第三章 三层架构和MVC

1. 什么是系统架构

有2种，三层架构与MVC。有区别既有联系，均是降低系统模块的耦合度。

2.什么是三层架构

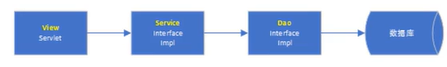
指：试图层View。服务层Servise，与持久层DAO,他们有不同的功能。

试图层View ：用于接受用户提交的请求的代码

服务层Servise ：系统的业务逻辑主要在这里完成

持久层DAO ：直接操作数据库的代码

注：为了更好的降低各层的耦合度，采用面向对象。即上层对下层的调用。是通过接口实现的。而下层对上层的真正服务提供者，是下层接口的实现类。服务标准（接口）是相同的，服务者（实现类）可以更换，实现层间解耦合。



3.总结：

系统的复杂度越来越高 复杂度==耦合度

Petsshop

视图层

业务逻辑层 Service Manager 约定由于配置 abc service

数据访问层 DAO 数据访问对象

开发中，业务才是核心

技术， 技术是辅助业务的

业务问题

业务是有复杂度，事物

三级 ：

普通业务 只开启一个事物

普通业务 开启三个事物

复杂业务 开启7个事物

多张表之间有个依赖关系 强关联系

Main

所有业务都写进main

一个类只做一种事

一个方法只做一件事

写且只写一次

三层架构和MVC

* MVC 即Model模型 view模式 及Controller控制器
* view模式：试图，为用户提供使用界面，与用户直接进行交互。
* model模型：承接数据，并对用户提交请求进行计算的模块，其分为二类，一类称为数据承载Bean， 一类称为业务处理Bean，所谓数据承接Bean是指实体类，专门用户承载业务数据的，如Student,User，等。而业务处理Bean则是指Service或Dao对象，专门用于处理用户提交请求的。
* Controller控制器：用于将用户请求转发给相应的Model进行处理，并根据Model的计算机结果向用户提供相应。

MVC架构程序的工作流程

* 用户通过View页面向服务端提出请求,可以是表单请求、超链接请求、AJAX 请求等
* 服务端Controller控制器接收到请求后对请求进行解析,找到相应的Model对用户请求进行处理
* Model处理后,将处理结果再交给Controller
* Controller在接到处理结果后,根据处理结果找到要作为向客户端发回的响应View页面。页面经渲染(数据填充)后,再发送给客户端。

总结：Model 模型是一个角色

数据模型 Entity

业务模型 Service

控制器

处理用户请求的

Servlet 服务器的小程序

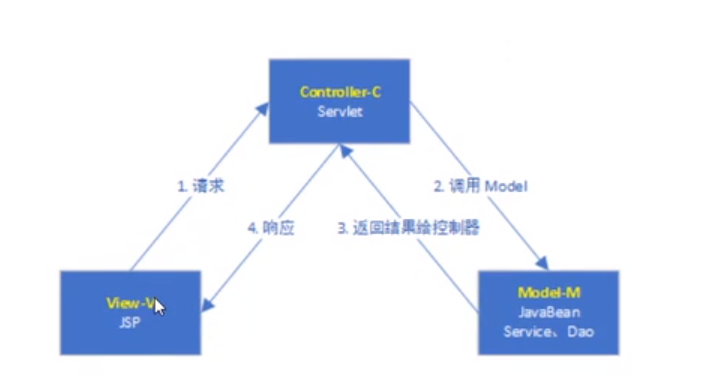
处理业务逻辑

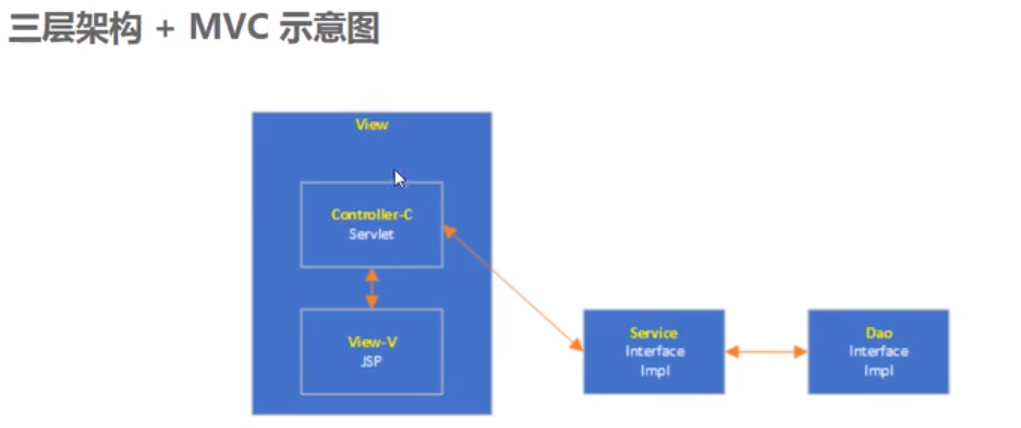
处理页面展示

JSP 处理页面展示 JSP就是一个Servlet JSP是一门技术

小脚本

<% %>





**说说你对设计原则的理解**

口诀

为了便于记忆,I我们可以使用一个口诀来记忆面向对象设计原则:开口合里最单依

开:开闭原则

口:接口隔离原则

合:组合/聚合原则

里:里式替换原则

最:最少知识原则(迪米特法则)

单:单- -职责原则

依:依赖倒置原则

**开闭原则(Open-Closed Principle, OCP)**

一个软件实体应当对扩展开发，对修改关闭说的是再设计一个模块的时候应当使这个模块可以在不被修改的前提下被扩展换言之,应当可以在不必修改源代码的情况下改变这个模块的行为,在保持系统一定稳定性的基础上,对系统进行扩展。这是面向对象设计(OOD)的基石,也是最重要的原则。

**接口隔离原则(Interface Segregation Principle, ISP)**

一个类对另外一 个类的依赖是建立在最小的接口上。

使用多个专门的接口比使用单一的总接口要好根据客户需要的不同，而为不同的客户端提供不同的服务是一种应当得到鼓励的做法，就像”看人下菜碟”-样,要看客人是谁，再提供不同档次的饭菜.

胖接口会导致他们的客户程序之间产生不正常的并且有害的耦合关系.当一个客户程序要求该胖接口进行一 个改动时，会影响到所有其他的客户程序.因此客户程序应该仅仅依赖他们实际需要调用的方法.

**组合/聚合复用原则(Composite/Aggregate Reuse Principle , CARP)**

在一个新的对象里面使用一些已有的对象使之 成为新对象的一部分;新的对象通过这些向对象的委派达到复用已有功能的目的.这个设计原则有另一个简短的表述:要尽量使用合成/聚合，尽量不要使用继承

**里氏代换原则(Liskov Substitution Principle , LSP)**

由Barbar Liskov (芭芭拉.里氏)提出,是继承复用的基石。

所有引用基类的地方必须透明的使用其子类的对象。只要父类能出现的地方子类也可以出现,而且替换为子类不会产生任何错误或异常,但是反过来就不行,有子类出现的地方,父类末必就能适应。

**最少知识原则(Least Knowledge Principle , LKP)**

一个对象应当对其他对象有尽可能少的了解.

没有任何一个其他的00设计原则象迪米特法则这样有如此之多的表述方式如下几种:

1. 只与你直接的朋友们通信(Only talk to your immediate friends)
2. 不要跟”陌生人”说(Don’t talk to strangers)

3.每一个软件单位对其他的单位都只有最少的知识，而且局限于那些本单位密切相关的软件单位

就是说如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用，如果其中的一个类需要调用另一个类的某一个方法的话，可以通过第三者转发这个调用。

**单一职责原则(Simple responsibility pinciple，SRP)**

就一个类而言，应该仅有一个引起它变化的原因，如果你能想到多 于一个的动机去改变一 个类， 那么这个类就具有多于一个的职责.应该把多于的指责分离出去，分别再创建一些类来完成每一 个职责.

**依赖倒置原则(Dependence Inversion Principle)**要求客户端依赖于抽象耦合.  
1.模块间的依赖通过抽象发生,实现类之间不发生直接的依赖关系,其依赖关系是通过接口或抽象类产生的。

2.接口或抽象类不依赖实现类实现类依赖接口或抽象类  
采用依赖倒置原则可以减少类间的耦合性,提高系统的稳定,降低并行开发引起的风险,提高代码的可读性和可维护性。

总结：

一件事

四则运算 加减乘除 计算（int a, int b , String ops）

面向对象设置原则

重构原则

事不过三，三责重构

提取方法

提取类

A 类

工具类

四则运算

B 类

四则运算

C 类

工具类

Add

**第三章：Bootstrap**

简单的框架或技术

复杂的框架或技术

技术选项

1. 搜索量，指数
2. 社区活跃度，需要文档完备性
3. 是否开源

渐进增强，向上兼容

优雅降级，向下兼容

布局

Table 表格布局 2秒 问题：必须表格里的内容加载完成才会显示

Div+css 代码 从上往下执行 调试布局难度大

网格布局

客户体验

**Bootstrap简介**

Bootstrap ,来自Twitter ,是目前很受欢迎的前端框架。Bootstrap 是基于HTML、CSS、 JavaScript 的,它简洁灵活,使得Web开发更加快捷。它是一个CSS/HTML框架。Bootstrap 提供了优雅的HTML和CSS规范。Bootstrap -经推出后颇受欢迎，一直是GitHub上的热[ ]开源项目。

**为什么使用Bootstrap ?**

* 移动设备优先:自Bootstrap3起,框架包含了贯穿于整个库的移动设备优先的样式。
* 浏览器支持:所有的主流浏览器都支持Bootstrap.
* 容易上手:只要您具备HTML和CSS的基础知识,您就可以开始学习Bootstrap.
* 响应式设计: Bootstrap的响应式CSS能够自适应于台式机、平板电脑和手机。
* 它为开发人员创建接口提供了一个简洁统一的解决方案。
* 它包含了功能强大的内置组件,易于定制。
* 它还提供了基于Web的定制。
* 它是开源的。

**Bootstrap网格系统**

Bootstrap提供了一套响应式、 移动设备优先的流式网格系统,随着屏幕或视口( viewport )尺寸的增加,系统会自动分为最多12列。  
**什么是网格( Grid ) ?** 在平面设计中,网格是一种由一 系列用于组织内容的相交的直线(垂直的、水平的)组成的结构(通常是二维的)。 它广泛应用于打印设计中的设计布局和内容结构。在网页设计中，它是一种用于快速创建一致的布局和有效地使用HTML和CSS的方法。  
简单地说,网页设计中的网格用于组织内容,让网站易于浏览,并降低用户端的负载。  
**什么是Bootstrap网格系统( Grid System) ?** Bootstrap官方文档中有关网格系统的描述:  
 Bootstrap包含了一-个响应式的、移动设备优先的、不固定的网格系统,可以随着设备或视口大小的增加而适当地扩展到12列。它包含了用于简单的布局选项的预定义类,也包含了用于生成更多语义布局的功能强大的混合类。  
让我们来理解一下， 上面的语句。Bootstrap 3是移动设备优先的,在这个意义上, Bootstrap代码从小屏幕设备(比如移动设备、平板电脑)开始,然后扩展到大屏幕设备(比如笔记本电脑、台式电脑)上的组件和网格。







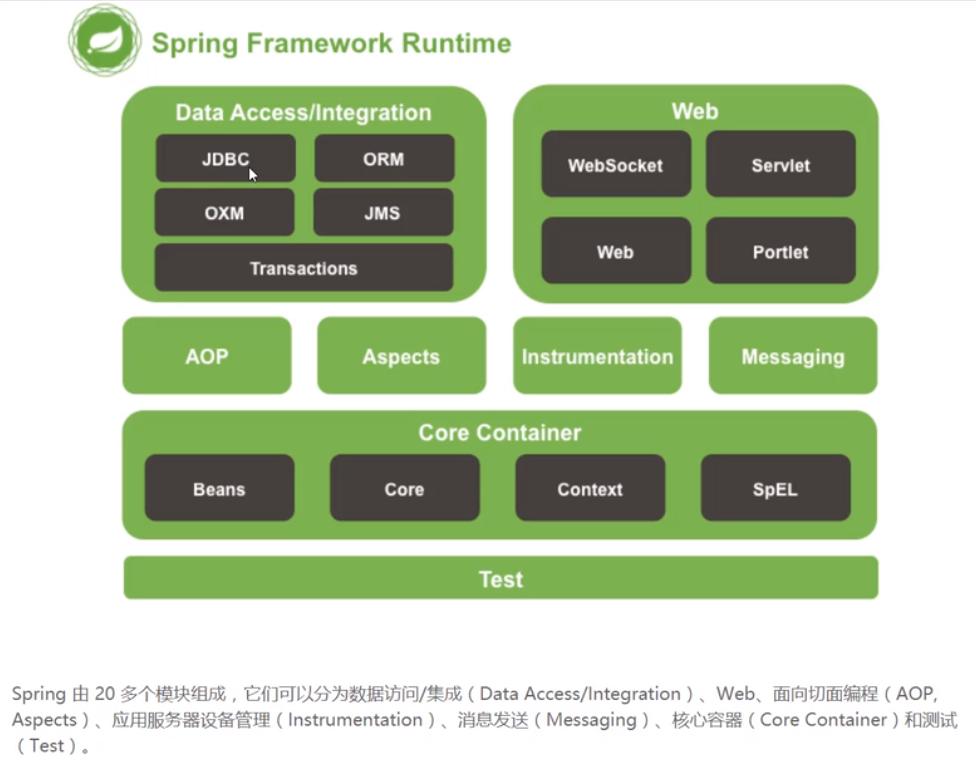
登录：

业务逻辑完备

独立的系统

拆分





Spring 是为了解决企业级开发的复杂度问题 就是耦合度

系统开发过程中

1. 主业务逻辑

银港业务

电商业务

物流业务

1. 系统级业务逻辑，交叉业务逻辑

JDBC连接数据库

1. 加载驱动
2. 创建连接
3. 开启事物
4. SQL操作 具体业务
5. 提交事物
6. 释放连接

UserService userService = new UserServiceImpl();

交给Spring 容器创建

Spring 胶水框架

ORM

1. MyBatis
2. Hibernate