# Javascript入门

## 引入

html：负责网页结构

css： 负责网页美观

javascript：负责用户与浏览器交互。

## javacript的来由

1994时，网景公司研发了livescript语言，领航者浏览器（把livescript语言植入到浏览器）

微软公司的IE浏览器，后来自己花钱20亿美金研发jscript

1995年，Sun公司，推出jdk1.1 .谈合作。livescript-》javascript

1998，美国在线收购网景。

2003，直接关闭网景。网景6-7亿美金。2千万美金，组建谋智基金（火狐）

javascript jscript 语法不同。 ECMA（欧洲制造商联盟）

scirpt的语法：

1．基础语法（ECMA规范）

2．BOM编程（没有统一）

3．DOM编程（没有统一）

# javascript基础语言

javascript语法体系

1．EMCA基础语法（统一）

2．BOM编程（不统一）

3．DOM编程（不统一）

## javascript使用

### 注释：

java 单行 // 多行 /\* \*/ 文档注释 /\*\* \*/

css 多行 /\* \*/

javascript 单行 // 多行 /\* \*/

常用的函数：

alert("提示框");

document.write("向浏览器输出内容");

### javascript的使用方法：

#### 1．内部javacript:

a)在html页面中使用<script>标签，在<script>标签体中写js内容

b)弊端：和html代码混杂在一起，不好维护，不好重用

#### 2．外部javascript（推荐使用）

导入外部js文件

注意： 不能使用<script src="01.j/s"> 空标签方式导入

**<script src="js文件"></script>**

## 变量和数据类型

### 1． 定义和赋值变量： var 变量名 = 值;

注意：

1．使用var关键词定义变量，var可以省略的，但是不建议省略var

2) 在js中可以重复定义变量，后面定义的变量会覆盖前面的变量。

3) js是弱类型语言，使用var来定义任何数据类型

4．js中变量的类型是由变量的值决定的，所以只定义不赋值的变量，就是一个未赋值变量（undefined），未赋值的变量不能使用

typeof(变量)： 查看变量的数据类型

### 2． js的数据类型分类：

a．number： 无论整数还是小数，都是number类型

b． string: 无论字符还是字符串，都是string类型

c．boolean

d．object 对象类型

#### 例：

**var a = 10;**

**var b = 3.14;**

**var c = 'c';**

**var d = "hello";**

**var e = true;**

**var f = new Object();**

**var是弱类型，可以是任何类型**

## 类型转换函数

**string->number(整数) : parserInt(变量)**

**string->number(小数): parserFloat(变量)**

**注意：**

**1．使用parseInt函数转换时，转换的内容中的第一个是数值类型，就可以继续转换，直到遇到非数组类型为止。**

#### 实例：

var a = "10";

a = "10a";

document.write("a的类型是(转换前)："+typeof(a)+"<br/>");

a = parseInt(a); //转换

document.write("a的类型是(转换后)："+typeof(a)+",值为："+a+"<br/>");

var b = "3.14";

b = parseFloat(b);

document.write("b的类型是(转换后)："+typeof(b)+",值为："+b+"<br/>");

## 运算符

在js中，布尔值可以用true用1来代表，false用0来代表，和c++一样。

Java不行

## 流程控制语句

if

swtich

for

while

do-while

with

**同C一样。**

for-in

**和java增强for一样**

**作用：**

**1．遍历数组**

**2．对象**

**注意：**

**遍历对象的时候，变量是属性的名称，如果获取属性值，则 p[i];**

var arr = [10,20,30,40];

for(var i in arr){

document.write(arr[i]+",");

}

## 函数

### 函数定义：

function 函数名称(形式参数列表){

语句

}

#### 调用函数:

函数名称(实际参数列表);

注意：

1．js的函数使用function定义，但是形式参数列表不能使用var关键词

2．js的函数可以有返回值，直接使用return关键词返回即可，不需要声明返回值类型

3) js没有方法重载的概念，后面定义的函数会覆盖前面的函数。

4．js中的形式参数和实际参数的数量可以不一致，依然可以调用。

5．js的每个函数中都隐藏了一个叫arguments的数组，这个数组用于接收函数调用时传递过来的实际参数值。

6)arguments数组接收完实际参数后，会逐一的依次从左到右赋值给形式参数,如果实际参数数量大于形式参数，则丢失剩下的实际参数

#### 实例：

function add(a,b,c) { // a=10 b=20 c undefined a+b+c=NaN

var result = a+b+c;

document.write("三个参数的结果为："+result);

}

**//js的每个函数中都隐藏了一个叫arguments的数组，这个数组用于接收函数调用时传递过来的实际参数值。**

add(10,20); //调用无错，输出NaN

add(10,20,30,40); //调用无错，输出60

## 基于对象编程

### String对象

#### 定义String对象

##### 方式一：

var str1 = new String("hello");

var str2 = new String("hello");

document.write("结果："+(str1==str2)+"<br/>");

//valueOf（）:该方法返回对象的内容

document.write("结果："+(str1.valueOf()==str2.valueOf()));

##### 方式二：

var str1 = "hello";

var str2 = "hello";

document.write("结果："+(str1==str2)+"<br/>");

#### 常用方法：

charAt(): 返回指定索引的内容

indexOf(): 返回首次出现指定字符的索引位置

lastIndexOf(): 返回最后出现指定字符的索引位置

fontcolor(): 直接给字符串添加颜色

replace(): 替换指定的字符串

split(): 使用指定字符串切割字符串，返回字符串数组

substring(start,end); 截取字符串，start：开始索引， end:结束索引

substr(strat,[length]) 截取字符串， start：开始索引 length: 截取的字符串长度

### Math对象

#### 常用的方法：

1．ceil()： 向上取整。 如果有小数部分的话，直接+1

2．floor(): 向下取整。如果有小数部分的话，直接丢失小数部分，保利整数位

3．round(): 四舍五入取整。满5进一

4．random(): 生成一个随机的0-1的小数 .包含0，不包含1

5．max(): 返回最大值

6．min(): 返回最小值

### Number对象

#### 定义Number对象

##### 方式一：

var num1 = new Number(20);

var num2 = new Number(20);

##### 方式二：

var num1 = 20;

var num2 = 20;

### Boolean对象

### Date对象

#### 常用方法：

getFullYear() 获取年份

getMonth() 获取月份

getDate() 获取日

date.getDay() 获取星期几

getHours 时

getMinutes 分

getSeconds 秒

##### 网页时钟实例：

<body>

当前的日期为：<span id="dateTip"></span>

</body>

<script type="text/javascript">

function genNewDate(){

//生成一个系统的当前时间，格式：2015-06-01 15:22:30

var date = new Date();

//当前日期时间的字符串

var dateStr = date.getFullYear()+"-"+(date.getMonth()+1)+"-"+date.getDate()+"&nbsp;"

+date.getHours()+":"+date.getMinutes()+":"+date.getSeconds();

//把日期字符串放到span中

**//得到span对象**

**var dateTip = document.getElementById("dateTip"); //读到了没有加载的标签，无法读取的**

**//设置span的innerHTMl属性给span赋值**

**dateTip.innerHTML=dateStr;**

}

//设置定时器，每个多少毫秒调用1次任务（函数）

window.setInterval("genNewDate()",1000);

</script>

### Array数组对象

#### 注意：

1．数组的长度会随着元素的添加而变化，不用担心出现索引位置越界的异常。

2) js的数组可以存放任意类型的元素。

#### 定义Array对象

##### 方式一：创建数组

//1.1 指定数组长度

//var arr = new Array(3);

//1.2 不指定数组长度、默认0个元素

//var arr = new Array();

//1.3 指定具体内容

//var arr = new Array(10,"hello",true);

##### 方式二： 不需要new，直接指定内容

var arr = [10,"java",false];

#### 常用的方法：

**join(字符串): 使用指定的字符串，把数组的所有元素连接在一起，最终组成一个新的字符串**

**reverse(): 反转数组中的元素**

##### 实例：

var arr = ["java","net","php","ios"];

var str = arr.join("-"); **//连接**

document.write(str);

for(var i=0;i<arr.length;i++){ //反转前

document.write(arr[i]+",");

}

document.write("<hr/>");

arr = arr.reverse(); **//反转**

for(var i=0;i<arr.length;i++){ //反转后

document.write(arr[i]+",");

}

## 自定义对象

**java： 使用class来定义对象**

**javascript： 使用function来定义对象**

#### 定义对象方式一：有参数的构造函数 （推荐）

//定义人对象

**function Person(name,age){ //this: 表示当前调用的对象**

**//定义属性 赋值**

**this.name = name;**

**this.age = age;**

**//定义方法**

**this.say = function(){**

**alert("这个对象的方法");**

**}**

**}**

#### 定义对象方式二： 无参数的构造函数

**//定义对象**

**function Person(){**

**}**

**//创建对象**

**var p = new Person();**

**//追加属性**

**p.name = "狗剩";**

**p.age = 14;**

**//追加方法**

**p.say = function(){**

**alert("狗剩的函数");**

**}**

#### 定义对象方式三：使用Object对象。 Object对象可以作为任意对象的模板

**//创建对象**

**var p = new Object();**

**//追加属性**

**p.name = "狗蛋";**

**p.age = 16;**

**//追加方法**

**p.say = function(){**

**alert("狗蛋的函数");**

**}**

#### 定义对象方式四：使用字面量的形式创建对象。 json语言（了解）

**//创建人对象**

**var p = {**

**//定义属性（属性名：属性值）**

**"name":"铁蛋",**

**"age": 20,**

**//定义方法**

**"say":function(){**

**alert("铁蛋的函数");**

**}**

**};**

//查看属性值

document.write(p.name);

document.write(p.age);

//调用对象方法

p.say();

## 原型

**给内置对象追加方法**

### 什么是原型？

1．js的每个对象中都包含有了原型属性（prototype）

2．如果往某个对象的原型属性添加了方法，那么添加的这个方法就会自动地添加到当前对象的方法中。

3) 原型的作用：**给内置对象prototype追加方法的**

**function Array(){**

**//属性**

**this.prototype = new Prototype(); //原型对象,默认存在**

**}**

**添加到原型对象，就会自动地添加到当前对象的方法中**

**function Prototype(){**

**this.search = function(){}**

**}**

### 给Array追加一个search和max方法

**Array.prototype.search** = function(target){

//遍历

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

if(this[i]==target){

return i;

}

}

return -1; //找不到就是-1

}

**Array.prototype.max** = function(){

//存储最大值

var max = this[0];

for(var i=1;i<this.length;i++){

if(this[i]>max){

max = this[i];

}

}

return max;

}

### 给String追加方法

#### 字符串转数组

String.prototype.toCharArray = function(){

//1.创建数组

var charArray = new Array();

//2.遍历字符

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

charArray[i]=this.charAt(i);

}

return charArray;

}

#### 字符串反转

String.prototype.reverse = function(){

//1.把String转为数组

var charArray = this.toCharArray();

//2.对数组进行反转

charArray = charArray.reverse();

//3.把数组变为String

return charArray.join("");

}

var str = "hellojava";

var arr = str.toCharArray();

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

document.write(arr[i]+",");

}

document.write("<hr/>");

str = str.reverse();

document.write(str);

# BOM编程

## 入门

BOM就是浏览器对象模型编程，通过javascript引擎提供的四个浏览器对象，操作浏览器，这叫BOM编程

## window对象（重点）

**window代表的是一个窗口。**

### 常用方法：

**open() 在一个窗口中打开页面**

**setInterval() 设置定时器（执行n次）**

**clearInterval() 清除定时器**

**setTimeout() 设置定时器(只执行1次)**

**clearTimeout() 清除定时器**

**alert() 提示框**

**confirm() 确认提示框**

**propmt() 输入提示框**

**注意：**

因为window对象使用非常频繁，所以当调用js中的window对象的方法时，**可以省略对象名不写**。

### 实例：

function testOpen(){

/\*

参数一： 打开的页面

参数二：打开的方式。 \_self: 本窗口 \_blank: 新窗口（默认）

参数三： 设置窗口参数。比如窗口大小，是否显示任务栏

\*/

**window.open("02.广告页面.html","\_blank","width=300px;height=300px;toolbar=0");**

}

var taskId;

function testInterval(){

/\*

定时器： 每隔n毫秒调用指定的任务（函数）

参数一：指定的任务（函数）

参数二：毫秒数

\*/

**taskId = window.setInterval("testOpen()",3000);**

}

function testClearInterval(){

/\*清除任务

参数一：需要清除的任务ID

\*/

**window.clearInterval(taskId);**

}

var toId;

function testTimeout(){

/\*设置定时任务\*/

**toId = window.setTimeout("testOpen()",3000);**

}

function testClearTimeout(){

/\*清除任务

参数一：需要清除的任务ID

\*/

**window.clearTimeout(toId);**

}

function testAlert(){

window.alert("提示框");

}

function testConfirm(){

/\*

**返回值就是用户操作**

**true: 点击了确定**

**false： 点击了取消**

\*/

**var flag = window.confirm("确认删除吗？一旦删除不能恢复，请慎重！");**

**if(flag){**

**alert("确定删除，正在删除中....");**

**}else{**

**alert("取消了操作");**

**}**

}

function testPrompt(){

/\*

输入提示框

\*/

**var flag = window.prompt("请输入你的U顿密码");**

**if(flag){**

**alert("密码正确，转账中...");**

**}else{**

**alert("取消了操作");**

**}**

}

## location对象

**location对象表示一个窗口中的地址栏**

### 常用属性及方法：

**href属性**： 代表的是地址栏的URL，可以获取和设置URL。URL表示统一资源定位符

**reload方法**： 刷新当前页面

### 实例：

function testHref(){

//alert(window.location.href);

/\*

通过修改location对象的href属性来实现页面的跳转

\*/

**window.location.href="02.广告页面.html";**

}

function testReload(){

//刷新当前页面

**window.location.reload();**

}

## history对象

history对象表示窗口的历史记录栏

### 常用方法：

**forward() 前进到下一页**

**back() 后退上一页**

**go() 跳转到某页（正整数：前进 负整数：后退）**

### 实例：

#### 前进一页

function testForward(){

**window.history.forward();**

**//window.history.go(1);**

}

#### 后退一页

function testBack(){

**window.history.back();**

**//window.history.go(-1);**

}

## screen对象

screen对象代表是一个屏幕

### 常用属性：

availWidth 屏幕宽度（排除了任务栏）

availHeight 屏幕高度（排除了任务栏）

width 屏幕宽度

height 屏幕高度

实例：

**document.write(window.screen.availWidth + "<br/>");**

**document.write(window.screen.availHeight + "<br/>");**

**document.write(window.screen.width + "<br/>");**

**document.write(window.screen．+ "<br/>");**

# 事件编程

## 回顾javase的事件编程

GUI编程事件三个要素：

事件源：按钮JButton JFrame

事件：KeyEvent WindowEvent

监听器：KeyListener WindowListener

GUI编程具体的步骤：

1．创建事件源

2．编程监听器

3．在事件源上注册监听器

javascript事件编程的三个要素：

1．事件源：html标签

2．事件 ：click dblclick mouseover。。。。

3．监听器： 函数

## javascript事件分类：

点击相关的：

**单击： onclick**

**双击： ondblclick**

焦点相关的：（鼠标选中为聚焦）

**聚焦： onfocus**

**失去焦点： onblur**

选项相关的：

**改变选项： onchange**

鼠标相关的：

**鼠标经过： onmouseover**

**鼠标移除： onmouseout**

页面加载相关的：

**页面加载： onload**

**这个事件是在加载完标签后再触发。通用用在body标签中，所以加载完body标签的内容触发。**

## 实例：

<script type="text/javascript">

function testClick(){

alert("触发了单击事件");

}

function testDblClick(){

alert("触发了双击事件");

}

function testFocus(){

//清空输入框的内容

var userName = document.getElementById("userName");

userName.value="";

}

function testBlur(){

//获取用户输入内容

var userName = document.getElementById("userName").value;

//检查用户是否存在

//显示内容到span

//获取到span

var userNameTip = document.getElementById("userNameTip");

if(userName=="eric"){

userNameTip.innerHTML = "用户已经被占用，请更改!".fontcolor("red");

}else{

userNameTip.innerHTML = "恭喜你，该用户可用！".fontcolor("green");

}

}

function testChange(){

//alert("选项改变了");

//获取选中的内容

var jiguan = document.getElementById("jiguan").value;

//alert(jiguan);

var city = document.getElementById("city");

//每次先清空城市select的内容

city.innerHTML = "";

if(jiguan=="广东"){

//把一些option选项放入到城市的select中

var arr = ["广州","珠海","深圳"];

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

city.innerHTML += "<option value='"+arr[i]+"'>"+arr[i]+"</option>";

}

}

if(jiguan=="广西"){

//把一些option选项放入到城市的select中

var arr = ["柳州","桂林","南宁"];

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

city.innerHTML += "<option value='"+arr[i]+"'>"+arr[i]+"</option>";

}

}

if(jiguan=="湖南"){

//把一些option选项放入到城市的select中

var arr = ["长沙","湘潭","郴州"];

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

city.innerHTML += "<option value='"+arr[i]+"'>"+arr[i]+"</option>";

}

}

}

function testOver(){

alert("鼠标经过了div");

}

function testOut(){

alert("鼠标移除了dvi");

}

function testLoad(){

alert("触发了onload方法");

}

</script>

<body onload="testLoad()">

<input type="button" value="单击" **onclick="testClick()"**/>

<input type="button" value="双击" **ondblclick="testDblClick()"**/>

<hr/>

请输入用户名：<input type="text" id="userName" value="4-12位字母或数字" **onfocus="testFocus()**"

**onblur="testBlur()"**/><span id="userNameTip"></span>

<hr/>

请选择籍贯

<!--onchange :表示这个select中的选项发送改变 -->

<select **onchange="testChange()"** id="jiguan">

<option value="广东">广东</option>

<option value="广西">广西</option>

<option value="湖南">湖南</option>

</select>

城市

<select id="city">

<option value="广州">广州</option>

<option value="珠海">珠海</option>

<option value="深圳">深圳</option>

</select>

<hr/>

<div id="div1" **onmouseover="testOver()"** **onmouseout="testOut()"**></div>

</body>

# DOM编程

## 概念

DOM（document Object Model）文档对象模型编程。

## 查询标签对象

**通过document对象获取，document代表一个html页面**

### #通过document对象的集合

作用： 获取多个或者同类的标签对象

#### 常用对象：

**all: 获取所有标签对象**

**forms： 获取所有form标签对象**

**images: 获取所有img标签对象**

**links: 获取所有a标签对象**

#### 实例：

var nodeList = document.all; //返回所有标签对象数组

var nodeList = document.forms; //返回所有form标签对象数组

var nodeList = document.images; //返回所有img标签数组

var nodeList = document.links; //返回所有a标签对象数组

### #通过关系查找标签对象

#### 常用对象：

**父节点： parentNode**

**子节点： childNodes**

**第一个子节点： firstChild**

**最后一个子节点： lastChild**

**下一个兄弟节点： nextSibling**

**上一个兄弟节点： previousSibling**

**节点类型： nodeType**

#### 注意事项：

1．childNodes属性,firstChild属性,lastChild属性获取的子节点包含了以下的元素：

a) 标签 1

b) 空格换行 3

c) html注释 8

在子节点中筛选出标签出来呢？ 可以使用 **nodeType属性： 节点类型**

**取出nodeType值为1的节点，就是标签对象**

#### 实例：

//得到a标签（获取第一个a标签对象）

var aNode = document.links[0];

//alert(aNode.nodeName);

//得到父标签

**var formNode = aNode.parentNode;**

alert(formNode.nodeName);

//得到form的子节点,返回数组

**var nodeList = formNode.childNodes;**

alert(nodeList.length);

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

//筛选出标签对象

**if(nodeList[i].nodeType==1){**

**document.write(nodeList[i].nodeName+"，节点类型："+nodeList[i].nodeType+"<br/>");**

**}**

}

//第一个子节点

**var firstChild = formNode.firstChild;**

alert(firstChild.nodeName);

//最后一个子节点

**var lastChild = formNode.lastChild;**

alert(lastChild.nodeName);

var aNode = document.links[1];

//得到下一个兄弟节点

**var next = aNode.nextSibling;**

alert(next.nodeName);

//得到上一个兄弟节点

**var previous = next.previousSibling;**

alert(previous.nodeName);

### #通过document方法查询标签对象

#### 常用方法：

**document.getElementById("id属性值"); 最常用**

注意：

1．使用该方法获取的标签一定要有id属性

2．在同一个html页面中不要出现两个同名的id

**documetn.getElementsByName("name属性值"); 获取同name属性名的标签列表**

注意：

1．使用该方法获取的标签一定要有name属性

**document.getElementsByTagName("标签名")； 获取相同标签名的标签列表**

#### 实例：

function getValue(){

//获取输入框的内容

//1.得到输入框标签对象

//var userName = document.getElementById("userName");

//2.返回的是标签数组

//var userName = document.getElementsByName("user")[0];

//3.返回标签对象数组

var userName = document.getElementsByTagName("input")[0];

//获取标签对象的value属性值

alert(userName.value);

}

## 修改标签对象属性

### 常用对象属性：

**innerHTML属性**：修改标签体内容 <span>**内容**</span> <div></div> <select></select>

**innerHTML : 设置的标签内的html。**

innerText: 设置标签内容的文本。（以淘汰）

**value属性：** **修改value属性值**。 input type="text"

**src属性：** **修改图片的src属性**。 <img src=""/>

**checked属性：** **修改单选或多项的默认值**。 <input type="radio/checked" checked=""/>

### 实例：

//获取span标签对象

var mySpan = document.getElementById("mySpan");

//设置标签内的HTML内容， 在IE和火狐可以

**mySpan.innerHTML = "<h3>标题</h3>";**

//获取input对象

var userName = document.getElementById("userName");

**userName.value = "name";**

function changeImg(){

//得到标签对象

var img = document.getElementsByTagName("img")[0];

**img.src="mm.jpg";**

}

function qunaxuna(){

/\*

var lq = document.getElementsByName("hobit")[0];

//alert(lq.checked);

//设置状态

lq.checked=true;

\*/

var hList = document.getElementsByName("hobit");

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

**hList[i].checked = true;**

}

}

## 标签对象方法

### 常用方法

添加相关的：

**document.createElement("标签名")； 创建节点对象**

**setAttribute("name","value")； 设置节点的属性**

插入相关的：

**appendChild("标签对象") ; 添加子节点对象**

**insertBefore("新标签对象","指定的对象"); 在指定的对象前面添加子节点对象**

删除：

**removeChild("标签对象"); 删除子节点**

### 实例：

//添加按钮

function addItem(){

//1.创建一个按钮对象

**var input = document.createElement("input"); // <input />**

//2.设置属性

**input.setAttribute("type","button"); // <input type="button"/>**

input.setAttribute("value","新按钮"); // <input type="button" value="新按钮"/>

//3.在html页面中插入这个新建的按钮对象

//3.1 得到body对象

var bodyNode = document.getElementsByTagName("body")[0];

//3.2 把新的按钮对象加入到body对象的子节点中(加到最后一个)

**bodyNode.appendChild(input);**

/\*

//3.2 把新的按钮对象加入到btn按钮的前面

var btn = document.getElementById("btn");

//参数一： 新的对象 参数二： 插入到前面的对象

**bodyNode.insertBefore(input,btn);**

\*/

}

//删除

function delItem(){

//找到需要删除的节点对象

var bodyNode = document.getElementsByTagName("body")[0];

//获取最后一个子标签

var lastChild = bodyNode.lastChild;

//删除子标签

**bodyNode.removeChild(lastChild);**

}

## 动态表格实例

<head>

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

<title>动态表格</title>

<script type="text/javascript">

var count = 2;

function addItem(){

/\*

<tr>

<td>1</td>

<td><input type="text"/></td>

<td><input type="password"/></td>

</tr>

\*/

//1.创建tr对象

var tr = document.createElement("tr");

//2.创建三个td对象

var td1 = document.createElement("td");

td1.innerHTML = count;

var td2 = document.createElement("td");

var input1 = document.createElement("input");

input1.setAttribute("type","text");

//把input放入td中

td2.appendChild(input1);

var td3 = document.createElement("td");

var input2 = document.createElement("input");

input2.setAttribute("type","password");

//把input放入td中

td3.appendChild(input2);

//3.把三个td放入tr中

tr.appendChild(td1);

tr.appendChild(td2);

tr.appendChild(td3);

//4.把tr放在tbody的子节点

var tbody = document.getElementsByTagName("tbody")[0];

tbody.appendChild(tr);

count++;

}

function delItem(){

if(count>2){

//查找需要删除的节点

//tbody最后一个子节点

var tbody = document.getElementsByTagName("tbody")[0];

var lastChild = tbody.lastChild;

tbody.removeChild(lastChild);

count--;

}

}

</script>

</head>

<body>

<table border="1" align="center" width="400px">

<thead>

<tr>

<th>编号</th>

<th>姓名</th>

<th>密码</th>

</tr>

</thead>

<tbody>

<tr>

<td>1</td>

<td><input type="text"/></td>

<td><input type="password"/></td>

</tr>

</tbody>

<tfoot>

<tr>

<td colspan="3"><input type="button" value="添加一行" onclick="addItem()"/><input type="button" value="删除一行" onclick="delItem()"/></td>

</tr>

</tfoot>

</table>

</body>

</html>

# 正则表达式

**创建正则表达式： var 变量 = /正则规则/;**

## 常用规则

[a-z]: 表示匹配字母

\* : 0或多个元素

+: 1个或多个元素

? : 0或1个元素

{n,m} 大于n,小于m的个数

**基本和java一样，看java正则即可。**

## 正则方法：

**test（）： 用于匹配指定的字符串．true：表示匹配成功 ； false; 表示匹配失败**

## 注意事项：

**在js的正则表达式中，如果遇到了符合规则的内容，就代表匹配成功！**

**如果需要和java一样完全匹配，需要添加边界符号**

**开始标记： ^**

**结束标记： $**

### 例：

var str = "12abc";

var reg = /^[0-9]{2}$/; **//不加边界符的话，只要str有两个连着的数字就会匹配成功**

if(reg.test(str)){

alert("成功");

}else{

alert("失败");

}

# XML入门

## 引入

HTML, 超文本标记语言。其语言语法较为松散。

1．标签不区分大小写的！

2．标签可以不匹配的。

由w3c组织，针对html的语法缺陷，去设计了另一门，叫xml语言。

html语言的标签是固定的。而XML标签是可以自定义的。

## XML语言

**XML全称Extend Markup lanuage，中文名：可扩展标记语言。**

**HTML XML**

概念 超文本标记语言 可扩展标记语言

标记数量 是固定的 是不固定的，可扩展的

作用 负责网页的结构 1．描述带关系的数据结构（作为软件的配置文件）

2．装载数据（作为小型的“数据库”）。

### XML作为配置文件实例：

properties配置文件： 键值对 name=eric password=123456

xml可以描述包含与被包含的关系。

<student>

<name></name>

</student>

<student>

<name></name>

</student>

配置使用场景：

1．框架（struts2 ， hibernate，spring），xml作为框架的配 置文件

2．tomcat服务器软件配置文件（web.xml server.xml）

# XML作用

## 作为软件配置文件

**PCweb服务器：**

**学生管理系统 - > 添加学生 -> name=张三,age=20 -> 提交到后台程序 -> 把数据存储到数据库中**

两台通讯条件；

IP地址和端口

java代码： 根据ip和端口连接数据库服务器(ip改动会改变代码)

**properties文件：**

ip=211.34.21.43

port=20

**xml文件：**

<host-list>

<host id="main">

<ip> 211.34.21.43</ip>

<port>20</port>

</host>

<host id="backup">

<ip> 211.34.21.44</ip>

<port>20</port>

</host>

</host-list>

PC数据库服务器1：oracle数据库 211.34.21.43 20

PC数据库服务器2：oracle数据库 211.34.21.44 20

**使用XML可以设置多个同样的配置对象。**

## 作为小型数据库

教师管理系统 教师信息： 姓名 工龄 邮箱地址

传输数据：

字符串： 张|三|20|zhangsan@qq.com|.......|...． （格式不是行业规范的，不通用）

xml格式：

<teacher-list>

<teacher>

<name></name>

<email></email>

<workage></workage>

....

</teacher>

<teacher>

<name></name>

<email></email>

<workage></workage>

....

</teacher>

<teacher-list> （xml是w3c组织的制定的标准）

获取字符串，截取字符串数据。

财务系统

发邮件给教师：

邮箱地址 姓名

方案1：财务系统中也存储了一套教师信息。数据维护成本比较高了。

方案2：财务系统使用的教师数据从教师管理系统中发送过去的。

**使用XML传输数据，格式统一方便解析出需要的数据。**

# XML语法

## 标签

<student>学生</student>

注意：

1．分带标签体的标签，也有空标签。

2．标签是区分大小写

3．标签一定要配对

4．标签中间不能含有空格

5．不能使用数字开头

6．如果没有指定名称空间的话， 标签名中间不能含有冒号

7．在同一个xml文件中，有且仅有一个根标签属性

<student id="1"></student>

注意：

1．一个属性分为属性名和属性值。属性名和值之间使用=号分割。

2．属性值一定要使用单引号或者双引号包含，不能省略，也不能单双混用！

3．一个标签内可以包含多个属性，但是不能出现同名的属性

## 注释

<!—注释内容 --->

## 文档声明

**<?xml version="1.0" encoding="码表"?>**

version: xml的版本号

encoding： 表示解析xml文件内容时使用的码表

xml文件的编码问题：

1．xml文件内容保存的编码（另存为 -> 选择编码保存）

2．解析xml文件内容的编码( <?xml version="1.0" encoding="utf-8"?> )

以上两个编码必须保存一致，才能避免中文乱码问题。

注意：

如果在ecplise开发工具上，编写xml文件，那么xml文件内容自动按照文档声明的encoding的编码保存，所以不会中文乱码问题.

## 转义字符

和html的转义字符一样。

## CDATA块

把CDATA块中的内容全部原样输入

**<![CDATA[ 内容 ]]>**

## 处理指令（不用了，过时了）

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

# XML约束（会看即可）

## 什么是约束？

xml语法： w3c组织对xml文件的编程规则规范。（w3c组织制定的）

xml约束： 由开发者指定的对xml文件内容规范。（开发者根据业务指定的）

## 约束分类

DTD约束：相对简单的，数据类型简单的

场景：hibernate struts2

Schema约束：相对复杂，功能很强大，数据类型非常丰富。schema的出现时为了替代DTD的。

## DTD约束

### 1．dtd的使用方法

内部dtd **<!DOCTYPE 约束根标签 [ dtd内容 ]>**

外部dtd **<!DOCTYPE 约束根标签 SYSTEM "dtd文件地址">**

### 2．语法

#### 约束标签

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

例：

<!ELEMENT note (heading, body+, from\*, to?)>

<!ELEMENT to ANY>

<!ELEMENT from EMPTY >

<!ELEMENT heading (#PCDATA)>

<!ELEMENT body (#PCDATA)>

##### 顺序问题：

(子元素名称 1,子元素名称 2,.....)： 一定要依次出现1,2,...

例： <!ELEMENT note **(heading, body+, from\*, to?)**>

##### 类别

(#PCDATA)： 普通字符串（不能包含子标签）

ANY： 任意内容（可以包含子标签）

EMPTY： 空标签

##### 数量问题：

子元素: 有且仅有1个

+ : 1个或多个

\* : 0个或多个

? : 0个或1个

例： <!ELEMENT note (**heading, body+, from\*, to?**)>

#### 约束属性

<!ATTLIST 元素名称 属性名称 属性类型 默认值>

例： <!ATTLIST body id ID #REQUIRED>

##### 属性类型：控制属性值

CDATA： 普通字符串

(en1|en2|en3...): 枚举中的一个值

ID： 唯一的值

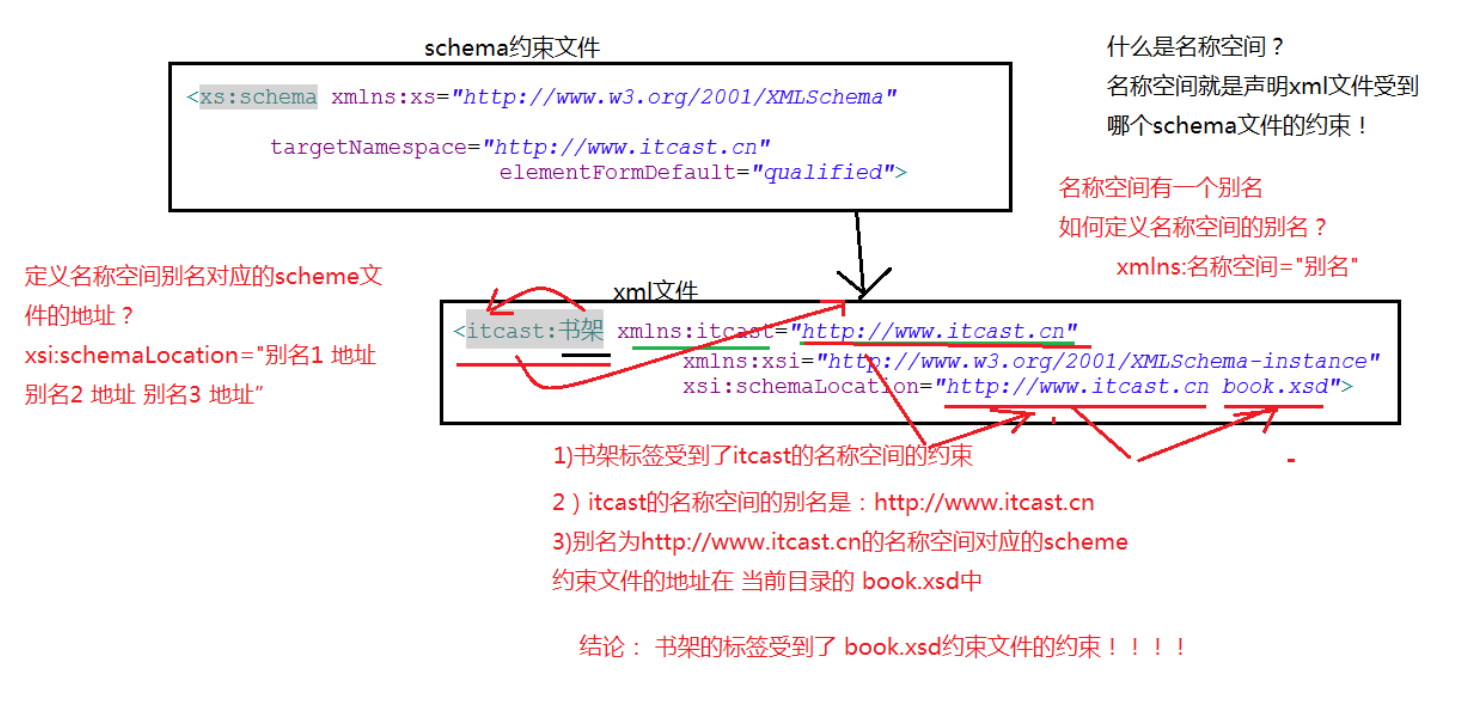
##### 默认值

#REQUIRED： 必须

#IMPLIED： 不是必须的

#FIXED value ： 固定的

## schema约束



### 重点：名称空间

**需要知道标签是被哪个约束文件约束。**

# XML解析

## xml解析方式

### DOM解析：

原理: xml解析引擎一次性把整个xml文件加载进内存，在内存创建一颗document树。

问题：大文件的话，使用dom解析效率比较低，甚至会导致内存溢出。

编码方式： 面向对象编程方式

### 2．SAX解析:

原理: 加载一点，读取一点，处理一点。优势：占用的内存比较小。

编程方式：基于事件编程方式。

### 两种解析的特点：

=============== DOM解析 vs SAX解析=====================

|  |  |  |
| --- | --- | --- |
|  | DOM解析 | SAX解析 |
| 原理 | 一次性加载进内存，构造docuemnt数，内存占用比较大 | 加载一点，读取一点，内存占用相对小 |
| 读取顺序 | 可以读取XML文件任意位置信息，甚至往回读 | 只能是从上往下依次读取，不能往回读 |
| 操作方式 | 可以进行查询，也可以进行修改 | 只能查询 |
| 编码方式 | 面向对象编程方式，更加适合java开发者 | 基于事件编程方式，相对java开发难理解 |

## DOM解析的工具

Sun官方jaxp工具

jdom工具

dom4j工具： 三大框架读取xml文件的程序。（非官方的，第三方的）

dom for java

iphone4s iphone for steven

## dom4j工具使用

1．到dom4j的官方下载 。dom4j-1.6.1.zip

2．导入dom4j的支持jar包、dom4j-1.6.1.jar核心包

# Dom4j解析方法：

## 读XML文件

//1．创建xml解析器对象

**SAXReader reader = new SAXReader();**

//2．读取xml文件

**Document doc = reader.read(new File("./src/contact.xml"));**

## 写XML文件

//1．创建输出流

**OutputStream outStream = new FileOutputStream("e:/contact.xml");**

//2．设置输出的格式

**OutputFormat format = OutputFormat.createPrettyPrint();**

//3．设置输出的编码格式

**format.setEncoding("utf-8");**

//4．创建输出对象

**XMLWriter writer = new XMLWriter(out,format);**

//5．写出数据

**writer.write(doc);**

### 一、设置输出的格式

**//OutputFormat format = OutputFormat.createCompactFormat();**

**//紧凑的格式.空格和换行去掉了！！ 系统上线了使用**

**OutputFormat format = OutputFormat.createPrettyPrint();**

**//漂亮的格式。包含空格和换行。 测试时使用**

### 二、 设置输出的编码格式

/\*\*作用：

\* 1．影响了xml的文档声明的encoding编码

\* 2．影响了xml内容保存的编码

\*/

**format.setEncoding("utf-8");**

## 修改xml信息的步骤

1．读取到原来的xml文件（document对象）

2．操作document对象，改变xml信息（docuement对象）

3．把修改后的document对象写出到xml文件中（覆盖原来的文件）

## 获取的方法：

### 获取标签：

**getRootElement() 获取根标签**

**getName() 获取标签名称**

**element("名称") 获取第一个指定名称的子标签**

**elements("名称") 获取指定名称的所有的子标签**

**elements() 获取所有子标签**

#### 实例：

//1)创建xml解析器对象

**SAXReader reader = new SAXReader(); // 快速声明 ctrl + 2放手 + l**

//2)读取xml文件

**Document doc = reader.read(new File("./src/contact.xml"));**

//1．读取根标签

**Element rootElem = doc.getRootElement();**

System.out.println(rootElem);

//2．获取标签名称

**System.out.println(rootElem.getName());**

//3．获取第一个子标签(根据指定的名称获取第一个子标签)

**Element conElem = rootElem.element("contact");**

//5．获取所有子标签（根据指定的名称获取所有同名子标签）

**List<Element> list = rootElem.elements("contact");**

//4．获取所有子标签（不指定名称）

**List<Element> List = rootElem.elements();**

//遍历List

//1．传统for循环

for(int i=0;i<list.size();i++){

list.get(i); //根据角标获取指定对象

}

//2．for-each循环

for(Element e: list){

System.out.println(e);

}

//3．迭代器

Iterator it = list.iterator();

while(it.hasNext()){ //hasNext(): 判断是否有下一个元素

it.next(); //next():取出当前对象

}

/\*\*

\* **注意，如果需要获取孙标签，首先得拿到子标签，再从子标签来获取孙标签！！！**

\*/

**Element nameElem = rootElem.element("contact").element("name");**

System.out.println(nameElem);

### 获取属性：

**atrributeValue("name") 获取属性值**

**atrribute("name") 根据属性名称获取属性对象**

**atrributes() 获取标签的所有属性对象**

**getName() 通过属性对象拿到属性名**

**getValue() 通过属性对象拿到属性值**

#### 实例：

//1)创建xml解析器

**SAXReader reader = new SAXReader();**

**Document doc = reader.read(new File("./src/contact.xml"));**

/\*\*

\* **注意：获取属性，必须先得到属性所在的标签**

\*/

Element conElem = doc.getRootElement().element("contact");

//1．在标签上获取属性值(根据属性名称获取对应的属性值)

**String value = conElem.attributeValue("id");**

System.out.println(value);

/\*\*

\* **通过集合方法get()拿到第二个标签**

\*/

**Element conElem2 = (Element)doc.getRootElement().elements().get(1);**

System.out.println(conElem2.attributeValue("id"));

//2．根据属性名称获取属性对象

//拿到标签对象

conElem = doc.getRootElement().element("contact");

//拿到属性对象

**Attribute idAttr = conElem.attribute("id");**

//通过属性对象拿到属性名

**String idName = idAttr.getName();**

//通过属性对象拿到属性值

**String idValue = idAttr.getValue();**

System.out.println(idName+"="+idValue);

//3．获取标签的所有属性对象

conElem = doc.getRootElement().element("contact");

**List<Attribute> attrList = conElem.attributes();**

for (Attribute attr : attrList) {

System.out.println(attr.getName()+"="+attr.getValue());

}

### 获取文本：

**getText() 获取当前标签的文本**

**elementText("name") 通过父标签获取指定子标签的文本内容**

#### 实例：

**SAXReader reader = new SAXReader();**

**Document doc = reader.read(new File("./src/contact.xml"));**

/\*\*

\* **注意**：

\* 1．**在xml文件中，空格和换行会作为xml的内容被解析到。**

\* **2．获取文本，要先获取文本所在的标签对象。**

\*/

//1．拿到所在标签上的文本内容

Element nameElem = doc.getRootElement().element("contact").element("name");

**String content = nameElem.getText();**

System.out.println(content);

//2．通过父标签获取指定子标签的文本内容

Element conElem = doc.getRootElement().element("contact");

**content = conElem.elementText("gender");**

System.out.println(content);

## 增加的方法：

**DocumentHelper.createDocument() 添加空文档**

**addElement("name") 添加标签 (根标签只能有一个)**

**addAtribute("name","value") 属性添加属性**

#### 实例：

//1．添加空文档

**Document doc = DocumentHelper.createDocument();**

//2．添加标签

**Element conListElem = doc.addElement("contact-list");**

//doc.addElement("contact-list"); //不能添加两个根标签！！！

//3．添加属性

**conElem.addAttribute("id", "001");**

//4．把文档写出到xml文件中FileOutputStream输出字节流

OutputStream out = new FileOutputStream("e:/contact.xml");

**OutputFormat format = OutputFormat.createPrettyPrint();**

**format.setEncoding("utf-8");**

**XMLWriter writer = new XMLWriter(out,format);**

writer.write(doc);

}

## 修改的方法：

**setValue("Value") 修改指定属性对象的属性值**

**addAttribute("同名属性") 添加同名属性，覆盖属性值**

**setText() 修改指定文本对象的文本内容**

#### 实例：

//修改属性

//1．先得到属性对象，再调用方法修改属性值

Element conElem = doc.getRootElement().element("contact");

Attribute idAttr = conElem.attribute("id");

**idAttr.setValue("003");**

//2．在标签中添加同名的属性，覆盖属性值

Element conElem = doc.getRootElement().element("contact");

**conElem.addAttribute("id", "004");**

//3．修改文本

Element nameElem = doc.getRootElement().element("contact").element("name");

**nameElem.setText("王五");**

//4．把文档写出到xml文件中

OutputStream out = new FileOutputStream("e:/contact.xml");

OutputFormat format = OutputFormat.createPrettyPrint();

format.setEncoding("utf-8");

XMLWriter writer = new XMLWriter(out,format);

writer.write(doc);

## 删除的方法：

**detach() 删除指定的标签或属性**

**getParent().remove(标签/属性) 通过父级标签删除指定的标签或属性**

#### 实例：

//1．删除标签

Element conElem = doc.getRootElement().element("contact");

**conElem.detach(); //自杀**

**conElem.getParent().remove(conElem); //他杀**

//2．删除属性

Attribute idAttr = doc.getRootElement().element("contact").attribute("id");

**idAttr.detach();**

//3．把文档写出到xml文件中

OutputStream out = new FileOutputStream("e:/contact.xml");

OutputFormat format = OutputFormat.createPrettyPrint();

format.setEncoding("utf-8");

XMLWriter writer = new XMLWriter(out,format);

writer.write(doc);

回顾重点内容：

XML解析-**DOM解析**

1．DOM解析的原理：

xml解析引擎把一个xml文件一次性加载进内存，在内存中创建一颗document对象树。通过树上的对象获取或设置xml文件信息

2．基于dom解析原理工具

dom4j工具

3．dom4j使用方法

查：

Document doc= new SAXReader().read("xml文件")；

标签：

element("name") 查询第一个子标签

elements("name") 查询指定名称的所有子标签

elements() 查询所有子标签

属性：

attributeValue("name") 根据属性名获取属性值

attribute("name") 根据属性名获取属性对象

getName() 属性名 / getValue() 属性值

文本：

getText() 获取当前标签的文本内容

elementText("name") 获取子标签文本

增：

文档： DocumentHelper.createDocumnt();

标签： addElement("name")

属性： addAttribute("name","value")

改：

属性值： setValue("修改值") / addAttribute("同名属性","修改值")

文本： setText("文本")

删：

标签、属性： detach() / getParent().remove(标签/属性)

# XPath技术

## 作用

**快速地查询到xml文件中节点。**

当xml文件中的标签层次结构很深，那么如何来快速地定位到所需的标签呢？这就用到了xpath技术！！

## 体验xpath技术

1．导入jaxen-1.1-beta-6.jar dom4j的xpath插件包

2．在dom4j中使用xpath

**List<Node> list = selectNodes("xpath表达式") 查询多个节点**

**Node node = selectSingleNode("xpath表达式") 查询一个节点**

## xpath表达式（重点）

/ 绝对路径 斜杠在最前面，代表xml文件的根。斜杠在中间，表示子元素。

// 相对路径 选择后代元素（不分层次结构）

\* 通配 选择所有元素

[ ] 条件 选择什么条件下的元素。例如 /AAA/BBB[1] 选择第一个BBB子元素

@ 属性 选取属性

= 内容（值）

and 逻辑与

text() 选取文本内

### 实例：

//读取xml文件

**Document doc = new SAXReader().read(new File("./src/contact.xml"));**

//利用xpath方法查询xml文件

String xpath = "";

// ---------> /

**xpath = "/contact-list"; //根标签contact-list**

**xpath = "/contact-list/contact"; //contact-list根标签下的contact子标签**

// ---------> //

**xpath = "//contact"; //选择所有contact标签（不分层次）**

**xpath = "//contact/name"; //选择所有父标签是contact的name标签**

// ---------> \*

**xpath = "/contact-list/\*"; //选择根据标签contact-list下的所有子标签**

**xpath = "/contact-list//\*"; //选择根标签contact-list下的所有后代标签（不分层次结构）**

// ---------> [ ]

**xpath = "//contact[1]"; // 第一个contact标签**

**xpath = "//contact[last()]"; //最后一个contact标签**

// ---------> @

**xpath = "//@id"; // 选择所有id属性**

**xpath = "//contact[@id]"; //选择所有包含id属性的contact标签**

// ---------> =

**xpath = "//contact[@id='002']"; //选择id属性值为002的contact标签**

// ---------> and

**//选取id属性为002,且name属性为eric的contact标签**

**xpath = "//contact[@id='002' and @name='eric']";**

// ---------> text() 选取文本

**xpath = "//contact[@id='001']/name[1]/text()"; //选择第一个name标签的文本**

**xpath = "//name[text()='陈六']"; //文本内容为”陈六“的name标签**

**List<Node> list = doc.selectNodes(xpath); // Node是Element的父类**

for (Node node : list) {

System.out.println(node);

}

## 案例

模拟登录效果：

用户输入用户名密码 -> 到“数据库”中是否存在用户名和密码的用户 ->

存在： 登录成功

不存在： 登录失败

public class Dem {

public static void main(String[] args) throws Exception {

//1．获取用户输入的用户名和密码

//注意： System.in是字节流，BufferedReader是字符流，字节流转字符流需要使用转换流

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("请输入用户名：");

String name = br.readLine();

System.out.println("请输入密码：");

String password = br.readLine();

//2．在user.xml中查询

//name标签文本为 ' rose ', password标签文本为' 123456 '的user标签

Document doc = new SAXReader().read(new File("./src/user.xml"));

//查询文本为' xxx '的name标签

**Element nameElem = (Element)doc.selectSingleNode("//user/name[text()='"+name+"']");**

//判断name标签是否存在

if(nameElem!=null){

//存在

//判断密码是否正确

Element userElem = nameElem.getParent();

//判断password子标签的文本内容

String dbpwd = userElem.elementText("password");

//数据库的密码和用户输入的密码匹配

if(password.equals(dbpwd)){

System.out.println("登录成功");

}else{

//不正确

System.out.println("密码错误，请重新输入！");

}

}else{

//不存在

System.out.println("该用户名不存在的！");

}

}

}

# SAX解析

## SAX解析工具

sun公司官方的sax解析工具。在jdk中包含sax解析工具的api.org.xml.sax.\*;

## SAX解析的开发步骤

事件编程模式三要求：

事件源： xml文件

事件： 解析到开始标签（包含属性），解析到结束标签，解析文本内容

监听器： 继承了DefaultHandler接口的对象

//1．创建SAXParser解析对象

**SAXParser parser = SAXParserFactory.newInstance().newSAXParser();**

//创建事件处理程序（一个继承了DefaultHandler接口的对象）

**MyDefaultHandler2 handler2 = new MyDefaultHandler();**

//2．解析xml文件

/\*\*

\* 参数一： 需要解析的xml文件

\* 参数二： 指定的继承了DefaultHandler接口的对象

\*/

**parser.parse(new File("./src/contact.xml"), handler2);**

## 常用方法

**startElement(String uri, String localName, String qName, Attributes attributes)**

**characters(char[] ch, int start, int length)**

**endElement(String uri, String localName, String qName)**

## SAX案例

### DefaultHandler实现对象

public class MyDefaultHandler1 extends DefaultHandler{

/\*\*

\* **遇到xml文档的开始位置触发此方法**

\*/

@Override

public void **startDocument()** throws SAXException {

System.out.println("MyDefaultHandler1.startDocument()");

}

/\*\*

\* **遇到每个开始标签触发次方法**

\* @param **qName: 表示当前读到的开始标签名称**

\* @param **attributes : 属性列表**

\*/

@Override

public void **startElement(String uri, String localName, String qName, Attributes attributes)** throws SAXException {

System.out.println("MyDefaultHandler1.startElement()->"+qName);

}

/\*\*

\* **遇到文本内容触发此方法**

\* 如何获取当前读到的内容？

\* **char[]: 表示到目前为止读到的文本内容**

\* **start: 表示当前内容的起始位置**

\* **length: 表示当前内容的长度**

\*/

@Override

public void **characters(char[] ch, int start, int length)** throws SAXException {

/\*\*

\* char[]内容： 张三 男 1341111122222 **（因为其会把所有读到的文本都写入数组）**

\*/

**//获取当前读到的内容**

**String content = new String(ch,start,length);**

System.out.println("MyDefaultHandler1.characters()->"+content);

}

/\*\*

\* **遇到每个结束标签时触发此方法**

\* @param **qName: 当前读到的结束标签名称**

\*/

@Override

public void **endElement(String uri, String localName, String qName)** throws SAXException {

System.out.println("MyDefaultHandler1.endElement()->"+qName);

}

/\*\*

\* **遇到xml文档的结尾触发此方法**

\*/

@Override

public void **endDocument()** throws SAXException {

System.out.println("MyDefaultHandler1.endDocument()");

}

}

### 存储contact.xml文件信息

public class MyDefaultHandler2 extends DefaultHandler{

//存储contact.xml文件信息

//当contact.xml读取完毕之后，这个变量就有了所有xml文件信息

private StringBuffer sb = new StringBuffer();

public String getContent(){

return sb.toString();

}

//开始标签

/\*\*

\* qName:开始标签的名称

\* attributes： 属性列表

\*/

@Override

public void startElement(String uri, String localName, String qName,

Attributes attributes) throws SAXException {

sb.append("<"+qName);

//属性列表

if(attributes!=null){

//遍历属性

for(int i=0;i<attributes.getLength();i++){

String name = attributes.getQName(i);//属性名称

String value = attributes.getValue(i);//属性值

sb.append(" "+name+"=\""+value+"\"");

}

}

sb.append(">");

}

//文本内容

@Override

public void characters(char[] ch, int start, int length)

throws SAXException {

//当前文本内容

String content = new String(ch,start,length);

sb.append(content);

}

//结束标签

//qName: 结束标签名称

@Override

public void endElement(String uri, String localName, String qName)

throws SAXException {

sb.append("</"+qName+">");

}

}

### 存储所有Contact对象信息

public class MyDefaultHandler3 extends DefaultHandler{

//用于存储所有Contact对象信息

private List<Contact> conList = new ArrayList<Contact>();

//用于存储一个contact标签中的信息

private Contact contact = null;

//返回封装好的List对象

public List<Contact> getList(){

return conList;

}

//用于记录当前标签是哪个

private String curTag;

/\*\*

\* 思路：

\* 1）创建一个新的Contact对象，用于封装contact标签信息

\* 2) 把当前读到的这个contact标签的信息封装到Contact对象中

\* 3）把封装好的Contact对象放入List中

\*/

//开始标签

@Override

public void startElement(String uri, String localName, String qName,

Attributes attributes) throws SAXException {

curTag = qName;

//1)在读到contact的开始标签时创建Contact对象

if(qName.equals("contact")){

//创建Contact对象

contact = new Contact();

//封装id属性

String id = attributes.getValue("id");

contact.setId(id);

}

}

//文本内容(注意：包含换行和空格)

@Override

public void characters(char[] ch, int start, int length)

throws SAXException {

//2)把contact子标签中的文本内容封装到Contact对象中

String content = new String(ch,start,length);

//如果当前标签是name

if("name".equals(curTag)){

contact.setName(content);

}

//如果当前标签是phone

if("phone".equals(curTag)){

contact.setPhone(content);

}

//如果当前标签是email

if("email".equals(curTag)){

contact.setEmail(content);

}

//如果当前标签是address

if("address".equals(curTag)){

contact.setAddress(content);

}

//如果当前标签是gender

if("gender".equals(curTag)){

contact.setGender(content);

}

}

//结束标签

@Override

public void endElement(String uri, String localName, String qName)

throws SAXException {

//把curTag置空

curTag = null;

//3)在读到contact结束标签时把Contact对象放入List中

if(qName.equals("contact")){

conList.add(contact);

}

}

}

# 总结：

1．xml语法：

标签、属性、文档声明、处理指令

**2．xml解析（运用）**

**Dom解析：原理…（面向对象）**

**2.1 dom解析的工具：dom4j工具**

**2.2 对xml文件增删改查 （方法）**

**2.3 xpath技术： 快速查询xml文件的节点（标签，属性，文本）**

**SAX解析：原理…（基于事件的）**

**2.1 DefaultHandler类**

**startElement(): 开始标签**

**characters（）:文本内容**

**endElement(): 结束标签**

3．xml约束（看懂）

dtd约束：简单

schema约束： 看得懂名称空间，如何找到scheme约束文件？

**xml约束语法查询w3c文档**