**2017-09-13 javaEE培训开始**

第一个任务，用反射、注解和泛型封装JDBC的CRUD。

<https://github.com/DuanJiaNing/JDBCEncapsulation.git>

**day1 2017-09-22**

**1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”);

2. 使用new String(byte[],charset)方法重组字符串

3. 过滤器统一过滤

**2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

2. 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。

3. 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。

4. 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)

5. 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）。

**3. ? 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");

**day2 2017-09-23**

**4. 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。

3.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。

4. 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>

6. session 无法跨浏览器获取数据。即更换浏览器会开启新的会话。

**5. 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

4.ServletContext在服务器关闭Web应用时销毁。

**\*\* 6.MySQL 外键**

<http://www.cnblogs.com/zunpeng/p/3878459.html>

**\*\* 7. MySQL级联查询**

**day3 2017-09-25**

**8. 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。

3 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。

4 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。

5 过滤器的链式结构：

为一个Web应用部署多个过滤器，这些过滤器链组成一个过滤器链，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。



6 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。

7 url-pattern 与 dispatcher 是且的关系，只有同时满足两者，该Filter才会生效

**9.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。

2.语法：

<% %> ：java脚本

3. 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。

4. 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。

5. 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

6. 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）。

**10. 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：

（1）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。

（2）初始化

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。

（3）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）。

（4）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>

**day4 2017-09-26**

**11.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。

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);

2. page代表当前页面，即自动生成的Servlet。

3. 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。

**12.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）。

**13.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>

**day5 2017-09-27**

**14.<%@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。

2. <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

\*\*  **23. 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;  
}

**\*\* 24. ajax的优/缺点**

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

**\*\* 25. 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**

**26. 文件上传，下载**

**27. 验证码**

**day8 2017-09-30**

**28. EL表达式**

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

**29. 分页**

**day9 2017-10-09(国庆收假)**

自己完成-留言管理系统

**30. 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。

**day9 2017-10-10**

自己完成-留言管理系统

**day9 2017-10-11**

自己完成-留言管理系统

**day9 2017-10-12**

自己完成-留言管理系统

**day9 2017-10-13**

**1. EL表达式**

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

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 >