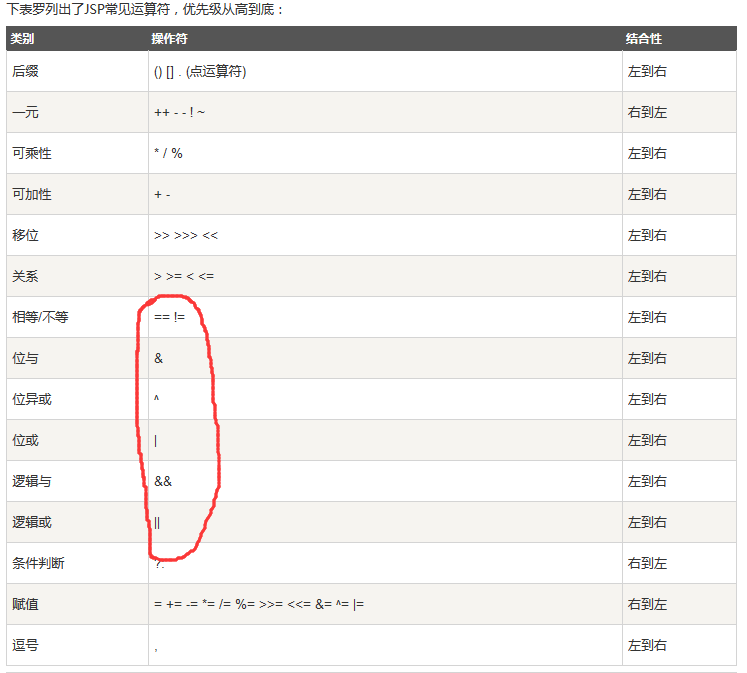
Jetty 是一个开源的servlet容器，它为基于Java的web容器。

**配置文件；**

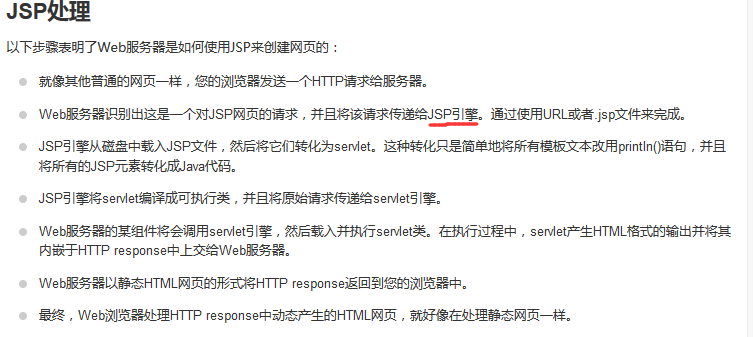
WEB-INF/classes/jdbc.properties



如果你使用的是txt等分散的组装环境，那么你想要调试代码的时候就最好不要使用异常处理先，直接看500 Server Error。通过自己删减注释代码来锁定出错的位置。

EL（Expression Language） 目的：为了使JSP写起来更加简单。表达式语言的灵感来自于 ECMAScript 和 XPath 表达式语言，它提供了在 JSP 中简化表达式的方法，让Jsp的代码更加简化。使用EL需要在jsp服务器里面提供支持。

JSP是一种Java servlet，主要用于实现Java web应用程序的用户界面部分。所以网页开发者们通过结合HTML代码、XHTML代码、XML元素以及嵌入JSP操作和命令来编写JSP。JSP（Java Server Page）必须有容器才能运行显示。

JSP实质上只是为HTML页面封装了对HTTP协议的Request对象和Rsponse对象而已。就比如说，获取上一个页面的数据、操作Session等。

JSP是在HTML里面写Java代码，而Servlet是在JAVA里面写HTML代码。Java applet 是一种当作单独文件跟网页一起发送的小程序，它通常用于在客户端运行，结果得到为用户进行运算或者根据用户互作用定位图形等服务。Servlet 的命名可以看出 sun 命名的特点，如 Applet 表示小应用程序；Scriptlet = Script + Applet，表示小脚本程序；同样 Servlet = Service + Applet，表示小服务程序。不过Servlet并不是想applet那样的独立程序，不存在main()，init()与destroy()决定了它的生存周期。

不能说javascript就是前台，有的javascript可以直接操作java代码的~比如dwr框架~

* JSP支持在静态HTML中嵌入3种标签:
  + *Scripting elements (scriptlets)*. （脚本标签）<%%>
  + *Directives*. （编译指示标签）<%@ %>
  + *Actions*. （动作或行为标签）<jsp:action\_name />
    - jsp:include --在页面被请求时包含进一个文件。
    - jsp:forward--让请求者可以向前到一个新的页面。
    - 使用java Bean，常需要使用userBean action

JavaBean的使用： 感觉就是变一个样地创建变量

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h2>Jsp 使用 JavaBean 实例</h2>

<jsp:useBean id="test" class="com.runoob.main.TestBean" />

<jsp:setProperty name="test"

property="message"

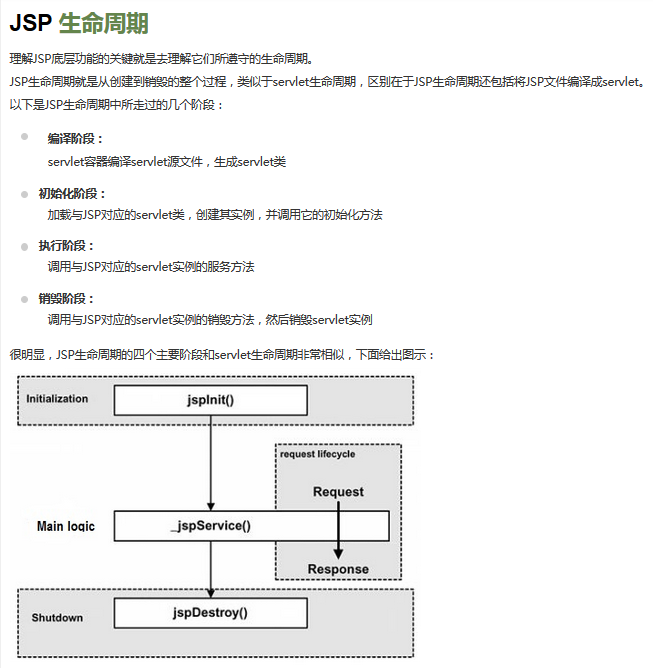
value="菜鸟教程..." />

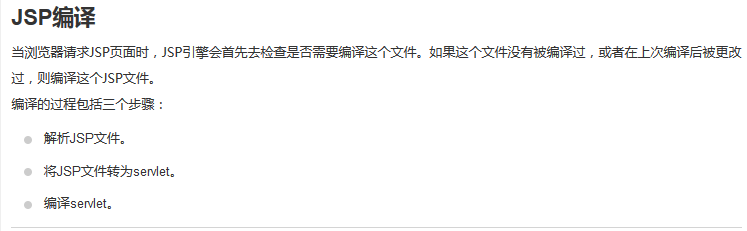
<p>输出信息....</p>

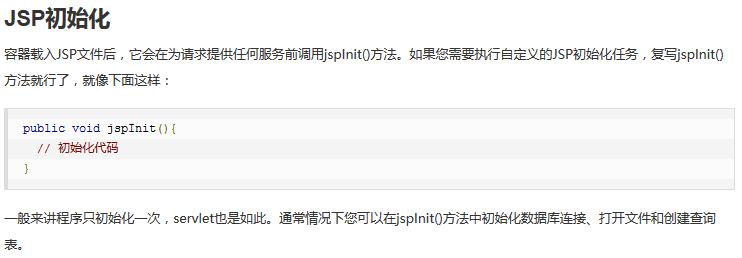
<jsp:getProperty name="test" property="message" />

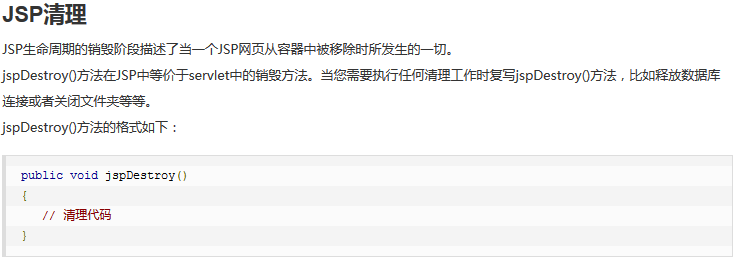
</body>

</html>









jar文件的热部署（类别热插拔）

上传部署了一个war.war，里面有index.jsp，那么应该访问war/index.jsp就好

<%%> <=> <jsp:scriptlet></jsp:scriptlet>

<%@ page language="java"; contentType="text/html; charset=UTF-8 %>

<%@ page import="java.util.\*" %>

<html>

<head>

<title>test</title>

</head>

<body>

<%

request.setCharacterEncoding("UTF-8"); //对于受到的请求的编码

String id = request.getParameter("ID"); //request.getParameter 获取从post或者get来的参数

out.println(id); //输出显示到网页out是可以直接使用的，这是jsp的预定义变量。但是这不是servlet中的预定义变量。

System.out.println("Debugging message"); 会输出到日志中。也就是idea的下面的窗口中。

%>

</body>

</html>

进一步结合：

<body>

<%

for ( int i = 0; i < 6; i++ ) {

%>

<TR>

<TD>Number</TD>

<TD><%= i+1 %></TD>

</TR>

<%

}

%>

</body>

通过JDBC数据库的连接：



第二种连击方式：

public Connection getConn() {  
 String driver = "org.mariadb.jdbc.Driver";  
 String url = "jdbc:mariadb://192.168.1.182:3306/hotel?useUnicode=true&characterEncoding=UTF-8";  
 String username = "root";  
 String password = "";  
 Connection conn = null;  
 try {  
 Class.*forName*(driver);  
 conn = DriverManager.*getConnection*(url, username, password);  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return conn;  
}

Connection conn = getConn();

见到过将数据库操作全都放到JDBC\_Operation类中，之后只需要创建这个类的对象，调用public函数，比如通过login(username,password) 的返回值判断是否应该登录成功。

String url="jdbc:mariadb://192.168.1.182:3306/yiplus?useUnicode=true&characterEncoding=UTF-8";  
Connection con=DriverManager.*getConnection*(url,"root","");

上面这种就是将数据库账号密码作为参数传入的  
Statement stmt=con.createStatement();  
String sql="select \* from result where username="+session.getAttribute("username");  
ResultSet rs=stmt.executeQuery(sql);

ResultSetMetaData rsm = rs.getMetaData();  
int colNum = rsm.getColumnCount();

for (int i = 1; i <= colNum; i++) {  
 String column\_name = rsm.getColumnName(i);

float value = rs.getFloat(column\_name);

}

如果使用绑定： 防SQL

pstsm = conn.prepareStatement("select userId from account where username=? and password=?");  
pstsm.setString(1, username);  
pstsm.setString(2, password);  
ResultSet rs = pstsm.executeQuery();

如果不是将数据库的驱动程序放到tomcat的lib目录下，而是导入项目的话需要自己在程序中注册会使用的驱动。下述网站介绍了三种注册jdbc驱动的方式：

http://blog.csdn.net/yanwushu/article/details/7574708

数据库按照字段名获取值：

while (rs.next())

{ rno = rs.getString("rno");

name = rs.getString("name");

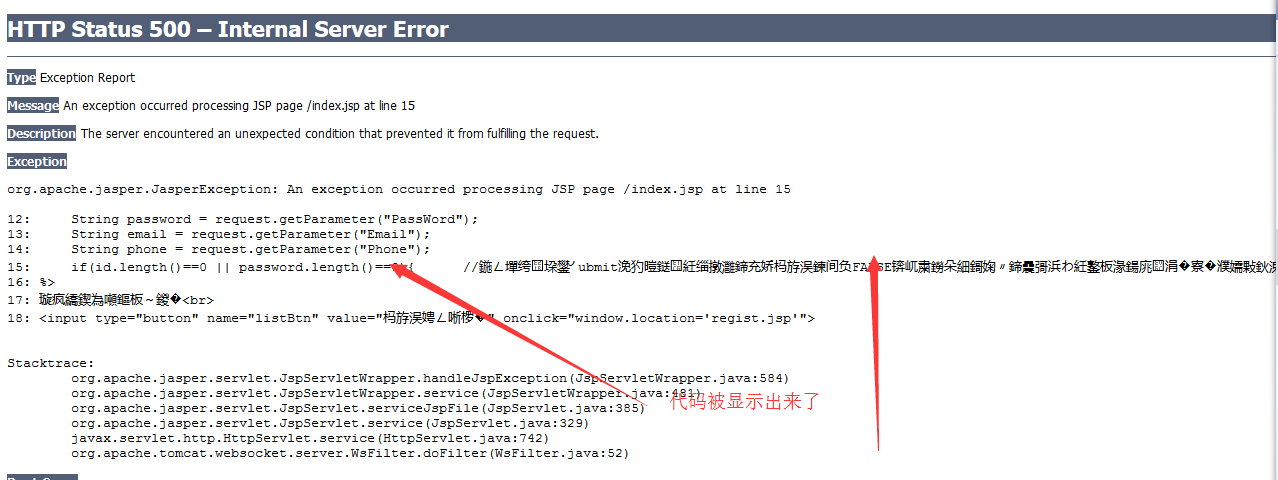
System.out.println(rno+" "+name);

}

数据库里面存的是int类型，那么读取的时候也可以getString，但是写入的时候必须setInt。

如果不是用字段，而是用index，那么就和pstsm.setString(1,code);一样，都是从1开始数的。

危险的JSP：



如果使用ODBC连接数据库，那么需要安装jar包。网上的教程一般解释说是在tomcat的安装目录下的lib，但是对于较老的版本，lib是在common\libs或者sever\libs，帮助陈科的时候是放在common\libs下，重新启动tomcat就可以在代码中引用了。

在JSP里面return;会结束继续执行

URL编码的处理：URL编码本身是基于其他编码的，所以就需要一个参数来指定编码方式

java.net.urldecoder.decode()

String mytext =   java.net.URLEncoder.encode("中国", "utf-8");

String mytext2 =   java.net.URLDecoder.decode(mytext, "utf-8");

1.直接使用函数进行处理：

表单提交分为GET和POST两种提交方式。两种方式的乱码解决又不一样，用POST提交只需要在接受的时候加上request.setCharacterEncoding("utf-8");而GET方法处理应该将接收过来的值打碎成ISO-8859-1编码的，然后再组装成UTF-8的，new String(request.getParameter(param).getBytes("iso8859-1"),"UTF-8")，或者在from表单加上 accept-charset="UTF-8"。

甚至自己重写HTML参数获取函数；http://blog.csdn.net/u012292938/article/details/49101081

2. get方式 可以再server.xml的8080端口 节点中增加编码(服务器自动进行解码处理)

打开你运行项目的tomcat 文件目录，找到 tomcat目录下的\apache-tomcat\conf\server.xml 文件，找到你的端口所在所在的xml配置 <Connector port="8080" protocol="HTTP/1.1"   
connectionTimeout="20000"   
redirectPort="8443" （新添加）URIEncoding="utf-8" /> 在其后面添加上 URIEncoding="utf-8",重启tomcat，再次运行

JSP指令： 用来设置整个jsp相关的相关属性

<%@ directive attribute="value" %>

<%@ page ... %>定义页面的依赖属性，比如脚本语言、error页面、缓存需求等等。

<%@ include ... %>：

<%@ include file="文件相对 url 地址" %>

<jsp:directive.include file="文件相对 url 地址" />

<%@include%>：页面请求之前预编译，所有代码包含进来之后，一起进行处理，把所有代码合在一起，编译成一个servlet。静态包含文件。

<jsp:include>：所有代码分别处理，在页面被请求的时候才编译，被编译成多个servlet，页面语法相对独立，处理完成之后再将代码的显示结果（处理结果）组合进来。是个动作，也就是动态包含文件。

<%@ taglib ... %> 引入标签库

<%@ taglib uri="uri" prefix="prefixOfTag" %>

<jsp:directive.taglib uri="uri" prefix="prefixOfTag" />

JSP声明的语法格式：

<%! declaration; [ declaration; ]+ ... %>

<jsp:declaration> xxxx</jsp:declaration>

举例：<%! int a, b, c;byte d; %>

JSP表达式：

<%= 表达式 %>

<jsp:expression>表达式</jsp:expression>

举例：今天的日期是: <%= (new java.util.Date()).toLocaleString()%>

JSP注释：

<%-- 注释 --%>不会被编译更不会被发送至浏览器



动态包含文件、动态产生JavaBean、XML，请求转发



application对象直接包装了servlet的ServletContext类的对象，这个对象在JSP页面的整个生命周期中都代表着这个JSP页面。这个对象在JSP页面初始化时被创建，随着jspDestroy()方法的调用而被移除。

config对象是 javax.servlet.ServletConfig 类的实例，直接包装了servlet的ServletConfig类的对象。

pageContext这个对象存储了request对象和response对象的引用。application对象，config对象，session对象，out对象可以通过访问这个对象的属性来导出。

JSP中的输出：

1. out是response对象。可以通过out.print()或者out.println()或者out.write()函数进行输出，out.flush()刷新输出流。 print方法是子类JspWriter，write是Writer类中定义的方法；System.out.println()用的是标准输出流，这个是输出在控制台上的

out.println(null)报错

out.println(request.getRemoteUser())却可以输出null

1. <%=表达式%> 这个东东放到HTML中就会将变量进行显示。

对request的操作：HttpServleRequest类

**Enumeration getAttributeNames() 返回本对象所有属性名称的集合**

**Object getAttribute(String name)** 回名称为name的属性值，如果不存在则返回null常用于servlet页面传递参数给jsp，也就是内部传值

**Enumeration getHeaderNames() 返回包头的集合**

**boolean isSecure()**

返回request是否使用了加密通道，比如HTTPS

**String getRemoteAddr()** 返回客户端的IP（代理）地址

**String getRemoteHost()** 返回客户端的完整名称 我试了下得到0:0:0:0:0:0:0:1

**String getRemoteUser()**

返回客户端通过登录认证的用户名，若用户未认证则返回null

**String getParameter(String name)** 返回此request中name指定的参数，若不存在则返回null（HTTP方法的输入数据，对于GET而言就是查询字符串）。checkbox被选中就会返回on（创建一个chekbox然后显示结果到GET就知道了），没有被选中就不会有值，所以使用这个函数获取的话就是null。

**getInputStream():**调用此方法来读取来自客户端的二进制数据流。

**String getHeader(String name)** 返回name指定的信息头

**String getQueryString()** 返回此 request URL包含的查询字符串 专门对于GET方法

**String getMethod()** 返回此request中的HTTP方法，比如 GET,，POST，或PUT

**String getCharacterEncoding()** 返回request的字符编码集名称

**int getContentLength()**

返回request主体所包含的字节数，若未知的返回-1

**Cookie[] getCookies()**

**返回客户端已有的 cookie数组**

**HttpSession getSession()**

返回request对应的session对象，如果没有，则创建一个

创建出来的一个例子：JSESSIONID=2A5E359B182EB09FE597DF8700B53769; Path=/; HttpOnly

**HttpSession getSession(boolean create)**

加了个限制，创建还需要create这个参数的值是true。

**String getRequestedSessionId()** 返回request指定的session ID

**String getServletPath()** 返回所请求的servlet路径

<%

Enumeration headerNames = request.getHeaderNames();

while(headerNames.hasMoreElements()) {

String paramName = (String)headerNames.nextElement();

out.print("<tr><td>" + paramName + "</td>\n");

String paramValue = request.getHeader(paramName);

out.println("<td> " + paramValue + "</td></tr>\n");

}

%>

对response的操作： HttpServletResponse类

**String encodeURL(String url)**

将URL编码，回传包含Session ID的URL

**boolean containsHeader(String name)**

返回指定的响应头是否存在

**void addHeader(String name, String value)**

添加指定名称的响应头和值

**void addCookie(Cookie cookie)**

添加指定的cookie至响应中

**void setHeader(String name, String value)**

使用指定名称和值设置响应头的名称和内容

**void sendError(int sc)**

使用指定的状态码向客户端发送一个出错响应，然后清除缓存

**void sendError(int sc, String msg)**

增加一个向客户端发送一个出错响应的String参数

**void setStatus(int sc)**

设置响应的状态码

**void setBufferSize(int size)**

设置响应体的缓存区大小

response.setContentType("text/html;charset=UTF-8");设置响应内容类型

可能没有对响应头部的内容的get

r**esponse.sendRedirect("login.jsp");**

重定向到指定页面

JSP过滤器：

jsp中和Servlet中的过滤器都是java类，过来不起可以动态地拦截请求和响应，以变换或使用包含在请求或响应中的信息。

过滤器通过 Web 部署描述符（web.xml）中的 XML 标签来声明，然后映射到您的应用程序的部署描述符中的 Servlet 名称或 URL 模式。Filter的执行顺序与在web.xml配置文件中的配置顺序一致，一般把Filter配置在所有的Servlet之前。



<?xml version="1.0" encoding="UTF-8"?>

<web-app>

<filter>

<filter-name>LoginFilter</filter-name>

<filter-class>com.xxxx.test.LogFilter</filter-class>

<init-param>元素用于为过滤器指定初始化参数

<param-name>Site</param-name>

<param-value>菜鸟教程</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>LogFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<servlet>

<!-- 类名 --> 指定了拦截的servlet名称

<servlet-name>DisplayHeader</servlet-name>

<!-- 所在的包 -->

<servlet-class>com.runoob.test.DisplayHeader</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>DisplayHeader</servlet-name>

<!-- 访问的网址 -->

<url-pattern>/TomcatTest/DisplayHeader</url-pattern>

</servlet-mapping>

</web-app>

/\*它的3个函数和jsp的后三个函数是类似的\*/

//导入必需的 java 库

import javax.servlet.\*;

import java.util.\*;

//实现 Filter 类

public class LogFilter implements Filter {

public void init(FilterConfig config) throws ServletException {

// 获取初始化参数

String site = config.getInitParameter("Site");

//handle

……

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws java.io.IOException, ServletException {

//handle

……

// 把请求传回过滤链

chain.doFilter(request,response);

}

public void destroy( ){

/\* 在 Filter 实例被 Web 容器从服务移除之前调用 \*/

}

}

Cookie: Cookie类

Set-Cookie: name=runoob; expires=Friday, 04-Feb-07 22:03:38 GMT;

path=/; domain=runoob.com

有效期、路径、域名、安全参数HttpOnly、键值对

处理中文的时候进行编码：

String str = java.net.URLEncoder.encode("中文"，"UTF-8");

String str = java.net.URLDecoder.decode("编码后的字符串","UTF-8");

一个cookie对象只能存储一个key和value

Cookie cookie = new Cookie("key","value");

response.addCookie(cookie);

**public void setDomain(String pattern)**

设置cookie的域名，比如w3cschool.cc

**public String getDomain()**

获取cookie的域名，比如w3cschool.cc

**public void setMaxAge(int expiry)**

设置cookie有效期，以秒为单位，默认有效期为当前session的存活时间

**public int getMaxAge()**

获取cookie有效期，以秒为单位，默认为-1 ，表明cookie会活到浏览器关闭为止

**public void setPath(String uri)**

设置cookie 的路径，默认为当前页面目录下的所有URL，还有此目录下的所有子目录

**public String getPath()**

获取cookie 的路径

**public String getName()**

返回 cookie的名称，名称创建后将不能被修改，就是获取key，我们创建以后能改动的也就只有对应的value了

**public void setValue(String newValue)**

设置 cookie的值

**public String getValue()**

获取cookie的值

**public void setSecure(boolean flag)**

指明cookie是否要加密传输

**public void setComment(String purpose)**

设置注释描述 cookie的目的。当浏览器将cookie展现给用户时，注释将会变得非常有用

**public String getComment()**

返回描述cookie目的的注释，若没有则返回null

Session： 不需要导入库，是内置对象

<%@ page session="false" %> 显式禁用掉session的使用。默认情况下jsp是允许会话跟踪也就是开启session的。

id是必备的默认属性，还有创建时间与最后访问时间

// 如果不存在 session 会话，则创建一个 session 对象

HttpSession session = request.getSession(true);

**public Enumeration getAttributeNames()**

返回session对象中所有的对象名称

**public Object getAttribute(String name) 居然可以存入Object对象，切记序列化，否则会过一会数据就像是没了一样**

返回session对象中与指定名称绑定的对象，如果不存在则返回null

**public void setAttribute(String name, Object value)**

使用指定的名称和值来产生一个对象并绑定到session中

**public void invalidate()**

将session无效化，解绑任何与该session绑定的对象

**public boolean isNew()**

返回是否为一个新的客户端，或者客户端是否拒绝加入session

**public String getId()**

返回session对象的ID

**public long getCreationTime() session很讲究时间控制**

返回session对象被创建的时间， 以毫秒为单位，从1970年1月1号凌晨开始算起

**public long getLastAccessedTime()**

返回客户端最后访问的时间，以毫秒为单位，从1970年1月1号凌晨开始算起

**public int getMaxInactiveInterval()**

返回最大时间间隔，以秒为单位，servlet 容器将会在这段时间内保持会话打开

**public void setMaxInactiveInterval(int interval)**

用来指定时间，以秒为单位，servlet容器将会在这段时间内保持会话有效

如果使用的是Tomcat，可以向下面这样配置web.xml文件：

<session-config>

<session-timeout>15</session-timeout>

</session-config>

单位是分钟

session无效以后再调用相关函数就会抛java.lang.illegalstateexception，getSession会重新创建，有一个新的JSESSIONID。

writer.write(""+session.getMaxInactiveInterval());

不能是（String）来进行强制转换，因为int不能直接转为String，但是使用这种方式就可以。这是一种字符串转换方法。

<script>  
 var errori ='<%=request.getParameter("error")%>';  
 if(errori=='yes'){  
 alert("账号或密码错误!");  
 }  
</script>

**Java Bean**：是带有一组私有属性及相应Getter、Setter的Java类

<jsp:useBean id=“reader” class=“包名.Reader” scope=“session” />

如果当前session上没有reader对象，实例化一个bean对象reader。id就是创建的变量的名字。

scope属性可以设置

page 表示该Bean当前页面内可以使用

request 表示该Bean在当期的客户请求内有效

session 表示该Bean对当前HttpSession内所有页面都有效

application 表示该Bean对该应用的所有页面都有效

<%@ page contentType=*"text/html; charset=GBK" %>*

<meta http-equiv=*"Content-Type" content="text/html; charset=gbk">*

<jsp:useBean id=*"output" class = "java.lang.String" scope = "request"></jsp:useBean>* 这一行直接创建了一个变量

<html>

<body>

<%=output%>

</body>

</html>

**JSP内建对象：**

* page 对象 page对象代表JSP本身
* config 对象
  + config 对象里存放着一些Servlet 初始的数据结构。
* request 对象
  + request 对象包含所有请求的信息，实现javax.servlet.http.HttpServletRequest接口。
* response 对象
  + response 对象主要将JSP 处理数据后的结果传回到客户端。  
    response 对象是实现javax.servlet.http.HttpServletResponse 接口。
* out 对象 out 对象能把结果输出到网页
* session 对象
  + session对象表示当前用户的会话(session)状况，实现javax.servlet.http.HttpSession接口。
* application对象
  + application对象最常被使用在存取环境的信息，  
    实现javax.servlet.ServletContext 接口。
* pageContext对象
  + pageContext对象能够存取其他隐含对象。

MVC中：

JavaBeans充当Model

JSP pages主要用于实现View

要想在jsp中使用自定义类，必须要定义包名，IDE中的default package就是没有包名，所以不能被import使用。自定义包名以后，web.xml中的servlet-class也要改改。

项目打war包：

<http://blog.csdn.net/mrhuangxiutao/article/details/64921583>

部署在服务器上面：

<https://www.cnblogs.com/george93/p/7478756.html>

一直部署失败，wireshark抓包分析才知道是因为jsp、servlet因为war包的原因，在访问目录前面有一个war解压后文件夹的名字，但是我卸载js文件中的XMLHTTPRequest没有考虑到这个。所以写项目的时候还是不要省略项目的名于HTTP URL中。