第5章 Web容器

Web容器是中间件的重要组成部分，它为处于其中的应用程序组件提供了一个环境。Web容器可以管理对象的生命周期、对象与对象之间的依赖关系，同时对动态语言进行解析，实现了系统软件和应用软件之间的连接。它的作用是将应用程序运行环境与操作系统隔离，从而简化应用程序开发，使程序开发者不用关心系统环境，而只需关注该应用程序在解决问题上的能力。本章将介绍Web容器的概念，介绍典型的Web应用服务器框架和Java EE事实上的标准框架Spring，以及与Web容器紧密相关的依赖注入和面向切面编程等技术。

5.1 Web服务器

5.1.1 Web服务器概述

Web服务器也称为WWW(World Wide Web)服务器，是驻留于因特网上的某种类型的计算机程序，也是因特网上发展最快和目前应用最广泛的服务。它起源于1989年3月，是由欧洲核子研究组织(European Organization for Nuclear Research，通常简称为CERN，即其法语名称的缩写)发展出来的主从结构分布式超媒体系统。通过Web服务器，用户使用简单的方法就可以迅速方便地获取丰富的信息资料。用户在通过Web浏览器访问信息资源的过程中，无须关心一些技术性的细节，且所访问的界面相对友好。Web服务器的出现标志着WWW时代的到来。Web服务器在因特网上一经推出就受到了热烈的欢迎，并得到了爆炸性的发展。

从技术角度上看，Web服务器是运行在互联网上的一个超大规模的分布式系统。其设计初衷是一个静态信息资源发布媒介，通过超文本标记语言(Hyper Text Markup Language，HTML)描述信息资源，通过统一资源标识符(Uniform Resource Locator，URL)定位信息资源，通过超文本传输协议(HyperText Transfer Protocol，HTTP)请求信息资源。HTML、URL和HTTP三个规范构成了Web的核心体系结构，是支撑着Web运行的基石。客户端(一般为浏览器)通过URL找到网站(如www.google.com)，发出HTTP请求，服务器收到请求后返回HTML页面。Web服务器基于TCP/IP等协议，Web网页在这个协议族之上将计算机的信息资源连接在一起，形成了万维网(World Wide Web)。

目前主流的Web服务器有Apache[①]、Nginx[②]、IIS[③]。而CGI(Common Gateway Interface)、JSP(Java Server Pages)、Servlets、ASP(Active Server Pages)等技术的发展增强了Web服务器

① https://www.apache.org/.

② https://www.nginx.com/.

③ https://www.iis.net/.

获取动态资源的能力，使得 Web 服务器朝着企业级应用方向发展。面对快速的业务变化，Web 容器为开发者提供了快速开发接口，使得开发人员只需关注业务本身即可写出可靠、符合业务需求的程序。

5.1.2 Web 服务器的工作原理

Web 服务器处理浏览器等 Web 客户端的请求，并返回相应响应以及向 Web 客户端提供文档和显示界面。Web 服务器采用的是浏览器/服务器(B/S)结构，其作用是整理和存储各种网络资源，并响应客户端软件的请求，把客户所需的资源传送到 Windows、UNIX 或 Linux 等平台上。Web 服务器传送页面使浏览器可以浏览，而应用程序服务器提供的是客户端应用程序可以调用的方法。换句话说，Web 服务器专门处理 HTTP 请求，而应用程序服务器通过多种协议为应用程序提供商业逻辑。

Web 服务器的工作过程一般可分成如下 4 个步骤：连接过程、请求过程、应答过程以及关闭连接(如图 5-1 所示)。连接过程是 Web 服务器及其浏览器之间建立一种连接的过程，用户可以通过查看套接字 socket 这个虚拟文件确定连接是否建立。请求过程是 Web 浏览器运用 socket 这个文件向其服务器提出各种请求。应答过程是 HTTP 把在请求过程中所提出来的请求传输到 Web 服务器，进而实施任务处理；然后运用 HTTP 把任务处理的结果传输到 Web 浏览器，同时在 Web 的浏览器上面展示上述所请求的界面。关闭连接是当应答过程完成以后，Web 服务器和其浏览器之间断开连接的过程。

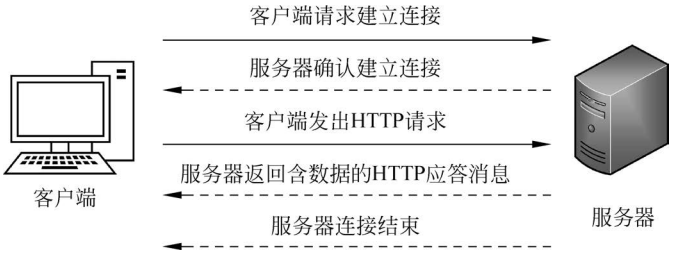


图 5-1 客户端与服务器连接示意图

在典型的 Web 应用中(如图 5-2 所示)，用户在浏览器中输入网址或单击链接，浏览器获得事件后与 Web 服务器建立 TCP 连接，这个过程就是上文所说的连接过程。如果连接成功，浏览器将用户事件按照 HTTP 格式打包成一个数据包，向服务器提出各种请求，这个过程是请求过程。服务端程序接到请求后，以 HTTP 格式解包请求，分析客户端请求，进行分类处理，如提供某种文件、处理数据等；最后将结果打包成 HTTP 格式，返回给浏览器端，这是应答过程。应答过程基于 HTTP，把请求传输到 Web 的服务器，进而实施任务处理。然后把任务处理的结果传输到 Web 浏览器，并在 Web 的浏览器上面展示上述所请求界面，其处理结果一般是 HTML 网页文件。当应答过程完成后，服务器和客户端浏览器会根据约定断开连接。服务器工作的四个步骤环环相扣、紧密相连，可以支持多个进程、多个线程以及多个进程与多个线程相混合的技术。

对于网站服务器而言，连接过程、请求过程和关闭连接都是较为标准的例行程序，关键点在于应答过程，即如何根据请求返回各种各样的结果网页。一般而言，网页可分为静态网页和动态网页两种。静态网页是预先存在服务器上的固定文件，动态网页则是服务器根据用户的

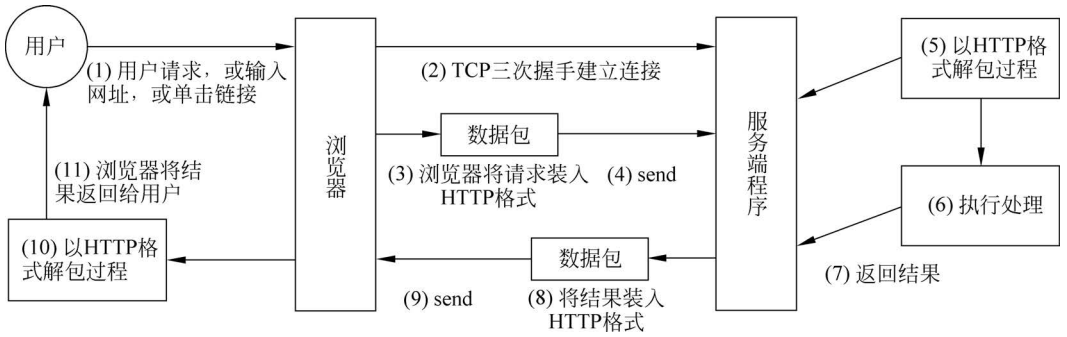


图 5-2　Web 服务器工作原理

请求动态组装而成的。对于不同的请求，动态网页返回的结果一般不同。因此，可以把网站服务器看作一个拥有各种车间和车床的工厂，该工厂可根据用户的订单生产个性化的产品。而网站开发人员的主要任务就是定义工厂的各种组件，安排流程，生产出用户需要的产品。

Web 服务器的代理模型(delegation model)相对简单。为了处理一个请求(request)，Web 服务器可以响应(response)一个静态页面或图片，进行页面跳转(redirect)，或者把动态响应(dynamic response)的产生委托(delegate)给一些其他的程序，例如 CGI 脚本、JSP 脚本、Servlets、ASP 脚本、服务器端 JavaScript，或者一些其他的服务器端技术。不管请求是什么，服务器端的程序通常产生一个 HTML 响应来让浏览器可以浏览。当一个请求被送到 Web 服务器里来时，它只单纯地把请求传递给可以处理该请求的程序(服务器端脚本)即可。Web 服务器只需提供一个可执行服务器端程序和返回程序所产生的响应的环境。

服务器端程序通常也具有事务处理、数据库连接和消息队列等功能。虽然 Web 服务器不支持事务处理或数据库的连接池，但它可以配置各种策略来实现容错性和可扩展性，如负载平衡、缓冲等。同时，Web 服务器也提供了很多处理请求、分配任务、封装显示界面的类。用户继承这些类，覆写相应的方法，即可加入个性化处理，同时符合基本的网页连接和处理流程。

5.1.3　Web 服务器和 MVC 框架

Web 服务器可基于不同的编程语言进行编程，自 2000 年以来各厂商和开源社区提出了多种框架和架构来优化 Web 服务器的工作过程。例如，SSH(Spring、Struts、Hibernate)、Django、Ruby on Rails 架构等都可以帮助开发者在开发、部署、维护 Web 应用程序时变得简单快捷。这些框架细节各不相同，但一个共同之处是会通过抽象，分离出服务器的基本执行流程：例行的标准化的流程由服务器来处理，而用户只需要负责定义个性化的、非标准化的流程。

Web 服务器通常采用 MVC(Model-View-Controller，模型-视图-控制器)框架来抽象客户端和服务器的访问流程。MVC 用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。控制器(Controller)是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。模型(Model)是应用程序中用于处理应用程序数据逻辑的部分，通常模型对象有对数据直接访问的权力，负责在数据库中存取数据。视图(View)是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的，能

为应用程序处理不同的数据视图。图 5-3 是一个 MVC 组件之间的合作场景，Controller 读取用户输入，Model 处理业务逻辑，View 根据 Model 的处理结果更新视图。

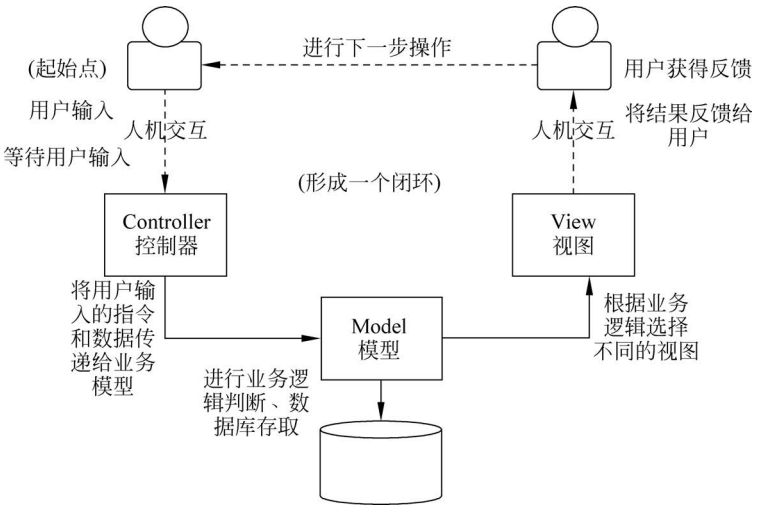


图 5-3　MVC 组件之间的合作

MVC 分层有助于管理复杂的应用程序，使得程序员在一段时间内专门关注一方面。例如，可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。MVC 的分层也简化了分组开发，不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

Java EE 是 MVC 框架的一个具体实现。如图 5-4 所示，Java EE 的组件也大体可以按照模型-视图-控制器的方式来进行划分。其中，控制器 Servlet 是 Java 编写的服务器端程序，主要功能是交互式地浏览和修改数据，生成动态 Web 内容；视图 JSP 部署于网络服务器上，可以响应客户端发送的请求，并根据请求内容动态地生成 HTML、XML 或其他格式文档的 Web 网页，然后返回给请求者。实体模型 Java Bean 封装数据操作，将功能、处理、值、数据库访问和其他任何可以用 Java 代码生成的对象进行打包，以供其他类使用。

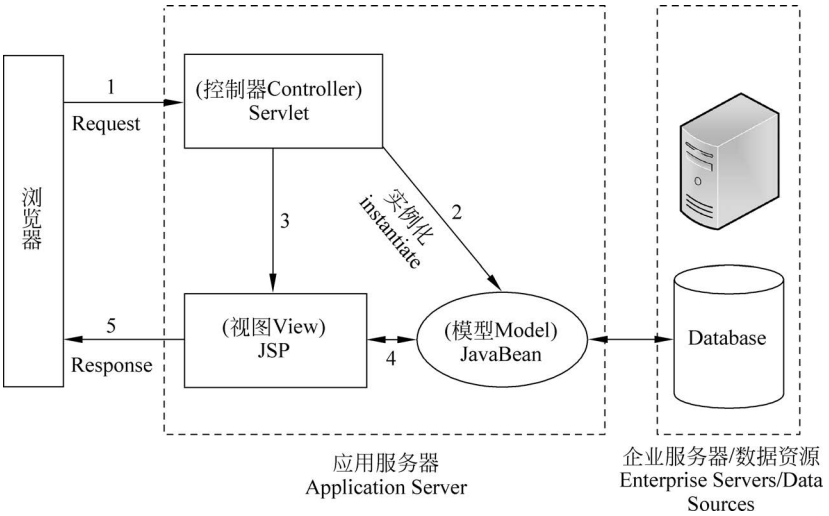


图 5-4　Java EE 技术架构示意图

5.2 Web 容器简介

5.2.1 容器的概念

用户访问 Web 服务器时，服务器会在内存中生成各种对象来处理请求和处理业务逻辑。这就出现了 Web 容器的概念。容器中的对象可以由一个用户定义的模板来生成。例如 JavaBean，它是一种 Java 语言写成的可重用组件。JavaBean 的类必须是具体的和公共的，并且具有无参数的构造器。由于可能有很多用户访问，因此内存中一般存在很多的对象实例。

Web 容器是一种服务程序，一般位于应用服务器之内，由应用服务器负责加载和维护。一个容器只能存在于一个应用服务器之内，一个应用服务器可以创建和维护多个容器。容器一般遵守可配置的原则，即容器的用户可以通过对容器参数的配置来达到自己的使用需求，而不需要修改容器的代码。如图 5-5 所示，容器是位于应用程序/组件和服务器平台之间的接口集合，使得应用程序/组件可以方便地部署到 Web 服务器上运行。在服务器的一个端口就有一个提供相应服务的程序处理从客户端发出的请求，如 Java 中的 Tomcat 容器、ASP 的 IIS 或 PWS 都是这样的容器。一个服务器可能不止一个容器，容器给处于其中的应用程序/组件(JSP、Servlet)提供一个环境，使组件直接跟容器中的环境变量交互，而不必关注其他的系统问题。

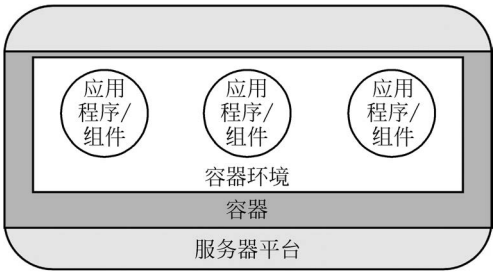


图 5-5 容器与服务器的联系

在 Spring 和 EJB(Enterprise Java Beans)框架结构中，都将中间件服务传递给耦合松散的 POJO(Plain Old Java Objects)对象，而容器则负责对象的创建、对象间的关联和对象的生命周期管理。通过容器的配置(如常见的 XML 配置文件)，可以定义对象的名称、产生方式、对象间的关联关系等。在启动容器之后，容器自动地生成这些对象，而无须用户编码来产生对象或建立对象与对象之间的依赖关系。Web 容器是自动化例行程序，是减少用户工作量的一个有效方法。它提供一个应用框架，屏蔽和掩藏例行的或者复杂的事物(如事务、安全或持久性)，支持弹性配置，使得开发者只把关注聚焦到应该的地方，是简化企业软件开发的关键。因此，一个设计良好的框架及中间件架构可以提高代码重用率、开发者的生产力及软件的质量。

5.2.2 解耦合、控制反转及依赖注入

1. 解耦合

在采用面向对象方法设计的软件系统中，底层实现是由 N 个对象组成，所有的对象通

过彼此合作，共同实现系统的业务逻辑。图 5-6 是软件系统中耦合的对象的示例，各模块之间互相依赖，耦合度较高。如果其中一个模块出了问题，就可能影响到整个系统的正常运转。并且随着工业级应用的规模越来越庞大，对象之间的依赖关系也越来越复杂，经常会出现对象之间的多重依赖性关系。对象之间耦合度过高的系统，不利于系统的维护。

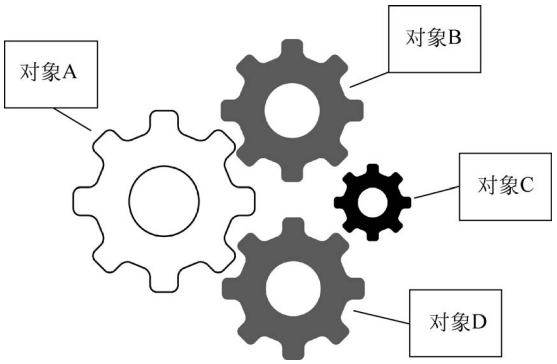


图 5-6　软件系统中耦合的对象

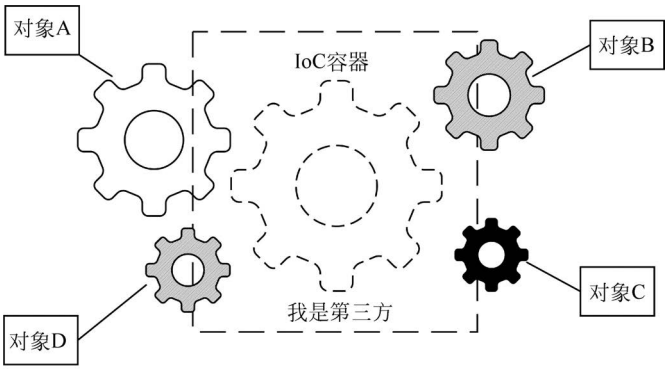
耦合关系不仅会出现在对象与对象之间，也会出现在软件系统的各模块之间，以及软件系统和硬件系统之间。如何降低系统之间、模块之间和对象之间的耦合度，是软件工程追求的目标之一。

2. 控制反转

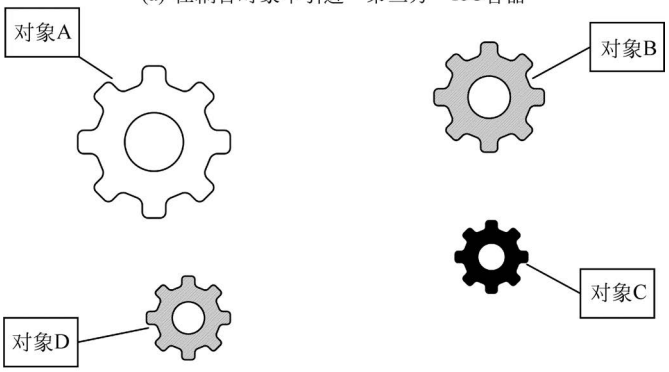
为了解决对象之间的耦合度过高的问题，软件专家 Michael Mattson 提出了反转控制(Inverse of Control，IoC)理论，用来实现对象之间的“解耦”。

在图 5-7(a)中，由于引进了中间位置的“第三方”IoC 容器，A、B、C、D 这 4 个对象没有了耦合关系。齿轮之间的传动全部依靠“第三方”了，全部对象的控制权全部交给“第三方”IoC 容器。所以，IoC 容器成了整个系统的关键核心。它起到一种类似“黏合剂”的作用，把系统中的所有对象黏合在一起发挥作用。如果没有这个“黏合剂”，对象与对象之间会彼此失去联系，这就是有人把 IoC 容器比喻成“黏合剂”的由来。而如果拿掉中间的容器(即容器无须开发人员实现)，如图 5-7(b)所示的就是开发人员要实现的整个系统所需要完成的内容。这时候，A、B、C、D 这 4 个对象之间去除了耦合关系，彼此毫无联系。当实现 A 的时候，无须再去考虑 B、C 和 D 了，对象之间的依赖关系已经降到了最低。如果实现了 IoC 容器，对于系统开发而言，参与开发的成员只要实现自己的类而无须关注其他的工作。

因此，控制反转，简单来说就是把复杂系统分解成相互合作的对象。这些对象类通过封装以后，内部实现对外部是透明的，不仅降低了解决问题的复杂度，而且可以灵活地被重用和扩展。IoC 借助于“第三方”实现具有依赖关系的对象之间的解耦。软件系统在没有引入 IoC 容器之前，如图 5-6 所示，对象 A 依赖于对象 B，那么对象 A 在初始化或者运行到某一点的时候，自己必须主动去创建对象 B 或者使用已经创建的对象 B。无论是创建还是使用对象 B，控制权都在自己手上。软件系统在引入 IoC 容器之后，创建对象的对象改变了，如图 5-7(a)所示，由于 IoC 容器的加入，对象 A 与对象 B 之间失去了直接联系。所以，当对象 A 运行到需要对象 B 的时候，IoC 容器会主动创建一个对象 B 注入对象 A 需要的地方。通过这两种情况的对比，不难看出：对象 A 获得依赖对象 B 的过程，由主动行为变为了被动行为，控制权颠倒过来了，这就是“控制反转”这个名称的由来。



(a) 在耦合对象中引进“第三方”IoC容器



(b) 没有耦合关系的对象

图 5-7　IoC 解耦过程

3. 依赖注入

2004 年，著名的软件专家 Martin Fowler 给“控制反转”取了一个更合适的名字叫作“依赖注入”(Dependency Injection，DI)。控制被反转之后，获得依赖对象的过程由自身管理变为由 IoC 容器主动注入。所谓依赖注入，就是由 IoC 容器在运行期间，动态地将某种依赖关系注入对象之中。所以，依赖注入(DI)和控制反转(IoC)是从不同的角度描述的同一件事情。通过引入 IoC 容器，利用依赖关系注入的方式，实现对象之间的解耦。而组件对象之间的解耦，将大大提高程序的灵活性、重用性和可维护性。目前，依赖注入(DI)和控制反转(IoC)理论已经被成功地应用到实践当中，很多的 Java EE 项目均采用了 IoC 框架产品。

从另一个角度看，依赖注入和控制反转都符合依赖倒置原则(Dependence Inversion Principle，DIP)。DIP 设计原则指出，高层模块不应该依赖底层模块的实现，而应该依赖底层模块的抽象，也就是接口。IoC 是一种设计模式，它指导程序员应该怎么做才能保证遵循 DIP 原则。它将底层模块的实例化和为高层模块提供底层实现这两件事交给第三方系统负责，而依赖注入是容器为高层模块提供底层实现的方式。

5.2.3　面向切面的编程

1. 面向切面的编程

AOP(Aspect Oriented Programming，面向切面的编程)是通过预编译方式和运行期动态代理的方式，实现程序功能的统一维护的一种技术。AOP 是面向对象程序设计(Object

Oriented Programming，OOP)的延续，也是 Spring 框架中的一个重要内容。利用 AOP 可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

面向切面的编程是促使软件进行关注点分离的一项技术。软件系统由许多不同的组件组成，每一个组件各负责一定功能。有些组件除了实现自身的核心功能之外，还负责额外的职责。如图 5-8 所示，持久化管理、日志管理、事务管理、调试管理和安全管理等系统服务经常会融入其他具有核心业务的组件中去。这些系统服务常被称为“方面”，因为它们跨越系统的多个组件。如果将这些关注点分散到多个组件中去，代码会出现一些问题。例如，实现系统关注点功能的代码重复出现在多个组件中，且组件会因为那些与自身核心业务无关的代码而变得混乱。

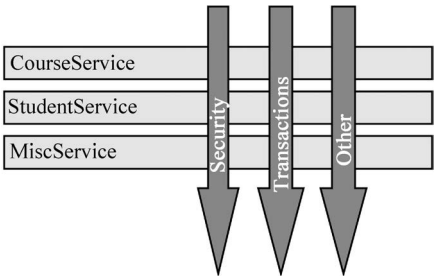


图 5-8 “方面”应用逻辑跨越多个应用程序对象

AOP 提供的解决方法是使这些服务模块化。AOP 将应用系统分为核心业务逻辑和横向通用逻辑两部分。横向通用逻辑也就是所谓的“方面”。如图 5-9 所示，在 Spring 中提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务(如审计和事务管理)进行内聚性的开发。应用对象只需实现业务逻辑，而无须负责其他的系统级关注点，如日志或事务支持。

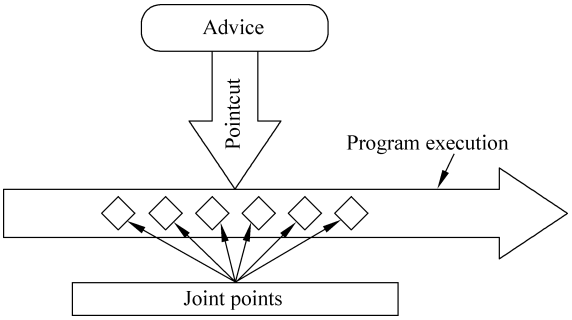


图 5-9 “方面”功能在一个或多个连接点处编织到程序的执行中

2. AOP 的核心概念

在 5.4.4 节将具体介绍面向切面的编程实例，此处先给出 AOP 的几个核心概念。

(1) 连接点(Join Points)：程序执行过程中某一个方面可以切入的点，如特定方法的调用或特定的异常被抛出。

(2) 通知(Advice)：在特定的连接点，AOP 框架执行的动作。各种类型的通知包括“Around”“Before”“Throws”和“After returning”等。许多 AOP 框架(包括 Spring)都是以

拦截器作通知模型，维护一个“围绕”连接点的拦截器链。“Around”通知是包围一个连接点（如方法调用）的通知。“Around”通知在方法调用前后完成自定义的行为，它们负责选择继续执行连接点，或返回自己的返回值，或抛出异常来短路执行（即通过控制条件跳过某些程序片段）。“Before”通知是在一个连接点之前执行的通知，但它不能阻止连接点前的执行（除非它抛出一个异常）。“Throws”通知是在方法抛出异常时执行的通知。Spring 提供了强制类型的“Throws”通知，因此用户可书写代码捕获感兴趣的异常（和它的子类），而不需要从 Throwable 或 Exception 强制类型转换。“After returning”通知是在连接点正常完成后执行的通知。例如，一个方法正常返回，没有抛出异常时。

（3）切入点（Pointcut）：指定通知将被引发的一系列连接点的集合。AOP 框架必须允许开发者指定切入点，例如，使用正则表达式来匹配连接点的集合。如果通知 Advice 是定义了横切时“What”和“When”，则切入点 Pointcut 定义了 Where。

（4）方面（Aspect）：通知和切入点合在一起称为方面，是一个关注点的模块化，这个关注点实现可能横切多个对象。例如，事务管理、安全管理等都是横切关注点。

（5）引入（Introduction）：引入允许程序添加新的方法或字段到被通知的类。Spring 允许引入新的接口到任何被通知的对象。例如，可为一个类引入 IsModified 接口和保存其状态数据的对象来记录对象被修改的状态，而不必更改原有类的代码。

（6）目标对象（Target Object）：包含连接点的对象，也被称作被通知或被代理对象。

（7）AOP 代理（AOP Proxy）：AOP 框架创建的对象，包含通知。在 Spring 中，AOP 代理可以是 JDK 动态代理或 CGLIB 代理。

（8）编织（Weaving）：把方面应用到目标对象，从而创建一个新的代理对象的过程。编织可以在编译时完成（例如使用 AspectJ 编译器），也可以在运行时完成。Spring 和其他纯 Java AOP 框架一样，在运行时完成织入。

5.3 Java EE 框架

5.3.1 概述

Java EE（Java Platform，Enterprise Edition）是 Sun 公司（2009 年 4 月被甲骨文收购）推出的企业级应用程序版本。这个版本以前称为 J2EE（1.2～1.4 版本），从版本 1.5 开始正式使用 Java EE 这个名字。它能够帮助我们开发和部署可移植、健壮、可伸缩且安全的 Web 应用和企业应用。这些应用通常设计为多层的分层应用，包括 Web 框架的前端层，提供安全和事务的中间层，以及提供持久性服务的后端层。这些应用程序具备快速响应性和适应用户需求增长的伸缩性。

Java EE 是在 Java SE 的基础上构建的，平台为每个层中的不同组件定义了 API，它提供 Web 服务、组件模型、管理和通信服务，可以用来实现企业级的面向服务体系结构（Service-Oriented Architecture，SOA）和 Web 2.0 应用程序。此外，还提供一些额外的服务，例如，命名、注入和跨平台的资源管理等。这些组件提前部署在提供运行期支持的容器中。容器为应用组件提供底层的 Java EE API 联合视图，Java EE 组件间通过容器的协议和方法彼此交互，或者与平台的服务交互。同时，容器可以透明地注入组件所需要的服务，例如，事务管理、安全检查、资源池和状态管理等。这种基于容器的模型和对资源的访问，使

得开发人员可以从底层基础任务中解脱出来，而聚焦于应用的模型和应用商业逻辑。

5.3.2 Java EE 框架组成

Java EE 将传统的 C/S 两层结构细分为四层：应用层、服务层、业务层、数据层，如图 5-10 所示。Java EE 应用层可以是浏览器网页，也可以是 Application 客户端，或者是 Applets(一种运行在浏览器 Java 虚拟机上的小程序)。服务层组件包括 Servlet、JSP、JSF。业务层组件一般是与业务需求相对应的代码，通常被称为 Enterprise JavaBeans。例如，如何从客户端接收信息、如何根据具体业务逻辑处理信息、数据以什么样的格式存储在数据库中。数据层可以是数据库或者一个企业级的信息系统，也可以用于业务数据的保存。

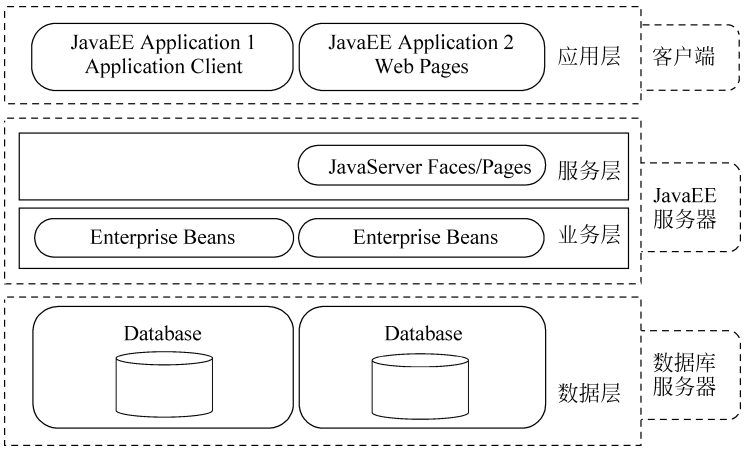


图 5-10 Java EE 体系结构

近年来工业界和开源社区涌现了许多的框架，使得创建企业的应用程序更加容易。服务层有 Struts、JSF、Tapestry、WebWork、Velocity 等框架。业务层可以用普通的 Java Beans，也可以用 EJB(Session Bean)。数据层可以选择 JDBC、ORMapping 框架(如 Hibernate、toplink 等)、SQLMapper tools(IBatis)、JDO、Entity Bean 等。

5.3.3 Java EE 的主要技术

Java EE 平台由一整套服务、应用程序接口和协议构成。它对开发基于 Web 的多层应用提供了功能支持，提供了一个基于组件的方法来加快设计、开发、装配和部署企业应用程序。下面简单介绍 Java EE 的一些技术规范。

1. JDBC

JDBC(Java Database Connectivity，Java 数据库连接)是 Java 语言中用来规范客户端程序如何访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法。它提供连接各种关系数据库的统一接口，可以为多种关系数据库提供统一访问。它由一组用 Java 语言编写的类和接口组成。JDBC 为工具/数据库开发人员提供了一个标准的 API，数据库开发人员能够用纯 Java API 编写数据库应用程序，为 JDBC 开发者屏蔽了一些细节问题。JDBC 对数据库的访问也具有平台无关性。Java 开发中，使用 JDBC 操作数据库需要如下几个步骤。

(1) 加载数据库驱动程序。

(2) 连接数据库。

(3) 操作数据库。

(4) 关闭数据库,释放连接。

JDBC 除了代表 Java 数据库编程接口以外,同时也是 Sun Microsystems 的商标。

2. JNDI

JNDI(Java Name and Directory Interface,Java 命名和目录接口)是 Sun 公司提供的一种标准,是用于从 Java 应用程序中访问名称和目录服务的一组 API。命名服务将服务名称与对象关联起来,从而使得开发人员在开发过程中可以使用名称来访问对象。目录服务是命名服务的一种自然扩展,两者之间的关键差别是目录服务中对象不但可以有名称还可以有属性(例如,用户有 E-mail 地址),而命名服务中对象没有属性。命名或目录服务允许程序集中管理共享信息的存储。

JNDI 为开发人员提供了查找和访问各种命名和目录服务的通用、统一的接口。它提供了一致的模型来存取和操作企业级的资源 DNS(Domain Name System)、轻型目录访问协议 LDAP(Lightweight Directory Access Protocol)、本地文件系统或应用服务器中的对象。屏蔽了企业网络所使用的各种命名和目录服务,使得应用程序更加一致和易于管理。类似 JDBC,JNDI 也是构建在抽象层上。JNDI 已经成为 Java EE 的规范之一,所有的 Java EE 容器都必须提供一个 JNDI 的服务。

3. EJB

EJB(Enterprise Java Beans)又被称为企业 Java Beans,是一个用来构筑企业级应用的服务器端可被管理组件。Sun 公司发布的文档中对 EJB 的定义是: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。J2EE 技术之所以赢得广泛重视的原因之一就是 EJB。它提供了一个框架来开发和实施分布式商务逻辑,显著地简化了具有可伸缩性和高度复杂的企业级应用程序的开发。EJB 规范定义了 EJB 组件在何时如何与它们的容器进行交互作用。容器负责提供公用的服务,例如,目录服务、事务管理、安全性、资源缓冲池以及容错性。

4. RMI

RMI(Remote Method Invoke,远程方法调用)是一种用于过程调用的应用程序编程接口,是纯 Java 的网络分布式应用系统的核心解决方案之一。它使用了序列化的方式在客户端和服务器之间传递数据。RMI 是一种被 EJB 使用的更底层的协议。

5. Java IDL/CORBA

Java IDL(Interface Definition Language,Java 接口定义语言)是 Java 2 开发平台中的 CORBA(Common Object Request Broker Architecture)功能扩展,可实现与异构对象的互操作性和连接性。它基本上是 JDK 提供的对象请求代理(Object Request Broker,ORB)。在 Java IDL 的支持下,开发人员可以将 Java 和 CORBA 集成在一起。可以创建 Java 对象并使之可在 CORBA/ORB 中展开,还可以创建 Java 类并和其他 ORB 一起服务 CORBA 对象客户。后一种方法提供了一种途径,通过它 Java 可以被用于将新的应用程序和旧的系统集合在一起。

6. JSP

JSP(Java Server Pages,Java 服务器页面)是由 Sun Microsystems 公司主导创建的一种动态网页技术标准。JSP 页面由 HTML(标准通用标记语言下的一个应用)代码和嵌入

其中的 Java 代码组成，将特定变动内容嵌入到静态的页面中，实现以静态页面为模板的动态网页。JSP 部署于网络服务器上，可以响应客户端发送的请求，并根据请求内容动态地生成 HTML、XML 或其他格式文档的 Web 网页，然后返回给请求者。JSP 技术以 Java 语言作为脚本语言，为用户的 HTTP 请求提供服务，并能与服务器上的其他 Java 程序共同处理复杂的业务需求。

7. Java Servlet

Servlet 是用 Java 编写的服务器端程序，主要功能是交互式地浏览和修改数据，生成动态 Web 内容。Servlet 运行于支持 Java 的应用服务器中。从实现上讲，Servlet 可以响应任何类型的请求，但绝大多数情况下，Servlet 只用来扩展基于 HTTP 的 Web 服务器。当 Servlet 被部署在应用服务器中后，由容器控制 Servlet 的生命周期。除特殊指定外，在容器启动的时候，Servlet 是不会被加载的，Servlet 只会在第一次请求的时候被加载和实例化。Servlet 在服务器的运行生命周期为：在第一次请求时被加载并执行一次初始化方法，接着执行正式运行方法，之后会被常驻，于是接下来每次被请求时直接执行正式运行方法，直到服务器关闭或被清理时执行一次销毁方法后实体销毁。

Java 服务器页面 JSP 是 Http Servlet 的扩展。由于 Httpservlet 大多是用来响应 HTTP 请求，并返回 Web 页面（例如 HTML、XML），所以在编写 Servlet 时会涉及大量的 HTML 内容，这给 Servlet 的书写效率和可读性带来很大障碍。JSP 通常是大多数的 HTML 代码中嵌入少量的 Java 代码，将程序员从复杂的 HTML 中解放出来，更专注于 Servlet 本身的内容。JSP 在首次被访问的时候被应用服务器转换为 Servlet，在以后的运行中，容器直接调用这个 Servlet，而不再访问 JSP 页面。所以，JSP 的实质仍然是 Servlet。

8. XML

XML（eXtensible Markup Language，可扩展标记语言）是一种用于标记电子文件使其具有结构性的标记语言。在电子计算机中，标记指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种的信息，例如文章等。可以选择国际通用的标记语言，例如 HTML，也可以使用像 XML 这样由相关人士自由决定的标记语言，这就是语言的可扩展性。

XML 可以从 HTML 中分离数据。即能够在 HTML 文件之外将数据存储在 XML 文档中，这样可以使开发者集中精力使用 HTML 做好数据的显示和布局，并确保数据改动时不会导致 HTML 文件也需要改动，从而方便维护页面。XML 也可以在不兼容的系统之间交换数据，可用于交换数据。XML 的发展和 Java 是相互独立的，但是，它和 Java 同样具有平台独立性。

9. JMS

JMS（Java Message Service，Java 消息服务）应用程序接口是一个 Java 平台关于面向消息中间件的 API，是用于和面向对象消息的中间件（Message Oriented Middleware，MOM）相互通信的应用程序接口。JMS 用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。JMS 是一个与具体平台无关的 API，绝大多数 MOM 提供商都对 JMS 提供支持，因此程序员可以在他们的分布式软件中实现面向消息的操作，这些操作将具有不同面向消息中间件产品的可移植性。JMS 提供企业消息服务，如可靠的消息队列、发布和订阅通信，以及有关推送/拉取（Push/Pull）技术的各个方面。

10. JTA

JTA(Java Transaction API)是 Java 平台上处理事务的应用程序接口。JTA 定义了一组标准 API,应用程序可以通过它访问各种事务监控。JTA 允许应用程序执行分布式事务处理,可以在两个或多个网络计算机资源上访问并且更新数据。JTA 事务比 JDBC 事务更强大。一个 JTA 事务可以有多个参与者,而一个 JDBC 事务则被限定在一个单一的数据库连接。

11. JTS

JTS(Java Transaction Service)是 Java 事务服务,规定了事务管理的实现方法,是 CORBA OTS 事务监控的基本实现。该事务管理器是在高层支持 JTA 规范,并且在较低层次实现 OMG OTS 规范[①]和 Java 映像。JTS 事务管理器为应用程序服务器、资源管理器、独立的应用以及通信资源管理器提供了事务服务。

12. Java Mail

Java Mail 是提供给开发者处理电子邮件相关的编程接口,它提供了一套邮件服务器的抽象类。Java Mail 不仅支持 SMTP 服务器,也支持 IMAP 服务器。

13. JAF

JAF(JavaBeans Activation Framework)是一个专用的数据处理框架,它用于封装数据,并为应用程序提供访问和操作数据的接口。JAF 的主要作用是让 Java 应用程序知道如何对一个数据源进行查看、编辑和打印等操作。应用程序通过 JAF 提供的接口可以完成:访问数据源中的数据、获取数据源数据类型、获知可对数据进行的操作,用户执行操作时,自动创建该操作的软件部件的实例对象。JavaMail 利用 JAF 来处理 MIME(Multipurpose Internet Mail Extensions)编码的邮件附件。MIME 的字节流可以被转换成 Java 对象,大多数应用都可以不需要直接使用 JAF。

除了上述 13 种技术规范之外,Java 中还有其他一些重要的技术规范。如 JMAPI(Java Management API)为异构网络系统、网络和服务管理的开发提供一整套丰富的对象和方法;JMF(Java Media Framework API)可以帮助开发者把音频、视频和其他一些基于时间的媒体放到 Java 应用程序或 Applet 小程序中去,为多媒体开发者提供了捕捉、回放、编解码等工具,是一个弹性的、跨平台的多媒体解决方案。从 JDK 1.5 开始,注解 Annotation 成为 Java 的重要特色。注解提供一种机制,将程序的元素,如类、方法、属性、参数、本地变量、包和元数据联系起来。这样编译器可以将元数据存储在 Class 文件中,虚拟机和其他对象可以根据这些元数据来决定如何使用这些程序元素或改变它们的行为。Java 管理扩展(Java Management Extensions,JMX)是一个管理和监控接口,用于管理应用程序、设备、系统等植入。JMX 可以跨越一系列异构操作系统平台、系统体系结构和网络传输协议,灵活地开发无缝集成的系统、网络和服务管理应用。Java 持久性接口(Java Persistence API,JPA)通过 JDK 5.0 注解或 XML 描述对象-关系表的映射关系,并将运行期的实体对象持久化到数据库中。

---

① Object Management Group's,Object Transaction Service(对象管理组,对象事务服务)。

5.3.4 企业级 Java 中间件

1. EJB 简介

JavaBeans 是 Java 技术中值得关注的技术。它是一个开放的标准的组件体系结构，它使用 Java 语言但独立于平台。JavaBean 是满足 JavaBeans 规范的 Java 类，通常定义了一个现实世界的事物或概念。JavaBean 的主要特征包括属性、方法和事件。在一个支持 JavaBeans 规范的开发环境中，可以可视化地操作 JavaBean，也可以使用 JavaBean 构造出新的 JavaBean。JavaBean 的优势还在于 Java 带来的可移植性。

EJB(Enterprise Java Bean)即企业级 JavaBean，是一个可重用的、可移植的 Java EE 组件。它将 JavaBean 概念扩展到 Java 服务端组件体系结构，这个模型支持多层的分布式对象应用。除了 EJB，典型的分布式结构还有前面章节介绍过的 DCOM 和 CORBA。

EJB 由封装了业务逻辑的多个方法组成。例如，一个 EJB 可以包括一个更新客户数据库中数据的方法的业务逻辑。多个远程和本地客户端可以调用这个方法。EJB 在容器中运行，允许开发者只关注于 Bean 中的业务逻辑而不用考虑像事务支持、安全性和远程对象访问等复杂和容易出错的事情。此外，EJB 以 POJO(Plain Ordinary Java Object，简单传统 Java 对象)的形式开发，开发者可以使用元数据注释来定义容器如何管理这些 Bean。

图 5-11 描述了 EJB 的环境构成。EJB 是以组件为基础的技术模型，组件运行在 EJB 容器之中，EJB 容器提供 EJB 组件运行的环境，并对 EJB 进行管理。EJB 容器一般包含在 EJB 服务器中，一个 EJB 服务器可以拥有一到多个 EJB 容器。EJB 容器可以在任何一个 EJB 服务器中部署，不需要定制专用的服务系统和容器。调用 EJB 组件的一方被称为 EJB 客户端，可以是运行在 Web 容器中的 JSP、Servlet 或其他的 Java 程序。EJB 客户端可以对 EJB 进行重新组装和重新定义，能够利用这一特性和其他组件一起构造出符合使用需求的应用系统，而且可以同时为多个系统提供服务。因此，EJB 结构平台完全是独立的，具有极强的独立性。

基于 EJB 开发出来的应用程序不用进行任何修改，即可被复制到另外的平台中。EJB 部署到应用服务器上之后，容器会自动生出三个对象，分别是 Home 对象、Remote 对象、Enterprise Java Bean 对象，客户端通过 JNDI 机制，绑定与定位 EJB，就可以调用它来完成各种功能。

2. EJB 的种类

EJB 中有三种类型的组件：会话 Bean(Session Bean)、实体 Bean(Entity Bean)和消息驱动 Bean(Message Driven Bean)。会话 Bean 执行独立的、解除耦合的任务，如检查客户的信用记录。实体 Bean 是一个复杂的业务实体，它代表数据库中存在的业务对象。消息驱动 Bean 用于接收异步 JMS 消息。以下简要介绍 EJB 3.0 规范中的这些类型。

(1) 会话 Bean

会话 Bean 主要负责业务逻辑的处理，代表着业务流程中“处理订单”这样的操作。“会话”意味着 Bean 只存在于某一时间段，而当服务器容器关闭或故障时将被销毁。根据会话状态的保持性，会话 Bean 可分为有状态或者无状态。

无状态会话 Bean 不维护会话状态，其服务任务须在一个方法调用中结束。它没有中间状态，也不保持追踪方法传递的信息。一个无状态业务方法的每一次调用都独立于它的前

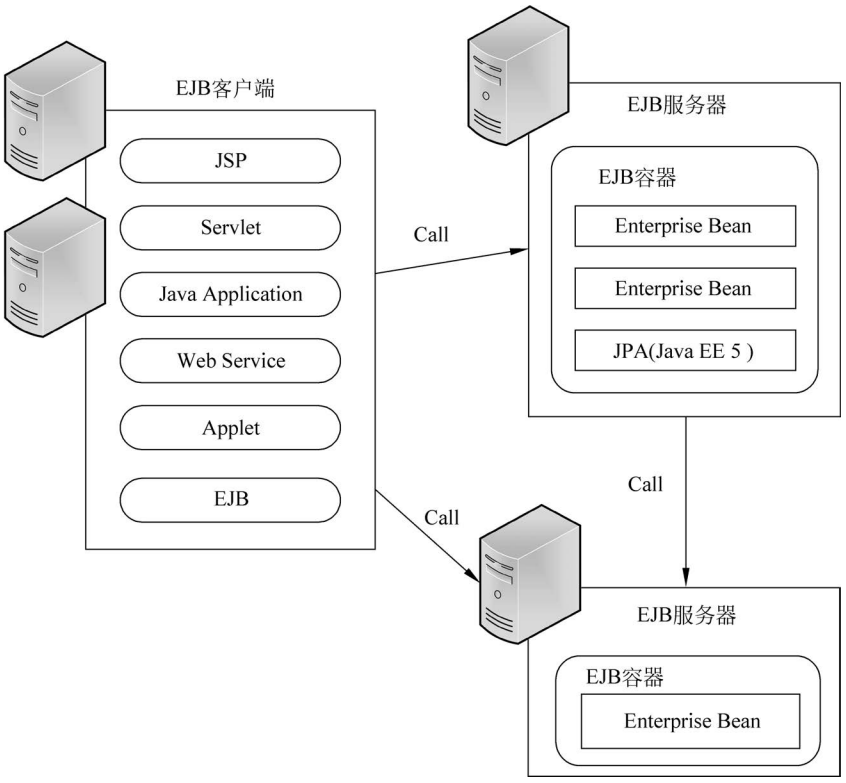


图 5-11　EJB 的环境构成

一个调用。例如，当计算税费额的方法被调用时，只需计算税费值并返回给调用者，没有必要存储税费的值以及调用者的信息。多个用户可以无差别地调用无状态会话 Bean 的方法。因此，为了能够重用一些无状态会话 Bean 的实例，一般会采用“会话池”技术。即容器可以维护一定数量的实例来为大量的客户端服务。会话池中的无状态会话 Bean 能够被共享，当客户端请求一个无状态的 Bean 实例时，它可以从池子中选一个空闲状态的 Bean 实例进行调用处理。当请求达到了会话池设置的最大数量，新请求将被加入队列，以等待无状态 Bean 的服务。EJB 通过添加@Stateless 标注来指定一个 Java Bean 作为无状态会话 Bean 被部署和管理。

有状态的会话 Bean 维护一个跨越多个方法调用的会话状态。当一个客户端请求一个有状态会话 Bean 实例时，客户端将会得到一个会话实例，该 Bean 的状态只为该客户端维持。例如在线购物篮应用，当客户开始在线购物时，从数据库获得客户的详细信息，定义购物篮列表。当客户从购物篮中增加或者移除商品等操作时，这些用户信息和购物篮信息，都将被再次被访问。不同的客户端调用，都将产生新的有状态的会话 Bean 实例。但会话 Bean 是暂时的，因为该状态在会话结束、系统崩溃或者网络失败时都不会被保留。通过向方法增加标注@Remove，可以告知容器某个方法调用结束后，该有状态会话 Bean 实例应该被移除。

（2）实体 Bean

实体 Bean 是管理持久性数据的一个对象，有可能使用几个相关的 Java 对象，可以通过主键实现唯一性。通过添加@Entity 注释，可以把某类指定为实体 Bean。实体 Bean 代表

数据库中的持久性数据，如客户表中的一行或者员工表中的一条员工记录。实体 Bean 还可以在多个客户端之间共享。如某个员工实体 Bean 可以由多个客户端用于计算某员工的年薪或者更新员工地址。与之前的版本相比，EJB 3.0 中的实体 Bean 是纯粹的 POJO，它表达和 Hibernate 持久化实体对象同样的概念。可通过注解来定义映射，注解分别是逻辑映射注解和物理映射注解，通过逻辑映射注解可以描述对象模型、类之间的关系等，而物理映射注解则描述了物理的模式、表、列、索引等。

(3) 消息驱动 Bean

消息驱动 Bean(MDB)提供了一个实现异步通信的方法，该方法比直接使用 Java 消息服务(JMS)更容易。可以通过创建 MDB 接收异步 JMS 消息。当一个业务执行的时间很长，而执行结果无须实时向用户反馈时，适合使用消息驱动 Bean。例如，订单成功后给用户发送一封电子邮件或发送一条短信等。对客户机来说，消息驱动 Bean 是一个在服务器上实现某些业务逻辑的 JMS 消息使用者。客户机发送消息到 JMS Destination(Queue 或 Topic)来访问消息驱动 Bean。在服务器端，EJB 容器处理 JMS 队列和上下文所要求加载处理的大部分工作。当 JMS 消息到达时，它向相关的 MDB 发送消息，激发该 MDB 来处理消息。消息驱动 Bean 实例没有会话状态，因此当不涉及服务客户机消息时，所有的 Bean 实例都是等同的。

5.4 Spring 框架

5.4.1 Spring 框架的历史

JavaEE(J2EE)应用程序的广泛实现是在 2000 年左右开始的。它的出现带来了诸如事务管理之类的核心中间层概念的标准化，但是在实践中并没有获得绝对的成功。其主要原因是其开发效率、开发难度和实际的性能都令人失望。特别是 EJB 要严格地继承各种不同类型的接口，类似的或者重复的代码大量存在，而配置相对复杂和单调。因此，对于大多数初学者来说，学习 EJB 实在是一件代价高昂的事情。较低的开发效率，极高的资源消耗，都造成了 EJB 的使用困难。

Spring 出现的初衷就是为了解决类似的这些问题，它的一个最大的目的就是使 Java EE 开发更加容易。Spring 兴起于 2003 年，是一个轻量级的 Java 开源框架，由 Rod Johnson 所著的 *Expert One-On-One J2EE Development and Design* 中阐述的部分理念和原型衍生而来。Spring 是为解决企业应用开发的复杂性而创建的，主要优势之一就是其分层架构。分层架构允许使用者选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。Spring 使用基本的 JavaBean 来完成以前只可能由 EJB 完成的事情。它可以独立或在现有的应用服务器上运行，用于服务器端的开发。

总的来说，Spring 是一个分层的 Java SE/EE 全栈式(full stack)轻量级开源框架。Spring 的核心是控制反转(Inversion of Control，IoC)、依赖注入(Dependency Injection，DI)和面向切面编程(Aspect Oriented Programming，AOP)等。同时，Spring 与 Struts、Hibernate 等单层框架不同，它致力于提供一个统一的、高效的方式构造整个应用，并且可以将单层框架以最佳的组合揉合在一起建立一个连贯的体系。因此，从简单性、可测试性和松耦合的角度而言，任何 Java 应用都可以从 Spring 中受益，其用途不限于服务器端的开发。

5.4.2 Spring 的体系结构

Spring 框架是一个分层架构，它包含一系列的功能要素并被分为大约 20 个模块。如图 5-12 所示，这些模块分为核心容器（Core Container）、面向切面编程（Aspect Oriented Programming）、设备支持（Instrument）、数据访问及集成（Data Access/Integration）、Web、报文发送（Messaging）、Test 等模块。

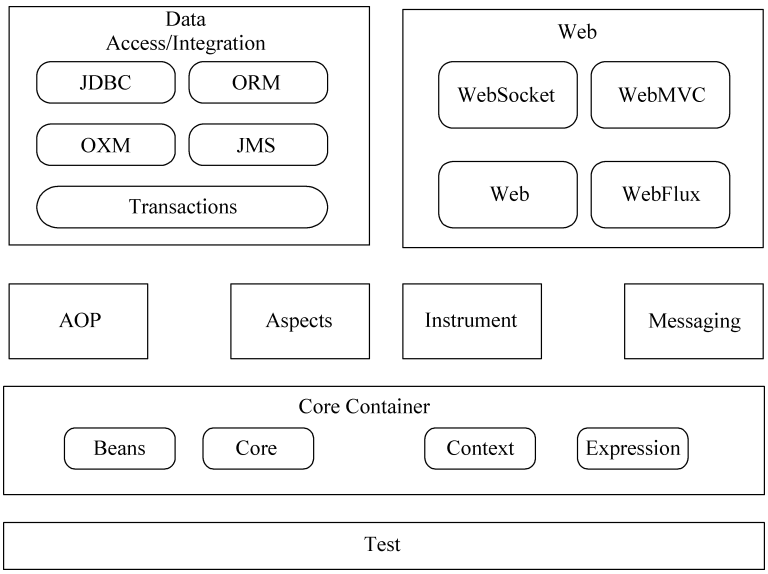


图 5-12　Spring 体系结构

1. 核心容器模块

Spring 核心容器提供 Spring 框架的基本功能，它由 Spring-Beans、Spring-Core、Spring-Context、Spring-Expression 4 个模块组成。

Spring 以 Bean 的方式组织和管理 Java 应用中的各个组件及其关系，它将管理对象称为 Bean，其 Beans 模块提供了工厂模式的经典实现 BeanFactory。BeanFactory 使用控制反转对应用程序的配置和依赖性规范与实际的应用程序代码进行了分离。

Spring-Core 是 Spring 的核心控制反转（IoC）和依赖注入（Dependency Injection，DI）的基本实现。控制反转是一种设计思想，即将设计好的对象交由容器控制，而不是传统的在对象内直接控制。

Spring-Context 模块建立在 Core 和 Beans 模块的基础之上，提供一个框架式的对象访问方式，是访问定义和配置的对象的媒介。它扩展了 BeanFactory，为它添加了 Bean 生命周期控制、框架事件体系以及资源加载透明化等功能。此外，该模块还提供了许多企业级支持，如邮件访问、远程访问、任务调度等。ApplicationContext 是该模块的核心接口，它是 BeanFactory 的超类，与 BeanFactory 不同，ApplicationContext 容器实例化后会自动对所有的单实例 Bean 进行实例化与依赖关系的装配，使之处于待用状态。

Spring-Expression 模块提供了强大的表达式语言去支持运行时查询和操作对象图。它是对 JSP 2.1 规范中规定的统一表达式语言（Unified EL）的扩展。该语言支持设置和获取属性值、属性分配、方法调用、访问数组、集合和索引器的内容、逻辑和算术运算、变量命名

以及从 Spring 的 IoC 容器中以名称检索对象。除此之外，它还支持列表投影、选择以及常用的列表聚合。

2. 面向切面编程模块

AOP 模块通过配置管理特性，直接将面向切面的编程功能集成到了 Spring 框架中，提供了一个符合 AOP 要求的面向切面的编程实现，允许定义方法拦截器和切入点，将代码按照功能进行分离，降低了它们之间的耦合性。同时，AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务，通过 AOP 模块，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

3. 设备支持模块

Spring-Instrument 模块提供了在特定应用程序服务器中使用的类工具支持和类加载器实现。它是基于 Java SE 中的 java.lang.instrument 进行设计的，是 AOP 的一个支援模块，主要作用是在 JVM(Java Virtual Machine)启用时，生成一个代理类，程序员通过代理类在运行时修改类的字节，从而改变一个类的功能，实现 AOP 的功能。

4. 数据访问及集成模块

Data Access/Integration 模块包括 JDBC(Java Database Connectivity)、ORM(Object Relational Mapping)、OXM(Object-to-XML-Mapping)、JMS(Java Message Service)和 Transactions 模块。

其中，Spring-JDBC 模块是 Spring 提供的 JDBC 抽象框架的主要实现模块，用于简化 Spring JDBC。主要提供 JDBC 模板方式、关系数据库对象化方式、SimpleJdbc 方式、事务管理来简化 JDBC 编程。它的主要实现类是 JdbcTemplate、SimpleJdbcTemplate 以及 NamedParameterJdbcTemplate。Spring-ORM 模块是 ORM 框架支持模块，主要集成 Hibernate、Java Persistence API (JPA) 和 Java Data Objects (JDO)，可用于资源管理、数据访问对象(DAO)的实现和事务策略。Spring-OXM 模块主要提供一个抽象层以支撑 OXM(Object-to-XML-Mapping，将 Java 对象映射成 XML 数据，或者将 XML 数据映射成 Java 对象)。Spring-JMS 模块提供对 JMS 的支持，能够发送和接收信息。Spring-Transactions 模块是 Spring JDBC 事务控制实现模块。该模块支持编程和声明式事务管理，用于实现特殊接口和所有 POJO(普通 Java 对象)的类。

5. Web 模块

Web 模块由 Spring-Web、Spring-Web MVC、Spring-WebSocket、Spring-Webflux 四个模块组成。Spring-Web 模块为 Spring 提供了最基础的 Web 支持，它主要建立于核心容器之上，通过 Servlet 或者 Listeners 来初始化 IoC 容器以及 Web 应用上下文。Spring-WebMVC 模块是一个 Web-Servlet 模块，实现了 Spring MVC(Model-View-Controller)的 Web 应用。Spring-WebSocket 提供了 WebSocket 功能，它提供了通过一个套接字实现全双工通信的功能。Spring-Webflux 是一个非阻塞函数式 Reactive Web 框架，可以用来建立异步的、非阻塞事件驱动的服务，并且它的扩展性非常好。

6. 报文发送模块

Messaging 模块仅包括一个模块，它的主要职责是为 Spring 框架集成一些基础的报文传送应用。

7. Test 模块

Test 模块也只由一个模块组成，主要为测试提供支持。

5.4.3 Spring 容器中的依赖注入

1. POJO 对象

Spring 和 EJB 框架结构都有一个共同的核心设计理念：将中间件服务传递给耦合松散的 POJO 对象(Plain Old Java Objects)。POJO 对象可以理解为简单的实体类，实际就是普通的 JavaBeans。POJO 有一些私有的(private)的参数作为对象的属性，然后针对每个参数定义了 get 和 set 方法作为访问的接口。

以下是一个 POJO 对象的例子。

```
    //Student.java
package xmu.edu.cn;

public class Student{
  private long id;
  private String name;
  public void setId(long id) {
    this.id = id;
  }
  public void setName(String name) {
    this.name = name;
  }
  public long getId() {
    return id;
  }
  public String getName() {
    return name;
  }
}
```

POJO 类没有任何特别之处，无须装饰，不继承自某个类。但是，它却可以作为 Spring 的组件(Component)使用，发挥强大的功能。

2. Spring 依赖注入

传统编程通过 new 关键字创建对象，对象须负责寻找或创建其依赖的对象，一般通过显式代码来进行对象关联。但在基于 Spring 的应用中，这些组件及对象生存在 Spring 容器中(如图 5-13 所示)，而容器将负责对象的创建、对象间的关联，并管理对象的生命周期。负责对象创建和关联的，就是 Spring 框架的“依赖注入”(Dependency Injection，DI)技术。

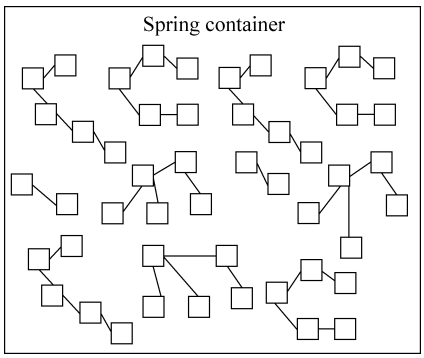


图 5-13　Spring 容器

DI 通过截取执行上下文或在运行时信息，为组件和对象注入它们之间的依赖关系。DI 能提供类似胶水的功能，自动地把某些对象“缠绕”起来实现对象之间的协作。对象本身并

不关心这种“缠绕”,对这种框架结构也没有什么依赖。因此,开发者可专注于业务逻辑和脱离框架的 POJO 类对象单元测试。除此之外,由于 POJO 类并不需要继承框架的类或实现其接口,开发者能够极其灵活地搭建继承结构和建造应用。以下通过一个简单的例子来说明 Spring 中的依赖注入技术。

首先,定义一个接口 Service,其提供服务方法 serve()。

```
//Service.java
package xmu.edu.cn;

public interface Service {
  void serve();
}
```

教师给学生上课。因此,定义教师类 Teacher 和学生类 Student。其中,教师类将实现 Service 接口,而学生类实现方法 learn()。类的定义非常简单,与普通的 Java 类基本相似,唯一不同的是多了@Component 的标注,告知 Spring 容器这是一个组件。

```
// Teacher.java
package xmu.edu.cn;
import org.springframework.stereotype.Component;

@Component
public class Teacher implements Service {
  private String course = "English";
  public void serve() {
    System.out.println("Giving the " + course + " lecture!");
  }
}

  // Student.java
package xmu.edu.cn;
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.*;

@Component
public class Student {
  private Service service;
  public Student(Service s){
    service = s;
  }
  public void learn(){
    service.serve();
  }
}
```

在定义了以上简单的类之后,Spring 可通过另外的配置文件,来定义各个类之间的关系。例如,Spring 可以通过 Java 代码、XML、注解,以及三者混合的模式来进行配置。在以下的 Java 代码中,标注@Configuration 说明这是一个 Java 方式的配置文件,标注@Bean 则告知 Spring 容器该方法将返回一个对象 Bean,并同时注册到容器的上下文环境中。该实例对象的 id 默认为类名的小写字符,Spring 容器生成的对象默认是仅有一份的,方法内的代

码将生成该类的实例对象；即使多次调用该方法，也将返回同一个实例。

```
//StudentConfig.java
package xmu.edu.cn;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;

@Configuration
public class StudentConfig {
  @Bean
  public Service service(){
    return new Teacher();
  }
  @Bean
  public Student student(){
    return new Student(service());
  }
}
```

因此，通过以上配置文件，Spring 容器将生成两个实例对象 Bean，id 分别是 service 和 student。以下的代码，则可以验证容器是否辅助生成了 student 对象，并进行调用。

```
// StudentJavaMain.java
package xmu.edu.cn;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class StudentJavaMain {
  public static void main(String[] args) throws Exception {
    ApplicationContext context =
          new AnnotationConfigApplicationContext(StudentConfig.class);
    Student s = context.getBean(Student.class);
    s.learn();
  }
}
```

基于 Java 配置使用 AnnotationConfigApplicationContext 类。该类是 ApplicationContext 接口的一个实现，能够注册所注释的配置类。在本例中，配置类是使用@Configuration 标注声明的 StudentConfig。因此，注册配置类将根据配置类 StudentConfig 所描述的逻辑初始化上下环境 context。Spring 将自动扫描标注为@Bean 的类并生成组件对象，其对应的 Bean 就是 service(类型为 Teacher)和 student(类型为 Student)。随后可以使用 getBean 方法来获取相关的 Bean，并调用其业务方法。编写 Java 的配置类并将其注册到 Spring 上下文非常简单。当然，也可以通过 XML 文件进行同样的配置。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="service" class="xmu.edu.cn.Teacher"/>
  <bean id="student" class="xmu.edu.cn.Student">
```

```
        <constructor-arg ref="service"></constructor-arg>
    </bean>
</beans>
```

其中，除了加载基本的 XML 模式，<beans>标签下还通过<bean>标签定义了 id 为 service 的 Teacher 对象和 id 为 student 的 Student 对象。同时，service 是 student 对象的构造函数的参数。当然，除了设置构造函数的参数，在 XML 配置文件中还可以设置对象的属性值和参数。也就是说，Spring 允许通过 Java 和 XML 的配置文件，动态设置各个对象的依赖和关联；而不必在源组件的代码中预先定义。其验证的代码如下。

```
//StudentXmlMain.java
package xmu.edu.cn;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class StudentXmlMain {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext("META-INF/spring/student-bean.xml");
        Student s = context.getBean(Student.class);
        s.learn();
        context.close();
    }
}
```

当容器中对象很多的时候，配置文件本身也是一个不小的工作。为了克服这个问题，Spring 容器引入了自动编织技术(Auto Wiring)。即容器根据对象自身的需求，自动匹配各对象之间的关联。自动编织技术的关键标注是@Autowired 和@ComponentScan。

```
// StudentAutoConfig.java
package xmu.edu.cn;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class StudentAutoConfig {
}
```

在新的配置文件中，并未显式标注容器内的组件，而是添加了@ComponentScan 标注。Spring 容器将从源代码中查看标注为@Component 的对象，并尝试自动匹配和关联各对象的关系。Teacher 类被标注为 Component，因此 Spring 容器将为之创建一个对象；Student 类也被标注为 Component，Spring 容器也将尝试为之创建一个对象。Spring 首先调用 Student 类的构造函数，该构造函数需要类型为 Service 的对象作为参数。因此，Spring 将自动从容器中寻找类型为 Service 的对象。如果找到，则自动把该对象注入作为 Student 类的构造函数的参数，自动构建和编织成功；否则，Spring 将抛出异常，提示无法自动创建该对象。

XML 配置文件，也可以注明自动编织。代码如下。

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< beans xmlns = "http://www.springframework.org/schema/beans"
            xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance"
            xmlns:context = "http://www.springframework.org/schema/context"
            xsi:schemaLocation = "http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring - beans - 3.0.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring - context - 3.0.xsd">
  < context:component - scan base - package = "xmu.edu.cn" />
</beans >
```

其中，<context:component-scan base-package="xmu.edu.cn"/>表示将进行组件扫描和自动编织，且 base-package 指出了具体扫描组件时的范围。

此外，Spring 还可以通过@Scope 注释 Bean 的使用范围，以及利用 initMethod 和 destroyMethod 管理 Bean 的生命周期。感兴趣的读者可以深入研究。

5.4.4 Spring 中的 AOP 编程

以下用一个例子来说明 Spring 容器中面向切面的编程。

首先，定义一个教师类 Teacher，其实现 Service 接口中的 serve()方法。即教师提供的服务是讲课。但是在讲课之前，一般要求学生要按要求入座，并把手机静音；课堂讲完了，还会问学生是否有问题，进行答疑。因此，按传统的编程方法，serve()方法定义如下。

```
//Service.java
package xmu.edu.cn;

public interface Service {
  void serve();
}

    //Teacher.java
package xmu.edu.cn;

import org.springframework.stereotype.Component;
@Component
public class Teacher implements Service {
  private String course = "English";
  public void serve() {
    //嵌入对学生对象的调用，使其坐好座位，手机静音
    System.out.println("students, please taking the seats… ");
    System.out.println("students, please silencing cell phones…");
    System.out.println("Giving the " + course + " lecture!");
    System.out.println("class is over, is there any questions?");
  }
}
```

可以发现，Teacher 类的 serve 方法包含很多与上课无关的代码，且须显式地调用 Student 对象来进行操作，代码耦合度高。如何能使老师只关注课堂，只负责认真讲课，而把学生入座之类的事情放到其他的地方呢？在这个例子中，老师上课是核心业务，而学生入

座、关手机等可以看作横向逻辑业务，即“方面”。因此，可以把 Teacher 方法中的 serve 方法作为连接点，在其之前和之后分别切入相应的关于学生的方法。

```
//Student.java
package xmu.edu.cn;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class Student {
  @Pointcut("execution( ** xmu.edu.cn.Teacher.serve(..))")
  public void giveLecture() {}
  @Before("giveLecture()")
  public void silenceCellPhones() {
    System.out.println("Silencing cell phones...");
  }
  @Before("giveLecture()")
  public void takeSeats() {
    System.out.println("Taking seats...");
  }
  @AfterReturning("giveLecture()")
  public void askQuestion() {
    System.out.println("Class is over, is there any Questions?");
  }
  @AfterThrowing("giveLecture()")
  public void haveClassAccident() {
    System.out.println("A teaching accident, ask for an investigation... ");
  }
}
```

在以上代码中，@Pointcut("execution(** xmu.edu.cn.Teacher.serve(..))")定义了一个连接点。execution(** xmu.edu.cn.Teacher.serve(..))声明了切入点表达式。execution()是最常用的切点函数，其语法如下。

execution(* com.sample.service.impl..*.*(..))整个表达式可以分为以下五个部分。

- execution()：表达式主体。
- 第一个 * 号：表示返回类型，* 号表示所有的类型。
- 包名：表示需要拦截的包名，后面的两个句点表示当前包和当前包的所有子包，com.sample.service.impl 包、子孙包下所有类的方法。
- 第二个 * 号：表示类名，* 号表示所有的类。
- *(..)：最后这个星号表示方法名，* 号表示所有的方法，后面括弧里面表示方法的参数，两个句点表示任何参数。

该连接点定义为函数 public void giveLecture() {}，以方便在其他地方引用。@Before("giveLecture()")和@AfterReturning("giveLecture()")则声明该函数在切入点之前及之后执行。方法 public void takeSeats()则是注入的 Advice。

在本例中，Teacher 是目标对象，Student 类中定义了“方面”。那如何织入呢？在 Spring 框架中使用配置文件，把方面应用到目标对象。如代码所示，配置文件 ClassRoomConfig 定义了 Student 和 Teacher 这两个 Bean，并进行自动扫描，在容器中生成对象。同时，标注 @EnableAspectJAutoProxy 声明了将创建代理对象，进行自动编织。

```
//ClassRoomConfig.java
package xmu.edu.cn;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration
@EnableAspectJAutoProxy
@ComponentScan
public class ClassRoomConfig {
  @Bean
  public Student student() {
    return new Student();
  }
  @Bean
  public Service teacher(){
    Teacher p = new Teacher();
    p.setName("Lai");
    return p;
  }
}
```

Teacher 类则简化为如下代码。教师无须关注学生的状况，可以只关注讲课。事实上，Teacher 类与 Student 类是独立的模块，Teacher 类甚至不用知道有学生类的存在。

```
//Teacher.java
package xmu.edu.cn;
import org.springframework.stereotype.Component;
@Component
public class Teacher implements Service {
  private String course = "English";
  public void serve() {              //只负责专心讲课
    System.out.println("Giving the " + course + " lecture!");
  }
}
```

可以用以下代码进行验证。

```
//StudentMain.java
package xmu.edu.cn;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class StudentMain {
  public static void main(String[] args) throws Exception {
    ApplicationContext context = new
```

```
            AnnotationConfigApplicationContext(ClassRoomConfig.class);
        Service teacher = context.getBean(Service.class);
        teacher.serve();
    }
}
```

运行结果如下。

```
Silencing cell phones...
Taking seats...
Lai : Giving a lecture...
Class is over, is there any Questions?
```

在以上示例中，Spring 通过配置文件把一些组件对象编织起来，并可以实现自动装配。这大大降低了各组件之间的耦合度，且让开发人员可以更加聚焦到自己的模块上。

小　　结

Web 容器是位于应用程序/组件和服务器平台之间的接口集合，它可以管理对象的生命周期、对象与对象之间的依赖关系，是减少用户工作量的一个有效方法。本章首先介绍了 Web 服务器的概念，并阐述其工作原理。接着介绍了 Web 容器及 Web 容器所用到的技术和思想——解耦合、控制反转、面向切面编程。然后介绍了 Java EE 和 Spring 框架的主要技术组成部分。最后通过编程案例，帮助读者学习 Web 容器编程和 AOP 编程的原理。

思　考　题

1. 简述 Web 服务器的工作原理。
2. MVC 框架由哪些部分组成？有哪些优点？
3. 什么是 Web 容器？请概述 Web 容器的作用。
4. 解释解耦、控制反转、依赖注入的概念。
5. 什么是面向切面的编程？其主要用于解决什么问题？
6. 在面向切面编程中，连接点、通知、切入点、方面、编织分别指什么？
7. 请简述 Java EE 框架的基本组成模块。
8. 请列出三种 Java EE 的主流技术并说明其内容。
9. EJB 容器由哪些模块组成？EJB 组件有什么特性？
10. Spring 产生的背景是什么？与 EJB 有哪些异同点？

实验 3　Spring 和 AOP 编程

一、实验目的

掌握 Spring MVC 编程，能够基于 Spring 或者 Spring Boot 实现简单的网页页面；同时掌握基于 Spring 的面向切面的编程。

二、实验平台和准备

操作系统：Windows 或 Linux 系统。

需要的软件：Java JDK(1.8 或以上版本)、Spring、IDEA 或者 Eclipse 等开发环境，安装 MySQL 数据库(https://www.mysql.com)。

三、实验内容和要求

构建一个校友的信息收集系统。系统有两个表格：管理员权限表 Admin，校友信息表 Alumni。Admin 数据表定义的字段有{id，名字，密码，权限}，Alumni 数据表定义的字段有{姓名、性别、生日、入学年份、毕业年份、工作城市/地区、工作单位、职务、手机、邮箱、微信}。

采用 Spring MVC 框架，构建该校友信息系统，实现以下功能。

(1) 随机生成 20 个录入员，生成 100 个校友用户信息。验证操作用户(录入人员等)的登录信息是否正确，把校友的数据插入到数据库中，对校友目录进行检索、修改、删除、统计等功能。同时，可对信息进行增删改的操作。操作用户登录后，显示本次登录的次数和上一次登录的时间，操作用户退出时，显示用户连接的时间长度，并把此次登录记录到数据库。

(2) 构建一个日志记录的切面。要求对于所有的 Alumni 表的查询操作，记录各个操作的时间、用户，读取内容存入 ReadLog 表中；对于所有的 Alumni 表的更新(更新和删除)操作，记录各个操作的时间、用户，修改的新值和旧值，存入 UpdateLog 表中。

(3) 构建一个用户记录的切面。要求对于所有的登录操作，记录各次登录的时间、用户，存入 UserLog 表中；对于所有的登出操作，记录各次登录的时间、用户，存入 UserLog 表中；对于用户新的输入操作，记录其表单值，存入 InsertLog 表中。

四、扩展实验内容

可自学前端编程和人机交互设计的知识，针对网页界面进行美化；学习关系型数据库的相关知识。

同时构建一个安全验证的切面。要求对于所有的 Alumni 表的查询操作，验证用户已经登录；如果用户没有登录，先导航到登录页面；对于所有的 Alumni 表的更新(更新和删除)操作，在 Read 权限的基础上验证用户具有 Update 的权限。如果没有，该操作取消，并导航到错误页面；对于所有的 Alumni 表的汇总和下载操作，验证用户具有 Aggregate 权限；如果没有，该操作取消，并导航到错误页面。