目录

[1 WEB开发 2](#_Toc39869418)

[1.1 JAVA网络编程基础 2](#_Toc39869419)

[1.1.1 TCP编程 2](#_Toc39869420)

[1.1.2 HTTP编程 2](#_Toc39869421)

[1.1.3 Session与Cookies 2](#_Toc39869422)

[1.2 Servlet 3](#_Toc39869423)

[1.2.1 Servlet类基础 3](#_Toc39869424)

[1.2.2 转发 4](#_Toc39869425)

[1.3 JSP 5](#_Toc39869426)

[1.3.1 基本使用 5](#_Toc39869427)

[1.3.2 调用方法 6](#_Toc39869428)

[1.4 其他WEB组件 6](#_Toc39869429)

[1.4.1 Filter 6](#_Toc39869430)

[1.4.2 Listener 7](#_Toc39869431)

[2 Spring MVC 8](#_Toc39869432)

[2.1 MVC模式介绍 8](#_Toc39869433)

[2.1.1 理论介绍 8](#_Toc39869434)

[2.1.2 控制器实现 8](#_Toc39869435)

[2.2 Spring MVC简介 9](#_Toc39869436)

[2.2.1 控制器实现 9](#_Toc39869437)

[2.3 一个Spring MVC项目的主要部分 10](#_Toc39869438)

[2.3.1 配置文件说明 10](#_Toc39869439)

[2.4 HandlerMapping配置 11](#_Toc39869440)

[2.4.1 配置文件方式 11](#_Toc39869441)

[2.4.2 注解方式 12](#_Toc39869442)

[2.5 ViewResolver配置 12](#_Toc39869443)

[2.5.1 编写Controller类 12](#_Toc39869444)

[2.5.2 编写view 12](#_Toc39869445)

# WEB开发

参考：https://www.liaoxuefeng.com/wiki/1252599548343744

## JAVA网络编程基础

### TCP编程

1. 服务端通过ServerSocket对象监听端口
2. 客户端通过Socket对象与服务端建立连接
3. 服务端每监听到一个连接，创建一个Socket对象，交由一个线程处理
4. Socket对象通过getInputStream()和getOutputStream()获取输入输出流，使用一般流操作进行数据传输

### HTTP编程

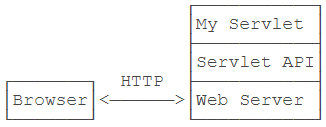
1. 创建HttpClient对象实例，用于发送请求
2. 创建HttpRequest请求对象实例，指定URL、header、body等信息
3. 通过HttpClient对象实例发送请求，返回HttpResponse对象

解析HttpResponse对象，获取返回结果

### Session与Cookies

1. HTTP是无状态协议，无法区分请求的发送者，因此服务端通过session和cookies完成用户身份的区分。
2. 大致流程：
   1. 客户端与服务端完成身份认证
   2. 服务端为客户端创建一个唯一session，保存这次的连接信息和客户端信息
   3. 服务端将session id放在HTTP报文的cookies字段中传给客户端
   4. 客户端之后的请求都要带上session id这个cookies字段，便于服务端确定其身份
3. Cookies是HTTP报文头部的一个字段，可以用于身份验证，也可以存储其他信息，便于两端交互
4. Session是一种会话概念，基于该概念有各种不同的实现
5. Session存储于服务端，cookies存储于服务端

## Servlet



1. 第三方工具包(如Tomcat)实现了HTTP交互底层的方法，叫做Web服务器或者Servlet容器
2. JavaEE提供了Servlet API，我们使用Servlet API编写自己的Servlet来处理HTTP请求，Web服务器实现Servlet API接口，实现底层功能
3. 部署时将我们的代码打包成 .war 文件放入Servlet容器(如Tomcat)的webapps目录下，Servlet容器会去实例化我们定义的Servlet类并注入相应依赖，并且负责请求与具体Servlet之间的映射
4. Servlet容器只会给每个Servlet类创建唯一实例，使用线程池以多线程的形式执行Servlet实例的doGet()或doPost()方法，要注意线程安全

### Servlet类基础

1. @WebServlet(urlPatterns)注解
   1. 指明该类是一个Servlet，并且指明URL映射
2. 每个Servlet类都继承于HttpServlet类，该类实现了各种类型请求的基本处理方法，最常用的就是doGet()和doPost()，根据自己的需求重写这些请求处理方法
3. 每个请求处理方法都有2个参数HttpServletRequest req和HttpServletResponse resp，分别代表着请求和响应
4. HttpServletRequest常用方法
   1. getMethod()：返回请求方法，例如，"GET"，"POST"；
   2. getRequestURI()：返回请求路径，但不包括请求参数，例如，"/hello"；
   3. getQueryString()：返回请求参数，例如，"name=Bob&a=1&b=2"；
   4. getParameter(name)：返回请求参数，GET请求从URL读取参数，POST请求从Body中读取参数；
   5. getContentType()：获取请求Body的类型，例如，"application/x-www-form-urlencoded"；
   6. getContextPath()：获取当前Webapp挂载的路径，对于ROOT来说，总是返回空字符串""；
   7. getCookies()：返回请求携带的所有Cookie；
   8. getHeader(name)：获取指定的Header，对Header名称不区分大小写；
   9. getHeaderNames()：返回所有Header名称；
   10. getInputStream()：如果该请求带有HTTP Body，该方法将打开一个输入流用于读取Body；
   11. getReader()：和getInputStream()类似，但打开的是Reader；
   12. getRemoteAddr()：返回客户端的IP地址；
   13. getScheme()：返回协议类型，例如，"http"，"https"
   14. setAttribute(name, value)：添加新的请求参数
5. HttpServletResponse
   1. 设置响应头
      1. setStatus(sc)：设置响应代码，默认是200；
      2. setContentType(type)：设置Body的类型，例如，"text/html"；
      3. setCharacterEncoding(charset)：设置字符编码，例如，"UTF-8"；
      4. setHeader(name, value)：设置一个Header的值；
      5. addCookie(cookie)：给响应添加一个Cookie；
      6. addHeader(name, value)：给响应添加一个Header，因为HTTP协议允许有多个相同的Header；
   2. 设置响应体
      1. getOutputStream()：获取输出字节流
      2. getWriter()：获取输出字符流
      3. 输出流写完后，调用flush()，刷新缓存，不要调用close()关闭输出流，这会将TCP连接断开

### 转发

1. 内部转发forward()，调用其他Servlet继续处理
   1. req.getRequestDispatcher(urlPatterns).forward(req, resp);
2. 外部转发，告诉浏览器请求另一个URL，属于不同的请求了
   1. resp.sendRedirect(redirectToUrl); //302，临时重定向到redirectToUrl
   2. resp.setStatus(HttpServletResponse.SC\_MOVED\_PERMANENTLY);

resp.setHeader("Location", redirectToUrl);

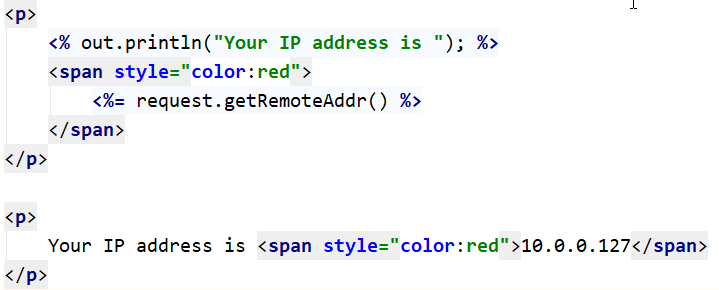
301，利用浏览器缓存进行永久重定向

## JSP

1. 使用HttpServletResponse返回一个HTML页面，需要构造复杂的字符串，可维护性较低，可返回一个.jsp文件进行替代
2. JSP文件类似于HTML文件，可读性较高，同时也可以插入JAVA代码动态构造
3. JSP本质就是一个Servlet，会自动编译成相应的Servlet类，完成HTML页面字符串的构造工作
4. 比较
   1. HTML：静态页面，可以直接打开
   2. JSP：本质是Java代码，先转译成Servlet，运行产生一个HTML页面

### 基本使用

1. Jsp中内置了几个变量，可以直接使用
   1. out：表示HttpServletResponse的PrintWriter
   2. session：表示当前HttpSession对象
   3. request：表示HttpServletRequest对象
2. 几种符号
   1. 注释：<%-- --%>
   2. Java代码： <% %>
   3. 变量的值： <%= xxx %>
3. 样例，下面两种写法是等价



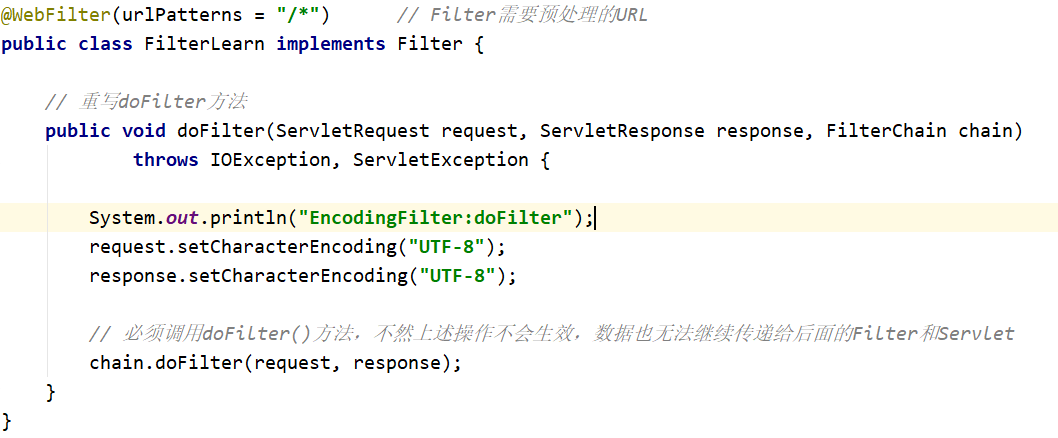
1. 调用其他包，实例化对象，实例化后的对象可以在JSP中使用
   1. 

### 调用方法

1. Servlet先对请求的HttpServletRequest和HttpServletResponse对象实例进行预处理
2. 利用内部转发机制传给相应的JSP进行处理
   1. req.getRequestDispatcher("/WEB-INF/user.jsp").forward(req, resp);
3. JSP处理完，返回给用户HTML页面

## 其他WEB组件

### Filter

1. 预处理类，对请求的HttpServletRequest和HttpServletResponse对象实例进行一些预处理操作
2. 用户请求 -> Web Server -> Filter 1 -> Filter 2 -> ... -> 相应的Servlet
3. 常用于用户验证，对请求进行拦截，验证通过后再交由Servlet处理
4. 

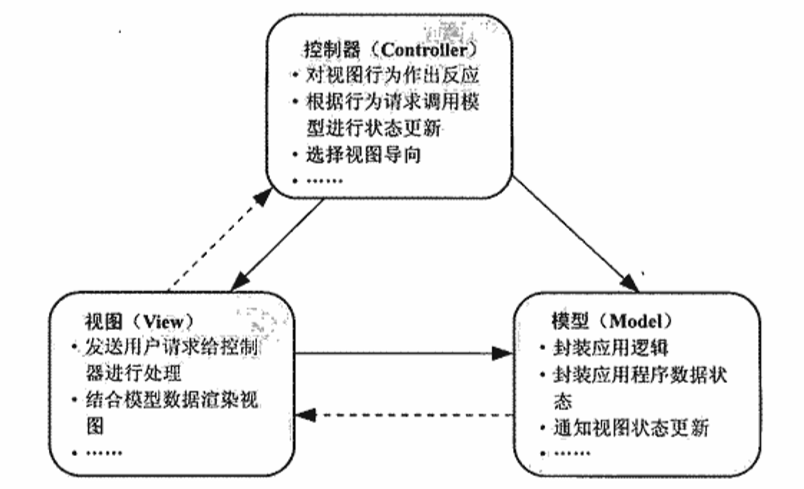
### Listener

1. 用于监听Web服务器事件，触发相应的处理操作
2. 使用@WebListener，有多种事件类型
   1. ServletContextListener 监听ServletContext的创建和销毁，每个APP服务器都会为其创建一个唯一的ServletContext实例
   2. HttpSessionListener：监听HttpSession的创建和销毁事件；
   3. ServletRequestListener：监听ServletRequest请求的创建和销毁事件；
   4. ServletRequestAttributeListener：监听ServletRequest请求的属性变化事件（即调用ServletRequest.setAttribute()方法）；
   5. ServletContextAttributeListener：监听ServletContext的属性变化事件（即调用ServletContext.setAttribute()方法）；

# Spring MVC

## MVC模式介绍

### 理论介绍

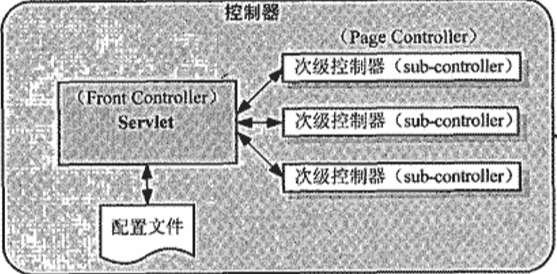


1. 控制器
   1. 响应视图请求，调用相应的模型进行逻辑处理
   2. 选择合适的视图返回给用户
2. 模型
   1. 封装具体的应用逻辑
   2. 通知视图处理完成，进行状态更新
3. 视图
   1. 将用户请求发送给控制器
   2. 根据模型通知，更新视图状态
4. 具体的WEB应用实现时，模型是无法将数据传递至视图，因此都是由控制器调用相应模型，获取返回数据，然后将数据传递给视图

### 控制器实现

传统JSP实现的MVC模型主要问题在于Servlet膨胀，无论是每个请求对应一个Servlet还是一个Servlet处理所有请求，在请求个数增加时都会有膨胀问题。

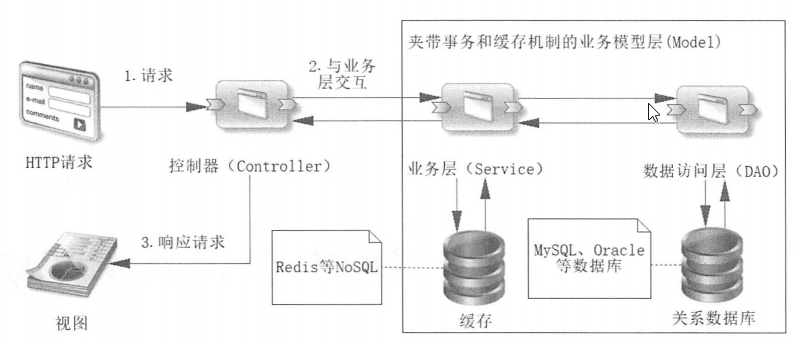
因此，现在的MVC模型对于控制器的实现都是通过两级控制器实现：Front Controller和Page Controller

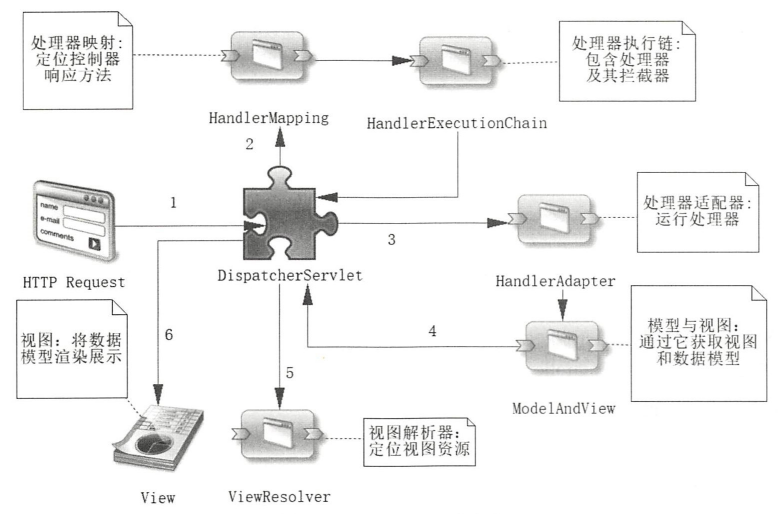


1. Front Controller：只负责加载配置文件中的映射信息，将请求转发给次一级的控制器
2. Page Controller：封装了具体请求的应用处理逻辑

## Spring MVC简介

Spring MVC最大的优势在于对请求处理期间涉及的各种关注点进行了合理而完全的分离，明确设置了不同的角色实现不同的关注点。





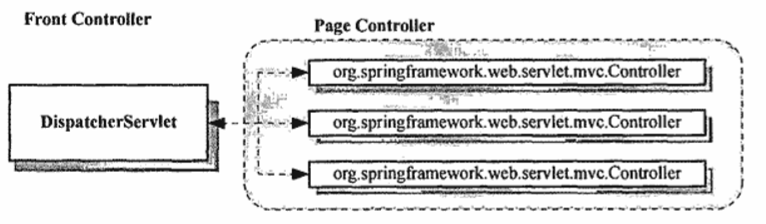
Spring MVC请求处理流程：

1. DispatcherServlet调用HandlerMapping，获取请求对应的HandlerExecutionChain对象，该对象包含相应控制器(controller)封装成的处理器(handler)
2. DispatcherServlet调用处理器适配器(HandlerAdapter)处理输入输出，调用HandlerExecutionChain中包含的处理器对象，返回ModelAndView
3. DispatcherServlet使用ViewResolver查找相应视图，并传入数据model进行渲染
4. 将渲染后视图返回给用户(哪个模块负责返回操作？)

Servlet容器和IoC容器的关系：

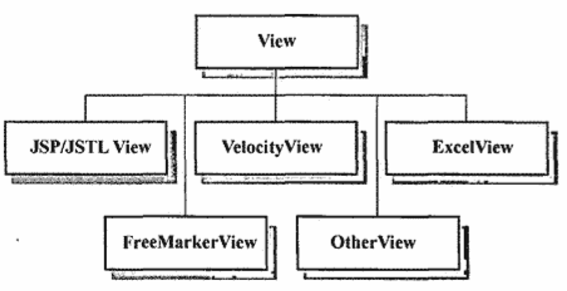
1. Web服务器提供Servlet容器
2. Servlet容器创建DispatcherServlet对象实例
3. DispatcherServlet对象实例负责创建并绑定IoC容器，调用相应的Bean

### 控制器实现



DispatcherServlet的处理请求流程分为三步：

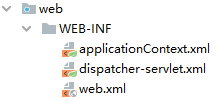
1. 使用HandlerMapping完成请求到具体处理逻辑Controller的映射
   1. HandlerMapping是一个接口，有很多具体实现类将请求URL绑定到Controller上，比如：
      1. RequestMappingHandlerMapping：通过注解绑定URL和Controller
      2. BeanNameUrlHandlerMapping、SimpleUrlHandlerMapping：通过配置文件绑定URL和Controller
2. 调用具体Controller完成Web请求的处理逻辑
   1. Controller处理方法返回值是一个ModelAndView对象，包含两部分信息：
      1. 视图的逻辑名称或具体的视图实例，DispatcherServlet能够通过视图的逻辑名称返回用户具体的视图
      2. 模型数据，用于渲染视图
3. 使用ViewResolver根据ModelAndView对象中的视图逻辑名称查找具体的View实现类，返回View实例给DispatcherServlet；然后DispatcherServlet将ModelAndView中的模型数据传给View实例，完成最终的视图渲染工作，并返回给用户。
   1. Spring MVC提供View接口，有各种不同的实现类来处理不同类型的视图(如jsp/vm/fm/xls/excel等)



* 1. DispatcherServlet负责请求与Controller的映射，ViewResolver负责视图逻辑名称与View实例之间的映射

## 一个Spring MVC项目的主要部分

### 配置文件说明



1. Spring MVC是在Java Web应用的基础上开发的，所以仍然保留着web.xml文件，用于初始化全局共享对象ServletContext容器，主要用于配置以下内容：
   1. IoC容器配置
   2. 监听器listener配置
   3. Servlet配置，使用DispatcherServlet，设置servlet.xml配置文件的位置和名称，一般IoC容器也是绑定到DispatcherServlet上
   4. servlet映射，全部请求都映射到DispatcherServlet上
2. applicationContext.xml
   1. 容器ApplicationContext的配置信息，包括应用所需要的中间层服务配置，比如JDBC、数据源定义、数据访问对象定义、数据服务对象定义、事务管理等
   2. 也可以配置我们自己创建的类，比如各种Controller，交由IoC容器来管理类的创建和依赖管理
3. dispatcher-servlet.xml
   1. DispatcherServlet的配置，主要包括两部分：HandlerMapping与ViewResolver

## HandlerMapping配置

1. HandlerMapping不止可以将请求映射到Controller上，还可以映射到其他次级控制器类型，统称为Handler
2. 一个DispatcherServlet可以使用多个HandlerMapping，不过要通过 order 属性设置其优先级
   1. 
3. 默认优先级是Integer.MAX\_VALUE，优先级最低
4. HandlerMapping接口有两大类具体实现类，一种通过配置文件指定URL与Controller的映射；一种通过注解指定映射。

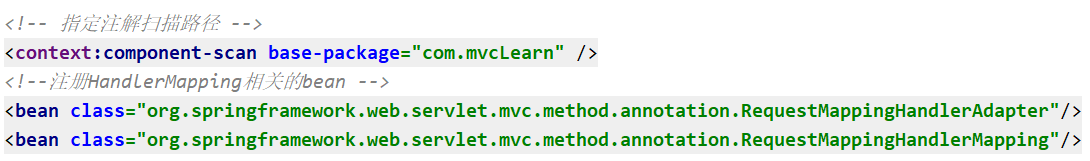
### 配置文件方式

1. 先在applicationContext.xml中注册各个Handler的bean，或者通过注解@Bean和@Controller注册bean
2. 在dispatcher-servlet.xml中创建HandlerMapping的bean，指明URL与Handler bean之间的映射关系



### 注解方式

1. 在web.xml中启动注解，指定扫描路径



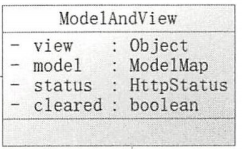
1. 在具体的类似使用@Controller标识处理器类，通过@RequestMapping、@GetMapping或@PostMapping指定映射

## ViewResolver配置

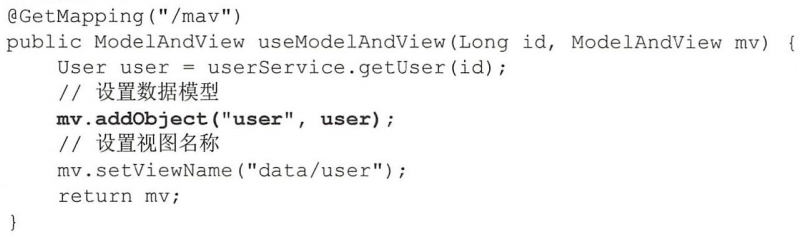
ViewResolver一般只需指定view视图文件的前缀与后缀，最终的视图文件路径=前缀+视图逻辑名称+后缀



## ModelAndView



### 数据模型model

1. ModelMap继承于LinkedHashMap
2. 对于控制器方法中出现的Model、ModelMap类型的参数，SpringMVC会自动为其创建数据模型对象，将其绑定到ModelAndView中进行返回
   1. 
   2. 

### 视图view

1. 视图分为两种，逻辑视图和非逻辑视图。逻辑视图需要视图解析器进一步定位，然后渲染；非逻辑视图可以直接渲染。
2. ModelAndView中view对象可以是一个字符串，也可以是一个视图对象，比如自定义的视图类型

# REST风格

1. Representational State Transfer
   1. Representational：资源
   2. State：表现层，资源表现形式
   3. Transfer：资源可以被新增、修改、删除
2. 特点
   1. 服务器存在一系列资源，每一额资源通过单独唯一的URL进行标识
   2. 客户端和服务器之间可以相互传递资源，资源会以某种表现层得以展示
   3. 客户端通过HTTP协议所定义的动作对资源进行操作，以实现资源的状态转换
3. HTTP操作

