# 乱码处理

1. 开始编码前统一设置：workspace + resource + jsp

Servlet中：

request.setCharacterEncoding(“utf-8”)

response.setCharacterEncoding(“utf-8”);

1. 使用new String(byte[],charset)方法重组字符串
2. 过滤器统一过滤

# 请求转发与重定向

1. Request.getRequestDispatcher(“”).forward(request,response)：

页面地址不变化，默认使用post提交！！，request，response对象被直接传递，在服务端执行，对于浏览器（用户）是透明的。

ServletContext.getRequestDispatcher(“”)只能使用绝对路径，而ServletRequest.getRequestDispatcher(“”)可以使用相对路径

1. Response.sendReDirect(“”)：

页面地址变化，默认使用get提交，request，response对象被丢弃，重新生成。此时request和response的编码设置也丢弃，注意设置编码。

1. sendRedirect()会新建request对象，所以上一个request中的数据会丢失。 更具体来说就是这样的：   
   redirect 会首先发一个response给浏览器, 然后浏览器收到这个response后再发一个requeset给服务器, 然后服务器发新的response给浏览器. 这时页面收到的request是一个新从浏览器发来的。  
   forward 发生在服务器内部, 在浏览器完全不知情的情况下发给了浏览器另外一个页面的response. 这时页面收到的request不是从浏览器直接发来了,可能己经用request.setAttribute在request里放了数据.在转到的页面可直接用request.getAttribute获得数据。
2. forward只能跳转到同一个Web应用中的资源，而sendRedirect可以跳转到任何URL

[*http://www.cnblogs.com/phpzxh/archive/2010/02/01/1661137.html*](http://www.cnblogs.com/phpzxh/archive/2010/02/01/1661137.html)

1. form 的action：

form的action提交方式既不是请求转发也不是重定向，form提交由客户端发起（重定向和转发都属于服务端动作）。

# ? 号传参

只能传递字符串；request.getParameter(“”)

？后有中文字符时不能直接传递，需要将中文字符的编码转为 ISO-8859-1

String msg = new String("登陆失败".getBytes("utf-8"), "ISO-8859-1");

# HttpSession 会话

1. 保存上下文信息，实现会话跟踪，针对每一个用户的值，变量保存到服务器端，通过sessionID区分不同的客户
2. session何时被删除：

程序调用HttpSession.invalidate()；

距离上一次调用时间超过MaxInactiveInterval时间（默认时间30 minute）；

服务器进程被停止；

关闭浏览器只会使存储在客户端浏览器内存中的seesion cookie失效，不会使服务端的session对象失效。

1. isNew: 如果会话尚未与客户端发生任何联系或者客户端选择不加入会话时返回true；

比如，如果客户端只使用基于cookie的会话，而浏览器恰恰禁用了cookie，那么无论何时调用都返回true，因为会话对浏览器而言始终是新的。

isNew返回false，只能说明用户曾经访问过该web应用，并不一定代表其曾经访问过我们的具体哪个servlet或jsp页面。

正确的做法是判断session中是否有特定key存在，及其值是否正确。

1. 关闭浏览器不会使session销毁，服务器需要为session设置一个失效时间。

<http://www.cnblogs.com/zhaoYuQing-java2015/p/4621384.html>

1. session 无法跨浏览器获取数据。即更换浏览器会开启新的会话。

# ServletContext

1. 一个 java web应用只创建一个ServletContext对象，一般用于在多个客户端(浏览器)之间共享数据。
2. ServletContext被定位于 <http://localhost:8080/project/>开始的所有请求被发送到与此应用关联的Web应用程序。
3. ServletContext提供对应用作用域与对象选择：

request：跟当前操作相关

session：更跟用户相关

application：跟项目全局信息相关

ServletContext在服务器关闭Web应用时销毁。

# javax.servlet.Filter

1. 过滤器位于客户端和Web应用程序之间，用于检查和修改两者之间流过的请求和响应。
2. 在请求到达Servlet/Jsp之前，过滤器截获请求；

在响应到达客户端之前，过滤器截获响应。

1. 多个过滤器形成一个过滤器链，过滤器链中不同过滤器的先后顺序由部署文件web.xml中过滤器映射<filter-mapping>的顺序决定。
2. 最先截获客户端**请求**的过滤器将最后截获Servlet/JSP的**响应信息**。
3. 过滤器的链式结构：

为一个Web应用部署多个过滤器，这些过滤器组成一个过滤器链，每个过滤器只执行某个特定的操作或检查，这样请求在到达被访问的目标之前，需要经过这个过滤器链。

1. 在过滤器的doFilter()方法实现中，任何出现在FilterChain的doFilter方法之前地方，request是可用的；在doFilter()方法之后response是可用的。
2. url-pattern 与 dispatcher 是且的关系，只有同时满足两者，该Filter才会生效

# JSP（java service page）java 服务器页面

1. Servlet 缺陷：

写html标签可读性差；

html可以转化为js，servlet不能转换；

编码处理麻烦；

jsp弥补了上述不足，jsp通过在标准的HTMl页面中嵌套java代码，使其静态的部分无需java控制，只有那些需要从数据库或动态生成的页面内容，才使用java脚本控制。

1. 语法：

<% %> ：java脚本

1. 每一个jsp页面就是一个Servlet实例。Servlet容器会负责将jsp转为servlet类。
2. 工作原理：

（1）如果是第一次请求页面，或页面发生改动，则Servlet容器首先将jsp页面转化为Servlet实例，再将其编译为class文件，编译过程会耗费时间。

（2）jsp容器负责调用从jsp转换而来的Servlet，这些Servlet负责响应用户的请求（提供服务），容器会为Servlet建立一个线程，如果有多个请求，则建立多个线程。

（3）容器执行Servlet的字节码文件，返回的最终方式为html格式的文件流。

1. jsp 页面转换为Servlet时仅将html内容用JspWriter（out）输出。转化结果文件查看路径：D:\Duan\tomcat\apache-tomcat-8.5.16\work\Catalina
2. 自动转换的Servlet的继承结构：\*\*\*\_jsp extends HttpJspBase extends HttpServlet：jsp 页面中的静态内容和动态内容都由 \_jspService 方法使用JspWriter（apache的实现为JspWriterImpl）对象输出到客户端（\_jspService 方法调用封装在HttpJspBase 的service 方法中）。

# Servlet

1. Servlet只在第一次使用（请求）时加载和实例化，由容器管理和控制servlet的生命周期。
2. Servlet一旦被加载，一般不会从容器中删除，直至服务器关闭或重新启动，但当容器做内存回收动作时，servlet有可能被删除，也正是这个原因，第一次访问Servlet所用的时间要大大多于以后访问所用的时间。
3. Servlet在服务器中的运行：加载 -> 初始化 -> 调用 -> 销毁。
4. Servlet不是单例的，不过一般一个Servlet只会有一个实例。
5. Servlet的工作过程：
6. 加载和实例化

如果Servlet容器还没有实例化一个Servlet对象，此时容器装载和实例化Servlet，创建一个实例，如果已经存在则不创建新实例。

初始化

1. 在产生Servlet实例后，容器负责调用该Servlet实例的init()方法，在处理用户请求之前，完成初始化。
2. 处理请求，当Servlet容器接收到请求时，运行对应Servlet实例的service()方法，service方法根据请求类型调用相应的方法（doPost，doGet等）。
3. 销毁，当容器决定将一个Servlet从服务器移除时（如Servlet文件更新时），便调用Servlet的destroy方法。

其中（1）、（2）、（4）在整个生命周期中只会被执行一次。

JSP/Servlet工作原理 ：<http://blog.csdn.net/insistgogo/article/details/20788749>

Servlet单实例多线程模式：<http://kakajw.iteye.com/blog/920839>

# JSP内置对象page和pageContext

* 1. javax.servlet.jsp.PageContext pageContext：代表jsp页面上下文，访问页面共享数据。
  2. 使用pageContext将属性设置到session范围中，仅PageContext 有该方法。pageContext.setAttribute(“page”,”first page”,PageContext.SESSION\_SCOPE);
  3. page代表当前页面，即自动生成的Servlet。
  4. pageContext.findAttribute(String name)：

该方法会依次在四个作用域中寻找，顺序为(由内而外)：page -> request -> session -> application，无需知道作用域就可以找到存在的数据。

# JSP内置对象application

javax.servlet.ServletContext application：

* 1. 全局作用范围，整个Web应用内共享，生命周期为应用启动到停止。服务器启动后就产生了一个application对象，当用户在访问Web应用的不同页面时application始终为同一个，直到Web应用停止。
  2. 存在于服务器的内存空间，在application对象的生命周期中，该应用的任何jsp页面都能获取到application对象绑定的参数。（不借助数据库就能实现聊天室功能）。

# JSP内置对象out

javax.servlet.jsp.JspWriter out：

实现类：org.apache.jasper.runtime.JspWriterImpl

JspWriter与PrintWriter异同：JspWriter可以在jsp页面中直接输出，亦可用 pageContext.getOut() （与内置对象 out 为同一个对象）方法获得；

同时用两者输出内容时，不管JspWriter与PrintWriter在程序中的顺序怎么样,始终先会输出PringWriter中的数据然后再输出JspWriter中的数据.这是因为out对象相当于插入到了PrintWriter前面的缓冲区中.out对象满足一定条件时,才会调用PrintWriter对象的print()方法,把out缓冲区中的内容输出到浏览器端。

<http://blog.csdn.net/zhd_superstar/article/details/6588222>

# 静态导入和动态导入

<%@include file=”index2.jsp”%>与<jsp:inclde page=”index3.jsp”></…> 异同

1. %@include - 静态包含：

（1）被包含页面中定义的java变量被修改声明为**包内权限的全局变量**（不静态包含时为\_JspService方法（HttpJspBase中的service调用）的局部变量）。方法原样包含。

（2）只对应生成一个Servlet文件（容器只转换生成一个java文件），两部分的内容被合并。

1. <jsp:include – 动态包含：内部实现为，request.getRequestDispatcher(resourcePath).include(request,new ServletResponseWrapperInclude(response,out));

其中RequestDispatcher的 apache实现为ApplicationDispatcher，最终调用方法为ApplicationDispatcher的doInclude方法。

RequestDispatcher接口的include方法定义注释为：将servlet、jsp page、html的内容包含进response，目标jsp和动态包含的jsp都会被转换为Servlet并编译。

需要传递参数时可使用<jsp:param name=”” value=””>标签，底层实现为？号传参。见**20.RequestDispatcher的include与forward的区别**

## 15. 内置对象out.print()与out.write()

out.print(100)：将内容转化为字符串打印到页面

out.write(100)：将原有内容输出（ASCI码）

## 16. Servlet自启动

web.xml

<servlet>

// 初始化参数

<init-param>

<param-name>isAdmin</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

1. load-on-startup：

（1）容器启动时自动加载Servlet（实例化并调用其init方法），值必须是一个整数，表示servlet的加载顺序（大于等于0）

（2）当值小于0或不指定时表示加载时机由容器自行决定。

（3）正数的值越小，该servlet的优先级越高，应用启动时就越先加载，当值相等时，容器会自己决定加载顺序。

## 17. 监听器

1. 用于监听ServletContext、HttpSession、ServletRequest等域对象的创建和销毁事件，还可以监听域对象的**属性**发生修改的事件，可以在事件发生前或者后做一些必要的处理。

2. 在web.xml中注册相应的监听即可，如果一个web.xml中注册了多个监听器，则监听器的启动顺序按照web.xml中的注册顺序启动。

3. 当web.xml同时有servlet、filter、listener时加载顺序为：listener -> filter -> servlet

****

## 18. Session的创建时机

<http://www.360doc.com/content/12/0511/12/1542811_210284774.shtml>

session不是一打开网站就立刻建立，它的建立需要基于下面两个条件中的任意一个：

（1）在servlet中手动调用getSession()（有就返回，没有就创建一个再返回），或getSession(true)（为true时同getSession方法，为false时有就返回，没有返回null）。jsp页面被容器自动转换为servlet时会在其\_JspService方法（该方法在有请求发生时才会调用）中调用getSession方法以为9大内置对象之一的session赋值。而\_JspService调用的前提为有请求发送到Servlet。

（2）jsp页面中设置<%@page session=”true”%>，默认情况下为true，可手动设置为false。

## 19. 设计模式之外观模式（Facade）

外观模式是为了**解决类与类之间的依赖关系**，像spring一样，可以将类和类之间的关系配置到配置文件中，而外观模式就是将他们的关系放在一个Facade类中，间接调用，降低了类之间的耦合度。

通过外观的包装，使应用程序只能看到外观现象，而不会看到具体的细节对象，这样会**降低程序的复杂度，提高程序的可维护性**。

用户只需直接与外观角色交互，用户与子系统之间的复杂关系由外观角色充当，降低耦合度。

如jsp 9大内置对象之：session、application、config

1. session：org.apache.catalina.session.**StandardSessionFacade**

实现HttpSession，同时内部组合有正真的HttpSession。

2. application：org.apache.catalina.core.**ApplicationContextFacade**

实现ServletContext，内部组合ApplicationContext（实现ServletContext），

3. config：org.apache.catalina.core.**StandardWrapperFacade**

实现ServletConfig，内部组合StandardWrapper（实现ServletConfig）；

## 20. 设计模式之装饰模式（Decorator或Wrapper）

对客户透明的方式**动态的给一个对象附加上更多的责任**。装饰模式的典型应用是javaI/O流。

要求装饰对象和被装饰对象实现同一个接口。

## 21.RequestDispatcher的include与forward的区别

（1）forward方法可以将http请求转发给其它的Web资源（jsp，servlet，html），并产生响应。

（2）include方法与forward非常类似，不同的是，include方法将请求转发给其它Servlet后，被调用的servlet虽然可以处理这个http请求，但最后的主导权仍然是原来的servlet。

（3）RequestDispatcher是Web资源包装器，可以把request传递到任意一个目标资源。

（4）include将资源包含到当前的输出中。

（5）注意：只有在尚未向客户端输出响应时才可以调用forward方法，如果页面缓存不为空在重定向前将自动清空缓存，否则将抛出一个IllegalStateException。

# day6 2017-09-28

ajax，cookie和session的关系，session创建时机e.g

<http://www.cnblogs.com/rush/archive/2012/05/15/2502264.html>

## 22. AJax （asynchronous javascript and xml）

<http://www.cnblogs.com/mingmingruyuedlut/archive/2011/10/18/2216553.html>

1. 是一种用于创建快速动态网页的技术，时一种无需重新加载整个网页的情况下，能够更新部分网页的技术，Ajax原则——**按需取数据。**

2. 通过在后台与服务器进行少量数据交换，Ajax可以实现异步更新，这意味着可以在不重新加载整个网页的情况下，对网页的部分进行更新。（传统的网页，如果不使用Ajax，更新网页内容需要重载整个页面）。

3. web的运作原理：一次HTTP请求对应一个页面，如果要让用户停留在当前页面，同时发送http请求，就必须用javaScript发送这个请求，接收到数据后，再用JavaScript更新页面，这样，用户就感觉自己仍然停留在当前页面但数据却可以不断更新。

4. Ajax包含的技术：

（1）使用css和xhtml来表示；

（2）使用DOM模型来交互和动态显示；

（3）使用XMLHttpRequest来和服务器进行异步通信；

（4）使用javaScript来绑定和调用。

5. 完全客户端的技术，而XMLHttpRequest正是处理了服务端与客户端通信的问题。

## 23. Ajax原理和XmlHttpRequest对象

通过XMLHttpRequest对象来向服务器发起异步请求，从服务端获得数据后使用javaScript来操作DOM更新页面。

javascript可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

1. XmlHttpRequest属性：

* onreadystatechange  每次状态改变所触发事件的事件处理程序。
* responseText     从服务器进程返回数据的字符串形式。
* responseXML    从服务器进程返回的DOM兼容的文档数据对象。
* status           从服务器返回的数字代码，比如常见的404（未找到）和200（已就绪）
* statusText       伴随状态码的字符串信息
* readyState       对象状态值

常见readyState状态码：

0 (未初始化) 对象已建立，但是尚未初始化（尚未调用open方法）

1 (初始化) 对象已建立，尚未调用send方法

2 (发送数据) send方法已调用，但是当前的状态及http头未知

3 (数据传送中) 已接收部分数据，因为响应及http头不全，这时通过responseBody和responseText获取部分数据会出现错误，

4 (完成) 数据接收完毕,此时可以通过通过responseXml和responseText获取完整的回应数据

## \*\*  24. Ajax浏览器差异（兼容）处理

差异主要体现在IE和其它浏览器之间：

function CreateXmlHttp() {  
  
 //非IE浏览器创建XmlHttpRequest对象  
 if (window.XmlHttpRequest) {  
 xmlhttp = new XmlHttpRequest();  
 }  
  
 //IE浏览器创建XmlHttpRequest对象  
 if (window.ActiveXObject) {  
 try {  
 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
 }  
 catch (e) {  
 try {  
 xmlhttp = new ActiveXObject("msxml2.XMLHTTP");  
 }  
 catch (ex) { }  
 }  
 }

return xmlhttp;  
}

## \*\* 25. ajax的优/缺点

<http://www.cnblogs.com/mingmingruyuedlut/archive/2011/10/18/2216553.html>

## \*\* 26. Cookie

[**http://blog.csdn.net/fangaoxin/article/details/6952954/**](http://blog.csdn.net/fangaoxin/article/details/6952954/)

1. Cookie通过在**客户端**记录信息确定用户身份，Session通过在**服务器**记录信息确定用户身份。

2. HTTP协议是无状态协议，一旦数据交换完毕，客户端与服务端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上**跟踪**会话。Cookie弥补了HTTP协议的无状态的不足，在Session出现之前，Cookie被大量使用。

3. 由于HTTP协议是一种无状态协议，服务器单从网络连接上无从知道客户机身份。怎么办呢？就给客户端颁发一个通行证吧，每人一个，无论谁都必须携带自己的通行证。这样服务器就能从通行证上确认客户身份。这就是cookie的工作原理。

4. Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

5. 在浏览器输入 javascript:alert(document.cookie) 就可以查看一个网站包含的cookie，需要有网。

6. Cookie功能需要浏览器的支持，如果浏览器不支持（大部分的手机浏览器），或者禁用了，Cookie就会失效，不同浏览器采用不同的方式保存Cookie，IE会保存在：C:\Documents and Settings\你的用户名\Cookies 目录下，以文件形式保存。

7. Cookie的不可跨域名性

Biadu和Google都会给浏览器颁发Cookie，那么浏览器访问Baidu时会将Google的Cookie也提交吗？**不会的。**

Cookie具有不可跨域名性，根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。浏览器区分的依据是域名。二者不能互相操作彼此的Cookie。

8. 单个Cookie的客户端限制是3k，即一个站点在客户端存放的Cookie不能超过 3 k；将登陆信息等重要信息存放在session，其它信息如果需要保留，可以存放在Cookie中。

<http://blog.csdn.net/axin66ok/article/details/6175522>

# day7 2017-09-29

## 27. 文件上传，下载

## 28. 验证码

# day8 2017-09-30

## 29. EL表达式

[**http://www.jb51.net/article/105314.htm**](http://www.jb51.net/article/105314.htm)

## 30. 分页

**day9 2017-10-09(国庆收假)**

自己完成-留言管理系统

## 31. js getElementById().value 与innerHTML的区别

表单用value，插入字段用innerHTML。有value属性的标签才能使用 .value。

# day10 2017-10-10

自己完成-留言管理系统

# day11 2017-10-11

自己完成-留言管理系统

# day12 2017-10-12

自己完成-留言管理系统

# day13 2017-10-13

## 1. EL表达式

[**http://www.jb51.net/article/20042.htm**](http://www.jb51.net/article/20042.htm)

**<%@page isELIgnored="false"%>**

Servlet容器内部会直接调用org.apache.jspager.runtime.PageContextImpl.proprietaryEvaluate(…)方法解析 el 表达式（内部为 ELContext 对象）

1它是一种简单的语言，基于可用的命名空间（PageContext 属性）、嵌套属性和对集合、操作符（算术型、关系型和逻辑型）的访问符、映射到 Java 类中静态方法的可扩展函数以及一组隐式对象。EL 提供了在 JSP 脚本编制元素范围外使用运行时表达式的功能。脚本编制元素是指页面中能够用于在 JSP 文件中嵌入 Java 代码的元素。它们通常用于对象操作以及执行那些影响所生成内容的计算。JSP 2.0 将 EL 表达式添加为一种脚本编制元素。

2 语法结构 ${expression}

3 [] 与 . 运算符

当要存取的属性名称中包含一些特殊字符，如 . 、- 或 ？ 等并非字母或数字的符号，就一定要使用 []。如：

${user.nick-name} 应改为 ${user[“nick-name”]}。

如果要动态取值，可以用 [] 来做，而 . 无法做到动态取值，如：

${sessionScope.user[data]} 中data是一个变量。

4 ${user}: 取出某一范围中名称为user的变量，因为并没有指定user变量的范围，所以它会依次从Page、Request、Session、Application范围（由内而外）中查找。如果中途找到，直接回传并中断查找，如果所有范围中都没有则回传null（打印时为””）。

5 el 中的有效表达式

文字：可以用位子表示特定数据类型的值，如true、false、24、-13、23.3、-1.8e4、’name’、”user”、null 等。

操作符：

算数：+、-、\*、/、%、div、mod、

逻辑：and、&&、or、||、!、not、

关系：==、eq、!=、ne、<=、le、>=、ge、

空：、、

条件：A ? B : C

5 隐式对象

JSP页面上下文：pageContext

请求参数的值映射到单个字符串（ServletRequest.getParameter(name)）：param

所有请求参数的值映射到一个字符串数组（ServletRequset）：paramValues

…

header

headerValues

cookie

initParam

…

pageScope

requestScope

sessionScope

applicationScope

6 ${pageContext} 将会输出 pageContext （调用其 toString）方法

注意 <%@ page isELIgnored="true" %> 表示是否禁用EL语言,TRUE表示禁止.FALSE表示不禁止.JSP2.0中默认的启用EL语言。

## 2. JSTL c

<http://blog.csdn.net/leolu007/article/details/45029627>

<%@ taglib prefix=”c” uri=<http://java.sun.com/jsp/jstl/core>” %>

1 <c:out value=”value” [escapeXml=”{true|false}”] [default=”defaultValue”]/>

2 <c:set value="value" var="varName" [scope="{page|request|session|application}"]/>

3 <c:remove var="varName" [scope="{page|request|session|application}"]/>

4 <c:catch [var=”varName”]>

可能抛异常的部分

<c:catch>

5 <c:if test=”booleanexpressopn” var=”var” [scope=”page|request|session|application”] />

6 <c:choose>

<c:when test="${product.onSale}">

${product.salesPrice} On sale!

</c:when>

<c:otherwise>

${product.price}

</c:otherwise>

</c:choose>

7 <c:forEach items="collection" [var="var"] [varStatus="varStatus"] [begin="startIndex"] [end="stopIndex"] [step="increment"]>

JSP elements

</c:forEach>

8 <c:forTokens items="stringFoTokens" delims="delimmmmiters" [var="varName"]

            [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>

 ...body内容....

</c:forTokens>

9 <c:import url="url" [context="context"]

[var="varName"] [scope="{page|request|session|application}"] [charEncoding="charEncoding"]>

内容

</c:import>

10 <c:url value="value" [context="context"] [var="varName"]

[scope="{page|request|session|application}"] />

语法2：本体内容代表查询字符串(Query String)参数

<c:url value="value" [context="context"] [var="varName"]

[scope="{page|request|session|application}"] >

<c:param> 标签

</c:url>

11 <c:redirect url="url" [context="context"] />

语法2：本体内容代表查询字符串(Query String)参数

<c:redirect url="url" [context="context"] >

<c:param>

</c:redirect >

## 2. JSTL fn

<%@ taglib prefix=”fn” uri=<http://java.sun.com/jsp/jstl/functions>%>

1. <fn:contains(str,substr)> boolean
2. containsIfnoreCase(str,substr) boolean
3. endsWith(str,suffix) boolean
4. escapeXml(str) String
5. indexOf(str,substr) int
6. join(arry,separator) String 将数组用指定连接符拼接并返回
7. length(array|Collection) int
8. replace(str,before,after) String
9. split(str,separator) Array
10. startsWith(str,prefix) boolean
11. subString(str,begin,end) String
12. subStringAfter(str,substr) String
13. subStringBefore(str,substr) String
14. to[Lower | Upper]case(str) String
15. trim(str) String 去除首尾空格

# day14 2017-10-16

## 1.jQuery

[**http://jquery.cuishifeng.cn/**](http://jquery.cuishifeng.cn/)

1 根据不同选择器找到对象，类、id，所有的div

* 类选择器：var es = $(‘button.className,#idName,div’)[0]; // dom对象

var ess = $(es); // jQuery对象

Object.prototype.toString.call(ess); // 精确输出对象类型

* id选择器：$(‘#idName’)
* 所有的div：$(‘div’)
* 所有的有id属性的input标签：$(‘input[id]’)
* 有name属性的input标签，且name为nm：$(‘input[name=”nm”]’)
* ul 下的指定列、偶数列、奇数列、3倍数加一的列（4,7,10…）：$(‘ul li-nth-child([2 | even | odd | 3n+1])’)
* ul 下的第一（最后）行：$(‘ul li-[first | last]-child’)
* $(select option:selected)

2 jQuery转为HTML\*Element：$(‘div’).get(0);

## 2.jQuery attr（jQuery 1.0）和prop（jQuery 1.6）

<http://www.jb51.net/article/89038.htm> 尽量使用prop()函数

1操作对象不同：attribute表示HTML文档节点的属性，property表示JS对象的属性。

attr方法依赖于Element的getAttribute()和setAttribute()方法，prop函数依赖的则是JS中原生的对象属性获取和设置方式。

2属性值类型不同：attr函数操作的是文档节点的属性，因此设置的属性值只能是字符串类型，如果不是也会调用其toString方法；prop函数设置的属性值可以包括数组对象在内的任意类型。

此外，对于表单元素的“checked”、“selected”、“disabled”等属性，在jQuery 1.6之前，attr()获取这些属性的返回值为Boolean类型：如果被选中(或禁用)就返回true，否则返回false。

但是从1.6开始，使用attr()获取这些属性的返回值为String类型，如果被选中(或禁用)就返回“checked”、“selected”或“disabled”，否则(即元素节点没有该属性)返回undefined。并且，在某些版本中，这些属性值表示文档加载时的初始状态值，即使之后更改了这些元素的选中(或禁用)状态，对应的属性值也不会发生改变。

# day15 2017-10-17

## jQuery

<http://jquery.cuishifeng.cn/>

添加类：$(…).addClass()

判断是否有类：$(…).hasClass()

移除类：$(…).removeClass()

## 2. jQuery attr prop css

1. w3c对于属性的定义包括：**标准属性**（id，class，style…），**语言属性**（lang（语言类型 en），dir（文本方向ltr|rtl）），元素**固有属性**（a的href，input的type等），不存在的属性叫**新增属性**。
2. attr( )：可以设置元素的属性值（固有属性，新增属性），可以获取元素本来就有的属性以及新增属性。如果要获取的属性没有设置，那么获取到的结果是 undefined；
3. prop( )：可以设置元素的属性值（固有属性，新增属性），可以获取元素本来就有的属性值，**如果是额外设置的其他属性，则无法通过prop（）获取。**
4. 对于**固有的HTMl属性**，如果在元素中没有设置（html中没有指定），那么 attr 获取的结果是undefined 而prop（js对象的属性）可以获取那个属性的值。
5. css（）设置和获取样式属性（即style中设置的值）。

## 3. json

## 4. json jsonp 区别

[**http://www.cnblogs.com/iovec/p/5312464.html**](http://www.cnblogs.com/iovec/p/5312464.html)

1. 同源策略（Same-Origin Policy）：它是由Netscape提出的一个著名的安全策略。现在所有支持JavaScript 的浏览器都会使用这个策略。所谓同源，就是必须协议、域名、端口都一致的，才叫做同源。<http://blog.csdn.net/xiaoxian8023/article/details/27817861>

如：<http://www.12306.cn> 和 <https://www.12306.cn> 协议不一致，不是同源。

直接访问浏览器会提示Access-Control-Allow-Origin，只要在服务端设置响应头即可，但存在安全隐患，因为存在通配符“\*”：response.setHeader(“Access-Control\_Allow\_Origin”,”\*”)。

1. jQuery的$.ajax支持跨域访问，只需将dataType设置为jsonp（JSON with padding），是json的一种“使用模式”， 可以让网页从别的网域获取资料。jsonp是采用的js的回调机制来实现的。使用方式也很简单，简略方式如下：

$.getJSON([http://\*\*\*\*.\*\*/\*\*/&callback=?,function(json){...})](http://****.**/**/&callback=?,function(json)%7b...%7d))；服务端需要获取callback的值并以如下方式返回数据：callback+”(“+jsondata+”)”; 格式是回调函数名+(json数组)。

1. jsonp就是简单的函数（javaScript）调用，数据被包裹在了参数中。
2. json是理想的数据交换格式，但没办法跨域直接获取，于是就将json包裹(padding)在一个合法的js语句中作为js文件传过去。这就是json和jsonp的区别，json是想要的东西，jsonp是达到这个目的而普遍采用的一种方法，当然最终获得和处理的还是json。所以说json是目的，jsonp只是手段。json总会用到，而jsonp只有在跨域获取数据才会用到。

# day16 2017-10-18

junit 4 需要两个jar： junit-4.12.jar hamcrest-all-1.3.jar

## 1. MyBatis

官方文档：<http://www.mybatis.org/mybatis-3/zh/getting-started.html>

1. 步骤

1 添加mybatis jar包

2 配置mybatis-config.xml文件：mybatis框架工作的设置和属性，可以使用 ${} 来直接引用properties标签中指定属性文件中的属性。

3 配置bean 对应的xml配置文件

1. mybatis中 # 和 $ 的区别

* #：是将传入的值当字符串处理，使用#可以很大程度上防止SQL注入（更常用），JDBC解析为一个占位符 ? 等。
* $：是将传入的数据直接显示，生成语句，在order by中就需要使用 $，仅为纯粹的字符串替换。

#{id}：创建一个预处理语句，相当于JDBC的？。

#与$的区别最大在于： #{} 传入值时，sql解析时，参数是带引号的，而${}传入值，sql解析时，参数是不带引号的。

\* 注意：用 ’ ’ 包住 ${} ，同时参数名为value

SELECT \* FROM user WHERE name LIKE '%${value}%'

#{beanFiledName} beanFiledName 为实体类（如User）的字段名。（insert操作，且parameterType为bean）

INSERT INTO user(name,password,state,age) VALUES(#{userName},#{userPassword},#{userState},#{userAge})

# {mapKeyName} mapKeyName 为传的数据（Map）的键名。（insert操作，且parameterType为map/hashmap）

INSERT INTO user(name,password,state,age) VALUES(#{n},#{p},#{s},#{a})

## 2. .properties

<http://blog.csdn.net/easy_gemini/article/details/8229811>

配置文件，使具有通用性。各种语言都有自己的配置文件类型，如Python的.ini，java的.properties 等。

格式语法：

* #为注释标记
* 以key和value键-值得方式保存数据，用=连接

如：

# 以下为数据库连接信息

driver=com.mysql.jdbc.driver

# 以下为…

name=tom

=两端不要有空格，需要用空格时使用\u0020（中文空格为\u3000）。

java.util包中的Properties类可处理.properties文件，getPropertiy(key)，setPropertiy(key,value)，load，clear等。

## 3. SQL中=、in、like区别

三者都可以用来进行数据匹配 。但三者并不相同。

* =： 等号是用来查找与单个值匹配的所有数据；

等号确切知道所要查找的内容，且为单一值时，可以使用等号运算符来进行数据比较。等号运算符中可以使用字符串、日期或数字。

* IN： IN 是用来查找与多个值匹配的所有数据；

当确切知道所要查找的内容，且为多个值时，可以使用 IN 子句来进行数据比较。IN 子句中也可以使用数字、字符串或日期。

* LIKE： 而LIKE用来查找与一个模式匹配的所有数据。

当无法确切知道所要查找的值，而是知道所要查找的数据符合的模式时，可以使用LIKE 子句进行匹配。一般来说，LIKE 子句仅在字符串类型时使用。

通配符：

%：替代0个或多个字符

\_：仅替代一个字符

[abcd]：字符列中的任何单一字符

^(或!)：取相反

## 4. MyBatis Mapper接口

在mybatis框架中，写dao层的mapper接口时，是不可以进行方法的重载的

# day17 2017-10-19

数据库三大范式：<http://www.cnblogs.com/linjiqin/archive/2012/04/01/2428695.html>

1 确保每列保持原子性

2 确保表中的每列都和主键相关 （复合主键）<http://blog.csdn.net/wangyuchun_799/article/details/49329649>

3 确保每列和主键直接相关，而不是间接相关

## 1 数据库表间的三种对应（映射）关系：一对一，一对多/多对一，多对多

[**http://www.cnblogs.com/abeam/p/7406285.html**](http://www.cnblogs.com/abeam/p/7406285.html)

* 一对一：夫妻关系，数据表中表现为外键关联，主键关联。A表中的一条记录对应B表中的一条记录。
* 一对多：存在最普遍的映射关系，如球队与球员。A表中的一条记录对应B表的多条记录，且A的主键作为B的外键。
* 多对多：一般需要张中间表，将两张表进行关联。如学生选课：实体表为学生表和课程表，关系表为选修表。

## 2 myBatis两种方式完成一对一和一对多查询

mybatis官网文档：<http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html>

数据库中表：



实体类（必须包含空参一个空参构造函数，数据库字段对应成员变量的getter和setter）：

Student.java：

int id; String name; Teacher tutor;

Teacher.java：

int id; String name; List<Student> students;

1. xml配置

需要在mybatis配置文件的mappers中注册XXXMapper.xml（resource），可将实体类在typeAliases中声明别名以简化xml配置。

XXXMapper.xml的数据库查询语句中使用 \* 可查询出resultMap中所有需要的column，不使用 \* 则需要将resultMap中用到的column都在select中声明：

select \* from …

select s.student\_id,s.student\_name,s.tutor\_id,t.tercher\_id,t.teacher\_name …

配置在resultMap标签中：

1. property为实体类对应的成员变量名
2. column为数据库对应字段的字段名
3. id为主键专用。

* 一对一

association为关联对象的映射，javaType为对应的类名（实体类或其他对应java类）



* 一对多

collection标签为关联集合的映射，ofType为集合中元素对应的类名。



调用示例：



1. java注解配置

创建对应的接口，XXXMapper.java，并在mybatis配置文件的mappers标签中注册（clas

）。

@Select：书写sql语句

@Results：相当于xml中的resultMap标签

property：实体类成员变量名

column：数据库对应字段名

id：true/false，是否为主键

one：一对一关系，用 @One 注解：指向另一个方法调用（select属性给定方法名）来获取值，fetchType指定获取方式，LAZY、EAGER、DEFAULT。

示例：



使用示例：



# day18 2017-10-20

## 1 动态SQL（myBatis）

1. if语句

test中放置判断语句，从前往后逐个判断。



示例解析：

1如果传入的Teacher对象的id属性不等于0，sql语句拼接AND teacher\_id = #{id}，这里的id为Teacher对象传入的值（虽然mybatis会通过getXXX将结果自动装箱，但由于Teacher的id为int，默认值为0，所以不能用null判断），之后再判断Teacher对象的name属性（String类型）是否为null，不是则sql拼接AND teacher\_name = #{name}。

2 若id=0或name为null，就不拼接语句，即默认查询所有。

3 传参时的Teacher传null，或直接调用无参方法时，不是调用其bean的无参构造器创建对象后传递，处理方法也不同…结论是动态sql中用到的参数都用null替代。

1. choose语句

相当于java的switch（choose）、case（when）、default（otherwise）





1. trim

对包含的内容加上prefix（前缀），suffix（后缀），prefixOverrides（忽略首部的某些内容），suffixOverrides（忽略尾部的某些内容）。



解析后SQL：

SELECT \* FROM teacher **WHERE teacher\_id = 1 AND teacher\_age BETWEEN 1 AND 150**

1. where

简化SQL的where判断，能智能处理and和or。

1 where元素在写入where元素的地方拼接一个WHERE，如果where中的条件都不满足会查出所有（丢弃WHERE条件）。

2 如果AND/OR开头会被智能的舍弃

3 不用考虑空格的问题



SELECT \* FROM teacher **WHERE teacher\_id = ? AND teacher\_age > 10**

1. set

主要用于更新操作。

功能类似于where元素，在sql语句前拼接SET，如果语句以逗号结束的话把逗号丢弃。





UPDATE teacher **SET** teacher\_name = ?, teacher\_age = ?, teacher\_level = ? **WHERE** teacher\_id = ?

SQL语法中有些自动如果值为null的话会出错，有了set元素我们就可以只更新那些修改了的字段。

1. foreach

主要用在构建IN条件。

item：迭代到的元素

index：迭代时的下标（规则同java中Collection下标）

collection：list、array、或map（与paramerType有关，可以不指定，让myBatis自己推断）

open：表示SQl语句以什么字符串作为开始标记

separator：每一次迭代到的元素拼接到SQL语句中时用什么字符串作为分隔符

colse：表示SQl语句以什么字符串作为结束标记





## 2 缓存机制-cache

<http://blog.csdn.net/luanlouis/article/details/41390801>

1. 一级缓存：基于PerpertualCache的HashMap本地缓存，作用域为session，当session被flush或close时，session中的所有Cache将被清空。

深入理解实现：<http://blog.csdn.net/luanlouis/article/details/41280959>

1. 二级缓存：和一级缓存机制一样，基于PerpetualCache的HashMap本地缓存，作用域为Mapper（NameSpace）.

深入理解实现：<http://blog.csdn.net/luanlouis/article/details/41408341>

1. <http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html#cache>

# day19 2017-10-21

## 1 mybatis的resultMap两张表字段名相同时的特殊处理

进行一次查询时，如果查询结果依赖的多张表拥有相同的字段名，此时需要进行特殊处理，否则mybatis无法正常映射：

1. 在SQL语句中对重复的字段取别名（AS），使之不重复。
2. resultMap中column使用SQL语句中取的别名



# Day20 2017-10-23

考试

# Day21 2017-10-24

讲解试题，机试题。

# Day22 2017-10-25

## Maven

<http://www.yiibai.com/maven/>

POM（项目对象模型）：是maven中工作的基本单位，是一个xml文件（pom.xml），包含pom.xml的项目是使用maven来构建的，它用来包含各种配置信息。

创建项目前，需决定项目组groupId，名字（artifactId）和版本（version），这些属性是项目仓库的唯一标识。

<http://blog.csdn.net/zhuxinhua/article/details/5788546>

<build/>：构建设置

<dependencies>依赖关系

<dependency>

<groupId><groupId/>

<artifactId><artifactId/>

<version><version/>

</ dependency >

<dependencies/>

## Spring MVC

Spring文档：<https://docs.spring.io/spring/docs/>

SpringMVC中文教程：<http://www.yiibai.com/spring_mvc/>

第二章 Spring MVC入门 —— 跟开涛学SpringMVC： <http://jinnianshilongnian.iteye.com/blog/1594806>

### Spring请求处理流程和架构

1. 关键对象：

前端控制器：DispatcherServlet

请求-处理器映射：HandlerMapping

处理器适配器：HandlerAdapter

视图解析器：ViewResolve

1. 请求处理流程





流程：用户发送请求 ——> 前端控制器（DispatcherServlet）—*根据请求信息选择页面控制器并委托处理*—> 页面控制器/处理器 —*将请求参数包装为****命令对象****，进行验证，传递给****业务对象****处理（获得****模型数据****）*—> 页面控制器/处理器 —*直接返回ModelAndView*—>前端控制器 —*根据回传数据装载页面,渲染****视图***—> 用户收到回馈

1. 架构



DispatcherServlet的核心代码在doDispatch方法中，doDispatch获取页面处理器，然后根据页面处理器获取对应的HandlerAdapter（以支持多种类型的请求），最后由HandlerAdapter调用页面处理器的方法。

HandlerMapping：找到保存url请求和处理函数间的映射关系。（如RequestMappingHandlerMapping，扫描当前请求和被@ReauestMapping注解的Controller中的方法，将方法和url关联）

HandlerAdapter：处理请求，适配器模式。（如RequestMappingHandlerAdapter，根据HandlerMapping找到对应的方法，并调用）

1. 解析用户请求(**图中第2点**) **——>**前端控制器收到请求后交给其它解析器处理，作为统一访问点，进行全局流程控制。通过HandlerMapping找到url和method间的映射关系（利于添加新的映射关系），并包装为Handler**Execution**Chain、Handler以及多个HandlerInterceptor。
2. 处理用户请求(**图中第3点**)**——>**前端控制器成功解析用户请求，将Handler交给处理适配器（HandlerAdapter），适配器将根据HandlerMapping获得的映射结果调用页面处理器（利于支持多种类型的处理器）的相应方法（**图中第4点**），完成处理；并返回一个ModelAndView对象（包含处理结果、逻辑视图名）。
3. 响应用户请求（**图中第5/6点**）**——>**ViewResolver将把逻辑视图解析为具体的View； **——>** View将传入的Model数据渲染到视图上（这里包装后的Model实际上是一个Map（HashMap）数据结构）。
4. 将控制权交回给DispatcherServlet，流程结束。

上述流程的1~4在DispatcherServlet的doDispatch中完成。5、6调用processDispatchResult方法中完成。

### Idea配置

<http://blog.csdn.net/zy0528smile/article/details/52922378>

<http://www.cnblogs.com/Sinte-Beuve/p/5730553.html>

### SpringMVC使用步骤

1. 在web.xml中注册DispatcherServlet并指定拦截所有请求（或指定请求）；
2. 指定配置文件：不指定的话默认加载“(servlet-name)-servlet.xml”，如：

…<servlet-name>MySpring<servlet-name/>，则会寻找MySpring-servlet.xml的配置文件，找不到就会出错（file doesn’t exists）。

可按如下方式配置：

<!--前端控制器DispatcherServlet的配置-->  
<**servlet**>  
 <**servlet-name**>springMVC</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 <**init-param**>  
 <!--Sources标注的文件夹下需要新建一个spring文件夹-->  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring/spring-mvc.xml</**param-value**>  
 </**init-param**>  
 <**load-on-startup**>1</**load-on-startup**>  
 <**async-supported**>true</**async-supported**>  
</**servlet**>  
  
<**servlet-mapping**>  
 <**servlet-name**>springMVC</**servlet-name**>  
 <**url-pattern**>/</**url-pattern**>  
</**servlet-mapping**>

1. 在spring-mvc.xml中配置视图处理器和其它杂项：

选择注解配置或xml配置；

视图处理器配置于<bean/>标签中；

配置自动扫描位置；

配置静态资源映射；

etc.

1. 开发处理器（Handler）/页面控制器

页面处理器有两种实现方式：参看[Spring MVC注解方式和实现接口方式实现页面控制器的不同](#_Spring_MVC注解方式和实现接口方式实现页面控制器的不同)

接口方式请求流程示例：



1. Java代码中引用bena
2. String[] cs = {  
    "/spring/spring-mvc.xml",  
   // "/spring/spring-bean.xml",  
    };  
    ApplicationContext context = new ClassPathXmlApplicationContext(cs);  
    UserService userService = (UserService) context.getBean("userService");  
    UserDao dao = userService.getUserDao();  
    *p*.accept(dao.queryUserById(2));

关键对象为ApplicationContext。

## Intellij IDEA 的Facets和Artifacts

<http://blog.csdn.net/gongsunjinqian/article/details/53018172>

## Spring MVC 对静态文件的引用

<http://www.cnblogs.com/dflmg/p/6393416.html>

如果在web.xml中配置DispatcherServlet的url-pattern为“/”，那么spring将捕获所有请求，这会使web应用在请求一些本地的静态资源时被spring以相同的方式处理，即在spring-mvc.xml中寻找其页面控制器，找不到就会产生404错误。有三种方法可以解决这种情况：

* 方法一：采用<mvc:default-servlet-handler/>

在springMVC-servlet.xml中配置<mvc:default-servlet-handler />后，会在Spring MVC上下文中定义一个org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler，它会像一个检查员，对进入DispatcherServlet的URL进行筛查，如果发现是静态资源的请求，就将该请求转由Web应用服务器默认的Servlet处理，如果不是静态资源的请求，才由DispatcherServlet继续处理。

* 方法二：采用<mvc:resources/>

<mvc:resources location="/,classpath:/META-INF/publicResources/" mapping="/resources/\*\*"/>

由Spring MVC框架自己处理静态资源，并添加一些有用的附加值功能。首先，<mvc:resources />允许静态资源放在任何地方，其次，<mvc:resources />依据当前著名的Page Speed、YSlow等浏览器优化原则对静态资源提供优化。

<!--静态资源映射-->  
<!--本项目把静态资源放在了WEB-INF的statics目录下，资源映射如下-->  
<**mvc:resources** mapping="/css/\*\*" location="/WEB-INF/statics/css/"/>  
<**mvc:resources** mapping="/js/\*\*" location="/WEB-INF/statics/js/"/>  
<**mvc:resources** mapping="/image/\*\*" location="/WEB-INF/statics/image/"/>

* 方法三：激活Tomcat的default servlet来处理静态文件

类似<mvc:default-servlet-handler/>方式。

* **<servlet-mapping>**
* **<servlet-name>**default**</servlet-name>**
* **<url-pattern>**\*.jpg**</url-pattern>**
* **</servlet-mapping>**
* **<servlet-mapping>**
* **<servlet-name>**default**</servlet-name>**
* **<url-pattern>**\*.js**</url-pattern>**
* **</servlet-mapping>**

这种方式要写在DispatcherServlet前面，让defaultServlet先拦截，请求就不会进入spring。

## Web.xml中的classpath

<http://blog.csdn.net/gjswxhb/article/details/21330225>

classpath代表 /WEB-INF/classes/这个路径，将一个java web工程发布为war（web Application Archive），解压后查看就能看到。

使用场景，SSM架构中，配置Spring的上下文环境：

<servlet>

<servlet-name>springMVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<!--Sources标注的文件夹下需要新建一个spring文件夹-->

<param-name>contextConfigLocation</param-name>

<param-value>**classpath:**spring/spring-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<async-supported>true</async-supported>

</servlet>

classpath:applicationContext.xml

也可以使用  /WEB-INF /classes/ applicationContext.xml  代替

classpath: 只会到class路径中查找文件

classpath\*: 不仅包含class路径，还包括jar文件中的class路径。

# Day23 2017-10-26

## XSD（xml schema definition）

1．  定义一个Xml文档中都有什么元素

2．  定义一个Xml文档中都会有什么属性

3．  定义某个节点的都有什么样的子节点，可以有多少个子节点，子节点出现的顺序

4．  定义元素或者属性的数据类型

5．  定义元素或者属性的默认值或者固定值

## Spring MVC注解方式和实现接口方式实现页面控制器的不同

参考文章：<http://blog.csdn.net/eson_15/article/details/51699103>

1. 注解方式（spring2.5之后支持）：

注解方式在配置文件（spring-mvc.xml）要进行如下声明：

<!--注解方式的映射器-->  
<**bean** class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>  
<!--注解方式的适配器-->  
<**bean** class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>

这种方式需将**页面处理器**对应的java类用@Controller注解，并在**具体方法**上用@ReuestMapping注解以指明该动作的访问url。

方法可返回ModelAndView对象将目标视图和数据包装其中，或返回String类型，仅仅指明目标视图名称。

使用注解方式时要**注意**：注解适配器和注解映射器必须配对使用

在SpringMVC中可以更简单的在配置文件中声明，<mvc:annotation-driven/>，此时注解适配器和映射器可以省略。

<**mvc:annotation-driven**/>

1. 实现接口方式

1 其中适配器有两种（可以同时使用），对应两种不同的接口实现；

<!--实现接口方式的适配器（HandlerAdapter） 表示所有实现了org.springframework.web.servlet.mvc.Controller-->  
<**bean** class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>  
<!--实现接口方式的适配器（HandlerAdapter） 表示所有实现了org.springframework.web.HttpRequestHandler-->  
<**bean** class="org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter"/>

接口方式要求一个页面控制器**只能有一个动作**（处理器），因为实现接口，其动作只能在那个方法中实现。（区别于注解方式的：一个页面控制器多个处理器）

2 映射器也有两种，拥有不同的灵活性：

* url和bean名字（bean将名字和页面控制器对应）对应，

<!--实现接口方式的映射器（HandlerMapping） 无需配置，springmvc可以默认启动-->  
<!-- HandlerMapping 表示将请求的URL和Bean名字（name）映射，如URL为 “上下文/hello”，则Spring配置文件必须有一个名字为“/hello”的Bean，上下文默认忽略。-->  
<**bean** class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

需额外配置<bean name=”” class=””/>将bean的名字和页面控制器对应，访问时就可通过name访问控制器。

* 简单映射器（指定url和页面控制器的对应关系）

<**bean** id="item1" class="com.duan.springweb.controller.Item1Controller"/>  
<**bean** id="item2" class="com.duan.springweb.controller.Item2Controller"/>  
<**bean** class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
 <**property** name="mappings">  
 <**props**>  
 <**prop** key="/item\_1">item1</**prop**>  
 <**prop** key="/item\_2">item2</**prop**>  
 </**props**>  
 </**property**>  
</**bean**>

将每一个页面控制器声明为bean，在SimpleUrlHandlerMapping中将访问url和bean对应；

两个映射器也可以一起使用，互不冲突，匹配到哪个就用哪个。

## <context:component-scan/> 自动扫描页面控制器

如果页面控制器很多的话，就需要配置很多的bean，因此spring-mvc提供了扫描包的方式进行自动配置。

<!--自动扫描该包，使SpringMVC认为包下用了@controller 注解的类是控制器 -->  
<**context:component-scan** base-package="com.duan.springweb.controller">  
 <**context:include-filter** type="annotation" expression="org.springframework.stereotype.Controller"/>  
</**context:component-scan**>

这样SpringMVC就能自动扫描@Controller注解的类（或者实现接口的类），该类就被认为是页面控制器，类中被@RequestMapping注解的方法则被认为具体的处理器（动作）；

## 视图名称的解析

<!-- 对模型视图名称的解析，即在模型视图名称添加前后缀(如果最后一个还是表示文件夹,则最后的斜杠不要漏了) 使用JSP  
ViewResolver 用于支持Servlet、JSP视图解析-->  
<**bean** id="defaultViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <**property** name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>  
 <**property** name="prefix" value="/WEB-INF/views/"/><!--设置JSP文件的目录位置-->  
 <**property** name="suffix" value=".jsp"/>  
</**bean**>

在页面控制器的相应处理方法中直接返回页面名称，或装入ModelAndView中返回，对视图名称的解析将通过上面的方式。（也可以配置其它的方式）

## SpringMVC提供的乱码解决过滤器

<!--表示规定字符编码为UTF-8-->  
<**filter**>  
 <**filter-name**>CharacterEncodingFilter</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.CharacterEncodingFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>encoding</**param-name**>  
 <**param-value**>utf-8</**param-value**>  
 </**init-param**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>CharacterEncodingFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

## SpringMVC主要注解

<http://jinnianshilongnian.iteye.com/blog/1594806>

## Spring

Spring教程：<http://www.yiibai.com/spring/load-multiple-spring-bean-configuration-file.html>

ApplicationContext有三个实现类：

ClassPathXmlApplicationContext：从classpath检索配置文件并装载，创建bean实例。

FileSystemXmlApplicationContext：从classpath以外的地方检索配置文件并装载，创建实例。

XmlWebApplicationContext：可以在web.xml中配置

## 依赖注入（DI）：

1. Setter注入

创建bean对应的类（必须有一个空参可访问的构造器），为类中需要注入的成员变量创建setter方法；

在配置文件中创建注入配置；

<**bean** id="user" class="com.duan.springweb.entity.User">  
 <**property** name="name" value="张二"/>  
 <**property** name="password" value="张二aabc"/>  
 <**property** name="state" value="0"/>  
 <**property** name="age" value="23"/>  
</**bean**>

其中property的name对应成员变量的名字（调用其setter方法），value为要注入的值（setter传入的实参）。

1. 构造器注入：

为Bean对应的类创建构造器，定义好传入的参数；

在配置文件中配置：

<**bean** id="user2" class="com.duan.springweb.entity.User">  
 <**constructor-arg** type="java.lang.String">  
 <**value**>王五</**value**>  
 </**constructor-arg**>  
 <**constructor-arg** type="java.lang.String">  
 <**value**>王五0001</**value**>  
 </**constructor-arg**>  
 <**constructor-arg** type="int">  
 <**value**>2</**value**>  
 </**constructor-arg**>  
 <**constructor-arg** type="int">  
 <**value**>23</**value**>  
 </**constructor-arg**>  
</**bean**>

<constructor-arg/>在构造器参数有歧义时需指定type属性以进行区分，否则会注入失败。

构造器歧义示例：

User(String name, int state, String password, int age) {

User(String name, String password, int state, int age) {

## 外部配置文件引用

<http://www.cnblogs.com/leiOOlei/archive/2013/12/25/3491042.html>

1 可使用<import/>标签引入其他的配置文件。

<**import** resource="spring-bean.xml"/>

2 创建ApplicationContext时传递用到的配置文件。

String[] cs = {  
 "/spring/spring-mvc.xml",  
 "/spring/spring-bean.xml",  
};  
ApplicationContext context = new ClassPathXmlApplicationContext(cs);

3 或者在一个配置文件中<import/>用到的所有配置文件，创建ApplicationContext只传入那个即可。

## Spring bean的引用

如果bean在不同的xml文件中，需要先将其所在的xml文件引入（import）;

<**import** resource="spring-bean.xml"/>  
<**bean** id="userDao" class="com.duan.springweb.dao.UserDaoImpl">  
 <**property** name="user" ref="user2"/>  
</**bean**>

在同一个文件中时，直接引用即可，也可放在独立的<ref local=””/>标签中（不作为<property/>标签的属性）。

注意：可以不显示的import，直接引用，但这需要在装载配置文件时装载所需的所有配置文件，即：

String[] cs = {  
 "/spring/spring-mvc.xml",  
 "/spring/spring-bean.xml",  
};  
ApplicationContext context = new ClassPathXmlApplicationContext(cs);

## Spring bean注入三种方式

1. 普通方法

<**bean** id="user" class="com.duan.springweb.entity.User">  
 <**property** name="name">  
 <**value**>"张二"</**value**>  
 </**property**>  
 <**property** name="password">  
 <**value**>"张二aabc"</**value**>  
 </**property**>  
 <**property** name="state">  
 <**value**>0</**value**>  
 </**property**>  
 <**property** name="age">  
 <**value**>23</**value**>  
 </**property**>  
</**bean**>

1. 快捷方法

<**bean** id="user" class="com.duan.springweb.entity.User">  
 <**property** name="name" value="张二"/>  
 <**property** name="password" value="张二aabc"/>  
 <**property** name="state" value="0"/>  
 <**property** name="age" value="23"/>  
</**bean**>

1. “p”方法
2. <**bean** id="user" class="com.duan.springweb.entity.User"  
    p:name="张二"  
    p:password="张二aabc"  
    p:state="0"  
    p:age="23"/>