上课不要接电话，不要上厕所，不要拿快递。

# CSS

## CSS简介

CSS:层叠样式表，是Cascade Style Sheet

优势：是页面的结构和表现样式分离。

## CSS引入方式

### 第一种引入方式：页面内，style引入方式

例如 ：

<style type="text/css">

a{

color:red;

text-decoration:none;

}

</style>

### 第二种：行内引入

例如：<a style="color:#0F0; text-decoration:none" href="#">新闻的标题1</a>

### 第三种：外部引入

1.使用link标签

例如：<link href="1css.css" rel="stylesheet"/>

[2.使用css里面的@import](mailto:2.使用css里面的@import)

例如：<style>

@import url("1css.css");

</style>

## CSS注释

/\*CSS注释\*/

<!—html注释-->

## 选择器

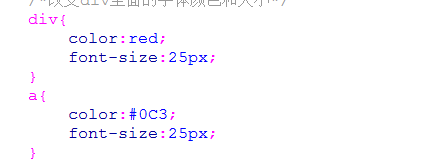
### 标签选择器

格式：标签名{

CSS样式代码；

}

例子：

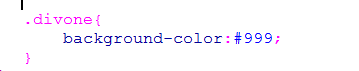


### 类选择器

注意：必须要给一个标签指定class属性

Class属性值：可以使用字母下划线数字的组合。但是不能以数字开头。

格式：.class名{ css样式}



### ID选择器

注意：必须要给标签指定ID属性值

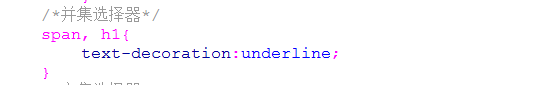
Id属性值：可以使用字母下划线数字的组合，但是不能以数字开头。

格式：#id值{css样式}

### 并集选择器

并集选择器的格式：标签1 ,标签2{ css样式代码}

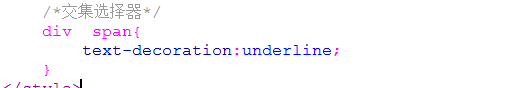
例如：



### 交集选择器

交集选择器的格式：标签1 标签2{css样式代码}

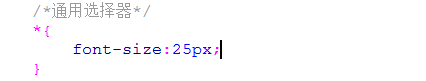
例如：



### 通用选择器

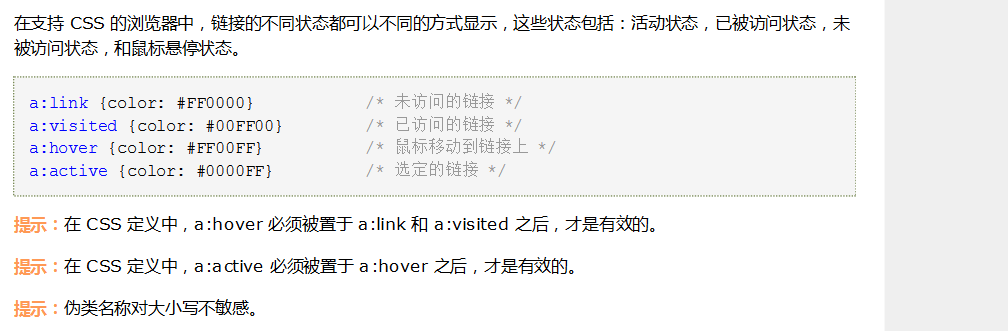
格式：\*{ css样式代码}

例如：



### 伪类选择器

语法：选择器:伪类{属性:值}



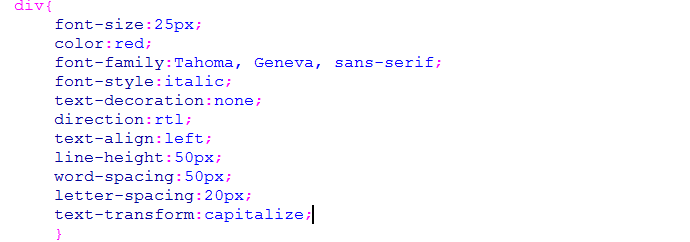
## CSS常用属性

## 背景



## 文本





# JavaScript

## BOM编程

### BOM编程基础

全称 Browser Object Model，浏览器对象模型。

JavaScript是由浏览器中内置的javascript脚本解释器程序来执行javascript脚本语言的。

为了便于对浏览器的操作，javascript封装了对浏览器的各个对象使得开发者可以方便的操作浏览器。

### BOM对象-window对象

Window 对象是 JavaScript 层级中的顶层对象。

Window 对象代表一个浏览器窗口或一个框架

Window 对象会在 <body> 或 <frameset> 每次出现时被自动创建。

Window对象包括：

* document 文档对象
* location 地址栏对象，包含当前url信息
* history 包含用户（在浏览器窗口中）访问过的 URL。
* screen：屏幕对象

#### window对象中的属性及方法

[moveto](mk:@MSITStore:E:\\文档\\DHTML.chm::/DHTMLref/methods/moveTo.html)() 将窗口左上角的屏幕位置移动到指定的 *x* 和 *y* 位置。

moveby() 相对于目前的位置移动。

resizeTo() 调整当前浏览器的窗口。

open() 打开新窗口显示指定的URL（有的浏览器中是打一个新的选项卡）

alert() 消息提示框，该消息框提供了一个“确定”按钮让用户关闭该消息框，并且该消息框是模式对话框，也就是说，用户必须先关闭该消息框然后才能继续进行操作。

confirm() 消息确认框，显示一个确认按钮和一个取消按钮。

prompt() 用户输入提示框

setTimeout(vCode, iMilliSeconds) 超时后执行代码。只执行一次

setInterval(vCode, iMilliSeconds) 定时执行代码，第一次也是先待，到时再执行。

clearInterval(iIntervalID) 根据定时任务id取消定时任务

#### location对象

location 对象包含有关当前 URL 的信息。

location中的重要方法：

href属性 设置或获取整个 URL 为字符串。

reload() 重新装入当前页面

#### screen对象

screen 对象是由 JavaScript runtime engine 自动创建的，包含有关客户机显示屏幕的信息。

属性：

availHeight 获取系统屏幕的工作区域高度，排除 Microsoft Windows 任务栏。

availWidth 获取系统屏幕的工作区域宽度，排除 Windows 任务栏。

height 获取屏幕的垂直分辨率。

width 获取屏幕的水平分辨率。

示例：

document.write("屏幕工作区: " + screen.availHeight + ", " + screen.availWidth + "<br>");

document.write("屏幕分辨率: " + screen.height + ", " + screen.width + "<br>");

#### history对象

包含用户（在浏览器窗口中）访问过的 URL

[length](mk:@MSITStore:E:\\help\\W3School离线手册(2015.07.17).chm::/www.w3school.com.cn/jsref/prop_his_length.asp)属性：返回浏览器历史列表中的 URL 数量

方法：



#### document对象

该对象代表整个文档页面

对象的集合：

all 获取页面所有元素对象

forms 获取页面所有表单对象

images 获取页面所有图片对象

links 获取所有超链接或area对象

具体的内容会在学习DOM时学习。

## 事件

基本上所有的HTML元素中都可以指定事件属性。

每个元素支持什么样事件应查询文档。

所有的事件属性都是以on开头，后面的是事件的触发方式，如：

onclick，表示单击

onkeydown，表示键按下

### 常用的事件类型

鼠标点击相关：

onclick 在用户用鼠标左键单击对象时触发。

ondblclick 当用户双击对象时触发。

onmousedown 当用户用任何鼠标单击对象时触发。

onmouseup 当用户在鼠标位于对象之上时释放鼠标触发。

鼠标移动相关：

onmouseout 当用户将鼠标指针移出对象边界时触发。

onmousemove 当用户将鼠标划过对象时触发。

onmouseover 鼠标被移到某元素之上

焦点相关的：

onblur 在对象失去输入焦点时触发。

onfocus 当对象获得焦点时触发。

其他：

onchange 当对象或选中区的内容改变时触发。

onload 在浏览器完成对象的装载后立即触发。

onsubmit 当表单将要被提交时触发，onsubmit 事件会在表单中的提交按钮被点击时发生

## DOM编程

全称Document Object Model，即文档对象模型。

DOM描绘了一个层次化的树，允许开发人员添加、删除、修改页面的某一部分。

浏览器在解析HTML页面标记的时候，其实不是按照一行一行读取并解析的，而是将HTML页面中的每一个标记按照顺序在内存中组建一颗DOM树，组建好之后，按照树的结构将页面显示在浏览器的窗口中

### 节点层次

HTML网页是可以看做是一个树状的结构，如下：

html

|-- head

| |-- title

| |-- meta

| ...

|-- body

| |-- div

| |-- form

| | |-- input

| | |-- textarea

... ... ...

这是一颗树，是由多个节点（Node）组成的，节点的类型有很多种。

节点最多有一个父节点，可以有多个子节点。

HTML DOM 定义了访问和操作HTML文档的标准方法。

document

代表当前页面的整个文档树。

访问属性

All 所有的标签

Forms 所有的表单

Images 所有的图片标签

Links 所有的超链接

Body

访问方法(最常用的DOM方法)

|  |  |
| --- | --- |
| *[createElement](mk:@MSITStore:E:\\my\\help\\javaWeb\\毕业\\day05\\day05\\day05\\DHTML.chm::/DHTMLref/methods/createElement.html)* | 为指定标签创建一个元素的实例 |
| *[getElementById](mk:@MSITStore:E:\\my\\help\\javaWeb\\毕业\\day05\\day05\\day05\\DHTML.chm::/DHTMLref/methods/getElementById.html)* | 获取对 [ID](mk:@MSITStore:E:\\my\\help\\javaWeb\\毕业\\day05\\day05\\day05\\DHTML.chm::/DHTMLref/properties/id.html) 标签属性为指定值的第一个对象的引用。 |
| *[getElementsByName](mk:@MSITStore:E:\\my\\help\\javaWeb\\毕业\\day05\\day05\\day05\\DHTML.chm::/DHTMLref/methods/getElementsByName.html)* | 根据 [NAME](mk:@MSITStore:E:\\my\\help\\javaWeb\\毕业\\day05\\day05\\day05\\DHTML.chm::/DHTMLref/properties/name_2.html) 标签属性的值获取对象的集合。 |
| *[getElementsByTagName](mk:@MSITStore:E:\\my\\help\\javaWeb\\毕业\\day05\\day05\\day05\\DHTML.chm::/DHTMLref/methods/getElementsByTagName.html)* | 获取基于指定元素名称的对象集合。 |

|  |
| --- |
| <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <script type=*"text/javascript"*>  // 获取dom 树, 获取document 对象.  **var** dom = window.document;  // all 获取页面中所有的标签节点 ,注释和文档类型约束.  **function** testAll() {  **var** allArr = dom.all;  alert(allArr.length);  **for** (**var** i = 0; i < allArr.length; i++) {  //获取节点名称  alert(allArr[i].nodeName);  }  }  // anchors 获取页面中的所有的锚连接.  **function** testAnchors() {  **var** anArr = dom.anchors;  alert(anArr.length);  }  // froms 获取所有的form 表单对象  **function** testForms() {  **var** formArr = dom.forms;  alert(formArr.length);  alert(formArr[0].nodeName);  }  // images  **function** testImages() {  **var** imageArr = dom.images;  alert(imageArr.length);  }  // links 获取页面的超链接.  **function** testLinks() {  **var** linkArr = dom.links;  //alert(linkArr.length);  **for** (**var** i = 0; i < linkArr.length; i++) {  //alert(linkArr[i].nodeName);  }  **for** (**var** i **in** linkArr) {  alert(i);  }  }  //testLinks();  // 获取页面的Body  **var** body = dom.body;  alert(body.nodeName);  </script>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>javascript</title>  </head>  <body onmousemove="test(this)">  <img src=*"xxx"* alt=*"这是一个美女"*/>  <img src=*"xxx"* alt=*"这是一个美女"*/>  <img src=*"xxx"* alt=*"这是一个美女"*/>  <a href=*"http://www.baidu.com"*>百度一下</a>  <a href=*"http://www.google.com"*>百度两下</a>  <a href=*"http://www.baigu.com"*>百谷一下</a>  <a name=*"one"*></a>  <a name=*"two"*></a>  <form>  <label>姓名:</label><!--默认不写type 就是文本输入框-->  <input type=*"text"*/>  </form>  </body>  </html> |

### 获取节点对象

document.getElementById("html元素的id")

document.getElementsByTagName("标签名") //根据标签名称获取元素，返回的元素集合

document.getElementsByName("html元素的name") //获取name名称相同的元素，返回数组元素集合，一般用于复选框。

示例：

1，得到所有的div元素并显示信息（innerHTML）。

2，得到所有div元素中id为"test"的结果。

#### 小练习

查看下面的页面，点击设置文本，给文本输入框动态的赋值”day day up”。点击显示图片按钮，给页面上的三个图片标签加载上图片，并且设置宽度和高度。点击设置div内容按钮，让name为info的div显示一句话“我是div，我可以显示内容”

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>document对象练习</title>  </head>  <body>  <input type="text" id="userName" value="请输入用户名..." /><input type="button" onclick="showText()" value="设置文本"/>  <hr/>    <img src="" />  <img src="" />  <img src="" />  <input type="button" onclick="showImage()" value="显示图片"/>  <hr/>  <div name="info"></div>  <div name="info"></div>  <div name="info"></div>  <input type="button" onclick="showDiv()" value="设置div内容"/>    </body>  </html> |

#### 小练习：全选全不选



1. 制作一个页面，效果图如上图所示
2. 通过点击上面的全选复选框，实现页面所有复选框的全选全不选效果。
3. 点击计算总金额按钮，计算所选中的复选框对应的金额的总和，并显示到倒数第二行的位置。

### 通过节点关系查找结点

从一个节点出发开始查找：

parentNode 获取当前元素的父节点。

childNodes 获取当前元素的所有下一级子元素。

firstChild 获取当前节点的第一个子节点。

lastChild 获取当前节点的最后一个子节点。

nextSibling 获取当前节点的下一个节点。（弟节点）

previousSibling 获取当前节点的上一个节点。（兄节点）

示例1：

firstChild属性最普遍的用法是访问某个元素的文本：

var text=x.firstChild.nodeValue;

示例2：

parentNode 属性常被用来改变文档的结构。

假设您希望从文档中删除带有 id 为 "maindiv" 的节点：

var x=document.getElementById("maindiv");

x.parentNode.removeChild(x);

#### 获取节点对象的信息

每个节点都包含的信息的，这些属性是：

nodeType 节点类型

nodeName 节点名称

nodeValue 节点值

nodeType 属性可返回节点的类型。

---------------------------------

元素类型 节点类型

------------------

元素 1 就是标签元素，例<div>..</div>

文本 3 标签元素中的文本

注释 8 表示为注释

nodeName

nodeName 属性含有某个节点的名称。

--------------------------------

元素节点的 nodeName 是标签名称

属性节点的 nodeName 是属性名称

文本节点的 nodeName 永远是 #text

文档节点的 nodeName 永远是 #document

nodeValue

对于文本节点，nodeValue 属性是所包含的文本。

对于属性节点，nodeValue 属性是属性值。

对于注释节点，nodeValue 属性注释内容。

nodeValue 属性对于文档节点和元素节点是不可用的。

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <script type=*"text/javascript"*>  //节点和节点之间的关系.  //获取dom树  **var** dom = window.document;  //获取指定id 的标签节点.  **function** test() {  **var** form = dom.getElementById("form1");  //获取父节点.  //alert(form.parentNode.nodeName);  // 获取子节点(Node 包含 文本,注释,标签)  **var** childArr = form.childNodes;  //alert(childArr.length);  /\*  for (var i = 0; i < childArr.length; i++) {  alert(childArr[i]);  }  \*/  // 获取第一个孩子.  **var** first = form.firstChild;  //alert(first);  //最后一个孩子.  **var** last = form.lastChild;  //alert(last);  // 获取下兄弟(获取弟弟)  **var** sibling = form.nextSibling;  //alert(sibling.nodeName);  // 获取大哥  **var** previous = form.previousSibling;  alert(previous.nodeName);  }  test();  </script>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />  <title>javascript</title>  </head>  <body onmousemove="test(this)">  <a>哈哈</a>  <form id=*"form1"*>  <label>姓名:</label>  <input type=*"text"* />  </form>  </body>  </html> |

### 操作节点

#### 创建、删除、追加、插入节点

document.createElement("标签名") 创建新元素节点

elt.setAttribute("属性名", "属性值") 设置属性

elt.appendChild(e) 添加元素到elt中最后的位置

elt.insertBefore(new, child); 添加到elt中，child之前。 // 参数1：要插入的节点 参数 2：插入目标元素的位置

elt.removeChild(eChild) 删除指定的子节点

##### 案例1

动态添加和删除附件信息

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>操作节点</title>  <style type="text/css">  td{  height:50px; width:250px;}  </style>  <script type="text/javascript">  function addRow(){  //创建节点，createElement()  var trNode=document.createElement("tr");  var td1=document.createElement("td");  td1.innerHTML="<input type='file'>";  var td2=document.createElement("td");  td2.innerHTML="<a href='#' onclick='delFile(this)'>删除附件</a>";  //给一个节点添加子节点 appendChild()  trNode.appendChild(td1);  trNode.appendChild(td2);    var tbodyNode=document.getElementsByTagName("tbody")[0];  var buttonTr=document.getElementById("addButton");  //在添加附件的行的上面，新插入一行 insertBefore  tbodyNode.insertBefore(trNode,buttonTr);  }  /\*\*  删除附件 removeChild()  \*/  function delFile(node){  //alert(node.parentNode.parentNode.parentNode.nodeName);  var trNode=node.parentNode.parentNode;  var tbodyNode=trNode.parentNode;  tbodyNode.removeChild(node.parentNode.parentNode);    }  </script>  </head>  <body>  <table style="border-collapse:collapse;" border="1px;">  <tr>  <td><input type="file"></td>  <td><a href="#" onclick="delFile(this)">删除附件</a></td>  </tr>  <tr id="addButton">  <td colspan="2">  <input type="button" value="添加附件" onclick="addRow()"/>  </td>  </tr>  </table>  </body>  </html> |

##### 案例2：两级联动

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>二级城市联动</title>  <script type="text/javascript">  function selCity(){  //二维数组  var subCity=[["--选择城市--"],["昌平区","海淀区","朝阳区"],["石家庄","保定市","邯郸市"],["浦东区","普陀区","虹口区"]];  var selid=document.getElementById("selid");  var index=selid.selectedIndex; //获取下拉列表选中项的索引    var subselid=document.getElementById("subselid"); //获取城市的选择下拉框的对象  //清空二级下拉列表以前的值  subselid.length=0;      var cityArray=subCity[index]; //根据选中的索引，获取二级城市数组  for(var i=0;i<cityArray.length;i++){  var option=document.createElement("option");//创建节点  option.innerHTML=cityArray[i]; //给节点赋值  subselid.appendChild(option); //追加子节点  }      }    </script>  </head>  <body>  <select id="selid" onchange="selCity()">  <option>--选择省市--</option>  <option>北京市</option>  <option>河北省</option>  <option>上海市</option>  </select>&nbsp;&nbsp;&nbsp;&nbsp;  <select id="subselid">  <option>--选择城市--</option>  </select>  </body>  </html> |

### 正则表达式

位置：

^ 开头

$ 结尾

次数：

\* 0或多个

+ 1或多个

? 0或1个

{n} 就是n个

{n,} 至少n个

{n,m} 最少n个，最多m个

通配符：

\d 任意数字

\D 任意非数字

\s 任意空白

\S 任意非空白

. 任意字符（除'\n'外）

组合：

[a-z]

[0-9]

等

组：

(正则) 匹配括号中正则表达式对应的结果，并暂存这个结果。

(?:正则) 匹配括号中正则表达式对应的结果，但不暂存这个结果。

\数字 使用第n个组匹配的结果

使用正则的工具（RegExp类与相关方法）

创建：

// 方式一

var regex = new RegExp("正则表达式", "标志");

// 方式二

var regex = /正则表达式/标志

参数说明：

正则表达式：

参见上面的规则

标志：

g （全文查找出现的所有 pattern）

i （忽略大小写）

m （多行查找）

方法：

Regexp.test( str )

String.replace( regex, str )

练习：校验表单

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <script>    //用户名的规则： 第一位是字母，只有由数字与字母 组成，6位。  function checkName(){  //获取到了用户名的值  var userName = document.getElementById("username").value;  var userSpan = document.getElementById("userId");  var reg = /^[a-z][a-z0-9]{5}$/i;  if(reg.test(userName)){  //符合规则  userSpan.innerHTML="正确".fontcolor("green");  return true;  }else{  //不符合规则  userSpan.innerHTML="错误".fontcolor("red");  return false;  }  }      //校验密码 6位  function checkPass(){  var password = document.getElementById("pwd").value;  if(password.length>0){  var reg = /^\w{6}$/;  var passSPan = document.getElementById("passId");  if(reg.test(password)){  //符合规则  passSPan.innerHTML="正确".fontcolor("green");  return true;  }else{  //不符合规则  passSPan.innerHTML="错误".fontcolor("red");  return false;  }  }    }      //检验密码是否正确  function ensurepass(){  var password1 = document.getElementById("pwd").value; //第一次 输入的密码  var password2 = document.getElementById("ensurepwd").value;  if(password2.length>0){  var enSpan = document.getElementById("ensure");  if(password1.valueOf()==password2.valueOf()){  enSpan.innerHTML="正确".fontcolor("green");  return true;  }else{  enSpan.innerHTML="错误".fontcolor("red");  return false;  }  }  }      //校验邮箱  function checkEmail(){  var email = document.getElementById("email").value;  var reg = /^[a-z0-9]\w+@[a-z0-9]{2,3}(\.[a-z]{2,3}){1,2}$/i; // .com .com.cn  var emailspan = document.getElementById("emailspan");  alert(reg.test(email));  if(reg.test(email)){  //符合规则  emailspan.innerHTML="正确".fontcolor("green");    return true;  }else{  //不符合规则  emailspan.innerHTML="错误".fontcolor("red");  return false;  }  }      //校验兴趣爱好： 至少要算中其中 的 一个。  function checkHoby(){  var likes = document.getElementsByName("like");  var hobySpan =document.getElementById("hobbySpan")  var flag = false;  for(var i = 0 ; i<likes.length ; i++){  if(likes[i].checked){  flag =true;  break;  }  }  if(flag){  //符合规则  hobySpan.innerHTML="正确".fontcolor("green");  return true;  }else{  //不符合规则  hobySpan.innerHTML="错误".fontcolor("red");  return false;  }  }        //总体校验表单是否可以提交了 如果返回的true表单才可以提交。 上面的表单项必须要每个都填写正确。  function checkForm(){  var userName = checkName();  var pass = checkPass();  var ensure = ensurepass();  var email = checkEmail();  var hoby = checkHoby();  if(userName&&pass&&ensure&&email&&hoby){  return true;  }else{  return false;  }    }            </script>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>正则表达式</title>  </head>  <body>  <form action="success.html" method="get" onsubmit="return checkForm()"> <!--如果表单提交时候触发的方法返回是false,那么该表单不允许提交，如果返回的是true允许提交 -->  <table border="1px" width="50%" align="center" cellspacing="0px" cellpadding="3px">  <tr>  <td width="25%">姓名:</td>  <td>  <input type="text" name="username" id="username" onblur="checkName()"/>  <span id="userId"></span>  </td>  </tr>  <tr>  <td >密码:</td><td>  <input type="password" name="pwd" id="pwd" onblur="checkPass()"/>  <span id="passId"></span>  </td>  </tr>  <tr>  <td>确认密码:</td><td>  <input type="password" name="ensurepwd" id="ensurepwd" onblur="ensurepass()" /> <span id="ensure"></span>  </td>  </tr>  <tr>  <td>邮箱</td><td>  <input type="text" name="email" id="email" onblur="checkEmail()"/>  <span id="emailspan"></span>    </td>  </tr>  <tr>  <td>性别</td><td>  <input type="radio" checked="ture" name="gender" id="male" value="male"/>  男  <input type="radio" name="gender" value="female"/>  女</td>  </tr>    <tr>  <td>爱好:</td><td>  <input type="checkbox" checked="checked" name="like" />  eat  <input type="checkbox" name="like" />  sleep  <input type="checkbox" name="like"/>  play  <span id="hobbySpan"></span>  </td>  </tr>  <tr>  <td>城市</td><td>  <select name="city" id="city">  <option value=""> 请选择</option>  <option value="bj"> 北京 </option>  <option value="gz"> 广州 </option>  <option value="sh"> 上海 </option>  </select>    </td>  </tr>  <tr>  <td>自我介绍</td><td> <textarea cols="15" rows="5" name="myInfo" id="myInfo"></textarea></td>  </tr>  <tr align="center">  <td colspan="2"><!--提交按钮-->  <input type="submit"/>  </td>  </tr>  </table>  </form>  </body>  </html> |

# Xml

## xml引入

Html:超文本标记语言，负责页面的结构

Css:层叠样式，负责网页的美观

Javascript:脚本语言，负责与用户交互

html语言的特点：

1)，由标签组成 <p></p>

2)，标签名不区分大小写

|  |
| --- |
| <html>  <head>  <title>this is title</title>  </head>  <body>  <p>html标签</p>  <P>html标签</P>  </body>  </html> |

我们在生活中会遇到大量有关系的数据，如图所示：

1. 中国
2. 北京
3. 湖南
4. 湖北
5. 海淀
6. 丰台
7. 长沙
8. 岳阳
9. 武汉
10. 荆州

以上数据是有层级关系的，那么这样的数据怎么交给计算机处理呢？

## xml概述

XML语言出现的根本目标在于描述上图那种，在现实生活中经常出现的有关系的数据

在XML语言中，它允许用户自定义标签。一个标签用于描述一段数据；一个标签可分为开始标签和结束标签，在开始标签和结束标签之间，又可以使用其它标签描述其它数据，以此来实现数据关系的描述

XML：Extensible Markup Language，翻译过来为可扩展标记语言。Xml技术是w3c组织发布的，目前推荐遵循的是W3C组织于2000发布的XML1.0规范

xml除了用于保存有关系的数据以外，它还经常用作软件的配置文件。

### xml的作用

#### 描述关系型数据(配置文件)

java代码：使用ip（255.43.12.54）地址和端口（1521），连接oracle数据库，保存学生数据。

把ip地址端口配置到xml文件：host.xml

|  |
| --- |
| <host id=”1”>  <ip>255.43.12.55</ip>  <port>1521</port>  </host>  <host id=”2”>  <ip>255.43.12.54</ip>  <port>1521</port>  </host> |

数据库服务器（PC）：

主服务器（255.43.12.54）：Oracle数据库软件（负载）

副服务器（255.43.12.55）：Oracle数据库软件

#### 数据的载体，小型的数据库

|  |
| --- |
| <teacher>  <name>张三</name>  <email>zhangsan@qq.com</email>  <workage>2</workage>  </teacher> |

#### 作为数据传输的格式

通过 XML，可以在不兼容的系统之间轻松地交换数据。

对开发人员来说，其中一项最费时的挑战一直是在因特网上的不兼容系统之间交换数据。

由于可以通过各种不兼容的应用程序来读取数据，以 XML 交换数据降低了这种复杂性

### xml与html的区别

|  |  |
| --- | --- |
| xml | html |
| 可扩展性标记语言 | 超文本标记语言 |
| 可以自定义标签，可以自定义标签属性 | 不能自定义标签，标签是固定的，是由w3c组织制定的，约100来个 |
| 作为软件的配置文件。  作为传输格式（xml,json）  可作为小型数据库，供用户管理数据 | 负责html的结构 |

## xml的语法

xml文件是以.xml结尾的

xml文件需要使用xml解析器去解析。浏览器内置了xml解析器

### 标签

语法：

<student></student> 开始标签 标签体内容 结束标签

<student/> 或 <student></student> 空标签。没有标签体内容

注意：

1. 标签严格区分大小写
2. 标签要正确配对
3. 标签名称不能以数字开头
4. 标签名不能带有空格
5. 在一个xml文件中，有且仅有一个根标签

### 属性

语法： <Student name="eric">student</Student>

注意：

1）属性值必须以引号包含，不能省略，也不能单双引号混用！！！

2）一个标签内可以有多个属性，但不能出现重复的属性名！！！

### 文档申明

语法：<?xml version=”1.0” encoding=”utf-8”?>

version:版本号，目前都固定写成1.0

encoding:解析xml文件的时候，查询的码表

### 注释

<!-- xml注释 -->

xml申明之前不能有注释

### 转义字符

|  |  |
| --- | --- |
| 特殊字符 | 转义字符 |
| < | &lt; |
| > | &gt; |
| “ | &quot; |
| & | &amp; |
| 空格 | &nbsp; |

### CDATA块

可以让一些需要进行包含特殊字符的内容统一进行原样输出

语法：

<![CDATA[需要原样输出的内容]]>

例如：

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <student>  <name>张三</name>  <desc><![CDATA[<html><color></color></html>]]></desc>  </student> |

### 处理指令

告诉xml解析器，如何解析xml文件

语法：

<? xml-stylesheet type=”text/css” href=”1.css”?>

告诉xml解析器该xml文档引用了哪个css文件。写css样式的时候，使用对应标签名的标签选择器。

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <?xml-stylesheet type="text/css" href="1.css"?>  <students>  <student id="1">  <name>张三</name>  <desc><![CDATA[<html><color></color></html>]]></desc>  </student>  <student id="2">  <name>李四</name>  <desc>学习特别认真</desc>  </student>  </students> |

|  |
| --- |
| student {  color:red;  font-size:20px;  width:150px;  height:100px;  display:block;/\*以块状显示该标签\*/  margin-top:40px;  background-color:green;  } |

需要提取xml的内容，可以使用xml-stylesheet指令

## xml解析

目前有两种解析方式：

* DOM解析
* SAX解析

### 解析工具:

DOM(文档对象模型)解析工具:

1）JAXP （oracle-Sun公司官方）

2）JDOM工具（非官方）

3）Dom4j工具（非官方）

三大框架（默认读取xml的工具就是Dom4j）

.......

SAX解析工具：

Sax解析工具（oracle-sun公司官方） (

)

### dom解析原理

xml解析器一次性把整个xml文档加载进内存，然后在内存中构建一颗Document的对象树，通过Document对象，得到树上的节点对象，通过节点对象访问（操作）到xml文档的内容。

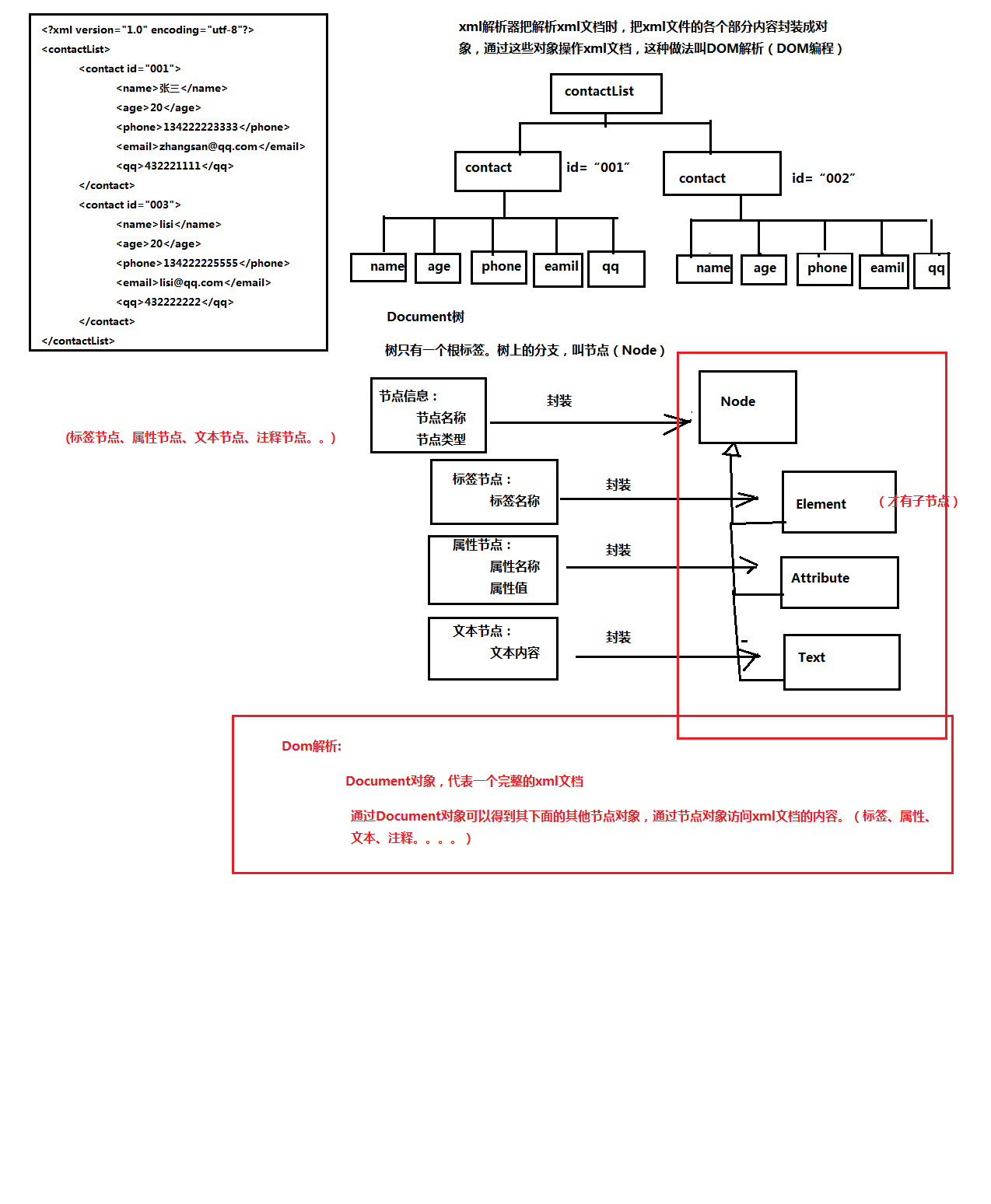
Node 节点可以分为以下几部分：

\* 1.标签节点 Element 它的名称为标签名称，值为标签的文本内容

\* 2.文本节点 Text 名称为null，值是文本内容

\* 3.注释节点 Comment 名称为null，值是注释内容

\* 4.属性节点 Attribute 名称为属性名称，值是属性值



### dom4j读取xml

1. 获取xml解析器 SAXReader saxReader=new SAXReader();
2. 获取dom对象 Document doc=saxReader.read(new File(“xml文件路径”));(注意，xml文件路径最好放到项目里面，写路径地址的时候，使用相对地址 ./表示当前项目路径)
3. nodeIterator() 获取当前document或者当前标签的所有子节点（注意：节点包括注释节点（注释节点name为null，值text为注释内容），标签节点，文本节点，属性节点）
4. elements():获取标签下面的所有子标签，返回list的标签集合
5. elementIterator():获取标签下面的所有子标签，返回标签迭代器
6. getRootElement() 获取当前文档的根标签
7. element("标签名字")获取某个标签下面指定名称的第一个标签对象
8. elementText("标签名字")获取某个标签下面指定名称的第一个标签的text内容
9. attribute("属性名称") 获取某个标签的指定名称的属性对象
10. attributeValue("属性名称") 获取某个标签的指定名称的属性对象值

#### 代码示例

|  |
| --- |
| **package** com.xml.dom.read;  **import** java.io.File;  **import** java.util.ArrayList;  **import** java.util.Iterator;  **import** java.util.List;  **import** org.dom4j.Attribute;  **import** org.dom4j.Document;  **import** org.dom4j.DocumentException;  **import** org.dom4j.Element;  **import** org.dom4j.Node;  **import** org.dom4j.io.SAXReader;  **import** org.junit.Test;  /\*\*  \* dom4j工具：第三方工具包  \*  \* 解析xml文件  \* **@author** Administrator  \*  \*/  **public** **class** Demo1 {    /\*\*  \* 获取xml文件的所有节点名称  \* nodeIterator  \*  \* Node 节点下面包含：  \* 1.标签 Element  \* 2.文本 Text  \* 3.注释 Comment  \* 4.属性 Attribute  \*  \*/  @Test  **public** **void** test1(){  //创建xml解析器对象  SAXReader saxReader=**new** SAXReader();  **try** {  //获取到document对象  Document doc=saxReader.read(**new** File("./src/student.xml"));  //nodeIterator() 获取当前document或者当前标签的所有子节点  Iterator<Node> nodeIt=doc.nodeIterator();  **while**(nodeIt.hasNext()){  Node node=nodeIt.next();//获取节点对象  System.*out*.println(node.getName());  System.*out*.println(node.getText());  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  **if**(node **instanceof** Element){  Element studentsEle=(Element)node;  Iterator<Node> studentsIt=studentsEle.nodeIterator();  **while**(studentsIt.hasNext()){  Node studentNode=studentsIt.next();  System.*out*.println(studentNode.getName());  }  }  }    } **catch** (DocumentException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    }  /\*\*  \* 获取标签下面的所有子标签  \*/  @Test  **public** **void** test2(){  //创建xml解析器对象  SAXReader saxReader=**new** SAXReader();  **try** {  //获取document对象  Document doc=saxReader.read(**new** File("./src/student.xml"));  //getRootElement() 获取当前document下面的跟标签  Element rootEle=doc.getRootElement(); //拿到document下面的根标签  System.*out*.println("根标签："+rootEle.getName());  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  //获取当前标签下面的所有子标签:elements()，elementIterator()  //Iterator<Element> studentItEle=rootEle.elementIterator();  getChildEle(rootEle);    } **catch** (DocumentException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    }    **public** **void** getChildEle(Element el){  List<Element> studentListEle=el.elements();  **for**(Element stu:studentListEle){  System.*out*.println(stu.getName());  getChildEle(stu);  }    }    /\*\*  \* 获取某个标签下面指定名称的第一个子标签 element("标签名")  \* 获取某个标签指定名称的属性 attribute("属性名字")  \*/  @Test  **public** **void** test3(){  //获取xml解析器对象  SAXReader saxReader=**new** SAXReader();  **try** {  //获取document对象  Document doc=saxReader.read(**new** File("./src/student.xml"));  Element rootElement=doc.getRootElement(); //获取doc的跟标签  //element("标签名字")获取某个标签下面指定名称的第一个标签  Element studentEle=rootElement.element("student");  System.*out*.println(studentEle.getName());  System.*out*.println(studentEle.getText());  //attribute("属性名称") 某个标签的指定名称的属性对象  //attributeValue("属性名称") 获取某个标签的指定名称的属性对象值  Attribute attribute=studentEle.attribute("id"); //student标签的id属性  System.*out*.println(attribute.getName()+"="+attribute.getText()); //getText()获取属性值    Element nameEle=studentEle.element("name");  System.*out*.println("name="+nameEle.getText());  System.*out*.println("desc="+studentEle.element("desc").getTextTrim());    } **catch** (DocumentException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }    @Test  **public** **void** test4(){  //获取xml解析器对象  SAXReader saxReader=**new** SAXReader();  List<Student> stuList=**new** ArrayList<Student>();  **try** {  //获取document对象  Document doc=saxReader.read(**new** File("./src/student.xml"));  Element rootElement=doc.getRootElement(); //获取doc的跟标签  List<Element> studentEleList=rootElement.elements(); //获取students下面的所有子标签  **for**(Element studentEle:studentEleList){  Student stu=**new** Student();  stu.setId(studentEle.attributeValue("id")); //id  //stu.setId(studentEle.attribute("id").getText()); //id  //stu.setName(studentEle.element("name").getText());//name  stu.setName(studentEle.elementText("name"));//name  stu.setAge(Integer.*valueOf*(studentEle.elementText("age")));//age  stu.setEmail(studentEle.element("email").getText()); //email  stu.setDesc(studentEle.element("desc").getText()); //desc  stuList.add(stu);  }    **for**(Student stu:stuList){  System.*out*.println(stu.toString());  }  } **catch** (DocumentException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

注意：使用dom4j工具解析xml需要导入dom4j的工具包

注意：使用junit测试，需要导入junit包。之后，在测试方法上声明@Test

### Dom4j操作xml文件

#### 新创建一个xml文件

1. DocumentHelper.*createDocument*() 创建document文档
2. addElement("Students"); 增加指定名称的标签
3. addAttribute("属性名", "属性值") 给标签新增指定的属性名称和属性值
4. nameEle.setText("文本内容"); 给标签设置指定的内容
5. OutputFormat.*createPrettyPrint*() 指定xml内容输出的文件格式：带有空格和换行，格式美观
6. OutputF*ormat.createCompactFormat()* 指定xml内容输出的文件格式：去掉空格和换行，一般用于正式的服务器上
7. format.setEncoding("utf-8"); //设置xml文件的编码和解析的编码
8. XMLWriter writer=**new** XMLWriter(outputStream, format);创建xml写入的对象
9. writer.write(doc); 写入document文档

|  |
| --- |
| @Test  **public** **void** test1(){  //创建Document文档  Document document=DocumentHelper.*createDocument*();  //创建students根标签  Element studentsEle=document.addElement("students");  //创建student标签，student标签是students的根标签  Element studentEle=studentsEle.addElement("student");  //给student标签增加id属性  studentEle.addAttribute("id", "1");      //创建student标签的子标签 name标签  Element nameEle=studentEle.addElement("name");  nameEle.setText("张林林");  //创建student标签的子标签 age标签  Element ageEle=studentEle.addElement("age");  ageEle.setText("20");  //创建student标签的子标签 email标签  Element emailEle=studentEle.addElement("email");  emailEle.setText("ss@163.com");  //创建student标签的子标签 desc标签  Element descEle=studentEle.addElement("desc");  descEle.setText("请添加对张林林的描述");  **try** {  //建立xml文件输出流  OutputStream outputStream=**new** FileOutputStream(**new** File("./src/student.xml"));  //设置输出的格式  OutputFormat format=OutputFormat.*createPrettyPrint*();//带有换行和空格，格式美观  //OutputFormat format=OutputFormat.createCompactFormat();//没有换行和空格，一般用于部署在服务器上的时候  format.setEncoding("utf-8"); //设置xml文件解析的编码格式  //创建xml writer  XMLWriter writer=**new** XMLWriter(outputStream, format);  writer.write(document);  writer.close();    } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    } |

#### 修改标签的属性或者文本值

1. attribute.setText("id2"); 给属性赋值
2. attribute.setValue("id2"); 给属性赋值 也可以通过增加同名属性的方法，修改属性值
3. element.setAttributeValue("id", "0002");给某个标签的指定属性赋值（已经弃用）
4. element.element("name").setText("ddddd") 给某个标签的文本设置值

|  |
| --- |
| @Test  **public** **void** test2(){  //创建xml解析器  SAXReader saxReader=**new** SAXReader();  //获取document文档  **try** {  Document doc=saxReader.read(**new** File("./src/student.xml"));  //获取根标签  Element rootEle=doc.getRootElement();  //获取students的第一个子标签 student  Element studentEle=rootEle.element("student");  //获取student下面标签名字为age的第一个子标签  Element ageEle=studentEle.element("age");  //修改age的文本值  ageEle.setText("18");    //修改student标签的id属性值为11  //studentEle.setAttributeValue("id","11");  Attribute idAttribute=studentEle.attribute("id");  //idAttribute.setText("11");  idAttribute.setValue("11");  //删除第一个student标签的id属性  idAttribute.detach();  //studentEle.remove(idAttribute);  //删除第一个student标签  studentEle.detach();  //rootEle.remove(studentEle);    //输出到xml文件  //建立输出流  FileOutputStream outputStream=**new** FileOutputStream(**new** File("./src/student.xml"));  //指定xml输出的格式  OutputFormat format=OutputFormat.*createPrettyPrint*();  //设置编码  format.setEncoding("utf-8");  //创建xml writer对象  XMLWriter writer= **new** XMLWriter(outputStream, format);  //将document对象写出到相应的xml文件里面  writer.write(doc);  writer.close();        } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    } |

#### 删除标签或者属性

rmElement.detach() 删除rmElement标签

element.remove(rmElement) 删除rmElement标签

attribute.detach() 删除attribute属性

element.remove(attribute); 删除element属性

案例

|  |
| --- |
| @Test  **public** **void** test2(){  //创建xml解析器  SAXReader saxReader=**new** SAXReader();  //获取document文档  **try** {  Document doc=saxReader.read(**new** File("./src/student.xml"));  //获取根标签  Element rootEle=doc.getRootElement();  //获取students的第一个子标签 student  Element studentEle=rootEle.element("student");  //获取student下面标签名字为age的第一个子标签  Element ageEle=studentEle.element("age");  //修改age的文本值  ageEle.setText("18");    //修改student标签的id属性值为11  //studentEle.setAttributeValue("id","11");  Attribute idAttribute=studentEle.attribute("id");  //idAttribute.setText("11");  idAttribute.setValue("11");  //删除第一个student标签的id属性  idAttribute.detach();  //studentEle.remove(idAttribute);  //删除第一个student标签  studentEle.detach();  //rootEle.remove(studentEle);    //输出到xml文件  //建立输出流  FileOutputStream outputStream=**new** FileOutputStream(**new** File("./src/student.xml"));  //指定xml输出的格式  OutputFormat format=OutputFormat.*createPrettyPrint*();  //设置编码  format.setEncoding("utf-8");  //创建xml writer对象  XMLWriter writer= **new** XMLWriter(outputStream, format);  //将document对象写出到相应的xml文件里面  writer.write(doc);  writer.close();        } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    } |

类 UserDao

//获取用户集合

getUserList(){

}

UserWrite

XXX()

//获取用户集合

创建一个xml文档，将集合里面的数据动态的写进去（要对应上相应标签和属性）

### xPath技术

XPath即为XML路径语言，它是一种用来确定[XML](http://baike.baidu.com/view/63.htm" \t "_blank)（[标准通用标记语言](http://baike.baidu.com/view/5286041.htm" \t "_blank)的子集）文档中某部分位置的语言。XPath基于XML的树状结构，提供在结构树中找寻节点的能力，它能够快速获取节点对象

#### Dom4j中使用xPath

1. 导入xPath支持jar包 : jaxen-1.1-beta-6.jar
2. 使用xpath方法

List<Node> selectNodes("xpath表达式"); 查询多个节点对象

Node selectSingleNode("xpath表达式"); 查询一个节点对象

#### xPath语法

* + - * / 绝对路径 表示从xml的根位置开始或子元素（一个层次结构）
      * // 相对路径  表示不分任何层次结构的选择元素。
      * \* 通配符 表示匹配所有元素
      * [] 条件 表示选择什么条件下的元素
      * @ 属性 表示选择属性节点
      * and 关系  表示条件的与关系（等价于&&）
      * text() 文本 表示选择文本内容

#### 案例

Xpath表达式练习

|  |
| --- |
| @Test  public void test1(){  SAXReader saxReader=new SAXReader();  String xpath="";  try {  Document doc=saxReader.read(new File("./src/student1.xml"));  // 1. / 绝对路径 表示从xml根位置或者子元素开始  xpath="/students"; //根标签 students  xpath="/students/student"; //根标签students下面的子标签student  //2. // 相对路径 不分任何层测结构  xpath="//name"; //所有的name标签  //3. \*通配符 表示匹配所有元素  xpath="/students/\*"; //students下面的所有子标签  xpath="/students//\*";//students下面的所有标签  //4. [] 条件 选择什么条件下的元素  xpath="//student[@id]"; //带有id属性的student标签  xpath="//student[@id='1']"; //id属性等于1的student标签  xpath="//student[3]"; //第三个student标签  xpath="//student[last()]"; //最后一个student标签  //5. @ 属性 表示选择属性节点  xpath="//@id"; //选择id属性节点对象，返回Attribute属性对象  xpath="//student[@id='2']"; //选择id属性值为2的student标签  xpath="//student[not(@id)]"; //选择不包含id属性的student标签  xpath="//student[@id='2' and @name='aa']"; //选择id属性值为2同时name属性值为aa标签  //6. text()表示选择文本内容  xpath="//name/text()"; //选择name标签下文本内容，返回Text对象  xpath="//student/name[text()='张林林']"; //选择student标签下name为张林林的name标签  xpath="/student";    List<Node> nodeList=doc.selectNodes(xpath);  for(Node node:nodeList){  System.out.println(node.getName());  }    } catch (DocumentException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |

1. 如下显示效果的html页面



页面结构如下：

|  |
| --- |
| <html>  <head>  <title>标签练习</title>  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />  </head>  <body>  <center><h1>学生班通讯录</h1></center>  <table border="1" align="center" id="contactForm" cellspacing="0">  <thead>  <tr><th>id</th><th>姓名</th><th>年纪</th><th>地址</th><th>性别</th></tr>  </thead>  <tbody>  <tr>  <td>001</td>  <td>杨震</td>  <td>18</td>  <td>北京</td>  <td>男</td>  </tr>  <tr>  <td>002</td>  <td>张林林</td>  <td>20</td>  <td>上海</td>  <td>男</td>  </tr>  <tr>  <td>002</td>  <td>黄蓉</td>  <td>12</td>  <td>深圳</td>  <td>女</td>  </tr>  </tbody>  </table>  </body>  </html> |

需求如下:通过xpath和dom4j的技术，实现读取该html文件，并且将标题和单元格内容都解析出来，单元格内容解析格式为：

001-杨震-18-北京-男

002-张林林-20-上海-男

002-黄蓉-12-深圳-女

|  |
| --- |
| @Test  **public** **void** test2(){  //创建xml解析器  SAXReader saxReader=**new** SAXReader();  **try** {  //获取document文档  Document document=saxReader.read(**new** File("./src/personList.html"));  String xpath="//h1";  Node node=document.selectSingleNode(xpath);  System.*out*.println("标题："+node.getText());  xpath="//tbody/tr";  List<Node> trNodeList=document.selectNodes(xpath);  **for**(Node trNode:trNodeList){  Element trEle=(Element)trNode;  System.*out*.print(((Element)trEle.elements("td").get(0)).getText());  System.*out*.print("-");  System.*out*.print(((Element)trEle.elements("td").get(1)).getText());  System.*out*.print("-");  System.*out*.print(((Element)trEle.elements("td").get(2)).getText());  System.*out*.print("-");  System.*out*.print(((Element)trEle.elements("td").get(3)).getText());  System.*out*.print("-");  System.*out*.print(((Element)trEle.elements("td").get(4)).getText());  System.*out*.println("");  }    } **catch** (DocumentException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  } |

### Sax解析

#### Sax解析和dom解析的区别

* DOM解析原理：

一次性把xml文档加载进内存，然后在内存中构建Document树。

对内存要求比较严。

缺点： 不适合读取大容量的xml文件，容易导致内存溢出。

* SAX解析原理： 加载一点，读取一点，处理一点。对内存要求比较低

|  |  |
| --- | --- |
| **DOM解析** | **SAX解析** |
| 原理： 一次性加载xml文档，不适合大容量的文件读取 | 原理： 加载一点，读取一点，处理一点。适合大容量文件的读取 |
| DOM解析可以任意进行增删改查 | SAX解析只能读取 |
| DOM解析任意读取任何位置的数据，甚至往回读 | SAX解析只能从上往下，按顺序读取，不能往回读 |
| DOM解析面向对象的编程方法（Node，Element，Attribute）,Java开发者编码比较简单。 | SAX解析基于事件的编程方法。java开发编码相对复杂。 |

#### SAX解析工具

* SAX解析工具- Sun公司提供的。内置在jdk中。org.xml.sax.\*
* SAXParser类： 用于读取和解析xml文件对象
* parse（[File](mk:@MSITStore:E:\\study\\API帮助文档\\jdk6.chm::/j2se6/api/java/io/File.html" \o "java.io 中的类) f, [DefaultHandler](mk:@MSITStore:E:\\study\\API帮助文档\\jdk6.chm::/j2se6/api/org/xml/sax/helpers/DefaultHandler.html" \o "org.xml.sax.helpers 中的类) dh）方法： 解析xml文件

参数一： File：表示 读取的xml文件。

参数二： DefaultHandler： SAX事件处理程序。使用DefaultHandler的子类

使用sax解析xml的步骤：

* 创建SAXParser对象：

SAXParser saxParser=SAXParserFactory.newInstance().newSAXParser();

* 调用parse方法  
   parser.parse(new File("./src/contact.xml"), new MyDefaultHandler());

|  |
| --- |
| @Test  **public** **void** test1(){  **try** {  //创建sax解析器 SAXParser 对象  SAXParser saxParser=SAXParserFactory.*newInstance*().newSAXParser();  MyHandle1 handle=**new** MyHandle1();  saxParser.parse(**new** File("./src/student.xml"),handle );  String content=handle.getContent();  System.*out*.println(content);  } **catch** (ParserConfigurationException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (SAXException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**catch**(IOException e){  e.printStackTrace();  }    } |

DefaultHandler类的API:

* void startDocument() : 文档开始的通知
* void endDocument() ：文档结束通知
* void startElement(String uri, String localName, String qName, Attributes attributes) ：标签开始通知
* void endElement(String uri, String localName, String qName) ：标签结束通知
* void characters(char[] ch, int start, int length) ： 标签中文本数据的通知

案例

|  |
| --- |
| package com.xml.sax;  import org.xml.sax.Attributes;  import org.xml.sax.SAXException;  import org.xml.sax.helpers.DefaultHandler;  public class MyHandle extends DefaultHandler {  /\*\*  \* 接收文档开始的通知  \*/  @Override  public void startDocument() throws SAXException {  // TODO Auto-generated method stub  System.out.println("startDocument----");  }    /\*\*  \* 接收标签开始的通知  \*/  @Override  public void startElement(String uri, String localName, String qName,  Attributes attributes) throws SAXException {  // TODO Auto-generated method stub  System.out.println("startElement----");  }    /\*\*  \* 接收标签结束的通知  \*/  @Override  public void endElement(String uri, String localName, String qName)  throws SAXException {  // TODO Auto-generated method stub  System.out.println("endElement----");  }    /\*\*  \* 接收 元素字符数据的通知  \*/  @Override  public void characters(char[] ch, int start, int length)  throws SAXException {  // TODO Auto-generated method stub  System.out.println("characters----");  }    /\*\*  \* 接收 文档结束的通知  \*/  @Override  public void endDocument() throws SAXException {  // TODO Auto-generated method stub  System.out.println("endDocument----");  }  } |

通过sax打印出完整的xml结构

|  |
| --- |
| package com.xml.sax;  import org.xml.sax.Attributes;  import org.xml.sax.SAXException;  import org.xml.sax.helpers.DefaultHandler;  public class MyHandle1 extends DefaultHandler {  private StringBuffer sb=new StringBuffer();    public String getContent(){  return sb.toString();  }    /\*\*  \* 接收文档开始的通知  \*/  @Override  public void startDocument() throws SAXException {  // TODO Auto-generated method stub  }    /\*\*  \* 接收标签开始的通知  \*/  @Override  public void startElement(String uri, String localName, String qName,  Attributes attributes) throws SAXException {  // TODO Auto-generated method stub  sb.append("<"+qName);  int len=attributes.getLength();// 当前标签的属性个数  for(int i=0;i<len;i++){  String attributeName=attributes.getQName(i); //属性名称  String attributeValue=attributes.getValue(i); //属性值  sb.append(" "+attributeName+"='"+attributeValue+"'");  }  sb.append(">");  }    /\*\*  \* 接收标签结束的通知  \*/  @Override  public void endElement(String uri, String localName, String qName)  throws SAXException {  // TODO Auto-generated method stub  sb.append("</"+qName+">");  }    /\*\*  \* 接收 元素字符数据的通知  \*/  @Override  public void characters(char[] ch, int start, int length)  throws SAXException {  // TODO Auto-generated method stub  String currentContent=new String(ch,start,length);  sb.append(currentContent);    }    /\*\*  \* 接收 文档结束的通知  \*/  @Override  public void endDocument() throws SAXException {  // TODO Auto-generated method stub  }    } |

## XML约束

### 引入

XML语法： 规范的xml文件的基本编写规则。（由w3c组织制定的）

XML约束： 规范XML文件数据内容格式的编写规则。（由开发者自行定义）

### XML约束技术

DTD约束：语法相对简单，功能也相对简单。学习成本也低。

Schema约束：语法相对复杂，功能也相对强大。学习成本相对高！！！（名称空间）

### DTD约束

1)导入dtd方式

* 内部导入

|  |
| --- |
| <!DOCTYPE note [  <!ELEMENT note (to,from,heading,body)>  <!ELEMENT to (#PCDATA)>  <!ELEMENT from (#PCDATA)>  <!ELEMENT heading (#PCDATA)>  <!ELEMENT body (#PCDATA)>  ]> |

* 外部导入

本地文件系统：

|  |
| --- |
| <!DOCTYPE note **SYSTEM** "note.dtd"> |

* 公共的外部导入：

|  |
| --- |
| <!DOCTYPE 根元素 **PUBLIC** "http://gz.itcast.cn/itcast.dtd"> |

2）DTD语法

约束标签

<!ELEMENT 元素名称 类别> 或 <!ELEMENT 元素名称 (元素内容)>

类别：

空标签： EMPTY。 表示元素一定是空元素。

普通字符串： (#PCDATA)。表示元素的内容一定是普通字符串（不能含有子标签）。

任何内容： ANY。表示元素的内容可以是任意内容（包括子标签）

(元素内容):

顺序问题：

<!ELEMENT 元素名称 (子元素名称 1,子元素名称 2,.....)>： 按顺序出现子标签

次数问题：

标签 ： 必须且只出现1次。

标签+ ： 至少出现1次

标签\* ： 0或n次。

标签？ ： 0 或1次。

约束属性

<!ATTLIST 元素名称 属性名称 属性类型 默认值>

默认值：

#REQUIRED 属性值是必需的

#IMPLIED 属性不是必需的

#FIXED value 属性不是必需的，但属性值是固定的

属性类型：控制属性值的

CDATA ：表示普通字符串

(en1|en2|..)： 表示一定是任选其中的一个值

ID：表示在一个xml文档中该属性值必须唯一。值不能以数字开头

### Schema约束

* Schema和dtd的区别：
  + XML Schema符合XML语法结构。
  + DOM、SAX等XML API很容易解析出XML Schema文档中的内容。
  + XML Schema对**名称空间**支持得非常好。
  + XML Schema比XML DTD支持更多的数据类型，并支持用户自定义新的数据类型。
  + XML Schema定义约束的能力非常强大，可以对XML实例文档作出细致的语义限制。
  + XML Schema不能像DTD一样定义实体，比DTD更复杂，但Xml Schema现在已是w3c组织的标准，它正逐步取代DTD。

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <xs:schema xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*  targetNamespace=*"http://www.jiyun.cn"*  elementFormDefault=*"qualified"*>  <xs:element name=*'书架'* >  <xs:complexType>  <xs:sequence maxOccurs=*'unbounded'* >  <xs:element name=*'书'* >  <xs:complexType>  <xs:sequence>  <xs:element name=*'书名'* type=*'xs:string'* />  <xs:element name=*'作者'* type=*'xs:string'* />  <xs:element name=*'售价'* type=*'xs:integer'* />  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:sequence>  </xs:complexType>  </xs:element>  </xs:schema> |

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <itcast:书架 xmlns:itcast=*"http://www.jiyun.cn"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.jiyun.cn demo.xsd"*>  <itcast:书>  <itcast:书名>JavaScript网页开发</itcast:书名>  <itcast:作者>张天书</itcast:作者>  <itcast:售价>28</itcast:售价>  </itcast:书>  </itcast:书架> |

* 名称空间：

在XML Schema中，每个约束模式文档都可以被赋以一个唯一的名称空间，名称空间用一个唯一的URI（Uniform Resource Identifier，统一资源标识符）表示

如下，在schema文件中，申明如下

|  |
| --- |
| <xs:schema xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*  targetNamespace=*"http://www.w3school.com.cn"*  xmlns=*"http://www.w3school.com.cn"*  elementFormDefault=*"qualified"*> |

targetNamespace元素用于指定schema文档中声明的元素属于哪个名称空间。

elementFormDefault元素用于指定，该schema文档中声明的根元素及其所有子元素都属于targetNamespace所指定的名称空间。

为了在一个XML文档中声明它所遵循的Schema文件的具体位置，通常需要在Xml文档中的根结点中使用schemaLocation属性来指定，例如

|  |
| --- |
| <?xml version=*"1.0"*?>  <note xmlns=*"http://www.w3school.com.cn"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.w3school.com.cn note.xsd"*>  <to>George</to>  <from>John</from>  <heading>Reminder</heading>  <body>Don't forget the meeting!</body>  </note> |

schemaLocation此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置，两者之间用空格分隔。

注意，在使用schemaLocation属性时，也需要指定该属性来自哪里。

在一个xml文档中，不同的标签可以受到不同的schema文档的约束。

1. 一个名称空间受到schema文档约束的情况

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <itcast:书架 xmlns:itcast=*"http://www.jiyun.cn"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.jiyun.cn book.xsd"*>  <itcast:书>  <itcast:书名>JavaScript网页开发</itcast:书名>  <itcast:作者>张天书</itcast:作者>  <itcast:售价>28</itcast:售价>  </itcast:书>  </itcast:书架> |

2）使用名称空间引入多个schema文档约束的情况

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <书架 xmlns="http://www.it315.org/xmlbook/schema"  xmlns:demo="http://www.it315.org/demo/schema"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.it315.org/xmlbook/schema http://www.it315.org/xmlbook.xsd  <http://www.it315.org/demo/schema> http://www.it315.org/demo.xsd">  <书>  <书名>JavaScript网页开发</书名>  <作者>张天书</作者>  <售价 demo:币种=”人民币”>28.00元</售价>  </书>  </书架> |

3）默认名称空间的情况

|  |
| --- |
| <书架 xmlns="http://www.it315.org/xmlbook/schema"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation=“http://www.it315.org/xmlbook/schema book.xsd">  <书>  <书名>JavaScript网页开发</书名>  <作者>张天书</作者>  <售价>28.00元</售价>  </书>  <书架> |

4）没有名称空间的情况

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <书架 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="xmlbook.xsd">  <书>  <书名>JavaScript网页开发</书名>  <作者>张天书</作者>  <售价>28.00元</售价>  </书>  </书架> |

# mysql数据库

## 引入

* 数据保存到内存：

优点：

1）读写非常快

缺点：

1）程序关闭导致数据丢失

* 数据保存到文件：

优点：

1）数据可以永久保存

缺点：

1）频繁地IO操作，效率不高！

2）数据管理不方便。例如查询某个数据需要全部读取出来，再匹配。

* 数据保存到**数据库软件**：

优点：

1）数据永久保存下来

2）数据管理非常方便。（例如查询非常快速和方便）

## 数据库概念

数据库，俗称数据的仓库。方便管理数据的软件（或程序）

目前常用以下几种数据库：

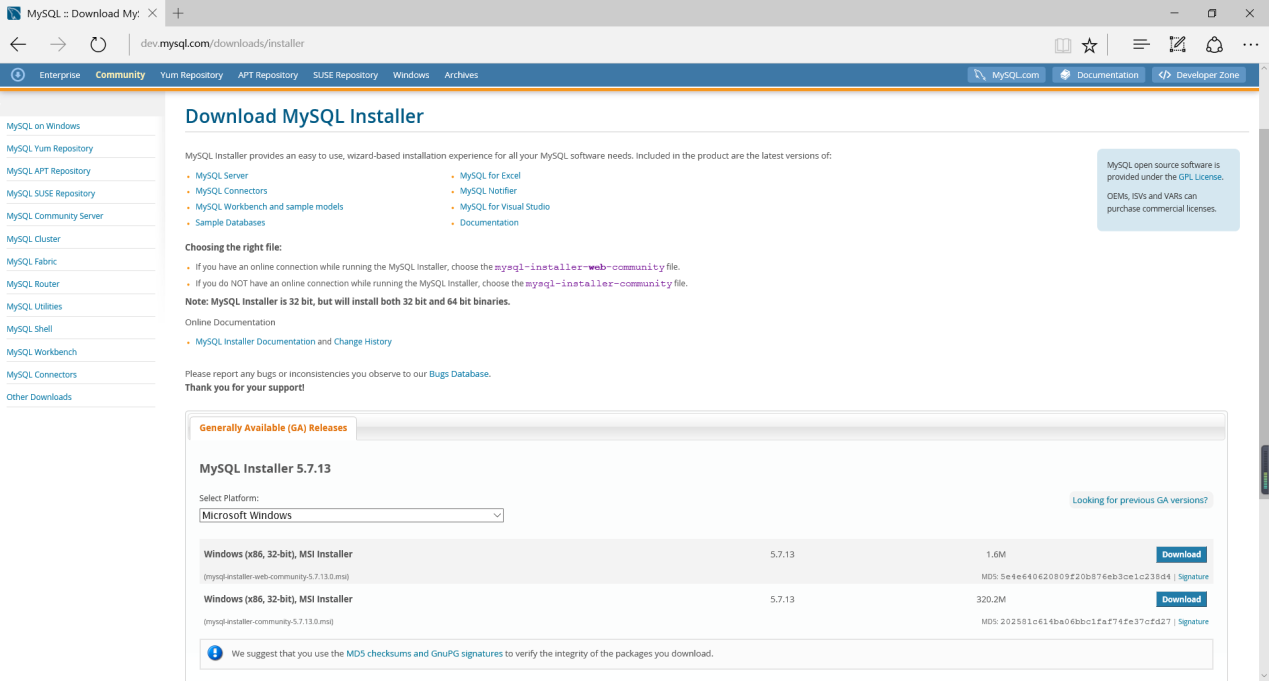
* Oracle，甲骨文公司的产品。 当前最流行应用最广泛的数据库软件。和java语言兼容非常好。适合中大型，中大应用。
* SQL Server: 是微软公司的产品。window平台应用非常广泛。和c#，net平台兼容非常好。
* DB2： IBM公司的产品。IBM服务器--> UNIX -> DB2- > Websphere
* MySQL: 开源组织的产品。甲骨文公司的产品。免费！！！和java语言兼容非常好！适合中小企业，中小应用

## Mysql使用

### 官网下载：

下载地址：<http://dev.mysql.com/downloads/installer/>

目前windows最高版本是5.7.13



### 安装mysql

Mysql默认的端口号是：3306

Mysql默认的管理员用户是：root

Sql脚本语句文件是以.sql结尾的

### mysql常用命令

* 连接本机数据库： mysql –u root –p 回车
* 连接远程数据库：mysql –h ip地址 –u root –p
* 数据库退出，使用 exit; quit
* 切换数据库：use 数据库名
* 更改用户密码：

mysql>use mysql;  
 mysql> UPDATE user SET password=PASSWORD("new password") WHERE user='username';  
 mysql> FLUSH PRIVILEGES;  
 mysql> exit;

* 显示表：show tables;
* 表的详细描述：describe tablename; des tablename
* 刷新数据库：flush privileges

## 数据库管理

要连接到一个数据库，有以下几大要素：

数据库服务器ip地址

数据库用户名

数据库密码

Mysql端口号 3306

数据库名称

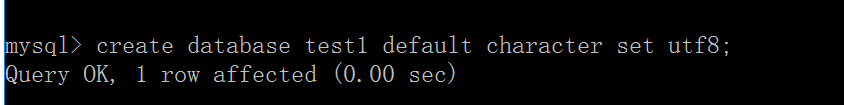
### 查询所有数据库

show databases;

|  |
| --- |
| mysql> show databases;  +--------------------+  | Database |  +--------------------+  | information\_schema | -- mysql元数据，基础数据  | mysql | -- mysql配置数据库，其中包含用户信息。（用户名和密码，权限管理）  | performance\_schema | -- mysql数据库软件的运行数据，日志信息，性能数据  | test | -- 测试数据库  | world |  +--------------------+  6 rows in set (0.00 sec) |

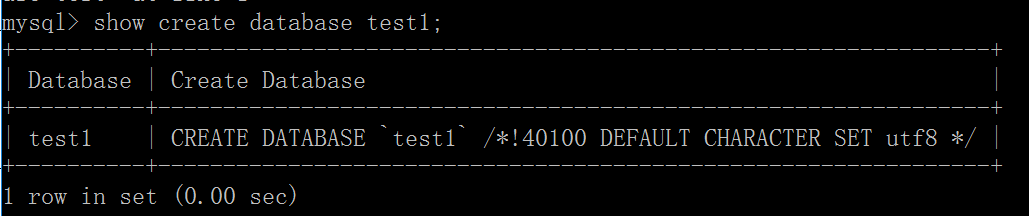
### 创建数据库

|  |
| --- |
| Create database 数据库名 default character set 字符集 |



查看数据库新创建的信息或者默认字符集

|  |
| --- |
| Show create database test; |



### 修改数据库

|  |
| --- |
| alter database 数据库名 default character set gbk; |

### 删除数据库

|  |
| --- |
| drop database 数据库名 |

### 创建表

|  |
| --- |
| -- 表代表着一类事物，表当中的每一列都代表着该事物的一个属性  创建表：创建一个学生表，存储学生信息：id,学生姓名，学生年龄，性别，所在地区，学生机试分数  create table tbl\_student(  id int, -- 主键id  stuName varchar(25), -- 学生姓名  age int,-- 学生年龄  gender char(1) ,-- 学生性别  city varchar(25)  ); |

### 查看表结构

|  |
| --- |
| -- 查看表结构  desc tbl\_student; |

### 删除表

|  |
| --- |
| -- 删除表：  drop table tbl\_student; |

### 修改表

1）添加字段

|  |
| --- |
| alter table tbl\_student add column operateScroe int;-- 添加一个字段 operateScore |

2）删除字段

|  |
| --- |
| alter table tbl\_student drop column opeScore; -- 删除字段 |

3）修改字段类型

|  |
| --- |
| alter table tbl\_student modify column opeScore double; -- 修改字段的数据类型 |

4）修改字段名称

|  |
| --- |
| alter table tbl\_student change column operateScroe opeScore int; -- 修改字段名称 |

5）修改表名称

|  |
| --- |
| alter table tbl\_stu rename to tbl\_student; -- 修改表的名称 |

## 数据管理

### 新增数据

|  |
| --- |
| -- 向表中插入数据  insert into tbl\_student values(1,'张三',12,'男','海淀区',90,60); -- 给所有字段按照顺序插入数据  insert into tbl\_student values(2,'李四',15,'男','昌平区',90,70); -- 给所有字段按照顺序插入数据  insert into tbl\_student values(3,'王五',18,'女','昌平区',90,80); -- 给所有字段按照顺序插入数据  insert into tbl\_student values(4,'赵六',18,'女','昌平区',90,90); -- 给所有字段按照顺序插入数据  insert into tbl\_student values(5,'葛尔丹',10,'女','朝阳区',50,60);  insert into tbl\_student(id,stuname,age,gender,opeScore,writteScore) values(6,'葛二蛋',8,'女',70,78);  insert into tbl\_student values(7,'小菲',10,'女','',50,60);  insert into tbl\_student(id,stuname,age,gender) values(2,'李四',18,'女'); -- 向指定字段插入数据  insert into tbl\_student values(5,'葛优',22,'女','朝阳区',80,60); |

### 修改数据

|  |
| --- |
| -- 修改数据  update tbl\_student set age=16,opeScore=98 where id=2; -- 根据条件修改数据 |

### 删除数据

|  |
| --- |
| -- 删除数据  -- delete from tbl\_student; -- 删除整张表的数据  delete from tbl\_student where id=2; -- 根据条件删除数据,不能删除约束，删除数据之后可以回滚  truncate table tbl\_student; -- 不能加条件，会删除掉数据的同时，删除掉表的约束。不建议使用，删除之后不可以回滚 |

## 数据查询

### 查询所有列

|  |
| --- |
| -- 查看表中的数据  select \* from tbl\_student; -- 查看表中的所有字段数据 \*  select stuname, opeScore from tbl\_student; -- 查看指定列的数据 |

### 查询指定列

|  |
| --- |
| -- 查看表中的数据  select \* from tbl\_student; -- 查看表中的所有字段数据 \*  select stuname, opeScore from tbl\_student; -- 查看指定列的数据 |

### 查询时添加常量列

|  |
| --- |
| -- 查询的时候添加常量列  select stuName,opeScore,'中国' from tbl\_student;  -- 给列名起别名 as 关键字 （as也可以省略），也可以给表起别名  select stuName,opeScore,'中国' as country from tbl\_student; |

### 查询时合并列

|  |
| --- |
| -- 查看id为1的学生的考试总分  select id,stuName ,(opeScore+writteScore) as totalScore from tbl\_student where id=1; -- 查询时合并列 |

### 查询时去除重复记录

|  |
| --- |
| -- 去除重复记录 ： 查看学生所在的区域  select distinct city from tbl\_student;  select distinct(city) from tbl\_student; |

## 条件查询

### 逻辑条件： and(与) or(或)

|  |
| --- |
| /\* 条件查询 逻辑条件用 and or \*/  -- 查询出年龄为15岁，同时在昌平区的学生  select \* from tbl\_student where age=15 and city='昌平区';  -- 查询单科及格的学生：机试成绩>80 或者 笔试成绩>80  select \* from tbl\_student where opeScore>80 or writteScore>80; |

### 比较条件

Sql语言中有以下比较条件：

* 大于：>
* 小于: <
* 大于等于: >=
* 小于等于: <=
* 等于:=
* 不等于:<>
* 在XX和XX之间：between XX and XXX 等价于大于等于XX，小于等于XXX

|  |
| --- |
| -- 比较条件 > >= < <= = <>不等于 (between xxx and xxx )>=且<=  -- 查看笔试成绩在80分到90分之间的学生信息  -- select \* from tbl\_student where writteScore>=80 and writteScore<=90;  select \* from tbl\_student where writteScore between 80 and 90; -- 包括两个临界值 |

### In

|  |
| --- |
| -- 字段值 in (值1,值2,值3......)  -- 查看在海淀区或者昌平区或者朝阳区的学生  select \* from tbl\_student where city in ('海淀区','昌平区','朝阳区'); |

### 判空条件

|  |
| --- |
| -- 查看地址不为空的学生  -- 判空条件 is null is not null 之时判断是否是null，跟空字符串''为关  select \* from tbl\_student where city<>'' and city is not null; |

### 模糊查询 like

|  |
| --- |
| -- 模糊查询 like %匹配多个字符 \_匹配一个字符  -- 查找姓葛的学生  select \* from tbl\_student where stuName like '葛%';  -- 查找名字里面带有葛的学生  select \* from tbl\_student where stuName like '%葛%';  -- 查找姓葛且名字是3个字的学生  select \* from tbl\_student where stuName like '葛\_\_'; |

## 聚合查询

常用的聚合函数：

总和：sum()0

平均:avg()

最大：max()

最小:min()

总个数：count()

|  |
| --- |
| /\*\* 聚合查询 count() max() min() sum() avg() \*\*/  -- 查看总共有多少个学生 count 8  select count(\*) from tbl\_student; -- 8个 一般不建议大家放\*，\*效率比较低  select count(city) from tbl\_student; -- 7个 count(字段) 会过滤掉null值的字段  select count(id) from tbl\_student; -- 建议用count(主键)  -- 查看机试分数最高的学生 max()  select max(opescore),stuname from tbl\_student;  -- 找机试分数最低的学生 min()  select min(opescore),stuname from tbl\_student;  -- 获取学生的机试总分 sum() 求和函数  select sum(opescore) from tbl\_student;  -- 获取学生机试成绩平均分 avg()  select avg(opescore) from tbl\_student; |

## 分页查询(limit 起始行，查询几行)

|  |
| --- |
| -- 分页查询 limit 参数1 参数2  -- 参数1：查询的起始索引位置  -- 参数2：每页查询的条数  -- 查询索引从0开始，查出6条数据  select \* from tbl\_student limit 0,6 ; |

Limit只有一个参数的时候，代表的每页查询的条数。

## 查询排序

|  |  |
| --- | --- |
| --order by  -- 语法 ：order by 字段 asc/desc  -- asc: 顺序，正序。数值：递增，字母：自然顺序（a-z）  -- desc: 倒序，反序。数值：递减，字母：自然反序(z-a)  -- 默认情况下，按照插入记录顺序排序  SELECT \* FROM student;  -- 查询排序 order by 字段 asc升序 desc 降序  -- 将学生按照笔试成绩从高到低排序  select \* from tbl\_student order by writteScore desc ;  select \* from tbl\_student order by writteScore desc,opeScore desc ; -- 多个字段排序，中间使用逗号分隔 |  |

## 分组查询（group by）

|  |
| --- |
| -- 分组 group by  -- 查看每个区有几个学生  select city,count(id) from tbl\_student group by city; |

## 分组查询后筛选

|  |
| --- |
| -- 分组后的条件筛选 使用having关键字:对分组之后的结果再进行筛选。where不能用于group by之后，它只能用于分组之前  -- 查看学生人数大于2的地区  select city,count(id) from tbl\_student group by city having count(id)>2; -- where 不能放在group by后面 |

## 数据约束

### 什么是数据约束

对用户操作的表的数据进行约束。

### 默认值

默认值：当字段没有插入值的时候，mysql自动给该字段分配指定的默认值。使用 default关键字

注意：

1）对默认值字段插入null是可以的。

2）对默认值字段可以插入非null

|  |
| --- |
| -- 默认值约束 ：当字段没有插入值的时候，mysql自动给该字段分配指定的默认值  -- 使用关键字 default  create table tbl\_user (  id int ,  userName varchar(25),  pwd varchar(25) default '123456', -- 给密码添加默认值约束  loginTime timestamp  ); |

### 非空

非空约束：not null 当定义字段的时候，对该字段进行非空约束，表示该字段插入值的时候，不能为Null，必须赋值(可以赋值空字符串)

|  |
| --- |
| -- 非空约束:not null 不允许字段为null，允许为空字符串  drop table tbl\_user;  create table tbl\_user (  id int ,  userName varchar(25),  pwd varchar(25) default '123456', -- 给密码添加默认值约束  address varchar(30) not null -- 给address添加不能为null的约束  ); |

### 唯一

Unique :使用唯一约束的字段，插入的值不能重复。

注意：

1. 唯一的子弹可以插入null
2. 唯一的字段可以插入多个null，唯一约束对Null没有约束

|  |
| --- |
| -- 唯一约束unique ，控制同一个字段的不同行的值不能重复  drop table tbl\_user;  create table tbl\_user (  id int unique , -- 给id加上唯一约束  userName varchar(25),  pwd varchar(25) default '123456', -- 给密码添加默认值约束  address varchar(30) not null -- 给address添加不能为null的约束  ); |

### 主键(非空+唯一)

主键约束：primary key 对一个字段加上主键约束，控制字段的值不能为null，而且必须唯一，不能重复。

注意：

主键字段，不能插入null，但是可以插入空字符串””，如果主键字段是int类型的，插入空字符串的时候，实际插入的是0值。但是因为有唯一约束，所以不能重复插入空字符串。

|  |
| --- |
| -- 主键约束 primary key ：字段值不能重复同时不能为空  drop table tbl\_user;  create table tbl\_user (  id int primary key , -- 给id加上主键约束约束，让id值唯一且不为null，可以为''  userName varchar(25),  pwd varchar(25) default '123456', -- 给密码添加默认值约束  address varchar(30) not null -- 给address添加不能为null的约束  ); |

### 自增长

自增长： auto\_increment

设置自增长之后的字段，可以不赋值，默认的自动增长1

注意：

1. 自增长只能用于数据类型的字段
2. 自增长如果用于小数点类型的字段，当该字段不赋值的时候，仍然是按照整数在递增的。
3. 自增字段只能被设置在主键字段上
4. 可以使用zerofill 来设置int类型的字段增长的时候的格式：使用0来填充剩下的位数。zerofill需要紧跟在int类型的后面

|  |
| --- |
| create table tbl\_class(  id int primary key auto\_increment,  className varchar(20),  classNum varchar(25)  );  insert into tbl\_class(className,classNum) values('大数据C1607G','507');  insert into tbl\_class(className,classNum) values('大数据C1606A','503');  insert into tbl\_class(className,classNum) values('大数据C1607H','708');  select \* from tbl\_class;  drop table tbl\_student;  create table tbl\_student(  id int primary key auto\_increment,  stuName varchar(25)not null,  age int default 23,  gender char(1) default '男',  classId int, -- classId 外键  constraint fk\_student\_class foreign key(classId) references tbl\_class(id) -- 外键约束  ); |

### 外键约束

外键约束：约束两张表的数据。

外键定义语法： constraint 外键约束名称 foreign key(外键字段) references 主表名称(引用字段)

被约束的表叫做副表，外键设置在副表上面，外键引用字段所在的表叫做主表。

注意：

1. 当有外键约束之后，添加数据的时候，先添加主表数据，再添加副表数据
2. 当有了外键约束，修改数据的时候，先改副表的数据，在修改主表的数据。
3. 当有了外键约束，删除数据的时候，也是先删除副表的数据，再删除主表的数据

|  |
| --- |
| -- 创建主表  create table tbl\_city (id int primary key,cityName varchar(25) );  -- 创建副表  create table tbl\_student(  id int primary key auto\_increment,  stuname varchar(25),  cityId int ,  -- 声明一个外键约束  constraint student\_city\_fk foreign key (cityId) references tbl\_student(id)  ); |

### 级联操作

问题： 当有了**外键约束**的时候，必须先修改或删除副表中的所有关联数据，才能修改或删除主表！但是，我们希望直接修改或删除主表数据，从而影响副表数据。可以使用级联操作实现！！！

级联修改：on update cascade

级联删除：on delete cascade

注意：级联操作必须在外键的基础上使用

|  |
| --- |
| -- 创建副表  create table tbl\_student(  id int primary key auto\_increment,  stuname varchar(25),  cityId int ,  -- 声明一个外键约束  constraint student\_city\_fk foreign key (cityid) references tbl\_city(id) on update cascade on delete cascade  ); |

## 数据库设计

### 引入

需求分析-需求设计(数据库设计，数据字典)-详细设计

### 设计的三大范式

设计原则： 建议设计的表尽量遵守三大范式。

第一范式：确保每个字段的原子性， 要求表的**每个字段必须是不可分割的独立单元**

学生表 : 姓名 -- 违反第一范式

露西|lucy

第二范式： 在第一范式的基础上，要求每张表只描述一件事情。表的每个字段都和表的主键有依赖。

学生表: 学生id 学生姓名 学生年龄 黄金价格 --违反第二范式

第三范式： 在第二范式基础，要求每张表的主键之外的其他字段都直接依赖主键。

学生表：学生id 学生姓名 学生年龄 老师编号

老师表：老师编号 老师姓名 老师工资

### 数据库设计案例

实现用户购物功能

* 用户一次性可以购买多件商品，每购买一次生成一个订单，订单包括：购买人姓名，购买时间，总金额，总件数，支付状态，收货信息
* 每件商品有商品编号，商品名称，商品原价格，商品现价
* 收货人信息包括：收件人姓名，电话，详细地址，邮政编码

子订单表

|  |  |  |  |
| --- | --- | --- | --- |
| 子订单编号1 | 商品1 | 2 | 订单编号1 |
| 子订单编号2 | 商品2 | 1 | 订单编号1 |
| 子订单编号3 | 商品3 | 1 | 订单编号1 |
| 子订单编号4 | 商品4 | 1 | 订单编号1 |
| 子订单编号5 | 商品5 | 1 | 订单编号1 |
|  |  |  | 订单编号2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 订单编号1 | 2016，8，15 | 总金额：500 | 总件数 6 | 购买用户 | 收获信息id |
| 订单编号2 | .。。 |  |  |  |  |

|  |  |
| --- | --- |
| 商品1 | 价格 |
|  |  |

|  |
| --- |
| Id，收货人姓名 电话 详细地址 邮政编码 |

## 关联查询

### 内连接查询

需求：查询学生及其所在城市(显示学生姓名，所在城市名称)

内连接两种形式

Select a.\*,b.\* from A,B where A.id=B.id;

Select a.\*,b.\* from A inner join B on A.id=B.id

### 左[外]连接

-- 需求： 查询每个城市的学生

-- 预期结果：

左[外]连接查询： 使用左边表的数据去匹配右边表的数据，如果符合连接条件的结果则显示，如果不符合连接条件则显示null

（注意： 左外连接：左表的数据一定会完全显示！）

左外连接语法：Left [outer] join 表 on 关联条件

### 右[外]连接

右[外]连接查询: 使用右边表的数据去匹配左边表的数据，如果符合连接条件的结果则显示，如果不符合连接条件则显示null

-- （注意： 右外连接：右表的数据一定会完成显示！）

右外连接语法：right [outer] join 表 on 关联条件

### 自连接

需求：查询学生及学生的班长

### 子查询

|  |
| --- |
| -- 查看所有的学生信息，及他所属的班级信息  select tbl1.\*,(select tbl2.className from tbl\_class tbl2 where tbl2.id=tbl1.classId) from  tbl\_student tbl1 |

## 存储过程

### 什么是存储过程

存储过程：带有业务逻辑的sql语句。里面有流程控制语句 if条件判断，while循环等等

### 存储过程的特点

1）执行效率非常快！存储过程是在数据库的服务器端执行的！！！

2）移植性很差！不同数据库的存储过程是不能移植。

### 存储过程语法

创建存储过程 :create procedure 存储过程名字 (参数)

调用存储过程 call 存储过程名字(参数)

删除存储过程 drop procedure 存储过程名字

定义存储过程的时候，参数定义：

* IN： 表示输入参数，可以携带数据到存储过程中
* OUT： 表示输出参数，可以从存储过程中返回结果
* INOUT： 表示输入输出参数，既可以输入功能，也可以输出功能

#### 输入参数的存储过程

|  |
| --- |
| /\*创建带有参数的存储过程\*/  delimiter $ -- 声明存储过程的结束符  create procedure pro\_test(in stuid int)  begin  select \* from tbl\_student where id=stuid;  end $;  call pro\_test(1);-- 调用存储过程  drop procedure pro\_test; -- 删除存储过程 |

#### 输出参数的存储过程

|  |
| --- |
| -- 带参数的存储过程 输出参数  delimiter $ -- 申明存储过程的结束符号  -- 创建不带参数的存储过程  create procedure pro\_test3(out num1 int)  begin  set num1=3;  end $  call pro\_test3(@num2);  select @num2; -- 查询会话参数值 |

#### 输入输出参数的存储过程

|  |
| --- |
|  |

#### 条件判断的存储过程

需求：输入一个整数，如果1，则返回“星期一”,如果2，返回“星期二”,如果3，返回“星期三”。其他数字，返回“错误输入”

|  |
| --- |
| delimiter $  create procedure pro\_test3(in num int,out str varchar(20))  begin  if num=1 then  set str='星期一';  elseif num=2 then  set str='星期二';  else  set str='输入错误';  end if;  end $  call pro\_test3(2,@str);  select @str; |

#### 带有循环功能的存储过程

需求： 输入一个整数，求和。例如，输入100，统计1-100的和

|  |
| --- |
| delimiter $  create procedure pro\_test4(in num int,out sumStr int)  begin  -- 定义局部变量 并且给默认值  declare i int default 1;  declare psum int default 0;  while i<=num do  set psum=psum+i;  set i=i+1;  end while;  set sumStr=psum;  end $  call pro\_test4(90,@str1);  select @str1; |

#### 使用查询的结果赋值给变量（into）

语法 select into 变量

|  |
| --- |
| /\*select into \*/  -- 带参数的存储过程 输出参数  delimiter $ -- 申明存储过程的结束符号  -- select into  create procedure pro\_test6(in stuid int ,out stuname1 varchar(20))  begin  select stuname into stuname1 from tbl\_student where id=stuid;  end $  call pro\_test6(1,@stuname1);  select @stuname1; -- 查询会话参数值  drop procedure pro\_test6; |

#### mysql变量

* 全局变量：mysql数据库内置的变量 （所有连接都起作用）
  + 查看所有全局变量： show variables
  + 查看某个全局变量： select @@变量名
  + 修改全局变量： set 变量名=新值
  + character\_set\_client: mysql服务器的接收数据的编码
  + character\_set\_results：mysql服务器输出数据的编码
* 会话变量: 只存在于当前客户端与数据库服务器端的一次连接当中。连接端口，会话变量丢失
  + -- 定义会话变量: set @变量=值
  + -- 查看会话变量： select @变量
* 局部变量：在存储过程中使用的变量就叫局部变量。只要存储过程执行完毕，局部变量就丢失！！

## 触发器

#### 触发器的作用

当操作了某张表时，希望同时触发一些动作/行为，可以使用触发器完成！！

需求：当用户评论了一条信息之后，该信息的评论数需要增加1

该业务涉及两张表，如下：

信息表：信息id，用户id，信息内容，发表时间，评论数，点赞数

评论表：评论id，信息id,用户id，评论内容，评论时间

#### 创建触发器（添加）

|  |
| --- |
| -- 创建触发器  create trigger tri\_comment\_add after insert on tbl\_comment for each row  update tbl\_info set commentNum=commentNum+1 where id=new.infoId; |

#### 创建触发器(修改)

|  |
| --- |
| create trigger tri\_comment\_update after update on tbl\_comment for each row  insert into tbl\_log(content,rowId) values("dsss",old.id); |

#### 触发器删除

|  |
| --- |
| create trigger tri\_comment\_update after delete on tbl\_comment for each row  insert into tbl\_log(content,rowId) values("删除数据了",old.id); |

#### 删除触发器

|  |
| --- |
| drop trigger tri\_comment\_update; |

## mysql权限

### password 函数：

password:md5 单向加密函数，不能解密

|  |
| --- |
| select password("123456"); |

### 创建用户并且分配权限

|  |
| --- |
| /\*  创建用户  lijie:用户名  %:表示该用户可以在任意机器上登陆mysql服务，localhost和127.0.0.1，表示只能本地  123456：登陆密码，如果密码为空，表示登陆不需要密码 identified by  \*/  create user 'lijie' @'%' identified by '123456';  /\*  授权，分配权限的语法： grant privileges ON databasename.tablename to 'username'@'host'  privileges: select 查询，insert 添加 update修改 drop create delete如果是所有权限，则使用all  databasename.tablename :表示用户对哪个数据库的哪个表有操作权限，如果是所有，用\*.\*表示  \*/  grant select on 0629db.tbl\_student to 'lijie'@'localhost';  /\*  修改某个用户的密码  \*/  update user set password=password('1234567') where user='root';  flush privileges; |

### 撤销用户权限

|  |
| --- |
| /\*  撤销用户的权限  revoke privilege ON databasename.tablename FROM 'username'@'host';  \*/  revoke select on 0629db.tbl\_student from 'lijie'@'localhost'; |

### 删除用户

|  |
| --- |
| DROP USER 'username'@'host'; |

## 数据库备份和还原

数据库备份（不需要登陆）：

|  |
| --- |
| mysqldump -u root -p 0630db\_test>d:0630db\_test.sql |

数据库还原(不需要登陆)：数据库还原的时候一定要指定数据库

|  |
| --- |
| mysql -u root -p 0630db\_test1<d:0630db\_test.sql |

# web开发入门

web：英语中是网页的意思

在Java中，动态web资源开发技术统称为Javaweb，我们课程的重点也是教大家如何使用Java技术开发动态的web资源，即动态web页面

## 服务器

从物理上来说，服务器就是一台PC机器。8核，8G以上，T来计算，带宽100M linux

web服务器：PC机器安装一个具有web服务的软件，称之为web服务器

数据库服务器：PC机器安装一个具有数据管理件服务的软件，称之为数据库服务器。

邮件服务器：PC机器安装一个具有发送邮件服务的软件，称之为邮件服务器。

## web服务软件

把本地的资源共享给外部外部访问

## 常见的市面上的web服务软件

javase的规范,包含IO流，线程，集合，socket编程。。。。

* + - WebLogic: BEA公司的产品。 收费的。支持JavaEE规范。
    - WebSphere： IBM公司的产品。收费的。支持JavaEE规范
    - JBoss: Redhat公司的产品。收费的。支持JavaEE规范
    - Tomcat： 开源组织Apache的产品。免费的。支持部分的JavaEE规范。（servlet、jsp。jdbc，但 ejb， rmi不支持）

## Tomcat基本使用

### 下载并安装

1）到apache官网。www.apache.org http://jakarta.apache.org(产品的主页)

2）

安装版：window （exe、msi） linux（rmp）

压缩版：window（rar，zip） linux（tar，tar.gz）学习时候使用

3）运行和关闭tomcat

3.1 启动软件

a）找到%tomcat%/bin/startup.bat ，双击这个文件

b）弹出窗口，显示信息（不要关闭次窗口）

c）打开浏览器，输出以下地址

http://localhost:8080

d）看到一只猫画面，证明软件启动成功！

3.3 关闭软件

a）找到%tomcat%/bin/shutdown.bat，双击这个文件即可！

c）打开浏览器，输出以下地址。看到“无法连接”（最好先清空浏览器缓存）

### tomcat软件使用的常见问题

1）闪退问题

原因：tomcat软件是java语言开发的。 tomcat软件启动时，会默认到系统的环境变量中查找一个名称叫JAVA\_HOME的变量。这个变量的作用找到tomcat启动所需的jvm。

解决办法； 到环境变量中设置JAVA\_HOME的变量

JAVA\_HOME= C:\Program Files\Java\jdk1.6.0\_30 (注意别配置到bin目录下)

2）端口占用的错误

原因： tomcat启动所需的端口被其他软件占用了！

解决办法：

a）关闭其他软件程序，释放所需端口

b）修改tomcat软件所需端口

找到并修改%tomcat%/conf/server.xml文件

|  |
| --- |
| <Connector port="8081" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8443" /> |

3）CATALINA环境变量问题

原因： tomcat软件启动后，除了查找JAVA\_HOME后，还会再查找一个叫CATALINA\_HOME变量，这个变量的作用是设置tomcat的根目录。

解决办法：建议不要设置CATALINA\_HOME变量。检查如果有的话，清除掉！！！

### 体验tomcat软件作用

webapps目录： tomcat共享目录。需要共享的本地资源放到此目录中。

### URL

URL全名叫统一资源定位符，用于定位互联网的资源。

问题： http://localhost:8081/myweb/test.html 看到文件？

http:// 协议。http协议。

localhost 域名。为了找到IP地址。域名与ip如何映射呢，会先到本地的 C:\Windows\System32\drivers\etc\hosts文件里面查找ip地址映射信息。

本地域名： localhost

外部域名：www.baidu.com

8081 端口。软件监听的

8080： tomcat默认的端口

3306：mysql数据库的端口

1521： orace数据库的端口。

/myweb: web应用的名称。默认情况下，在webapps目录下找

/test.html ： 资源名称。

## Tomcat的目录结构

|-bin: 存放tomcat的命令。

catalina.bat 命令：

startup.bat -> catalina.bat start

shutdown.bat - > catalina.bat stop

|- conf: 存放tomcat的配置信息。其中server.xml文件是核心的配置文件。

|-lib：支持tomcat软件运行的jar包。其中还有技术支持包，如servlet，jsp

|-logs：运行过程的日志信息

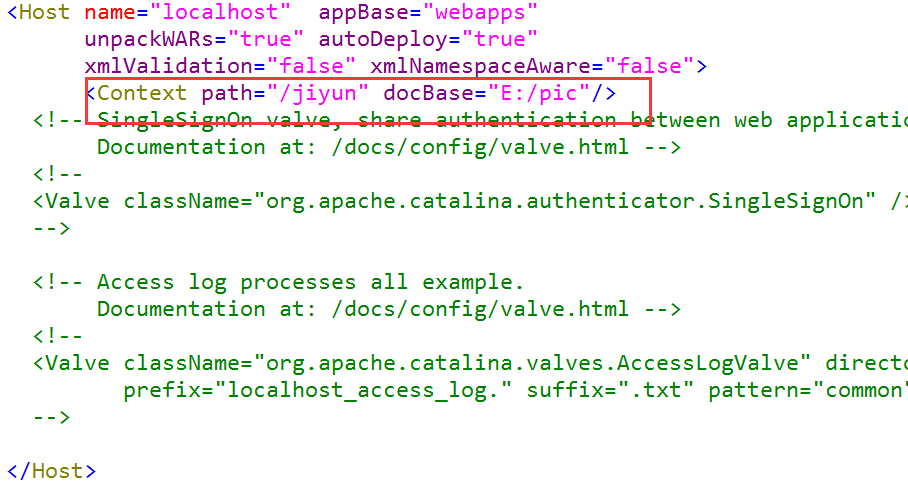
|-temp: 存放tomcat运行时产生的临时文件

|-webapps： 共享资源目录。web应用目录。（注意不能以单独的文件进行共享，必须放在文件夹里面）

|-work： tomcat的运行目录。jsp运行时产生的临时文件就存放在这里

## Tomcat虚拟目录的配置

找到tomcat下面的server.xml，在其中个的host节点里面，增加<Context>节点，如下图所示：



# 使用myEclipse开发动态资源

## 课程回顾

Web:页面网页的意思

Javaweb:在java当中，开发动态web资源的技术，统称为javaweb。

Web应用程序：浏览器可以访问的程序，也称为web应用。

tomcat应用服务器：把本地的资源共享给他人访问。

下载-官网-分清楚版本（windows（安装版的和压缩版的），linux）

在windows里面，需要通过/bin/startup.bat，开启tomcat服务

Tomcat默认端口是：8080

<url:http://ip>地址：端口号/项目名称

bin/shutdown.bat关闭tomcat服务。

在开发工具myeclipse里面，如何配置tomcat关联，以及如何启动tomcat服务。

对外共享的资源，放在tomcat下面的webapps里面。

## 为什么需要javaweb技术

Javaweb技术是开发动态web资源的。

静态资源： 源代码不会发生改变，不能与用户进行动态数据交互。

动态资源：当用户多次访问这个资源，资源的源代码可能会发生改变，它可以和用户进行动态交互。

## 动态资源的开发技术

Servlet : 用java语言来编写动态资源的开发技术。

Servlet特点：

1）普通的java类，继承HttpServlet类，重写doGet或者doPost()方法

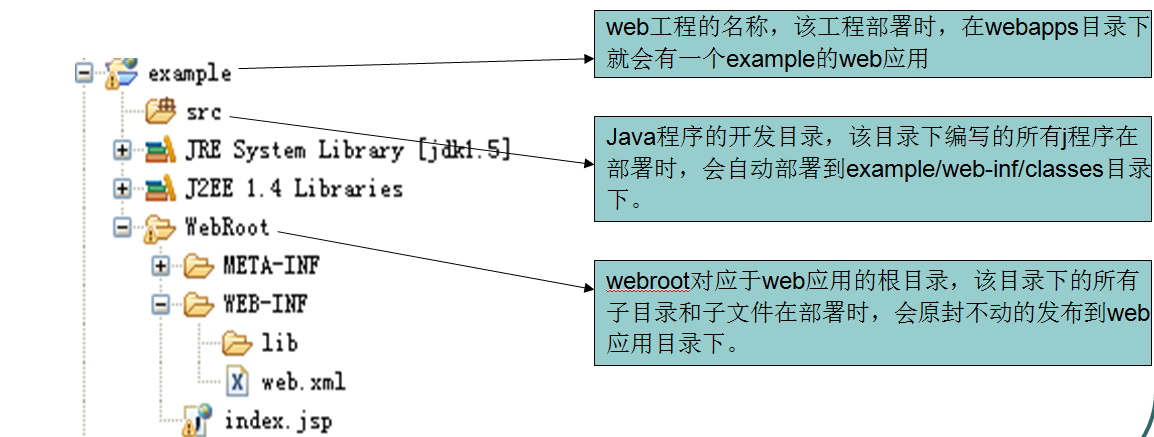
2）Servlet类只能交给tomcat服务器运行！！！！（开发者自己不能运行！！！）

## 使用myeclipse编写Servlet

1. **新建web项目**

Myeclipse中，File->new->Web Project

Web Project的目录结构如下：



1. **新建Servlet类**

在src下面，新建一个class，继承HttpServlet类，重写doGet方法

/\*\*

\* 第一个servlet程序

\* **@author** APPle

\*

\*/

**public** **class** HelloServlet **extends** HttpServlet{

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)

**throws** ServletException, IOException {

//解决中文乱码问题

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

//向浏览器输出内容

resp.getWriter().write("这是第一个servlet程序。当前时间为："+**new** Date());

}

}

1. 在配置web.xml中配置servlet

找到项目->WebRoot->WEB-INF->web.xml

|  |
| --- |
| <!--begin 配置servlet程序 -->  <servlet>  <!-- servlet名字 -->  <servlet-name>Example</servlet-name>  <!-- servlet类名：包名+类名 -->  <servlet-class>com.Example</servlet-class>  </servlet>  <servlet-mapping>  <!-- servlet名字，必须和上面一致 -->  <servlet-name>Example</servlet-name>  <!-- servlet访问地址 -->  <url-pattern>/example</url-pattern>  </servlet-mapping>  <!--end 配置servlet程序 --> |

1. 关联tomcat服务器

window-> Preferences - > MyEcplise -> servers -> Tomcat 6.x (注意一定要enable)

如果没有设置过jdk，需要在Tomcat6.x下面设置jdk，选择jdk的安装目录。

1. 部署web项目
2. 启动tomcat
3. 访问servlet

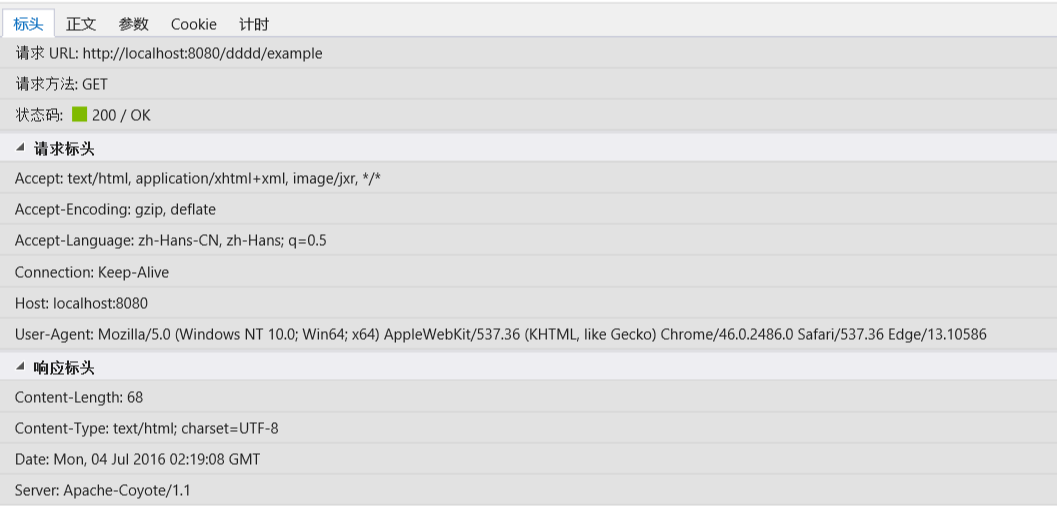
# Http协议

## http协议概念

http协议：超文本传输协议，是一种数据传输的规范

## http协议内容查看工具

1. IE，右键 ，检查元素，在网络里面可以查看到协议内容



1. 360浏览器，右键，审查元素，网络

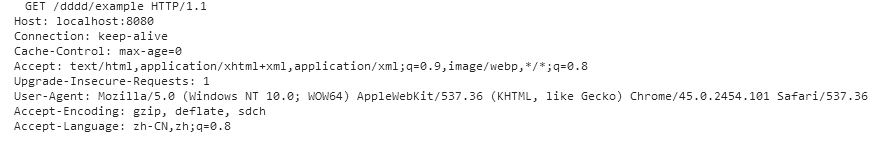


1. 通过系统自带的telnet工具来查看
2. 命令行输入 telnet localhost 8080 //连接到服务器
3. Ctrl+] 回车，可以看到回显
4. 输入请求的内容，然后回车，就可以看到服务器的响应信息

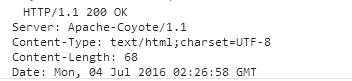
|  |
| --- |
| GET /FirstWeb/wel HTTP/1.1  Host: localhost:8080 |

## Http协议内容

请求内容：



响应内容：



今天作业：

* 新建一个javaweb项目
* 编写一个servlet类，可以接收浏览器的请求，并且做出响应。
* 将开发工具和tomcat应用服务器关联
* 部署编写好的javaweb项目
* 访问编写好的servlet类
* 新建一个jsp页面，页面里面定义好表单，通过表单将用户输入的数据提交到servlet，并且将响应结果输出到页面。

## http请求

|  |
| --- |
| GET /FirstWeb/wel?username=abc HTTP/1.1 -- 请求行  Host: localhost:8080 --请求头  Connection: keep-alive  Cache-Control: max-age=0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  Upgrade-Insecure-Requests: 1  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36  Accept-Encoding: gzip, deflate, sdch  Accept-Language: zh-CN,zh;q=0.8 |

### 请求行

GET /FirstWeb/wel?username=abc HTTP/1.1

#### http协议版本

http1.0：当前浏览器客户端与服务器端建立连接之后，只能发送一次请求，一次请求之后连接关闭。

http1.1：当前浏览器客户端与服务器端建立连接之后，可以在一次连接中发送多次请求。（基本都使用1.1）

#### 请求的资源

URL: 统一资源定位符。http://localhost:8080/FirstWeb/wel。只能定位互联网资源。是URI 的子集。

URI： 统一资源标记符。/FirstWeb/wel 。用于标记任何资源。可以是本地文件系统，局域网的资源

C:\\1.txt（//192.168.14.10/myweb/index.html），可以是互联网。

问题：

一个web页面中，使用img标签引用了三幅图片，当客户端访问服务器中的这个web页面时，客户端总共会访问几次服务器，即向服务器发送了几次HTTP请求。

#### 请求方式

常见的请求方式： GET 、 POST、 HEAD、 TRACE、 PUT、 CONNECT 、DELETE

常用的请求方式： GET 和 POST

表单提交：

|  |
| --- |
| <form action="提交地址" method="GET/POST">  <form> |

GET 和 POST 区别

**1）GET方式提交**

a）地址栏（URI）会跟上参数数据。以？开头，多个参数之间以&分割。

|  |
| --- |
| **GET** /day09/testMethod.html?name=eric&password=123456 HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive |

b）GET提交参数数据有限制，不超过1KB。

c）GET方式不适合提交敏感密码。

d）注意： 浏览器直接访问的请求，默认提交方式是GET方式

**2）POST方式提交**

a）参数不会跟着URI后面。参数而是跟在请求的实体内容中。没有？开头，多个参数之间以&分割。

|  |
| --- |
| **POST** /day09/testMethod.html HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive  name=eric&password=123456 |

b）POST提交的参数数据没有限制。

c）POST方式提交敏感数据。

### 请求头

|  |
| --- |
| Accept: text/html,image/\* -- 浏览器接受的数据类型  Accept-Charset: ISO-8859-1 -- 浏览器接受的编码格式  Accept-Encoding: gzip,compress --浏览器接受的数据压缩格式  Accept-Language: en-us,zh- --浏览器接受的语言  Host: www.it315.org:80 --（必须的）当前请求访问的目标地址（主机:端口）  If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT --浏览器最后的缓存时间  Referer: http://www.it315.org/index.jsp -- 当前请求来自于哪里  User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0) --浏览器类型  Cookie:name=eric -- 浏览器保存的cookie信息  Connection: close/Keep-Alive -- 浏览器跟服务器连接状态。close: 连接关闭 keep-alive：保存连接。  Date: Tue, 11 Jul 2000 18:23:51 GMT -- 请求发出的时间 |

### 实体内容

只有POST提交的参数会放到实体内容中

### HttpServletRequest对象

HttpServletRequest对象作用是用于获取请求数据。

核心的API：

请求行：

request.getMethod(); 请求方式

request.getRequetURI() / request.getRequetURL() 请求资源

request.getProtocol() 请求http协议版本

请求头：

request.getHeader("名称") 根据请求头获取请求值

request.getHeaderNames() 获取所有的请求头名称

实体内容:

request.getInputStream() 获取实体内容数据

### 获取浏览器的类型（user-agent）

request.getHeader

Firefox：火狐

Trident：IE浏览器

Chrome：谷歌浏览器

### 传递的请求参数如何获取

GET方式： 参数放在URI后面

POST方式： 参数放在实体内容中

获取GET方式参数：

request.getQueryString();

获取POST方式参数：

request.getInputStream();

问题：但是以上两种不通用，而且获取到的参数还需要进一步地解析。

所以可以使用统一方便的获取参数的方式：

核心的API：

request.getParameter("参数名"); 根据参数名获取参数值（注意，只能获取一个值的参数）

request.getParameterValue("参数名“)；根据参数名获取参数值（可以获取多个值的参数）

request.getParameterNames(); 获取所有参数名称列表

### 请求参数编码问题

http协议对字符的默认编码是iso-8859-1,不支持中文。一般都使用utf-8进行统一编码

修改POST方式参数编码：

request.setCharacterEncoding("utf-8"); //设置实体内容里面的编码

修改GET方式参数编码：

手动解码：String name = new String(name.getBytes("iso-8859-1"),"utf-8");

全局的，修改tomcat服务器里面的URIEncoding

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

# Http响应

* **一个HTTP响应代表服务器向客户端回送的数据，它包括：**

**一个响应行、若干响应头（消息头）、以及实体内容** ，其中的一些消息头和实体内容都是可选的，消息头和实体内容之间要用空行隔开。

|  |
| --- |
| HTTP/1.1 200 OK --响应行  Server: Apache-Coyote/1.1 --响应头（key-vaule）  Content-Length: 24  Date: Fri, 30 Jan 2015 01:54:57 GMT  --一个空行  this is hello servlet!!! --实体内容 |

## 响应行

* http协议版本
* 状态码: 服务器处理请求的结果（状态），它是一个三位的十进制数

|  |  |
| --- | --- |
| **状态码** | **含义** |
| 100～199 | 表示成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程 |
| 200～299 | 表示成功接收请求并已完成整个处理过程，常用200 |
| 300～399 | 为完成请求，客户需进一步细化请求。例如，请求的资源已经移动一个新地址，常用302、307和304 |
| 400～499 | 客户端的请求有错误，常用404 |
| 500～599 | 服务器端出现错误，常用 500 |

常见的状态：

200 ： 表示请求处理完成并完美返回

302： 表示请求需要进一步细化。

404： 表示客户端访问的资源找不到。

500： 表示服务器的资源发送错误。（服务器内部错误）

* 状态描述

响应状态的具体文字描述。

## 常见的响应头

|  |
| --- |
| Location: http://www.it315.org/index.jsp -表示重定向的地址，该头和302的状态码一起使用。  Server:apache tomcat ---表示服务器的类型  Content-Encoding: gzip -- 表示服务器发送给浏览器的数据压缩类型  Content-Length: 80 --表示服务器发送给浏览器的数据长度  Content-Language: zh-cn --表示服务器支持的语言  Content-Type: text/html; charset=GB2312 --表示服务器发送给浏览器的数据类型及内容编码  Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT --表示服务器资源的最后修改时间  Refresh: 1;url=http://www.it315.org --表示定时刷新  Content-Disposition: attachment; filename=aaa.zip --表示告诉浏览器以下载方式打开资源（下载文件时用到）  Transfer-Encoding: chunked  Set-Cookie:SS=Q0=5Lb\_nQ; path=/search --表示服务器发送给浏览器的cookie信息（会话管理用到）  Expires: -1 --表示通知浏览器不进行缓存  Cache-Control: no-cache  Pragma: no-cache  Connection: close/Keep-Alive --表示服务器和浏览器的连接状态。close：关闭连接 keep-alive:保存连接 |

## HttpServletResponse对象

Web服务器收到客户端的http请求，会针对每一次请求，分别创建一个用于代表请求的HttpServletRequest对象、和代表响应的HttpServletResponse对象

服务器端对客户端的响应封装在HttpServletResponse对象中。

可以通过HttpServletResponse对象的相关方法修改和设置响应信息：

设置响应状态：

response.setStatus() 设置状态码

resp.sendError(500);//既可以设置响应状态码，又可以设置响应的内容为状态码对应的内容

响应头：

response.setHeader("name","value") 设置响应头

实体内容：

response.getWriter().writer(); 发送字符实体内容

response.getOutputStream().writer() 发送字节实体内容

|  |
| --- |
| @Override  /\*\*  \* tomcat将客户端的请求数据封装在HttpServletRequest对象  \* tomcat将对客户端的响应信息封装在HttpServletResponse对象  \*  \*/  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  // response.setStatus() 设置状态码  //resp.setStatus(404); //手动设置响应状态码为404  //resp.sendError(500);//既可以设置响应状态码，又可以设置响应的内容为状态码对应的内容    resp.setHeader("Server", "JBoss"); //更改响应头信息    //resp.getWriter().write("sdfffffffffffffffff"); //向客户端输出字符内容  resp.getOutputStream().write("study".getBytes()); //向客户端输出字节内容  } |

## 案例- 请求重定向（Location）

方式一：

设置响应头的状态码为 302

设置location地址。

方式二： resp.sendRedirect("");

|  |
| --- |
| @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  //实现重定向的方式1：实现重定向到一个新页面，需要状态码为302和设置响应头location  resp.setStatus(304);  resp.setHeader("Location", "/FirstWeb/aa.jsp");    //实现重定向的方式2  //resp.sendRedirect("/FirstWeb/aa.jsp"); //比较常用的    } |

## 案例- 定时刷新(refresh)

隔1秒之后，刷新到一个新页面

|  |
| --- |
| resp.setHeader("Refresh", "2;/day02/a.html"); |

设置当前页面没隔1秒刷新一次

|  |
| --- |
| resp.setHeader("Refresh", "2"); |

## 案例-content-Type作用

服务器发送给浏览器的数据类型和内容编码。

1. 服务器发送给浏览器html形式的数据类型 text/html
2. Xml形式的数据类型 text/xml 图片：image/CXXX
3. 数据类型怎么查看，tomcat的web.xml可以查看所有支持的内容类型
4. 发送数据乱码（字符数据乱码和字节数据乱码怎么解决）（字节数据乱码，有两种方式）

|  |
| --- |
| @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  //resp.getWriter().write("dddd");  resp.setHeader("content-type", "text/html;charset=utf-8");  //resp.setHeader("content-type", "text/xml"); //tomcat服务器具体支持的数据类型，可以到tomcat/conf/web.xml里面寻找。  //resp.setContentType("text/xml");    //resp.setCharacterEncoding("utf-8"); //设置响应内容的编码  //resp.setHeader("content-type", "text/html;charset=utf-8");//服务器发送给浏览器的数据类型及内容编码    resp.getWriter().write("<html><head><title>content-ty的作用</title></head><body><h1>content-type到底有什么作用呢？</h1></body></html>");  } |

HttpServletRequest getParameter() getParameterValues()

HttpServletResponse .getWriter.write() getOutputStream.write(); sendRedirect();

解决乱码问题

# Servlet编程

Servlet:开发动态web资源的技术

## Servlet类的编写

1. 编写java类，继承HttpServlet类
2. 重写doGet方法和doPost方法
3. 在web.xml中配置servlet
4. 部署servlet项目
5. 启动tomcat服务
6. 访问servlet

## Servlet类的映射

|  |
| --- |
| <servlet-mapping>  <!-- servlet的内部名称，一定要和上面的内部名称保持一致！！ -->  <servlet-name>FirstServlet</servlet-name>  <!-- servlet的映射路径（访问servlet的名称） -->  <httP://localhost:8080/duty/login>xxx.shtml **<url-pattern>first</url-pattern>**  **<url-pattern>\*.html</url-pattern>**  </servlet-mapping> |

精确匹配 /first http://localhost:8080/day10/first

/itcast/demo1 http://localhost:8080/day10/itcast/demo1

模糊匹配 /\* http://localhost:8080/day10/任意路径

/itcast/\* http://localhost:8080/day10/itcast/任意路径

\*.后缀名 http://localhost:8080/day10/任意路径.do

\*.do

\*.action

\*.html(伪静态)

注意：

1）url-pattern要么以 / 开头，要么以\*开头。 例如， itcast是非法路径。

2）不能同时使用两种模糊匹配，例如 /jiyun/\*.do 是非法路径

3）当有输入的URL有多个servlet同时被匹配的情况下：

3.1 精确匹配优先。（长的最像优先被匹配）

3.2 以后缀名结尾的模糊url-pattern优先级最低！！！

|  |
| --- |
| **对于如下的一些映射关系：**   * Servlet1 映射到 /abc/\* * Servlet2 映射到 /\* * Servlet3 映射到 /abc * Servlet4 映射到 \*.do   **问题：**   * 当请求URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应   **Servlet1。**   * 当请求URL为“/abc”时，“/abc/\*”和“/abc”都匹配，哪个servlet响应   **Servlet3。**   * 当请求URL为“/abc/a.do”时，“/abc/\*”和“\*.do”都匹配，哪个servlet响应   **Servlet1**。   * 当请求URL为“/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应   **Servlet2。**   * 当请求URL为“/xxx/yyy/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应   **Servlet2。** |

## servlet缺省路径

servlet的缺省路径（<url-pattern>/</url-pattern>）是在tomcat服务器内置的一个路径。该路径对应的是一个**DefaultServlet**（缺省Servlet）。这个缺省的Servlet的作用是用于解析web应用的静态资源文件。

问题： URL输入http://localhost:8080/day10/index.html 如何读取文件？？？？

1）到当前day10应用下的web.xml文件查找是否有匹配的url-pattern。

2）如果没有匹配的url-pattern，则交给tomcat的内置的DefaultServlet处理

3）DefaultServlet程序到day10应用的根目录下查找是否存在一个名称为index.html的静态文件。

4）如果找到该文件，则读取该文件内容，返回给浏览器。

5）如果找不到该文件，则返回404错误页面。

结论： 先找动态资源，再找静态资源

## Sevlet的生命周期（重点）

### 引入

Servlet的生命周期： servlet类对象什么时候创建，什么时候调用什么方法，什么时候销毁。

以前的对象： new Student（）； stu.study(); stu=null;

Servlet程序的生命周期由tomcat服务器控制的！！！！

Tomcat是如何调用servlet的呢？

|  |
| --- |
| * Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：   1. Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。   2. 装载并创建该Servlet的一个实例对象。   3. 调用Servlet实例对象的init()方法。   4. 创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。   5. WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。 |

### Servlet重要的四个生命周期方法

（1）构造方法： 创建servlet对象的时候调用。默认情况下，第一次访问servlet的时候创建servlet对象只调用1次。证明servlet对象在tomcat是单实例的。

(2) init方法： 创建完servlet对象之后，初始化servlet的时候调用。只调用1次。

（3）service方法： 每次发送servlet请求的时候调用。调用n次。

（4）destroy方法： 销毁servlet对象的时候调用。停止服务器或者重新部署web应用时销毁servlet对象。只调用1次。

细节：

|  |
| --- |
| * Servlet是一个供其他Java程序（Servlet引擎）调用的Java类，它不能独立运行，它的运行完全由Servlet引擎来控制和调度。 * 针对客户端的多次Servlet请求，通常情况下，服务器只会创建一个Servlet实例对象，也就是说Servlet实例对象一旦创建，它就会驻留在内存中，为后续的其它请求服务，直至web容器退出，servlet实例对象才会销毁。 * 在Servlet的整个生命周期内，Servlet的init方法只被调用一次。而对一个Servlet的每次访问请求都导致Servlet引擎调用一次servlet的service方法。对于每次访问请求，Servlet引擎都会创建一个新的HttpServletRequest请求对象和一个新的HttpServletResponse响应对象，然后将这两个对象作为参数传递给它调用的Servlet的service()方法，service方法再根据请求方式分别调用doXXX方法。 |

## Servlet自动加载

默认情况下，第一次访问servlet的时候创建servlet对象。如果servlet的构造方法或init方法中执行了比较多的逻辑代码，那么导致用户第一次访问sevrlet的时候比较慢。

可以通过改变servlet创建对象的时间:提前加载到web应用中

在servlet的配置信息中，加上一个<load-on-startup>

|  |
| --- |
| <servlet>  <servlet-name>LifeDemo</servlet-name>  <servlet-class>gz.itcast.c\_life.LifeDemo</servlet-class>  <init-param>  <!-- 让servlet对象自动加载 -->  <load-on-startup>1</load-on-startup> 注意： 整数值越大，创建优先级越低！！  </servlet> |

## 有参的init方法和无参的init方法

有参数的init()方法：servlet里面生命周期中的方法

无参数的init()方法：提供给开发人员进行重写的。

|  |
| --- |
| /\*\*  \* 生命周期中的init方法  \*/  @Override  **public** **void** init(ServletConfig config) **throws** ServletException {  // **TODO** Auto-generated method stub  **super**.init(config);  }    /\*\*  \* 无参init方法，sun公司为我们开发人员定义好的，只需要我们自己去重写  \* 平常，我们自己做的一些初始化信息，放到无参的init方法里面  \*/  @Override  **public** **void** init() **throws** ServletException {  // **TODO** Auto-generated method stub  System.*out*.println("无参init()");  } |

## Servlet多线程

注意： servlet对象在tomcat服务器是单实例多线程的。

因为servlet是多线程的，所以当多个servlet的线程同时访问了servlet的共享数据，如成员变量，可能会引发线程安全问题。

解决办法：

1）把使用到共享数据的代码块进行同步（使用synchronized关键字进行同步）

2）建议在servlet类中尽量不要使用成员变量。如果确实要使用成员，必须同步。而且尽量缩小同步代码块的范围。（哪里使用到了成员变量，就同步哪里！！），以避免因为同步而导致并发效率降低。

Servlet学习：

HttpServletRequest 请求对象：获取请求信息

HttpServletResponse 响应对象： 设置响应对象

ServletConfig对象 servlet配置对象

ServletContext对象； servlet的上下文对象

## ServletConfig对象

### 作用

ServletConfig对象: 主要是用于加载servlet的初始化参数。在一个web应用可以存在多个ServletConfig对象（一个Servlet对应一个ServletConfig对象）

### 对象创建和得到

创建时机： 在创建完servlet对象之后，在调用init方法之前创建。

得到对象： 直接从有参数的init方法中得到！！！

### servlet的初始化参数配置

|  |
| --- |
| <servlet>  <servlet-name>ConfigDemo</servlet-name>  <servlet-class>gz.itcast.f\_config.ConfigDemo</servlet-class>  <!-- 初始参数： 这些参数会在加载web应用的时候，封装到ServletConfig对象中 -->  <init-param>  <param-name>path</param-name>  <param-value>e:/b.txt</param-value>  </init-param>  </servlet> |

注意： servlet的参数只能由当前的这个sevlet获取！！！！

ServletConfig的API：

java.lang.String getInitParameter(java.lang.String name) 根据参数名获取参数值

java.util.Enumeration getInitParameterNames() 获取所有参数

ServletContext getServletContext() 得到servlet上下文对象

java.lang.String getServletName() 得到servlet的名称

## ServletContext对象

### 引入

ServletContext对象 ,叫做Servlet的上下文对象。表示一个当前的web应用环境。一个web应用中只有一 个ServletContext对象。

### 对象创建和得到

创建时机：加载web应用时创建ServletContext对象。

得到对象： 从ServletConfig对象的getServletContext方法得到

我们设计：

创建ServletConfig对象

public void init( ServletConfig config，ServletContext context ){ 多了一个参数

得到ServletConfig对象

得到ServletContext对象；

}

Sun公司设计

1）创建ServletContext对象 ServletContext context = new ServletContext()

2）创建ServletConfig对象 ServetConfig config = new ServletConfig();

config.setServletContxt(context);

class ServletConfig{

ServletContext context;

public ServletContext getServletContxt(){

return contxt;

}

}

public void init( ServletConfig config ){

得到ServletConfig对象

从ServletConfig对象中得到ServletContext对象

SerlvetContext context = config.getServletContext();

}

### ServletContext对象的核心API(作用)

得到web应用路径：

context.getContextPath(); /day13

request.getContextPath(); 等价于上面的代码

得到web应用参数：

context.getInitParameter("name");

context.getInitParameterNames(); //获取web应用参数名称

域对象:

context.setAttribute("name",Object): 保存数据，可以保存任何类型的。

context.getAttribute("name") 得到数据

context.removeAttribue("name") 清除数据

转发

context.getRequestDispatcher("路径").forward(request,response); //转发，不能转发当前web应用以外的资源。response.sendRedirect可以重定向到web资源以外的互联网资源。

request.getRequestDispacher("路径").forward(request,response); 等价于上面的代码

得到web应用中的资源文件

context.getRealPath("路径")

context.getResourceAsStream("路径");

### 得到web应用路径

java.lang.String getContextPath() 用在请求重定向的资源名称中

### 得到web应用的初始化参数（全局）

java.lang.String getInitParameter(java.lang.String name) --得到web应用的初始化参数

java.util.Enumeration getInitParameterNames()

web应用参数可以让当前web应用的所有servlet获取！！！

### 域对象有关的方法

域对象：作用是用于保存数据，获取数据。可以在不同的动态资源之间共享数据。

案例：

Servlet1 Servlet2

name=eric

response.sendRedirect("/Servlet2?name=eric") String request.getParameter("name");

保存到域对象中 从域对象获取

Student

方案1： 可以通过传递参数的形式，共享数据。局限：只能传递字符串类型。

方案2： 可以使用域对象共享数据，好处：可以共享任何类型的数据！！！！！

ServletContext就是一个域对象！！！！

保存数据：void setAttribute(java.lang.String name, java.lang.Object object)

获取数据： java.lang.Object getAttribute(java.lang.String name)

删除数据： void removeAttribute(java.lang.String name)

ServletContext域对象：作用范围在整个web应用中有效！！！

所有域对象：

HttpServletRequet 域对象

ServletContext域对象

HttpSession 域对象

PageContext域对象

### 转发

RequestDispatcher getRequestDispatcher(java.lang.String path)

1）转发

a）地址栏不会改变

b）转发只能转发到当前web应用内的资源

c）可以在转发过程中，可以把数据保存到request域对象中

2）重定向 HttpServletResponse.sendRedirect(“url”);

a）地址栏会改变，变成重定向到地址。

b）重定向可以跳转到当前web应用，或其他web应用，甚至是外部域名网站。

c）不能在重定向的过程，把数据保存到request中。重定向过程中，请求的参数信息丢失

结论： 如果要使用request域对象进行数据共享，只能用转发技术！！！

总结：

Servlet编程:

Servlet生命周期（重点）

其他都是应用的东西（敲代码练习）

## Web应用更改名字

更改web项目的名字的时候，记得更改部署的项目名字

## Java读取properties文件

在web项目中， . 代表在tomcat/bin目录

在java项目中，.代表项目根路径

路径不能写死，可以通过getRealPath读取,返回资源文件的绝对路径

|  |
| --- |
| //加载数据库配置文件  @Override  **public** **void** init() **throws** ServletException {  // **TODO** Auto-generated method stub ./src/  // .代表java运行命令所在的目录 ./  //File file=new File("./src/db.properties");  String path=**this**.getServletContext().getRealPath("/WEB-INF/classes/db.properties");  File file=**new** File(path);  **try** {  FileInputStream input=**new** FileInputStream(file);  Properties pro=**new** Properties();  pro.load(input);  user=pro.getProperty("user");  pwd=pro.getProperty("pwd");  System.*out*.println(user+","+pwd);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  } |

## 作业：

提示：

添加学生：

设计一个添加学生的页面

保存servlet：

AddServlet （接收页面数据（getParameter（）），使用jdbc保存到数据库中）

查询在QueryServlet:查询出数据库中所有的学生信息，重定向到ShowServlet，在showServlet中将数据输出到页面上

# 会话管理

## 软件中的会话

一次会话： 打开浏览器 -> 访问一些服务器内容 -> 关闭浏览器

登录场景：

打开浏览器 -> 浏览到登陆页面 -> 输入用户名和密码 -> 访问到用户主页(显示用户名)

修改密码（输入原密码）

修改收货地址

.......

问题：在此处登录会话过程中产生的数据（用户会话数据）如何保存下来呢？

购物场景：

打开浏览器 -> 浏览商品列表 -> 加入购物车(把商品信息保存下来) -> 关闭浏览器

打开浏览器-> 直接进入购物车 -> 查看到上次加入购物车的商品 -> 下订单 -> 支付

问题： 在购物会话过程中，如何保存商品信息？？

会话管理： 管理浏览器客户端 和 服务器端之间会话过程中产生的会话数据。

域对象： 实现资源之间的数据共享。

request域对象

context域对象

登录场景：

小张： 输入“张三” （保存数据： context.setAttribute("name","张三")） -> 用户主页（显示“张三”）

小李： 输入“李四”(保存数据：context.setAttribute("name","李四")) -> 用户主页（显示“李四”）

问题： context是所有用户公有的资源！！！会覆盖数据。

小张： 输入“张三”（保存数据： request.setAttribute("name","张三")）- > 用户主页（显示“张三”） 问题： 一定要使用转发技术来跳转页面！！！

解决办法： 可以使用session域对象来保存会话数据！！！

## 会话技术

Cookie技术：会话数据保存在浏览器客户端。

Session技术：会话数据保存在服务器端。

## Cookie技术

### 特点

Cookie技术：会话数据保存在浏览器客户端。

### Cookie技术核心

Cookie类：用于存储会话数据

1）构造Cookie对象

Cookie(java.lang.String name, java.lang.String value)

2）设置cookie

void setPath(java.lang.String uri) ：设置cookie的有效访问路径

void setMaxAge(int expiry) ： 设置cookie的有效时间，单位是秒

void setValue(java.lang.String newValue) ：设置cookie的值

3）发送cookie到浏览器端保存

void response.addCookie(Cookie cookie) : 发送cookie

4）服务器接收cookie

Cookie[] request.getCookies() : 接收cookie

### Cookie原理

1）服务器创建cookie对象，把会话数据存储到cookie对象中。

new Cookie("name","value");

2） 服务器发送cookie信息到浏览器

response.addCookie(cookie);

举例： set-cookie: name=eric (隐藏发送了一个set-cookie名称的响应头)

3）浏览器得到服务器发送的cookie，然后保存在浏览器端。

4）浏览器在下次访问服务器时，会带着cookie信息

举例： cookie: name=eric (隐藏带着一个叫cookie名称的请求头)

5）服务器接收到浏览器带来的cookie信息

request.getCookies(); 返回Cookie[]

### Cookie的细节

1）void setPath(java.lang.String uri) ：设置cookie的有效访问路径。有效路径指的是cookie的有效路径保存在哪里，那么浏览器在有效路径下访问服务器时就会带着cookie信息，否则不带cookie信息。

2）void setMaxAge(int expiry) ： 设置cookie的有效时间。

正整数：表示cookie数据保存浏览器的缓存目录（硬盘中），数值表示保存的时间。

负整数：表示cookie数据保存浏览器的内存中。浏览器关闭cookie就丢失了！！

零：表示删除同名的cookie数据

3）Cookie数据类型只能保存非中文字符串类型的。可以保存多个cookie，但是浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。

### 案例- 显示用户上次访问的时间

|  |
| --- |
| public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  //1.创建Cookie对象  Cookie cookie1 = new Cookie("name","eric");  //Cookie cookie2 = new Cookie("email","jacky@qq.com");  //Cookie cookie1 = new Cookie("email","eric@qq.com");      /\*\*  \* 1）设置cookie的有效路径。默认情况：有效路径在当前web应用下。 /day11  \*/  //cookie1.setPath("/day11");  //cookie2.setPath("/day12");    /\*\*  \* 2)设置cookie的有效时间  \* 正整数：表示cookie数据保存浏览器的缓存目录（硬盘中），数值表示保存的时间。  负整数：表示cookie数据保存浏览器的内存中。浏览器关闭cookie就丢失了！！  零：表示删除同名的cookie数据  \*/  //cookie1.setMaxAge(20); //20秒，从最后不调用cookie开始计算  cookie1.setMaxAge(-1); //cookie保存在浏览器内存（会话cookie）  //cookie1.setMaxAge(0);      //2.把cookie数据发送到浏览器（通过响应头发送： set-cookie名称）  //response.setHeader("set-cookie", cookie.getName()+"="+cookie.getValue()+",email=eric@qq.com");  //推荐使用这种方法，避免手动发送cookie信息  response.addCookie(cookie1);  //response.addCookie(cookie2);  //response.addCookie(cookie1);      //3.接收浏览器发送的cookie信息  /\*String name = request.getHeader("cookie");  System.out.println(name);\*/  Cookie[] cookies = request.getCookies();  //注意：判断null,否则空指针  if(cookies!=null){  //遍历  for(Cookie c:cookies){  String name = c.getName();  String value = c.getValue();  System.out.println(name+"="+value);  }  }else{  System.out.println("没有接收cookie数据");  }    } |

### Cookie重点掌握：

Cookie存数据

|  |
| --- |
| / \*\*  \* 将用户信息存放到会话对象Cookie里面  \* Cookie里面允许存储的值只能是字符串类型  \* cookie值是保存在浏览器内存里面的  \* cookie有效时间可以设置 setMaxAge() 单位秒，整数，负数，0  \*/  Cookie cookie=**new** Cookie("userName", user.getUserName()); //创建cookie对象，并保存值  cookie.setMaxAge(60\*60);//设置cookie值保存的有效时间  resp.addCookie(cookie); //将cookie发送出去 |

Cookie取数据：

|  |
| --- |
| //cookie的名字 cookie.getName() cookie的值：cookie.getValue()  Cookie[] cookies=request.getCookies();  String userName="";  **for**(**int** i=0;i<cookies.length;i++){  **if**(cookies[i].getName().equals("userName")){  userName=cookies[i].getValue();  }  } |

Cookie值默认是存在浏览器端内存中

Cookie的局限：

1）Cookie只能存非中文字符串类型。不能保存对象，不能保存引用数据类型。

2）只能存非中文。

3）1个Cookie的存储的数据大小有限制

### 本章作业

1. 请详细描述cookie的特点及细节
2. 设置cookie的有效路径，使用什么方法，该方法需要注意什么？
3. 设置cookie的有效时间，使用什么对象里面的什么方法？使用该方法设置值的时候，需要注意什么？
4. 设置cookie的值，使用什么方法？
5. 获取cookie的名称和值，分别使用什么方法？
6. 机试:通过cookie技术实现显示用户上次登录的时间，要有良好的业务逻辑。

## Session技术

### 引入

Cookie的局限：

1）Cookie只能存非中文字符串类型。不能保存对象，不能保存引用数据类型。

2）只能存非中文。

3）1个Cookie的容量不超过4KB。但是浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB

如果要保存非字符串，超过4kb内容，只能使用session技术！！！

Session特点：

会话数据保存在服务器端。（内存中）

### Session技术核心

HttpSession类：用于保存会话数据

1）创建或得到session对象

HttpSession getSession()

HttpSession getSession(boolean create)

1. 设置session对象

void setMaxInactiveInterval(int interval) ： 设置session的有效时间，设置为-1的时候表示session永远不过期;

void invalidate() ： 销毁session对象

java.lang.String getId() ： 得到session编号

3）保存会话数据到session对象

void setAttribute(java.lang.String name, java.lang.Object value) ： 保存数据

java.lang.Object getAttribute(java.lang.String name) ： 获取数据

void removeAttribute(java.lang.String name) ： 清除数据

### Session原理

问题： 服务器能够识别不同的浏览者！！！

代码解读：HttpSession session = request.getSession();

1）第一次访问创建session对象，给session对象分配一个唯一的ID，叫JSESSIONID

2）把JSESSIONID作为Cookie的值发送给浏览器内存保存

3）第二次访问的时候，浏览器带着JSESSIONID的cookie访问服务器

4）服务器得到JSESSIONID，在服务器的内存中搜索是否存放对应编号的session对象。

5）如果找到对应编号的session对象，直接返回该对象

6）如果找不到对应编号的session对象，创建新的session对象，同时分配一个sessionId，然后继续走1的流程

结论：通过JSESSION的cookie值在服务器找session对象！！！！！

### Sesson细节

1）java.lang.String getId() ： 得到session编号

2）两个getSession方法：

getSession(true) / getSession() : 创建或得到session对象。没有匹配的session编号，自动创 建新的session对象。

getSession(false): 得到session对象。没有匹配的session编号，返回null

|  |
| --- |
| HttpSession session1=req.getSession(true);//如果存在sesion对象，则获取对应的session对象，如果不存在，就重新创建session对象 |
| HttpSession session=req.getSession(**false**); // 获取session对象，如果存在，就取出对应的session对象，如果不存在，则为空 |

3）void setMaxInactiveInterval(int interval) ： 设置session的有效时间，单位是秒

4)session对象销毁时间：

a. 默认情况30分钟服务器自动回收

b.手动修改session回收时间，设置setMaxInactiveInterval(int interval) (如果设置-1的话，表示永远不过期)

c. 全局修改session有效时间，在web.xml里面设置全局的session有效时间

|  |
| --- |
| <!-- 修改session全局有效时间:分钟 -->  <session-config>  <session-timeout>1</session-timeout>  </session-config> |

d.手动销毁session对象

void invalidate() ： 销毁session对象

5)移除一个session对象的某一个值：removeAttribute(name);

|  |
| --- |
| //移除某个session的值  session.removeAttribute("name"); |

6）如何避免浏览器的JSESSIONID的cookie随着浏览器关闭而丢失的问题

|  |
| --- |
| /\*\*  \* 手动发送一个硬盘保存的cookie给浏览器  \*/  Cookie c = **new** Cookie("JSESSIONID",session.getId());  c.setMaxAge(60\*60);  response.addCookie(c); |

### 用户登录及退出案例

1. 定义一个登录页面
2. 用户在登录页面，点击登录按钮，跳转到登录servlet
3. 在登录servlet里面，判断用户名密码是否正确，正确的话，跳转到首页servlet。用户名密码不正确，则向客户端输出“用户名或密码错误”
4. 在首页servlet里面获取到用户名，然后向客户端输出“欢迎您，某某某！”
5. 注意避免乱码问题

# Jsp

## Jsp基础

JSP全称是Java Server Pages

### Jsp引入

Servlet的作用： 用java语言开发动态资源的技术！！！

Jsp的作用：用java语言（+html语言）开发动态资源的技术！！！

Jsp就是servlet！！！

它相比html而言，html只能为用户提供静态数据，而Jsp技术允许在页面中嵌套java代码，为用户提供动态数据

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!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=ISO-8859-1"*>  <title>Insert title here</title>  </head>  <body>    <%  String name="ddd";  out.print(name);  %>  </body>  </html> |

### Jsp的特点

1）jsp的运行必须交给tomcat服务器！！！！

tomcat的work目录： tomcat服务器存放jsp运行时的临时文件

2）jsp页面既可以写html代码，也可以写java代码。

（html页面不能写java代码 。而jsp页面可以写java代码）

### 体验jsp页面作用

需求：显示当前时间到浏览器上

可以把jsp页面当做html页面在tomcat中访问！！！

### Jsp的执行过程

问题： 访问http://localhost:8080/day12/01.hello.jsp 如何显示效果？

1）访问到01.hello.jsp页面，tomcat扫描到jsp文件，在%tomcat%/work把jsp文件翻译成java源文件

(01.hello.jsp -> \_01\_hello\_jsp.java) （翻译）

2）tomcat服务器把java源文件编译成class字节码文件 （编译）

（\_01\_hello\_jsp.java -> \_01\_hello\_jsp.class）

3）tomcat服务器构造\_01\_hello\_jsp类对象

4）tomcat服务器调用\_01\_hello\_jsp类里面方法，返回内容显示到浏览器。

### 疑问

问题： 为什么Jsp就是servlet！！！

Jsp翻译成的java类继承 org.apache.jasper.runtime.HttpJspBase

HttpJspBase类继承了javax.servlet.http.HttpServlet

结论： Jsp就是一个servlet程序！！！

servlet的技术可以用在jsp程序中( HttpServletRequest ,response,servletConfig，servletContext,cookie,sessin)

Servlet的生命周期：

1）构造方法（第1次访问）

2）init方法（第1次访问）init()

3）service方法

4）destroy方法

Jsp的生命周期

1）翻译： jsp->java文件

2）编译： java文件->class文件（servlet程序）

3）构造方法（第1次访问）

4）init方法（第1次访问）：\_jspInit()

5）service方法：\_jspService()

6）destroy方法：\_jspDestroy()

## Jsp语法

### Jsp模板

jsp页面中的html代码就是jsp的模板

### Jsp表达式

语法：<%=变量或表达式或者常量%>

作用： 向浏览器输出变量的值或表达式计算的结果

注意：

1）表达式的原理就是翻译成out.print(“变量” );通过该方法向浏览器写出内容

2）表达式后面不需要带分号结束。

### Jsp的脚本

语法：<%java代码 %>

作用： 执行java代码

单个脚本片断中的Java语句可以是不完整的，但是，多个脚本片断组合后的结果必须是完整的Java语句

注意：

1）原理把脚本中java代码原封不动拷贝到\_jspService方法中执行。

### Jsp的声明

语法：<%! 变量或方法 %>

作用： 声明jsp的变量或方法

注意:

* + 1. 变量翻译成成员变量，方法翻译成成员方法。（jsp脚本不能定义java方法，因为脚本代码是加载在\_jspService()里面）

|  |
| --- |
| <!-- jsp声明 -->  <%!  String age="20";  **public** **void** add(){  System.out.print("aaa");  }  %> |

### Jsp的注释

语法： <%-- jsp注释 --%>

注意;

1)html的注释会被翻译和执行。而jsp的注释不能被翻译和执行。

## Jsp指令

* + JSP指令（directive）是为JSP引擎而设计的，它们并不直接产生任何可见输出，而只是告诉引擎如何处理JSP页面中的其余部分。在JSP 2.0规范中共定义了三个指令：
  + page指令
  + include指令
  + taglib指令
* JSP指令的基本语法格式：

<%@ 指令 属性名="值" %>

例如include指令：<%@ include 属性名=“值”%>

### include指令

作用： 在当前页面用于包含其他页面

语法： <%@include file="common/header.jsp"%>

注意：

1）原理是把被包含的页面（header.jsp）的内容翻译到包含页面(index.jsp)中,合并成翻译成一 个java源文件，再编译运行！！，这种包含叫静态包含（源码包含）

2）如果使用静态包含，被包含页面中不需要出现全局的html标签了！！！（如html、head、 body）

### page指令

page指令的作用： 告诉tomcat服务器如何翻译jsp文件

<%@ page

language=*"java"* --告诉服务器使用什么动态语言来翻译jsp文件，

import=*"java.util.\*"* --告诉服务器java文件使用什么包，多个包之间用逗号分割

pageEncoding=*"utf-8"* --告诉服务器使用什么编码翻译jsp文件（成java文件）

contentType=*"text/html; charset=utf-8"--* 服务器发送浏览器的数据类型和内容编码

注意：在开发工具中，以后只需要设置pageEncoding即可解决中文乱码问题

%>

一个指令有多个属性，可以分开写，也可以写到一起

分开写

|  |
| --- |
| <%@page import=*"java.util.Date"*%>  <%@page import=*"java.text.SimpleDateFormat"*%> |

|  |
| --- |
| <%@page import=*"java.util.Date,java.text.SimpleDateFormat"*%> |

**其他page指令：**

异常错误相关的：

errorPage: 指定当前jsp页面的错误处理页面。

isErorrPage: 指定当前页面是否为错误处理页面。false，不是错误处理页面，则不能使用

exception内置对象；true，是错误处理页面，可以使用exception内置对象。

配置全局的错误处理页面：在web.xml里面进行配置

|  |
| --- |
| <!-- 全局错误处理页面配置 -->  <error-page>  <error-code>500</error-code>  <location>/common/500.jsp</location>  </error-page>  <error-page>  <error-code>404</error-code>  <location>/common/404.html</location>  </error-page> |

session: 是否开启session功能。false，不能用session内置对象；true，可以使用session内置对象。

buffer: jsp页面的缓存区大小。

isELIgnore： 是否忽略EL表达式。

### taglib指令

Taglib指令用于在JSP页面中导入标签库

## JSP内置对象

### 什么是内置对象？

在jsp开发中，会频繁使用到一些对象，例如HttpSession,ServletConfig,ServletContext,HttpServletRequet。如果我们每次要使用这些对象都去创建这些对象就显示非常麻烦。所以Sun公司设计Jsp时，在jsp页面加载完毕之后就会自动帮开发者创建好这些对象，而开发者只需要直接使用这些对象调用方法即可！，这些创建好的对象就叫内置对象！！！！

举例：

servlet:

HttpSession session = request.getSession(true); （需要开发者做）

jsp:

tomcat服务器： HttpSession session = request.getSession(true);(不需要开发者做)

开发者做的： session.getId();

### 九大内置对象

内置对象名 类型

request HttpServletRequest

response HttpServletResponse

config ServletConfig

application ServletContext

session HttpSession

exception Throwable

page Object(this)

out JspWriter

pageContext PageContext

这其中有四个域对象

pageContext

request

session

application

### Out对象

out对象类型，JspWriter类，相当于带缓存的PrintWriter

PrintWriter： wrier(内容)： 直接向浏览器写出内容。 // 可以通过response.getWriter()来获取

JspWriter: writer(内容): 向jsp缓冲区写出内容

当满足以下条件之一，缓冲区内容写出：

1）缓冲区满了

2）刷新缓存区

3）关闭缓存区

4）执行完毕jsp页面

buffer属性默认缓存是8kb

### pageContext对象

pageContext对象的类型是PageContext，叫jsp的上下文对象

pageContext对象的用处：可以获取其他八个内置对象，本身可以作为域对象来使用

#### 可以获取其他八个内置对象

|  |
| --- |
| <%@ 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>pageContext</title>  </head>  <body>  <%-- pageContext PageContext --%>  <%-- 1.可以作为一个域对象，来保存数据和获取数据，作资源共享 2.可以通过他，来获取其他八大内置对象--%>  <%=request==pageContext.getSession()%> <%-- 通过pageContext获取request对象 --%>  <br>  <%=session==pageContext.getSession()%> <%-- 通过pageContext获取session对象 --%>  <%  request.getAttribute("");  read(pageContext);  %>  <%!  **public** **void** read(PageContext pageContext){  pageContext.getRequest().getAttribute("");  pageContext.getSession().getAttribute("");  }  %>  </body>  </html> |

使用场景： 在自定义标签的时候，PageContext对象频繁使用到！！！

#### 本身是一个域对象

ServletContext context域

HttpServletRequet request域

HttpSession session域 --Servlet学习的

PageContext page域 --jsp学习的

作用： 保存数据和获取数据，用于共享数据

#保存数据

1）默认情况下，保存到page域

pageContext.setAttribute("name");

/ 2）可以向其它域对象保存数据

pageContext.setAttribute("name",域范围常量)

|  |
| --- |
| pageContext.setAttribute("msg", "i am request", PageContext.REQUEST\_SCOPE); //可以向其他域对象里面存数据 |

#获取数据

1）默认情况下，从page域获取

pageContext.getAttribute("name")

2）可以从四个域中获取数据

pageContext.getAttribute("name",域范围常量)

|  |
| --- |
| pageContext.getAttribute("msg", PageContext.REQUEST\_SCOPE);//从其他域里面取数据 |

域范围常量:

PageContext.PAGE\_SCOPE

PageContext.REQUEST\_SCOPE

PageContext..SESSION\_SCOPE

PageContext.APPLICATION\_SCOPE

3）自动在四个域中搜索数据

pageContext.findAttribute("name");

顺序： page域 -> request域 -> session域- > context域（application域）

### Jsp中的四个域对象

四个域对象：

pageContext page域 PageContext

request request域 HttpServletRequest

session session域 HttpSession

application context域 ServletContext

1）域对象作用：

保存数据 和 获取数据 ，用于数据共享。

2）域对象方法：

setAttribute("name",Object) 保存数据

getAttribute("name") 获取数据( request.getParameter(“”) 获取参数值，而request.getAttribute(“”)从域对象里面取值)

removeAttribute("name") 清除数据

3）域对象作用范围：

page域： 只能在当前jsp页面中使用（当前页面）

request域： 只能在同一个请求中使用（转发）

session域： 只能在同一个会话（session对象）中使用（私有的）

context域： 只能在同一个web应用中使用。（全局的）

### Jsp的最佳实践

Servlet技术： 开发动态资源。是一个java类，最擅长写java代码

jsp技术： 开发动态资源。通过java代码最擅长输出html代码。

做项目的一个逻辑：

1. 用户在前端输入
2. 后台接收用户输入数据
3. 做业务逻辑处理
4. 将处理结果返回到前端

Jsp:

控制前端页面结构

通过域对象来获取后台传递过来的数据和结果

把数据展示到页面上

Servlet:

接收用户的数据

业务逻辑处理

将处理结果及数据保存到域对象里面

通过转发和重定向，跳转到前端页面

## EL表达式

### EL作用

jsp的核心语法： jsp表达式 <%=%>和 jsp脚本<% %>。

以后开发jsp的原则： 尽量在jsp页面中少写甚至不写java代码。

**使用EL表达式替换掉jsp表达式**

**EL表达式作用： 向浏览器输出域对象中的变量值或表达式计算的结果！！！**

**语法： ${变量或表达式}**

### EL语法

1）输出基本数据类型变量

1.1 从四个域获取

${name}

1.2 指定域获取

${pageScope.name}

域范围： pageScope / requestScope / sessionScope / applicationScope

2）输出对象的属性值

Student

语法：${对象.属性}

3）输出集合对象

List 和 Map

|  |
| --- |
| <%--使用EL获取List对象 --%>  ${list[0].name } - ${list[0].age }<br/>  ${list[1].name } - ${list[1].age }<br/>  ${list[2].name } - ${list[2].age }  <%--使用EL获取Map对象 --%>  ${map['100'].name } - ${map['100'].age }<br/>  ${map['101'].name } - ${map['101'].age }<br/>  ${map['102'].name } - ${map['102'].age }<br/> |

4）EL表达式计算

算术表达式：${10+5 }

比较运算:${10>5}

逻辑运算:${true && false}

判空：${name!=null} ${name!=””} ${empty name}

## jsp标签

### jsp标签的作用

jsp标签作用：替换jsp脚本。

1）流程判断（if for循环）

2）跳转页面（转发，重定向）

3）。。。。。

### Jsp标签分类

1）内置标签（动作标签）： 不需要在jsp页面导入标签

2）jstl标签： 需要在jsp页面中导入标签

3）自定义标签 ： 开发者自行定义，需要在jsp页面导入标签

### 动作标签

转发标签： <jsp:forward />

参数标签： <jsp:pararm/>

包含标签： <jsp:include/>

原理： 包含与被包含的页面先各自翻译成java源文件，然后再运行时合并在一起。

（先翻译再合并），动态包含

静态包含 vs 动态包含的区别？

1） 语法不同

静态包含语法： <%@inclue file="被包含的页面"%>

动态包含语法：　<jsp:include page="被包含的页面">

2）参数传递不同

静态包含不能向被包含页面传递参数

动态包含可以向被包含页面传递参数

3）原理不同

静态包含： 先合并再翻译

动态包含： 先翻译再合并

### JSTL标签

JSTL (全名：java standard tag library - java标准标签库 )

**核心标签库 （c标签库） 天天用**

**国际化标签（fmt标签库）**

**EL函数库（fn函数库）**

xml标签库（x标签库）

sql标签库（sql标签库）

### 使用JSTL标签步骤

1） 导入jstl支持的jar包（标签背后隐藏的java代码）

注意：使用javaee5.0的项目自动导入jstl支持jar包

2）使用taglib指令导入标签库

<%@taglib uri=*"tld文件的uri名称"* prefix=*"简写"* %>

3）在jsp中使用标签

核心标签库的重点标签：

保存数据：

<c:set></c:set>

|  |
| --- |
| <%--set标签 :保存数据(保存到域中)默认保存到page域 --%>  <c:set var=*"name"* value=*"rose"* scope=*"request"*></c:set> |

获取数据：

<c:out value=*""*></c:out>

|  |
| --- |
| <%--out标签： 获取数据（从域中）  default： 当value值为null时，使用默认值  escapeXml: 是否对value值进行转义，false，不转义，true，转义（默认）  --%>  <c:out value=*"*${msg}*"* default=*"*<h3>*标题3*</h3>*"* escapeXml=*"true"*></c:out> |

单条件判断

<c:if test=*""*></c:if>

多条件判断

<c:choose></c:choose>

<c:when test=*""*></c:when>

<c:otherwise></c:otherwise>

循环数据

<c:forEach></c:forEach>

<c:forTokens items=*""* delims=*""*></c:forTokens>

|  |
| --- |
| <%--  begin="" : 从哪个元素开始遍历，从0开始.默认从0开始  end="": 到哪个元素结束。默认到最后一个元素  step="" ： 步长 (每次加几) ，默认1  items=""： 需要遍历的数据（集合）  var=""： 每个元素的名称  varStatus=""： 当前正在遍历元素的状态对象。（count属性：当前位置，从1开始）    --%>  <c:forEach items=*"*${list}*"* var=*"student"* varStatus=*"varSta"*>  序号：${varSta.count} - 姓名：${student.name } - 年龄：${student.age}<br/>  </c:forEach>  <c:forEach items=*"*${map}*"* var=*"entry"*>  ${entry.key } - 姓名： ${entry.value.name } - 年龄：${entry.value.age }<br/>  </c:forEach> |

|  |
| --- |
| <c:set value=*"java-sql-html"* var=*"str"*></c:set>  <c:forTokens items=*"*${str }*"* delims=*"-"* var=*"str"*>  ${str }  </c:forTokens> |

重定向

<c:redirect></c:redirect>

## 自定义标签

### 第一个自定义标签开发步骤

1）编写一个普通的java类，继承SimpleTagSupport类，叫**标签处理器类**

|  |
| --- |
| /\*\*  \* 标签处理器类  \* **@author** APPle  \* 1）继承SimpleTagSupport  \*/  **public** **class** ShowIpTag **extends** SimpleTagSupport{  **private** JspContext context;    /\*\*  \* 传入pageContext  \*/  @Override  **public** **void** setJspContext(JspContext pc) {  **this**.context = pc;  }  /\*\*  \* 2）覆盖doTag方法  \*/  @Override  **public** **void** doTag() **throws** JspException, IOException {  //向浏览器输出客户的ip地址  PageContext pageContext = (PageContext)context;    HttpServletRequest request = (HttpServletRequest)pageContext.getRequest();    String ip = request.getRemoteHost();    JspWriter out = pageContext.getOut();    out.write("使用自定义标签输出客户的IP地址："+ip);    }  } |

2）在web项目的WEB-INF目录下建立itcast.tld文件，这个tld叫标签库的声明文件。（参考核心标签库的tld文件）

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <taglib xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-jsptaglibrary\_2\_1.xsd"*  version=*"2.1"*>  <!-- 标签库的版本 -->  <tlib-version>1.1</tlib-version>  <!-- 标签库前缀 -->  <short-name>itcast</short-name>  <!-- tld文件的唯一标记 -->  <uri>http://gz.itcast.cn</uri>  <!-- 一个标签的声明 -->  <tag>  <!-- 标签名称 -->  <name>showIp</name>  <!-- 标签处理器类的全名 -->  <tag-class>gz.itcast.a\_tag.ShowIpTag</tag-class>  <!-- 输出标签体内容格式 -->  <body-content>scriptless</body-content>  </tag>  </taglib> |

3） 在jsp页面的头部导入自定义标签库

<%@taglib uri=*"http://gz.itcast.cn"* prefix=*"itcast"*%>

4） 在jsp中使用自定义标签

<itcast:showIp></itcast:showIp>

### SimpleTag类的方法介绍

|  |
| --- |
| * setJspContext方法   + 用于把JSP页面的pageContext对象传递给标签处理器对象 * setParent方法   + 用于把父标签处理器对象传递给当前标签处理器对象 * getParent方法   + 用于获得当前标签的父标签处理器对象 * setJspBody方法   + 用于把代表标签体的JspFragment对象(即标签体内容)传递给标签处理器对象 * doTag方法   + 用于完成所有的标签逻辑，包括输出、迭代、修改标签体内容等。在doTag方法中可以抛出javax.servlet.jsp.SkipPageException异常，用于通知WEB容器不再执行JSP页面中位于结束标记后面的内容。 |

### JspFragment类介绍

javax.servlet.jsp.tagext.JspFragment类是在JSP2.0中定义的，它的实例对象代表JSP页面中的一段符合JSP语法规范的JSP片段，这段JSP片段中不能包含JSP脚本元素，即<%%>

* getJspContext方法

用于返回代表调用页面的JspContext对象

* public abstract void invoke(java.io.Writer out)

用于执行JspFragment对象所代表的JSP代码片段

参数out用于指定将JspFragment对象的执行结果写入到哪个输出流对象中，如果传递给参数out的值为null，则将执行结果写入到JspContext.getOut()方法返回的输出流对象中。(简而言之，可以理解为写给浏览器)

### 自定义标签的作用

1. 控制标签体内容是否输出

|  |
| --- |
| //1.1 得到标签体内容  JspFragment jspBody = **this**.getJspBody();    /\*\*  \* 执行invoke方法： 把标签体内容输出到指定的Writer对象中  \*/  //1.2 往浏览器输出内容，writer为null就是默认往浏览器输出  //JspWriter out = this.getJspContext().getOut();  //jspBody.invoke(out);  jspBody.invoke(**null**);//等价于上面的代码 |

2）控制标签余下内容是否输出

|  |
| --- |
| /\*\*  \* 2)控制标签余下内容是否输出  \* 输出: 什么都不干！  \* 不输出： 抛出SkipPageException异常  \*/  **throw** **new** SkipPageException(); |

3）控制重复输出标签体内容

4）改变标签体内容

5）制作带属性的标签

步骤：

* 1. 在标签处理器中添加一个成员变量和setter方法
  2. 在TLD文件中描术标签的属性

|  |
| --- |
| //1.声明属性的成员变量  **private** Integer num;    //2.关键点： 必须提供公开的setter方法，用于给属性赋值  **public** **void** setNum(Integer num) {  **this**.num = num;  } |

### Tld文件示例

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <taglib xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-jsptaglibrary\_2\_1.xsd"*  version=*"2.1"*>  <!-- 标签库的版本 -->  <tlib-version>1.1</tlib-version>  <!-- 标签库前缀 -->  <short-name>jy</short-name>  <!-- tld文件的唯一标记 -->  <uri>http://bj.jiyun.cn</uri>  <!-- 一个标签的声明 -->  <tag>  <!-- 标签名称 -->  <name>showIp</name>  <!-- 标签处理器类的全名 -->  <tag-class>gz.itcast.a\_tag.ShowIpTag</tag-class>  <!-- 输出标签体内容格式 -->  <body-content>scriptless</body-content>  </tag>      <tag>  <name>demoTag</name>  <tag-class>gz.itcast.a\_tag.DemoTag</tag-class>  <body-content>scriptless</body-content>  <!-- 属性声明 -->  <attribute>  <!-- 属性名称 -->  <name>num</name>  <!-- 是否必填 -->  <required>true</required>  <!-- 是否支持EL表达式 -->  <rtexprvalue>false</rtexprvalue>  </attribute>  </tag>  </taglib> |

### Tld文件的标签属性

|  |  |  |
| --- | --- | --- |
| **元素名** | **是否必须指定** | **描 述** |
| **description** | **否** | **用于指定属性的描述信息。** |
| **name** | **是** | **用于指定属性的名称。属性名称是大小写敏感的，并且不能以jsp、\_jsp、java和sun开头。** |
| **required** | **否** | **用于指定在JSP页面中调用自定义标签时是否必须设置这个属性。其取值包括true和false，默认值为false，true表示必须设置，否则可以设置也可以不设置该属性。** |
| **rtexprvalue** | **否** | **rtexprvalue是runtime expression value（运行时表达式）的英文简写，用于指定属性值是一个静态值或动态值。其取值包括true和false，默认值为false，false表示只能为该属性指定静态文本值，例如"123"；true表示可以为该属性指定一个JSP动态元素，动态元素的结果作为属性值，例如JSP表达式<%=value %>。** |
| **type** | **否** | **用于指定属性值的Java类型。** |

### 输出标签体内容格式

|  |
| --- |
| <!-- 一个标签的声明 -->  <tag>  <!-- 标签名称 -->  <name>showIp</name>  <!-- 标签处理器类的全名 -->  <tag-class>gz.itcast.a\_tag.ShowIpTag</tag-class>  <!-- 输出标签体内容格式 -->  <body-content>scriptless</body-content>  </tag> |

|  |
| --- |
| * JSP：表示支持jsp脚本表达式取值，即<%= %>。只能在传统标签用，简单标签不支持！ * empty：没有标签体 * scriptless：标签体可以包含 el 表达式和 JSP 动作元素，但不能包含 JSP 的脚本元素 * tagdependent：表示标签体交由标签本身去解析处理。若指定 tagdependent，在标签体中的所有代码都会原封不动的交给标签处理器，而不是将执行结果传递给标签处理器 |

### 自定义标签案例

1. 高仿核心标签库：c:if

|  |
| --- |
| package com;  import java.io.IOException;  import javax.servlet.jsp.JspException;  import javax.servlet.jsp.JspWriter;  import javax.servlet.jsp.SkipPageException;  import javax.servlet.jsp.tagext.JspFragment;  import javax.servlet.jsp.tagext.SimpleTagSupport;  public class IfTag extends SimpleTagSupport {  private boolean test;  public void setTest(boolean test) {  this.test = test;  }  @Override  public void doTag() throws JspException, IOException {  // TODO Auto-generated method stub  if(test){ // test结果是true，就输出if标签里面的标签体内容  //获取标签体内容  JspFragment jspFragment=this.getJspBody();  JspWriter out=this.getJspContext().getOut();  //输出标签体内容  jspFragment.invoke(out);  }  /\*//不让标签体外的东西输出  throw new SkipPageException();\*/  }  } |

# JDBC技术

## 什么是jdbc？

使用java代码（程序）发送sql语句的技术，就是jdbc技术！！！！

SUN公司为**了简化、**统一对数据库的操作，定义了一套Java操作数据库的规范，称之为JDBC。

JDBC全称为：Java Data Base Connectivity（java数据库连接），它主要由接口组成。

开发JDBC应用需要以上2个包的支持外，还需要导入相应JDBC的数据库实现(即数据库驱动)

## 使用jdbc发送sql几要素

登录数据库服务器（连接数据库服务器）

数据库的IP地址

端口

数据库用户名

密码

要操作的数据库名

## Jdbc连接数据库

### 方式1：通过创建驱动程序类对象

|  |
| --- |
| /\*\*  \* 第一种方法  \* @throws Exception  \*/  @Test  public void test1() throws Exception{  //1.创建驱动程序类对象  Driver driver = new com.mysql.jdbc.Driver(); //新版本  //Driver driver = new org.gjt.mm.mysql.Driver(); //旧版本    //设置用户名和密码  Properties props = new Properties();  props.setProperty("user", user);  props.setProperty("password", password);    //2.连接数据库，返回连接对象  Connection conn = driver.connect(url, props);    System.out.println(conn);  } |

### 方式2：通过驱动管理类

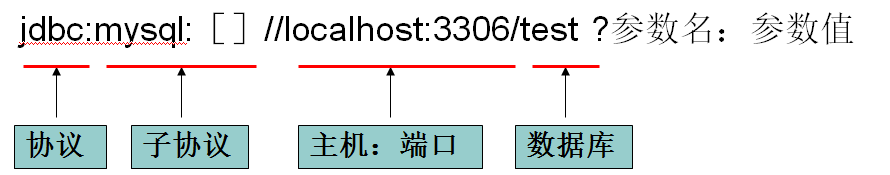
|  |
| --- |
| /\*\*  \* 使用驱动管理器类连接数据库(注册了两次，没必要)  \* @throws Exception  \*/  @Test  public void test2() throws Exception{  Driver driver = new com.mysql.jdbc.Driver();  //Driver driver2 = new com.oracle.jdbc.Driver();  //1.注册驱动程序(可以注册多个驱动程序)  DriverManager.registerDriver(driver);  //DriverManager.registerDriver(driver2);    //2.连接到具体的数据库  Connection conn = DriverManager.getConnection(url, user, password);  System.out.println(conn);    } |

### 方式3：通过加载驱动程序类

|  |
| --- |
| /\*\*  \* （推荐使用这种方式连接数据库）  \* 推荐使用加载驱动程序类 来 注册驱动程序  \* @throws Exception  \*/  @Test  public void test3() throws Exception{    //通过得到字节码对象的方式加载静态代码块，从而注册驱动程序  Class.forName("com.mysql.jdbc.Driver");  //2.连接到具体的数据库  Connection conn = DriverManager.getConnection(url, user, password);  System.out.println(conn);  } |

### 数据库URL

* URL用于标识数据库的位置，程序员通过URL地址告诉JDBC程序连接哪个数据库，URL的写法为：



* 常用数据库URL地址的写法：
  + **Oracle写法：jdbc:oracle:thin:@localhost:1521:sid**
  + **SqlServer—jdbc:microsoft:sqlserver://localhost:1433; DatabaseName=sid**
  + **MySql—jdbc:mysql://localhost:3306/sid**
* Mysql的url地址的简写形式： **jdbc:mysql:///sid**
* 常用属性：useUnicode=true&characterEncoding=UTF-8

## Connection接口

* 它用于代表数据库的链接，Collection是数据库编程中最重要的一个对象，客户端与数据库所有交互都是通过connection对象完成的，这个对象的常用方法：
  + createStatement()：创建向数据库发送sql的statement对象。
  + prepareStatement(sql) ：创建向数据库发送预编译sql的PrepareSatement对象。
  + prepareCall(sql)：创建执行存储过程的CallableStatement对象。
  + setAutoCommit(boolean autoCommit)：设置事务是否自动提交。
  + commit() ：在链接上提交事务。
  + rollback() ：在此链接上回滚事务

## Statement接口

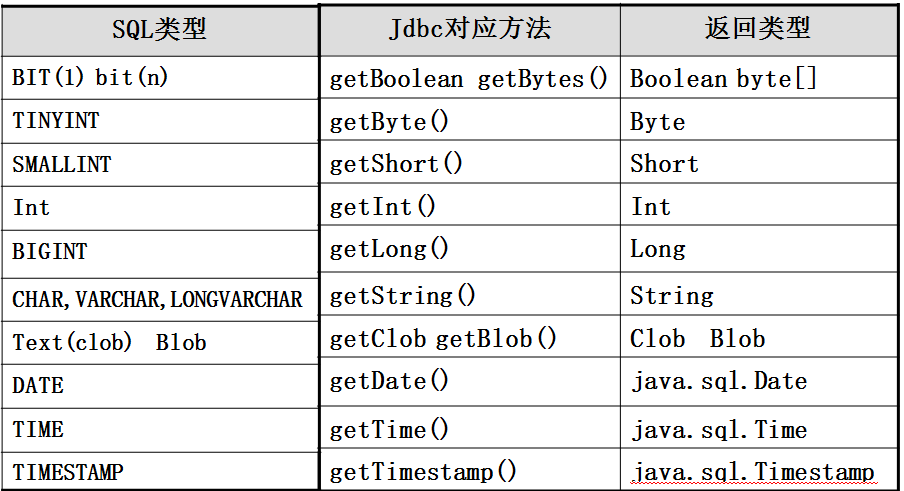
* Statement对象用于向数据库发送SQL语句， Statement对象常用方法：
  + executeQuery(String sql) ：用于向数据发送查询语句。
  + executeUpdate(String sql)：用于向数据库发送insert、update或delete语句
  + execute(String sql)：用于向数据库发送任意sql语句
  + addBatch(String sql) ：把多条sql语句放到一个批处理中。
  + executeBatch()：向数据库发送一批sql语句执行

## ResultSet接口

* ResultSet用于代表SQL语句的执行结果。Resultset封装执行结果时，采用的类似于表格的方式，它维护了一个指向表格数据行的**游标，**初始的时候，游标在第一行之前，调用ResultSet.next() 方法，可以使游标指向具体的数据行，进行调用方法获取该行的数据。
* ResultSet中用于获取数据的get方法：
  + 获取任意类型的数据
    - getObject(int index)
    - getObject(string columnName)
  + 获取指定类型的数据，例如：
    - getString(int index)
    - getString(String columnName)
  + 提问：数据库中列的类型是varchar，获取该列的数据调用什么方法？Int类型呢？bigInt类型呢？Boolean类型？

注意：列的索引从1开始。

* 常用数据类型转换表

****

* 操作结果集的方法
  + next()：移动到下一行
  + previous()：移动到前一行
  + absolute(int row)：移动到指定行
  + beforeFirst()：移动resultSet的最前面。
  + afterLast() ：移动到resultSet的最后面

## 使用statement对数据库进行CRUD

1. **使用statement完成数据增加**

|  |
| --- |
| Statement st = conn.createStatement();  String sql = "insert into user(….) values(…..) ";  **int** num = st.executeUpdate(sql);  **if**(num>0){  System.*out*.println("插入成功！！！");  } |

1. **使用statement完成数据修改**

|  |
| --- |
| Statement st = conn.createStatement();  String sql = “update user set name=‘’ where name=‘’";  **int** num = st.executeUpdate(sql);  **if**(num>0){  System.*out*.println(“修改成功！！！");  } |

1. **使用statement完成数据删除**

|  |
| --- |
| Statement st = conn.createStatement();  String sql = “delete from user where id=1;  **int** num = st.executeUpdate(sql);  **if**(num>0){  System.*out*.println(“删除成功！！！");  } |

1. **使用statement完成数据读取**

|  |
| --- |
|  |

案例：编写程序对User表进行增删改查操作

## 释放资源

* Jdbc程序运行完后，切记要释放程序在运行过程中，创建的那些与数据库进行交互的对象，这些对象通常是ResultSet, Statement和Connection对象。
* 特别是Connection对象，它是非常稀有的资源，用完后必须马上释放，如果Connection不能及时、正确的关闭，极易导致系统宕机。Connection的使用原则是尽量晚创建，尽量早的释放。（使用原则：**尽量晚的创建，尽量早的释放**）
* 为确保资源释放代码能运行，资源释放代码也一定要放在finally语句中

## 读取数据库连接配置信息

|  |
| --- |
| /\*\*  \* 静态代码块中（只加载一次）  \*/  **static**{  **try** {  //读取db.properties文件  Properties props = **new** Properties();  /\*\*  \* . 代表java命令运行的目录  \* 在java项目下，. java命令的运行目录从项目的根目录开始  \* 在web项目下， . java命令的而运行目录从tomcat/bin目录开始  \* 所以不能使用点.  \*/  //FileInputStream in = new FileInputStream("./src/db.properties");    /\*\*  \* 使用类路径的读取方式  \* / : 斜杠表示classpath的根目录  \* 在java项目下，classpath的根目录从bin目录开始  \* 在web项目下，classpath的根目录从WEB-INF/classes目录开始  \*/  InputStream in = JdbcUtil.**class**.getResourceAsStream("/db.properties");    //加载文件  props.load(in);  //读取信息  *url* = props.getProperty("url");  *user* = props.getProperty("user");  *password* = props.getProperty("password");  *driverClass* = props.getProperty("driverClass");      //注册驱动程序  Class.*forName*(*driverClass*);  } **catch** (Exception e) {  e.printStackTrace();  System.*out*.println("驱程程序注册出错");  }  } |

关于Class.getResourceAsStream(String path)

|  |
| --- |
| Class.getResourceAsStream(String path) ： path 不以’/'开头时默认是从此类所在的包下取资源，以’/'开头则是从  ClassPath根下获取。其只是通过path构造一个绝对路径，最终还是由ClassLoader获取资源。  **getResourceAsStream 用法大致有以下几种：**  第一： 要加载的文件和.class文件在同一目录下，例如：com.x.y 下有类me.class ,同时有资源文件myfile.xml  那么，应该有如下代码：  me.class.getResourceAsStream("myfile.xml");  第二：在me.class目录的子目录下，例如：com.x.y 下有类me.class ,同时在 com.x.y.file 目录下有资源文件myfile.xml  那么，应该有如下代码：  me.class.getResourceAsStream("file/myfile.xml");  第三：不在me.class目录下，也不在子目录下，例如：com.x.y 下有类me.class ,同时在 com.x.file 目录下有资源文件myfile.xml  那么，应该有如下代码：  me.class.getResourceAsStream("/com/x/file/myfile.xml"); |

## PreparedStatement接口

* PreperedStatement是Statement的孩子，它的实例对象可以通过调用Connection.prepareStatement()方法获得，相对于Statement对象而言：
  + PreperedStatement可以避免SQL注入的问题。
  + Statement会使数据库频繁编译SQL，可能造成数据库缓冲区溢出。PreparedStatement 可对SQL进行预编译，从而提高数据库的执行效率。
  + 并且PreperedStatement对于sql中的参数，允许使用占位符的形式进行替换，简化sql语句的编写。

## 使用PreparedStatement执行sql语句

|  |
| --- |
| **public** **class** Demo1 {  /\*\*  \* 增加  \*/  @Test  **public** **void** testInsert() {  Connection conn = **null**;  PreparedStatement stmt = **null**;  **try** {  //1.获取连接  conn = JdbcUtil.*getConnection*();    //2.准备预编译的sql  String sql = "INSERT INTO student(NAME,gender) VALUES(?,?)"; //?表示一个参数的占位符    //3.执行预编译sql语句(检查语法)  stmt = conn.prepareStatement(sql);    //4.设置参数值  /\*\*  \* 参数一： 参数位置 从1开始  \*/  stmt.setString(1, "李四");  stmt.setString(2, "男");    //5.发送参数，执行sql  **int** count = stmt.executeUpdate();    System.*out*.println("影响了"+count+"行");    } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException(e);  } **finally** {  JdbcUtil.*close*(conn, stmt);  }  }    /\*\*  \* 修改  \*/  @Test  **public** **void** testUpdate() {  Connection conn = **null**;  PreparedStatement stmt = **null**;  **try** {  //1.获取连接  conn = JdbcUtil.*getConnection*();    //2.准备预编译的sql  String sql = "UPDATE student SET NAME=? WHERE id=?"; //?表示一个参数的占位符    //3.执行预编译sql语句(检查语法)  stmt = conn.prepareStatement(sql);    //4.设置参数值  /\*\*  \* 参数一： 参数位置 从1开始  \*/  stmt.setString(1, "王五");  stmt.setInt(2, 9);    //5.发送参数，执行sql  **int** count = stmt.executeUpdate();    System.*out*.println("影响了"+count+"行");    } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException(e);  } **finally** {  JdbcUtil.*close*(conn, stmt);  }  }    /\*\*  \* 删除  \*/  @Test  **public** **void** testDelete() {  Connection conn = **null**;  PreparedStatement stmt = **null**;  **try** {  //1.获取连接  conn = JdbcUtil.*getConnection*();    //2.准备预编译的sql  String sql = "DELETE FROM student WHERE id=?"; //?表示一个参数的占位符    //3.执行预编译sql语句(检查语法)  stmt = conn.prepareStatement(sql);    //4.设置参数值  /\*\*  \* 参数一： 参数位置 从1开始  \*/  stmt.setInt(1, 9);    //5.发送参数，执行sql  **int** count = stmt.executeUpdate();    System.*out*.println("影响了"+count+"行");    } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException(e);  } **finally** {  JdbcUtil.*close*(conn, stmt);  }  }    /\*\*  \* 查询  \*/  @Test  **public** **void** testQuery() {  Connection conn = **null**;  PreparedStatement stmt = **null**;  ResultSet rs = **null**;  **try** {  //1.获取连接  conn = JdbcUtil.*getConnection*();    //2.准备预编译的sql  String sql = "SELECT \* FROM student";    //3.预编译  stmt = conn.prepareStatement(sql);    //4.执行sql  rs = stmt.executeQuery();    //5.遍历rs  **while**(rs.next()){  **int** id = rs.getInt("id");  String name = rs.getString("name");  String gender = rs.getString("gender");  System.*out*.println(id+","+name+","+gender);  }    } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException(e);  } **finally** {  //关闭资源  JdbcUtil.*close*(conn,stmt,rs);  }  }  } |

**推荐使用PreparedStatement**

## 实现页面分页

## CallableStatement执行存储过程

|  |
| --- |
| /\*\*  \* 使用CablleStatement调用存储过程  \* **@author** APPle  \*  \*/  **public** **class** Demo1 {  /\*\*  \* 调用带有输入参数的存储过程  \* CALL pro\_findById(4);  \*/  @Test  **public** **void** test1(){  Connection conn = **null**;  CallableStatement stmt = **null**;  ResultSet rs = **null**;  **try** {  //获取连接  conn = JdbcUtil.*getConnection*();    //准备sql  String sql = "CALL pro\_findById(?)"; //可以执行预编译的sql    //预编译  stmt = conn.prepareCall(sql);    //设置输入参数  stmt.setInt(1, 6);    //发送参数  rs = stmt.executeQuery(); //注意： 所有调用存储过程的sql语句都是使用executeQuery方法执行！！！    //遍历结果  **while**(rs.next()){  **int** id = rs.getInt("id");  String name = rs.getString("name");  String gender = rs.getString("gender");  System.*out*.println(id+","+name+","+gender);  }    } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException(e);  } **finally** {  JdbcUtil.*close*(conn, stmt ,rs);  }  }    /\*\*  \* 执行带有输出参数的存储过程  \* CALL pro\_findById2(5,@NAME);  \*/  @Test  **public** **void** test2(){  Connection conn = **null**;  CallableStatement stmt = **null**;  ResultSet rs = **null**;  **try** {  //获取连接  conn = JdbcUtil.*getConnection*();  //准备sql  String sql = "CALL pro\_findById2(?,?)"; //第一个？是输入参数，第二个？是输出参数    //预编译  stmt = conn.prepareCall(sql);    //设置输入参数  stmt.setInt(1, 6);  //设置输出参数(注册输出参数)  /\*\*  \* 参数一： 参数位置  \* 参数二： 存储过程中的输出参数的jdbc类型 VARCHAR(20)  \*/  stmt.registerOutParameter(2, java.sql.Types.*VARCHAR*);    //发送参数，执行  stmt.executeQuery(); //结果不是返回到结果集中，而是返回到输出参数中    //得到输出参数的值  /\*\*  \* 索引值： 预编译sql中的输出参数的位置  \*/  String result = stmt.getString(2); //getXX方法专门用于获取存储过程中的输出参数    System.*out*.println(result);  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException(e);  } **finally** {  JdbcUtil.*close*(conn, stmt ,rs);  }  }  } |

## 使用jdbc实现批处理

当需要向数据库发送一批SQL语句执行时，应避免向数据库一条条的发送执行，而应采用JDBC的批处理机制，以提升执行效率

实现批处理，有两种方式

* **Statement.addBatch(sql);**
  + 优点：可以向数据库发送多条不同的sql语句
  + 缺点：sql语句没有预编译。如果发送多条相同的sql语句但是参数不同的，需要重复写多条sql语句
* **PreparedStatement.addBatch();**
  + 优点：发送的是预编译的sql语句，执行效率高
  + 缺点：只能应用在sql语句相同，参数不同的批处理中
* 执行批处理需要用到的方法
  + executeBatch()方法：执行批处理命令
  + clearBatch()方法：清除批处理命令

|  |
| --- |
| /\*\*  \* **使用Statement.addBatch(sql)来批量处理数据**  \* 执行批处理：executeBatch();  \* 清空批处理指令：clearBatch();  \* 能同时执行不同的sql语句  \* 没有预编译，执行效率低  \*/  @Test  **public** **void** test1(){  Connection conn=**null**; //数据库连接对象  Statement statement=**null**; //Statement对象  **try** {  conn=JdbcUtil.*getConnection*();  statement=conn.createStatement();  String username="lucy111";  String pwd="123111";  //插入数据的sql语句  String sql=" insert into tbl\_user(username,pwd) values ('"+username+"','"+pwd+"')"; //向用户表插入数据  String sql2="insert into tbl\_student (age,name) values(24,'张三')"; //朝学生表插入数据  //调用addBatch(sql)方法添加批处理语句  /\*//要插入100个用户  for(int i=0;i<100;i++){  statement.addBatch(sql);  }\*/  //使用批处理朝两个不同的表插入数据  statement.addBatch(sql);  statement.addBatch(sql2);  //执行批处理，使用Statement对象的executeBatch()  statement.executeBatch();  //清除批处理指令 clearBatch()  statement.clearBatch();  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement);  }  } |

|  |
| --- |
| /\*\*  \* **使用PreparedStatement.addBatch()来批量处理数据**  \* 它可以进行sql预编译，提交数据库的执行效率  \* 缺点：只能执行相同的sql语句  \* 执行批处理 excuteBatch()  \* 清空批处理指令 clearBatch();  \*/  @Test  **public** **void** test2(){  Connection conn=**null**; //数据库连接对象  PreparedStatement statement=**null**; //预编译的PreparedStatement对象  **try** {  conn=JdbcUtil.*getConnection*();  String sql="insert into tbl\_user(username,pwd) values(?,? ) ";  statement=conn.prepareStatement(sql);    //先添加到批处理  **for**(**int** i=0;i<100;i++){  statement.setString(1, "lucy"+i); //设置用户名  statement.setString(2, "1233"+i); //设置密码  //添加到批处理  statement.addBatch();  }  //批处理执行  statement.executeBatch();  //清空批处理指令  statement.clearBatch();  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement);  }  } |

## 使用jdbc处理大数据

在实际开发中，程序需要把大文本或二进制数据保存到数据库

* 基本概念：大数据也称之为LOB(Large Objects)，LOB又分为：
  + clob和blob
    - clob用于存储大文本。
    - blob用于存储二进制数据，例如图像、声音、二进制文等。
* 对MySQL而言只有blob，而没有clob，mysql存储大文本采用的是Text，Text和blob分别又分为：
  + - TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT
    - TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB

**大数据的类型：**





对于MySQL中的Text类型，可调用如下方法设置：

|  |
| --- |
| PreparedStatement.setCharacterStream(index, reader, length);  //注意length长度须设置，并且设置为int型 |

对于MySQL中的Text类型，可调用如下方法获取：

|  |
| --- |
| reader = resultSet. getCharacterStream(i);  reader = resultSet.getClob(i).getCharacterStream();  string s = resultSet.getString(i); |

对于MySQL中的BLOB类型，可调用如下方法设置

|  |
| --- |
| PreparedStatement. setBinaryStream(i, inputStream, length); |

对MySQL中的BLOB类型，可调用如下方法获取：

|  |
| --- |
| InputStream in = resultSet.getBinaryStream(i);  InputStream in = resultSet.getBlob(i).getBinaryStream(); |

|  |
| --- |
| **package** bigData;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.FileReader;  **import** java.io.FileWriter;  **import** java.io.InputStream;  **import** java.io.Reader;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** org.junit.Test;  **import** util.JdbcUtil;  /\*\*  \* jdbc操作大数据字段  \* **@author** dell1  \*  \*/  **public** **class** Demo1 {    /\*\*  \* 存储二进制数据  \* 对应的字段在数据库中设置的类型为longblob  \*/  @Test  **public** **void** test1(){  Connection conn=**null**;  PreparedStatement statement=**null**;  **try** {  conn=JdbcUtil.*getConnection*();  String sql="insert into tbl\_user (img) values(?)";  statement=conn.prepareStatement(sql);  //存储二进制数据  FileInputStream input=**new** FileInputStream(**new** File("E:\\pic\\美女\\11.jpg"));  statement.setBinaryStream(1, input);  //执行sql语句  statement.executeUpdate();  input.close();  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement);  }  }    /\*\*  \* 获取数据库中二进制类型的字段值  \* 对应的字段在数据库中设置的类型为 longblob  \*/  @Test  **public** **void** test2(){  Connection conn=**null**;  PreparedStatement statement=**null**;  **try** {  conn=JdbcUtil.*getConnection*();  String sql="select img from tbl\_user where id=132 ";  statement=conn.prepareStatement(sql);  //执行查询的sql语句，获得查询结果集  ResultSet rs= statement.executeQuery();  **if**(rs.next()){  InputStream input=rs.getBinaryStream(1); //获取二进制类型的字段，使用getBinaryStream（）方法  **int** len=-1;  **byte**[] buffer=**new** **byte**[1024];  FileOutputStream output=**new** FileOutputStream(**new** File("D:\\0801.jpg"));  **while**((len=input.read(buffer))!=-1){  output.write(buffer, 0, len);  }  output.close();  input.close();  }      } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement);  }  }      /\*\*  \* 存储大文本数据  \* 对应的字段在数据库中设置的类型为longtext  \*/  @Test  **public** **void** test3(){  Connection conn=**null**;  PreparedStatement statement=**null**;  **try** {  conn=JdbcUtil.*getConnection*();  String sql="insert into tbl\_user (instru) values(?)";  statement=conn.prepareStatement(sql);  //存储大文本数据  FileReader reader=**new** FileReader(**new** File("D:\\aa.txt"));  statement.setCharacterStream(1, reader); //设置大文本数据  //执行sql语句  statement.executeUpdate();  reader.close();  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement);  }  }    /\*\*  \* 读取大文本数据  \* 对应的字段在数据库中设置的类型为longtext  \*/  @Test  **public** **void** test4(){  Connection conn=**null**;  PreparedStatement statement=**null**;  ResultSet rs=**null**;  **try** {  conn=JdbcUtil.*getConnection*();  String sql="select instru from tbl\_user where id=133 ";  statement=conn.prepareStatement(sql);  rs=statement.executeQuery(); //得到结果集  **if**(rs.next()){  Reader reader=rs.getCharacterStream(1); //得到reader对象  FileWriter writer=**new** FileWriter(**new** File("D:\\aa1.txt")); //拿到writer  **char**[] charbuff=**new** **char**[1024];  **int** len=-1;  **while**((len=reader.read(charbuff))!=-1){  writer.write(charbuff, 0, len);  }  writer.close();  reader.close();  }  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement,rs);  }  }    } |

## 获取数据库自动生成的主键

|  |
| --- |
| /\*\*  \* 获取插入行的自增长主键值  \*/  @Test  **public** **void** test3(){    Connection conn=**null**; //数据库连接对象  PreparedStatement statement=**null**; //预编译的PreparedStatement对象  **try** {  conn=JdbcUtil.*getConnection*();  String sql="insert into tbl\_user(username,pwd) values(?,? ) ";  //statement=conn.prepareStatement(sql);  **statement=conn.prepareStatement(sql,Statement.*RETURN\_GENERATED\_KEYS*);** //预编译的时候，增加 参数值Statement.RETURN\_GENERATED\_KEYS  //给参数赋值  statement.setString(1, "ddd"); //设置用户名  statement.setString(2, "233s"); //设置密码  //执行sql语句  statement.executeUpdate();  //根据getGeneratedKeys 获得插入行的自增长主键  **ResultSet rSet=statement.getGeneratedKeys();**  **if**(rSet.next()){  **int** keyId=rSet.getInt(1); //获取返回的主键值  System.*out*.println("获取到的自增长的主键为："+keyId);  }    } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  JdbcUtil.*close*(conn, statement);  }    } |

* **注：此参数仅对insert操作有效。**

## JDBC事务

事务指逻辑上的一组操作，组成这组操作的各个单元，要不全部成功，要不全部不成功

### 事务的ACID特性

* **原子性（Atomicity）**原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
* **一致性（Consistency）**事务必须使数据库从一个一致性状态变换到另外一个一致性状态。
* **隔离性（Isolation）**事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。
* **持久性（Durability）**持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响

### JDBC控制事务语句

* **Connection**.setAutoCommit(false); 设置事物不自动开启，需要手动开启，也可以使用 start transaction命令
* **Connection**.rollback(); 设置事物回滚，也可以用命令 rollback
* **Connection**.commit(); 提交事务，也可以使用命令 commit

注意，默认情况下，setAutoCommint(true)为true，默认数执行完每条sql语句，会自动提交事务。也就是说每个SQL操作都是一个事务。  
 一个独立SQL操作什么时候算执行完毕，JDBC规范是这样规定的：  
 对数据操作语言(DML，如insert,update,delete)和数据定义语言(如create,drop)，语句一执行完就视为执行完毕。  
 对select语句，当与它关联的ResultSet对象关闭时，视为执行完毕。  
 对存储过程或其他返回多个结果的语句，当与它关联的所有ResultSet对象全部关闭，所有update count(update,delete等语句操作影响的行数)和output parameter(存储过程的输出参数)都已经获取之后，视为执行完毕

### 事务隔离级别

**JDBC定义了五种事务隔离级别：**

* TRANSACTION\_NONE :不支持事务
* TRANSACTION\_READ\_UNCOMMITTED :允许脏读、不可重复读和幻读。
* TRANSACTION\_READ\_COMMITTED :禁止脏读，但允许不可重复读和幻读。
* TRANSACTION\_REPEATABLE\_READ :禁止脏读和不可重复读，单运行幻读。
* TRANSACTION\_SERIALIZABLE: 禁止脏读、不可重复读和幻读。

什么是脏读、幻读和不可重复读

1. 脏读 ：脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。

例如：

        1.Mary的原工资为1000, 财务人员将Mary的工资改为了8000(但未提交事务)   
        2.Mary读取自己的工资 ,发现自己的工资变为了8000，欢天喜地！   
        3.而财务发现操作有误，回滚了事务,Mary的工资又变为了1000   
          像这样,Mary记取的工资数8000是一个脏数据。

1. 不可重复读 ：是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。   
       例如：   
       1.在事务1中，Mary 读取了自己的工资为1000,操作并没有完成   
       2.在事务2中，这时财务人员修改了Mary的工资为2000,并提交了事务.   
       3.在事务1中，Mary 再次读取自己的工资时，工资变为了2000
2. 幻读 : 是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。   
      例如：  
      目前工资为1000的员工有10人。   
      1.事务1,读取所有工资为1000的员工，然后进行数据的修改。   
      2.这时事务2向employee表插入了一条员工记录，工资也为1000   
      3.事务1再次读取所有工资为1000的员工 共读取到了11条记录

|  |
| --- |
| //设置隔离级别  conn.setTransactionIsolation(Connection.TRANSACTION\_SERIALIZABLE); |

### 设置事务回滚点

|  |
| --- |
| Savepoint sp = conn.setSavepoint(); //设置回滚点  Conn.rollback(sp); //回滚到指定的点  Conn.commit(); //回滚后必须要提交 |

## BeanUtils组件

### 简介

程序中对javabean的操作很频繁， 所以apache提供了一套开源的api，方便对javabean的操作！即BeanUtils组件。

BeanUtils组件， 作用是简化javabean的操作！

用户可以从[www.apache.org下载BeanUtils](http://www.apache.org下载BeanUtils)组件，然后再在项目中引入jar文件！

**使用BenUtils组件：**

1. 引入commons-beanutils-1.8.3.jar核心包
2. 引入日志支持包: commons-logging-1.1.3.jar

### 实例, 基本用法

方法1： 对象属性的拷贝

BeanUtils.copyProperty(admin, "userName", "jack");

BeanUtils.setProperty(admin, "age", 18);

方法2： 对象的拷贝

BeanUtils.copyProperties(newAdmin, admin);

方法3： map数据拷贝到javabean中

【注意：map中的key要与javabean的属性名称一致】

BeanUtils.populate(adminMap, map);

## Java获取元数据

元数据：数据库、表、列的定义信息

**获取数据库元数据**

* Connection.getDatabaseMetaData()
* DataBaseMetaData对象
  + getURL()：返回一个String类对象，代表数据库的URL。
  + getUserName()：返回连接当前数据库管理系统的用户名。
  + getDatabaseProductName()：返回数据库的产品名称。
  + getDatabaseProductVersion()：返回数据库的版本号。
  + getDriverName()：返回驱动驱动程序的名称。
  + getDriverVersion()：返回驱动程序的版本号。
  + isReadOnly()：返回一个boolean值，指示数据库是否只允许读操作。

**获取参数元数据**

* PreparedStatement . getParameterMetaData()
  + 获得代表PreparedStatement元数据的ParameterMetaData对象。
  + Select \* from user where name=? And password=?
* ParameterMetaData对象
  + getParameterCount()
    - 获得指定参数的个数
  + getParameterType(int param)
    - 获得指定参数的sql类型

**获取结果集元数据**

* ResultSet. getMetaData()
  + 获得代表ResultSet对象元数据的ResultSetMetaData对象。
* ResultSetMetaData对象
  + getColumnCount**()**
    - 返回resultset对象的列数
  + getColumnName(int column)
    - 获得指定列的名称
  + getColumnTypeName(int column)
    - 获得指定列的类型

|  |
| --- |
| //1. 数据库元数据  @Test  **public** **void** testDB() **throws** Exception {  // 获取连接  Connection conn = JdbcUtil.getConnection();  // 获取数据库元数据  DatabaseMetaData metaData = conn.getMetaData();// alt + shift + L 快速获取方法返回值    System.*out*.println(metaData.getUserName());  System.*out*.println(metaData.getURL());  System.*out*.println(metaData.getDatabaseProductName());  }    //2. 参数元数据  @Test  **public** **void** testParams() **throws** Exception {  // 获取连接  Connection conn = JdbcUtil.getConnection();  // SQL  String sql = "select \* from dept where deptid=? and deptName=?";  // Object[] values = {"tom","888"};    PreparedStatement pstmt = conn.prepareStatement(sql);  // 参数元数据  ParameterMetaData p\_metaDate = pstmt.getParameterMetaData();  // 获取参数的个数  **int** count = p\_metaDate.getParameterCount();      // 测试  System.*out*.println(count);  }    // 3. 结果集元数据  @Test  **public** **void** testRs() **throws** Exception {  String sql = "select \* from dept ";    // 获取连接  Connection conn = JdbcUtil.getConnection();  PreparedStatement pstmt = conn.prepareStatement(sql);  ResultSet rs = pstmt.executeQuery();  // 得到结果集元数据(目标：通过结果集元数据，得到列的名称)  ResultSetMetaData rs\_metaData = rs.getMetaData();    // 迭代每一行结果  **while** (rs.next()) {  // 1. 获取列的个数  **int** count = rs\_metaData.getColumnCount();  // 2. 遍历，获取每一列的列的名称  **for** (**int** i=0; i<count; i++) {  // 得到列的名称  String columnName = rs\_metaData.getColumnName(i + 1);  // 获取每一行的每一列的值  Object columnValue = rs.getObject(columnName);  // 测试  System.*out*.print(columnName + "=" + columnValue + ",");  }  System.*out*.println();  }    } |

## 项目架构

JSP

Servlet 接受页面的请求，然后响应到页面

Service 业务逻辑处理

Dao 做数据库操作 –baseDao

Bean 实体类 entity

### 使用元数据简化jdbc代码

* 业务背景：系统中所有实体对象都涉及到基本的CRUD操作：
  + 所有实体的CUD操作代码基本相同，仅仅发送给数据库的SQL语句不同而已，因此可以把CUD操作的所有相同代码抽取到工具类的一个update方法中，并定义参数接收变化的SQL语句。
  + 实体的R操作，除SQL语句不同之外，根据操作的实体不同，对ResultSet的映射也各不相同，因此可义一个query方法，除以参数形式接收变化的SQL语句外，可以使用策略模式由qurey方法的调用者决定如何把ResultSet中的数据映射到实体对象中。

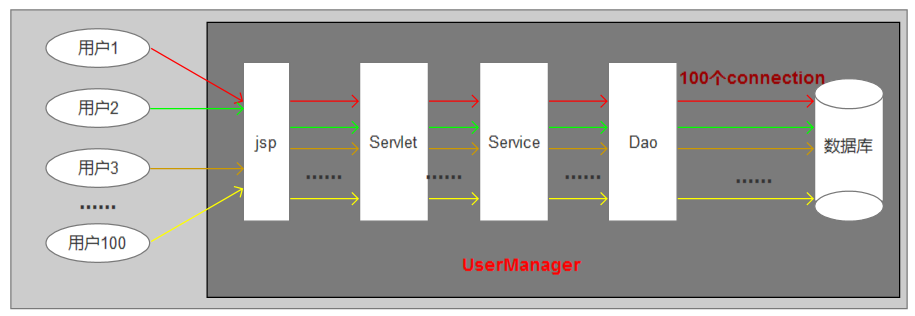
# 连接池

## 引入：

平常获取数据库连接的方式：

|  |
| --- |
| **static** String *url* = "jdbc:mysql://localhost:3306/StudentManager"; //数据库地址  **static** String *un* = "root"; //登陆数据库的账号  **static** String *upwd* = "admin"; //登陆数据库的密码  Class.*forName*("com.mysql.jdbc.Driver");//加载数据库的驱动  Connection *conn* = DriverManager.*getConnection*(*url*,*un*,*upwd*);//连接数据库 |

假如我们有一个应用程序叫UserManager，同时有100个用户访问的时候对应的流程图



问题：每个用户访问UserManager都会去创建一个数据库连接对象Connection，那么这样做到底好不好呢？

各个数据库支持的最大连接数：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据库 | 最大连接数默认 | 极限值 | 如何查看和修改 |
| MySql | 100 | 16384 | 查看：show variables like '%max\_connections%';  修改：打开MySQL配置文件 my.ini 或my.cnf查找max\_connections=100修改即可 |
| Oracle | 150 | ORACLE的连接数理论上是没有上限的，设置多少都可以。实际上与服务器的配置有关，配置越强悍，连接数可以设置的上限越高。 | select value from v$parameter where name = ‘processes’; |

## 为什么会产生连接池的概念？

**创建一个数据库连接的流程**

 Java 应用程序调用 DriverManager.getConnection(String url,Stringuser,Strnigpwd)。

 JDBC 厂商代码（驱动程序或者 DataSource 实现）请求一个来自 JVM 的socket连接。

 JVM 需要检查底层调用的安全方面。例如，swing(applet) 只被允许和产生它们的服务器进行通讯。

 如果允许，调用需要穿过主机网络接口到公司 LAN 上。

 调用可能需要穿过防火墙到达internet。

 调用最终到达目的子网络，在那里它可能需要穿过另一个防火墙。

 调用到达数据库主机。

 数据库服务器处理新的连接请求。

 许可证服务器可能需要查询来确定是否有适当的许可。

 数据库初始化新的客户机连接，包括所有内存和操作系统开销。

 返回调用被送回 JDBC 客户机（在那里它必须穿过所有防火墙和路由器）。

 JVM 收到返回调用，然后创建适当的 Connection 对象。

 请求的 Java 应用程序收到 Connection 对象

既然创建一个数据库连接这么麻烦，为什么不重用连接对象，而是要关闭呢？

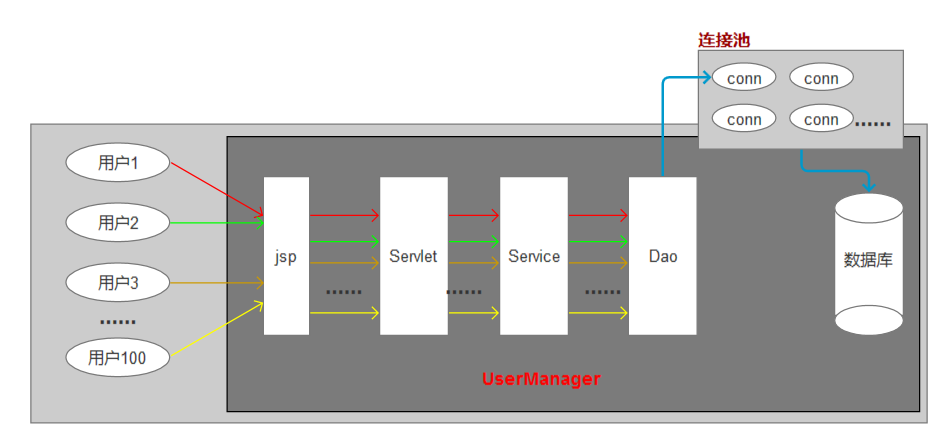
1. 减少反复打开和关闭数据库连接的过程，节省服务器资源
2. 提升服务器运行的效率（建立连接流程复杂，连接时间变长，连接的时间可能会比sql语句执行的时间还长）

3、可以通过连接池的管理机制监视数据库的连接的数量﹑使用情况，为系统开发﹑测试及性能调整提供依据(数据统计可以通过友盟开放平台)。

## 不使用线程池可能引发的问题？

假设网站一天10万访问量，数据库服务器就需要创建10万次连接，极大的浪费数据库的资源，并且极易造成数据库服务器内存溢出、宕机(dang 服务器挂了、服务器累死了)。

## 有连接池之后的流程



## 连接池中到底应该放置多少个连接，怎么管理？

### 关键参数：

**最小连接数**（minConnection）服务器启动时连接池所创建的连接数

**最大连接数**（maxConnection）

注意：

不是创建的连接对象越多越好，具体设置多少，要看系统的访问量和业务需求！

### 管理策略

**有动态和静态两种策略**。动态即每隔一定时间就对连接池进行检测，如果发现连接数量小于最小连接数，则补充相应数量的新连接，以保证连接池的正常运转。静态是发现空闲连接不够时再去检查。

## 开源的、常用的数据库连接池

现在很多WEB服务器(Weblogic, WebSphere, Tomcat)都提供了DataSoruce的实现，即连接池的实现。**通常我们把DataSource的实现，按其英文含义称之为数据源，数据源中都包含了数据库连接池的实现。**

### DBCP（tomcat默认使用DBCP，这个是Tomcat自己的实现）

需要的jar包：(在tomcat中配置是不需要导入jar包的)

**commons-dbcp.jar**：连接池的实现(这个不是tomcat的实现)

**Commons-pool.jar**：连接池实现的依赖库

### 独立使用步骤1：（需要导入jar包）

在src目录下加入dbcp的配置文件：dbcp.properties

|  |
| --- |
| #数据库驱动  driverClassName=com.mysql.jdbc.Driver  #数据库连接地址  url=jdbc\:mysql\://localhost/test  #用户名  username=root  #密码  password=admin  #连接池的最大数据库连接数。设为0表示无限制  maxActive=30  maxIdle=10  #最大建立连接等待时间。如果超过此时间将接到异常。设为-1表示无限制  maxWait=1000  #超过removeAbandonedTimeout时间后，是否进行没用连接（废弃）的回收（默认为false，调整为true)  removeAbandoned=true  #超过时间限制，回收没有用(废弃)的连接（默认为 300秒）  removeAbandonedTimeout=180 |

### 独立使用步骤2：

导入jar包

### 独立使用步骤3：

|  |
| --- |
| public void testDbcp() throws Exception {  // DBCP连接池核心类  BasicDataSource dataSouce = new BasicDataSource();  // 连接池参数配置：初始化连接数、最大连接数 / 连接字符串、驱动、用户、密码  dataSouce.setUrl("jdbc:mysql:///jdbc\_demo"); //数据库连接字符串  dataSouce.setDriverClassName("com.mysql.jdbc.Driver"); //数据库驱动  dataSouce.setUsername("root"); //数据库连接用户  dataSouce.setPassword("root"); //数据库连接密码  dataSouce.setInitialSize(3); // 初始化连接  dataSouce.setMaxActive(6); // 最大连接  dataSouce.setMaxIdle(3000); // 最大空闲时间    // 获取连接  Connection con = dataSouce.getConnection();  con.prepareStatement("delete from admin where id=3").executeUpdate();  // 关闭  con.close();  }    @Test  // 2. 【推荐】配置方式实现连接池 , 便于维护  public void testProp() throws Exception {  // 加载prop配置文件  Properties prop = new Properties();  // 获取文件流  InputStream inStream = App\_DBCP.class.getResourceAsStream("db.properties");  // 加载属性配置文件  prop.load(inStream);  // 根据prop配置，直接创建数据源对象  DataSource dataSouce = BasicDataSourceFactory.createDataSource(prop);    // 获取连接  Connection con = dataSouce.getConnection();  con.prepareStatement("delete from admin where id=4").executeUpdate();  // 关闭  con.close();  } |

## C3P0

c3p0是一个开源的JDBC连接池，它实现了数据源和JNDI绑定。c3p0一般是与Hibernate，Spring等框架一块使用的，也可以单独使用。

**需要的jar包：**

**c3p0.jar**

### 实现第一步：

在src目录下加入C3P0的配置文件：c3p0-config.xml

|  |
| --- |
| <c3p0-config>  <!-- C3P0的缺省(默认)配置， -->  <!-- 如果在代码中“ComboPooledDataSourceds = new ComboPooledDataSource();”这样写就表示使用的是C3P0的缺省(默认)配置信息来创建数据源 -->  <default-config>  <property name="driverClass">com.mysql.jdbc.Driver</property>  <property name="jdbcUrl">jdbc:mysql://localhost:3306/test</property>  <property name="user">root</property>  <property name="password">admin</property>  <!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default:3 -->  <property name="acquireIncrement">5</property>  <!--初始化的连接数，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->  <property name="initialPoolSize">10</property>  <!--连接池中保留的最小连接数 -->  <property name="minPoolSize">5</property>  <!--连接池中保留的最大连接数。Default:15 -->  <property name="maxPoolSize">20</property>  <!--定义在从数据库获取新连接失败后重复尝试的次数。Default: 30 -->  <property name="acquireRetryAttempts">30</property>  <!--两次连接中间隔时间，单位毫秒。Default: 1000 -->  <property name="acquireRetryDelay">1000</property>  <!--连接关闭时默认将所有未提交的操作回滚。Default: false -->  <property name="autoCommitOnClose">false</property>  </default-config>  <!-- C3P0的命名配置， -->  <!-- 如果在代码中“ComboPooledDataSourceds = new ComboPooledDataSource("MySQL");”这样写就表示使用的是name是MySQL的配置信息来创建数据源 -->  <named-config name="MySQL">  <property name="driverClass">com.mysql.jdbc.Driver</property>  <property name="jdbcUrl">jdbc:mysql://localhost:3306/test</property>  <property name="user">root</property>  <property name="password">admin</property>  <property name="acquireIncrement">5</property>  <property name="initialPoolSize">10</property>  <property name="minPoolSize">5</property>  <property name="maxPoolSize">20</property>  </named-config>  </c3p0-config> |

### 实现第二步：

导包

### 实现第三步

|  |
| --- |
| public void testCode() throws Exception {  // 创建连接池核心工具类  ComboPooledDataSource dataSource = new ComboPooledDataSource();  // 设置连接参数：url、驱动、用户密码、初始连接数、最大连接数  dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/jdbc\_demo");  dataSource.setDriverClass("com.mysql.jdbc.Driver");  dataSource.setUser("root");  dataSource.setPassword("root");  dataSource.setInitialPoolSize(3);  dataSource.setMaxPoolSize(6);  dataSource.setMaxIdleTime(1000);    // ---> 从连接池对象中，获取连接对象  Connection con = dataSource.getConnection();  // 执行更新  con.prepareStatement("delete from admin where id=7").executeUpdate();  // 关闭  con.close();  }    @Test  //2. XML配置方式，使用C3P0连接池管理连接  public void testXML() throws Exception {  // 创建c3p0连接池核心工具类  // 自动加载src下c3p0的配置文件【c3p0-config.xml】  ComboPooledDataSource dataSource = new ComboPooledDataSource();// 使用默认的配置  PreparedStatement pstmt = null;    // 获取连接  Connection con = dataSource.getConnection();  for (int i=1; i<11;i++){  String sql = "insert into employee(empName,dept\_id) values(?,?)";  // 执行更新  pstmt = con.prepareStatement(sql);  pstmt.setString(1, "Rose" + i);  pstmt.setInt(2, 1);  pstmt.executeUpdate();  }  pstmt.close();  // 关闭  con.close();    } |

# 过滤器（Filter）

过滤器，设计执行流程：

1. 用户访问服务器
2. 过滤器： 对Servlet请求进行拦截
3. 先进入过滤器， 过滤器处理
4. 过滤器处理完后， 再放行， 此时，请求到达Servlet/JSP
5. Servlet处理
6. Servlet处理完后，再回到过滤器, 最后在由tomcat服务器响应用户；

## 过滤器相关Api

|-- interface Filter 过滤器核心接口

Void init(filterConfig); 初始化方法，在服务器启动时候执行

Void doFilter(request,response,filterChain); 过滤器拦截的业务处理方法

Void destroy(); 销毁过滤器实例时候调用

|-- interface FilterConfig 获取初始化参数信息

|  |  |
| --- | --- |
| String | **[getInitParameter](mk:@MSITStore:E:\\API\\servlet-api_en.chm::/javax/servlet/FilterConfig.html" \l "getInitParameter(java.lang.String))**(java.lang.String name) |
| Enumeration | **[getInitParameterNames](mk:@MSITStore:E:\\API\\servlet-api_en.chm::/javax/servlet/FilterConfig.html" \l "getInitParameterNames())()** |

|-- interface FilterChain 过滤器链参数；一个个过滤器形成一个执行链；

void doFilter(ServletRequest request, ServletResponse response) ; 执行下一个过滤器或放行

|  |
| --- |
| /\*\*  \* 过滤器，测试  \* **@author** Jie.Yuan  \*  \*/  **public** **class** HelloFilter **implements** Filter{    // 创建实例  **public** HelloFilter(){  System.*out*.println("1. 创建过滤器实例");  }  @Override  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {  System.*out*.println("2. 执行过滤器初始化方法");    // 获取过滤器在web.xml中配置的初始化参数  String encoding = filterConfig.getInitParameter("encoding");  System.*out*.println(encoding);    // 获取过滤器在web.xml中配置的初始化参数 的名称  Enumeration<String> enums = filterConfig.getInitParameterNames();  **while** (enums.hasMoreElements()){  // 获取所有参数名称：encoding、path  String name = enums.nextElement();  // 获取名称对应的值  String value = filterConfig.getInitParameter(name);  System.*out*.println(name + "\t" + value);  }  }  // 过滤器业务处理方法： 在请求到达servlet之前先进入此方法处理公用的业务逻辑操作  @Override  **public** **void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** IOException, ServletException {  System.*out*.println("3. 执行过滤器业务处理方法");  // 放行 (去到Servlet)  // 如果有下一个过滤器，进入下一个过滤器，否则就执行访问servlet  chain.doFilter(request, response);    System.*out*.println("5. Servlet处理完成，又回到过滤器");  }  @Override  **public** **void** destroy() {  System.*out*.println("6. 销毁过滤器实例");  }  } |

|  |
| --- |
| <!-- 过滤器配置 -->  <filter>  <!-- 配置初始化参数 -->  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  <init-param>  <param-name>path</param-name>  <param-value>c:/...</param-value>  </init-param>    <!-- 内部名称 -->  <filter-name>hello\_filter</filter-name>  <!-- 过滤器类的全名 -->  <filter-class>cn.itcast.a\_filter\_hello.HelloFilter</filter-class>  </filter>  <filter-mapping>  <!-- filter内部名称 -->  <filter-name>hello\_filter</filter-name>  <!-- 拦截所有资源 -->  <url-pattern>/\*</url-pattern>  </filter-mapping> |

## 对指定的请求拦截

|  |
| --- |
| **/\* 表示拦截所有的请求** |
| <filter-mapping>  <filter-name>hello\_filter2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

默认拦截的类型：(直接访问或者重定向)

<dispatcher>REQUEST</dispatcher>

拦截转发：

<dispatcher>FORWARD</dispatcher>

拦截包含的页面(RequestDispatcher.include(/page.jsp); 对page.jsp也执行拦截)

<dispatcher>INCLUDE</dispatcher>

拦截声明式异常信息：

<dispatcher>ERROR</dispatcher>

|  |
| --- |
| <!-- 配置第二个过滤器 -->  <!-- 演示： 拦截指定的请求 -->  <filter>  <filter-name>hello\_filter2</filter-name>  <filter-class>cn.itcast.a\_filter\_hello.HelloFilter2</filter-class>  </filter>  <filter-mapping>  <filter-name>hello\_filter2</filter-name>  <!-- 1. 拦截所有  <url-pattern>/\*</url-pattern>  -->    <!-- 2. 拦截指定的jsp  <url-pattern>/index.jsp</url-pattern>  <url-pattern>/list.jsp</url-pattern>  -->  <!-- 拦截所有的jsp  <url-pattern>\*.jsp</url-pattern>  -->  <!-- 3. 根据servlet的内部名称拦截  <servlet-name>IndexServlet</servlet-name>  -->  <!-- 拦截指定的servlet  <url-pattern>/index</url-pattern>  -->    <!-- 4. 指定拦截指定的类型 -->  <url-pattern>/\*</url-pattern>  <dispatcher>REQUEST</dispatcher>  <dispatcher>FORWARD</dispatcher>  </filter-mapping> |

工作区间编码、项目编码、request/response、数据库编码一致！

## 案例

### 编码统一处理

|  |
| --- |
| /\*\*  \* 将ServletRequest和ServletResponse强制转换成HttpServletRequest和HttpServletResponse  \*/  **final** HttpServletRequest req=(HttpServletRequest)request;  HttpServletResponse resp=(HttpServletResponse)response;  //设置请求参数的编码  req.setCharacterEncoding(encoding); //只针对post方式有效    //针对get方式的请求数据    /\* \* 出现GET中文乱码，是因为在request.getParameter方法内部没有进行提交方式判断并处理。  \* String name = request.getParameter("userName");  \*  \* 解决：对指定接口的某一个方法进行功能扩展，可以使用代理!  \* 对request对象(目标对象)，创建代理对象！\*/  HttpServletRequest proxy = (HttpServletRequest) Proxy.*newProxyInstance*(  req.getClass().getClassLoader(), // 指定当前使用的累加载器  **new** Class[]{HttpServletRequest.**class**}, // 对目标对象实现的接口类型  **new** InvocationHandler() {  @Override  **public** Object invoke(Object proxy, Method method, Object[] args)  **throws** Throwable {  // **TODO** Auto-generated method stub  // 定义方法返回值  Object returnValue = **null**;  // 获取方法名  String methodName = method.getName();  // 判断：对getParameter方法进行GET提交中文处理  **if** ("getParameter".equals(methodName)) {    // 获取请求数据值【 <input type="text" name="userName">】  String value = req.getParameter(args[0].toString()); // 调用目标对象的方法  // 获取提交方式  String methodSubmit = req.getMethod(); // 直接调用目标对象的方法    // 判断如果是GET提交，需要对数据进行处理 (POST提交已经处理过了)  **if** ("GET".equals(methodSubmit)) {  **if** (value != **null** && !"".equals(value.trim())){  // 处理GET中文  value = **new** String(value.getBytes("ISO-8859-1"),encoding);  }  }  **return** value;  }  **else** {  // 执行request对象的其他方法  returnValue = method.invoke(req, args);  }    **return** returnValue;  }  });    //设置响应结果的编码  resp.setContentType("text/html;charset="+encoding);    //放行  chain.doFilter(proxy, response); |

### 登陆权限判断

<http://localhost:8080/day18/stuDel?stuId=13>

<http://localhost:8080/day18/index.jsp>

怎么判断用户是否已经登陆成功

去session获取用户，如果能获取到，代表已经登陆过，如果没有获取到，或者session为空，代表用户没有登陆

|  |
| --- |
| @Override  **public** **void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** IOException, ServletException {  // **TODO** Auto-generated method stub  /\*\*  \* 将ServletRequest和ServletResponse强制转换成HttpServletRequest和HttpServletResponse  \*/  HttpServletRequest req=(HttpServletRequest)request;  HttpServletResponse resp=(HttpServletResponse)response;  //http://localhost:8080/day18/login.jsp  String reqUri=req.getRequestURI(); // /day18/login.jsp  reqUri=reqUri.substring(reqUri.lastIndexOf("/")); // /login.jsp  **if**(!"/login.jsp".equals(reqUri) && !"/login".equals(reqUri)){  //获取session对象  HttpSession session=req.getSession(**false**);  **if**(session==**null**){ //用户未登陆  //跳转到登陆界面  resp.sendRedirect(req.getContextPath()+"/login.jsp");  }**else**{  UserBean user=(UserBean)session.getAttribute("user"); //去session里面获取user信息  **if**(user!=**null**){ //用户已经登陆  //放行  chain.doFilter(request, response);  }**else**{  //跳转到登陆界面  resp.sendRedirect(req.getContextPath()+"/login.jsp");  }  }  }**else**{ //如果是login.jsp的请求，直接放行  chain.doFilter(request, response);  } |

# 监听器

## 概述

监听器： 主要是用来监听特定对象的创建或销毁、属性的变化的！

是一个实现特定接口的普通java类！

对象：

自己创建自己用 (不用监听)

别人创建自己用 （需要监听）

Servlet中哪些对象需要监听？

request / session / servletContext

分别对应的是request监听器、session相关监听器、servletContext监听器

监听器(listener)

监听器接口：

一、监听对象创建/销毁的监听器接口

Interface ServletRequestListener 监听request对象的创建或销毁

Interface HttpSessionListener 监听session对象的创建或销毁

Interface ServletContextListener 监听servletContext对象的创建或销毁

二、监听对象属性的变化

Interface ServletRequestAttributeListener 监听request对象属性变化: 添加、移除、修改

Interface HttpSessionAttributeListener 监听session对象属性变化: 添加、移除、修改

Interface ServletContextAttributeListener 监听servletContext对象属性变化

三、session相关监听器

Interface HttpSessionBindingListener 监听对象绑定到session上的事件

Interface HttpSessionActivationListener(了解) 监听session序列化及反序列化的事件

实现一个监听器的步骤：

1. 写一个普通的类，实现对应的监听器接口
2. 在web.xml里面配置监听器类

## 生命周期监听器

Interface ServletRequestListener 监听request对象的创建或销毁

Interface HttpSessionListener 监听session对象的创建或销毁

Interface ServletContextListener 监听servletContext对象的创建或销毁

生命周期监听器： 监听对象的创建、销毁的过程！

监听器开发步骤：

1. 写一个普通java类，实现相关接口；
2. 配置(web.xml)

### ServletRequestListener

监听request对象的创建或销毁。

代码：

|  |
| --- |
| /\*\*  \* 监听request对象的创建或销毁  \* **@author** Jie.Yuan  \*  \*/  **public** **class** MyRequestListener **implements** ServletRequestListener{  // 对象销毁  @Override  **public** **void** requestDestroyed(ServletRequestEvent sre) {  // 获取request中存放的数据  Object obj = sre.getServletRequest().getAttribute("cn");  System.*out*.println(obj);  System.*out*.println("MyRequestListener.requestDestroyed()");  }  // 对象创建  @Override  **public** **void** requestInitialized(ServletRequestEvent sre) {  System.*out*.println("MyRequestListener.requestInitialized()");  }  } |
| **Web.xml**  <!-- 监听request对象创建、销毁 -->  <listener>  <listener-class>cn.itcast.a\_life.MyRequestListener</listener-class>  </listener> |
|  |

### HttpSessionListener

监听session对象的创建或销毁。

### ServletContextListener

监听servletContext对象的创建或销毁。

## 属性监听器

监听:request/session/servletContext对象属性的变化！

### ServletRequestAttributeListener

### HttpSessionAttributeListener

### ServletContextAttributeListener

## 其他监听器： session相关监听器

### HttpSessionBindingListener

监听对象绑定/解除绑定到sesison上的事件!

步骤：

对象实现接口； 再把对象绑定/解除绑定到session上就会触发监听代码。

作用：

(上线提醒！)

|  |
| --- |
| /\*\*  \* 监听此对象绑定到session上的过程，需要实现session特定接口  \* **@author** Jie.Yuan  \*  \*/  **public** **class** Admin **implements** HttpSessionBindingListener {  // 对象放入session  @Override  **public** **void** valueBound(HttpSessionBindingEvent event) {  System.*out*.println("Admin对象已经放入session");  }  // 对象从session中移除  @Override  **public** **void** valueUnbound(HttpSessionBindingEvent event) {  System.*out*.println("Admin对象从session中移除！");  }  } |

# 国际化

## Locale 本地化

Java提供了一个本地化的对象！封装当前语言、国家、环境等特征！

|  |
| --- |
| **public** **class** App {  @Test  //1. 本地化对象:Locale  // 封装语言、国家信息的对象，有java.util提供  **public** **void** testLocale() **throws** Exception {  // 模拟中国语言等环境  //Locale locale = Locale.CHINA;  Locale locale = Locale.*getDefault*(); // 当前系统默认的语言环境  System.*out*.println(locale.getCountry()); // CN 国家的简称  System.*out*.println(locale.getDisplayCountry()); // 国家名称  System.*out*.println(locale.getLanguage()); // zh 语言简称    // 模拟美国国家  Locale l\_us = Locale.*US*;  System.*out*.println(l\_us.getCountry());  System.*out*.println(l\_us.getDisplayCountry());  }  } |

## 国际化

### 静态数据国际化

网站中显示的固定文本的国际化： “用户名”“密码“

国际化的软件：

1. 存储所有国家显示的文本的字符串
   1. 文件: properties
   2. 命名： 基础名\_语言简称\_国家简称.properties

例如：msg\_zh\_CN.properties 存储所有中文

msg\_en\_US.properties 存储所有英文

配置文件的名字，除了语言和国家简称，其它的必须要保持一致

1. 程序中获取

ResourceBundle类，可以读取国际化的资源文件!

|  |
| --- |
| // 国际化 - 静态数据  @Test  **public** **void** testI18N() **throws** Exception {    // 中文语言环境  Locale locale = Locale.*US*;    // 创建工具类对象ResourceBundle  ResourceBundle bundle = ResourceBundle.*getBundle*("cn.itcast.f\_i18n.msg", locale);  // 根据key获取配置文件中的值  System.*out*.println(bundle.getString("hello"));  System.*out*.println(bundle.getString("username"));  System.*out*.println(bundle.getString("pwd"));    } |

### Jsp页面国际化

|  |
| --- |
| <html>  <head>  <%  ResourceBundle bundle = ResourceBundle.getBundle("cn.itcast.f\_i18n.msg",request.getLocale());  %>  <title><%=bundle.getString("title") %></title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  </head>    <body>  <form name=*"frmLogin"* action=*"*${pageContext.request.contextPath }*/admin?method=login"* method=*"post"*>  <table align=*"center"* border=*"1"*>  <tr>  <td><%=bundle.getString("username") %></td>  <td>  <input type=*"text"* name=*"userName"*>  </td>  </tr>  <tr>  <td><%=bundle.getString("pwd") %></td>  <td>  <input type=*"password"* name=*"pwd"*>  </td>  </tr>  <tr>  <td>  <input type=*"submit"* value=*"*<%=bundle.getString("submit") %>*"*>  </td>  </tr>  </table>  </form>  </body>  </html> |

### Jsp页面国际化 – 使用jstl标签

JSTL标签：

**核心标签库**

**国际化与格式化标签库**

<fmt:setLocale value=""/> 设置本地化对象

<fmt:setBundle basename=""/> 设置工具类

<fmt:message></fmt:message> 显示国际化文本

格式化数值

<fmt:formatNumber pattern="#.##" value="100.99"></fmt:formatNumber>

格式化日期:

<fmt:formatDate pattern="yyyy-MM-dd" value="${date}"/>

|  |
| --- |
| <html>  <head>  <!-- 一、设置本地化对象 -->  <fmt:setLocale value="${pageContext.request.locale}"/>  <!-- 二、设置工具类 -->  <fmt:setBundle basename=*"cn.itcast.f\_i18n.msg"* var=*"bundle"*/>  <title><fmt:message key=*"title"* bundle="${bundle}"></fmt:message></title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  </head>    <body>  <form name=*"frmLogin"* action=*"*${pageContext.request.contextPath }*/admin?method=login"* method=*"post"*>  <table align=*"center"* border=*"1"*>  <tr>  <td><fmt:message key=*"username"* bundle="${bundle}"></fmt:message></td>  <td>  <input type=*"text"* name=*"userName"*>  </td>  </tr>  <tr>  <td><fmt:message key=*"pwd"* bundle="${bundle}"></fmt:message></td>  <td>  <input type=*"password"* name=*"pwd"*>  </td>  </tr>  <tr>  <td>  <input type=*"submit"* value=*"*<fmt:message key=*"submit"* bundle="${bundle}"/>*"*>  </td>  </tr>  </table>  </form>  </body>  </html> |

格式化标签

|  |
| --- |
| <%  request.setAttribute("date",**new** Date());  %>    <body>  <!--  格式化金额  格式: 0.00 保留2为小数，会自动补0  #.## 保留2为小数，不自动补0  -->  <fmt:formatNumber pattern=*"#.##"* value=*"100.99"*></fmt:formatNumber>    <!-- 格式化日期 -->  <fmt:formatDate pattern=*"yyyy-MM-dd"* value=*"*${date}*"*/>  </body> |

# 文件上传与下载

## 文件上传

使用Apache提供的文件上传组件:FileUpload组件

FileUpload组件:

1. 下载源码

2. 项目中引入jar文件

commons-fileupload-1.2.1.jar 【文件上传组件核心jar包】

commons-io-1.4.jar 【封装了对文件处理的相关工具类】

|  |
| --- |
| **try** {  // 文件上传工厂  FileItemFactory fileFactory=**new** DiskFileItemFactory();  //文件上传核心工具类  ServletFileUpload fileUpload=**new** ServletFileUpload(fileFactory);  //设置文件名的编码  fileUpload.setHeaderEncoding("UTF-8");  **if**(fileUpload.*isMultipartContent*(req)){ //是文件上传表单    List<FileItem> fileItemList=fileUpload.parseRequest(req);  **for**(FileItem fileItem:fileItemList){  String fieldName=fileItem.getFieldName(); //表单元素的name值  String fileName=fileItem.getName(); //文件名称    fileName=fileName.substring(fileName.lastIndexOf("\\"));  InputStream stream=fileItem.getInputStream();  **int** len=-1;  **byte**[] buff=**new** **byte**[1024];  String path=**this**.getServletContext().getRealPath("/upload");  File file=**new** File(path, fileName);  FileOutputStream output=**new** FileOutputStream(file);  **while**((len=stream.read(buff))!=-1){  output.write(buff, 0, len);  }  output.close();  stream.close();  }  }**else**{ //不是文件上传表单    }  } **catch** (FileUploadException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } |

## 文件下载

|  |
| --- |
| // 获取用户下载的文件名称(url地址后追加数据,get)  String fileName = request.getParameter("fileName");  fileName = **new** String(fileName.getBytes("ISO8859-1"),"UTF-8");    // 先获取上传目录路径  String basePath = getServletContext().getRealPath("/upload");  // 获取一个文件流  InputStream in = **new** FileInputStream(**new** File(basePath,fileName));    // 如果文件名是中文，需要进行url编码  fileName = URLEncoder.*encode*(fileName, "UTF-8");  // 设置下载的响应头  response.setHeader("content-disposition", "attachment;fileName=" + fileName);    // 获取response字节流  OutputStream out = response.getOutputStream();  **byte**[] b = **new** **byte**[1024];  **int** len = -1;  **while** ((len = in.read(b)) != -1){  out.write(b, 0, len);  }  // 关闭  out.close();  in.close(); |