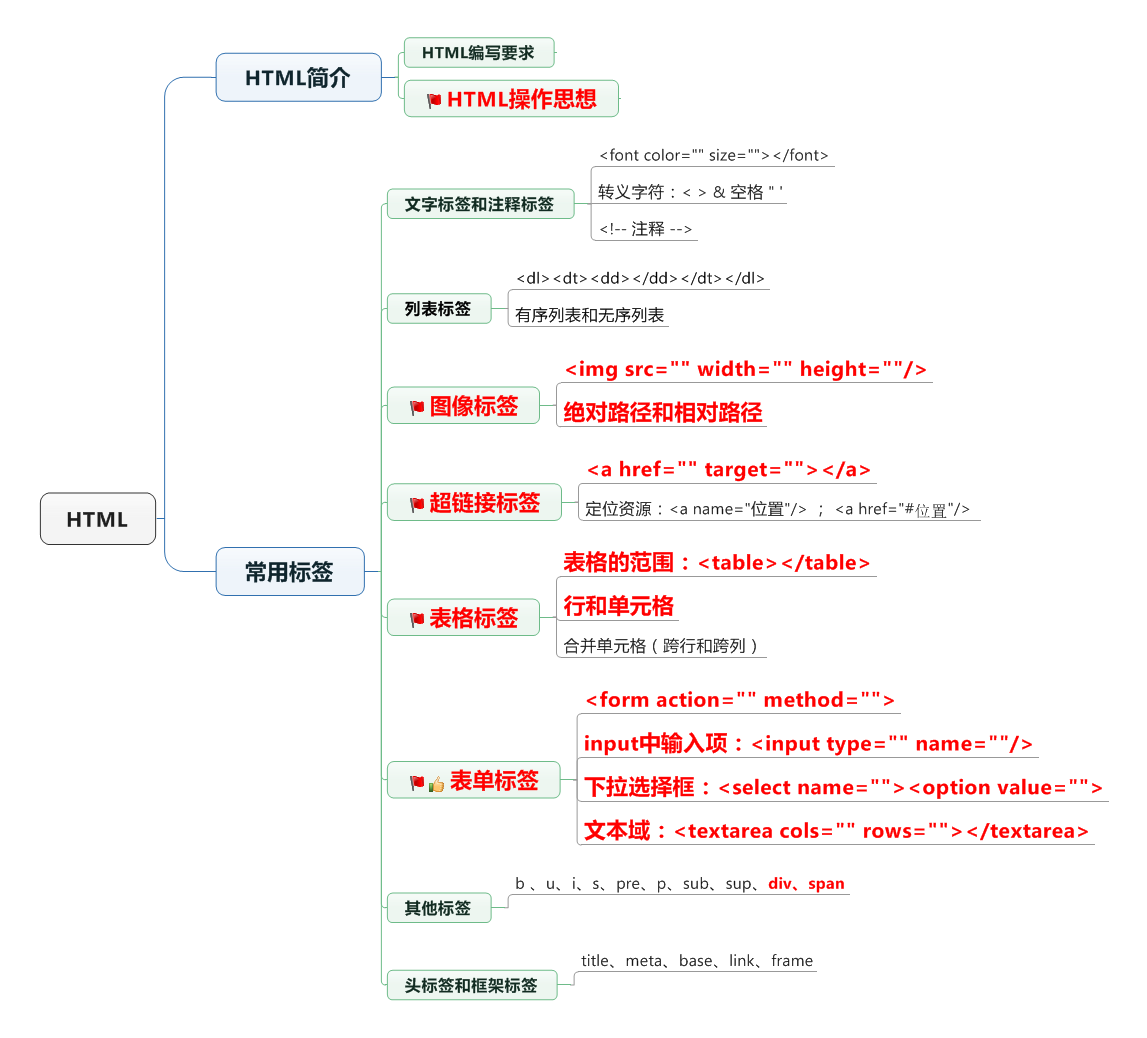
Day1 Html



1、html的简介

\* 什么是html？

- HyperText Markup Language：超文本标记语言，网页语言

\*\* 超文本：超出文本的范畴，使用html可以轻松实现这样操作

\*\* 标记：html所有的操作都是通过标记实现的，标记就是标签，<标签名称>

\*\* 网页语言：

\* 第一个html程序。

- 创建java文件.java

\*\* 先编译，然后再运行（jvm）

- html后缀是 .html .htm

\*\* 直接通过浏览器就可以运行

- 代码

\*\* 这是我的<font size="5" color="red">第一个html程序！</font>

\* html的规范（遵循）

1、一个html文件开始标签和结束的标签 <html> </html>

- 定义一个java方法 { }

2、html包含两部分内容

（1）<head> 设置相关信息</head>

（2）<body> 显示在页面上的内容都写在body里面</body>

3、html的标签有开始标签，也要有结束标签

- <head></head>

4、html的代码不区分大小写的

5、有些标签，没有结束标签 ，在标签内结束

- 比如 换行 <br/> <hr/>

\* html的操作思想（\*\*\*\*\*\*理解）

网页中有很多数据，不同的数据可能需要不同的显示效果，这个时候需要使用标签把要操作的数据包起来（封装起来），

通过修改标签的属性值实现标签内数据样式的变化。

一个标签相当于一个容器，想要修改容器内数据的样式，只需要改变容器的属性值，就可以实现容器内数据样式的变化。

html中常用的标签

2、文字标签和注释标签

\* 文字标签：修改文字的样式

- <font></font>

- 属性：

\* size: 文字的大小 取值范围 1-7,超出了7，默认还是7

\* color：文字颜色

- 两种表示方式

\*\* 英文单词：red green blue black white yellow gray......

\*\* 使用十六进制数表示 #ffffff : RGB

- 通过工具实现不同的颜色 #66cc66

\* 注释标签

- java注释几种？三种

- html的注释 ： <!-- html的注释 -->

3、标题标签、水平线标签和特殊字符

\* 标题标签

- <h1></h1> <h2></h2> <h3></h3> .......<h6></h6>

- 从h1到h6，大小是依次变小，同时会自动换行

\* 水平线标签

- <hr/>

- 属性

\*\* size: 水平线的粗细 取值范围 1-7

\*\* color: 颜色

- 代码

<hr size="5" color="blue"/>

\* 特殊字符

- 想要在页面上显示这样的内容 <html>:是网页的开始！

- 需要对特殊字符进行转义

\* < &lt;

\* > &gt;

\* 空格：&nbsp;

\* & : &amp;

4、列表标签

- 比如现在显示这样的效果

传智播客

财务部

学工部

人事部

\*\* <dl> </dl>: 表示列表的范围

\*\* 在dl里面 <dt></dt>: 上层内容

\*\* 在dl里面 <dd></dd>：下层内容

- 代码

<dl>

<dt>传智播客</dt>

<dd>财务部</dd>

<dd>学工部</dd>

<dd>人事部</dd>

</dl>

- 想要在页面上显示这样的效果

1. 财务部

2. 学工部

3. 人事部

a. 财务部

b. 学工部

c. 人事部

i. 财务部

ii. 学工部

iii. 人事部

\*\* <ol></ol> : 有序列表的范围

- 属性 type：设置排序方式 1(默认) a i

\*\* 在ol标签里面 <li>具体内容</li>

- 代码

<ol>

<li>财务部</li>

<li>学工部</li>

<li>人事部</li>

</ol>

- 想要在页面上显示这样的效果

特殊符号（方框） 财务部

特殊符号（方框） 学工部

\*\* <ul></ul> : 表示无序列表的范围

属性： type: 空心圆circle 、实心圆disc 、实心方块square ，默认disc

在ul里面 <li></li>

- 代码

<ul>

<li>财务部</li>

<li>学工部</li>

<li>人事部</li>

</ul>

5、图像标签（\*\*\*\*\*\*\*）

\* <img src="图片的路径"/>

- src: 图片的路径

- width：图片的宽度

- height：图片的高度

- alt: 图片上显示的文字，把鼠标移动到图片上，停留片刻显示内容

\*\* 有些浏览器下不显示（没有效果）

示例：

<html>

<head>

<title>HTML示例</title>

</head>

<body>

<img src="img\w02.jpg" alt="这是一个美女"/>

<!-- <img src="w02.jpg" width="300" height="400" alt="这是图片上的文字"/>-->

</body>

</html>

6、路径的介绍

\* 分类：两类

\*\* 绝对路径

- C:\Users\asus\Desktop\0413\day01\code\a.jpg

- http://www.baidu.com/b.jpg

\*\* 相对路径

- 一个文件相对于另外一个文件的位置

- 三种：

\*\* html文件和图片在一个路径下，可以直接写文件名称

- <img src="b1.jpg" alt="这是一个美女"/>

\*\* 图片在html的下层目录

在html文件中，使用img文件夹下面的a.jpg

-- C:\Users\asus\Desktop\0413\day01\code\ 4.html

-- C:\Users\asus\Desktop\0413\day01\code\ img\a.jpg

\*\*\* 在html中使用图片 4.html和img在一个路径下

img\a.jpg

\*\* 图片在html文件的上层目录

--- C:\Users\asus\Desktop\0413\day01\ code\4.html

--- C:\Users\asus\Desktop\0413\day01\ c.png

\*\*\* html文件所在的目录和图片是一个目录

\*\* 图片和html文件是什么关系？

- 图片在html的所在目录的上层目录 day01

\*\* 怎么表示上层路径 ../

- ../: day01

-- ../c.png

\*\* 想要表示上层的上层 ../../

8、超链接标签（\*\*\*\*\*\*）

\* 链接资源(\*\*\*\*\*\*)

- <a href="链接到资源的路径"> 显示在页面上的内容 </a>

\*\* href: 链接的资源的地址

\*\* target：设置打开的方式 ，默认是在当前页打开

-- \_blank : 在一个新窗口打开

-- \_self: 在当前页打开 默认

- 当超链接不需要到任何的地址 在href里面加#

- <a href="#">这是一个超链接2</a>

\* 定位资源

\*\* 如果想要定位资源：定义一个位置

<a name="top">顶部</a>

\*\* 回到这个位置

<a href="#top">回到顶部</a>

\*\* 引入一个标签 pre：原样输出

9、表格标签（\*\*\*\*重要的标签\*\*\*\*\*）

\* 可以对数据进行格式化，使数据显示更加清晰

\* <table></table>: 表示表格的范围

- border：表格线

- bordercolor：表格线的颜色

- cellspacing：单元格直接的距离

- width：表格的宽度

- height：表格的高度

\*\* 在table里面 <tr></tr>

- 设置对齐方式 align： left center right

\*\*\* 在tr里面 <td></td>

- 设置显示方式 align： left center right

\*\*\* 使用th也可以表示单元格

- 表示可以实现居中和加粗

\* 代码

<table border="1" bordercolor="blue" cellspacing="0" width="200" height="150">

\*\* 表格的标题

<caption></caption>

\*\* 合并单元格

- rowspan：跨行

\*\* <td rowspan="3">人员信息</td>

- colspan：跨列

\*\* <td colspan="3">人员信息</td>

**10、表单标签（\*\*\*\*今天最重要的标签\*\*\*\*）**

\* 可以提交数据到开心网的服务器，这个过程可以使用表单标签实现

\* <form></form>: 定义一个表单的范围

- 属性

\*\* action： 提交到地址，默认提交到当前的页面

\*\* method: 表单提交方式

- 常用的有两种 get和post，默认是get请求

\*\*target属性:规定在何处打开 action URL

* 

\*\* 面试题目： get和post区别

1、get请求地址栏会携带提交的数据，post不会携带（请求体里面。在第七天时候讲http协议时候）

2、get请求安全级别较低，post较高

3、get请求数据大小的限制，post没有限制

\*\* enctype：一般请求下不需要这个属性，做文件上传时候需要设置这个属性（第22天时候讲文件上传）

\*\* 输入项：可以输入内容或者选择内容的部分

- 大部分的输入项 使用 <input type="输入项的类型"/>

\*\*\*\*\*\*\* 在输入项里面需要有一个name属性

\*\*\* 普通输入项：<input type="text"/>

\*\*\* 密码输入项：<input type="password"/>

\*\*\* 单选输入项：<input type="radio"/>

- 在里面需要属性 name

- name的属性值必须要相同

- 必须有一个value值

\*\*\*\* 实现默认选中的属性

-- checked="checked"

\*\*\* 复选输入项：<input type="checkbox"/>

- 在里面需要属性 name

- name的属性值必须要相同

- 必须有一个value值

\*\*\*\* 实现默认选中的属性

-- checked="checked"

\*\*\* 文件输入项（在后面上传时候用到）

- <input type="file"/>

\*\*\* 下拉输入项（不是在input标签里面的）

<select name="birth">

<option value="1991">1991</option>

<option value="1992">1992</option>

<option value="1993">1993</option>

</select>

- 默认选择

\*\*\* selected="selected"

\*\*\* 文本域

<textarea cols="10" rows="10"></textarea>

\*\*\* 隐藏项（不会显示在页面上，但是存在于html代码里面）

<input type="hidden" />

\*\*\* 提交按钮

<input type="submit"/>

<input type="reset" value="重置注册"/>

<input type="button" value="普通按钮"/>

<input type="submit" value="注册"/>

- file:///C:/Users/asus/Desktop/0413/day01/code/10-表单标签一.html

?sex=on&love=on&love=on&birth=1991

当在输入项里面写了name属性之后

- file:///C:/Users/asus/Desktop/0413/day01/code/10-表单标签一.html

?phone=2222&pwd=3333&sex=on&love=on&birth=1993&tex=aaaaaaaa&hid=

- file:///C:/Users/asus/Desktop/0413/day01/code/10-表单标签一.html?

phone=130111111&pwd=123456&sex=nv&love=y&love=p&love=pp&birth=1992&tex=good+love&hid=

\*\* ?输入项name的值=输入的值&

\*\* 参数类似于key-value形式

\*\*\* 使用图片提交

<input type="image" src="图片路径"/>

\*\*\* 重置按钮： 回到输入项的初始状态

<input type="reset"/>

\*\*\* 普通按钮(和明天讲js在一起使用的)

<input type="button" value=""/>

11、案例：使用表单标签实现注册页面

- 使用表格实现页面效果

- 超链接不想要他有效果 href="#"

- 如果表格里面的单元格没有内容， 使用空格作为占位符 &nbsp;

- 使用图片提交表单 <input type="image" src="图片的路径"/>

12、html中的其他的常用标签的使用

b : 加粗

s ：删除线

u ：下划线

i ：斜体

pre ：原样输出

sub : 下标

sup : 上标

p ：段落标签 比br标签多一行

====明天css时候一直使用

div ：自动换行

span：在一行显示

13、html的头标签的使用

\* html两部分组成 head和body

\*\* 在head里面的标签就是头标签

\*\* title标签：表示在标签上显示的内容

\*\* <meta>标签：设置页面的一些相关内容

- <meta name="keywords" content="毕姥爷，熊出没，刘翔">

<meta http-equiv="refresh" content="3;url=01-hello.html" />

\*\* base标签：设置超链接的基本设置

- 可以统一设置超链接的打开方式

<base target="\_blank"/>

\*\* link标签：引入外部文件

\*\* 明天css，可以使用link标签引入css文件

14、框架标签的使用（会用）

\* <frameset>

- rows:按照行进行划分

\*\* <frameset rows="80,\*">

- cols:按照列进行划分

\*\* <frameset cols="80,\*">

\* <frame>

- 具体显示的页面

- <frame name="lower\_left" src="b.html">

\*\*\* 使用框架标签时候，不能写在body里面，使用了框架标签，需要把body去掉

\* <frameset rows="80,\*"> //把页面划分成上下两部分

<frame name="top" src="a.html"> //上面页面

<frameset cols="150,\*"> //把下面部分划分成左右两部分

<frame name="lower\_left" src="b.html"> //左边的页面

<frame name="lower\_right" src="c.html"> //右边的页面

</frameset>

</frameset>

\* 如果在左边的页面超链接，想让内容显示在右边的页面中

- 设置超链接里面属性 target值设置成名称

- <a href="01-hello.html" target="right">超链接1</a>

15、a标签的扩展（了解）

- 百度是网络资源

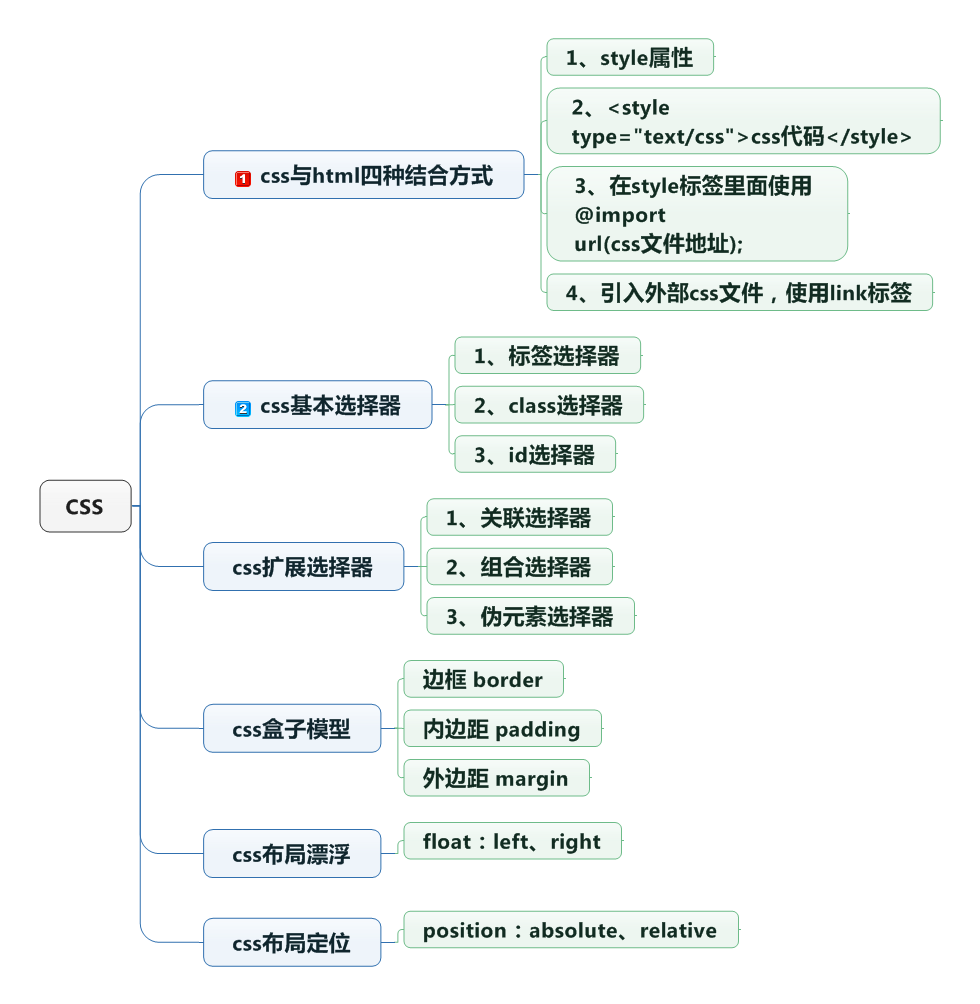
- 当a标签里面访问网络资源时候，必须要加一个协议 http：表示一个网络的公共协议，

如果加上http协议之后，自动识别访问资源是一个网络资源

- 当浏览器里面找到相关协议，首先看这个协议是不是公共协议http。

如果不是公共协议，会去本地电脑找支持这个协议的应用程序。

Day2 CSS和JS



CSS

1、css的简介

\* css： 层叠样式表

\*\* 层叠：一层一层的

\*\* 样式表：

很多的属性和属性值

\* 是页面显示效果更加好

\* CSS将网页内容和显示样式进行分离，提高了显示功能。

2、css和html的结合方式（四种结合方式）

（1）在每个html标签上面都有一个属性 style，把css和html结合在一起

- <div style="background-color:red;color:green;">

（2）使用html的一个标签实现 <style>标签，写在head里面

\* <style type="text/css">

css代码;

</style>

\* <style type="text/css">

div {

background-color:blue;

color: red;

}

</style>

（3）在style标签里面 使用语句（在某些浏览器下不起作用）

@import url(css文件的路径);

- 第一步，创建一个css文件

<style type="text/css">

@import url(div.css);

</style>

（4）使用头标签 link，引入外部css文件

- 第一步 ，创建一个css文件

- <link rel="stylesheet" type="text/css" href="css文件的路径" />

\*\*\* 第三种结合方式，缺点：在某些浏览器下不起作用，一般使用第四种方式

\*\*\* 优先级（一般情况）

由上到下，由外到内。优先级由低到高。

\*\*\* 后加载的优先级高

\*\*\* 格式 选择器名称 { 属性名：属性值；属性名：属性值；…….}

3、css的基本选择器（三种）

\*\* 要对哪个标签里面的数据进行操作

（1）标签选择器

\* 使用标签名作为选择器的名称

div {

background-color:gray;

color:white;

}

（2）class选择器

\* 每个html标签都有一个属性 class

- <div class="haha">aaaaaaa</div>

- .haha {

background-color: orange;

}

（3）id选择器

\* 每个html标签上面有一个属性 id

- <div id="hehe">bbbbb</div>

- #hehe {

background-color: #333300;

}

\*\*\* 优先级

style > id选择器 > class选择器 > 标签选择器

4、css的扩展选择器

（1）关联选择器

\* <div><p>wwwwwwww</p></div>

\* 设置div标签里面p标签的样式，嵌套标签里面的样式

\* div p {

background-color: green;

}

（2）组合选择器

\* <div>1111</div>

<p>22222</p>

\* 把div和p标签设置成相同的样式，把不同的标签设置成相同的样式

\* div,p {

background-color: orange;

}

（3）伪元素选择器(了解，浏览器的兼容性比较差)

\* css里面提供了一些定义好的样式，可以拿过来使用

\* 比如超链接

\*\* 超链接的状态

原始状态 鼠标放上去状态 点击 点击之后

:link :hover :active :visited

\*\* 记忆的方法

lv ha

\*超链接的状态

- 原始 :link

- 悬停 :hover

- 点击 :active

- 点击之后 :visited

<html>

<head>

<title>HTML示例</title>

<style type="text/css">

/\*原始状态\*/

a:link {

background-color: red;

}

/\*悬停状态\*/

a:hover {

background-color: green;

}

/\*点击状态\*/

a:active {

background-color: blue;

}

/\*点击之后的状态\*/

a:visited {

background-color: gray;

}

</style>

</head>

<body>

<a href="http://www.sina1.com.cn" target="\_blank">CSS概述和与HTML的结合方式超链接一</a>

</body>

</html>

5、css的盒子模型

\*\* 在进行布局前需要把数据封装到一块一块的区域内（div）

（1）边框

border: 2px solid blue;

border：统一设置

上 border-top

下 border-bottom

左 border-left

右 border-right

（2）内边距

padding:20px;

使用padding统一设置

也可以分别设置

上下左右四个内边距

（3）外边距

margin: 20px;

可以使用margin统一设置

也可以分别设置

上下左右四个外边距

6、css的布局的漂浮(了解)

float：

\*\* 属性值

left :　 文本流向对象的右边

right :　 文本流向对象的左边

7、css的布局的定位（了解）

position：

\*\* 属性值

- absolute ：

\*\*\* 将对象从文档流中拖出

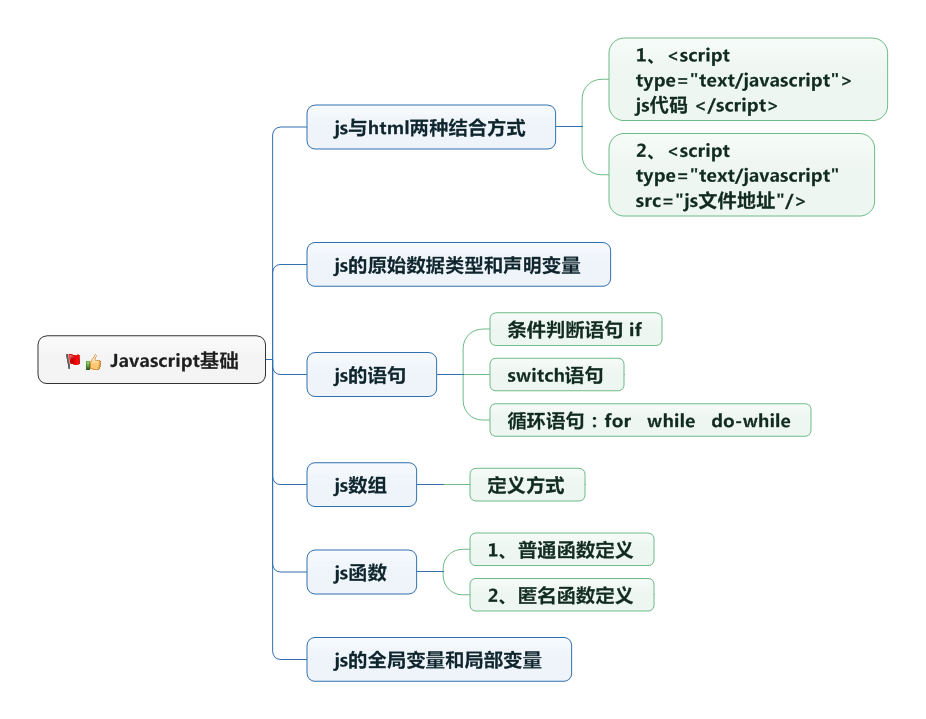
\*\*\* 可以是top、bottom等属性进行定位

- relative ：

\*\*\* 不会把对象从文档流中拖出

\*\*\* 可以使用top、bottom等属性进行定位

**JS:**

****

**1、javascript的简介**

**\* 是基于对象和事件驱动的语言，应用与客户端。**

**- 基于对象：**

**\*\* 提供好了很多对象，可以直接拿过来使用**

**- 事件驱动：**

**\*\* html做网站静态效果，javascript动态效果**

**- 客户端：专门指的是浏览器**

**\* js的特点**

**（1）交互性**

**- 信息的动态交互**

**（2）安全性**

**- js不能访问本地磁盘的文件**

**（3）跨平台性**

**- java里面跨平台性，虚拟机**

**- 只有能够支持js的浏览器，都可以运行**

**\* javascript和java的区别（雷锋和雷峰塔）**

**（1）java是sun公司，现在oracle；js是网景公司**

**（2）JavaScript 是基于对象的，java是面向对象**

**（3）java是强类型的语言，js是弱类型的语言**

**- 比如java里面 int i = "10";**

**- js: var i = 10; var m = "10";**

**（4）JavaScript只需解析就可以执行，而java需要先编译成字节码文件，再执行。**

**\* javascript的组成（下面js）**

**三部分组成**

**（1）ECMAScript**

**- ECMA : 欧洲计算机协会**

**- 有ECMA组织制定的js的语法，语句.....**

**（2）BOM**

**- broswer object model: 浏览器对象模型**

**（3）DOM**

**- document object model：文档对象模型**

**2、js和html的结合方式（两种）**

**第一种：**

**- 使用一个标签 <script type="text/javascript"> js代码; </script>**

**第二种：**

**- 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="1.js"></script>**

**\*\* 使用第二种方式时候，就不要在script标签里面写js代码了，不会执行。**

**3、js的原始类型和声明变量**

**\*\* java的基本数据类型 byte short int long float double char boolean**

**\*\* 定义变量 都使用关键字 var**

**\*\* js的原始类型（五个）**

**- string: 字符串**

**\*\*\* var str = "abc";**

**- number：数字类型**

**\*\*\* var m = 123;**

**- boolean：true和false**

**\*\*\* var flag = true;**

**- null**

**\*\*\* var date = new Date();**

**\*\*\* 获取对象的引用，null表示对象引用为空 ，所有对象的引用也是object**

**- undefined**

**\*\*\* 定义一个变量，没有赋值**

**\*\*\* var aa;**

**\*\* typeof(); 查看当前变量的数据类型**

**（**

ECMAScript 有 5 种原始类型（primitive type），即 Undefined、Null、Boolean、Number 和 String。

对变量或值调用 typeof 运算符将返回下列值之一：

* undefined - 如果变量是 Undefined 类型的
* boolean - 如果变量是 Boolean 类型的
* number - 如果变量是 Number 类型的
* string - 如果变量是 String 类型的
* object - 如果变量是一种引用类型或 Null 类型的

注释：您也许会问，为什么 typeof 运算符对于 null 值会返回 "Object"。这实际上是 JavaScript 最初实现中的一个错误，然后被 ECMAScript 沿用了。现在，null 被认为是对象的占位符，从而解释了这一矛盾，但从技术上来说，它仍然是原始值。

这里需要注意的是：alert(typeof null);  
//输出的是object 。null被认为是对象的占位符，但仍然算做原始数据类型

另一种只有一个值的类型是 Null，它只有一个专用值 null，即它的字面量。值 undefined 实际上是从值 null 派生来的，因此 ECMAScript 把它们定义为相等的。

alert(null == undefined); //输出 "true"

尽管这两个值相等，但它们的含义不同。undefined 是声明了变量但未对其初始化时赋予该变量的值，null 则用于表示尚未存在的对象（在讨论 typeof 运算符时，简单地介绍过这一点）。如果函数或方法要返回的是对象，那么找不到该对象时，返回的通常是 null。

2.Null类型  
Null类型（空型）只有一个值就是：null。  
<script type=*"text/javascript"*>  
alert(null == undefined);  
//显示true  
</script>  
undefined实际上是从值null派生来的，所以显示true  
undefined是声明了变量但是未赋值，null是找不到对象。

**）**

**4、js的语句**

**- java里面的语句：**

**\*\* if判断**

**\*\* switch语句**

**\*\* 循环 for while do-while**

**-js里面的这些语句**

**\*\* if判断语句**

**\*\*\*\* =:表示赋值**

**\*\*\*\* ==：表示判断**

**\*\* switch语句**

**- java里面支持数据类型 string支持吗？在jdk1.7开始支持**

**- js里面都支持**

**- switch(a) {**

**case 5:**

**break;**

**case 6:**

**break;**

**default:**

**......**

**}**

**\*\* 循环语句 for while do-while**

**- while循环**

**\*\*\*\* var i = 5;**

**while(i>1) {**

**alert(i);**

**i--;**

**}**

**- for循环**

**\*\*\* for(int i=0;i<=10;i++) { }**

**for(var mm=0;mm<=3;mm++) {**

**alert(mm);**

**}**

**\*\* i++ ++i和java里面一样**

**5、js的运算符**

**\*\* += ： x+=y; ===> x=x+y;**

**\*\* js里面不区分整数和小数**

**var j = 123;**

**alert(j/1000\*1000);**

**// j/1000\*1000 在java里面得到结果是 0**

**// 在js里面不区分整数和小数，123/1000=0.123 \* 1000 = 123**

**\*\* 字符串的相加和相减的操作**

**var str = "123";**

**\*\* 如果相加时候，做是字符串连接**

**\*\* 如果相减，做的是相减的运算**

**\* //字符串的操作**

**var str = "456";**

**//alert(str+1); //在java里面操作的结果是 4561 ，在js里面还是 4561**

**alert(str-1); //相减时候，执行减法的运算**

**\* 提示NaN:表示不是一个数字**

**\*\* boolean类型也可以操作**

**\*\*\* 如果设置成true，相当于这个值是1**

**\*\*\* 如果设置成false，相当于这个值是0**

**\*\* == 和 === 区别**

**\*\* 做判断**

**\*\* == 比较的只是值**

**\*\* === 比较的是值和类型**

**\*\* 引入知识**

**直接向页面输出的语句（可以把内容显示在页面上）**

**\* document.write("aaa");**

**document.wirte("<hr/>");**

**\*\* 可以向页面输出变量，固定值和html代码**

**6、实现99乘法表（输出到页面上）**

**\* document.write("<table border='1' bordercolor='blue'>");**

**//循环行 9**

**for(var i=1;i<=9;i++) {**

**document.write("<tr>");**

**//循环列**

**for(var j=1;j<=i;j++) {**

**document.write("<td>");**

**//运算**

**document.write(j+"\*"+i+"="+i\*j);**

**document.write("</td>");**

**}**

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

**document.write("</tr>");**

**}**

**document.write("</table>");**

**- document.write里面是双引号，如果设置标签的属性需要使用单引号**

**- document.write可以输出变量，还可以输出html代码**

**7、js的数组**

**\* 什么是数组？**

**- 使用变量，var m = 10;**

**- java里面的数组 定义 int[] arr = {1,2,3};**

**\* 定义方式（三种）**

**第一种： var arr = [1,2,3]; var arr = [1,"4",true];**

**第二种：使用内置对象 Array对象**

**var arr1 = new Array(5); //定义一个数组，数组的长度是5**

**arr1[0] = "1";**

**第三种：使用内置对象 Array**

**var arr2 = new Array(3,4,5); //定义一个数组，数组里面的元素是3 4 5**

**\* 数组里面有一个属性 length：获取到数组的长度**

**\* 数组可以存放不同的数据类型的数据**

**8、js的函数**

**\*\* 在java里面定义方法**

**public 返回类型void /int 方法名(参数列表) {**

**方法体;**

**返回值;**

**}**

**public int add(int a,int b) {**

**int sum = a+b;**

**return sum;**

**}**

**\*\* 在js里面定义函数（方法）有三种方式**

**\*\*\*\* 函数的参数列表里面，不需要写var，直接写参数名称**

**第一种方式：**

**\*\*\*\* 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（根据实际需要）;**

**}**

**\*\*\*\* 代码**

**//使用第一种方式创建函数**

**function test() {**

**alert("qqqqq");**

**}**

**//调用方法**

**//test();**

**//定义一个有参数的方法 实现两个数的相加**

**function add1(a,b) {**

**var sum = a+b;**

**alert(sum);**

**}**

**//add1(2,3);**

**//有返回值的效果**

**function add2(a,b,c) {**

**var sum1 = a+b+c;**

**return sum1;**

**}**

**alert(add2(3,4,5));**

**第二种方式：**

**\*\*\*\* 匿名函数**

**var add = function(参数列表) {**

**方法体和返回值;**

**}**

**\*\*\*\* 代码**

**//第二种方式创建函数**

**var add3 = function(m,n) {**

**alert(m+n);**

**}**

**//调用方法**

**add3(5,6);**

**第三种方式：（用的少，了解）**

**\*\*\* 动态函数**

**\*\*\* 使用到js里面的一个内置对象 Function**

**var add = new Function("参数列表","方法体和返回值");**

**9、js的全局变量和局部变量**

**\*\* 全局变量：在script标签里面定义一个变量，这个变量在页面中js部分都可以使用**

**- 在方法外部使用，在方法内部使用，在另外一个script标签使用**

**\*\* 局部变量：在方法内部定义一个变量，只能在方法内部使用**

**- 如果在方法的外部调用这个变量，提示出错**

**- SCRIPT5009: “nn”未定义**

**12-js的局部变量.html, 行18 字符3**

**\*\* ie自带了一个调试工具，ie8及其以上的版本中，键盘上 F12，在页面下方出现一个条**

**10、script标签放在的位置**

**\* 建议把script标签放到 </body>后面**

**\* 如果现在有这样一个需求：**

**在js里面需要获取到input里面的值，如果把script标签放到head里面**

**会出现问题。**

**html解析是从上到下解析的，script标签放到的是head里面，直接在里面取input里面的值，**

**因为页面还没有解析到input那一行，肯定取不到。**

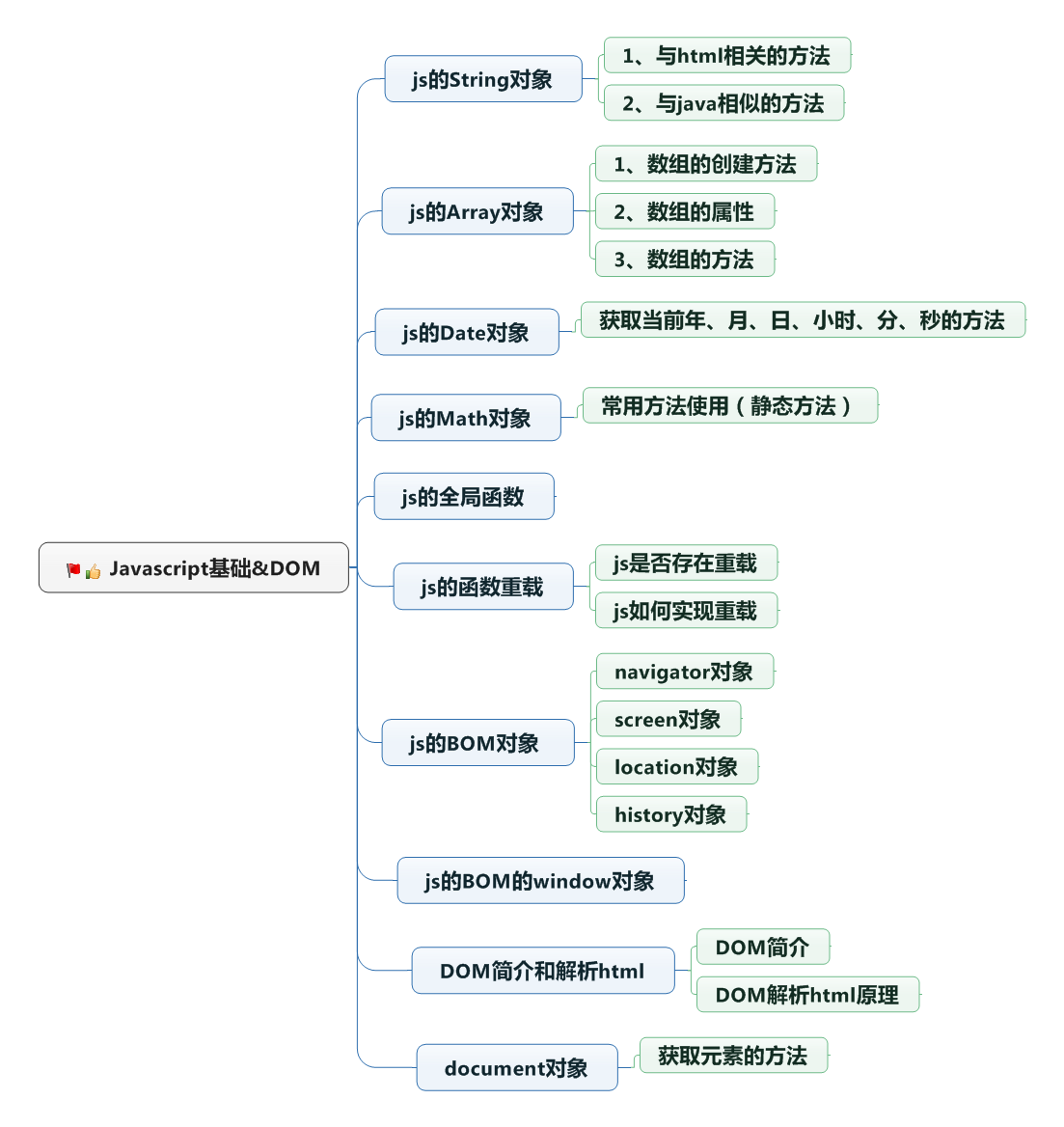
**11、js的重载**

**\* 什么是重载？方法名相同，参数列表不同**

**- java里面有重载，肯定有**

**\* js里面是否有重载？没有**

**Day 3**

****

**1、js的String对象**

**\*\* 创建String对象**

**\*\*\* var str = "abc";**

**\*\* 方法和属性（文档）**

**\*\*\* 属性 length：字符串的长度**

**\*\*\* 方法**

**（1）与html相关的方法**

**- bold()：加粗**

**- fontcolor(): 设置字符串的颜色**

**- fontsize(): 设置字体的大小**

**- link(): 将字符串显示成超链接**

**\*\*\*\* str4.link("hello.html")**

**- sub() sup(): 下标和上标**

**（2）与java相似的方法**

**- concat(): 连接字符串**

**\*\* //concat方法**

**var str1 = "abc";**

**var str2 = "dfg";**

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

**- charAt():返回指定指定位置的字符串**

**\*\* var str3 = "abcdefg";**

**document.write(str3.charAt(20)); //字符位置不存在，返回空字符串**

**- indexOf()： 返回字符串位置**

**\*\* var str4 = "poiuyt";**

**document.write(str4.indexOf("w")); //字符不存在，返回-1**

**- split()：切分字符串，成数组**

**\*\* var str5 = "a-b-c-d";**

**var arr1 = str5.split("-");**

**document.write("length: "+arr1.length);**

**- replace() ： 替换字符串**

**\* 传递两个参数：**

**-- 第一个参数是原始字符**

**-- 要替换成的字符**

**\* var str6 = "abcd";**

**document.write(str6);**

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

**document.write(str6.replace("a","Q"));**

**- substr()和substring()**

**\* var str7 = "abcdefghuiop";**

**//document.write(str7.substr(5,5)); //fghui 从第五位开始，向后截取五个字符**

**\*\*\* 从第几位开始，向后截取几位**

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

**document.write(str7.substring(3,5)); //de 从第几位开始到第几位结束 [3,5)**

**\*\*\* 从第几位开始，到第几位结束，但是不包含最后哪一位**

**2、js的Array对象**

**\*\* 创建数组（三种）**

**- var arr1 = [1,2,3];**

**- var arr2 = new Array(3); //长度是3**

**- var arr3 = new Array(1,2,3); //数组中的元素是1 2 3**

**- var arr = []; //创建一个空数组**

**\*\* 属性：length：查看数组的长度**

**\*\* 方法**

**- concat方法： 数组的连接**

**\* var arr11 = [1,2,3];**

**var arr12 = [4,5,6];**

**document.write(arr11.concat(arr12));**

**- join()：根据指定的字符分割数组**

**\* var arr13 = new Array(3);**

**arr13[0] = "a";**

**arr13[1] = "b";**

**arr13[2] = "c";**

**document.write(arr13);**

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

**document.write(arr13.join("-"));**

**- push():向数组末尾添加元素，返回数组的新的长度**

**\*\* 如果添加的是一个数组，这个时候把数组当做一个整体字符串添加进去**

**\* //push方法**

**var arr14 = new Array(3);**

**arr14[0] = "tom";**

**arr14[1] = "lucy";**

**arr14[2] = "jack";**

**document.write("old array: "+arr14);**

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

**document.write("old length:"+arr14.length);**

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

**document.write("new length: "+arr14.push("zhangsan"));**

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

**document.write("new array: "+arr14);**

**\* var arr15 = ["aaa","bbb","ccc"];**

**var arr16 = ["www","qqq"];**

**document.write("old array:"+arr15);**

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

**document.write("old length:"+arr15.length);**

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

**document.write("new length:"+arr15.push(arr16));**

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

**document.write("new array: "+arr15);**

**for(var i=0;i<arr15.length;i++) {**

**alert(arr15[i]);**

**}**

**- pop()：表示 删除最后一个元素，返回删除的那个元素**

**\* var arr17 = ["zhangsan","lisi","wangwu","zhaoliu"];**

**document.write("old array: "+arr17);**

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

**document.write("return: "+arr17.pop());**

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

**document.write("new array: "+arr17);**

**- reverse():颠倒数组中的元素的顺序**

**\* var arr17 = ["zhangsan","lisi","wangwu","zhaoliu"];**

**document.write("old array: "+arr17);**

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

**document.write("return: "+arr17.pop());**

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

**document.write("new array: "+arr17);**

**//reverse方法**

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

**var arr18 = ["zhangsan1","lisi1","zhaoliu1","niuqi1"];**

**document.write("old array: "+arr18);**

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

**document.write("new array:"+arr18.reverse());**

**3、js的Date对象**

**\*\* 在java里面获取当前时间**

**Date date = new Date();**

**//格式化**

**//toLocaleString() //2015年4月17日 11:17:12**

**\*\* js里面获取当前时间**

**var date = new Date();**

**//获取当前时间**

**var date = new Date();**

**document.write(date); // Fri Apr 17 10:47:46 UTC+0800 2015**

**//转换成习惯的格式**

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

**document.write(date.toLocaleString());**

**\*\* 获取当前的年方法**

**getFullYear()：得到当前的年**

**\*\*\*\* document.write("year: "+date.getFullYear());**

**\*\* 获取当前的月方法**

**getMonth()：获取当前的月**

**\*\*\* 返回的是 0-11月，如果想要得到准确的值，加1**

**\*\*\*\* var date1 = date.getMonth()+1;**

**document.write("month: "+date1);**

**\*\* 获取当前的星期**

**getDay()：星期,返回的是 (0 ~ 6)**

**\*\* 外国朋友，把星期日作为一周的第一天，星期日返回的是 0**

**而星期一到星期六 返回的是 1-6**

**\*\* document.write("week: "+date.getDay());**

**\*\* 获取当前的日**

**getDate()：得到当前的天 1-31**

**\*\* document.write("day: "+date.getDate());**

**\*\* 获取当前的小时**

**getHours()：获取小时**

**\*\* document.write("hour: "+date.getHours());**

**\*\* 获取当前的分钟**

**getMinutes()：分钟**

**\*\* document.write("minute: "+date.getMinutes());**

**\*\* 获取当前的秒**

**getSeconds(): 秒**

**\*\* document.write("second: "+date.getSeconds());**

**\*\* 获取毫秒数**

**getTime()**

**返回的是1970 1 1 至今的毫秒数**

**\*\* 应用场景：**

**\*\*\* 使用毫秒数处理缓存的效果（不有缓存）**

**http://www.baidu.com?毫秒数**

**4、js的Math对象**

**\* 数学的运算**

**\*\* 里面的都是静态方法，使用可以直接使用 Math.方法()**

**\*\* ceil(x): 向上舍人**

**\*\* floor(x)：向下舍人**

**\*\* round(x)：四舍五入**

**\*\* random()：得到随机数（伪随机数）**

**- 得到0-9的随机数**

**Math.random()\*10**

**Math.floor(Math.random()\*10));**

**5、js的全局函数**

**\* 由于不属于任何一个对象，直接写名称使用**

**\*\* eval() ： 执行js代码（如果字符串是一个js代码，使用方法直接执行）**

**\*\*\*\* var str = "alert('1234');";**

**//alert(str); ----alert('1234')**

**eval(str);----1234**

**\*\* encodeURI() ：对字符进行编码**

**- %E6%B5%8B%E8%AF%95%E4%B8%AD%E6%96%87aaa1234**

**decodeURI() ：对字符进行解码**

**encodeURIComponent() 和 decodeURIComponent()**

**\*\* isNaN():判断当前字符串是否是数字**

**-- var str2 = "aaaa";**

**alert(isNaN(str2));**

**\*\*\* 如果是数字，返回false**

**\*\*\* 如果不是数字，返回true**

**\*\* parseInt()：类型转换**

**\*\* var str3 = "123";**

**document.write(parseInt(str3)+1);**

**6、js的函数的重载**

**\*\* 什么是重载？方法名相同，参数不同**

**\*\* js的重载是否存在？ 不存在**

**\*\* 调用最后一个方法**

**\*\* 把传递的参数保存到 arguments数组里面**

**\*\* js里面是否存在重载？(面试题目)**

**（1）js里面不存在重载。**

**（2）但是可以通过其他方式模拟重载的效果 （通过arguments数组来实现）**

**\*\*\* function add1() {**

**//比如传递的是两个参数**

**if(arguments.length == 2) {**

**return arguments[0]+arguments[1];**

**} else if (arguments.length == 3) {**

**return arguments[0]+arguments[1]+arguments[2];**

**} else if (arguments.length == 4) {**

**return arguments[0]+arguments[1]+arguments[2]+arguments[3];**

**} else {**

**return 0;**

**}**

**}**

**7、js的bom对象**

**\*\* bom：broswer object model: 浏览器对象模型**

**\*\* 有哪些对象？**

**\*\*\* navigator： 获取客户机的信息（浏览器的信息）**

**- navigator.appName**

**- document.write(navigator.appName);**

**\*\*\* screen: 获取屏幕的信息**

**- document.write(screen.width);**

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

**document.write(screen.height);**

**\*\*\* location: 请求url地址**

**- href属性**

**\*\*\*\* 获取到请求的url地址**

**- document.write(location.href);**

**\*\*\*\* 设置url地址**

**- 页面上安置一个按钮，按钮上绑定一个事件，当我点击这个按钮，页面可以跳转到另外一个页面**

**- location.href = "hello.html";**

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

**- 鼠标点击事件 onclick="js的方法;"**

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

**//href**

**function href1() {**

**//alert("aaaa");**

**location.href = "hello.html";**

**}**

**\*\*\* history：请求的url的历史记录**

**- 创建三个页面**

**1、创建第一个页面 a.html 写一个超链接 到 b.html**

**2、创建b.html 超链接 到 c.html**

**3、创建c.html**

**- 到访问的上一个页面**

**history.back();**

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

**- 到访问的下一个页面**

**history.forward();**

**history.go(1);**

**\*\*\*\* window（\*\*\*\*）**

**\* 窗口对象**

**\* 顶层对象（所用的bom对象都是在window里面操作的）**

**\*\* 方法**

**- window.alert() : 页面弹出一个框，显示内容**

**\*\* 简写alert()**

**- confirm()： 确认框**

**- var flag = window.confirm("显示的内容");**

**- prompt()： 输入的对话框**

**- window.prompt("please input : ","0");**

**- window.prompt("在显示的内容","输入框里面的默认值");**

**- open() :　打开一个新的窗口**

**\*\* open("打开的新窗口的地址url","","窗口特征，比如窗口宽度和高度")**

**- 创建一个按钮，点击这个按钮，打开一个新的窗口**

**- window.open("hello.html","","width=200,height=100");**

**- close(): 关闭窗口(浏览器兼容性比较差)**

**- window.close();**

**- 做定时器**

**\*\* setInterval("js代码",毫秒数) 1秒=1000毫秒**

**- 表示每三秒，执行一次alert方法**

**window.setInterval("alert('123');",3000);**

**\*\* setTimeout("js代码",毫秒数)**

**- 表示在毫秒数之后执行，但是只会执行一次**

**- 表示四秒之后执行js代码，只会执行一次**

**window.setTimeout("alert('abc');",4000);**

**\*\* clearInterval(): 清除setInterval设置的定时器**

**var id1 = setInterval("alert('123');",3000);//通过setInterval会有一个返回值**

**clearInterval(id1);**

**\*\* clearTimeout() : 清除setTimeout设置的定时器**

**var id2 = setTimeout("alert('abc');",4000);**

**clearTimeout(id2);**

**8、js的dom对象（\*\*\*\*）**

**\* dom：document object model: 文档对象模型**

**\*\* 文档：**

**超文本文档（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型文档**

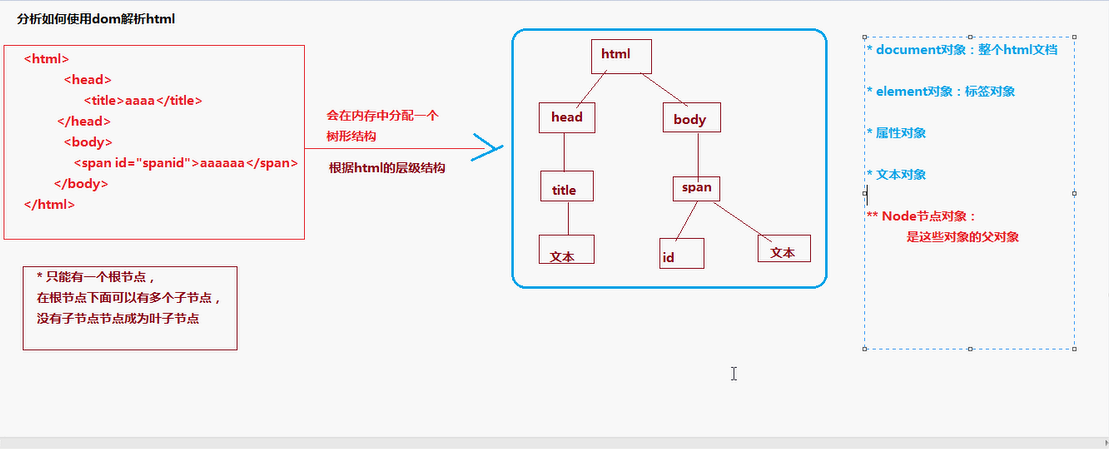
**\*\*\* 可以使用js里面的dom里面提供的对象，使用这些对象的属性和方法，对标记型文档进行操作**

**\*\*\* 想要对标记型文档进行操作，首先需要 对标记型文档里面的所有内容封装成对象**

**-- 需要把html里面的标签、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**

**- 画图分析，如何使用dom解析html**



**\*\*\* 解析过程**

**根据html的层级结构，在内存中分配一个树形结构，需要把html中的每部分封装成对象，**

**- document对象：整个文档**

**- element对象：标签对象**

**- 属性对象**

**- 文本对象**

**-- Node节点对象：这个对象是这些对象的父对象**

**\*\*\* 如果在对象里面找不到想要的方法，这个时候到Node对象里面去找**

**DOM模型有三种：**

**DOM level 1：将html文档封装成对象。**

**DOM level 2：在level 1的基础上添加新的功能，例如：对于事件和css样式的支持。**

**DOM level 3：支持xml1.0的一些新特性。**

**\* 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：专门指的是js的语法语句（ECMAScript）**

**9、document对象**

**\* 表示整个的文档**

**\*\* 常用方法**

**\*\*\*\* write()方法：**

**（1）向页面输出变量（值）**

**（2）向页面输出html代码**

**- var str = "abc";**

**document.write(str);**

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

**\*\*\*\* getElementById();**

**- 通过id得到元素（标签）**

**- //使用getElementById得到input标签**

**var input1 = document.getElementById("nameid"); //传递的参数是标签里面的id的值**

**//得到input里面的value值**

**alert(input1.name); //标签对象.属性名称**

**//向input里面设置一个值value**

**input1.value = "bbbbb";**

**\*\*\*\* getElementsByName();**

**- 通过标签的name的属性值得到标签**

**- 返回的是一个集合（数组）**

**- //使用getElementsByName得到input标签**

**var inputs = document.getElementsByName("name1"); //传递的参数是 标签里面的name的值**

**//alert(inputs.length);**

**//遍历数组**

**for(var i=0;i<inputs.length;i++) { //通过遍历数组，得到每个标签里面的具体的值**

**var input1 = inputs[i]; //每次循环得到input对象，赋值到input1里面**

**alert(input1.value); //得到每个input标签里面的value值**

**}**

**\*\*\*\* getElementsByTagName("标签名称");**

**- 通过标签名称得到元素**

**- //演示getElementsByTagName**

**var inputs1 = document.getElementsByTagName("input"); //传递的参数，是标签名称**

**//alert(inputs1.length);**

**//遍历数组，得到每个input标签**

**for(var m=0;m<inputs1.length;m++) {**

**//得到每个input标签**

**var input1 = inputs1[m];**

**//得到value值**

**alert(input1.value);**

**}**

**\*\*\*\* 注意地方**

**\*\*\*\* 只有一个标签，这个标签只能使用name获取到，这个使用，使用getElementsByName返回的是一个数组，**

**但是现在只有一个元素，这个时候不需要遍历，而是可以直接通过数组的下标获取到值**

**//通过name得到input标签**

**var inputs2 = document.getElementsByName("name11")[0];**

**alert(inputs2.value);**

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

**alert(inputss.value);**

**10、案例：window弹窗案例**

**- 实现过程**

**1、创建一个页面**

**\*\* 有两个输入项和一个按钮**

**\*\* 按钮上面有一个事件：弹出一个新窗口 open**

**2、创建弹出页面**

**\*\* 表格**

**\*\* 每一行有一个按钮和编号和姓名**

**\*\* 按钮上有一个事件：把当前的编号和姓名，赋值到第一个页面相应的位置**

**\*\*\*\*//实现s1方法**

**function s1(num1,name1) {**

**//需要把num1和name1赋值到window页面**

**//跨页面的操作 opener：得到创建这个窗口的窗口 得到window页面**

**var pwin = window.opener; //得到window页面**

**pwin.document.getElementById("numid").value = num1;**

**pwin.document.getElementById("nameid").value = name1;**

**//关闭窗口**

**window.close();**

**}**

**－　opener:属性，获取创建当前窗口的窗口**

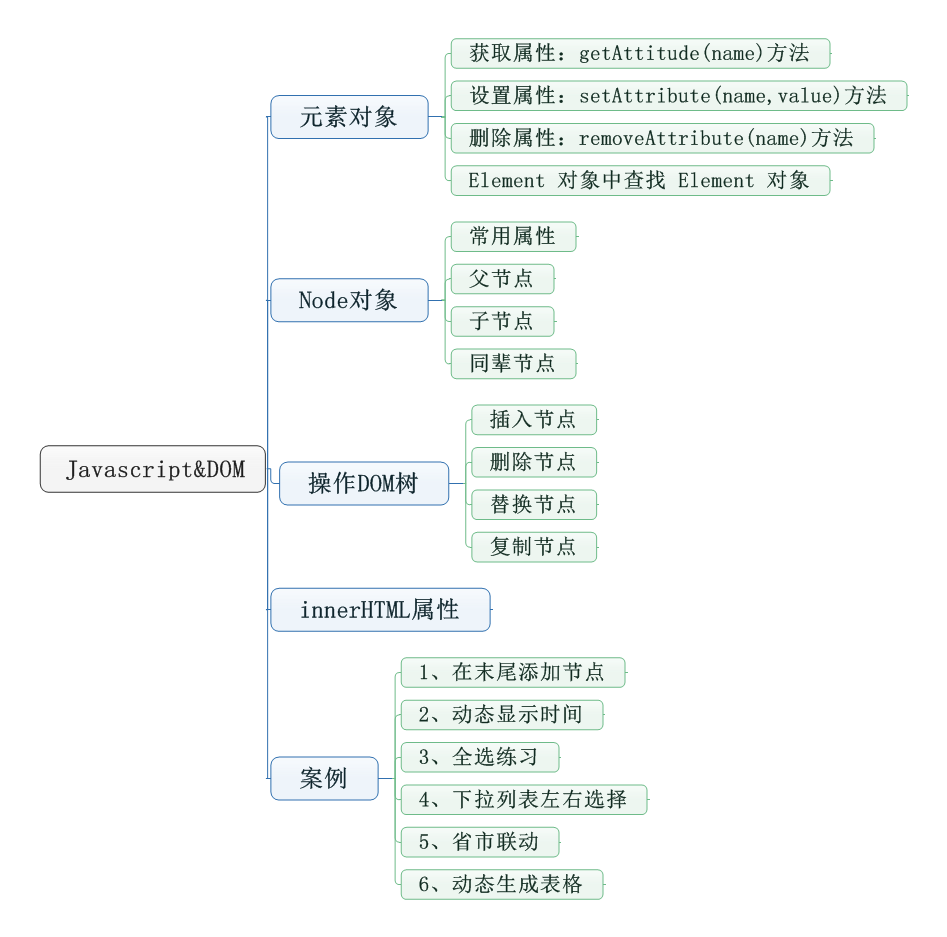
**- 做这个案例时候会有一个问题**

**\*\*\* 由于我们现在访问的是本地文件，js安全性，谷歌浏览器安全级别很高，不允许访问本地文件**

**\*\*\* 在实际开发中，没有这样的问题，实际中不可能访问本地的文件。**

**\*\*\*** [**http://www.baidu.com**](http://www.baidu.com)

**Day4**

****

**1、案例一：在末尾添加节点**

**第一步：获取到ul标签**

**var ul1 = document.getElementById("ulid");**

**第二步：创建li标签**

**document.createElement("标签名称")方法**

**第三步：创建文本**

**document.createTextNode("文本内容");**

**第四步：把文本添加到li下面**

**使用 appendChild方法**

**第五步：把里添加到ul末尾**

**使用 appendChild方法**

**2、元素对象（element对象）**

**\*\* 要操作element对象，首先必须要获取到element，**

**- 使用document里面相应的方法获取**

**\*\* 方法**

**\*\*\* 获取属性里面的值**

**getAttribute("属性名称")**

**- var input1 = document.getElementById("inputid");**

**//alert(input1.value);**

**alert(input1.getAttribute("value"));**

**\*\*\* 设置属性的值**

**input1.setAttribute("class","haha");**

**\*\*\* 删除属性**

**input1.removeAttribute("name");**

**\*\* 不能删除value**

**\*\*\* 想要获取标签下面的子标签**

**\*\* 使用属性 childNodes，但是这个属性兼容性很差**

**\*\* 获得标签下面子标签的唯一有效办法，使用getElementsByTagName方法**

**- var ul11 = document.getElementById("ulid1");**

**//获取ul下面的子标签**

**//var lis = ul11.childNodes;**

**//alert(lis.length);**

**var lis = ul11.getElementsByTagName("li");**

**alert(lis.length);**

**3、Node对象属性一**

**\* nodeName**

**\* nodeType**

**\* nodeValue**

**\* 使用dom解析html时候，需要html里面的标签，属性和文本都封装成对象**

**\* 标签节点对应的值**

**nodeType： 1**

**nodeName： 大写标签名称 比如SPAN**

**nodeValue: null**

**\* 属性节点对应的值**

**nodeType： 2**

**nodeName： 属性名称**

**nodeValue: 属性的值**

**\* 文本节点对应的值**

**nodeType： 3**

**nodeName： #text**

**nodeValue: 文本内容**

**//获取标签对象**

**var span1 = document.getElementById("spanid");**

**//alert(span1.nodeType); // 1**

**///alert(span1.nodeName); // SPAN**

**///alert(span1.nodeValue); // null**

**//属性**

**var id1 = span1.getAttributeNode("id");**

**//alert(id1.nodeType); // 2**

**//alert(id1.nodeName); // id**

**//alert(id1.nodeValue); // spanid**

**//获取文本**

**var text1 = span1.firstChild;**

**alert(text1.nodeType); // 3**

**alert(text1.nodeName); // #text**

**alert(text1.nodeValue); // 内容**

**4、Node对象的属性二**

**<ul>**

**<li>qqqqq</li>**

**<li>wwww</li>**

**</ul>**

**\* 父节点**

**- ul是li的父节点**

**- parentNode**

**- //得到li1**

**var li1 = document.getElementById("li1");**

**//得到ul**

**var ul1 = li1.parentNode;**

**alert(ul1.id);**

**\* 子节点**

**- li是ul的子节点**

**- childNodes：得到所有子节点，但是兼容性很差**

**- firstChild：获取第一个子节点**

**- //获取ul的第一个子节点 id=li1**

**//得到ul**

**var ul1 = document.getElementById("ulid");**

**//第一个子节点**

**var li1 = ul1.firstChild;**

**alert(li1.id);**

**- lastChild：获取最后一个子节点**

**//得到最后一个子节点**

**var li4 = ul1.lastChild;**

**alert(li4.id);**

**\* 同辈节点**

**- li直接关系是同辈节点**

**- nextSibling: 返回一个给定节点的下一个兄弟节点。**

**previousSibling：返回一个给定节点的上一个兄弟节点。**

**- //获取li的id是li3的上一个和下一个兄弟节点**

**var li3 = document.getElementById("li3");**

**//alert(li3.nextSibling.id);**

**alert(li3.previousSibling.id);**

**5、操作dom树**

**\*\* appendChild方法**

**- 添加子节点到末尾**

**- 特点：类似于剪切黏贴的效果**

**\*\* insertBefore(newNode,oldNode)方法**

**- 在某个节点之前插入一个新的节点**

**- 两个参数**

**\* 要插入的节点**

**\* 在谁之前插入**

**- 插入一个节点，节点不存在，创建**

**1、创建标签**

**2、创建文本**

**3、把文本添加到标签下面**

**- 代码**

**/\***

**1、获取到li13标签**

**2、创建li**

**3、创建文本**

**4、把文本添加到li下面**

**5、获取到ul**

**6、把li添加到ul下面（在<li>貂蝉</li>之前添加 <li>董小宛</li>）**

**\*/**

**//获取li3 标签**

**var li13 = document.getElementById("li13");**

**//创建li**

**var li15 = document.createElement("li");**

**//创建文本**

**var text15 = document.createTextNode("董小宛");**

**//把文本添加到li下面 appendChild**

**li15.appendChild(text15);**

**//获取到ul**

**var ul21 = document.getElementById("ulid21");**

**//在<li>貂蝉</li>之前添加 <li>董小宛</li>**

**//insertBefore(newNode,oldNode)**

**ul21.insertBefore(li15,li13);**

**\*\*\* 不存在 没有insertAfter()方法**

**\*\* removeChild方法：删除节点**

**- 通过父节点删除，不能自己删除自己**

**- /\***

**1、获取到li24标签**

**2、获取父节点ul标签**

**3、执行删除（通过父节点删除）**

**\*/**

**//获取li标签**

**var li24 = document.getElementById("li24");**

**//获取父节点**

**//两种方式 1、通过id获取 ； 2、通过属性 parentNode获取**

**var ul31 = document.getElementById("ulid31");**

**//删除（通过父节点）**

**ul31.removeChild(li24);**

**\*\* replaceChild(newNode,oldNode)方法: 替换节点**

**- 不能自己替换自己，通过父节点替换**

**- 两个参数**

**\*\* 第一个参数：新的节点（替换成的节点）**

**\*\* 第二个参数：旧的节点（被替换的节点）**

**- 代码**

**/\***

**1、获取到li34**

**2、创建标签li**

**3、创建文本**

**4、把文本添加到li下面**

**5、获取ul标签（父节点）**

**6、执行替换操作 （replaceChild(newNode,oldNode)）**

**\*/**

**//获取li34**

**var li34 = document.getElementById("li34");**

**//创建li**

**var li35 = document.createElement("li");**

**//创建文本**

**var text35 = document.createTextNode("张无忌");**

**//把文本添加到li下面**

**li35.appendChild(text35);**

**//获取ul**

**var ul41 = document.getElementById("ulid41");**

**//替换节点**

**ul41.replaceChild(li35,li34);**

**\*\* cloneNode(boolean): 复制节点**

**- //把ul列表复制到另外一个div里面**

**/\***

**1、获取到ul**

**2、执行复制方法 cloneNode方法复制 true**

**3、把复制之后的内容放到div里面去**

**\*\* 获取到div**

**\*\* appendChild方法**

**\*/**

**//获取ul**

**var ul41 = document.getElementById("ulid41");**

**//复制ul，放到类似剪切板里面**

**var ulcopy = ul41.cloneNode(true)**

**//获取到div**

**var divv = document.getElementById("divv");**

**//把副本放到div里面去**

**divv.appendChild(ulcopy);**

**\*\* 操作dom总结**

**\* 获取节点使用方法**

**getElementById()：通过节点的id属性，查找对应节点。**

**getElementsByName()：通过节点的name属性，查找对应节点。**

**getElementsByTagName()：通过节点名称，查找对应节点**

**\* 插入节点的方法**

**insertBefore方法：在某个节点之前插入**

**appendChild方法：在末尾添加，剪切黏贴**

**\* 删除节点方法**

**removeChild方法：通过父节点删除**

**\* 替换节点方法**

**replaceChild方法：通过父节点替换**

**6、innerHTML属性**

**\* 这个属性不是dom的组成部分，但是大多数浏览器都支持的属性**

**第一个作用：获取文本内容**

**\*\*\* //获取span标签**

**var span1 = document.getElementById("sid");**

**alert(span1.innerHTML);**

**第二个作用：向标签里面设置内容（可以是html代码）**

**\*\*\* //向div里面设置内容 <h1>AAAAA</h1>**

**//获取到div**

**var div11 = document.getElementById("div11");**

**//设置内容**

**div11.innerHTML = "<h1>AAAAA</h1>";**

**7、案例二：动态显示时间**

**\* 得到当前的时间**

**var date = new Date(); //得到不是常规的格式**

**var d1 = date.toLocaleString(); //格式化**

**\* 需要让页面每一秒获取时间**

**setInterval方法 定时器**

**\* 显示到页面上**

**每一秒向div里面写一次时间**

**\* 使用innerHTML属性**

**\* 代码**

**function getD1() {**

**//当前时间**

**var date = new Date();**

**//格式化**

**var d1 = date.toLocaleString();**

**//获取div**

**var div1 = document.getElementById("times");**

**div1.innerHTML = d1;**

**}**

**//使用定时器实现每一秒写一次时间**

**setInterval("getD1();",1000);**

**8、案例三：全选练习**

**\*\* 使用复选框上面一个属性判断是否选中**

**- checked属性**

**- checked=true：选中**

**- checked=false：不选中**

**\* 创建一个页面**

**\*\* 复选框和按钮**

**- 四个复选框表示爱好**

**- 还有一个复选框操作 全选和选不选，也有一个事件**

**\*\* 三个按钮，分别有事件**

**- 全选**

**- 全不选**

**- 反选**

**\* 全选操作**

**步骤：**

**/\***

**1、获取要操作的复选框**

**- 使用getElementsByName()**

**2、返回是数组，**

**- 属性 checked判断复选框是否选中**

**\*\*\* checked = true; //表示选中**

**\*\*\* checked = false;//表示不选中**

**- 遍历数组，得到的是每一个checkbox**

**\* 把每一个checkbox属性checked=true**

**\*/**

**\* 全不选操作**

**步骤**

**/\***

**1、获取到要操作的复选框**

**2、返回的是数组，遍历数组**

**3、得到每一个复选框**

**4、设置复选框的属性 checked=false**

**\*/**

**\* 反选操作**

**步骤**

**/\***

**1、获取到要操作的复选框**

**2、返回数组，遍历数组**

**3、得到每一个复选框**

**4、判断当前的复选框是选中还是不选中**

**- if(love1.checked == true) {}**

**5、如果选中，属性checked设置成false，否则，设置成true**

**\*/**

**\* 使用复选框实现全选和全不选**

**步骤**

**/\***

**1、得到上面那个复选框**

**- 通过id获取到**

**2、判断这个复选框是否是选中**

**- if条件，checked==true**

**3、如果是选中，下面是全选**

**4、如果不是选中，下面是全不选**

**\*/**

**9、案例四：下拉列表左右选择**

**\* 下拉选择框**

**<select>**

**<option>111</option>**

**<option>111</option>**

**</select>**

**\* 创建一个页面**

**\*\* 两个下拉选择框**

**- 设置属性 multiple属性**

**\*\* 四个按钮，有事件**

**\* 选中添加到右边**

**步骤**

**/\***

**1、获取select1里面的option**

**- getElementsByTagName()返回是数组**

**- 遍历数组，得到每一个option**

**2、判断option是否被选中**

**- 属性 selected，判断是否被选中**

**\*\* selected= true: 选中**

**\*\* selected= false：没有选择**

**3、如果是选中，把选择的添加到右边去**

**4、得到select2**

**4、添加选择的部分**

**- appendChild方法**

**\*/**

**\* 全部添加到右边**

**步骤**

**/\***

**1、获取第一个select下面的option对象**

**2、返回数组，遍历数组**

**3、得到每一个option对象**

**4、得到select2**

**5、添加到select2下面**

**- appendChild方法**

**\*/**

**\* 选中添加到左边**

**步骤**

**/\***

**1、获取select2里面的option对象**

**2、返回是数组，遍历数组**

**3、得到每一个option对象**

**4、判断option是否被选中**

**- if条件 属性 selected == true：选择**

**5、获取select1**

**6、添加到select1里面**

**- 使用appendChild方法**

**\*/**

**\* 全部添加到左边**

**步骤**

**/\***

**1、获取select2里面的option对象**

**2、返回是数组，遍历数组**

**3、得到每一个option对象**

**4、获取到select1**

**5、添加到select1里面**

**- 使用appendChild方法**

**\*/**

**10、案例五：省市联动**

**\* 创建一个页面，有两个下拉选择框**

**\* 在第一个下拉框里面有一个事件 ：改变事件 onchange事件**

**- 方法add1(this.value);表示当前改变的option里面的value值**

**\* 创建一个二维数组，存储数据**

**\* 每个数组中第一个元素是国家名称，后面的元素是国家里面的城市**

**\***

**/\***

**1、遍历二维数组**

**2、得到也是一个数组（国家对应关系）**

**3、拿到数组中的第一个值和传递过来的值做比较**

**4、如果相同，获取到第一个值后面的元素**

**5、得到city的select**

**6、添加过去（使用）appendChild方法**

**- 创建option（三步）**

**\*/**

**/\***

**由于每次都要向city里面添加option**

**第二次添加，会追加。**

**\* 每次添加之前，判断一下city里面是否有option，如果有，删除**

**\*/**

**11、案例六：动态生成表格**

**\*创建一个页面：两个输入框和一个按钮**

**\*代码和步骤**

**/\***

**1、得到输入的行和列的值**

**2、生成表格**

**\*\* 循环行**

**\*\* 在行里面循环单元格**

**3、显示到页面上**

**- 把表格的代码设置到div里面**

**- 使用innerHTML属性**

**\*/**

**//获取输入的行和列**

**var h = document.getElementById("h").value;**

**var l = document.getElementById("l").value;**

**//把表格代码放到一个变量里面**

**var tab = "<table border='1' bordercolor='blue'>";**

**//循环行**

**for(var i=1;i<=h;i++) {**

**tab += "<tr>";**

**//循环单元格**

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

**tab += "<td>aaaaaaa</td>"**

**}**

**tab += "</tr>";**

**}**

**tab += "</table>";**

**//alert(tab);**

**//得到div标签**

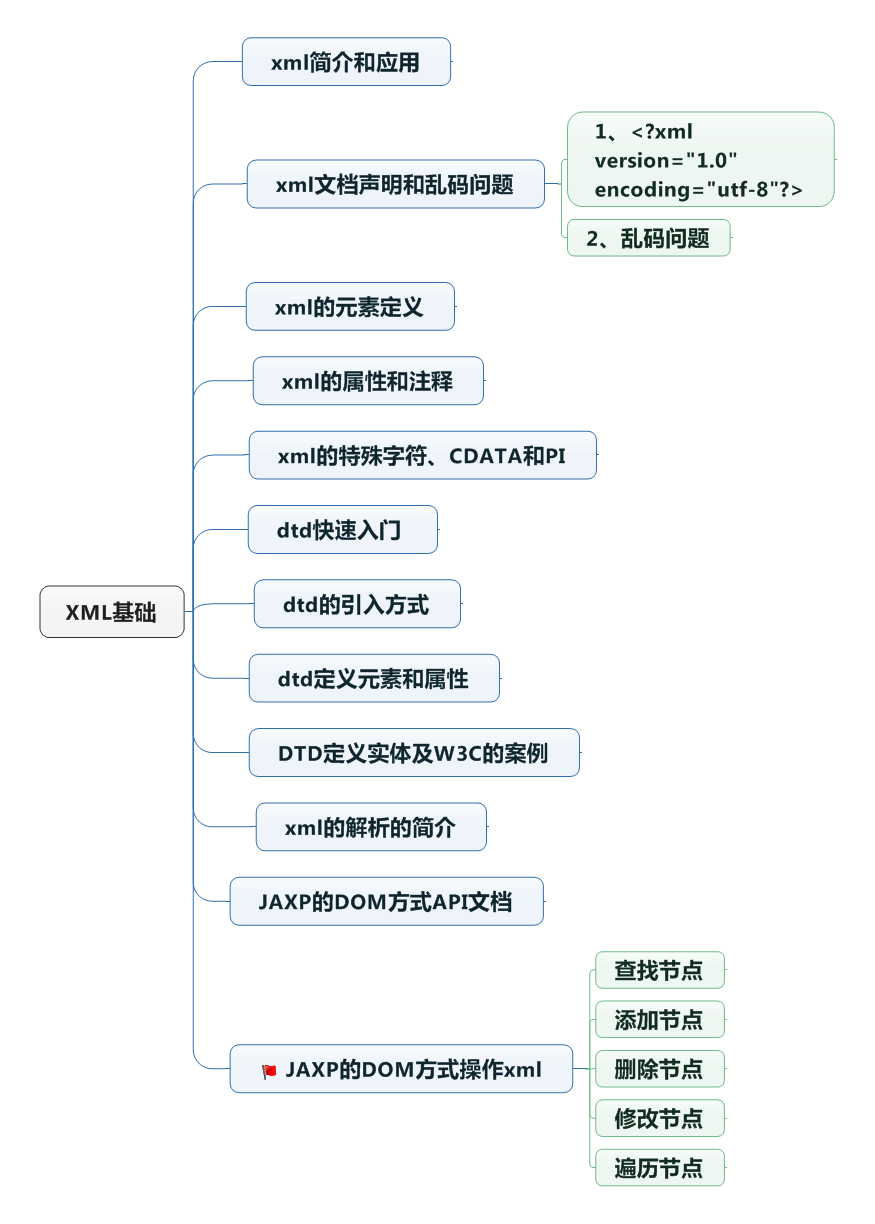
**var divv = document.getElementById("divv");**

**//把table的代码设置到div里面去**

**divv.innerHTML = tab;**

**}**

**DAY 5**

****

**1、表单提交方式**

**\* 使用submit提交**

**<form>**

**.....**

**<input type="submit" />**

**</form>**

**\* 使用button提交表单**

**- 代码**

**//实现提交方法**

**function form1() {**

**//获取form**

**var form1 = document.getElementById("form1");**

**//设置action**

**form1.action = "hello.html";**

**//提交form表单**

**form1.submit();**

**}**

**\* 使用超链接提交**

**- 代码z/**

**<a href="hello.html?username=123456">使用超链接提交</a>**

**\* onclick：鼠标点击事件**

**onchange：改变内容（一般和select一起使用）**

**onfocus:得到焦点 (ie5、某些版本的ie6)**

**onblur:失去焦点**

**2、xml的简介（了解）**

**\* eXtensible Markup Language：可扩展标记型语言**

**\*\* 标记型语言：html是标记型语言**

**- 也是使用标签来操作**

**\*\* 可扩展：**

**- html里面的标签是固定，每个标签都有特定的含义 <h1> <br/> <hr/>**

**- 标签可以自己定义,可以写中文的标签 <person></person、<猫></猫>**

**\* xml用途**

**\*\* html是用于显示数据，xml也可以显示数据（不是主要功能）**

**\*\* xml主要功能，为了存储数据**

**\* xml是w3c组织发布的技术**

**\* xml有两个版本 1.0 1.1**

**- 使用都是1.0版本，（1.1版本不能向下兼容）**

**3、xml的应用**

**\* 不同的系统之间传输数据**

**\*\* qq之间数据的传输**

**\*\* 画图分析过程**

**\* 用来表示生活中有关系的数据**

**\* 经常用在文件配置**

**\* 比如现在连接数据库 肯定知道数据库的用户名和密码，数据名称**

**\* 如果修改数据库的信息，不需要修改源代码，只要修改配置文件就可以了**

**4、xml的语法**

**（1）xml的文档声明（\*\*\*）**

**\* 创建一个文件 后缀名是 .xml**

**\* 如果写xml，第一步 必须要有 一个文档声明（写了文档声明之后，表示写xml文件的内容）**

**\*\* <?xml version="1.0" encoding="gbk"?>**

**\*\*\* 文档声明必须写在 第一行第一列**

**\* 属性**

**- version：xml的版本 1.0(使用) 1.1**

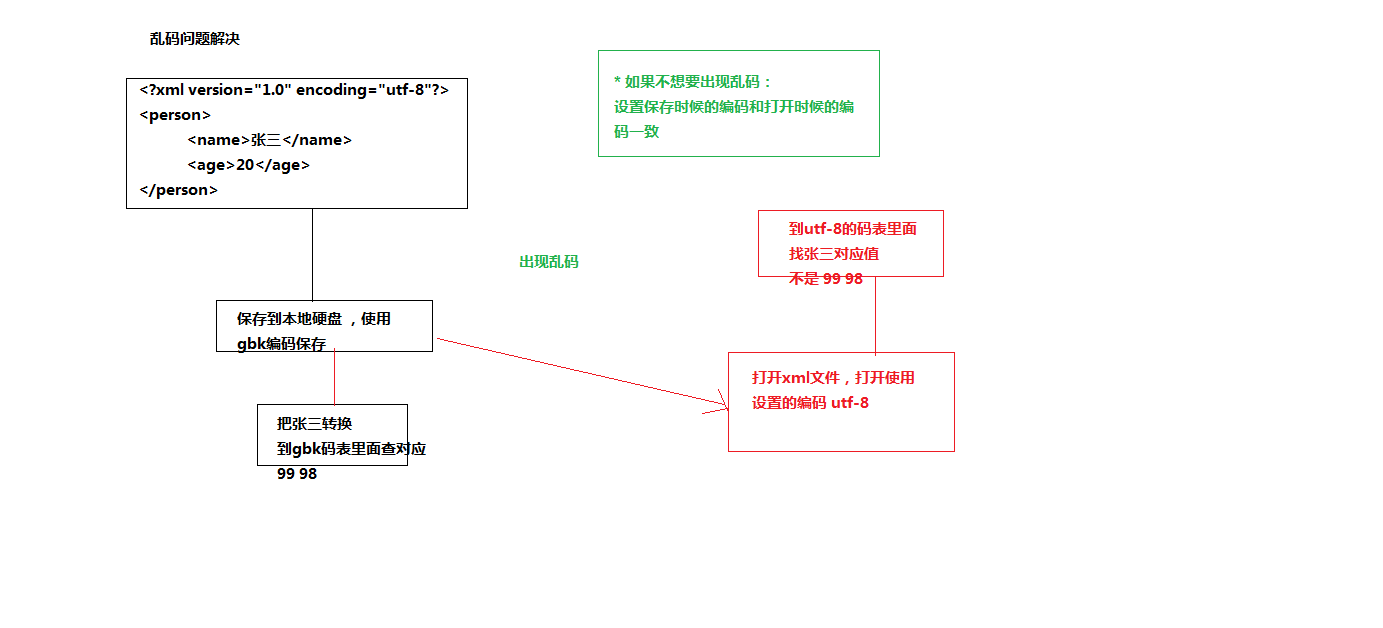
**- encoding：xml编码 gbk utf-8 iso8859-1(不包含中文)**

**- standalone：是否需要依赖其他文件 yes/no**

**\* xml的中文乱码问题解决**

**\*\* 画图分析乱码问题**

**\*\* 保存时候的编码和设置打开时候的编码一致，不会出现乱码**

****

**（2）定义元素（标签）（\*\*\*）**

**（3）定义属性（\*\*\*）**

**（4）注释（\*\*\*）**

**（5）特殊字符（\*\*\*）**

**（6）CDATA区（了解）**

**（7）PI指令（了解）**

**5、xml的元素（标签）定义（\*\*\*\*\*）**

**\*\* 标签定义**

**\*\* 标签定义有开始必须要有结束：<person></person>**

**\*\* 标签没有内容，可以在标签内结束 ; <aa/>**

**\*\* 标签可以嵌套，必须要合理嵌套**

**\*\*\* 合理嵌套 <aa><bb></bb></aa>**

**\*\*\* 不合理嵌套 <aa><bb></aa></bb>: 这种方式是不正确的**

**\*\* 一个xml中，只能有一个根标签，其他标签都是这个标签下面的标签**

**\*\* 在xml中把空格和换行都当成内容来解析，**

**\*\*\*\* 下面这两段代码含义是不一样的**

**\* <aa>1111111</aa>**

**\* <aa>**

**11111111111**

**</aa>**

**\*\* xml标签可以是中文**

**\*\* xml中标签的名称规则**

**（1）xml代码区分大小写**

**<p> <P>:这两个标签是不一样的**

**（2）xml的标签不能以数字和下划线(\_)开头**

**<2a> <\_aa>: 这样是不正确的**

**（3）xml的标签不能以xml、XML、Xml等开头**

**<xmla> <XmlB> <XMLC>: 这些都是不正确的**

**（4）xml的标签不能包含空格和冒号**

**<a b> <b:c> : 这些是不正确的**

**6、xml中属性的定义（\*\*\*\*\*）**

**\* html是标记型文档，可以有属性**

**\* xml也是标记型文档，可以有属性**

**\* <person id1="aaa" id2="bbb"></person>**

**\*\* 属性定义的要求**

**（1）一个标签上可以有多个属性**

**<person id1="aaa" id2="bbb"></person>**

**（2）属性名称不能相同**

**<person id1="aaa" id1="bbb"></person>：这个是不正确，不能有两个id1**

**（3）属性名称和属性值之间使用= ，属性值使用引号包起来 （可以是单引号，也可以是双引号 ）**

**（4）xml属性的名称规范和元素的名称规范一致**

**7、xml中的注释（\*\*\*\*\*）**

**\* 写法 <!-- xml的注释 -->**

**\*\* 注意的地方**

**\*\*\*\* 注释不能嵌套**

**<!-- <!-- --> -->**

**<!-- <!-- <sex>nv</sex>--> -->**

**\*\* 注释也不能放到第一行，第一行第一列必须放文档声明**

**8、xml中的特殊字符（\*\*\*\*\*）**

**\* 如果想要在xml中现在 a<b ,不能正常显示，因为把<当做标签**

**\* 如果就想要显示，需要对特殊字符 < 进行转义**

**\*\* < &lt;**

**> &gt;**

**9、CDATA区（了解）**

**\* 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，不需要转义了**

**\*\* 写法**

**<![CDATA[ 内容 ]]>**

**- 代码**

**<![CDATA[ <b>if(a<b && b<c && d>f) {}</b> ]]>**

**\*\* 把特殊字符，当做文本内容，而不是标签**

**10、PI指令（处理指令）（了解）**

**\* 可以在xml中设置样式**

**\* 写法： <?xml-stylesheet type="text/css" href="css的路径"?>**

**\* 设置样式，只能对英文标签名称起作用，对于中文的标签名称不起作用的。**

**\*\* xml的语法的总结**

**所有 XML 元素都须有关闭标签**

**XML 标签对大小写敏感**

**XML 必须正确地嵌套顺序**

**XML 文档必须有根元素(只有一个)**

**XML 的属性值须加引号**

**特殊字符必须转义 --- CDATA**

**XML 中的空格、回车换行会解析时被保留**

**11、xml的约束**

**\* 为什么需要约束？**

**\*\* 比如现在定义一个person的xml文件，只想要这个文件里面保存人的信息，比如name age等，但是如果在xml文件中**

**写了一个标签<猫>，发现可以正常显示，因为符合语法规范。但是猫肯定不是人的信息，xml的标签是自定义的，需要技术来**

**规定xml中只能出现的元素，这个时候需要约束。**

**\* xml的约束的技术 ： dtd约束 和 schema约束 （看懂）**

**12、dtd的快速入门**

**\* 创建一个文件 后缀名 .dtd**

**步骤：**

**（1）看xml中有多少个元素 ，有几个元素，在dtd文件中写几个 <!ELEMENT>**

**（2）判断元素是简单元素还是复杂元素**

**- 复杂元素：有子元素的元素**

**<!ELEMENT 元素名称 (子元素)>**

**- 简单元素：没有子元素**

**<!ELEMENT 元素名称 (#PCDATA)>**

**（3）需要在xml文件中引入dtd文件**

**<!DOCTYPE 根元素名称 SYSTEM "dtd文件的路径">**

**\*\* 打开xml文件使用浏览器打开的，浏览器只负责校验xml的语法，不负责校验约束**

**\*\* 如果想要校验xml的约束，需要使用工具（myeclipse工具）**

**\*\* 打开myeclipse开发工具**

**\*\*\* 创建一个项目 day05**

**\*\*\* 在day05的src目录下面创建一个xml文件和一个dtd文件**

**\*\*\* 当xml中引入dtd文件之后，比如只能出现name age，多写了一个a，会提示出错**

**13、dtd的三种引入方式**

**（1）引入外部的dtd文件**

**<!DOCTYPE 根元素名称 SYSTEM "dtd路径">**

**（2）使用内部的dtd文件**

**- <!DOCTYPE 根元素名称 [**

**<!ELEMENT person (name,age)>**

**<!ELEMENT name (#PCDATA)>**

**<!ELEMENT age (#PCDATA)>**

**]>**

**（3）使用外部的dtd文件（网络上的dtd文件）**

**<!DOCTYPE 根元素 PUBLIC "DTD名称" "DTD文档的URL">**

**- 后面学到框架 struts2 使用配置文件 使用 外部的dtd文件**

**- <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"**

**"http://struts.apache.org/dtds/struts-2.0.dtd">**

**14、使用dtd定义元素**

**\* 语法： <!ELEMENT 元素名 约束>**

**\* 简单元素：没有子元素的元素**

**<!ELEMENT name (#PCDATA)>**

**\*\*\* (#PCDATA): 约束name是字符串类型**

**\*\*\* EMPTY : 元素为空（没有内容）**

**-<!ELEMENT sex EMPTY>**

**- <sex></sex>**

**\*\*\* ANY:任意**

**\* 复杂元素：**

**<!ELEMENT person (name,age,sex,school)>**

**- 子元素只能出现一次**

**\* <!ELEMENT 元素名称 (子元素)>**

**\* 表示子元素出现的次数**

**+ : 表示一次或者多次**

**? ：表示零次或者一次**

**\* ：表示零次或者多次**

**\* 子元素直接使用逗号进行隔开 ,**

**\*\* 表示元素出现的顺序**

**\* 子元素直接使用|隔开**

**\*\* 表示元素只能出现其中的任意一个**

**15、使用dtd定义属性**

**\* 语法： <!ATTLIST 元素名称**

**属性名称 属性类型 属性的约束**

**>**

**\* 属性类型**

**- CDATA: 字符串**

**- <!ATTLIST birthday**

**ID1 CDATA #REQUIRED**

**>**

**- 枚举 ： 表示只能在一定的范围内出现值，但是只能每次出现其中的一个**

**\*\* 红绿灯效果**

**\*\* (aa|bb|cc)**

**- <!ATTLIST age**

**ID2 (AA|BB|CC) #REQUIRED**

**>**

**- ID: 值只能是字母或者下划线开头**

**- <!ATTLIST name**

**ID3 ID #REQUIRED**

**>**

**\* 属性的约束**

**- #REQUIRED：属性必须存在**

**- #IMPLIED：属性可有可无**

**- #FIXED: 表示一个固定值 #FIXED "AAA"**

**- 属性的值必须是设置的这个固定值**

**- <!ATTLIST sex**

**ID4 CDATA #FIXED "ABC"**

**>**

**- 直接值**

**\* 不写属性，使用直接值**

**\* 写了属性，使用设置那个值**

**- <!ATTLIST school**

**ID5 CDATA "WWW"**

**>**

**16、实体的定义**

**\* 语法： <!ENTITY 实体名称 "实体的值">**

**\*\*\* <!ENTITY TEST "HAHAHEHE">**

**\*\*\* 使用实体 &实体名称; 比如 &TEST;**

**\*\* 注意**

**\* 定义实体需要写在内部dtd里面，**

**如果写在外部的dtd里面，有某些浏览器下，内容得不到**

**Schema：**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**schema xmlns="http://www.w3.org/2001/XMLSchema"  
targetNamespace="http://www.itcast.cn/20151111"  
elementFormDefault="qualified"**>  
 <**element name="person"**>  
 <**complexType**>  
 <**sequence**>  
 *<!-- <all> -->  
 <!-- <choice> -->* <**element name="name" type="string" maxOccurs="unbounded"**></**element**>  
 <**element name="age" type="int"**></**element**>  
 *<!-- </choice> -->  
 <!-- </all> -->* </**sequence**>  
 <**attribute name="id1" type="int" use="required"**></**attribute**>  
 </**complexType**>  
 </**element**>  
</**schema**>

**17、xml的解析的简介（写到java代码）（\*\*\*今天最重要的内容\*\*\*\*\*）**

**\* xml是标记型文档**

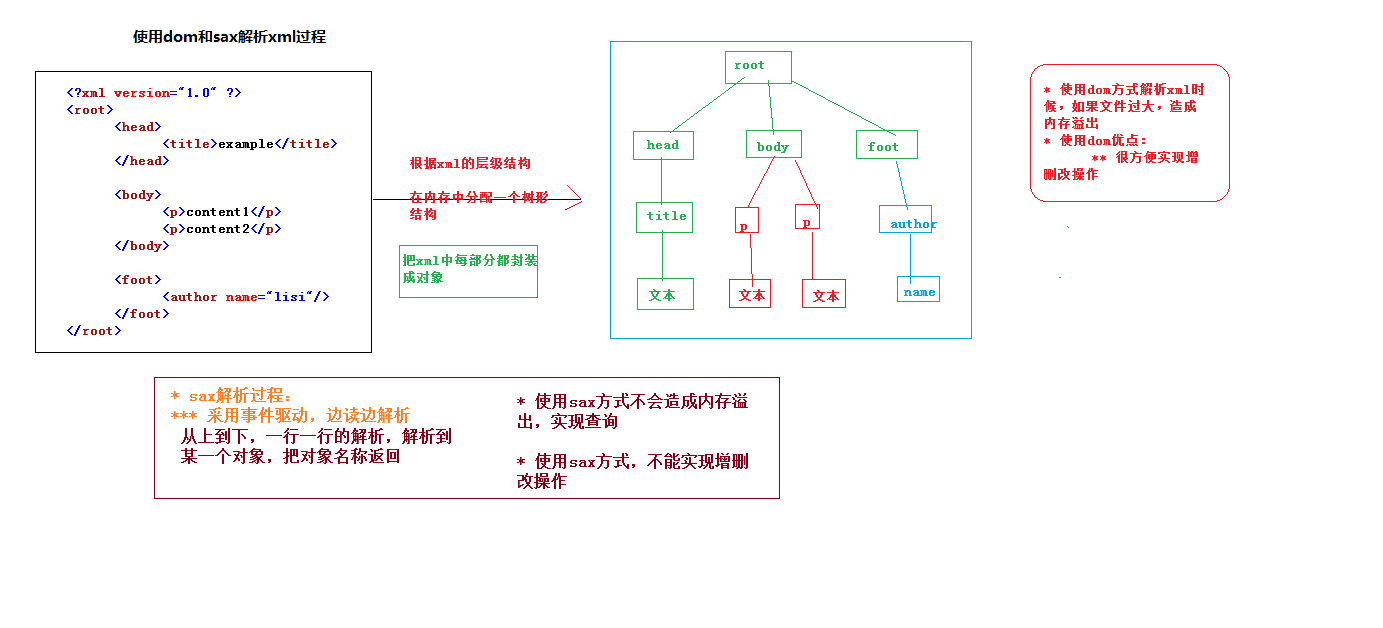
**\* js使用dom解析标记型文档？**

**- 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，属性和文本都封装成对象**

**- document对象、element对象、属性对象、文本对象、Node节点对象**

**\* xml的解析方式（技术）：dom 和 sax**

**\*\* 画图分析使用dom和sax解析xml过程**

****

**\*\*\* dom解析和sax解析区别：**

**\*\* dom方式解析**

**\* 根据xml的层级结构在内存中分配一个树形结构，把xml的标签，属性和文本都封装成对象**

**\* 缺点：如果文件过大，造成内存溢出**

**\* 优点：很方便实现增删改操作**

**\*\* sax方式解析**

**\* 采用事件驱动，边读边解析**

**- 从上到下，一行一行的解析，解析到某一个对象，返回对象名称**

**\* 缺点：不能实现增删改操作**

**\* 优点：如果文件过大，不会造成内存溢出，方便实现查询操作**

DOM会一行一行的读取XML文档，最终会把XML文档所有数据存放到Document对象中。SAX也是一行一行的读取XML文档，但是当XML文档读取结束后，SAX不会保存任何数据，同时整个解析XML文档的工作也就结束了。

但是，SAX在读取一行XML文档数据后，就会给感兴趣的用户一个通知！例如当SAX读取到一个元素的开始时，会通知用户当前解析到一个元素的开始标签。而用户可以在整个解析的过程中完成自己的业务逻辑，当SAX解析结束，不会保存任何XML文档的数据。

优先：使用SAX，不会占用大量内存来保存XML文档数据，效率也高。

缺点：当解析到一个元素时，上一个元素的信息已经丢弃，也就是说没有保存元素与元素之间的结构关系，这也大大限制了SAX的使用范围。如果只是想查询XML文档中的数据，那么使用SAX是最佳选择！

**\* 想要解析xml，首先需要解析器**

**\*\* 不同的公司和组织提供了 针对dom和sax方式的解析器，通过api方式提供**

**\*\*\* sun公司提供了针对dom和sax解析器 jaxp**

**\*\*\* dom4j组织，针对dom和sax解析器 dom4j（\*\*\* 实际开发中\*\*\*\*）**

**\*\*\* jdom组织，针对dom和sax解析器 jdom**

**18、jaxp的api的查看**

**\*\* jaxp是javase的一部分**

**\*\* jaxp解析器在jdk的javax.xml.parsers包里面**

**\*\* 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，**

**此类的实例可以从 DocumentBuilderFactory.newDocumentBuilder() 方法获取**

**- 一个方法，可以解析xml parse("xml路径") 返回是 Document 整个文档**

**- 返回的document是一个接口，父节点是Node，如果在document里面找不到想要的方法，到Node里面去找**

**- 在document里面方法**

**getElementsByTagName(String tagname)**

**-- 这个方法可以得到标签**

**-- 返回集合 NodeList**

**createElement(String tagName)**

**-- 创建标签**

**createTextNode(String data)**

**-- 创建文本**

**appendChild(Node newChild)**

**-- 把文本添加到标签下面**

**removeChild(Node oldChild)**

**-- 删除节点**

**getParentNode()**

**-- 获取父节点**

**NodeList list**

**- getLength() 得到集合的长度**

**- item(int index)下标取到具体的值**

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

**list.item(i)**

**}**

**getTextContent()**

**- 得到标签里面的内容**

**DocumentBuilderFactory： 解析器工厂**

**- 这个类也是一个抽象类，不能new**

**newInstance() 获取 DocumentBuilderFactory 的实例。**

**\*\*\* sax：**

**SAXParser：解析器类**

**SAXParserFactory: 解析器工厂**

**19、使用jaxp实现查询操作**

**\*\*\* 查询xml中所有的name元素的值**

**\* 步骤**

**//查询所有name元素的值**

**/\***

**\* 1、创建解析器工厂**

**DocumentBuilderFactory.newInstance();**

**\* 2、根据解析器工厂创建解析器**

**builderFactory.newDocumentBuilder();**

**\* 3、解析xml返回document**

**\* Document document = builder.parse("src/person.xml");**

**\* 4、得到所有的name元素**

**使用document.getElementsByTagName("name");**

**\* 5、返回集合，遍历集合，得到每一个name元素**

**- 遍历 getLength() item()**

**- 得到元素里面值 使用 getTextContent()**

**\* \*/**

**\*\*\* 查询xml中第一个name元素的值**

**\* 步骤**

**/\***

**\* 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**

**\***

**\* 4、得到所有name元素**

**\* 5、使用返回集合，里面方法 item，下标获取具体的元素**

**NodeList.item(下标)： 集合下标从0开始**

**\* 6、得到具体的值，使用 getTextContent方法**

**\***

**\* \*/**

**20、使用jaxp添加节点**

**\*\*\* 在第一个p1下面（末尾）添加 <sex>nv</sex>**

**\*\*步骤**

**/\***

**\* 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**

**\***

**\* 4、得到第一个p1**

**\* - 得到所有p1，使用item方法下标得到**

**\* 5、创建sex标签 createElement**

**\* 6、创建文本 createTextNode**

**\* 7、把文本添加到sex下面 appendChild**

**\* 8、把sex添加到第一个p1下面　appendChild**

**\***

**\* 9、回写xml**

**\* \*/TransformerFactory transformerFactory = TransformerFactory.newInstance();**

**Transformer transformer = transformerFactory.newTransformer();**

**transformer.transform(new DOMSource(document), new StreamResult("src/person.xml"));**

**21、使用jaxp修改节点**

**\*\*\* 修改第一个p1下面的sex内容是nan**

**\*\* 步骤**

**/\***

**\* 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**

**\***

**\* 4、得到sex item方法**

**\* 5、修改sex里面的值**

**\*\*\* setTextContent方法**

**\***

**\* 6、回写xml**

**\* \*/**

**22、使用jaxp删除节点**

**\*\*\* 删除<sex>nan</sex>节点**

**\*\* 步骤**

**/\***

**\* 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**

**\***

**\* 4、获取sex元素**

**\* 5、获取sex的父节点 使用getParentNode方法**

**\* 6、删除使用父节点删除 removeChild方法**

**\***

**\* 7、回写xml**

**\* \*/**

**23、使用jaxp遍历节点**

**\*\* 把xml中的所有元素名称打印出来**

**\*\* 步骤**

**/\***

**\* 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**

**\***

**\* ====使用递归实现=====**

**\* 4、得到根节点**

**\* 5、得到根节点子节点**

**\* 6、得到根节点子节点的子节点**

**\* \*/**

**\*\* 遍历的方法**

**//递归遍历的方法**

**private static void list1(Node node) {**

**//判断是元素类型时候才打印**

**if(node.getNodeType() == Node.ELEMENT\_NODE) {**

**System.out.println(node.getNodeName());**

**}**

**//得到一层子节点**

**NodeList list = node.getChildNodes();**

**//遍历list**

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

**//得到每一个节点**

**Node node1 = list.item(i);**

**//继续得到node1的子节点**

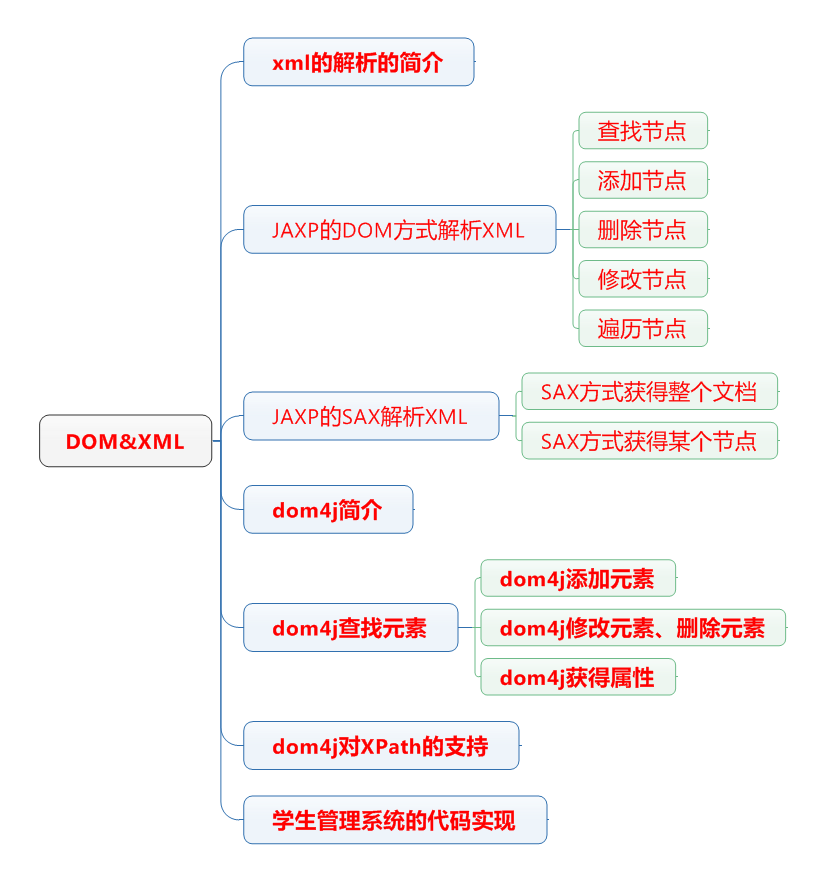
**//node1.getChildNodes()**

**list1(node1);**

**}**

**}**

**DAY6**

****

**1、schema约束**

**dtd语法： <!ELEMENT 元素名称 约束>**

**\*\* schema符合xml的语法，xml语句**

**\*\* 一个xml中可以有多个schema，多个schema使用名称空间区分（类似于java包名）**

**\*\* dtd里面有PCDATA类型，但是在schema里面可以支持更多的数据类型**

**\*\*\* 比如 年龄 只能是整数，在schema可以直接定义一个整数类型**

**\*\*\* schema语法更加复杂，schema目前不能替代dtd**

**2、schema的快速入门**

**\* 创建一个schema文件 后缀名是 .xsd**

**\*\* 根节点 <schema>**

**\*\* 在schema文件里面**

**\*\* 属性 xmlns="http://www.w3.org/2001/XMLSchema"**

**- 表示当前xml文件是一个约束文件**

**\*\* targetNamespace="http://www.itcast.cn/20151111"**

**- 使用schema约束文件，直接通过这个地址引入约束文件**

**\*\* elementFormDefault="qualified"**

**步骤**

**（1）看xml中有多少个元素**

**<element>**

**（2）看简单元素和复杂元素**

**\* 如果复杂元素**

**<complexType>**

**<sequence>**

**子元素**

**</sequence>**

**</complexType>**

**（3）简单元素，写在复杂元素的**

**<element name="person">**

**<complexType>**

**<sequence>**

**<element name="name" type="string"></element>**

**<element name="age" type="int"></element>**

**</sequence>**

**</complexType>**

**</element>**

**（4）在被约束文件里面引入约束文件**

**<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xmlns="http://www.itcast.cn/20151111"**

**xsi:schemaLocation="http://www.itcast.cn/20151111 1.xsd">**

**\*\* xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**-- 表示xml是一个被约束文件**

**\*\* xmlns="http://www.itcast.cn/20151111"**

**-- 是约束文档里面 targetNamespace**

**\*\* xsi:schemaLocation="http://www.itcast.cn/20151111 1.xsd">**

**-- targetNamespace 空格 约束文档的地址路径**

**\* <sequence>：表示元素的出现的顺序**

**<all>: 元素只能出现一次**

**<choice>：元素只能出现其中的一个**

**maxOccurs="unbounded"： 表示元素的出现的次数**

**<any></any>:表示任意元素**

**\* 可以约束属性**

**\* 写在复杂元素里面**

**\*\*\*写在　</complexType>之前**

**－－**

**<attribute name="id1" type="int" use="required"></attribute>**

**- name: 属性名称**

**- type：属性类型 int stirng**

**- use：属性是否必须出现 required**

**\* 复杂的schema约束**

**<company xmlns = "http://www.example.org/company"**

**xmlns:dept="http://www.example.org/department"**

**xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://www.example.org/company company.xsd http://www.example.org/department department.xsd"**

**>**

**\* 引入多个schema文件，可以给每个起一个别名**

**<employee age="30">**

**<!-- 部门名称 -->**

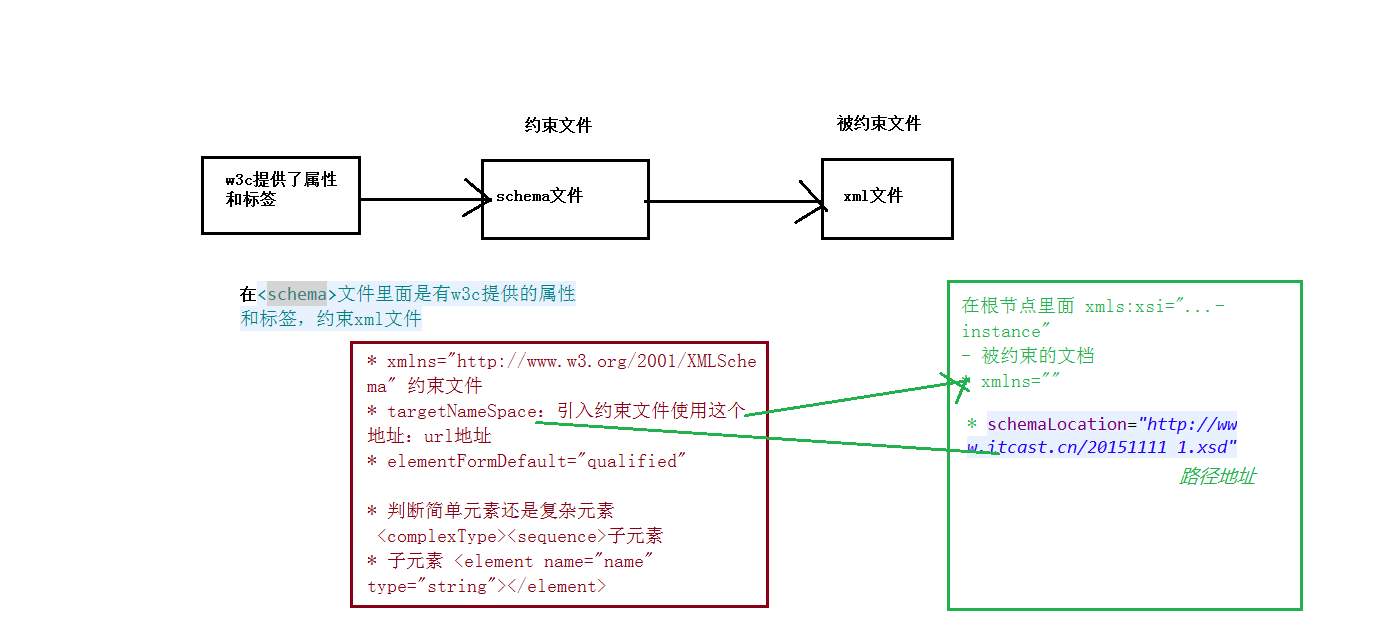
**<dept:name>100</dept:name>**

**\* 想要引入部门的约束文件里面的name，使用部门的别名 detp:元素名称**

**<!-- 员工名称 -->**

**<name>王晓晓</name>**

**</employee>**

****

**3、sax解析的原理（\*\*\*\*\*\*\*\*）**

**\* 解析xml有两种技术 dom 和sax**

**\* 根据xml的层级结构在内存中分配一个树形结构**

**\*\* 把xml中标签，属性，文本封装成对象**

**\* sax方式：事件驱动，边读边解析**

**\* 在javax.xml.parsers包里面**

**\*\* SAXParser**

**此类的实例可以从 SAXParserFactory.newSAXParser() 方法获得**

**- parse(File f, DefaultHandler dh)**

**\* 两个参数**

**\*\* 第一个参数：xml的路径**

**\*\* 事件处理器**

**\*\* SAXParserFactory**

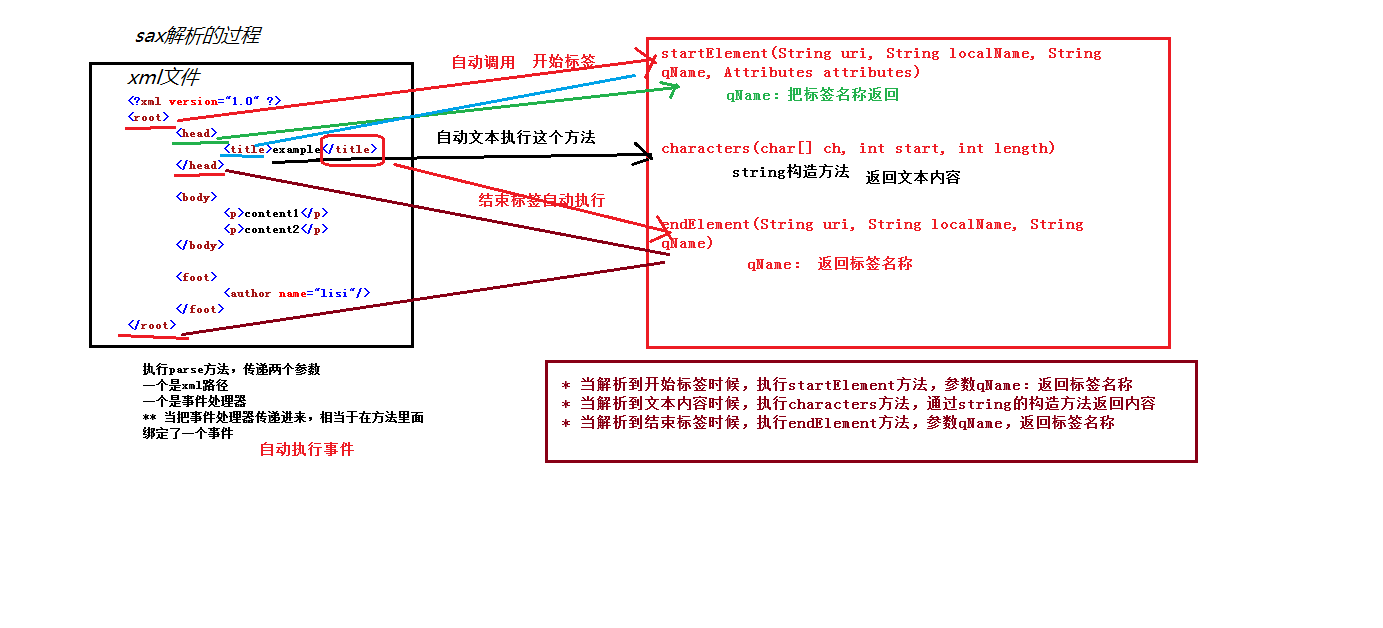
**实例 newInstance() 方法得到**

**\* 画图分析一下sax执行过程**

**\* 当解析到开始标签时候，自动执行startElement方法**

**\* 当解析到文本时候，自动执行characters方法**

**\* 当解析到结束标签时候，自动执行endElement方法**

****

**4、使用jaxp的sax方式解析xml（\*\*会写\*\*\*）？？没看**

**\* sax方式不能实现增删改操作，只能做查询操作**

**\*\* 打印出整个文档**

**\*\*\* 执行parse方法，第一个参数xml路径，第二个参数是 事件处理器**

**\*\*\* 创建一个类，继承事件处理器的类，**

**\*\*\*重写里面的三个方法**

**\* 获取到所有的name元素的值**

**\*\* 定义一个成员变量 flag= false**

**\*\* 判断开始方法是否是name元素，如果是name元素，把flag值设置成true**

**\*\* 如果flag值是true，在characters方法里面打印内容**

**\*\* 当执行到结束方法时候，把flag值设置成false**

**\* 获取第一个name元素的值**

**\*\* 定义一个成员变量 idx=1**

**\*\* 在结束方法时候，idx+1 idx++**

**\*\* 想要打印出第一个name元素的值，**

**- 在characters方法里面判断，**

**-- 判断flag=true 并且 idx==1，在打印内容**

**5、使用dom4j解析xml（重要）**

**\* dom4j，是一个组织，针对xml解析，提供解析器 dom4j**

**\* dom4j不是javase的一部分，想要使用第一步需要怎么做？**

**\*\*\* 导入dom4j提供jar包**

**-- 创建一个文件夹 lib**

**-- 复制jar包到lib下面，**

**-- 右键点击jar包，build path -- add to build path**

**-- 看到jar包，变成奶瓶样子，表示导入成功**

**\* 得到document**

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

**2、 Document document = 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里面去找**

**\* document里面的方法 3、getRootElement() ：获取根节点 返回的是Element**

**\*4、 Element也是一个接口，父接口是Node**

**\* element(qname)**

**\*\* 表示获取标签下面的第一个子标签**

**\*\* qname：标签的名称**

**\* elements(qname)**

**\*\* 获取标签下面是这个名称的所有子标签（一层）**

**\*\* qname：标签名称**

**\* elements()**

**\*\* 获取标签下面的所有一层子标签**

**\*5回写xml**

OutputFormat format = OutputFormat.*createPrettyPrint*();  
XMLWriter xmlWriter = **new** XMLWriter(**new** FileOutputStream(path), format);  
xmlWriter.write(document);  
xmlWriter.close();

**6、使用dom4j查询xml**

**\* 解析是从上到下解析**

**\* 查询所有name元素里面的值**

**/\***

**1、创建解析器**

**2、得到document**

**3、得到根节点 getRootElement() 返回Element**

**4、得到所有的p1标签**

**\* elements("p1") 返回list集合**

**\* 遍历list得到每一个p1**

**5、得到name**

**\* 在p1下面执行 element("name")方法 返回Element**

**6、得到name里面的值**

**\*5、 getText方法得到值**

**\*/**

**public class** Dom4jUtils {  
   
 **public static final** String ***PATH*** = **"src/p1.xml"**;  
  
 *//返回document* **public static** Document getDocument(String path) {  
 **try** {  
 *//创建解析器* SAXReader reader = **new** SAXReader();  
 *//得到document* Document document = reader.read(path);  
 **return** document;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
   
 *//回写xml的方法* **public static void** xmlWriters(String path,Document document) {  
 **try** {  
 OutputFormat format = OutputFormat.*createPrettyPrint*();  
 XMLWriter xmlWriter = **new** XMLWriter(**new** FileOutputStream(path), format);  
 xmlWriter.write(document);  
 xmlWriter.close();  
 }**catch**(Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

*//获取第一个p1里面的属性id1的值***public static void** getValues() **throws** Exception {  
 */\*  
 \* 1、得到document  
 \* 2、得到根节点  
 \* 3、得到第一个p1元素  
 \* 4、得到p1里面的属性值  
 \* \*/  
 //得到document* Document document = Dom4jUtils.*getDocument*(Dom4jUtils.***PATH***);  
 *//得到根节点* Element root = document.getRootElement();  
 *//得到第一个p1元素* Element p1 = root.element(**"p1"**);  
 *//得到p1里面的属性值* String value = p1.attributeValue(**"id1"**);  
 System.***out***.println(value);  
}

*//修改age的值*age.setText(**"300"**);

*//创建元素 使用*Element school = DocumentHelper.*createElement*(**"school"**);  
*//在school下面创建文本*school.setText(**"ecit"**);

*//在p1下面直接添加元素*Element sex1 = p1.addElement(**"sex"**);  
*//在sex下面添加文本*sex1.setText(**"nv"**);

*//得到p1下面的name元素*Element name1 = p1.element(**"name"**);  
*//得到name的值*String s1 = name1.getText();

**\* 查询第一个name元素的值**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\* 3、得到根节点**

**\***

**\* 4、得到第一个p1元素**

**\*\* element("p1")方法 返回Element**

**\* 5、得到p1下面的name元素**

**\*\* element("name")方法 返回Element**

**\* 6、得到name元素里面的值**

**\*\* getText方法**

**\* \*/**

**\* 获取第二个name元素的值**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\* 3、得到根节点**

**\***

**\* 4、得到所有的p1**

**\*\* 返回 list集合**

**\* 5、遍历得到第二个p1**

**\*\* 使用list下标得到 get方法，集合的下标从 0 开始，想要得到第二个值，下标写 1**

**\* 6、得到第二个p1下面的name**

**\*\* element("name")方法 返回Element**

**\* 7、得到name的值**

**\*\* getText方法**

**\* \*/**

**7、使用dom4j实现添加操作**

**\* 在第一个p1标签末尾添加一个元素 <sex>nv</sex>**

**\* 步骤**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\* 3、得到根节点**

**\***

**\* 4、获取到第一个p1**

**\* 使用element方法**

**\* 5、在p1下面添加元素**

**\* 在p1上面直接使用 addElement("标签名称")方法 返回一个Element**

**\* 6、在添加完成之后的元素下面添加文本**

**\* 在sex上直接使用 setText("文本内容")方法**

**\* 7、回写xml**

**\* 格式化 OutputFormat,使用 createPrettyPrint方法，表示一个漂亮的格式**

**\* 使用类XMLWriter 直接new 这个类 ，传递两个参数**

**\*\*\* 第一个参数是xml文件路径 new FileOutputStream("路径")**

**\*\*\* 第二个参数是格式化类的值**

**\* \*/**

**8、使用dom4j实现在特定位置添加元素**

**\* 在第一个p1下面的age标签之前添加 <school>ecit.edu.cn</schlool>**

**\* 步骤**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\* 3、得到根节点**

**\* 4、获取到第一个p1**

**\***

**\* 5、获取p1下面的所有的元素**

**\* \*\* elements()方法 返回 list集合**

**\* \*\* 使用list里面的方法，在特定位置添加元素**

**\* \*\* 首先创建元素 在元素下面创建文本**

**- 使用DocumentHelper类方法createElement创建标签**

**- 把文本添加到标签下面 使用 setText("文本内容")方法**

**\* \*\*\* list集合里面的 add(int index, E element)**

**\* - 第一个参数是 位置 下标，从0开始**

**\* - 第二个参数是 要添加的元素**

**\* 6、回写xml**

**\* \*/**

**\*\* 可以对得到document的操作和 回写xml的操作，封装成方法**

**\*\* 也可以把传递的文件路径，封装成一个常量**

**\*\*\* 好处：可以提高开发速度，可以提交代码可维护性**

**- 比如想要修改文件路径（名称），这个时候只需要修改常量的值就可以了，其他代码不需要做任何改变**

**9、使用dom4j实现修改节点的操作**

**\* 修改第一个p1下面的age元素的值 <age>30</age>**

**\* 步骤**

**/\***

**\* 1、得到document**

**\* 2、得到根节点，然后再得到第一个p1元素**

**\* 3、得到第一个p1下面的age**

**element("")方法**

**\* 4、修改值是 30**

**\* \* 使用setText("文本内容")方法**

**\* 5、回写xml**

**\***

**\* \*/**

**10、使用dom4j实现删除节点的操作**

**\* 删除第一个p1下面的<school>ecit</school>元素**

**\* 步骤**

**/\***

**\* 1、得到document**

**\* 2、得到根节点**

**\* 3、得到第一个p1标签**

**\* 4、得到第一个p1下面的school元素**

**\* 5、删除（使用p1删除school）**

**\* \* 得到school的父节点**

**- 第一种直接得到p1**

**- 使用方法 getParent方法得到**

**\* 删除操作**

**- 在p1上面执行remove方法删除节点**

**\* 6、回写xml**

**\* \*/**

**11、使用dom4j获取属性的操作**

**\* 获取第一个p1里面的属性id1的值**

**\* 步骤**

**/\***

**\* 1、得到document**

**\* 2、得到根节点**

**\* 3、得到第一个p1元素**

**\* 4、得到p1里面的属性值**

**- p1.attributeValue("id1");**

**- 在p1上面执行这个方法，里面的参数是属性名称**

**\* \*/**

**12、使用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： 表示一层一层的，AAA下面 DDD下面的BBB**

**\* 第二种形式**

**//BBB： 表示和这个名称相同，表示只要名称是BBB，都得到**

**\* 第三种形式**

**/\*: 所有元素**

**\* 第四种形式**

**\*\*　BBB[1]：　表示第一个BBB元素**

**××　BBB[last()]：表示最后一个BBB元素**

**\* 第五种形式**

**\*\* //BBB[@id]： 表示只要BBB元素上面有id属性，都得到**

**\* 第六种形式**

**\*\* //BBB[@id='b1'] 表示元素名称是BBB,在BBB上面有id属性，并且id的属性值是b1**

**13、使用dom4j支持xpath具体操作**

**\*\* 默认的情况下，dom4j不支持xpath**

**\*\* 如果想要在dom4j里面是有xpath**

**\* 第一步需要，引入支持xpath的jar包，使用 jaxen-1.1-beta-6.jar**

**\*\* 需要把jar包导入到项目中**

**\*\* 在dom4j里面提供了两个方法，用来支持xpath**

**\*\*\* selectNodes("xpath表达式")**

**- 获取多个节点**

**\*\*\* selectSingleNode("xpath表达式")**

**- 获取一个节点**

**\*\* 使用xpath实现：查询xml中所有name元素的值**

**\*\* 所有name元素的xpath表示： //name**

**\*\* 使用selectNodes("//name");**

**\*\* 代码和步骤**

**/\***

**\* 1、得到document**

**\* 2、直接使用selectNodes("//name")方法得到所有的name元素**

**\***

**\* \*/**

**//得到document**

**Document document = Dom4jUtils.getDocument(Dom4jUtils.PATH);**

**//使用selectNodes("//name")方法得到所有的name元素**

**List<Node> list = document.selectNodes("//name");**

**//遍历list集合**

**for (Node node : list) {**

**//node是每一个name元素**

**//得到name元素里面的值**

**String s = node.getText();**

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

**}**

**\*\* 使用xpath实现：获取第一个p1下面的name的值**

**\* //p1[@id1='aaaa']/name**

**\* 使用到 selectSingleNode("//p1[@id1='aaaa']/name")**

**\* 步骤和代码**

**/\***

**\* 1、得到document**

**\* 2、直接使用selectSingleNode方法实现**

**\* - xpath ： //p1[@id1='aaaa']/name**

**\* \*/**

**//得到document**

**Document document = Dom4jUtils.getDocument(Dom4jUtils.PATH);**

**//直接使用selectSingleNode方法实现**

**Node name1 = document.selectSingleNode("//p1[@id1='aaaa']/name"); //name的元素**

**//得到name里面的值**

**String s1 = name1.getText();**

**System.out.println(s1);**

**14、实现简单的学生管理系统**

**\*\* 使用xml当做数据，存储学生信息**

**\*\* 创建一个xml文件，写一些学生信息**

**\*\* 增加操作**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\* 3、获取到根节点**

**\* 4、在根节点上面创建stu标签**

**\* 5、在stu标签上面依次添加id name age**

**\*\* addElement方法添加**

**\* 6、在id name age上面依次添加值**

**\* \*\* setText方法**

**\* 7、回写xml**

**\* \*/**

**\*\* 删除操作（根据id删除）**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\***

**\* 3、获取到所有的id**

**\* 使用xpath //id 返回 list集合**

**\* 4、遍历list集合**

**\* 5、判断集合里面的id和传递的id是否相同**

**\* 6、如果相同，把id所在的stu删除**

**\***

**\* \*/**

**\*\* 查询操作（根据id查询）**

**/\***

**\* 1、创建解析器**

**\* 2、得到document**

**\***

**\* 3、获取到所有的id**

**\* 4、返回的是list集合，遍历list集合**

**\* 5、得到每一个id的节点**

**\* 6、id节点的值**

**\* 7、判断id的值和传递的id值是否相同**

**\* 8、如果相同，先获取到id的父节点stu**

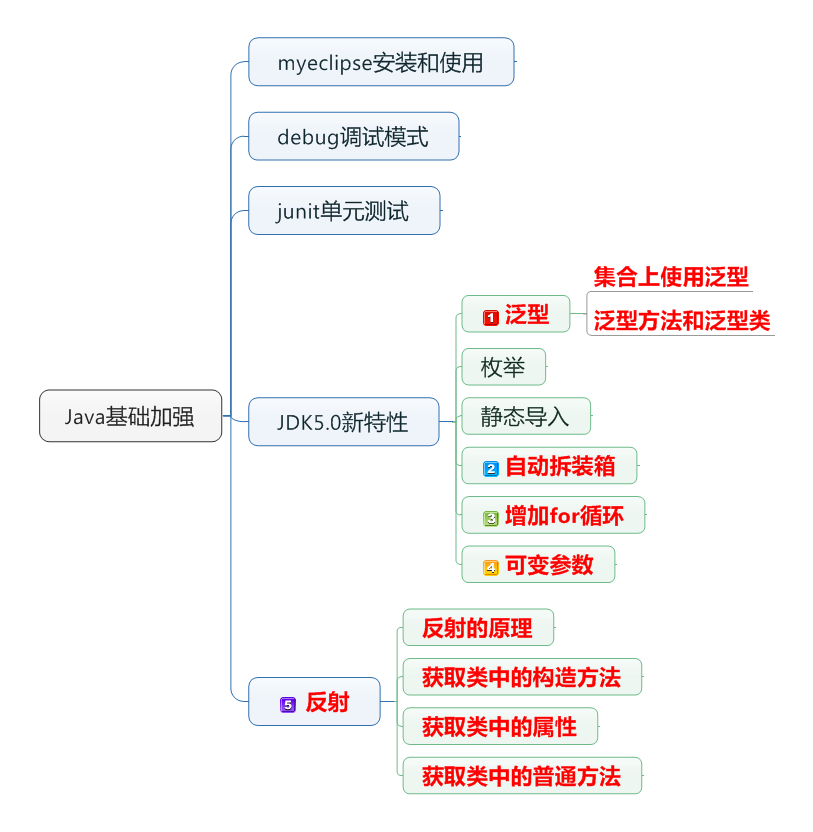
**\* 9、通过stu获取到name age值**

**\*\* 把这些值封装到一个对象里面 返回对象**

**\***

**\* \*/**

**DAY7**

****

**1、myeclipse的安装和使用**

**\* eclipse：是一个免费的开发工具**

**\* myeclipse：是一个收费的插件，破解myeclipse，**

**\*\* 安装目录的要求： 不能有中文和空格**

**\*\* 安装完成之后，选择一个工作空间 ，这个工作空间不能有中文和空格**

**\* 破解myeclipse**

**\*\* 运行run.bat文件，但是运行之前，必须要安装jdk，通过配置环境变量**

**\* myeclipse的使用**

**\* 创建一个工程**

**- 类型 java project web project**

**- 选择依赖的jdk，可以使用myeclipse自带的jdk，或者可以使用安装的jdk**

**\* 创建包 package**

**- cn.itcast.test XX.XX.XX**

**\* 在包里面创建一个类**

**- 类的命名规范：**

**\*\* 首字母要大写**

**比如： TestDemo1 UserManager**

**\* 在类里面创建方法**

**public void test1(参数列表) {**

**方法体或者返回值;**

**}**

**- 方法的命名规范**

**首字母小写 比如：addNum()**

**\* 定义变量**

**- 变量的命名规范**

**\*\* 首字母小写，第二个单词的首字母要大写 ，比如 userName**

**\* 这些命名还有一种方式**

**\*\* 使用汉语拼音命名 yonghuming mima**

**\*\* 不能把汉语拼音和英文字母混合使用**

**userMing**

**\* 命名的最基本的原则：看到名字知道是什么含义**

**\* 代码需要有缩进**

**\* 运行程序 run as java application**

**debug as java application**

**2、debug的调试模式（断点调试模式）**

**\* 使用这种模式，调试程序（看到程序里面数据的变化）**

**\* 使用debug第一步需要设置一个断点（让程序运行停止在这一行）**

**- 显示出来行号**

**- 双击左边，出现一个圆点，表示设置了一个断点**

**\* 使用debug as方式，运行程序**

**- 提示是否进入到调试界面，yes**

**- 在断点那一个，有一个绿色条，表示程序停止在这一行，没有向下运行**

**\* 可以让程序向下执行，**

**- 使用 step over 快捷键是 F6（单步执行）**

**- resume F8：表示调试结束，直接向下运行**

**\*\* 比如当前的断点之后还有断点，跳到下一个断点，**

**\*\*　如果当前断点后面没有断点，程序直接运行结束**

**\* debug另外一个用途**

**\*\* 查看程序的源代码**

**\*\* F5 step into：进入到方法**

**\*\* F7 step return :返回**

**3、myeclipse的快捷键的使用**

**\* 代码提示 alt /**

**\* 快速导包 ctrl shift o**

**\* 单行注释 ctrl /**

**\* 去掉单行注释 ctrl /**

**\* 多行注释 ctrl shift /**

**\* 去掉多行注释 ctrl shift \**

**\* 删除行 ctrl d**

**4、junit的使用**

**\* 单元测试**

**\* 测试对象是 是一个类中的方法**

**\* juint不是javase的一部分，想要使用导入jar包**

**\*\* 但是，在myeclipse中自带了junit的jar包**

**\* 首先junit版本 3.x 4.x**

**\* 单元测试方法时候，方法命名规则 public void 方法名() {}**

**\* 使用注解方式运行测试方法, 在方法的上面**

**\*\* @Test：表示方法进行单元测试**

**--- @Test**

**public void testAdd1() {**

**TestJunit test01 = new TestJunit();**

**test01.testAdd(2, 3);**

**}**

**- 选中方法名称，右键运行 点击run as --- junit test**

**- 当出现绿色条，表示方法测试通过**

**- 当出现了红棕色条，表示方法测试不通过**

**--- 要运行类中的多个测试方法，点击类中的其他位置，run as --- junit test**

**\*\* @Ignore ：表示这个方法不进行单元测试**

**\*\* @Before: 在每个方法执行运行**

**\*\* @After：在每个方法之后运行**

**\*\* 断言（了解）**

**- Assert.assertEquals("测试期望的值", "方法运行的实际的值")**

**jdk5.0新特性**

**jdk 1.1 1.2 1.4 5.0**

**\*\* 泛型、枚举、静态导入、自动拆装箱、增强for、可变参数**

**\*\* 反射**

**5、泛型的简介**

**\* 为什么要使用泛型？**

**- 一般使用在集合上**

**\*\* 比如现在把一个字符串类型的值放入到集合里面，这个时候，这个值放入到集合之后，失去本事的类型，只能是object类型，**

**这个时候，比如想要对这个值进行类型转换，很容易出现类型转换错误，怎么解决这个问题，可以使用泛型来解决**

**\* 在集合上如何使用泛型**

**- 常用集合 list set map**

**- 泛型语法 集合<String> 比如 List<String>**

**\* 在泛型里面写是一个对象，String 不能写基本的数据类型 比如int (\*\*\*\*)**

**\*\* 写基本的数据类型对应包装类**

**byte -- Byte**

**short -- Short**

**int -- Integer**

**long -- Long**

**float -- Float**

**double -- Double**

**char -- Character**

**boolean -- Boolean**

**\* 在list上使用泛型**

**list的三种实现 ArrayList linkedList Vector**

**代码：**

**@Test**

**public void testList() {**

**List<String> list = new ArrayList<String>();**

**list.add("aaa");**

**list.add("bbb");**

**list.add("ccc");**

**//遍历list集合 有几种方式 三种**

**//普通for循环 迭代器 增强for**

**//普通for循环**

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

**String s = list.get(i);**

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

**}**

**System.out.println("=================");**

**//使用增强for**

**for (String s1 : list) {**

**System.out.println(s1);**

**}**

**System.out.println("=================");**

**//使用迭代器遍历**

**Iterator<String> it = list.iterator();**

**while(it.hasNext()) {**

**System.out.println(it.next());**

**}**

**\* 作业1： ArrayList linkedList Vector 这三个区别**

**\* 在set上使用泛型**

**代码：**

**//泛型使用set集合上**

**@Test**

**public void testSet() {**

**Set<String> set = new HashSet<String>();**

**set.add("www");**

**set.add("qqq");**

**set.add("zzz");**

**//set.add("qqq");**

**//遍历set 有几种方式 两种**

**//迭代器 增强for**

**//使用增强for遍历**

**for (String s2 : set) {**

**System.out.println(s2);**

**}**

**System.out.println("=================");**

**//使用迭代器遍历**

**Iterator<String> it1 = set.iterator();**

**while(it1.hasNext()) {**

**System.out.println(it1.next());**

**}**

**}**

**\* 在map上面使用泛型**

**- map结构：key-valu形式**

**代码：**

**//在map上使用泛型**

**@Test**

**public void testMap() {**

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

**map.put("aaa", "111");**

**map.put("bbb", "222");**

**map.put("ccc", "333");**

**//遍历map 有几种遍历方式 两种**

**// 1、获取所有的key，通过key得到value 使用get方法**

**// 2、获取key和value的关系**

**//使用第一种方式遍历**

**//获取所有的key**

**Set<String> sets = map.keySet();**

**//遍历所有key返回的set**

**for (String key : sets) {**

**//通过key得到value**

**String value = map.get(key);**

**System.out.println(key+" : "+value);**

**}**

**System.out.println("==============");**

**//得到key和value的关系**

**Set<Entry<String, String>> sets1 = map.entrySet();**

**//遍历sets1**

**for (Entry<String, String> entry : sets1) {**

**//entry是key和value关系**

**String keyv = entry.getKey();**

**String valuev = entry.getValue();**

**System.out.println(keyv+" : "+valuev);**

**}**

**}**

**6、泛型使用在方法上**

**\* 定义一个数组，实现指定位置上数组元素的交换**

**\* 方法逻辑相同，只是数据类型不同，这个时候使用泛型方法**

**\* /\***

**\* 使用泛型方法 需要定义一个类型 使用大写字母表示 T :这个T表示任意的类型**

**\* 写在返回值之前 void之前 <T>**

**\* =======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;**

**}**

**\*\* 作业2： 实现一个泛型方法，接受任意一个数组，颠倒数组中所有元素**

**7、泛型在类上的使用（了解）**

**\* 在一个类上定义一个类型，这个类型可以在类里面直接使用**

**\* public class TestDemo04<T> {**

**//在类里面可以直接使用T的类型**

**T aa;**

**public void test11(T bb) {}**

**//写一个静态方法 在类上面定义的泛型，不能再静态方法里面使用**

**public static <A> void test12(A cc) {}**

**}**

**8、枚举的简介**

**\* 什么是枚举？**

**\*\* 需要在一定的范围内取值，这个值只能是这个范围内中的任意一个。**

**\*\* 现实场景：交通信号灯，有三种颜色，但是每次只能亮三种颜色里面的任意一个**

**\* 使用一个关键字 enum**

**\*\* enum Color3 {**

**RED,GREEN,YELLOW;**

**}**

**\* 枚举的构造方法也是私有的**

**\* 特殊枚举的操作（了解）**

**\*\* 在枚举类里面有构造方法**

**\*\* 构造方法里面有参数，需要在每个实例上面都写参数**

**\*\* 在枚举类里面有抽象方法**

**\*\* 在枚举的每个实例里面都重写这个抽象方法**

**enum Color11 {**

**RED("red"){**

**@Override**

**public void print1() {**

**System.out.println("red");**

**}**

**},GREEN("green"){**

**@Override**

**public void print1() {**

**System.out.println("green");**

**}**

**},YELLOW("yellow") {**

**@Override**

**public void print1() {**

**System.out.println("yellow");**

**}**

**};**

**private Color11(String name) {}**

**//抽象方法**

**//当在枚举里面写了抽象方法之后，需要在每个实例上面都实现抽象方法**

**public abstract void print1();**

**}**

**9、枚举的api的操作**

**\*\* name() ：返回枚举的名称**

**\*\* ordinal() ：枚举的下标，下标从0开始**

**\*\* valueOf(Class<T> enumType, String name) ：得到枚举的对象**

**\*\* 还有两个方法，都是这两个方法不在api里面，编译的时候生成两个方法**

**\*\*\* valueof(String name) 转换枚举对象**

**\*\*\* values() 获得所有枚举对象数组**

**\* 练习：枚举对象、枚举对象下标、枚举对象名称表示之间的转换**

**- //知道枚举的对象，得到枚举名称和下标**

**@Test**

**public void test1() {**

**//得到枚举对象**

**Color100 c100 = Color100.RED;**

**//枚举名称**

**String name = c100.name();**

**//枚举的下标**

**int idx = c100.ordinal();**

**System.out.println(name+" "+idx);**

**}**

**- //知道枚举的名称，得到枚举的对象和下标**

**@Test**

**public void test2() {**

**String name1 = "GREEN";**

**//得到对象**

**Color100 c1 = Color100.valueOf(name1);**

**//枚举下标**

**int idx1 = c1.ordinal();**

**System.out.println(idx1);**

**}**

**- //知道枚举的下标，得到枚举的对象和名称**

**@Test**

**public void test3() {**

**int idx2 = 2;**

**//得到枚举的对象**

**Color100[] cs = Color100.values();**

**//根据下标得到对象**

**Color100 c12 = cs[idx2];**

**//得到枚举的名称**

**String name = c12.name();**

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

**}**

**10、静态导入(了解)**

**\* 可以在代码里面，直接使用静态导入方式，导入静态方法或者常量**

**\* import static XX.XX.xxx**

**\* import static java.lang.System.out;**

**import static java.util.Arrays.sort;**

**\*\* 比如现在实现一个计算器 在Math类里面**

**11、自动拆装箱**

**\* 装箱**

**\*\* 把基本的数据类型转换成包装类**

**\* 拆箱**

**\*\* 把包装类转换成基本的数据类型**

**\*\* //自动装箱**

**Integer i = 10;**

**//自动拆箱**

**int m = i;**

**\*\* 在jdk1.4里面如何实现装箱和拆箱**

**- //在jdk1.4里面实现拆装箱**

**public void test1() {**

**//装箱**

**Integer m = new Integer(10);**

**//拆箱**

**int a = m.intValue();**

**}**

**\*\* jdk是会向下兼容**

**- 比如 jdk1.4里面写的代码，这个时候到5.0里面也可以运行**

**\*\* 练习：向下兼容**

**== 执行的结果是会调用 doSomething(double m)**

**== 首先在jdk1.4里面肯定调用这个方法，如果调用下面的方法，需要类型转换，但是jdk1.4不能实现自动拆装箱**

**== 由于jdk是向下兼容，所以，在jdk1.4调用这个方法，在jdk5.0里面还是会调用这个方法**

**public static void main(String[] args) {**

**doSomething(10);**

**}**

**public static void doSomething(double m) {**

**System.out.println("double......");**

**}**

**public static void doSomething(Integer a){**

**System.out.println("integer.....");**

**}**

**\*\* 记住：八种基本的数据类型对应的包装类**

**\* int --- Integer**

**\* char--- Character**

**12、增强for循环（\*\*\*\*\*）**

**\* 语法 for(遍历出来的值 : 要遍历的集合) {}**

**- for(String s : list) {**

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

**}**

**\* 使用场景： 数组；实现Iterable接口的集合 可以使用增强for循环**

**\* 在集合上使用增强for循环遍历**

**list set 实现了Iterator接口，所以可以使用增强for循环**

**map不能使用增强for循环，没有实现Iterator接口，所以不能使用增强for循环**

**\* 增强for循环出现目的：为了替代迭代器**

**\*\* 增强for底层就是迭代器实现的**

**13、内容补充**

**（1）泛型擦除**

**\* 首先泛型只是出现在源代码阶段，当编译之后泛型不存在了**

**（2）练习：实现一个泛型方法，接受任意类型的数组，颠倒数组中所有元素**

**代码**

**public static <T> void reverses(T[] arr1) {**

**/\***

**\* 基本思想：把第一个元素和最后一个元素交换位置，把第二个元素和倒数第二个元素交换位置。。。。**

**\* 交换 长度/2**

**\* \*/**

**//遍历数组**

**for(int i=0;i<arr1.length/2;i++) {**

**/\*int temp = arr1[0];**

**arr1[0] = arr1[arr1.length-1];\*/**

**T temp = arr1[i];**

**arr1[i] = arr1[arr1.length-i-1];**

**arr1[arr1.length-i-1] = temp;**

**}**

**}**

**14、可变参数**

**\* 可变参数可以应用在什么场景：**

**\*\* 实现两个数的相加，实现三个数的相加 四个数的相加**

**-- 如果实现的多个方法，这些方法里面逻辑基本相同，唯一不同的是传递的参数的个数，可以使用可变参数**

**\* 可变参数的定义方法 数据类型...数组的名称**

**\* 理解为一个数组，这个数组存储传递过来的参数**

**- 代码**

**public static void add1(int...nums) {**

**//nums理解为一个数组，这个数组存储传递过来的参数**

**//System.out.println(nums.length);**

**int sum = 0;**

**//遍历数组**

**for(int i=0;i<nums.length;i++) {**

**sum += nums[i];**

**}**

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

**}**

**\* 注意的地方**

**（1）可变参数需要写在方法的参数列表中，不能单独定义**

**（2）在方法的参数列表中只能有一个可变参数**

**（3）方法的参数列表中的可变参数，必须放在参数最后**

**- add1(int a,int...nums)**

**15、反射的原理（\*\*\*\*\*\*\*\*理解\*\*\*\*\*\*\*\*）**

**\* 应用在一些通用性比较高的代码 中**

**\* 后面学到的框架，大多数都是使用反射来实现的**

**\* 在框架开发中，都是基于配置文件开发**

**\*\* 在配置文件中配置了类，可以通过反射得到类中的 所有内容，可以让类中的某个方法来执行**

**\* 类中的所有内容：属性、没有参数的构造方法、有参数的构造方法、普通方法**

**\* 画图分析反射的原理**

**\* 首先需要把java文件保存到本地硬盘 .java**

**\* 编译java文件，成.class文件**

**\* 使用jvm，把class文件通过类加载加载到内存中**

**\* 万事万物都是对象，class文件在内存中使用Class类表示**

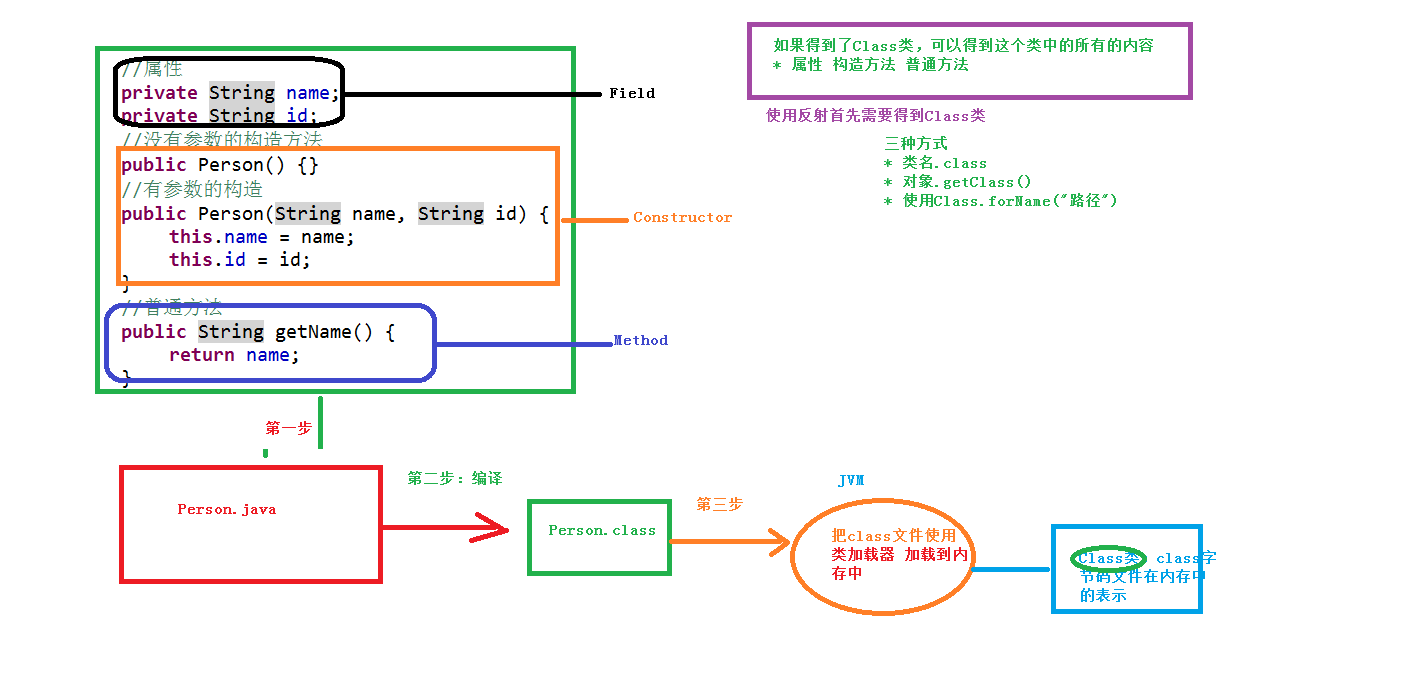
**\* 当使用反射时候，首先需要获取到Class类，得到了这个类之后，就可以得到class文件里面的所有内容**

**- 包含属性 构造方法 普通方法**

**\* 属性通过一个类 Filed**

**\* 构造方法通过一个类 Constructor**

**\* 普通方法通过一个类 Method**

****

**16、使用反射操作类里面的无参数的构造方法（\*\*会写\*\*）**

**\* 首先获取到Class类**

**- // 获取Class类**

**Class clazz1 = Person.class;**

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

**Class clazz3 = Class.forName("cn.itcast.test09.Person");**

**\* 比如： 要对一个类进行实例化，可以new，不使用new，怎么获取？**

**- //得到Class**

**Class c3 = Class.forName("cn.itcast.test09.Person");**

**//得到Person类的实例**

**Person p = (Person) c3.newInstance();**

**\* 代码**

**//操作无参数的构造方法**

**@Test**

**public void test1() throws Exception {**

**//得到Class**

**Class c3 = Class.forName("cn.itcast.test09.Person");**

**//得到Person类的实例**

**Person p = (Person) c3.newInstance();**

**//设置值**

**p.setName("zhangsan");**

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

**}**

**17、使用反射操作有参数的构造方法（\*\*会写\*\*）**

**//操作有参数的构造方法**

**@Test**

**public void test2() throws Exception {**

**//得到Class**

**Class c1 = Class.forName("cn.itcast.test09.Person");**

**//使用有参数的构造方法**

**//c1.getConstructors();//获取所有的构造方法**

**//传递是有参数的构造方法里面参数类型，类型使用class形式传递**

**Constructor cs = c1.getConstructor(String.class,String.class);**

**//通过有参数的构造方法设置值**

**//通过有参数的构造方法创建Person实例**

**Person p1 = (Person) cs.newInstance("lisi","100");**

**System.out.println(p1.getId()+" "+p1.getName());**

**}**

**18、使用反射操作属性（\*\*会写\*\*）**

**\* //操作name属性**

**@Test**

**public void test3() {**

**try {**

**//得到Class类**

**Class c2 = Class.forName("cn.itcast.test09.Person");**

**//得到name属性**

**//c2.getDeclaredFields();//表示得到所有的属性**

**//得到Person类的实例**

**Person p11 = (Person) c2.newInstance();**

**//通过这个方法得到属性，参数是属性的名称**

**Field f1 = c2.getDeclaredField("name");**

**//操作的是私有的属性，不让操作，需要设置可以操作私有属性setAccessible(true),可以操作私有属性**

**f1.setAccessible(true);**

**//设置name值 set方法，两个参数：第一个参数类的实例，第二个参数是设置的值**

**f1.set(p11, "wangwu"); //相当于 在 p.name = "wangwu";**

**System.out.println(f1.get(p11)); //相当于 p.name**

**}catch(Exception e) {**

**e.printStackTrace();**

**}**

**}**

**19、使用泛型操作普通方法（\*\*会写\*\*）**

**\* 使用Method类表示普通方法**

**\* 代码**

**//操作普通方法 ，比如操作 setName**

**@Test**

**public void test4() throws Exception {**

**//得到Class类**

**Class c4 = Class.forName("cn.itcast.test09.Person");**

**//得到Person实例**

**Person p4 = (Person) c4.newInstance();**

**//得到普通方法**

**//c4.getDeclaredMethods();//得到所有的普通方法**

**//传递两个参数：第一个参数，方法名称；第二个参数，方法里面参数的类型**

**Method m1 = c4.getDeclaredMethod("setName", String.class);**

**//让setName方法执行 ，执行设置值**

**//使用invoke(p4, "niuqi");传递两个参数：第一个参数，person实例；第二个参数，设置的值**

**//执行了invoke方法之后，相当于，执行了setName方法，同时通过这个方法设置了一个值是niuqi**

**m1.invoke(p4, "niuqi");**

**System.out.println(p4.getName());**

**}**

**\* //操作的私有的方法 ，需要设置值是true**

**\* //m1.setAccessible(true);**

**\* 当操作的方法是静态的方法时候，因为静态方法调用方式是 类名.方法名，不需要类的实例**

**\* 使用反射操作静态方式时候，也是不需要实例**

**\* 在invokie方法的第一个参数里面，写一个 null**

**- m1.invoke(null, "niuqi");**

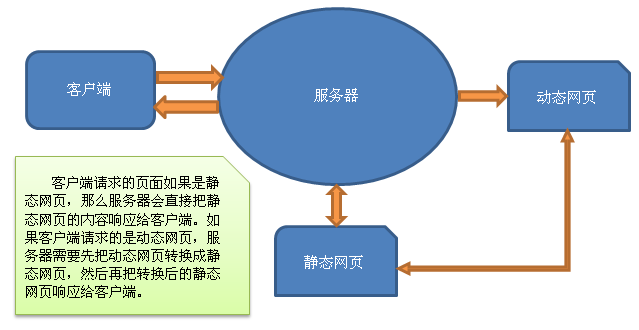
**Day 8**

Web资源介绍

* html：静态资源；
* JSP/Servlet：动态资源。

当然，除了JavaWeb程序，还有其他Web程序，例如：ASP、PHP等。

**动态静态资源区别：**

**‘****’**

访问Web资源

打开浏览器，输入URL：

* 协议名://域名:端口/路径，例如：http://www.itcast.cn:80/index.html

Web服务器的作用是接收客户端的请求，给客户端作出响应。

对于JavaWeb程序而已，还需要有JSP/Servlet容器，JSP/Servlet容器的基本功能是把动态资源转换成静态资源，当然JSP/Servlet容器不只这些功能，我们会在后面一点一点学习。

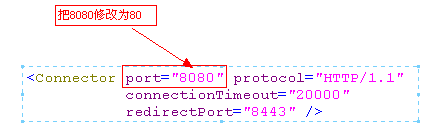
我们需要使用的是Web服务器和JSP/Servlet容器，通常这两者会集于一身。下面是对JavaWeb服务器：

* Tomcat（Apache）：当前应用最广的JavaWeb服务器；
* JBoss（Redhat红帽）：支持JavaEE，应用比较广；EJB容器
* GlassFish（Orcale）：Oracle开发JavaWeb服务器，应用不是很广；
* Resin（Caucho）：支持JavaEE，应用越来越广；
* Weblogic（Orcale）：要钱的！支持JavaEE，适合大型项目；
* Websphere（IBM）：要钱的！支持JavaEE，适合大型项目；

**Tomcat:**

配置端口号

打开%CATALANA\_HOME%\conf\server.xml文件：



http默认端口号为80，也就是说在URL中不给出端口号时就表示使用80端口。当然你也可以修改为其它端口号。

当把端口号修改为80后，在浏览器中只需要输入：<http://localhost>就可以访问Tomcat主页了。

## Web应用（重点）

静态网站：

* 在webapps目录下创建一个目录（命名必须不包含中文和空格），这个目录称之为项目目录；
* 在项目目录下创建一个html文件；

动态网站：

* 在webpass目录下创建一个项目目录；
* 在项目目录下创建如下内容：
* WEB-INF目录；
* 在WEB-INF目录下创建web.xml文件
* 创建静态或动态页面

###### 配置外部应用（了解）

　　原来我们的项目放到webapps下，现在我放到外面，也希望tomcat可以找到它！

也可以把应用放到Tomcat之外，这就是外部应用了。例如我们把上面写的hello应用从webapps目录中剪切到C盘下，即C:/hello。现在hello这个Web应用已经不在Tomcat中了，这时我们需要在tomcat中配置外部应用的位置，配置的方式一共有两种：

* conf/server.xml：打开server.xml文件，找到<Host>元素，在其中添加<Context>元素，代码如下：

server.xml

|  |
| --- |
|  |

1. path：指定当前应用的名称；
2. docBase：指定应用的物理位置；
3. 浏览器访问路径：<http://localhost:8080/itcast_hello/index.html>。

* conf/catalana/localhost：在该目录下创建itcast\_hello.xml文件，在该文件中编写<Context>元素，代码如下：

|  |
| --- |
|  |

1. 文件名：指定当前应用的名称；
2. docBase：指定应用的物理位置；
3. 浏览器访问路径：<http://localhost:8080/itcast_hello/index.html>。

###### 理解server.xml（了解）

<Server>

<Service>

<Connector>

<Engine>

<Host>

<Context>

* <Server>：根元素，表示整个服务器的配置信息；
* <Service>：<Server>的子元素，在<Server>中只能有一个<Service>元素，它表示服务；
* <Connector>：<Service>的子元素，在<Service>中可以有N个<Connector>元素，它表示连接。
* <Engine>：<Service>的子元素，在<Service>中只能有一<Engine>元素，该元素表示引擎，它是<Service>组件的核心。
* <Host>：<Engine>的子元素，在<Engine>中可以有N个<Host>元素，每个<Host>元素表示一个虚拟主机。所谓虚拟主机就像是真的主机一样，每个主机都有自己的主机名和项目目录。例如<Host name=”localhost” appBase=”webapps”>表示主机名为localhost，这个主机的项目存放在webapps目录中。访问这个项目下的主机时，需要使用localhost主机名，项目都存放在webapps目录下。
* <Context>：<Host>元素的子元素，在<Host>中可以有N个<Context>元素，每个<Context>元素表示一个应用。如果应用在<Host>的appBase指定的目录下，那么可以不配置<Context>元素，如果是外部应用，那么就必须配置<Context>。如果要为应用指定资源，也需要配置<Context>元素。

我们可以把<Server>看作是一个大酒店：

* <Service>：酒店的服务部门；
* <Connector>：服务员；
* <Engine>：后厨；
* <Host>：后厨中的一个区，例如川菜区是一个<Host>、粤菜区是一个<Host>；
* <Context>：后厨的一个厨师。

用户发出一个请求：http://localhost:8080/hello/index.jsp。发现是http/1.1协议，而且还是8080端口，所以就交给了处理这一请求的“服务员（处理HTTP请求的<Connector>）”，“服务员”再把请求交给了“后厨（<Engine>）”，因为请求是要一盘水煮鱼，所以由“川菜区（<Host>）”负责，因为“大老王师傅<Context>”做水煮鱼最地道，所以由它完成。

* <Connector>：关心请求中的http、和8080；
* < Host>：关心localhost；
* <Context>：关心hello

**HTTP协议：**

###### 请求协议

请求协议的格式如下：

|  |
| --- |
| 请求首行；  请求头信息；--》头名称：头值  空行；  请求体。 |

**GET请求没有请求体。**

**响应：**

**响应行(协议/版本 状态码 状态码的解析)**

**响应头（key/value格式）**

**空行**

**响应正文**

**编码原理：**

**username=%E5%BC%A0%E4%B8%89&password=123**

**-128~127、-128~127、-128~127、-128~127、-128~127、-128~127**

**字节+128 --> 转换成16进制 --> 添加%这个前缀**

**36 --> %A4**

POST请求是可以有体的，而GET请求不能有请求体。

* Referer: http://localhost:8080/hello/index.jsp：请求来自哪个页面，例如你在百度上点击链接到了这里，那么Referer:http://www.baidu.com；如果你是在浏览器的地址栏中直接输入的地址，那么就没有Referer这个请求头了；
* Content-Type: application/x-www-form-urlencoded：表单的数据类型，说明会使用url格式编码数据；url编码的数据都是以“%”为前缀，后面跟随两位的16进制，例如“传智”这两个字使用UTF-8的url编码用为“%E4%BC%A0%E6%99%BA”；
* Content-Length:13：请求体的长度，这里表示13个字节。
* keyword=hello：请求体内容！hello是在表单中输入的数据，keyword是表单字段的名字。

Referer请求头是比较有用的一个请求头，它可以用来做统计工作，也可以用来做防盗链。

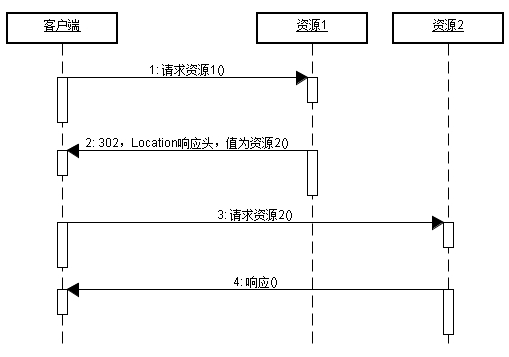
**统计工作**：我公司网站在百度上做了广告，但不知道在百度上做广告对我们网站的访问量是否有影响，那么可以对每个请求中的Referer进行分析，如果Referer为百度的很多，那么说明用户都是通过百度找到我们公司网站的。

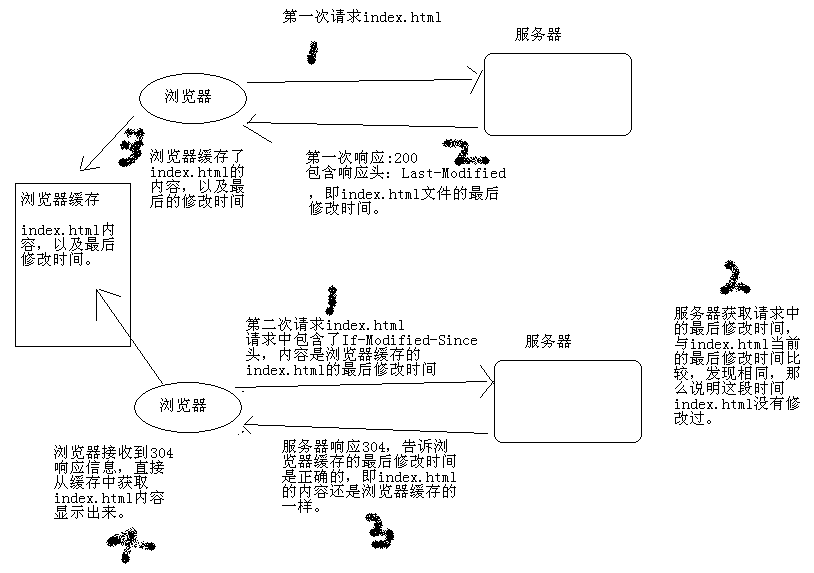
**防盗链**：我公司网站上有一个下载链接，而其他网站盗链了这个地址，例如在我网站上的index.html页面中有一个链接，点击即可下载JDK7.0，但有某个人的微博中盗链了这个资源，它也有一个链接指向我们网站的JDK7.0，也就是说登录它的微博，点击链接就可以从我网站上下载JDK7.0，这导致我们网站的广告没有看，但下载的却是我网站的资源。这时可以使用Referer进行防盗链，在资源被下载之前，我们对Referer进行判断，如果请求来自本网站，那么允许下载，如果非本网站，先跳转到本网站看广告，然后再允许下载。

响应码

响应头对浏览器来说很重要，它说明了响应的真正含义。例如200表示响应成功了，302表示重定向，这说明浏览器需要再发一个新的请求。

* 200：请求成功，浏览器会把响应体内容（通常是html）显示在浏览器中；
* 404：请求的资源没有找到，说明客户端错误的请求了不存在的资源；
* 500：请求资源找到了，但服务器内部出现了错误；
* 302：重定向，当响应码为302时，表示服务器要求浏览器重新再发一个请求，服务器会发送一个响应头Location，它指定了新请求的URL地址；
* 304：当用户第一次请求index.html时，服务器会添加一个名为Last-Modified响应头，这个头说明了index.html的最后修改时间，浏览器会把index.html内容，以及最后响应时间缓存下来。当用户第二次请求index.html时，在请求中包含一个名为If-Modified-Since请求头，它的值就是第一次请求时服务器通过Last-Modified响应头发送给浏览器的值，即index.html最后的修改时间，If-Modified-Since请求头就是在告诉服务器，我这里浏览器缓存的index.html最后修改时间是这个，您看看现在的index.html最后修改时间是不是这个，如果还是，那么您就不用再响应这个index.html内容了，我会把缓存的内容直接显示出来。而服务器端会获取If-Modified-Since值，与index.html的当前最后修改时间比对，如果相同，服务器会发响应码304，表示index.html与浏览器上次缓存的相同，无需再次发送，浏览器可以显示自己的缓存页面，如果比对不同，那么说明index.html已经做了修改，服务器会响应200。
* 重定向原理：

****



HTML中指定响应头

在HTMl页面中可以使用<meta http-equiv="" content="">来指定响应头，例如在index.html页面中给出<meta http-equiv="Refresh" content="3;url=http://www.itcast.cn">，表示浏览器只会显示index.html页面3秒，然后自动跳转到<http://www.itcast.cn>。

操作图片：

*// 得到图片缓冲区*BufferedImage bi = **new** BufferedImage(150, 70, BufferedImage.***TYPE\_INT\_RGB***);  
  
*// 得到它的绘制环境（这张图片的笔）*Graphics2D g2 = (Graphics2D) bi.getGraphics();  
g2.setColor(Color.***WHITE***);*//设置颜色*g2.fillRect(0, 0, 150, 70);*//填充整张图片(其实就是设置背景色)*g2.setColor(Color.***RED***);  
g2.drawRect(0, 0, 150-1, 70-1);  
  
g2.setFont(**new** Font(**"宋体"**, Font.***BOLD***, 18));*//设置字体*g2.setColor(Color.***BLACK***);*//设置颜色*g2.drawString(**"HelloWorld"**, 3, 50);*//向图片上写字符串*ImageIO.*write*(bi, **"JPEG"**, **new** FileOutputStream(**"F:/a.jpg"**));*//保存图片*

**GET请求：**

**1. 请求数据长度不能超过1K**

**2. 请求参数会在浏览器地址栏中显示出来**

**3. GET请求没有请求体（请求正文），参数是追加在url后面**

**POST请求：**

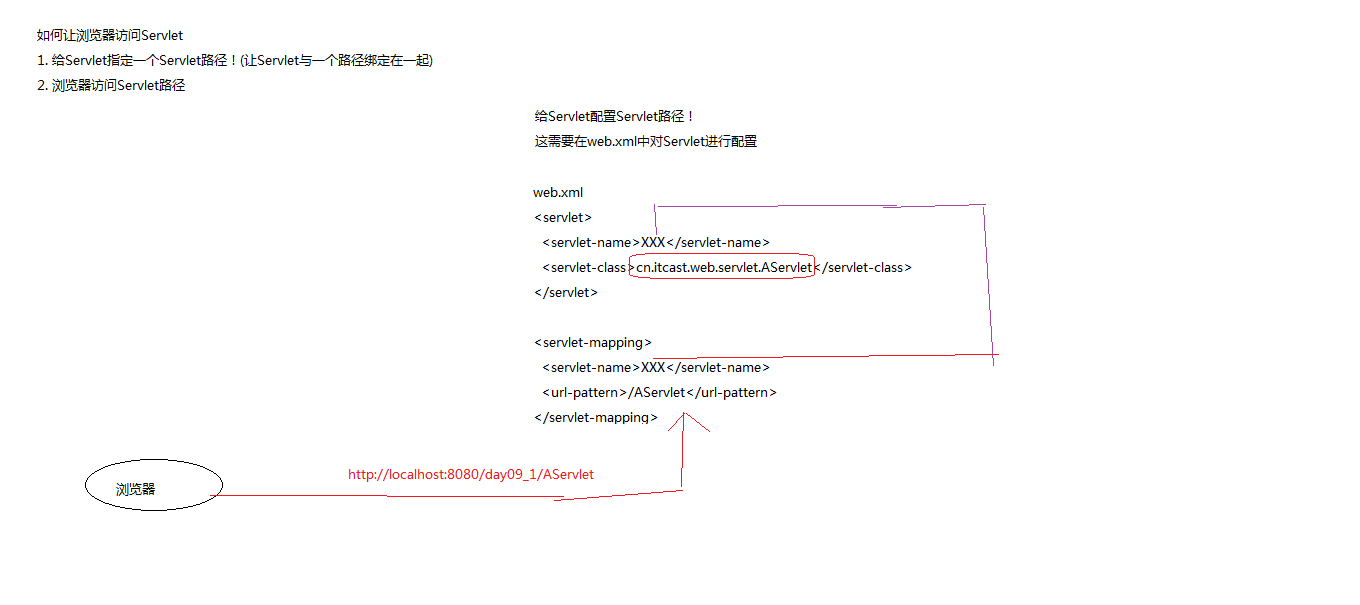
**1. 请求数据没有上限**

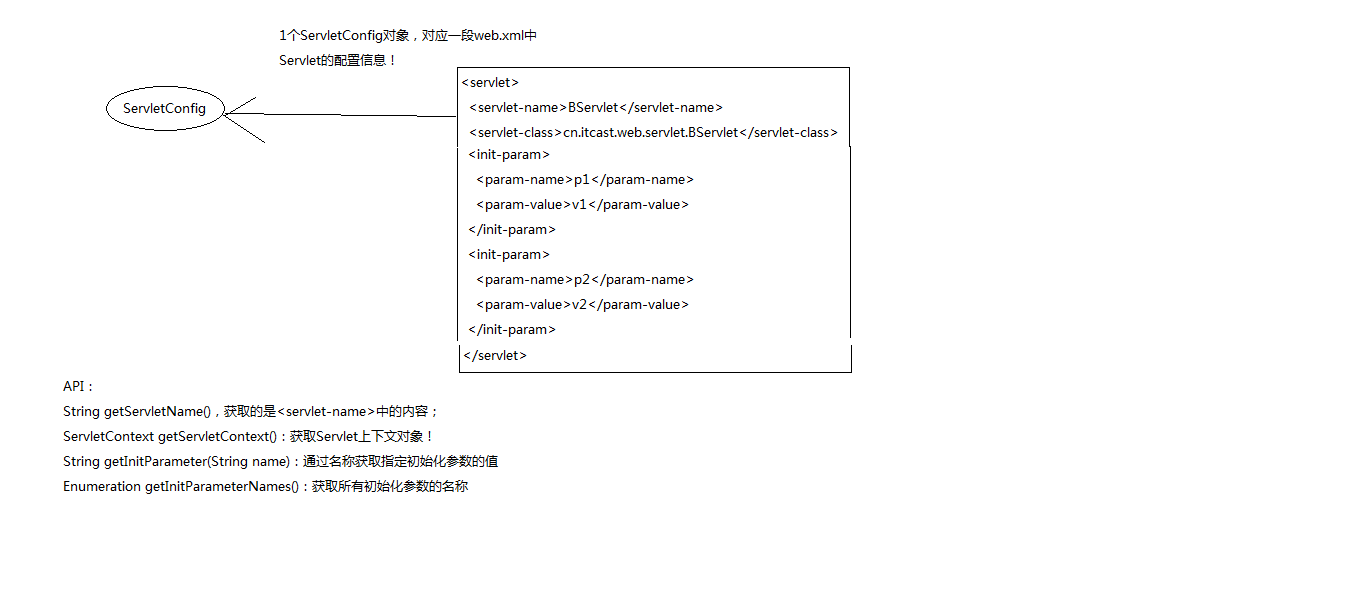
**2. 请求参数不会在浏览器地址栏中显示**

**3. 有请求体（请求正文）**

**4. 请求参数默认使用URL编码**

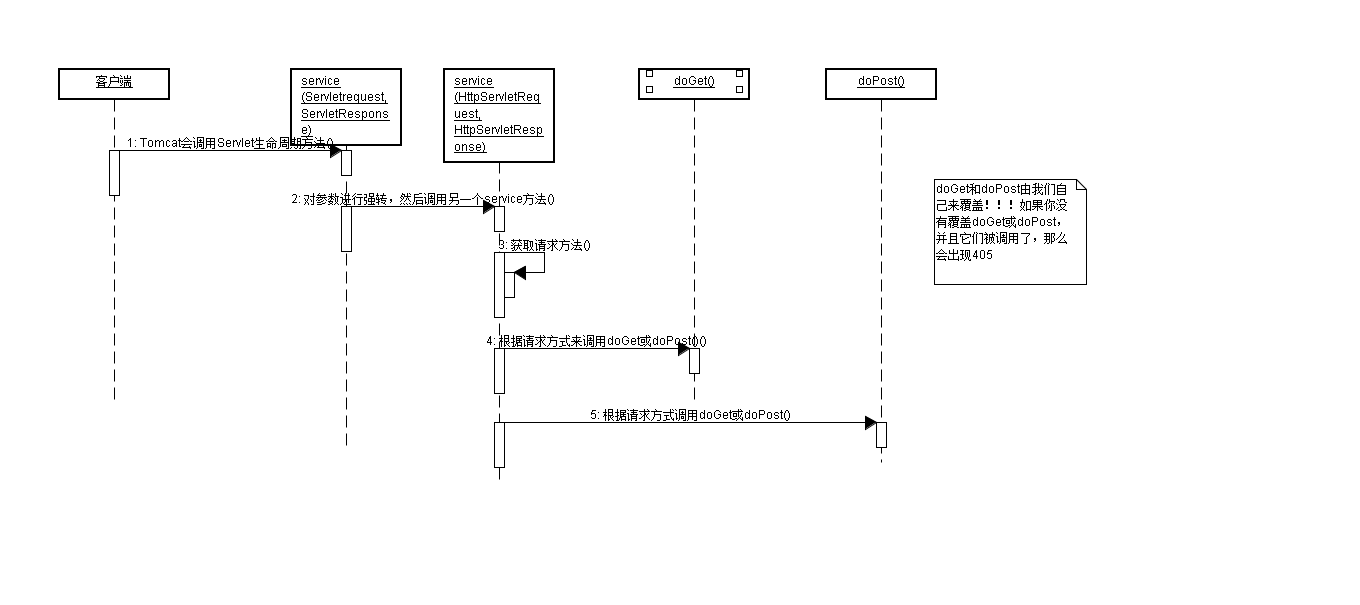
**Day9-Servlet**

****

****

****

**以后我们的类就是继承HttpServlet类，重写doGet（）和doPost，然后Tomcat会调用service将参数强制改成HttpServletRequest等，然后通过request得到请求的方式GET和POST,然后根据请求方式调用doGet（）或者doPost方法。**

****