作者：**Monica Pawlan**
翻译：环球联动开发组（Global Empower Inc.）

J2EE™ 蓝图摘要

莫尼卡·宝兰

如果您已经安装了 J2EE™ Platform（Enterprise Edition）软件，并通读了各种可用的指南来帮助您学习使用该软件，那么，您可能已经准备开始为 J2EE 平台编写第一个多层企业应用程序了。在这个时候，您可能对应用设计应当在何处着手、以及如何判断哪些 J2EE API 最能满足您的要求上存有疑惑。

您可以得到帮助。《为企业版 Java™2 平台设计企业应用程序》（*Designing Enterprise Applications for the Java™ 2 Platform, Enterprise Edition ISBN 0-201-70277-0*）描述了 J2EE 平台，并提出了一个编程模型，可以帮助您在为 J2EE 平台设计多层企业应用程序时作出最佳决策。为了理论联系实际，第 10 章举了一个电子商务宠物商店应用程序的例子，描述了该编程模型是如何指导设计的。该书和该宠物商店应用程序一起被称作 J2EE Blueprints（J2EE 蓝图）。

本文是 J2EE 蓝图的一个摘要，向您介绍该编程模型，并帮助您掌握它。您可以访问 J2EE 下载页面，下载该书的 PostScript 或者 PDF 格式的版本，或者下载书中作为例子的宠物商店应用程序。该书印刷本将在 2000 年 7 月上市。

- J2EE 基本架构：概述（J2EE Architecture—A Bird's-Eye View）
- 可再用应用组件（Reusable Application Components）
- 设计用户界面和引擎（Designing the User Interface and Engine）
- 设计基于 Web 的应用（Designing Web-Based Applications）
- Servlets 和 JSP 页面（Servlets and JSP Pages）
- 模型、视图、控制器设计范式（Model, View, Controller Design Pattern）
- J2EE 应用中的 Enterprise Beans（Enterprise Beans in J2EE Applications）
- 宠物商店应用程序（Pet Store Application）

J2EE 基本架构：概述

电子商务和信息技术的快速发展及对它的需求给应用程序开发人员带来了新的压力。必须以比以前更少的金钱、更少的资源来更快地设计、建立和生产企业应用程序。

为了降低成本，并加快企业应用程序的设计和开发，J2EE 平台提供了一个基于组件的方法，来设计、开发、装配及部署企业应用程序。J2EE 平台提供了多层的分布式的应用模型、组件再用、一致化的安全模型以及灵活的事务控制。您不仅可以比以前更快的速度向市场推出创造性的客户解决方案，而且，您的平台独立的、基于组件的 J2EE 解决方案不会被束缚在

任何一个厂商的产品和 API 上。

J2EE 规范定义了以下种类的组件：

- 应用的客户组件（Application client components）
- Enterprise JavaBeans™ 组件（Enterprise JavaBeans™ components）
- Servlets 及 JavaServer Pages™（JSP 页面）组件（也被称作 Web 组件）（Servlets and JavaServer PagesTM (JSP pages) components）
- Applets

一个多层的分布式的应用模型意味着应用逻辑被根据功能而划分成组件，并且可以在同一个服务器或不同的服务器上安装组成 J2EE 应用的这些不同的组件。一个应用组件应被安装在什么地方，取决于该应用组件属于该多层的 J2EE 环境中的哪一层。这些层是客户层、Web 层、 业务层及企业信息系统层（EIS）。

客户层（Client Tier）

J2EE 应用可以是基于 Web 的，也可以是不基于 Web 的。在一个基于 Web 的 J2EE 应用中，用户的浏览器在客户层中运行，并从一个 Web 服务器上下载 WEB 层中的静态 HTML 页面或由 JSP 或 servlets 生成的动态 HTML 页面。

在一个不基于 Web 的 J2EE 应用程序中，一个独立客户程序，或者不运行在一个 HTML 页面中，而是运行在其它一些基于网络的系统（比如手持设备或汽车电话）中的 applet 程序，在客户层中运行，并在不经过 Web 层的情况下访问 enterprise beans。该不基于 Web 的客户层可能也包括一个 JavaBeans 类来管理用户输入，并将该输入发送到在企业层中运行的 enterprise bean 类来处理。根据 J2EE 规范，JavaBeans 类不被视为组件。

为 J2EE 平台编写的 JavaBeans 类有实例变量和用于访问实例变量中的数据的“get 和 set 方法”。以此种方式使用的 JavaBeans 类在设计和实现上通常都是简单的，但是它们必须符合 JavaBeans 规范中列出的命名和设计约定。

Web 层

J2EE Web 组件可以由 JSP 页面、基于 Web 的 applets 以及显示 HTML 页面的 servlets 组成。调用 servlets 或者 JSP 页面的 HTML 页面在应用程序组装时与 Web 组件打包在一起。就像客户层一样，Web 层可能包括一个 JavaBeans 类来管理用户输入，并将输入发送到在业务层中运行的 enterprise beans 类来处理。

运行在客户层的 Web 组件依赖容器来支持诸如客户请求和响应及 enterprise bean 查询等。

业务层

作为解决或满足某个特定业务领域（比如银行、零售或金融业）的需要的逻辑的业务代码由运行在业务层的 enterprise beans 来执行。一个 enterprise bean 从客户程序处接收数据，对数据进行处理（如果需要），再将数据发送到企业信息系统层存储。一个 enterprise bean 还从存储中检索数据，并将数据送回客户程序。

运行在业务层的 enterprise beans 依赖于容器来为诸如事务、生命期、状态管理、多线程及资源存储池提供通常都非常复杂的系统级代码。

业务层经常被称作 Enterprise JavaBeans（EJB）层 。业务层和 Web 层一起构成了 3 层 J2EE 应用的中间层，而其它两层是客户层和企业信息系统层.

企业信息系统层

企业信息系统层运行企业信息系统软件，这层包括企业基础设施系统，例如企业资源计划（ERP）、大型机事务处理（mainframe transaction processing）、数据库系统及其他遗留信息系统（legacy information systems）。J2EE 应用组件因为某种原因(例如访问数据库)可能需要访问企业信息系统。

注意：J2EE 平台的未来版本将支持 Connector 架构，该架构是将 J2EE 平台连接到企业信息系统上的一个标准 API。

查询服务（lookup services）

因为一个 J2EE 应用程序的组件是单独运行的，并且往往在不同的设备上运行，因此，需要一种能让客户层和 Web 层代码查询并引用其他代码和资源的方法。客户层和 Web 层代码使用 Java 命名和目录接口（JNDI）来查询用户定义的对象（例如 enterprise beans）、环境条目（例如一个数据库驱动器的位置）、企业信息系统层中用于查找资源的 JDBC™ DataSource 对象，以及消息连接。

安全和事务管理（Security and Transaction Management）

诸如安全和事务管理这样的应用行为可以在部署时在 Web 和 enterprise bean 组件上进行配置。这个特征将应用逻辑从可能随装配而变化的配置设定中分开了。

安全

J2EE 安全模型允许配置一个 Web 或 enterprise bean 组件，使系统资源只能由授权的用户访问。例如，一个 Web 组件可以被配置成提示输入用户名和密码。一个 enterprise bean 组件可以被配置成只让特定团体中的成员调用其某些方法。或者，一个 servlet 组件可以被配置成让某个组织中的所有人都能访问其某些方法，同时只让该组织中的某些享有特权的人访问一些方法。同样是该 servlet 组件，可以针对另外一个环境而被配置成让每个人都能访问其所有方法，或者仅让选定的少数人访问其所有方法。

事务管理（Transaction Management）

J2EE 事务模型使得能够在部署时定义构成一个单一事务的方法之间的关系，以使一个事务中的所有方法被处理成一个单一的单元。这是我们所希望的，因为一个事务是一系列步骤，这些步骤要么全部完成，要么全部取消。

例如，一个 enterprise bean 可能有一组方法，使我们可以通过从第一个账户借出并存入第二个账户的方式而将钱从第一个账户转移到第二个账户。我们希望全部的操作被作为一个单元对待，这样，如果在借出之后存入之前发生了故障，该借出操作被取消。

事务属性是在装配期间定义在一个组件上的。这使得能将来自多个应用组件的方法归到一个事务中，这说明，我们可以轻易变更一个 J2EE 应用程序中的应用组件，并重新指定事务属性，而不必改变代码或重新编译。

在设计应用组件时，要记住，尽管 enterprise beans 有一个可使应用组件的容器自动启动多步事务的机制，但是 applet 和应用的客户容器可能并不支持这一点。然而，applet 和应用客户容器总是能够调用支持这一点的一个 enterprise bean。

还应当注意，JSP 页面和 servlets 没有被设计成是事务的，它们通常应当将事务工作交给一个 enterprise bean 来完成。然而，如果事务工作在一个 JSP 页面或 servlet 中是必须的，那么此种工作也应当是非常有限的。

可再用应用组件（Reusable Application Components）

J2EE 组件（applets、应用的客户、enterprise beans、JSP 页面及 servlets）都被打包成模块，并以 Java ARchive（JAR）文件的形式交付。一个模块由相关的组件、相关的文件及描述如何配置组件的配置描述文件组成。

例如，在组装过程中，一个 HTML 页面和 servlet 被打包进一个模块之中，该模块包含该 HTML 文件、servlet 组件及相关的配置描述文件，并以一个 Web ARchive（WAR）文件的形式交付，该 WAR 文件是一个带.war 扩展名的标准 JAR 文件。使用模块使得利用相同的组件中的某些组件来组装不同的 J2EE 应用程序成为可能。

例如，一个 J2EE 应用程序的 Web 版可能有一个 enterprise bean 组件，还有一个 JSP 页面组件。该 enterprise bean 组件可以与一个应用客户组件结合，以生成该应用程序的非 Web 版本。这不需要进行额外的编码，只是一个装配和部署的问题。

并且，可再用组件使得将应用开发和部署过程划分成由不同的角色来完成成为可能，这样不同的人或者公司就能完成封装和部署过程的不同部分。J2EE 平台定义了如下角色：

J2EE 产品提供商（J2EE Product Provider）
设计并使 J2EE 平台、API 和在 J2EE 规范中定义的其他特征能被其它公司或人购得的公司。

应用组件提供商（Application Component Provider）
创建用于 J2EE 应用程序的 Web 组件、enterprise bean 组件、applets 或应用客户程序的公司或个人。在装配过程中，应用组件文件、接口及类被打包进一个 JAR 文件中。

应用程序装配商（Application Assembler）
从组件提供商获得应用组件 JAR 文件，并将它们组装成一个 J2EE 应用的 Enterprise ARchive（EAR）文件的公司或个人，此种文件是一个带.ear 扩展名的标准文件。

应用装配商提供与该应用程序相关的整体信息，并使用验证工具来检验 EAR 文件的内容是正确的。组装和部署信息存储在一个基于文本的配置描述文件中，此种文件使用 XML 标记来标记该文本。应用装配商可以按照第 7 章“设计企业应用”的描述直接编辑该配置描述文件，或者使用一个能通过交互式选择来正确添加 XML 标记的装配和配置工具来编辑该配置

描述文件。

部署商（Deployer）
部署 J2EE 应用程序的公司或个人。职责包括设定事务控制、安全属性，并根据应用组件提供商提供的指示来标明一个 enterprise bean 是自己处理自身的存储，还是由一个容器来处理等。

部署涉及配置和安装。在配置过程中，部署商遵循应用组件提供商提供的指示来解决外部依赖问题，定义安全设定，以及分配事务属性。在安装过程中，部署商将应用组件安装到服务器上，并生成容器特定的类和接口。

系统管理员（System Administrator）
配置并管理运行 J2EE 应用程序的计算环境和网络基础设施，并监督运行时环境的人员。

工具提供商（Tool Provider）
生产被组件提供商、装配商及部署商使用的用于进行开发、组装和打包的工具的公司或个人。

设计用户界面和引擎（Designing the User Interface and Engine）

在为 J2EE 应用程序设计用户界面和后端引擎时，我们需要决定是让该程序基于 Web，还是不基于 Web。在做出这个决定时，我们可能希望考虑平台配置、下载速度、安全、网络流量和网络服务。

例如，包含有用户界面并且经常被大量用户访问的一个 applet 可能需要花很长的时间才能被下载下来，这让用户沮丧。然而，如果知道该 applet 要运行在一个公司的内部网内的受控环境中，那么，在这种情况下，该 applet 将拥有一个完全可接受的下载速度。

另一个考虑是，繁重的处理应当在哪里执行。例如，如果客户程序在一个蜂窝电话或呼机中执行，服务器应当完成尽量多的计算和数据处理，而客户程序只应显示结果就可以了。然而，设计在一个强大的台式机平台上运行的大型财务分析系统则应当在客户机上完成其复杂计算。

应用的客户程序和 applet 用户界面通常都是用 Swing API 创建的，该 API 可从标准版 Java2 平台（Java 2 Platform，Standard Edition）中获得。Swing API 提供了一整套 GUI 组件（表格、树形结构、按钮等），这些组件可以被用来实现一种比用一个典型的 HTML 页面所能实现的更为交互的体验。Swing 也支持 HTML 文本组件，这个组件可以被用来显示来自一个服务器的响应。

客户程序可以直接访问 enterprise bean 层或企业信息系统层。但应谨慎实现这种程序。绕过 EJB 层的程序可以使用 JDBC API 来访问一个关系型数据库，但应被限制于对数据库表格进行维护等这样的管理任务上。

设计基于 Web 的应用程序（Designing Web-Based Applications）

基于 Web 的应用程序是基于浏览器的，并且，如果它们运行在 Internet 上的话，可能被全世界的人访问。当设计一个基于 Web 的应用程序时，

不仅需要决定用什么来处理内容和应用逻辑（HTML、XML、JSP 页面及 servlets），而且还应当考虑使该应用程序国际化。

一个国际化的基于 Web 的应用程序向用户提供了选择一种语言，然后，根据该选定语言加载应用的正文的方式。对被支持的每种语言而言，应用正文都被存储在一个外部文件中，并且，与另外一个文件的关键词相对应。应用代码使用这些关键词以及选定的语言来加载正确的文本。国际化 API 还提供类来根据选定的语言来格式化日期和金钱。

一旦制订了使应用程序国际化的细节，我们就可以决定用什么来实现它了。总体来说，一个基于 Web 的应用程序使用 HTML 来显示数据;用 XML 来定义数据以使其可被另一个程序读取并处理;使用 JSP 页面或 servlets 来管理用户与业务层或存储层之间的数据流。

可以在 J2EE 平台上实现的基于 Web 的应用程序有四种。从简单到复杂排列，它们是：

- 基本 HTML
- 带基本 JSP 页面或 servlets 的 HTML
- 带 JavaBean 类的 JSP 页面
- 将应用逻辑根据功能划分成区域的高度结构化的应用

当设计一个基于 Web 的应用程序时，需要决定用什么来建立它。并且，如果我们是从建立一个简单的应用程序开始着手的话，如果认为以后会给该应用程序添加功能，那么，设计应当适应今后发展的需要。下面的内容通过比较 JSP 页面和 servlets，来帮助我们做出某些设计决定。

Servlets 和 JSP 页面

Servlets 是实现动态内容的一种简便的、平台独立的、Web 服务器独立的方式。JSP 页面是开发 servlets 的一种基于文本的、以显示为中心的方式。JSP 页面提供了 servlets 的所有好处，并且，当与一个 JavaBeans 类结合在一起时，提供了一种使内容和显示逻辑分开的简单方式。

分开内容和显示逻辑的好处是，更新页面外观的人员不必懂得 Java 代码，而更新 JavaBeans 类的人员也不必是设计网页的行家里手。JSP 页面和 servlets 都比公共网关接口（CGI）更可取，因为 CGI 不是平台独立的，使用了更多系统开销，而且没有一个简单的方法来访问参数数据，并将这些数据发送给一个程序。

可以用带 JavaBeans 类的 JSP 页面来定义 Web 模板，以建立一个由具有相似的外观的页面组成的网站。JavaBeans 类完成数据提供，这样在模板中就没有 Java 代码，这意味着这些模板可以由一个 HTML 编写人员来维护。

在选择使用一个 servlet，还是一个 JSP 页面时，要记住的是，servlets 是一个程序设计工具，它最适用于不需要频繁修改的低级应用功能;而 JSP 页面则通过以显示为中心的描述性的方法将动态内容和逻辑结合在一起。

对于一个使用一个 JSP 页面的简单的基于 Web 的应用程序，我们可以使用定制标记或者 scriptlets，而不是使用 JavaBeans 类来将内容与应用逻辑结合起来。定制标记被打包到一个标记库中，并被引入到一个 JSP 页面中。Scriptlets 是直接嵌入在 JSP 页面中的很小的 Java

代码段。

模型、视图和控制器架构（Model, View, Controller Architecture）

在基于组件的 J2EE 平台充分内置了灵活性的情况下，剩下的问题可能是如何组织应用程序以实现简单高效的应用程序升级和维护，以及如何让不懂程序代码的人员避开程序数据。答案就在模型、视图和控制架构（MVC）的使用之中。MVC 这样的架构是一个描述重现的问题及其解决方案的设计范式，但问题每次重现时，解决方案都不是完全相同。

MVC 设计范式包括三种对象：
模型（model）提供应用业务逻辑（enterprise bean 类）;视图（view）则是其在屏幕上的显示（HTML 页面、JSP 页面、Swing GUI）;控制器则是 servlet、JavaBean 或 session bean 类，它用于管理用户与视图发生的交互。我们可以将控制器想象成处在视图和数据之间，对视图如何与模型交互进行管理。

通过使视图完全独立于控制器和模型，就可以轻松替换前端客户程序。并且，通过将控制器和模型代码保持在视图之外，那些不理解这些代码的人员就不能改变他们不应改变的东西。将控制器和模型分开可以在不影响模型的情况下改变控制器，也可以在不影响控制器的情况下改变模型。

例如，如果应用的前端是一个 HTML 页面，一个 HTML 专家就可以更新它。如果使用一个 JSP 页面，将控制器的代码放到一个 JavaBean 或 session bean 类中，或使用动作标记（action tags），这样，JSP 页面就仅包含 JSP 代码了。

J2EE 应用程序中的 Enterprise Beans

当编写管理特定业务功能（比如追踪雇员资料或进行复杂财务计算）的 J2EE 应用程序时，请将完成这些任务的业务逻辑放置在 EJB 层的 enterprise beans 中。以这种方式，我们就可以使代码集中在解决手边的业务问题，而利用 enterprise bean 容器来支持低层服务，比如状态管理、事务管理、线程管理、远程数据访问和安全等。

将业务逻辑与低层系统逻辑分开意味着容器可以在运行时创建和管理 enterprise bean。 按照规范编写的任何 enterprise bean，都可以根据其在一个特定的 J2EE 应用程序中将被如何使用来对其事务管理或安全属性进行配置，并可以被部署到任何一个与规范兼容的容器中。可再用组件使不必改变和重新编译 enterprise bean 代码成为可能。

一个 enterprise bean 由接口和类组成。客户程序通过 enterprise bean 的 home 和远程接口来访问 enterprise bean 的方法。Home 接口提供了创建、删除和定位 enterprise bean 的方法，而远程接口则提供了业务方法。在部署时，容器由这些接口来创建类，使客户能够创建、删除、定位或调用位于 enterprise bean 上的业务方法。enterprise bean 类提供了业务方法、创建方法和查寻方法的实现。如果 enterprise bean 管理它自己的持久性的话，还为其生命期方法提供了实现。

有两种 Enterprise beans：entity beans 和 session beans。

一个 session bean 代表与客户程序的一个短暂的会话，而且可能执行数据库读写操作。一个 session bean 可能会自己调用 JDBC，或者它可能使用 entity bean 来完成此种调用。在后者这种情况下，这个 session bean 是该 entity bean 的客户。一个 session bean 的域包含会话状态，而且是短暂的。如果服务器或者客户程序崩溃，该 session bean 就丢失了。这种模式通常被用于像 PL/SQL 这样的数据库程序设计语言上。

一个 entity bean 代表一个数据库中的数据及作用于该数据的方法。在一个关系型数据库中的雇员信息表中，每一行都有一个 bean 来代表。entity beans 是事务的，并且是长寿命的。只要数据留在数据库中，entity bean 就存在。这个模式可以被很容易地用于关系型数据库，而不仅限于对象数据库。

Session beans 可以是有状态的，也可以是无状态的。一个有状态的 session bean 包含代表客户程序的会话状态。该会话状态是该 session bean 实例的域值加上这些域值所引用到的所有对象。有状态 session beans 并不代表在一个持久数据存储中的数据，但是，它可以代表客户程序访问和更新数据。

无状态 session beans 没有用于某个特定客户程序的任何状态信息。它们通常被用于提供不保持任何特定状态的服务器端行为。无状态 session beans 要求更少的系统资源。一个提供一种一般服务，或用于表示被存储的数据的一个被共享的视图的业务对象是无状态 session bean 的一个例子。

因为 enterprise beans 占用可观的系统资源和带宽，可能希望将某些业务对象构造成数据访问对象或值对象。数据访问对象完成诸如代表客户程序访问数据库等工作。值对象用于代表容纳数据字段并提供简单的“get 和 set”方法来访问这些数据的一个结构。

另外，可以将程序构造成使用 enterprise bean 在客户和 EJB 层的其它部分之间承担通信的任务。

一个使用容器管理的持久性来访问一个关系型数据库的 enterprise bean，并不要求在 bean 的代码中使用任何 JDBC 2.0 API 来进行数据库访问，因为容器完成了这些工作。然而，如果使用 bean 管理的持久性，或者我们要访问一个非关系型数据库的企业信息系统，那么我们就必须在 bean 中提供相应的代码来完成这些工作。

在一个 enterprise bean 使用 bean 管理的持久性来访问一个数据库的情况下，必须使用 JDBC 2.0 API 代码来实现该 enterprise bean 的生命期方法，以便处理数据的加载和存储，以及在运行时系统和持久数据存储之间维持数据的一致性。

一个使用 bean 管理的持久性的 enterprise bean，或一个需要访问企业信息系统的 Web 组件必须提供合适的代码。这些代码可能是用于进行数据库访问的 JDBC 2.0 API；或是用于访问一个特定企业信息系统的企业信息系统 API；或是用于抽象企业信息系统 API 的复杂性和低层细节的一个访问对象，或是用于访问企业信息系统资源的一个连接对象。

尽管 Web 层使用 HTTP 或 HTTPS 来在各层之间传输数据，但是，EJB 层使用的是 RMI-IIOP。RMI-IIOP 是一个完整的分布式计算协议，能让任何访问一个 enterprise bean 的客户层程序或 Web 层程序直接访问 EJB 层的服务。这些服务包括用于查找和引用 enterprise beans 的 JNDI，发送和接收异步消息的 Java Message Service（JMS），以及用于关系型数据库访问的 JDBC。

宠物商店应用程序（Pet Store Application）

《设计企业应用程序》这本书的第 10 章将 J2EE 应用编程模型应用到一个宠物商店示例应用程序上。您可以从 java.sun.com 网站的下载页面下载该演示版，并且您会在该下载的顶层目录中找到配置和安装指导。

在第 10 章，我们从规范制定和设计阶段到实现，再到指定安全和事务属性的装配和部署阶段，对应用的开发过程进行了描述。第 10 章的第一节描述了应用的情景，然后进一步讨论了架构、功能的划分、向各层分配功能以及在各层中分解对象。

Monica Pawlan 是 JDC 写作队伍的成员。她拥有 2D 及 3D 图形技术、安全以及数据库产品的背景，喜爱探索新兴技术。

Date Jun 2000