1. html
2. 简介：
3. 超文本标记语言
4. Hyper Text Markup Language
5. 网页语言
6. 超文本：超出文本的范畴，使用html可以轻松实现
7. 标记：html所有操作都通过标记实现 <标签>
   * 1. html直接通过浏览器 直接运行
8. html的规范
   1. 一个开始和结束的标签
   2. html包含两部分内容
      1. <head>显示信息</head>
      2. <body>显示页面上的内容</body>
   3. 不区分大小写
   4. 有的标签 没有结束标签
      1. <br/>换行
      2. <hr/>一条线
9. html操作思想
   1. 网页中有许多不同的数据 需要不同的显示效果
10. html常用标签
    1. <font></font>
    2. 属性 size：文字的大小1—7
       * 1. color:文字颜色 两种方式
            1. 英文单词
            2. 试用rgb表示 #加16进制
    3. 注释标签
       * 1. <!-- --!>
    4. 标题 标签<h1></h1> 里面的数字是变的 数字越大标题越小 标题标签会自动换行
    5. 水平线标签 <hr/> size粗细 color颜色
    6. 特殊符号转译 &lt; --- <
       * + 1. &gt; ---- >
         1. 空格也需要转译 &nbsp; ----" " 一个空格
11. 列表标签
    1. <dl></dl> 无序列表范围
       1. 在dl里面 <dt></dt>上层内容
          * 1. <dd></dd>下层内容
    2. <ol></ol> 有序列表范围
       1. ol属性 type:设置排序方式 1，数字 a,英文 i，罗马字
       2. 在ol里面 <li></li>
    3. 想要加特殊符号 使用<ul></ul>标签 默认为原点·
       1. 更改 type 改变默认的 特殊符号
       2. 在ul里面 <li></li>
12. 图像标签 重点
    1. <img src="地址"></img>
    2. 属性： width height 宽高属性 alt="鼠标停留在图片上 会显示文字 用的少 兼容性差“
13. 路径的介绍
    1. 分类：两类1，绝对路径 C：\Use\io\lxa.jpg http://www.baidu.com
       * + 1. 2，相对路径 一个文件相对于另一个文件的位置 img\iii.jpg
           2. ../ 表示html的上层路径
14. 超链接 <href target="\_blank"></href> 跳转到空白的页面 “\_self”在现在的页面打开
    * + 1. target = “#” 什么也不打开
15. 定位资源
    1. 如果想要定位资源 ： 定义一个位置
       1. <a name="top">顶部</a>
       2. 回到顶部 <a href="#top">回到顶部</a>
16. <pre></pre> 可以进行换行的 pre 按照里面的内容进行换行
17. 表格标签
    1. <table border="设置线 1 大小" bordercolor="线的颜色" cellspacing="线之间的空隙大小 设置为0 没有空隙" width="" height="">
       1. <tr可以设置显示方式 align="center"字体居中 > <td></td> <td></td></tr> align——排列
       2. <tr></tr>
    2. </table>
    3. tr表示一行 td 表示单元格
       1. th也可以表示单元格 th可以实现 居中 加粗
       2. 表格的标题<caption></caption> 设置标题
       3. 合并单元格
       4. rowspan ="3" 跨行合并3个单元格
       5. colspan
18. 表单标签\*\*\*\*
    1. 可以提交数据到服务器 这个过程可以使用表单标签
    2. \*\*<form action="提交到哪里 写出路径 01-hello.html 提交到这个页面里" method="get或 post”/form> ： 定义一个表单的范围
       1. 输入项：可以输入内容
          1. 大部分输入项 使用<input type="输入项的类型"/>
          2. \*\* 普通输入项：<input type="text"/>
             1. eg：

<form>

手机号码:<input type="text"/>

</form>

* + - 1. \*\* 密码输入项：<input type="password"/>
      2. \*\* 单选输入项 必须要有value <input type="radio" name="sex" cheaked="checked 这个属性可以默认选中 多选也是一样"/>女 <input type="radio" name="sex"/>男 需要属性 name name属性必须一样才能单选
      3. \*\* 复选输入项 必须要有value <input type="checkbox" name="love"/>羽毛球 <input type="checkbox" name="love"/> 网球
      4. \*\* 文件输入项 <input type="file"/> 可以插入本地文件
      5. \*\* 下拉输入项(不在input标签里） <select name="birth">

<option value="1998" selected="selected 实现默认选中'>1998<option/>

<option value="1997">1997<option/>

<option value="1996">1996<option/>

</select>

* + - 1. \*\* 文本域 <textarea cols="有机行" rows="有几列"></textarea>
      2. \*\* 隐藏项 <input type="hidden"/> 具体用途之后讲 可以写一些属性传递到servlet
      3. \*\* 重置按钮 <input type="reset" value="重置按钮" 回到初始 />
      4. \*\* 普通按钮 <input type="button" value="普通按钮"
      5. 通过图片进行上传 \*\*
         1. <input type="image" src="图片路径"/>
      6. \*\* 提交按钮 <input type="submit"/> 注册 提交之后网页 出现？后的值被提交到服务器上
    1. \*\*\*\*\* 每个标签里都要有 name 属性 \*\*\*\*
       1. \*\* get 和 post 的区别： get请求地址栏会携带提交数据 post不会携带提交数据

get 请求的安全级别低 get 请求数据有大小的限制 post没有带下的限制

还有一个 e

* 1. 使用表单写注册界面：
     1. 创建一个html页面 ：
  2. <h2>注册</h2>
  3. <form action="要提交的页面">
     1. <table width="100%" border="1" >
        1. <TR>
        2. <td>注册邮箱</td>
        3. <td><input type="text" name="mail"/></td>
        4. </TR>
        5. <tr>你可以使用<a href="#">手机号注册</a></tr>
        6. <tr><input type="password" name="pw"/></tr>
        7. <tr>姓名：<input type="text" name="name"/></tr>
        8. <tr><input type="radio' value="sex" checked="cheaked" /></tr>
        9. <tr><select>
           1. <option value="1998">1998</option>
           2. <option value="2000' selected="selected" ></option>
           3. </select>
        10. </tr>
            1. <td><img src="验证码 "/><td><input type="text"/></td>
        11. <tr>
        12. <tr><td>&nbsp;</td><td><input type="image" src="地址"/></tr>
     2. </table>
  4. </form>
  5. html中其他常用标签的使用
     1. <b>留心昂</b> 加粗
     2. <u>尤文图斯</u> 下划线
     3. <s>yiyiyi</s> 删除线
     4. <i></i> 斜体
  6. <pre>原样输出 文本里面有换行 也会还行
  7. <sub> 下标
  8. <sup> 上标
  9. \*\*\*\*\*\*\*\*\*\*\*\*
  10. <div> 可以自动换行
  11. <span> 都会在一行显示
  12. <p> 段落标签

1. html 头标签的使用 ：
   1. 在head里的标签 叫头标签
      1. title: 表示在标签显示的
      2. meta： 可以设置页面的相关的内容
      3. <meta name="keywords" contenet="liuxinang,asasooo 在早期的搜索引擎会找keyword关键字搜索内容 现在不用了" >
2. 框架标签 ： 用的时候要 删除 <body> 直接写在head外面
   1. <frameset> rows 按照行进行划分 <frameset rows="80,\*"> \*表示剩下的部分都是
      1. eg: <frameset rows="100,\*">
         * 1. <frame name="" src="a.html"/>
           2. <frame name="" sec="b.html"/>
         1. </frameset>
   2. 有点类似j2se的frame插入panel label 等等。。。
   3. 现在基本没人用这个 都是用div
      1. 块级元素
      2. 在浏览器中显示：
      3. 块级元素自占一行
      4. 上面第二个元素并没有和第一个元素在同一行显示，而是自动换行，每个块级元素都自占一行
3. html中的中文乱码问题：utf-8 和 GB2312 的区别 系统默认一定不会有中文乱码
   1. DAY1总结 ：
      1. html操作思想 使用标签把要操作的数据 修改标签的属性值 进行修改
      2. 常用标签 font属性 size 1—7 color 英文 或 16进制
         * 1. 标题标签 h1....h6 越来越小 会自动换行
           2. 注释 <!-- -->
           3. 列表标签 <dl><dt上层 ></dt><dd下层></dd></dl>
           4. 有序<ol type="1,a,i"><li></li><ol>
           5. 无序<ul><li></li></ul>
           6. 图像标签 <img src="" width height alt="鼠标放上去显示的文字"/> alt兼容差
           7. 路径 相对路径 在同一目录直接写 下一集目录 ？？？\。。。 上一层目录 ../
           8. 超链接标签 <a href="路径" target="\_self现在页面打开 \_blank新窗口打开 默认\_self"/>
           9. 表格标签 \*\*\*\*\*
           10. <table> <tr><td></td>或<th></th>加粗居中单元格 </tr> </table>
           11. 表单标签 最重要\*\*\* 注册登录可以使用表单标签
           12. <from></from>属性: 表示表单范围

action : 提交到的地址

method : 提交方式 post get.

enctype:上传时用到

multipart/form-data：

1、当我们上传的含有非文本内容，即含有文件（txt、MP3等）的时候，需要将form的enctype设置为multipart/form-

data。

将表单中的数据变成二进制数据进行上传，所以这时候这时如果用request是无法直接获取到相应表单的值的

* + - * 1. 输入项 :

普通输入项 <input type="text"/.

密码 type="password"

单选 ;radio

复选 ： checkbox

下拉框 <select name=""><option value=""></opton></select>

文本域 :<ttxtarea cols="用行和列定义大小" rows="" name=""></textarea>

文件 file

提交按钮 ; submit

重置 : reset

使用图片提交 ： <input type="image" src=""/>

普通按钮 button

隐藏项 : hidden

* 1. div自动换行 span 在同一行显示
  2. 框架标签 不是重点

css与js js是重点

css 层叠样式表 层叠：强调优先级 样式表：有很多属性值

是页面的显示效果很好 分离属性 使代码很好维护‘

css 和 html 的结合方式 有4种：

1，每个html上都有style属性 把css与html结合在一起

<div style="background-color:red;color:字体颜色 之间用,隔开">帅帅啊啊</div>

2,使用 <style>标签 写在<head>里

<head> <style type="text/css'>

div{ background-color:red;} 要用大括号 指明对象

</style></head>

<body><div>asdhAHD</div> </body>

3,在style标签里 使用语句 这样可以直接修改css 实现分离 可以直接改css文件 实现大面积更改

@import url(css文件路径） 后缀名就是 .css 在css文件里直接写 代码 div{background-color:red;}

在html中进行引用 ：写在head里面

<style type="text/css">

@import ur(XXX.css);

</style>

4,\*\*\* 使用头标签link 来进行实现 也要写在head里面

<link rel="stylesheet(样式表)" type="text/css" href="css文件路径"/>

第3中结合方式 在一些浏览器下不兼容 很少用 一般用地4种结合方式

css的优先级 ：

单独在标签内定义的 优先级最高

由上到下 后加载的优先级最高

css的选择器 ：

基本选择器：

标签选择器：使用标签名作为选择器的名称

class选择器：每个html标签都有一个属性

<head><style type="text/css'>

div.haha{ 指定元素进行更改}

.haha {只要class为haha的 都更改样式 }

</style></head>

<div class="haha">asd</div>

id选择器 ：

每个html都有id属性

<head><style type="text/css">

div#haha{ 这样设置 }

#haha { 只要id为haha的 都更改样式 }

</style></head>

<div id="haha">asda</div>

基本选择器优先级 ： 后加载的优先级高

style选择区 》id选择器 》 class选择器 》标签选择器

css扩展选择器 ：

关联选择器：<div><p>asdasd</p></div> <p>dasd</p> 有两个p标签 指定divp 变化

<style type="text/css"> div p { 之间用空格隔开 &nbsp; }

组合选择器 ： <div>213a</div> <p>SDaa</p> 把div和p设置成同一种样式

<style type="text/css" > div,p{用逗号，隔开}

伪元素选择器 ： css提供了定义好的样式 可以直接使用

比如使用超链接{ ：link 未访问前的样式原始状态 ：hover 鼠标放上去的状态 ：active点击　：ｖｉｓｉｔｅ鼠标之后的状态 }

　　　<style type="text/css"> a:link{ 更改未访问前的样式原始状态 的样式}

记忆的方法： lv ha

css 盒子模型 ： 有三个重要概念 { 边框 内边距 外边距 }

边框 border 统一设置 { 上 border-top 下 bprder-bottom 左右同理}

内边距（文字到边界的距离） padding:统一设置 四面的距离 padding-left:设置左边的页边距

外边距 margin 可以同意设置 -left 等等。。。

css的布局的漂浮：float（用的少 因为可能不兼容） ：有两个属性值 left right 相反的

left 文本流向对象的右边 right 文本流向左边

css布局的定位：

postion : 属性值 : static ，absolute ， relative

absolute : 将文件从文件流中拖出，绝对定位 right top bottom

relative : 不会把对象从文档流中拖出

eg: div{

postion : absolute;

图文混排案例（新闻推送样式 图片左边 文字右边

把文字图片放进div区域

<style type="text/css" >

#imgtext{width 350px,;height:300px;border:2px dashed orange;}

#text{float:right; color:green;}

<div id="imgtext">

<div><img src=""/></div>

<div id="text">大时代</div>

</div>

让字写在图片上（图像签名）

<style type=="text/css">#text{postion:absolute; 定位字体 进行绝对定位 }

<div><img id="img" src="" width="250" height="200"/></div>

<div id="text">hello </div>

内容总结 ：

1.css和html的4中结合方式 \*\*\*\*\* { 使用style属性 直接写 用：；隔开}

使用<style 标签

使用<style 进行 @import url(外部文件 用的少

使用 <link rel="stylesheet" type="text/css" href="地址 " 使用头标签

2.css优先级 一般后加载的优先级高

3.css的基本选择器 \*\*\*\*\*\* 标签选择器 class选择器。加名称 id选择器#加名称

优先级: style > id > class > 标签

4.css的扩展选择器

关联选择器 div 李嵌套p div p {}

组合选择器 div,p{}

伪元素选择器 超链接状态

原始 ：link

悬停 : hover

点击 : active

点击之后 : visited

5.盒子模型

边框 border:2px solid red ;

内边距 pading :20px

外边距 margin:20px

上下左右 ： top bottom right left

对数据进行操作 需要把数据放 一个区域里 (一般用div

6.布局的漂浮

float : left right 这里的定位是相反的！！

7.布局的定位

postion:absolute ；绝对定位 从文档流拖出

relative ：不会拖出

图文混排—— 设置float

图像签名—— 把文字显示在图片上 吧文字的div设置为 postion：absolute。

javascript

简介：基于对象和事件驱动的语言 （应用于客户端

——基于对象：提供了对象 可以直接使用

——事件驱动：html做的网站都是静态的效果 js做的都是动态的效果

——客户端：指浏览器 或者 像 apicloud

特点 ：

交互性 ：信息的动态交互

安全性 ：js不能直接访问本地磁盘的文件

跨平台性 ：浏览器都可以运行

js只需要解析就可以运行

js的组成 ：三部分

ECMAScript：ECMA 是欧洲计算机协会 制定语法

BOM : broswer object model -- 浏览器对象模型，浏览器里的空间等等

DOM : document object model -- 文件对象模型，网页里的内容可以对内容进行操作

js 和 html的结合方式 （有两种

1 使用一个标签 <script type="text/javascript">

alert("alert可以弹出一个框");

</script>

2 使用script标签 引入一个外部的js文件

\*\*\*创建一个js文件 写js代码

<script type="text/javascript" src=" 引入js文件路径"> 若在此处写入js代码 不会执行 </script>

原始类型和声明变量

Java的基本数据类型 byte short int boolean long double float

定义变量 都是用关键字 var

Js 的原始类型 5个——string number boolean null undifined

string：字符串 eg{var str = "abc";

number：数字类型 var m = 123;

boolean：true false var flag=false;

null： var date = new Date(); 表示获取对象的引用 null表示对象引用为空 所有对象的应用也是object

undifined: 定义了一个变量 没有赋值

typeof（）; 可以查看当前变量的类型

js里面的语句 ：

if(a==5){ alert("5");}

switch(a){

case 5 : break;

case 6: break;

default:....} 跟java一样、

循环语句 while do while for 都有

js的运算符

js里没有小数 123/100 = 1.23

var str ="1233"

alert(str+1); ----"12331" 跟java相同

alert(str-1); ----"1232" 直接进行减法 若str不是数字进行相减会提示NaN 报错

boolean

var flag = true：等于1

如果设置成true值为1 如果设置false值为0

var a = "5";

if(a==5){alert("5")}else{alert("other")} 输出5

if(a===5){alert("5")}else{alert("other")} 输出other

\*\*\* 两个等号比较值 三个等号比较值和类型

引入知识： 直接向页面输出语句（直接显示在页面上

document.write("直接输出在页面上");

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

document.write("<hr/>"); 输出变量固定值和html代码

js的数组 ( 可以存放不同的数据类型

定义方式 3种

第一种 ： var arr = [1,2,3,4，“as”，true];

第二种 ： 使用内置对象Array对象

var arr1 = new Array(5); 有5个元素

arr1[0] = "1";

第三种 ： 使用内置对象 Array

var arr2 = new Array(3,4,"d"); 传入多个表示直接定义

有length属性

js里的定义函数 有三种方式

1， 使用到一个关键字 function

function 方法名 （参数列表）{

方法体；

返回值 可有可无（跟Java不同）

}

2.匿名函数

function （参数列表 ）{

方法体；返回值；}

var add3 = function(){}

可以这样调用匿名函数

3.使用js里面内置的对象 function （用的少)

new function("参数列表","方法体返回值”)；

JS的全局变量和局部变量

全局变量 ： 在scropt标签里的定义一个变量 这个变量在页面js部分可以使用

局部变量 ： 只能在一部分使用 在方法内部使用

script标签位置 （可以放在任何的位置)

一般放在</body>的后面 一定会执行

js没有重载\*\*\* 但可以用其他方式模拟重载

JavaScript基础 & DOM

js的String对象

创建String对象 var str = "abc";

方法和属性

<script type="text/javascript">

方法：

与html相关的方法

与java相似的方法

与html相关的方法

bold() 加粗 str.bold();

fontcolor() 字体颜色 str.fontcolor();

fontsize(1-7) 设置字体大小

link(url) 将字符串显示为超链接{ str.link("hellp.htm");

sup sub 上下标

document.write(s1.sub());

与java相似的方法

concat 方法（合并字符串： var str1 = "ac"; var str2 = "dd";

document.write(str1.concat(str2));

charAt 返回制定位置的字符串 ：

var str = "acbf";

document.write(str.charAt(0)); 输出a 若字符的位置不存在 则返回空字符

indexOf 返回指定字符所在的位置 若没有则返回-1

split 切分字符 根据制定字符切分成数组 { var str = "a-f-g"; var arr = str.split("-"); document.write(arr.length);

replace 替换字符串 （ 要传递第个参数 第一个替换第二个　） { var str = "acb"; str2.replace("a","g"); //输出 gcb

substr() 和 substring() 区别 ：

var str = "abcdefghuiop";

document.write(str.substr(5,3)); 输出 fgh

document.write(str.substring(5,3)); 输出de

substr 从第几位开始向后截取几位

substring 从第几位开始截取到第几位 不包含第二位上的字符

js的数组 Array对象

创建数组 3种

var arr = [1,2,3];

var arr = new Array(3);

var arr = new Array(1,"a",2);

——————

var arr = []; 创建一个空的数组

var arr = new Array(); 创建一个空的数组

concat方法 连接数组 arr1.concat(arr2);

join() 根据指定字符分割数组 可以将默认的 , 分割符替换{

var arr = new Array(3);

arr[0] = "a";arr[1] = "b";arr[2] = "c";

document.write(arr.join("-"));

push（） 向数组末尾添加元素 \*\*返回数组新的长度

如果传入数组的话 先将要传入的数组转化为字符串 在将字符串传入

var arr = ["a"，"b","c","tom"];

document.write(arr.push("lxa")); //应该输出长度 为什么是数组？？？

//输出 a,b,c,tom,lxa

pop() 会返回数组最后的元素 并将数组最后的元素删除

var arr = ['b','tom',"ann"];

document.write(arr.pop()); 输出：ann

document.write(arr); 输出：b,tom

reverse() 元素倒叙排列

var arr = ["lxa","zjy","hhh"];

document.write(arr.reverse()); 输出： hhh,zjy,lxa

shuffle() 随机打乱

js的Date对象 时间对象

java里：Date date = new Date(); //格式化 转化成习惯的时间 —— toLocaleString()

js里获取当前的时间

var date = new Date(); document.write(date); 得出的格式不习惯

转化为习惯的格式：

toLocaleString() : document.write(date.toLocaleString()); // 输出 ：2018年2月13日 13：13：22

得到具体时间的方法

获取年的方法 ： getFullYear（）; 返回四位数字的年份

获取月的方法 : getMonth() : \*\*只能的到0-11 所以要＋1

获取星期的方法 ：getDay() : 星期，返回的是 0-6 \*\*星期日返回的是-0 星期一到星期六返回1-6

获取当前的日 ： getDate() : 得到天 1-31

获取当前的小时 （以下几个方法最后都有s） ： getHours（） ：

获取当前的分钟 ： getMinutes()

获取当前秒： getSeconds()

获取毫秒 ： getTime() --- 返回1970。1.1 至今的毫秒数

应用场景 ： 可以用于来处理缓存的效果 不让有缓存 { http://www.baidu.com?毫秒数 用毫秒数作为参数

Math对象 （进行数学的运算

这个对象都是静态方法 直接使用Math.方法();

abs()绝对值

ceil()进行向上舍入

floor() 向下舍入

round() 四舍五入 （java里是rint 方法 ）

random（）随机数 （ 伪随机数 返回0-1之间的随机数

max（a,b） 返回大的 min 返回小的

pow （a,b) a的b次幂

js的全局函数 不属于任何一个对象

直接写名称就可以使用

eval() 直接写就行 ： 执行js代码 （可以使用这个方法直接来执行）： eval("laert("111")"); 直接弹出111

encodeURI()decodeURI() 对字符进行编码 解码 中文乱码需要解码

var ste = "中文";

var encode = encodeURI(str) 变成乱码; var decode = decodeURI(encode) 解码乱码 ;

isNaN() 判断当前的字符是不是数字

var str = "aaa" isNaN(str) 返回true

\*\*\* 是数字返回false 不是数字返回true

parseInt() 类型转换 字符转化为number JS没有int

parseFloat() 意思差不多

js里没有float类型 ？ 方法啥意思？

js函数的重载 ： \*\*\* 不存在重载 但是可以通过其他方式 模拟重载 在每个函数里都有数组arguments

面试题目 ： 不存在重载 但可以通过 数组arguments模拟重载

eg : function add () { if(arguments.length == 2) { return .... }

js的bom对象 ： 浏览器对象模型 Browser

对象 ：

navigator （英文翻译 领航员 ：

navigator.appName 显示浏览器名称 （ 没有加“（）”）

screen ：屏幕的信息 screen.width screen.height 像素高 宽

location ： 请求的url地址

href属性 ： 获取到请求的url地址

设置url地址 ：页面上安置一个按钮 按钮可以打开新网页

需要一个鼠标点击时间 onclick="js方法（）";

<input type="button" value="tiaozhuan" onclick="href1();"/>

<script type="javascript/css" >

function href1 (){location.href = "hello.html";}

</script>

history ： 历史对象 请求的url的历史记录

eg: 创建a.html 超链接到 b.html

创建 b.html 超链接到 c.html

创建<input type="button" value="next" onclick="back();">

<script type="javascript/css" > function back(){history.back(); //回到上一个页面}

function next(){ history.forward(); //到下一个页面

创建 c.html

window \*\*\* 窗口对象 顶层对象（所有bom都是在window里进行操作

方法 ：

alert() 全称是 window.alert() 弹出一个框 显示内容 可以简写为alert

confirm() 确认框 ： <script type="javascript/css" >var flag = window.confirm("这是确认框"); </script>

会返回 确定 和 取消 两个值

prompt() 输入对话框 ： 很少用 有两个参数 在框里面显示的内容 和 输入框的默认值

open() 打开一个新的窗口 ：

open("打开新窗口的url地址"，"","窗口特征 高 宽 width=100，height = 200")

close() 关闭窗口

做定时器：

setInterval("js代码"，"毫秒数") 在制定周期 调用js代码 一定要设置id

window.setInterval("alert("111")","3000") 每3秒 alert一次111

setTimeout("js代码"，"毫秒数") 在指定的毫秒数 后执行 一会执行一次 一定要设置id

window.setTimeout("alert("111")","3000") 3秒后 alert 111

clearInterval(setInterval 的 id) ：清楚 setInterval

clearTimeout(setTimeout 的 id) ： 清除 setTimeout

js的dom 文档对象模型（都要封装为对象

document object model

文档 ： 超文本标记文档 html xml

对象 ： 提供了属性和方法

模型 ： 使用属性和方法操作超文本标记文档

\*\*\* 可以使用js里的方法提供的对象 对标记行文档进行操作

想要对标记行文档进行操作 首先需要 对标记型文档里面的所有内容封装成对象

js的dom对象：学dom里面的属性方法

需要把html里的标签 属性 文本 内容都封装成对象

要想对标记型文档进行操作 解析标记型文档

分析如何使用dom解析html ：

会在内存中分配一个树形结构 根据html层级结构 在内存中分配一个树形结构 把html中的每一个部分都封装为对象 （只能有一个根节点 在根节点下面多个子节点 如果下面没有子节点 就叫做叶子节点



DHTML：是很多技术的简称

Html 封装数据

Css 使用属性和属性值设置样式

Dom 操作html文档 ( 标记型文档

Javascript 专门指js语法语句

document对象 整个文档

每个载入浏览器的html文件 都将成为一个document文件

write方法：

想页面输出变量值

向页面输出html代码

\*\*\*\*\* 返回的是数组类型 需要遍历 就算只有一个元素依然返回数组Object

getElementById() 方法： 不返回数组 直接返回标签 准确！\*\*\*

通过id得到元素（标签

使用getElementById得到input标签

var input1 = document.getElementById("nameid");

//得到input里面的value值

alert(input1.name);

input1.value = "cccc";

//向input里设置一个值value

getElementsByName() 方法 一般会返回一个数组Object

<input type="text" name="n1" value="aaa"/>

<input type="text" name="n1" value="bbb"/>

<input type="text" name="n1" value="ccc"/>

<script type="text/javascript">

var inputs = document.getElenemtsByName("n1"); // 返回一个数组

</script type="text/javascript">

getElementsByTagName() 方法 ： 通过标签类型名称 返回数组object

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

input1 是一个object数组

\*\*\* 只有一个标签 这个标签只能使用name获取 使用getElementsByName返回的是一个数组 但是只有一个元素 这时不需要遍历 可以直接通过数组获取内容

案例 ： window弹窗案例 ( 会有一个问题： 这个过程访问了本地文件 chrome不能用... 实际开发中不会反问本地文件 不会出现这个问题）

实现过程 ：

创建一个页面 有输入项和按钮 按钮上事件：弹出一个新窗口（表格 每一个有一个编号和姓名，按下按钮后再上一个页面输入内容）

需要夸页面操作 需要方法opener

opener（）方法 ： 表示可以得到创建这个窗口的上一个窗口

var pwin = window.opener; //返回上一个页面 \*\*\*



在末尾添加节点：

<ul>

<li>123</li><li>22</li><li>33</li>

</ul>

若要添加节点 1 创建li标签 2，创建文本 3.把本文加入 li 4，把li加入ul

具体方法 ：

function add(){

var ul1 = document.getElementById("");(对它操作先要获取它

//接下来创建标签

var li1 = document.createElement("直接写标签的名称 li");

//接下来创建文本

var text = document.createTextNode("这里是文本");

//吧文本添加到li下面

li1.appendChild(text); //Node 英文翻译——节点

}

1，获取ul标签 getElementById

2，创建li标签 createElement

3，创建文本 createTextNode

4，把文本添加到li下 appendChild

Element对象：

要操作element对象 要先获取到element

使用document 相应的方法

获取属性的值： 先获取标签

var input1 = document.getElementById("");

获取value ：

1，input1.value

2,input1.getAttribute 也可以获取 更好用

3，input1.setAttribute("要设置的属性的名称","要设置的值")

attribute 英文翻译——属性

——移除属性 removeAttribute（"要移除的属性名称"）；\*\*此方法removeAttribute不能移除value

若想要获取到ul下面所有的子标签

<script type="text/javascript">

//获取ul标签

var ul1 = document.getElementById("");

//获取下面的子标签

((( 不要用！！！！ —— var array = ul1.childNodes; ( 这个方法返回的是 数组 \*\*\* 浏览器的兼容性差

var lis = ul1.getElementsByTagName("要查找的标签");

\*\*\* 这是获得标签下子标签的 唯一 有效方法！！！

</script>

Node对象属性：

nodeName

nodeType

nodeValue

这三个方法 在不同环境下的应用：

标签节点对应的值：

nodeType；1 （最重要 要记住\*\*\*

nodeName：大写的标签名称 比如SPAN

nodeValue：null

属性节点：

nodeType：2

nodeName：属性的名称

nodeValue：属性值

文本节点：

nodeType：3

nodeName：#text

nodeValue：文本内容

使用dom解析html时，需要html里面的标签 属性和文本都封装成了对象

获取文本的方法： firstChild 后面没有（）！！！

Node对象的属性：

父节点 子节点 同辈节点:关系显而易见

父节点的属性：parentNode 得到父节点

子节点：childNodes：得到所有子节点，兼容性差 \*\*\*不要使用

firstChild 获取第一个子节点

lastChild 获得最后一个节点

其他属性：nextSibling：返回指定节点的下一个兄弟节点

previousSibling ： 返回一个制定节点的上一个兄弟节点

Sibling 翻译——兄弟姐妹

操作DOM树：

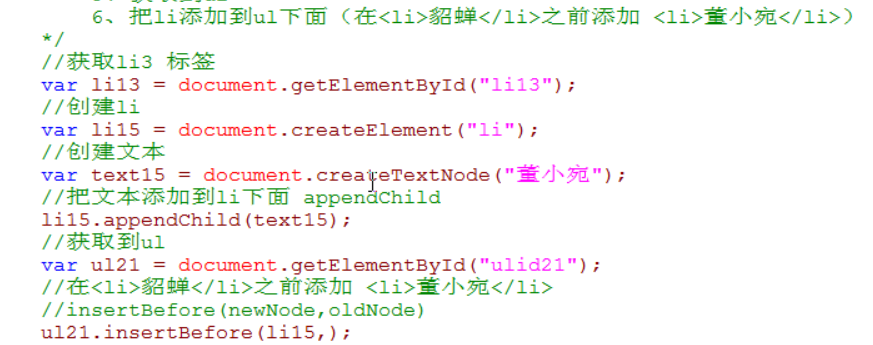
appendChild 方法 append翻译——追加

添加子节点到末尾 特点：类似于剪切\*\*\*

insertBefore（newNode,oldNode) 在某个节点之前插入新的节点 此方法必须用在父标签上

两个参数：\*要插入的节点

\*再睡之前插入

插入一个节点 节点不存在， 创建 1，标签 2，文本 3，把文本添加到标签下面

没有insertAfter方法

\*\*\* removeChild 删除节点，用通过父节点进行删除，removeChild要在父节点上进行使用

eg{

<script type="text/javascript"/>

function remove(){ /\*1，获取li标签 2，获取父节点标签 3，执行删除

var li2 = document.getElementById("");

var ul2 = li2.parentNode 或 var li2 = document.getElementById();

ul2.removeChild("li2");

\*\*\* replaceChild（newNode,oldNode）替换方法 也是通过父标签进行

不能替换自己 ， 要通过父节点

第一个参数 ——新节点

第二个参数——旧节点

eg{

1，获取li

2，创建新标签

3，创建文本

4，把文本添加到li下

5，进行替换

<script type="text/javascript"/>

function replace(){

var liOld = document.getElementById();

var liNew = document.createElement("li");

var text = document.createTextNode("lxa");

liNew.appendChild(text);

var parentUl = document.getElementById("ul") 或 liOld.parentNode;

parentUl.replaceChild(liNew,liOld);

}

\*\*\* cloneNode（boolean） 复制节点，里面参数 boolean类型 表示是否复制节点

eg{

1,获取ul

2，执行复制cloneNode true

3，把复制的内容放到div里

获取div

appendChild方法

var ul = document.getElementById（）;

var ulcopy = ul.cloneNode(true);

var div111 = document.getElementById();

div111.appendChild(ulcopy);

}

\*\*\* insertBefore : 在某个节点之前插入

appendChild ： 在末尾添加，剪切

removeChild ： 通过父节点删除

replaceChild ： 替换通过父节点

cloneNode ： 复制节点

innerHTML属性：

不是DOM组成部分，大部分浏览器都支持

1，获取文本内容

2，向标签里面设置内容

<script type="text/javascript"/>

var span1 = document.getElementById();

span1.innerHTML; —— 获取标签里的文本内容

div22 空的div

div22.innerHTML = "<h>lxalxa</h>"; —— 直接写代码

/\* 添加表格

var tab = "<table><tr><td>asd</td></tr></table>";

div22.innerHTML = tab ;

动态显示时间：设置计时器 用innerHTML写入时间

function time(){

var date = new Date();

var d = date.toLocalString(); 此处为字符串类型

var div1 = document.getElementById();

div1.innerHTML = d ;

}

setInterval("time();",1000)

&&& 总结：

getAttribute

setAttribute

获取子标签 唯一 方法 getElementByTagName

nodeType

nodeName

nodeValue

appendChild 可以添加到末尾 有剪切黏贴的效果

insertBefore（新节点，旧节点） 执行添加通过父节点进行

removeChild 通过父节点

replaceChild（新节点，旧节点）

cloneNode（true）boolean表示复制子节点

innerHTML\*\*\*

可以获取文本内容，可以向标签里设置内容可以设置html代码

全选练习： 使用复选框上的checked属性

创建一个页面

复选框和按钮（按钮上有事件

四个复选框表示爱好

还有 全选 反选 撤销 按钮

<body>

<input type="checkbox" id="boxid"/>全选/全不选

<input type="checkbox" name="love"/>篮球

<input type="checkbox" name="love"/>足球

<input type="checkbox" name="love"/>羽毛球

<input type="checkbox" name="love"/>乒乓球

<input type="buttom" value="全选" onclick="all();"/>

<input type="buttom" value="全不选" onclick="no();"/>

<input type="buttom" value="反选" onclick=:other();:/>

</

<script type="text/javascript">

function all(){

var arr = document.getElementsByName("love"); 返回的是数组 Elements 有s!!

遍历数组

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

var arrNumber = arr[a];

arrNumber.checked=true ; 通过checkbox里的 checked属性进行操作

}

function on(){

var love = document.getElementsByName("love"):

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

var love = love[i];

love.checked = false ;

}

}

function other(){

var love = document.getElementsByName("love"):

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

var love = love[i];

if(love.checked==true){love.checked=false}一个等号赋值 两个等号判断

相反。。。

}

案例：下拉列表左右选择（添加到。。

<body>

<div>

<select id="select1" multiple="multiple"(\*\* 可以把内容都显示出来）style="width:100px;height:100px;">

<option>aaaa</option>

<option>abbba</option>

<option>accca</option>

</select>

<input type="buttom" value="添加到右边" onclick="add();"/>

<input type="buttom" vale= "全部添加到左边" onclick="alladd();"/>

</div>

<div style="float:left;">

<select id="select2" multiple="multiple"(\*\* 可以把内容都显示出来）style="width:100px;height:100px;">

<option>dddda</option>

</select>

</div>

</body>

<script type="text/javascript">

function add(){

var select1 = document.getElementById("select1");

var select1 = document.getElementById("select2");

var arr = select1 .getElementsByTagName("option"); 返回数组

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

var option1=arr[i];

if(option1.selected ==true){select2.appendChild(option1); \*\*\* i-- ; 这样能一直使用！！！因为appendChild方法类似于剪贴 所以不需要删除}

}

function addall(){

var s2 = document.getElementById();

var s = document.getElementById();

var ops=s.getElementByTagName();

for(var h==0;h<ops.length;h++){var op1=ops[h];s2,appendChild(op1);h--;此处同理};

}

案例： 省市联动 （选北京 有 海淀昌平。。 选上海有静安宝山。。

创建一个二维数组来存放数据，第一个是国家名称，第二个国家城市 ，

<body>

<select id="country">

<option>中国</option>

<option>美国</option>

<option>意大利</option>

</select>

<select id="city">

</select>

<script type=”text/javascript”>

Var arr = new Array(3); 此处为二维数组

ar[0]=[“中国”，”上海“，”杭州”]；

ar[1]=[“美国”，”底特律“，”纽约”]；

ar[2]= [“意大利”，”米兰“，”都灵”]；

function add( val ){ 1，遍历数组 2，获取第一个值对照 3，获取后面的值4，得到citySelect 5，加到第二个select，要创建option

var city1 = document.getElementById(“city”)

for(var I = 0 ;i<arr.length;i++){ //得到二维数组里的每一个值

var arr1 = arr[i];

var coutry = arr1[0];

if( val == coutry) {//得到后面的元素

// 先将之前存在city里的option删除掉 \*\*\*

Var city1 = document.getElemenetById(“city”);

Var arr2 = city1.getElementByTagName();’

for(var j=1;j<arr2.length;j++){ //这里为什么不是 0 ?？？

var value = arr2[j];

city1.removeChild(value);

}

For(var I = 1i<arr1.length;i++){

Var value1=arr1[i];

Var option = document.createElement(“option”);

Var text = document.createTextNode(value1);

Option.appendChild(text)

City1.appendChild(option)

}

}

案例：动态生成表格 ( tab的写法是重点

<input type="text" id="h"/>

<input type="text" id="l"/>

<input type="buttom" value="生成" onclick="add();"/>

function add(){

//得到输入的数字

生成表格 进行循环 显示到页面上-把表格的代码设置到页面上 （使用innerHTML

var h = document.getElementById()。value；

var l = document.getElementById().value;

var tab = "<table>";

循环： for(var i=1 ; i<=h;i++){

tab+="<tr>";

for(var j=1;j<=l;j++){

tab+="<td></td>";

}

tab+="</table>";

var div11 = document.getElementById();

\*\*\* div11.innerHTML = tab;

}

总结： 在末尾添加节点： 创建一个标签 createElement

创建文本额 createTextNode

添加文本之标签 appendChild——类似剪切黏贴效果

获取标签下的自标签：getElementByTagName

Node对象： nodeType nodeValue nodeName

操作DOM树：

父节点：parentNode 以下方法都是从父标签进行操作

insertBefore —— 没有nsertAfter方法

removeChild

replaceChild

innerHTML：获取文本内容

标签里设置内容

案例：动态时间 setInterval

全选连续 checked

下拉列表左右选择 multiple：让下拉框的内容全显示出来

\*\*\* 省市联动 二维数组 事件onchange（改变事件 方法 add(this.value//当前的value值)

动态生成表格 innerHTML属性 使用一个变量存放HTML代码

XML

表单的提交方式 \*\*\*

1.使用submit提交

<from >

<input type="submit"/>

</from>

2.buttom进行提交

<from默认提交到当前页面>

<input type="buttom" onclick="tijiao();"/>

</from>

function tijiao(){

//获取form

document.getElementById();

//提交form,设置action

form.action="页面地址" 提交到的页面，不写就是自己的页面

form.submit();

}

3.超链接进行提交

<a href="hello.html(提交到的页面)/usename=123(要提交的值)">点我提交</a>

如何实现动静结合？？？

4.onclick 鼠标点击事件

onchange改变内容（一般和select使用）

onfocus 得到焦点

onblur 失去焦点

<input type="text" name="text1" value="填写" onfocus="focus1()" onblur="onblur1()";

function focus1(){

var input1 = document.getElementById();

//效果类似于搜索框 一点之前里面的内容就没了

Input1.value = “空的”;

}

function onblur1(){

var input1 = document.getElementById();

input1.value = “听说Creep很好听” // 类似于网易云音乐

}

XML的简介：

什么是XML ——可扩展标记语言，使用标签进行操作 一般用于储存数据 交换数据

html里的标签都是固定的 都有特定的含义

xml里的标签可以自己定义 可以写中文

xml的主要功能是 存储数据

xml是w3c组织发布的技术

XML的应用： 3个地方

不同的系统之间进行传输数据

qq之间数据的传输——发送的内容发送到 服务器 之间的传输格式：xml 有利于程序的维护

用来表示生活中有关系的数据

经常用在配置文件

XML的语法：

xml的文档声明 \*\*\*

创建一个后缀.xml的文件

如果写xml 一定要有文档声明

文档声明必须写在第一行 <?xml version=”1.0” encoding=”utf-8”?>

属性 version 版本用1.0

encoding：编码类型 gbk utf-8

\*\*\* 文档声明：<?xml version="1.0"encoding="gbk"?>

定义元素（标签 \*\*\*

定义属性 \*\*\*

特殊字符 \*\*\*

注释 \*\*\*

CDATA区

PI指令

xml元素的标签 （可以随便定义标签

标签的定义：有开始必须有结束

包含标签主体

只能有一个跟标签，其他的标签都是下面的自标签

<aa>111</aa>

<aa>

111

</aa>

这两个是不一样的，xml中把空格换行都当成内容来解析

xml不能以\_开头

xml的标签不能包含空格 冒号 并且区分大小写

xml属性的定义：

xml也是标记文档 可以有属性

<person id="a"></person>

xml的注释：<!-- 注释内容 -->

注释不能放在第一行 第一行必须放文档声明

xml的特殊字符：（类似html

大于号 > 小于号 不能正常显示 需要进行转义

< &lt;

> &gt;

xml的CDATA区：

什么叫CDATA ： 可以解决多个字符都需要转义的操作 if(a<b && b<c && d>f)

把这些内容放到CDATA区里面 不需要转义

写法： <![CDATA[内容]]>

把特殊字符 当作文本内容 而不是标签

xml的PI指令： 通过PI指令引入css文件

<?xml-stylesheet type="text/css" href="css的路径"> 这个没什么用 一般xml都用于存储 对中文的标签不起作用

所有xml都必须有关闭标签

必须有一个根元素，必须嵌套 (类似于树结构

属性值加引号

空格回车都会当作内容来解析

<?xml version="1.0" encoding="utf-8"?> 声明文件

xml的约束：（主要通过dtd进行约束

为什么要约束：

比如现在定义了有个person的xml文件，只想要这个文件里面保存人的信息，比如name age，若添加了一个 其他的元素例如dick 可以正常显示，因为符合与法规范，但是不应该有这个元素，所以需要约束xml

xml的约束技术： dtd的约束 schema的约束 —— 两种约束

dtd约束; 是为了进行程序之间的数据交换而建立的标记语法

dtd的快速入门：

创建一个文件 后缀名： .dtd

1，看xml之中有多少个元素，有几个元素在dtd元素中写几个<!ELEMENT>

2，判断元素是简单元素 还是 复杂元素 ，若没有子元素就是简单元素 ，反之为复杂元素

如果是 复杂元素 <!ELEMENT 元素名（子元素）>

简单元素 <!ELEMENT 元素名>

3，需要在xml文件中引入dtd文件

<!DOCTYPE 根元素名称 SYSTEM"dtd路径”>

打开xml文件使用浏览器 浏览器只负责校验xml文件 不负责校验约束文件

若想校验xml的约束 需要使用工具（myeclipse工具）

dtd的引入方法：

1，引入外部dtd文件

2，使用内部dtd文件

<!DOCTYPE 根元素名称 [<！ELEMENT><><>]>

3,使用外部的dtd文件

dtd的约束：<!ELEMENT name(#PCDATA)> 表示必须是字符串类型

使用dtd定义元素

语法：EMPTY：元素为空（没有内容

ANY：任意

复杂元素：

<!ELEMENT 元素名 （子元素，子元素。。。）》

子元素只能出现一次

表示子元素出现次数：

+：表示出现一次或者多次

？：表示0次或者1次

\*：表示0次或者多次

总结：

表单提交方式 用buttom 得到form 定义方法 form.action = "提交地址" form.submint()

用href提交 <a href="网址?提交内容">

onclick 鼠标点击事件 onchange改变内容 onfocus得到焦点 onblur失去焦点

xml简介：可扩展标记性语言

应用在 数据传输 表示生活中有关系的数据 配置文件

文件声明必须写在第一行 <?xml version="1.0" encoding=""?>

xml区分大小写 空格换行将被解析 注释不能放在第一行 不能嵌套

特殊字符转义

CDATA和PI

DTD可以进行约束

创建dtd文件.dtd文件

引入dtd文件：<!DOCTYPE 根元素名 SYSTEM "dtd文件路径">

dtd文件不会被自动校验 可以使用myeclipse进行校验

dtd的引入方式：

外部引入 <!DCTYPE 根源宿命 SYSTEM "dtd路径">

内部 <!DOCTYPE 根元素名 [<!ELEMENT ...>]>

使用外部文件 <!DCTYPE 根元素 PUBLIC "DTD名称" "DTD文件的url">

dtd定义元素： （#PCDATA）表示字符串类型 EMPTY表示空

使用dtd定义属性：

语法： <!ATTLIST 元素名称 dtd的定义之间一定要有空格

属性名 属性类型 属性约束

， 逗号表示顺序

<!ELEMENT age(#PCDATA)>想在这个属性里面定义类型

<!ATTLIST age

ID1（属性名称） CDATA>

属性约束 ：#REQUIRED 表示必须出现

#IMPLIED 表示属性可有可无

#FIXED "这里是固定的属性值" 表示一个固定值

"这里是值"

<age ID="">若没有写id 就会默认为上面的值

属性类型： CDATA 表示为普通文本类型 —— 字符串

枚举 ： （aa|bb|cc） 表示只能在一定的范围内出现值 ，但只能出现其中一个

类似红绿灯效果

#REQUIRED 表示必须出现

ID：值只能是字母 下划线 开头

定义引用实体： 在DTD中定义 在xml中使用

语法： <!ENTITY 实体名称 “实体内容”>

引用方式： （在xml中使用） &实体名称

<!ENTITY 实体名称 "内容"> 在xml里引用 &实体名称; （ 这里区分大小写

引入后在xml里直接显示 内容

\*\* 注意 ： 定义实体需要写在内部 dtd中，如果写在外部dtd中 有可能用不了

xml解析的简介（用java \*\*\*\*\*\*\*\*

\* xml的标记型文档

js使用dom解析，根据html的层级结构 在内存中分配一个树形结构，把html的标签 属性 都封装对象

有 document element 属性对象 文本对象 node节点对象

## \*\*\*\* xml的解析技术

dom ， sax 两种解析技术

dom： 根据xml的层级结构 在内存中分配树形结构 ，dom解析文件如果文件过大，造成内存溢出，优点： 方便进行增删改

sax： 采用事件驱动边读边解析 读到一个 返回一个

从上到下 一行一行解析 解析到某一个对象 把对象返回

使用sax不会造成stackoverflow 实现查询

使用sax 不能实现增删改 操作

想要解析xml 需要解析器

针对dom sax 的解析器 通过api的方式提供

sun 提供了解析器 —— jaxp （权威

dom4j提供了 —— dom4j （实际用的最多

jdom提供——jdom

jaxp的api查看：

是javase的一部分（都在 javax.xml.parsers 里

有四个类：分别是针对dom和sax解析使用的类

dom : documentBuilder ；解析器类

这个类是一个抽象类 不能new

方法：

item（下标） 根据下表得到具体值

getParentNode（）得到父节点

getElementsByTagName() 得到标签

createElement() 创建标签

appendChild() 添加自标签

createTextNode 创建文本

removeChild（）从父节点删除

DocumentBuilderFactory ；解析器工厂

sax ； SAXParser；解析器类

使用jaxp实现查询的操作

创建一个xml文件

<person>

<p1>

<name>lxa</name>

</p1>

<p1>

<name>lll</name>

</p1>

</person>

sax :

SAXPerser

### 用jaxp 的dom 来解析xml：

这时候可以创建一个java的class

public class jaxpTest{

public static void main(String arg[]){

1创建解析器工厂

2根据解析器工厂创建解析器

3解析xml 返回document

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance(); instance——实例

DocumentBuilder builder = dbf.newDocumentBuilder();

Document（需要导入w3c document类） doc = builder.parse("src/person.xml"); (返回一个document

Parse ----- 解析

//随后需要得到所以的name元素

NodeList list = doc.getElementsByTagName();

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

Node name1 = list.item(i);

Item()返回节点

//得到name元素里的值

String s = name1.getTextContent(); 得到标签里的内容

System.out.print(s);

}

使用jaxp查询某一个节点

查询xml中的第一个name元素的值

使用jaxp添加节点

想在第一个p1的下面 添加一个sex标签

4,得到p1

5，创建sex标签

6，创建文本createTextNode

7，把文本创建到sex appendChild

8，放入sex

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = dbf.newDocumentBuilder();

Document doc = builder.parse(地址); \*\*\* 导入w3c的document包

得到所有的p1

NodeList nl = document.getElementsByTagName("p1");

Node p1 = list.item(0) 表示第一个p1

Element sex = document.createElement("sex");

Text t = document.createTextNode("man")

sex.appendChild(t);

p1.appendChild(sex);

\*\*\* 最后一定要有 回写 操作

\*\*\* Transformer 抽象类

TransformerFactory tf = TransformerFactory.newInstance();

Transformer t = tf.newTransformer();

t.transform(new DOMSource(document),new StreamResult(xml地址))

完成添加

jaxp修改节点

修改第一个p下面的sex内容是男

java文件：

public static void modifySex() throws Exception { modify——修改

1,创建解析器工厂 解析器

2，解析xml 返回document

3，得到sex item方法

4，修改sex里面的值

5，setTextContent 方法

6，回写xml文件

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.parse("文件路径") ; parse —— 从文本分析

Node sex = doc.getElementsByName().item(第几个);

sex.setTextContent("修改该内容");

TransformerFactory tf = TransformerFactory.newInstance();

Transformer t = tf.newTransformer();

t.transform(new DOMSource(doc),new StreamResult("修改xml的地址"));

jaxp删除节点 sex节点

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.parse("xml地址");

Node sex1 = doc.getElementsByTagName("sex").item(0);

Node parent = sex1.getParentNode();

parent.removeChilder(sex1);

TransformerFactory tf = TransformerFactory.newInstance();

Transformer t = tf.newTransformer();

t.transform(new DOMSource(doc

),new StreamResult("src//person.xml"));

jaxp遍历节点

打印出所有节点

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.parse("src//person.xml");

//编写一个方法进行遍历

list(doc);

public static void list(Node doc){

if(doc.getNodeType() == Node.ELEMENT\_NODE){ 判断是不是element类型

NodeList li = doc.getChildNodes();

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

Node node1 = list.item(i);

//继续得到node1的子节点

list(node1); 使用递归方法。自己调用自己。

//xml解析会把所有东西都解析出来

}

}

总结：

jaxp是重点

表单提交方式 buttom提交 超链接提交 事件

xml的文档声明 <?xml version="1.0" encoding="utf-8"?> 必须放在第一行

CDATA与PI

dtd约束 schema约束

#PCDATA 字符串

#REQUIRED 必须出现

xml解析 dom容易内存溢出 可以增删改

sax不能操作

获取全部元素getElementsByTagName返回的是数组

setTextContent 修改标签里的内容

getParentNode removeChild

递归遍历方法

Schema的介绍

xml的schema约束

xml有 dtd schema 两种约束 ： 一个xml中只能有一个dtd 有多个schema

dtd语法：<!ELEMENT 元素名称 约束 >

schema语法 ：符合xml的语法规范

xml的语句

一个xml可以有多个schema 用名称空间来进行区分 （ 类似于包名称

dtd里面有PCDATA 但是在xml里面可以有更多的数据类型

在schema里面可以直接定义一个整数类型

schema语法更加复杂 目前不能替代dtd

schema的快速入门：

创建schema文件 —— 写入语句

看xml中里面有多少元素

也要遵循xml规范 <?version=1.0 encoding="utf-8"?>

schema文件是一个约束文件 —— xml是被约束文件（schema本生就是xml文件’

\*\* 在schema文件里面使用提供属性的标签

既然xml schema 都是xml文件 那如何得知谁是约束文档？

<schema xmlns="" targetNamespace="" elementFormDefault="">

xmlns在一个schema文件里面 叫做xmlns属性 表示这是一个约束文件

targetNamespace:直接通过这个地址引入约束文件

首先看xml中有多少元素：

<element name="">

看简单元素和复杂元素：

如果是复杂元素：<complexType> <sequence> 子元素 </sequence(排列顺序的意思）> </complexType>

简单元素 写复杂元素

在被约束的文件里面 引入被约束文件

<person（标签名） xmlns:xsi="http://www.3c.org/2001/XMLSchema-instance"

xmlns="http://http://www.itcast.cn/12333 (这里是 schema文件的路径"

xsi:schemaLoction="约束文档的地址路径

\*\* schema文件：

xmlns="http://www.w3.org/2001/XMLSchema" 表示是约束文件

targetNameSpace ： 引入约束文件 使用 地址：url地址

elementFormDefault="qualified"

\*\* 在xml里面导入schema文件：

在根节点里面 xmls:xsi="http://www.w3.org/2001/XMLSchema-instance" 这些在被约束的根节点里

xmlns=""

schemaLoction="" 这里写的是schema的路径地址

schema 规范：

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

<schema> .... </schema>

判断是简单元素还是复杂元素

简单元素：

<element name="xml里的标签名" type="数据类型">

复杂元素：

<all> 表示只能出现一次

<choice> 表示只能 出现其中的任意一个 不能全都出现

<maxOccurs="unbounded（表示出现次数" 这个属性要写在element中 表示这个元素标签可以多次出险

<any></any> : 表示可以出现任意元素

<attribute name="id1" type="int" use="required"></attribute> 这表示在 约束文件的声明标签里 强制要求一个属性值 id1 类型int

要卸载复杂元素里面 卸载</complexType> 之前

required——必须的

Schema的名称空间 （ ：xsi 用于区分 schema文件

若想要引入多个schema文件 需要给每个起一个别名

sax的解析原理： parser——剖析器

解析xml 有两种技术 dom sax

dom:根据xml的层级结构 分配一个属性结构 吧xml中的标签属性 文本都封装成对象

sax：是事件驱动的 边读边解析

在java.xml.parsers 包中

SAXParsers 由于是抽象类 不能使用new

sax解析的过程 ：

xml文件

///￥￥￥ ArrayList 适于读取 LinkedList适于添加

使用jaxp的sax方式解析xml

这是xml文件:

<?version=”1.0” encoding=”utf-8”?>

//声明文件

<person>

<p1>

<name>lxa</name>

<age>20</age>

</p1>

<p1>

<name>zjy</name>

<age>19</age>

</p1>

</person>

创建一个java :

Public class Sax (Sax 解析模式)

Public static void main(String arg[] ){

1,创建解析器工厂

2，创建解析器

3，执行parse方法

4，创建一个class 要继承 DefaultHandler

5，重写里面的三个方法

SAXParserFactory saxpf = SAXParserFactory.newInstance();

SAXParser saxp = saxpf.newSAXParser();

saxParser.parse(“xml地址”,此处是事件处理器);

//parse方法 :

第一个参数xml路径 第二个参数 事件处理器

重写新的事件处理器 的三个方法。。

class MYDefault extends DefaultHandler { //自定义事件处理器 。

// 右键 source -》 override 找到需要用的方法

startElement

characters

endElement 这三个方法。。

实例：

获取所有的name元素的值 通过一个boolean进行筛选

// 接上内容

Class myDefault2 extends DefaultHandler{

重写方法 :

startElement 这个方法 返回 qName

用qName 判断

If(“name”.equals(qName)){

//写入内容 }

\*\*\* Dom4j 来解析 xml

Dom4j是一个组织 针对xml的解析 提供了dom4j

Dom4j 不是javase的一部分 想要使用需要 :

导入don4j提供的jar包

* + - * 1. 创建一个文件夹lib
        2. 复制jar包到libxia
        3. 右键点击 jar build path --add to buildpath
        4. 看到jar包 变成一个奶瓶的样子 说明导入成功

得到document

SAXReader s = new SAXReader();

Document doc = reader.read(url)；

Document的父接口是Node

如果在document里面找不到想要的方法 到Node里去找

Document里面的方法getRootElement 获取根节点 返回Element

Element 也是一个接口 父接口是 Node

getParent 获取父节点

使用dom4j来查询xml

查询所有name元素的值：

1. 创建解析器
2. 得到document
3. 得到根节点 getRootElement

\*\*\* element elements 获取的都是第一层的子标签 !！！！

1. 得到所有p1 element(qName)获得标签下面的第一个子标签 只返回一个Element elements(qName) 获得标签下面的所有是qName的子标签 返回list elements() 获取标签下面的一层子标签
2. 得到name 并得到里面的值

下面是一个java文件

//查询xml之中所有name元素的值

Public static void seletName(){

//创建解析器 SAXReader sax = new SAXReader();

Document doc = sax.read(“xml路径”); 这里是don4j的document

Element root = doc.getRootElement();

//得到p1

List<Element> list = Root.elements(“p1”);

//遍历list ….

得到p1下面name元素‘

Element e = Element.element(“name”);

String s = e.getText();

Public static void seletSin(){

创建解析器 SAXReader s = new SAXReader();

Document doc = s.read(“xml路径”);

Element root = Doc.getRootElement();

Element p1 = root.element(“p1”)’

Element name = p1.element(“name”);

得到name的值

String s = name.getText();

获取第二个selectSecond（）{

SAXReader s = new SAXReader();

Document doc = s.read();

Element root = doc.getRootElement();

List list = root.elements();

Element e = List.get(1 );

Element name = e.element(“name”);

String s = name.getText();

使用dom4j 实现 添加操作 \*\*\* 一定要会写

XMLWriter 直接new 传递两个参数 FileOutputStream 2,格式化类的值 OutputFormat

在p1末尾添加sex标签

Public static void addSex(){

SAXReader s = new SAXReader();

Document doc = s.read();

Element root = doc.getRootElement();

Element p1 = root.element(“p1”);

//用jaxp添加的时候要先创建 现在可以直接进行添加

Element sex1 = P1.addElement(“sex”);

//在sex 下面添加文本

Sex1.setText(“man”);

//回写xml 需要用到XMLWriter 可以new

定义：XMLWriter w = new XMLWriter(OutputStream,OutputFormat);

OutputFormat f = OutputFormat.createPrettyprint();

XMLWriter w = new XMLWriter(new FileOutputStream(),f);

w.write(doc);

w.close();

在特定位置 添加元素 \*\*\* 使用add(int index,e element) 进行添加

第一个参数是 位置 从0开始

第二个参数是 要添加的元素

在第一个p1下面的age标签之前 <school>yuxin</school>

Public static void add(){

SAXReader s = new SAXReader();

Document doc = s.read(“xml地址”);

Element e = Doc.getRootElement();

Element p1 = e.element(“p1”);

List<Element> list = p1.elements();

//这里需要创建一个元素

Element school = DocumentHelper.createElement(“school”);

School.setText(“yuxin”);

List.add(1,school);

//回写 XMLWriter 是dom4j 里的

OutputFormat f = OutputFormat.createPrettyPrint();

XMLWriter w = new XMLWriter(new FileOutputStream(“xml地址”),f);

w.write(doc);

w.close();

}

Dom4j里面封装方法的操作

可以提高代码的可维护性 提高效率

//以下是读取document

Public static Document getDocument(String path){

SAXReader s = new SAXReader():

Try{

Document doc = S.read(path);

Return doc ;

}catch(){}

Return null ;

}

//以下是回写

Public static void xmlWriters(String path,Document doc ){

OutputFormat f = OutputFormat.createPrettyPrint();

XMLWriter xml = new XMLWriter(new FileOutputStream(path),f);

Xml.writer(doc);

Xml.close();

}

·使用dom4j 实现修改的操作

Public static void modiftAge(){

SAXReader saxreader = new SAXReader();

Document doc = saxreader.read(“xml地址”);

Element root = doc.getRootElement();

Element p1 = root.element(‘’p1’);

Element age = p1.element(“age”);

Age.setText(“21”);

OutputFormat f = OutputFormat.createPrettyPrint();

XMLWriter xmlwriter = new XMLWriter(new FileOutputStream(“xml地址”,f);

Xmlwriter.write(doc);

Xmlwriter.close();

}

\*\*\*\*\* ctrl shift + o 快速导入包

使用dom4j 来实现删除节点的操作 必须通过父节点进行删除

例如删除第一个school元素

Public static void deletElement(){

SAXReader saxreader = new SAXReader();

Document doc = saxreader.read(“”);

Element root = doc.getRootElement();

Element p1 = root.element(“p1”);

Element sch = p1.element(“school”);

// 用remove方法

P1.remove(sch);

OutputFormat f = OutputFormat.createPrettyPrint();

XMLWriter xmlw = new XMLWriter(new FileOutputStream(),f);

Xmlw.write();

Xmlw.close();

}

使用dom4j 来获得属性的要求   
 例如要获取p1标签里面的id的值

Public static void getId(){

SAXReader saxreader = new SAXReader();

Document doc = saxreader.read();

Element root = doc.getRootElement();

Element p1 = root.element(“p1”);

// 用attributeValue 方法：

String s = P1.attributeValue(“id”);

System.out.println(s);

}

又是需要一层一层解析 层数过多消耗大

提供了dom4j 支持xpath的操作

可以直接获取到某个元素

第一种形式 :

/AAA/DDD/BBB 表示一层一层 AAA下面 DDD下面的BBB

第二种形式

//BBB 表示和这个名称相同 表示只要是名称是BBB 都得到

第三种形式：

/\* 表示所有元素

实例：第四种形式

/AAA/DDD[1] 第一个DDD元素

/FFF[last()] 表示最后一个FFF元素

第五种形式：

//@id 表示标签上有id属性都得到

//BBB@id 表示BBB上有id属性 都得到

第六种形式：

//AAA[@id=’a’] 表示有属性id且值为a的元素

//BBB[@name=’lxa’] 表示有属性name 且值为lxa的元素

\*\*\* 以下摘自w3c



使用dom4j支持xpath的具体操作

默认的情况下 dom4j不支持xpath

1. 引入支持xpath的jar包
2. 使用jaxen-1.1-beta-6.jar (复制到lib目录 右键buildpath
3. 在dom4j里面提供了方法 用来支持xpath

\*\*\*\*\* 重要方法：引号里的内容都是xpath语法！！

个人认为比elements element 好用一些

* + 1. selectNodes(“xpath表达式”) 获取多个节点
    2. selectSingleNode(“xpath表达式”) 获取一个节点

练习 : 使用xpath来进行查询xml里name元素的值

所有name元素 的xpath表示 //name

需要使用selectNodes(“//name”) 括号里直接写xpath表达式

Public static void selectName(){

SAXReader saxreader = new SAXReader();

Document doc = Saxreader.read(“xml路径”);

//List 是 util 包 Node是dom4j包

List<Node> list = Doc.selectNodes(“//name”);

//遍历list

For(Node node : list ){

//node是每一个name元素

String s = node.getText();

}

//用以上方法写 会省去很多步骤

}

练习 : 获取第一个p1下面name的值

使用了selectSingleNode()方法

Public static void getp1(){

SAXReader sax = new SAXReader();

Document doc = sax.read();

Node name = doc.selectSingleNode(“//p1[@id=’aaa’]/name”);

\*\*\* [@id=’a’]表示属性id为a的进行选择

//得到name里面的值

String s = name.getText();

System.out.println(s);

}

实现简单的学生管理系统 ( 用xml当作数据库

创建一个xml文件 写入学生信息 在myeclipse

总结：

使用xml当作数据 写学生信息

增加操作

创建解析器

得到doc

获取根节点 创建标签

添加自标签

回写\*\*\*

删除操作

创建解析器

。。。同上

使用xpath selectNodes 选出id一致的标签

用getparent remove方法进行删除

回写

查询操作

基本步骤同上

Xpath “.//\*” 查询自标签 gettext进行打印或者赋值

回写

注意回写操作：XMLWriter(new FileOutputStream(),OutputFormat //这个是抽象类 不能new);

myEclipse 的安装与使用：

eclipse 是免费的开发工具

myeclipse是收费的插件 ( = = 。

工作空间不能有中文 和 空格

创建类的首字母要大写！！

Debug的调试模式

断点调试模式 可以看程序里面数据的变化

使用debug的第一步需要设置断点

F6 可以使程序继续运行

F8 表示调试结束

直接向下运行 如果下面有断点 则在断点出暂停 ，没有断点就直接继续运行

Debug也可以查看源代码 F7可以进入源代码

Myeclipse的快捷键：

常用快捷键：

Ctrl shift o : 快速导入包

Ctrl shift / : 多行注释

Ctrl shift \ : 取消多行注释

右键source – format 格式化 有缩进效果

/\* syso alt+/ 可以快速补全 system.out.println();

Junit的使用

单元测试：测试对象是一个类中的方法

juint 不是javase的一部分被 需要导入jar（但是myeclipse里自带了juint

首先juint版本

@Test : 在方法上面写

单元测试 的方法 : 方法命名规则 public void 方法名(){}

使用注解方法：运行点击run as – joint test

当出现了 绿色 表示测试通过

@Ignore : 忽略下面的方法不去测试

@Before:

@After :

断言：（用的很少

Assert.assertEquals(期望的值 ， 方法运行实际的值)；

JDK5.0 新特性：

泛型 枚举 静态导入 自动拆装箱 增强for 可变参数 反射

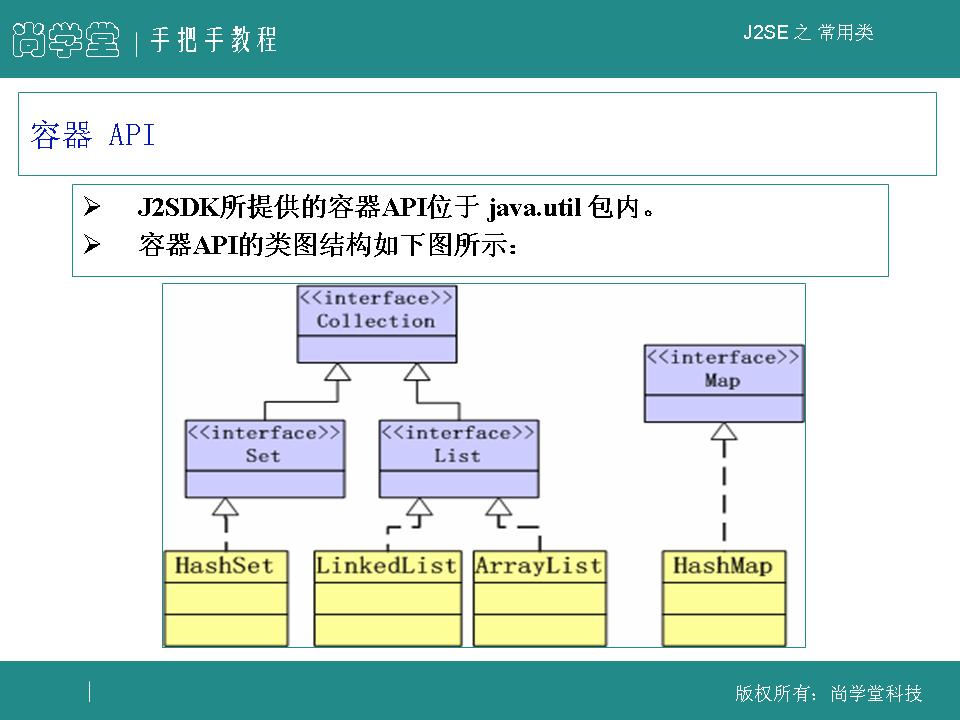
泛型：一般是用在集合上

比如现在把一个字符串类型的值放入集合，取值时不用进行类型转换

解决了类型转换的问题

再集合上如何使用泛型：

常见集合：list set map ctrl shift o 快速导入包



几种常见的遍历方式：

For(int I ; I < ? ;i++){}

For(String s :list){}

Iterator I = list.iterator();

泛型使用在set集合上：set是无序的 且不能重复

Set<String> t = new HashSet<String>();

遍历map：

Set<String> s = map.keySet();

For(String key :s){

String s1 = map.get(key);

# Java中map的entrySet()方法返回的是什么内容啊？

简单的理解，就是Entry.set是获得一个set集合，并且泛型是Map.Entry的对象结果集，这样你就可以通过Set集合来进行遍历，是一种Map功能的增强。使用这种方式你可以在不知道key的情况下遍历Map对象。

泛型里面的对象 必须是包装类

不能写 int 要写 Integer

short Short

byte Byte

char Character…

泛型使用在方法上的：

定义一个数组 实现制定位置上数组的交换

实例：

Public static void main(String arg[]){

Int[] arr = {10,11,12,13,14};

Swap1(arr,1,3);

}

Private static void swap1(int[] arr,int a,int b){

//这里需要定义一个中间变量。。。

若方法和逻辑相同 只是数据类型不同 可以使用泛型方法

使用泛型方法 需要定义一个类型 使用大写字母表示 R : 这个T表示任意类型

写在返回值之前void之前

表示定义了一个类型 这个类型是T

在下面就可以使用这个类型了T

Public static <T> void swap1(T[] arr,int a ,int b){

T temp = arr[a];

Arr[a] = arr[b];

Arr[b] = temp ;

/\*temp 意为 临时文件\*/

} \*\*\* 这里面的T表示任意类型 ,

而且泛型方法里面数据类型也一定要用包装类型

枚举简介：

什么是?

需要在一定范围内取值 这个值只能是这个范围中的任意一个

例如：交通信号灯

Public static final col Red = new Col();

Public static final col Greem = new Col();

Public static final col Yellow = new Col();

//枚举 :

Enum Col2{

Red,Green,Yellow;

}

\*\*\*\* 枚举的构造方法也是私有的

如果构造方法里面有参数 则需要在每个实例里面写参数

特殊枚举的操作 :

在枚举的类里面 有抽象的方法

当我们在枚举里面写下 抽象方法 需要在每个实例上都实现抽象方法

枚举api的操作：

Name（）返回枚举名称

Ordinal（） 返回枚举的下标

valueOf（Class<T> enumType ,String s ） 返回枚举的对象

valueOf（String s ） 返回枚举的对象

values（）获得枚举对象的数组 返回Object数组

eg：

Enum Col2{

Red,Green,Yellow;

}

Public void test(){

Col2 col = Col2.Red;

String name = col.name() 返回String 名称

Int index = col.oradinal(); 返回下标

//知道枚举对象的名称 得到枚举对象 下标

ValueOf方法

//根据下标得到对象

Col2[] list = Col2.values();

Col2 col = list[下标];

静态导入：（容易出错

可以在代码里直接使用静态方法

自动拆装箱：

装箱: 把基本的数据类型 转化成包装类

拆箱：把包装类转化成数据类型

增强for循环：

实现Iterable接口可以使用增强for循环

只有list set 可以使用增强for循环

Map不能使用增强for ( 因为没有Iterable接口)

增强for循环是为了替代迭代器（增强for的底层其实就是迭代器

语法： for(遍历的类 名称：要进行遍历的集合){}

内容补充

泛型的擦除：

首先泛型只是出现在源代码阶段

练习：实现一个泛型方法 接受任意类型的数组 颠倒数组所有元素

Public static void main(String arg[]){

Integer[] arr = {2,3,1,4};

Reverses(arr);

}

Public static <T> void reverses(<T>[] arr){

For(int i=0 ;i<arr.length/2;i++){

//进行交换

}

可变参数：

可变参数 可以应用在：

实现两个数的相加

实现三个数的相加

四个数的相加

定义一个方法实现两个数的相加

Public void add(int a,int b){

Int sum = a+b;

System.out.println(sum);

}

实现三个数相加

Public void add2(int a ,int b,int c){

Int sum1 = a+b+c;

System.out.println(sum1);

}

以上方法的逻辑基本相同

太麻烦 可以使用可变参数来实现

Public void add(int…nums){

//nums表示可以理解为一个数组 用来储存传递过来的参数

\*\*\* 可变参数必须写在参数列表里 不能单独定义

}

Public void add1(int a ,int…nums) 也可以这样写

\*\*\* 可变参数必须放在参数列表最后

DEBUG调试模式：

F6 单步执行

F8 结束断点 后面有断点到下一个断点

\*\*\* 泛型方法：

\*\* 枚举要会用

\*\*\* 自动拆装箱

拆箱： 数据类型——包装类型

装箱： 包装类型——数据类型

增强for循环：

底层的实现是迭代器

可变参数 :

写法：int…nums

可变参数 的应用场景：逻辑相同

参数列表里定义可变参数 且只能有一个可变参数

\*\*\* 反射：

应用在一些通用性高的代码中：

之后的框架：大部分由反射实现

框架的底层难以理解

框架都是基于配置文件来进行开发

在配置文件中配置了类，可以通过反射得到类中的所有方法

Eg：

Public class Person{

Private String name ;

Private String id ;

Public Person(){

//没有参数的构造方法

}

Public Person(int a){

}

可以通过反射来得到所有方法参数…

当写下一个java文件后 先保存 在编译

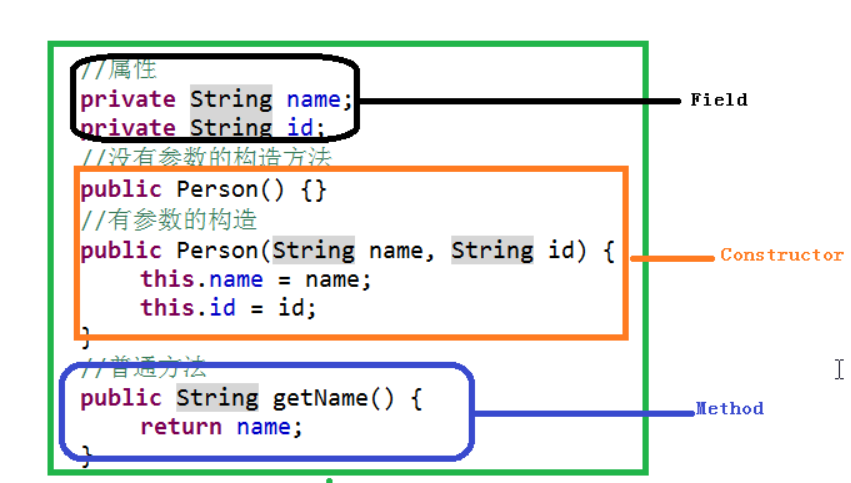
.java->.class->把class文件加载到内存 (使用了类加载器)

如果得到了class类，可以得到这个类中所有的内容

使用反射 :

首先得到class类

* 有三种方式：
  + 1类明.class
  + 2对象.getClass()
  + 3使用Class.forName(“路径”)



使用反射操作类里面的属性：

Class clazz1 = Person.class;

Class clazz2 = new Person().getClass();

Class clazz3 = Class.forName(“路径”);

对于一个类进行实例化 可以new 不使用new 怎么获取？？

得到class

Class c = Class.forName(“路径”);

Person p = c.newInstance();

//这样就获得了person的实例。

若操作有参数的构造方法:

Class c = Class.forName();

Constructor cs = c.getConstructor(String.class); //这里面的String.class表示String的参数

//通过有参数的构造方法来创建了person的实例

Person p = (Person)cs.newInstance(“这里是参数”)；

使用反射来操作属性：

得到class类 :

类名.class

对象.getClass

使用Class.forName 方法

操作无参数构造方法：

当对类进行实例化时，Person p = (Person)class.newInstance();

有参数的构造方法：

Class c = Class.forName();

Constructor cs = c.getConstructor(String.class//这个是参数的类型.class);

Person p = (Person)cs.newInstance(“传入String类”);

操作普通方法：

得到Class类“

Class c = Class.forName();

Person p = (Person)c.newInstance();

Method m = c.getDeclaredMethod(“这里是方法名”,String.class(这是方法参数的类型);

m.invoke(p,”这里是设置的值”); //执行方法>

\*\*\* m.setAccessible(true) 表示可以使用私有方法

总结：

泛型 集合使用泛型 泛型方法泛型类

自动拆装箱 向下兼容

要理解反射的原理

软件体系结构

常见的软件结构 BS CS

CS: 客户端 服务器 安全性高 例如：qq

BS：浏览器 服务器 不需要安装客户端，不用升级

WEB服务器：

Tomcat不支持JAVAEE

可以通过tomcat目录下的：startup shutdown 来进行开启关闭tomcat

\*\*\* 如果startup 一点就消失 说明JAVA\_HOME 不存在



WEB 应用：

在webapps下创建一个目录 ( 不能包含中文 空格)

这个目录是项目目录

在项目下创建一个html文件

创建一个html文件

在项目目录下创建如下内容：

WEB-INF目录：安全目录

在里面创建一个xml文件叫做：web.xml（这个文件是用来初始化配置信息

http 协议：

协议的甲乙方就是 客户端 服务器

可以理解为双方通信的格式！

请求协议

响应协议

请求头：

Referer: 在百度里点击链接 则请求头就是百度（地址栏里发生的地址转跳 不算

Referer的作用: 统计访问来源， 统计访问量

Referer可以防盗链

响应协议：

响应首行：

响应头信息：

\*\* 响应码：

200 成功

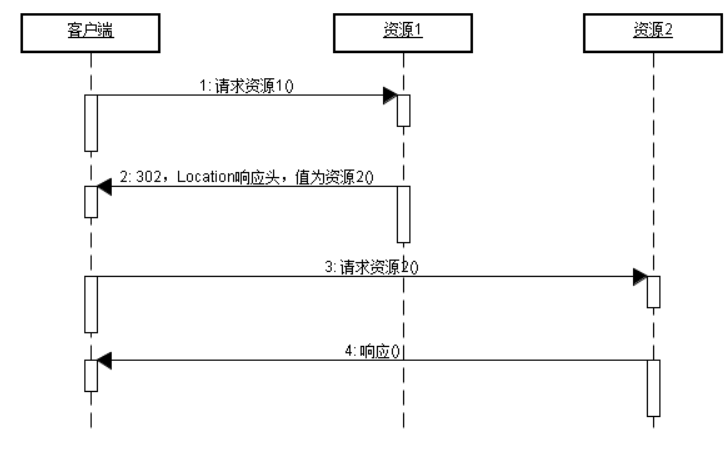
404 请求的资源没有找到

405 不支持访问方式

403 Forbidden //服务器收到请求，但是拒绝提供服务

500 请求资源找到了 但服务器内部出现了错误

302 \*重定向\* 表示服务器要求再发送一个请求 服务器的会发送一个响应头 Location 制定新的请求url地址



304 ：可以节省传输成本 直接用缓存 是比较Last-Modified 和 If-Modified-Since 的时间与真是文件真是的时间一样是 服务器会返回304 而却不会影响正文



4开头的都是客户端的错误

响应头： Last-Modified :最后修改时间

If-Modified-Since:吧上一次请求的index.html的最后修改时间还给服务器

Refresh: 自动刷新

Eg:Refresh:3;url=http://lxa.com (3秒后 转跳到lxa 这个页面

<head><meta http-equiv="refresh" content="5" /></head>

使用java绘制图片：

相关的类：Image ImageIO BufferedImage Icon ImageIcon

验证码的作用 : 防止恶意注册

软件体系结构：BS CS 浏览器服务器 客户端服务器

Tomcat是web服务器 jsp容器servlet容器

默认端口号：8080 网页的默认是：80

Web应用：

目录结构：



状态码： 200 404 304 302 500 Referer请求头 有来源网址信息 盗链 统计访问次数等作用

Refresh 自动刷新 Eg:Refresh:3;url=http://lxa.com (3秒后 转跳到lxa 这个页面

//\*\* 反射存在的意义就是在执行一段程序时如果需要执行其他类或代码 反射可以直接调用实例化需要用的代码 不用停止现在正在运行的代码再来使用需要用的代码\*\*//

**\*\*\*\***

**同步**和**异步**通常用来形容一次方法调用。

**同步方法**调用一旦开始，调用者必须等到方法调用返回后，才能继续后续的行为。

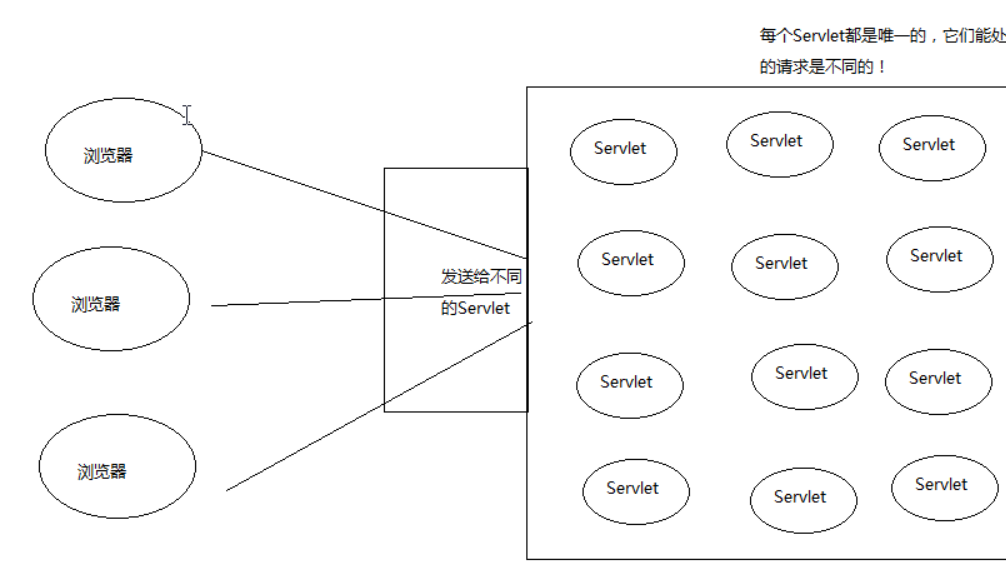
**异步方法**调用更像一个消息传递，一旦开始，方法调用就会立即返回，调用者就可以继续后续的操作。而，异步方法通常会在另外一个线程中，“真实”地执行着。整个过程，不会阻碍调用者的工作。

Servlet：

是javaweb三大组建之一 是服务器小程序 由Applet衍生出来

每个servlet都是唯一的 他们的功能都是不同的

多线程并发访问



Servlet需要：

接受请求数据

处理请求

完成响应

实现servlet的方式：

实现servlet的三种方式：

实现javax.servlet.Servlet接口

继承javax.servlet.GenericServlet

继承javax.servlet.http.HttpServlet

\*\* 我们通常会使用继承HttpServlet 比较方便

这几个方法httpServlet是GenericServlet的子类 GenericSerlvet是调用了Serlvet的接口

Servlet接口定义了5种方法：

* init() 只在第一次初始化调用生成实例化一次
* service() 每次访问调用一次 传递request response
* destroy() servlet销毁时调用
* getServletConfig()
* getServletInfo()

## init() 只会在第一次启动进行实例化servlet时调用

在Servlet实例化后，Servlet容器会调用init()方法来初始化该对象，主要是为了让Servlet对象在处理客户请求前可以完成一些初始化工作，例如：建立数据库的连接，获取配置信息等。对于每一个Servlet实例，init()方法只能被调用一次。init()方法有一个类型为ServletConfig的参数，Servlet容器通过这个参数向Servlet传递配置信息。Servlet使用ServletConfig对象从Web应用程序的配置信息中获取以名-值对形式提供的初始化参数。另外，在Servlet中，还可以通过ServletConfig对象获取描述Servlet运行环境的ServletContext对象，使用该对象，Servlet可以和它的Servlet容器进行通信。

在GenericServlet抽象类中的Init方法还有一个没有参数的方法 用于进行改写等等 原因是 有ServletConfig参数的方法在使用时候容易忘记 this.ServletConfig = ServletConfig 这里的Servlet是传入的参数

## service()

容器调用service()方法来处理客户端的请求。要注意的是，在service()方法被容器调用之前，必须确保init()方法正确完成。容器会构造一个表示客户端请求信息的请求对象（类型为ServletRequest）和一个用于对客户端进行响应的响应对象（类型为ServletResponse）作为参数传递给service()。在service()方法中，Servlet对象通过ServletRequest对象得到客户端的相关信息和请求信息，在对请求进行处理后，调用ServletResponse对象的方法设置响应信息。

## destroy()

当容器检测到一个Servlet对象应该从服务中被移除的时候，容器会调用该对象的destroy()方法，以便让Servlet对象可以释放它所使用的资源，保存数据到持久存储设备中，例如将内存中的数据保存到数据库中，关闭数据库的连接等。当需要释放内存或者容器关闭时，容器就会调用Servlet对象的destroy()方法，在Servlet容器调用destroy()方法前，如果还有其他的线程正在service()方法中执行容器会等待这些线程执行完毕或者等待服务器设定的超时值到达。一旦Servlet对象的destroy()方法被调用，容器不回再把请求发送给该对象。如果需要改Servlet再次为客户端服务，容器将会重新产生一个Servlet对象来处理客户端的请求。在destroy()方法调用之后，容器会释放这个Servlet对象，在随后的时间内，该对象会被java的垃圾收集器所回收。

## getServletConfig()

该方法返回容器调用init()方法时传递给Servlet对象的ServletConfig对象，ServletConfig对象包含了Servlet的初始化参数。

## getServletInfo()

返回一个String类型的字符串，其中包括了关于Servlet的信息，例如，作者、版本和版权。该方法返回的应该是纯文本字符串，而不是任何类型的标记。

如何使用浏览器访问servlet：

1. 给servlet指定一个servlet路径
2. 浏览器访问servlet路径
   1. 给servlet配置路径：

需要在web.xml中对servlet进行配置

<load-on-startup>0 意味着在tomcat启动时 就对servlet进行实例化 这样可以避免在第一次访问servlet时进行实例化Servlet会造成反应缓慢。



Mapping英文翻译——映射



ServletConfig可以获取配置信息

一个ServletConfig对象 对应一段web.xml

ServletConfig对象的功能：

String getServletName() 获取servlet-name标签的内容

ServletContext getServletContext() 获取servlet上下对象

/\*\*父类的private 子类能否继承？

正确的回答是：  
如果一个子类继承了父类，那么这个子类拥有父类所有的成员属性和方法，即使是父类里有private属性的变量，子类也是继承的，只不过不能使用，也就是说，它继承了，但是没有使用权，似乎又点矛盾，用我们通俗的说法就是 只能看，不能用

抽象类和接口的区别：抽象类由abstract关键字来修饰，接口由interface关键字来修饰。抽象类中除了有抽象方法外，也可以有数据成员和非抽象方法；而接口中所有的方法必须都是抽象的，接口中也可以定义数据成员，但必须是常量。

定义常量：方法一采用接口(Interface)的中变量默认为static final的特性。

方法二采用了[**Java**](http://lib.csdn.net/base/java)5.0中引入的Enum类型。

方法三采用了在普通类中使用static final修饰变量的方法。

GenericServlet 是一个抽象类 generic英文翻译——类的

GenericServlet

实现了Servlet接口，并帮我们做了一些常用操作

1.init方法 妥善的保存config对象并实现getServletInfo,getServletConfig，  
2.增加一个空参init方法,供开发人员初始化，为了防止开发人员重写 原生init方法  
3.service方法空实现 => 声明成抽象（强制开发人员实现该方法）  
4.destory方法空实现   
5.实现了servletConfig接口. 接口中的方法直接调用config实现类实现.

HttpServlet

HttpServlet的原理:

继承于GenderServlet

方法：

Void service(ServletRequset,ServletResponse) 生命周期方法

强转两个参数为http协议相关的类型

Void service(HttpServletRequest,HttpServletResponse)

参数已经是http协议相关的 使用起来更加方便

Void doGet(){}

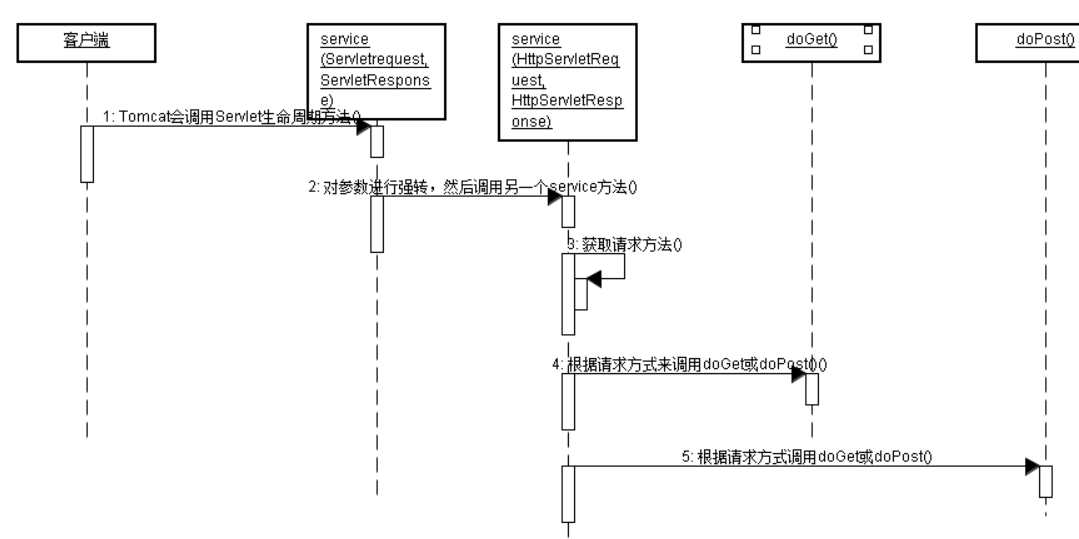
Void doPost(){}

工作原理:

Tomcat会调用Servlet生命周期方法 对参数进行转化 然后调用另外一个service方法 获取请求方法 根据请求方式来调用doGet 或 doPost

DoGet doPost 由我们自己进行覆盖

如果没有覆盖doGet doPost 就会出现405



Servlet与线程安全 ( 优点就是速度快

不要再servlet中创建成员。创建局部变量即可

可以创建无状态成员。

可以创建只读对象成员。

让servlet在启动服务器Tomcat时 就启动servlet

在<servlet>标签之下加入<load-on-startup>0</load-on-startup>



在web.xml中加入图中段落

# servlet 有参数的init方法和无参数的init方法

在servlet生命周期中，首先是构造firstservelt类，2调用有参数的init方法

但是在学习的时候，我们发现存在inti有参数和init无参数的两种方法

有参数是给服务器初始化servlet时调用的

public void init(ServletConfig config) throws ServletException {  
        this.config = config;  
        this.init();  
    }

无参数给开发者使用的

public void init() throws ServletException {  
        // NOOP by default  
    }

如果我们重写有参数的，如果忘记完成this.config = config;即super.config = config这行代码，就有可能出现空指针异常

<url-pattern>用来指定servlet的访问路径 就是url 必须是’/’开头

Web.xml文件的继承（了解

每个javaweb应用都有一个web.xml 文件

如果访问路径不存在时 会执行DefaultServlet 返回404 没有找到页面

如果需要访问jsp扩展名的文件 需要通过JSPservlet进行访问

\*\*\* session 的过期时间是30mins

Servlet必须要有无参数构造类

ServletContext（重要） 使用它来进行数据的传递

一个项目只有一个servletcontext对象

服务器会为每个应用创建一个servletContext对象

在创建服务器启动完成

在销毁服务器是关闭

获取ServletContext ：



域对象功能：用于在servlet之间传递数据

Javaweb四大域对象：

PageContext

ServletRequest

HttpSession

ServletContext

所有域对象都有存取数据的功能 因为域对象内部有一个map 用来存储数据 下面是servletcontext对象用来操作数据的方法。

Void setAttribute(String name,Object value) 用来存储一个对象 也可以称之为储存一个域属性 例如：

servletContext.setAttribute(“name”,”lxa”); 在域属性，域属性名称为xxx 属性为xxx。请注意，如果多次调用方法 并且使用相同的name 那么回覆盖上一次的值 这一特性与map相同

Object getAttribute(String attributeName) : 用来获取servletContext中的属性 当前在获取之前需要先去储存才行，例如： String value = (String ) servletContext.getAttribute(“name of attribute”) /\*\*由于返回的是object 所以要进行强行转化！！

Void removeAttribute(String name) 用来移除servletContext中的域属性 如果参数name指向的域属性不存在 那么方法什么都不做

Enumeration getAttributeNames() 获取所有属性的名称

/\*

hashCode方法实际上返回的就是对象的存储地址。   
可以先记下结论：   
1、如果两个对象相同，那么它们的hashCode值一定要相同；   
2、如果两个对象的hashCode相同，它们并不一定相同   
3.两个对象不相同，他们的hashCode值不一定不相同。   
4.两个对象的hashCode值不相同，他们一定是两个不同的对象

\*/

演示向servletContext中保存数据：



演示从servletContext中获取数据：



以上两个方法是在不同的两个类的doGet方法里实现 在访问对应servlet时 自动调用实现

获取应用初始化参数

Servlet也可以获取初始化参数 但是它只是局部参数 也就是说 一个servlet只能获取自己的初始化参数

也可以配置 \*公共的初始化参数\* 为所有的servlet来进行使用 必须需要用servletContext来进行获取

演示获取公共的参数

1. 得到servletContext
2. 调用getInitParameter（String）得到初始化参数



公共初始化参数 的定义方式：

在web.xml 文件里的 根目录的第一层下

<param-context>

<param-name (这个是初始化参数的name) > XXX</param-name>

<param-value (这个是初始化参数的calue)>YYYY</param-value>

</param-context>

定义完成后需要重新载入tomcat

获取相关的资源

使用servletContext获取资源真实路径

可以使用servletContext对象来获取web应用目录下的资源 例如在Hello应用的根目录下创建 a.txt 文件 现在想在servlet中获取这个资源 就可以使用ServletContext来进行获取

获取a.txt的真实路径： String realpath = servletContext.getRealPath(“这里是文件的名称”)

返回的值 是文件的绝对路径

获取b.txt的真实路径 String realpath = servletContext.getRealPath(“WEB-INF/b.txt”) 说明b.txt 在web-inf 的文件目录下

获取资源流

InPutStream in = this.getServletContext().getResourceStream(“需要读取的文件路径”)

获取文件的资源

练习 : 访问量的统计

一个项目中所有的资源访问 都要对访问量进行累加

创建一个int类型的变量 用来保存访问量 然后保存到servletContext的域中 这样可以保存所有的servlet都可以访问到。

\*\* 最初servletContext中没有保存访问量相关的属性

\*\* 当网站的第一次被访问时 创建一个变量设置值为1 然后保存到servletContext之中

当以后的访问 就可以从servletContext中访问这个变量 然后+1

获取servletContext对象 查看count的属性 进行操作

PrintWriter out = response.getWriter(); 想浏览器输出的方式



获取类路径下的资源：获取类路径资源 对于javaweb项目而言 就是 WEB-INF/classes目录下的文件

Class

ClassLoader

得到ClassLoader

调用其getResourceAsStream() 得到一个InputStream

ClassLoader cl = this.getClass().getClassLoader();(从本class中调用classloader 借刀杀人 - - )

InputStrean in = cl.getResourceAsStream(“classes目录下的文件名”)

\*\*\* String s = IOUtils.toString(in); //读取输入流内容 转化成String返回

#### 反射获取class对象的三种方式：

Person person = new Person();

//1、通过Object类的getClass()方法：（需要先实例化一个对象）

Class clazz1 = person.**getClass()**;

//2、通过对象实例方法获取对象：（需要先实例化一个对象）

Class clazz2 = person.**class**;

//3、类的全路径：（不许呀实例对象）

Class clazz3 = Class.**forName("com.cn.Person")**;

对于有空的构造函数的类可以直接用字节码文件获取实例：

　　Object objt = clazz.newInstance();　　//会调用空参构造器（如果没有则会报错）；

对于没有空的构造函数的类则需要先获取到他的构造对象，在通过该构造方法类获取实例

　　1、获取构造函数

　　　　Constroctor const = clazz3.getConstructor(String.class,int.class);

　　2、通过构造器对象的newInsttance方法进行对象的初始化

　　　　Object obj = const.newInstance("tom",30);

关于getClass方法：

Java的每个类都带有一个运行时类对象，该Class对象中保存了创建对象所需的所有信息。  
可以用.class返回此 Object 的运行时类Class对象，也可以用getClass()获得。  
获得此对象后可以利用此Class对象的一些反射特性进行操作，  
例如：  
this.getClass().newInstance(); //用[缺省构造函数](https://www.baidu.com/s?wd=%E7%BC%BA%E7%9C%81%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dWnyD4rHR1PH-hrHI9nvfv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTzrjbvnjm4)创建一个该类的对象  
this.getClass().getInterfaces(); //获得此类实现的接口信息  
this.getClass().getMethods();//获得此类实现的所有公有方法  
  
Class.forName(" ... JDBC driver class name...."); // [Class类](https://www.baidu.com/s?wd=Class%E7%B1%BB&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dWnyD4rHR1PH-hrHI9nvfv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTzrjbvnjm4)的静态方法forName, 向DiverManager注册这个JDBC driver类

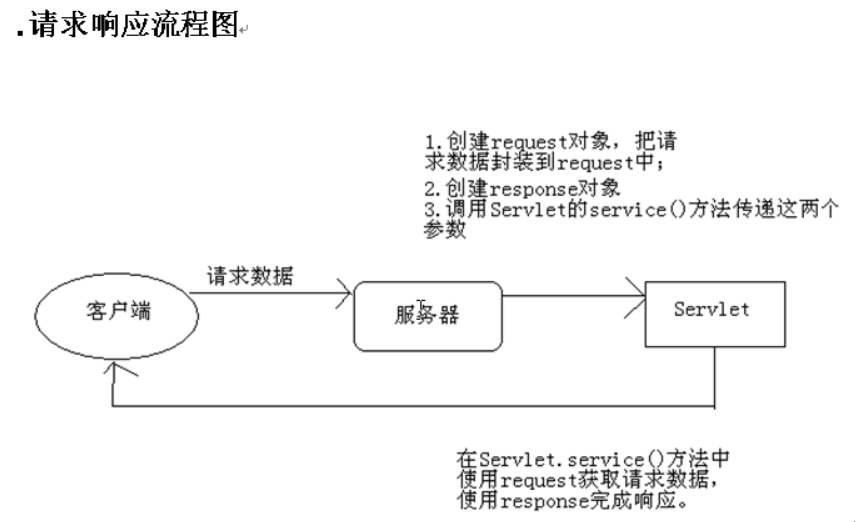
内容：

Response

Request

编码

路径



服务器创建request对象 把请求数据封装到request中

创建response对象 调用servlet的service()方法传递参数

在serlvet.service() 方法中使用request获取请求的数据 使用response成完请求的响应

服务器请求的流程：

服务器每次收到请求时 都会为这个请求开启一个新的线程

服务器会把客户端的请求数据封装到request对象中 request就是这个请求的数据的载体

服务器还会创建response对象 这个对象与客户端连接在一起 它可以用来向客户端发送响应

Response request 在请求结束的时候都会自动销毁

服务器可以同时接受多个请求

Response request 都是由服务器进行创建的

http的响应结构 response :

状态码 200成功 404客户端错误 403访问被拒绝 500服务器错误 302重定向 304未修改可以使用之前的缓存

响应头

响应体

<html>

….

</html>

Response 其类型为HttpServletResponse

ServletResponse 与协议无关的类型

HttpServletResponse 与http协议相关的类型

sendError (int sc) 发送错误的状态码

sendError (int sc ,String msg) 发送错误的状态码 并返回文本

sendStatus (int sc) 发送成功的状态码

\*\*\* 注意都是send不是set

status 英文翻译 --- 状态

response的响应头相关的方法：

响应头： Content-Type, Refresh, Location 等等…

setHeader(String name,String value)适用于单值响应头

addHeader(String name,String value) 适用于多值响应头

response.addHeader(“aa”,”A”);

response.addHeader(“aa”,”B”); //分开进行设置

setIntHeader(String name,int value) 适用于单值响应头设置int类型值

addIntHeader(String name,int value) 适用于多值响应头设置int类型值

setDateHeader(String name,long value) 适用于单值响应头设置 毫秒 类型值

这里的value的类型是long 是因为浏览器无法处理date()类型的参数数据

这里的 参数 1000 = 1 秒

可以设置 response.setDateHeader(“expires”,1000\*60\*60\*24) 进行设置过期时间 这里是1天过期

使用response完成重定向

使用到 response.setHeader(“Location”,”/Hello/Next”这是重定向的地址页面

response.sendStatus(302); 发送重定向状态码

设置定时刷新

设置一个Refresh 表示定时刷新

response.setHeader(“Refresh”,”5; /Hello/B 这里是转跳到的页面地址”);

response.setHeader(“Refresh”,”5;http://www.baidu.com”);

禁止浏览器进行缓存

Cache-Control ,pragma ,expires

eg:

response.setHeader(“Cache-Control”,”no-cache”);

response.setHeader(“pragma”,”no-cache”);

response.setDateHeader(“expires”,”0”);

<meta> 标签可以充当响应头

response 有两个流

getOutputStream

getWriter

\*\* 这两个流不能一起使用 会抛出异常

PrintWriter writer = response.getWriter();

ServletOutputStream sos = response.getOutputStream();

response的快捷重定向方法：

sendRedirect(String local)方法

//快捷重定向的方法

response.sendRedirect(“http://www.baidu.com”);

request

request格式:

请求行

请求头

空行

请求体

获取常用地址：

获取客户端IP 案例： 可以封IP

request.getRemoteAddr()

获取请求方式：

request.getMethod() 可能是post ， get

获取请求头

getHeader(String name ) 返回String 适用于单值

getIntHeader(String name) 适用于int返回值的单值请求头

getDateHeader(String name) 适用于好免得返回值的请求头

案例：

通过User-Agent 识别用户的浏览器类型

request.getHeader(“User-Agent”) 返回String值 里面是浏览器信息等等

获取请求url：

<http://localhost:8080/day01/Servlet?usename-xxx&password=yyy>

String getScheme 获取协议：http

String getServerName 获取服务器: localhost

String getServerPort 获取端口号： 8080

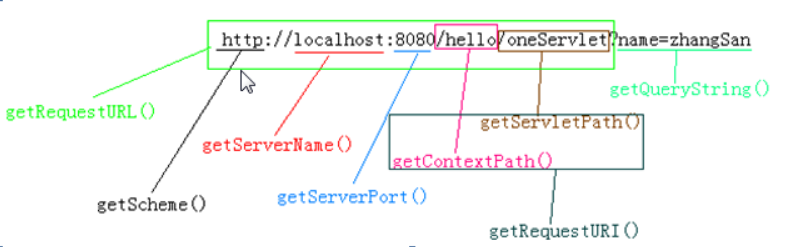
String getContextPath 获取项目名 /day01

String getServletPath 获取Servlet路径 /XXX/Servlet

String getQueryString 获取参数部分 就是问号后面的内容 usename=xxx&password=yyy

String getRequestURL 获取请求URL 等于不包含参数的整个请求路径:

<http://localhost:8080/day01/Servlet>



Referer : 可以告诉 请求来自的地址

可以统计来访的页面

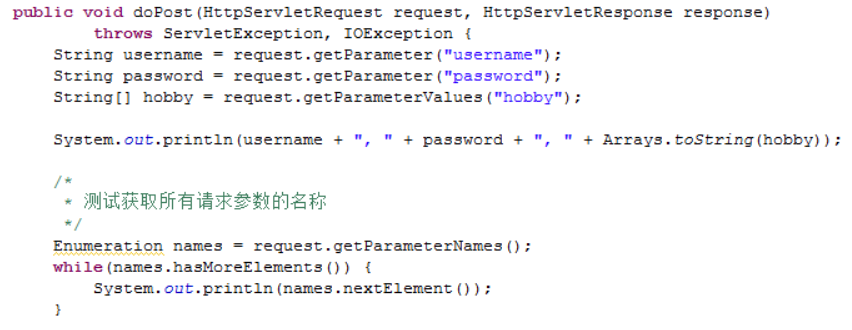
可以防盗链



请求参数：

由客户端发送给服务器

请求参数可能在请求提 可能在url中（get post 两种不同方法）



Servlet  eroor：HTTP method GET is not supported by this URL

**错误提示：**

 type: Status report

message: HTTP method GET is not supported by this URL

description: The specified HTTP method is not allowed for the requested resource (HTTP method GET is not supported by this URL).

**原因：**

1，继承自HttpServlet的Servlet没有重写对于请求和响应的处理方法：doGet或doPost等方法；默认调用父类的doGet或doPost等方法；

2，父类HttpServlet的doGet或doPost等方法覆盖了你重写的doGet或doPost等方法；

不管是1或2，父类HttpServlet的doGet或doPost等方法的默认实现是返回状态代码为405的HTTP错误表示对于指定资源的请求方法不被允许。

**解决方法：**

1，子类重写doGet或doPost等方法；

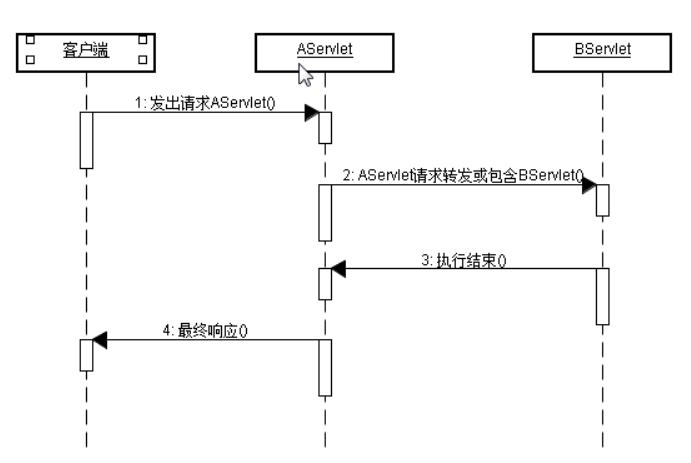
 2，在你扩展的Servlert中重写doGet或doPost等方法来处理请求和响应时 不要调用父类HttpServlet的doGet或doPost等方法，即去掉super.doGet(request, response)和super.doPost(request, response);

我的解决方法是： 删去super代码

请求转发 请求包含

有些任务一个servlet完成不了 会将请求发送给其他servlet

客户端发送请求 Aservlet解决不了 AServlet 请求转发BServlet BServlet结束执行 返还AServlet AServlet最终响应回客户端



RequestDispatcher rd = request.getRequestDispatcher(“/MyServlet” 这个是要传递到servlet的路径 );

请求转发： rd.forward(request,response);

有时候一个请求需要多个servlet 需要使用转发和包含：

#### javaweb之请求转发和请求包含

**·**关于请求转发和请求包含我们首先得知道无论是请求转发还是请求包含，都表示由多个Servlet共同来处理一个请求。

  例如Servlet1来处理请求，然后Servlet1又转发给Servlet2来继续处理这个请求。下面用例子测试下：

**-----请求转发**

在AServlet中，把请求转发到BServlet：

**[java]** [view plain](https://blog.csdn.net/duyifei01/article/details/51935018) [copy](https://blog.csdn.net/duyifei01/article/details/51935018)

**public** **class** AServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

        System.out.println("AServlet");

        RequestDispatcher rd = request.getRequestDispatcher("/BServlet");

        rd.forward(request, response);

    }

}

**[java]** [view plain](https://blog.csdn.net/duyifei01/article/details/51935018) [copy](https://blog.csdn.net/duyifei01/article/details/51935018)

**public** **class** BServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

        System.out.println("BServlet");

    }

}

**结果：**

Aservlet

BServlet

**-----请求包含**

在AServlet中，把请求包含到BServlet：

**[java]** [view plain](https://blog.csdn.net/duyifei01/article/details/51935018) [copy](https://blog.csdn.net/duyifei01/article/details/51935018)

**public** **class** AServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

        System.out.println("AServlet");

        RequestDispatcher rd = request.getRequestDispatcher("/BServlet");

        rd.include(request, response);

    }

}

**[java]** [view plain](https://blog.csdn.net/duyifei01/article/details/51935018) [copy](https://blog.csdn.net/duyifei01/article/details/51935018)

**public** **class** BServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

        System.out.println("BServlet");

    }

}

结果：

Aservlet

BServlet

请求转发与请求包含比较

1.如果在AServlet中请求转发到BServlet，那么在AServlet中就不允许再输出响应体，即不能再使用response.getWriter()和response.getOutputStream()向客户端输出，这一工作应该由BServlet来完成；如果是使用请求包含，那么没有这个限制；

2.请求转发虽然不能输出响应体，但还是可以设置响应头的，例如：response.setContentType(”text/html;charset=utf-8”);

3.请求包含大多是应用在JSP页面中，完成多页面的合并；

4.请求请求大多是应用在Servlet中，转发目标大多是JSP页面

转发 留头不留体 （请求头 请求体）

包含 留头留体

最后来比较一下请求转发与重定向：

--- 请求转发是一个请求，而重定向是两个请求；

--- 请求转发后浏览器地址栏不会有变化，而重定向会有变化，因为重定向是两个请求；

--- 请求转发的目标只能是本应用中的资源，重定向的目标可以是其他应用；

--- 请求转发对AServlet和BServlet的请求方法是相同的，即要么都是GET，要么都是POST，因为请求转发是一个请求；

---重定向的第二个请求一定是GET

重定向由于是两次请求 所以会丢失request 不能使用request域进行传值

request域：

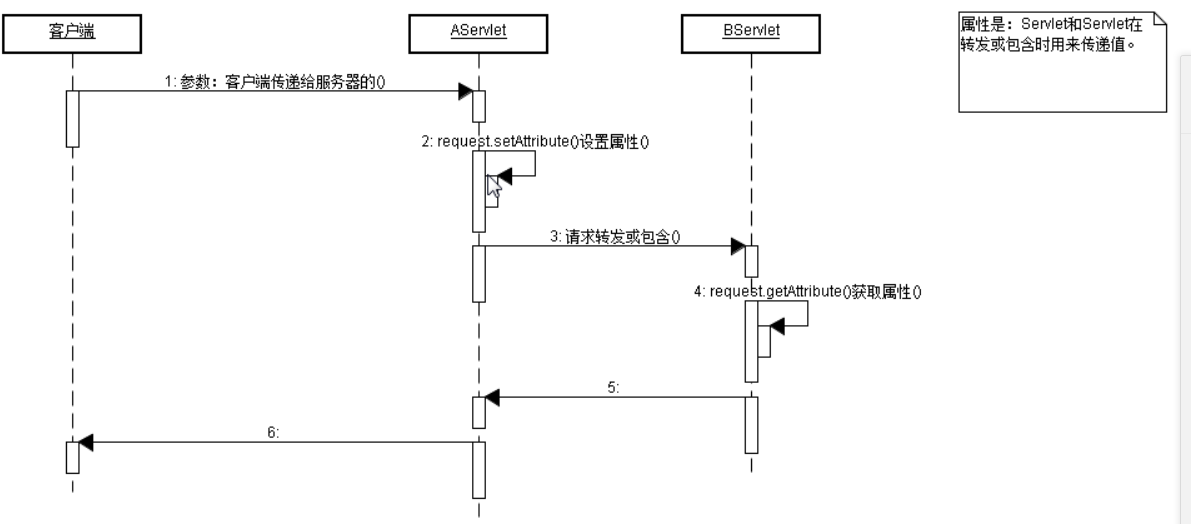
Servlet中的三大域对象： request session application 都下有三个方法：

void setAttribute(String name ,Object value);

Object getAttribute(String name);

void removeAttribute(String name);

在请求转发包含时 : 使用setAttribute getAttribute 来进行传值



请求转发 重定向的区别：

请求转发是一个请求一次响应 而重定向是两次请求

请求转发地址栏不会发生变化

请求转发只能转发到本项目其他Servlet 而重定向不只能重定向到此项目的地址

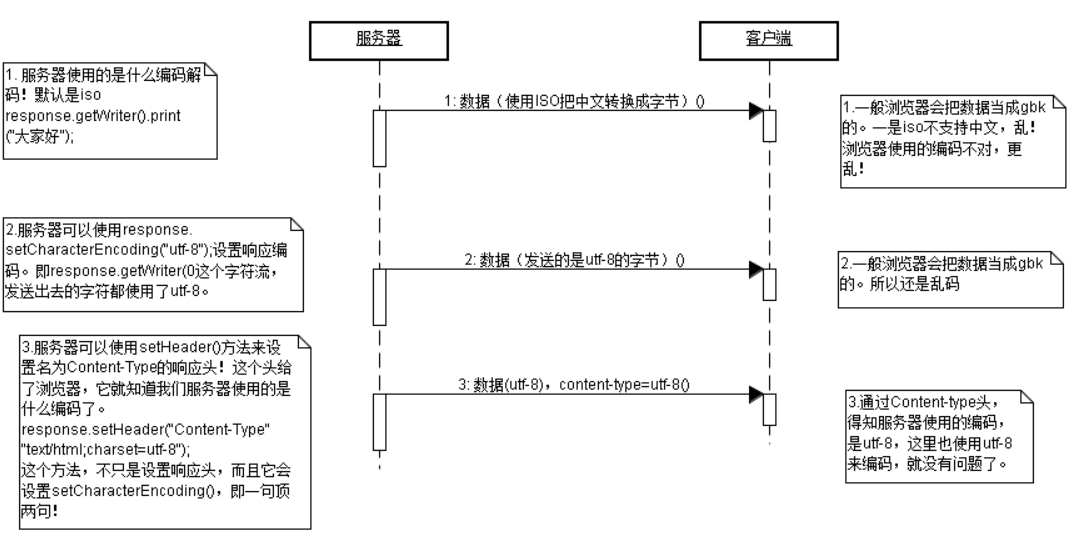
请求转发是服务器端的行为 之需要给转出的servlet路径 而重定向需要给出requestURI 即包命项目名

请求转发的效率高于重定向 因为只有一次请求

如果需要地址栏的变化 那么必须使用重定向

需要在下一个servlet中获取request域中的数据 必须要用请求转发！重定向的request在第一次请求后就会丢失

编码：



一定要 直接 response.setContentType(“text/html;charset=utf-8”)

想要不出现乱码 :

在使用getWriter()方法之前 先调用response.setContentType(“text/html;charset=utf-8”);

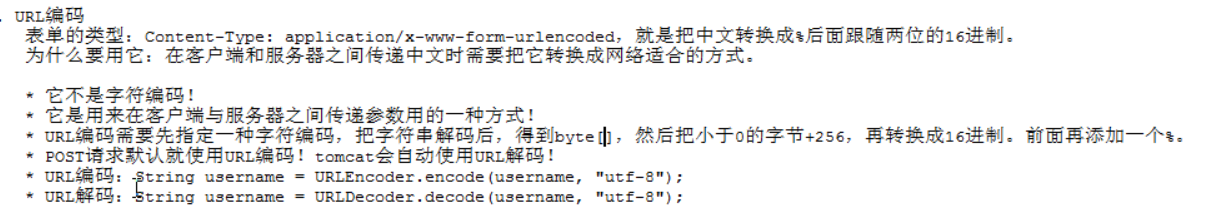
请求编码：

客户端传递参数的编码：

在页面上点击表单或者链接 大部分参数的编码都是 utf-8 因为大部分的页面都是 utf-8 的页面

在地址栏里面输入的参数 大部分都是gbk 但是 基本上没有人会在地址栏里面进行传递信息

URL编码：



路径：

web.xml里面配置的<url-pattern>路径 这个叫servlet路径

要么\*开头 要么/开头

转发和包含的路径：

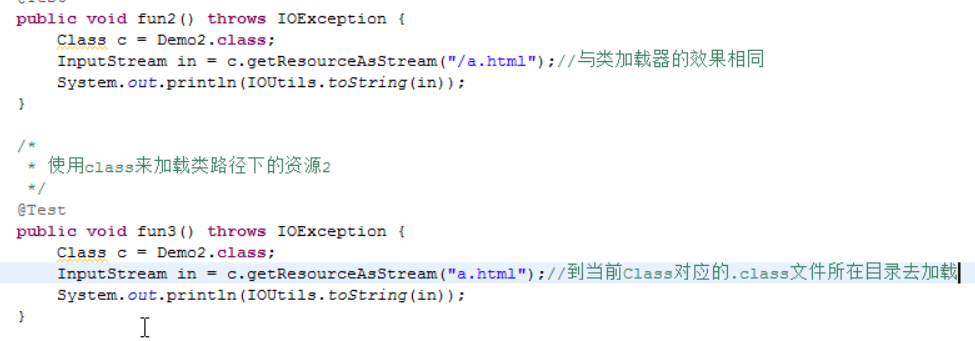
是服务器的路径 AServlet 到 BServlet 不会到客户端

以“/”开头 意为： <http://localhost:8080/项目名/>

request.getRequestDispatcher().forward(“/BServlet”)；

重定向路径：

客户端路径 : 要加上项目名 若以“/” 意为http://localhost:8080/…



/\* 浏览器只能看懂html

jsp

jsp其实就是servlet

Servlet  
 缺点：不适合配置html响应体

优点: 动态资源 可以编程

html

缺点： html是静态页面 不能包含动态信息

有点： 不能为输出html标签而发愁

jsp

优点： 在原有的html上添加java脚本 构成jsp页面

jsp servlet的分工：

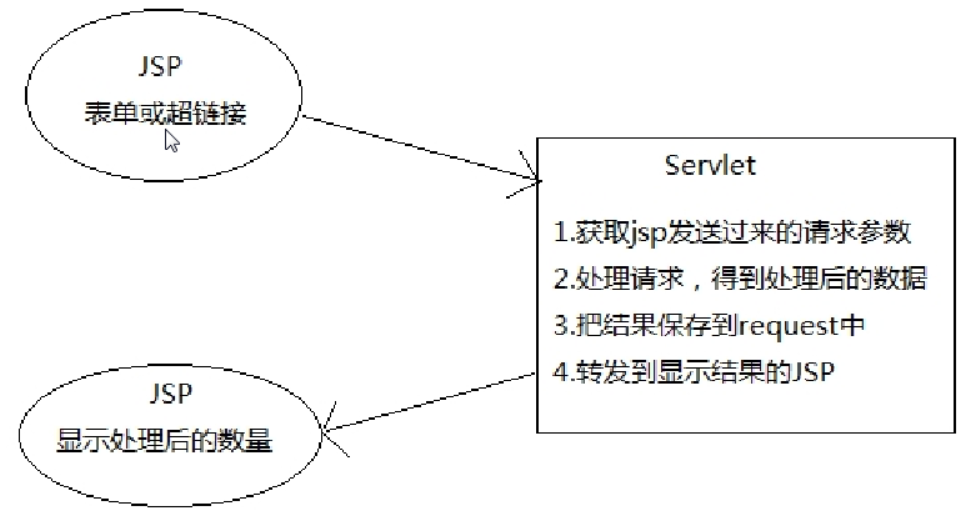
jsp：

作为请求发起页面 例如显示表单 超链接

作为请求结束页面 例如显示数据

Servlet：

作为请求中处理数据的环节



jsp其实就是html放上java代码

jsp中有 9 个对象 不用创建直接可以使用

request对象

request对象不用创建直接可以使用

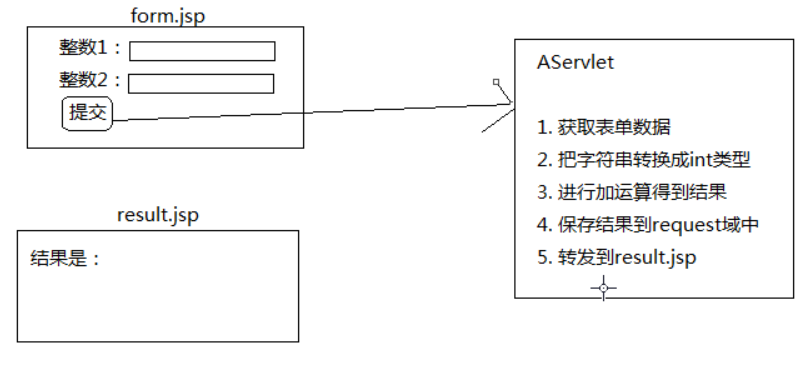
3种java脚本

<% 。。。。 %> java代码片段 用于定义0~N条java代码

<%! ….. %> java声明 用来创建类的成员变量 和 成员方法

<%= 。。。%> java表达式

演示jsp java脚本 servlet jsp分工合作：



JSP原理：

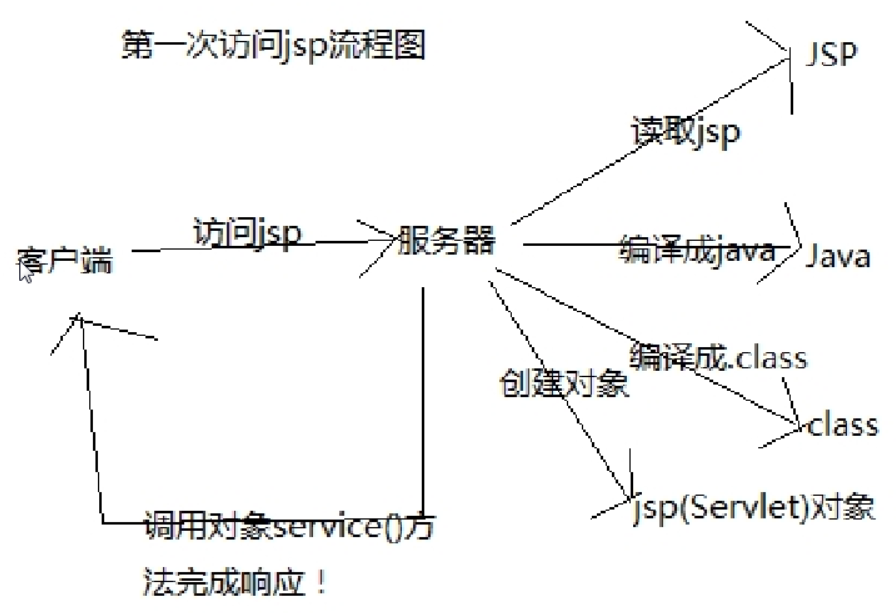
jsp其实就是一个Servlet

当jsp文件第一次访问时 服务器把jsp转化成java文件

然后把java编译生成.class

然后调用它的service()方法

第二次请求同一个jsp时 直接调用service()方法

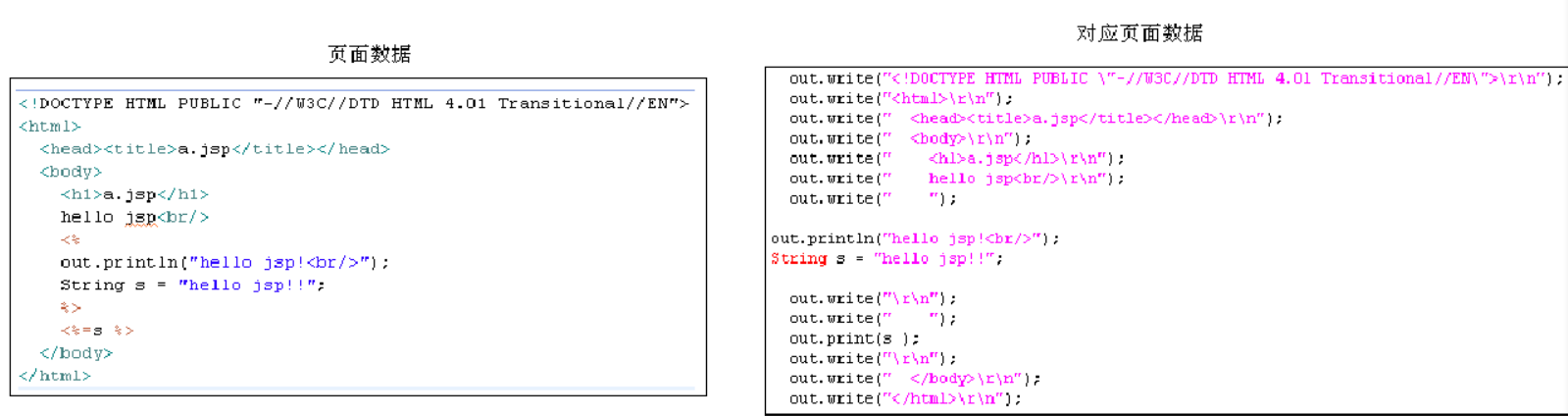


jsp转化为java文件：

会在头部为9个基本类型进行赋值

<% 里面的java语句会直接进行写入

<%= 里面的语句 调用out.print(之前的内容)



jsp中的注释 <%--- ---%>

翻译成.java 后不存在

jsp中的注释 <!—XXX -->

会传到到浏览器 在浏览器上不会显示 但是在浏览器上显示源代码上会显示

### Cookie

是服务器保存到客户端的东西

Cookie是http协议制定的 先由服务器保存Cookie到浏览器 在下次浏览器请求服务器时把上一次请求得到Cookie再归还给服务器

由服务器创建保存到客户端浏览器的键值对 服务器保存Cookie的响应头 ：Set-Cookie :aaa=XXX

当浏览器请求服务器时 会把该服务器保存的Cookie随请求发送给服务器 浏览器归还Cookie的请求头：Cookie：aaa=AAA;bbb=BBB (此处只返回一个头 用；隔开信息)

Http协议定义：

1个Cookie最大4kb

1个服务器最多可以向浏览器保存20个Cookie

1个浏览器最多保存300个Cookie

(大多数Cookie都违反了规定

Cookie不安全

服务器可以使用Cookie跟踪客户端状态

\*\*\*Cookie是不能跨浏览器的

Cookie的详解：

Cookie的maxAge : cookie的最大生命 cookie.setMaxAge(6)

maxAge>0 浏览器会把Cookie保存到硬盘上 有效时间 单位 1秒

maxAge=0 浏览器会马上删除这个cookie

maxAge<0 cookie只会在浏览器内存中存在 浏览器关闭时 cookie 被销毁

Cookie的path：

Cookie的path并不是设置这个Cookie客户端的保存路径

Cookie的path由服务器创建Cookie时决定

\*\* 浏览器访问服务器的路径 如果包含某个Cookie的路径 那么就会归还这个Cookie

例如：

aCookie.path = /day1/; bCookie.path=day1/jsp/;cCookie.path=/day1/jsps/cookies/;

访问/day1/index.jsp 时 归还： aCookie

访问/day1/jsps/index.jsp时 归还：aCookie,bCookie

访问/gay1/jsps/cookies/a.jsp 时 归还：aCookie,bCookie,cCookie

Cookie的域 domain：

domain用来指定Cookie的域名

当多个二级域中共享Cookie时才会有用

例如：[www.baidu.com](http://www.baidu.com) zhidao.baidu.com news.baidu.com 之间可以共享Cookie时可以使用domain

设置damain为：cookie.setDomain(“.baidu.com”);

设置path为：cookie.setPath(“/”);

设置path为/ 防止path被固定

###### HttpSession (保存在服务器端

HttpSession由Javaweb提供

用来会话跟踪的类

HttpSession是三大域之一

三大域：request session application

它们都有setAttribute getAttribute removeAttribute 三个方法

会话：会话范围是某个用户从首次访问服务器开始 到该用户关闭浏览器结束

会话：一个用户对服务器的多次连贯性请求 所谓连贯性请求 就是该用户多次请求中间没有关闭的服务器

获取session：HttpSession session = request.getSession();

session保存id :

一般会保存到cookie中

如果cookie被禁用 会保存到url ：href='/JavaWeb\_Session\_Study\_20140720/servlet/BuyServlet;jsessionid=96BDFB9D87A08D5AB1EAA2537CDE2DB2?id=3'

HttpSession 原理：

Session默认生命周期 ( 最大不活动周期 ) 30mins

服务器是如何实现一个session为一个用户浏览器服务的？

 　　服务器创建session出来后，会把session的id号，以cookie的形式回写给客户机，这样，只要客户机的浏览器不关，再去访问服务器时，都会带着session的id号去，服务器发现客户机浏览器带session id过来了，就会使用内存中与之对应的session为之服务。

配置session最大活动时间：

session用得最多的方法是：getAttribute setAttribute removeAttribute

Session的其他方法:  
 String getId() : 获取sessionId

int getMaxInactiveInterval() : 获取session可以得最大不活动时间 默认为30mins

void invalidate() : 让session失效！ 调用这个方法会被session失效 session失效后 客户端再次创建请求 服务器会返回一个新的session ( 这个方法 常常用在推出登陆

Boolean isNew() : 查看session是否是新的

图形验证码：

1. 创建图片缓冲区 设置宽高
2. 得到图片的绘制环境（得到画笔Graphics）
3. 保存图片

BufferedImage bi = new BufferedImage(70,35,BufferedImage.TYPE\_INT\_RGB);

Graphics g = bi.getGraphics();

g.setColor(Color.RED);

g.fillRect(0,0,70,35);

g.setColor(Color.WHITE);

g.drawString(“Hello”,2,2); // 其中2，2表示x,y轴的坐标

ImageIO.write(bi,”jpeg”,new FileOutputStream(“要存放的地址”);

JSP三大指令：

page ——》最复杂

include——》静态包含

taglib——》导入标签库

以下是page指令的属性： <%@page aaa=”XXX” (z这样就设置了一个方法)

pageEncoding：指定当前jsp页面的编码

contentType：它表示添加一个响应头：Content-Type！ 等同于response.setContentType(“text/html;charset=utf-8”);

如果两个属性只提供一个 那么另一个的默认值为设置的那一个

如果两个都没有设置 那么默认为iso

import 效果跟java import一样

errorPage ： 如果这个页面出错 转跳到指定的页面 errorPage=”b.jsp”

这个使用的请求转发 地址栏不变

isErrorPage 用来标注这个页面是处理错误的页面

isErrorPage=”true” \*\*\* 标注之后就可以在这个jsp里面使用Exception这个域对象

九个内置对象 不用获取直接可以使用

out jsp输出流 向客户端响应

page 当前jsp对象 Object page = this ;

config 对应真身ServletConfig

pageContext 一个顶 9 个

request HttpServletRequest

response HttpServletResponse

exception Throwable

session HttpSession

application ServletContext

pageContext

一个顶9个

servlet中的三大域 JSP有四大域

ServletContext：整个应用程序的范围

session ： 一个会话 一个用户一个回话 多个用户不是一个session

request ：一个请求连

pageContext ： 一个JSP页面 这个域实在当前的jsp页面 和当前的jsp页面中使用的标签之间共享数据

域对象

可以代理其他 8 个 域 ： pageContext.setAttribute(“aa”,”AA”,PageContext.SESSION\_SCOPE ) 这样就能把数据set到想要放进的域里

全域查找\*\*\*\* ： pageContext.findAttribute(“XXX”) 从小到大查找 request 开始查找 。。。

可以获取其他8个内置对象

## 什么是内置对象?

在jsp开发中会频繁使用到一些对象,如ServletContext HttpSession PageContext等.如果每次我们在jsp页面中需要使用这些对象都要自己亲自动手创建就会特别的繁琐.SUN公司因此在设计jsp时,在jsp页面加载完毕之后自动帮开发者创建好了这些对象,开发者只需要使用相应的对象调用相应的方法即可.这些系统创建好的对象就叫做内置对象.

在servlet程序中,如果开发者希望使用session对象,必须通过request.getSession()来得到session对象;而在jsp程序中,开发中可直接使用session(系统帮我们创建好的session对象的名字就叫session)调用相应的方法即可,如:session.getId().

可以直接调用 不用先得到

九大内置对象：

1. request        HttpServletRequest
2. response       HttpServletResponse
3. config         ServletConfig
4. application    ServletContext
5. session        HttpSession
6. exception      Throwable
7. page           Object(this)   当前页面对象
8. out            JspWriter   输出流
9. pageContext    PageContext    一个顶8个

## JSP中四大域对象

### 分类:

**[plain]** [view plain](https://blog.csdn.net/bighuan/article/details/68925746) [copy](https://blog.csdn.net/bighuan/article/details/68925746)

1. ServletContext     context域
2. HttpServletRequet  request域
3. HttpSession        session域     --前三种在学习Servlet时就能接触到
4. PageContext        page域     --jsp学习的

域对象的作用:保存数据,获取数据,共享数据.

### 保存数据:

**[plain]** [view plain](https://blog.csdn.net/bighuan/article/details/68925746) [copy](https://blog.csdn.net/bighuan/article/details/68925746)

1. pageContext.setAttribute("内容");//默认保存到page域
2. pageContext.setAttribute("内容",域范围常量);//保存到指定域中
3. //四个域常量
4. PageContext.PAGE\_SCOPE
5. PageContext.REQUEST\_SCOPE
6. PageContext.SESSION\_SCOPE
7. PageContext.APPLICATION\_SCOPE

### 获取数据:

**[plain]** [view plain](https://blog.csdn.net/bighuan/article/details/68925746) [copy](https://blog.csdn.net/bighuan/article/details/68925746)

1. pageContext.getAttribute("内容");

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

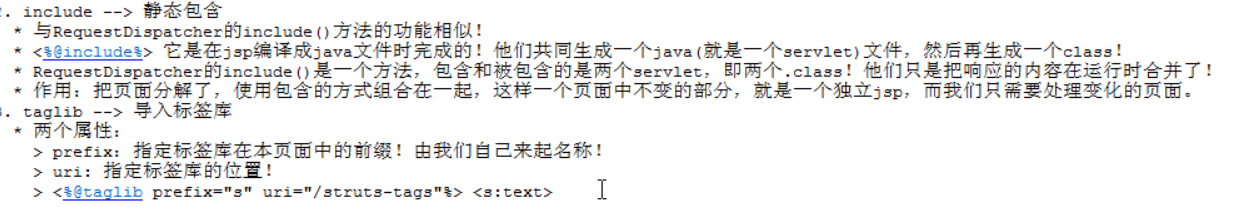
//自动在四个域中搜索数据 pageContext.findAttribute("内容");//在四个域中自动搜索数据,顺序:page域->request域->session域->application域(context域) 默认全域查找

这里 如果session还存在 就从session 里面进行查找

### 域作用范围:

**[plain]** [view plain](https://blog.csdn.net/bighuan/article/details/68925746) [copy](https://blog.csdn.net/bighuan/article/details/68925746)

1. page域:    只能在当前jsp页面使用                (当前页面)
2. request域: 只能在同一个请求中使用               (转发)
3. session域: 只能在同一个会话(session对象)中使用  (私有的)
4. context域: 只能在同一个web应用中使用            (全局的)



Session对象  
    （1）什么是Session对象 （验证登录状态）  
    Session对象是一个JSP内置对象，它在第一个JSP页面被装载时自动创建，完成会话期管理。从一个客户打开浏览器并连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。当一个客户访问一个服务器时，可能会在这个服务器的几个页面之间切换，服务器应当通过某种办法知道这是一个客户，就需要Session对象。  
    （2）Session对象的ID  
    当一个客户首次访问服务器上的一个JSP页面时，JSP引擎产生一个Session对象，同时分配一个String类型的ID号，JSP引擎同时将这换个ID号发送到客户端，存放在Cookie中（如果cookie被禁用 就存放在URL），这样Session对象，直到客户关闭浏览器后，服务器端该客户的Session对象才取消，并且和客户的会话对应关系消失。当客户重新打开浏览器再连接到该服务器时，服务器为该客户再创建一个新的Session对象。  
    （3）Session对象的常用方法  
    ● public String getId()：获取Session对象编号。  
    ● public void setAttribute(String key,Object obj)：将参数Object指定的对象obj添加到Session对象中，并为添加的对象指定一个索引关键字。  
    ● public Object getAttribute(String key)：获取Session对象中含有关键字的对象。  
    ● public Boolean isNew()：判断是否是一个新的客户。

invalidate()方法 可以销毁session （ 多用于注销）

Application对象  
    （1）什么时Application对象  
    服务器启动后就产生了这个Application对象，当客户再所访问的网站的各个页面之间浏览时，这个Application对象都时同一个，直到服务器关闭。但是与Session对象不同的时，所有客户的Application对象都时同一个，即所有客户共享这个内置的Application对象。  
    （2）Application对象的常用方法  
    ● setAttribute(String key,Object obj)：将参数Object指定的对象obj添加到Application对象中，并为添加的对象指定一个索引关键字。  
    ● getAttribute(String key)：获取Application对象中含有关键字的对象。

**application –> ServletContext：**

相同点：

其实servletContext和application 是一样的，就相当于一个类创建了两个不同名称的变量。

在servlet中ServletContext就是application对象。

大家只要打开jsp编译过后生成的Servlet中的 \_jspService()方法就可以看到如下的声明：

ServletContext application = null;

application = pageContext.getServletContext();

不同点：

两者的区别就是application用在jsp中，servletContext用在servlet中。

application和page request session 都是JSP中的内置对象，

在后台用ServletContext存储的属性数据可以用application对象获得。

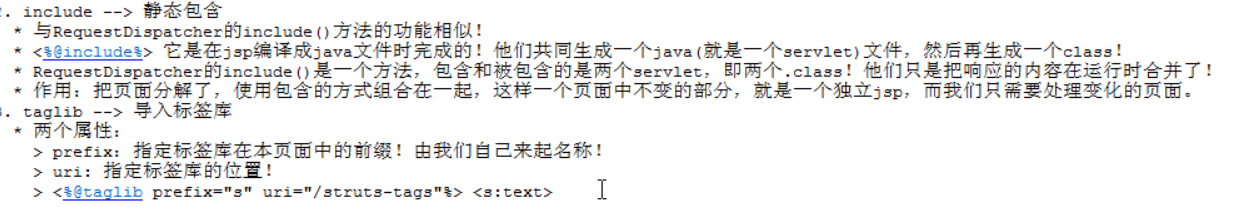
而且application的作用域是整个Tomcat启动的过程。

例如: ServletContext.setAttribute("username",username);

则在JSP网页中可以使用 application.getAttribute("username");

来得到这个用户名。

Out对象  
   Out对象时一个输出流，用来向客户端输出数据。Out对象用于各种数据的输出。其常用方法如下。  
    ● out.print()：输出各种类型数据。  
    ● out.newLine()：输出一个换行符。  
    ● out.close()：关闭流。



一、静态包含指令<%@include file=“fileurl”%>

1、两个jsp页面的<%@page contentType=“text/html；charset=gbk”%>应该保持一致

2、不能通过fileurl向被包含的jsp页面传递参数，因为此静态包含是发生在jsp页面转换为servlet的转换期间，此时的参数是服务器端设置的死的参数，完全没有经过客户端，这种参数是没有意义的，如<%@include  file=“fileurl?user=admin”%>，而且此时会报错。

3、包含的jsp页面与被包含的jsp页面共用一个request内置对象。

      比如说在客户端访问包含页面时地址栏后面直接加上参数后传递，这种形式的传参是客户端送来的，两个页面都能够访问此参数。我们可以通过这两个页面合成的servlet中可以看到有传递的参数成为servlet的成员变量。

4、包含的jsp页面与被包含的jsp页面最好没有重复的html标签。否则会发生覆盖现象。

二、动态包含<jsp :include page=“a.jsp”/>与静态包含<%@include  file=“fileurl”%>的区别

**1.**动态包含用的元素是page，而且有两种形式。静态包含用的是file,只有一种形式。

**2.**生成的文件不同，静态的包含是将两个jsp文件二合一，生成一个以包含页面命名的servlet和class文件，动态包含的两个jsp文件各自生成自己的servlet和class文件。

**3.** 传参方式一：<jsp:include page=“a.jsp?param=123”/>时被包含的jsp页面是可以访问该参数的。

**4.**传参方式二：

            <jsp:include page=“a.jsp”>

                     <jsp:param name=“” value=“”>

                    <jsp:param name=“” value=“”>

           </ jsp:include >

**5.**在客户端访问包含页面时地址栏后面直接加上参数后传递，这种形式的传参是客户端送来的，但是这两个页面的request对象不是同一个，因为**3中**已经说了包含的页面可以向被包含的页面传递参数，所以被包含的request对象含的参数个数应该大于等于包含页面的参数个数的。所以它们各有各的request对象。而且被包含的jsp页面可以访问传到包含页面的参数。

**6.**动态包含只有在执行到它的时候才加载，所以它才叫动态包含。

JSP动作标签：

这些jsp的动作标签 与html提供的标签有区别

动作标签是由 服务器 来解释执行！ 实在服务器端执行的！

html由浏览器来进行执行

<jsp:forward> : 转发 与RequestDispather 的forward 一样 ，只不过这个在jsp中使用

<jsp:include> : 包含 基本内容同上

jsp动作标签csdn详解：<https://blog.csdn.net/qq_29028175/article/details/53729048>

使用<jsp:useBean class=”bean地址” id=”bean名字”></jsp:useBean><jsp:getProperty name=”bean 的名字” property=”property的name”> 定义在useBean标签外 ： 不管有没有实例化bean都执行 要是定义在useBean标签里面 只有实例化了才会执行

JavaBean：

JavaBean的规范：  
 必须要有默认构造器

提供get/set 方法 （如果只有get方法 那么是只读属性）

属性：有getset方法的成员 还可以没有成员 只有get/set方法 属性名称由get /set 来决定 ！而不是成员的名称 （ getName(){ return username;）这里的属性名称是name 而不是username

方法名称符合规定

JavaBean的内省 ：

就是通过反射操作javabean’ 但他比使用反射要方便

BeanInfo info = Introspector.getBeanInfo(“反射路径”);

BeanInfo 可以得到 PropertyDiscritpor[]

BeanInfo 是一个 JavaBean的信息类

可以得到属性描述符对象 info.getPropertyDiscritpor()

PropertyDiscriptor.getReadMethod / getWriteMethod

内省类 – Bean信息 – 属性描述 – 属性get/set对应的Method – 可以反射

使用javabean的jar包beanutil可以省去很多代码

/\*java中class.forName()和classLoader都可用来对类进行加载。  
class.forName()前者除了将类的.class文件加载到jvm中之外，还会对类进行解释，执行类中的static块。  
而classLoader只干一件事情，就是将.class文件加载到jvm中，不会执行static中的内容,只有在newInstance才会去执行static块。  
Class.forName(name, initialize, loader)带参函数也可控制是否加载static块。并且只有调用了newInstance()方法采用调用构造函数，创建类的对象\*/

forName方法会将class加载到jvm中 会执行static代码

JSP中的javabean标签：

<jsp:useBean:>

<jsp:useBean id=”user1” class=”包名” scope=”page 指定的域”>

<jsp:setProperty>

<jsp:setProperty property=”username \*\* 这个是属性名” name=”user1 \*\*这是bean的名称” value=”XXX \*\*这个是username属性对应的属性值”> 在user1的bean里设置username = XXX

<jsp:getProperty property=”name” name=”user1” > 在user1 的bean里面查找name的属性值

EL表达式：  
 是jsp的内置语言

${request\_xxx }

当属性不存在时 返回空字符串 不是null

el表达式 可以做全域查找

${xx} 查找xx的属性值

也可以指定域来进行查找  
 ${pageScope.xx} 在page里面进行查找

JSP2.0开始不让使用Java脚本 而是使用el表达式 动态标签

el要替代的是<%= … %>

el只能做输出标签

${xxx} 全域查找 顺序先pageContext request session application（注意：如果session还存在那么还是在session中进行查找）

EL可以输出的东西 都在11个内置对象之中

JavaBean导航：、



**BeanUtils工具包：**

 　　由上述可看出，内省操作非常的繁琐，所以所以Apache开发了一套简单、易用的API来操作Bean的属性——BeanUtils工具包。  
　　BeanUtils工具包：下载：<http://commons.apache.org/beanutils/>，注意：应用的时候还需要一个logging包<http://commons.apache.org/logging/>  
　　使用BeanUtils工具包完成上面的测试代码：

**[java]** [view plain](https://blog.csdn.net/zhoudaxia/article/details/33783321) [copy](https://blog.csdn.net/zhoudaxia/article/details/33783321)

1. **package** com.peidasoft.instrospector;
3. **import** java.lang.reflect.InvocationTargetException;
4. **import** org.apache.commons.beanutils.BeanUtils;
5. **import** org.apache.commons.beanutils.PropertyUtils;
7. **public** **class** BeanInfoTest {
9. /\*\*
10. \* @param args the command line arguments
11. \*/
12. **public** **static** **void** main(String[] args) {
13. UserInfo userInfo = **new** UserInfo();
14. userInfo.setUserName("peida");
15. **try** {
16. BeanUtils.setProperty(userInfo, "userName", "peida");
17. System.out.println("set userName:" + userInfo.getUserName());
18. System.out.println("get userName:" + BeanUtils.getProperty(userInfo, "userName"));
19. BeanUtils.setProperty(userInfo, "age", 18);
20. System.out.println("set age:" + userInfo.getAge());
21. System.out.println("get age:" + BeanUtils.getProperty(userInfo, "age"));
22. System.out.println("get userName type:" + BeanUtils.getProperty(userInfo, "userName").getClass().getName());
23. System.out.println("get age type:" + BeanUtils.getProperty(userInfo, "age").getClass().getName());
24. PropertyUtils.setProperty(userInfo, "age", 8);
25. System.out.println(PropertyUtils.getProperty(userInfo, "age"));
26. System.out.println(PropertyUtils.getProperty(userInfo, "age").getClass().getName());
27. PropertyUtils.setProperty(userInfo, "age", "8");  // IllegalArgumentException
28. } **catch** (IllegalAccessException | NoSuchMethodException | InvocationTargetException e) {
29. e.printStackTrace();
30. }
31. }
32. }

运行结果：

**[java]** [view plain](https://blog.csdn.net/zhoudaxia/article/details/33783321) [copy](https://blog.csdn.net/zhoudaxia/article/details/33783321)

1. set userName:peida
2. get userName:peida
3. set age:18
4. get age:18
5. get userName type:java.lang.String
6. get age type:java.lang.String
7. 8
8. java.lang.Integer
9. Exception in thread "main" java.lang.IllegalArgumentException: Cannot invoke com.peidasoft.instrospector.UserInfo.setAge on bean **class**
10. 'class com.peidasoft.instrospector.UserInfo' - argument type mismatch - had objects of type "java.lang.String" but expected signature "int"
11. at org.apache.commons.beanutils.PropertyUtilsBean.invokeMethod(PropertyUtilsBean.java:2181)
12. at org.apache.commons.beanutils.PropertyUtilsBean.setSimpleProperty(PropertyUtilsBean.java:2097)
13. at org.apache.commons.beanutils.PropertyUtilsBean.setNestedProperty(PropertyUtilsBean.java:1903)
14. at org.apache.commons.beanutils.PropertyUtilsBean.setProperty(PropertyUtilsBean.java:2010)
15. at org.apache.commons.beanutils.PropertyUtils.setProperty(PropertyUtils.java:896)
16. at com.peidasoft.instrospector.BeanInfoTest.main(BeanInfoTest.java:32)
17. Caused by: java.lang.IllegalArgumentException: argument type mismatch
18. at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
19. at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
20. at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
21. at java.lang.reflect.Method.invoke(Method.java:483)
22. at org.apache.commons.beanutils.PropertyUtilsBean.invokeMethod(PropertyUtilsBean.java:2116)
23. ... 5 more

　　说明：

1. 获得属性的值，例如，BeanUtils.getProperty(userInfo, "userName")，返回字符串。  
　　2. 设置属性的值，例如，BeanUtils.setProperty(userInfo, "age", 8)，参数是字符串或基本类型自动包装。设置属性的值是字符串，获得的值也是字符串，不是基本类型。

3. BeanUtils的特点：  
　　1). 对基本数据类型的属性的操作：在WEB开发、使用中，录入和显示时，值会被转换成字符串，但底层运算用的是基本类型，这些类型转到动作由BeanUtils自动完成。  
　　2). 对引用数据类型的属性的操作：**首先在类中必须有对象，不能是null，例如，private Date birthday=new Date();。操作的是对象的属性而不是整个对象**，例如，BeanUtils.setProperty(userInfo, "birthday.time", 111111);

**[java]** [view plain](https://blog.csdn.net/zhoudaxia/article/details/33783321) [copy](https://blog.csdn.net/zhoudaxia/article/details/33783321)

1. **package** com.peidasoft.Introspector;
2. **import** java.util.Date;
4. **public** **class** UserInfo {
6. **private** Date birthday = **new** Date(); // 引用类型的属性，不能为null
8. **public** **void** setBirthday(Date birthday) {
9. **this**.birthday = birthday;
10. }
11. **public** Date getBirthday() {
12. **return** birthday;
13. }
14. }

**[java]** [view plain](https://blog.csdn.net/zhoudaxia/article/details/33783321) [copy](https://blog.csdn.net/zhoudaxia/article/details/33783321)

1. **package** com.peidasoft.Beanutil;
3. **import** java.lang.reflect.InvocationTargetException;
4. **import** org.apache.commons.beanutils.BeanUtils;
5. **import** com.peidasoft.Introspector.UserInfo;
7. **public** **class** BeanUtilTest {
8. **public** **static** **void** main(String[] args) {
9. UserInfo userInfo=**new** UserInfo();
10. **try** {
11. BeanUtils.setProperty(userInfo, "birthday.time","111111");  // 操作对象的属性，而不是整个对象
12. Object obj = BeanUtils.getProperty(userInfo, "birthday.time");
13. System.out.println(obj);
14. }
15. **catch** (IllegalAccessException e) {
16. e.printStackTrace();
17. }
18. **catch** (InvocationTargetException e) {
19. e.printStackTrace();
20. }
21. **catch** (NoSuchMethodException e) {
22. e.printStackTrace();
23. }
24. }
25. }

　　3. PropertyUtils类和BeanUtils不同在于，运行getProperty（取出来是object类型）、setProperty操作时，没有类型转换，使用属性的原有类型或者包装类。由于age属性的数据类型是int，所以方法PropertyUtils.setProperty(userInfo,"age", "8")会爆出数据类型不匹配，无法将值赋给属性。

EL 可以输出11个内置对象  
 其中10个是map=（键值对应） pageContext不是map 就是pageContext

param:对应参数 是一个map key参数名 value是单个参数值

paramValues:对应参数 是一个map key参数名 value是多个参数值

header：对应请求头 是map key是头名称 value单个头值 适用于单只请求头

headerValues：对应请求头 是map key是头名称 value多个头值 适用于多只请求头

initParam： 获取<context-param> 里面的参数

\*cookie：Map<String ,Cookie> 类型 其中key是cookie的name value是cookie的对象

\*\* 这里的${cookie.username(指定是那个cookie，此处返回cookie对象 ).value(\*\*必须加value才是返回的参数值}

\*pageContext : ${pageContext.request.contextPath} 获取request里面的项目名 /day01

\*\*\*以后的href action都要用${pageContext.request.contextPath}这个el语句 以防项目名的变动

或使用： <c:url value=”/页面名”/>

el函数库：

导入标签库： <%@ taglib prefix=”fn”uri=<http://java.sun.com/jsp/jstl/function>%>

String toUpperCase(String input) : input转化为大写

String toLowerCase(String input) :input转化为小写

int indexOf(String input ,String substring) :返回substring在input里面的位置 没有的话返回-1

boolean contains(String input ,String substring) :input是否包含substring

boolean containsIgnoreCase(String input ,String substring) :input是否包含substring（忽略大小写）

Boolean startsWith(String input ,String substring) :input是否是substring开头

Boolean endsWith(String input ,String substring) :input是否是substring结束

String substring(String input ,int beginIndex ,int endIndex) :input中从beginIndex截取到endIndex（留头不留尾）

String substringAfter(String input ,String substring) :获取input中substring后面的字符

String substringBefore(String input ,String substring) :获取input中substring前面的字符

String escapeXml(String input) :把input中的’<’,’>’,’&’,’ ’ ’,’ ’进行转译

String trim(String input) :去除前后空格

String replace(String input , String substringBefore , String substringAfter) :在input中用substringAfter替换substringBefore

String[] split(String input ,String delimiters) :用delimiters分割input 返回字符串数组

int length(Object obj) :返回长度 对象是Object

String join(String array[] ,String separator) :用separator连接array[]

taglib ———标签库

cookie的值是cookie类型 el语法中 ${cookie.XX.value} 才能去除object对象

JSTL标签语言：

JSTL是EL表达式的扩展

使用jstl需要导入jar包

导入标签

jar包

在jsp页面中 <%@taglib prefix=”前缀” uri=”路径”%>

core核心标签库： 习惯用c作为前缀（要导入<http://java.sun.com/jsp/jstl/core>这个jar）

out 和 set

<c:out value=”@{aaa(表示全域查找aaa)}”

<c:out value=”@{aaa}” default=”xxx”/> 全域查找 没有就输出xxx

out的作用是输出

value：可以使字符串敞亮 也可以是el表达式

default：若要输出的内容为null 会输出default指定的值

set 设置属性值

var：变量名

value：变量值 可以使el表达式

scope：域 默认为page 可选值：page request session application

<c:set var=”a” value=”hello”/> 默认存在page域

<c:set var=”x” value=”xa” scope=”session”/>

remove

在域里面删除对象

var 变量名

scope 如果不给出scope 删除所有域的变量

<c:remove var=”a”/> 删除所有域里name为a的属性

url

value：指定一个路径 它会在路径前面自动添加项目名

<c:url value=”/index.jsp”/> 输出：/day1/index.jsp

子标签：<c:param> 给url后面添加参数

<c:url value=”/index.jsp”>

<c:param name=”username” value=”lxa”/>

</c：url> 这样就能传入参数 可以为参数做url编码

<a href=”<c:url value=”/index.jsp”/>”>点击这里回到主页</a>

两个标签不是一种 一个是浏览器执行 一个是服务器执行

if

标签属性必须是一个boolean值

<c:if test=”布尔类型” > 这里是执行语句… </c:if>

choose

<c:set var=”score” value=”@{param.score}”/>

<c:choose>

<c:when test=”${score=10}” > 分数是10 </c:when>

<c:when test=”${score=20}” > 分数是20 </c:when>

<c:otherwise>不及格</c:otherwise>

</c:choose> 类似于if else

forEach （只能循环域中的对象）

用来循环遍历数组集合

还可以用计数的方式循环

var 循环变量

begin 设置循环变量从几开始

end 设置循环变量到哪里结束

step 步长 等同于 i+=X 默认值为1

<c:forEach var=”i” begin=”1” end=”10” step=”2(步长)”>

${i}

</c:forEach>

用来输出数组 集合

<c:forEach items=”@{strs}” var=”str”>

${str }

</c:forEach> strs是数组或者集合  
 等同于：

for(String str:strs){}

循环状态

使用varStatus创建循环变量

属性： count 循环元素个数

index 元素下表

first 是否是第一个元素

last 是否是最后一个元素

current 当前元素

FMT：

格式化标签库

<fmt:formatDate 用来格式化如期

<fmt:formatDate value=”指定一个date类型变量” pattern=”yyyy-mm-dd HH:mm:ss”/>

格式化数字：

<fmt:formatNumber value=”” pattern=”0.00(表示要去小数点后两位 并且四舍五入 不足的 补 0”/>

<fmt:formatNumber value=”” pattern=”#.##(表示取两位 四舍五入 但是不足的 不补0”/>

自定义标签 ：

要减少使用<%。。。。%> 这样的代码

自定义标签的步骤：

标签处理类（标签也是一个对象 需要有类）

tld文件 是一个xml

页面中使用<%taglib%> 来制定tld的位置

标签处理类：（创建自定义标签之后继承simpleTag类）

simpleTag接口：

void doTag 每次执行标签都会调用这个方法

JSPTag getParent 返回父标签（非生命周期方法）

void setParent(JspTag) 设置父标签 void setJspBody(JspFragment) 设置标签体

void setJspContext(JspContext) 设置jsp上下文对象 它儿子是pageContext

所有setXXX方法都在doTag之前调用 所以在doTag里面都可以使用

其中doTag 会在其他三个方法之后被tomcat调用

eg:

public class MyTag implement SimpleTag{

private PageContext pc;

private JspFragment jf ;

public void doTag(){

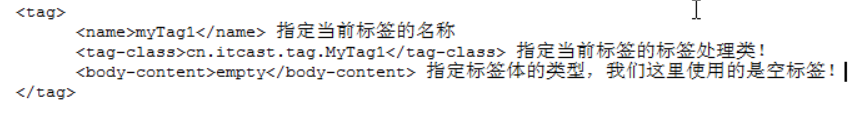
PageContext.getOut().print(“Hello!”);

}

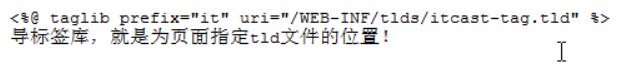
public

tld文件的配置：

一般放在WEB-INF之下 保证客户端访问不到（web-inf被称为安全目录 在浏览器里不能进行访问 ）



页面中指定tld文件的位置 一个tld文件里面可以部署多个标签



SimpleTagSupport 实现了SimpleTag接口 把tomcat传的值已经配置好了

直接重写doTag就可以 在this.get…. 来调用配置好的参数等等

simpleTagSupport使得自定义接口更加简单便捷

继承simpleTagSupport

有标签体的内容：

empty： 表示没有标签体

scriptless ：只能是el表达式 也可以是其他标签

不执行标签下内容的标签（是下面所有的标签都不会进行执行）

在标签处理类中的doTag 中使用SkipPageException来结束！

tomcat会调用标签处理类的doTag方法 得到exception会跳过其他内容

自定义标签 --- 有属性的标签

<c:if test=”test是boolean类型”>

\*步骤：

给标签处理类定义属性

给tld文件进行属性配置

public class … extends simpleTagSupport {

private Boolean text ；定义属性

public void setTest(boolean test){

this.getJspBody().invoke(null); (如果使用的输出流是null 表示使用的就是当前页面的out

}

为标签处理类中添加属性

为标签处理类添加属性 属性至少有一个set方法 这个set方法会在doTag方法之前被tomcat执行 所在的doTag方法就能够使用属性了

给tld文件进行属性配置

在<tag>标签下添加字标签：

<attribute>

<name>test</name> 属性名必须一致

<required>true</required> 是否必须出现属性值

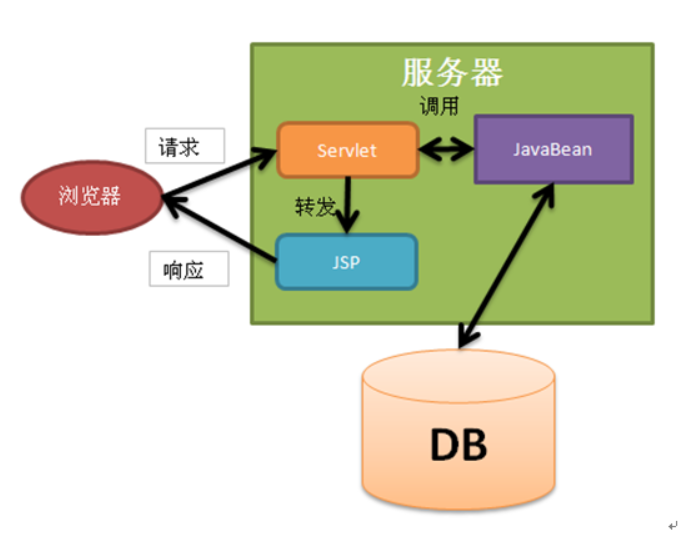
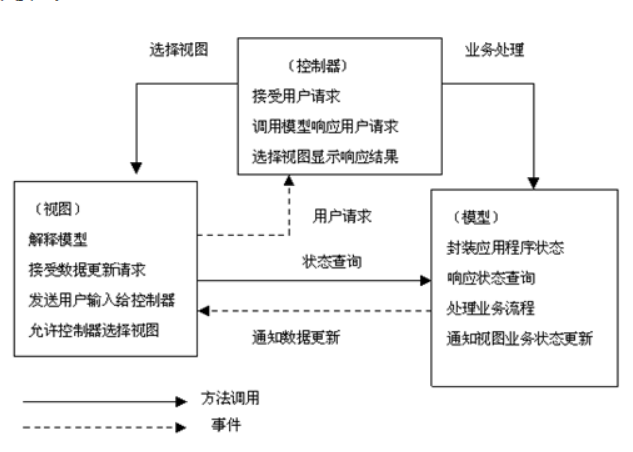
<rtexprvalue> </rtexprvalue>在使用标签时能不能使用表达式来动态指定数据 （可能是类似el语句之类的定西）

</attribute>

MVC:设计模式（模型视图控制器）

应用广泛 所有BS项目都在使用

相互分离 互不相干 设计模式

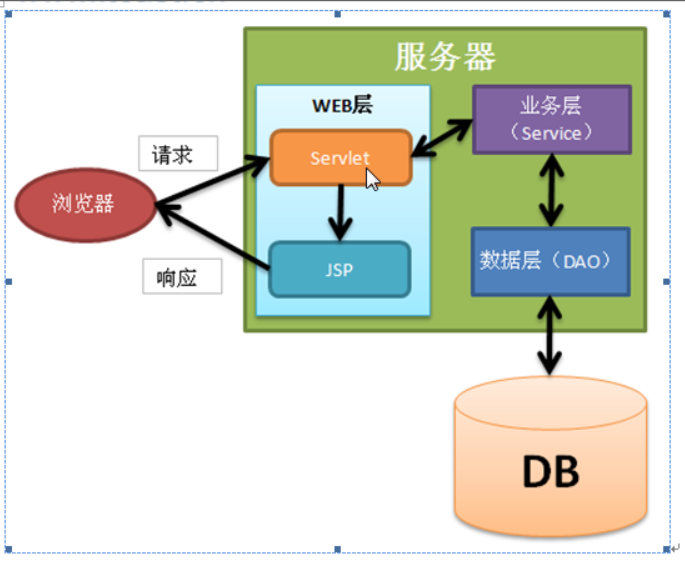


JavaWeb三层架构

web层 ：与web相关的内容(Servlet JSP Servlet相关的API)

业务层 ：业务对象 service

数据层 ：操作数据 又叫DAO dataaccessobject



MySQL：

use XXX（数据库名称 进入数据库）

SQL语句：

ddl 数据库 表 的结构操作

dml 对表的记录进行更新（增删改）

dcl 对用户的创建 授权

DDL:

查看所有数据库 show databases（复数）

使用数据库 use XXX

创建数据库 create database [if not exists] XXX [charset=utf8] [里面的内容是可选的]

删除数据库 drop database [if exists] XXX

修改数据库编码 alter database XXX character set utf8 更改为utf8

数据类型：

int 整数

double 浮点数 double(5,2)表示最多五位数 有两位小数

decimal 浮点数 不会出现精度缺失 （多用于金钱）

char 固定长度的字符串类型 char(25) 不足25则会自动补全

varchar 可变长度的字符串类型 varchar(25) 不足25的不会自动补全

char 由于是 固定的 所以使用比varchar方便

text 字符串类型 容量大

blob 大字节类型

data 日期类型 格式为 YYYY-MM-DD

time 时间类型 格式为 hh:mm:ss

timestamp 时间戳类型

表：

创建表 （前提是先要进入一个数据库）

create table [if not exists] 表明

查询表 desc XXX表明

修改表：

添加列 ：

alter table XXX

add( name varchar(50) ); 添加一条列

更改列属性：

alter table XXX

modify name varchar(10) ;修改为10长度

修改表名 ：

alter table XXX rename to NEWXXX

查询表记录

select \* from

插入信息：

insert into XXX表名 (name , age , sex//这里是table里有的数据名称) values(‘lxa’,21,’nan’);

insert into XXX values(‘lxa’,21,’nan;); 也可以不写数据名称 但要一一对应 不建议这样写 可读性降低

插入时 可以不指定所有列 其他为默认值

修改表：一定不要忘了 中间哟个set

update XXX set age=age+1 , id=1 where name=’LXA’ or id=111 ; where 关键字的使用

update XXX set age=age+1 where age between 18 and 40 ;

update XXX set age=age+1 where age>=18 and age<=40 ;

只要 数值 = null 用来筛选null 要用is null 不要用where XXX=null 用 where XXX is null ；

删除记录：

delete from XXX where（按条件删除） name is null ；

delete from XXX ； 全删了 一定要加where 不然全删了

DCL

一般情况下 一个项目创建一个用户 一个项目对应一个用户（这个用户只能对这个数据库有权限 其他项目不归他管）

create user 用户名@IP地址 identified by ’密码’；

用户只能在指定的IP地址上登录

create user 用户名@’%’ identified by ‘密码’；

表示可以在任意ip下登录

给用户授权：

grant 权限1，权限2，权限n on 数据库名称 to 用户名@ip地址

grant all(表示把所有权限都给) on DATABASE.\*(表示有所有数据库权限) to USER ;

权限： create alter drop insert update delete select

撤销权限：

revoke 权限1 权限2 权限n on DATABASE from USERNAME@IP地址 ；

revoke all on mydatabase from root@loaclhost ；

查看权限：

show grants for USERNAME@IP地址；

删除用户：

drop user USERNAME@IP地址；

基本查询（列控制）：

select \* from 表名 ；

表示查询所有。

select name ，age from table ； 指定查询

select distinct name from table ； distinct 表示不显示重复的数据

\*\*\* select \*,age+10 from table ; 表示在列出所有后 在加上一行age+10后的数据

任何类型和null相加 都变成null

select \*,age+ifnull(数据的名称,0/要变成的值) from table；

select concat(‘我是’,name,’今年’，age，‘岁。’) from table;

concat用于连接字符

select name as 姓名，job as 工作 from table ；

给列的名字 起别名（这里可以省去as）

\*\* select name xingming , age from table ； 这样执行不会出错 不是少了 逗号 是理解为起了别名

模糊查询： （关键字 like \_表示一个字符 %表示0~n个字符）

select \* from list where name like ‘Liu\_’; 模糊查询 表示含有Liu 后面是一个字

select \* from list where name like ‘Zheng\_\_’ 两个下划线 表示Zheng后面有两个字符

select \* from list where name like ‘%Ang’ ； 表示匹配0~n个字符

select \* from list where name like ‘Liu%’ 表示匹配0~n个字符

select \* from list where name like ‘%小%’ 表示查询 有 小 字的元素

排序： （关键字 order by） asc升序排列 desc降序排列

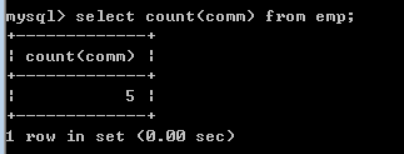
select \* from list ORDER BY age ASC； 表示进行升序排列

select \* from list ORDER BY age ASC ,sal DESC , comm ASC ; 表示 在 第一个条件相同时 用逗号后面的第二个条件进行 比较 ， 再次相同时 用逗号后面的再次进行比较 以此类推。

\*\* net stop mysql ; 停止mysql

聚合函数：(纵向查询)

select count(comm//要查询的列) from list ; 表示comm项里面不为null的都进行计数



select sum(comm) from list ; 计算comm的和

select min(comm) from list ；找出comm的最小那个 显示

select avg(comm)from list ;计数comm的平均值 null当做0。

分组查询：（关键组 group by）

根据group by后面的元素（分组列） 进行分组

关键字： having 后面跟 分组后条件

分组前的条件用 where

ifnull: ifnull(XX,’none’) 如果XX为null 在表中显示为 none

limit关键字（只在mysql可以使用）用于进行分页查询

select \*from list limit 0,5; 表示从0行查（类似于下标 都要+1） 查5行

分页查询： 大量数据不能一下全显示 类似百度 会进行分页查询

若； 每页8行记录 要查17页的记录

limit (17-1)\*8 , 8 ;

Mysql编码：

show variables like ‘char%’ 显示字符类型

character\_set\_client=utf8 这说明不论发送什么编码 mysql都当成utf8对待

若 编码不同意 会报错

每次设置编码之后 再次打开还是原来的状态。。

解决方法\*\* ： 在总配置文件里面解决

my.ini default-character-set=gbk 进行设置 （主要解决 client result的编码问题）

Mysql的备份 恢复：（命令）

mysqldump –u用户名 –p密码(已经登陆了就不需要) mydb3>c:a.sql (mydb3数据库名 c:a.sql 传出路径 后缀必须是.sql

\*\*\* 恢复数据库：(前提时已经进行了数据库的存储 格式为.sql)

想要恢复之前drop的内容

* 1. 先退出
  2. mysql –uroot –p123 mydb<c:/a.sql
  3. 这样就导入了之前存储的数据库

source c:/a.sql 这样使用source语句也可以恢复 （导入）

约束：  
 主键（primary key）： 非空 唯一 被引用

主键不能为空

通过主键 引用

主键的值不能重复

如何在修改表时 设置主键 ：

alter table {

add添加

modify修改

change修改

drop删除

rename to 改名}

alter table XXX add primary key(要设置的元素名)；

删除主键： alter table XXX drop primary key;

主键自增长：

主键必须 非空 唯一

name INT PRIMARY KEY AUTO\_INCREMENT 自动增长

概念模型：  
 对象模型：在java里是domain

关系模型： 在数据库中表！

概念模型在java中成为实体类（javaBean）

外键约束：（外键必须约束主键）

外键必须是另一个表的主键

外键可以重复 外键可以为空

一张表中可以引用多个外键

也可以在同一张表上进行约束

--外键约束：

alter table 表名

add constraint FK\_字段名--"FK"为外键的缩写

foreign key (字段名) references 关联的表名(关联的字段名) --注意'关联的表名'和'关联的字段名'

--删除创建的约束:

alter table 表名

drop constraint 约束名--约束名为你前面创建的如：PK\_字段这样的约束名

--注意：如果约束是在创建表的时候创建的，则不能用命令删除

--只能在'企业管理器'里面删除

一对一关系：  
 可以通过 两个表的主键 进行外键对应 形成1对1 关系

从表的主键 是外键

多对多关系：（与要用到 中间表）

创建 学生表 create table stu ( sid int primary key , name varchar(10) ) ;

create table tea (tid int primary key ,name varchar(10));

创建关联表（有两个外键）：\*\* 所有关系都要在这个表 里面体现

使用关联表 创建多对多关系

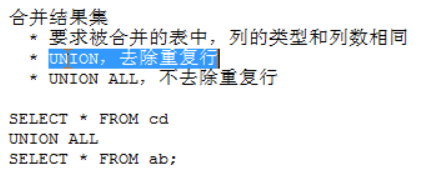
create table mid(t+ sid int ,tid int ,

constraint fk\_stu foreign key(sid) references stu(sid)；

constraint fk\_tea foreign key(tid) references tea(tid)；);

多表查询：

合并结果集： 就是一个表格 （ 要求两个表 列数 列类型 \*完全相同 ）

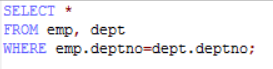


连接查询： 一次查询多个表

内连接： 会出现许多 没用的数据 （叫做 笛卡尔积）

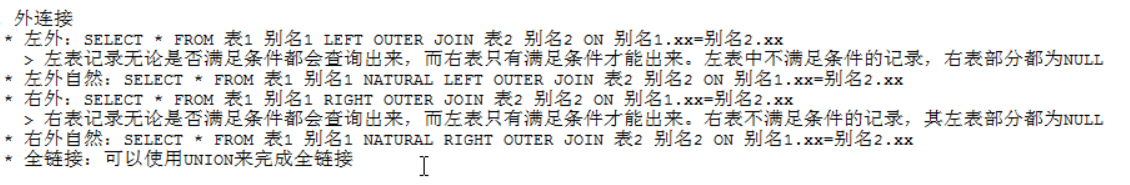
\*\* 可以用 where 语句 去除没有用的信息 （去除 笛卡尔积）\*要打表名的前缀

也可以 给表起别名



外链接：

select \* from XXX1 natural join XXX2 ; （自然内连接：可以省去写条件 直接去除笛卡尔积 但是可读性降低）



子查询：

例如：在本公司查询工资最高的员工信息 （在where中 不能有函数语句）

select\*from XXX where sal=( select max（sal） from XXX);

也可以 select \*from (select from XXX where sal=1000) where name=XXX;

例如：找出工资高于平局工资的人

select\*from XXX where sal>(select avg(sal) from XXX(这个表时单行单列));

如果是多行单列的 被查询结果 时： 可以使用ALL ANY

单行多列： select \* from XXX where (job,sal) in (select job,sal from XXX where name=’lxa’); 通过获取lxa 的sal job 列表 获取XXX表格里面相同的对象

一张表也可以自身链接。

#### JDBC：使用java代码向服务器发送sql语句：

链接步骤：

1. 导入jar包
2. 加载驱动类： Class.forName(“类名”);
3. 给出url username password
4. 使用diverManager类得到connection对象

jdbc四大配置参数：

driverClassName : com.mysql.jdbc.Driver

url : jdbc:mysql://localhost:3306/XXX 要使用的数据库名称

username: 用户名

password:密码

driverClassName = com.mysql.jdbc.Driver

url = jdbc:mysql://localhost:3306/数据库名称

加载驱动类 Class.forName(com.mysql.jdbc.Driver);

Connection conn=DriverManager.getConnection(“jdbc:mysql://localhost:3306/XXX”,username,password);

\*\* 对数据进行增删改

通过connection 对象创建声明

\*\* statement语句向数据库发送sql语句

Statement sta = conn.createStatement();

创建sql语句：

String sql = “insert into XXX values(‘lxa’,21)”;

\*\*\* sql语句不用加;

ResultSet 结果集

ResultSet rs = sta.executeQuery(sql);

ResultSet有一个行光标

next()方法可以把行光标向下移动一行



next方法返回一个boolean类型 表示 当前行是否存在

ResultSet 提供了许多 getXXX方法：

\*\*\*\*\*\*\* 使用结束之后 一定要 将ResultSet Statement Connection 关闭

DriverManager：

可能出现的异常：

ClassNotFoundException：没有导入jar com.mysql.jdbc.Driver打错了

Statement 方法：

executeQuery 查询 返回 ResultSet结果集

executeUpdate 更新 返回int类型 说明执行了几行

execute 方法可以执行所有sql语句 返回boolean 不会直接返回结果集（少用）

ResultSet方法：

next 向下移动一行

last 移动到最后一行

getRow 返回int 这是第几行

beforeFrist 移动到第一行的上一行

absolute（int ow） 移动到X行 返回boolean 是否成功

获取结果集数据：

得到元素据：rs.getMetaData() 返回ResultSetMetaData；

获取结果集列数： int getColumnCount() rs.getMetaData().getColumnCount();

获取指定列的列名:Sting rs.getMetaData().getColumnName(int columnIndex);

PreparedStatement

防止sql攻击

提高可读性 可维护性

提高效果

学习PreparedStatement：(用问好替换)

给出sql模板

PreparedStatement pstmt = conn.prepareStatement(sql);

pstmt.setSting/setInt=XXX;

调用pstmt的executeQuery executeUpdate 都没有参数

jdbcUtils:

作用：总是进行获取连接 过于繁琐

将参数放入配置文件

加载配置文件

static{

File f = new File();

Properties p = new Properties();

p.load(f);

Class.forName(p.getProperty(“键名”));

}

只在第一次加载时初始化 防止多次加载造成的浪费

return 🡪DriverManager.getConnection(jdbc:mysql://localhost:3306/XXX,username,password);

DAO模式：（感觉就是往麻烦了设计。。）

将访问数据库的代码封装起来

通过配置文件 反射 接口等

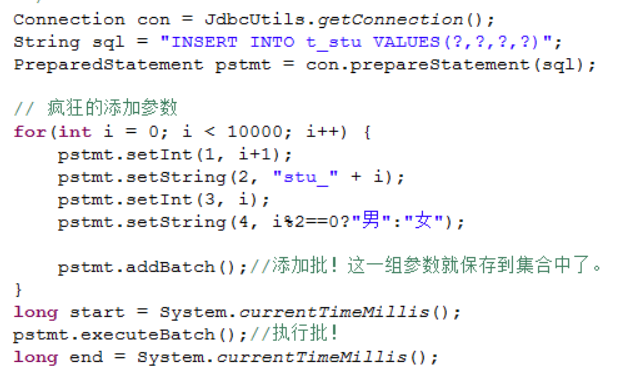
大数据;

把mp3保存到数据库

批处理：

用循环向pstmt添加sql语句

调用pstmt的执行方法向数据库发送



主要语法： addBatch 添加

executeBatch 执行

Batch——批处理

事务： ACID （eg：转账系统 安全）

原子性：不可再分割 一起生一起死

一致性：无论成功与否数据库与业务保持一致

隔离性：并发操作不会产生影响 保持一致

持久性：数据到达数据库之后不能出问题 （如果数据达到数据库 数据库崩溃 数据也能恢复）

mysql中操作事务：

关键字：start transaction ; 事务开始

commit ; 事务提交

rollback ; 回滚 （事务回滚能恢复到事务开始的状态 但是如果事务commit 就不能回滚 因为事务成功提交了）

JDBC中处理事务：

在JDBC中处理事务 都是通过connection完成的

connection的三个事务相关方法：

setAutoCommit(boolean) 设置是否为自动提交事务 如果为true（默认为true）表示自动提交 也就是每执行sql语句都是一个单独的事务 如果设置为false 那么相当于开启了事务

Connection.setAutoCommit(false)； 表示开启事务

commit();表示提交事务

rollback(); 回滚事务 如果没有commit 那么可以回滚到执行事务之前的状态



同一个事务必须要在同一个connection操作

在分离式DAO时 一定要要同一个Connection （通过传递Connection保持一致）

sql JAR包的东西都要出现在DAO业务中

把所有对Connection的操作隐藏起来

事务的隔离级别：

事物的并发读问题：

脏读： 读取到另一个事务未提交的数据，读取到了脏数据

不可重复读：对同一个数据连续两次的读取不一致 因为中间另一个事务做了修改

幻读（虚读）： 对同一个表两次查询不一样 因为另一个事务插入了记录

四大隔离级别：

SERIALIZABLE（串行化）：怒会出现并发问题 性能最差 但是最安全

数据连接池：（作用是可重用）

往往用map来实现值

初始大小：10个

最小空闲连接数：3个

增量：一次创建的最小单位（5个）

最大空闲连接数：12个

最大连接数：20个

最大等待时间：1000毫秒

连接池：创建连接 销毁连接

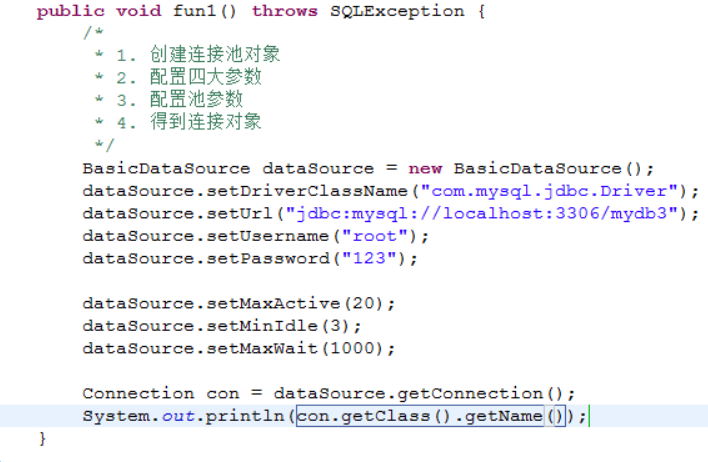
连接池使用四大参数完成配置

JAR包 ： bdpc需要pool

连接池 必须实现javax.sql.DataSource接口!

连接池 必须实现四大参数

连接池 必须返回connection对象 他的close（）方法不同 调用close不是关闭 而是归还



连接池内部使用了四大参数链接对象 即mysql驱动提供的Connection

连接池使用mysql的连接对象进行装饰 对close（）方法增强

装饰之后的connection的close（）方法 用来把当前的连接池归还

close（）方法会将链接归还给连接池

https://blog.csdn.net/shuaihj/article/details/14223015

装饰者模式：

对象增强的手段：

继承

装饰者模式

动态代理

使用继承会使类class增多 这是应该使用装饰者模式（动态装配）

装饰者模式：

咖啡 a = new 加糖（）；

咖啡 b = new 加糖（a）; 对a进行装饰

继承的缺点： 增强的内容不能动

被增强的对象不能改变

在io流中大量应用装饰者模式

四大流：InputStream OutputStream Reader Writer

FileInputStream 是一个节点流 就是和一个磁盘上的文件资源进行绑定

BufferedInputStream 是一个装饰流 创建时一定要有底层对象 不管给的是什么流 都会添加缓冲区

new bufferedInputStream(new InputStream(“…”));

对象流： ObjectInputStream(new InputStream(“…”));

装饰： 不知道被增强对象的具体类型时 可以使用

class MyConnection implemenyts Connection {

private Connection conn ;

public MyConnection(Connection c ){

//通过构造器传递底层对象

this.c = c ;

}

public Statement createStatement(){

return c.createStatement();

}

public void close(){

//把当前连接归还给池

}

}

C3P0 连接池：

免费 功能强于dbcp

C3P0的连接池： ComboPolledDataSource

\*\*\* 都要导入jar包



C3P0的配置文件：（最好使用C3P0连接池）

文件名称必须叫 c3p0-config.xml

连接时 会自动生成配置文件

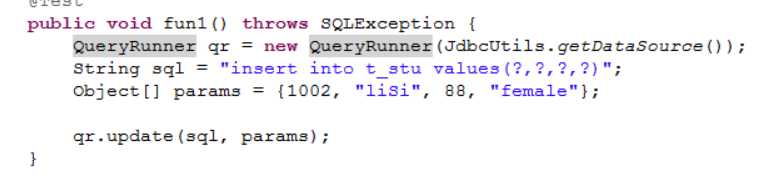
JNDI ： java命名 目录接口

1、JNDI 提出的目的是为了解藕，是为了开发更加easy维护，easy扩展。easy部署的应用。   
2、JNDI 是一个sun提出的一个规范(相似于jdbc),详细的实现是各个j2ee容器提供商。sun   仅仅是要求，j2ee容器必须有JNDI这种功能。  
3、JNDI 在j2ee系统中的角色是“交换机”，是J2EE组件在执行时间接地查找其它组件、资源或服务的通用机制。   
4、JNDI 是通过资源的名字来查找的，资源的名字在整个j2ee应用中(j2ee容器中)是唯一的。

ThreadLocal

ThreadLocal是解决线程安全问题一个很好的思路，它通过为每个线程提供一个独立的变量副本解决了变量并发访问的冲突问题。在很多情况下，ThreadLocal比直接使用synchronized同步机制解决线程安全问题更简单，更方便，且结果程序拥有更高的并发性。

dbUtils：





dbutils总结

QueryRunner方法：

int updata(String sq1,Object[] params) 执行增删改

T query(String sql,ResultSetHandler rsh ,Object[]…params) 执行查询

int updata(Connection conn , String sql , Object…params) 需要调用者提供connection 说明此方法不在管理connection 可以支持事务

T Query(Connection conn ,String sql ,ResultSetHandler rsh,Object…params)

ResultSetHandler 接口：

BeanHandler 构造器需要一个Class类型参数 用来把结果转化成指定的类型javaBean对象

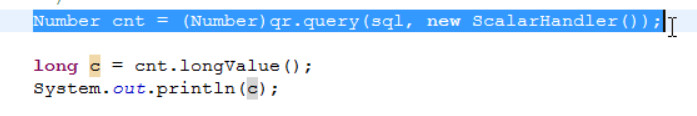
BeanListHandler 多行 ： 构造器需要一个Class类型的参数 用来把一行结果转化成指定的javaBean 多行就转化成多个javaBean

MapHandler 单行 ： 把一行结果转化成Map对象

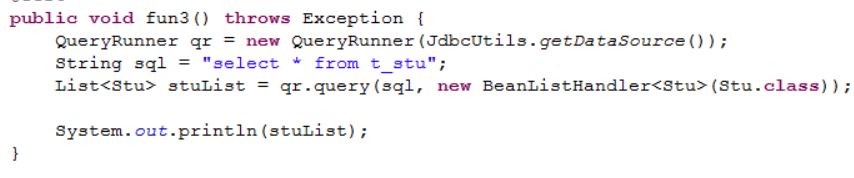
一行记录： 一个Map

MapListHandler 多行 ： 多行多个Map List<Map>

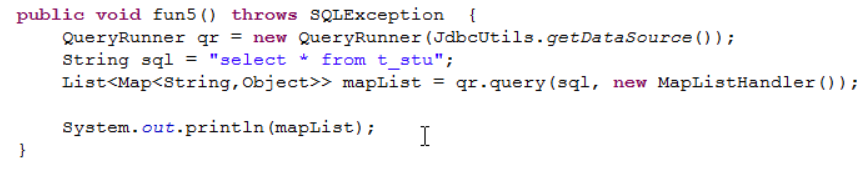
ScalarHandler 单行单列 ： 通常与select count(\*) from XXX 这个语句结果是单行单列 返回一个Long类型(如果想要转化成别的类型 先转化成 Number类型 在转化成别的数字类型



BeanListHandler： 用于单行多行记录



MapListHandler： 用于单行多行记录



BaseServlet

我们希望在一个servlet中可以有多个请求处理方式

客户端请求时 要多发送参数 告诉是用什么方法

请求处理方法：除了名称之外 都和service相同

请求处理方法 参数 声明异常 都要与 service相同

在service里面获取参数 来得到请求的方法

客户端必须传递 叫做method的参数

{之后要进行if判断 并进行调用}

\*\* 认为可以不用if判断 而是通过参数名 反射 方法

需要得到当前类的Class对象 ： Class c = this.getClass();

Method m = c.getMethod(参数名称 , HttpSerlvetRequest.class,HttpServletResponse.class);

m.invoke(这里写入参数 ); // 执行方法

\*\* 正常调用方法 this.addUser(参数 )

反射调用方法 this.addUSer(this,参数)；

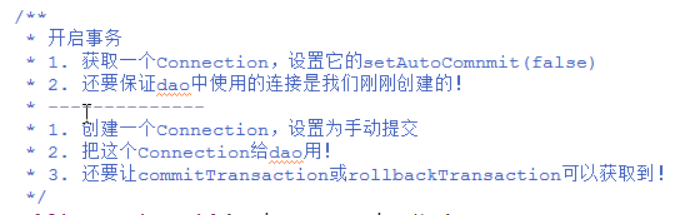
BaseServlet往往都会被继承

BaseServlet：servlet常常会转发重定向

在一个BaseServlet方法 返回 String 例如：“f:/index.jsp”

通过 冒号 分割字符串 前面f转发 r重定向 后面是路径

在service中处理事务：



/\*\* trim()他的所用是去掉字符序列左边和右边的空格 \*\*/

Connection不能出现在service中

在jbdcUtilsw中使用threadlocal来进行事务处理（防止多线程报错）

beginTransaction : 要在threadLocal中创建connection

commitTransaction ： 要remove Connection

ThreadLocal想一个存储单元

项目开始第一步：

导入原型 只有页面没有功能的页面

导包：mysql驱动 c3p0 dbutils

表单有target 超链接也有target target：内容在那里显示

<base>

InvocationTargetException 这个异常 说明反射调用的方法里面出现了错误 直接去查控制台报错

分页：

什么是分页

可以减少信息的传递 进行有用信息传递

页面数据都是servlet传递过来的

servlet 当前 pageCode pc

pc 如果页面没有传递当前页面

总页数 totalPage tp

tp 总记录/每页记录数

总记录数 totalRecord tr

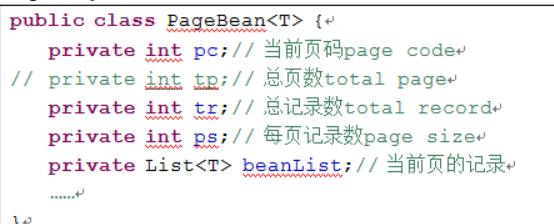
tr 总记录数

每页的记录数 是业务数据 可以自行更改

数据的传递：

把分页数据封装在JavaBean之中 叫做分页Bean

例如：pageBean



/\*\* limit A,B； 从第a行开始查找 向后查找b个记录

分页在各层中的处理：

页面 ： 传递当前页码 pc

servlet： 创建pageBean对象 给pageBean赋值 传递当前页面

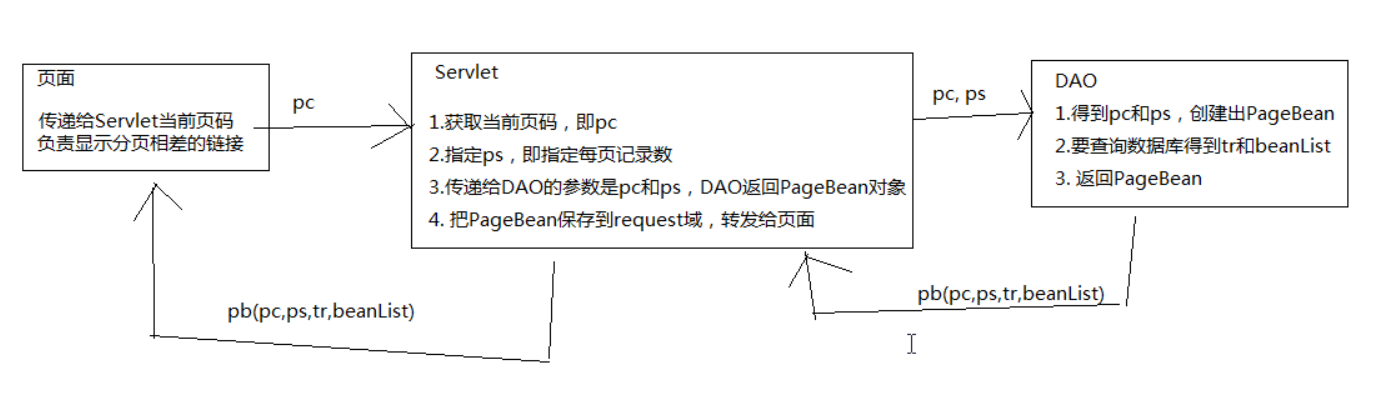
servlet需要给dao传递pc ps

service： 中转站

dao ：使用select count(\*) from … 获取记录总数

BeanList select \* from XXX limit X,Y; 装载数据

分页工序流程图：



Servlet:

获取request的当前页面 pc

获取每页记录数 ps

调用service的方法 将参数传递

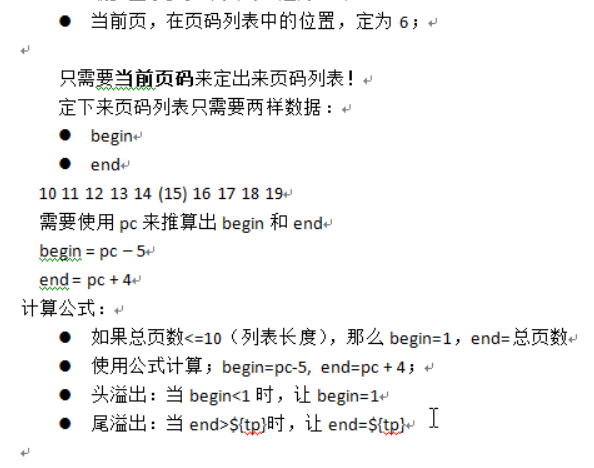
DAO：

创建一个pageBean 用queryRunner进行查找

将数据封装到PageBean之中

显示分页页码列表：

可以显示10个页码



需要把条件以String 的形式 保存到pagebean交给servlet

分页难

JavaWeb监听器：

javaweb三大组建：

servlet

listener

filter

listener监听器：

是一个接口 由我们来实现

需要注册 注册在按钮上

监听器中的方法 会在特殊事件发生时调用

javaWeb中的监听器：

事件源 三大域：

ServletContext：在服务器开启时出生 关闭时结束 每个项目只有一个

生死监听： ServletContextListener 有两个方法 一个在出生时调用 一个而在死亡时调用

属性监听： ServletContextAttributeListener 有三个方法： 一个在添加属性调用 一个在替换属性时调用 一个在属性被remove时调用

HttpSession：会话监听每个用户都有一个 request.getSession时会创建一个session 但是任何一个jsp页面都会自动创建一个session

生死监听：HttpSessionListener： 个方法 在出生时调用 在死亡是调用

属性监听：HttpSessionAttributeListener 3个方法 一个在添加属性时调用 一个在替换时调用 一个在remove时调用

ServletRequest： 请求发出时创建 但是请求静态资源时不会创建

生死监听：HttpServletRequest 2个方法 出生时调用 死亡时调用

属性监听：HttpServletRequest 3个方法 添加属性调用 替换时调用 移除时调用



void attributeAdded(ServletContextAttributeEvent e \*事件源\*) 添加属性

void attributeReplaced(ServletContextAttributeEvent e ) 替换属性

void attributeRemoved(ServletContextAttributeEvent e ) 移除属性

HttpSession :

生命周期监听： HttpSessionListener

方法： void sessionCreated(HttpSessionEvent e)

void sessionDestoryed(HttpSessionEvent e)

属性监听：

SessionAttributeListener它的三个方法基本同上

ServletRequest

生命周期 ServletRequestListener 两个方法同上

void requestInitialized(ServletRequestEvent e) 创建request时

void requestDestroyed(ServletRequestEvent e)

属性监听：

void attributeAdded(ServletRequestAttributeEvent e) …

javaWeb中完成编写监听器：

写一个监听类 要求必须实现摸个监听器接口

注册 在web.xml中配置完成注册

感知监听(都与HttpSession相关)

用来添加到javaBean中 不是添加到三大域

这两个监听器都不需要在web.xml 中注册

要在JavaBean中添加监听 需要添加接口 HttpSessionBuindingListener

Session在服务器停止时 会自动生成序列化文件保存到硬盘 再次开启服务器时 会再生Session不必担心由于服务器停止 而造成session丢失

Session的钝化 活化：

当session长时间不使用 会将session存放到硬盘上

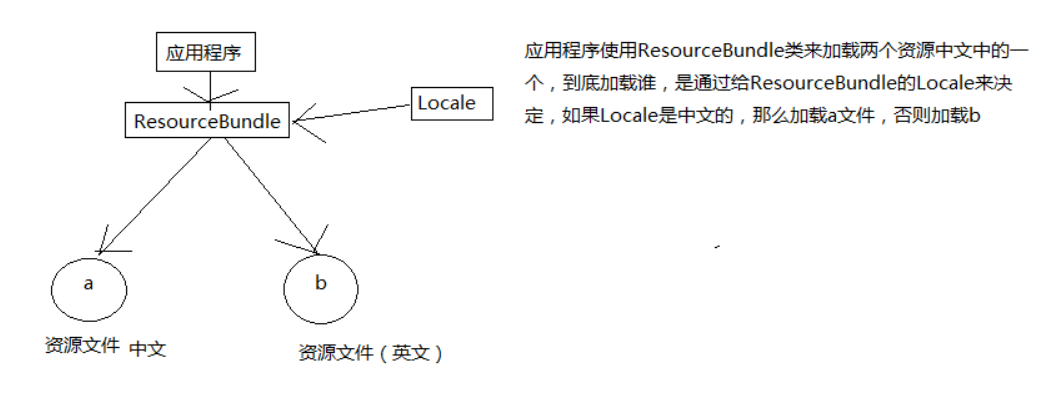
需要session的时候会从硬盘上抓取 称为session的钝化 活化

国际化

根据不同的地图 改变语言

需要把字符串都写成变量

通过配置文件 加载字符串





JavaWeb三大组件：servlet listener filter （都需要在web.xml中配置 感知监听器不用配置 直接放在javabean中）

过滤器：

会在一组资源的前面执行

可以让求情达到目标资源 也可以不让请求达到资源

\*\*\* 过滤器有拦截请求的能力

过滤器如何编写：

写一个类实现Filter接口

在web。xml中配置

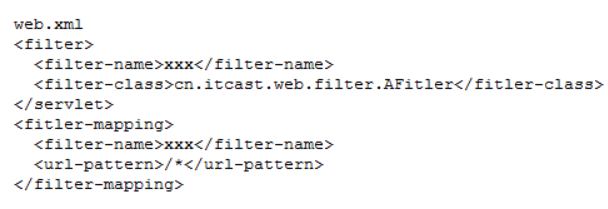
Filter接口：

void init(FilterConfig) 创建之后 马上执行 Filter会在服务区启动时创建

void destroy() 在销毁前执行 服务器结束前执行

void doFilter(ServletRequest,ServletResponse,FilterChain) 每次过滤时都会执行

Filter 是单例的 如同servlet



FilterConfig ServletConfig相似：

都可以获取初始化参数 getInitParameter()

获取过滤器的名称：getFilterName

获取application: getServletContext()

FilterChain :

方法 doFilter(ServletRequest ,ServletResponse)放行

FilterChain.daFilter(…)在doFilter方法中放行 停止拦截

/\*application不是[JAVA](https://www.baidu.com/s?wd=JAVA&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3P163mWnsmWIWnjD1nv7h0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHfknjmvPjR)上的...是[JSP](https://www.baidu.com/s?wd=JSP&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3P163mWnsmWIWnjD1nv7h0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHfknjmvPjR)中的...  
  
它和page request session application都是[JSP](https://www.baidu.com/s?wd=JSP&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3P163mWnsmWIWnjD1nv7h0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHfknjmvPjR)中的内置对象...  
  
在后台用ServletContext存储的[属性数据](https://www.baidu.com/s?wd=%E5%B1%9E%E6%80%A7%E6%95%B0%E6%8D%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3P163mWnsmWIWnjD1nv7h0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHfknjmvPjR)可以用application对象获得..  
  
而且application的[作用域](https://www.baidu.com/s?wd=%E4%BD%9C%E7%94%A8%E5%9F%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3P163mWnsmWIWnjD1nv7h0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHfknjmvPjR)是整个Tomcat启动的过程...  
  
例如: ServletContext.setAttribute("username",username);  
  
则在JSP网页中可以使用 application.getAttribute("username");  
  
来得到这个用户名....\*/

多过滤器：

**FilterChain#doFilter()方法：**

**执行目标资源 或是执行下一个过滤器 如果没有下一个过滤器那么执行的是目标资源 如果有 那么执行下一个过滤器**

**过滤器的四种拦截方式：**

1. **拦截请求**
2. **拦截转发**
3. **拦截包含**
4. **拦截错误**

**过滤器的四种拦截方法：**

**<dispatcher>FORWARD</dispatcher>**

**<dispatcher>REQUEST</dispatcher>**

**<dispatcher>INCLUDE</dispatcher>**

**<dispatcher>ERROR</dispatcher>**

**Filter的执行顺序： 根据web.xml 的写入顺序**

Filter的应用场景：

执行目标资源之前做预处理工作 例如设置密码 这种通常都会放行 只是在目标资源执行之前做一些准备工作

通过条件判断是否放行 例如校验用户是否登录 或者此IP是否被禁用

在目标资源执行后 做一些后续的特殊处理工作 例如把目标资源输出的数据进行处理

thread.Join把指定的线程加入到当前线程，可以将两个交替执行的线程合并为顺序执行的线程。比如在线程B中调用了线程A的Join()方法，直到线程A执行完毕后，才会继续执行线程B（类似于插队。。）

### 泛型的擦除

Java泛型是JDK 5引入的一个特性，它允许我们定义类和接口的时候使用参数类型，泛型在集合框架中被广泛使用。类型擦除是泛型中最让人困惑的部分，本篇文章将阐明什么是类型擦除，以及如何使用它。

# 一个常见错误

[复制代码](javascript:void(0);)

package simplejava;

import java.util.ArrayList;

public class Q29 {

public static void main(String[] args) {

ArrayList<String> al = new ArrayList<String>();

al.add("a");

al.add("b");

accept(al);

}

public static void accept(ArrayList<Object> al) {

for (Object o : al)

System.out.println(o);

}

}

[复制代码](javascript:void(0);)

以上代码看起来是没问题的，因为String是Object的子类。然而，这并不会工作，编译不会通过，并提示如下错误：  
The method accept(ArrayList<Object>) in the type Q29 is not applicable for the arguments (ArrayList<String>)

# List<Object>和List<String>

原因在于类型擦除。记住：Java的泛型机制是在编译级别实现的。编译器生成的字节码在运行期间并不包含泛型的类型信息。

在编译之后，List<Object>和List<String>将变成List，Object和String类型信息对于JVM来说是不可见的。在编译阶段，编译器发现它们不一致，因此给出了一个编译错误。

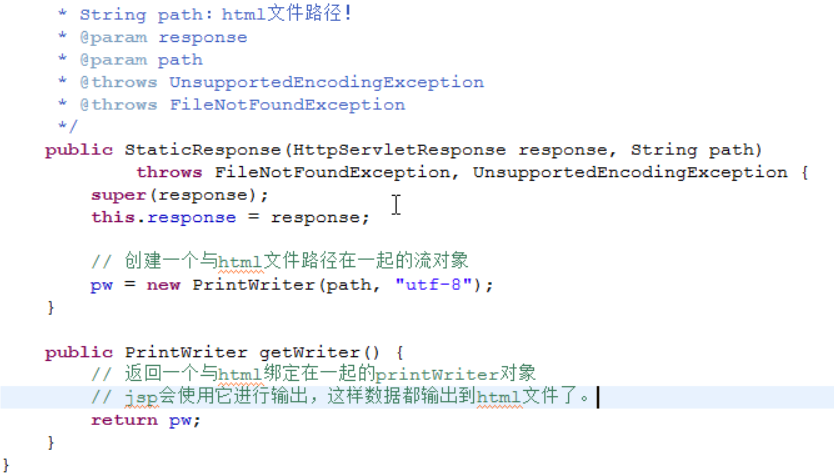
分IP统计网站访问次数

页面静态化：不会轻易发生变化）

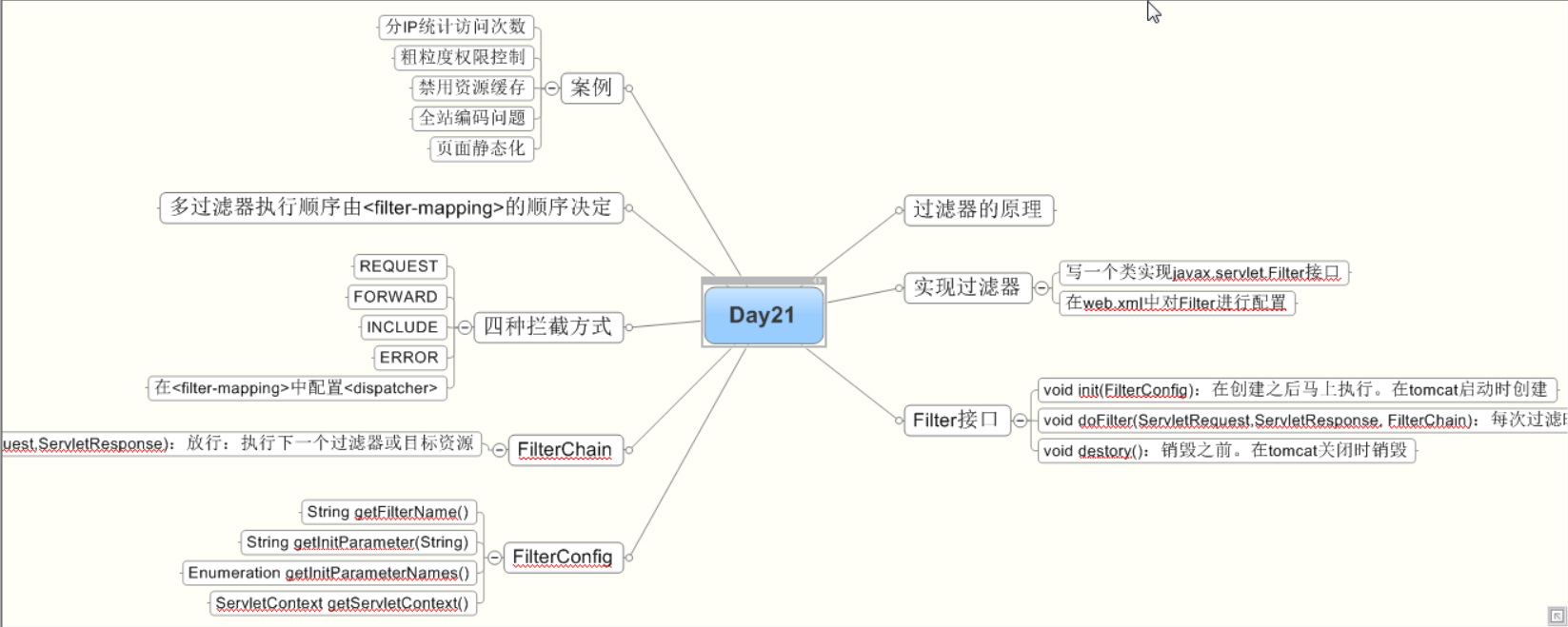
首次访问去数据库获取数据 然后保存到html中

第二次访问不用再去数据库获取 直接可以显示

性能好







上传：

客户端表单限制

上传对servlet有要求

表单会发发生变化

method=”post” 必须

enctype=”multipart/form-data”叫做多部件 二进制上传

表单中使用添加文件单项 <input type=”file” name=”XXX”/>

上传对Servlet的限制：

request.getParametere() 这个方法作废（永远返回null）

多部件表单：

每隔出多个部件 即一个表单项一个部件

一个部件中自己包含请求头请求体 空行

\*普通表单项：

1个头 Content-Disposition 表单项名称

\*文件表单项

2个头 Content-Disposition 表单项名称

content-Type: 文件类型 。。。

这个小组件 帮助我们解析上传数据：

上传三步：

相关类

工厂：DiskFileItemFactory

解析器：ServletFileUpload

表单项：FileItem

创建工厂：DiskFileItemFactory f = new DiskFileItemFactory();

创建解析器：ServletFileUpload sfd = new ServletFileUpload(f);

使用解析器来解析request 得到FileItem集合：List<FileItem> fileItemList = sfu.parseRequest(request);

FileItem:

Boolean isFormField():是否为普通表单 返回true为普通表单 false为文件表单

String getFieldName():返回当先表单项的名字

String getString(String charset):返回表单项的值

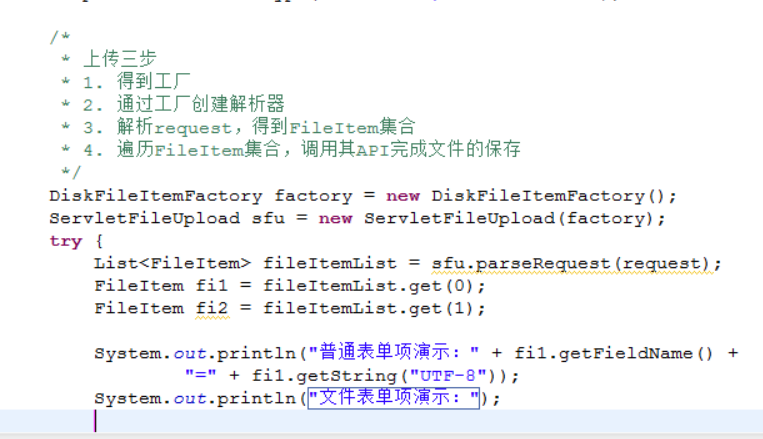
String getName():返回上传文件的名称

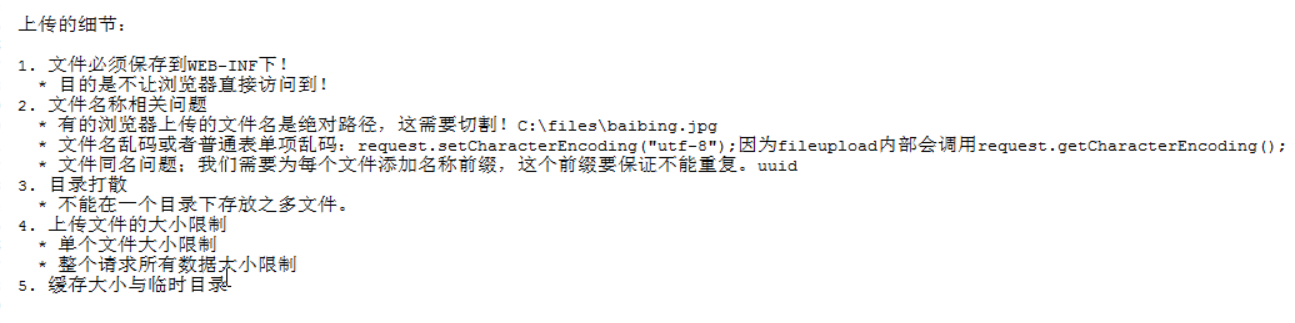
long getSize():返回上传文件的字节数

InputStream getInputStream():返回上传文件对应的输入流

void write(File destFile):把上传的文件内容保存到指定的文件中

文件上传演示：





文件一定要上传到web-inf 目录下 （此为安全目录 ）

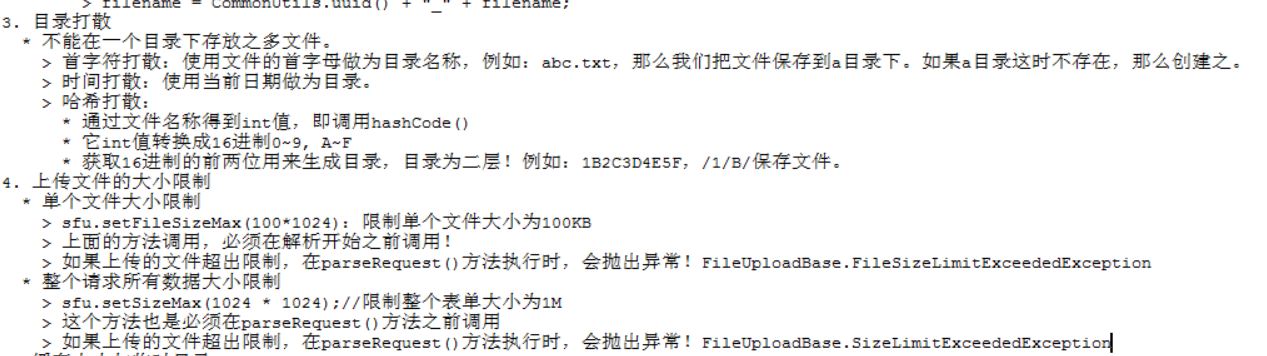
文件打散：不能都放在一个文件夹下

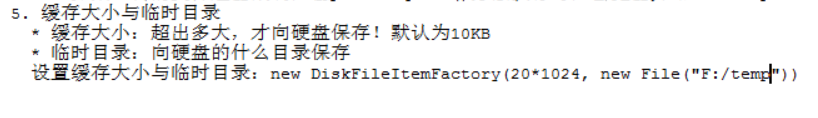
上传文件大小的限制：

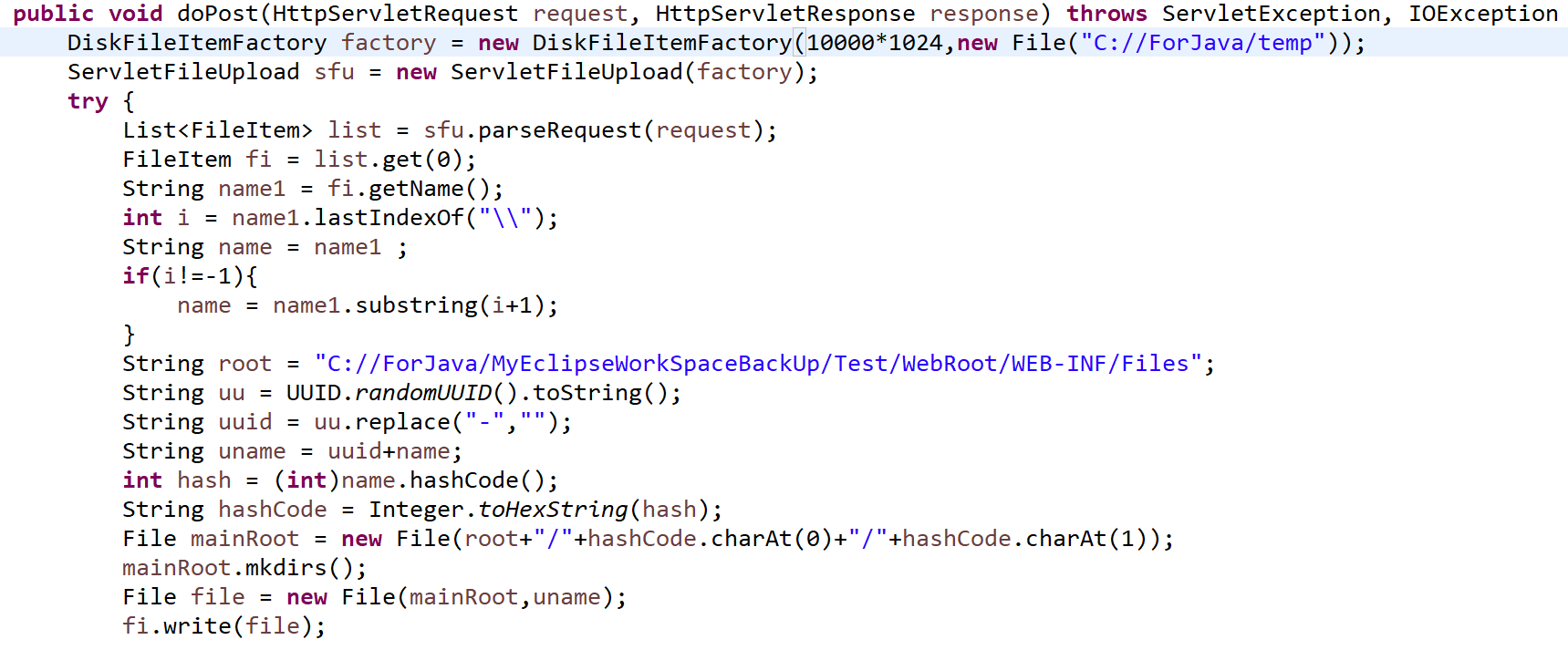
一个表单有多个文件表单

单个文件大小为：100kb

上面的方法调用必须在解析开始之前 在sfu.parseRequest()之前







下载：

下载就是向客户端相应字节数据

把文件变成字节数组 使用response.getOutputStream()

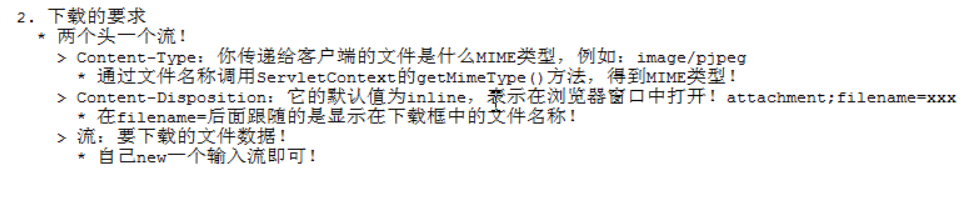
下载要求： 两个头 一个流

Content-Type : 你传递给浏览器的文件的类型mime类型 例如 ：image/pjpeg

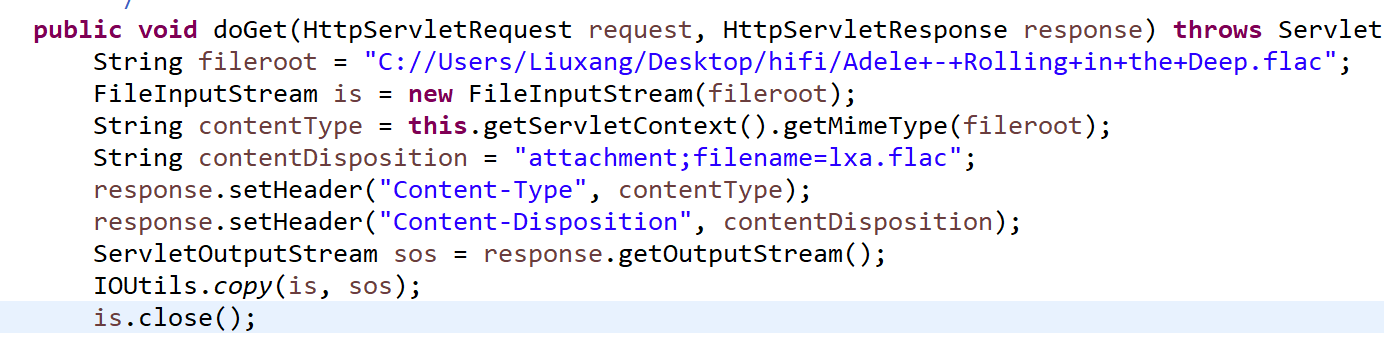
Content-Disposition ： 它的默认值为inline , 表示浏览器窗口打开

流： 要下载的文件数据

attachment——附件



示例代码：



关于文件名称乱码的解决：

response.setHeader("Content-Disposition", "attachment;filename=" + **new** String(filename.getBytes("GBK"),"ISO8859\_1"));

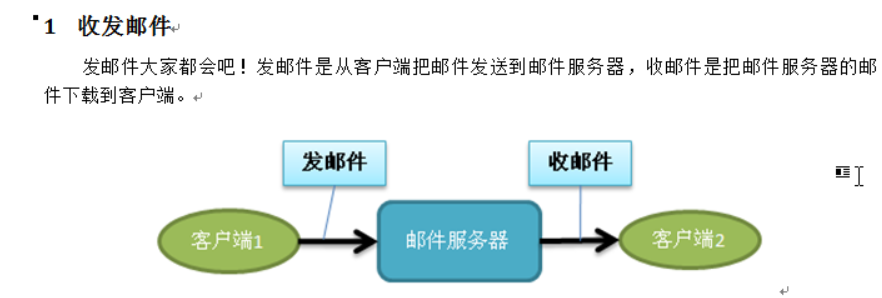
可以上网下载对应的工具类jar包进行文件的字符转化

邮件的概述：

javaMail是api

邮件相关的协议：

收邮件



邮件服务器的名称：(smtp 发邮件协议 pop3收邮件协议

smtp服务器的端口号25 服务器的名称 smtp.xxx.com

pop3服务器端口号110 服务器的名称 pop3.xxx.com

JavaMail：

导入jar

mail.jar

activation

核心类

session 如果你得到了它 表示已经连接上服务器 与Connection相似

MimeMessage 表示一个邮件对象 可以用他调用setFrom() 设置发件人 收件人

TransPort 只有一个功能 ——发邮件

得到Session：

使用session的静态方法getInstance(Properties，Authenticator)

一定需要传入参数

AJAX：阿贾克斯（异步的js xml）

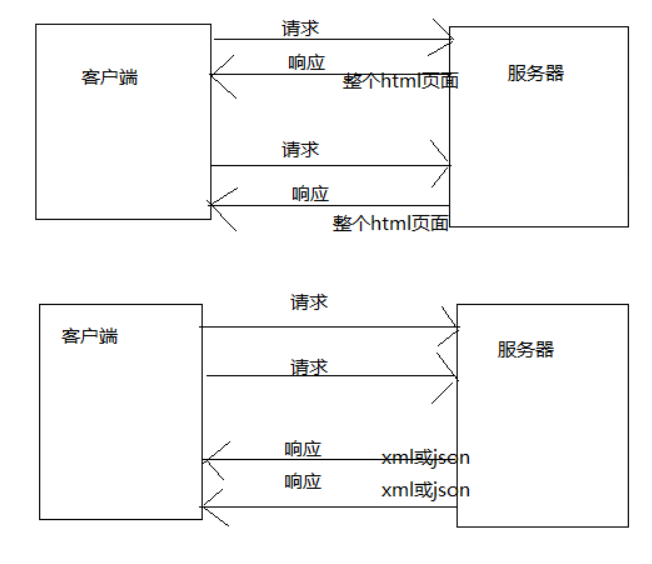
asynchronous javascript and xml

同步：发一个请求之后 等到恢复 在发送第二个请求

异步：发一个请求之后 不必等待 直接可以发送下一个请求

可以使用js接受服务器的响应 使用js局部刷新

服务器给了护短响应一般是整个页面 一个html 在ajax中因为时局部刷新 服务器就不用相应整个页面 而只是数据 大多数使用json



ajax的常见场景：（增强用户体验

百度搜索 扩充常见内容

ajax优点：

异步交互：增强用户体验

性能：因为服务器无需再相应整个页面 只需要响应部分内容 压力减轻了

缺点：

ajax不能应用在所有场景

ajax无端的增多了服务器的相应次数

ajax发送异步请求：

第一步

ajax其实只需要学习一个对象 XMLHttpRequest 掌握了它 就掌握了ajax！

得到XMLHttpRequest

大多数浏览 都支持var XMLHttp = new XMLHttpRequest();

指定ie6.0得到XMLHttpRequest : var xmlHttp = new ActiveXObject(“Msxml2.XMLHTTP”);

第二部

xmlHttp.open() 打开服务器的链接 需要三个参数

请求方式 get , post

请求的url 指定服务器资源

请求是否异步 true表示异步请求

xmlHttp.open(“GET”,”/XXServlet”,true);

第三部

xmlHttp.send() 如果参数填null可能会造成浏览器无法发送

参数： 具体请求体内容

第四部

得到异步响应

在xmlHttp对象的一个事件上注册监听器

onreadystatechange

xmlHttp一共有5个状态：

0 初始化未完成 刚创建未调用open方法

1 请求开始 调用了open 没有调用send

2 调用完了send

3 服务器开始响应 但不表示响应结束

4 服务器响应结束！

只关心最后的4 状态

得到xmlHttp的状态 : var state = xmlHttp.readyState;返回01234

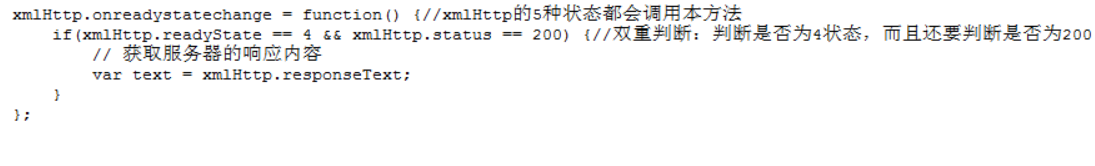
得到服务器的响应状态码：

var status = xmlHttp.status; 返回200 404 500 302 304

得到服务器响应内容

var content = xmlHttp.responseText;得到服务器的响应的文本格式的内容

var content = xmlHttp.respnseXML;得到服务器的响应的xml响应内容 是Document对象



get请求没有请求体 但一定要传入null

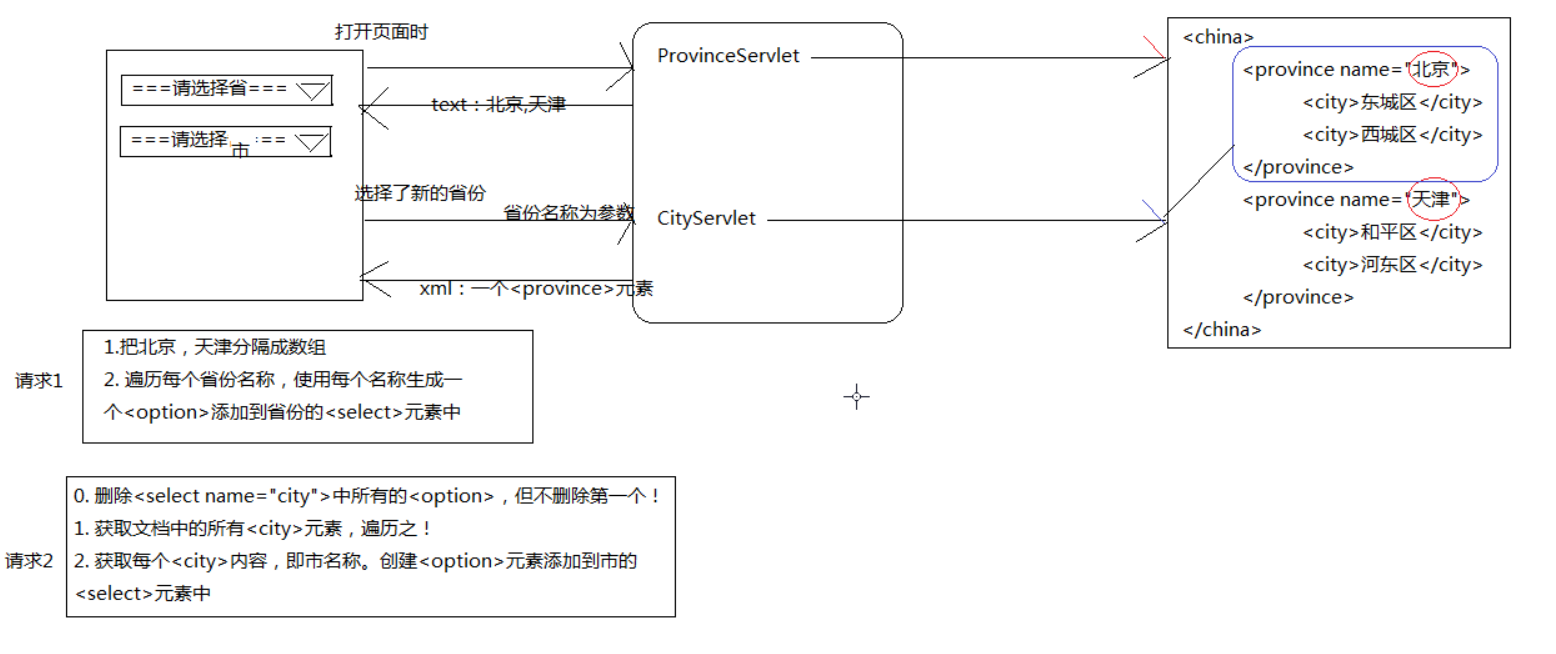
/\*\*\* 我在使用 window.onload 方法总是无法运行 所以直接

<body onload=XXX();>这样进行解决 在javascript里构造 function XXX(){…} \*\*\*/

响应内容为xml：

var doc = xmlHttp.responseXML; 得到的是Document对象

省事联动案例：



XStream:  
 作用：可以把JavaBean转化为xml

XStream的jar：

核心jar: xstream-1.4.7.jar

必须依赖jar: xpp3\_min-1.1.4c（这是一个速度很快的xml解析器）

使用步骤：

XStream xstream = new XStream();

String xmlStr = xstream.toXML(这里是javabean);

在大多数时间 都需要转化别名：

默认List元素 希望List对应自己想要的名称

xstream.alias(“别名”,City.class) 这样指定别名

alias——别名的意思

把标签的 Name转化为属性名

<ctiy>

<name>BEIJIN</name>

</city>

转化为：

<city name=”BEJIN”></city>

xstream.useAttributeFor(XXX.class这是需要转化到的标签 ,”name”这是要将子标签转化为属性的子标签)

四步发生异步请求

去除不想要的javabean属性

xstream.omitField(City.class,”这里是想要才删除的属性名臣个“)；

总结： （其实我不明白xstream的意义 。。。）

使用别名： .alias(“别名”,List.class)让List生成的元素名为 XX

让类的成员变成元素的属性: .useAttributeFor(XXX.class ，“name”);把XXX类的名为name的成员转化为属性名 包含在XXX的属性里面

去除无用的集合属性（去除集合类型的成员）：

去除Collection类型的名称

xmlstream.addImplicitCollection(XXX.class,”cities”);

会使XXX类中名为cities的Collectin类型的标签不会生成

去除一些不想要的累的指定成员 让其不生成xml元素：

xstream.omitField(City.class,”EnglishName”);

再生成的xml中不会出现EnglishName的对应元素

JSON

由js提供的数据交换格式

json的语法：

{}是对象

属性名必须使用双引号括起来 单引号不行！！！！！

属性值：

null

数值

字符串

数组 使用[]括起来

Boolean true和false

若想:ab”c 需要转义 “ab/”c”

在servlet之间进行数据传递 需要使用到转义 （传递字符串

XML的可读性较好

JSON的解析能力好

JSON更加受欢迎

JSON-LIB 工具：一定需要导入jar包

可以把JavaBean转化为JSON

核心类：

JSONObject 是一个map类型

JSONObject map = new JSONObject();

map.put(“name”,“lxa”);

map.put(这里也可以是一个JavaBean)

String s = map.toString();

SYSO(s); 会打印出来JSON字符串

JSONArray 是一个list类型

生成ajax的小工具

function ajax(method ,url , asyn , params)

如果是POST方法 需要添加setRequestHeader

asyn——异步

网上书城项目：

搭建架构：导入原型

用户模块 分类模块 购物车模块 图书模块 订单模块

功能分析：

前台（用户模块 – 注册，激活，登陆，退出（session销毁）

分类

图书（查询所有 分类查询 精准查询

购物车（添加条目 删除条目 删除所有条目

订单（生成订单 过往订单（在过往订单中按id查询订单） 付款 确认收货 付款功能

\*\* 后台：

管理员登陆（一般不需要注册 都写在数据库里

查看图书 按id查询 上传新的图书（要添加图片 并校验大小） 删除 更改

订单 查询订单 按状态查询 发货

框架的搭建：

导入包 创建package 创建数据表

用户模块：

创建相关模块

用户注册

用户激活

用户登陆

用户退出

分类模块

分类模块的相关创建

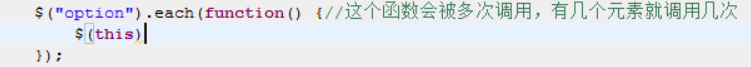
JQuery



元素选择器 返回一个jquery对象 不是Dom的Element对象 但是jquery对象包含多个dom对象

$(“[name=lxa]”)查找属性为name且值为lxa的元素

$(“:option[name=lxa]”).attr(“id”);获取这个元素的id的值



JQuery对于ajax的操作：

$(function(){

$(“#xxx”id选择器).blur(function(){

var value = $(“.xxx”class选择器).val();选择值

$.ajax({表示进行ajax

[url:”/XXServlet](url:)”要请求的servlet

data:{val:value}这时传递的对象 在servlet中使用getrParamter获取

async:true;表示 是否为异步请求

cache:false,表示 是否缓存 一般不缓存

type:”POST”请求方式（大写）

dataType:”json/xml/text”返回值的类型

success:function(result){表示请求成功之后执行的方法

alert(“XXX”);

};

)

)

}；)

#### 基础加强：

泛型：

具有一个或多个类型的变量的类 称之为泛型

class A<T> 在创建泛型时 需要为其类型变量赋值

A<String> a = new A<String>(); 如果创建类型实例时 不给赋值泛型 那么会报错

泛型方法： 具有一个或多个泛型的方法

class A<T>{public T fun(T a)}

这个方法不是泛型方法 是泛型类里的一个方法

public <T> T fun(T t) ---》 这个是泛型方法

泛型方法不一定在泛型类之中

泛型在类中 方法中的使用：

泛型可以在成员类型上使用

class A<T>

泛型可以在返回值 参数 可以在方法上使用

public T a1(T t)

泛型可以在局部变量的引用类型上使用

public void fun(){

T t = …

泛型的继承和实现

class A<T>

class AA extends A<String>() 若是继承了泛型类 子类不是泛型类

通配符：

List<? extends Object> list= new ArrayList<String>()；

只能出现在左边 不能出现在new时

通配符的含义：？表示一个不确定的类型

它的值会在调用时确定下来

List<? extends Number>

传递的泛型只能是Number或其子类

通配符的优点：

可以使泛型更加通用 尤其是调用形参的通配符

public void fun(List<?> list) 避免了许多问题

public void fun(List<? super Integer>)

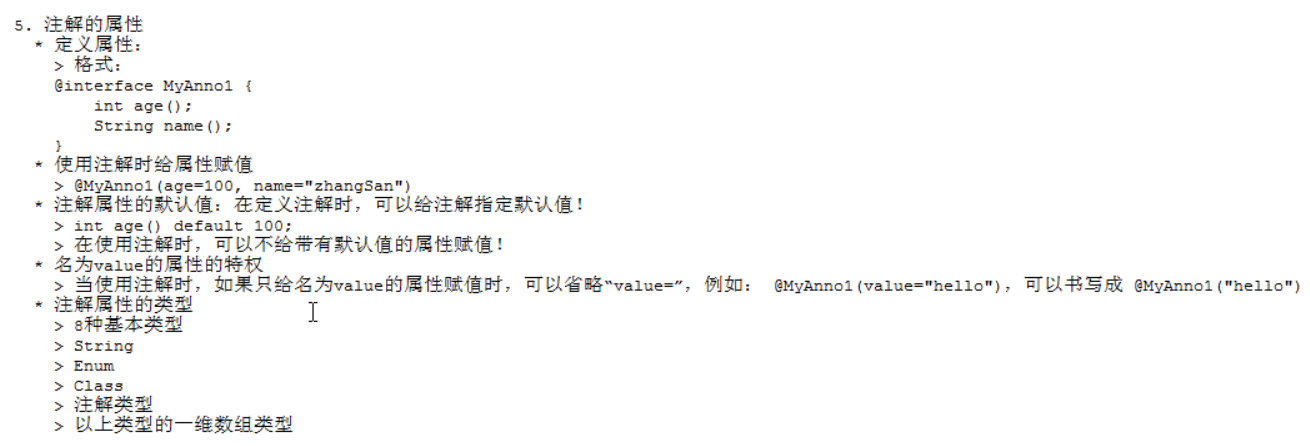
能传递Integer和Integer的父类型



注解：（替代xml进行数据的储存）

给框架看的语言：

所有注解都是Annotation的子类！



注解的作用目标以及保存策略

让一个注解 作用目标只能在类上 这就叫做目标的限定

在定义注解时 给注解添加注解@Target

@Target(value={ELementType.METHOD,MlementType.TYPE}) 这里面存放枚举类型

@interface MYAnno1{} 用Target来限定MYAnnol

保留策略：

源代码文件 SOURCE 注解只存在在源代码中 编译时被忽略

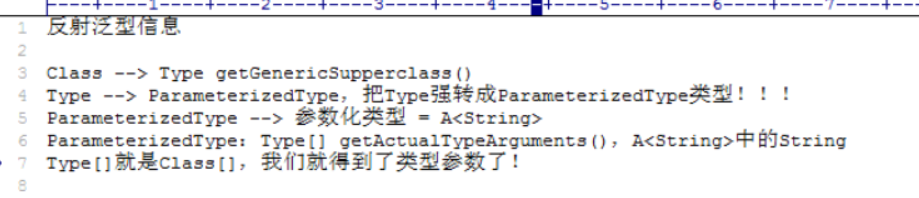
字节码文件 CLASS注解在源代码中存在 然后编译时会把注解信息放到CLAss文件 但是JVM加载时 会忽略注解！

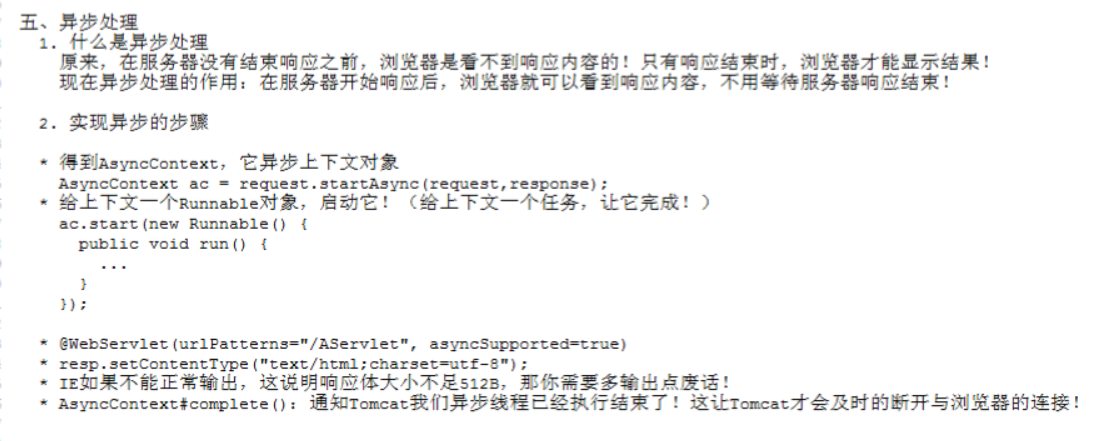
JVM码 RUNTIME 注解在源代码 字节码文件中存在 并在JVM加载类是 会把注解加载到JVM内存中

￥ 限定注释的保留策略

@Retention(RetentionPolicy.RUNTIME)

@interface XXX{} 在注解的上一行进行定义

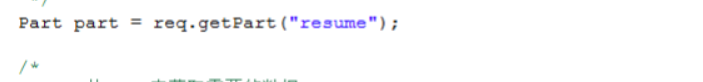




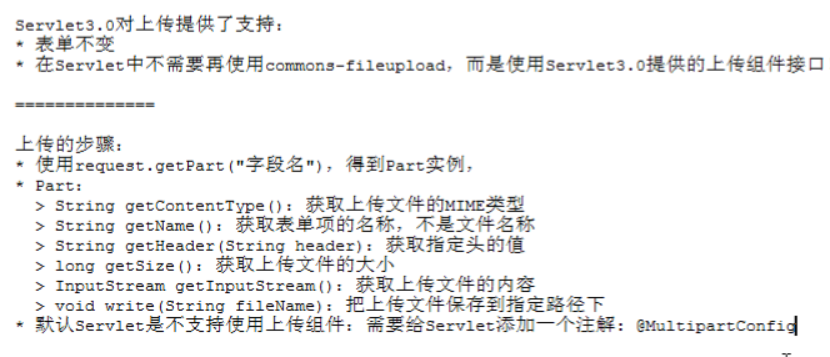
servlet异步：定义

C:\Users\Liuxang\AppData\Local\Temp\1528047809(1).png

servlet上传：



如上 获取表单中name为resume的对象 Part 类型 part.write(“目标路径”) 进行写入



动态代理 ：

最终目的是学习AOP

OOP是面向对象变成

AOP 面向切面编程

类加载器：ClassLoader

类加载器的分类：

引导 负责加载类库

扩展 负责加载扩展jar包

系统 负载加载应用下的class （通常就是自己写的class）

存在各层级的委托机制：

假若代码出现 new c();

系统会请求上一层级的扩展

扩展会继续向上请求引导

随后引导在类库寻找对应的类 如果没有向下传递null

扩展在扩展的jar包 下寻找对应的c 若没有

在系统中寻找 还没有的话 报错

这样做的效果就是 安全

#每一个线程都有自己的类加载器

自定义类加载器：可以对自己的class进行加密

Tomcat 的类加载器：

提供两个类加载器

服务器加载器： tomcat下的专门加载lib下的类

应用加载器： 还有一个专门加载 /WEB-INF/classes 应用加载器

引导—扩展—系统—服务器加载器—应用加载器

服务器 应用加载器 ：先自己加载 若找不对对应的class 在寻求其他加载器