JavaWeb

尚硅谷 大数据

版本 V1.0

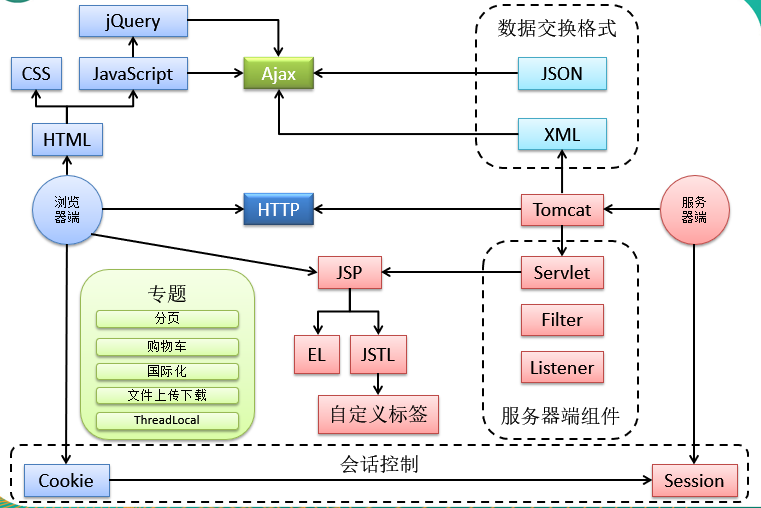
# 一、JavaWeb简介

## 1. 学习目的

整个javaWeb阶段的内容通过实际的案例贯穿学习， 所涉及到的技术知识点会在案例中根据不同的需求引入。该阶段的学习目标是了解javaWEB的整个技术体系，掌握常用的技术知识点。

JavaWeb阶段的学习主要是为了掌握JavaWeb核心的内容。

## 2. JavaWeb的技术体系



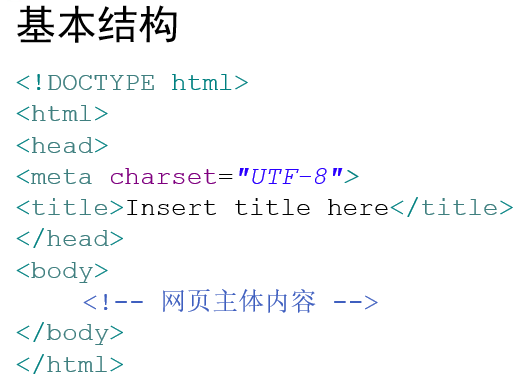
# 二、HTML和CSS简介

## 1. HTML简介

使用HTML完成一个简单的登录页面的开发！

HTML指的超文本标记语言(Hyper Text Markup Language)，是一种用来描述网页的语言。超文本指的是除了可以包含文字之外，还可以包含图片、链接、音乐、视频、程序等内容。

HTML网页的组成:



HTML+CCC+JS三者的关系：一般将一个网页分成三个部分：结构(HTML)、表现(CSS)、行为(JavaScript)；

HTML的基本语法：

①标签不能交叉嵌套

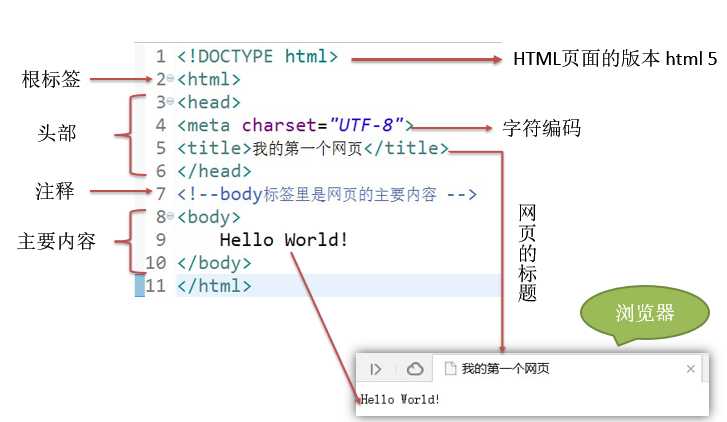
②标签必须正确关闭

③属性必须有值，属性值必须加引号

④注释不能嵌套

⑤HTML不区分大小写





示例：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>  这是我的第一个Html  <video src=*"test.mp4"* controls=*"controls"* ></video>  <img alt=*"尚硅谷"* src=*"atguigu.jpg"*/>  </body>  </html> |

## 2. 常见的HTML标签

|  |  |
| --- | --- |
| 标签 | 作用 |
| <html> | 文档的根标签 |
| <head> | 文档的头标签 |
| <body> | 文档的体标签 |
| <a> | 超链接标签 |
| <form> | 表单标签 |
| <table> | 表格标签 |
| <span> | 用来组合文档中的行内元素，以便于通过样式来格式化它们 |
| <style> | 为文档定义样式信息 |
| <br> | 换行 |
| <hr> | 分割线 |

## 3. CSS

### 3.1 CSS简介

CSS 指层叠样式表 (Cascading Style Sheets)。CSS允许你创建一些规则，来指定元素中的内容将会如何显示。例如，可以指定页面背景颜色，字体的大小、颜色等。

CSS的基本语法：

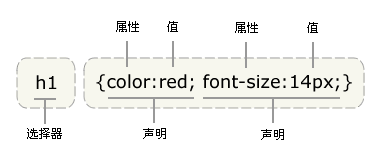
selector {declaration1; declaration2; ... declarationN }

CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明组成。选择器负责选取文档中的要素标签。

声明负责为他们指定样式属性。

例如：

h1 {color:red; font-size:14px;}



这个语法代表为文档中所有的<h1>标签，设置其字体大小为14像素，颜色为红色。

### 3.2 CSS的设置方式

css的设置方式：

①通过<Style>标签设置

<style type=*"text/css"*>**h1** {color: *red*; font-size: *14px*; }</style>

②在标签的属性中使用style属性设置样式

<h1 style="color: blue; ">鸡你太美</h1>

③如果页面中要设置的样式过多，可以将样式信息独立写在一个css文件中，通过<link>标签来引入

<link type="text/css" href="my.css" rel="stylesheet ">

### 3.3 CSS的选择器

元素(标签)选择器

例子：p {color: red; font-size: 12px;}

id选择器

例子：#id {color: red; font-size: 12px;}

类选择器

例子：.class{color: red; font-size: 12px;}

选择器分组

例子： p ，#id，.class {color: red; }

## 4. 简单的登录页面



<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body bgcolor=*"pink"*>

<h1>欢迎登录</h1>

<form action=*""*>

用户名称：<input name=*"username"* value=*""* type=*"text"*>

<br>

用户密码：<input name=*"password"* value=*""* type=*"password"*>

<br>

<input type=*"submit"* value=*"提交"*>

</form>

</body>

</html>

# 三、登录功能的实现

## 1.服务器简介

页面需要发布到服务器上，才可以被用户所访问。因此首先需要搭建服务器环境！

Web服务器主要用来接收客户端发送的请求和响应客户端请求。

请求：浏览器（客户端）向服务器发送信息

响应：服务器向（客户端）浏览器返回信息

请求和响应是成对出现的。

作为JavaWeb程序来说，还需要有Servlet容器，容器的主要作用就是调用java程序处理用户发送的请求，并响应指定的资源，关于Servlet容器我们还需要一点一点学习。开发JavaWeb项目Web服务器和Servlet容器是必须的，通常情况二者是合在一起的。

所谓web资源即放在Internet网上供外界访问的文件或程序，又根据它们呈现的效果及原理不同，将它们划分为**静态资源**和**动态资源**。

**静态web资源**：静态网页 HTML、CSS文件、文本、音频、视频；

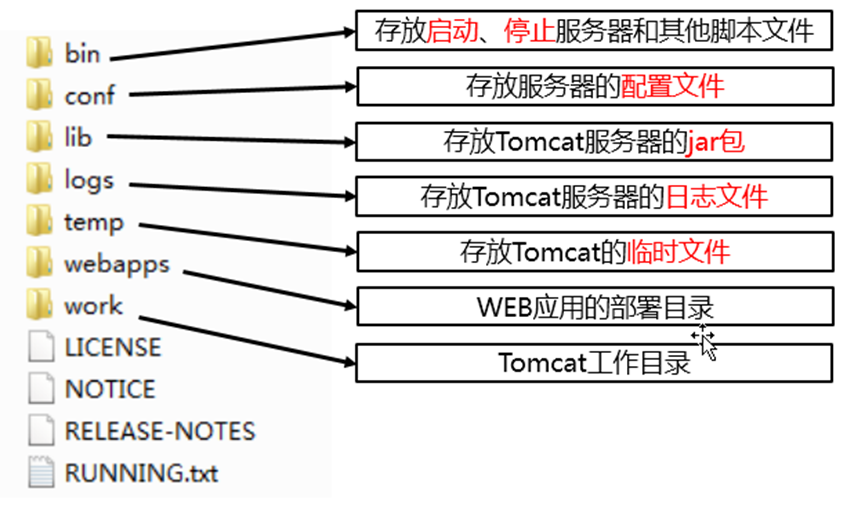
**动态web资源**：jsp、Servlet、php程序；

常见的Web服务器:

* Web服务器主要用来接收客户端发送的请求和响应客户端请求。
* **Tomcat（Apache）**：当前应用最广的JavaWeb服务器；
* JBoss（Redhat红帽）：支持JavaEE，应用比较广EJB容器 –> SSH轻量级的框架代替
* GlassFish（Orcale）：Oracle开发JavaWeb服务器，应用不是很广；
* Resin（Caucho）：支持JavaEE，应用越来越广；
* Weblogic（Orcale）：要钱的！支持JavaEE，适合大型项目；
* Websphere（IBM）：要钱的！支持JavaEE，适合大型项目

## 2.Tomcat服务器的安装及配置

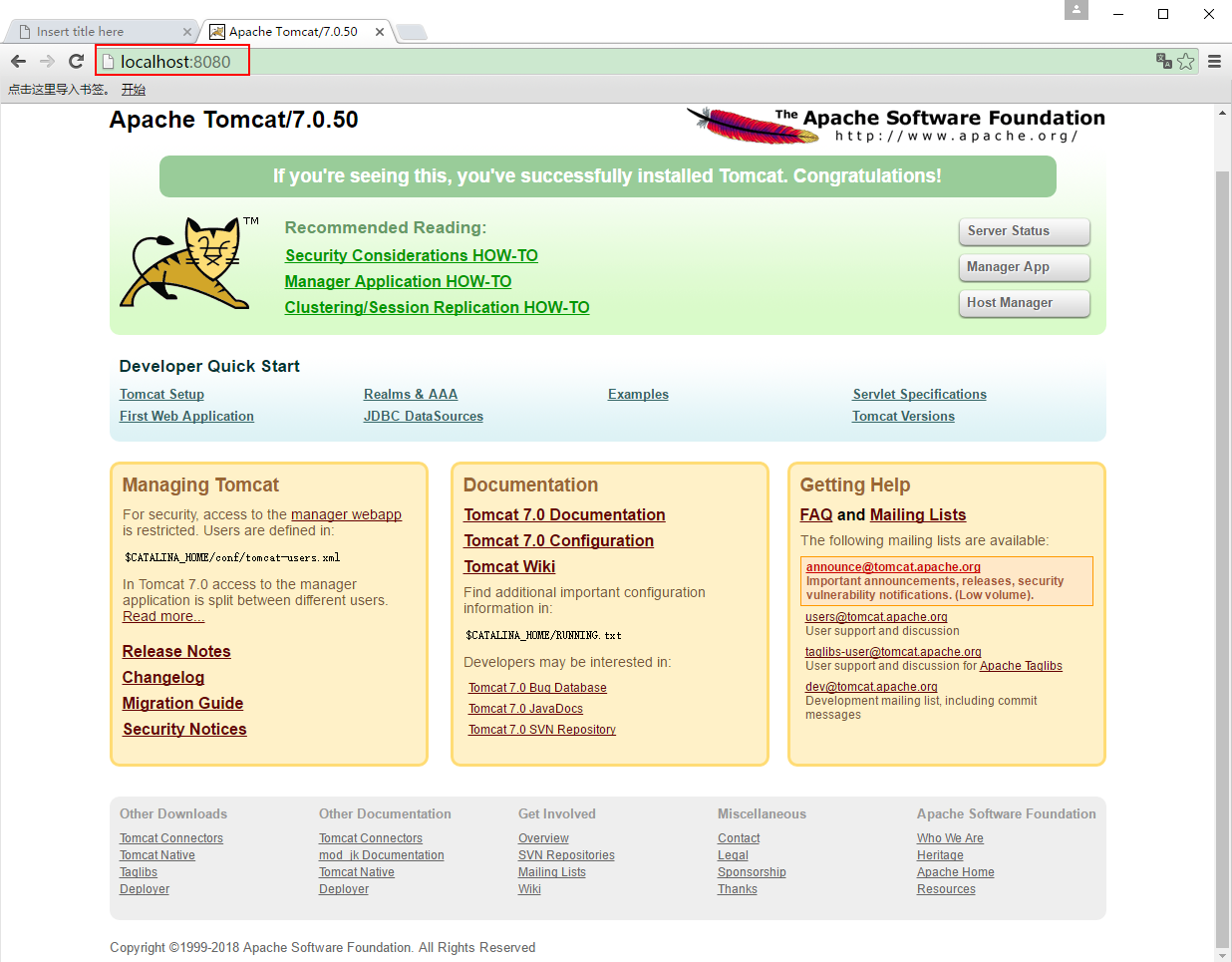
* 将Tomcat的安装包解压到磁盘的任意位(非中文无空格)
* Tomcat的运行需要正确配置JAVA\_HOME
* Tomcat服务的目录结构



启动：在Tomcat安装目录下的bin目录，双击startup.bat

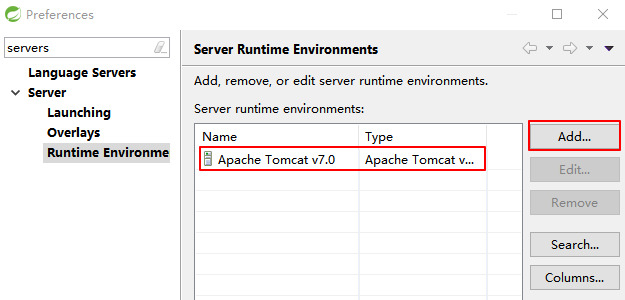
停止：在Tomcat安装目录下的bin目录，双击shutdown.bat

启动Tomcat后，访问<http://localhost:8080>后，看到如下页面即tomcat启动成功：

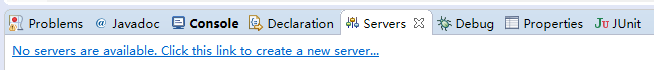


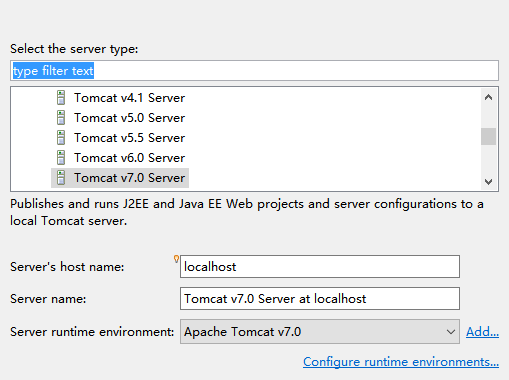
## 3.在eclipse中配置tomcat

* 在Eclipse中配置运行环境

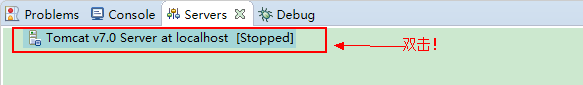


* Window----show view----Server:在Eclipse中创建新的Server





配置Tomcat的工程部署目录：

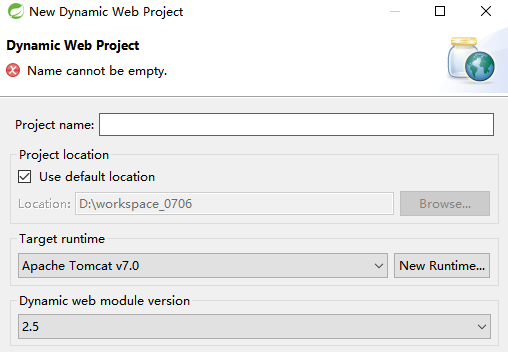


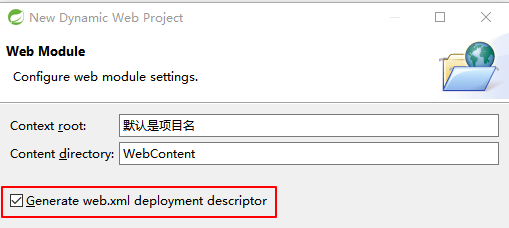
## 4.创建动态的web工程

注意：

Target runtime选择自己配置的Tomcat服务器版本；

Dynamic web module version选择2.5版本；





## 5.Servlet

### 5.1 简介

Servlet是Sun公司制定的一套技术标准，包含与Web应用相关的一系列接口，是Web应用实现方式的宏观解决方案。而具体的Servlet容器负责提供标准的实现。

Servlet作为服务器端的一个组件，它的本意是“服务器端的小程序”。Servlet的实例对象由Servlet容器负责创建；Servlet的方法由容器在特定情况下调用；Servlet容器会在Web应用卸载时销毁Servlet对象的实例。

Servlet主要完成接收请求、处理请求和完成响应三个步骤！

### 5.2 开发一个Servlet

Servlet实现类由我们编写，而由Web服务器（Servlet容器）调用，每个Servlet都必须实现javax.servlet.Servlet

源码导入：Servlet是一个接口，servlet是java定义的一套服务器规范，由各大服务器厂商来实现，例如如果使用的是tomcat，就需要导入tomcat的源码

第一种：实现Serlvet接口，重写Service方法，其中ServletRequest代表请求报文，ServletResponse代表响应报文

**public** **class** LoginServlet **implements** Servlet{

@Override

**public** **void** destroy() {

}

@Override

**public** ServletConfig getServletConfig() {

**return** **null**;

}

@Override

**public** String getServletInfo() {

**return** **null**;

}

@Override

**public** **void** init(ServletConfig arg0) **throws** ServletException {

}

@Override

**public** **void** service(ServletRequest arg0, ServletResponse arg1) **throws** ServletException, IOException {

}

}

原生的Servlet接口中，有很多我们不需要使用的方法，因此，在实际使用中，我们通常使用继承HttpServlet来创建一个Servlet：

**public** **class** LoginServlet **extends** HttpServlet{

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

}

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

}

}

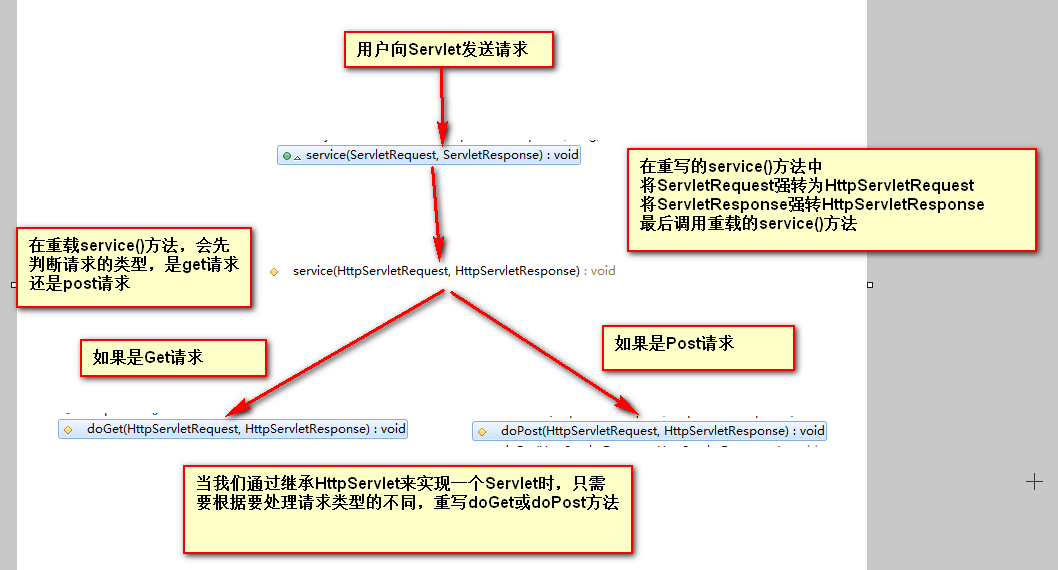
Servlet类的相关方法:

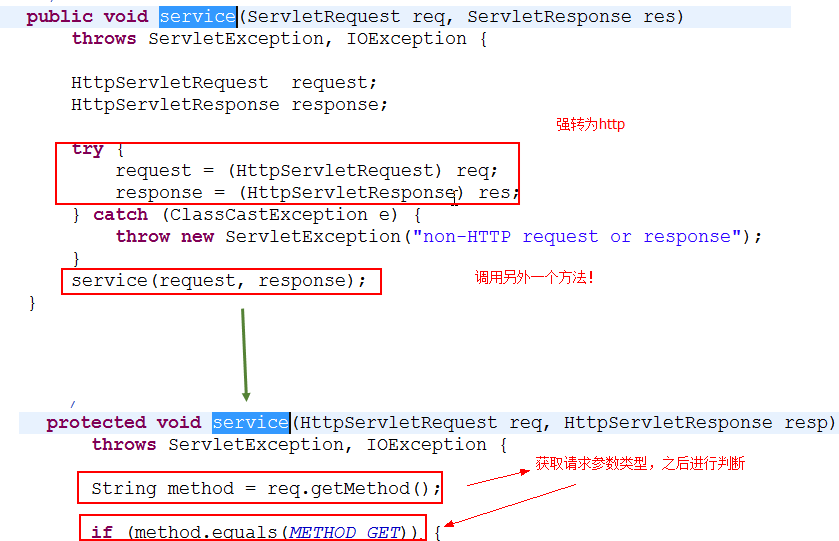
* doGet Servlet中用于处理get请求的方法
* doPost Servlet中用于处理post请求的方法

处理请求的doGet、doPost方法的重要参数：

HttpServletRequest代表请求报文对象；

HttpServletResponse：代表响应报文对象；





### 5.3 接受请求

哪些Servlet可以处理哪些请求，需要在工程的web.xml中配置！

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>com.atguigu.web.servlet.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

三个属性：

<servlet-class>: Servlet的全类名，服务器使用全类名来创建Servlet的实例。

<servlet-name>:Servlet的别名

<url-pattern>: Servlet可以处理的请求

### 5.4 处理请求

在继承了HttpServlet的方法中，要处理哪些请求，只需要重写哪些方法即可。如处理GET请求，就重写doGet()方法。

HttpServletRequest代表封装的浏览器发送给服务器的请求报文。

        常用方法：

String getParameter(String paramName)：获取指定请求参数的值；

String getMethod()：获取请求方法，例如GET或POST；

String getHeader(String name)：获取指定请求头的值；

String getContextPath():获取项目名

void setCharacterEncoding(String encoding)：设置请求体的编码！因为GET请求没有请求体，所以这个方法只只对POST请求有效。当调用request.setCharacterEncoding(“utf-8”)之后，再通过getParameter()方法获取参数值时，那么参数值都已经通过了转码，即转换成了UTF-8编码。所以，这个方法必须在调用getParameter()方法之前调用！

### 5.5 完成响应

HttpServletResponse代表：封装的服务器发送给浏览器的响应报文。

主要功能：

①向浏览器写入响应结果：

reponse.getWriter().print();

②完成重定向操作

response.sendRedirect();

常用方法：

PrintWriter getWriter()：获取字符响应流，使用该流可以向客户端输出响应信息。

ServletOutputStream getOutputStream()：获取字节响应流，向客户端响应字节数据。

void setCharacterEncoding(String encoding)：用来设置字符响应流的编码，例如在调用setCharacterEncoding(“utf-8”);之后，再response.getWriter()获取字符响应流对象，这时的响应流的编码为utf-8，使用response.getWriter()输出的中文都会转换成utf-8编码后发送给客户端。

void setHeader(String name, String value)：向客户端添加响应头信息。

void setContentType(String contentType)：该方法是setHeader(“content-type”, “xxx”)的简便方法，即用来添加名为content-type响应头的方法。如果响应数据为文本类型，那么还要台同时设置编码，例如setContentType(“text/html;chartset=utf-8”)表示响应数据类型为文本类型中的html类型，并且该方法会调用setCharacterEncoding(“utf-8”)方法。

## 6.登录功能的实现

### 6.1 准备工作

Sql建表语句：

CREATE TABLE tbl\_user(

id INT(10) PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(20),

password VARCHAR(20)

);

数据库连接信息属性文件：

user=root

password=guodai

jdbcUrl=jdbc:mysql:///query\_data

driverClass=com.mysql.jdbc.Driver

获取数据库连接的工具类：

**public** **class** ConnectionUtil {

**private** **static** String *driverClass*=**null**;

**private** **static** String *jdbcUrl*=**null**;

**private** **static** String *username*=**null**;

**private** **static** String *password*=**null**;

**private** **static** Properties *properties*=**new** Properties();

**private** **static** ThreadLocal<Connection> *threadLocal* =**new** ThreadLocal<>();

**static** {

**try** {

InputStream is = ConnectionUtil.**class**.getClassLoader().getResourceAsStream("db.properties");

//加载外部配置文件

*properties*.load(is);

//设置属性

*driverClass*=*properties*.getProperty("driverClass");

*jdbcUrl*=*properties*.getProperty("jdbcUrl");

*username*=*properties*.getProperty("user");

*password*=*properties*.getProperty("password");

//注册驱动

Class.*forName*(*driverClass*);

}**catch** (Exception e) {

e.printStackTrace();

}

}

// 获取连接，确保当前线程只有一个连接

**public** **static** Connection getConnection() **throws** SQLException {

Connection connection = *threadLocal*.get();

**if** (connection==**null**) {

connection=DriverManager.*getConnection*(*jdbcUrl*, *username*, *password*);

}

**return** connection;

}

**public** **static** **void** closeConnection() **throws** SQLException {

Connection connection = *threadLocal*.get();

**if** (connection !=**null** && !connection.isClosed()) {

connection.close();

}

*threadLocal*.set(**null**);

}

}

Dao层实现：

**public** **class** UserDaoImpl **implements** UserDao{

@Override

**public** User findUserByUserNameAndPassword(String username, String password) {

**try** {

Connection connection = ConnectionUtil.*getConnection*();

String sql="select id,username,password from tbl\_user where username=? and password=?";

PreparedStatement ps=connection.prepareStatement(sql);

ps.setString(1, username);

ps.setString(2, password);

ResultSet rs = ps.executeQuery();

User user=**null**;

**if** (rs.next()) {

user=**new** User();

user.setUsername(rs.getString("username"));

user.setPassword(rs.getString("password"));

}

**return** user;

} **catch** (SQLException e) {

e.printStackTrace();

}**finally** {

**try** {

ConnectionUtil.*closeConnection*();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**return** **null**;

}

}

### 6.2 分析

①页面表单的action属性发送请求到Servlet；

②Servlet处理请求，获取表单传入的参数

③调用Dao，验证参数

④根据验证结果，判断是否成功

### 6.3 表单

<form action=*"login"* method=*"get"*>

用户名称：<input name=*"username"* value=*""* type=*"text"*><br>

用户密码：<input name=*"password"* value=*""* type=*"password"*><br>

<input type=*"submit"* value=*"提交"*>

</form>

### 6.4 Servlet

**public** **class** LoginServlet **extends** HttpServlet{

**private** UserDao ud;

**public** LoginServlet() {

ud=**new** UserDaoImpl();

}

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

System.***out***.println("接受到了页面发来的请求！");

String username = req.getParameter("username");

String password = req.getParameter("password");

User user = ud.findUserByUserNameAndPassword(username, password);

**if** (user==**null**) {

System.***out***.println("登录失败！");

}**else** {

System.***out***.println("登录成功！");

}

}

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

}

}

## 7. 完成响应

使用response将信息，写回给浏览器！

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

System.***out***.println("接受到了页面发来的请求！");

String username = req.getParameter("username");

String password = req.getParameter("password");

User user = ud.findUserByUserNameAndPassword(username, password);

**if** (user==**null**) {

resp.getWriter().print("登录失败");

}**else** {

resp.getWriter().print("登录成功");

}

}

## 8.中文字符乱码的解决

①get请求乱码：

修改Server的server.xml,在8080端口号处添加属性URIEncoding= utf-8；

<Connector URIEncoding=*"UTF-8"* connectionTimeout=*"20000"* port=*"8080"* protocol=*"HTTP/1.1"* redirectPort=*"8443"*/>

②post请求乱码：

在第一次获取请求参数之前，设置编码字符集：req.setCharacterEncoding("utf-8");

③响应乱码：

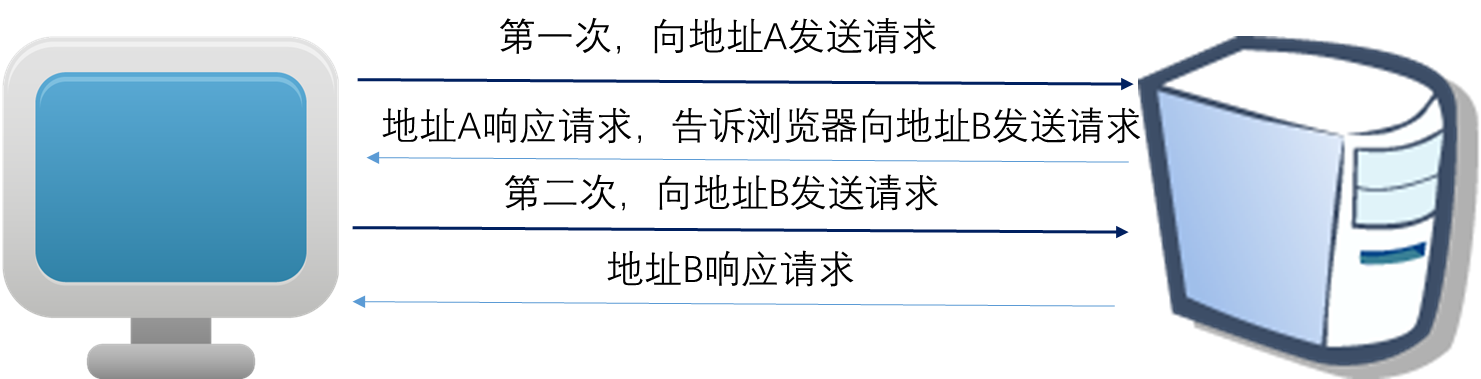
在获取响应的流之前，设置响应的数据类型：resp.setContentType("text/html;charset=utf-8");

## 9.重定向和转发

### 9.1 重定向

重定向的原理：

servlet发送给浏览器一个特殊的响应，这个响应的响应状态吗为302，在响应头上有location属性，为需要重定向的地址。此时，浏览器会向指定的地址再次发送请求！

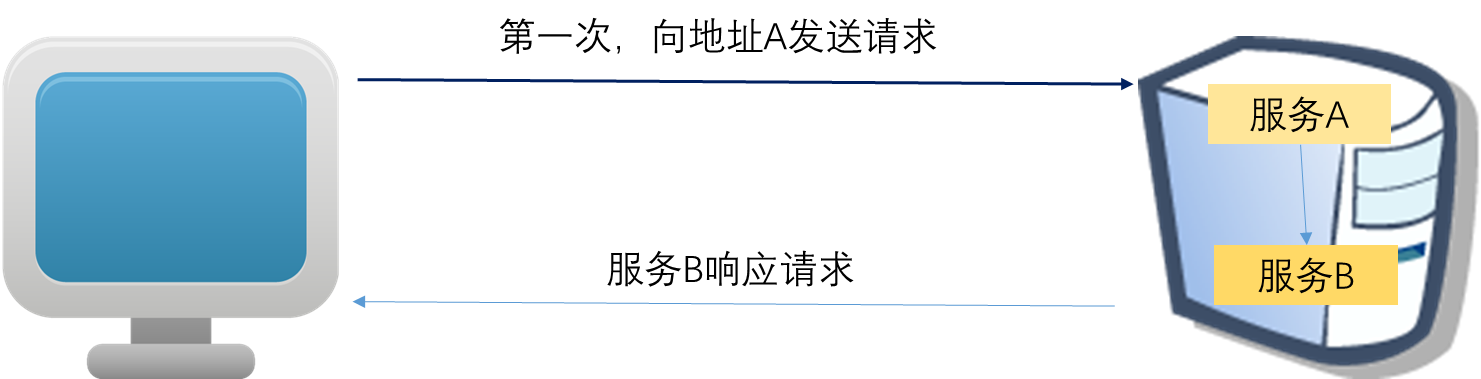


如何重定向： repsonse.sendRedirect(“xxx”);

### 9.2. 转发

请求的转发request.getRequestDispatcher("路径").forward(request,response);

转发的定义：转发指的是当前Servlet不去处理用户的请求，而是在服务器内部调用其他的资源替他处理请求。



### 9.3 转发和重定向的对比

|  |  |  |
| --- | --- | --- |
|  | 转发 | 重定向 |
| 浏览器发送请求次数 | 1次 | 2次 |
| 发生的位置 | 服务器内部 | 浏览器 |
| 浏览器地址栏 | 不改变 | 改变 |
| 浏览器感知 | 不知道 | 知道 |
| 是否可以传递数据 | 可以 | 不可以 |
| 如何操作 | 通过request对象 | 通过response对象 |
| 目标资源：WEB-INF下的资源 | 能访问 | 不能访问 |

总结：

①能用重定向就不用转发；

②当需要传递请求中的数据的时候，只能使用转发！

## 10. JSP

### 10.1 JSP简介

JSP全称Java Server Pages，顾名思义就是运行在java服务器中的页面，也就是在我们JavaWeb中的动态页面，其本质就是一个Servlet。其本身是一个动态网页技术标准，它的主要构成有HTML网页代码、Java代码片段、JSP标签几部分组成，后缀是.jsp。

相比于Servlet，JSP更加善于处理显示页面，而Servlet跟擅长处理业务逻辑，两种技术各有专长，所以一般我们会将Servlet和JSP结合使用，Servlet负责业务，JSP负责显示。

一般情况下， 都是Servlet处理完的数据，转发到JSP，JSP负责显示数据的工作。

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

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

<body>

</body>

</html>

### 10.2 JSP的脚本片段

脚本片段是嵌入到JSP中Java代码段，格式以<%开头，%>结尾，两个%号之间就可以编写Java代码了。

<% System.out.println("这是一个JSP页面！"); %>

### 10.3 JSP的表达式

JSP表达式用来直接将Java变量输出到页面中，格式以<%=开头，以%>结尾，中间是我们要输出的内容。

<%=**new** Date() %>

### 10.4 JSP隐含对象

* JSP的隐含对象
  + out（JspWriter）：相当于response.getWriter()获取的对象，用于在页面中显示信息。
  + config（ServletConfig）：对应Servlet中的ServletConfig对象。
  + page（Object）：对应当前Servlet对象，实际上就是this。
  + pageContext（PageContext）：当前页面的上下文，也是一个域对象。
  + exception（Throwable）：错误页面中异常对象
  + request（HttpServletRequest）：HttpServletRequest对象
  + response（HttpServletResponse）：HttpServletResponse对象
  + application（ServletContext）：ServletContext对象
  + session（HttpSession）：HttpSession对象

隐含对象是指一个JSP对象在初始化的时候，就已经赋值过的属性。



### 10.5 JSP的原理

实际上访问JSP，访问的并不是JSP本身，而是JSP所对应的那个Java类（Servlet）。当我们第一次访问JSP时，服务器会先将JSP翻译为一个.java，然后将java文本编译为class文件，实际上就是这个类来处理的请求在以后多次访问时，直接调用该类对象处理请求。

JSP需要经过Tomcat的JspServlet进行解析，所以，浏览器无法直接访问JSP。

jsp -翻译-> java -编译-> class。



## 11. EL表达式

* EL表达式
* EL是JSP内置的表达式语言，用以访问页面的上下文以及不同作用域中的对象 ，取得对象属性的值，或执行简单的运算或判断操作。EL在得到某个数据时，会自动进行数据类型的转换。
* **EL表达式用于代替JSP表达式(<%= %>)在页面中做输出操作。**
* EL表达式仅仅用来读取数据，而不能对数据进行修改。
* 使用EL表达式输出数据时，**如果有则输出数据，如果为null则什么也不输出。**
* **EL表达式的语法:**

**${ EL表达式 （可完成取值、简单的判断、简单的运算等）}**

* EL取值的四个域:

pageScope

requestScope

sessionScope

applicationScope

Scope为域的意思，xxxScope为获取指定范围存放数据的Map。示例：

${ pageScope .user}：从当前页面存放数据的map中获取user;

${requestScope.user}：从当前请求存放数据的map中获取user;

${sessionScope.user}：从当前会话存放数据的map中获取user;

${applicationScope.user}：从当前应用存放数据的map中获取user;

EL取值示例：

①获取域中的对象可以直接使用对象名，如获取域中名字为user的对象：${user}

②获取对象的属性值可以直接通过“对象.属性名”：${user.name}

注意：这里的属性名是get和set方法对应的属性值，并不是对象中的变量名。

比如：如果获取name属性值，那么该对象中必定要存在一个getName()方法。

③获取Map中属性时可以以直接通过属性的key：${map.key}

EL运算实例：

①使用EL进行基本的数学运算： ${2+3}

②判断是否为空： ${empty xxx}

EL表达式 中隐含11个对象，这11个对象我们都可以直接使用！！！

|  |  |  |
| --- | --- | --- |
| **隐含对象** | **类型** | **说明** |
| pageContext | javax.servlet.jsp.PageContext | 就是JSP页面中的pageContext对象 |
| **pageScope** | java.util.Map<String,Object> | Page范围属性名和属性值 |
| **requestScope** | java.util.Map<String,Object> | Request范围属性名和属性值 |
| **sessionScope** | java.util.Map<String,Object> | Session范围属性名和属性值 |
| **applicationScope** | java.util.Map<String,Object> | Web应用范围属性名和属性值 |
| **param** | **java.util.Map<String,String>** | **对应一个请求参数** |
| paramValues | java.util.Map<String,String[]> | 对应一组请求参数 |
| header | java.util.Map<String,String> | 请求头 |
| headerValues | java.util.Map<String,String[]> | 请求头返回字符数组 |
| cookie | java.util.Map<String, Cookie> | 获取某个cookie对象 |
| initParam | java.util.Map<String, String> | 获取web.xml中<context-param>初始化参数 |

## 12. JSTL（JSP standard tag library）

所谓自定义标签库就是指可以在JSP页面中以类似于HTML标签的形式调用Java中的方法。使用方法和我们JSP动作标签类似。JSTL由5个标签库组成，但是最常用的是核心标签库。

使用：

①使用JSTL必须在项目中导入两个jar包：

* taglibs-standard-impl-1.2.1.jar
* taglibs-standard-spec-1.2.5.jar

②然后还需要在JSP页面中通过taglib标签引入标签库：

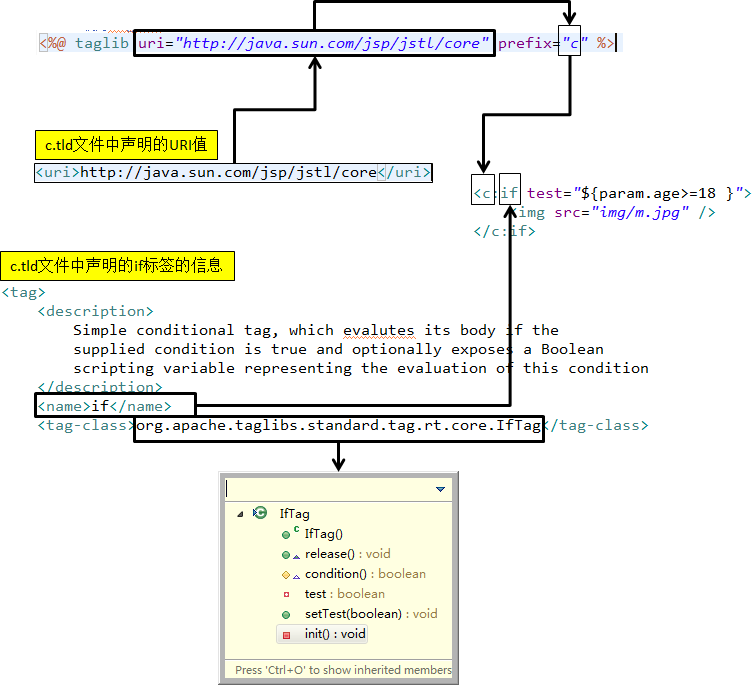
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

prefix用来指定前缀名(库名)，我们通过该名来使用JSTL，uri相当于库的唯一标识，因为JSTL由多个不同的库组成，使用该属性指定要导入哪个库。

③使用JSTL：<c:out value="hello"></c:out>

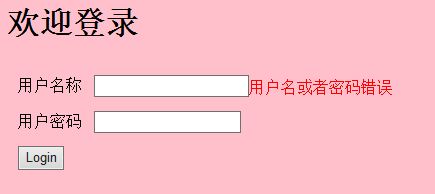
这个例子标识，调用前缀为c的标签的out方法，向页面中输出value属性中的字符串。JSTL的使用非常像html标签。JSTL一般结合EL表达式才能发挥最大的作用

原理：



## 13. 完成错误信息提示

页面中错误提示的功能效果：



思路：

①页面发起请求，将用户名和密码作为参数提交Servlet处理

②Servlet处理完成之后，根据用户名进行判断

③如果用户名错误，则将错误信息，封装到请求域中，转发到jsp页面

④在JSP页面使用EL表达式取出响应的值

## 14. JavaScript

### 14.1 JavaScript简介

在1995年时，由[Netscape](http://baike.baidu.com/view/153922.htm" \t "_blank)公司的[Brendan Eich](http://baike.baidu.com/view/2135520.htm" \t "_blank)，在[网景导航者](http://baike.baidu.com/view/1350596.htm" \t "_blank)浏览器上首次设计实现而成。[Netscape](http://baike.baidu.com/view/153922.htm" \t "_blank)在最初将其脚本语言命名为[LiveScript](http://baike.baidu.com/view/2373233.htm" \t "_blank)，因为[Netscape](http://baike.baidu.com/view/153922.htm" \t "_blank)与[Sun](http://baike.baidu.com/subview/24856/10322735.htm" \t "_blank)合作，网景公司管理层希望它外观看起来像[Java](http://baike.baidu.com/subview/29/12654100.htm" \t "_blank)，因此取名为JavaScript。

* 特性
* 脚本语言。JavaScript是一种解释型的脚本语言,C、C++、Java等语言先编译后执行, 而JavaScript是在程序的运行过程中逐行进行解释。
* 基于对象。JavaScript是一种基于对象的脚本语言,它不仅可以创建对象,也能使用现有的对象。
* 简单。JavaScript语言中采用的是弱类型的变量类型,对使用的数据类型未做出严格的要求,是基于Java基本语句和控制的脚本语言。
* 动态性。JavaScript是一种采用事件驱动的脚本语言,它不需要经过Web服务器就可以对用户的输入做出响应。
* 跨平台性。JavaScript脚本语言不依赖于操作系统,仅需要浏览器的支持。因此一个JavaScript脚本在编写后可以带到任意机器上使用,前提是机器上的浏览器支 持JavaScript脚本语言,目前JavaScript已被大多数的浏览器所支持。

### 14.2 使用位置

①只需要在head 标签中，或者在body标签中， 使用<script> 标签来书写JavaScript代码;

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Login JSP</title>

<script type=*"text/javascript"*>

**var** str="hello js";

alert(str);

</script>

</head>

②将js代码写在一个独立的文件中，使用<script>标签来引入

<script type=*"text/javascript"* src=*"my.js"*></script>

### 14.3 Js中的变量

|  |  |
| --- | --- |
| javaScript的变量类型 | |
| 数值类型 | Number(int short long double float) |
| 字符串类型 | String |
| 对象类型 | object |
| 布尔类型 | Boolean |
| 函数类型 | function |
| Js中的特殊值 | |
| Undefined | 未定义（所有变量未初始化时的初始值都是undefined） |
| Null | 空值 |
| NAN | Not a number 非数值 |

### 14.4 JS中的事件

|  |  |
| --- | --- |
| JavaScript的事件驱动 | |
| 用户事件 | 用户操作，例如单击、鼠标移入、鼠标移出等 |
| 系统事件 | 由系统触发的事件，例如文档加载完成 |
| 常见的事件 | |
| Onload | 加载完成事件 ,经常用于在页面加载完成之后。做一些初始化准备工作 |
| Onclick | 单击事件 ,常用于按钮点击后的响应 |
| Onblur | 常用于当输入框失去焦点后，可以验证输入框中的内容是否合法，或有效。 |
| onfocus | 聚焦后触发 |

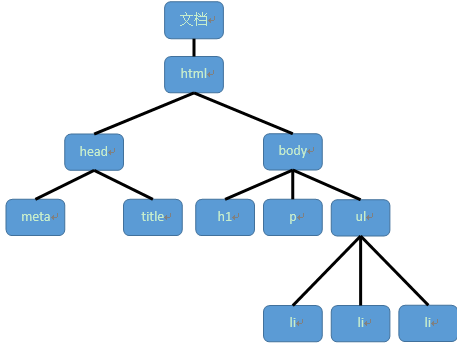
### 14.5 BOM和DOM

BOM （Browser Object Model） 浏览器对象模型。浏览器对象模型提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。BOM由多个对象组成，其中代表浏览器窗口的Window对象是BOM的顶层对象，其他对象都是该对象的子对象

常用的对象(window的子对象)：document、history 、location 、screen、navigator 、frames

DOM（Document Object Model） 文档对象模型。document对象是window对象的一个属性，代表当前HTML文档，包含了整个文档的树形结构。获取document对象的本质方法是：window.document，而“window.”可以省略。

DOM树



|  |  |  |
| --- | --- | --- |
| **DOM元素查询** | | |
| **功能** | **API** | **返回值** |
| **根据id值查询** | document.getElementById(“id值”) | 一个具体的元素节点 |
| **根据标签名查询** | document.getElementsByTagName(“标签名”) | 元素节点数组 |
| **根据name属性值查询** | document.getElementsByName(“name值”) | 元素节点数组 |

## 15.完成错误消息提示

<script type=*"text/javascript"*>

**function** clearSpan(){

**var** es=document.getElementById("errorspan");

es.innerHTML="";

}

</script>

用户名称：<input name=*"username"* value=*""* type=*"text"* onchange=*"clearSpan()"*>

<span id=*"errorspan"*>${requestScope.errorinfo}</span>

# 三、用户校验功能的实现

## 1. Ajax

### 1.1 Ajax简介

AJAX 是 Asynchronous JavaScript And XML 的简称。直译为，异步的JS和XML。

AJAX的实际意义是，不发生页面跳转、异步载入内容并改写页面内容的技术。

AJAX也可以简单的理解为通过JS向服务器发送请求。

同步处理和异步处理

* 同步处理

AJAX出现之前，我们访问互联网时一般都是同步请求，也就是当我们通过一个页面向服务器发送一个请求时，在服务器响应结束之前，我们的整个页面是不能操作的，也就是直观上来看他是卡主不动的。

这就带来了非常糟糕的用户体验。首先，同步请求时，用户只能等待服务器的响应，而不能做任何操作。其次，如果请求时间过长可能会给用户一个卡死的感觉。最后，同步请求的最大缺点就是即使整个页面中只有一小部分内容发生改变我们也要刷新整个页面。

* 异步处理

而异步处理指的是我们在浏览网页的同时，通过AJAX向服务器发送请求，发送请求的过程中我们浏览网页的行为并不会收到任何影响，甚至主观上感知不到在向服务器发送请求。当服务器正常响应请求后，响应信息会直接发送到AJAX中，AJAX可以根据服务器响应的内容做一些操作。

使用AJAX的异步请求基本上完美的解决了同步请求带来的问题。首先，发送请求时不会影响到用户的正常访问。其次，即使请求时间过长，用户不会有任何感知。最后，AJAX可以根据服务器的响应信息局部的修改页面，而不需要整个页面刷新。

### 1.2 异步请求对象

异步请求对象:

XMLHttpRequest对象是AJAX中非常重要的对象，所有的AJAX操作都是基于该对象的。

XMLHttpRequest对象用来封装请求报文，我们向服务器发送的请求信息全部都需要封装到该对象中。

这里需要稍微注意一下，XMLHttpRequest对象并没有成为标准，但是现在的主流浏览器都支持该对象，而一些如IE6的老版本浏览器中的创建方式有一些区别，但是问题不大。

### 1.3 使用Ajax

①创建



②请求



③响应



④onReadyStateChange



## 2. 使用Ajax判断用户名是否存在

**function** sendAjax(){

**var** param=document.getElementById("username").value;

//创建异步请求对象

**var** request=**new** XMLHttpRequest();

//封装异步请求参数

request.open("GET","validLogin?username="+param,**true**);

//发送异步请求

request.send()

//如需获得来自服务器的响应，请使用 XMLHttpRequest 对象的 responseText

request.onreadystatechange=**function**(){

**if**(request.readyState==4 && request.status==200){

document.getElementById("errorspan").innerHTML=request.responseText;

}

}}

用户名称：<input id=*"username"* name=*"username"* value=*""* type=*"text"* onchange=*"clearSpan()"* onblur="sendAjax()"><span id=*"errorspan"*>${requestScope.errorinfo}</span>

服务端：

①DAO:

@Override

**public** **boolean** validLogin(String username) {

**try** {

Connection connection = ConnectionUtil.*getConnection*();

String sql="select id username from tbl\_user where username=? ";

PreparedStatement ps=connection.prepareStatement(sql);

ps.setString(1, username);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

**return** **true**;

}

**return** **false**;

} **catch** (SQLException e) {

e.printStackTrace();

}**finally** {

**try** {

ConnectionUtil.*closeConnection*();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**return** **false**;

}

②Servlet:

**public** **class** ValidLoginServlet **extends** HttpServlet {

**private** UserDao ud;

**public** ValidLoginServlet() {

ud=**new** UserDaoImpl();

}

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

String username = request.getParameter("username");

System.***out***.println(username);

**boolean** validResult = ud.validLogin(username);

response.setContentType("text/html;charset=utf-8");

**if** (validResult) {

response.getWriter().print("用户已经存在！");

}**else** {

response.getWriter().print("用户名可以使用！");

}

}

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

doGet(request, response);

}

}

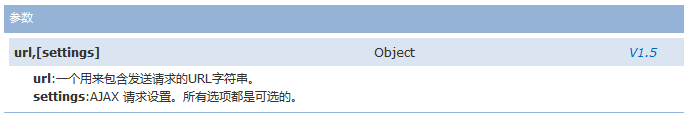
## 3. 使用Jquery框架来发送异步请求

JQuery是当前比较主流的 JavaScript 库，封装了很多预定义的对象和实现函数，帮助使用者建立有高难度交互的页面，并且兼容大部分主流的浏览器。

JQuery对同样提供了对Ajax的支持，可以更加方便快速的进行Ajax的开发，相关的方法有$.get $.post $.ajax等。

$.ajax方法

* jQuery 底层 AJAX 实现。简单易用的高层实现见 $.get, $.post 等。$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该函数，除非你需要操作不常用的选项，以获得更多的灵活性。最简单的情况下，$.ajax()可以不带任何参数直接使用。
* $.ajax方法的参数



对于settings请求设置来说，所有选项都是可选的，详见jQuery手册

示例：

$(**function**(){

$("#username").blur(**function**(){

**var** data="username="+$("#username").val();

$.ajax({

type:"GET",

url:"validLogin",

data:data,

success:**function**(data){

$("#errorspan").html(data);

}

});

});

});

# 四、登录功能的实现

## 1. 准备工作



需要用到的数据：

CREATE TABLE tbl\_employee(

id INT(11) PRIMARY KEY AUTO\_INCREMENT,

last\_name VARCHAR(255),

gender VARCHAR(10),

email VARCHAR(255)

);

INSERT INTO tbl\_employee(last\_name,gender,email) VALUES('Tom','male','Tom@163.com');

INSERT INTO tbl\_employee(last\_name,gender,email) VALUES('Jack','male','Jack@163.com');

INSERT INTO tbl\_employee(last\_name,gender,email) VALUES('Marry','female','Tom@163.com');

分析：①页面发起请求，Servlet处理请求，调用Dao查出数据

②将数据放入请求域中

③转发请求到指定页面

④在指定页面遍历数据，放入表格

## 2. 使用JSTL遍历数据

表格样式：

<table border="0" cellpadding="3" cellspacing="1" bgcolor="#c1c1c1" style="text-align:center">

<tr bgcolor="#FFFFFF">

<td>ID</td>

<td>LASTNAME</td>

<td>GENDER</td>

<td>EMAIL</td>

</tr>

</table>

使用JSP脚本片段和表达式

<h1 align=*"center"*>员工信息列表</h1><br>

<table border=*"0"* cellpadding=*"3"* cellspacing=*"1"* bgcolor=*"#c1c1c1"* style="text-align:*center*" align=*"center"*>

<tr bgcolor=*"#FFFFFF"*>

<td>ID</td>

<td>LASTNAME</td>

<td>GENDER</td>

<td>EMAIL</td>

</tr>

<% List<Employee> emps=(List<Employee>)request.getAttribute("emps");

**for**(Employee e:emps){ %>

<tr bgcolor=*"#FFFF00"*>

<td><%=e.getId() %></td>

<td><%=e.getLastName() %></td>

<td><%=e.getGender()%></td>

<td><%=e.getEmail() %></td>

</tr>

<%

}

%>

</table>

<table border=*"0"* cellpadding=*"3"* cellspacing=*"1"* bgcolor=*"#c1c1c1"* style="text-align:*center*" align=*"center"*>

<tr bgcolor=*"#FFFFFF"*>

<td>ID</td>

<td>LASTNAME</td>

<td>GENDER</td>

<td>EMAIL</td>

</tr>

<c:forEach var=*"e"* items=*"*${emps}*"*>

<tr bgcolor=*"#FFFF00"*>

<td>${e.id}</td>

<td>${e.lastName }</td>

<td>${e.gender }</td>

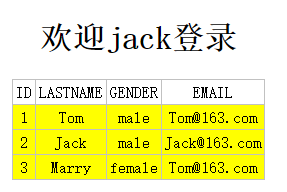
<td>${e.email }</td>

</tr>

</c:forEach>

</table>

结果：



## 3.Cookie

### 3.1 简介

* Cookie
* HTTP是无状态协议，服务器不能记录浏览器的访问状态，也就是说服务器不能区分中两次请求是否由一个客户端发出。这样的设计严重阻碍的Web程序的设计。如：在我们进行网购时，买了一条裤子，又买了一个手机。由于http协议是无状态的，如果不通过其他手段，服务器是不能知道用户到底买了什么。而Cookie就是解决方案之一。
* Cookie实际上就是服务器保存在浏览器上的一段信息。浏览器有了Cookie之后，每次向服务器发送请求时都会同时将该信息发送给服务器，服务器收到请求后，就可以根据该信息处理请求。
* Cookie的用途

网上商城购物车

用户登录状态的保持

* Cookie的限制性
  + Cookie最为请求或响应报文发送，无形中增加了网络流量。
  + Cookie是明文传送的安全性差。
  + 各个浏览器对Cookie有限制，使用上有局限

### 3.2 使用

①创建：

Cookie cookie = **new** Cookie("cookie-name", "cookie-Value");

②设置浏览器保存Cookie

response.addCookie(cookie);

③获取Cookie

// 获取所有cookie对象

Cookie[] cookies = request.getCookies();

// 如果没有cookie，则返回null。

**if** (cookies != **null**) {

// 有cookie则遍历

**for** (Cookie cookie : cookies) {

response.getWriter().write("Cookie名：" + cookie.getName()

+ "<br/>Cookie值：" + cookie.getValue() + "<br/><br/>");

}}

④修改：

// 创建一个已存在key的Cookie对象

Cookie cookie = **new** Cookie("cookie-name", **null**);

// 修改Cookie的值

cookie.setValue("this is new value");

// 通知浏览器保存修改

response.addCookie(cookie);

response.getWriter().write("Cookie…已修改值");}

⑤设置有效时间

cookie.setMaxAge(60 \* 60 \* 24 \* 7);//一周

setMaxAge()如果是负数，就表示浏览器关闭，就会过期；0则是，收到响应，就过期。正数表示多少秒后过期。

Cookie默认的有效时间，就是浏览器一关闭，Cookie就过期！

## 4. Session

### 4.1 Session的简介

Session是JSP中九大内置对象之一；

Session是一个域对象；

Session是服务器端用来保存用户数据的一种技术，且基于Cookie实现。

使用Cookie有一个非常大的局限，就是Cookie中保存的信息是有限的（4KB），如过Cookie过多，则无形中增加了客户端与服务器端的数据传输量。

Session代表客户端和服务器的一次会话。通过request.getSession()可以创建或获取当前的session对象。

第一次访问是创建Session，之后都是获取Session。

Session被创建后，同时还会有一个名为JSESSIONID的Cookie被创建, 这个Cookie的默认时效就是当前会话。

之后浏览器每次访问项目时都会携带该Cookie。当我们再次调用时会根据该JSESSIONID获取已经存在的Cookie，而不是在创建一个新的Cookie。如果Cookie中有JSESSIONID，但是JSESSIONID没有对应的Session存在，则会重新创建一个HttpSession对象，并重新设置JSESSIONID。

### 4.2 Session的有效时间

Session在浏览器中的失效，可以在全局的web.xml中设置，默认是30分钟！其有效时间指的是，最后一次访问的时间开始统计，而不是从创建的时间来统计。

  <!-- ==================== Default Session Configuration ================= -->

  <!-- You can set the default session timeout (in minutes) for all newly   -->

  <!-- created sessions by modifying the value below.                       -->

    <session-config>

        <session-timeout>30</session-timeout>

    </session-config>

## 5. 免输用户名登录功能的实现

需求：用户第一次登录时，需要输入用户名和密码；之后在1天内的每一次登录，都无需再输入用户名。

技术：使用Cookie来解决。

①在登录的LoginServlet中，判断，如果用户输入了用户名，就将用户名放入新创建的Cookie中，响应给客户端

//在第一次获取请求参数之前，设置请求的编码

req.setCharacterEncoding("utf-8");

String username = req.getParameter("username");

String password = req.getParameter("password");

User user = ud.findUserByUserNameAndPassword(username, password);

**if** (user==**null**) {

req.setAttribute("errorinfo", "请输入用户名！");

req.getRequestDispatcher("/index.jsp").forward(req, resp);

}**else** {

Cookie cookie = **new** Cookie("username", username);

cookie.setMaxAge(60\*60\*24);

resp.addCookie(cookie);

List<Employee> emps = ed.getAllEmployees();

req.setAttribute("emps", emps);

req.getRequestDispatcher("/WEB-INF/pages/list.jsp").forward(req, resp);

}

②在登录页面使用EL表达式的内置Cookie对象来取出指定cookie的值

<form id=*"form"* action=*"login"* method=*"post"*>

用户名称：<input id=*"username"* name=*"username"* value=*"*${cookie.username.value}*"* type=*"text"* onchange=*"clearSpan()"* >

<span id=*"errorspan"*>${requestScope.errorinfo}</span>

<br>

用户密码：<input name=*"password"* value=*"123456"* type=*"password"*>

<br>

<input type=*"submit"* value=*"提交"*>

</form>

## 6. 使用Session实现全局登录机制

需求： 如果用户已经登录，在首页则显示登录后的页面；如果用户没有登录，则显示登录页面！

①LoginServlet:

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

System.***out***.println("接受到了页面发来的请求！");

//在第一次获取请求参数之前，设置请求的编码

req.setCharacterEncoding("utf-8");

String username = req.getParameter("username");

String password = req.getParameter("password");

User user = ud.findUserByUserNameAndPassword(username, password);

**if** (user==**null**) {

req.setAttribute("errorinfo", "请输入用户名！");

req.getRequestDispatcher("/index.jsp").forward(req, resp);

}**else** {

Cookie cookie = **new** Cookie("username", username);

cookie.setMaxAge(60\*60\*24);

resp.addCookie(cookie);

//将用户名放入session

HttpSession session = req.getSession();

session.setAttribute("user", username);

List<Employee> emps = ed.getAllEmployees();

req.setAttribute("emps", emps);

req.getRequestDispatcher("/pages/list.jsp").forward(req, resp);

}

}

②index.jsp页面：

<h1>欢迎${sessionScope.user}登录</h1>

<c:if test=*"*${**empty** sessionScope.user}*"*>

<form id=*"form"* action=*"login"* method=*"post"*>

用户名称：<input id=*"username"* name=*"username"* value=*"*${cookie.username.value}*"* type=*"text"* onchange=*"clearSpan()"* >

<span id=*"errorspan"*>${requestScope.errorinfo}</span>

<br>

用户密码：<input name=*"password"* value=*"123456"* type=*"password"*>

<br>

<input type=*"submit"* value=*"提交"*>

</form>

</c:if>

<c:if test=*"*${**not empty** sessionScope.user}*"*>

<a href=*"showAllEmps"*>查看所有员工</a>

</c:if>

③点击超链接查看员工的Servlet处理：

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

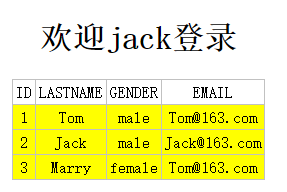
List<Employee> emps = ed.getAllEmployees()

request.setAttribute("emps", emps);

request.getRequestDispatcher("/pages/list.jsp").forward(request, response);

}

效果：



# 五、主页面访问权限控制

## 1. Filter

### 1.1 Filter简介

服务器端负责**拦截**和**过滤**请求的组件，它可以根据一定的**预设的条件**，将请求**放行**到目标资源，或者**重新派发**到其他位置。

实现一个Filter需要实现Filter接口。

### 1.2 Filter使用

创建：自定义类，实现Filter接口。

配置拦截：在web.xml中配置

<filter>

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

<filter-class>com.atguigu.web.filter.LoginFilter</filter-class>

</filter>

<filter-mapping>

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

<url-pattern>/login</url-pattern>

</filter-mapping>

预设条件： 在doFilter中，自己设定判断条件；

符合条件放行: chain.doFilter(request,response);

### 1.3 Filter的生命周期

 ①创建对象：**Web应用加载**时就创建对象——这一点和Servlet不同！

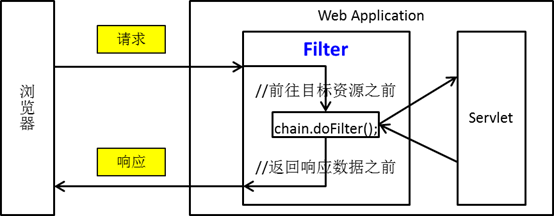
 ②初始化：在创建对象之后，立即执行——init()

 ③拦截到浏览器请求：调用doFilter()方法

          每一次拦截请求都没有重新创建对象，说明Filter也是**单实例**，**多线程**运行的。

 ④清理工作：在卸载Web应用之前，释放Filter对象的时候执行——destroy()

### 1.4 运行流程



### 1.5 Filter链

 ①当多个Filter拦截同一个资源时，参与的所有Filter就会形成一个Filter链

 ②Filter链中每个Filter调用的顺序是根据filter-mapping元素配置的顺序决定的

 ③FilterChain接口

     [1]代表：Filter链中，当前Filter对象所在的位置

     [2]对象获取：Servlet容器创建，通过doFilter()方法以参数的形式传入进来

     [3]功能：调用chain.doFilter()方法实现

         (1)调用当前Filter链中的下一个Filter

         (2)如果当前Filter就是Filter链中的最后一个，那么就会将请求放行到目标资源

## 2. 页面权限控制

问题： 如果直接访问员工列表展示页面，则会有如下效果：



解决：通过Filter来拦截请求，如果直接访问，则重定向到登录页面。

@Override

**public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

**throws** IOException, ServletException {

HttpServletRequest req=(HttpServletRequest)request;

HttpServletResponse resp=(HttpServletResponse) response;

String requestURL = req.getRequestURL().toString();

**if** (requestURL.endsWith("list.jsp")) {

System.***out***.println("拦截直接访问的请求！");

resp.sendRedirect("/JavaWeb");

}**else** {

chain.doFilter(request, response);

}

}

## 3. Listener

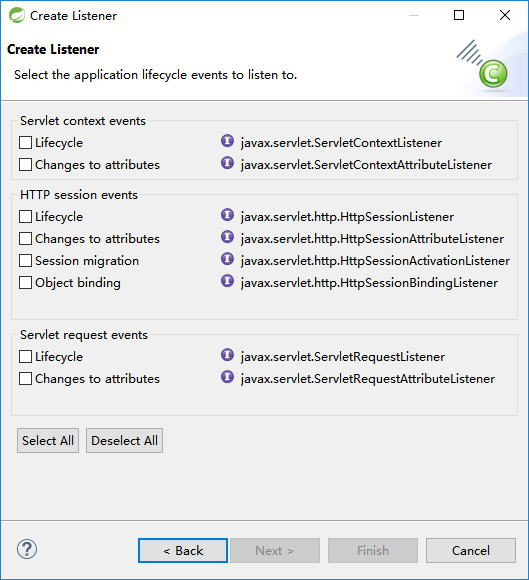
监听器就是实时监视一些事物状态的程序，我们称为监听器。

Listener(监听器)和Servlet和Filter一样，都是javaweb的三大组件之一。

JavaEE中的监听器，按作用分类，有三大类：

|  |  |
| --- | --- |
| 生命周期监听器 | |
| javax.servlet.ServletContextListener |  |
| javax.servlet.ServletRequestListener |  |
| javax.servlet.http.HttpSessionListener |  |
| 属性监听器 | |
| javax.servlet.ServletContextAttributeListener | ServletContext属性监听器 |
| javax.servlet.http.HttpSessionAttributeListener | HttpSession属性监听器 |
| javax.servlet.ServletRequestAttributeListener | Request属性监听器 |
| 对象监听器 | |
| javax.servlet.http.HttpSessionActivationListener | 保存到Session域中的对象，活化和钝化的监听器 |
| javax.servlet.http.HttpSessionBindingListener | 对象被绑定到Session域中的监听器 |

需要实现什么样的功能，就实现什么样的监听器接口。然后在web.xml中进行配置！



## 4. 在线人数统计

①每发起一次会话，记录用户数为1

②登录的用户数量，需要共享给所有的用户，即放入application域中！

③创建Listenner，监听session的生命周期，每创建一个session，统计数字自增1

④在页面上使用EL取出指定的登录数

①创建Listenner，监听session的生命周期



并在web.xml中注册：

<listener>

<listener-class>com.atguigu.web.listener.MyListener</listener-class>

</listener>

②当session创建时，记录用户数加1

**public** **class** MyListener **implements** HttpSessionListener {

**public** MyListener() {

}

**public** **void** sessionCreated(HttpSessionEvent sessionEvent) {

ServletContext sc = sessionEvent.getSession().getServletContext();

Object count = sc.getAttribute("count");

**if** (count==**null**) {

sc.setAttribute("count", 1);

}**else** {

sc.setAttribute("count", (Integer)count+1);

}

}

**public** **void** sessionDestroyed(HttpSessionEvent sessionEvent) {

ServletContext sc = sessionEvent.getSession().getServletContext();

Object count = sc.getAttribute("count");

sc.setAttribute("count", (Integer)count-1);

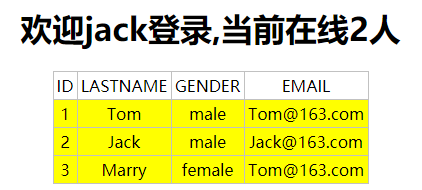
}

}

③在页面取出count

<h1 align="center">欢迎${sessionScope.user}登录,当前在线${applicationScope.count}人</h1>

在线人数统计功能展示



# 六、XML和JSON

## 1. XML

### 1.1 简介

XML称为可扩展的标记性语言。

作用：

* 用来保存数据，并且这些数据内容具有自我描述性。
* xml用作工程或者框架的配置文件。
* 作为数据传输的一种数据格式。

### 1.2 xml语法

①文档声明



文档声明是一个XML文件必不可少的部分，主要声明了使用的版本和编码。

②注释：<! - -xxxx - ->

③XML标签的语法规则

* + 标签名称可以含有字母、数字和其他字符，但是不能以数字和标点符号开头；
  + 标签名称中不能有空格；
  + 标签必须正确结束；
  + 标签大小写敏感，且必须正确闭合，正确嵌套！
  + 标签头中的属性值，必须使用””引起来；

示例：

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

<!-- xml声明，其中version为版本号，encoding为编码 -->

<books>

<!-- 在标签头上可以声明属性，属性必须使用""引起来 -->

<book id=*"001"*>

<name>《西游记》</name>

<!-- XML中如果出现特殊字符需要使用转义或使用CDATA文本区进行包装

使用转义字符：如 < : &lt;

> : &gt;

使用CDATA：

-->

<author><![CDATA[<吴承恩>]]></author>

<price>88.88</price>

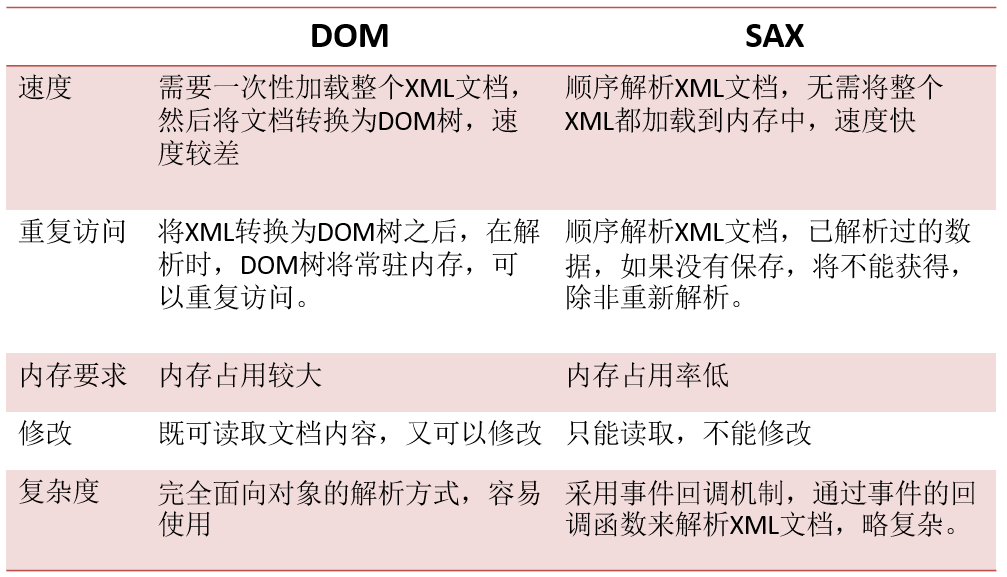
</book>

</books>

### 1.3 XML文档的解析

可以使用DOM（Document Object Model）或SAX（Simple API for XML）这两种技术，将XML文件中的内容转换为Java对象。

DOM和SAX的对比：



Dom4J的解析示例：

## 2. JSON

### 2.1. 简介

JSON是一种轻量级的数据交换格式。

Json定义格式：

var 变量名 = {

“key” : value , // Number类型

“key2” : “value” , // 字符串类型

“key3” : [] , // 数组类型

“key4” : {}, // json 对象类型

“key5” : [{},{}] // json 数组

};

Json对象的访问：Json对象中，如果要访问指定的属性，只需要按照 【对象名.属性名】的格式就可以访问。

### 2. 2 JSON对象和JSON字符串的转换

JSON.stringify(jsonObject): 此方法可以把一个json对象转换成为json字符串；

JSON.parse(jsonString); 此方法可以把一个json字符串转换为json对象。

### 2.3. 在java中使用JSON

#### 2.3.1 java对象和json的转换

①Java对象转JSON字符串

Person person = **new** Person("Jack", "男", 11);

//使用一个Gson对象转换

Gson gson = **new** Gson();

//转换Java对象为Json字符串

String jsonStr = gson.toJson(person);

//{"name":"Jack","gender":"男","age":11}

②JSON字符串转Java对象

//使用一个Gson对象转换

Gson gson = **new** Gson();

//转换Java对象为Json字符串

String jsonStr ="{\"name\":\"Jack\",\"gender\":\"男\",\"age\":11}" ;

Person person = gson.fromJson(jsonStr, Person.**class**);

//Person [name=Jack, gender=男, age=11]

#### 2.3.2 java对象list集合和json的转换

①List集合转Json字符串

//使用一个Gson对象转换

Gson gson = **new** Gson();

List<Person> persons=**new** ArrayList<>();

persons.add( **new** Person("Jack1", "男", 11));

persons.add( **new** Person("Jack2", "男", 11));

persons.add( **new** Person("Jack3", "男", 11));

/\*

\* [{"name":"Jack1","gender":"男","age":11},

\* {"name":"Jack2","gender":"男","age":11},

\* {"name":"Jack3","gender":"男","age":11}]

\*/

String jsonStr = gson.toJson(persons);

②Json字符串转List集合

Gson gson = **new** Gson();

String jsonStr = "[{\"name\":\"Jack1\",\"gender\":\"男\",\"age\":11},{\"name\":\"Jack2\",\"gender\":\"男\",\"age\":11},{\"name\":\"Jack3\",\"gender\":\"男\",\"age\":11}]";

List<Person> persons = gson.fromJson(jsonStr, **new** TypeToken<ArrayList<Person>>() {}.getType());

#### 2.3.3 java对象map集合和json的转换

①Map转Json字符串

Gson gson = **new** Gson();

Map<String, Person> map=**new** HashMap<>();

map.put("person1", **new** Person("Jack1", "男", 11));

map.put("person2", **new** Person("Jack2", "男", 11));

map.put("person3", **new** Person("Jack3", "男", 11));

String jsonStr = gson.toJson(map);

②Json字符串转Map

Map<String, Person> map1 = gson.fromJson(jsonStr, new TypeToken<HashMap<String, Person>>(){}.getType());

### 2.4. Ajax请求Json

①页面

<a id=*"getJson"* href=*"GetJsonServlet"*>Ajax获取Json</a>

<table id=*"table"* border=*"0"* cellpadding=*"3"* cellspacing=*"1"* bgcolor=*"#c1c1c1"* style="text-align:*center*" align=*"center"*>

</table>

②后台处理

**public** **class** HandleJson **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** EmployeeDao ed;

**public** HandleJson() {

ed=**new** EmployeeDaoImpl();

}

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

List<Employee> allEmployees = ed.getAllEmployees();

Gson gson = **new** Gson();

String json = gson.toJson(allEmployees);

response.setContentType("text/html,charset=utf-8");

response.getWriter().print(json);

}

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

doGet(request, response);

}

}

③前端处理

$(**function**(){

$("#getJson").click(**function**(){

$.ajax({

url:"GetJsonServlet",

type:"GET",

//设置dataType为json，会自动转换为JSON对象

dataType:"json",

success:**function**(data){

alert(data);

**var** str="<tr bgcolor='#FFFF00'><td>ID</td><td>LASTNAME</td><td>GENDER</td><td>EMAIL</td></tr>";

**for**(**var** i=0;i<data.length;i++){

**var** emp=data[i];

str+="<tr><td>"+emp.id+"</td><td>"+emp.lastName+"</td><td>"+emp.gender+"</td><td>"+emp.email+"</td></tr> ";

}

$("#table").html(str);

}

});

**return** **false**;

});

});