**Java WEB**

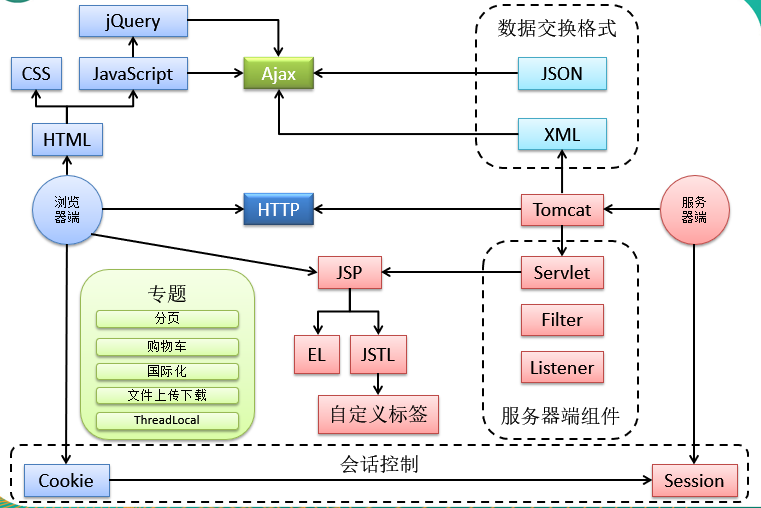
尚硅谷java研究院

版本：V 1.0

# 第1章 简介

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。

# 第2章 JavaWeb的技术体系



用户浏览器输入地址，根据地址找web.xml中的对应的servlet，找到处理。找不到找页面。再找不到404。

# 第3章 登录页面的开发

## 3.1 涉及的技术知识点

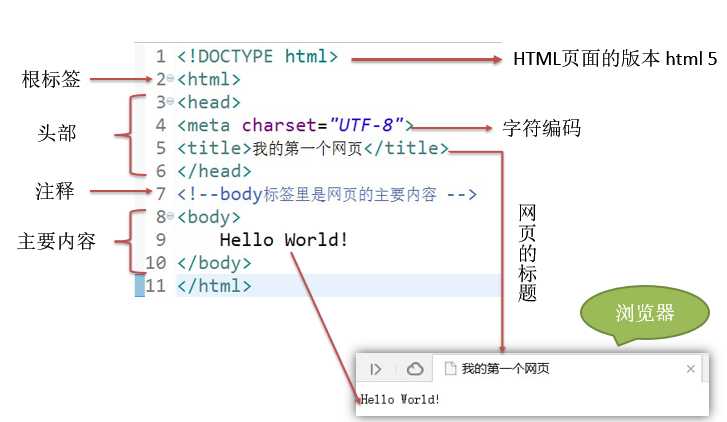
1. HTML、CSS

## 3.2 HTML是什么?

1. 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。
2. HTML网页的组成:



1. 常用的HTML标签
2. html 根标记
3. head 头标记
4. body 体标记
5. h 标题标签
6. a 超链接
7. base标签
8. table 表格
9. **form 表单**
10. 一个基本结构的HTML页面



1. 登录页面的示例



# 第4章 登录功能实现-环境的搭建

## 4.1 涉及的技术知识点

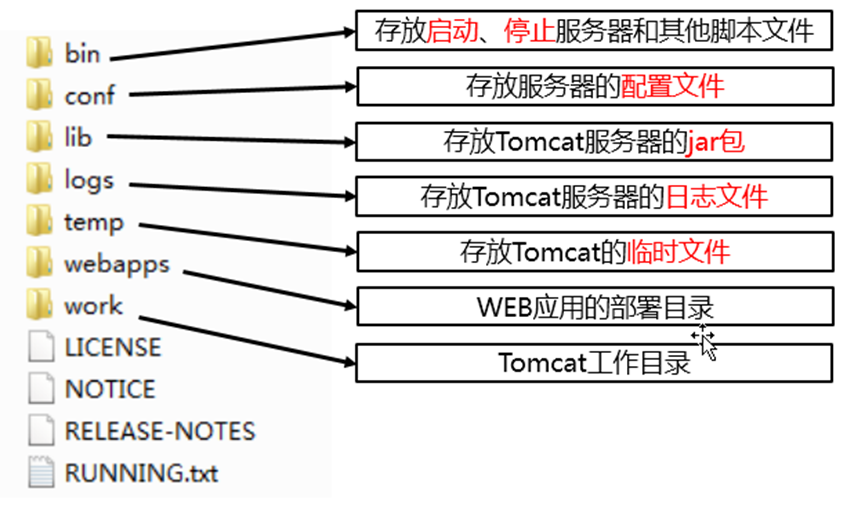
1. WEB服务器
2. 动态的web工程

## 4.2 Web服务器

1. 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。
2. **Tomcat（Apache）**：当前应用最广的JavaWeb服务器；
3. JBoss（Redhat红帽）：支持JavaEE，应用比较广EJB容器 –> SSH轻量级的框架代替
4. GlassFish（Orcale）：Oracle开发JavaWeb服务器，应用不是很广；
5. Resin（Caucho）：支持JavaEE，应用越来越广；
6. Weblogic（Orcale）：要钱的！支持JavaEE，适合大型项目；
7. Websphere（IBM）：要钱的！支持JavaEE，适合大型项目

## 4.3 Tomcat服务器的安装及配置

1. 将Tomcat的安装包解压到磁盘的任意位置(非中文无空格)
2. Tomcat服务的目录结构



1. 配置环境变量，方便Tomcat的启动关闭（**可选**）
2. 新建环境变量CATALINA\_HOME=解压目录

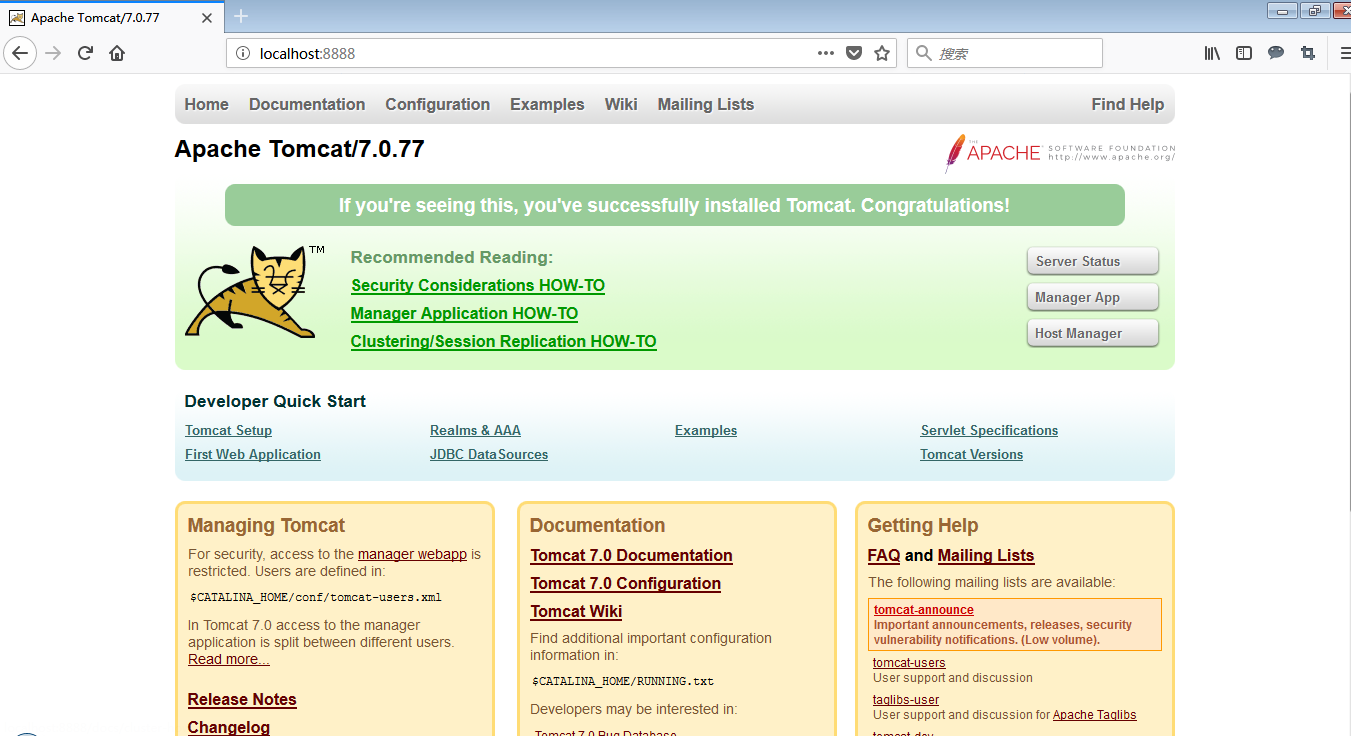


1. 在Path环境变量中加入Tomcat解压目录\bin目录



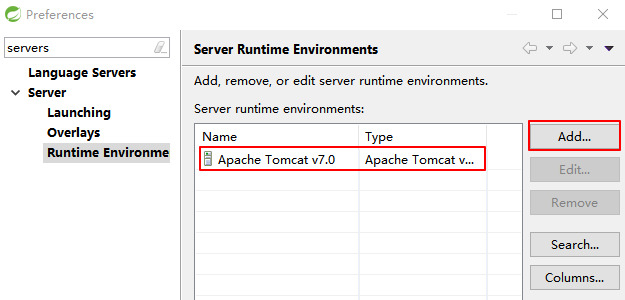
1. 在命令行中运行catalina run或者 startup启动Tomcat服务器，在浏览器地址栏访问如下地址进行测试

http://localhost:8080

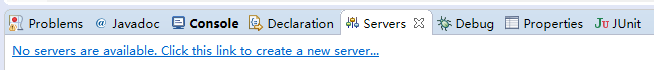


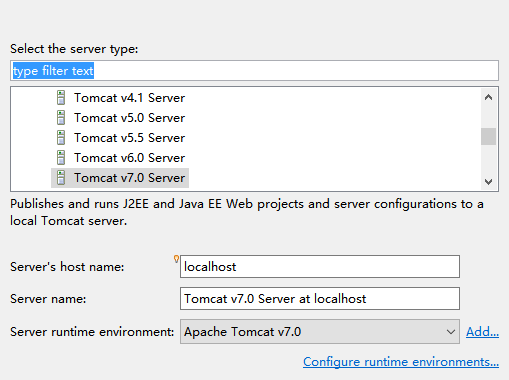
## 4.4 在eclipse中配置tomcat

1. 在Eclipse中配置运行环境



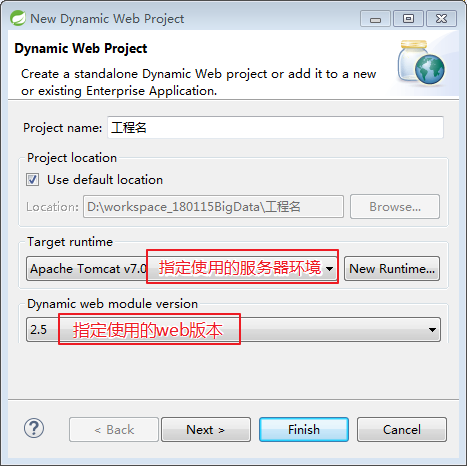
1. 在Eclipse中创建新的Server

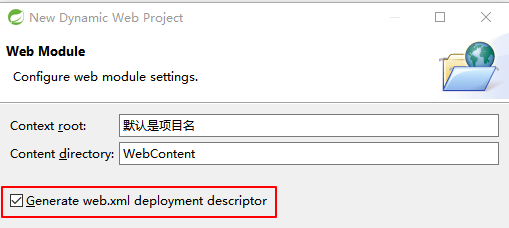




## 4.5 创建动态的web工程

1. 在Eclipse中点击File ->New->Dynamic Web Project





# 第5章 登录功能实现-LoginServlet

## 5.1 涉及的技术知识点

1. Servlet
2. Request请求对象
3. Response响应对象
4. 转发
5. 重定向

## 5.2 什么是Servlet?

1. 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。
2. 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。
3. 简单可以理解为 **Servlet就是用来处理客户端的请求的.**

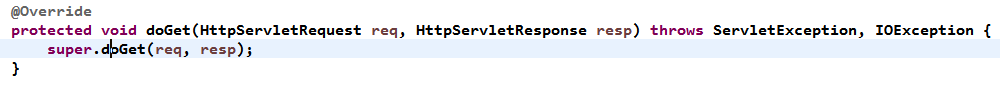
## 5.3 Servlet开发规则

1. 实际编码通过继承HttpServlet来完成Servlet的开发

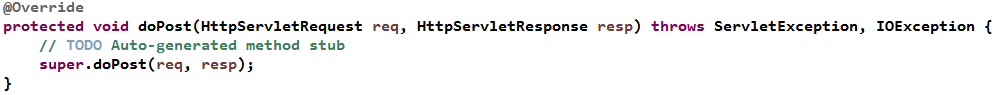


## 5.4 Servlet类的相关方法:

1. doGet Servlet中用于处理get请求的方法

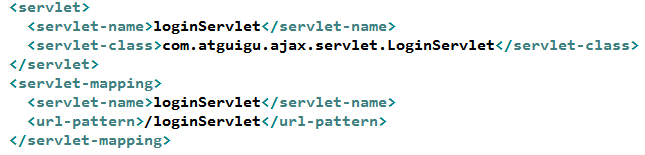


1. doPost Servlet中用于处理post请求的方法



1. service
2. 在Servlet的顶层实现中，在service方法中调用的具体的doGet或者是doPost
3. 在实际开发Servlet的过程中，可以选择重写doGet以及doPost 或者 直接重写service方法来处理请求。

## 5.5 Servlet在web.xml中的配置



## 5.6 获取请求参数值

1. HttpServletRequest
2. 该接口是ServletRequest接口的子接口，封装了HTTP请求的相关信息，由Servlet容器创建其实现类对象并传入service(ServletRequest req, ServletResponse res)方法中。以下我们所说的HttpServletRequest对象指的是容器提供的HttpServletRequest实现类对象。
3. HttpServletRequest对象的主要功能有

获取请求参数

获取项目虚拟路径

将请求转发给另外一个URL地址 [转发]

## 5.7 响应结果

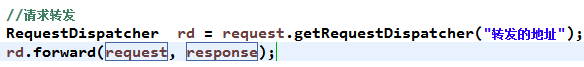
1. HttpServletResponse
2. 该接口是ServletResponse接口的子接口，封装了HTTP响应的相关信息，由Servlet容器创建其实现类对象并传入service(ServletRequest req, ServletResponse res)方法中。以下我们所说的HttpServletResponse对象指的是容器提供的HttpServletResponse实现类对象
3. 主要功能

使用PrintWriter对象向浏览器输出数据

实现请求的重定向[重定向]

## 5.8 请求转发

1. Servlet接收到浏览器端请求后，进行一定的处理，先不进行响应，而是在服务器端内部“转发”给其他Servlet程序继续处理。在这种情况下浏览器端只发出了一次请求，浏览器地址栏不会发生变化，用户也感知不到请求被转发了。
2. 转发请求的Servlet和目标Servlet共享同一个request对象。
3. 实现转发的API



## 5.9 请求重定向 redirect

1. Servlet接收到浏览器端请求并处理完成后，给浏览器端一个特殊的响应，这个特殊的响应要求浏览器去请求一个新的资源，整个过程中浏览器端会发出两次请求，且浏览器地址栏会改变为新资源的地址。
2. 重定向的情况下，原Servlet和目标资源之间就不能共享请求域数据了
3. 实现重定向的API



## 5.10 重定向与转发的区别

|  |  |  |
| --- | --- | --- |
|  | 转发 | 重定向 |
| 浏览器地址栏 | 不改变 | 改变 |
| 发送请求次数 | 1 | 2 |
| 能否共享request对象数据 | 能 | 否 |
| 目标资源：WEB-INF下的资源 | 能访问 | 不能访问 |

## 5.11 中文乱码的解决方式

请求：get (1) 修改tomcat 中 server.xml 中

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

（2）username = new String(username.getBytes("iso8859-1"),"utf-8");

先用iso8859-1编码，然后使用utf-8解码

Post 请求

request.setCharacterEncoding("UTF-8");

响应乱码：

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

## 5.12 相对路径与绝对路径

<!-- 使用base标签后 该页面中所有访问的地址前都会加上 href中的内容 -->

<base href="http://localhost:8080/05\_WEB\_Servlet\_login/">

</head>

<body>

<!--

解决路径问题：使用绝对路径 （访问地址的完整路径- http://localhost:8080/项目名/页面、Servlet）

1.使用base标签

<base href="http://localhost:8080/05\_WEB\_Servlet\_login/">

2.直接在访问的地址前面加 "/" - 表示使用的就是绝对路径

注意：

如果是服务器解析 "/"(路径的最前面加的) ： http://localhost:8080/项目名/

①web.xml中的url-pattern

②转发

如果是浏览器解析"/"(路径的最前面加的) : http://localhost:8080/

①重定向

②页面中的标签：超连接<a href=""> , <form action="">等。

-->

<!-- 浏览器不能直接访问web-inf下面的内容 转发可以 -->

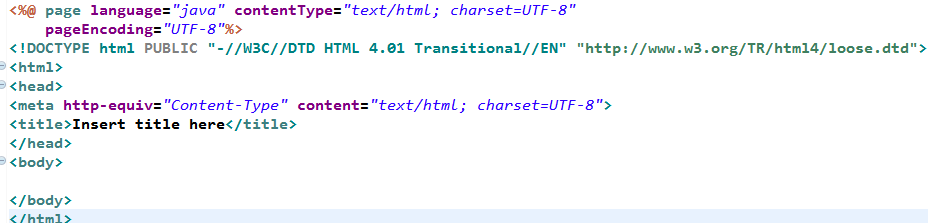
# 第6章 登录功能实现-页面中错误提示

## 6.1 涉及的技术知识点

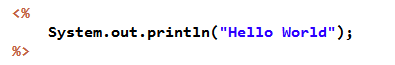
1. Jsp页面
2. EL表达式
3. JS简单应用

## 6.5 JSP页面

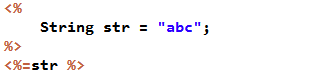
1. JSP全称Java Server Pages，顾名思义就是运行在java服务器中的页面，也就是在我们JavaWeb中的动态页面，其本质就是一个Servlet。
2. 其本身是一个动态网页技术标准，它的主要构成有HTML网页代码、Java代码片段、JSP标签几部分组成，后缀是.jsp
3. 相比于Servlet，JSP更加善于处理显示页面，而Servlet跟擅长处理业务逻辑，两种技术各有专长，所以一般我们会将Servlet和JSP结合使用，Servlet负责业务，JSP负责显示。
4. 一般情况下， 都是Servlet处理完的数据，转发到JSP，JSP负责显示数据的工作
5. JSP的基本语法:



1. JSP的脚本元素
2. 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



1. JSP的表达式
2. 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



1. JSP的隐含对象
   1. out（JspWriter）：相当于response.getWriter()获取的对象，用于在页面中显示信息。
   2. config（ServletConfig）：对应Servlet中的ServletConfig对象。
   3. exception（Throwable）：错误页面中异常对象
   4. response（HttpServletResponse）：HttpServletResponse对象
   5. **pageContext**（PageContext）：当前页面的上下文，也是一个域对象。
   6. page（Object）：对应当前Servlet对象，实际上就是this。
   7. **request**（HttpServletRequest）：HttpServletRequest对象，也是域对象
   8. **session**（HttpSession）：HttpSession对象，也是域对象
   9. **application**（ServletContext）：ServletContext对象，也是域对象
2. EL表达式
3. 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。
4. **EL表达式用于代替JSP表达式(<%= %>)在页面中做输出操作。**
5. EL表达式仅仅用来读取数据，而不能对数据进行修改。
6. 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。**
7. **EL表达式的语法:**



1. EL取值的四个域:

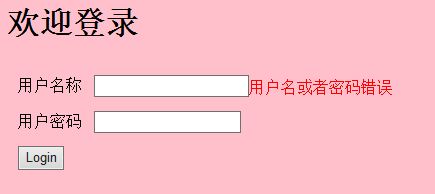
pageScope

requestScope

sessionScope

applicationScope

## 6.6 页面中错误提示的功能效果



## 6.7 JavaScript

1. 在1995年时，由[Netscape](http://baike.baidu.com/view/153922.htm)公司的[Brendan Eich](http://baike.baidu.com/view/2135520.htm)，在[网景导航者](http://baike.baidu.com/view/1350596.htm)浏览器上首次设计实现而成。[Netscape](http://baike.baidu.com/view/153922.htm)在最初将其脚本语言命名为[LiveScript](http://baike.baidu.com/view/2373233.htm)，因为[Netscape](http://baike.baidu.com/view/153922.htm)与[Sun](http://baike.baidu.com/subview/24856/10322735.htm)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)，因此取名为JavaScript。
2. 特性
3. 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。
4. 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。
5. 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。
6. 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。
7. 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。

1. 编写位置
2. 编写到HTML中<script>标签中。



1. 写在外部的.js文件中。然后通过script标签引入。



1. JavaScript的事件驱动
2. 用户事件：用户操作，例如单击、鼠标移入、鼠标移出等
3. 系统事件：由系统触发的事件，例如文档加载完成。
4. 常用的事件:

onload

onclick

onblur

onfocus

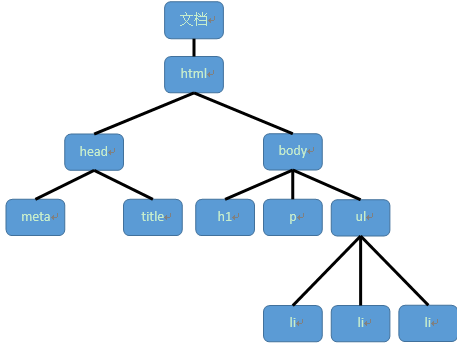
onmouseover

onmouseout

1. BOM
2. Borwser Object Model 浏览器对象模型
3. 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
4. 常用的对象(window的子对象)

document history location screen navigator frames

1. DOM
2. Document Object Model 文档对象模型
3. document对象: 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。
4. DOM树



1. 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 |

# 第7章 注册功能实现-异步的表单校验

## 7.1 涉及的技术知识点

1. Ajax

## 7.2 Ajax

1. AJAX 是 Asynchronous JavaScript And XML 的简称。直译为，异步的JS和XML。
2. 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。
3. AJAX也可以简单的理解为通过JS向服务器发送请求。

## 7.3 异步处理

1. 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住不动的。

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**。

1. 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**。

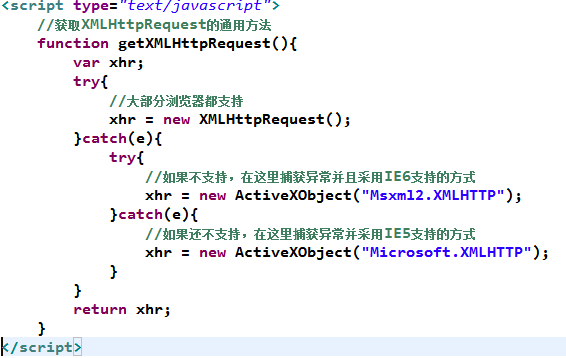
## 7.4 异步请求对象

1. 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。

### 7.4.1 Xhr对象的获取



### 7.4.2 Xhr对象的方法

* 1. open(method,url,async)

open()用于设置请求的基本信息，接收三个参数。

* + - 1. method
         * 请求的方法：get或post
         * 接收一个字符串
      2. url
         * 请求的地址，接收一个字符串
      3. Assync
         * 发送的请求是否为异步请求，接收一个布尔值。
         * true 是异步请求
         * false 不是异步请求（同步请求）
  1. send(string)

send()用于将请求发送给服务器，可以接收一个参数

* + - 1. string参数
         * 该参数只在发送post请求时需要。
         * string参数用于设置请求体
  1. setRequestHeader(header,value)

用于设置请求头

* + - 1. header参数
         * 字符串类型，要设置的请求头的名字
      2. value参数
         * 字符串类型，要设置的请求头的值

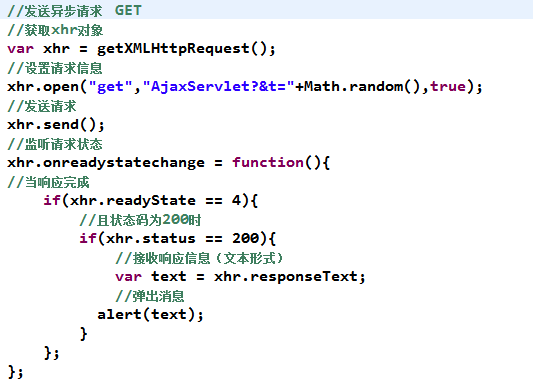
### 7.4.3 XMLHttpRequest对象的属性

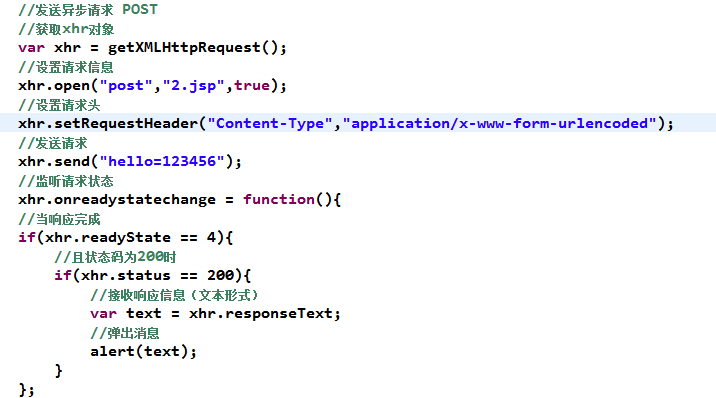
1. readyState
2. 描述XMLHttpRequest的状态
3. 一共有五种状态分别对应了五个数字：
   * + - 0 ：请求尚未初始化，open()尚未被调用
       - 1 ：服务器连接已建立，send()尚未被调用
       - 2 ：请求已接收，服务器尚未响应
       - 3 ：请求已处理，正在接收服务器发送的响应
       - 4 ：请求已处理完毕，且响应已就绪。
   1. status
      1. 请求的响应码
         * 200 响应成功
         * 404 页面没有找到
         * 500 服务器内部错误

… … … …

* 1. onreadystatechange

1. 该属性需要指向一个函数
2. 该函数会在readyState属性发生改变时被调用
   1. responseText
3. 获得字符串形式的响应数据。
   1. responseXML（用的比较少）
4. 获得 XML 形式的响应数据。
   1. 示例代码





### 7.4.4 使用jQuery框架来发送异步请求

1. 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.

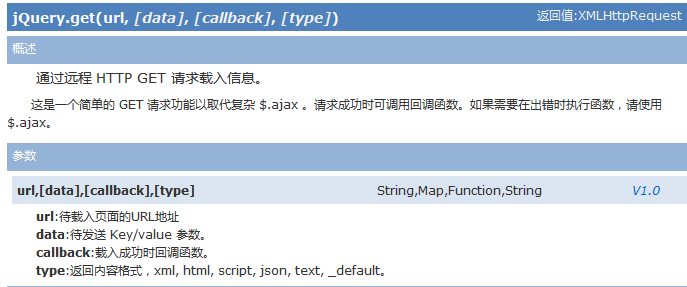
jQuery的对象的本质就是dom对象的数组/集合

1. jQuery对象与dom对象的相互转换

JS转jQuery: var jObj = $(dObj);

jQuery转JS: var dObj = jObj[0] 或者 var dObj = jObj.get(0)

1. $.get方法



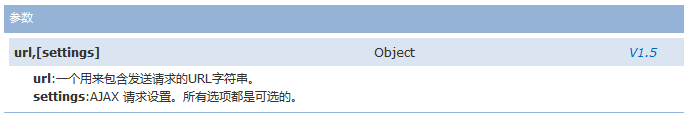
1. $.post方法



1. $.ajax方法

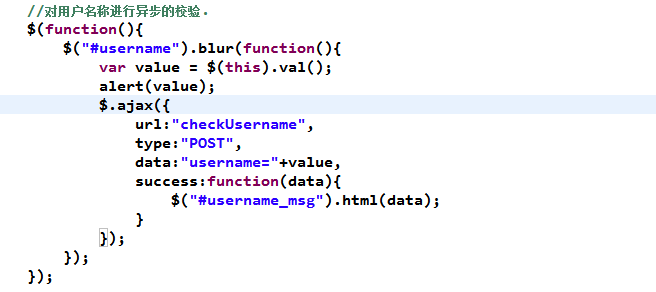
jQuery 底层 AJAX 实现。简单易用的高层实现见 $.get, $.post 等。$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该函数，除非你需要操作不常用的选项，以获得更多的灵活性。最简单的情况下，$.ajax()可以不带任何参数直接使用。

$.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

1. 具体的示例代码



# 第8章 登录功能实现-登录成功跳转主页面

## 8.1 涉及的技术知识点

1. Session会话 Cookie
2. JSTL标签

## 8.2 Cookie

1. 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。
2. **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。
3. Cookie的用途

用户登录状态的保持

广告推荐

1. Cookie的限制性
   1. Cookie作为请求或响应报文发送，无形中增加了网络流量。
   2. Cookie是明文传送的安全性差。
   3. 各个浏览器对Cookie有限制，使用上有局限
   4. Cookie的值只能是String类型，不能保存对象
2. Cookie的具体使用
3. 创建cookie



1. 读取cookie



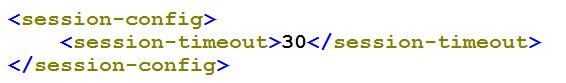
## 8.3 Session

1. 使用Cookie有一个非常大的局限，就是如果Cookie很多，则无形的增加了客户端与服务端的数据传输量。而且由于浏览器对Cookie数量的限制，注定我们不能再Cookie中保存过多的信息，于是Session出现。
2. Session的作用就是在服务器端保存一些用户的数据，然后传递给用户一个名字为JSESSIONID的Cookie，这个JESSIONID对应这个服务器中的一个Session对象，通过它就可以获取到保存用户信息的Session。
3. Session的工作原理
   1. Session的创建时机是在request.getSession()方法第一次被调用时。
   2. Session被创建后，同时还会有一个名为JSESSIONID的Cookie被创建。
   3. 这个Cookie的默认时效就是当前会话。
   4. 简单来说，Session机制也是依赖于Cookie来实现的
4. Session的具体使用



1. Session的时效问题

Session默认有效时间为30分钟，可以在服务器的web.xml配置中修改.



## 8.4 具体功能展示



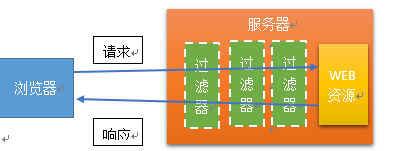
# 第9章主页面访问权限控制

## 9.1 涉及的技术知识点

1. 过滤器

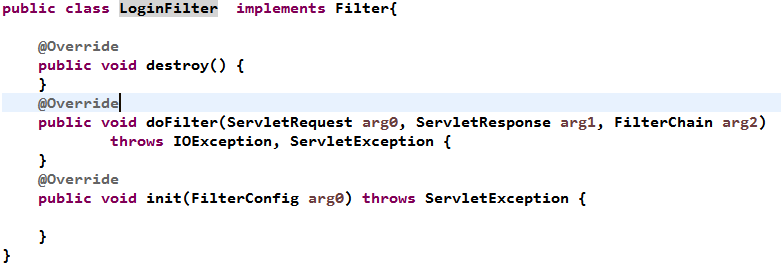
## 9.2 过滤器

1. 对于WEB应用来说，过滤器是一个驻留在服务器中的WEB组件，他可以截取客户端和WEB资源之间的请求和响应信息。WEB资源可能包括Servlet、JSP、HTML页面等
2. 当服务器收到特定的请求后，会先将请求交给过滤器，程序员可以在过滤器中对请求信息进行读取修改等操作，然后将请求信息再发送给目标资源。目标资源作出响应后，服务器会再次将响应转交给过滤器，在过滤器中同样可以对响应信息做一些操作，然后再将响应发送给浏览器。
3. 也就是说过滤器可以在WEB资源收到请求之前，浏览器收到响应之前，对请求和响应信息做一些相应的操作。
4. 在一个WEB应用中可以部署多个过滤器，多个过滤器就组成了一个过滤器链，请求和响应必须在经过多个过滤器后才能到达目标

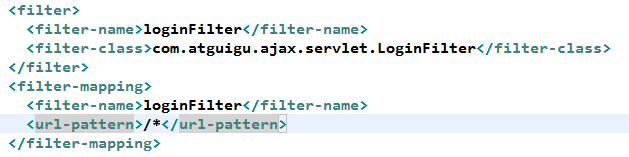


## 9.3 过滤器的使用

1. 通过实现Filter接口完成过滤器的开发



1. Filter在web.xml中的配置



## 9.4 主页面访问权限控制要求

1. 在进入主页面必须进行登录状态的判断，如果未登录状态不允许进入主界面。
2. 登录状态的判断再过滤器中实现，更为通用，而且可拔插。

# 第10章 xml

## 11.1 xml简介

1. XML--可扩展标记语言eXtensible Markup Language
2. 由W3C组织发布，目前推荐遵守的是W3C组织于2000年发布的XML1.0规范
3. XML的使命，就是以一个统一的格式，组织有关系的数据，为不同平台下的应用程序服务
4. XML用来传输和存储数据，HTML用来显示数据
5. XML没有预定义标签，均为自定义标签

## 11.2 xml用途

1. 配置文件

JavaWeb中的web.xml

C3P0中的c3p0-config.xml

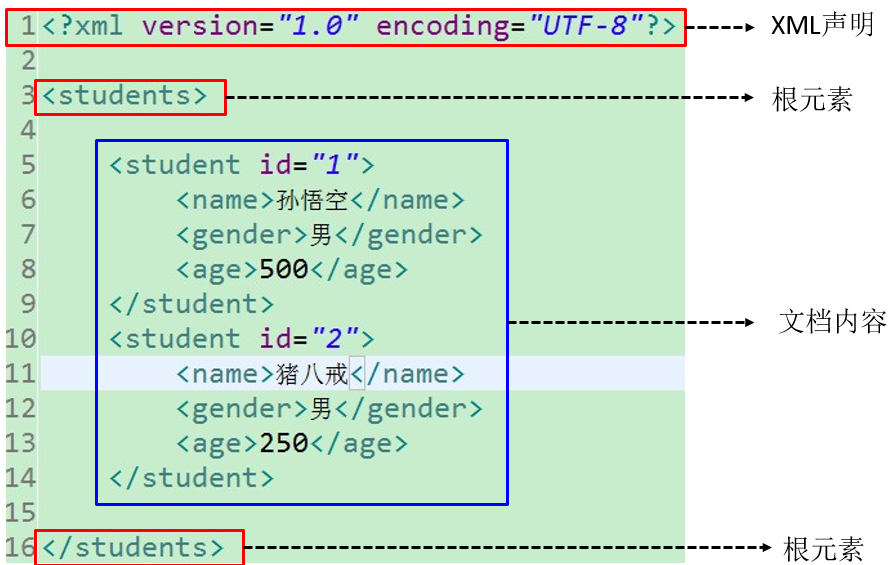
1. 数据交换格式

Ajax

WebService

1. 数据存储

保存关系型数据



## 11.3 xml基本语法

1. XML文档组成
2. XML声明

version属性指定XML版本，固定值是1.0

encoding指定的字符集，是告诉解析器使用什么字符集进行解码，而编码是由文本 编辑器决定的

1. CDATA区

当XML文档中需要写一些程序代码、SQL语句或其他不希望XML解析器进行解析 的内容时，就可以写在CDATA区中

XML解析器会将CDATA区中的内容原封不动的输出

CDATA区的定义格式：<![CDATA[…]]>

1. 语法规则
2. XML声明要么不写，要写就写在第一行，并且前面没有任何其他字符
3. 只能有一个根标签
4. 标签必须正确结束
5. 标签不能交叉嵌
6. 严格区分大小写
7. 属性必须有值，且必须加引号
8. 标签不能以数字开头
9. 注释不能嵌套

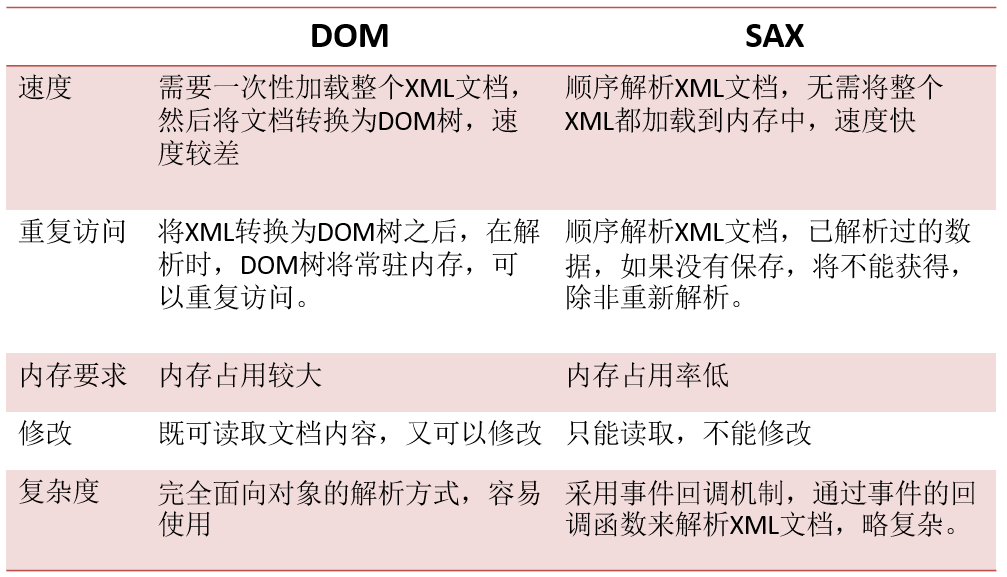
## 11.4 xml解析

1. XML解析是指通过解析器读取XML文档，解释语法，并将文档转化成对象
2. 常用的解析方式

DOM（Document Object Model）

SAX（Simple API for XML）

1. DOM 和SAX解析的对比



1. Dom4j解析示例

解析

|  |
| --- |
| //1.创建解析器对象  SAXReader saxReader = **new SAXReader();**  //2.解析xml文件获取document对象  Document document = saxReader.read("students.xml");  //3.得到根元素  Element root = document.getRootElement(); |

# 第11章 JSON

## 12.1 JSON 简介

1. AJAX一开始使用的是XML的数据格式，XML的数据格式非常简单清晰，容易编写，但是由于XML中包含了过多的标签，以及十分复杂的结构，解析起来也相对复杂，所以目前来讲，AJAX中已经几乎不使用XML来发送数据了。取而代之的是一项新的技术JSON。
2. JSON是JavaScript Object Notation 的缩写，是JS提供的一种数据交换格式。
3. JSON对象本质上就是一个JS对象，但是这个对象比较特殊，它可以直接转换为字符串，在不同语言中进行传递，通过工具又可以转换为其他语言中的对象。
4. 例，有如下一个JSON对象：
5. {“name”:”sunwukong” , ”age”:18 , ”address”:”beijing” }
6. 这个对象中有三个属性name、age和address
7. 如果将该对象使用单引号引起了，那么他就变成了一个字符串
8. ‘{“name”:”sunwukong” , ”age”:18 , ”address”:”beijing” }’
9. 变成字符串后有一个好处，就是可以在不同语言之间传递。
10. 比如，将JSON作为一个字符串发送给Servlet，在Java中就可以把JSON字符串转换为一个Java对象。

## 12.2 JSON通过6种数据类型来表示

1. 字符串
   * + 例子：”字符串”
     + 注意：不能使用单引号
2. 数字：
   * + 例子：123.4
3. 布尔值：
   * + 例子：true、false
4. null值:
   * + 例子：null
5. 对象
   * + 例子：{“name”:”sunwukong”, ”age”:18}
6. 数组
   * + 例子：[1,”str”,true]

## 12.3 在JS中操作JSON

1. 创建JSON对象
   * + var json = {“name1”:”value1”,”name2”:”value2” , “name3”:[1,”str”,true]};
     + var json = [{“name1”:”value1”},{“name2”:”value2”}];
2. JSON对象转换为JSON字符串
   * + JSON.stringify(JSON对象)
3. JSON字符串转换为JSON对象
   * + JSON.parse(JSON字符串)

## 12.4 在Java中操作JSON

1. 在Java中可以从文件中读取JSON字符串，也可以是客户端发送的JSON字符串，所以第一个问题，我们先来看如何将一个JSON字符串转换成一个Java对象。
2. 首先解析JSON字符串我们需要导入第三方的工具，目前主流的解析JSON的工具大概有三种json-lib、jackson、gson。三种解析工具相比较json-lib的使用复杂，且效率较差。而Jackson和gson解析效率较高。使用简单，这里我们以gson为例讲解。
3. Gson是Google公司出品的解析JSON工具，使用简单，解析性能好。
4. Gson中解析JSON的核心是Gson的类，解析操作都是通过该类实例进行。
5. JSON字符串转换为对象

|  |
| --- |
| String json = "{\"name\":\"张三\",\"age\":18}";  Gson gson = **new** Gson();  //转换为集合  Map<String,Object> stuMap = gson.fromJson(json, Map.**class**);  //如果编写了相应的类也可以转换为指定对象  Student fromJson = gson.fromJson(json, Student.**class**); |

1. 对象转换为JSON字符串

|  |
| --- |
| Student stu = **new** Student("李四", 23);  Gson gson = **new** Gson();  //{"name":"李四","age":23}  String json = gson.toJson(stu);    Map<String , Object> map = **new** HashMap<String, Object>();  map.put("name", "孙悟空");  map.put("age", 30);  //{"age":30,"name":"孙悟空"}  String json2 = gson.toJson(map);    List<Student> list = **new** ArrayList<Student>();  list.add(**new** Student("八戒", 18));  list.add(**new** Student("沙僧", 28));  list.add(**new** Student("唐僧", 38));  //[{"name":"八戒","age":18},{"name":"沙僧","age":28},  {"name":"唐僧","age":38}]  String json3 = gson.toJson(list);  // 如果将一个数组格式的json字符串转换成java对象需要用到  //Gson提供的一个匿名内部类： TypeToken  TypeToken tk= new TypeToken<List<User>>(){};  List<User> list2 = gson.fromJson(json,tk.getType());  System.out.println(list2.get(0)); |

## 12.5 JQuery 异步请求返回JSON数据

1. Servlet 返回json数据

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  // 创建EmployeeDao对象  EmployeeDao employeeDao = **new** EmployeeDaoImpl();  // 调用EmployeeDao中获取所有员工的方法  List<Employee> employees = employeeDao.getEmployees();  //创建Gson对象  Gson gson = **new** Gson();  //将List转换为JSON字符串  String json = gson.toJson(employees);  response.setContentType("text/html;charset=UTF-8");  response.getWriter().write(json);  } |

1. 页面中处理 json数据

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"* src=*"script/jquery-1.7.2.js"*></script>  <script type=*"text/javascript"*>  $(**function**(){  //给按钮绑定单击事件  $("#btnId").click(**function**(){  //发送Ajax请求  //设置请求地址  **var** url = "GetEmployeesByAjax";  $.post(url,**function**(data){  **var** str = "<tr><th>Id</th><th>LastName</th><th>Email</th><th>Salary</th><th>Dept</th></tr>";  **for**(**var** i= 0 ;i <data.length;i++){  **var** emp = data[i];  str+="<tr align='center'><td>"  +emp.id+  "</td><td>"  +emp.lastName+  "</td><td>"  +emp.email+  "</td><td>"  +emp.salary+  "</td><td>"  +emp.dept+  "</td></tr>"  }  $("#tb").html(str);  },"json");  });  });  </script>  </head>  <body>  <button id=*"btnId"*>通过发送Ajax请求获取数据库中所有的员工信息</button>  <h1 align=*"center"*>员工信息</h1>  <table id=*"tb"* border=*"1"* align=*"center"* cellspacing=*"0"* cellpadding=*"10"*>  </table>    </body>  </html> |