# HTML

超文本标记语言（Hyper Text Markup Language）简写：HTML。

Html通过标签来标记要显示的网页中的各个部分。网页文件本身是一种文本文件，

通过在文本文件中添加标记符，可以告诉浏览器如何显示其中的内容（如：文字如何处理，画面如何安排，图片如何显示等）。

页面的组成分为：**内容（结构）、表现和行为**。

内容：是我们在页面中可以看到的数据。我们称之为内容。一般内容 我们使用html技术来展示。

表现：指的是这些内容在页面上的展示形式。比如说。布局，颜色，大小等等。一般使用CSS技术实现

行为：指的是页面中元素与输入设备交互的响应。一般使用javascript技术实现。

**简单HTML语言的结构**

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> <html><!-- html标签表示html页面内容的开始 --> <head><!-- head标签表示头信息。head标签中主要包含三部分内容：title标题，css样式，js内容 -->  <!-- meta 是元信息，  Content-Type 表示告诉浏览器当前文件内容是什么类型  text/html; 表示我是text文本，具体是html标签  charset=UTF-8 告诉浏览器使用UTF-8编码来查看 -->  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>学习页面</title><!-- title标题 --> </head> <body><!-- body是整个页面空白区域，页面要显示的主体内容 --> hello world! </body> </html><!-- html标签表示html页面内容的结束 --> |

**HTML注释**

HTML语言的注释格式为：<!—注释内容 -->，HTML语言的注释可以在页面中右键查看源代码时看到。

**HTML路径**

在JavaSE中，相对路径是从工程根目录开始算起，绝对路径则是从盘符开始算起。而在HTML语言中，相对目录中：

**相对路径**

./ 表示当前文件所在的目录

../ 表示上一级目录

../../ 表示上上一级目录

资源名 表示当前目录下的资源（./资源名这种情况下./可以省略）

**绝对路径**

<http://ip:port/工程名/>资源名

在HTML语言中的绝对路径和Java中的绝对路径表示方式不相同，实际开发中，只能使用绝对路径或base标签+相对路径。

**base标签**

base标签可以设置页面中所有相对路径跳转时，需要参照的路径，只要设置了base标签，相对路径就会忽略掉浏览器地址栏中的地址。

|  |
| --- |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <!-- base设置相对路径的参照路径，href属性设置参照的路径 -->  <base href="http://localhost:8080/day07/a/b/"/> </head> <body>  这是a下的b下的c.html页面<br/>  <a href="../../index.html">跳到index.html</a> </body> |

直接通过a标签跳转时浏览器地址栏中的地址是http://localhost:8080/day07/a/b/c.html

跳转回去的相对路径是../../index.html

所有相对路径在跳转时都需要参照本地浏览器地址栏中的地址来进行跳转，其结果为

http://localhost:8080/day07/index.html

因此可以跳转回去。

**特殊字符**

HTML中某些字符是预留的，浏览器会误认为它们是标签，因此要想正确的显示这些预留的字符，就要在HTML源码中使用字符实体（**character entities**）。字符实体类似于：&entity\_name或&#entity\_number，使用字符实体名的好处是易于记忆，但是坏处在于不是所有的浏览器都支持该字符实体。

HTML会截短页面中的空格，如果文本中有10个连续的空格，那么在显示之前，浏览器就会删除其中的9个空格，只会保留一个。此时如果想要增加空格的数量就要使用表示空格的字符实体&nbsp。

**字符实体名称对大小写敏感！！！**

**常见字符实体**

|  |  |  |  |
| --- | --- | --- | --- |
| 显示结果 | 描述 | 实体名称 | 实体编号 |
|  | 空格 | &nbsp; | &#160; |
| < | 小于号 | &lt; | &#60; |
| > | 大于号 | &gt; | &#62; |
| & | 和号 | &amp; | &#38; |
| " | 引号 | &quot; | &#34; |
| ' | 撇号 | &apos; (IE不支持) | &#39; |
| ￠ | 分（cent） | &cent; | &#162; |
| £ | 镑（pound） | &pound; | &#163; |
| ¥ | 元（yen） | &yen; | &#165; |
| € | 欧元（euro） | &euro; | &#8364; |
| § | 小节 | &sect; | &#167; |
| © | 版权（copyright） | &copy; | &#169; |
| ® | 注册商标 | &reg; | &#174; |
| ™ | 商标 | &trade; | &#8482; |
| × | 乘号 | &times; | &#215; |
| ÷ | 除号 | &divide; | &#247; |

**标签**

标签是HTML语言的重要组成部分，可以说HTML语言就是标签语言。HTML中的标签对大小写不敏感，每个标签都拥有自己的属性（基本属性和时间属性），且标签根据其形式可以分为单标签和双标签。

单标签格式：<标签名 /> <br /> 换行、<hr />水平线

双标签格式: <标签名>封装的数据</标签名>

**使用标签时需要注意标签的语法**

1. 标签不能交叉嵌套；
2. 表签必须正确关闭，虽然浏览器会自动修复部分没有关闭的标签，但是存在发生错误的可能；
3. 没有文本内容的标签属于单标签，必须要在标签内有“/”符号来关闭标签；
4. 属性必须有值且属性值必须使用双引号包裹；
5. 注释不能嵌套。

**块元素**

每个块元素通常都会独自占据一行或多行，可以对其设置宽度、高度、对齐等属性，常用于网页布局和网页结构的搭建。常见的块元素有<h1>~<h6>、<p>、<div>、<ul>、<ol>、<li>等，其中<div>标记是最典型的块元素。

# CSS

CSS 是「层叠样式表单」，是用于(增强美化)控制网页样式并允许将样式信息与网页内容分离的一种标记性语言。HTML决定网页的骨架，CSS决定网页的外表。

选择器

浏览器根据“选择器”决定受CSS样式影响的HTML元素（标签）。

属性

要改变的样式名，并且每个属性都有一个值。属性和值被冒号分开，并由花括号包围，这样就组成了一个完整的样式声明（declaration），例如：p {color: blue}

多个声明

如果要定义不止一个声明，则需要用分号将每个声明分开。虽然最后一条声明的最后可以不加分号(但尽量在每条声明的末尾都加上分号)

例如：

p{

color:red;

font-size:30px;

}

在编写CSS代码时，习惯于每行只描述一个属性，便于阅读。

CSS注释：

/\* 注释内容 \*/

和Java中的块注释写法相同

## CSS和HTML的结合

**方式一**

在HTML语言中标签的style属性上设置“key：value value”的形式修改标签样式。这种方式的缺点在于如果页面中的标签、样式过多，则代码量非常庞大；而且这种方式写出来的代码可读性很差；最后还没有代码的复用性可言。

|  |
| --- |
| <div style="border: 1px solid red; width: 200px; height: 200px; background-color: blue;">  div标签1</div> <div style="border: 1px solid red; width: 200px; height: 200px; background-color: green;">  div标签2</div> <span style="border: 1px solid red;">  span标签1</span> <span style="border: 1px solid red;">  span标签2</span> |

**方式二**

在HTML语言中的head标签中，使用style标签来自定义各种自己需要的CSS样式，格式如下：

xxx{

key：value value；

}

这种方式的缺点在于只能在同一个html页面中复用代码，而不能在多个页面中复用CSS代码；而且不方便维护，实际项目中会有成千上万个页面，需要到每个页面中去修改，工作量过大。

|  |
| --- |
| <head>  <!-- style标签专门用来定义css代码 -->  <style type="text/css">  /\* 这是css与html结合使用的第二种方式 ，在style标签的内部全部CSS语言的内容，因此在这里的注释就和HTML注释不同 \*/  div { border: 1px solid red }  span { border: 1px solid red }  </style>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>标题</title> </head> <body> <div>div标签1</div><div>div标签2</div> <span>span标签1</span><span>span标签2</span> </body> |

**方式三**

将CSS样式代码单独编写在一个CSS文件中，然后再使用link标签向HTML文件中引入该CSS文件。示例如下：

|  |
| --- |
| <link rel="stylesheet" type="text/css" href="./styles.css" /> |

CSS文件内容

|  |
| --- |
| /\* 这是css与html结合使用的第三种方式 \*/ div {border: 1px solid red;width:200px;height:200px} span {border: 1px solid green} |

## CSS选择器

CSS的优先级有两种

按照选择器搜索精确度：行内样式 > ID选择器 > 类选择器 > 元素选择器

就近原则：哪个离得近,就选用哪个的样式

### 标签名选择器

标签名{

属性:值;

}

|  |
| --- |
| <head>  <title>CSS选择器</title>  <style type="text/css">  div {border: 1px solid yellow;color: blue;font-size: 30px}  span {color: yellow;font-size: 20px;border: 5px dashed blue}  </style> </head> <body> <div>div标签1</div><div>div标签2</div> <div>div标签1</div><div>div标签2</div> <span>span标签1</span><span>span标签2</span> </body> |

### id选择器

#id属性值{

属性:值;

}

|  |
| --- |
| <head>  <meta charset="UTF-8">  <title>ID选择器</title>  <style type="text/css">  #id001 {color: blue;font-size: 30px;border: 1px yellow solid}  #id002 {color: red;font-size: 20px;border: 5px blue dotted}  </style> </head> <body> <div id="id001">div标签1</div><div id="id002">div标签2</div> </body> |

### class选择器

.class属性值{

属性:值;

}

|  |
| --- |
| <head>  <meta charset="UTF-8">  <title>class类型选择器</title>  <style type="text/css">  .class01 {color: blue;font-size: 30px;border: 1px solid yellow}  .class02 {color: gray;font-size: 26px;border: 1px solid red}  </style> </head> <body> <div class="class02">div标签class01</div><div>div标签</div> <span class="class02">span标签class01</span><span>span标签2</span> </body> |

### 组合选择器

选择器1,选择器2,选择器n{

属性:值;

}

多个选择器之间如果使用逗号隔开则表示根据其最大交集选择；如果是使用空格隔开则表示根据其最小并集选择。

|  |
| --- |
| <head>  <meta charset="UTF-8">  <title>class类型选择器</title>  <style type="text/css">  .class01,#id01 {color: blue;font-size: 20px;border: 1px yellow solid}  </style> </head> <body> <div class="class01">div标签1</div><div>div标签2</div> <span id="id01">span标签1</span><span>span标签2</span> </body> |

### 通配符选择器

通配符选择器用“\*”号表示，是所有选择器中作用范围最广的，能匹配页面中所有的元素。

|  |
| --- |
| \* {  margin: 10px; /\* 定义外边距\*/  padding: 10px; /\* 定义内边距\*/ } |

### 属性选择器

### 后代选择器

爷爷选择器 孙子选择器 找出所有的后代

### 子元素选择器

父选择器 > 儿子选择器

### 伪类选择器

通常都是用在A标签上

## CSS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，这个区域的专业术语叫盒子。

边框（border）分为上 下 左 右

内边距（padding）分为上 下 左 右

外边距（margin）分为上 下 左 右

float（漂浮）

none :　 默认值。对象不飘浮

left :　 文本流向对象的右边

right :　 文本流向对象的左边

clear（清除）

none :　 默认值。允许两边都可以有浮动对象

left :　 不允许左边有浮动对象

right :　 不允许右边有浮动对象

both :　 不允许有浮动对象

position

static :　 默认值。无特殊定位，对象遵循HTML定位规则

absolute :　 将对象从文档流中拖出，使用 left ， right ， top ， bottom 等属性相对于其最接近的一个最有定位设置的父对象进行绝对定位。如果不存在这样的父对象，则依据 body 对象。而其层叠通过 z-index 属性

relative :　 对象不可层叠，但将依据 left ， right ， top ， bottom 等属性在正常文档流中偏移位置

## CSS常用样式

颜色

color:red;

边框

border:1px solid red;

宽度

width:20px;

高度

height:20px;

背景颜色

background-color:red;

字体样式

font:font-style font-weight font-size\line-height font-family

综合设置字体样式，必须按顺序书写，不需设置的属性可以省略，但是必须保留font-size和font-family属性，否则font属性将不起作用。

font-style:normal\italic\oblique

normal：默认值，浏览器会显示标准的字体样式

italic：浏览器会显示斜体的字体样式

oblique：浏览器会显示倾斜的字体样式

font-weight

字体粗细

font-size:20px;

字体大小

font-family:KaiTi;

可以同时指定多个字体并以逗号隔开，浏览器不支持第一个字体则会尝试下一个。各种字体之间必须使用英文状态下的逗号隔开，中文字体需要加英文状态下的引号，英文字体一般不需要加引号。当需要设置英文字体时，英文字体名必须位于中文字体名之前。如果字体名中包含空格、#、$等符号，则该字体必须加英文状态下的单引号或双引号。尽量使用系统默认字体，保证在任何用户的浏览器中都能正确显示。

在 CSS 中设置字体名称，直接写中文是可以的。但是在文件编码（GB2312、UTF-8 等）不匹配时会产生乱码的错误。

为此，在 CSS 直接使用 Unicode 编码来写字体名称可以避免这些错误。使用 Unicode 写中文字体名称，浏览器是可以正确的解析的。

font-family: "\5FAE\8F6F\96C5\9ED1"，表示设置字体为“微软雅黑”

|  |  |  |
| --- | --- | --- |
| **字体名称** | **英文名称** | **Unicode 编码** |
| 宋体 | SimSun | \5B8B\4F53 |
| 新宋体 | NSimSun | \65B0\5B8B\4F53 |
| 黑体 | SimHei | \9ED1\4F53 |
| 微软雅黑 | Microsoft YaHei | \5FAE\8F6F\96C5\9ED1 |
| 楷体\_GB2312 | KaiTi\_GB2312 | \6977\4F53\_GB2312 |
| 隶书 | LiSu | \96B6\4E66 |
| 幼园 | YouYuan | \5E7C\5706 |
| 华文细黑 | STXihei | \534E\6587\7EC6\9ED1 |
| 细明体 | MingLiU | \7EC6\660E\4F53 |
| 新细明体 | PMingLiU | \65B0\7EC6\660E\4F53 |

字间距

letter-spacing

letter-spacing属性用于定义字间距，所谓字间距就是字符与字符之间的空白。其属性值可为不同单位的数值，允许使用负值，默认为normal。

**单词间距**

word-spacing

word-spacing属性用于定义英文单词之间的间距，对中文字符无效。和letter-spacing一样，其属性值可为不同单位的数值，允许使用负值，默认为normal。

word-spacing和letter-spacing均可对英文进行设置。不同的是letter-spacing定义的为字母之间的间距，而word-spacing定义的为英文单词之间的间距。

**行间距**

line-height

line-height属性用于设置行间距，就是行与行之间的距离，即字符的垂直间距，一般称为行高。line-height常用的属性值单位有三种，分别为像素px，相对值em和百分比%，实际工作中使用最多的是像素px。

DIV居中

margin-left:auto;

margin-right:auto;

文本

text-align:center\left\center;

文本位置

text-decoration

text-decoration属性用于设置文本的下划线，上划线，删除线等装饰效果，其可用属性值如下：

none：没有装饰（正常文本默认值）。

underline：下划线。

overline：上划线。

line-through：删除线。

另外，text-decoration后可以赋多个值，用于给文本添加多种显示效果，例如希望文字同时有下划线和删除线效果，就可以将underline和line-through同时赋给text-decoration。

text-indent

text-indent属性用于设置首行文本的缩进，其属性值可为不同单位的数值、em字符宽度的倍数、或相对于浏览器窗口宽度的百分比%，允许使用负值, 建议使用em作为设置单位。

表格细线

border:1px solid black;/\*设置边框\*/

border-collapse:collapse;/\*边框合并\*/

列表去修饰

list-style:none;

**空白符**

white-space:normal\pre\nowrap

使用HTML制作网页时，不论源代码中有多少空格，在浏览器中只会显示一个字符的空白。在CSS中，使用white-space属性可设置空白符的处理方式，其属性值如下：

normal：常规（默认值），文本中的空格、空行无效，满行（到达区域边界）后自动换行。

pre：预格式化，按文档的书写格式保留空格、空行原样显示。

nowrap：空格空行无效，强制文本不能换行，除非遇到换行标记<br />。内容超出元素的边界也不换行，若超出浏览器页面则会自动增加滚动条。

work-break:normal\break-all\keep-all

自动换行

normal 使用浏览器默认的换行规则。

break-all 允许在单词内换行。

keep-all 只能在半角空格或连字符处换行。

word-wrap:normal\break-word

属性允许长单词或 URL 地址换行到下一行normal

normal 只在允许的断字点换行（浏览器保持默认处理）。

break-word 在长单词或 URL 地址内部进行换行。

# XML

xml指的是扩展标记语言（**EXtensible Markup Language**），类似HTML。其设计宗旨是传输数据，而非显示数据；xml标签没有被预定义，需要自定义标签；被设计为具有自我描述性，且是W3C的推荐标准。

xml的作用在于存储数据；作为项目、模块或框架的配置文件，作为数据的传输格式。

## XML语法

### 文档声明

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

version表示xml的版本；encoding表示编码集（文档声明中还有standalone=“yes/no”用来表示这个xml文件是否是独立的xml文件），<?xml要连在一起，否则会报错。

### XML元素

xml元素指的是从开始标签直到结束标签的部分，元素可以包含其他元素、文本或者两者混合，元素也可以拥有属性。

一个xml文档有且仅有一个根标签，其他标签都是这个跟标签的子标签或孙标签。

一个标签众可以嵌套若干子标签，但是所有的标签必须合理的嵌套，不允许交叉嵌套。

对于XML标签中出现的所有空格和换行，XML解析程序都会当作标签内容进行处理。

#### 命名规则

1. 名称可以有字母、数字以及其他字符
2. 名称不能以数字、标点符号、xml、XML、Xml开始
3. 名称不能包含空格
4. 区分大小写

#### 元素（标签）种类

**单标签**

<标签名 属性=“值” 属性=“值”……/>

**双标签**

<标签名 属性=“值” 属性=“值”……>文本数据或子标签</标签名>

### XML属性

xml的标签属性和html 的标签属性非常类似，**可以提供元素的额外信息。**在标签上书写属性，一个标签上可以书写多个属性，**每个属性的值必须使用 （单/双）引号 引起来**。属性名称的命名规范与元素的命名规范相同；元素中的属性是不允许重复的。

### XML语法规则

1. 所有xml标签都必须要有关闭标签，也就是闭合（不同于HTML，浏览器不会自动修复xml中的错误标签）。
2. xml标签对大小写敏感，因此一对对应的标签必须完全一致。
3. xml标签必须正确的嵌套。
4. xml文档必须有根元素。
5. xml属性值必须加引号
6. xml中特殊字符若果要表示为文本，则需要使用转义方式表示，如“<”写为“&gt；”和“>”写为”&lt；”。

### XML注释

xml的注释和HTML一样：<!—注释 -->，不允许嵌套。

### 文本区域（CDATA区）

CDATA即character data的缩写。

该区域中的任何内容都会被视为文本内容，其格式为<![CDATA[数据内容]]>，其中的数据内容会被当成文本处理。

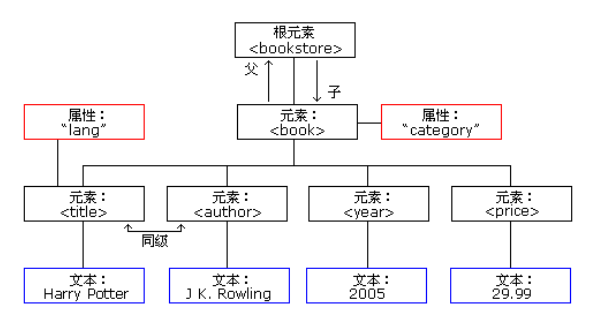
<![CDATA[

<this is normal text>

]]>

## XML解析技术

不管是html文件还是xml文件它们都是标记型文档，都可以使用w3c组织制定的dom技术来解析。xml文档对象模型定义访问和操作xml文档的标准方法，dom将xml作为一个属性结构，而树叶被定义为节点。dom对象表示的是整个文档，可以是html文档，也可以是xml文档。



**早期JDK为我们提供了两种xml解析技术Dom和Sax简介（已经过时）**

DOM解析技术是W3C组织制定的，而所有的编程语言都对这个解析技术使用了自己语言的特点进行实现。

Java对dom技术解析标记也做了实现。DOM技术处理大型文件时其性能下降的非常厉害。这个问题是由DOM的树结构所造成的，这种结构占用的内存较多，而且DOM必须在解析文件之前把整个文档装入内存,适合对XML的随机访问

sun公司在JDK5版本对dom解析技术进行升级：SAX（ Simple API for XML ）。它跟W3C制定的解析不太一样。它是以类似事件机制通过回调告诉用户当前正在解析的内容，一行一行的读取xml文件进行解析，不会创建大量dom对象，所以在解析xml时在内存的使用和性能上都优于Dom解析。不同于DOM,SAX是事件驱动型的XML解析方式。它顺序读取XML文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML文件，适合对XML的顺序访问

**第三方的解析**：

jdom 在dom基础上进行了封装 、**dom4j** （现在主流解析xml文档的技术）又对jdom 进行了封装。pull 主要用在Android 手机开发，是在跟sax非常类似都是事件机制解析xml文件。

## XML约束

在XML技术里，可以编写一个文档来约束一个XML文档的书写规范，这称之为XML约束。

### DTD约束

### Schema约束

#### Schema概述

XML Schema是用一套预先规定的XML元素和属性创建的，这些元素和属性定义了XML文档的结构和内容模式。 XML Schema规定XML文档实例的结构和每个元素/属性的数据类型

Schema相对于DTD的明显好处是，XML Schema文档本身也是XML文档，而不是像DTD一样使用自成一体的语法

XML Schema 文件自身就是一个XML文件，但它的扩展名通常为.xsd，和XML文件一样，一个XML Schema文档也必须有一个根结点，但这个根结点的名称为Schema。

编写了一个XML Schema约束文档后，通常需要把这个文件中声明的元素绑定到一个URI地址上，这个URI地址叫namespace名称空间，以后XML文件就可以通过这个URI（即名称空间）引用绑定指定名称空间的元素。

**XML Schema VS DTD：**

XML Schema符合XML语法结构。

DOM、SAX等XML API很容易解析出XML Schema文档中的内容。

XML Schema对名称空间支持得非常好。

XML Schema比XML DTD支持更多的数据类型，并支持用户自定义新的数据类型。

XML Schema定义约束的能力非常强大，可以对XML实例文档作出细致的语义限制。

XML Schema不能像DTD一样定义实体，比DTD更复杂，但Xml Schema现在已是w3c组织的标准，它正逐步取代DTD。

**Schema和DTD区别**

XML从SGML中继承了DTD，并用它来定义内容的模型，验证和组织元素。同时，它也有很多局限：

DTD不遵守XML语法；

DTD不可扩展；

DTD不支持命名空间的应用；

DTD没有提供强大的数据类型支持，只能表示很简单的数据类型。

Schema完全克服了这些弱点，使得基于Web的应用系统交换XML数据更为容易。下面是它所展现的一些新特性：

Schema完全基于XML语法，不需要再学习特殊的语法；

Schema能用处理XML文档的工具处理，而不需要特殊的工具；

Schema大大扩充了数据类型，支持booleans、numbers、dates and times、URIs、integers、decimal numbers和real numbers等；

Schema支持原型，也就是元素的继承。如：我们定义了一个“联系人”数据类型，然后可以根据它产生“朋友联系人”和“客户联系”两种数据类型；

Schema支持属性组。我们一般声明一些公共属性，然后可以应用于所有的元素，属性组允许把元素、属性关系放于外部定义、组合；

开放性。原来的DTD只能有一个DTD应用于一个XML文档，现在可以有多个Schema运用于一个XML文档。

#### Schema名称空间

在XML Schema中，每个约束模式文档都可以被赋以一个唯一的名称空间，名称空间用一个唯一的URI（Uniform Resource Identifier，统一资源标识符）表示。 在Xml文件中书写标签时，可以通过名称空间声明（xmlns），来声明当前编写的标签来自哪个Schema约束文档。

注意：名称空间的名字语法容易让人混淆，尽管以 http:// 开始，那个 URL 并不指向一个包含模式定义的文件。事实上，这个 URL：http://www.itcast.cn根本没有指向任何文件，只是一个分配的名字。

为了在一个XML文档中声明它所遵循的Schema文件的具体位置，通常需要在Xml文档中的根结点中使用schemaLocation属性来指定。schemaLocation此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置，两者之间用空格分隔。在使用schemaLocation属性时，也需要指定该属性来自哪里。

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www. itcast.cn"

elementFormDefault="qualified"

attributeFormDefault="qualified"

>

<xs:schema>

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

elementFormDefault元素用于指定局部元素是否受到该schema指定targetNamespace所指定的名称空间限定

attributeFormDefault元素用于指定局部属性是否受到该schema指定targetNamespace所指定的名称空间限定。

#### Schema元素

简单类型元素

简单类型元素只能包含字符内容。这些字符可以被约束为特殊的预定义类型或派生类型。例如，可以指定一个简单元素的内容必须是日期、整数、字符串或者仅仅是一个字符或者一系列字符

复杂类型元素

复杂类型元素是包含子元素内容或者属性的元素。

# jQuery

## 概述

jQuery指的是JavaScript和Query，是用于辅助JavaScript开发的js类库，其核心思想是write less,do more(写得更少,做得更多)，所以它实现了很多浏览器的兼容问题。jQuery是免费、开源的，jQuery的语法设计可以使开发更加便捷，例如操作[文档](http://baike.baidu.com/view/55621.htm)对象、选择[DOM](http://baike.baidu.com/view/14806.htm)元素、制作动画效果、事件处理、使用[Ajax](http://baike.baidu.com/view/1641.htm)以及其他功能。

### 使用方式

#### load()

要想使用JQuery ，必须得先添加它的依赖。

1. 导入jquery的支持

<script type="text/javascript" src="js/jquery-1.11.3.min.js"></script>

2. 在早前，我们要使用ajax 去执行请求， 需要写代码很多，请求数据回来后，要赋值显示，也是比较复杂。 但是JQuery 只要简单的一行代码即可。

$(#div01).load("Demo01");

解释起来的话是：

$(#div01) ---> 代表 找到 页面上 id 为 div01 的节点

load("Demo01"); 代表去加载这个路径的值。 当然这里写的是servlet的相对路径地址。

#### get()

字节请求，不带任何参数 ：

$.get("url");

需要带上参数的话与平常的get请求参数传递一样。

$.get("url?name=zhangsan&age=18");

-------------------以下获取数据----------------------------

$.get("url?name=zhangsan&age=18", function(data , status){

alert("data="+data+"--status=="+status);

});

如果想要获取服务器反馈的数据，需要在get里面在加一个参数。 给定一个方法即可。服务器响应后，会执行该方法。

注意，方法里面的参数格式固定， data , status 。 data : 代表服务器响应过来的数据， status 代表这次请求的响应状态码

#### post()

直接，请求不带上任何数据

$.post("Demo01" );

请求，带上数据 。

$.post("Demo01" , {name:"xx" , age:99});

请求，带上数据，并且获取响应回来的数据

$.post("Demo01" , {name:"xx" , age:99} , function(data ,status){

alert("data="+data+"--status="+status);

});

### jQuery对象和dom对象的区别及转换

Dom对象是通过HTML DOM获取到的元素对象；Jquery对象本质就是dom对象的集合；两者各自只能调用自己的方法。

Dom对象转jquery对象：使用核心函数包装dom对象 $(domObject)

Jquery对象转dom对象：通过索引取出指定位置上的元素 $jqueryObj[index]

### 简单案例

#### 仿百度提示

onkeyup="keyup01()"

function keyup01() {

var words = $("#word").val();

alert("输入内容22："+words);

}

上下等同，效果一样。

$(function(){

//最id为 word的元素，进行 键盘弹起事件监听。

$("#word").keyup(

//一旦有键盘弹起了，那么将会执行下面的代码。

function(){

var words = $(this).val();

alert("输入内容："+words);

}

);

});

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

结果代码：

function keyup01() {

var words = $("#word").val();

$.post("SearchServlet" , {word:words} , function(data){

//alert(data);

if(words.length>0){

//显示div

$("#div01").show();

$("#div01").html(data);

}else{

//隐藏div

$("#div01").hide();

}

});

}

#### 省市联动

**XML**

js代码请求。

//当文档准备完毕，就会调用该方法

$(function() {

//执行监听，当这个元素的值发生改变，就会执行内部的方法

$("#province").change(function() {

//那么要做什么呢？ 其实是要把这个元素的值上传到服务器上。

var id = $(this).val();

$.post("CityServlet" , {id:id} ,function(data){

$(data).find("city").each(function() {

var id = $(this).children("id").text();

var cname = $(this).children("cname").text();

$("#city").append("<option value='"+id+"'>"+cname);

});

});

});

});

选则器：

$("#province") : 代表着选则 id为 province 的元素。

遍历：

$(data).find("city") 找出data数据中的所有city ，

$(data).find("city").each(function(){}） . 对每一个city进行遍历，并且执行括号中的方法

$(this).children("id").text(); 遍历得到的每一个city里面，再取它的孩子 id , 然后拿它的值。

取值：

html() ---- 获取或者设置 元素中的 html 内容。 可以获取 p标签中的内容。

如： <p>

我是div.demo中第一个P元素：

<a href="#">我在第一个P里面</a>

</p>

text() ---- 获取元素中间的文字值， 如 ： <a href="">黑马训练营 </a> 得到 黑马训练营

val() --- 获取元素上的 value=""的值 <input name="name" value="zhangsna"> 得到张三

**JSON**

//当文档准备完毕，就会调用该方法

$(function() {

//执行监听，当这个元素的值发生改变，就会执行内部的方法

$("#provice").change(function() {

//那么要做什么呢？ 其实是要把这个元素的值上传到服务器上。

var id = $(this).val();

$("#city").html("<option value=''>-请选择-");

$.post("CityServlet02" , {id:id} ,function(data){

$(data).each(function(i , city) {

$("#city").append("<option value='"+city.id+"'>"+city.cname);

});

},"json");

});

});

写法上与前面的xml 基本一样。 区别就在于，对数据的处理上，有所不同而已。 注意： json的写法，一定要在后面跟上数据格式，否则无法遍历出来。

### 赋值

val("aa")

针对带有value属性的元素进行赋值， 其实就是给该元素的 value赋值

html("aa")

针对某一个元素，可以使用元素左右包括起来一段值的元素赋值 。如：<div> </div>

并且该方法可以写html代码。 如：<font> </font>

text("aa")

该方法针对那些可以在两个标签中写值的赋值工作 。如 ： <div>aaaaaaaaaaaaaa</div>

## jQuery核心函数

$ 是jquery的核心函数，能完成jquery的很多功能，$()就是调用$这个函数。$的本质如下：

ƒ ( selector, context ) {

// The jQuery object is actually just the init constructor 'enhanced'

return new jQuery.fn.init( selector, context, rootjQuery );

}

传入参数为[函数]时：

$(function(){}); 页面加载完成之后

传入参数为[HTML字符串]时：

$("<span>12</span>"); 表示创建了span标签对象。

传入参数为[选择器字符串]时：$(“#id”)

$(“#id属性值”); 表示按id属性值查找标签对象

$(“标签名”); 表示按标签名查找标签对象集合

$(“.class属性值”); 表示按class类型查找标签对象集合。

传入参数为[DOM对象]时：

$(dom对象) 转成为jquery对象

<!--一个script标签只能完成一项功能，因此需要两个或更多script标签-->  
<script type="text/javascript" src="script/jquery-1.7.2.js"></script>  
<script>  
 console.log($);  
 $(function () {//表示页面加载完成后  
 let btnObj = $("#btnId");//查找该id的标签对象  
 btnObj.click(function () {//绑定单击事件  
 console.log("this is jquery");  
 })  
 });  
</script>

## jQuery对象和Dom对象

### 本质

#### jQuery对象

jQuery对象的本质上是一个dom对象数组和一系列jQuery功能函数的组合。其通过alert显示的格式为[object Object]。

通过JQuery提供的API创建的对象，是JQuery对象；

通过JQuery包装的Dom对象，也是JQuery对象；

通过JQuery提供的API查询到的对象，是JQuery对象。

#### Dom对象

dom对象在alert 的时候格式如下[object htmlXxxElement]。

通过getElementById()查询出来的标签对象是Dom对象；

通过getElementsByName()查询出来的标签对象是Dom对象；

通过getElementsByTagName()查询出来的标签对象是Dom对象；

通过createElement() 方法创建的对象，是Dom对象。

### 区别

dom对象的属性和方法不能给jQuery对象使用，jQuery对象的属性和方法也不能给dom对象使用。

### 转换

#### dom转化为jquery

1、先有dom对象

2、$( dom对象 ); var $obj = $(dom对象);

#### jquery转为dom

1、先有jquery对象

2、通过下标: var dom = $obj[下标]；

## jQuery选择器

### 基本选择器

**#ID**  id选择器：根据id查找标签对象

**.class**  类选择器：根据class查找标签对象

**element**  标签选择器：根据标签名查找标签对象

**\***  任意选择器：表示任意的，所有的元素

**selector1，selector2** 组合选择器：合并选择器1，选择器2的结果并返回

**注意**

1. id选择器返回的肯定是单个的对象，其他的选择器返回的是jQuery对象的集合；
2. 返回的是jQuery对象，查询到的是响应的dom对象，但是会被封装为jQuery对象后返回；
3. 组合选择器中的selector可以是任意类型。

### 层级选择器

**ancestor descendant** 后代选择器：在给定的祖先元素下匹配所有的后代元素

**parent > child**  子元素选择器：在给定的父元素下匹配所有的子元素

**prev + next**  相邻元素选择器：匹配所有紧接在 prev 元素后的 next 元素

**prev ~ sibings**  兄弟元素选择器：匹配 prev 元素之后的所有 siblings 元素

### 过滤选择器

#### 基本过滤器

**:first** 获取第一个元素

**:last** 获取最后个元素

**:not(selector)** 去除所有与给定选择器匹配的元素

**:even** 匹配所有索引值为偶数的元素，从 0 开始计数

**:odd** 匹配所有索引值为奇数的元素，从 0 开始计数

**:eq(index)** 匹配一个给定索引值的元素

**:gt(index)** 匹配所有大于给定索引值的元素

**:lt(index)**  匹配所有小于给定索引值的元素

**:header**  匹配如 h1, h2, h3之类的标题元素

**:animated**  匹配所有正在执行动画效果的元素

#### 内容过滤器

**:contains(text)**  匹配包含给定文本的元素

**:empty**  匹配所有不包含子元素或者文本的空元素

**:parent**  匹配含有子元素或者文本的元素

**:has(selector)**  匹配含有选择器所匹配的元素的元素

#### 可见性过滤器

**:hidden**  匹配所有不可见元素，或者type为hidden的元素

**:visible** 匹配所有可见元素

#### 属性过滤器

[attribute] 匹配包含给定属性的元素。

[attribute=value] 匹配给定的属性是某个特定值的元素

[attribute!=value] 匹配所有不含有指定的属性，或者属性不等于特定值的元素

[attribute^=value] 匹配给定的属性是以某些值开始的元素

[attribute$=value] 匹配给定的属性是以某些值结尾的元素

[attribute\*=value] 匹配给定的属性是以包含某些值的元素

[selector][selector]... 复合属性选择器，需要同时满足多个条件时使用

#### 表单过滤器

:input 匹配所有 input, textarea, select 和 button 元素

:text 匹配所有 文本输入框

:password 匹配所有的密码输入框

:radio 匹配所有的单选框

:checkbox 匹配所有的复选框

:submit 匹配所有提交按钮

:image 匹配所有img标签

:reset 匹配所有重置按钮

:button 匹配所有input type=button <button>按钮

:file 匹配所有input type=file文件上传

:hidden 匹配所有不可见元素display:none 或 input type=hidden

#### 表单对象属性过滤器

:enabled 匹配所有可用元素

:disabled 匹配所有不可用元素

:checked 匹配所有选中的单选，复选，和下拉列表中选中的option标签对象

:selected 匹配所有选中的option

## jQuery元素筛选

### 过滤

eq(index|-index) 选出索引为index的元素（0开始）,负号表示可以倒着选（-1开）

first() 选出第一个元素

last() 选出最后一个元素

hasClass(class) 是否含有class值

filter(expr|obj|ele|fn) 按照表达式过滤

is(expr|obj|ele|fn) 是否满足表达式

has(expr|ele) 是否含有表达式的元素

not(expr|ele|fn) 判断一个元素是不是符合表达式

slice(start,[end]) 表示从start开始选择直到end,只传递一个start表示从start开始直至结束

### 查找

children([expr]) 查找所有子元素，传入表达式，表示满足表达式的子元素

closest(expr,[con]|obj|ele) 表示查找和当前元素最接近的元素

find(expr|obj|ele) 表示查找元素，查找的是后代元素

next([expr]) 查找下一个元素

nextall([expr]) 查找下面所有的元素

nextUntil([exp|ele][,fil]) 查找相邻相邻元素一致到结束

offsetParent() 返回第一个匹配元素用于定位的父节点。这返回父元素中第一个其position设为relative或者absolute的元素。此方法仅对可见元素有效。

parent([expr]) 返回父元素

parents([expr]) 返回所有祖先元素

parentsUntil([exp|ele][,fil]) 返回所有祖先元素直到满足表达式为止

prev([expr]) 返回之前的那个元素

prevall([expr]) 返回之前所有的兄弟元素

prevUntil([exp|ele][,fil]) 返回之前所有兄弟元素直到满足表达式为止

siblings([expr]) 返回满足表达式的同辈元素

### 串联

add(expr|ele|html|obj[,con]) 并联关系。选择当前的元素和add条件中的元素，返回的是这些元素的集合

## jQuery的属性操作

**html()** 设置和获取起始标签和结束标签中的内容，有参表示设置，无参表示获取 innerHTML一样

**text()** 设置和获取起始标签和结束标签中的文本，有参表示设置，无参表示获取 innerText一样

**val()** 设置和获取表单项的value属性值，有参表示设置，无参表示获取 value属性

**attr()**设置和获取属性值。不推荐操作checked、selected、readOnly、disabled这些属性（这些属性在dom对象中值都是true和false的情况。使用attr操作都会返回undefined的情况），attr方法可以操作自定义属性。

**prop()**设置和获取属性值。只推荐操作checked、selected、readOnly、disabled这些属性（这些属性在dom对象中值都是true和false的情况。使用attr操作都会返回undefined的情况）

## CSS样式操作

**addClass()**  添加样式

**removeClass()**  删除样式

**toggleClass()**  有就删除，没有就添加

**offset()**  设置和获取坐标信息

## Dom对象的增删改操作

### 内部插入

appendTo() a.appendTo(b) 把a追加到 b 子元素，成为b的最后一个子元素

prependTo() a.prependTo(b) 把a 插入到b子元素前面。成为b第一个子元素

### 外部插入

insertAfter() a.insertAfter(b); 在b后面插a ba

insertBefore() a.insertBefore(b); 在b前面插a ab

### 替换

replaceWith() a.replaceWith(b) b替换a

replaceAll() a.replaceAll(b) a替换b

### 删除

remove() a.remove() 删除a

empty() a.empty() a被清空

## jQuery动画操作

### 基本动画

**show()**  显示正在隐藏的元素

**hide()**  隐藏正在显示的元素

**toggle()**  隐藏就显示，可见就隐藏。

第一个参数是动画执行的时长，毫秒为单位

第二个参数是回调函数（动画执行完就自动执行的函数。叫回调函数）。

### 淡入淡出动画

**fadeIn()** 淡入 慢慢可见

**fadeOut()** 淡出 慢慢消失

**fadeTo()**  在指定的时间内将透明度修改到指定的值，0全透明 ，1完全可见，0.5半透明

**fadeToggle()**  切换调用fadeIn和fadeOut

第一个参数是动画执行的时长，毫秒为单位

第二个参数是回调函数（动画执行完就自动执行的函数。叫回调函数）。

## jQuery事件操作

### 页面加载事件

**$(function(){})和window.onload = function(){}的区别**

他们分别是在什么时候触发？

jQuery的页面加载，是在浏览器内核解析完标签创建完标签对象之后就调用；

原生js的页面加载，是在浏览器内核解析完标签创建好dom对象，还要等标签中需要显示的内容加载完成。

他们触发的顺序？

先执行jQuery的页面加载，再执行原生的js页面加载。

他们执行的次数？

原生js因为是赋值操作，所以只会执行最后一次页面加载；

jQuery的页面加载完成之后，会将所有注册的函数全部一次调用。

### 事件相关函数

click() 绑定单击事件

mouseover() 鼠标移入

mouseout() 鼠标移除

bind() 给元素绑定事件（可以一次绑定多个）

one() 给元素绑定只响应一次的事件

live() 给匹配了选择器的元素绑定事件，哪怕这个元素是后面代码创建的

unbind() 取消事件的绑定，和bind()相反的操作

### 事件的冒泡

什么是事件的冒泡？

事件的冒泡是指，父子元素同时监听同一个事件。当触发子元素的事件的时候，同一个事件也被传递到了父元素的事件里去响应。

那么如何阻止事件冒泡呢？

在事件函数体内，return false; 可以阻止事件的冒泡传递。

### javaScript事件对象

事件对象，是封装有触发的事件信息的一个javascrip对象。在给元素绑定事件的时候，在事件的function( event )参数列表中添加一个参数，这个参数名，我们习惯取名为event，这个event就是javascript传递参事件处理函数的事件对象。

**原生javascript获取事件对象**

window.onload = function(){  
 document.getElementById("areaDiv").onclick = function(event){  
 console.log(event);  
 }  
}

**Query代码获取事件对象**

$(function(){  
 $("#areaDiv").click(function(event){  
 console.log(event);  
 });  
});

**使用bind同时对多个事件绑定同一个函数，获取当前操作事件**

$(function(){  
 $("#areaDiv").bind("mouseover mouseout",function(event){  
 console.log("bind");  
 if (event.type == "mouseover") {  
 console.log("鼠标进来了");  
 } else if (event.type == "mouseout") {  
 console.log("鼠标出去了");  
 }  
 });  
});

# Json

## 概述

Json(**JavaScript Object Notation**)即JavaScript对象表示法，是存储和交换文本信息的语法，类似于XML，但是比XML更小、更快、更易解析。Json采用完全独立于语言的文本格式，很多语言都提供了对Json的支持（包括C、C++、C#、Java、JavaScript、Python等）。

### json结构

• “名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为对象（object），纪录（record），结构（struct），字典（dictionary），哈希表（hash table），有键列表（keyed list），或者关联数组 （associative array）。

• 值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）。

### json形式

对象是一个无序的“‘名称/值’对”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间使用“,”（逗号）分隔。

数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。

值（value）可以是双引号括起来的字符串（string）、数值(number)、true、false、 null、对象（object）或者数组（array）。这些结构可以嵌套。

字符串（string）是由双引号包围的任意数量Unicode字符的集合，使用反斜线转义。一个字符（character）即一个单独的字符串（character string）。字符串（string）与C或者Java的字符串非常相似。

## Json在JavaScript中的使用

### Json的定义

json是由键值对组成。键必须使用引号引起来，键和值之间使用冒号进行分隔，多组键值对之间使用逗号进行分隔，由花括号（大括号）包含所有的键值对。

json的值可以有很多种数据类型：number类型、String类型、boolean类型、json对象、js的数组、json数组。

“key1” : value , // Number类型

“key2” : “value” , // 字符串类型

“key3” : [] , // 数组类型

“key4” : {}, // json 对象类型

“key5” : [{},{}] // json 数组

|  |
| --- |
| let jsonVar = {  "key1":1,  "key2":"str2",  "key3":[3, "str3", true],  "key4":{"key4\_1":41, "key4\_2":"str4\_2"},  "key5":[{"key5\_1\_1":511, "key5\_1\_2":"str512"}, {"key5\_2\_1":521, "key5\_2\_2":"str522"}] } |

### Json的访问

|  |
| --- |
| console.log(jsonVar.key1); console.log(jsonVar.key2); console.log(jsonVar.key3[2]); console.log(jsonVar.key4.key4\_2); console.log(jsonVar.key5[0].key5\_1\_2); |

### Json的转换

在JavaScript中json和字符串转换会用到两个方法：JSON.stringify()和JSON.parse()。

|  |
| --- |
| let jsonString = JSON.stringify(jsonVar); //json对象转json字符串 let jsonObj = JSON.parse(jsonString); //json字符串转json对象 |

## Json在Java中的使用

json在Java中使用需要导入Google提供的用来操作json的工具包：gson.jar。

### Json和JavaBean转换

|  |
| --- |
| Person person = new Person(1, "java"); Gson gson = new Gson(); //使用gson的toJson方法可以将JavaBean对象转换为字符串 String s = gson.toJson(person); //使用gson的fromJson方法可以将json字符串转换为JavaBean对象，传入参数为json字符串和转换的对象类型 Person newPerson = gson.fromJson(s, Person.class); |

### Json和List转换

|  |
| --- |
| public void method() {  List<Person> persons = new ArrayList<>();  persons.add(new Person(1, "p1"));  persons.add(new Person(2, "p2"));  persons.add(new Person(3, "p3"));  Gson gson = new Gson();  String listJson = gson.toJson(persons);  //将json字符串转换为集合时需要传入TypeToken对象并调用getType方法  //此时需要该对象继承TypeToken类  List<Person> persons2 = gson.fromJson(listJson, new PersonListType().getType()); }  class PersonListType extends TypeToken<ArrayList<Person>> {} |

因为PersonListType类只在fromJson方法中使用一次，因此这里使用匿名内部类更为简洁，避免额外创建Java类。

### Json和Map转换

|  |
| --- |
| public void method() {  Map<String, Person> map = new HashMap<String, Person>();  map.put("p1", new Person(1, "person1"));  map.put("p2", new Person(2, "person2"));  map.put("p3", new Person(3, "person3"));  Gson gson = new Gson();  String mapJsonString = gson.toJson(map);  //为了避免每次将json字符串转换为集合都要新建一个空类，这里可以使用匿名内部类的方式获取TypeToken的对象  Map<String, Person> map2 = gson.fromJson(mapJsonString, new TypeToken<HashMap<String, Person>>(){}.getType()); } |

# Ajax

## 概述

Ajax（**Asynchronous Javascript And XML**）即异步JavaScript和XML，是一种创建交互式网页应用的网页开发技术，是一种浏览器通过js异步请求，局部更新页面的技术。Ajax不是新的编程语言，而是一种使用现有标准的新方法。

1.使用CSS和XHTML来表示。

2.使用DOM模型来交互和动态显示。

3.使用XMLHttpRequest来和服务器进行异步通信。

4.使用javascript来绑定和调用。

在上面几中技术中，除了XmlHttpRequest对象以外，其它所有的技术都是基于web标准并且已经得到了广泛使用，XMLHttpRequest虽然目前还没有被W3C所采纳，但是它已经是一个事实的标准，因为目前几乎所有的主流浏览器都支持它。

### 作用

通过在后台与服务器进行少量数据交换，Ajax可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页（不使用 Ajax）如果需要更新内容，必需重载整个网页面。

传统的网页，如果需要更新内容，必须加载整个网页。如果只需要对网页上的某个内容进行局部刷新， 那么就需要使用到Ajax了。它可以让我们无需重新加载全部网页内容，即可完成对某个部分的内容执行刷新。最典型的的例子，莫过于大家在注册网站的时候，输入的用户名，会自动的提示我们，该用户已被注册。

Ajax技术一般可以用在用户登陆，页面逐步加载等方面。

### 内部原理

Ajax 并不是一项新技术，而是包装了现有的技术，然后使用他们来完成工作而已。

以判断用户名是否已被注册为例。

传统方式：

1. 输入用户名，

2. 点击一个按钮，校验。

3. 把数据提交给服务器

4. 服务器在后台帮助我们完成校验，并且反馈信息。

5. 我们在浏览器上提示用户，给出结果

Ajax方式：

ajax方式与前面的方式其实从要做的事情来说，是一样的。 ajax也没有牛到，不用去访问服务器就知道你的用户名是否已被占用。那么它是如何工作的呢?

1. 通过JS 获取咱们的输入框文本内容 document.getElementById("username").value

2. 通过XmlHttpRequest 去执行请求。 XmlHttpRequest 其实就是 XML + http + Request 的组合。

3. 请求结束后，收到结果， 再使用 js 去完成提示。

4. 可以在顺便配合 css 样式来增加提示效果。

## 使用方式

### 请求访问

#### Get请求

1. 创建XmlHttpRequest 对象。 这部分创建对象的代码，由于需要针对不同的浏览器， 需要做出判断，并且还没有什么提示， 所以大家可以不用自己写。 往后直接拷贝即可。

function ajaxFunction(){

var xmlHttp;

try{ // Firefox, Opera 8.0+, Safari

xmlHttp=new XMLHttpRequest();

}catch (e){

try{// Internet Explorer

xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");

}catch (e){

try{

xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

catch (e){}

}

}

return xmlHttp;

}

2. 发送请求

//不带数据，直接请求。

//获取 xmlhttprequest 对象

var request = ajaxFunction();

// 参数一： 执行 get 请求 ， 参数二 ： 请求的地址 ， 参数三：　是否是异步请求。

request.open("GET", "Demo01", true);

//发送请求。

request.send();

--------------------------------以下带上数据--------------------------------------

//获取 xmlhttprequest 对象

var request = ajaxFunction();

// 参数一： 执行 get 请求 ， 参数二 ： 请求的地址 ， 参数三：　是否是异步请求。

request.open("GET", "Demo01?name=zhangsan&age=18", true);

request.send();

如果发送请求的同时，还想获取数据，那么代码如下

//执行get请求

function get() {

//1. 创建xmlhttprequest 对象

var request = ajaxFunction();

//2. 发送请求

request.open("GET" ,"/day16/DemoServlet01?name=aa&age=18" ,true );

//3. 获取响应数据 注册监听的意思。 一会准备的状态发生了改变，那么就执行 = 号右边的方法

request.onreadystatechange = function(){

//前半段表示 已经能够正常处理。 再判断状态码是否是200

if(request.readyState == 4 && request.status == 200){

//弹出响应的信息

alert(request.responseText);

}

}

request.send();

}

#### Post请求

Post请求 和 Get请求基本相似，区别就在于 数据传输方式不同。 Get方式是直接在地址的后面拼接的。 但是Post的方式是通过send 方法传输过去的。 并且还要设置一个请求头。

基本上，如果明白了Http的协议，那么对Post请求的代码理解起来就不是那么困难了。

1. 创建XmlHttpRequest 对象。 这部分创建对象的代码，由于需要针对不同的浏览器， 需要做出判断，并且还没有什么提示， 所以大家可以不用自己写。 往后直接拷贝即可。

function ajaxFunction(){

var xmlHttp;

try{ // Firefox, Opera 8.0+, Safari

xmlHttp=new XMLHttpRequest();

}catch (e){

try{// Internet Explorer

xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");

}catch (e){

try{

xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

catch (e){}

}

}

return xmlHttp;

}

2. 发送请求

//不带数据，直接请求。

//获取 xmlhttprequest 对象

var request = ajaxFunction();

// 参数一： 执行 get 请求 ， 参数二 ： 请求的地址 ， 参数三：　是否是异步请求。

request.open("POST", "Demo01", true);

//发送请求。

request.send();

--------------------------------以下带上数据--------------------------------------

//获取 xmlhttprequest 对象

var request = ajaxFunction();

// 参数一： 执行 get 请求 ， 参数二 ： 请求的地址 ， 参数三：　是否是异步请求。

request.open("POST", "Demo01", true);

//设置请求头，其实就是表示传输的是一个经过url编码的form表单数据

request.setHeader("Content-Type" , "application/x-www-form-urlencoded");

request.send("name=zhangsan&age=18");

function post() {

//1. 创建对象

var request = ajaxFunction();

//2. 发送请求

request.open( "POST", "/day16/DemoServlet01", true );

//想获取服务器传送过来的数据， 加一个状态的监听。

request.onreadystatechange=function(){

if(request.readyState==4 && request.status == 200){

alert("post："+request.responseText);

}

}

//如果使用的是post方式带数据，那么 这里要添加头， 说明提交的数据类型是一个经过url编码的form表单数据

request.setRequestHeader("Content-type","application/x-www-form-urlencoded");

//带数据过去 ， 在send方法里面写表单数据。

request.send("name=aobama&age=19");

}

### 获取响应

数据提交到的两种请求Get 和 Post基本上都会使用后， 就应该想想获取数据的问题了。 前面我们都一直很无私的往服务器提交数据， 那么服务器返回的那些数据，我们又该怎么通过Ajax去拿呢?

这里就要靠xmlhttprequest的事件 onreadystatechange ， 这是用来监听我们请求的一些状态， 比如： 成功了还是失败了。 如果成功，想获取数据，得靠xmlhttprequest的responseText 或 responseXML 属性

完整例子：

//获取 xmlhttprequest 对象 代码跟以前一样，此处不再赘述。

var request = ajaxFunction();

// 参数一： 执行 get 请求 ， 参数二 ： 请求的地址 ， 参数三：　是否是异步请求。

request.open("GET", "Demo01", true);

//对请求的状态 进行监听。

request.onreadystatechange = function(){

if(request.readyState == 4 && request.status == 200 ){

//设置节点id为 myDiv的标签 结果为 我们请求得到的响应文字

document.getElementById("myDiv").innerHTML=request.responseText;

}

}

//发送请求。

request.send();

### 异步校验用户名

Ajax Get:

var request = ajaxFunction();

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

request.onreadystatechange=function(){

if (request.readyState==4 && request.status==200){

var result = request.responseText;

if(result == 1){

document.getElementById("span1").innerHTML="<font color='red'>用户名已被占用</font>";

}else{

document.getElementById("span1").innerHTML="<font color='green'>用户名可以使用</font>";

}

}

}

request.open("GET", "RegisterServlet?name="+name, true);

request.send();

字符串比较：

if(result == '1'){

document.getElementById("span1").innerHTML="<font color='red'>用户名已被占用</font>";

}else{

document.getElementById("span1").innerHTML="<font color='green'>用户名可以使用</font>";

}

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

Ajax Post:

function checkName02(){

var request = ajaxFunction();

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

request.onreadystatechange=function(){

if (request.readyState==4 && request.status==200){

var result = request.responseText;

if(result == '1'){

document.getElementById("span1").innerHTML="<font color='red'>用户名已被占用22</font>";

}else{

document.getElementById("span1").innerHTML="<font color='green'>用户名可以使用22</font>";

}

}

}

request.open("POST", "RegisterServlet", true);

request.setRequestHeader("Content-type","application/x-www-form-urlencoded");

request.send("name="+name);

}

## js的Ajax请求

JavaScript原生的Ajax请求需要创建XMLHttpResquest对象，然后调用该对象的open方法设置请求参数，最后调用该对象的send方法发送请求，在send方法之前绑定onreadystatechange事件，处理请求完成后的操作。该对象的open方法的三个参数分别为：

method 请求的类型，get或post

url 请求的资源地址

async true（异步）或false（同步）

该对象的send方法当且仅当请求方式为post请求时可以传入String类型的参数。

<**script type="text/javascript"**>  
 **function** *ajaxRequest*() {  
 //创建XMLHttpRequest对象  
 **let** xmlHttpRequest = **new *XMLHttpRequest***();  
 //调用open方法设置请求参数  
 xmlHttpRequest.open(**"get"**, **"http://localhost:8080/day605/ajaxServlet?action=javaScriptAjax"**, **true**);  
 //在send发送方法前绑定onreadystatechange事件，处理请求完成后操作  
 xmlHttpRequest.**onreadystatechange** = **function** () {  
 //判断请求已完成，而且是成功的请求  
 **if** (xmlHttpRequest.**readyState** == 4 && xmlHttpRequest.**status** == 200) {  
 // document.getElementById("div01").innerHTML = xmlHttpRequest.responseText;  
 **let** jsonObj = ***JSON***.parse(xmlHttpRequest.**responseText**);  
 ***document***.getElementById(**"div01"**).**innerHTML** = **"编号："** + jsonObj.**id** + **" , 姓名："** + jsonObj.**name**;  
 }  
 }  
 //调用发送请求  
 xmlHttpRequest.send();  
 }  
</**script**>

<**body**>  
<**button onclick="***ajaxRequest*()**"**>ajax request</**button**>  
<**div id="div01"**></**div**>  
</**body**>

**protected void** javaScriptAjax(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 Person person = **new** Person(1,"xmm");  
 Gson gson = **new** Gson();  
 String pjson = gson.toJson(person);  
 response.getWriter().write(pjson);  
}

编写原生的JavaScript要写很多的代码，还要考虑浏览器兼容问题，所以使用起来非常的不方便。之后一般会使用JavaScript的框架Jquery来解决这个问题，它有很好的Ajax解决方案。

## jQuery的Ajax请求

jQuery针对Ajax有四个方法可以来简化代码的编写，分别为$.ajax、$.get、$.post和$.getJSON。

### $.ajax方法

**url** 请求的资源地址

**type** 请求的方式，get或post

**data** 请求的参数，name=value&name=value或{key1:value,key2:value}

success 响应的函数（成功的回调函数）

error 失败的回调函数

**dataType** 返回的数据类型，常用值是text、xml、json

**$**.**ajax**({  
 **url**: **"http://localhost:8080/day605/ajaxServlet"**,  
 **type**: **"GET"**,  
 **data**: **"action=jqueryAjax"**,  
 //响应成功时data为获取到的响应数据  
 success: **function** (data) {  
 **$**(**"#msg"**).html(**"编号："** + data.**id** + **" , 姓名："** + data.**name**);  
 ***console***.log(**"jqueryAjax"**);  
 },  
 //类型设置为json对于获取到的响应数据就不用再转换为json了  
 **dataType**: **"json"**});

**protected void** jqueryAjax(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 System.**out**.println("jqueryAjax");  
 Person person = **new** Person(1,"xmm");  
 Gson gson = **new** Gson();  
 String pjson = gson.toJson(person);  
 response.getWriter().write(pjson);  
}

#### $.ajax用法

请求体为json格式

当要发送的数据结构比较复杂，使用这个用法，此时handler方法必须使用Spring中的@RequestBody注解

|  |
| --- |
| // 封装函数执行批量删除 function doBatchRemove(adminIdArray, pageNum, keyword) {  // 以请求体方式发送adminIdArray需要将adminIdArray转换为JSON字符串  var requestBody = JSON.stringify(adminIdArray);  // 发送Ajax请求执行删除  $.ajax({  "url": "admin/batch/remove.json", // 请求的URL地址  "type": "post", // 请求方式  "data": requestBody, // 请求体数据  "contentType": "application/json;charset=UTF-8", // 设置请求体的内容类型  "dataType":"json", // 将服务器返回的数据按照JOSN格式解析  "success":function(response) { // 请求处理成功时执行的回调函数（响应状态码200）  ……  }); } |

请求体像表单那样提交请求参数

当要发送的数据比较简单，就是键值对，建议使用这个用法。但是请注意：如果是像表单提交数据这样发送请求参数，那么handler还是正常按照请求参数接收数据，不能使用@RequestBody注解。

|  |
| --- |
| // 发送Ajax请求执行删除 $.ajax({  "url": "admin/batch/remove.json", // 请求的URL地址  "type": "post", // 请求方式  "data": {  "empName":"tom",  "empAge":"20",  "empSalary":"1000.5"  }, // 请求体数据  // contentType的默认值就是application/x-www-form-urlencoded，可以省略  "contentType": "application/x-www-form-urlencoded",  ……  ) |

### $.get和$.post方法

**url** 请求的资源地址

**data** 请求的参数，name=value&name=value或{key1:value,key2:value}

**callback** 响应的函数（成功的回调函数）

**type** 返回的数据类型，常用值是text、xml、json

**$**(**"#getBtn"**).click(**function** () {  
 **$**.get(  
 **"http://localhost:8080/day605/ajaxServlet"**,  
 {**"action"**: **"jqueryGet"**},  
 **function** (data) {  
 **$**(**"#msg"**).html(**"编号："** + data.**id** + **" , 姓名："** + data.**name**);  
 ***console***.log(**"jqueryGet"**);  
 },  
 **"json"** );  
});

**protected void** jqueryGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 System.**out**.println("jqueryGet");  
 Person person = **new** Person(1,"xmm");  
 Gson gson = **new** Gson();  
 String pjson = gson.toJson(person);  
 response.getWriter().write(pjson);  
}

**$**(**"#postBtn"**).click(**function** () {  
 **$**.post(  
 **"http://localhost:8080/day605/ajaxServlet"**,  
 {**"action"**: **"jqueryPost"**},  
 **function** (data) {  
 **$**(**"#msg"**).html(**"编号："** + data.**id** + **" , 姓名："** + data.**name**);  
 ***console***.log(**"jqueryPost"**);  
 },  
 **"json"** );  
});

**protected void** jqueryPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 System.**out**.println("jqueryPost");  
 Person person = **new** Person(1,"xmm");  
 Gson gson = **new** Gson();  
 String pjson = gson.toJson(person);  
 response.getWriter().write(pjson);  
}

### $.getJSON方法

**url** 请求的资源地址

**data** 请求的参数，name=value&name=value或{key1:value,key2:value}

**callback** 响应的函数（成功的回调函数）

**$**.getJSON(  
 **"http://localhost:8080/day605/ajaxServlet"**,  
 //serialize方法可以获取所有表单信息并以键值对的形式存储  
 **"action=jquerySerialize&"** + **$**(**"#form01"**).serialize(),  
 **function** (data) {  
 **$**(**"#msg"**).html(**"编号："** + data.**id** + **" , 姓名："** + data.**name**);  
 ***console***.log(**"jquerySerialize"**);  
 }  
);

**protected void** jquerySerialize(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 System.**out**.println("jquerySerialize");  
 System.**out**.println(request.getParameter("username"));  
 System.**out**.println(request.getParameter("password"));  
 Person person = **new** Person(1,"xmm");  
 Gson gson = **new** Gson();  
 String pjson = gson.toJson(person);  
 response.getWriter().write(pjson);  
}

### $.ajax()请求的异步和同步

异步方式下$.ajax()请求函数不会等待服务器响应，同步方式下$.ajax()请求函数一定会等服务器返回响应后再执行后续操作。

不管是Java程序还是浏览器，在同一个线程内，按顺序执行的操作一定是同步的，因为A方法调用B方法，A必须等B方法执行完成，才能够继续执行后续操作。如果能够开启多个线程，那个各个线程内部是同步的，各个线程之间是并行推进，同时执行，谁也不用等谁，这就是异步执行模式。

# Pagination

pagination是一个基于jQuery的框架，专门用于处理前端页面中的分页导航条。

num\_edge\_entries边缘页数

num\_display\_entries主题页数

callback:pageselectCallback指定回调函数pageselectCallback，用户点击按钮时跳转页面，调用这里的回调函数完成页面的跳转。

items\_per\_page每页显式数据数量（size和pagesize区分最后一页时的数据数量）

jsp先翻译为java文件，再编译为class文件。

&keyword=${param.keyword}中${param.keyword}一定要写在双引号里面，否则编译无法通过。

将pagination框架中的最后一行有效代码中的回调函数opts.callback(current\_page,this);注释掉。

# zTree