**Spring MVC起步**

**2018年5月30日**

[第一章 MVC简介 3](#_Toc516045350)

[第一节 Spring MVC起步课程简介 3](#_Toc516045351)

[第二节 前端控制器 3](#_Toc516045352)

[第三节 MVC概念 4](#_Toc516045353)

[第二章 Spring MVC中的基本概念 5](#_Toc516045354)

[第一节 Spring MVC的静态概念 5](#_Toc516045355)

[第二节 Spring MVC的动态概念 7](#_Toc516045356)

[第三章 配置Maven环境 8](#_Toc516045357)

[第一节 Maven介绍 8](#_Toc516045358)

[第二节 Maven的安装 11](#_Toc516045359)

[第三节 Maven的配置 12](#_Toc516045360)

[第四节 用Maven创建项目 13](#_Toc516045361)

[第五节 Hello Spring MVC 15](#_Toc516045362)

[第四章 Spring MVC实操 20](#_Toc516045363)

[第一节 从配置文件开始 20](#_Toc516045364)

[第二节 Controller—基础代码 24](#_Toc516045365)

[第三节 Controller—现代方式 25](#_Toc516045366)

[第四节 Controller—传统方式 27](#_Toc516045367)

[第五节 Binding 28](#_Toc516045368)

[第六节 FileUpload—单文件上传 31](#_Toc516045369)

[第七节 JSON（上） 33](#_Toc516045370)

[第八节 JSON（中） 35](#_Toc516045371)

[第九节 JSON（下） 37](#_Toc516045372)

[第五章 总结 40](#_Toc516045373)

[第一节 总结 40](#_Toc516045374)

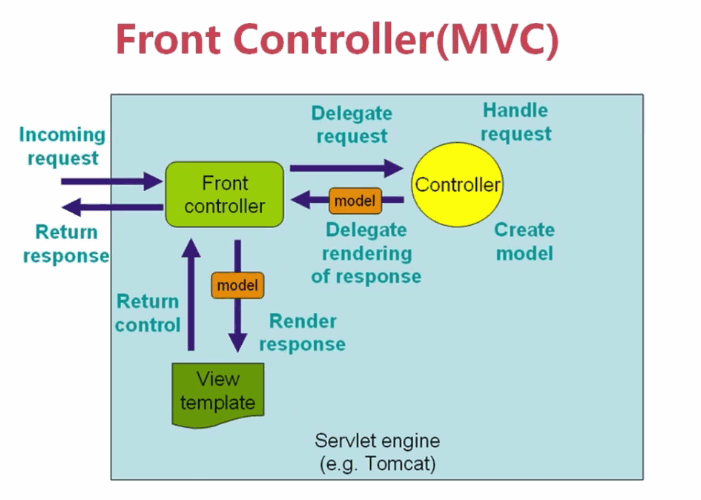
# 第一章 MVC简介

MVC的演进，概念及优点

## 第一节 Spring MVC起步课程简介

## 第二节 前端控制器

1. MVC不是java独有的概念，在java诞生之前，Smalltalk，Python都有MVC的概念
2. MVC是一种开发web应用程序的通用架构方式
3. Front Controller（MVC）：由Spring团队书写的一种设计模式，一种MVC的实现方式，如下图：



**分发调度**

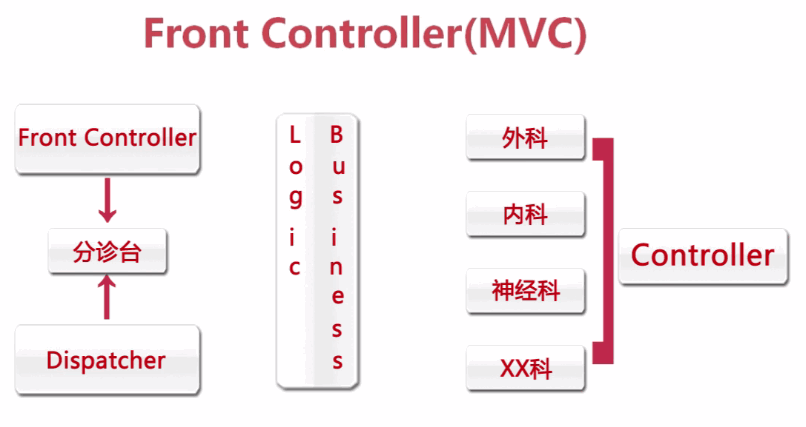
**页面呈现**

**业务数据抽取**

* 1. 用户端的请求通过http协议到达前端控制器
  2. 前端控制器了解这个请求应当被谁处理（控制器）
  3. 控制器了解业务逻辑的细节，调用业务逻辑生成了业务数据，并将业务数据返回给了前端控制器
  4. 此时，前端控制器将业务数据分发给业务视图，由业务视图呈现最终的用户页面
  5. 将呈现好的页面返回给前端控制器，并最终将这个页面返回给浏览器端

1. 前端控制器也称之为调度器

图例：



因为分诊台是在具体科室之前接待病人，所以这种模式称之为前端控制器

1. MVC本质
   1. MVC的核心思想是业务数据抽取同业务数据呈现相分离

## 第三节 MVC概念

1. 什么是MVC？
   1. Model-View-Controller的缩写
   2. View：视图层

为用户提供UI，重点关注数据的呈现

* 1. Model：模型层

业务数据的信息表示，关注支撑业务的信息构成，通常是多个业务实体的组合

* 1. Controller：控制层

通过调用业务逻辑产生合适的数据（Model），同时将数据传递给视图层用于呈现

* 1. 什么是MVC？
     1. MVC是一种架构模式，它使得我们的程序”程序分层”,”分工合作”,”既相互独立”,”又协同工作”， 所以才会有前后端工程师之分
     2. MVC是一种思考方式：
        1. 需要将什么信息展示给用户？如何布局？调用哪些业务逻辑？

**M V C**

# 第二章 Spring MVC中的基本概念

## 第一节 Spring MVC的静态概念

1. DispatcherServlet:

注：Dispatcher：调度员/分配器

Spring MVC作为前端控制器的一种实现形式，DispatcherServlet就是前端控制器

Controller

1

DispatcherServlet

3

2

View

Model

1. Controller:

Generate（生成）

Model

Controller

1. HandlerAdapter

Knows

Use

HandlerAdapter

Controller

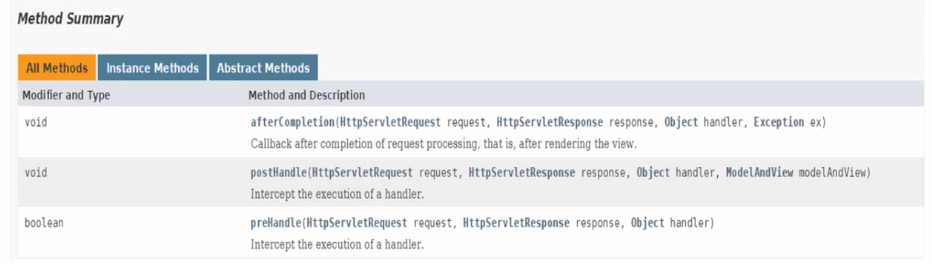
DispatcherServlet

* 1. HandlerAdapter是在DispatcherServlet内部使用的一个类，其实就是Controller的一个表现形式，在Spring MVC中并没有一个interface叫Controller，当然Spring MVC有一个annotation（标签），这个标签是为了识别Controller而引入的
  2. 在DispatcherServlet中调用的Controller最终实际上是以Handler形式出现的
  3. 将各种不同的handler适配成DispatcherServlet可以使用的handler
  4. Handler是Dispatcher调用Controller的一种中间过渡对象
  5. Mapping是DispatcherServlet与Controller之间映射关系的一种类

1. HandlerInterceptor（Interceptor：拦截器）（接口）

就是在需要被拦截的对象的两侧加入一些料

官方文档：



1. HandlerMapping
   1. Help DispatcherServlet to get the right controller

告诉我们请求到来之后，由那个Controller来相应请求

Controller1

HandlerMapping

Controller2

* 1. Wrap controller with HandlerInterceptor（用HandlerInterceptor包装控制器），这也是HandlerMapping需要考虑的事情

HandlerInterceptor

结论：

* + - 1. 在HandlerMapping工作完毕之后，它可以给DispatcherServlet一个HandlerAdapter，这个HandlerAdapter里面就包括了核心的内容，是某个具体的Controller的实例
      2. 这个Controller需要被包裹的HandlerInterceptor，它们会构成一个执行的链条往下走

1. HandlerExecutionChain

HandlerMapping返回的东西包含了HandlerInterceptor和HandlerAdapter，它们之间构成了一个执行链条

Execution Chain（执行链）:（用JAVA的反射机制实现的）

postHandler

controller Method

afterCompletion

preHandle

1. ModelAndView：认为它是Model的具体表现，因为无论是使用Model还是Map，最后，DispatcherServlet统统都会转化为ModelAndView

ModelAndView

Map

Model

Model classes in Spring MVC

1. ViewResolver（视图解析器）

决定DispatcherServlet需要用哪一个视图进行视图呈现

Help DispatcherServlet to Resolve the right View to render page.

作用：根据我们的配置，找出我们需要的那个视图对象

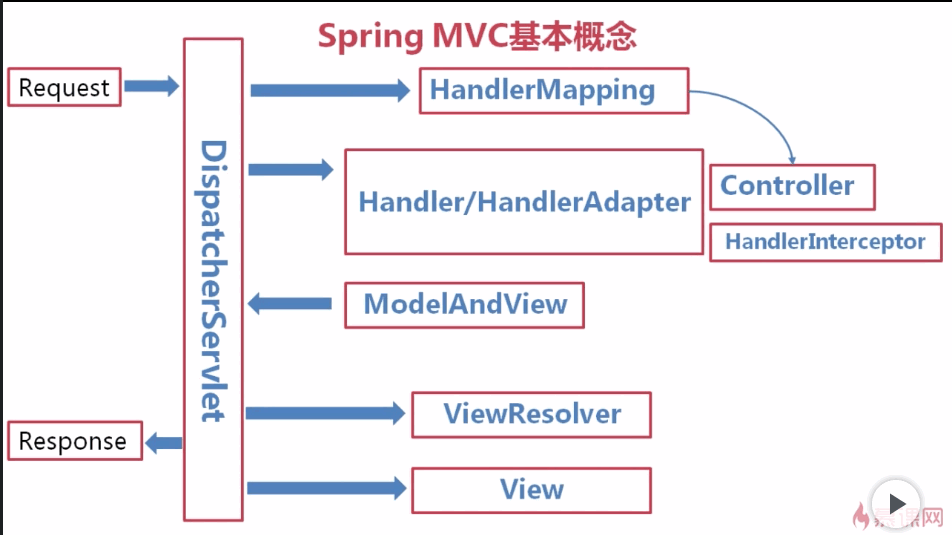
1. View（V in MVC）

Responsible for page rendering

负责视图的呈现

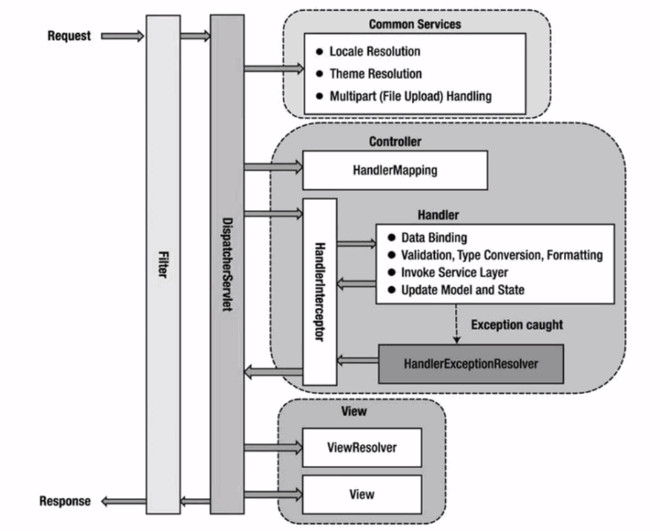
## 第二节 Spring MVC的动态概念

1. 图解：



注：只有Controller必须要写，其他的都只是配置就好了

1. Spring MVC模块划分图解（另一角度）



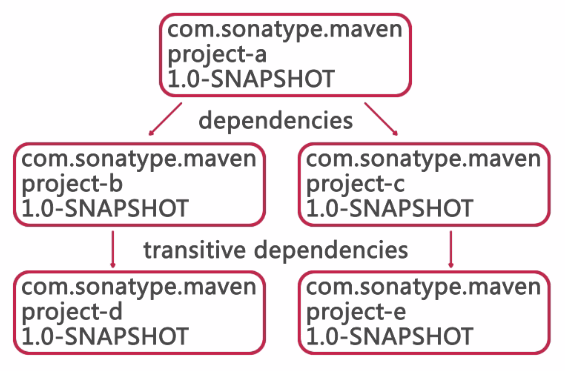
# 第三章 配置Maven环境

## 第一节 Maven介绍

1. Spring MVC项目搭建（使用Maven）
2. Maven是一个构建依赖管理的工具，一下大致讲解三部分内容
   1. POM（Project Object Model）
   2. Dependency Management（依赖管理）
   3. Coordinates（坐标）
3. POM（Project Object Model）
   1. POM仅仅是一个xml文件（pom.xml）,也就是一个主要的配置文件，在配置文件之中会维护很多的信息，来完成我们所需要的功能，比如说：
      1. 依赖管理
      2. 生命周期
      3. 插件的需要
      4. 等等
   2. 这个xml中包含了很多的信息，比如：
      1. Dependencies,
      2. Developers
      3. Organization
      4. Licenses
      5. 等等
   3. POM截图示例：



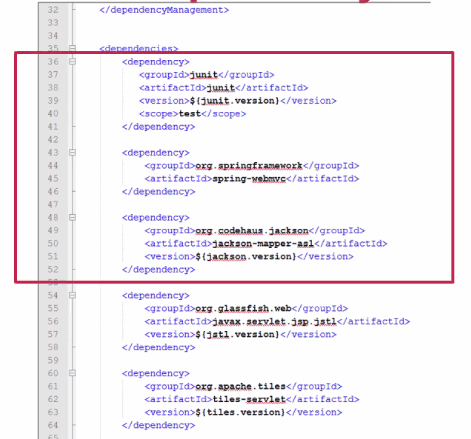
1. Dependency
   1. 图解



* 1. /WEB-INF/lib（jar包存放位置）

这里面可能会包含其他项目中的jar包，因为与其他项目有依赖关系，比如引入了Spring的相关jar包，但是Spring的jar包依赖于其他jar包，此时就需要引入其他的jar包

* 1. 如果手动导入相关依赖包，费时费力，但是可以使用Maven的依赖管理来完成这件事情，仅仅是在pom中配置一些依赖项
     1. 这里在配置文件中写入依赖spring-webmvc之后，spring MVC依赖的其他包Maven会为我们自动载入



1. 实现上面的功能的原理是Coordinates（坐标）（仅仅是一个简单的答案）
   1. 如果将Maven理解为一个仓库，那么所有的软件提供商其实就是将他们的产品发布到我们的仓库之中，由于提供商很多，如何定位某一个软件，就需要坐标这个概念
   2. Maven指定了Coordinates有几种属性：
      1. groupId
      2. artifactId
      3. version
      4. packaging（可以缺省，缺省值为 JAR，表示依赖的是一个JAR包，其他值可能是一个war包，也可能是一个POM对象，POM对象就是用来处理传递依赖的事情）

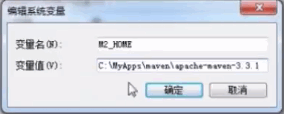
注：这4个属性构成了Maven的坐标，并且可以**唯一标识**一个产品（即为包），可以形象比喻为Maven仓库中的GPS系统，具体示例如上图：

上图中version是用变量来替换它，上面的scope并不是Coordinates的属性，有其他用途

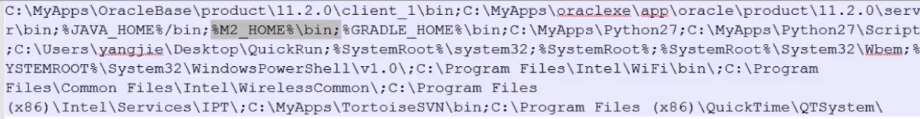
用一个xml文件就可以管理整个项目的依赖管理，包括直接依赖于传递依赖

## 第二节 Maven的安装

1. Maven安装简介（分为三步）
   1. 下载并解压Maven
      1. 在Maven官网选择bin.zip（Binary zip）的下载即可
   2. 配置环境变量（M2\_HOME，Path）
      1. M2\_HOME



* + 1. Path



* 1. 打开cmd命令行测试：

mvn -version

* 1. 配置Maven配置文件（本地仓库路径，镜像）

详见下节

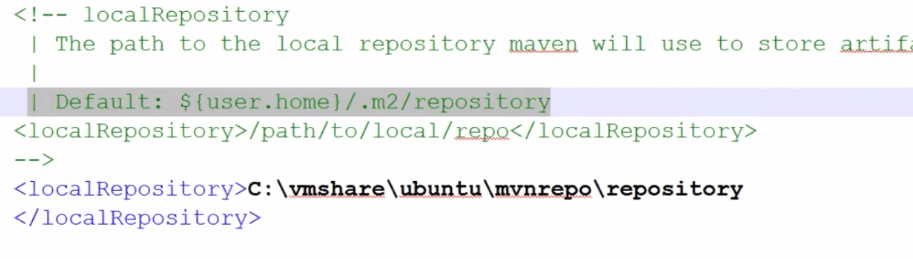
## 第三节 Maven的配置

1. 进入安装目录，打开conf文件夹，找到settings.xml文件，将该文件copy，并复制到%HOME%\.m2目录下，如果只是刚刚安装了maven，而没有运行Maven的任何命令，比如说create等等，那么在本地可能还看不见这个目录，此时有以下解决方法：
   1. 手动创建这个目录（在某个目录下直接新建.m2文件夹，一般都是Users目录下的当前用户名的文件夹中）
   2. 等运行了某个Maven命令之后自动创建

注：

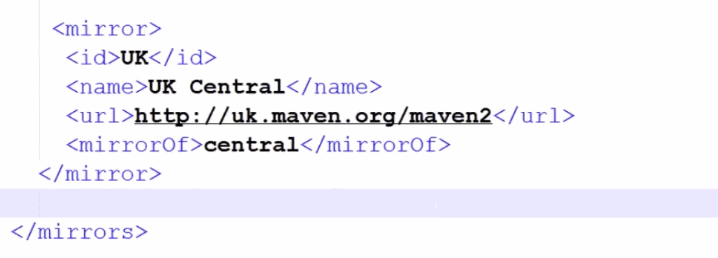
* + 1. 输入 echo %M2\_HOME% 命令查看设置的环境变量，输入mvn -v 查看maven的版本
    2. 输入 mvn help:system 命令然后回车，等其执行完后就可以在 C:\Users\Admin\.m2\repository 看到 Maven 下载的一些文件（必须联网）
    3. 放在.m2目录下是因为，如果Maven版本升级时，就不需要再次修改配置文件了

1. **这是一个可选项，如果不对这一项进行配置，它的软件库就会安装在默认路径中，如果希望放在其他位置，可以选择配置这一项**



1. **mirrors（镜像）：**

**通常指一个中心或者核心东西的副本，Maven的核心仓库的副本就称之为 mirrors，这里按照官方的推荐，使用了一个英国的镜像，每次都去这个镜像仓库去下载软件包，降低了中心仓库的负载，中心仓库为了避免大量的访问造成服务器的问题，可能会拒绝我们的访问，但使用镜像就会使访问的成功率大大提升**

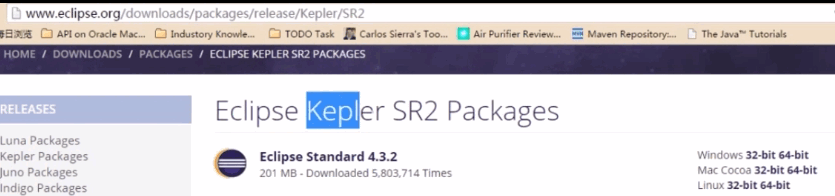
****

## 第四节 用Maven创建项目

1. Eclipse集成Maven（并使用Maven创建一个Spring MVC的开发环境），主要有以下几个步骤：
   1. 下载eclipse（kepler（4.3版本），自带m2e（Maven插件）），否则需要手动将m2e插件集成到开发的IDE中（百度搜索）

<https://blog.csdn.net/u011711295/article/details/39377233>

<https://www.cnblogs.com/nxming/p/6404829.html> **（推荐）**



* 1. 配置eclipse
     1. Window->preferences->Maven->Installations->add，找到Maven安装路径，到bin的上一级目录，点击确定
     2. Window->preferences->Maven->User Settings确保User Settings的配置文件目录是配置了mirrors的那个配置文件
     3. 可以在命令行做一些简单的操作，例如：

mvn archetype:generate –DgroupId=imooc-arthur –DartifactId=spring-mvc-study –DarchetypeArtifactId=maven-archetype-webapp

分析：让Maven为我们自动创建一个目录结构，这样的一个目录结构，

就是一个典型的**web应用的目录结构**，这里就是一个Maven命令

的典型格式

注意：通过mvn调用了Maven的运行环境，后面紧接着的（红色部分）

是调用了某种插件，并且将插件上的某种目标运用，后边是一连串的插件所需要的参数

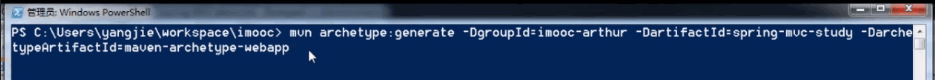
注意：比较重要的就是groupId和artifactId属性，都是坐标中的一个元素，

artifactId的值最终会成为我们所创建的目录结构的根目录的名称，也就是我们最终会创建一个名字叫做spring-mvc-study的目录，来作为项目的根目录

注意：archetypeArtifactId：这个参数指明了我们创建目录类型的一个类型，

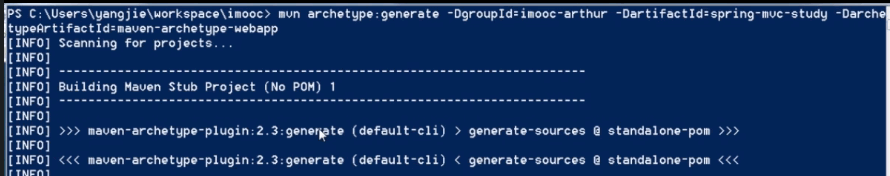
**上面是一个webapp的类型，是一个标准的值**

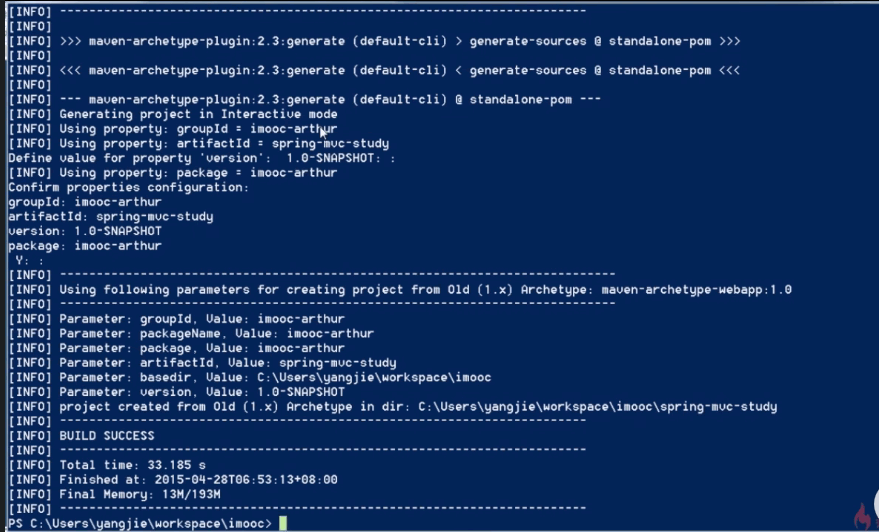
注意：应当在需要创建目录的位置，打开命令行输入命令



注意：执行过程中，可能会提示，让输入未指定的参数，若想选择默认值，

直接按回车键即可



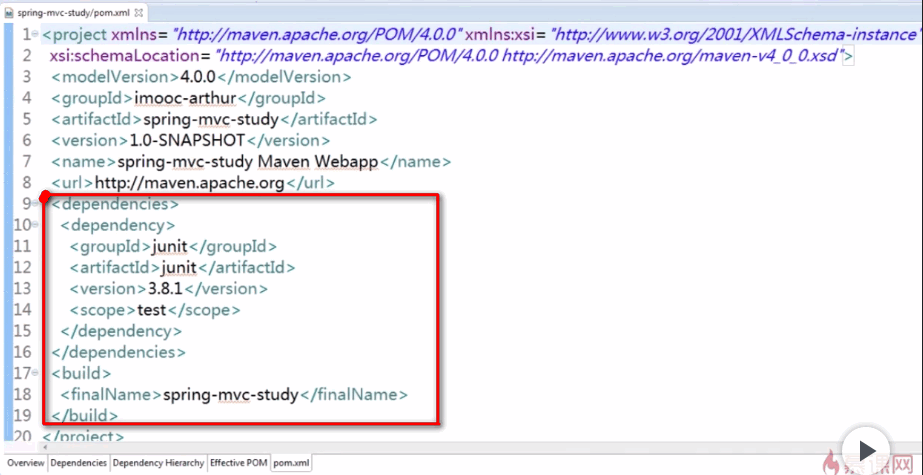


* 1. Hello Spring MVC

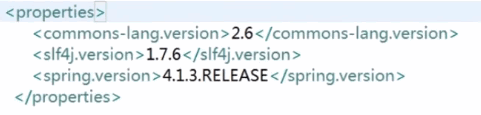
详见下节

## 第五节 Hello Spring MVC

1. 在eclipse中选择导入，选择Maven菜单下的Existing Maven Projects，选择上一节中通过cmd命令创建的工程
2. 查看Properties中的java build path中的order and Export选项卡，查看java目录是否缺失，如果缺失，则需要手动配置：
   1. 在main文件夹下新建java文件夹
   2. 打开pom.xml
   3. Packaging选项中默认为war，可以修改成jar或其他
   4. 点击下方的pom.xml选项卡，下图是默认生成的结构：



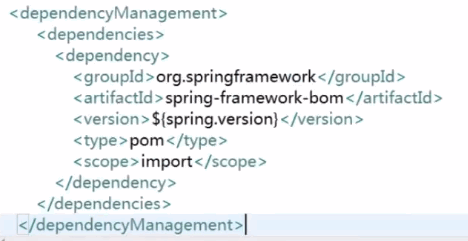
* 1. 对于上图需要做一些改变，可以把红色方框内的内容去掉，然后加入我们自己需要的东西，比如说：
     1. 变量（例如可以用来替代我们需要用到的一些工程版本）



这样在我们整个工程，如果依赖比较复杂，工程比较大的情况下，会比较清晰的理解到我们所使用的依赖软件包的版本

* + 1. 依赖管理（在Maven中是比较核心的东西）

（dependencyManagement）



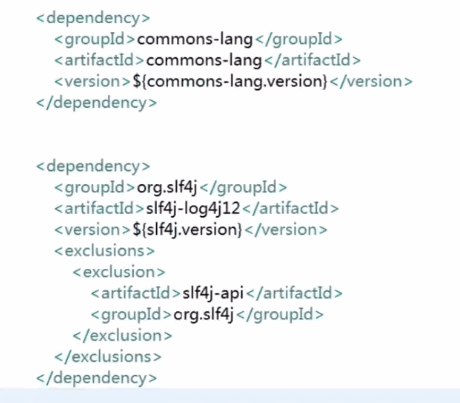
注：Maven可以为我们去管理一些传递的依赖，如果spring MVC和**Hibernate**都依赖了Spring包，当这两个依赖的Spring的版本不一样的时候，上图的配置就发挥了作用，会选择version标签中指定的版本，而标签中的变量，在第一点的图片上有声明这个变量

* + 1. 依赖管理

（dependencies）

依赖管理是我们这个项目所依赖的包，都存放在这个标签中





* + 1. 插件



为我们提供了一些额外的功能，但这些功能和项目本身没有什么太大的关系吗，比如上图，jetty用来测试和部署

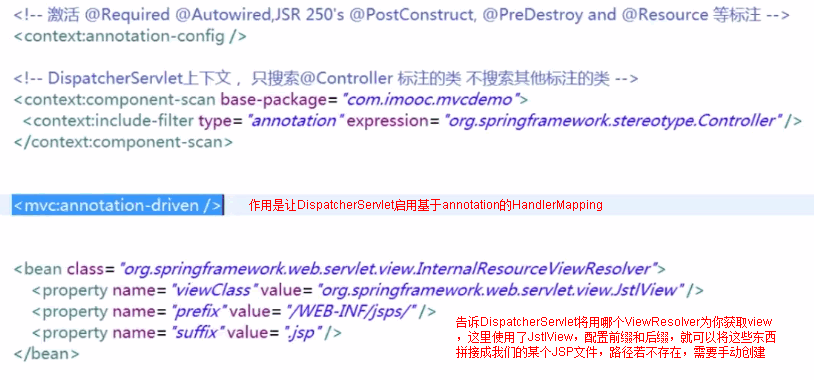
1. 配置web.xml文件

由于要使用Spring MVC，所以我们需要一个Servlet和Servlet-mapping将我们Spring的前端控制器配入我们的web环境

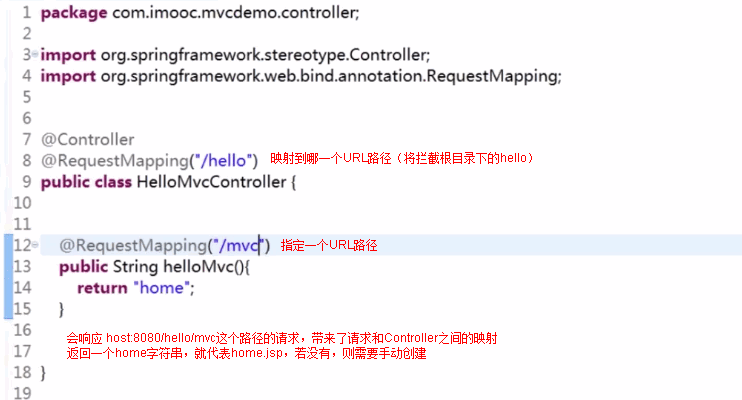


上图改变了配置文件的默认路径，改变之后的路径若不存在，需要手动创建，xml也需要手动创建（因为使用的是Spring MVC，所以该文件是一个标准的Spring的配置文件）

1. 配置$servlet-name$-servlet.xml（一个标准的Spring配置文件）

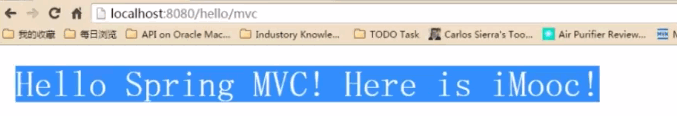


1. 代码实例：



创建完成jsp之后可以在body标签中加入页面元素，若编码集不符合要求，也可以修改

在命令行中进入项目所在路径，这个路径下应该有工程的pom文件，此时在该目录下输入命令：mvn jetty:run，并运行，来启动jetty，为我们提供一个web工程运行的容器环境，当提示信息中显示Start Jetty Server的时候说明Maven已经成功启动，这行信息上方几行中会提示所使用的端口号



此时说明环境已经搭建成功

# 第四章 Spring MVC实操

## 第一节 从配置文件开始

1. 用Spring MVC进行开发——从配置文件开始1
   1. 打开web.xml（web应用的部署描述）



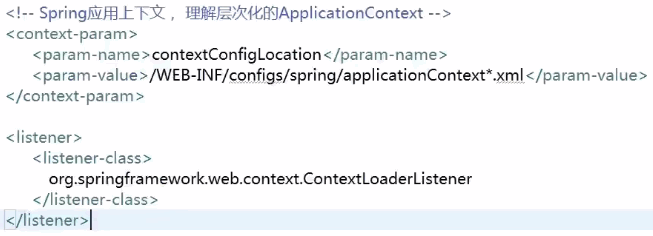
**上图所示的xml是由Maven自动生成的xml中的一部分，它使用了**

**web-app-2.3的标准，这个标准之下，我们的JSP页面会默认的将我们的EL表达式语言关闭，所以最好替换掉，使用2.4以上版本，如下图**



display-name是可以更改的

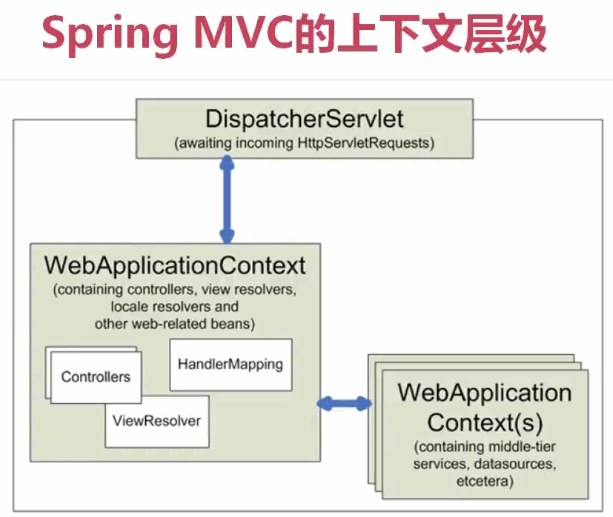
* 1. 加入Spring的声明



* 1. 注：Spring和下面的DispatcherServlet都有自己的相关的配置文件，这样的两组配置文件就构成了它们不同的应用上下文的层次
  2. DispatcherServlet



* 1. Spring MVC的上下文层级图解：



* + 1. WebApplicationContext(s)是根，是ContextLoaderListener所加载形成的上下文，它为我们提供了我们所有应用公共所使用的一些组件和服务，比如Service层，DAO层等等，这些服务应当是被整个应用所共享的，它不应该被局限在某一个DispatcherServlet上下文之中
    2. 另外一部分很显然就是我们与特定DispatcherServlet相关的上下文，比如我们的MVC Dispatcher，与它相关的Controllers、ViewResolver、HandlerMapping等等，可能有多个DispatcherServlet的情况，而既然使用了公共的、通用的上下文，就是为了在多个DispatcherServlet的情况下所共有
    3. 为什么会出现多个DispatcherServlet？

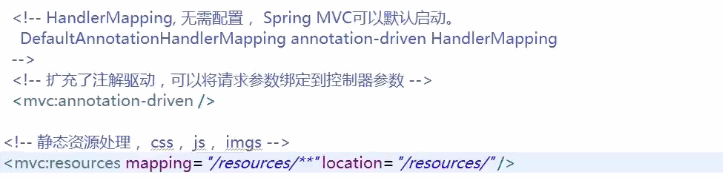
因为互联网世界越来越复杂，所需要提供的服务业越来越多，通过不同的DispatcherServlet来分发，可以更好的服务不同类型的请求（比如慕课网用户的请求、慕课网为其他的机器提供一些webservice的方式、为他们提供课程的检索服务等等）

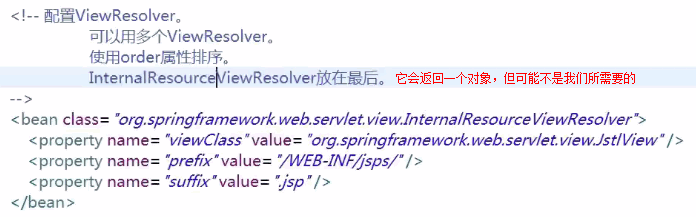
* 1. <servlet-mapping>标签中有一个<url-pattern>标签，这个样式为我们指定了不同url来源的划分依据
     1. /：可以响应来自人类用户的请求
     2. /ws/：可以响应来自机器的请求

1. 用Spring MVC进行开发——从配置文件开始2
   1. 打开mvc-dispatcher-servlet.xml
   2. 图解如下：

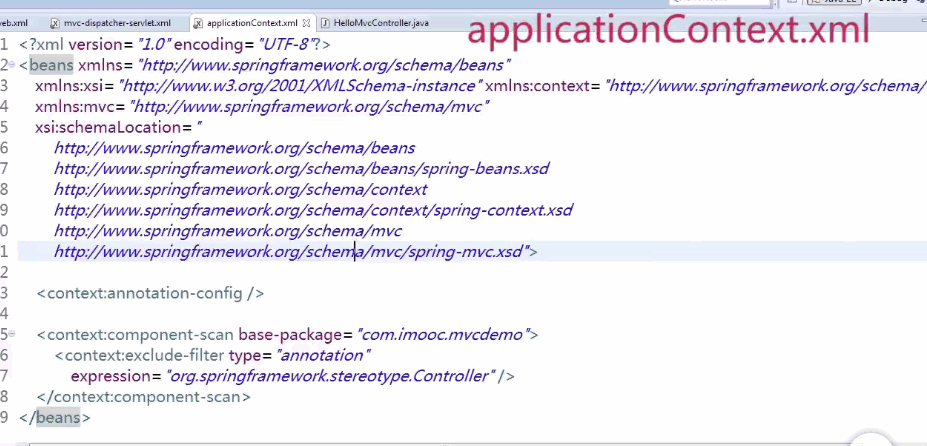








1. 用Spring MVC进行开发——从配置文件开始3
   1. 打开ApplicationContext.xml（Spring相关）
   2. 图解如下：

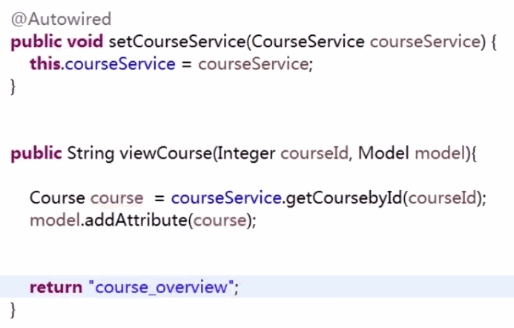


它就是用来放置它或者它的兄弟的配置文件就共同组成了我们整个应用中通用的组件的Spring Bean管理

**注意：这里是exclude-filter，第2点中是include-filter**

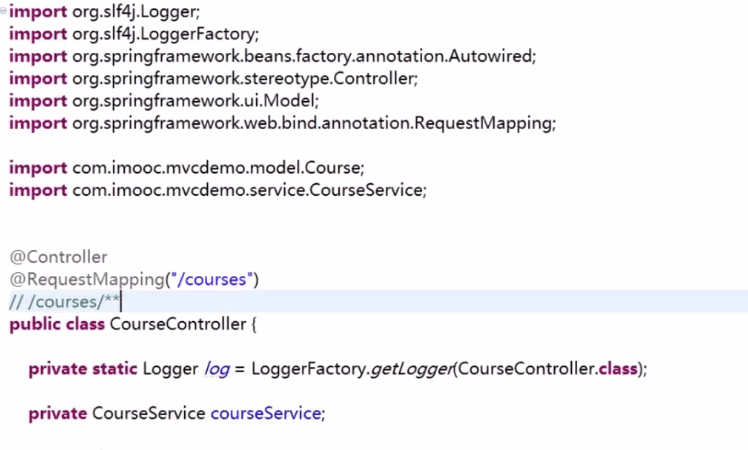
## 第二节 Controller—基础代码

1. Log4j.properties放在resources下，这是一种标准的放置方式
2. Log4j需要的包是org.slf4j.Logger
3. Course：课程
4. Controller中部分代码：具体代码见Demo

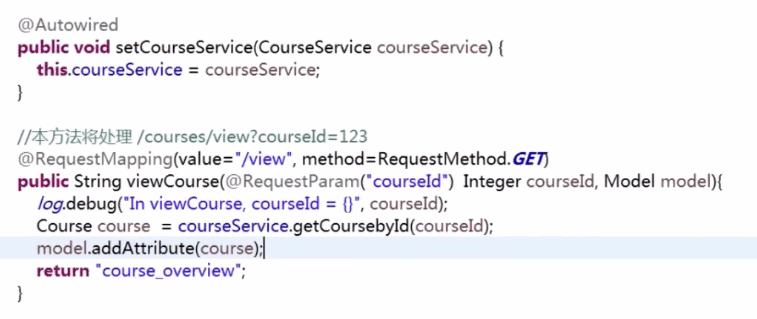


## 第三节 Controller—现代方式

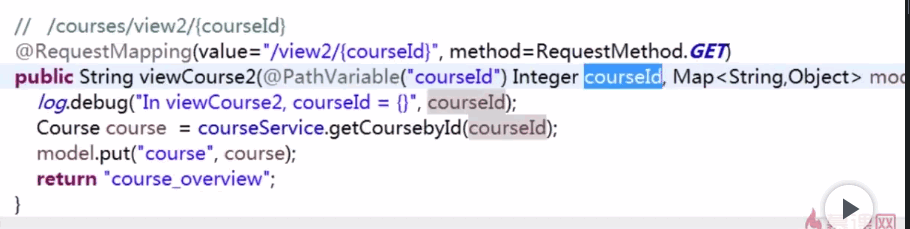
1. @Controller：告诉Spring MVC的dispatcherServlet这是一个Controller，此时这个类会被dispatcherServlet的上下文所管理，并且完成它的依赖注入
2. @RequestMapping：指明这个Controller应该负责处理哪一个或者哪一种类型的URL，因此在类级别上添加该注解（处理根URL），任何该根URL下的路径都会被该Controller所拦截,比如@RequestMapping（"/courses"）



1. @RequestMapping：在方法级别上添加，比如@RequestMapping（value="/view", method=RequestMethod.GET）,将处理/courses/view，并且是由GET方法（方式）传递过来的请求
2. 若该URL请求传递了参数，，比如/courses/view？coueseId=123，此时在参数级别上加注释（在参数上面绑定的annotation），@RequestParam("courseId"),其属性可以显示的指明属性的名称，也就是我们查询字符串的查询变量，对于变量是否正确获取，可以通过日志验证，数据放到model中，在页面上可以通过${course.title}获取



1. Rest风格
   1. <https://www.cnblogs.com/zgqys1980/p/4037468.html>
   2. <https://blog.csdn.net/marvel__dead/article/details/71835477>
2. Rest风格书写代码
   1. {}表示是一个路径变量



注：被遮住的是model

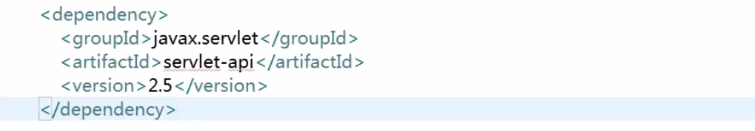
注：此时参数级别的annotation是@PathVariable，而且方法级别的annotation需要显示声明路径的形式，同时在@PathVariable指明该路径变量，对应上图中的入参Integer courseId，这样Spring就为我们完成了变量的绑定及类型的转换

* 1. Rest风格的URL下图：

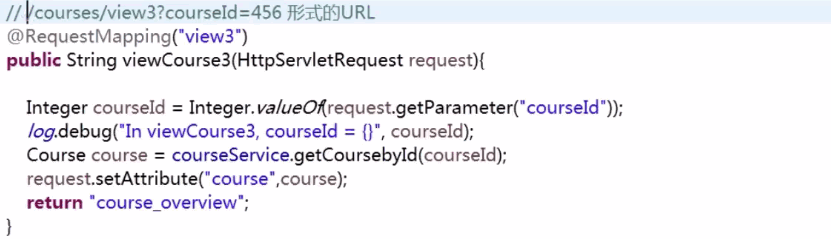
C:\Users\zhangming\Desktop\4.png

## 第四节 Controller—传统方式

1. 若想在代码中使用最古老的HttpServletRequest之类的对象，在Spring MVC中也可以进行类似的集成
2. 在依赖管理中引入servlet API



1. 代码示例：



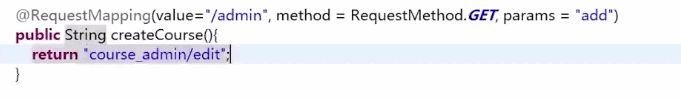
1. Controller要点总结
   1. @Controller
   2. RequestMapping
   3. URL template(@RequestParam and @PathVariable)
   4. HttpServletRequest and/or HttpSession

## 第五节 Binding

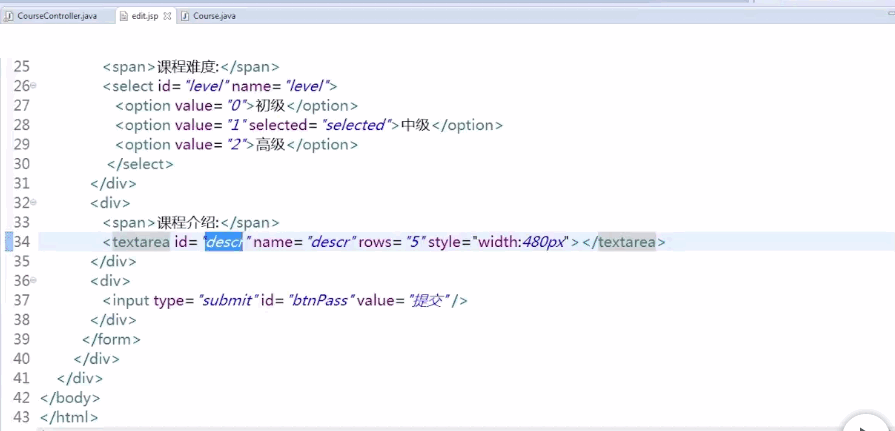
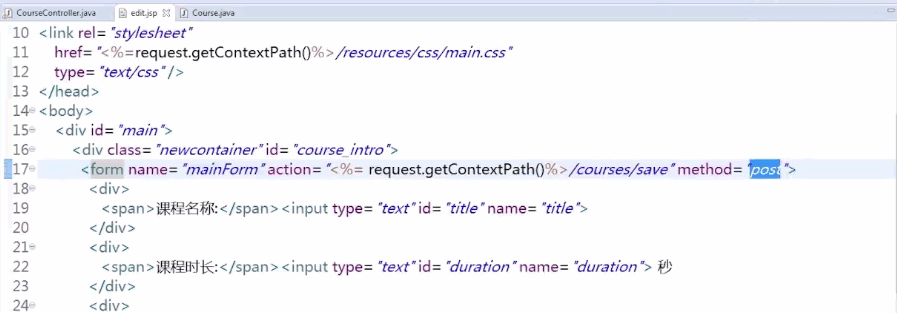
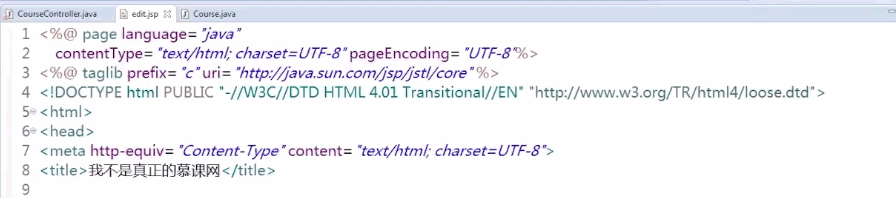
1. 何为Bingding（绑定）

将请求中的字段按照名字匹配的原则填入模型对象

1. Controller示例：（新建）



1. JSP示例：



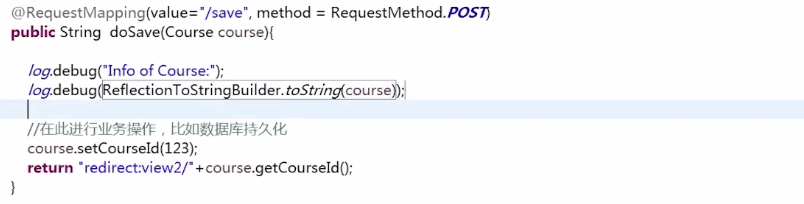
**需要注意的一点是：这些页面控件的name属性，应当和我们模型对象中的属性名称一致**

1. Controller示例：（保存）

在古老的代码编写方式中，我们可以采用比如一个名叫ResquestDispacter的类来进行转发，或者是重定向

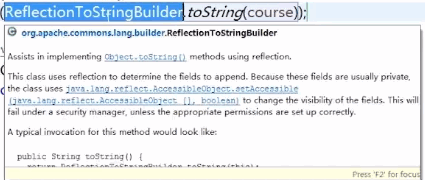
在SPV中实现请求转发和请求重定向非常简单，以请求重定向为例，不需要调用任何特殊的类，仅仅需要使用一个字符串

Return “redirect:view2”

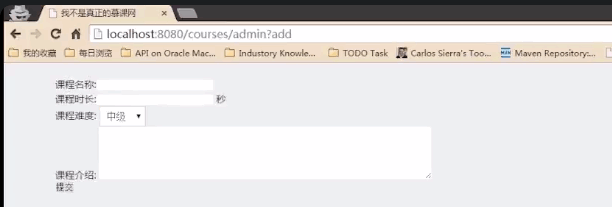


上图的返回形式，是通过rest风格，直接返回某个CourseId所对应的界面

这个类在我们进行一些日志输出来调试我们的程序的时候非常的常见，它的toString方法接收的参数就是某个对象，它会将这个对象所代表的意义的内容以键值对的形式为我们输出在日志当中

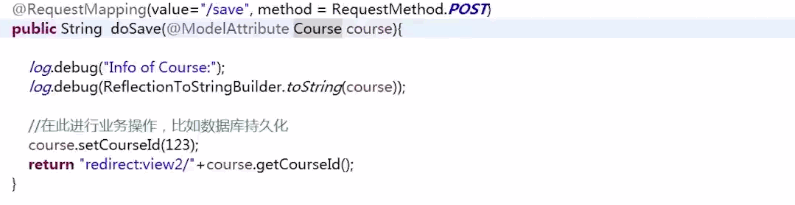


1. 新增页面示例：（value和params与第二点相对应）



点击提交按钮，请求将到达Controller，由拦截器进行拦截，进行处理之后，页面会跳转到展示页面

1. 在Spring MVC中还为我们提供了一个annotation标签，这个标签也可以为我们做类似的工作，这个标签称之为@ModelAttribute（这是一个放在方法参数级别的annotation），通过它也可以完成绑定参数的过程



与上图相比，只是加了一个注解

1. 总结
   1. @ModelAttribute on Method
   2. redirect（重定向）/forward（请求转发）

## 第六节 FileUpload—单文件上传

1. Spring MVC为我们提供了文件上传的内置支持，将它作为一个公共的服务，我们只需要通过一些简单的配置，就可以使用Spring MVC暴露给我们的接口来实现文件上传的工作
2. 打开mvc-dispatcher-servlet.xml文件
   1. 首先找到与dispatcherServlet相关的上下文，在上下文中需要配置如下Bean



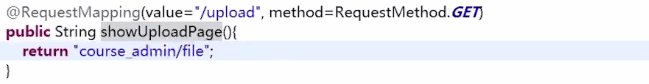
文件大小设置为上限：200M（括号内为字节数）

resolveLazily：延迟加载

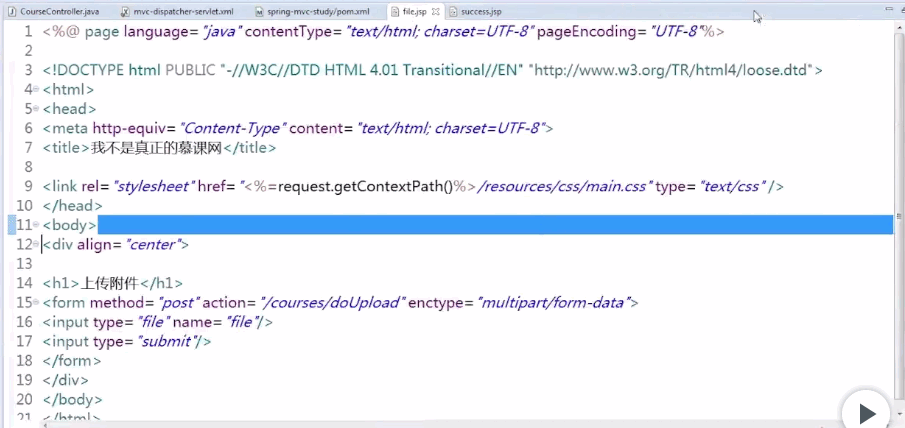
注：这个Bean背后所依赖的是Apache的commons-fileupload这么一个包，所以需要在依赖管理中引入这个包，（pom.xml）如下图：



1. Controller代码实例：



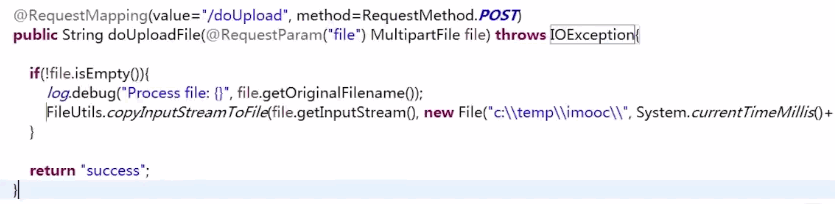
1. JSP实例：



Jsp中有一个**file元素**和一个**提交按钮**

唯一不同之处在于表单中有一个属性是enctype，这个属性是进行文件上传的时候必须显示指明的一个属性，如果没有这个属性，无法完成我们的文件上传工作

1. 点击上传Controller实例



**未截取部分见C小点的第二小点的图例**

* 1. 这个file对应jsp页面的file元素的name属性，这个参数的类型会被转换成MultipartFile。
  2. 对于文件上传的方式，我们通常都是希望使用POST方式
  3. FileUtils所在的包如下图

C:\Users\zhangming\Desktop\1.png

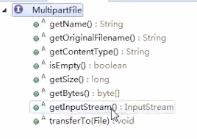
可以做文件的拷贝以及文件流的操作

* + 1. 第一个参数代表我们所要拷贝的源文件所代表的流，一个InputStream
    2. 第二个参数可以new一个File对象，也有两个参数，分别为文件位置和文件名称，如下图所示

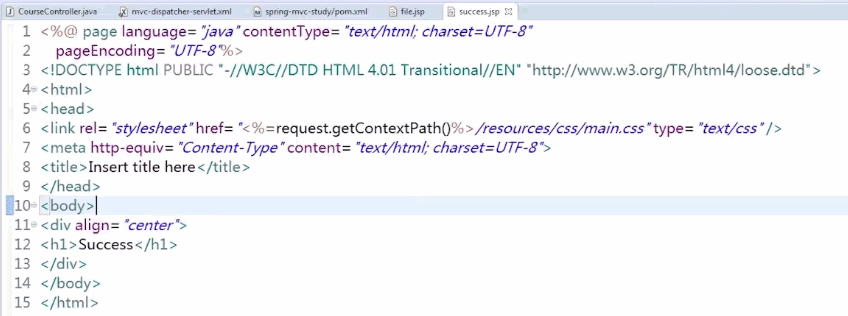
C:\Users\zhangming\Desktop\2.png

1. MultipartFile类的包及方法

C:\Users\zhangming\Desktop\8.png



1. 上传成功返回的JSP实例：



1. Spring MVC并没有限制我们Controller方法的形式，我们可以用任意的参数，任意的名称，任意的返回值和任意的异常等等

## 第七节 JSON（上）

1. JSON的定义（JavaScript Object Notation） is a lightweight data-interchange format

一种轻量级的数据交换**格式**

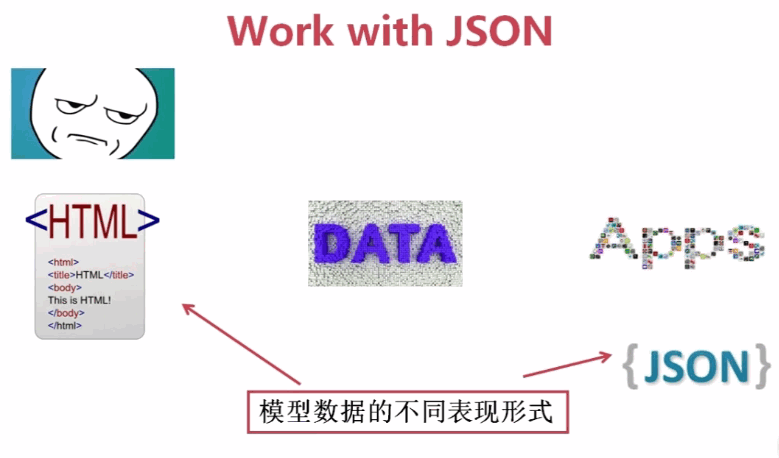
1. JSON示例



* 1. 具有XML一样良好的结构的特点，同时又没有那么冗长的书写格式，也就是说它融合了对人和对机器服务的各种优点
  2. 正是基于这种优点，JSON在Restful Web Service中发挥了重要的作用，也使得Restful Web Service成为了逐渐成为了业界对Web Service的一种事实的标准，逐渐的取代了非常笨重的SOAP
  3. SOAP：

<https://baike.baidu.com/item/%E7%AE%80%E5%8D%95%E5%AF%B9%E8%B1%A1%E8%AE%BF%E9%97%AE%E5%8D%8F%E8%AE%AE/3841505?fr=aladdin&fromid=4684413&fromtitle=SOAP>

1. 对于JSON的初步理解

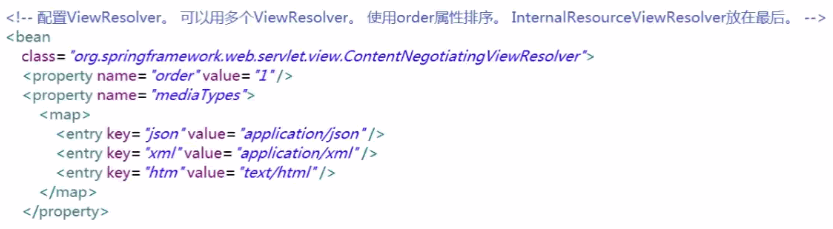


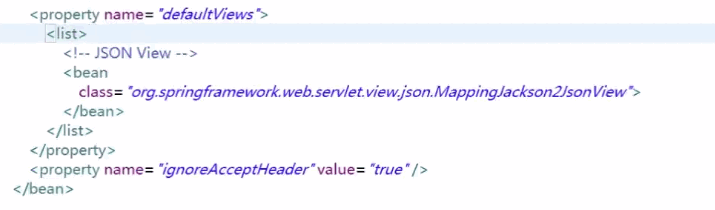
1. Spring MVC使用了一种ViewResolver的机制（ContentNegotiatingViewResolver）来处理这种对**相同数据不同呈现方式要求的应用场景，**用这样一个ViewResolver来处理我们不同的数据呈现格式
2. 左边是人类用户，右边是机器用户



## 第八节 JSON（中）

1. 如果想在Spring MVC中启用对于JSON的支持，首先需要配置一个ViewResolver，它可以让我们的Spring MVC将相同的数据呈现为不同的数据表现形式
2. ViewResolver代码示例



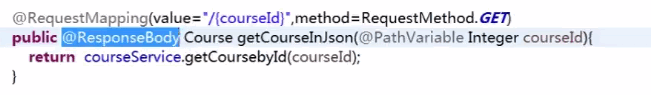


1. 引入依赖



在项目工程中实现java Bean到jackson的转换

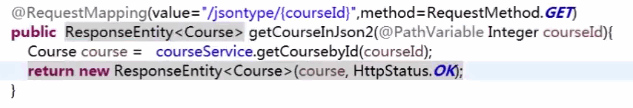
1. Controller代码实例：



从浏览器端访问一个URL请求，这个URL请求把一个模型数据（课程数据）呈现成了一种JSON的数据格式

@ResponseBody：说明这个course会被我们的响应所使用

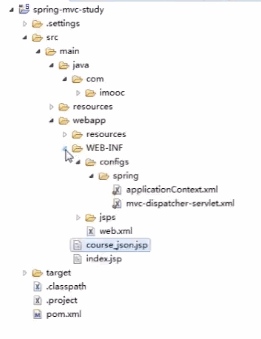
1. @ResponseBody方式并不是唯一的方式，我们只能说它是最简洁的方式，因为它没有太多的其他配置就可以帮助我们完成这项事情
2. 类似也有@RequestBody来获取页面通过JSON方式提交的格式
3. 不仅可以使用GET方法来查询数据，使用POST或者DELETE方法来修改数据也是可以的，它可以将我们的JSON数据推送到服务器端
4. 另一种方法示例：



这种方法只需返回一个泛型类即可，泛型为模型的类型，这样就可以向浏览器返回一个JSON格式的数据

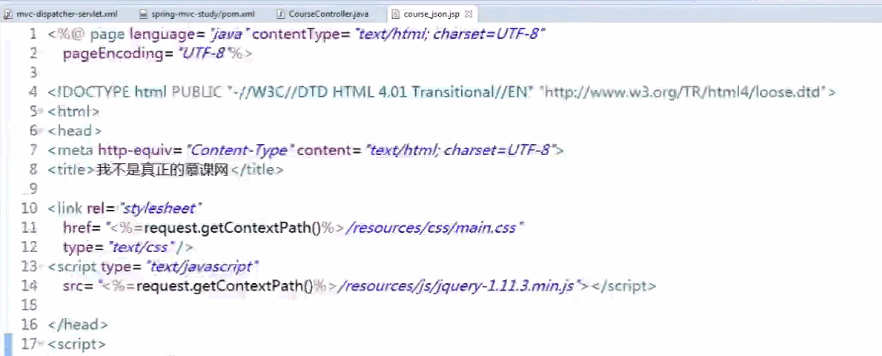
## 第九节 JSON（下）

1. URL直接访问一个JSP页面，传入参数，然后通过参数取到了模型数据，展示了页面，这是用异步的方式获取的数据，从而在前端通过JavaScript代码来完成我们页面整合的一种方式，这也是当今很多应用单页面开发模式所常用的一种方式。
2. 这种方式彻底的将我们后端的业务逻辑和前端的页面呈现相分离，甚至都不会在某一个方法中去组装很多的数据模型
3. 比如说某个页面，我们既需要课程，也需要教师等等，但其实这两个业务模型是相对独立的，在这种情况下我们这种模式就将更好的将我们的业务逻辑和页面呈现相分离，我的某个方法只需要异步的取一下课程信息即可，另外的方法再去异步的获取讲师信息即可
4. 这样通过在前端通过对返回的JSON数据做一个JavaScript的处理，就可以动态的生成当前的页面
5. 具体实现：
   1. 放置位置

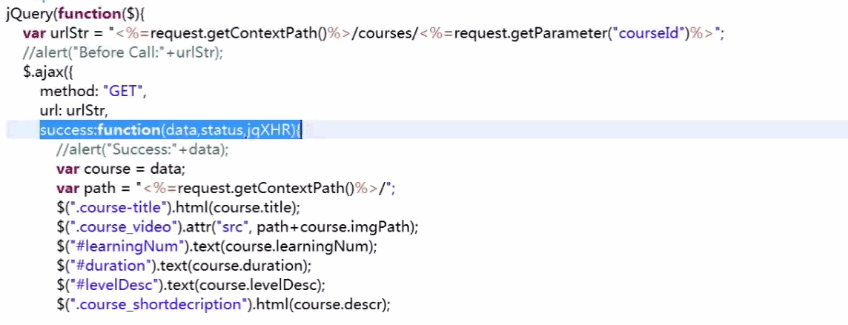


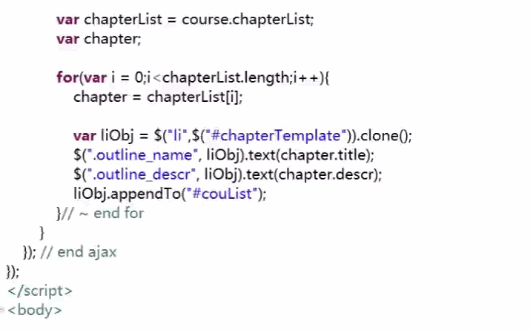
注：course\_json.jsp这个文件是不能放在WEB-INF目录下的，因为WEB-INF下的都是私有的，所以一定要暴露在公共领域（webapp下）

* 1. JSP代码示例



上图引用了在静态资源下的jquery的js文件





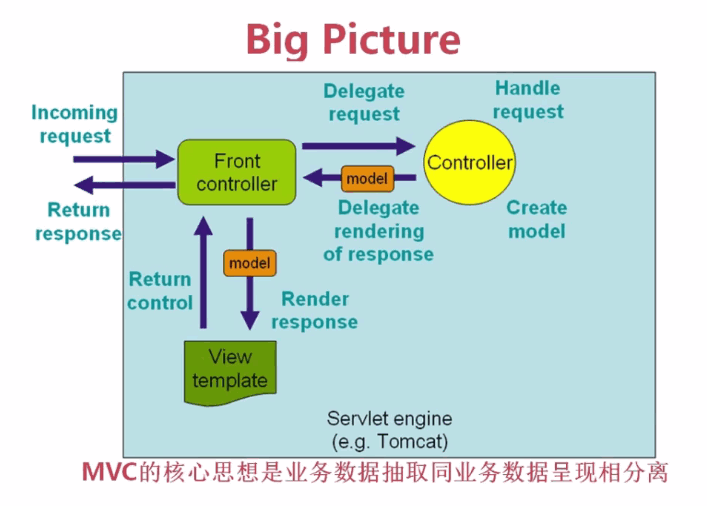
下面还有一些代码，但那些只是一些元素的布局（框架），没有任何的数据，故没有截图

我们通过jquery的这个方式，在页面加载完成之后，通过ajax这个方法去异步的访问服务器端的请求，就是上图中的URL（course下的某个课程），是一种restful的格式，之后指定了GET方法（因为服务器端是GET方法），通过ajax方式访问服务端请求，返回数据之后，通过jquery的动态加载DOM的方式，就实现了页面的呈现

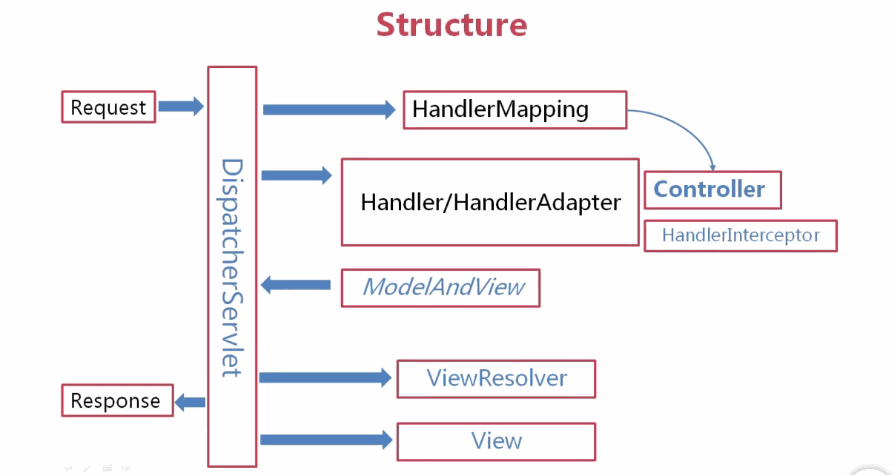
# 第五章 总结

## 第一节 总结

1. 宏观（Big Picture）



1. 中观（Structure）



1. 微观（Details）



1. 微观（Details）—知识点云图

