现，代码量大，移植性、重用性也比较差。构件技术通过分离服务支持逻辑，将对服务的需求实现为描述性文件，从而使编程简单，代码量小，移植性和重用性高。构件技术的引用，最终实现了 J2EE 的根本目标：简化分布式应用的开发，同时使 J2EE 与其它中间件技术相比具有明显优势和特点。J2EE 构件模型定义了四种类型的构件：位于客户端的应用客户构件(Application Client Component)和 Applet 构件(Applet Component)；位于服务器端表示层的 Web 构件(Web Component)，由 JSP、Servlet 构成；位于服务器端业务逻辑处理层的 EJB(Enterprise Java Bean)。一个完整的企业应用就是由这四种类型的构件组装部署而成。容器(Container)为构件提供了运行支撑环境。相应于四种类型的构件，J2EE 定义了四种类型的容器：应用客户容器(Application Client Container)、Applet 容器(Applet Container)、Web 容器(Web Container)和 EJB 容器(EJB Container)。

其四，J2EE 集成了大量技术，从而具有强大功能，包括：提供消息服务的 JMS(Java Message Service)，提供安全服务的 JAAS(Java Authentication and Authorization Service)和 JACC(Java Authorization Service Provider Contract for Containers)，提供 XML 解析服务的 JAXP (Java API for XML Parsing)，提供数据库访问服务的 JDBC，提供名字与目录服务的 JNDI(Java Naming and Directory Interface)、提供事务处理服务的 JTA (Java Transaction API)，提供自动邮件服务的 Java Mail 和 JAF (Javabean Activation Framework)，提供 Web Service 服务的 JAX-RPC (Java API for XML-based RPC)、SAAJ (SOAP with Attachments API for Java) 和 JAXR(Java API for XML Registries)，提供管理服务的 JMX (Java Management Extention)，以及提供遗留系统集成服务的 Connector 机制。

J2EE 应用虽然具有四层结构，但在实际应用中，可根据应用需求选择多种应用模式。如，Web 应用模式，客户端-服务器 Web 层-数据库；或者三层结构模式，客户端-服务器业务逻辑处理层-数据库；甚至两层结构模式，客户端-数据库。多种应用模式，为应用提供了多种灵活的选择，满足不同应用的需求。

其五，J2EE 具有强大的互操作与集成能力。在网络互通的社会，任何一种网络应用都需要考虑到与基于其它技术的应用间的互操作。如果成为信息的孤岛，则无疑选择了一条死亡之路。J2EE 通过支持多种协议和数据格式，提供了强大的互操作功能。J2EE 互操作模型如图 2 所示。

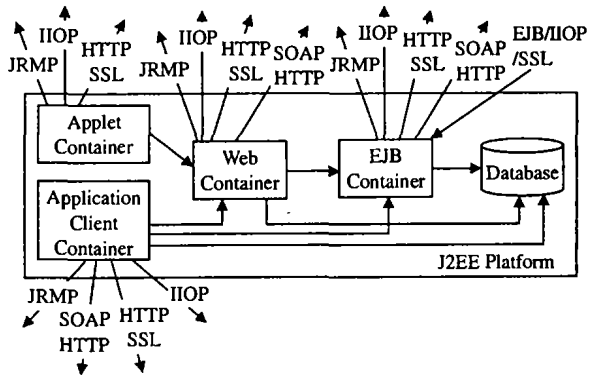

图 2　J2EE 互操作模型

在 J2EE 互操作模型中，主要支持三种类型的协议。首

先，为实现基本的网络互联并支持 Web Service，J2EE 应用服务器必须支持 Internet 和 Web 协议，包括 TCP/IP 协议族、HTTP、SSL、TLS、SOAP 等。其次，J2EE 对部分 OMG 协议提供支持。CORBA 作为目前一种主流的分布式应用开发平台，具有强大的集成能力。为了能与 CORBA 分布式应用互操作，同时也为了通过与 CORBA 的互操作，进而间接继承 CORBA 的强大互操作性，J2EE 支持 IIOP 协议、基于 IIOP 的 EJB 互操作协议、CORBA 互操作名字服务协议。其中，IIOP 协议是 CORBA 在对象互操作过程中采用的底层通讯协议，通过对该协议的支持，可以达到与异构应用底层协议的互操作。基于 IIOP 和 CORBA 安全互操作协议的 EJB 互操作协议，主要用于实现来自不同 J2EE 中间件提供商的 EJB 之间的互操作。基于 IIOP 和 COSNaming 协议的 CORBA 互操作名字服务协议，实现了 J2EE 应用与 CORBA 应用在定位对象方面的互操作。最后 J2EE 支持 JRMP，该协议是 Java 分布对象互操作的底层通讯协议，通过支持该协议，实现与 Java 分布对象的互操作。除了以上三类协议，J2EE 还定义了必须支持的数据格式，这些格式用于规范构件之间交换的数据，包括 XML、HTML、图形文件、压缩文件和可执行文件。显然，J2EE 平台是一个开放式平台，可提供各种异构客户、异构应用、异构平台间的互操作。

3 服务器端构件技术：EJB

构件技术是 J2EE 的核心技术，其中服务器端构件技术 EJB 又是四种构件中地位最重要而且发展也最完善的构件模型，它体现了构件技术的基本思想，反映了基于构件开发分布应用的基本思路。

EJB[3,4]是基于软件重用思想，在对象基础上发展起来的移植性更强、编程更简单的一种可重用的软件实体。构件的概念早已有之，如 Java Bean 和 ActiveX。但由于这些构件仅实现简单逻辑，缺乏对事务、安全等服务的支持，因此不能满足开发企业级应用的需求。为了在企业级应用的开发中引入构件，构件模型从简单构件模型扩展为企业级 Java 平台上的服务器端构件模型，Enterprise Java Bean，简称 EJB。

基于 EJB 的分布式应用开发过程，由传统的复杂对象开发、建立工程、编译、链接，转变为简单构件开发、应用组装、应用部署过程。

EJB 开发：一个 EJB 开发者开发的 EJB 通常由 Bean 实现对象、远程接口、Home 接口和部署描述文件构成。从功能角度来看，与普通对象一样，Bean 实现对象实现了业务逻辑处理功能。但与对象不同的是，EJB 将业务处理逻辑与服务支持逻辑分离，Bean 实现对象仅实现业务逻辑，而通过基于 XML 的部署描述文件精确描述了所需的事务、安全等服务，包括服务类型和服务策略，最终由容器根据描述提供所需服务支持逻辑。另外，虽然 EJB 从客户角度来看是一个分布对象，但 Bean 实现对象却是一个本地对象，不具备任何网络特性，也没有任何体现分布特性的代码。与普通分布对象的开发相比，EJB 开发者无需面向中间件的 API 进行网络编程，这无疑进一步简化了开发过程。远程接口定义了 Bean 实现对象向外部客户提供的服务接口信息。Home 接口定义了外部客户用于创建、删除以及查询 EJB 的接口信息。在应用部署时，J2EE 平台将根据 Home 接口和远程接口等信息自动生成 Home 对象和 EJB 对象，两者可被视为容器的一部分。EJB 对象实现了远程接口，对外接收客户发向 Bean 的服务请求，对内将请求转发给 Bean 实现对象。分布对象 EJB 对象，作为 Bean 的请求代理，一方面使得 Bean 间接具有网络特性，可被远程访问，另一方面，也通过截获请求从而向 Bean 提供了隐式的管理和服务。作为管理构件的工厂对象，Home 对象实现了 Home 接口。因为 EJB 对象作为 Bean 的代理，直接面向客户。因此 Home 对象中的创建、删除、查询方法实际是提供给客户用于创建、删除、查询 EJB 对象的。但真正运行于容器中的 Bean 实例的创建和删除，与之没有直接联系，由容器根据对资源的管理而独立实现。例如，当客户调用 Home 对象的构件创建方法时，实际仅创建了一个新的 EJB 对象，至于后端真正接收请求的 Bean 可能是通过重用构件实例获得的。EJB 对象和 Home 对象的分布特性最终使 EJB 成为一个分布对象，可被远程创建并访问。在开发的最后阶段，EJB 开发者将 Bean 实现对象、远程接口、Home 接口和部署描述文件打成一个 Jar 格式的压缩包，称为构件包。EJB 以构件包的形式流通、交易和被组装重用。对于不同类型的构件以及来自于不同提供商的 J2EE 平台，构件包中还可能包括诸如主关键字对象和一些平台专有文件等。

应用组装：根据具体应用需求，1 个或多个 EJB 可被组装成一个完整应用，以组装包的形式存在。组装包也可继续参与组装构成更大更复杂的应用。在组装中，构件间的调用依赖关系体现在构件的实现代码中，整个应用对服务以及数据库等资源的需求体现在组装级的部署描述文件中。构件技术提出的应用组装概念是非常有意义的，将促进软件产业向大规模工厂制造模式的发展。但 EJB 构件的组装模式还需要完善，原因在于没有在接口级定义构件间的调用依赖关系，导致构件间的调用隐含在构件实现代码中，因此给构件的组装重用和移植都带来了不便。

应用部署：构成应用的 EJB 通过应用部署，最终运行在 EJB 容器中。在部署过程中，J2EE 平台将根据远程接口、Home 接口以及部署描述文件等信息，自动生成 EJB 对象、Home 对象以及其它辅助对象，并完成所需外部资源的绑定等工作，从而使构件在容器的支持下正常响应远程客户的请求，获得声明的所需服务，并接收容器其它方面的管理。至此，基于 EJB 的应用进入正常运行状态。

客户在使用 Bean 所提供的服务时，首先通过 JNDI 获得 Home 对象的引用，进而通过 Home 对象创建一个 Bean 的代理对象 EJB 对象。客户通过向 EJB 对象发送服务请求从而激活 Bean 的相应方法，并获得响应。EJB 的工作原理如图 3。

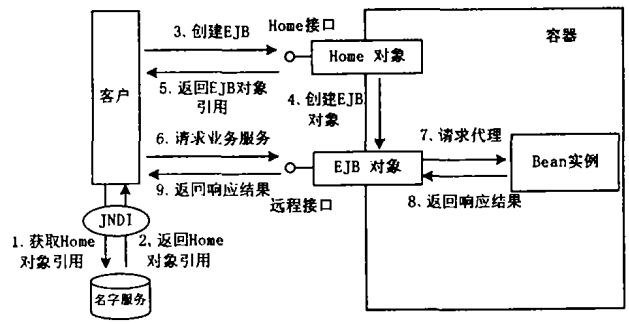

图 3 EJB 工作原理

就在 EJB 对象的请求截获和代理点上，容器为 Bean 隐式地提供了大量服务和管理，包括系统资源管理、生命周期管理、状态管理，以及部署描述文件中要求实现的安全服务、事务服务和持久状态服务等，如图 4 所示。

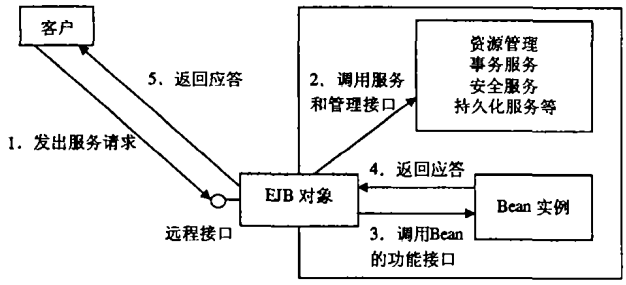

图 4 容器隐式提供的服务及管理

显然，基于构件的分布式应用与传统分布式应用的区别主要在于对服务支持逻辑的分离，服务的声明方式实现。以事务处理为例，若实现银行出纳业务的逻辑，传统应用要求在对象实现中对事务服务接口进行直接编程，实现事务控制，这就是服务的编程方式，如图 5 所示。而在基于构件的应用中，Bean 开发者只需声明所需服务，无需实现任何事务控制代码，在部署时由 J2EE 平台自动生成相应代码，并在运行过程中隐式实现构件所需的事务控制，这就是服务的声明方式，如图 6 所示。EJB 容器目前支持事务服务、安全服务以及持久化服务的声明方式。

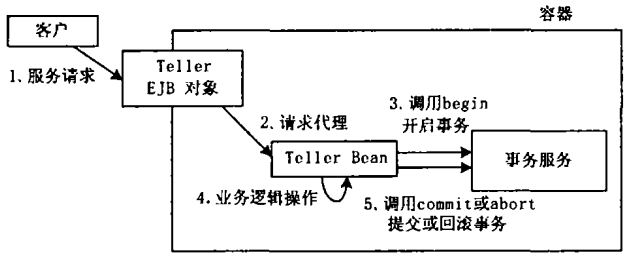

图 5 事务服务的编程方式

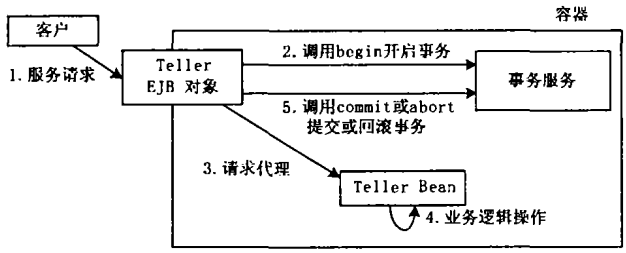

图 6 事务服务的声明方式

引入构件后，应用由构件组装而成。针对应用的多样性，构件也分为多种类型，充分满足了各种企业应用的需求。

会话 Bean(Session Bean)：对业务处理过程建模，细分为两种子类型：

· 无状态的会话 Bean(Stateless Session Bean)：所实现的业务处理过程不跨越多个方法，因此无需保存多个方法调用之间的状态，典型的如实现压缩、解压缩等功能的 Bean。

· 有状态的会话 Bean(Stateful Session Bean)：所实现的业务处理过程跨越多次方法调用，需要在多次方法调用之间保存状态，典型的如在电子商务中实现购物小车逻辑的 Bean，就需要在客户从登录到多次选中商品、放弃商品的整个选购过程中保存客户信息及其选购商品的信息。

实体 Bean(Entity Bean)：对数据建模，并实现数据访存等操作逻辑，是持久存储设备中的企业数据在内存中的一个视图。对实体 Bean 的创建、删除和查询最终映射到对持久数据的创建、删除和查询。显然实体 Bean 的实现中存在大量为实现持久性对数据库的访存逻辑，根据这些数据库访存逻辑实现主体的不同，实体 Bean 又分为两种子类型：

· Bean 管理持久性的实体 Bean(Bean-Managed Persistence Entity Bean)：由构件开发者针对具体持久存储设备的 API，如 JDBC 或其它专有接口，将持久数据的访存逻辑和操作逻辑硬编码到构件实现中，简称为 BMP 的实体 Bean。因为是硬编码方式，存在构件与持久存储设备之间的绑定问题，很难切换持久存储设备，重用构件。BMP 方式实际对应着持久化服务实现的编程方式。

· 容器管理持久性的实体 Bean(Container-Managed Persistence Entity Bean)：在构件部署时，容器根据来源于部署描述符的持久存储设备的具体位置信息和驱动程序信息以及需要持久化的数据域信息等，自动生成对持久存储设备的访存和操作代码，简称为 CMP 的实体 Bean。显然，CMP 方式减少了构件的代码量，构件开发者无需编写任何对持久数据的访存和操作代码。CMP 方式成功的解除了构件与持久存储设备之间的绑定关系，提高了重用性。在面向不同的持久存储设备重用构件时，只需修改 XML 格式的部署描述符文件，而无需重新修改并编译构件代码。虽然 CMP 方式提供了很大的便利，但相应也损失了实现的灵活性。因为 J2EE 平台提供商通常只针对某种持久存储设备，如数据库，提供 CMP 方式的持久性管理，所以如果存储设备不是数据库而是文件或 LDAP 等平台所不支持的设备，则需采用 BMP 方式。CMP 方式实际对应着持久化服务实现的声明方式。

消息驱动 Bean(Message-Driven Bean)：会话 Bean 和实体 Bean 都是基于 RMI-IIOP 这种同步调用机制进行通讯的，为了解决某些应用场景不适合使用同步调用方式的问题，在 EJB2.0 规范中提出了基于异步消息通讯机制的消息驱动 Bean。类似会话 Bean，消息驱动 Bean 也用于实现业务过程。区别在于，对消息驱动 Bean 的访问只能通过向其发送消息这种异步方式实现，如接收股票交易、信用卡验证等消息的 Bean。

虽然 EJB 是服务器端构件技术，但它反映了构件技术的基本特点。首先，应用开发遵循功能划分、人尽其才的原则。在基于构件的应用开发过程中，有多个角色共同参与：构件开发者、应用组装者、应用部署者，以及 J2EE 平台提供者。构件开发者仅负责实现构件的业务逻辑；应用组装者了解整个应用需求，负责确定所需构件以及组装方式；应用部署者熟悉应用部署的环境，负责确定应用所需的各种外部资源，如数据库；J2EE 平台提供者则是对各种服务和平台实现非常擅长的人，通常是中间件提供商、应用服务器提供商。不难看出，这种分工合作的方式，使每个角色都在从事自己擅长的工作，从而提高了整个软件的可靠性。其次，服务采用声明方式实现，一方面，极大减轻了应用开发者的工作量，提高了应用开发效率；同时，由于各种服务的实现与底层平台结合紧密，服务的声明方式又降低了应用对底层平台的依赖性，增加了移植性和重用性。最后，构件提出的应用组装概念对软件产业的推动和变革也是不容忽视的。

显而易见，基于构件开发分布应用具有代码量小、易于移植、可重用性强、可靠性高、开发容易快捷的优势。

4 几种基于构件的中间件比较

目前基于构件的分布式中间件技术有多种选择，主流技

(下转第 117 页)

HA 监控进程在系统中占有重要的地位。如果要保证整个系统的高可用性，那么首先必须保证 HA 监控进程的稳定性。在 HA 监控进程的实现过程中，尽可能做到功能简单清楚，减少与其他应用软件的依赖关系，作为一个相对独立的模块运行，这样能够有助于其稳定性的提高。

对程序的测试结果表明，系统从备用切换为主用所需时间小于100ms，在系统切换过程中，I/O 板与系统板之间的通信基本不受影响。在发生故障的系统板恢复正常后，重新查询系统状态，作为备用系统开始运行。

结论 随着计算机和网络的飞速发展，计算机软件在各个行业得到了越来越广泛的研究和应用，尤其在一些关键行业的关键应用上，要求系统必须具有保护业务关键数据和维持应用程序的高可用性的能力。高可用系统对硬件和软件都有一定的要求，在硬件方面，要求系统具有冗余硬件，并且能够在不影响系统运行的前提下，实现部件的热插拔；在软件方面，要求系统具有故障监测和故障处理功能，尽可能减少数据丢失和服务中断，提高系统的可用性。

本文所讨论的 Motorola CPX8216系统在双系统配置下完全符合高可用系统对硬件的要求，而 HA-Linux 则提供了一个集成的 HA 应用软件开发平台。在此平台上，本文详细讨论了一个 HA 电信应用软件的设计与实现，测试结果证明系统能够自动检测系统控制节点和服务进程的错误、失效，并且当发生这种情况时能够自动将系统控制和服务切换到备用节点，以提供不间断的服务。在不同的应用系统中，其 HA 监控软件具有很大的相似性，因此本文所给出的 HA 监控软件具有一定的可重用性，进一步的改进方向主要有在主备用系统之间采用检查点(check point)技术[1,4]，定期将主用程序的运行状态信息传送给备用系统。当主用系统宕机时，备用系统能够从最近的检查点处恢复运行。另一方面，故障监控和切换程序的稳定性还需进一步提高。

参 考 文 献

1 Preslan K, et al. Scalability and Failure Recovery in a Linux Cluster File System. In: 4th Annual Linux Showcase and Conf. Atlanta, Georgia, Oct. 2000. 169～180
2 Malkhi D, Reiter M. An Architecture for Survivable Coordination in Large Distributed Systems. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(2): 187～202
3 Gray J, Siewiorek D. High-availability computer systems. IEEE Computer, 1991, 24(9): 39～48
4 Burke T. High-Availability Cluster Checklist. Linux Journal, no. 80, Dec. 2000
5 http://www.motorola.com/computer/literature
6 Chen L, Avizienis A. N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation. Fault Tolerant Computing, FTCS-8, 1978. 3～9
7 Cristian F. Understanding Fault-Tolerant Distributed Systems. Communications of the ACM, 1991, 34(2): 57～78
8 Fabre J-C, Perennou T. A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach. IEEE Trans. on Computers, 1998, 47(1): 78～95
9 Dias D M, Kish W, Mukherjee R, Tewari R. A Scalable and Highly Available Web Server. In: Proc. 41st IEEE Intl. Computer Society Conf. (COMPCOM), Feb. 1996. 85～92
10 Gray C, Cheriton D. Leases: An efficient fault tolerant mechanism for distributed file cache consistency. In: Proc. 12th Symp. on Operating Systems Principles, Dec. 1989. 202～210

(上接第16页)

术有三种：CORBA、J2EE、Microsoft .Net。这三者在长期的发展过程中，互相借鉴，因此具有相似的体系结构和实现技术[5～7]，应根据具体应用需求进行选择。

J2EE 平台构筑在 Java 语言之上，因此利用了很多 Java 语言及 Java 虚拟机的优势。EJB 构件模型简单易学，基于 J2EE 开发分布式应用具有简便、快速的特点。J2EE 对 Web 应用提供了良好的支持，同时基于 J2EE 开发的应用显然可运行在不同的硬件平台上。J2EE 平台的强大功能、简单易学，以及多家大公司的成熟产品支持无疑使 J2EE 成为开发基于 Web 的企业应用的首要选择。但 J2EE 显然无法对除 Java 之外的其他语言提供支持。同时还需要指出的是 J2EE 构件模型的缺陷。EJB 构件仅通过远程接口定义了构件向外界提供服务的接口，却没有对其所需的服务进行刻画和定义，从而导致构件间的依赖关系全部隐含在构件的实现代码中。因此，当需要组装重用构件，或移植部分应用时，必须研究构件代码，抽取出该构件与其他构件间的依赖关系，只有在新的环境中提供具有同样接口、同样功能的依赖构件，才可以完成重用以及移植。显然，这种移植性与重用性不能满足人们的需求。值得一提的是，CORBA 构件模型 CCM[7](CORBA Component Model)，在构件接口对提供服务和所需服务都进行了刻画，从而真正实现了基于构件的应用组装。

Microsoft .Net 构筑在 Windows 操作系统之上，因此操作系统提供了很多特殊的支持以及性能上的优化。另外 .Net 引入的对多语言的支持，也是优越于 J2EE 的一个特性。但是 .Net 作为一个产品，专属于一家公司，仅针对一个平台，与 J2EE 作为一个开放式规范，多家公司支持，多种产品实现的现状对比起来，使得人们对基于 .Net 应用的日后维护、升级，以及与其它产品的兼容性都有所担忧[5,6]。

CORBA 的优势在于它对异构性的屏蔽，强大的集成能力，不象 J2EE 局限于 Java 语言，不象 Microsoft .Net 局限于 Windows 操作系统。CORBA 构件模型 CCM，具有更加完整的模型结构。CCM 不仅通过刻面(Facet)定义了构件向外界提供的服务接口，允许客户从不同角度使用同一构件的不同功能；CCM 还通过接插口机制(Receptacle)和事件机制(Event Source/Event Sink)分别定义了构件之间的依赖连接关系[7]，构件的重用性、移植性明显提高。CORBA 缺点也很明显，就是比较复杂，学习周期较长。虽然有产品支持，但缺乏易于使用的开发工具，从而导致开发基于 CORBA 的应用比较困难而且易于出错。CCM 的出现虽然将极大简化基于 CORBA 的服务器端应用开发，但 CCM 规范仍然比较复杂，目前尚存在很多问题，没有成熟的产品，有待研究和完善。

总结 基于构件的 J2EE 中间件明显简化了分布式应用的开发。构件化思想的引入，改变了开发应用的传统模式，由传统的复杂对象开发，建立工程，编译，链接的过程转变为简单构件开发，应用组装，部署的过程。无论从应用开发的效率，质量，还是灵活性、可用性来说，基于构件的应用开发都比传统基于对象的应用开发具备明显优势，而且能充分满足企业分布应用的服务需求。EJB 的简单易用初步展示了构件的无限魅力，但还存在缺陷，对构件自描述能力的深入研究，对构件模型的继续完善，将对软件的开发、重用产生深远影响，并将促进软件产业向大规模工厂制造模式发展。

参 考 文 献

1 Sun Microsystems, Inc. Java(tm) 2 Platform Enterprise Edition Specification, v1.3. Final Draft - 10/20/00
2 Sun Microsystems, Inc. Java(tm) 2 Platform Enterprise Edition Specification, v1.4. Proposed Final Draft 2- 11/8/02
3 Roman E. Mastering Enterprise Java Bean. John Wiley & Sons, Inc. 1999
4 Roman E. Mastering Enterprise Java Bean. Second Edition. John Wiley & Sons, Inc. 2002
5 Roman E, Oberg R. The Technical Benefits of EJB and J2EE technologies over COM+ and Windows DNA. Dec. 1999
6 Vawter C, Roman E. J2EE vs. Microsoft. NET: A comparison of building XML-based web services. June, 2001
7 Object Mnagement Group. Corba Component Model Specification V3.0. Formal/2002-06-65