中间件技术基础与Java实践

Middleware Technology

第六章 Web容器

李会格 讲师

E-mail: 1034434100@qq.com

前言

- 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。

大纲

- **Web服务器**
  - ✓ **Web服务器概述**
  - ✓ **Web服务器的工作原理**
  - ✓ **Web服务器和MVC框架**
- **Web容器简介**
  - ✓ **容器的概念**
  - ✓ **解耦合、控制反转及依赖注入**
  - ✓ **面向切面的编程AOP**
- **Java EE框架**
  - ✓ **Java EE概述**
  - ✓ **Java EE框架组成**
  - ✓ **Java EE的主要技术**
  - ✓ **企业级Java中间件**
- **Spring框架**
  - ✓ **Spring框架的历史**
  - ✓ **Spring的体系结构**
  - ✓ **Spring容器和依赖注入**
  - ✓ **Spring容器和AOP编程**

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。它起源于1989年3月，是由欧洲量子物理实验室CERN所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，无需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。

Web服务器技术概述

- 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。
  - 超文本标记语言(Hyper Text Markup Language, HTML)描述信息资源
  - 统一资源标识符(Uniform Resource Locator, URL)定位信息资源
  - 超文本转移协议(HyperText Transfer Protocol, HTTP)请求信息资源

- 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)。

Web服务器概述

- 目前主流的Web服务器有Apache 、 Nginx 、 IIS 。

- Web服务器的设计初衷是一个静态信息资源发布媒介，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。

大纲

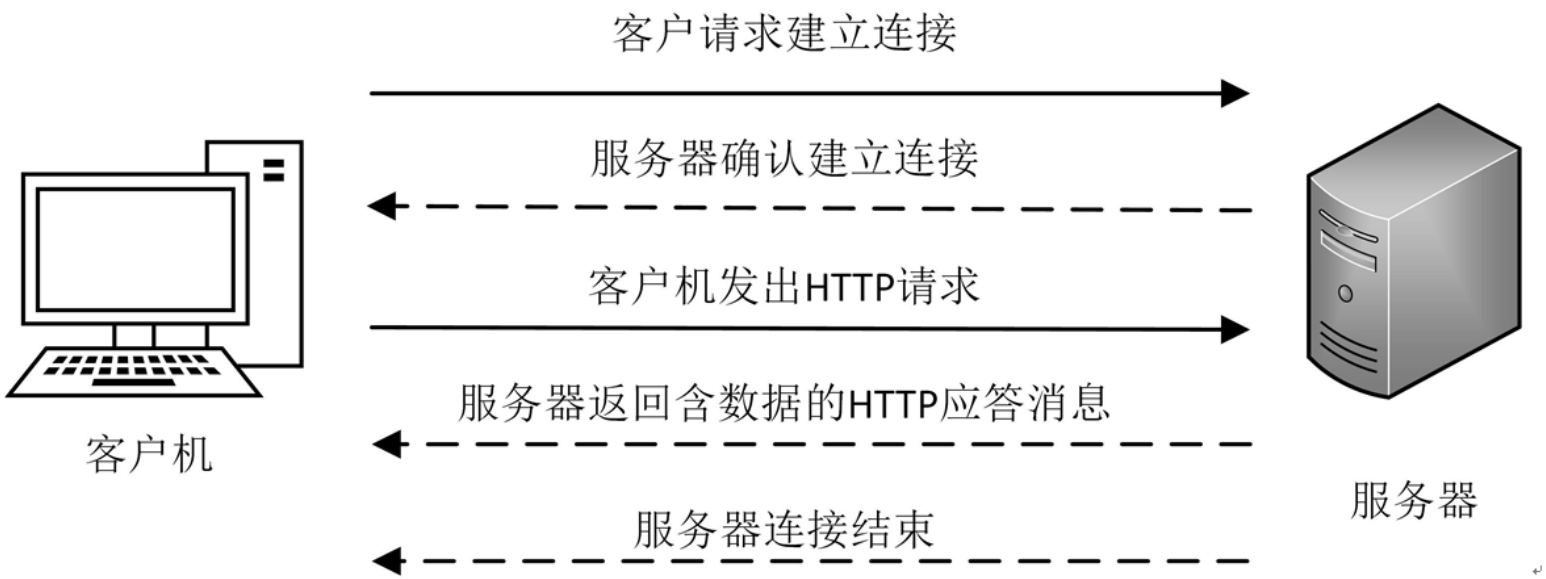
- **Web服务器**
  - Web服务器概述
  - **Web服务器的工作原理**
  - Web服务器和MVC框架
- **Web容器简介**
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- **Java EE框架**
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- **Spring框架**
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

Web服务器架构

- Web服务器采用的是浏览器/服务器（B/S）结构，其作用是整理和储存各种网络资源，并响应浏览器等Web客户端的请求，返回客户所需的资源，向Web客户端提供文档和显示界面。

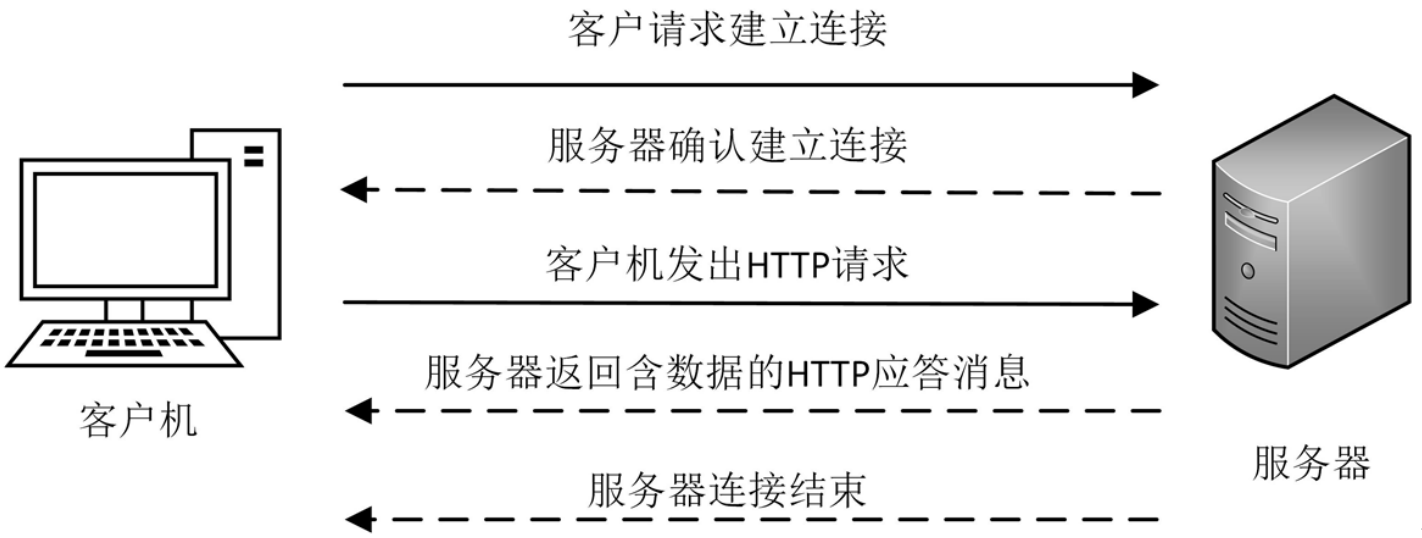
- Web服务器传送页面使浏览器可以浏览，而应用程序服务器提供的是客户端应用程序可以调用的方法。换句话说，Web服务器专门处理HTTP请求，而应用程序服务器通过多种协议为应用程序提供商业逻辑。

Web服务器工作过程



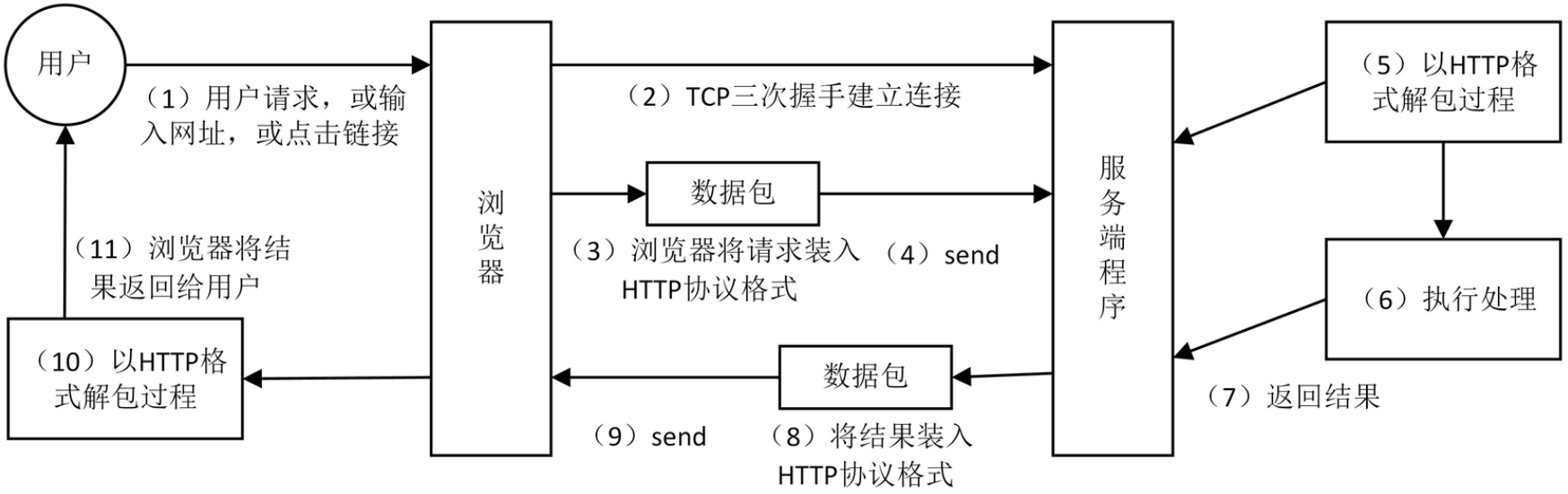
- Web服务器的工作过程一般可分成如下4个步骤：连接过程、请求过程、应答过程以及关闭连接。
  - 连接过程是Web服务器及其浏览器之间建立一种连接的过程，用户可以通过查看套接字socket这个虚拟文件确定连接是否建立。

Web服务器工作过程



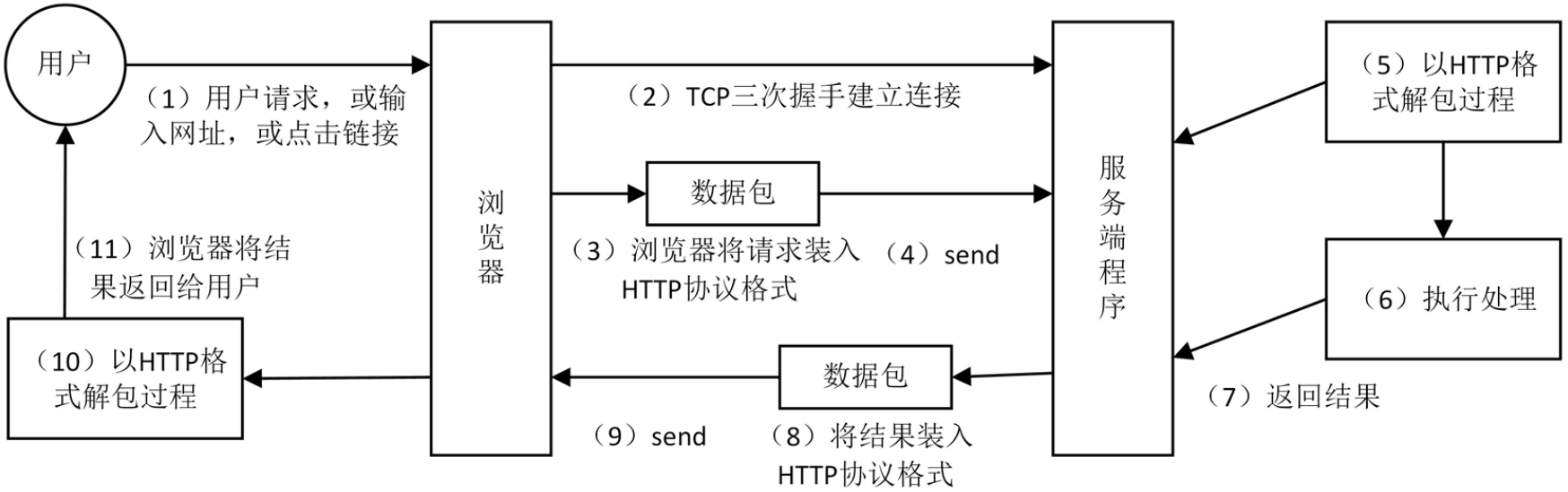
- 请求过程是Web浏览器运用socket这个文件向其服务器提出各种请求。
- 应答过程是HTTP协议把在请求过程中所提出来的请求传输到Web服务器，进而实施任务处理；然后运用HTTP协议把任务处理的结果传输到Web浏览器，同时在Web的浏览器上面展示上述所请求的界面。
- 关闭连接是当应答过程完成以后，Web服务器和其浏览器之间断开连接的过程。

Web服务器工作原理



- 在典型的Web应用中，用户在浏览器中输入网址或点击链接，浏览器获得事件后与Web服务器建立TCP连接，(1)(2)就是连接过程。
- 如果连接成功，浏览器将用户事件按照HTTP协议格式打包成一个数据包，向服务器提出各种请求，(3)(4)是请求过程。

Web服务器工作原理



- 服务端程序接到请求后，以HTTP协议格式解包请求，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协议格式，返回给浏览器端，(5)-(11)是应答过程。
- 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、紧密相联，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。

Web服务器应答过程

- 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请求动态组装而成。对于不同的请求，动态网页返回的结果一般不同。

- 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）给一些其它的程序例如CGI脚本、JSP脚本、Servlets、ASP脚本、服务器端JavaScript，或者一些其它的服务器端技术。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。

大纲

- **Web服务器**
  - Web服务器概述
  - Web服务器的工作原理
  - **Web服务器和MVC框架**
- **Web容器简介**
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- **Java EE框架**
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- **Spring框架**
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

Web服务器框架

- Web服务器可基于不同的编程语言进行编程，近来各厂商和开源社区提出了各种框架和架构来优化Web服务器的工作过程。比如SSH架构（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。

MVC框架

- Web服务器通常采用MVC框架（Model View Controller，模型-视图-控制器）来抽象客户端和服务器的访问流程。

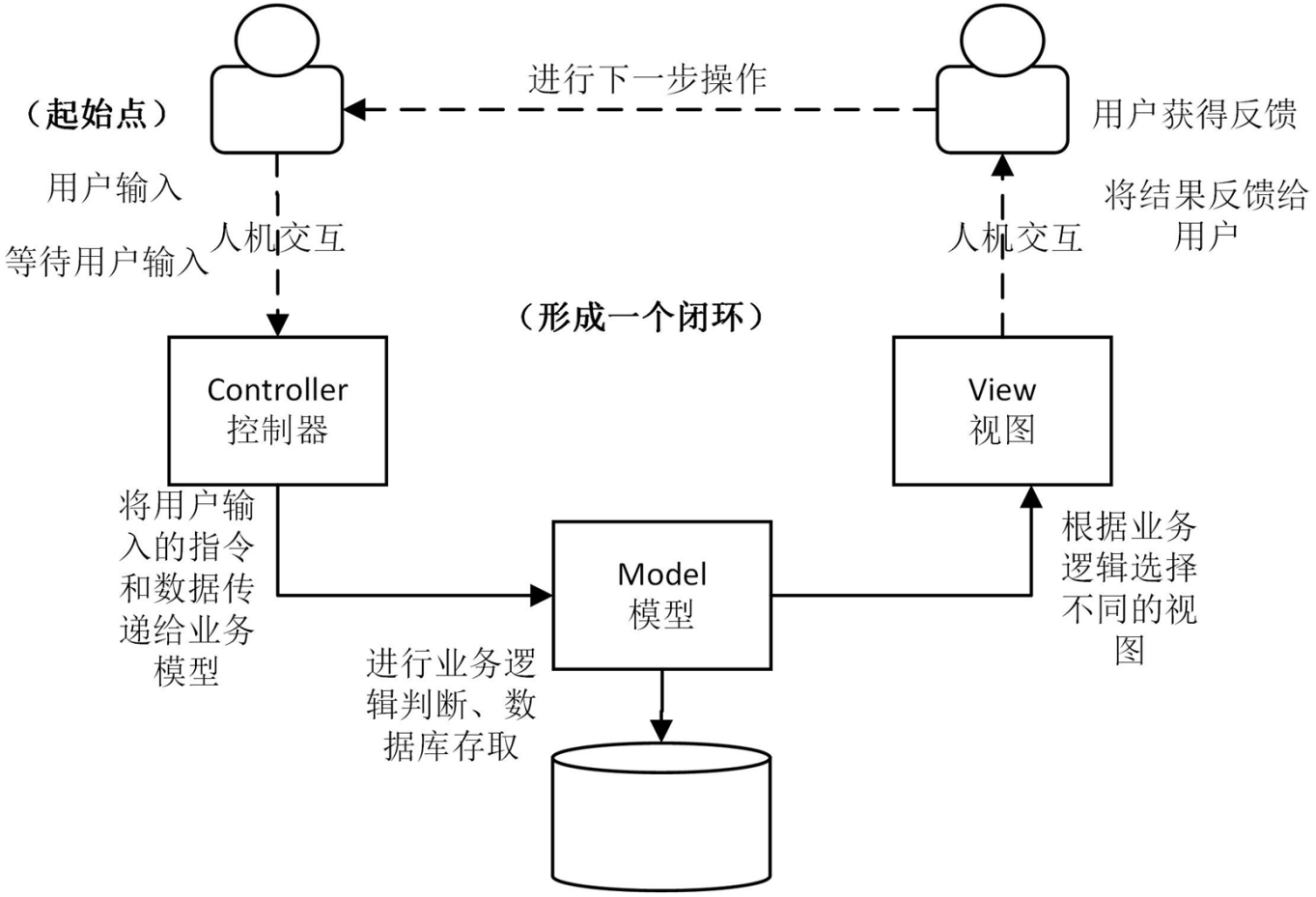
- MVC用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

MVC框架

- 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。

MVC框架图示

- 下图是一个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。

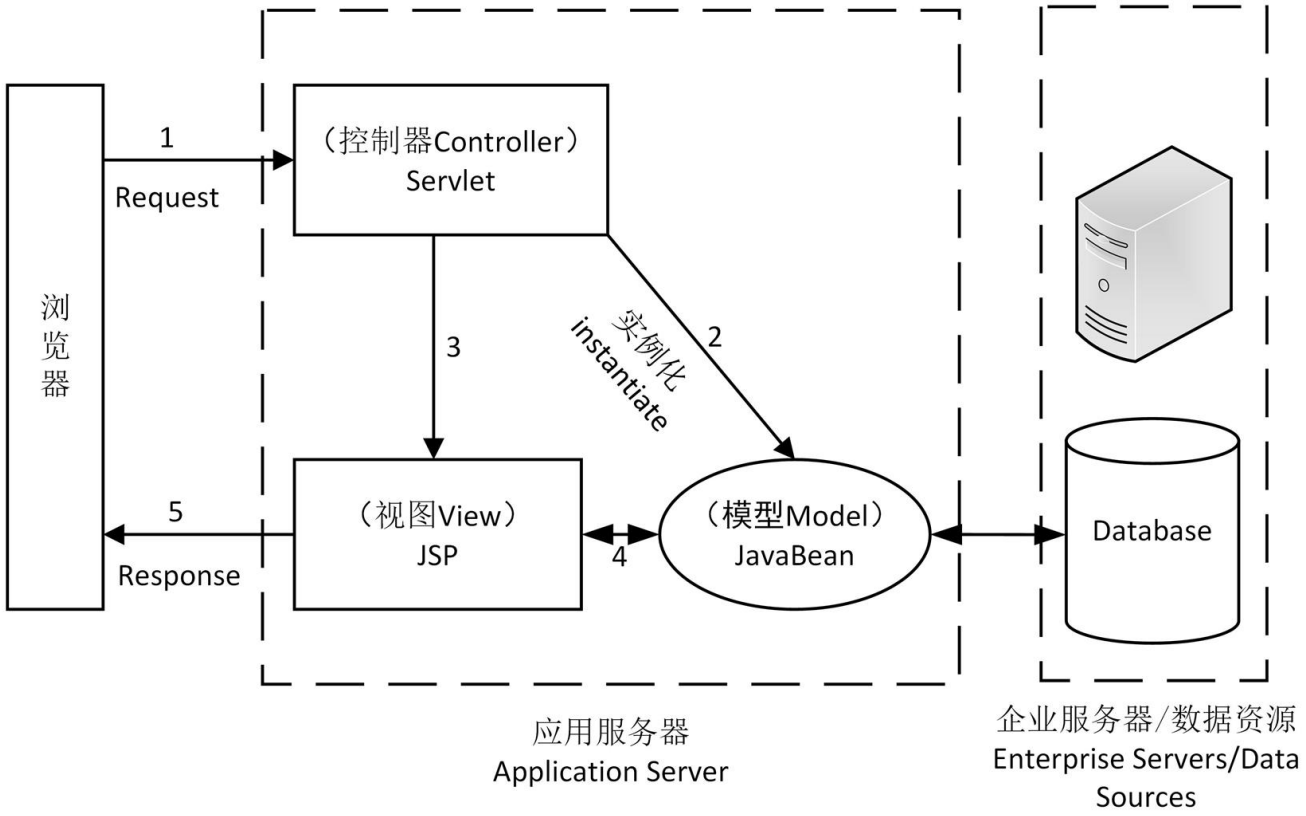


MVC框架优点

- MVC 分层有助于管理复杂的应用程序，使得程序员在一个时间内专门关注一个方面。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。

MVC框架实例



- JavaEE是MVC框架的一个具体实现。如图所示，JavaEE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。

大纲

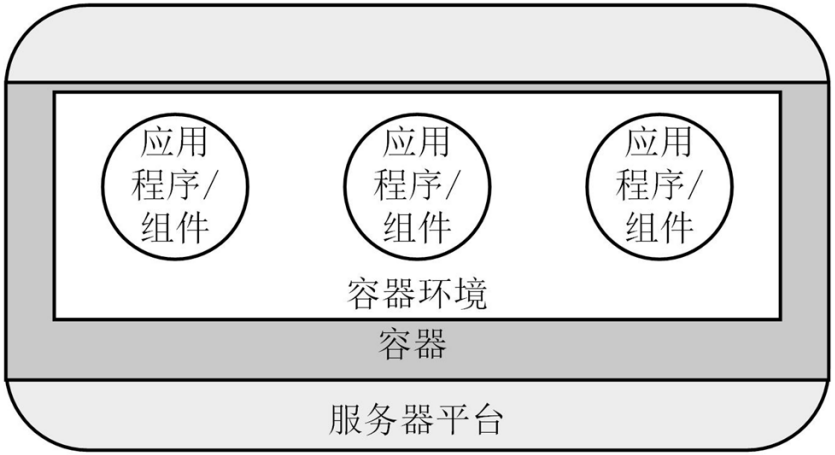
- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

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强求和处理业务逻辑。这也出现了Web容器的概念。
- 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。

Web容器

- 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：
  - ✓ 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，不需要修改容器的代码。



- 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、Servelet）提供环境，是组件直接跟容器中的环境变量交互，而不必关注其他系统问题。

容器的优点

- 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 Object）对象，而容器负责对象的创建、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，而无需用户编码来产生对象或建立对象与对象之间的依赖关系。

容器的优点

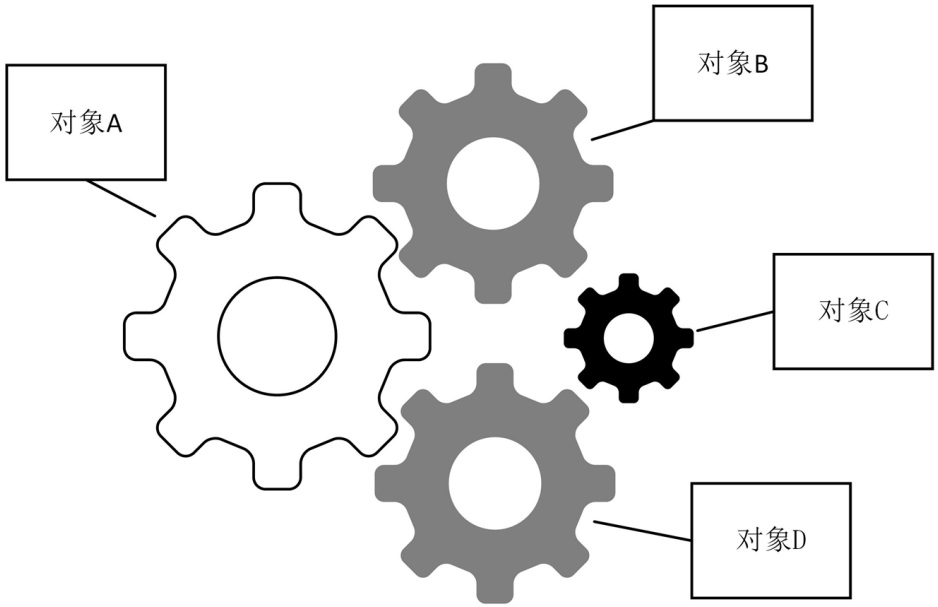
- Web容器是自动化例行程序。
- Web容器提供一个应用框架。
  - 屏蔽和掩藏例行的或者复杂的事物（如事务、安全或持久性）；
  - 支持弹性配置，使得开发者只把关注聚焦到应该的地方，是简化企业软件开发的关键。
- 因此，容器技术可以提高代码重用率、开发者的生产力及软件的质量。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

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，一同实现系统的业务逻辑。

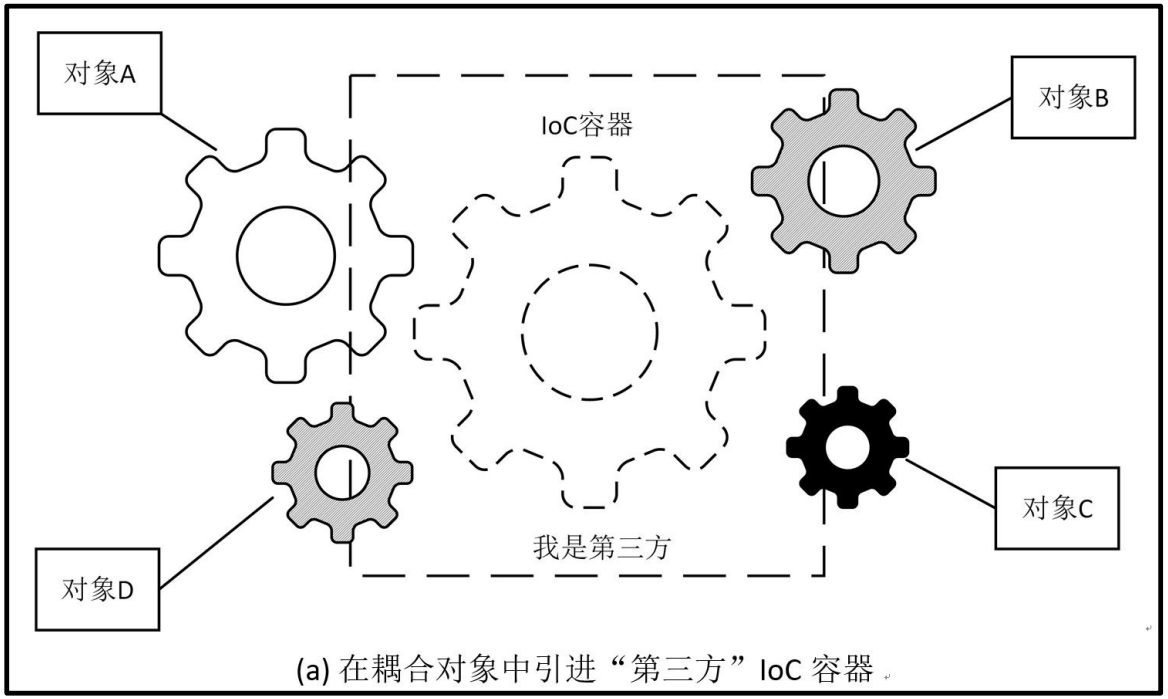


- 右图是软件系统中耦合的对象的示例，各模块之间互相依赖，耦合度较高，对象之间耦合度过高的系统，不利于系统的维护。

  - 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。

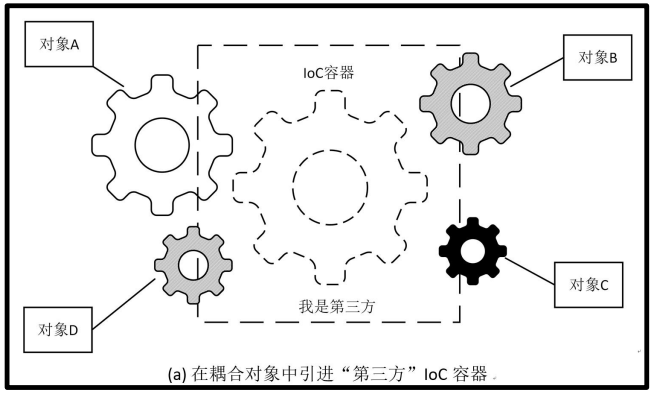
解耦合、控制反转

- 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 IoC（Inverse of Control）理论，用来实现对象之间的“解耦”。



(a) 在耦合对象中引进“第三方”IoC 容器

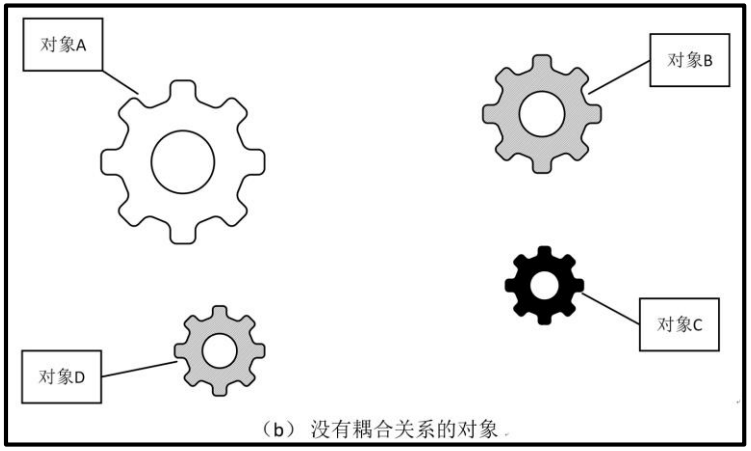
解耦合、控制反转



(a) 在耦合对象中引进“第三方”IoC 容器

- 在图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。它起一种类似“粘合剂”的作用，把系统中的所有对象粘合在一起发挥作用。

解耦合、控制反转



（b）没有耦合关系的对象。

- 而如果拿掉中间的容器（即容器无需开发人员实现），图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需关注其他的工作。

解耦合、控制反转

- 在没有引入IoC容器之前，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
- 在引入IoC容器之后，对象A与对象B之间失去了直接联系。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注入到对象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。

控制反转

- 把复杂系统分解成相互合作的对象。这些对象通过封装以后，内部实现对外部是透明。
  - 问题的复杂度降低；
  - 对象可以灵活地被重用和扩展。

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做“依赖注入”(Dependency Injection, DI)。
  - 所谓依赖注入，就是IoC容器在运行期间，动态地将某种依赖关系注入到对象之中。

- 所以，依赖注入(DI)和控制反转(IoC)是从不同的角度的描述的同一件事情。
  - 控制反转是一种设计模式，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。

- 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。
- 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。

依赖注入、控制反转

- 依赖注入和控制反转都符合依赖倒置(Dependence Inversion Principle, DIP)的设计原则。
  - 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。

- 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JavaEE项目均采用了IoC框架产品。

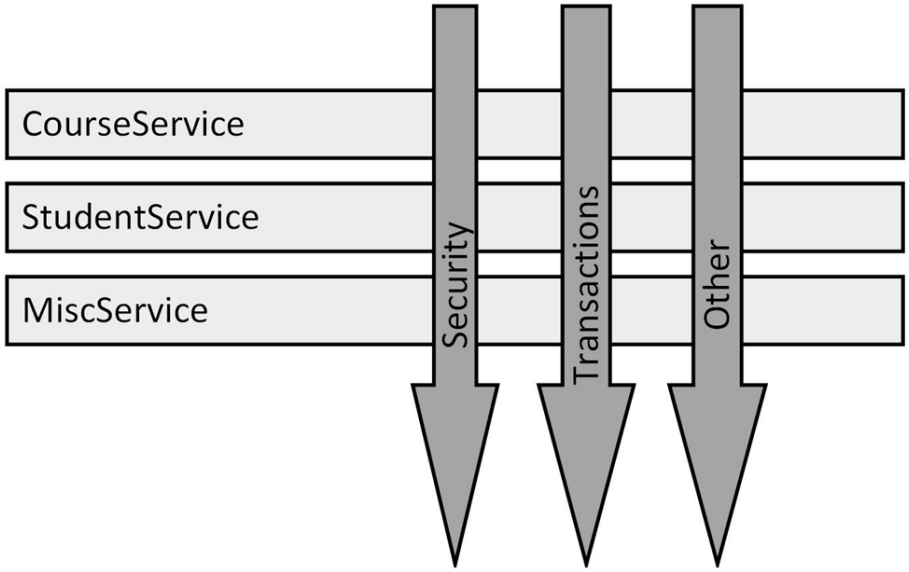
大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

AOP概述

- AOP（Aspect Oriented Programming）即面向切面的编程，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OOP(Object Oriented Programming)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。

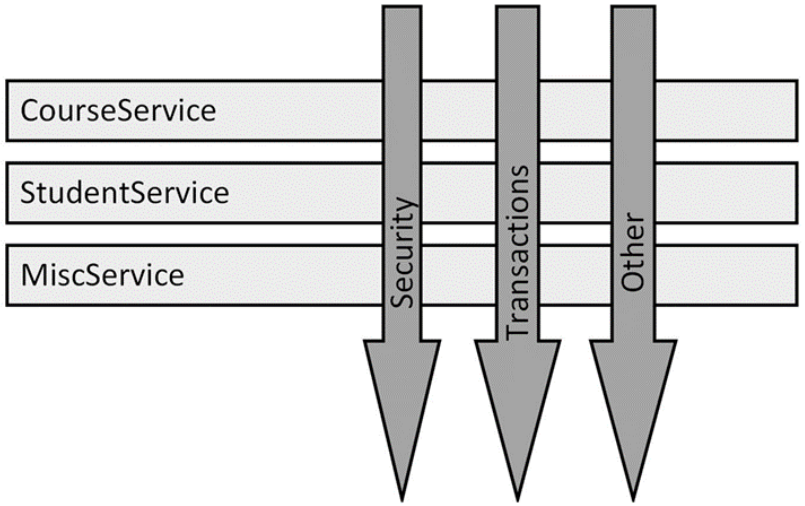
AOP由来



- 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到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。

AOP由来

- 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。比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。



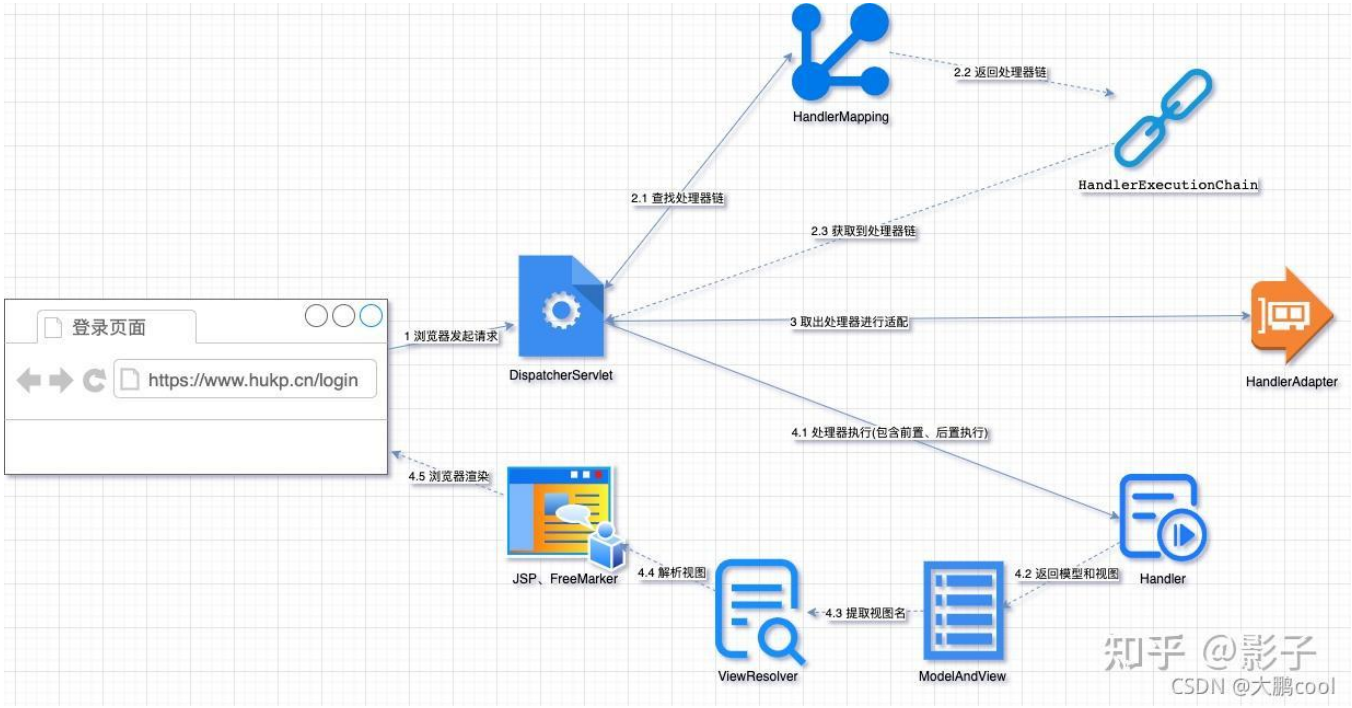
- AOP提供的解决方法是使这些服务模块化。
  - AOP将应用系统分为核心业务逻辑和横向通用逻辑两部分。通用逻辑，也就是所谓的“方面”。
  - 在Spring中提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务（例如审计和事务管理）进行内聚性的开发。
  - 应用对象只需实现业务逻辑，而无需负责其它的系统级关注点，例如日志或事务支持。

AOP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”通知。许多AOP框架（包括Spring）都是以拦截器做通知模型，维护一个“围绕”连接点的拦截器链。

- 拦截器Interceptor是Spring框架中提供的，拦截处理器中需要执行预先设定的代码。



拦截器的的功能

前端发起请求时，根据业务需要执行预先设定的代码。

- 用户校验：校验用户是否携带JWT令牌，且该令牌是否合法。
- 权限控制：校验用户是否包含访问权限。

AOP核心概念

- “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。

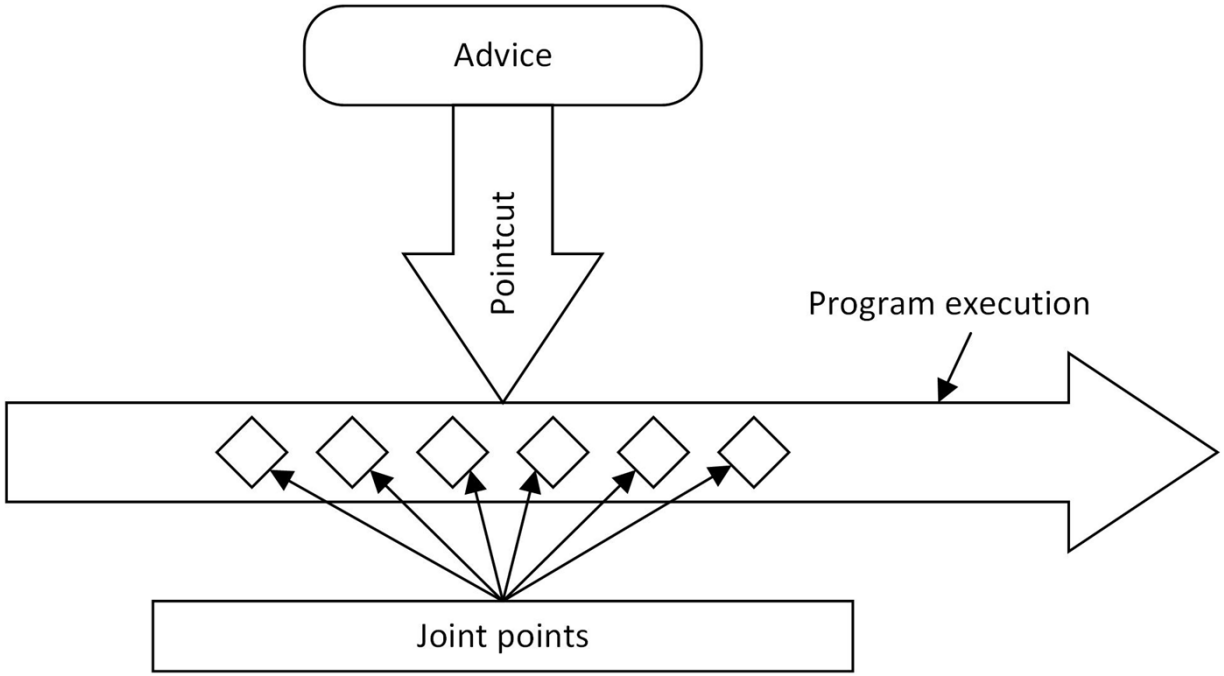
3. 切入点（Pointcut）：指定通知将被引发的一系列连接点的集合。AOP框架必须允许开发者指定切入点.

- 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则定义了Where。

常用正则表达式

用户名	/^[a-z0-9_-]{3,16}$/
密码	/^[a-z0-9_-]{6,18}$/
十六进制值	/^#?([a-f0-9]{6}|[a-f0-9]{3})$/
电子邮箱	/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/ /^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/
URL	/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
IP 地址	/((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)/ /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
HTML 标签	/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
删除代码\\注释	(?<!http:|\S)//.*$
Unicode编码中的汉字范围	/^[\u2E80-\u9FFF]+$/

AOP核心概念



方面功能在一个或多个连接点处编织到程序的执行中

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。比如，事务管理、安全管理等都是横切关注点。

AOP核心概念

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，你可为一个类引入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。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

Java EE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。

- Java EE 是在 Java SE 的基础上构建的，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。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。

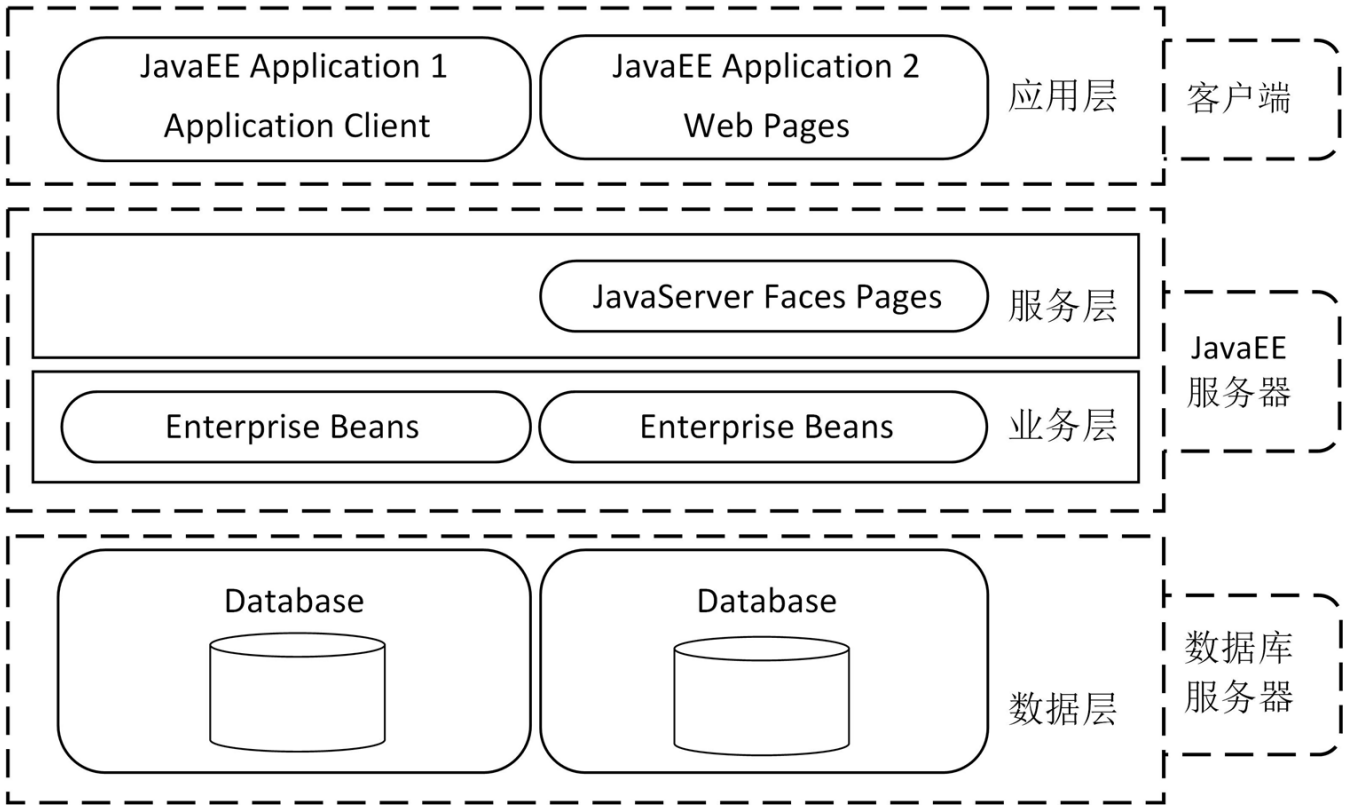
Java EE概述

- 此外，Java EE还提供一些额外的服务，比如命名、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。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。
  - 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，比如事务管理、安全检查、资源池和状态管理等。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

Java EE框架组成



Java EE体系结构

- 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。

Java EE框架组成

- 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。比如，如何从客户端接受信息、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。

➢ 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。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

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（Java数据库连接）
2. JNDI（Java命名和目录接口）
3. EJB（企业Java Beans）
4. RMI（远程方法调用）
5. Java IDL（Java接口定义语言）
6. JSP（Java服务器页面）
7. Java Servlet
8. XML（可扩展标记语言）
9. JMS（Java消息服务）
10. JTA（Java事务API）
11. JTS（Java事务服务）
12. Java Mail
13. JAF（数据处理框架）

1. JDBC

- JDBC（Java Database Connectivity）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。

- 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。

2. JNDI

- JNDI（Java Name and Directory Interface，Java命名和目录接口）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。

- JNDI为开发人员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。

- JNDI已经成为JavaEE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。
  - 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。
  - 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。

4. RMI和5. Java IDL

- 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。

- Java IDL（Java 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ORB（Object Request Broker）
  - 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类并和其它ORB一起展开的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 公司主导创建的一种动态网页技术标准。
  - JSP页面由大多数的HTML代码和嵌入其中的少量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它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。
  - 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协议的Web服务器。
  - 在容器启动的时候，servlet是不会被加载的，只有在第一次请求时被加载并执行一次初始化方法，之后会常驻，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。
  - 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技术将程序员从复杂的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和9. JMS

- XML（eXtensible Markup Language，可扩展标记语言）是一种标记语言。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具有平台独立性，可以在不兼容的系统之间交换数据。

- JMS（Java Message Service，Java消息服务）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（MOM, Message Oriented Middleware）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、以及有关推拉（Push/Pull）技术的各个方面。
  - 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。

10. JTA和 11. JTS

- JTA（Java Transaction API，Java事务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。

- JTS（Java Transaction Service，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 和Java印象。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和13. JAF

- JavaMail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。

- 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。

其他技术规范

- 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，比如：
  - 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。
  - 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。

其他技术规范

- 从在JDK1.5开始，注解Annotation成为Java的重要的特色。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，虚拟机和其它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。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

EJB简介

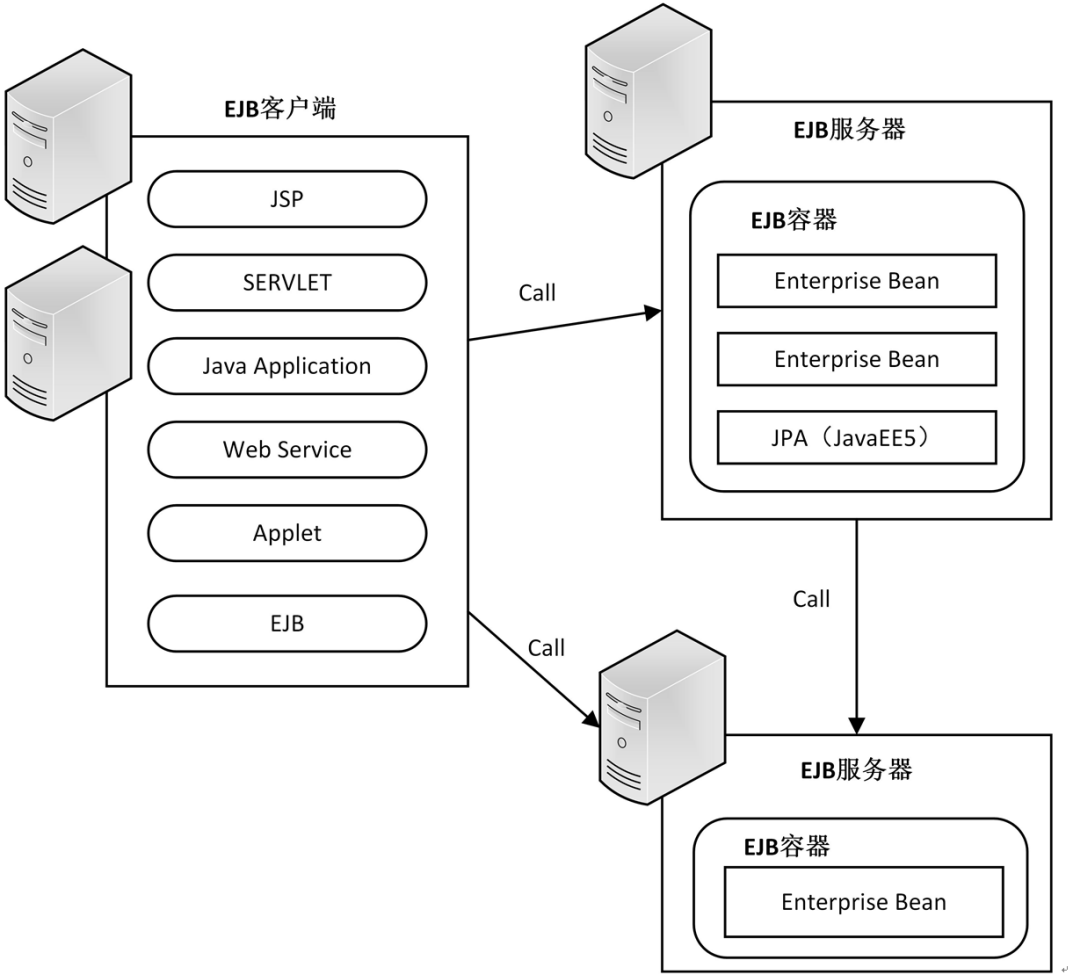
- JavaBeans是Java技术中的一个开放的标准的组件体系结构，它使用Java语言但独立于平台。
- JavaBean是满足JavaBeans规范的Java类，通常定义了一个现实世界的事物或概念。
  - JavaBean的主要特征包括属性、方法和事件。
  - 在一个支持JavaBeans规范的开发环境中，可以可视化地操作JavaBean，也可以使用JavaBean构造出新的JavaBean。
  - JavaBean的优势还在于Java带来的可移植性。

- 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简介

- 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）即简单传统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。

EJB环境构成



- EJB是以组件为基础，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，和其他组件一起构造出符合使用需求的应用系统。因此，EJB结构平台完全是独立的，具有极强的独立性。

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。

- 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

- 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。无状态会话Bean不维护会话状态，一个无状态业务方法的每一次调用都独立于它的前一个调用。多个用户可以无差别的调用无状态会话Bean的方法。
  - 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。

- 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。

有状态会话Bean

- 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。

实体Bean

- 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，EJB3.0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。

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。当一个业务执行的时间很长，而执行结果无需实时向用户反馈时，适合使用消息驱动Bean。比如，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。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

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。

- Spring兴起于2003年，是一个轻量级的Java开源框架， Spring出现的初衷是为了使Java EE开发更加容易。
  - 其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。

Spring框架概述

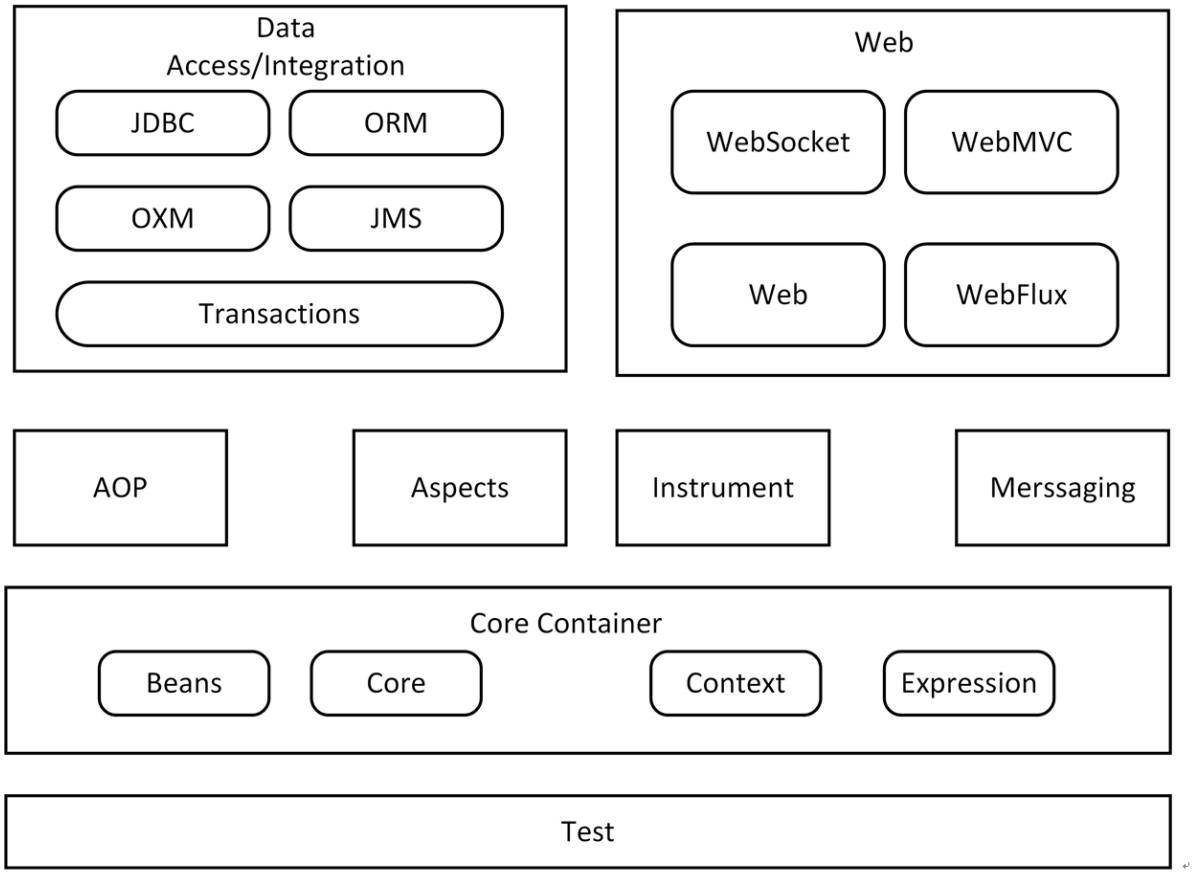
- 总的来说，Spring是一个分层的Java SE/EE 全栈式（full stack）轻量级开源框架。
  - Spring的核心是控制反转（Inversion of Control，IoC）、依赖注入（Dependency Injection，DI）和面向切面编程（Aspect Oriented Programming，AOP）等等。
  - 同时，Spring与Struts、Hibernate等单层框架不同，它致力于提供一个统一的、高效的方式构造整个应用，并且可以将单层框架以最佳的组合揉和在一起建立一个连贯的体系。

- 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。

大纲

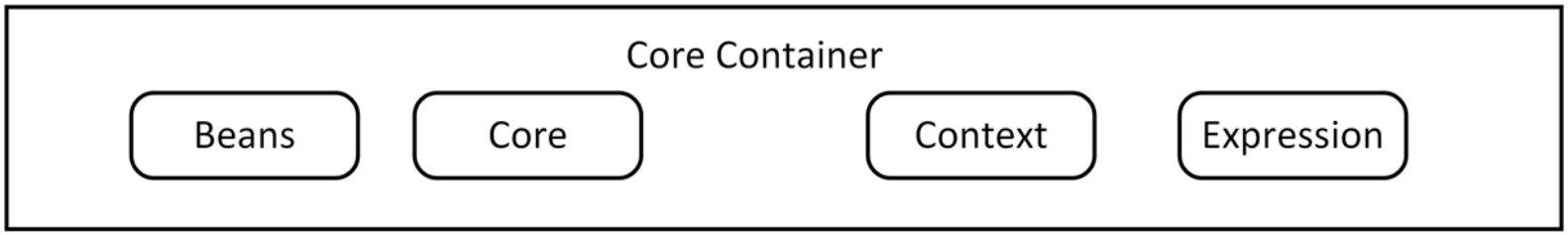
- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

Spring的体系结构



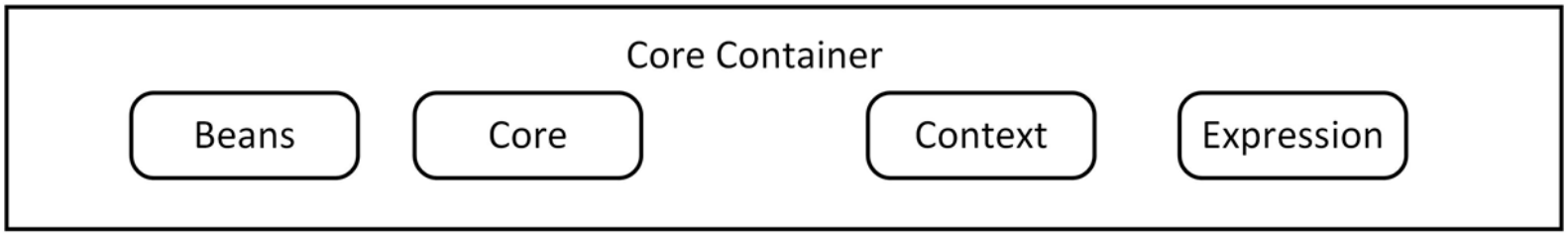
- Spring框架是一个分层架构，它包含一系列的功能要素并被分为大约20个模块。

1. 核心容器



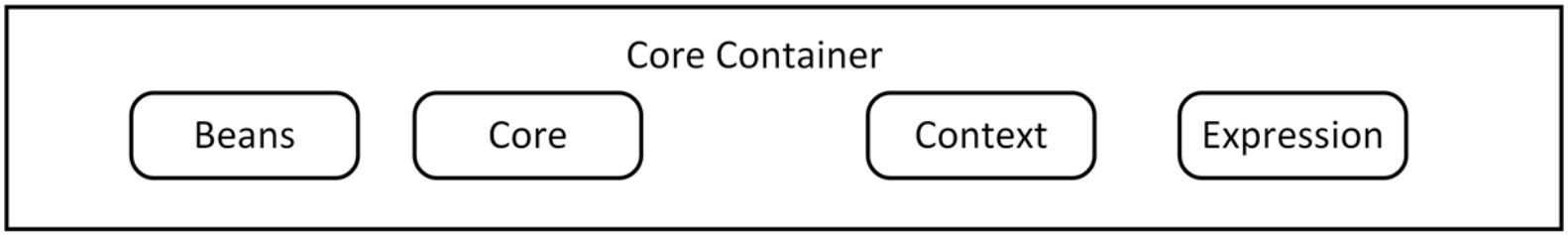
- 核心容器提供Spring框架的基本功能，它由4个模块组成。
  - Spring将管理对象称之为bean，以bean的方式组织和管理Java应用中的各个组件及其关系，其Beans模块提供了工厂模式的经典实现BeanFactory。BeanFactory 使用控制反转对应用程序的配置和依赖性规范与实际的应用程序代码进行了分离。
  - Spring-Core是Spring的核心控制反转IoC和依赖注入DI(Dependency Injection)的基本实现。控制反转是一种设计思想，即将设计好的对象交由容器控制，而不是传统的在对象内直接控制。

1. 核心容器



- 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。

1. 核心容器



- 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JSP2.1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。

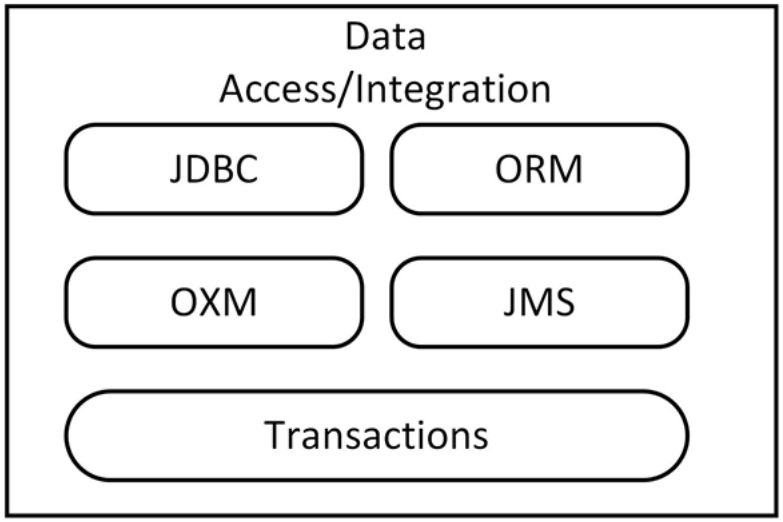
2. AOP和3. Instrument

- 面向切面编程模块
  - AOP模块通过配置管理特性，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。

- 设备支持模块
  - Spring-Instrument模块是基于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. Data Access/Integration

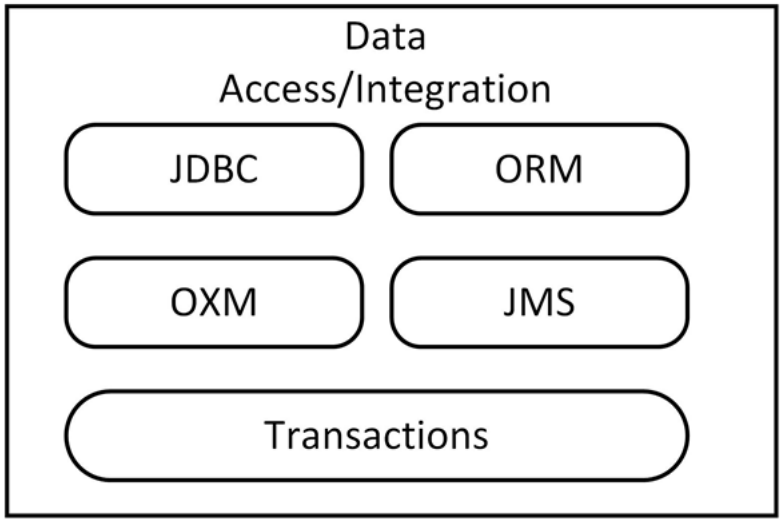
- 其中，Spring-JDBC（Java Database Connectivity）模块是Spring提供的JDBC抽象框架的主要实现模块，用于简化Spring JDBC。主要提供JDBC模板方式、关系数据库对象化方式、SimpleJdbc方式、事务管理来简化JDBC编程。



- Spring-ORM （Object Relational Mapping）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。

4. Data Access/Integration

- Spring-OXM模块主要提供一个抽象层以支撑OXM。OXM 是Object-to-XML-Mapping的缩写，将java对象映射成 XML数据，或者将XML数据映射成 java对象。

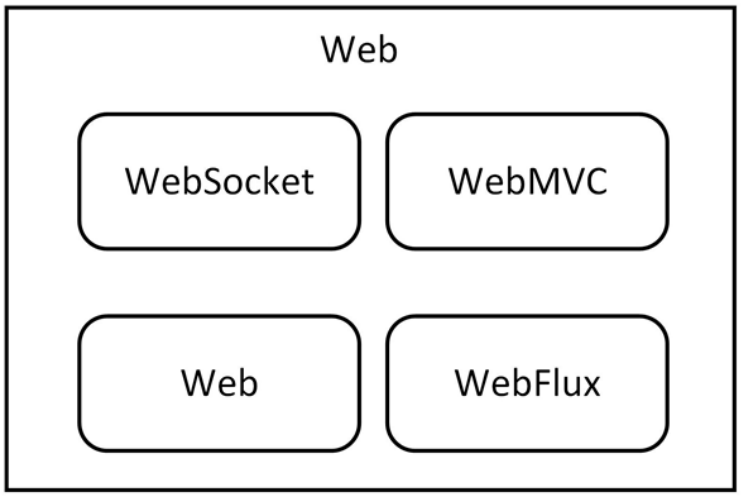


- Spring-JMS模块提供对JMS的支持，能够发送和接受信息。

- 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

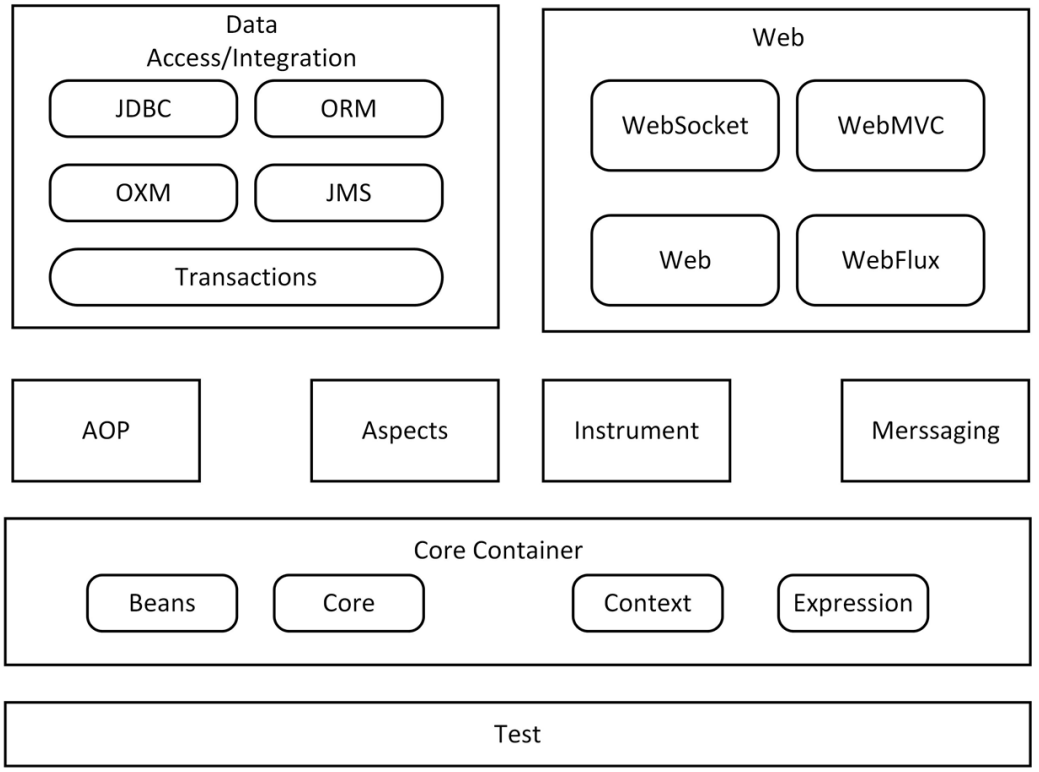
- Web模块由四个模块组成。
  - Spring-Web模块为Spring提供了最基础Web支持，它主要建立于核心容器之上，通过 Servlet或者Listeners来初始化IoC容器以及Web应用上下文。



  - 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. Messaging和7. Test

- 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。
- Test模块也只由一个模块组成，主要为测试提供支持。
- Spring体系结构回顾



大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

POJO对象

- Spring和EJB框架结构都有一个共同核心设计理念：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。
  - POJO类没有任何特别之处，无需装饰，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。

POJO对象例子

```java
//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;
    }
}
```

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，一般通过显示代码来进行对象关联。

- 在基于Spring的应用中，这些组件及对象生存在Spring容器中，由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。
  - DI能提供类似胶水的功能，自动的把某些对象“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”，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。

DI案例:接口

- 以下，我们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();
}
```

DI案例：老师类

- 教师给学生上课。因此，定义教师类Teacher和学生类Student。其中教师将实现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，唯一不同的时多了@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!");
    }
}
```

DI案例：学生类

```
// 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();
    }
}
```

DI案例：配置文件

- 在定义了以上的简单的类之后，Spring可通过另外的配置文件，来定义各个类之间的关系。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， 以下是java代码的配置方式：

```
//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());
    }
}
```

DI案例:配置文件

- ✓ 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。因此，通过以上的配置文件，Spring容器将生成两个实例对象Bean，id分别是service和student。

DI案例:测试代码

- 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。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。

```
// 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();
    }
}
```

DI案例:XML配置

- 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，在student中，还为student对象的构造函数指定了引用参数service。

DI案例:XML配置和测试代码

- ✓ 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，动态的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。

➢ 测试代码如下

```
//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();
    }
}
```

DI案例:自动编织

- 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component-scan。

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

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

DI案例:XML自动编织

- Xml配置文件，也可以注明自动编织。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。

```
<?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>
```

- 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。感兴趣的同学可以深入研究。

大纲

- Web服务器
  - Web服务器概述
  - Web服务器的工作原理
  - Web服务器和MVC框架
- Web容器简介
  - 容器的概念
  - 解耦合、控制反转及依赖注入
  - 面向切面的编程AOP
- Java EE框架
  - Java EE概述
  - Java EE框架组成
  - Java EE的主要技术
  - 企业级Java中间件
- Spring框架
  - Spring框架的历史
  - Spring的体系结构
  - Spring容器和依赖注入
  - Spring容器和AOP编程

AOP案例

- 以下用一个例子来说明Spring容器中面向切面的编程。

- 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();
}
```

AOP案例:传统方法

```
//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方法包含了很多的与上课无关的代码，且须显式地调用Student对象来进行操作，代码耦合度高。

AOP案例:定义切面

- 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，关手机等可以看做是横向逻辑业务，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() {}
```

AOP案例:定义切面

```
    @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... ");
    }
}
```

AOP案例:切入点表达式

- 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，其语法如下所示：
- ✓ 表达式为 execution (* com.sample.service.impl..*.*(..))，则
  1. execution()：表达式主体。
  2. 第一个*号：表示返回类型，*号表示所有的类型。
  3. 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，即com.sample.service.impl包、子孙包下所有类的方法。
  4. 第二个*号：表示类名，*号表示所有的类。
  5. *（..）：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。

AOP案例:织入

- 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。

```
//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();
    }
```

AOP案例:织入

```
//ClassRoomConfig.java
    @Bean
    public Service teacher(){
        Teacher p=new Teacher();
        p.setName("Lai");
        return p;
    }
}
```

- ✓ 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。

AOP案例: AOP老师类

- Teacher类则简化为如下代码。教师无需关注学生的状况，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!");
    }
}
```

AOP案例:测试代码

```
//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?
```

本章小结

- 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JavaEE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。

参考文献

- https://blog.csdn.net/m0_74436895/article/details/140840574

作业3

- 根据课程内容，对Spring中的AOP编程部分内容进行仿真实现。要求同上次一样。