SpringMVC原理介绍

# MVC及SpringMVC知识概要

## 什么是MVC框架？MVC框架的设计思想？

## SpringMVC概述

## SpringMVC中的DispatcherServlet

## SpringMVC的原理及执行过程

## SpringMVC中的逻辑视图、物理视图和视图解析器ViewResolver

# 什么是MVC框架？

## Model-View-Controller：MVC

**MVC是一种设计模式。模型-视图-控制器（MVC）**是一个众所周知的以设计界面应用程序为基础的设计模式。它主要通过**分离模型、视图及控制器**在应用程序中的角色将**业务逻辑**从界面中解耦。

## MVC框架的三部分功能介绍

通常：

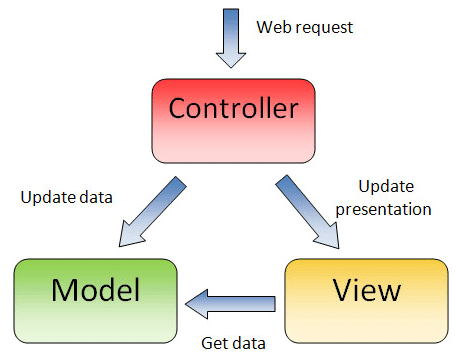
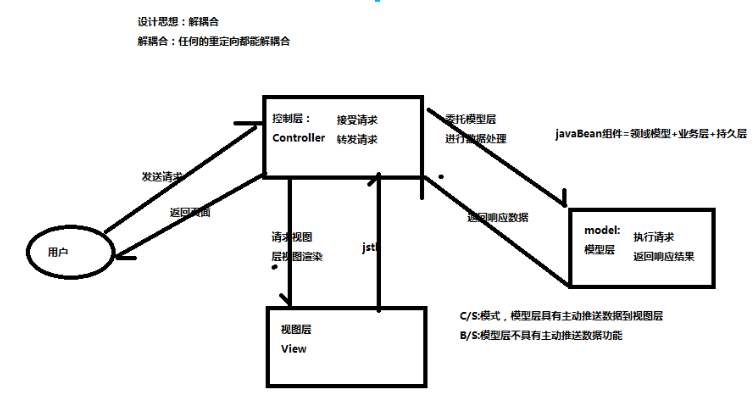
### 模型负责封装应用程序数据在**视图层**展示。

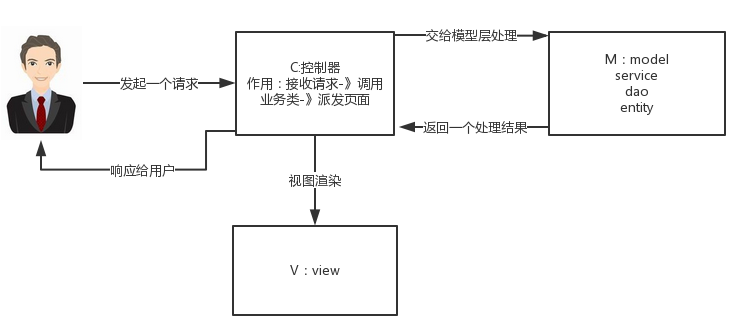
### 视图仅仅只是展示这些数据，不包含任何业务逻辑。

### 控制器负责接收来自用户的请求，并调用后台服务（manager或者dao）来处理业务逻辑。处理后，后台业务层可能会返回了一些数据在视图层展示。控制器收集这些数据及准备模型在视图层展示。

## MVC框架的设计思想：解耦合

**MVC模式的设计思想**是将**业务逻辑**从界面中分离出来，允许它们**单独改变而不会相互影响**。



M-Model 模型（完成业务逻辑：有javaBean构成，service+dao+entity）

V-View 视图（做界面的展示 jsp，html……）

C-Controller 控制器（接收请求—>调用模型—>根据结果派发页面）

在**Spring MVC**应用程序中，模型通常由**POJO对象**组成，**它在业务层中被处理，在持久层中被持久化**。视图通常是用JSP标准标签库（JSTL）编写的JSP模板。**控制器部分是由Dispatcher servlet负责**，在本教程中我们将会了解更多它的相关细节。

**MVC的设计思想：解耦合**。任何的重定向都能解耦合。

### 控制层Controller： **接受请求，转发请求**。

### 模型层Model：执行请求，返回响应结果。

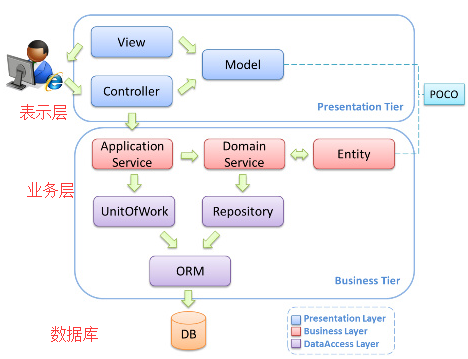
### 视图层View：

C/S模式：模型层会主动推送数据到视图层，已经过时；

B/S模型：模型层不会主动推送数据到视图层。

## Web应用3层架构：数据-业务-表示

一些开发人员认为**业务层和DAO层类**是MVC模型组件的一部分。我对此持有不同的意见。我不认为业务层及DAO层类为MVC框架的一部分。通常一个web应用是3层架构，即**数据-业务-表示**。**MVC实际上是表示层的一部分。**



# Spring MVC概述

## SpringMVC简单介绍

**Spring MVC框架**是一个MVC框架，通过实现**Model-View-Controller模式**来很好地将数据、业务与展现进行分离。从这样一个角度来说，Spring MVC和Structs、Structs2非常类似。Spring MVC的设计是围绕**DispatcherServlet**展开的，**DispatcherServlet负责将请求派发到特定的handler**。通过可配置的hander mappings、view resolution、locale以及theme resolution来处理请求并且转到对应的视图。

有一个疑惑就是，**springMVC和spring是什么样的关系呢**？这个在百度百科上有一个很好的解释：意思是说，springMVC是spring的一个后续产品，其实就是**spring在原有基础上，又提供了web应用的MVC模块**，可以简单的把springMVC理解为是spring的一个模块（类似AOP，IOC这样的模块），网络上经常会说springMVC和spring无缝集成，其实springMVC就是spring的一个子模块，所以根本不需要同spring进行整合。

**Spring MVC有基于注解版与基础.xml版的两种用法**，不过现在的企业级开发基本都使用的是注解版，没别的原因，就是方便而已。因此后面的代码示例，都是基于注解版本的，想了解基于.xml版本的Spring MVC的朋友可以自行上网查询。

spring MVC属于SpringFrameWork的后续产品，已经融合在**Spring Web Flow**里面。

Spring 框架提供了构建 Web 应用程序的**全功能 MVC 模块**。使用 Spring 可插入的 **MVC 架构**，可以选择是使用内置的 Spring Web 框架还是 Struts 这样的 Web 框架。通过策略接口，Spring 框架是高度可配置的，而且包含多种视图技术，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText 和 POI。Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。

Spring MVC 分离了**控制器、模型对象、分派器以及处理程序对象**的角色，这种分离让它们更容易进行定制。

## SpringMVC的组成

Spring的MVC框架主要由**DispatcherServlet、处理器映射(HandlerMapping)、处理器(控制器)、视图解析器(ViewResolver)、视图(View)**组成。

## SpringMVC中的接口介绍

### **DispatcherServlet**接口：

Spring提供的**前端控制器**，所有的请求都有经过它来**统一分发**。在DispatcherServlet将请求分发给Spring Controller之前，需要借助于Spring提供的**HandlerMapping**定位到具体的Controller。

### HandlerMapping接口：

能够完成客户请求到Controller映射。

### Controller接口：

需要为并发用户处理上述请求，因此实现Controller接口时，必须保证线程安全并且可重用。

Controller将处理用户请求，这和Struts Action扮演的角色是一致的。一旦Controller处理完用户请求，则返回**ModelAndView对象**给DispatcherServlet前端控制器，ModelAndView中包含了**模型（Model）**和**视图（View）**。

从宏观角度考虑，**DispatcherServlet是整个Web应用的控制器**；从微观考虑，**Controller是单个Http请求处理过程中的控制器**，而ModelAndView是Http请求过程中返回的模型（Model）和视图（View）。

### ViewResolver接口：

Spring提供的**视图解析器**（**ViewResolver**）在Web应用中查找View对象，从而将相应结果渲染给客户。

# SpringMVC的运行原理

## SpringMVC的执行过程1

1. 客户端请求提交到**DispatcherServlet**;

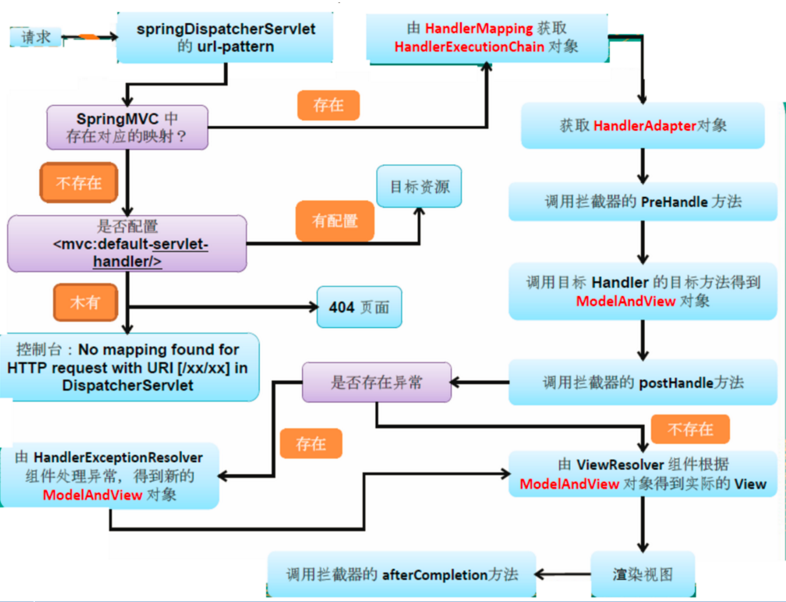
2. 由DispatcherServlet控制器查询一个或多个**HandlerMapping**，找到处理请求的Controller;

3. DispatcherServlet将请求提交到Controller ;

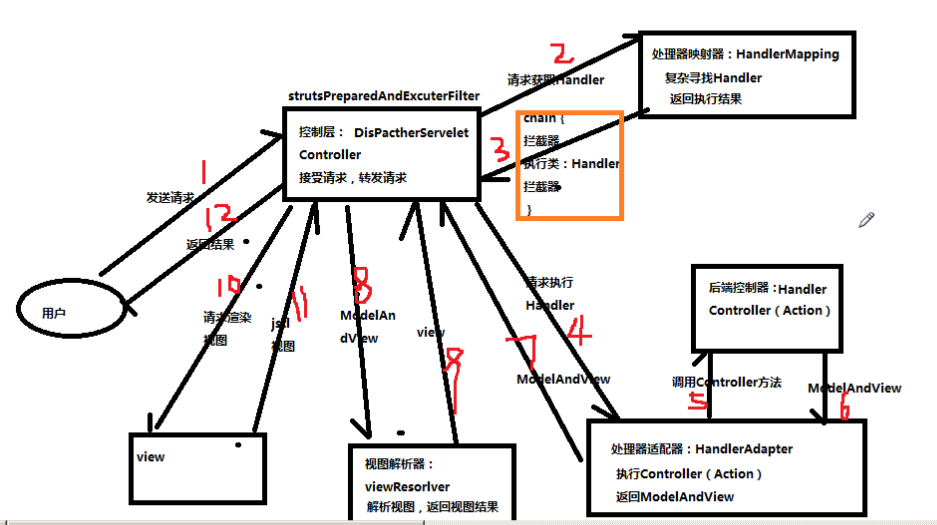
4. Controller调用**业务逻辑处理**后，返回ModelAndView对象；

5. DispatcherServlet查询一个或多个**ViewResoler视图解析器**，找到ModelAndView指定的物理视图；

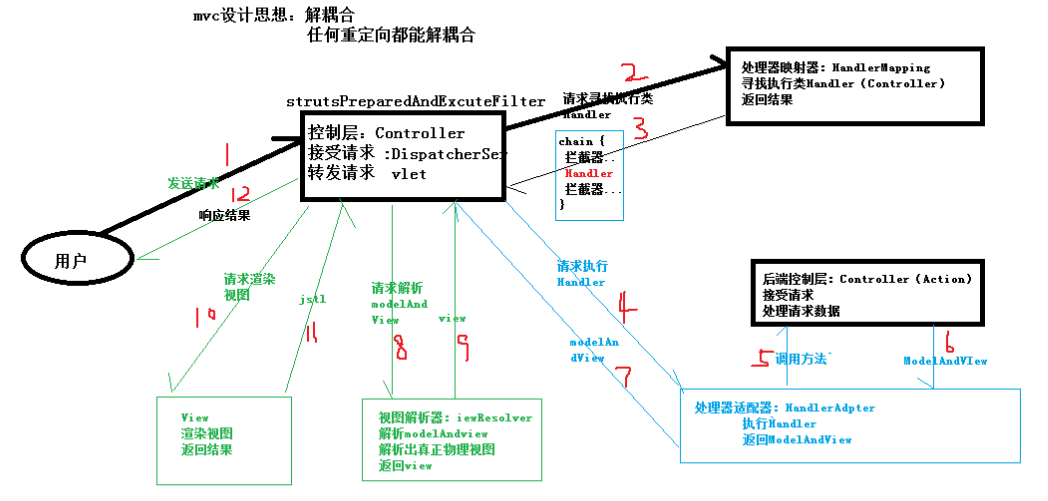
6. 视图负责将结果显示到客户端；



## SpringMVC的执行过程2



## SpringMVC执行过程3

、

## SpringMVC工作原理图4：最完整。

执行流程：

1、 用户发送请求至**前端控制器DispatcherServlet**。

2、 DispatcherServlet收到请求后，请求**HandlerMapping处理器映射器去查找处理器(Handler)：**通过xml配置或者注解进行查找。

3、 处理器映射器找到具体的处理器之后，生成HandlerExecutionChain执行链(包括**处理器对象及处理器拦截器(如果有则生成))**返回给DispatcherServlet。

4、 DispatcherServlet调用**HandlerAdapter处理器适配器去执行处理器(Handler)**。

5、 HandlerAdapter经过适配调用具体的处理器Handler(也叫后端控制器)。

6、 Handler处理器执行完向HandlerAdapter处理器适配器返回**ModelAndView**。

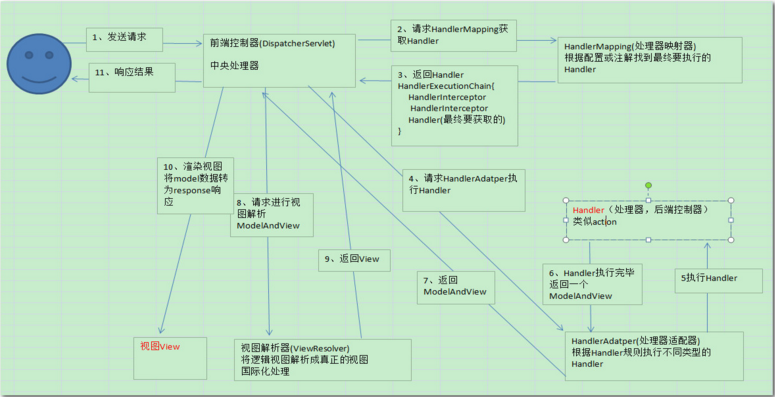
7、 HandlerAdapter将处理器Handler执行结果ModelAndView返回给**DispatcherServlet**。

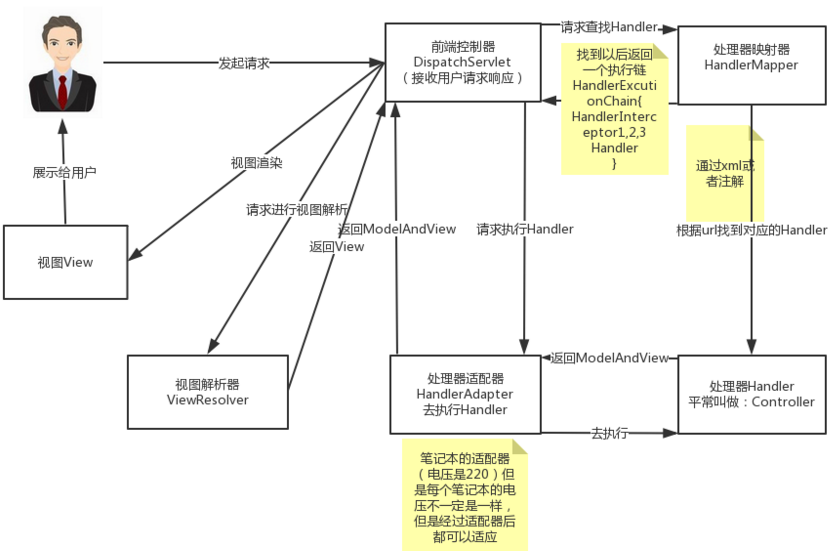
8、 DispatcherServlet将ModelAndView传给**ViewReslover视图解析器**。

9、 ViewReslover解析后返回具体View。

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11、 DispatcherServlet响应用户。





# SpringMVC的组件

以下组件通常使用框架提供实现：

**DispatcherServlet**：作为**前端控制器，整个流程控制的中心**，控制其它组件执行，统一调度，降低组件之间的耦合性，提高每个组件的扩展性。

**HandlerMapping**：通过扩展**处理器映射器**实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

**HandlAdapter**：通过扩展**处理器适配器**，支持更多类型的处理器。

**ViewResolver**：通过扩展**视图解析器**，支持更多类型的视图解析，例如：jsp、freemarker、pdf、excel等。

简单理解：

**前端控制器（DispatcherServlet）**：接收请求，响应结果，相当于电脑的CPU。

**处理器映射器（HandlerMapping）**：根据URL去查找处理器；

**处理器（Handler）**：（需要程序员去写代码处理逻辑的）

**处理器适配器（HandlerAdapter）**：会把处理器包装成适配器，这样就可以支持多种类型的处理器，类比笔记本的适配器（适配器模式的应用）

**视图解析器（ViewResovler）**：进行视图解析，多返回的字符串，进行处理，可以解析成对应的页面

组件：

## 前端控制器DispatcherServlet（不需要工程师开发）,由框架提供

作用：接收请求，响应结果，相当于转发器，中央处理器。有了dispatcherServlet减少了其它组件之间的耦合度。

用户请求到达前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

## 处理器映射器HandlerMapping(不需要工程师开发),由框架提供

作用：根据请求的url查找Handler

HandlerMapping负责根据用户请求找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

## 处理器适配器HandlerAdapter

作用：按照特定规则（HandlerAdapter要求的规则）去执行Handler

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

## 处理器Handler(需要工程师开发)(Controller中的方法)

注意：编写Handler时按照HandlerAdapter的要求去做，这样适配器才可以去正确执行Handler

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。

由于Handler涉及到具体的用户业务请求，所以一般情况需要工程师根据业务需求开发Handler。

## 视图解析器View resolver(不需要工程师开发),由框架提供

作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。 springmvc框架提供了很多的View视图类型，包括：jstlView、freemarkerView、pdfView等。

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由工程师根据业务需求开发具体的页面。

## 视图View(需要工程师开发jsp...)

View是一个接口，实现类支持不同的View类型（jsp、freemarker、pdf...）

## 核心架构的具体流程步骤如下：

1、首先用户发送请求——>**DispatcherServlet**，前端控制器收到请求后自己不进行处理，而是委托给其他的解析器进行处理，作为统一访问点，进行全局的流程控制；

2、DispatcherServlet——>HandlerMapping， HandlerMapping 将会把请求映射为HandlerExecutionChain 对象（包含一个Handler 处理器（页面控制器）对象、多个HandlerInterceptor 拦截器）对象，通过这种策略模式，很容易添加新的映射策略；

3、DispatcherServlet——>HandlerAdapter，HandlerAdapter 将会把**处理器**包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；

4、HandlerAdapter——>处理器功能处理方法的调用，HandlerAdapter 将会根据适配的结果调用真正的处理器的功能处理方法，完成功能处理；并返回一个ModelAndView 对象（包含模型数据、逻辑视图名）；

5、ModelAndView的逻辑视图名——> **ViewResolver**， ViewResolver 将把逻辑视图名解析为具体的View，通过这种策略模式，很容易更换其他视图技术；

6、View——>渲染，View会根据传进来的**Model模型**数据进行渲染，此处的Model实际是一个Map数据结构，因此很容易支持其他视图技术；

7、返回控制权给DispatcherServlet，由DispatcherServlet返回响应给用户，到此一个流程结束。

## 下边两个组件通常情况下需要开发：

Handler：处理器，即后端控制器用controller表示。

View：视图，即展示给用户的界面，视图中通常需要标签语言展示模型数据。

# DispatcherServlet(SpringMVC前端控制器)

**DispatcherServlet**是整个Spring MVC的核心，它负责接收HTTP请求组织协调Spring MVC的各个组成部分。其主要工作有以下三项：

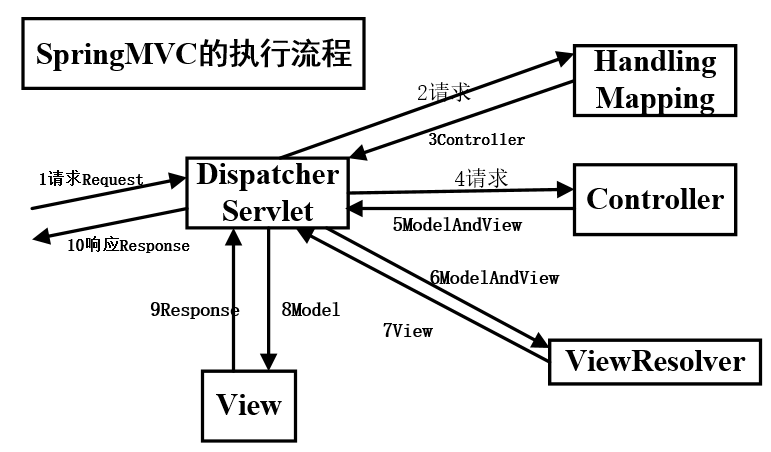
### 截获符合特定格式的URL请求。

### 初始化DispatcherServlet上下文对应的**WebApplicationContext**，并将其与业务层、持久化层的WebApplicationContext建立关联。

### 初始化Spring MVC的各个组成组件，并装配到**DispatcherServlet**中。

在简单的Spring MVC应用程序中，控制器是唯一的你需要在**Java web部署**描述文件（即**web.xml**文件）中配置的Servlet。**Spring MVC控制器 ——通常称作DispatcherServlet**，**实现了前端控制器设计模式**，**并且每个web请求必须通过它以便它能够管理整个请求的生命周期**。

当一个web请求发送到Spring MVC应用程序，**DispatcherServlet首先接收请求**，然后它组织那些在Spring web**应用程序上下文配置**的（例如实际请求处理控制器和视图解析器）或者**使用注解配置的组件**，所有的这些都需要处理该请求。



**在Spring3.0中定义一个控制器类，这个类必须标有@Controller注解。**当有@Controller注解的控制器收到一个请求时，它会寻找一个**合适的handler方法**去处理这个请求。这就需要控制器通过**一个或多个handler映射**去把每个请求映射到**handler方法**。为了这样做，**一个控制器类的方法需要被@RequestMapping注解装饰，使它们成为handler方法。**

**handler方法处理完请求后，它把控制权委托给视图名与handler方法返回值相同的视图。**为了提供一个灵活的方法，**一个handler方法的返回值并不代表一个视图的实现而是一个逻辑视图**，即没有任何文件扩展名。你可以将这些逻辑视图映射到正确的实现，并将这些实现写入到上下文文件，这样你就可以**轻松的更改视图层代码甚至不用修改请求handler类的代码**。为**一个逻辑名称**匹配正确的文件是**视图解析器**的责任。一旦控制器类已将一个视图名称解析到一个视图实现，它会根据视图实现的设计来渲染对应对象。

# 视图解析器ViewResolver

resolver英 [rɪ'zɒlvə] 美 [rɪ'zɒlvə] n.解析器 如ViewResolver视图解析器

## 在springmvc-config.xml中配置视图解析器：

*<!-- 视图解析器 -->*<**bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  
 <**property name="prefix" value="/WEB-INF/static/"**></**property**>  
 <**property name="suffix" value=".jsp"**></**property**>  
</**bean**>

## 物理视图

视图分为逻辑视图和物理视图，它们之间的转化通过**视图解析器**来完成。

**视图解析器:**：SpringMVC提供的视图解析器为

**org.springframework.web.servlet.view.InternalResourceViewResolver。**

逻辑视图：在Handler方法返回的是逻辑视图，如welcome

物理视图：视图解析器解析出真正的物理视图：**prefix+逻辑视图+suffix**；如**/WEB-INF/static/welcome.jsp**

## 使用示例：

物理视图不需要配置视图解析器，只有逻辑视图才需要配置逻辑解析器，一般为了简单，采用的是逻辑视图解析器，需要配置逻辑视图解析器。

@RequestMapping(value=**"/hello"**)  
**public** ModelAndView hello(){  
 *// 创建准备返回的ModelAndView对象，该对象通常包含了****返回视图的路径、模型的名称以及模型对象***ModelAndView mv = **new** ModelAndView();  
 */\*添加模型数据 可以是任意的POJO对象\*/* mv.addObject(**"message"**, **"Hello World!"**);  
 *// 设置逻辑视图名，视图解析器会根据该名字解析到具体的视图页面* **mv.setViewName("welcome");**  
 *// 返回ModelAndView对象。* **return** mv;  
}



通过ModelAndView的setViewName方法添加路径：由于已经配置了视图解析器，这里只需要设置逻辑视图welcome,就可以转换为物理视图：**/WEB-INF/static/welcome.jsp。**

如果没有配置视图解析器，则需要mv.setViewName("/WEB-INF/static/welcome.jsp ");

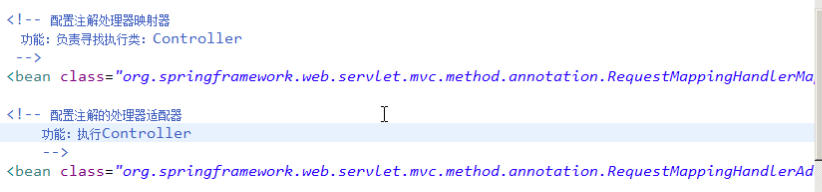
# 处理器映射器和处理器适配器。

## 处理器映射器：…annotation.RequestMappingHandlerMapping

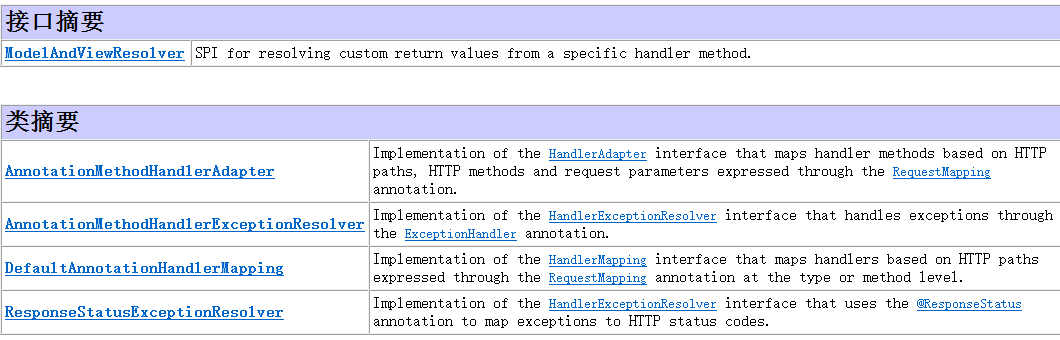
功能：映射到对应的处理器Handler

## 处理器适配器：…annotation.RequestMappingHandlerAdapter

功能：对对应的处理器Handler适配



## 包org.springframework.web.servlet.mvc.annotation



# SpringMVC与struts2的区别

1、springmvc基于**方法**开发的，struts2基于**类**开发的。springmvc将url和controller里的方法映射。映射成功后springmvc生成一个Handler对象，对象中只包括了一个method。方法执行结束，形参数据销毁。springmvc的controller开发类似web service开发。

2、springmvc可以进行单例开发，并且建议使用单例开发，struts2通过类的成员变量接收参数，无法使用单例，只能使用多例。

3、经过实际测试，struts2速度慢，在于使用struts标签，如果使用struts建议使用jstl。

struts的执行流程：

