

前

端

笔

记

# Html

Html：超级文本标记语言（HyperText Markup Language），在浏览器上运行的一种标记语言。

就是给文本加上含有语义的标签。

接下来应该学习更多具体语义标签：

## 一、结构（固定的结构）

<html>

<head>

<title></title>

</head>

<body></body>

## 二、Html中的标签

标签：用“<>”包起来的内容。

Html标签：<html></html>

由于现在学习的html页面还是很简单的，为了能够让页面更加的“充实”我们要学习更多的标签。

### h系列的标签(Header)：h1,h2,h3,h4,h5,h6

作用：把页面上的文字加上标题的语义。

代码：

h1,h2,h3,h4,h5,h6:将文字放大，加粗.并且逐渐变小。（h1标签是最大的。）

注意：一个页面只能有一个h1标签。

### p标签(Paragraph)：段落标签

作用：给页面的上一段文字加上段落的说语义

代码：

<p>这是段落</p>

特点：页面上的一个段落，单独占一行，并且与上下内容之间存在距离。

### hr标签(Horizontal Rule)

作用：在页面显示一条横线。

代码：

<hr />

特点：在页面显示一条横线，默认占整行。

### br标签(break)

作用：换行。

代码：

<br />

特点：换行。

### b(Bold),u(Underlined),i(Italic),s(Strikethrough),strong,em,del,ins

作用：

b：加粗

u: 下划线

i: 倾斜

s: 删除线

建议不要使用，因为这些标签没有语义。

作用：

strong:加粗

ins:下划线

em:倾斜

del:删除线

可以使用

### img标签(image)：

作用：在页面显示一张图片。

代码：

<img src=”图片的路径” />

特点：就是显示图片.

属性：

|  |  |
| --- | --- |
| src | 图片显示的路径 |
| alt | 如果图片加载不出来会显示这个属性中的文字 |
| title | 介绍这张图片 |

### a标签(Anchor)：锚

a标签的其它名称：

超级链接，超链接，锚链接。

作用：可以在一个页面跳转到另一个页面。

代码：

<a href=”页面的路径”></a>

注意：在a标签之内必须要写上文字，如果没有，那么这个标签在页面上找不到。

a标签的其它用法：

1.0可以不跳转（跳转到当前页面）href=”#”

2.0可以跳转到另外的页面。

3.0可以在当前页面进行定位。

1. 设置a标签的href属性为“#id名”，
2. 在页面的指定位置加入一个目标标签（可以是任意标签）
3. 必须给这个标签设置一个id名：<p id=”mubiao”>这是目标</p>

4.0在跳转的页面进行定位。

5.0可以进行下载。（强烈不推荐使用）

a标签的属性：

|  |  |
| --- | --- |
| href | a标签跳转的目标地址 |
| target | \_blank:保留原始页面，再进行跳转  \_self:在当前页面进行跳转 |
| base | 为页面上所有的a标签设置跳转的方式（base标签一般放在titile标签下面） |

### 列表

#### 有序列表

有序列表即为有排列顺序的列表，其各个列表项按照一定的顺序排列定义，有序列表的基本语法格式如下：

<ol>

<li>列表项1</li>

<li>列表项2</li>

<li>列表项3</li>

......

</ol>

#### 无序列表

无序列表的各个列表项之间没有顺序级别之分，是并列的。其基本语法格式如下：

<ul>

<li>列表项1</li>

<li>列表项2</li>

<li>列表项3</li>

......

</ul>

自定义列表

定义列表常用于对术语或名词进行解释和描述，定义列表的列表项前没有任何项目符号。其基本语法如下：

<dl>

<dt>名词1</dt>

<dd>名词1解释1</dd>

<dd>名词1解释2</dd>

...

<dt>名词2</dt>

<dd>名词2解释1</dd>

<dd>名词2解释2</dd>

...

</dl>

### 表格

#### 创建表格

在HTML网页中，要想创建表格，就需要使用表格相关的标签。创建表格的基本语法格式如下：

<table>

<tr>

<td>单元格内的文字</td>

...

</tr>

...

</table>

在上面的语法中包含三对HTML标签，分别为 <table></table>、<tr></tr>、<td></td>，他们是创建表格的基本标签，缺一不可，下面对他们进行具体地解释。

1、<table>< </table>：用于定义一个表格。

2、<tr>< </tr>：用于定义表格中的一行，必须嵌套在 <table> </table>标签中，在 <table> </table>中包含几对 <tr> </tr>，就有几行表格。

3、<td> </td>：用于定义表格中的单元格，必须嵌套在< <tr> </tr>标签中，一对 <tr> </tr>中包含几对< <td> </td>，就表示该行中有多少列（或多少个单元格）。

4、thead：表格头部

5、tbody：表格主体

6、tfoot：表格底部

7、caption表格标题

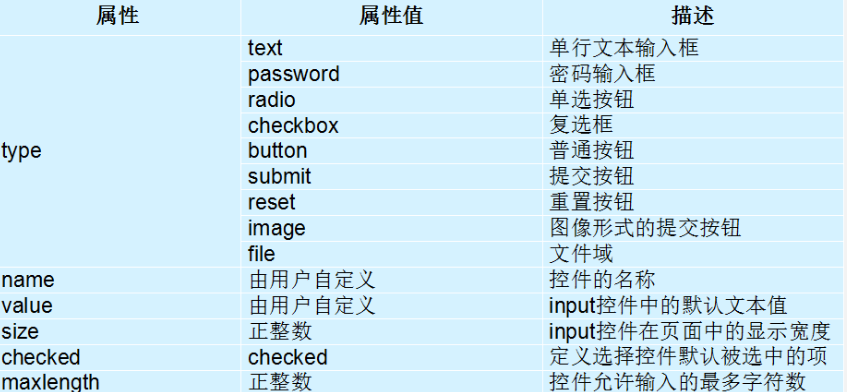
#### 表格属性



### 表单控件

#### input控件

<input />标签为单标签，type属性为其最基本的属性，其取值有多种，用于指定不同的控件类型。除了type属性之外，<input />标签还可以定义很多其他的属性，其常用属性如下表所示。



#### label 标签

label 标签为 input 元素定义标注（标记）。

作用： 用于绑定一个表单元素, 当点击label标签的时候, 被绑定的表单元素就会获得输入焦点

如何绑定元素呢？

for 属性规定 label 与哪个表单元素绑定。

<label for="male">Male</label>

<input type="radio" name="sex" id="male" value="male">

#### textarea 标签

如果需要输入大量的信息，就需要用到<textarea></textarea>标签。通过textarea控件可以轻松地创建多行文本输入框，其基本语法格式如下：

<textarea cols="每行中的字符数" rows="显示的行数">

文本内容

</textarea>

#### select标签

使用select控件定义下拉菜单的基本语法格式如下

<select>

<option>选项1</option>

<option>选项2</option>

<option>选项3</option>

...

</select>

<select></select>中至少应包含一对<option></option>。

在option 中定义selected =" selected "时，当前项即为默认选中项。

#### button标签

<button> 按钮 </button> 单独的一个表单控件，就是普通按钮的意思。 只不过它是双标签哦！

#### form表单

在HTML中，form标签被用于定义表单域，即创建一个表单，以实现用户信息的收集和传递，form中的所有内容都会被提交给服务器。创建表单的基本语法格式如下：

<form action="url地址" method="提交方式" name="表单名称">

各种表单控件

</form>

常用属性：

Action 在表单收集到信息后，需要将信息传递给服务器进行处理，action属性用于指定接收并处理表单数据的服务器程序的url地址。

method 用于设置表单数据的提交方式，其取值为get或post。

name 用于指定表单的名称，以区分同一个页面中的多个表单。

注意： 每个表单都应该有自己表单域。

## 三、当前的主流文档类型：

|  |  |  |
| --- | --- | --- |
| 文档类型 | 分类 | 备注 |
| HTML | HTML Strict DTD | 请求比较严格的html类型 |
| HTML Transitional DTD | 相对而言比较规范不太严禁 |
| Frameset DTD | 框架级的类型 |
| xHTML | HTML Strict DTD | 请求比较严格的html类型 |
| HTML Transitional DTD | 相对而言比较规范不太严禁 |
| Frameset DTD | 框架级的类型 |

xHtml(html2.0版本（是一种相对html语法更加严谨的hmtl）):

注意：将来在使用DOCTYPE的时候建议使用html5的类型。每个页面都要设置一个doctype，如果不设置，浏览器会默认开启quirks mode（怪异模式）解析（quirks mode（怪异模式）是浏览器为了兼容很早之前针对旧版本浏览器设计、并未严格遵循 W3C 标准的网页而产生的一种页面渲染模式）。

### Html标签

作用所有html中标签的一个根节点。

### Head标签

作用：用于存放：

title,meta,base,style,script,link

注意：每个head标签中都必须有一个有title标签，其它的可选。

### Body标签

作用：用于存放所有的html的结构标签：

p,h,a,b,u,i,s,em,del,ins,strong,ul,li,ol,

### Title标签：

作用：让页面拥有一个属于自己的标题。

### meta 标签：

常用用法：

1. Description：可以描述页面，可以用来使用百度程序（网络爬虫）来收录关键信息，以此提高页面的排名。
2. Keywords：关键词，可以用来提高页。面的关健词的比重（前升排名的一种方式。）
3. 字符集（编码格式）：

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

Charset（字符集的格式）：UTF-8.

注意：字符集（文字在电脑是存储的字典）：

电脑是不能直接存储文字的，一般情况下电脑存储的是这个文字在“字典”里对应的位置。

gb2313---->gbk;

国标2313---->国标扩

### gbk,utf-8之间的区别：

相同点：都是字符的编码格式。

区别：

utf-8:收录是全世界所有的语言中的文字。

gbk:收录了汉字，片假名。

大小：

utf-8>gbk

性能：

uft-8<gbk

约定：将来我们在整个学习过程中都用utf-8;

utf-8存储一个汉字占3个字节，

gbk存储一个汉字上2个字节

## 四、标签的分类

双标签：有开始有结束，开始和结束之间是存在内容

<h1></h1>,<p></p>

单标签：只有一个标签，自己闭合自己。

<hr /> ,<br />

## 五、标签与标签之间的关系

嵌套关系：一个标签包含另外一个标签，他们之间构造父子关系。

并列关系：两个标签并列，他们构造兄弟关系。

注意：将来在书写代码的时候如果两个标签之间的关系是嵌套关系，最好通过代码直接反映出来（子元素相对于父元素有一个缩进）。如果是并列关系，最好要有对齐。

## 六、标签的属性

<img src=”a.jpg” />

属性为 HTML 元素提供附加信息，img标签中的src就是img标签的一个属性。

## 七、路径问题

第一种：绝对路径

带有盘符的路径：C:\上课内容\上课视频\01html第一天\4-源代码\1.jpg.

第二种：相对路径

由页面是一个文件，图片 也是一个文件，而现在需要在页面上输出图片，所以需要得到图片相对于文件的路径.

a.如果页面与图片在同一目录下面：



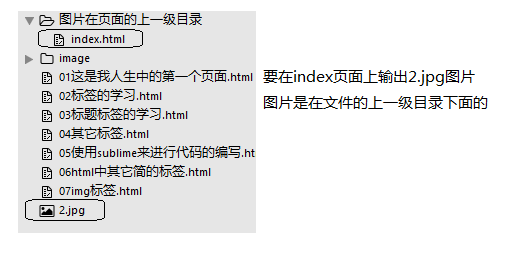
<img src=”2.jpg” />

b.如果页面在图片一上级目录：



<img src=”image/1.jpg” />

c.如果图片在页面的上一级目录：

  
 <img src=”../2.jpg” />

只要不出意外情况都是用相对路径：因为相对路径的可移植性要强。

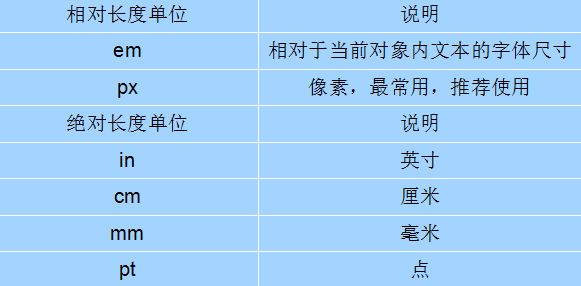
# CSS

## 一、字体设置及选择器

### font属性的设置

#### font-size字体大小

font-size属性用于设置字号，该属性的值可以使用相对长度单位，也可以使用绝对长度单位。其中，相对长度单位比较常用，推荐使用像素单位px，绝对长度单位使用较少



**Google和火狐默认字体大小为：16px；**

#### font-family字体

font-family属性用于设置字体。网页中常用的字体有宋体、微软雅黑、黑体等，例如将网页中所有段落文本的字体设置为微软雅黑，可以使用如下CSS样式代码：

p{ font-family:"微软雅黑";}

可以同时指定多个字体，中间以逗号隔开，表示如果浏览器不支持第一个字体，则会尝试下一个，直到找到合适的字体。

#### font-weight字体粗细

字体加粗除了用 b 和 strong 标签之外，可以使用css 来实现，但是css 是没有语义的。

font-weight属性用于定义字体的粗细，其可用属性值：normal、bold、bolder、lighter、100~900（100的整数倍）。

#### font-style字体风格

字体倾斜除了用 i 和 em 标签之外，可以使用css 来实现，但是css 是没有语义的。

font-style属性用于定义字体风格，如设置斜体、倾斜或正常字体，其可用属性值如下：

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。

### CSS中注释写法

CSS规则是使用 /\* 需要注释的内容 \*/ 进行注释的，即在需要注释的内容前使用 “/\*” 标记开始注释，在内容的结尾使用 “\*/”结束。

### 3、颜色的不同写法

a、直接写英文单词yellow red等等

b、16进制表示#aabbcc或者#abc

c、rgb（）如rgb（0，0，0）

d、rgba（0,0,0,0.5）a为透明度

### css各类选择器

#### 标签选择器

标签选择器写法

标签名{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

效果：会将所有的这类标签都加上相同的css属性

#### 类选择器

类选择器的写法

.类名{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

类选择器最大的优势是可以为元素定义单独或相同的样式。

将拥有这个类名的所有元素加上css样式

命名的取值规范只能是：0-9，a-z，A-Z，\_，-，不能以数字开头

#### id选择器

#id名{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

元素的id值是唯一的，只能对应于文档中某一个具体的元素。

将拥有这个id名的唯一元素加上css样式

### 关于text属性的设置

#### line-height行间距

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。

#### text-align文本内容水平对齐

text-align属性用于设置文本内容的水平对齐，相当于html中的align对齐属性。其可用属性值如下：

left：左对齐（默认值）

right：右对齐

center：居中对齐

#### text-indent首行文本缩进

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。

## 复合选择器及伪元素

### 复合选择器

#### 交集选择器

选择器选择器{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

注意：两个选择器之间是没有空格的。

#### 并集选择器

选择器 ，选择器{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

将两种类型的元素全部选择出来。

#### 后代选择器

选择器 选择器{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

注意：两个选择器之间是要有空格的。

#### 子代选择器

选择器>选择器{属性1:属性值1; 属性2:属性值2; 属性3:属性值3; }

注意：两个选择器之间是要有大于号隔开。

### 元素的三种显示方式

块转行内：display:inline;

行内转块：display:block;

块、行内元素转换为行内块： display: inline-block;

### 伪类和伪元素的使用

#### 伪类

|  |  |
| --- | --- |
| :link | 伪类将应用于未被访问过的链接 |
| :hover | 伪类将应用于有鼠标指针悬停于其上的元素。 |
| :active | 伪类将应用于被激活的元素，如被点击的链接、被按下的按钮等。 |
| :visited | 伪类将应用于已经被访问过的链接 |
| :focus | 伪类将应用于拥有键盘输入焦点的元素。（ie8以上支持） |

链接伪类，他们规定执行有顺序的，不能随便更改位置：

要按照 :link --> :visited --> :hover --> :active 的顺序。

#### text-decoration检索或设置对象中的文本的装饰

text-decoration : none || underline || blink || overline || line-through

none : 　无装饰

underline : 　下划线

line-through : 　删除线

overline : 　上划线

#### 伪元素

更改第一个字或字母的样式：first-letter

更改第一行文本的样式：first-line

向前面添加：before

向后面添加：after

### 4、css三大特性：继承性、层叠性、优先级、权重

|  |  |
| --- | --- |
| 继承或者\* 的贡献值 | 0,0,0,0 |
| 每个元素（标签）贡献值为 | 0,0,0,1 |
| 每个类，伪类贡献值为 | 0,0,1,0 |
| 每个ID贡献值为 | 0,1,0,0 |
| 每个行内样式贡献值 | 1,0,0,0 |
| 每个!important贡献值 | ∞ 无穷大 |

## 背景及盒模型

### background背景

#### 背景颜色

background-color：

背景图片

语法：

background-image : none | url (url)

参数：

none : 　无背景图（默认的） url : 　使用绝对或相对地址指定背景图像

background-image 属性允许指定一个图片展示在背景中（只有CSS3才可以多背景）可以和 background-color 连用。 如果图片不重复地话，图片覆盖不到地地方都会被背景色填充。 如果有背景图片平铺，则会覆盖背景颜色。

#### 背景平铺

语法：

background-repeat : repeat | no-repeat | repeat-x | repeat-y

参数：

repeat : 　背景图像在纵向和横向上平铺（默认的）

no-repeat : 　背景图像不平铺

repeat-x : 　背景图像在横向上平铺

repeat-y : 　背景图像在纵向平铺

设置背景图片时，默认把图片在水平和垂直方向平铺以铺满整个元素。

repeat-x : 　背景图像在横向上平铺

repeat-y : 　背景图像在纵向平铺

#### 背景位置

语法：

background-position : length || length

background-position : position || position

参数：

length : 　百分数 | 由浮点数字和单位标识符组成的长度值。请参阅长度单位 position : 　top | center | bottom | left | center | right

说明：

设置或检索对象的背景图像位置。必须先指定background-image属性。默认值为：(0% 0%)。 如果只指定了一个值，该值将用于横坐标。纵坐标将默认为50%。第二个值将用于纵坐标。

注意：

position 后面是x坐标和y坐标。 可以使用方位名词或者 精确单位。

如果和精确单位和方位名字混合使用，则必须是x坐标在前，y坐标后面。比如 background-position: 15px top; 则 15px 一定是 x坐标 top是 y坐标。

#### 背景附着

background-attachment：scroll（滚动）

background-attachment：fixed（固定）

### 盒子模型

#### border边框

语法：

border : border-width || border-style || border-color

边框属性—设置边框样式（border-style）

边框样式用于定义页面中边框的风格，常用属性值如下：

none：没有边框即忽略所有边框的宽度（默认值）

solid：边框为单实线(最为常用的)

dashed：边框为虚线

dotted：边框为点线

double：边框为双实线

表格边框问题

border-collapse:collapse; 表示边框合并在一起。

border-spacing:px; 定义边框之间的间距。

#### padding内边距

padding-top:上内边距

padding-right:右内边距

padding-bottom:下内边距

padding-left:左内边距

|  |  |
| --- | --- |
| **值的个数** | **表达意思** |
| 1个值 | padding：上下左右边距 比如padding: 3px; 表示上下左右都是3像素 |
| 2个值 | padding: 上下边距 左右边距 比如 padding: 3px 5px; 表示 上下3像素 左右 5像素 |
| 3个值 | padding：上边距 左右边距 下边距 比如 padding: 3px 5px 10px; 表示 上是3像素 左右是5像素 下是10像素 |
| 4个值 | padding:上内边距 右内边距 下内边距 左内边距 比如: padding: 3px 5px 10px 15px; 表示 上3px 右是5px 下 10px 左15px 顺时针 |

#### margin外边距

margin属性用于设置外边距。 设置外边距会在元素之间创建“空白”， 这段空白通常不能放置其他内容。

margin-top:上外边距

margin-right:右外边距

margin-bottom:下外边距

margin-left:上外边距

margin:上外边距 右外边距 下外边距 左外边

取值顺序跟内边距相同。

#### 宽度高度

使用宽度属性width和高度属性height可以对盒子的大小进行控制。

width和height的属性值可以为不同单位的数值或相对于父元素的百分比%，实际工作中最常用的是像素值。

大多数浏览器，如Firefox、IE6及以上版本都采用了W3C规范，符合CSS规范的盒子模型的总宽度和总高度的计算原则是：

**盒子的总宽度= width+左右内边距之和+左右边框宽度之和**

**盒子的总高度= height+上下内边距之和+上下边框宽度之和**

注意：

1、宽度属性width和高度属性height仅适用于块级元素，对行内元素无效（ img 标签和 input除外）。

2、计算盒子模型的总高度时，还应考虑上下两个盒子垂直外边距合并的情况。

## 浮动与定位

### 浮动

选择器{float:属性值;}

|  |  |
| --- | --- |
| **属性值** | **描述** |
| left | 元素向左浮动 |
| right | 元素向右浮动 |
| none | 元素不浮动（默认值） |

### 清除浮动

清除浮动主要为了解决父级元素因为子级浮动引起内部高度为0 的问题。

其实本质叫做闭合浮动更好一些, 记住，清除浮动就是把浮动的盒子圈到里面，让父盒子闭合出口和入口不让他们出来影响其他元素。

在CSS中，clear属性用于清除浮动，其基本语法格式如下：

选择器{clear:属性值;}

|  |  |
| --- | --- |
| **属性值** | **描述** |
| left | 不允许左侧有浮动元素（清除左侧浮动的影响） |
| right | 不允许右侧有浮动元素（清除右侧浮动的影响） |
| both | 同时清除左右两侧浮动的影响 |

#### 额外标签法

是W3C推荐的做法是通过在浮动元素末尾添加一个空的标签例如 <div style=”clear:both”></div>，或则其他标签br等亦可。

**优点：** 通俗易懂，书写方便

**缺点：** 添加许多无意义的标签，结构化较差。 我只能说，w3c你推荐的方法我不接受，你不值得拥有。。。

#### 给父级添加overflow属性

可以通过触发BFC的方式，可以实现清除浮动效果。（BFC后面讲解）

可以给父级添加： overflow为 hidden|auto|scroll 都可以实现。

**优点：** 代码简洁

**缺点：** 内容增多时候容易造成不会自动换行导致内容被隐藏掉，无法显示需要溢出的元素。

#### 使用after伪元素

使用方法：

.clearfix:after { content: "."; display: block; height: 0; clear: both; visibility: hidden; }

.clearfix {\*zoom: 1;} ie6、7 专有

**优点**： 符合闭合浮动思想 结构语义化正确

**缺点**： 由于IE6-7不支持:after，使用 zoom:1触发 hasLayout。

注意： content:"." 里面尽量跟一个小点，或者其他，尽量不要为空，否则再firefox 7.0前的版本会有生成空格。

#### 给伪元素添加空白字符

content:"\200B";这个参数，Unicode字符里有一个“零宽度空格”，即 U+200B，代替原来的“.”，可以缩减代码量。而且不再使用visibility:hidden。

.clearfix:after {content:"\200B"; display:block; height:0; clear:both; }

.clearfix { \*zoom:1; }.

当然有些网站也用 content:"\0200" 的，都是空格的意思。

#### 使用before和after双伪元素

使用方法：

.clearfix:before,.clearfix:after{

content:".";

display:table;

}

.clearfix:after{

clear:both;

}

.clearfix{\*zoom:1;}

优点： 代码更简洁

缺点： 由于IE6-7不支持:after，使用 zoom:1触发 hasLayout。

### 定位

元素的定位属性主要包括定位模式和边偏移两部分。

#### 1、边偏移

在CSS中，通过边偏移属性top、bottom、left或right，来精确定义定位元素的位置，其取值为不同单位的数值或百分比。具体解释如下表所示：

|  |  |
| --- | --- |
| **边偏移属性** | **描述** |
| top | 顶端偏移量，定义元素相对于其父元素上边线的距离 |
| bottom | 底部偏移量，定义元素相对于其父元素下边线的距离 |
| left | 左侧偏移量，定义元素相对于其父元素左边线的距离 |
| right | 右侧偏移量，定义元素相对于其父元素右边线的距离 |

也就说，以后定位要和这边偏移搭配使用了， 比如 top: 100px; left: 30px; 等等

#### 2、定位模式

在CSS中，position属性用于定义元素的定位模式，其基本语法格式如下：

选择器{position:属性值;}

position属性的常用值

|  |  |
| --- | --- |
| **值** | **描述** |
| static | 自动定位（默认定位方式） |
| relative | 相对定位，相对于其原文档流的位置进行定位（不脱标） |
| absolute | 绝对定位，相对于其上一个已经定位的父元素进行定位（脱标） |
| fixed | 固定定位，相对于浏览器窗口进行定位（脱标） |

**总结**

|  |  |  |  |
| --- | --- | --- | --- |
| **定位模式** | **是否脱标占有位置** | **是否可以使用边偏移** | **移动位置基准** |
| 静态static | 不脱标，正常模式 | 不可以 | 正常模式 |
| 相对定位relative | 不脱标，占有位置 | 可以 | 相对自身位置移动 |
| 绝对定位absolute | 完全脱标，不占有位置 | 可以 | 相对于定位父级移动位置 |
| 固定定位fixed | 完全脱标，不占有位置 | 可以 | 相对于浏览器移动位置 |

#### 3、叠放次序z-index

z-index的默认属性值是0，取值越大，定位元素在层叠元素中越居上。

如果取值相同，则根据书写顺序，后来居上。

后面数字一定不能加单位。

只有相对定位，绝对定位，固定定位有此属性，其余标准流，浮动，静态定位都无此属性，亦不可指定此属性。

#### 4、display显示

display 设置或检索对象是否及如何显示。

display : none 隐藏对象 与它相反的是 display:block 除了转换为块级元素之外，同时还有显示元素的意思。

特点： 隐藏之后，不再保留位置。

#### 5、visibility可见性

设置或检索是否显示对象。

visible : 　对象可视

hidden : 　对象隐藏

特点： 隐藏之后，继续保留原有位置。（停职留薪）

#### 6、overflow溢出

检索或设置当对象的内容超过其指定高度及宽度时如何管理内容。

visible : 　不剪切内容也不添加滚动条。

auto : 　 超出自动显示滚动条，不超出不显示滚动条

hidden : 　不显示超过对象尺寸的内容，超出的部分隐藏掉

scroll : 　不管超出内容否，总是显示滚动条

## 界面样式及精灵图

### 1、界面样式

#### cursor鼠标样式

设置或检索在对象上移动的鼠标指针采用何种系统预定义的光标形状。

cursor : default 小白 | pointer 小手 | move 移动 | text 文本

尽量不要用hand 因为 火狐不支持 pointer ie6以上都支持的尽量用

是绘制于元素周围的一条线，位于边框边缘的外围，可起到突出元素的作用。

#### outline轮廓

outline : outline-color ||outline-style || outline-width

但是我们都不关心可以设置多少，我们平时都是去掉的。

****最直接的写法是 ： outline: 0;****

#### ****resize防止拖拽****

resize：none 这个单词可以防止 火狐 谷歌等浏览器随意的拖动 文本域。

#### vertical-align垂直居中

****vertical-align 不影响块级元素中的内容对齐，它只针对于 行内元素或者行内块元素，特别是行内块元素， 通常用来控制图片和表单等。****

|  |  |
| --- | --- |
| 模式 | 单词 |
| 基线对齐：  默认的是文字和图片基线对齐 | vertical-align: baseline; |
| 垂直居中：  默认的是文字和图片基线对齐 | vertical-align: middle; |
| 顶部对齐：  默认的是文字和图片基线对齐 | vertical-align: top; |

### 2、精灵图

CSS 精灵其实是将网页中的一些背景图像整合到一张大图中（精灵图），然而，各个网页元素通常只需要精灵图中不同位置的某个小图，要想精确定位到精灵图中的某个小图，就需要使用CSS的background-image、background-repeat和background-position属性进行背景定位，其中最关键的是使用background-position属性精确地定位。

## CSS文本属性及优化

### 1、margin负值

margin是可以设置为负数的

定位后元素的层级要高于标准流的层级

### 2、给网站添加favicon图标

<link rel="shortcut icon" href=“favicon.ico">

### 3、logo优化

1、我们要在logo里面放一个h1标签，就是为提权，说明这个盒子很重要。

2、我们继续在h1 里面放一个链接a 要和 logo盒子一样大。

3、注意 a链接 里面要放上网站标题。（是为了优化）

4、链接里面的文字对用户体验不好，这里不需要看见。 有两种方法实现：

5、利用text-indent:-2000em; 或者 利用padding 挤开盒子 并且overflow 切割

6、给链接添加 title 提示文本 增强用户体验。

### 4、文本属性

#### letter-spacing字间距

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。

#### word-spacing单词间距

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。

#### word-break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。

#### white-space空白符处理

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。(重要)

#### text-overflow文字溢出

text-overflow : clip | ellipsis

设置或检索是否使用一个省略标记（...）标示对象内文本的溢出

clip : 　不显示省略标记（...），而是简单的裁切

ellipsis : 　当对象内文本溢出时显示省略标记（...）

#### 文本超出以省略号代替

white-space: nowrap;（强制一行显示）

overflow: hidden;（溢出隐藏）

text-overflow: ellipsis;（溢出显示省略号）

### 5、css条件注释语法的使用

<!--[if IE]> <p>你在非IE中将看不到我的身影</p> <![endif]-->

<!--[if IE 7]>只能被 IE7 识别;<![endif]-->

<!--[if lte ie 版本号]>小于等于ie版本号<![endif]-->

<!--[if ! ie]><!-->要判断非ie<!--<![endif]-->

# JS

## 一、流程控制语句

**程序三种基本结构--**顺序结构、选择结构、循环结构

### 1、选择结构

**if语句**

if (判断条件) {

执行语句

}else if (判断条件) {

执行语句

}else{

执行语句

}

**switch语句**

case里必须是具体的值，但是如果表达式可以返回一个具体的值true或者false也可以是表达式，这时switch里面是true

switch(true) {

case 表达式：

执行语句

break；

case 表达式：

执行语句

break；

default：

执行语句

}

switch(表达式) {

case 取值：

执行语句

break；

case 取值：

执行语句

break；

default：

执行语句

}

三元表达式：判断条件？条件为真的结果：条件为假的结果；

### 2、循环结构

**for语句:1初始化表达式、2控制表达式、3循环后表达式**

for(var i = 0; i < n ; i ++){

执行语句

}

**while语句**

var i= 0；

which (判断条件){

执行语句

}

**do while 语句 先执行后判断至少执行一次**

var i = 0;

do {

执行语句

}while(判断条件)

## 二、基本类型和复杂类型

基本数据类型（值类型）：直接存储值

number、string、boolean、undefined、null（基本类型只有这五个）

复杂数据类型（引用类型）：存储引用

### 一、数组

1、数组的声明

①通过构造函数

var arr = new Array ();

②通过字面量

var arr = [];

2、数组的赋值和取值

赋值：arr [索引号] = 1；

批量赋值 for循环

取值：arr [索引号]；

3、数组的遍历（重点）

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

防止稀疏数组 arr (arr.length) = i;

4、转换数组

|  |  |  |
| --- | --- | --- |
|  | **console.log(arr.);打印输出** |  |
| console.log(arr); | [1, 2, 3] | 先以对象的形式输出，刷新后把直接量输出，其实就是调用了valueOf |
| **valueOf()** | [1, 2, 3] | 某种程度上说 它就是调用了toString |
| **toString()** | 1,2,3 | 调用了join，把每一项的值取出来用逗号拼接 |
| **join("-")** | 1-2-3 | 不传参数，默认用逗号拼接1,2,3，传入参数，会用这个参数拼接 |

5、检测数组（掌握）

|  |  |
| --- | --- |
| var arr = [1, 2, 3]; | **console.log打印输出的返回值** |
| **typeOf arr;** | **object** |
| **arr instanceof Array;** | **true** |
| **{} instanceof Array** | **false** |
| **Array.isArray([]);** | **true** |
| **Array.inArray({});** | **false** |
| **Array.prototype.isArray;** |  |
| **Array.isArray;** |  |
| **Ovject.prototype.toString.cell(arr);** | **[object Array]** |

6、增删方法

|  |  |
| --- | --- |
| **push(1,2,3,4);** | 从后面添加元素，可以同时加入多个 |
| console.log(arr.push(1,2,3,4)); | 返回值是新数组的长度 |
| **unshift(a);** | 从前面添加 可以添加多个 |
| console.log(arr.unshift(a)); | 返回值是新数组的长度 |
| **shift();** 无参数 | 从前面删除一个 |
| console.log(arr.shift()); | 返回删除的元素 |
| **pop()**; 无参数 | 从后面删除最后一个 |
| console.log(arr.pop()); | 返回删除的元素 |

**找到数组中第一个a的位置：arr. indexOf(a);**

**找到数组中最后一个a的位置arr. lastindexOf(a);**

找到数组中所有的a的位置

function getIndex (arr,str) {

var index = -1;

do {

index = arr.indexOf(str, index + 1);

console.log(index);

if (index ! = -1) {

console.log(index);

}

}while (index ! = -1);

}

在数组中元素出现的次数

var arr = ["c", "a", "z", "a", "x", "a"]  
 var o = {}; //属性名 元素--> 属性值 次数

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

var item = arr[i];

if (o[item]) { /能进来说明有值，没有进来说明之前没有出现过

o[item] ++; //等于o[item] = o[item] + 1;

}else {

o[item] = 1; //能进来说明是undefined，也就是说之前没有出现过，这是第一次出现

}

}

7、操作方法

concat();把参数拼接到当前数组

slice[begin，end);从当前数组中把要的东西复制一份，不影响原来的数组

splice(begin，删除个数，插入的新元素);删除或替换当前数组的某些元素，返回这些元素

8、位置方法

indexOf(); 从前往后寻找元素在数组中的位置

lastIndexOf(); 从后往前找，没有找到返回-1

9、迭代方法

every();

filter();

forEach(); 对数组遍历

for (var i = 0; i < arr.length; i ++){  
 console.log(arr[i]);}

arr.forEach(function (element, index, array) {

console.log(element);

})

map();

some();

**把原数组的值全部复制给新数组，在进行各项操作之后不会对原素组产生影响，原数组不变**

function sort(arr, fn) {

var newArr = [];

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

newArr[newArr.length] = arr[i];

}

for (var i = 0; i < newArr.length - 1; i ++) {

var flag = true;

for (var j = 0; j < newArr.length - 1 - i; j ++){

if (fn(newArr[j] > newArr[j + 1]) > 0){

var temp = newArr[j];

newArr[j] = newArr[j+1];

newArr[j+1] = temp;

flag = false;

}

}

if (flag) {

break;

}

}

return newArr;

}

var array = [3, 2, 1];

console.log(arr);

console.log(sort(array, function (a, b){

return a - b;

}));

console.log(arr);

10、sort 方法（了解）

原理：

比较的趟数

function sort(arr, fn) {

for (var i = 0; i < arr.length - 1; i ++) {

var flag = true;

两两比较的次数

for (var j = 0; j < arr.length - 1 - i; j ++){

if (fn(arr[j] > arr[j + 1]) > 0){

var temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

flag = false;

}

}

if (flag) {

break;

}

}

return arr;

}

var array = [3, 2, 1];

原数组[3,2,1]

原数组[3,2,1]

console.log(arr);

console.log(sort(array, function (a, b){

return a - b;

sort方法返回的[1,2,3]

sort方法返回的[1,2,3]

}));

console.log(arr);

sort方法执行后的原数组没有发生改变[3,2,1]

a - b从小到大排列

b - a从大到小排列

a.length - b.length 字符串从少到多排列

a.age - b.age 年龄从小到大排列

sort方法执行后的原数组发生了改变[1,2,3]

### 二、函数

1、函数的定义和调用

函数的定义

①函数的声明：function 函数名 () {

函数体（要执行的代码）

}

②函数表达式：var 函数名 = function () {

函数体

};

函数的调用：函数名 ();（要调用先引用）

2、函数的参数

有参数的函数的声明：function 函数名 (参1，参2) { //函数声明时的参数只是一个占位符，是形式参数

函数体

}

有参数的函数的调用：函数名 (参1，参2) //函数调用时传入的参数是具体的数值，是实际参数

形参和实参没有关系，只不过在调用的时候把实参的值复制了一份赋值给了形参

3、函数返回值

①return 要返回的值；（函数的返回值是什么，调用这个函数就相当于调用了什么）

②函数在执行完成return语句后便会退出函数，后面的代码不会执行

③如果只打印函数名，会通过函数名找到函数体，如果想打印的是函数的返回值，那么一定要加小括号()

4、匿名函数

匿名函数：函数定义完后就赋值给了一个变量，通过变量名来调用这个函数，所以就没有必要起名字

5、递归函数

函数自己调用自己的编程技巧

6、回调函数(重点)

被当做函数传递的函数

### 三、对象

1、键值对

一种对应关系，通过键能够方便地找到值 key:value; 对象也是键值对

属性名：属性值、索引：元素的值

2、对象的声明

①通过构造函数

var obj = new.Object();

②通过字面量

var obj = {};

3、属性

用来描述对象的特征 一般是名词 对应变量

定义：对象.属性名 = 要赋的值；

调用：对象.属性名；

4、方法

用来描述对象的行为 一般是动词 是一种函数（属于某个对象的函数就叫方法）

定义：对象.方法名 = function(){ //函数体 }；

调用：对象.方法名；

5、访问属性的两种方式（重点）

①点语法（属性名是什么就必须写什么）：obj.属性名；

②中括号（通过字符串找属性名，字符串可以通过编程拼接）：obj["属性名"];

6、遍历的两种方式（重点）

①for循环

②for-in

for(var k in json) { 语句 }

k变量代表的是json中的各个属性（key）和 var i = 0中的i是一个意思 名字不同而已

json JavaScript Object Notation（JavaScript对象标记法）是仿照JS中对象字面量的格式去书写的一串 用来记录对象数据的**字符串**，可以用于数据传输。

7、Null

null常被用于期望一个对象，但是当前先不附加任何值的情况

undefined派生自null

undefined == null //true

undefined === null //false

### 四、字符串的方法

字符串的所有方法都不会修改字符串本身

1、操作方法（重点）

slice(参1，参2) 从start开始，截取到end位置，end取不到；参1：开始的位置 ；参2：结束的位置[start,end).遇到负数把length和负数相加

substring() 从start开始，截取到end位置，end取不到；遇到负数，把负数变为0，两个参数交换位置

substr() 从参1，start开始，截取参2，length个字符

charAt() 获取指定位置处的字符

str[0]; 和charAt 等效，IE8+支持

trim() 只能去除字符串前后的空白

split() 把字符串切割成字符数组

大小写转换：

to(Locale)UpperCase() 转换大写

to(Locale)LowerCase() 转换小写

2、查找字符串中所有的o 的位置

var str= "dmsosekkdsoosns";

varindex= -1；

do{

index = str.indexOf("o",index + 1);

console.log(index);

}while(index != -1);

3、将字符串中所有的o替换成！

do{

str = str.**replace**("o","!");

}while(str.indexOf("o")!= -1);

console.log(str);

4、统计字符串中每个字符出现的次数

var o = {};

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

var item = str.charAt(i);

if (o[item]) {

o[item] ++; //如果以前有了值，让这个值加一

}else {

o[item] = 1; //如果之前没有，现在进来就有了一次了

}

}

for (var key in o) {

console.log(key+"出现了" + o[key] + "次");

}

5、去掉字符串中所有的空格

var str = " aaa di djsoa jdi wd ";

str = str.trim(); //trim可以去除两边的空格

var arr = str.split(" ");//按照空格分隔

str = arr.join(" ");

6、replaceClassName: 替换类名，封装函数（了解）

function replaceClassName(element, oldStr, newStr) {

//把类名这个字符串按照空格分隔，把里面每一个类名做判断

var arr = element.className.split(" ");

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

if (arr[i] === oldStr) {

arr[i] = newStr;

}

}

element.className = arr.join (" ");

}

7、getElementsByClassName

function getElementsByClassName(element,className) {

if(element.getElementsByClassName) {

return element.getElementsByClassName(className);

}else{

var filterArr = [];

var elements = element.getElementsByTagName("\*");

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

var nameArr = elements[i].className.split(" ");

for (var j =0; j < nameArr.length; j ++) {

if (nameArr[j] === className) {

filterArr.push(elements[i]);

break;

}

}

}

return filterArr;}}

## 三、作用域、预解析和声明提升

预解析

解析器在执行当前作用域下的代码之前有一个预解析的过程

预解析的时候 会将变量和函数的声明提升到当前作用域的顶部 然后才执行

声明提升

变量提升：解析器会将变量提升到作用域的最上面，但是变量的赋值不会提升

函数提升：解析器会率先读取函数声明，并使其在执行任何代码之前可用

另外解析器会先找var再找function，因此如果变量和函数重名后找到的函数会把先找到的变量覆盖（如果代码执行后有赋值语句，该值又会把函数覆盖）

变量作用域

最外层的是全局变量

函数内的是局部变量

特殊：函数内部不加var的是全局变量

递归概念：程序调用自身的编程技巧称为递归

回调函数：函数也是一种普通的数据类型，因此函数也可以被当作参数传递，被当作参数传递的函数叫做回调函数

## 四、构造函数创建对象

@this和new

利用new 关键字可以声明新的对象。new 关键字可以让构造函数中this的指向改变，并让构造函数把this返回。

@构造函数

构造函数也是函数，只不过会默认返回一个对象。

@构造函数创建对象

通过构造函数创建对象更方便（不需要创建对象并返回）。更重要的是可以通过instanceof来判断实例的类型了。

### 工厂模式（了解）

function Student (name, age, sex, score) {

var stu = new Object();

stu.name = name;

stu.age = age;

stu.sex = sex;

stu.score = score;

stu.sayHi = function () {

console.log("大家好我叫" + this.name + "我今年" + this.age);

};

return stu;

}

var stu = new Student("zs", 18, 1, 100);

stu.sayHi();

### 构造函数模式（重点）

function Student (name, age, sex, score) {

var stu = new Object(); 这回就不需要手动的创建对象了，直接使用this

this.name = name;

this.age = age;

this.sex = sex;

this.score = score;

this.sayHi = function () {

console.log("大家好我叫" + this.name + "我今年" + this.age);

};

return stu; 也不需要返回对象了,因为new 可以让this返回

}

调用：

var stu 1= new Student("zs", 18, 1, 100);

var stu 2= new Student("zs", 18, 1, 100);

stu1.sayHi();

相同的函数保存了好几份，没有必要，通过原型属性可以解决这个问题

stu2.sayHi();

console.log(stu1.sayHi === stu2.sayHi);

通过构造函数模式创建出来的对象，可以通过instanceof 来辨别具体类型

## 五、原型属性：prototype

如果希望所有的数组对象都能调用某个方法，可以在这个对象的构造函数的原型属性上绑定方法

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

~~console.log("去吧，皮卡丘！");~~

};

var arr = new Array();

arr.jump()

通过原型属性，把公共的方法只保存一份，可以节省资源

### 原型模式：

function Student (name, age, sex, score) {

this.name = name;

this.age = age;

this.sex = sex;

this.score = score;

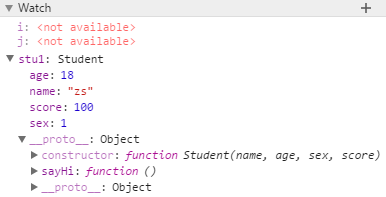
}

Student.prototype.sayHi = function () {

console.log("大家好我叫" + this.name + "我今年" + this.age);

};





调用：

var stu 1= new Student("zs", 18, 1, 100);

var stu 2= new Student("zs", 18, 1, 100);

stu1.sayHi();

stu2.sayHi();

console.log(stu1.sayHi === stu2.sayHi);

console.log(stu1);

**prototype 是构造函数的原型属性**

**\_\_proto\_\_ 是对象的原型属性,一环套一环，**

**形成了一个链条，就是原型链，通过原型链**

**可以让它现原形**

## 六、深层复制数组（了解）

function deepClone(arr) {

var newArr = [];

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

newArr[newArr.length] = arr[i];

}

return newArr;

[100, 2, 1]

浅层复制改变新数组会改变原数组

}

var arr1 = [3, 2, 1];

~~var arr2 = arr1;~~

[1, 2, 1]

[1, 2, 3]

深层复制改变新数组不会改变原数组

var arr2 = deepClone(arr1);

arr[0] = 100;

console.log(arr1);

console.log(deepClone(arr1).sort(function (a, b){

return a - b;

}));

## 七、形参和实参（重点）

**调用函数的时候 是把实际参数的值 复制一份 赋值给形参 它们只是值一样 没有任何关系**

function fn(a, b) { -->形式参数

a = 100; //对形参进行改变

//console.log(a);//形参改变了

return a + b;

}

var a = 10;

var b = 20;

fn(a, b);//实际参数

//console.log(a);//实参不会改变

对形参的改变一定不会影响实参 这个说法是不严谨的

上个例子中的sort（arr,fn）

var array = [3, 2, 1];//实际参数

console.log(array);//把实际参数array复制了一份 赋值给了形式参数arr 我们是对形式参数arr进行的修改

console.log(sort(array, function (a, b) {

return a - b;

}));//这个函数内部只是对形式参数arr进行了修改

console.log(array); 实际参数也改变了

## JS面试题

**1、谈谈你对递归的认识？**

递归：程序调用自身的编程技巧称为递归，自己调用自己。

function factorial(num) {

return ( num <= 1 ) ? 1 ： num \* factorial(num - 1)；

}

console.log(factorial(8))；

**2、简述forin 循环的特点及使用场景？**

for...in 语句用于对数组或者对象的属性进行循环操作。 for ... in 循环中的代码每执行一次，就会对数组的元素或者对象的属性进行一次操作。 注意：forin循环不会按照属性的下标来排列输出。

for (变量 in 对象){

在此执行代码

}

“变量”用来指定变量，指定的变量可以是数组元素，也可以是对象的属性。

**3、split()join()的区别？**

split() 方法通过把字符串分割成子字符串来把一个 String 对象分割成一个字符串数组。

语法 str.split([separator][， limit])

参数 separator 指定用来分割字符串的字符（串）。separator 可以是一个字符串或正则表达式。 如果忽略 separator，则返回整个字符串的数组形式。如果 separator 是一个空字符串，则 str 将会把原字符串中每个字符的 数组形式返回。 参数limit是一个整数，限定返回的分割片段数量。split 方法仍然分割每一个匹配的 separator，但是返回的数组 只会截取最多 limit 个元素。

"|a|b|c".split("|") //将返回[""， "a"， "b"， "c"]

"hello".split("") //可返回 ["h"， "e"， "l"， "l"， "o"]

"hello".split(""， 3) //可返回 ["h"， "e"， "l"]

join() 方法将数组中的所有元素连接成一个字符串。

语法 str = arr.join([separator = '，'])

参数separator 可选，用于指定连接每个数组元素的分隔符。分隔符会被转成字符串类型；如果省略的话，默认为 一个逗号。如果 seprator 是一个空字符串，那么数组中的所有元素将被直接连接。

例子： var a = ['Wind'， 'Rain'， 'Fire']；

var myVar1 = a.join()； // myVar1的值变为"Wind，Rain，Fire"

var myVar3 = a.join(' + ')； // myVar3的值变为"Wind + Rain + Fire"

var myVar4 = a.join('')； // myVar4的值变为"WindRainFire"

**4、如何消除一个数组里面重复的元素？**

var arr1 =[1,2,2,2,3,3,3,4,5,6],

arr2 = []; for(var i = 0,len = arr1.length; i< len; i++){

if(arr2.indexOf(arr1[i]) < 0){

arr2.push(arr1[i]);

}

}

document.write(arr2); // 1,2,3,4,5,6

**5、请说出==和===的区别？**

== 判断内容是否相等 不比较类型 console.log(1 == "1")； true

=== 判断内容相等 且类型也相等 console.log(1 ==="1")； false

**6、请列举字符串操作的方法？**

charCodeAt() 方法返回一个整数，代表指定位置字符的Unicode编码；

charAt() 方法返回指定索引位置处的字符。如果超出有效范围的索引值返回空字符串；

slice() 方法返回字符串的片段；

substring() 方法返回位于String对象中指定位置的子字符串。

substr() 方法返回一个从指定位置开始的指定长度的子字符串。

indexOf() 方法返回String对象内第一次出现子字符串位置。如果没有找到子字符串，则返回-1；

lastIndexOf() 方法返回String对象中字符串最后出现的位置。如果没有匹配到子字符串，则返回-1；

search() 方法返回与正则表达式查找内容匹配的第一个字符串的位置。

concat() 方法返回字符串值，该值包含了两个或多个提供的字符串的连接；

split() 将一个字符串分割为子字符串，然后将结果作为字符串数组返回；

# DOM

## 一、事件

### 1、阻止a标签的默认行为（重点）

return false;

### 事件三要素

事件源.事件 = function（）{事件处理程序}；

### 3、批量操作

①批量获取元素（通过标签名） getElementsByTagName ）

②批量添加事件 for循环遍历

replace 方法

replace (要找的值，要替换的值)

function 函数名（element, 要找的值，要替换的值）{

element.className = element.className.replace(要找的值，要替换的值)

}

Document Object Model 文档对象模型，就是把HTML文档模型化，当作对象来处理

## 二、DOM结构

### 1、概念

文档(Document)：就是指HTML或者XML文件

节点(Node)：HTML文档中的所有内容都可以称之为节点，常见的节点有

元素节点 属性节点 文本节点 注释节点

元素(Element)：HTML文档中的标签可以称为元素

### 2、结构概念

父节点 当前节点的父级

子节点 当前节点的子级

兄弟节点 和当前节点同属一个父级的节点

### 3、.获取元素

getElementById

getElementsByTagName 通过标签名寻找一类元素（伪数组）

### 4、设置属性

元素对象.属性名 = “属性值”;

<标签 属性名=”属性值”>

### 5、绑定事件

事件三要素

事件源.事件 = function(){ 事件处理程序 };

对象方法中的this永远指的是该方法所属的那个对象

### 6、常用属性（重点）

a常用标签属性

DOM对象的属性和HTML的标签属性几乎是一致的，常用的有src、title、className、href和style

b内部文本属性

innerHTML 获取和设置标签中的内容，设置的内容会当作节点对象被解析到DOM树上

innerText 获取和设置标签中的内容，设置的内容会被当作普通文本（有兼容性问题，旧版ff不支持）

textContent获取和设置标签中的内容，设置的内容会被当作普通文本（有兼容性问题，旧版IE不支持）

c常用表单属性

常见的表单元素属性有： type、value、checked、selected、disabled

type设置input元素的类型

value设置input元素的值

checked设置input元素是否选中

selected 设置下拉列表select中的option是否被选中

disabled 设置input元素是否被禁用

获取焦点事件是onfocus 失去焦点事件是onblur

d自定义属性--必须以data-开头

getAttribute() 获取标签属性

setAttribute() 设置标签属性

removeAttribute() 移除标签属性

### 7、节点（重点）

childNodes //子节点

children //子元素

nextSibling //下一个兄弟节点

nextElementSibling //下一个兄弟元素 有兼容性问题

previousSibling//上一个兄弟节点

previousElementSibling //上一个兄弟元素 有兼容性问题

firstChild //第一个节点

firstElementChild //第一个子元素 有兼容性问题

lastChild //最后一个子节点

lastElementChild //最后一个子元素 有兼容性问题

parentNode //父节点 （一定是元素节点，所以无需处理）

### 8、样式属性

style属性是对象，style对象的属性是字符串

style只能获取和设置行内样式

JS中 - 不能作为标识符，涉及到计算的时候必须直接通过JS加样式

### 9、动态创建元素（掌握）

①插入和移除节点

在父元素中的最后追加子元素：father.appendChild(要追加的元素);

在父元素中的某个子元素前面插入子元素：father.insertBefore(要插入的元素,插到这个元素的前面);

从父元素中移除子元素：father.removeChild(要移除的子元素);

②js动态创建结构

方式一：直接在文档中书写 document.write(“内容”)

方式二：改变元素对象内部的HTML：innerHTML=”内容”

**使用 字符串** 或者 **数组** 先拼接再用innerHTML去添加到页面

方式三：创建或克隆节点并追加，参数是一个布尔值 true表示深层复制 false是浅层复制

document.createElement()

document.cloneNode()

## 三、BOM

### **1、window.onload**

加载完成事件

onload 加载完成，指的是页面所有资源都加载完成

涉及到图片内容的一定要写在onload里面

### **2、addLoadEvent**

给同一个对象注册相同的事件会相互抵消，给window追加onload事件不会顶掉之前的函数

function addLoadEvent（fn） {

var oldOnload = window.onload;

//检查现在的window.onload是否绑定了事件

if (typeof oldOnLoad === "function") {//说明之前绑定了事件

window.onload = function () {

oldOnLoad();//之前的要执行

fn ();//传入的新的要绑定的将来也要执行

};

}else {

等同于

window.onload = function () {

fn();

}

window.onload = fn();

}

}

### 3、open方法

①url

②windowName窗口名

\_self 当前窗口

\_black 新窗口

③窗口的一些属性（长、宽、背景）

### 4、close 方法

window.close 关闭当前窗口

win.close 关闭调用它的窗口

### setTimeout

延时定时器

setTimeout（参1，参2）参1：回调函数，要执行的代码，参2：延迟的时间,毫秒

### 6、clearTimeout

（要清理的定时器） 清理定时器

clearTimeout（）

var timerId = null;

var btn1 = document.getElementById("btn1");

var btn2 = document.getElementById("btn2");

btn1.onclick = function () {

timerId = setTimeout(function () {

console.log("嘣")

}，3000);

};

btn2.onclick = function () {

clearTimeout(timerId);

};

### 7、setInterval 间歇定时器（重点）

setInterval（参1，参2）参1：回调函数，要执行的代码，参2：延迟的时间,毫秒

点击按钮，让按钮中的文字倒计时

var num = 5;

var timer = null;

btn.onclick = function () {

this.disabled = true;//1、禁用按钮

timer = setInterval(function () {//2、设置按钮的值

num--;

btn.value = num + "秒后可再次发送";//一定注意此时不能用this，定时器的回调函数是被window调用的

if (num === 0) {

clearInterval(timer);

btn.value = "点击发送验证码"；

btn.disable = false;//此时不需要再禁用了

num = 5;

}

},1000);

};

clearInterval(要清理的定时器) 清理间歇定时器

### 8、window的location对象（重点后期获取数据经常用到）

location.href = "要跳转的地址";

location.reload();让页面重新加载

location.hash; 锚点

location.host; 服务器

location.hostname; 服务器名

location.pathname

### 9、navigator对象（了解）

navigator.userAgent

### 10、history 对象

history.forward(); 前进

history.back(); 后退

### 11、screen 对象，屏幕对象（重点）

screen.width; 屏幕的宽度

screen.height; 屏幕的高度

screen.availWidth; 可用宽度

screen.availHeight; 可用高度

### 12、date日期对象（掌握）

date.toString(); Sat Jul 23 2016 15:47:38 GMT+0800 （中国标准时间）

date.valueOf();这是一个13位的数字，从1970-1-1 00:00:00:000 开始到现在的毫秒值，方便运算

var date = Date.now(); 返回数值，13位的数字

var date = +new Date (); 加了+号转换成13位数字，不加就是字符串 同date.toString或console.log（date）;

可以接受三种参数：2003,10,1 日期的每一部分

"2003-10-1" 字符串的日期格式

表示日期的毫秒形式 1128094003232

var date = Date.parse("2003-10-1"); 转化成毫秒形式，格式不对返回NaN

toDateString(); Sat Jul 23 2016

toTimrString(); 15:47:38 GMT+0800

toLocaleDateString(); 2016/7/23

toLocaleTimeString(); 下午3:52:11

getTine();返回毫秒数 getMilliseconds(); 当前的毫秒

getSecond(); 秒

getMinutes(); 分

getHours(); 时

getDay(); 返回周几0-6

getDate(); 返回当前月的第几天

getMonth(); 返回0-11

getFullYear(); 返回4位的年份 2016

## DOM面试题

**1、DOM操作——怎样添加、移除、移动、复制、创建和查找节点?**

（1）创建新节点

createDocumentFragment() //创建一个DOM片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

（2）添加、移除、替换、插入、复制

appendChild() //添加

removeChild() //移除

replaceChild() //替换

insertBefore() //在已有的子节点前插入一个新的子节点

cloneNode() //复制

（3）查找

getElementsByTagName() //通过标签名称

getElementsByName()//通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包括id等于 name值的)

getElementById() //通过元素Id，唯一性

**2、列举浏览器对象模型BOM里常用的至少4个对象，并列举window对象的常用方法至少5个？**

对象：Window、 document、 location、 screen、 history、 navigator 、data

方法：Alert()、 confirm() 、prompt()、 open() 、close()

**3、document.ready和onload有什么区别？**

document.ready和onload的区别——JavaScript文档加载完成事件。页面加载完成有两种事件:

一是ready，表示文档结构已经加载完成（不包含图片等非文字媒体文件）

二是onload，指示页面包含图片等文件在内的所有元素都加载完成。

jQuery中$(function(){/\* do something\*/});他的作用或者意义就是:在DOM加载完成后就可以可以对DOM进行操作。一般情况先一个页面响应加载的顺序是，域名解析-加载html-加载js和css-加载图片等其他信息。

**4、****当一个DOM节点被点击时候，我们希望能够执行一个函数，应该怎么做？**

直接在DOM里绑定事件：<div onclick=”test()”></div>

在JS里通过onclick绑定：xxx.onclick = test

通过事件添加进行绑定：addEventListener(xxx, ‘click’, test)

**5、window.location.search() 返回的是什么？**

    答：查询(参数)部分。除了给动态语言赋值以外，我们同样可以给静态页面,并使用javascript来获得相信应的参数值

        返回值：?ver=1.0&id=timlq 也就是问号后面的！

**6、简述列举文档对象模型DOM里document的常用的查找访问节点的方法做简单说明？**

document.getElementById 根据元素id查找元素

document.getElementByName 根据元素name查找元素

document.getElementTagName 根据指定的元素名查找元素

# 网页特效

## 一、offset系列属性

### 1、offset -- 偏移量（重点）

相对于原来的位置移动了多少，用于获取元素真实的自身的位置和大小，是数值可以直接计算，是只读属性，无法设置

offsetWidth

offsetHeight = height + padding + border；

style 可以获取或者设置行内样式，是字符串而且有单位

offsetParent 距离自身最近的带有定位的父级，

如果父级没有定位继续往前找，直到找到body

parentNode 父节点，只和结构有关

offsetLeft 从自身的border的左侧到父级的padding左侧

### 2、Math对象

Math.abs();取绝对值

Math.round();就近取整 Math.round(-1.5)--> -1;

### 3、动画

原理：每次让对象向前移动一点，连起来形成动画，一秒25帧就有动画效果了

公式：对象当前位置 = 对象当前位置 + 每个时间段要移动的长度

leader = leader + step；

注意：

判断运动方向

一定要加定位

完善终点检测

终点清除定时器

手动设置对象位置到终点

调用开始先清理定时器，防止多次调用

动画函数的封装：

function animate(obj, target) {  
 clearInterval(obj.timer);

if (leader < target) {

step = step;

}else {

step = - step;

}

obj.timer = setInterval(function () {

var leader = obj.offsetLeft;

var step = 10;

step = leader < target ? step : -step;

if (Math.abs(target - leader) >= Math.abs(step)) {

leader = leader + step;

obj.style.left = leader + "px";

}else {

obj.style.left = target + "px";

clearInterval(obj.timer);

}}, 15);}

动画轮播图具体的思路：

1、要做事先找人

把所有能用到的标签全部获取到

2、动态生成结构：按钮li

2.1 根据ul 中的li 动态生成ol中的li，ul中有几个li 生成ol中几个li

2.2设置li中的文本，把li追加到ol中

2.3获取ol中的li

2.4根据第一张图，动态克隆，并追加到ul最后

3、鼠标经过按钮，移动ul到指定的位置

3.1鼠标经过按钮，按钮排他（干掉所有人，留下我自己）

3.2经过按钮，ul移动到指定位置

指定位置：-this.index \* imgWidth和当前按钮的索引以及图片的宽度有关，向左移动是负数

①设置索引②取图片的宽度③调用动画函数

4、鼠标点击箭头

4.1 显示出箭头

4.2 点击右箭头，把ul移动到指定位置

①鼠标点击事件

②ul移动到指定位置：-pic \* imgWidth 和图片索引和图片宽度有关

③判断图片移动到最后一张后（假的第一张），瞬间调到开头，然后做从第一张到第二张的动画

ul.style.left = 0; //瞬间移动到开头

pic = 0;//索引后归零，后续让ul从第一张渐渐移动到第二张

4.3点击左箭头

当图片移动到第一张后，瞬间跳到最后，然后做从最后一张到倒数第二张的动画

5、按钮也要跟着左右箭头移动

根据当前图片的索引，计算出下一个应该亮起的按钮的索引

干掉所有人、、留下相应的

6、添加自动滚动

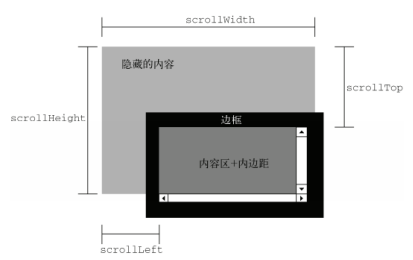
每隔一秒播放下一张（让点击右箭头的事件每隔一秒执行一次）

鼠标放在图片上自动滚动停止，离开后继续自动滚动

鼠标经过图片清除自动滚动，离开时继续自动滚动

在图片自动滚动时，经过按钮放开后，图片会接着自动滚动时的顺序继续滚动，而不是从经过按钮的图片开始滚动：把记录显示按钮的索引变为当前按钮的索引

把记录显示图片的索引变为当前图片的索引



## 二、scroll系列属性（重点）

offset 元素自身的大小

scroll 元素内部内容的大小

scrollTop 顶部被卷去的内容的高度

scrollLeft 左侧被卷去的内容的宽度

onscroll 滚动事件

### 1、页面滚动坐标（重点）

var scrollTop = window.pageYoffset || document.documentElement.scrollTop || document.body.scrollTop || 0

document.body 获取的是body标签，是对象

document.head 获取的是head 标签，是对象

document.title 获取的是title 标签中的内容，是字符串

document 没有html 这个属性，html 标签是documentElement

### 2、封装自己的scroll

封装一个函数，调用这个函数 .top 就可以获取被卷去的头部的高度 .left 就可以获取被卷去的左侧的宽度

function scroll() {

var scrollTop = window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop || 0;

var scrollLeft= window.pageXOffset || document.documentElement.scrollLeft || document.body.scrollLeft || 0;

var o = {};

o.top = scrollTop;

o.left = scrollLeft;

return o;

}

优化方案

function scroll() {

return {

top = window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop || 0;

left= window.pageXOffset || document.documentElement.scrollLeft || document.body.scrollLeft || 0;

};

}

### 3、缓动动画

缓动动画公式：step = (target - leader) / 10 leader = leader + step

缓动函数的封装：让任意对象移动到指定位置

function animate(obj, target){

clearInterval(obj.timer);

obj.timer = setInterval(function(){

var leader = obj.offsetLeft;

var step = (target - leader) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

leader = leader + step;

obj.style.left = leader + "px";

if (leader === target) {

clearInterval(obj.timer);

}

},15);

};

访问属性的两种方式：obj.name、obj["name"]

获取计算后样式属性：window.getComputedStyle

ie 6,7,8 :currentStyle

function getStyle (obj,attr) {

if (window.getComputedStyle) {

return window.getComputedStyle(obj,null)[attr];

} else {return obj.currentStyle[attr];}}

封装缓动框架

function animate(obj, json, fn) {

clearInterval(obj.timer);

属性名k

obj.timer = setInterval(function () {

属性值json[k]

var flag = true;

for (var k in json) { //json {attr : taget}

if (k === "opacity") { //opacity要特殊处理

//opacity没有单位 参与运算自动转换成数值 所以不用parsetInt

//取值范围 0-1 0.1 0.33 33 为了让以前的计算公式生效 要扩大100倍

var leader = getStyle(obj, k) \* 100;

var target = json[k] \* 100;

var step = (target - leader) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

leader = leader + step;

obj.style[k] = leader / 100; //opacity没有单位

} else if (k === "zIndex") {

obj.style.zIndex = json[k]; //层级不需要渐变 直接设置即可

} else {

var leader = parseInt(getStyle(obj, k)) || 0;

var target = json[k];

var step = (target - leader) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

leader = leader + step;

obj.style[k] = leader + "px";

}

if (leader != target) {

flag = false;

}

}

if (flag) {

clearInterval(obj.timer);

if (fn) {

fn();

}

}

}, 15);

}

//全部属性都到达目标值才能清空

function getStyle(obj, attr) {

if (window.getComputedStyle) {

return window.getComputedStyle(obj, null)[attr];

} else {

return obj.currentStyle[attr];

}

}

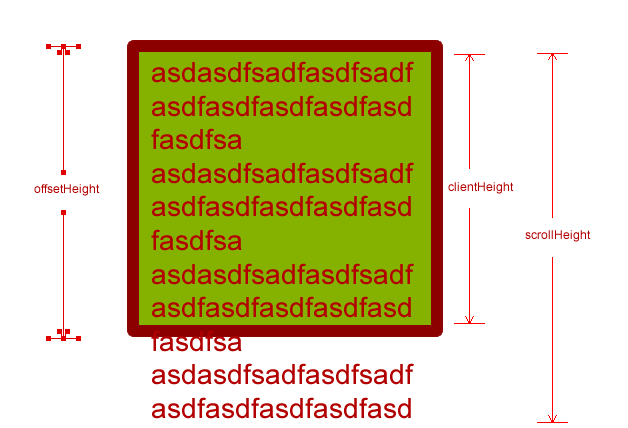
## 三、client系列

### 1、client 可视（重点）

偏移offsetWidth: width + padding + border

卷曲scrollWidth: width + padding 不包含border 内部内容的大小

可视clientWidth: width + padding 不包含border



### 2、网页可视区兼容性写法的封装（所有的浏览器）

function client() {

return {

width :window.innerWidth || document.documentElement.clientWidth || document.body.clientWidth || 0,

height :window.innerHeight || document.documentElement.clientHeight || document.body.clientHeight || 0;

}

}

window.onresize 窗体大小发生改变事件

### 响应式布局

a A zqQ ZQAZAzqqaaAq q Q qAA qqqqqqqqq

window.onresize = function () {

clearTimeout();

timer = setTimeout(responsive, 500)

};

respinsive();

function responsive() {

if(client()).width > 960){

document.body.style.backgroundColor = "red";

document.body.innerHTML = "computer";

} else if (client()).width > 640){

document.body.style.backgroundColor = "green";

.body.innerHTML = "tabletr";

}else {

document.body.style.backgroundColor = "yellow";

document.body.innerHTML = "mobile";

}}

### 4、函数节流

my 要使用animate 中的代码，要先加载animate

方法：windo.scrollTo(x轴坐标，y轴坐标) 窗体滚动

## 四、事件对象

1、实际上，事件发生的时候，是浏览器让我们注册的这个浏览器执行的，浏览器在调用这个函数的时候会传入一个参数

2、形参和实参可以不一致，如果传入了实参没有形参接收也不会出错，只是实参没有用

### 1、事件对象的兼容性写法

属性

参数

document.onclick = function (event) {

局部变量

var event = event || window.event;

};

### 2、三个重要坐标（重点）

clientX clientY 在浏览器上的位置

pageX pageY 在页面上的位置 page

page 有兼容问题，client 没有，我们可以用client 来计算page = client + screen.left

var pageX = event.pageX || event.clientX + document.documentElement.scrollLeft;

var pageY = event.pageY || event.clientY + document.documentElement.scrollTop;

screenX screenY 电脑屏幕

onmousedown 鼠标按下事件

onmouseup 鼠标弹起事件

onmousemove 鼠标移动事件

event.stopPropagation();停止传播，防止冒泡

event.cancelBubble 阻止冒泡 ie678

event.target 通过事件目标可以找到事件的源头

判断是否选中文字

var txt = window.getSelection ? window.getSelection().toString() : document.selection.createRange().text;

### 3、事件委托

参1：要委托事件的父元素，参2：事件类型，参3：事件发生在哪些标签上，参4：发生事件后要执行的代码

绑定事件的两种方式：

1、普通事件

### 4、事件监听器（重点）

addEventListener(参1，参2，参3) 参1：事件的类型，参2：事件处理函数，参3：是否使用捕获

是否使用捕获：false 不使用捕获，是冒泡的顺序，从子级到父级

true 使用捕获，从父级到子级

解绑事件的两种方式：

直接顶掉

事件监听器的

## 五、正则表达式

声明和使用

### 1、通过构造函数定义

var 变量名= new RegExp(/表达式/);

### 2、通过直接量定义(只能使用一次)

var 变量名= /表达式/; 变量名可以是 regEx

常用方法，可用于检测传入的字符串是否符合该规则并返回布尔值  
 exp.test("要检测的字符串") true、false 可以匹配数字，字母不行，只要有想要的（数字）就是true

预定义类

. [^\n\r] 除了换行和回车之外的任意字符

\d [0-9] 数字字符digit

\D [^0-9] 非数字字符

\w [a-zA-Z0-9\_] 单词字符(所有的字母数字和\_) word

\W [^a-zA-Z0-9\_] 非单词字符

\s [\f\r\n\t\v] 不可见字符 space-->空字符串、空格、换行

\S [^\f\r\n\t\v] 可见字符

转义符

\f 表示换页 form feed

\t 表示水平制表符 table

\v 表示垂直制表符 vertical table

| 表示或， /f|boot/.test("foot");这个表达式表示的意思是 f或者 boot 只不过foot里面有了f所以会返回true

1.1.3字符类

@字符类

简单类[abc] 表示该位置可以出现的字符，/[abc]/.test("a"); 返回true，可以是a或b或c

负向类[^abc] 表示该位置不可以出现的字符

范围类[a-e] 表示该位置可以出现的字符的范围

组合类[a-xA-E0-9] 范围类的组合

@验证密码强度

边界 量词 括号

@边界

^ 会匹配行或者字符串的起始位置

^只有在[]内才表示非 在外边表示开始

$ 会匹配行或字符串的结尾位置

^$在一起 表示必须是这个（精确匹配），必须是我要的那个

@量词 自定义出现的次数放在 {} 中

"\*" 重复零次或多次 x>=0

"+" 重复一次或多次 x>=1

"?" 重复零次或一次 x=(0||1)

{n} n次 x=n

{n,} 重复n次或更多 x>=n

{n,m} 重复出现的次数比n多但比m少 n<=x<=m

@括号总结

()表示一组

[]表示一个字符的位置

{}表示次数

常见项目的匹配

常见项目的匹配网上有很多（例如搜索常用正则表达式大全），无需记忆，能看懂即可

匹配国内电话号码：@验证座机

/^0\d{2,3}-\d{7,8}$/

匹配姓名：@验证姓名 /^[\u4e00-\u9fa5]{2,}$/

匹配腾讯QQ号：

/^[1-9]\d{4,10}$/

匹配手机号：

/^(13[0-9]|14[57]|15[0-9]|18[0-9])\d{8}$/

匹配邮箱：

/^\w+([+-.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$/

function check (inp, reg) {

inp.onblur = function () {

if (reg.test(this.value)){

this.nextSibling.innerHTML = "输入正确";

this.nextSibling.className = "";

} else {

this.nextSibling.innerHTML = "输入不正确";

}

};

}

### 3、封装自己的trim方法

@replace() 方法

格式：

字符串对象.replace(正则式或字符串，替换的目标字符)

返回值:

替换后的新字符串

@封装自己的trim()方法

function trim(str) {

return str.replace(/^\s+/g,"").replace(/\s+$/g,"");

}

----->>

function trim(str) {

return str.replace(/^\s+|\s+$/g,"");

}

键盘事件：

onkeydown: 键盘按下，按着不放会一直执行

onkeyup: 键盘弹起

onkeypress: 键盘按下，用的是Ascll码

获取焦点：

检测按下的是哪一个键：e.keyCode

元素获取焦点：obj.focus();

## 网页特效面试题

**1、请谈下团购倒计时如何实现？**

团购倒计时页面端的效果比较好实现，主要是样式和时间的操作，重要的考虑时间要和服务器端同步，其实 这个效果也可以基于服务器端推送技术来实现。

**2、轮播图有哪几种？如何实现？**

纯CSS可以实现轮播图；JS实现轮播图；Boostrap

**3、JavaScript的事件流模型都有什么，以及怎么阻止他们？**

1、原始事件模型 普通的事件绑定，比如事件赋值，按钮上绑定事件等

2、DOM2事件模型

addEventListener("eventType"，"handler"，"true!false")；

removeEventListner("eventType"，"handler"，"true!false")；

气泡模型（与ie有点区别） ，冒泡和捕获

3、IE模型 气泡模型

attachEvent("eventType"，"handler")

detachEvent("eventType"，"handler") 与dom2不同的是eventType有on前缀

**4、什么是事件冒泡/捕获？**

事件冒泡： 事件到达事件目标之后不会结束，会逐层向上冒泡，直至document对象，也就是子元素事件的触发会影响父元素的事件；

开关事件冒泡：

A，开启事件冒泡：event.addEventListener(eventName，handler，false)；

B，关闭事件冒泡：假设传统方式事件的返回值为e，就可以通过e.stopPropagation()来关闭事件冒泡；

C ,return false; 同时具有阻止冒泡和阻止默认行为的功能

事件捕获：父元素的事件会影响子元素的事件；

开启事件捕获：event.addEventListener(eventName，handler，true)

**5、 如果需要手动写动画，你认为最小时间间隔是多久，为什么？（阿里）**

多数显示器默认频率是60Hz，即1秒刷新60次，所以理论上最小间隔为1/60＊1000ms ＝ 16.7ms

**6、 documen.write和 innerHTML的区别?**

document.write只能重绘整个页面   
 innerHTML可以重绘页面的一部分

# jQuery

## 一、jQuery入口函数和js入口函数的区别

jQuery的入口函数：$(function(){}); 或者 $(document).ready(function(){}); jQuery === $;

$ 在jQuery中就是一个函数，根据传入参数类型的不同，分别进行不同的处理，来实现不同的功能

js入口函数指的是：window.onload = function() {};

jQuery对象和DOM对象的相互转换

$("#btn")本质上存储的就是DOM对象，所以可以使用[]的形式，来获取数组里面的元素

DOM对象此处指的是：使用js操作DOM返回的结果。

var btn = document.getElementById(“btnShow”); // btn就是一个DOM对象

jQkuery对象此处指的是：使用jQuery提供的操作DOM的方法返回的结果。

jQuery拿到DOM对象后又对其做了封装，让其具有了jQuery方法的jQuery对象，说白了，就是把DOM对象重新包装了一下。

DOM对象转换成jQuery对象：

var $btn1 = $(btn); // 此时就把DOM对象btn转换成了jQuery对象$btn1

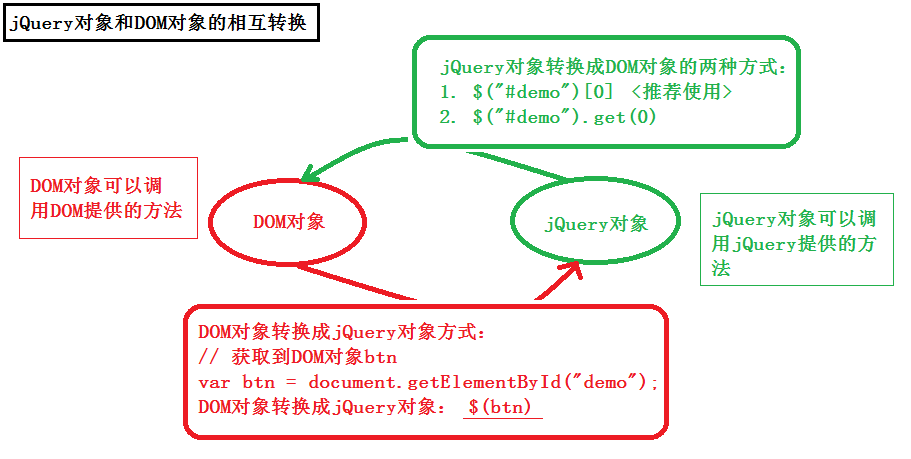
// $(document）.ready(function(){}); // 调用入口函数

// 此处是将document这个js的DOM对象，转换成了jQuery对象，然后才能调用jQuery提供的方法：ready

jQuery对象转换成DOM对象：

// 第一种方式var btn1 = $btn[0]; // 此时就把jQuery对象$btn转换成了DOM对象btn1

// 第二种方式var btn2 = $btn.get(0);// 此时就把jQuery对象$btn转换成了DOM对象btn2



## 二、jQuery选择器（重点）

### 1、基本选择器

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| # | Id选择器 |
| . | 类选择器 |
| element | 标签选择器 |

### 2、层级选择器

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| 空格 | 后代选择器 |
| > | 子代选择器 |
| one+two | 选取所有one元素同辈的下一个元素two（如果有几个one，就有几个two，除非紧挨着one后面没有同级元素two） |
| one~two | 选取one元素同辈的two的所有元素 |

### 3、基本过滤选择器

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| :first | 选取第一个元素 |
| :last | 选取最后一个元素 |
| :not(selector) | 选取不是这个元素的其他元素 |
| :eq(index) | 选择索引号为index的一个元素，index从0开始 |
| :odd | 选择索引号为奇数的所有元素，index从0开始 |
| :even | 选择索引号为偶数的所有元素，index从0开始 |
| :gt(index) | 选取索引大于index的元素 |
| :lt(index) | 选取索引小于index的元素 |
| :header | 选取所有的标题元素，如h1,h2,h3 |
| :animated | 选取当前正在执行动画的所有元素 |
| :focus | 选取当前获取焦点的元素 |

### 4、筛选选择器（方法）

|  |  |
| --- | --- |
| 符号(名称) | 说明 |
| find(selector) | 查找指定元素的所有后代元素（子子孙孙） |
| children() | 查找指定元素的直接子元素（亲儿子元素） |
| siblings() | 查找所有兄弟元素（不包括自己） |
| :parent() | 查找父元素（亲的） |
| :nth-child(index/even/odd/equation) | 选取父元素下的第index个子元素或者奇偶元素，index从1算起 |
| :first-child | 获取个父元素的第一个子元素 |
| :last-child | 选取父元素的最后一个子元素 |
| :eq(index) | 查找指定元素的第index个元素，  index是索引号，从0开始 |
| index() | 获取指定元素的索引号（从0开始的） |
| prevAll() | 获取指定元素之前的所有兄弟元素 |
| nextAll() | 获取指定元素之后的所有兄弟元素 |
| 内容过滤选择器 | |
| :contains(text) | 选取含有文本内容为“text”的元素 |
| :empty | 选取不包含子元素或者文本的空元素 |
| :has(selector) | 选取含有选择器所匹配的元素的元素 |
| :parent() | 选取含有子元素或者文本的元素 |
| 可见性过滤选择器 | |
| :hidden | 选取所有不可见的元素 |
| :visible | 选取所有可见的元素 |
| 属性过滤选择器 | |
| [ attribute ] | 选取拥有此属性的元素 |
| [ attribute = value ] | 选取属性的值为value的元素 |
| [ attribute != value ] | 选取属性的值不等于value的元素 |
| [ attribute ^= value ] | 选取以属性的值value开始的元素 |
| [ attribute $= value ] | 选取属性的值以value结束的元素 |
| [ attribute \*= value ] | 选取属性的值含有value的元素 |
| [ attribute |= value ] | 选取属性等于给定字符串或以该字符串为前缀（该字符串后跟一个连字符‘-’）的元素 |
| [ attribute ~= value ] | 选取属性的值含有value的元素 |

检查某个元素是否存在的方法：if($("#tt").length > 0) {}

## 三、样式操作 （重点）

### 1、样式属性操作

.css()

作用：设置或获取元素的样式属性值

a设置单个样式：$(selector).css(“color”, “red”);

b设置多个样式：$(selector).css({“color”: “red”, “font-size”: “30px”})

var cssStyle = {“color”: “red”, “font-size”: “30px”};$(selector).css(cssSTyle);

c获取样式属性操作：$(selector).css(“font-size”);

### 2、类操作

添加类样式：

$(selector).addClass(className) 为指定元素添加类classNam

移除类样式：

$(selector).removeClass(className) 为指定元素移除类 className

$(selector).removeClass(); 不指定参数，表示移除被选中元素的所有类

判断有没有类样式：

$(selector).hasClass(calssName) 判断指定元素是否包含类 className

切换类样式：

$(selector).toggleClass(className) 为指定元素切换类，该元素有类则移除，没有指定类则添加。

## 四、jQuery 动画（掌握）

### 1、隐藏显示动画

（1）show() 参数可以有两个，第一个是动画的执行时长，第二个是动画执行完成后的回调函数。

参数一可以是数值类型或者字符串类型

参数二表示：动画执行完后立即执行的回调函数

$(selector).show(2000, function() {});

hide方法作用：让匹配元素隐藏掉

### 2、滑入滑出动画

（1）滑入动画效果（联想：生活中的卷帘门）

作用：让元素以下拉动画效果展示出来

$(selector).slideDown(speed,callback);默认值：400毫秒（同样适用于fadeIn/slideDown/slideUp）

（2）滑出动画效果

作用：让元素以上拉动画效果隐藏起来

$(selector).slideUp(speed,callback);

（3）滑入滑出切换动画效果

$(selector).slideToggle(speed,callback);

### 3、淡入淡出动画

（1）淡入动画效果

作用：让元素以淡淡的进入视线的方式展示出来

$(selector).fadeIn(speed, callback);

（2） 淡出动画效果

$(selector).fadeOut(1000);作用：让元素以渐渐消失的方式隐藏起来

（3）淡入淡出切换动画效果

作用：通过改变不透明度，切换匹配元素的显示或隐藏状态

$(selector).fadeToggle('fast',function(){});

（4）改变不透明度到某个值

与淡入淡出的区别：淡入淡出只能控制元素的不透明度从 完全不透明 到完全透明；而fadeTo可以指定元素不透明度的具体值。并且时间参数是必需的！

作用：调节匹配元素的不透明度

$(selector).fadeTo(1000, .5); 参1：时长，参2：不透明值、 0全透，1全不透

有规律的体现：

jQuery提供的这几个动画效果控制的CSS属性包括：高度、宽度、不透明度。{height:400px; width:300px; opacity:.4;}这三个CSS属性的共性是：属性值只有一个，并且这个值是数值（去掉单位后）。

### 4、自定义动画

注意：所有能够执行动画的属性必须只有一个数字类型的值。

$(selector).animate({params},speed,callback);

作用：执行一组CSS属性的自定义动画

参1：要执行动画的CSS属性（必选）

参2：执行动画时长（可选）

参3：动画执行完后立即执行的回调函数（可选）

### 5、stop()停止动画

$(selector).stop(clearQueue,jumpToEnd);

参1：是否清空所有的后续动画

参2：是否立即执行完当前正在执行的动画

解释：

当调用stop()方法后，队列里面的下一个动画将会立即开始。但是，如果参数clearQueue被设置为true，那么队列面剩余的动画就被删除了，并且永远也不会执行。

如果参数jumpToEnd被设置为true，那么当前动画会停止，但是参与动画的每一个CSS属性将被立即设置为它们的目标值。比如：slideUp()方法，那么元素会立即隐藏掉。如果存在回调函数，那么回调函数也会立即执行。

注意：如果元素动画还没有执行完，此时调用sotp()方法，那么动画将会停止。并且动画没有执行完成，那么回调函数也不会被执行。

## 五、jQuery节点操作

### 1、操作元素节点（掌握）

（1）动态创建元素

var spanNode = $(“<span>我是一个span元素</span>”);

（2）添加元素append() 参数jQuery对象

$(selector).append($node);// 在$(selector)中追加$node

作用：在被选元素的最后一个子元素（或内容）后面插入内容（页面中存在或者创建出来的元素都可以）

如果是页面中存在的元素，那么调用append()后，会把这个元素从原先的位置移除，然后再插入到新 的位置。如果是给多个目标追加一个元素，那么append()方法的内部会复制多份这个元素，然后追加到多个目标里面去。

（3）$(selector).append('<div></div>');// 在$(selector)中追加div元素，参数为htmlString

①appendTo()

$(selector).appendTo(node);把$(selector)追加到node中去

②prepend()

$(selector).prepend(node);作用：在元素的第一个子元素前面追加内容或节点

③after()

$(selector).after(node);作用：在被选元素之后，作为兄弟元素插入内容或节点

④before()

$(selector).before(node);作用：在被选元素之前，作为兄弟元素插入内容或节点

**（4）html创建元素（推荐使用，重点）**

作用：①设置内容，参数为普通字符串

②创建元素，如果传入的是符合html规则的字符串，就会创建出相应的元素

$(selector).html(‘<span>传智播客</span>’);// 动态创建元素

③获取内容$(selector).html();// 获取html内容，有什么返回什么

调用这个方法，首先会清空原来的内容，然后再设置新的内容

（5）清空元素

$(selector).empty();// 清空指定元素的所有子元素（光杆司令）// 没有参数

$(selector).html(“”);// “自杀” 把自己（包括所有内部元素）从文档中删除掉

$(selector).remove();

（6）复制元素

(selector).clone();// 复制$(selector)所匹配到的元素// 返回值为复制的新元素

总结：推荐使用html(“<span></span>”)方法来创建元素或者html(“”)清空元素

### 2、操作form表单

（1）属性操作

①设置属性：

$(selector).attr("title", ''美女");参1：要设置的属性名称，参2：属性值，参2不存在返回属性对应的值

$(selector).prop(); 参2不存在返回false

用法和attr 一样，设置input和button的disabled属性，以及checkbox的checked属性 、selected属性，这几种动态改变的属性必须用prop

②获取属性：

$(selector).attr(“title”);参数：要获取的属性的名称，返回要获取的属性对应的值

此时，返回指定属性的值

③移除属性：

$(selector).removeAttr(“title”); // 参数为：要移除的属性的名称

④：selected 选择selected中被选中的元素

注意：设置selected、checked、disabled要使用.prop()方法。

（2）值和内容

①val()方法：

作用：设置或返回表单元素的值，例如：input,select,textarea的值

$(selector).val();// 获取匹配元素的值，只匹配第一个元素

$(selector).val(“具体值”);// 设置所有匹配到的元素的值

另外：val()能使select、checkBox、radio相应的选项被选中 val(' 上海 ');

②text() 方法:

作用：设置或获取匹配元素的文本内容

$(selector).text();//获取操作不带参数（注意：这时候会把所有匹配到的元素内容拼接为一个字符串）

$(selector).text(“我是内容”);//参数表示要设置的文本内容，会清空原有内容

### 3、尺寸位置操作

（1）高度操作height() ： 作用：设置或获取匹配元素的高度值

$(selector).height(200);//带参数表示设置高度 $(selector).height();//不带参数获取高度

（2）宽度操作width() ： 作用：设置或获取匹配元素的宽度值

(selector).width(200);//带参数表示设置宽度 $(selector).width();//不带参数获取宽度

（3）坐标值操作

①offset() 作用：获取或设置元素相对于文档左上角的位置

$(selector).offset();// 无参数表示获取，返回值为：{top:num,left:num}，值是相对于document的位置

$(selector).offset({left:100, top: 150});// 设置，参数数值类型不带单位

如果没有定位，会设置position：relative，如果有非static，则不会设置，但会把原来的值给覆盖掉

注意点：设置offset后，如果元素没有定位(默认值：static)，则被修改为relative

②scrollTop() 作用：获取或者设置元素垂直方向滚动的位置

$(selector).scrollTop();// 无参数表示获取偏移

$(selector).scrollTop(100);// 有参数表示设置偏移，参数为数值类型

对scrollTop的理解：

垂直滚动条位置 是可滚动区域 在 可视区域上方的 被隐藏区域的高度。

如果滚动条在最上方没有滚动 或者 当前元素没有出现滚动条，那么这个距离为0

③scrollLeft() 作用：获取或者设置元素水平方向滚动的位置

$(selector).scrollLeft(100);

## 六、jQuery事件机制（重点）

jQuery的事件机制，指的是：jQuery对JavaScript操作DOM事件的封装，包括了：事件绑定、事件解绑、事件触发。

### 1、事件绑定

简单事件绑定 >> bind事件绑定 >> delegate事件绑定 >> on【重点】

1、简单事件绑定：VC

click() 单击事件

mouseenter() 鼠标进入事件

mouseleave() 鼠标离开事件

dbclick() 双击事件

change() 改变事件，如：文本框值改变，下拉列表值改变等

focus() 获得焦点事件

blur() 失去焦点事件

keydown() 键盘按下事件

2、on方式

语法：$(selector).on(events,[selector],[data],function());

// 第一个参数：events，绑定事件的类型

// 第二个参数：selector, 执行事件的后代元素

// 第三个参数：data，传递给处理函数的数据，事件触发的时候通过event.data来使用(用处不大)

// 第四个参数：function()，事件处理函数

绑定多个事件

$(selector).on(“click mouseenter”, function(){});

表示给$(selector)匹配的元素绑定单击和鼠标进入事件

### 2、事件解绑 off

off解绑on方式绑定的事件（重点）。

$(selector).off(); 解绑匹配元素的所有事件

$(selector).off(“click”); 解绑匹配元素的所有click事件,自身的和动态的。

$(selector).off(“click”,“\*\*”); 参1：要解绑的事件，参2：解绑动态创建的事件。自身绑定的事件还在

### 3、事件触发

事件触发的时候只会触发当前元素被执行的事件，单击按钮触发div的单击事件

1、$(selector).click(); 简单事件触发，触发 click事件

2、trigger()方法触发事件，触发浏览器的默认行为

$("button").on("click",function(){

$("div").trigger("click");

});

$(selector).trigger(“click”);参数：要触发事件的名称

3、triggerHandler触发 事件响应方法，不触发浏览器行为

$(selector).triggerHandler(“focus”);比如:文本框获得焦点是浏览器的默认行为

### 4、事件对象

$(function(event){}): event 事件对象

event.data 传递给事件处理程序的额外数据

event.delegateTarget 代理绑定事件的元素

event.currentTarget 绑定事件的元素，等同于this，this：哪个元素触发的事件，this就指向这个元素

event.target  触发事件的元素，不一定===this

event.pageX 鼠标相对于页面的位置

event.clientX  距离浏览器的位置

event.screenX 相对于电脑屏幕

event.offsetX 相对于定位的父元素，或者body

event.stopPropagation()； 阻止事件冒泡

event.preventDefault();  阻止浏览器默认行为

event.type 事件类型：click，dbclick…

event.which 鼠标的按键类型：左1 中2 右3

event.keyCode 键盘按键代码

事件执行的顺序：

如果是单击的按钮，首先会触发按钮的单击事件，然后再触发div的单击事件

阻止事件冒泡：

event.stopPropagation()； 阻止事件冒泡

return false; 同时具有阻止冒泡和阻止默认行为的功能

### 5、链式编程

链式编程代码示例：

$(“li”).parent(“ul”).parent(“div”).siblings(“div”).children(“div”).html(“内容”);

链式编程原理：return this;

通常情况下，只有设置操作才能把链式编程延续下去。因为获取操作的时候，会返回获取到的相应的值，无法返回 this。

end(); // 结束当前链最近的一次过滤操作，并且返回匹配元素之前的一次状态

### 6、隐式迭代

// 设置操作$(“div”).css(“color”, “red”);

// 获取操作$(“div”).css(“color”); // 返回第一个元素的值

1、隐式迭代的意思是：在方法的内部会为匹配到的所有元素进行循环遍历，不用我们再进行循环

如果获取的是多元素的值，大部分情况下返回的是第一个元素的值。

2、each方法

大部分情况下是不需要使用each方法的，因为jQuery的隐式迭代特性。

如果要对每个元素做不同的处理，这时候就用到了each方法

作用：遍历jQuery对象集合，为每个匹配的元素执行一个函数

$(selector).each(function(index,element){});参1:当前元素在所有匹配元素中的索引号,参2：当前元素（DOM对象）

### 7、多库共存（了解）

此处多库共存指的是：jQuery占用了$ 和jQuery这两个变量。当在同一个页面中引用了jQuery这个js库，并且引用的其他库（或者其他版本的jQuery库）中也用到了$或者jQuery这两个变量，那么，要保证每个库都能正常使用，这时候就有了多库共存的问题。

// 模拟另外的库使用了 $ 这个变量

// 此时，就与jQuery库产生了冲突

var $ = { name : “itecast” };

解决方式：$.noConflict();

// 作用：让jQuery释放对$的控制权，让其他库能够使用$符号，达到多库共存的目的。此后，只能使用jQuery来调用jQuery提供的方法

### 8、jQuery插件机制

第三方插件

jQuery.color.js

animate()自定义动画：不支持背景的动画效果

animate动画支持的属性列表

使用步骤：

1.引入jQuery文件

2.引入插件

3.使用插件

全局jQuery函数扩展方法

$.pluginName = function() {};

jQuery对象扩展方法

$.fn. pluginName = function() {};

## jQuery面试题

**25、简述在jQuery中.eq()和.get()的异同？**

相同：

get() ：取得其中一个匹配的元素。数字序号表示取得第几个匹配的元素

eq()：获取第N个元素，下标都是从0开始， 用法基本相同。

不同：

eq返回的是一个jquery对象； 返回的是jQuery对象，就可以继续调用其他方法。

get返回的是一个html 对象数组；不能调用jQuery的其他方法；

1. **jQuery各个版本有哪些差异？**

**目前jQuery有三个大版本：**

1.x：兼容ie678,使用最为广泛的，官方只做BUG维护，功能不再新增。因此一般项目来说，使用1.x版本就可以了，最终版本：1.12.4 (2016年5月20日)

2.x：不兼容ie678，很少有人使用，官方只做BUG维护，功能不再新增。如果不考虑兼容低版本的浏览器可以使用2.x，最终版本：2.2.4 (2016年5月20日)

3.x：不兼容ie678，只支持最新的浏览器。除非特殊要求，一般不会使用3.x版本的，很多老的jQuery插件不支持这个版本。目前该版本是官方主要更新维护的版本。

**3、原生JS的window.onload与Jquery的$(document).ready(function() {})，$(function() {})有什么不同？**

1.执行时间

window.onload必须等到页面内包括图片的所有元素加载完毕后才能执行。 $(document).ready()是DOM结构绘制完毕后就执行，不必等到加载完毕。

1. 编写个数不同

window.onload不能同时编写多个，如果有多个window.onload方法，只会执行一个 $(document).ready()可以同时编写多个，并且都可以得到执行

1. 简化写法

window.onload没有简化写法 $(document).ready(function(){})可以简写成$(function(){})；

**4、Jquery.on这个方法怎么看？**

答： jQuery.on()方法可以给匹配元素（可以是多个）绑定一个或多个函数, off 可以解除绑定。

**5、Jquery与jQuery UI有啥区别？**

\*jQuery是一个js库，主要提供的功能是选择器，属性修改和事件绑定等等。   
 \*jQuery UI则是在jQuery的基础上，利用jQuery的扩展性，设计的插件。 提供了一些常用的界面元素，诸如对话框、拖动行为、改变大小行为等等

**6、描述一下.attr()和.prop()方法的区别。**

.attr()方法是操作属性节点，.prop()方法是操作获取到的对应js对象的属性。在遇到要获取或设置checked、selected、readonly和disabled等属性时，用prop方法显然更好。.prop()方法更高效，因为.attr()方法要访问DOM。

# HTML5+CSS3

## 一、HTML5新增标签和重新定义的标签

### 1、结构标签 （重点）

块级元素，有意义的div

<article> 定义一篇文章，强调独立性

<header> 定义一个页面或一个区域的头部  
 <nav> 定义导航链接

<section> 定义一个区域，例如将一块内容分成几段

<aside> 定义页面内容部分的侧边栏  
 <hgroup> 定义文件中一个区块的相关信息，里面放h系列的标签，最好h3

<figure> 定义一组媒体内容以及它们的标题

<figcaption> 定义figure元素的标题

<footer> 定义一个页面或一个区域的底部

<dialog> 定义一个对话框类似微信

### 2、多媒体标签（掌握）

<video> 定义一个视频

<audio> 定义音频内容

<source> 定义媒体资源

<canvas> 定义图片

<embed> 定义外部的可交互的内容或插件 比如 flash

### 3、Web 应用标签（了解）

<menu> 命令列表

火狐浏览器支持

<menuitem> menu命令列表标签（嵌入系统）

<command> menu标记定义一个命令按钮

<meter> 实时状态显示：气压、气温（ 状态标签）

属性：min、max、value、low、high value在low和high中间正常，之外会变色警告

<progrss> 任务过程：安装、加载（状态标签）进度条

<datalist> 为input标记定义一个下拉列表，配合option

<detalis> 标记定义一个元素的详细内容，配合dt、dd，有下拉框

### 4、注释标签

<ruby> 定义注释或音标

<rp> 告诉那些不支持Ruby元素的浏览器如何去显示，注释括号

<rt> 定义对ruby的注释内容文本，注释在ruby上面就像汉语汉字上的拼音

### 5、其他标签

<keygen> 定义表单里一个生成的键值（加密信息传送）

<mark> 定义有标记的文本（黄色选中状态）

<output> 定义一些输出类型，计算表单结果配合oninput事件

### 6、重新定义的标签

<dd> 问题

<dt> 描述

<hr> 表示主题结束，是一条水平线

<small> 表示小字体，例如注释或者法律条款

<menu> 定义用户界面的菜单，配合commond或者menuitem使用

## 二、HTML5视频、音频播放事件属性与API控件

### 1、video、audio标签

<video src = "movie.mp4" controls = "controls"><video>

或者<video controls = "controls">

<source src = "movie.mp4">

<video>

|  |  |  |
| --- | --- | --- |
| 属性 | 值 | 描述 |
| autoplay | autoplay（可省略） | 视频自动播放 |
| controls | controls（可省略） | 向用户显示播放控件 |
| width | px | 播放器宽度 |
| height | px | 播放器高度 |
| loop | loop、数字 | 播放完是否继续播放、播放次数 |
| preload | proload | 是否等待加载完再播放 |
| src | url | 视频url地址 |
| poster | imgurl | 加载等待的画面图片 |
| autobuffer | autobuffer | 设置为浏览器缓冲方式，不设置autoplay时有效 |

### 2、HTML5视频API 控件

（1）获得video标签

①通过DOM对象 var video = document.getElementById("videoID");

②通过jQuery 的方法 var video = $("#videoID")[0];

video标签的属性

载入视频：load()

播放视频：play()

暂停:pause()

快进10秒：currentTime +=10

播放速度增加：playbackRate ++

播放速度增加0.1：playbackRate += 0.1

音量增加：volume += 0.1

静音：muted = true

（3）事件

canplay

duration 媒体长度

timeupdate 媒体当前位置

## 三、表单

### 1、表单输入类型

|  |  |  |
| --- | --- | --- |
| 类型 | 使用示例 | 含义 |
| email | <input type="email"> | 输入邮箱格式 |
| tel | <input type="tel"> | 输入手机号码格式 |
| url | <input type="url"> | 输入url格式 |
| number | <input type="number"> | 输入数字格式 |
| search | <input type="search"> | 搜索框（体现语义化） |
| range | <input type="range"> | 自由拖动滑块 |
| color | <input type="color"> | 拾色器 |
| time | <input type="time"> | 小时 |
| date | <input type="date"> | 年月日 |
| datetime | <input type="datetime"> | 时间输入框 |
| month | <input type="month"> | 年月 |
| week | <input type="week"> | 年周 |

### 2、表单元素

|  |  |
| --- | --- |
| 元素 | 含义 |
| <datalist> | 数据列表 |
| <keygen> | 生成加密字符串 |
| <output> | 输出结果 |
| <meter> | 度量器 |

### 3、表单属性

|  |  |  |
| --- | --- | --- |
| 属性 | 用法 | 含义 |
| placeholder | <input type="text" placeholder="请输入用户名"> | 占位符 |
| autofocus | <input type="text" autofocus> | 自动获得焦点 |
| multiple | <input type="file" multiple> | 多文件上传 |
| autocomplete | <input type="text" autocomplete="off"> 值：off、on | 自动完成 |
| form | <input type="text" form="某表单ID"> |  |
| novalidate | <form novalidate></form> | 关闭验证 |
| required | <input type="text" required> | 必填项 |
| pattern | <input type="text" pattern="\d"> | 自定义验证 |

四、DOM扩展

### 1、获取元素

①document.getElementsByClassName ('class') 通过类名获取元素，以伪数组形式存在。

②document.querySelector('selector') 通过CSS选择器获取元素，符合匹配条件的第1个元素。

③document.querySelectorAll('selector') 通过CSS选择器获取元素，以伪数组形式存在。

### 2、类名操作

①Node.classList.add('class') 添加class

②Node.classList.remove('class') 移除class

③Node.classList.toggle('class') 切换class，有则移除，无则添加

④Node.classList.contains('class') 检测是否存在class

Node指一个有效的DOM节点，是一个通称。

### 3、自定义属性

在HTML5中我们可以自定义属性，其格式如下data-\*=""，例如

data-info="我是自定义属性"，通过Node.dataset['info'] 我们便可以获取到自定义的属性值。

Node.dataset是以对象形式存在的，当我们为同一个DOM节点指定了多个自定义属性时，Node.dataset则存储了所有的自定义属性的值。

假设某元素 <div id="demo" data-name="itcast" data-age="10">

var demo = document.querySelector('#demo');

1、读取自定义属性 demo.dataset（所有的属性及属性值）或者 demo.dataset['age']（age属性的值）

2、设置demo.dataset['name'] = 'web developer'

## 五、新增API

### 1、网络状态（了解）

window.online 用户网络连接时被调用

window.offline 用户网络断开时被调用

2、全屏  
 Node.requestFullScreen() 开启全屏显示

document.cancelFullScreen() 关闭全屏显示，只能通过document才能关闭  
 document.fullScreen检测当前是否处于全屏

全屏伪类选择器

:full-screen .box {}、:-webkit-full-screen {}、:moz-full-screen {}

### 3、文件读取（掌握）

通过FileReader对象我们可以读取本地存储的文件，使用 [File](https://developer.mozilla.org/zh-CN/docs/DOM/File" \t "_blank" \o "此页面仍未被本地化, 期待您的翻译!)对象来指定所要读取的文件或数据。其中File对象可以是来自用户在一个 [<input>](https://developer.mozilla.org/zh-CN/docs/HTML/Element/input" \t "_blank" \o "This article hasn't been written yet. Please consider contributing!)元素上选择文件后返回的[FileList](https://developer.mozilla.org/zh-CN/docs/DOM/FileList" \t "_blank" \o "一个FileList对象通常来自于一个HTML input元素的files属性,你可以通过这个对象访问到用户所选择的文件.该类型的对象还有可能来自用户的拖放操作,查看DataTransfer对象了解详情.)对象，也可以来自由拖放操作生成的  [DataTransfer](https://developer.mozilla.org/zh-CN/DragDrop/DataTransfer" \t "_blank" \o "En/DragDrop/DataTransfer)。

实例化一个读取器：var reader = new FileReader();

读取文件reader.readAsDateURL();

reader.readAsBinaryString();

reader.readAsTEXT();  
 事件监听onload 当文读取完成时调用

属性result 文件读取结果

### 4、拖拽（掌握）

①拖拽元素：页面中设置了draggable="true"属性的元素，其中<img>、<a>标签默认是可以被拖拽的

②目标元素：页面中任意的元素

③事件监听：根据元素类型不同，需要设置不同的事件监听

a拖拽元素

ondrag 应用于拖拽元素，整个拖拽过程都会调用

ondragstart 应用于拖拽元素，当拖拽开始时调用，只触发一次

ondragleave 应用于拖拽元素，当鼠标离开拖拽元素时调用

ondragend 应用于拖拽元素，当拖拽结束时调用

b目标元素

ondragenter 应用于目标元素，当拖拽元素进入时调用

ondragover 应用于目标元素，当停留在目标元素上时调用，

ondrop 应用于目标元素，当在目标元素上松开鼠标时调用

ondragleave 应用于目标元素，当鼠标离开目标元素时调用

### 5、地理定位（了解）

①获取当前地理信息（只获取一次）

navigator. geolocation.getCurrentPosition(successCallback, errorCallback, options)

②重复获取当前地理信息（多次）

navigator. geolocation.watchPosition(successCallback, errorCallback, options)

当成功获取地理信息后，会调用succssCallback，并返回一个包含位置信息的对象position。

position.coords.latitude纬度

position.coords.longitude经度

position.coords.accuracy精度

position.coords.altitude海拔高度

当获取地理信息失败后，会调用errorCallback，并返回错误信息error

可选参数 options 对象可以调整位置信息数据收集方式

a) enableHighAccuracy 高精度模式 true、false

b) timeout 超时设置，单位为ms

c) maximumAge表示浏览器重新获取位置信息的时间间隔，单位为ms

### 6、历史管理（了解）

提供window.history，对象我们可以管理历史记录，可用于单页面应用，Single Page Application，可以无刷新改变网页内容。

①pushState(data, title, url) 追加一条历史记录

data对象，用于存储自定义数据，通常设为null

title网页标题，基本上没有被支持，一般设为空

url 以当前域为基础增加一条历史记录，不可跨域设置

②replaceState(data, title, url) 与pushState()基本相同，不同之处在于replaceState()，只是替换当前url，不会增加/减少历史记录。

③事件监听

onpopstate事件，当前进或后退时则触发，通过事件对象ev.state可以读取到存储的数据。

### 7、Web存储（本地存储）(重点)

① a、设置、读取方便

b、容量较大，sessionStorage约5M、localStorage约20M

c、只能存储字符串，可以将对象JSON.stringify() 编码后存储

②window.sessionStorage

a、生命周期为关闭浏览器窗口

b、在同一个窗口下数据可以共享

③window.localStorage

a、永久生效，除非手动删除

b、可以多窗口共享

④方法详解

setItem(key, value) 设置存储内容

getItem(key) 读取存储内容

removeItem(key) 删除键值为key的存储内容

clear() 清空所有存储内容

key(n) 以索引值来获取存储内容

⑤其它

WebSQL、IndexDB

### 8、应用缓存

HTML5中我们可以轻松的构建一个离线（无网络状态）应用，只需要创建一个cache manifest文件。

①优势

a、可配置需要缓存的资源

b、网络无连接应用仍可用

c、本地读取缓存资源，提升访问速度，增强用户体验

d、减少请求，缓解服务器负担

②缓存清单

一个普通文本文件，其中列出了浏览器应缓存以供离线访问的资源，推荐使用.appcache为后缀名，添加MIME类型

AddType text/cache-manifest .appcache

例如我们创建了一个名为demo.appcache的文件，然后在需要应用缓存在页面的根元素(html)添加属性manifest="demo.appcache"，路径要保证正确。

③manifest文件格式

a、顶行写CACHE MANIFEST

b、CACHE: 换行 指定我们需要缓存的静态资源，如.css、image、js等

c、NETWORK: 换行 指定需要在线访问的资源，可使用通配符

d、FALLBACK: 换行 当被缓存的文件找不到时的备用资源 ./online.html ./offline.html

当online没有用offline替换

④其他

b、可以指定多个CACHE: NETWORK: FALLBACK:，无顺序限制

c、#表示注释，只有当demo.appcache文件内容发生改变时或者手动清除缓存后，才会重新缓存。

d、chrome 可以通过chrome://appcache-internals/工具和离线（offline）模式来调试管理应用缓存

9、多媒体

## 六、CSS3选择器（重点）

### 1、属性选择器

①E[attr] 表示存在attr属性即可；

②E[attr=val] 表示属性值完全等于val；

③E[attr\*=val] 表示在属性值的“任意”位置包含val字符；

④E[attr^=val] 表示在属性值的“开始”位置包含val字符；

⑤E[attr$=val] 表示在属性值的“结束”位置包含val字符；

### 2、伪类选择器

①以某元素相对于其父元素或兄弟元素的位置。

E:first-child第一个子元素

E:last-child最后一个子元素

E:nth-child(n) 第n个子元素

E:nth-last-child(n) 同E:nth-child(n) 相似，只是倒着计算

②目标伪类选择器

E:empty 选中没有任何子节点的E元素；（使用不是非常广泛）

E:target 结合锚点进行使用，处于当前锚点的元素会被选中；

E:not () 选中除了当前锚点的其他元素

### 3、伪元素选择器

E::first-letter文本的第一个单词或字（如中文、日文、韩文等）；

E::first-line 文本第一行；

E::selection 可改变选中文本的样式；

是一个行内元素，需要转换成块元素

":" 与 "::" 区别在于区分伪类和伪元素,在开发中使用":";

## 七、css3颜色（了解）

RGBA、HSLA，可以设置颜色值的透明度，相较opacity，它们不具有继承性，即不会影响子元素的透明度。

H 色调 取值范围0~360，0/360表示红色、120表示绿色、240表示蓝色

S 饱和度 取值范围0%~100%

L 亮度 取值范围0%~100%

A 透明度 取值范围0~1

关于透明度：

1、opacity只能针对整个盒子设置透明度，子盒子及内容会继承父盒子的透明度；

2、transparent 不可调节透明度，始终完全透明background-color：transparent

## 八、文本（掌握）

text-shadow，可分别设置偏移量、模糊度、颜色（可设透明度）。

参1：水平偏移量 正值向右 负值向左；

参2：垂直偏移量 正值向下 负值向上；

参3：模糊度（模糊半径）是不能为负值；

参4：颜色

## 九、边框（掌握）

边框圆角：border-radius：0 0 0 0 / 0 0 0 0; 前四个是x轴圆角半径，后面四个是Y轴圆角半径

边框阴影：border-shadow

参1、水平偏移量 右正左负；

参2、垂直偏移量 下正上负；

参3、模糊半径是不能为负值；

参4、颜色

参5、inset可以设置内阴影；

边框图片：border-image

参1：source：图片路径

参2：slice：切割，包含4参数（a,b,c,d）按照上右下左顺序

参3：repeat是否平铺 包含三个值：stretch拉伸（默认值）、repeat重复、round平铺

参4：width宽度

参5：偏移位置

## 十、盒模型（重点）

CSS3中可以通过box-sizing 来指定盒模型，即可指定为content-box、border-box，这样我们计算盒子大小的方式就发生了改变。

可以分成两种情况：

1、box-sizing: border-box 计算方式为content = width – border - padding

2、box-sizing: content-box 计算方式为content = width

box-shadow属性

取值：水平偏移量、垂直偏移量、模糊度、外延值、color、inset（内阴影）

box-reflect盒子倒影属性:可以对盒子模型进行倒影设置。

值： direction（above、below、left、right）、offset（图片与倒影间隔）、mask-box-image(url)

## 十一、背景（掌握）

背景在CSS3中也得到很大程度的增强，比如背景图片尺寸、背景裁切区域、背景定位参照点、多重背景等。

### 1、background-size设置背景图片的尺寸

cover会自动调整缩放比例，保证图片始终填充满背景区域，如有溢出部分则会被隐藏。

contain会自动调整缩放比例，保证图片始终完整显示在背景区域。

也可以使用长度单位或百分比

### 2、background-origin设置背景定位的原点

border-box 以边框做为原点；

padding-box 以内边距做为原点；

content-box 以内容区做为原点；

### 3、background-clip设置背景图片的裁切区域

border-box 裁切边框以内为背景区域；

padding-box裁切内边距以内为背景区域；

content-box 裁切内容区做为背景区域；

background-break 设置背景图片进行平铺时的循环方式

bounding-box 在整个元素中进行平铺

each-box 在每一行中进行平铺

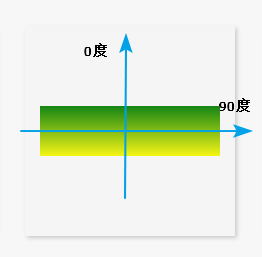
continuous 下一行中的图像紧接着上一行中的图像继续平铺

## 十二、渐变（掌握）

### 1、线性渐变

linear-gradient线性渐变指沿着某条直线朝一个方向产生渐变效果。

取值：方向、起止色、渐变范围



### 2、径向渐变

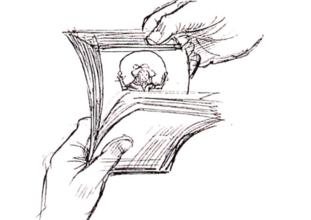
radial-gradient径向渐变指从一个中心点开始沿着四周产生渐变效果

取值：圆半径、圆中心、起止色、渐变范围

## 十三、过渡（重点）

过渡可以实现元素不同状态间的平滑过渡（补间动画），经常用来制作动画效果。

帧动画：通过一帧一帧的画面按照固定顺序和速度播放。如电影胶片



补间动画：自动完成从起始状态到终止状态的的过渡。

特点：当前元素只要有“属性”发生变化时，可以平滑的进行过渡。e

过渡属性在A、B状态都可以，如果都要慢慢变化就设在A上,hover或者JS添加class

简写：第一个时间是执行时间，第二个是延迟时间，其他的没有顺序要求，但是最好按照下面的顺序书写

transition-property设置过渡属性

取值：none 没有元素

all 所有元素

index 指定的css属性

transition-duration设置过渡时间，从旧属性转换到新属性花费的事件

transition-delay设置过渡动画延迟时间

transition-timing-function设置过渡速度（几个取值肉眼很难看出来区别）

取值：ease 缓解效果，等同于cubic-bezier(0.25, 0.1, 0.25, 1.0)函数，即立方贝塞尔

linear 线性效果，等同于cubic-bezier(0.0, 0.0, 0.1, 1.0)函数

ease-in 渐显效果，等同于cubic-bezier(0.42, 0, 1.0, 1.0)函数

ease-out 渐隐效果，等同于cubic-bezier(0, 0, 0.58, 1.0)函数

ease-in-out 渐显渐隐效果，等同于cubic-bezier(0.42, 0, 0.58, 1.0)函数

cubic-bezier 特殊的立方贝塞尔曲线

## 十四、转换（掌握）

### 1、2D转换

transform：用来设置2D或3D转换

转换可以实现元素的位移、旋转、变形、缩放，甚至支持矩阵方式

①移动 translate(x, y) 改变元素的位置，相对原来的位置的变化，如果为百分比则是相对于盒子的大小；

translate3d(x,y,z) 3d空间的位置移动

②缩放 scale(x, y) 水平和垂直方向的缩放，x、y的取值可为小数，如果只有一个参数即缩放的倍数；

③旋转 rotate(deg) 可以对元素进行旋转，正值为顺时针，负值为逆时针；旋转时元素的坐标也会跟着改变

④倾斜 skew(Ydeg, Xdeg) 可以使元素按一定的角度进行倾斜，会改变元素的形状

⑤矩阵变换matrix(x, y) 改变元素的位置

### 2、3D转换

（1）3D坐标轴

用X、Y、Z分别表示空间的3个维度，三条轴互相垂直。如下图



（2）3D呈现

transform-style:preserve-3D; 内嵌元素，并且必须经过Z变形（transform）

透视

perspective :辅助性帮我们去理解元素在转换过程中的视觉效果

(0-1000)设置给父盒子，值越大，效果越不明显，看起来近大远小

3、backface-visibility

设置元素背面是否可见

## 十五、动画（重点）

### 1、定义动画序列：

a、通过@keyframes指定动画序列名称，主要先定义再调用；

b、通过百分比将动画序列分割成多个节点；（0%-100%、from-to）

c、在各节点中分别定义各属性

d、通过animation将动画应用于相应元素；

### 2、调用：animation

a、animation-name设置动画名称

b、animation-duration动画持续时间

c、animation-delay动画延时时间

d、animation-timing-function动画执行速度，linear（匀速）、ease等

e、animation-play-state动画播放状态，play、paused（暂停）、running等

f、animation-direction动画播放方向，默认值normal，alternate动画逆播（动画先正向播放再反向播放）

g、animation-fill-mode动画执行完毕后状态，forwards（结束时的状态）backwards（执行之前的状态）等

h、animation-iteration-count动画执行次数，infinite（无限次）等

i、steps(60) 表示动画分成60步完成

## 十六、伸缩布局:控制元素对齐方式（掌握）

主轴：Flex容器的主轴主要用来配置Flex项目，默认是水平方向

侧轴：与主轴垂直的轴称作侧轴，默认是垂直方向的

方向：默认主轴从左向右，侧轴默认从上到下

主轴和侧轴并不是固定不变的，通过flex-direction可以互换。

### 1、必要元素：

a、指定一个盒子为伸缩盒子 display: flex（父元素）

b、设置属性来调整此盒的子元素的布局方式 例如 flex-direction

c、明确主侧轴及方向

### 2、各属性详解

a、flex-direction调整主轴方向（默认为水平方向）

row、column、row-reverse行倒序排列、column-reverse列倒序排列

b、justify-content调整主轴对齐

flex-start、flex-end、center、space-between、 space-around

c、align-items调整侧轴对齐

flex-start、flex-end、center、flex-stretch、space-between、 space-around

d、flex-wrap控制是否换行

nowrap、wrap

e、align-content堆栈（由flex-wrap:wrap产生的独立行）对齐

flex-start 、flex-end、center

space-around 行平均分布、space-between 两端对齐、space-streach 拉伸对齐

f、flex-flow是flex-direction、flex-wrap的简写形式

g、flex控制子元素的缩放比例，分配的是剩余空间（独立属性）

h、order控制子元素的排列顺序

## 十七、多列布局（了解）

-webkit-columns-width 定义每列的宽度

-webkit-count 定义列数

-webkit-column-gap 定义两栏之间的间距uii

属性：length 长度 = column-rule-length

style 样式 = column-rule-style

color 颜色 = column-rule-color

-webkit-column-rule 设置列之间的分割线

-webkit-column-span 定义跨列显示 默认1，all 横跨所有列

-webkit-column-fill 定义列的高度是否统一，auto，balance 统一设置成内容最多的那一列的高度

## 十八、web字体（掌握）

@font-face 指定自定义字体

font-family 给自定义字体起名字

src : url () 引入字体文件

## 十九、兼容性（了解）

通过http://caniuse.com/ 可查询CSS3各特性的支持程度，一般兼容性处理的常见方法是为属性添加私有前缀，如不能解决，应避免使用，无需刻意去处理CSS3的兼容性问题。

## 二十、H5C3面试题

**1、请描述一下cookies，sessionStorage和localStorage的区别**

sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。 localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。 web storage和cookie的区别 Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。 除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要[前端开发](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)者自己封装setCookie，getCookie。但是Cookie也是不可以或缺的：Cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生。

**2、html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？** HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

拖拽释放(Drag and drop) API

语义化更好的内容标签（header,nav,footer,aside,article,section）

音频、视频API(audio,video)

画布(Canvas) API

地理(Geolocation) API

本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；sessionStorage 的数据在浏览器关闭后自动删除

表单控件，calendar、date、time、email、url、search

新的技术webworker, websocket, Geolocation

移除的元素

纯表现的元素：basefont，big，center，font, s，strike，tt，u；

对可用性产生负面影响的元素：frame，frameset，noframes；

**3、如何处理HTML5新标签的浏览器兼容问题？**

支持HTML5新标签：

IE8/IE7/IE6支持通过document.createElement方法产生的标签， 可以利用这一特性让这些浏览器支持HTML5新标签， 当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

<!--[if lt IE 9]>

<script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>

<![endif]-->

**4、CSS 选择符有哪些？哪些属性可以继承？优先级算法如何计算？ CSS3新增伪类有那些？**

1.id选择器（ # myid）

2.类选择器（.myclassname）

3.标签选择器（div, h1, p）

4.相邻选择器（h1 + p）

5.子选择器（ul > li）

6.后代选择器（li a）

7.通配符选择器（ \* ）

8.属性选择器（a[rel = "external"]）

9.伪类选择器（a: hover, li:nth-child）

优先级为:

!important > id > class > tag

CSS3新增伪类举例：

p:first-of-type 选择属于其父元素的首个 <p> 元素的每个 <p> 元素。

p:last-of-type 选择属于其父元素的最后 <p> 元素的每个 <p> 元素。

p:only-of-type 选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。

p:only-child 选择属于其父元素的唯一子元素的每个 <p> 元素。

p:nth-child(2) 选择属于其父元素的第二个子元素的每个 <p> 元素。

:enabled :disabled 控制表单控件的禁用状态。

:checked 单选框或复选框被选中。

**5、什么是 FOUC（无样式内容闪烁）？你如何来避免 FOUC？**

FOUC - Flash Of Unstyled Content 文档样式闪烁

<style type="text/css" media="all">@import "../fouc.css";</style>

而引用CSS文件的@import就是造成这个问题的罪魁祸首。IE会先加载整个HTML文档的DOM，然后再去导入外部的CSS文件，因此，在页面DOM加载完成到CSS导入完成中间会有一段时间页面上的内容是没有样式的，这段时间的长短跟网速，电脑速度都有关系。

解决方法简单的出奇，只要在<head>之间加入一个<link>或者<script>元素就可以了。

**6、介绍一下box-sizing属性？**

box-sizing属性主要用来控制元素的盒模型的解析模式。默认值是content-box。

content-box：让元素维持W3C的标准盒模型。元素的宽度/高度由border + padding + content的宽度/高度决定，设置width/height属性指的是content部分的宽/高

border-box：让元素维持IE传统盒模型（IE6以下版本和IE6~7的怪异模式）。设置width/height属性指的是border + padding + content

标准浏览器下，按照W3C规范对盒模型解析，一旦修改了元素的边框或内距，就会影响元素的盒子尺寸，就不得不重新计算元素的盒子尺寸，从而影响整个页面的布局。

**7、’data-’属性的作用是什么？如何获取’data-’属性的值？**

答：data-属性是H5中的，用来自定义属性，通过dataset属性获取。 不是所有的浏览器都支持，不支持用getAttribute获取。

# canvas 标签（了解）

## 一、canvas主要属性和方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| width | 设置canvas的宽度 |
| height | 设置canvas的长度 |

|  |  |
| --- | --- |
| 方法 | 描述 |
| save() | 保存当前环境的状态 |
| restore() | 返回之前保存过的路径状态和属性 |
| createEvent() |  |
| getContext() | 返回一个对象，指出访问绘图功能必要的API |
| toDataURL() | 返回canvas图像的URL |

## 二、颜色、样式和阴影属性和方法

|  |  |
| --- | --- |
| 属性 | 方法 |
| fillStyle | 设置或返回 填充绘画的颜色、渐变或模式 |
| strokeStyle | 设置或返回 笔触的颜色、渐变或模式 |
| shadowColor | 设置或返回 阴影的颜色 |
| shadowBlur | 设置或返回用于阴影的模糊级别 |
| shadowOffsetX | 设置或返回阴影距形状的水平距离 |
| shadowOffsetY | 设置或返回阴影距形状的垂直距离 |
| 方法 | 描述 |
| createLinearGradient() | 创建线性渐变（用在画布内容上） |
| createPattern() | 在指定的方向上重复指定的元素 |
| createRadialGradient() | 创建放射状/环形的渐变（用在画布内容上） |
| addColorStop() | 规定渐变对象中的颜色和停止位置 |

## 三、线条样式属性和方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| lineCap | 设置或返回线条的结束端点样式 |
| lineJoin | 设置或返回两条线相交时，所创建的拐角类型 |
| lineWidth | 设置或返回当前的线条宽度 |
| miterLimit | 设置或返回最大斜接长度 |

## 四、矩形方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| rect() | 创建矩形 |
| fillRect()（建议使用这种方式画矩形） | 绘制"被填充"的矩形 |
| strokeRect() | 绘制矩形（无填充） |
| clearRect() | 在给定的矩形内清除指定的像素 |

## 五、路径方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| fill() | 填充当前绘图（路径） |
| stroke() | 绘制已定义的路径 |
| beginPath() | 起始一条路径，或重置当前路径 |
| moveTo() | 把路径移动到画布中的指定点，不创建线条 |
| closePath() | 创建从当前点回到起始点的路径 |
| lineTo() | 添加一个新点，创建从该点到最后指定点的线条 |
| clip() | 从原始画布剪切任意形状和尺寸的区域 |
| quadraticCurveTo() | 创建二次贝塞尔曲线 |
| bezierCurveTo() | 创建三次方贝塞尔曲线 |
| arc() | 创建弧/曲线（用于创建圆形或部分圆） |
| arcTo() | 创建两切线之间的弧/曲线 |
| isPointInPath() | 如果指定的点位于当前路径中，返回布尔值 |

## 六、转换方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| scale() | 缩放当前绘图至更大或更小 |
| rotate() | 旋转当前绘图 |
| translate() | 重新映射画布上的 (0,0) 位置 |
| transform() | 替换绘图的当前转换矩阵 |
| setTransform() | 将当前转换重置为单位矩阵。然后运行 transform() |

## 七、文本属性和方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| font | 设置或返回文本内容的当前字体属性 |
| textAlign | 设置或返回文本内容的当前对齐方式 |
| textBaseline | 设置或返回在绘制文本时使用的当前文本基线 |

|  |  |
| --- | --- |
| 方法 | 描述 |
| fillText() | 在画布上绘制"被填充的"文本 |
| strokeText() | 在画布上绘制文本（无填充） |
| measureText() | 返回包含指定文本宽度的对象 |

## 八、图像绘制方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| drawImage() | 向画布上绘制图像、画布或视频 chrome不支持 |

### 1、像素操作方法和属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| width | 返回 ImageData 对象的宽度 |
| height | 返回 ImageData 对象的高度 |
| data | 返回一个对象，其包含指定的 ImageData 对象的图像数据 |

|  |  |
| --- | --- |
| 方法 | 描述 |
| createImageData() | 创建新的、空白的 ImageData 对象 |
| getImageData() | 返回 ImageData 对象，该对象为画布上指定的矩形复制像素数据 |
| putImageData() | 把图像数据（从指定的 ImageData 对象）放回画布上 |

### 2、图像合成属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| globalAlpha | 设置或返回绘图的当前 alpha 或透明值 |
| globalCompositeOperation | 设置或返回新图像如何绘制到已有的图像上 |

### 3、其他方法

|  |  |
| --- | --- |
| 属性 | 描述 |
| globalAlpha | 设置或返回绘图的当前 alpha 或透明值 |
| globalCompositeOperation | 设置或返回新图像如何绘制到已有的图像上 |

智能表单的使用和规范

|  |  |
| --- | --- |
| Input表单的type新属性值 |  |
| Type="email" | 限制用户输入必须为Email类型 |
| Type="url" | 限制用户输入必须为URL类型 |
| Type="date" | 限制用户输入必须为日期类型 |
| Type="time" | 限制用户输入必须为时间类型O |
| Type="month" | 限制用户输入必须为月类型O |
| Type="week" | 限制用户输入必须为周类型O |
| Type="number" | 限制用户输入必须为数字类型 |
| Type="range" | 产生一个滑动条的表单 |
| Type="search" | 产生一个搜索意义的表单 配合results="n"属性 C |
| Type="color" | 生成一个颜色选择表单 |

新增的表单属性

|  |  |  |
| --- | --- | --- |
| 属性 | 值 | 描述 |
| required | required | 表单拥有该属性表示其内容不能为空，必填 |
| placeholder | 提示文本 | 表单的提示信息，存在默认值将不显 |
| autofocus | autofocus | 自动聚焦属性，页面加载完成自动聚焦到指定表单 |
| Pattern | 正则表达式 | 输入的内容必须匹配到指定正则 |

css3新增的文本属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| text-overflow | 设置或检索是否使用一个省略标记（...）标示对象内文本的溢出 |
| text-align | 设置或检索对象中文本的对齐方式 |
| text-transform | 检索或设置对象中的文本的大小写 |
| text-decoration | 复合属性。检索或设置对象中的文本的装饰，如下划线、闪烁等 |
| text-decoration-line | 检索或设置对象中的文本装饰线条的位置。 |
| text-decoration-color | 检索或设置对象中的文本装饰线条的颜色。 |
| text-decoration-style | 检索或设置对象中的文本装饰线条的形状。 |
| text-shadow | 设置或检索对象中文本的文字是否有阴影及模糊效果 |
| text-fill-color | 设置或检索对象中的文字填充颜色 |
| text-stroke | 复合属性。设置或检索对象中的文字的描边 |
| text-stroke-width | 设置或检索对象中的文字的描边厚度 |
| text-stroke-color | 设置或检索对象中的文字的描边颜色 |
| tab-size | 检索或设置对象中的制表符的长度 |
| word-wrap | 设置或检索当当前行超过指定容器的边界时是否断开转行 |

text-overflow属性

作用：设定内容溢出状态下的文本处理方式。

取值：

clip： 默认值

当对象内文本溢出时不显示省略标记（...），而是将 溢出的部分裁切掉。

ellipsis：

当对象内文本溢出时显示省略标记（...）。

注意:该属性需要和over-flow:hidden属性(超出处理)还有white-space:nowrap(禁止换行)配合使用，否则无法看到效果

text-align属性

作用:设定文本对齐方式

取值：

left ：默认值 内容左对齐。

center：内容居中对齐。

right： 内容右对齐。

justify： 内容两端对齐。写本文档时仅Firefox能看到正确效果

start： 内容对齐开始边界。（CSS3）

end： 内容对齐结束边界。（CSS3）

text-transform属性

作用：设定文本的大小写等形式的转换

取值：

none： 默认值 无转换

capitalize： 将每个单词的第一个字母转换成大写

uppercase： 转换成大写

lowercase： 转换成小写

full-width： 将左右字符设为全角形式（CSS3）不支持

full-size-kana：将所有小假名字符转换为普通假名（CSS3）不支持

例如：土耳其语。

text-decoration属性

作用：设定文本修饰线。

取值：

[ text-decoration-line ]：不支持

指定文本装饰的种类。相当于CSS1时的text-decoration属性

[ text-decoration-style ]：不支持

指定文本装饰的样式。

[ text-decoration-color]：不支持

指定文本装饰的颜色。

blink： 指定文字的装饰是闪烁。 仅opera和firefox

例如:text-decoration : overline CSS1实例

text-decoration : #F00 double overline CSS3实例

备注：目前主要浏览器都没有实现上述属性，但是依然可以使用CSS1的实例方式

text-decoration-line属性, 备注:目前大部分浏览器未实现该属性。

作用：设定文本修饰线的位置。

取值：

none：默认值

指定文字无装饰

underline：

指定文字的装饰是下划线

overline：

指定文字的装饰是上划线

line-through：

指定文字的装饰是贯穿线

text-decoration-style属性, 备注:目前大部分浏览器未实现该属性。

作用：设定文本修饰线的样式。

取值：

solid：默认值、实线

double： 双线

dotted： 点状线条

dashed： 虚线

wavy： 波浪线

text-shadow属性

作用:设定文本的阴影效果

取值：

none：默认值，无阴影

text-stroke属性

作用:文本边框颜色，指定文字描边部分的颜色

取值:

[ text-stroke-width ]：

设置或检索对象中的文字的描边厚度

[ text-stroke-color ]：

设置或检索对象中的文字的描边颜色

tab-size属性

作用:设定一个tab缩进键，在页面中的显示长度。

取值:　默认值：８

长度或者整数值

整数值 : z-index:1 此处的1就是整数值，不需要单位，类似倍数。

长度: margin:10px 此处的10px是长度值。  
 注意:该属性只在<pre>标签之内(预格式化状态)显示才会有效。因为浏览器会自动忽略空白字符。

opera和火狐浏览器需要使用浏览器私有前缀。

word-wrap属性

作用:溢出文本(特指类英文文字)的处理方式。

取值:

normal： 默认值

允许内容顶开或溢出指定的容器边界。

break-word：

内容将在边界内换行。如果需要，单词内部允许断行。

# 移动web开发重难点

## 一、适配问题解决方案：流式布局 + viewport（重点）

### 1、流式布局

百分比布局，通过设置盒子的宽度为百分比来根据屏幕的大小进行伸缩，特点是不受固定像素的限制，内容向两侧填充

### 2、viewport

在移动端用来承载网页的这个区域，就是我们的视觉窗口，也叫viewport（视口）, 这个区域可设置高度宽度，可按比例放大缩小，而且能设置是否允许用户自行缩放。

<meta name = "viewport">让浏览器知道设置了viewport，viewport只在移动端识别

### 3、viewport 参数

width：定义viewport的宽度，有一个特殊值 device-width当前设备的宽度

initial-scale：默认的初始缩放比

设置1.0，和PC端页面的显示比例一致，一比一显示在移动端

user-scalable：是否允许用户自行缩放 1-yes，0-no

maximum-scale：最大缩放比

minimum-scale：最小缩放比

### 4、构建标准的移动端web开发页面

<meta name = "viewport" content = "width = device-width,initial-scale = 1.0, user-scalable = 0">

快捷键—meta：vp

## 二、在写项目要注意的问题

base样式定义：reset css 重置默认的、浏览器自带的一些样式（所有的标签和伪类），目的是保持各种终端显示一致

### 1、在移动端特殊的设置（了解）

①清楚点击高亮效果

-webkit-tap-highlight-color: transparent;

②设置所有的盒子的宽度以边框开始计算

在移动端通常使用的是百分比布局，那么这样的布局如果使用border或者padding会使容器的宽度超出屏幕的宽度产生滚动条。

-webkit-box-sizing: border-box;

box-sizing: border-box;

③清除input默认的样式

- webkit-appearance: none;

### 2、最小宽度、最大宽度的限制

min-width：300px; //适配小屏幕设备，为了有较好的布局效果。

max-width：640px;//保证页面在尺寸比较大的设备当中保证页面的效果也就是清晰度

### 3、物理像素和图标模糊度问题

解决方案，采用压缩图标尺寸的方式来解决。

如果是Img使用直接设置宽高的方式来压缩。

如果是背景使用的是设置background-size的方式来压缩

### 4、搜索按钮调用

在移动端调用输入法，弹出的小键盘enter键会变成搜索按钮

<form action = "#">

<input type = "search" placeholder = "提示" />

</form>

### 5、结构性伪类原则器---同类型选择器

E:first-of-type匹配同类型中的第一个元素E。

E:last-of-type匹配同类型中的最后一个元素E。

E:nth-of-type(n) 匹配同类型中的第n个元素E。

## 三、Touch事件（重点）

### 1、touchstart：当手指触碰到屏幕的时候触发

绑定事件的方法：dom.addEventListener("touchstart", function(e){});

事件返回的e对象包含那些移动端特有的属性：

targetTouches 目标元素的所有当前触摸

changedTouches 页面上最新更改的所有触摸

touches 页面上的所有触摸/2

### 2、touchmove：当手指在屏幕上滑动时连续触发

绑定事件的方法：dom.addEventListener("touchmove", function(e){});  
 事件返回的e对象包含那些移动端特有的属性：

originalEvent  是jquery 封装的事件。

targetTouches 目标元素的所有当前触摸

changedTouches 页面上最新更改的所有触摸

touches 页面上的所有触摸

在滑动的时候不断给图片盒子做定位，来达到滑动的效果

定位的位置，当前的图片的定位加上移动的距离

在开始滑动的时候就要清楚定时器，move的时候清除过渡，-index\*width + distanceX/3

### 3、touchend：当手指在屏幕上结束滑动时触发

绑定事件的方法：dom.addEventListener("touchend", function(e){});

事件返回的e对象包含那些移动端特有的属性：

changedTouches 页面上最新更改的所有触摸

touchcancel：系统停止跟踪触摸时候会触发，系统被动终止，例如alert。

注意：在touchend事件的时候event只会记录changedtouches

clientX:触摸目标在视口中的X坐标。

clientY:触摸目标在视口中的Y坐标。

pageX：触摸目标在页面中的x坐标。

pageY：触摸目标在页面中的y坐标。

screenX:触摸目标在屏幕中的x坐标。

screenY:触摸目标在屏幕中的y坐标。

## 四、过渡和动画结束时间（掌握）

### 1、transitionEnd：过渡结束后触发。

绑定事件的方法：

dom.addEventListener("webkitTransitionEnd", function(e){});

dom.addEventListener("transitionEnd", function(e){});

### 2、animationEnd：动画结束后触发。

绑定事件的方法：

dom.addEventListener("webkitAnimationEnd", function(e){});

dom.addEventListener("animationEnd", function(e){});

### 3、gesture事件

gesturestart：当一个手指触摸屏幕之后，第二个手指再触摸屏幕时触发。

gesturechange：当上面的事件触发后立即触发。

gestureend：第二根手指离开屏幕时触发，之后将不会再次触发gesturechange。

在event当中会返回另外两个参数

scale 根据两个手指的滑动距离计算的缩放比例 初始1

rotation根据两个手指的滑动距离计算的旋转角度 初始 0

### 全屏单页布局

html,body{height:100%}

## 五、响应式开发（重点）

### 1、响应式布局

就是一个网站能够兼容多个终端。

### 2、响应式开发的原理

CSS3中的Media Query（媒介查询，and之后一定要加空格@media screen and (min-width: 768px) and (max-width: 992px) {}; 通过查询screen的宽度来指定某个宽度区间的网页布局。

超小屏幕（移动设备）768px以下

小屏设备 768px-992px

中等屏幕 992px-1200px

大屏设备 1200px以上

### 3、响应式开发和移动web开发的区别

|  |  |  |
| --- | --- | --- |
| 开发方式 | 移动web开发+PC开发 | 响应式开发 |
| 应用场景 | 一般在已经有PC段的网站，开发移动站的时候，只需单独开发移动端 | 针对新建站的一些网站，现在要求适配移动端，所以就一套页面兼容各种终端，灵活 |
| 开发 | 针对性强，开发效率高 | 兼容各种终端，效率低 |
| 适配 | 只是陪移动设备，pad上体验相对较差 | 可以适配各种终端 |
| 效率 | 代码简洁，加载快 | 代码相对复杂，加载慢 |

### 4、bootStrap（掌握）

bootStrap框架----当前最流行的前端UI框架（有预制界面组件）

特点：组件简洁大方，代码规范精简，界面自定义性强

优点：

有自己的生态圈，不断的更新迭代

提供了一套简洁、直观、强悍的组件

标配准化的html+css编码规范

让开发更简单，提高了开发的效率。

1、布局容器

.container固定宽度并且支持响应式布局的容器

默认margin:55px; padding:15px;

.container-fluid类用于 100% 宽度，占据全部视口（viewport）的容器。

2、删格系统

行：row 通过自身的拉伸来填充父容器的内边距

删格参数：

|  | 超小屏幕 手机(<768px) | 小屏幕 平板(≥768px) | 中等屏幕(≥992px) | 大屏幕 (≥1200px) |
| --- | --- | --- | --- | --- |
| 栅格系统行为 | 总是水平排列 | 开始是堆叠在一起的，当大于这些阈值时将变为水平排列C | | |
| .container最大宽度 | None（自动） | 750px | 970px | 1170px |
| 类前缀 | .col-xs-\* | .col-sm-\* | .col-md-\* | .col-lg-\* |
| 列数(column) | 12 | | | |
| 可嵌套 | 是 | | | |
| 偏移（Offsets） | .col-xs-offset-\* | | | |
| 可见、隐藏 | .visible-xs-\* .hidden-xs | | | |

3、媒体查询

在响应式开发中使用媒体查询中的screen查询浏览器的宽度来定义不同宽度区间的样式和布局。

代码：

@media screen and (min-width: 768px){}

@media screen and (min-width: 992px){}

@media screen and (min-width: 1200px){}

或者

@media (max-width: 767px){}

@media (min-width: 768px) and (max-width: 991px){}

@media (min-width: 992px) and (max-width: 1199px){}

@media (min-width: 1200px){}

4、结构选择器

div a{}

div > a {}

div + div {} 选择目标元素的下一个元素

div ~ div {} 选择目标元素之后的所有同级元素

4、Bootstrap常用样式

1、container类

用于定义一个固定宽度且居中的版心

2、row类

每一个列默认有一个15px的左右外边距，row类的一个作用就是通过左右-15px屏蔽掉它

xs : 超小屏幕 手机 (<768px)

sm : 小屏幕 平板 (≥768px)

md : 中等屏幕 桌面显示器 (≥992px)

lg : 大屏幕 大桌面显示器 (≥1200px)

3、hidden 类

hidden-xs,hidden-sm,hidden-md,hidden-lg在不同的屏幕下隐藏。

4、text-\* 类

text-center 文本居中

text-left 文本左对齐

text-right 文本右对齐

5、pull-\* 类

pull-left 左浮动类

pull-right 右浮动类

6、center-block 类

让一个固定宽度的元素居中。

7、display: table-cell

## 六、rem（重点）

相对长度单位，相对于html根元素font-size计算值得倍数

test布局 //白青衣/rem插件/rem/rem.css

flexible布局//白青衣/rem插件/flexible/rem.css

淘宝这个方案可以在任意设计稿尺寸下使用（地址：https://github.com/amfe/lib-flexible）

1）除font-size外，

其它大小都根据750标注稿的尺寸，转换成rem单位的值，转换方法为：

标注稿尺寸 / 标注稿基准字体大小；

2）标注稿基准字体大小 = 标注稿宽度 / 10，

如标注稿宽为750，标注稿基准字体大小为75；标注稿宽为640，标注稿基准字体大小为64；

em的大小是根据父元素的font-size来设置的

rem的大小是根据html标签的font-size来设置的

标签属性

提供给屏幕阅读器的属性，我们可以忽略

role、 aria-\*、 class="sr-only"

指定插件类型，和控制的目标元素

data-toggle

data-target

a标签也可以通过href来指定被控制的目标元素

轮播图

1、自适应图片：

在pc端使用的是背景图片，在移动端使用的是图片

2、在写静态demo的时候使用的是响应式工具来完成的图片响应，但是会加载两种图片那么这时候就需要做图片的响应式了，首先我们准备了json 中有两种数据，然后ajax请求 到数据做缓存 如果请求过就直接取缓存的，然后通过模版引擎解析成html，最终把解析完成的html渲染在页面当中。

## 七、自定义字体：声明自定义字体的名字，引入字体文件，指定字体文件的格式（掌握）

@font-face{

font-family: "wjs";

src: url(../fonts/MiFie-Web-Font.eot) format("embedded-opentype");

src: url(../fonts/MiFie-Web-Font.svg) format("svg");

src: url(../fonts/MiFie-Web-Font.ttf) format("truetype");

src: url(../fonts/MiFie-Web-Font.woff) format("woff");

}

wjs\_icon{font-family: wjs; }

通过伪类来定义一个字体图标，字体图标设计出来的时候一般都会有文档参考

wjs\_icon\_phone:before{  
 content: "\e908";

}

## 八、 less（掌握）

### 一、简介

1、什么是less[中文网](<http://lesscss.cn/>)

css的预处理语言，LESS 包含一套自定义的语法及一个解析器，用户根据这些语法定义自己的样式规则，这些规则最终会通过解析器，编译生成对应的 CSS 文件。

2、使用less

less文件只有在被编译后才能够被浏览器识别并使用

3、客户端调用方式在.less中@charset "utf-8";

①link引用.less文件 type = "type/less",引用less.js插件,不建议使用

l ess.js 解析less文件动态加载head，异步加载less文件

②直接引用less.css

/\*\*/会编译在css文件当中

//不会编译在css中

### 二、语法（了解）

1、变量

LESS 允许开发者自定义变量，变量可以在全局样式中使用，使得样式修改起来更加简单。

@mainColor: #E93223;

body{color: @mainColor}

2、Mixin混入

Mixin（混入）特性，它是多重继承的一种实现，在 LESS 中，混入是指在一个 CLASS 中引入另外一个已经定义的 CLASS，就像在当前 CLASS 中增加一个属性一样。

.red{color: @mainColor};

.border{border: 1px solid #ccc}

////方法混合.redBorder(){color: @mainColor;border: 1px solid #ccc}

样式混合（通过class）：.mixin-class{.red();.border();}

3、嵌套

在我们书写标准 CSS 的时候，遇到多层的元素嵌套这种情况时，我们要么采用从外到内的 选择器嵌套定义，要么采用给特定元素加 CLASS 或 ID 的方式

#wjs\_banner{

>div.item{

img\_box{  
 redBox();

&:hover{

color:@mainColor;

}

}

}

}

4、导入Import

@import "base";//引入其他的less文件

.f\_left{float: @right}

5、运算及内置函数

在我们的 CSS 中充斥着大量的数值型的 value，比如 color、padding、margin 等，这些数值之间在某些情况下是有着一定关系的，那么我们怎样利用 LESS 来组织我们这些数值之间的关系呢？

<http://www.1024i.com/demo/less/reference.html>

运算：

@back:#333;

.test{

border: 1px solid @back\*2

}

内置函数：

desaturate(@color, 10%);

lighten(@color, 10%);

darken(@color, 10%);

fadein(@color, 10%);

fade(@color, 10%);

## 移动端面试题

**1、能否简述一下如何使一套设计方案，适应不同的分辨率，有哪些方法可以实现？**

流式布局：也就是百分比布局(viewport)， 使用非固定像素来定义网页内容，通过盒子的宽度设置成百分比来根据屏幕的宽度来进行伸缩，不受固定像素的限制，内容向两侧填充。

响应式开发： 就是一个网站能够兼容多个终端。CSS3中的Media Query（媒介查询)通过查询screen的宽度来指定某个宽度区间的网页布局。 由于响应式开发显得繁琐些，一般使用第三方响应式框架来完成，比如bootstrap来完成一部分工作。

**响应式开发和移动web开发的区别**

|  |  |  |
| --- | --- | --- |
| 开发方式 | 移动web开发+PC开发 | 响应式开发 |
| 应用场景 | 一般在已经有PC段的网站，开发移动站的时候，只需单独开发移动端 | 针对新建站的一些网站，现在要求适配移动端，所以就一套页面兼容各种终端，灵活 |
| 开发 | 针对性强，开发效率高 | 兼容各种终端，效率低 |
| 适配 | 只是陪移动设备，pad上体验相对较差 | 可以适配各种终端 |
| 效率 | 代码简洁，加载快 | 代码相对复杂，加载慢 |

**2、响应式布局的时候，轮播图使用两张不同的图片去适配大屏幕和超小屏幕，还是一张图片进行压缩适配不同终端，说明原因？**

最好使用两张不同大小的图片去适配大屏幕和超小屏幕，这样可以针对不同设备的屏幕大小，来加载响应的图片，减少超小屏幕设备的网络流量消耗，加快响应速度，同时防止图片在大屏幕下分辨率不够导致失真的问题。

**3、Bootstrap中最多可以分多少列？lg、md、sm、xs这几个屏幕宽度的界限是多少？**

答：12列 .col-xs- 超小屏幕手机 (<768px) .col-sm- 小屏幕平板 (≥768px) .col-md- 中等屏幕桌面显示器 (≥992px) .col-lg- 大屏幕大桌面显示器 (≥1200px)

**4、移动端事件与pc端有什么区别？**

移动端新增了触摸事件。没有鼠标事件和hover事件。 另外包括移动端弹出的手机键盘的处理，这样的问题在PC端都是遇不到的。/5、

**5、zepto与jquery有什么区别？**

Zepto中没有为原型定义extend方法而jQuery有。

Zepto新添了touch事件，zepto是模块化的库

Zepto是不支持IE浏览器的

**6、怎么适配各种机型？**

设置viewport进行缩放、响应式做法、流式布局、rem能等比例适配所有屏幕

# 面向对象

## 一、面向对象的基本概念

js 是一个基于对象的多范式的编程语言

多范式: 编程风格

面向过程的方式编程

面向对象的方式编程

函数式 ...

-> 函数式的一个特点: 递归与链式

jQuery 就是典型的链式编程风格

$( 'div' ).css( 'border', '1px solid red' )

.css( 'width', '400px' )

什么是面向对象

### 1、面向对象的概念

面向：将脸朝向......-> 关注，用

面向过程：关注编程步骤，细节的编程方式（自己动手完成）

面向对象：关注对象，找到对象，让它帮你做，等结果

面向过程就是亲力亲为, 事无巨细, 面面俱到, 步步紧跟, 有条不紊

面向对象就是找一个对象, 指挥得结果

面向对象不是面向过程的替代, 而是面向过程的封装

### 2、对象是什么

A.对象是存储数据的数据集，是提供功能的功能集，对象可以用来保存数据

B.使用对象来添加一些特定的功能

var itcast = { tag:function(){}, addCss:function(){}}

使用方法： itcast.tag(参数)

### 3、在js中，对象就是键值对的集合，名词

### 4、使用面向对象编程（实际开发的时候如果需要完成一个功能）

①首先考虑系统是否提供了对象（使用现有对象）

document, 标签对象

②如果系统没有提供，就自己创建一个对象或者第三方（自定义对象）

注意：函数==>使用js代码编写的一个普通的function就是函数

方法==>使用对象添加的功能（函数）就是方法

## 二、面向对象的特性

1、抽象性：抽取我们所需要的数据信息

2、封装性：留下一些能使用的特性

3、继承性

## 三、值类型与引用类型

数字 + 上下文 = 信息

内存就是一个可以存储数字（数据）的盒子

### 1、JavaScript 中的数据类型

A基本类型（值类型）: 数字, 布尔, 字符串

B复合类型:（引用类型 : 对象( 数组, 时间, 正则, Object , 函数等 )

C空类型: Undefined, Null

复合类型除函数其它的无法通过typeof获得数据类型，要使用Object.prototype.toString.apply()

### 2、数据存储模型:“盒子” 与数据的逻辑结构

A.基本数据类型和空类型的存储模型：一个方格里面放一个数据

B.复合类型的存储模型：

复合类型的对象是一个单独的内存区域, 对象有什么属性, 那么内存区域中就有什么数据。变量 p 只存储该对象的 '地址'. 所以 p 不是真正存储数据的区域.

### 3、值类型与引用类型的存储特征（掌握）

①值类型的数据, 只需要开辟一段内存存储数据即可

var a = 123; var b = 'abc'; var c = true;

②对象类型( 引用类型 ). 对象才是真正的数据, 需要占据单独的内存.

而变量名只是存储着对象的内存地址( 引用 ).

即创建一个对象并赋值, 实际上需要两块内存空间. 一个存储数据( 对象 ),

另一个存储变量名以引用对象.

### 4、值类型与引用类型的赋值与传参的特性

-> 赋值: 将原变量中的数据拷贝一份, 然后赋值给新的变量中

① 值类型//拷贝的是实际的数据

② 引用类型//把变量的引用地址拷贝一份给新的变量，此时新的变量旧的变量都是指向 了同一个对象

var o1 = { num: 123 };

var o2;

// 赋值

o2 = o1; // 将 o1 中存储的内容 拷贝一份 存储到 o2 中

// o1 中存储的是引用, 或 '地址'

o1 和 o2 虽然是不相同的两个变量, 即两个独立的盒子. 但是由于存储的地址相同.

在访问数据的时候, o1 与 o2 也是访问同一个数据, o1 将数据修改了, o2 再读取,

读取的就是修改后的数据.

-> 函数参数传递

-> 什么是函数参数传递

在调用函数的时候, 会将参数中表示的数据拷贝一份, 然后给参数赋值

function foo( num ) {}

// 调用

var a = 123;

foo( a ); // 调用函数就是要执行函数// 将 a 中存储的数据 拷贝一份

// 进入 函数 foo// 给参数赋值, 相当于 num = a

// 进入函数体, 开始执行函数

总结：值类型引用类型的数据的传参

参数是值类型，函数中对参数的任何改变都不会影响到函数

参数是引用类型，函数中对参数对象的改变会影响到函数外的实参

### 5、 深拷贝与浅拷贝

-> 什么是拷贝: 就是创建一个与目标数据一模一样的数据

-> 案例:

var p = { name: '张三' };

var p1 = p; // 是否存在拷贝

// 一般描述拷贝是指拷贝对象

p1 = {};

p1.name = p.name;

// 才是拷贝

-> 案例, 给 对象 p 提供一个 clone 方法, 完成拷贝

-> 案例:

有一辆汽车 Car: name=保时捷

有一个人 Person : name=张三

-> 如果对象的属性也是一个引用类型, 拷贝的时候不重新创建一个新的对象来实现该属性的拷贝, 那么就是浅拷贝.

即任何不完全的拷贝都是浅拷贝

-> 将两个对象完全从内存中隔离开, 就是深拷贝. 即每一个引用属性, 以及引用属性的引用属性, ... 全部拷贝出来.

## 四、构造函数( 构造器 controctor )的作用（重点）

### 1、js 中对象的动态特性

动态的为对象添加属性

### 2、方式：点语法与关联数组语法

o.name = 'jim'; // 点语法赋值

console.log( o.name ); // 点语法取值

o[ 'name' ] = 'tom'; // +关联数组语法赋值

console.log( o[ 'name' ] ); // 关联数组语法取值

问题:o[ name ] = 'jack'; // 如果 name 是一个变量, 里面存储的是 字符串, 也是可以的

### 3、例创建一个 Person 对象

var p = {}; // 什么都没有的对象

p.name = '张三';

p.age = 30;

p.gender = '男';

-> 简化: 提供一个创建 Person 对象的函数

function createPerson( name, age, gender ) {

var p = {};

p.name = name;

p.age = age;

p.gender = gender;

return p;

}

var p1 = createPerson( 'jim', 19, '男' );

var p2 = createPerson( 'lily', 18, '女' );

这个( 这种类型 )的函数就是用来创建对象的, 即生产对象. 常常将这类函数

称为 '工厂函数'

工厂方法：

工厂方法就是一个函数，这个函数就是用来创建并返回一个对象，而且每次创建和返回的对象的特征都是一样的，像流水线下来的一样，这样的“流水线作业生产对象”函数叫做工厂方法。

### 4、构造方法创建对象：（构造器：用来创建对象的一个函数）

-> 构造器中不需要 return 语句. 一般也可以不写

-> 调用构造器的时候, 使用 new 运算符引导

-> 在构造器中 this 表示当前“new”出来的对象，给对象提供成员使用 this.xxx 的 方式

-> 将 createPerson 改造成构造器

// 构造器的定义、结构

function 构造函数名（参数）{

this.属性 = 属性值；

}

var p = new 构造函数名（）；

function createPerson( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

// 调用构造器创建对象

var p = new createPerson( '李磊', 19, '男' );

-> 构造器创建对象的本质: 还是使用对象的动态特性

-> 首先执行 new 运算符. 即创建对象. 可以简单看做为 {} 的简写

var p = new ... '相当于' var p = {}

-> 调用构造器. 并隐含一个参数, 即刚刚创建的对象.

-> 在构造器中使用 this 引用刚刚创建出来的对象.

-> 构造器结束是 默认返回 this

-> 补充说明

-> 构造器的名字, 一般使用 Pascal 命名规则( 首字母大写的 )

-> 一般为了与其他面向对象的编程语言( C++, C#, Java )保持一致. 称构造函数名为类名

function Person( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

// 调用构造器创建对象

var p = new Person( '李磊', 19, '男' );

## 五、 异常与捕获（了解）

-> 异常

简单的说, 就是代码在执行过程中出现的错误. 并不是代码的语法错误.

-> 一旦出现异常, 其后的代码, 不再执行

-> try-catch 语法

1) try-catch 形态

try {

有可能出现错误的代码

} catch ( e ) {

处理宠物的异常

}

2) try-catch-finally 形态

finally {

不管错误会不会发生，都会执行的代码

}

无论是出现异常，try语法结束的时候都

-> 自定义抛出异常

一般可以封装函数完成特定的功能. 例如 tag 函数

function tag ( tagName ) {

if ( tagName 不是字符串 ) 抛出异常.

return document.getElementsByTagName ( tagName );

}

-> 抛出异常的语法

throw 对象

## 六、原型

### 1. 为什么需要原型：减少内存的浪费，提高效率，使代码更加安全

构造器创建对象的时候, 实际上会有成员重复

如果使用 构造器 this.方法名 = function .... 方式创建对象. 那么每一个对象

对应的方法就会重复.

解决办法就是让这个方法( 函数 )共享

①将函数写到外面, 那么 Person 在初始化对象的时候就不会再创建一个函数了.

只需要将 外面的函数引用 交给对象即可.

缺点: 一个对象可能有 n 多方法. 如果将所有的东西 都放到外面, 与其他库

冲突的几率就会变大. 所以不宜采取该方法.

②将所有的方法( 函数 )都绑定到一个对象中.

③js 原生就支持解决该问题的办法：使用prototype属性

每一个函数都有 一个属性 prototype，该属性指向一对象（Fun.prototype）.

(重点) 每一个由该函数作为构造器创建的对象, 都会默认连接到该对象上.

如果访问对象的方法, 而对象中没有定义, 就会在这个构造函数.prototype

表示的对象中去找.

-> prototype 就是原型之意

原型就是被克隆的人，构造函数是克隆的机器，实例对象是克隆体

### 2. 原型就是 构造函数的 prototype 属性, 常常将其称为 原型属性.

原型就是 实例对象的 原型对象，实例对象是由原型对象复制而来的一个对象

例:

function Person () {} // 有了构造函数, 和 原型

var p = new Person(); // 有了实例

### 3. 一般如何使用原型对象

-> 简单的说就是将共享的方法放到原型中, 而独有数据与行为放在当前对象里

-> 例:

Person( name, age, gender, sayHello, eat, run )

-> 通过动态特性接给原型对象添加成员:这种方式添加的话，原型依然会有constructor的属性，并且指向原来的构造函数

-> 通过Person.prototype指向一个新对象替换旧原型对象：原型里面没有constructor属性，需要手动去添加这个属性（constructor:函数名） 属性, 表示对应的构造函数 )

constructor：构造器，原型的

原型属性--函数

原型对象--实例对象

### 4. \_\_proto\_\_：用来访问原型的一个属性

-> 访问

使用构造函数, 就使用 prototype 属性访问原型

使用实例对象, 就使用非标准的 \_\_proto\_\_ 属性访问原型(对象名.constructor.prototype)

## 七、 继承（掌握）

-> 什么是继承

自己没有, 别人有, 拿过来自己用, 就好像自己的一样.

-> 原型与实例对象

在 js 中, 方法定义在原型对象中, 而属性定义在实例对象中

调用方法的时候, 实例对象本身是没有该成员的, 但是依旧可以调用

该方法, 好像这个方法就是该实例对象的一样. 因此, 我们称该实例对象

继承自 原型对象

-> 任何一个实例, 都是继承自其原型对象的. 即原型式继承.

### 1、为什么需要继承

-> 编程的发展

复用( 重复使用 )

div 标签对象 nodeName, nodeType, nodeName, ...

appendChild, insertBefore, getElementsByTagName, ...

a 标签对象

baseElement

-> js 运行效率

共享特性

复用

### 2、 属性访问原则( 重点 )

1) 对象在调用方法或访问属性的时候, 首先在当前对象中查询. 如果有该成员使用并停止查找.

2) 如果没有该成员就在其原型对象中查找. 如果有该成员即使用, 并停止查找.

3) 如果还没有就到 该对象的 原型对象 的 原型对象中查找.

...

4) 最后会查到 Object.prototype 上. 如果还没有即 返回 undefined.

### 3、 如果修改原型对象中的属性值会怎样

给当前对象的原型提供的属性赋值, 实际上是给当前对象添加了该属性的新成员

并不会修改运行对象中的成员.

### 4、混入( mix ): 将一个对象中的成员加到另一个对象中

var o1 = { name: '张三' };

var o2 = { age: 19 };

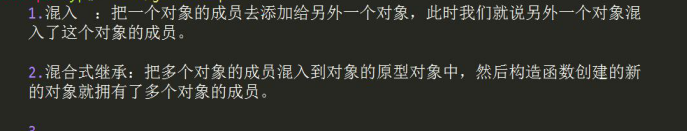
-----------

o2.name = o1.name

利用这个简单的赋值, 就可以

混入使得 o2 具有了 age 和 o1 中的 name. 即将 o1 混入到 o2 中

混入也是一种继承的方式



### 5、如何使用代码实现混入

// 考虑需要执行, 写函数即可

// 由于将 一个 对象混入到 另一个对象. 所以有两个参数

私有

function \_\_mix\_\_ ( obj(原对象), obj1(新对象) ) {//将obj1的成员加入到obj中去

// 如何获得 obj1 中的所有成员?

for ( var k in obj1 ) {

obj[ k ] = obj1[ k ];

}

}

在函数中有一个默认的对象arguments存在，它里面存储着所有传入的参数

我的extend方法可以带多个参数，如果是多个参数，要求将所有对象的成员都加到this当前对象上

argument对象：这个对象在每一个函数里都有，代表的就是函数的所有参数，是一个数组

var o = {

extend: function(obj){

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

for (var k in arguments[i]){

this[k] = arguments[i][k];

}

}

}

}

### 6、 原型式继承

-> 写一个构造函数, 如果需要将其实例继承自某个特定的对象 o. 那么

只需要设置该构造函数的 prototype 属性为 o 即可

function Person (){}

var o = { ... }

Person.prototype = o; // 继承

### 7、 混合式继承

-> 将多个对象的各个功能混合到一起, 加到构造函数的原型对象上，这些功能是从原型继承而来，所以叫做混合式继承

步骤：

A：给原型添加一个extend方法

function 构造函数 (){}

构造函数.prototype.extend = function(obj){

for (var k in obj){

this[k] = obj[k];

}

};

B.调用extend方法去混入多个对象的功能

构造函数.prototype.extend{

(多个对象)

方法: function(){}

}; //此时原型中就会混入多个对象的功能了

C.使用构造函数创建对象

var test = new 构造函数();

## 八、原型链（重点）

原型链结构：实例对象 -> 对象的原型对象 -> object的原型对象 ->null 链式结构( 原型链 )

### 1.新对象 Object.create( 原对象 );（了解）

以对象为原型创建一个新的对象，用在只有实例对象的时候

var newObj = Object.create(oldObj)

低版本兼容：先判断有没有，如果没有再加上

function createWithObject (obj) {

if (Object.create) {

return Object.create(obj);

}else{

function F(){}

F.prototype = o;

return new F();

}

}

### 2. 对象的原型链

-> 凡是对象都有原型

-> 构造函数 Person 创建的对象 p 有原型 Person.prototype

-> Person.prototype 的原型对象是Object.prototype

-> 结论

1) Person.prototype 是 实例 p 的原型对象, 使用 \_\_proto\_\_ 也可以访问对象的原型对象

2) Person.prototype 的 原型对象是 Person.prototype.\_\_proto\_\_

3) Person.prototype.\_\_proto\_\_ 里的 constructor 是 Object. 所以

Person.prototype.\_\_proto\_\_ 就是 Object.prototype

4) Object.prototype.\_\_proto\_\_ 是 null. 因此表明 Object.prototype 就是顶级.

-> 链式

p --> Person.prototype( p.\_\_proto\_\_ ) --> Object.prototype --> null

-> 系统内置的原型链

[] --> Array.prototype --> Object.prototype --> null

/./ --> RegExp.prototype --> Object.prototype --> null

... ...

### 3. 绘制数组的原型结构

var arr = [];

// 等价

var arr = new Array();

arr --> Array.prototype --> Object.prototype --> null

### 4. 练习:

根据下面代码绘制对象的原型链结构

1) function Person( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

function Student () {}

Student.prototype = new Person( '张三', 19, '男' );

var stu = new Student();

2) function Person() {}

var p1 = new Person();

var p2 = new Person();

### 5. {} 对象的原型链结构

在 js 中 对象 一般都有字面量

123, '123'

数组: []

正则表达式对象: /./

函数: function () {}

...

对象也有字面量: {}

{} --> Object.prototype --> null

注意: {} 与 new Object() 含义相同

### 6. 动态函数 Function

-> 动态函数就是在运行的过程中, 将一段字符串作为代码运行.

由于字符串可以随意的拼接. 因此得到动态的执行.

-> 定义动态函数, 并执行

-> 使用 Function 构造函数, 创建函数.

Function 是一个构造函数. new Function 得到 一个函数

-> 语法

var demo = prompt("");

var demo2 = new Function( arg0, arg1, ..., argN, body )

demo2();

Function 的所有的参数, 除了最后一个以外, 都是生成的函数的参数

最后一个参数是 函数体

### 7. 函数的相关的一些参数

①arguments对象：参数对象，保存了调用函数时传入的所有参数，使用数组的索引访问参数.

例如: 写一个函数, 在参数中写任意个参数, 最后求其和

function sum () {

// 所有的参数都会存储到 arguments 中

var sum = 0;

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

sum += arguments[ i ];

}

return sum;

}

使用arguments改写extend函数

函数名.prototype.extend = function(){

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

for(var k in arguments[i]){

this[k] = arguments[i][k];

}

}

}

②函数名.length, 即函数的 length 属性. 表示 定义函数时, 参数的个数

如果定义函数的时候, 定义了参数. 但是调用的时候又没有传递该参数. 那么该参数在函数内就是 undefined

③函数.name 返回的是函数名

### 8. 函数的引用 callee 与 caller（掌握）

js 中函数也是一个对象

-> callee被调用者 在函数的内部, 实现函数递归的时候，一般使用callee表示 当前函数 的引用，

function fn(){

arguments.callee()

}

fn();

-> caller 表示调用函数，调用者

### 9. eval 函数（立即执行函数，自调用函数）通过ajax获取请求，获得数据

eval 函数与 Function 功能类似. eval 可以直接将字符串作为代码来执行.

语法:

eval( 语句字符串 )，在eval函数中，使用字符串声明的变量，在Evelyn函数外面可以立即执行

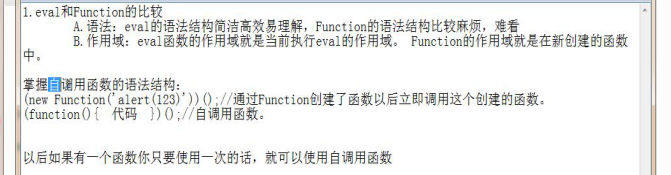
注意, 它好像与当前代码处于同一个作用域

语法上：eval 语法简洁，直接调用即执行

### Function

创建了一个新的，所有的东西，都只在函数内部起作用，如果要执行，需要调用

除非：1）变量没有声明，直接使用②return



将json字符串变成对象，三种做法

eval做法 var o1= eval（"("+data+")"）;（必须加（）），用来执行jsdaima 的，所有，json字符串中的{}读作了代码快的开始和结束，所有一定要加上（）；

Function 做法var o2 = (new function('return'+data))();

3）var o3 = JSON.parse（dara）; 低版本的浏览器可能不兼容，要求json字符串是严格格式字符串（键必须用引号引起来）

### 11. 在 js 中 函数 是 Function 的实例

function Person() {}

var p = new Person();

p 是 构造函数 Person 的实例

在 该角度去看, 函数就是对象, Function 就是构造函数

得到 构造-实例-原型 三角形

函数是一个函数对象，认为是一个实例对策function Person(){}

Function是函数的构造函数，，由Function创建函数实例对象，Function是自己的实例，任意一个函数是它的对象，是函数对象创建了自己

相当于：Person =



function Person() {} 为例

### 12. 给定一个任意的对象, 将其原型链包括对象和函数的结构完整的分析处理

### 13. instanceof 运算符

-> a instanceof b -> 判断b是否在a的原型链上

-> instanceof ?

-> 错觉: 判断某一个对象是否为某一个构造函数所创建出来的

语法：

对象 instanceof 构造函数（结果是布尔值）

判断构造函数的原型属性是否在对象的原型链上

### 14. 通过原型链的学习, 可以重新定义 js 的继承

js 的继承: 就是利用对象的动态特性添加成员, 或直接替换对象的方式修改原型链结构. 使得当前对象的原型链上的对象具有某些成员. 那么我的当前对象就可以使用这些成员了.

p -> Person.prototype -> Object.prototype -> null

p -> Person.prototype -> {} -> Object.prototype -> null

过多的依赖原型链继承, 会损耗 性能

如果必须使用原型链继承, 最好提供一些快速访问的方法

加入我希望所有的函数都具有 inherit 功能

function Person() {}

var Student = Person.inherit( 成员 );

// 生成函数 Student, 但是 Student 里面的成员( this.XXX )

// 由参数来提供, 而 Student 的对象, 继承自 Person 的对象

如果希望所有的函数都有 inherit 方法, 那么可以给 Function.prototype 添加该方法

## 九、代码的预解析（掌握）

-> 预解析：在代码执行之前先把代码解释分析一遍，处理一些核心问题，为了提高效率

提前的翻译解释, 在运行代码之前的一个解释.

-> 为什么需要它

-> 编译型语言: C, C++, C#, Java

就是需要一个 "翻译" 程序, 将源代码翻译成计算机可以读懂的二进制数据( 指令 ).

然后存储成可执行文件.

提前翻译好, 运行时直接执行得结果

-> 解释型( 脚本型 ): JavaScript, SQL, ...

代码在执行的时候, 有一个翻译程序, 读一句代码执行一句代码. 再读一句代码,

再执行一句代码.

一句一句的翻译执行. 每次运行都需要翻译一次.

-> 代码在执行之前, 需要快速的 "预览" 一遍. 那么可以尽可能提高执行效率.

### 1. 在 js 中预解析的特点

-> 代码是如何执行的: 读取 js 文本, 预解析, 一句一句地执行

-> js 在预解析的过程中完成了声明部分的标记与变量作用域的设定

### 2. 什么是 js 中的声明

-> 简单的说就是让 js 执行引擎知道有什么东西( 标识符 )

console.log( num ); // error: num is not defined

num(); // error: is not function

即代码在执行之前的预解析, 首先让 js 的执行引擎知道在当前运行环境中

有什么东西( 名字, 标识符 )是可以被使用的. 它是变量, 还是函数等?

-> 在 js 中有哪些声明:

1) 标识符的声明( 变量的声明 )

2) 函数的声明

-> 变量的声明:

语法: var 变量名;

目的: 告诉解释器, 有一个名字是一个变量, 在当前环境中可以被使用.

语句: 就是可以执行的东西.

var a = 123; 是一个语句

在使用 var 声明变量, 同时完成赋值的时候. 实际上, 预解析将其做了一定处理:

-> 凡是读取到 var 的时候, 就检查 var 紧跟的名字是否已经标记了

-> 如果没有标记, 就表明这个名字是一个标识符, 需要被标记

-> 如果已经被标记了, 那么 这个 var 被忽略

结论:

var a;

var a = 10;

等价

var a;

a = 10;

var a = 123;

var a = 456;

var a = 789;

等价于

var a = 123;

a = 456;

a = 789;

如果在代码中有多个 var 后面紧跟的名字是一样的. 那么只有第一个 var 起作用.

后面的所有 var 都会被自动的忽略.

-> 变量名提升

代码再预解析的时候会去关注var声明的变量，函数的声明，标记这个变量或者函数

-> 函数的声明

-> 函数的各种定义形式

-> ①函数声明式:

function func () {

console.log( '使用声明式定义' );

}

-> ②函数表达式式( 匿名函数, 字面量函数, lambda 函数 ):

var func = function () {

console.log( '使用表达式式定义' );

};

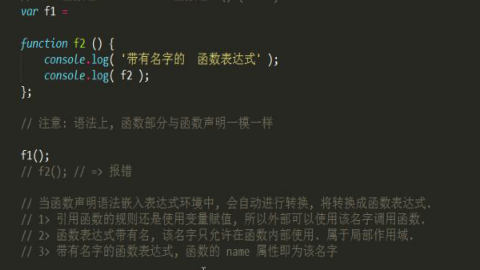
func();

使用这个方式定义函数，实际上是利用函数是js中的一个数据类型的特点，利用赋值，使用变量存储函数的引用，此时没有函数的声明，但是有变量的声明

--> ③带有函数名的函数表达式



f2外部无法使用只能在函数内部使用，如果要调用改函数要使用f1()调用，f2也是函数的name属性值



-> ...

-> 特点:

1> 函数的声明是独立于语句. 不需要加分号结束. 也不能嵌入到代码表达式中.

2> 表达式式, 本质上是使用函数表达式( 字面量 )给变量赋值. 因此它是语句.

-> 表达式:

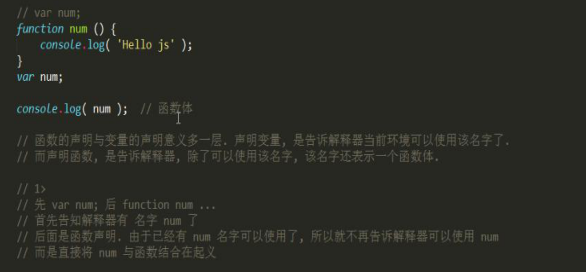
-> 将运算符与操作数连接起来的式子.

-> 就是一个有结果的代码单元( 不包括语句 )

var a; // 声明, 不是语句, 也没有结果

123 // 字面量, 有值, 是表达式. 是常量表达式

a = 123 // 赋值, 有值, 就是被赋值的那个值. 是赋值表达式.





function () {}

函数名提升：代码预解析的时候会去找到声明的函数，提前把函数放在作用于里面

### 3. 如果将变量的声明与函数的声明放在一起有些需要注意的情况

1) 函数的声明实际上包含两部分

1> 告诉解释器 xxx 名字已经可以使用( 函数名, 标识符 )

2> 告诉解释, 这个名字代表着一个函数( 变量里存储着函数的引用 )

2) 当函数声明与变量声明冲突的时候. 只看谁先有数据.

### 4. 一个在新版本的浏览器中的特性

if ( true ) {

function foo() {

console.log( true );

}

} else {

function foo() {

console.log( false );

}

}

foo();

函数声明不要放在代码块里面 吧

### 5. 词法作用域

-> 作用域: 就是变量可以使用到不能使用的范围

-> 块级作用域:

-> 块: 代码块, 即 { }

-> 变量的使用从定义开始, 到其所在的块级作用域结束

-> js 是词法作用域

在js中，只有函数可以改变作用域的范围

-> 词法: 就是定义, 书写代码的规则.

-> 所以 所谓的 词法作用域, 就是 在书写代码的时候, 根据书写代码的结构

就可以确定数据的访问范围的作用域.

-> js 不受 块的影响, 即使在块中定义声明变量, 在块的外面依旧可以使用

console.log( num ); // => undefined

{

var num = 123;

}

console.log( num ); // => 123

-> 所谓的 js 的词法作用域, 就是

①在没有函数的情况下，所有的变量的访问规则依据预解析规则

②只有函数可以限定范围. 其他的不行.

③在函数内部是一个独立的完整的作用范围结构.

④在函数内部，允许再定义函数，同时两个层次的函数都是作用域的独立体

⑤在函数内优先访问内部声明的变量, 如果没有才会访问外部的.

⑥允许在函数内访问函数外的变量，前提是函数内没有改变量的声明

-> 结论:

词法作用域就是描述变量的访问范围:

1> 在代码中只有函数可以限定作用范围. 允许函数访问外部的变量. 反之不允许.

2>

3> 所有变量的访问规则, 按照预解析规则来访问

### 6. 案例1

var num = 123;

function f1 () {

console.log( num );

}

function f2 () {

console.log( num );

var num = 456;

f1();

console.log( num );

}

f2();

1> 读取代码预解析. 得到 num, f1, f2

2> 逐步的执行代码

1) 赋值 num = 123; 注意 f1 和 f2 由于是函数, 所以也有数据.

2) 调用 f2.

进入到函数体内. 相当于做一次预解析. 得到 num. 注意, 此时有内外两个 num

执行每一句代码

-> 打印 num. 因为函数内部有声明 num. 所以此时访问的是函数内部的 num. 未赋值, 得到 undefined

-> 赋值 num = 456

-> 调用 f1(). 调用函数的规则也是一样. 首先看当前环境中是否还有函数的声明. 如果有直接使用. 如果

没有, 则在函数外面找, 看时候有函数. 此时在函数 f2 中没有 f1 的声明. 故访问的就是外面的 f1 函数

-> 跳入 f1 函数中. 又要解析一次. 没有得到任何声明.

-> 执行打印 num. 当前环境没有声明 num. 故在外面找. 外面的是 123. 所以打印 123.

函数调用结束, 回到 f2 中.

-> 继续执行 f2, 打印 num. 在 f2 的环境中找 num. 打印 456.

任务:

var num = 123;

function f1 () {

console.log( num );

}

function f2 () {

console.log( num ); // => 123 , 456, 456

num = 456;

f1();

console.log( num );

}

f2();

### 7. 案例2

(function ( a ) {

console.log( a );

var a = 10;

console.log( a );

})( 100 );

拆解

( 函数 ) ( 100 )

第一个圆括号就是将函数变成表达式

后面一个圆括号就是调用该函数

var func = function ( a ) {

console.log( a );

var a = 10;

console.log( a );

}

func( 100 );

注意: 函数定义参数, 实际上就是在函数最开始的时候, 有一个变量的声明

function ( a ) { ... }

其含义就是, 在已进入函数体, 在所有操作开始之前( 预解析之前 )就有了该变量的声明.

由于已经有了 a 参数的声明. 所以在代码中 var a = 10 是重复声明. 其声明无效.

所以上面的代码, 等价于

var func = function ( a ) {

console.log( a ); // => 100

a = 10;

console.log( a ); // => 10

}

func( 100 );

// 变式

(function ( a ) {

console.log( a );

var a = 10;

console.log( a );

function a () {

console.log( a );

}

a();

})( 100 );

1> 直接调用

2> 进入到函数中, 已有声明 a 并且其值为 100

3> 在函数内部预解析. 得到 一个结论. 函数声明是两个步骤.

1) 让当前环境中, 有变量名 a 可以使用. 但是不需要. 因为已经有 a 的声明了

2) 让 a 指向函数. 相当于

var a;

function a () {}

...

4> 开始逐步执行每一句代码

1) 打印 a. 所以打印函数体

2) 赋值 a = 10

3) 打印 a, 打印出 10

4) 如果让 a 调用, 那么报错 error: a is not function

### 8. 作用域链规则

-> 什么是作用域链

链指的就是访问规则

function foo() {

console.log( num );

}

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

function func () {

function foo() {

console.log( num );

}

foo();

}

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

function F () {

function func () {

function foo() {

console.log( num );

}

foo();

}

func();

}

... ...

由于这种一环套一环的访问规则, 这样的作用域构成一个链式结构. 所以直接称其为作用域链.

-> 作用域链是用来做变量查找的. 因此变量可以存储什么东西. 链中就应该有什么东西.

换句话说就是, 链里面存储的是各种对象. 可以将其想象成对象的序列( 数组 )

### 9. 绘制作用域链的规则

1> 将所有的 script 标签作为一条链结构. 标记为 0 级别的链.

2> 将全局范围内, 所有的声明变量名和声明函数名按照代码的顺序标注在 0 级链中.

3> 由于每一个函数都可以构成一个新的作用域链. 所以每一个 0 级链上的函数都延展出 1 级链.

4> 分别在每一个函数中进行上述操作. 将函数中的每一个名字标注在 1 级链中.

5> 每一条 1 级链中如果有函数, 可以再次的延展出 2 级链. 以此类推.

### 10. 分析代码的执行

当作用域链绘制完成后. 代码的的分析也需要一步一步的完成.

1> 根据代码的执行顺序( 从上往下, 从左至右 )在图中标记每一步的变量数据的变化

2> 如果需要访问某个变量. 直接在当前 n 级链上查找变量. 查找无序.

3> 如果找到变量, 直接使用. 如果没有找到变量在 上一级, n - 1 级中查找.

4> 一直找下去, 知直到 0 级链. 如果 0 级链还没有就报错. xxx is not defined.

## 十、闭包（重点）

### 一、相关概念

1、闭包：闭包就是一个封闭的隔离的空间。js中：函数所构成的外部无法直接访问的区域

function foo () {}

闭包的本质：利用作用域访问规则的不可逆性，构成一个单向的空间。

为什么要学习闭包

因为有一些数据是比较敏感的，会有安全的需求，不能让用户得到，所以将它们放到闭包里面，让用户更加安全的访问。

### 二、闭包的间接访问--访问闭包中的数据

1、访问闭包中数据的方法：

在函数内部新定义一个函数，在这个新函数中return返回我们需要的原始数据，然后return返回这个函数。// 返回一个对象

2、案例：

function foo () {

var num = 123; // 原始数据

function func () {

return num; // 它就是原始数据 num

}

return func;

}

var f = foo(); // f中存储的是foo里面定义的函数的引用，即，可以用f来调用里面的函数

凡是调用一次f，就是调用一次foo中定义的函数，就会利用变量的词法作 用域规则返回foo中的num，即间接访问了闭包中的数据。

//foo只能调用一次，可以创建一个原始数据，但是声明的函数f可以重复调 用，每一次调用都可以获得闭包中的数据的值

### 三、函数是对象--可以作为一般的数据来使用

函数是基本的对象类型，可以作为变量赋值，可以作为参数使用，也可以作为返回值使用

函数返回函数：return 函数名；

赋值

function foo() {

......（函数体）

}

var func = foo; //函数体只有一个，是将函数的引用赋值给了func，因此，func和foo指向同一个函数，func和foo都可以调用该函数

函数作为参数进行传递，使用方法是调用----像正常的变量一样

function foo (fn) {

if (typeof fn === 'function'){

fn();

}

}

function func1() {}

var func2 = function () {};

foo(func1);

foo(func2); //函数名存储的函数引用，因此将函数作为参数传递时，直接传递函数名即可

函数作为返回值使用

function foo() {}

function func() {}

return func;

}

var f = foo();

f();

foo()();

function foo() {

return function() {};

}

### 四、闭包应用

1、 闭包代码的基本结构

function foo () {

var num = 123;

return function () {

return num;

}

}

2、如何返回函数中的多个数据

通过一个对象将返回的数据的多个方法包含起来，然后返回这个对象就可以了

请编写代码让外部可以访问到name，age，hobby

再编写代码让外部可以修改hobby

function func() {

var name="csw";

var hobby = "吃饭，睡觉";

return {

get\_name:function(){ return name; },

get\_hobby:function(){return hobby;},

set\_hobby:function(value){ hobby = value; }

};

}

var obj = func(); //obj保存的是返回的对象的引用，对象本身在func函数的空间里面

obj.set\_hobby("打豆豆");//通过set方法修改hobby。

var name = obj.get\_name();

#### 3、闭包的核心内容

1> 带有私有数据的函数、对象

function foo () {

var num = 123;

return function () {

// 可以访问 num

}

}

var func = foo();

// 称 func 是一个 带有私有数据的 函数 // 称 func 带有缓存

2> 视情况提供访问数据的通道（返回的函数，返回的对象）

4、闭包的应用\_\_沙箱模式

沙箱就是一个隔离的执行环境

**沙箱模式：一个自调用的函数，我们写在这个函数里面的变量不会被外部的内容所影响到，这样代码即可以正常使用和运行，也会使我们的数据更加的安全。**

有时为了使得代码更加简洁, 会引入很多变量，定义变量越多, 出现冲突的可能性越大.

(function () {

// 沙箱模式

// 所有的代码写在这里

})();

5、闭包的应用\_\_模拟 window.onload 事件的追加与移除

o.addEvent( function ... )

o.removeEvent( fn )

模拟window.onload去制作一个对象，希望通过这个对象添加函数，删除函数，当页面加载完成以后，按照（私有数组存储函数）的顺序逐个的执行这些函数。

var itcastload = (function () {

var data = [];

window.onload = function () {

for ( var i = 0; i < data.length; i++ ) { // 依次执行 数组中的 方法

data[ i ]();

}

};

return {

addEvent: function ( fn ) {

data.push( fn );

},removeEvent:function( fn ){ //遍历 data，发现与fn相同的就删除

for(var i = data.length - 1;i>=0;i-- ){//删除以后arr.length就发生了变化，倒过来循环

if(fn === data[i]){

data.splice(i,1); //arr.splice(从第几个元素开始，删除的个数)

}

}

}

};

})();

itcastload.addEvent(function(){console.log()}; // 先执行

console.log(); // 后执行

function f () {console.log()}

itcastload.removeEvent(f){};

//itcastload.removeEvent(){function f () {console.log()}};

//console.log({} =={}); false!! 解释执行，解释成完全不同的对象

#### 6、利用闭包模拟一个缓存结构

cache 对象, 可以使用 cache[ key ] = value 存储数据, cache[ key ] 获得数据，当 cahche 里面 的数据达到 1024 条, 将最早放进去的数据移除，cache = {} 可以存取数据, 但是不能限定数据的 长度，所以如果需要限定数据, 就是在加入数据的时候判断, 是否已超过尺寸，如果是, 则移除, 如果不是, 则什么也不做

将 cache 做成函数（函数本身也是对象）, 添加数据使用 cache( key, value )

function cache ( key, value ) {

（可以在这里加上限定长度的代码）

cache[ key ] = value;

}

cache( 'attr', function ( node, attr ) {

return node.getAttribute( 'name' ) == attr;

} );

cache( '\_\_name\_\_', '张三' );

//由于需要记录键的数量，并且需要记录添加数据的先后顺序，所有首先考虑有序的数组。

//因此需要让 cache 函数带有缓存功能（数组）

var cache = (function () {

var data = [] , max = 3;

function cache ( key, value ) {

if( data.length >= 3){//做判断, 如果超出范围, 则, 将最开始加入的 移除

var temp = data.shift();//将数组 第 0 项元素移除的 splice, shift

delete cache[temp];

}

data.push(key);

cache[ key ] = value;

}

return cache;

})();

cache(key, value);

//使用缓存

var v = cache (key);

if (!v) {

v = ...;

cache(key, v);

};

处理缓存的函数

function createCache () {

var data = [], max = 3;

function cache ( key, value ) {

if ( data.length >= 3 ) {

var temp = data.shift();

delete cache[ temp ];

}

data.push( key );

cache[ key ] = value;

}

return cache;

}

var kvCache = createCache();



## 十一、函数调用模式

函数模式、方法模式、构造器模式、上下文模式

### 1、 函数模式

函数的定义方式

声明式 function func();

表达式式 var func = function(){};

Function new Function();

-> 单独独立调用的就是函数

函数名( 参数 )

-> this 表示全局对象

-> 任何自调用函数都是函数模式

语法：

function foo() {

。。。

console.log(this); //此时的this指的是window

}

foo();

### 2、方法调用 模式

方法本身就是函数, 但是方法不是单独独立的调用, 而是通过一个对象引导调用.

对象.方法( 参数 )

-> this 表示引导方法的对象

函数是独立的，方法需要对象引导，方法调用一定要有宿主对象（宿主对象可以是对象、函数、数组等），方法一定是某个对象的方法

语法：

function foo() {

console.log(this);

}

var o = { age: 19}; //这里o是宿主对象

o.foo = foo; //这里不能加()，如果加()就是调用foo函数的返回值赋值给o.foo

foo();

o.foo(); //此时this返回的是对象 o

function func () {

console.log(this);

}

func.foo = foo; //这里func是宿主对象

foo(); //函数调用

func.foo(); //方法调用，此时this返回的是函数func（函数整体）

### 3、构造器模式( constructor构造函数模式, 构造方法模式)

使用 new 关键字引导var p = new Person();

构造函数的执行过程，首先通过new创建一个对象，当创建出来对象以后，对象的原型结构就已经确定好了，但是对象里面没有任何成员，它实际上是一个空对象，当它调用构造函数的时候，它把对象的引用交给构造函数，构造函数里面的this就会指向这个函数，然后通过this去执行了this.name,this.age的赋值，实际上就是利用对象的动态特性给当前对象添加了成员

语法：

function Person () {

this.name = 'Jack';

this.age = 19;

}

var p = new Person();

-> 返回值

如果没有return 语句, 那么 构造函数 默认返回 this（即当前对象Person）

如果在构造函数中return 的是基本类型( return num, return 1223 ). 则忽略return,返回this

如果在构造函数中return 的是引用类型, 那么返回该引用类型数据, 而忽略 this

关于构造函数结合性的一个小结

1> 如果调用构造函数的时候构造函数没有参数, 可以省略 圆括号

var p = new Person;

2> 如果希望创建对象并直接调用其方法

( new Person () ).sayHello()

-> 可以省略调整结核性的圆括号

new Person().sayHello()

-> 如果想要省略构造函数的圆括号, 就必须添加结核性的圆括号

(new Person).sayHello()

### 4、上下文调用模式 === 环境调用模式

就是一种方法在不同环境下, 可以同时实现函数模式与方法模式，该方式调用不会污染宿主对象

apply 形式 函数名.apply( ... )

call 形式 函数名.call( ... )

上下文调用存在的目的就是为了实现借用方法

apply调用语法：

-->不带参数的上下文调用模式

function foo () {

console.log( this );

}

var o = { name: 'jim' };

foo(); //函数调用模式

o.func = foo;

o.func(); //方法调用模式

foo.apply( null ) 或 foo.apply() // 函数模式

foo.apply( o ) //方法模式, 注意需要提供一个宿主对象

-> 带有参数的函数如何实现上下文调用

function foo ( num1, num2 ) {

console.log( this );

return num1 + num2;

}

var res1 = foo( 123, 567 );// 函数调用模式

foo.apply(null, [123, 567); //apply函数调用

// 方法调用模式

var o = { name: 'jim' };

o.func = foo;

var res2 = o.func( 123, 567 );

foo.apply(o, [123, 567);//apply函数调用

函数模式：函数名.apply( null, [] );

方法模式：函数名.apply(对象名, []); //这里的对象就是宿主对象，函数的参数放在数组中

call 调用

在使用 apply 调用的时候, 函数参数, 必须以数组的形式存在. 但是有些时候数组封装比较复杂

call 调用与 apply 完全相同, 唯一不同是是参数不需要使用数组

foo( 123, 567 );

foo.apply( null, [ 123, 567 ] );

foo.call( null, 123, 567 );

### 5、借用构造方法实现继承

function Person ( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

传统的继承

function Student () {}

Student.prototype = new Person();

借用构造方法的继承

function Student ( name, age, gender, course ) {

Person.call( this, name, age, gender );

this.course = course;

}

var p = new Student ( 'jim', 19, 'male', '前端' );

## 十二、ES5

### 1、函数的 bind 方法（ES5）

bind 就是 让函数绑定对象的一种方法

函数本身就是可以调用，但是其如果想要作为方法调用，就必须传入宿主对象，并且使用call或apply形式，但是bind使得函数可以与某一个对象绑定起来，那么在调用函数的时候，就好像是该对象在调用方法

语法： 函数.bind (对象)

假如返回一个函数foo，那么调用返回的函数foo，就好像对象在调用该方法一样

var t = document.getElementsByTagName.bind(document);//让t包含函数体的同时也包含对象

arr = [];

arr.push.apply(arr, t('p'));

arr.push.apply(arr, t('p'));

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

arr[i].style.border = '1px solid red';

}

### 2、Object.prototype 的成员

constructor

hasOwnProperty 判断该属性是否为自己提供

propertyIsEnumerable 判断属性是否可以枚举 for in

isPrototypeOf 判断是否为原型对象

toString, toLocaleString, valueOf

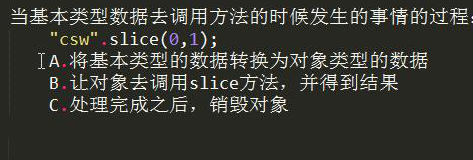
### 3、包装对象

在开发中常常会使用基本数据类型, 但是基本数据类型没有方法, 因此 js 引擎会在需要的时候，自动的将Number，String，Boolean三个基本类型转换成对象类型，可以使用charAt, substr, slice, ...等方法

执行过程：

"abc".charAt( 1 )

"abc" -> new String( "abc" )



.charAt( 1 ) 返回结果

s 就被销毁

当 基本类型.方法 的时候. 解释器首先将基本类型转换成对应的对象类型, 然后调用方法。 方法执行结束后, 这个对象就被立刻回收

在 apply 和 call 调用的时候, 也会有转换发生。上下文调用的第一个参数必须是对象. 如果传递的是数字就会自动转换成对应的包装类型

### 4、getter 与 setter 的语法糖

语法糖: 为了方便开发而给出的语法结构

var o = (function () {

var num = 123;

return {

get\_num: function () {

return num;

},

set\_num: function ( v ) {

num = v;

}

};

})();

o.get\_num(); //获得数据 => 希望o.num 形式，简单

o.set\_num( 456 ); //设置 => 希望o.num = 456 形式，简单

var o = (function () {

var num = 123;

return {

get num () { // get 名字 () { 逻辑体 }

return num;

}, 语法糖

set num ( v ) { // set 名字 ( v ) { 逻辑体 }

num = v;

}

};

})();

### 5、 ES5 中引入的部分数组方法

forEach遍历数组

语法：数组.forEach(fn) 例如 arr.forEach(function (v, i){};

map 映射，返回一个数组，数组的每一个元素就是map函数中的fn的返回值

语法：数组.map(fn(v, i)) v是数据，i是索引

filter 筛选

语法：数组.filter(function (v, i)){return true/false}

some

判断数组中至少有一个数据符合要求，就返回true，否则就返回false

every

必须满足所有的元素都符合要求才会返回true

indexOf

在数组中查找元素，如果含有该元素，返回元素的需求（索引），否则返回-1

lastIndexOf

从右往左找元素

例题：上下文对象的实际应用

需求：获得div与p标签，并添加边框border: 1px solid red

方法① var t = document.getElementsByTagName;

var p\_list = t.apply(document, ['div']);

var div\_list = t.apply(document, ['div']);

var arr = [];

[].push.apply(arr, p\_list);

[].push.apply(arr, div\_list);

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

arr[i].style.border = '1px solid red';

}

方法② var t = document.getElementsByTagName, arr = [];

arr.push.apply(arr, t.apply(document, ['p']));

arr.push.apply(arr, t.apply(document, ['div']));

arr.forEach(function (v){

v.style.border = '1px solid red';

})

方法③ var t = document.getElementsByTagName.bind(document);//让t包含函数体的同时也包含对象

arr = [];

arr.push.apply(arr, t('p'));

arr.push.apply(arr, t('p'));

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

arr[i].style.border = '1px solid red';

}

数组合并数组的方法

var arr1 = [1,2,3,4];

var arr2 = [5,6];

arr1.push(arr2); //这样是将arr2数组作为一个元素追加到arr1中，结果arr1[1,2,3,4,arr2]

[].push.apply(arr1, arr2);

面试题

var length = 10;

function fn() {

console.log( this.length );

}

var obj = {

length: 5,

method: function ( fn ) {

fn();

arguments[ 0 ]();//对象，接受传递的参数

}

};

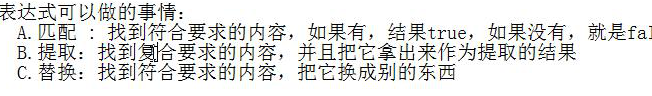
obj.method( fn, 1 );

## 十三、正则

### 1. 正则表达式

regular expression一个有规则的表达式

正则表达式就是 一个用于查找的 含有 匹配字符串 或 匹配元字符的 字符串



### 2. 正则表达式对象

js 正则表达式对象 就是 由 正则表达式创建的对象, 该对象可以进行 匹配, 提取 和 替换.

创建正则表达式对象

1) 构造函数

var regex = new RegExp( 正则表达式字符串[, 匹配模式] );

2) 字面量

var regex = /正则表达式/;

### 3. 使用正则表达式进行匹配

语法:

正则对象.test( 字符串 ) -> bool

如果参数字符串中含有 复合 正则匹配的 子字符串, 就返回 true, 否则返回 false

### 4.基本元字符

. 表示任一个非换行的字符

案例: foot \_ 是 food, foot 写成 foo.

() 表示分组和提高优先级

[] 表示一个字符, 出现在 [] 中的字符

用法: [abc] 出现 [] 中的任意一个字符

匹配 a, 或 b, 或 c

| 或

用法: 正则表达式|正则表达式

转义字符: \

表示点: \.

表示[]: \[ \]

表示(): \( \)

表示\ : \\

### 5. 限定元字符

\* 紧跟前面的一个字符或一组字符出现 0 次到多次

+ 紧跟在前面的字符出现 1 次到多次

? 紧跟在前面的字符出现 0 次或 1 次

{数字} 紧跟在前面的字符出现指定次数

a{3} aaa

{数字,} 紧跟在前面的字符至少出现指定次数

a{3,} aaa, aaaaaaaaa, aaaaaaaaaaaaaa, ...

{数字, 数字} 紧跟在前面的字符出现的次数范围

a{1, 3} a, aa, aaa

### 6. 首尾元字符

^ 表示必须以 xxx 开头

$ 表示 必须以 xxx 结尾

### 7. 简写元字符

\s 空白字符, 包括空格, tab, 回车换行等

\S 非空白字符

js 中常常使用 [\s\S] 表示任意字符

\w 表示字符, 包含字母, 数字, 下划线.

\W 非字符

\d 数字

\D 非数字

### 8. 提取

使用正则表达式可以进行匹配, 使用 exec 可以将匹配到的数据提取出来

语法:

正则表达式对象.exec( 字符串 ) -> 封装成数组

1）. 循环提取

全局模式：匹配正则表达式的一种方式，将一个字符串中所有符合的要求的字符串提取出来，

1> 正则表达式需要使用全局模式

var r = new RegExp( '正则', 'g' );

var r = /正则/g;

2> 调用 exec 首先获得第一个匹配项，一直调用下去, 就可以得到所有的匹配项，直到最后全部匹配完, 如果还用该方法, 则返回 null

案例： 在 'abc123def456ghij789klum' 提取数字

var str= 'abc123def456ghij789klum';

var r = /\d+/g;

while ( res = r.exec( str ) ) {

// 操作 res, 处理捕获的结果

}

2）. 将匹配到的结果进行解析

假如有一个字符串: itcast@itcast.cn

匹配提取邮箱, 同时将邮箱地址部分的名字和主机名都提取出来

3）. 匹配但是不去捕获的元字符

(?:其他正则表达式内容)

4）. 要截取一个字符串中的一个 html 标签

var str = '123<div>456</div>78<span>9</span>0';

1> 要截取 html 标签, 标签使用 <>. 因此正则表达式写成 <\w+>

2> 与之匹配的标签名是什么?

反向引用：如果在一个正则表达式中使用组匹配到某一个数据, 允许在该正则表达式中使用

'\数字' 的方式引用该组

<(\w+)>.\*<\/\1>

### 9. 贪婪模式

贪婪模式：凡是在正则表达式中, 涉及到次数限定的, 一般默认都是尽可能的多匹配.

取消贪婪模式： 在次数限定符（+，\*，？，{}）后面加上 ?

注意: 贪婪模式性能会略高于非贪婪模式, 所以开发的时候. 一般不考虑贪婪的问题

只有代码匹配结果出现问题了, 一般一次多匹配了, 才会取消贪婪

多个贪婪在一起的时候的强度

\d+\d+\d+

1234567

如果取消贪婪模式

\d+?\d+\d+

### 10. 否定元字符

语法:

[^字符]

不为这些字符的字符

[^abc] 不是 a 也不是 b 也不是 c 的字符

### 11. 字符串的替换

语法:

字符串.replace( 查找字符串, 替换字符串 ) -> 字符串

1> 字符串替换方法

字符串1.replace( 字符串2, 字符串3 )

在 字符串1 中找到 字符串2, 将其替换成 字符串3, 返回替换后的字符串

特点: 只替换第一个找到的字符串

2> 正则表达式替换

字符串.replace( 正则表达式, 字符串 ) -> 字符串

1. 简单替换

'aaaaa-------bbbbbb------ccccc'.replace( /-+/, '-' ) -->'aaaaa-bbbbbb------ccccc'

'aaaaa-------bbbbbb------ccccc'.replace( /-+/g, '-' ) -->'aaaaa-bbbbbb-ccccc'

2. 分组替换

可以在替换字符串中, 使用 $数字 来引用替换的数据

'abc123'.replace( /(\d+)/, 'd$1' ) -> 'abcd123'

'1991-1-1'

中国: 1991年1月1日

欧洲: 1/1/1991

... : 1991 1 1

'1991-1-19'.replace( /(\d+)-(\d+)-(\d+)/, '$1年$2月$3日' )

3> 函数参数用法(了解)

语法:

字符串.replace( 正则表达式, fn )

'我的邮箱是:abc@itcast.cn, 你的邮箱是: defghijk@itcast.cn'

.replace( /\w+@\w+(\.\w+)+/g, '\*' );

'我的邮箱是:abc@itcast.cn, 你的邮箱是: def@itcast.cn'

.replace( /(\w+)@(\w+(\.\w+)+)/g, function ( s, g1, g2, g3 ) {

return '\*';

} );

// 要求 只显示 第一个字符, 其余的都是用 \* 表示

'我的邮箱是:abc@itcast.cn, 你的邮箱是: defghijklim@itcast.cn'

.replace( /(\w+)@(\w+(\.\w+)+)/g, function ( s, g1, g2, g3 ) {

// 将用户名 变成 a\*\*\*\*\* 的形式

var first = g1.charAt( 0 );

var start = [];

for ( var i = 0; i < g1.length - 1; i++ ) {

start.push( '\*' );

}

return first + start.join('') + '@' + g2;

} );

### 12. 案例

1> 写一个正则表达式匹配 身份证号码

身份证是 18 位数字

省 市 区 出生年月 随机编码X

1) 首先是要做匹配, 就一定要使用 ^ $

2) 是 18 位数字( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ). 可以使用 [0123456789]

还可以使用 0|1|2|3|4|5|6|7|8|9

3) 要求是 18 位. 限定 18 位, 即 {18}

组合: ^[0123456789]{18}$

如果最后一位是x

前面 17 位数字: ^[0123456789]{17}$

后面要么是数字: [0123456789]

要么是 x: x

写法: [0123456789x]

[0123456789]|x

0|1|2|3|4|5|6|7|8|9|x

^[0123456789]{17}[0123456789x]$

2> 匹配邮箱

12345678901@qq.com

abcdefg@126.com

abcdefg@163.com

abc@yahoo.com.cn

.cc

.org

.edu

.中文

...

名字 @ 主机名

1) 是要验证邮箱, 那么就需要使用 ^ $

2) 名字:

数字与字母

[0123456789]

[abcdefghi...]

[ABCDEFG...]

[] 里面的字符如果是编码序号连续的可以使用连字符连接

数字: [0-9]

[9-0] 错误的, 编码逆序

字母: [a-z]

[A-Z]

整合: [0-9a-zA-Z]

名字的变式方法: [0-9a-zA-Z]+

3>主机名

主机名也是一串字符串或数字

但是它多了一个 .com .cn

3.1) 只有名字 [0-9a-zA-Z]+

3.2) 只含有一个 .什么

开始 \.

中间 [0-9a-zA-Z]+

只含有一个名字: \.[0-9a-zA-Z]+

3.3) 含有多个名字

.com.con.cc.c1.c2.c3

即 .什么 出现一次到多次

(\.[0-9a-zA-Z]+)+

最后主机名可以写成

[0-9a-zA-Z]+(\.[0-9a-zA-Z]+)+

最后整合一下

^[0-9a-zA-Z]+@[0-9a-zA-Z]+(\.[0-9a-zA-Z]+)+$

4>. 匹配一个数字

1> 匹配一个数字

[0-9]+

1) 由于是匹配, 包含 ^ $

2) 首先第一个字符不允许是 0, 所以第一个可以写成 [1-9]

3) 后面的数字就是 [0-9]

4) 要求后面的数字出现 0 次到多次, 以匹配任意的 非 0 数字: [1-9][0-9]\*

5) 由于还需要考虑 0, 因此写成 [1-9][0-9]\*|0

6) 考虑 | 优先级最低: ^([1-9][0-9]\*|0)$) 考虑正负

^(-?[1-9][0-9]\*|0)$

2> 匹配一个指定范围的数字

匹配 0 ~ 255 为例

[0-255] 匹配的是0, 1, 2, 5

如果要匹配指定范围的数字, 那么需要将字符串结构进行分类

1) 0 要匹配的, 所以在正则表达式中有 0 这一项

2) 任意的 2 位数, 即 [1-9][0-9]

3) 任意的 1 位数, 即 [0-9], 可以将 第 1) 结论合并

4) 考虑 3 位数的时候, 只允许出现 1xx 的任意数, 而 2xx 的有限制

因此在分组, 考虑 1xx 的任意数, 可以写成: 1[0-9][0-9]

5) 考虑 2xx 的数字, 在 200 到 250 之间允许任意取. 所以

写成: 2[0-4][0-9]

6) 考虑 250 到 255, 写成 25[0-5]

综合一起:

^([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])$

匹配赢 IPv4

127.0.0.1

192.168.1.1

^(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]))$

3> 匹配小数

要求, 小数不允许以 0 结尾

3.1415

[0-9]\*[1-9]

^((-?[1-9][0-9]\*|0)(\.[0-9]\*[1-9])?)$

jq 中有一属性过滤选择器

[value=name] [value] [value !=name] ...

了解

\[[a-zA-Z]((!|\||~|^|$|)=...|)\]

案例:

var str = '我有一个邮箱, 是 itcast@itcast.cn, 还有 abc@126.com, 和 1234567@qq.com'

要求将里面的邮箱全部提取出来, 然后将名字还有主机名也取出来

// 循环提取的正则表达式

var r = /[a-zA-Z\d]+@[a-zA-Z\d]+(\.[a-zA-Z\d]+)+/g;

// 如果需要将匹配到的结果再次分解, 可以使用分组的形式, 在正则中分组即可

var r = /([a-zA-Z\d]+)@([a-zA-Z\d]+(\.[a-zA-Z\d]+)+)/g;

// 注意, 正则表达式中, 分组是有编号的. 从左往右数 '('. 从 1 开始依次编号

// 匹配提取的结果中, 对应的编号就可以获得分解的数据

## 面向对象面试题

**1、说说你对闭包的理解**

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

闭包有三个特性：

1.函数嵌套函数

2.函数内部可以引用外部的参数和变量

3.参数和变量不会被F垃圾回收机制回收

**2、js继承方式及其优缺点**

原型链继承的缺点

一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

**3、Javascript中callee和caller的作用？**

caller是返回一个对函数的引用，该函数调用了当前函数；   
 callee是返回正在被执行的function函数，也就是所指定的function对象的正文。

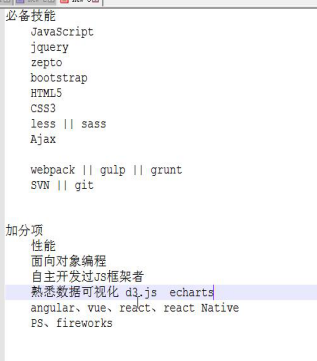
**4 . .call() 和 .apply() 的含义和区别？**

call() 和 apply() 是预定义的函数方法。两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。 后面的参数都是传递给当前对象的参数。

  区别：apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。   
 Object.call(this,obj1,obj2,obj3)   
 Object.apply(this,arguments)   
 在 JavaScript 严格模式(strict mode)下, 在调用函数时第一个参数会成为 this 的值， 即使该参数不是一个对象。 在 JavaScript 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

**5 . new操作符具体干了什么呢?**

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。   
 2、属性和方法被加入到 this 引用的对象中。   
 3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。



# AJAX

## 一、服务器与客户端（掌握）

### 1、服务器和客户端说白了，都是电脑

服务器：能够提供某种服务的机器（计算机），主要给上网用户提供后台服务

客户端：具有向服务器索取服务能力的终端，主要用于普通上网用户

### 2、服务器

定义：使计算机具备提供某种服务能力的应用软件，称为服务器软件，通过安装相应的服务软件，然后进 行配置后就可以使计算具备了提供某种服务的能力。

## 二、网络相关概念

### 1. IP地址

IP就是给每个连接在互联网上的主机分配的一个32位地址。

查看本机的IP： ipconfig 或者ipconfig -all

查看域名的IP： ping

### 2. 域名

由于IP地址是基于数字，不方便记忆，于是便用域名来代替IP地址，域名就是方便IP的的别名，替代品

一个域名可以对应多个IP（负载均衡）

一个IP可以对应多个域名（虚拟主机）

### 3. 端口

端口是计算机与外界通讯交流的出口，每个端口对应不同的服务。说白了就是服务所对应的出口。

## 三、网站

### 1、静态网站

1. 全部由HTML（标准通用标记语言的子集）代码格式页面组成的网站，所有的内容包含在网页文件中。

一般文件名均以htm、html、shtml等为后缀。

2. 优点：

a、静态网页的内容相对稳定，因此容易被搜索引擎检索

3. 缺点：

a、没有数据库的支持，在网站制作和维护方面工作量较大

b、静态网页的交互性较差，在功能方面有较大的限制 。

### 2、动态网站

1. 动态网站并不是指具有动画功能的网站，而是指网站内容可根据不同情况动态变更的网站，

一般情况下动态网站通过数据库进行架构。

一般动态网站体现在网页一般是以asp，jsp，php，aspx等结尾。

2. 优点：

a、动态网页以数据库技术为基础，可以大大降低网站维护的工作量。

b、交互能力强，网页会根据用户的要求和选择而动态改变和响应。

c、维护方便，即无需手动操作，便会自动生成新的页面

3. 缺点

a、动态网页中的“？”对搜索引擎检索存在一定的问题，

因此采用动态网页的网站在进行搜索引擎推广时需要做一定的技术处理才能适应搜索引擎的要求

## 四、php基础语法（）掌握

### 1、基础

1. 所有的PHP相关的代码都要写在<?php … ?>里面

echo 的作用就是向页面当中输入字符串，换行 '<br>'

2. PHP 文件的默认文件扩展名是 ".php"。

3. PHP脚本在服务器上执行。AJAX也是

所有咱们现在写代码都要放在Apache服务器下打开

4. PHP 语句必须以分号结尾（;）

变量声明与字符串拼接

1. PHP的变量声明以及变量名的规则

PHP中定义变量使用的 $ 操作符；

变量命名的规则：字母数字下划线，不能以下划线开头，对大小写敏感

2. PHP中的字符拼接使用“ . ”，js中用“+”；

3. PHP中的单引号把包含在其中的变量当做普通的字符串来处理，不会给它赋值，直接输出

PHP中的双印号把包含在其中的变量当做变量解析成变量值，$变量 后面不能直接跟其他字符串

### 2、JSON字符串转化成对象：

var json = '{"name":"lily"}'; json中的数据必须用双引号

var object = JSON.parse(json数据);

数组

数组的两种形式：

$arr = array(1,2,3);默认key是从0 开始的整数

$arr = array("a"=>"1", "b"=>"2","c"=>"3");

echo $arr["a"];//1

1. echo --- 输出简单数据类型，如字符串、数值

2. print\_r($数组名) --- 输出复杂数据类型，如数组

3. var\_dump($数组名) --- 输出详细信息，如对象、数组

$arr = array(1,2,3);

echo $arr[0]; //1

print\_r($arr); //Array([0]=>1 [1]=>2 [2]=>3)

array (size=5) //0 => int 2

1 => int 3

2 => int 5

$arr1 = array('name' => 'lily, "age" =>'18');

echo $arr1['name']; //lily

print\_r($arr1); //Array([user]=>lily [age]=>18)

var\_dump($arr1); // array(size=2)

'user' => string 'lily' (length=4)

'age' => string '18' (length=2)s

### 3、二维数组

$arr = array();

$arr[0] = array(1,2,3);

$arr[1] = array(4,5,6);

print\_r($arr); //Array ( [0] => Array ( [0] => 1 [1] => 2 [2] => 3 ) [1] =>

Array ([apple]=>Array ([color] => red [shape] => round) )

数据类型与数组遍历

1. php的数据类型与JavaScript的数据类型是类似的，都是弱类型语言

字符串、整型、浮点型、布尔型、数组、对象、NULL

2. gettype() 内置函数，用来判断变量的类型

echo gettype($arr);

3. count()是内置函数，用来计算数组的长度

$arr = array(123,456,789);

for($i=0; $i<count($arr);$i++) {

echo $arr[$i].'<br>';

}

foreach($arr as $value) {

echo $value.'-----<br>';

}

foreach($arr as $key => $value){

echo $key.'===='.$value.'<br>';

}

### 4、php基础语法 ---- 函数

1. 自定义函数，语法类似与JavaScript

2. 系统函数直接调用，不需要声明

3. json\_encode() --- 将数组和对象转换为字符串的方法;

php基础语法 ---- get请求参数获取

get[],post[]应用于表单处理

$\_GET是PHP内置好的专门用来接数据用的一个全局数组，它包含了表单提交上来的数据

1. http协议的常用请求方式：（增删改查）

a、get 用来从服务器获取数据（参数一般作为查询条件）

b、post 用来添加数据（表单）

c、put 用来修改数据

d、delete 用来删除数据

2. $\_GET['abc']得到url地址中传递的参数的值

3. 注意：绝不能使用 GET 来发送密码或其他敏感信息

### 5、php基础语法 ---- post请求参数获取

1. form 默认请求方式就是get请求，get请求会把表单数据作为url的参数进行提交

form 的method设置成post

2. 设置服务响应的文件类型：（如果请求的数据乱码，要在php文件中设置）

1. header("Content-Type: text/plain; charset=utf-8");

2. header("Content-Type: text/html; charset=utf-8");

3. $\_POST也是PHP内置好的专门用来接数据用的一个全局数组

GET和POST请求方式的差异

GET没有请求主体，使用xhr.send(null)

GET可以通过在请求URL上添加请求参数

POST有请求主体，可以通过xhr.send('name=itcast&age=10')

POST需要设置请求头

GET效率更好（应用多）

GET大小限制约4K，POST则没有限制

get 方式提交不太安全，post方式相对来说比较安全

### 6、php基础语法----后台接口

1. 将数组和对象转换为字符串的方法 --- json\_encode();

2. 将字符串转换为对象的方式 --- json\_decode();

3. 如何更好的理解接口这个概念

接口说白了就是后台返回特定格式数据，而不是一个完整的页面

就是从后台到前台返回一些数据

## 五、原生ajax请求（重点）

### 1、原生Ajax实现页面局部更新

什么是Ajax-----Asynchronous Javascript And XML(异步JavaScript和XML)，是一种创建交互式网页应用的网页开发技术，是多种技术的集合：JavaScript(XMLHTTPRequest) html css + 服务端

在不刷新页面的情况下，浏览器悄悄地、异步地向服务器发出HTTP请求。

服务器收到请求后，传回新的格式化数据回来（通常是JSON）。

浏览器解析JSON，通过DOM将新数据呈递显示，页面仅局部刷新

通过后台与服务器进行少量数据交换，Ajax可以使网页实现局部更新，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。传统的网页如果需要更新内容，必须重载整个网页页面

### 2、原生Ajax详解-xhr对象创建

1. 标准下 --- xhr = new XMLHttpRequest();

2. IE6 --- xhr = new ActiveXObject('Microsoft.XMLHTTP');

兼容if(window.XMLHttpRequest){

xhr = new XMLHttpRequest();

else{

xhr = new ActiveXObject('Microsoft.XMLHTTP');

}

### 3、原生Ajax详解-请求参数分析

1. open()方法

xhr.open("get","test.txt",true);

调用open方法并不会真正发送请求，而只是启动一个请求以备发送。

它接受三个参数：要发送的请求的类型、请求的URL、表示是否异步的布尔值。

注意：如果是get请求，那么请求参数必须在url中传递

encodeURI()用来对中文参数进行编码，防止乱码。

post请求参数通过send传递，不需要通过encodeURI()转码，但是必须需要设置请求头参数

2. send()方法

如果要发送请求，用send()方法。

要发送特定的请求，需要调用send()方法。

它接受一个参数，即要作为请求主体发送的数据。

如果不需要通过请求主体发送数据，则必须传入null，不能留空。

get请求是只有头部，没有主体的；而post请求有请求主体。

3. xhr对象有一个重要的属性，就是readyState属性，表示“就绪状态”；

就是 --- xhr.readyState

readyState取值只有5种值：0、1、2、3、4

0 (uninitialized) 开始初始化

1 (loading) XMLHttpRequest已经发送了请求

2 (loaded) XMLHttpRequest浏览器已经收到了服务器响应的数据

3 (interactive) 正在解析数据

4 (complete) 数据解析完成，可以使用了



4. 只要readyState属性值发生了变化，就会触发一个事件onreadystatechange事件。

此时，可以用

xhr.onreadystatechange = function(){}

来捕获readyState改变之后做的事情。

5. Ajax一旦用send方法发出HTTP请求之后，

当readyState为4的时候，就会有一个属性产生 --- xhr.status

表示的是请求的文件的状态码

1\*\* ---- 消息

2\*\* ---- 代码请求成功 //200表示相应成功

3\*\* ---- 重定向

4\*\* ---- 请求错误 //404没有找到请求的资源

5\*\* ---- 服务器错误 //500服务器端错误

6. responseText

获得字符串形式的响应数据。

如果来自服务器的响应并非 XML，请使用 responseText 属性。

7. responseXML

获得 XML 形式的响应数据。

8. XMLHttpRequest 对象用于和服务器交换数据。

9. setRequestHeader --- 向请求添加 HTTP 头

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

### 4、原生Ajax详解-响应状态分析

原生Ajax详解：

1、创建XMLHttpRequest对象

var xhr = new XMLHttpRequest();

准备发送

encodeURI()用来对中文参数进行编码，防止乱码

var param = 'username' + uname + '&password' + pw;

xhr.open('get','./01check.php?'+encodeURI(param),true);

参数1、请求方式（get获取数据，post提交数据）

参数2、请求地址（如果是get请求那么请求参数必须在url中传递）

参数3、同步或者异步标志位，默认是true表示异步，false表示同步

(post请求)xhr.open('post','./01check.php',true); //post请求，参数通过send传递，不需要通过encodeURI转码

执行发送动作

xhr.send(null); //get请求这里需要添加null参数

(post请求)xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded);

(post请求)xhr.send(param); //post请求参数在这里传递，并且不需要转码

4、指定回调函数

xhr.onreadystatechange = function(){ //该函数调用的条件就是readyState状态从2-4

if(xhr.readyState == 4){ //表示服务器返回的数据已经可以使用了，但是这个数据不一定是正确的

if(xhr.status == 200){//表示服务器返回的数据是正常的，不是200表示数据是错误的

var data = xhr.responseText;

var info = document.getElementById('info');

if(data == '1'){

info.innerHTML = '登录成功';

}else if(data == '2'){

info.innerHTML = '用户名或者密码错误';

}

}

}

}

## 六、json（重点）

### 1、json数据格式

1、什么是JSON

- JavaScript 对象表示法

- 是存储和交换文本信息的语法

- 轻量级的文本数据交换格式

2、JSON数据和普通的JS对象的区别

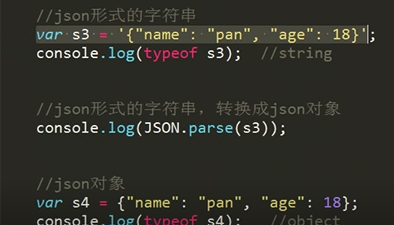
- json 对象没有变量

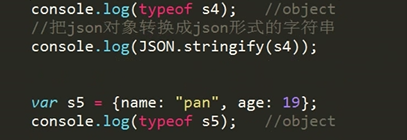
- json 形式的数据结尾没有分号

- json 数据中的键必须用双引号包括

3、JSON和XML对比

- JSON 比 XML 更小、更快，更易解析





js对象，键没有引号

### 2、json数据解析

1、把JSON文本转换为JavaScript对象

JSON 最常见的用法之一，是从 web 服务器上读取 JSON 数据（作为文件或作为 HttpRequest），将 JSON 数据转换为 JavaScript 对象，然后在网页中使用该数据。

2、为什么要转换

在数据传输过程中，json是以文本，即字符串的形式传递的，而JS操作的是JSON对象，所以，JSON对 象和JSON字符串之间的相互转换是关键。

3、转换的方法

**JSON.parse() 把json形式的字符串转成对象**

JSON.stringify() 把对象转成字符串

eval() 把字符串解析成JS代码并执（不安全）

4、php开发json形式数据接口

**json\_encode() --- 把数组转换成JSON形式的字符串**

### 3、同步与异步理解

同步：等待一个请求完成，后续进程被阻塞，只有等请求返回后，才能执行后续的语句

异步：不等待请求完成，立即执行后续语句，请求返回后，再执行回调函数里面的语句

多线程：就是指计算机CPU有一定的物理结构，能够在物理上，同时计算两个东西。一边算加法，一遍算乘法。 程序是并发的。

单线程：就是计算机每时每刻，只有一个计算任务在执行。程序不是并发的。

同步与异步底层原理分析

js事件处理机制：

浏览器单线程，浏览器事件队列

事件队列中的任务执行的条件

定时函数（延时时间已经达到）

事件函数（特定事件被触发）

ajax的回调函数（服务器有数据响应）

拓展

单线程也能异步。就是你的程序只用一个线程，也就是说计算机同一时刻只能计算一个任务。那为什么还 能异步呢？？原因是，异步的时候，CPU把请求已经交给了磁盘这个物理硬件，而CPU还在以单线程运行。 比 如世界著名 Node.js语言，就是非常著名的单线程、异步的语言。

多线程也能同步。多线程，就是CPU能够同时计算多个命令，但是程序就是选择傻等，你管得着么？比 如 PHP是世界上最最注明的多线程语言，但是里面提供的读取文件方法，全是同步的。

## 七、ajax封装（了解）

### 1、Ajax初步封装

抽取函数 --- 学会抽取函数

实际开发中，重复利用的代码块也要封装成一个函数

理解回调函数的调用方式 --- 写5遍，写熟

array\_key\_exists() -- 这里的array\_key\_exists用来判断数组中没有对应键

### 2、jQuery框架-Ajax相关API基本使用

常用的jQuery ajax参数：

$.ajax({

type : "",

url : "",

data : "",

dataType : "",

beforeSend: function(xhr){

$('').html('正在拼命加载...');

}

success : function(data){

}

});

## 八、跨域（掌握）

### 1、什么是跨域

同源策略是浏览器的一种安全策略，所谓同源指的是请求URL地址中的协议、域名和端口号都相同，只要其中之一不相同就是跨域。

同源策略主要是为了保证浏览器的安全性

在同源策略下，浏览器不允许Ajax跨域获取服务器数据

说简单点：跨域是指从一个域名的网页去请求另一个域名的资源。只要协议、域名、端口有任何一个的不同，就被当作是跨域。

2、跨域解决方案

jsonp，99%的公司都在使用jsonp。

利用script标签的跨域能力，这就是jsonp的基础。

### 2、JSONP的原理

1、JSONP的原理（1）

静态script标签的src属性进行跨域请求

如果使用src这种方法进行跨域的话，一定要把这段获取数据的代码在上面先行加载，使用数据的方法放 在后加载

这种方法的缺点就是非常的不灵活，这种方法已经极少使用，了解即可

2、JSONP的原理--动态（2）

利用js构造一个script标签，把json的url赋给script的src属性，把script插入到dom里，让浏览器去获取

动态创建script标签，通过标签的src属性发送请求

动态创建script标签发出去的请求是异步请求

服务器响应的内容是【函数调用】foo(实参)

3、JSONP的原理（3）

1、jsonp的本质：动态创建script标签，然后通过它src属性发送跨域请求，然后服务器响应的数据格式为【函数调用(foo(实参))】

所以在发送请求之前必须先声明一个函数，并且函数的名字与参数中传递的名字要一致，这里声明的函数是由服务器响应的内容，（实际就是一段js代码-函数调用）来调用

2、注意：ajax和jsonp其实本质上是不同的东西。

ajax的核心是通过XMLHttpRequest获取非本页内容，

而jsonp的核心则是动态添加<script>标签来调用服务器提供的js脚本。

**3、贺师俊 --- JSONP 的工作原理是什么**

就是利用<script>标签没有跨域限制的“漏洞”（历史遗迹啊）来达到与第三方通讯的目的。

当需要通讯时，本站脚本创建一个<script>元素，地址指向第三方的API网址，

形如： <script src="http://www.example.net/api?param1=1&param2=2"></script>

并提供一个回调函数来接收数据（函数名可约定，或通过地址参数传递）。

第三方产生的响应为json数据的包装（故称之为jsonp，即json padding），形如：

callback({"name":"hax","gender":"Male"})

这样浏览器会调用callback函数，并传递解析后json对象作为参数。本站脚本可在callback函数里处理所传入的数据。

#### 4、定义函数

一定要写在调用JSONP之前。也就是，调用任何外部js，都没有函数声明头的提升。

jsonp属性的作用就是自定义参数名字（callback=abc,这里的名字指的是等号前面的键，后端根据这个键获取方法名，jQuery的默认参数名称是callback） //等号前面的是后台获取数据abc用的,GET(callback)

jsonpCallback属性的作用就是自定义回调函数的名字，（callback=abc，这里的名字指的是等号后面的值）

//等号后面的是定义回调函数的名字

## 九、模板引擎：（掌握）

模板 + 数据-->静态页面片段-->插入到页面中

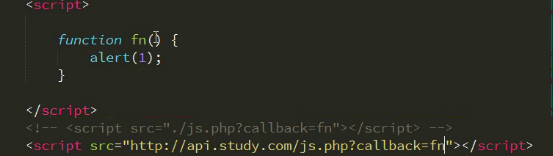
artTemplate 高效的原因：

### 1、预编译

artTemplate的编译赋值过程是在渲染之前完成的

### 2、更快的字符串相加方式

artTemplate根据JavaScript引擎特性采用了数组push拼接字符串和+=两种不同的字符串拼接方式，在IE6-8的浏览器下，数组push方法拼接字符串会比+=快，但是现代浏览器使用+=会比数组push方法快









change([[data],fn])

当元素的值发生改变时，会发生 change 事件。

该事件仅适用于文本域（text field），以及 textarea 和 select 元素。当用于 select 元素时，change 事件会在选择某个选项时发生。当用于 text field 或 text area 时，该事件会在元素失去焦点时发生。

:get(index)所有大于0 的元素

### 3、把？后面的参数取出来，方法

## ajax面试题

**1.什么是HTTP协议？**

A、HTTP协议（HyperText Transfer Protocol，超文本传输协议）是因特网上应用最为广泛的一种网络传输协议，所有的WWW文件都必须遵守这个标准。   
 B、HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。   
 C、HTTP是在网络上传输HTML的协议，用于浏览器和服务器的通信。   
 D、HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。

**2.ajax请求时，如何解析json数据？json格式如何在js中解析？**

在js中解析json有两种方式：eval和JSON.parse。 鉴于安全性考虑，使用parse更靠谱。   
 eval不仅解析了json字符串，还执行了json字符串中的js方法。eval不会判断json字符串是否合法，JSON.parse则会判断。所以使用eval方式是非常危险的，一般使用JSON.parse方式。

**3、如何解决跨域问题**

①JSONP：

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

②CORS

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

③通过修改document.domain来跨子域

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

④使用window.name来进行跨域

window对象有个name属性，该属性有个特征：即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的

⑤使用HTML5中新引进的window.postMessage方法来跨域传送数据

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

**4、XML和JSON的区别？**

(1).数据体积方面。

JSON相对于XML来讲，数据的体积小，传递的速度更快些。

(2).数据交互方面。

JSON与JavaScript的交互更加方便，更容易解析处理，更好的数据交互。

(3).数据描述方面。

JSON对数据的描述性比XML较差。

(4).传输速度方面。

JSON的速度要远远快于XML。

**5、创建ajax过程**

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.

(2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.

(3)设置响应HTTP请求状态变化的函数.

(4)发送HTTP请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

**6、HTTP和HTTPS**

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性

**7、HTTP状态码**

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 请求成功

201 请求成功并且服务器创建了新的资源

202 服务器已接受请求，但尚未处理

301 请求的网页已永久移动到新位置。

302 临时性重定向（请求的资源临时从不同的 URI响应请求）。

303 临时性重定向，且总是使用 GET 请求新的 URI。

304 自从上次请求后，请求的网页未修改过。

400 错误请求（请求的参数错误或者服务器不理解请求的语法）

401 请求未授权。

402 该状态码是为了将来可能的需求而预留的

403 禁止访问。

404 找不到如何与 URI 相匹配的资源。

500 服务器端错误。

502：网关或代理无效/无响应，网络错误

503 服务器端暂时无法处理请求（可能是过载或维护）。

**软件开发流程（软件开发生命周期）**

**生命周期：**孕育、诞生、成长、成熟、结束

**第一步：可行性分析与项目开发计划**

这个阶段主要确定软件开发的目标及其可行性，明确要解决的问题及解决办法，以及解决问题需要的费用、资源、时间。要进行问题定义、可行性分析，制定项目开发计划。《**项目开发计划**》

**第二步：需求分析（设计师）**

需求分析是明确软件系统要做什么，确定软件系统的功能、性能、数据、和界面等要求。《**软件需求说明书**》

**第三步：设计**

**概要设计**就是设计软件的结构，明确软件系统由那些模块组成，这些模块的层次结构、调用关系以及模块的功能，同时确定数据结构和数据库结构。

**详细设计**是对每个模块完成的功能进行具体的描述，把功能描述转变为精确地、结构化的过程描述，既该模块的控制结构或者说逻辑结构。

**《概要设计说明书》、《数据库设计说明书》、《接口设计》、《详细设计说明书》**

**第四步：编码**

编码就是把模块的控制结构转化为程序代码，该阶段需要编码规范。

开发者根据《软件系统详细设计报告》中对数据结构、算法分析和模块实现等方面的设计要求，开始具体的编写程序工作，分别实现各模块的功能，从而实现对目标系统的功能、性能、接口、界面等方面的要求。

**第五步：测试**

测试编写好的系统。《软件测试计划》、《测试用例》、《软件测试报告 》

**第六步：软件交付准备**

在软件测试证明软件达到要求后，软件开发者应向用户提交开发的目标安装程序、数据库的数据字典、《用户安装手册》、《用户使用指南》、需求报告、设计报告、测试报告等双方合同约定的产物。

**第七步：验收**

用户验收。

**第八步：软件维护**

\* 软件数据库管理

\* 用户跟踪培训

\* 故障分析解决

**第九步：软件升级**

\* 需求调整分析

\* 软件功能拓展

\* 系统优化

**第十步：报废处置**

\* 软件不能适应业务发展

\* 新软件项目立项

\* 企业数据详细备份

**[前端开发](http://lib.csdn.net/base/jquery" \o "jQuery知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**面试知识点大纲：

**HTML&CSS：**   
 对Web标准的理解、浏览器内核差异、兼容性、hack、CSS基本功：布局、盒子模型、选择器优先级及使用、 **[HTML5](http://lib.csdn.net/base/html5" \o "HTML5知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、**[CSS3](http://lib.csdn.net/base/css3" \o "CSS3知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、移动端适应

**[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)：**   
 数据类型、面向对象、继承、闭包、插件、作用域、跨域、原型链、模块化、自定义事件、内存泄漏、事件机 制、异步装载回调、模板引擎、Nodejs、JSON、ajax等。

**其他：**   
 HTTP、安全、正则、优化、重构、响应式、移动端、团队协作、可维护、SEO、UED、**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/dongyang0311/article/details/_blank)**、职业生涯

作为一名前端工程师，无论工作年头长短都应该必须掌握的**知识点**：

一、DOM结构 —— 两个节点之间可能存在哪些关系以及如何在节点之间任意移动。   
二、DOM操作 ——如何添加、移除、移动、复制、创建和查找节点等。   
三、事件 —— 如何使用事件，以及IE和标准DOM事件模型之间存在的差别。   
四、XMLHttpRequest —— 这是什么、怎样完整地执行一次GET请求、怎样检测错误。   
五、严格模式与混杂模式 —— 如何触发这两种模式，区分它们有何意义。   
六、盒模型 —— 外边距、内边距和边框之间的关系，及IE8以下版本的浏览器中的盒模型   
七、块级元素与行内元素 —— 怎么用CSS控制它们、以及如何合理的使用它们   
八、浮动元素——怎么使用它们、它们有什么问题以及怎么解决这些问题。   
九、HTML与XHTML——二者有什么区别，你觉得应该使用哪一个并说出理由。   
十、JSON —— 作用、用途、设计结构。