# JavaScript

## String类型

字符方法：charAt(index),charCodeAt(index),

## 易忘知识点

### XMLHttpRequest对象的几个状态

XMLHttpRequest的readyState属性表示请求/响应过程的当前活动阶段

0：未初始化。尚未调用open()方法；

1：启动。已经调用方法，尚未调用send()方法；

2：发送。已经调用send()方法，尚未收到响应；

3：接收。已经收到部分响应数据；

4：完成。已经收到全部响应数据，而且已经可以在客户端使用

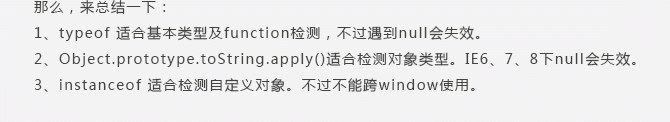
每当readystate状态变化是会触发readystatechange事件

### 类型

对象类型Object（Function、Array、Date）、Number、String、Boolean、Undefined

typeof 运算返回类型：

number string Boolean undefined function object



### toFixed

toFixed方法可把Number四舍五入为指定小数位数的数字

<**script**>  
 **var *num***=300;  
 **if**(***num***===3){  
 ***num***\*=100;  
 }**else if**(***num***===300){  
 ***num***=(***num***/100).toFixed(2);  
 }  
 alert(***num***);  
</**script**>

### IE中的dom0和attachEvent的区别

DOM0作用域当前元素的作用域

attachEvent作用域为Window

### call()/apply()、bind()、caller

http://blog.csdn.net/xxb2008/article/details/7862956

三个方法的作用是改变上下文的this，call()/apply()的区别在于传递参数的方式，call的参数一个个传递，而apply的参数则以数组或者arguments的形式传递，bind与call和apply的区别在于call和apply都是直接调用函数，而bind则返回一个函数。caller为调用对函数体的引用。

### Json

语法：JSON语法可以表示三种类型数据：

简单值；

对象；

数组。

JSON对象方法，

**stringify():对象序列换为JSON字符串**

<**script**>  
**var *book***={  
 **title**:**"professional Javascript"**,  
 **authors**:[**"Nicholas C.zakas"**],  
 **edition**:3,  
 **date**:**null**,  
 **year**:2011  
};  
 **var *bookJSON***=*JSON*.*stringify*(***book***);  
 alert(***bookJSON***);  
</**script**>

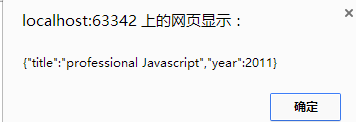
{"title":"professional Javascript","authors":["Nicholas C.zakas"],"edition":3,"date":null,"year":2011}

（1）默认情况下JSON.stringify()输出的JSON字符串不包含任何空格字符或缩进，在序列化JavaScript对象时，所有函数原型成员都会被有意忽略，不体现在结果中。此外值为undefined的任何属性都会被跳过。结果中最终都是值为有效JSON数据类型的实例属性。

（2）stringify第二个参数为过滤参数

如果过滤参数是数组，那么JSON.stringify的结果就只会包含数组中列出的是属性。

**var *book***={  
 **title**:**"professional Javascript"**,  
 **authors**:[**"Nicholas C.zakas"**],  
 **edition**:3,  
 **date**:**null**,  
 **year**:2011  
};  
 **var *bookJSON***=*JSON*.*stringify*(***book***,[**"title"**,**"year"**]);  
 alert(***bookJSON***);  
</**script**>



如果过滤参数是函数时，函数接收两个参数，属性名（键）属性值，

<**script**>  
**var *book***={  
 **title**:**"professional Javascript"**,  
 **authors**:[**"Nicholas C.zakas"**,**"lyx"**],  
 **edition**:3,  
 **date**:**null**,  
 **year**:2011  
};  
 **var *bookJSON***=*JSON*.*stringify*(***book***,**function**(key,value){  
 **switch** (key){  
 **case "authors"**:  
 **return** value.join(**"&"**);  
 **case "year"**:  
 **return** 500;  
 **default** :  
 **return** value;  
 }  
 });  
 alert(***bookJSON***);  
</**script**>

{"title":"professionalJavascript","authors":"Nicholas C.zakas&lyx","edition":3,"date":null,"year":500}

（2）stringify()的第三个参数用于控制结果中的缩进和空白符，最多不可以超过10个空白符，如果参数值超过10则按10计算。以下代码为空白符为3的示例：

**var *book***={  
 **title**:**"professional Javascript"**,  
 **authors**:[**"Nicholas C.zakas"**,**"lyx"**],  
 **edition**:3,  
 **date**:**null**,  
 **year**:2011  
};  
 **var *bookJSON***=*JSON*.*stringify*(***book***,**null**,3);  
 alert(***bookJSON***);

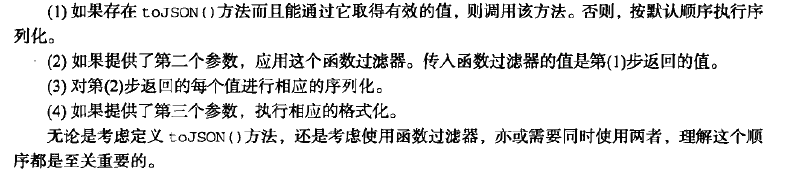


如果缩进参数是一个字符串而非数组，则将此字符串用于JSON字符串中的缩进字符，不超过10个：

**var *bookJSON***=*JSON*.*stringify*(***book***,**null**,**'----'**);  
alert(***bookJSON***);



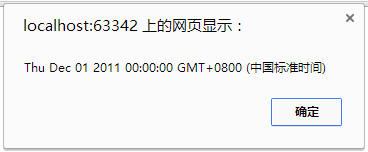
(4)如果stringify不够用可以在对象中定义toJSON()方法，如果对象中年存在toJSON方法，而且通过调用JSON.stringify()，序列化该对象的顺序如下：



parse():把JSON字符串解析为原生JavaScript值，

JSON.parse()方法也可以接收另外一个参数，该参数是一个函数，将在对个键值对上调用。

<**script**>  
**var *book***={  
 **title**:**"professional Javascript"**,  
 **authors**:[**"Nicholas C.zakas"**,**"lyx"**],  
 **edition**:3,  
 **date**:**null**,  
 **date**:**new** *Date*(2011,11,1)  
};  
 **var *bookJSON***=*JSON*.*stringify*(***book***);  
 **var *bookCopy***=*JSON*.*parse*(***bookJSON***,**function**(key,value){  
 **if**(key==**"date"**){  
 **return new** *Date*(value);  
 }**else**{  
 **return** value;  
 }  
 });  
 alert(***bookCopy***.**date**);  
</**script**>



### 数据类型

1. 数值（Number）
2. 布尔型（Boolean）
3. 字符串型（String）
4. 对象（Object）
5. 数组（Array）
6. 空值（Null）
7. 未定义（Undefined）

### Silence

截断字符串

var str=”hello world”;

alert(str.slice(3)); //”lo world”

alert(str.slice(3,-4));//”lo w”

alert(str.slice(3,7));//”lo w”

<http://www.cnblogs.com/chinafine/archive/2009/02/26/1398771.html>

### setTimeOut() setInterval()

window.setTimeout()是经过指定的某个时间段后执行一次指定的方法

window.setInterval()是循环的执行指定的方法，循环的间隔是指定的时间段，

然后通过window.clearInterval()方法清除循环执行。

### Array排序reverse()/sort()方法

Reverse()重排序方法，用法

var values=[1,2,3,4,5];

values.reverse();//5,4,3,2,1

sort()排序方法；

用法：

Var values=[1,4,2,6,3,5];

Function compare(value1,value2){

Return value2-value1；  
}

Values.sort(compare);//1,2,3,4,5,6

### Arrary迭代方法

every()对数组中的每一项运行给定函数，如果该函数每一项都返回true则返回true；

filter()对数组中每一项运行给定函数，返回该数组会返回true的项组成的数组；

forEach()对数组的每一项运行给定函数，该函数没有返回值；

map()对数组的每一项运行给定函数，返回每次函数调用的结果组成的数组；

some()对数组中的每一项运行给定函数，如果函数对任一项运行返回true则返回true。

**var *numbers***=[1,2,3,4,5,6];  
 **var *b***=***numbers***.every(**function**(item,index,array){  
 **return** item >2;  
});  
 alert(***b***);//false

### JavaScript arguments对象

1. 在JavaScript中，arguments是一个比较特别的内置对象，实际上是当前函数的一个内置属性。arguments非常类似Array，但实际上有不是Array。

**function** *funcArg*(){  
 alert(**typeof** arguments);*//object*}  
*funcArg*();

1. arguments对象的长度是有实参个数而不是形参个数来决定的，对于arguments和值都在的情况下，两者的值是同步的，对于一个无值得情况，值不会得到同步。

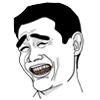
**function** *funcArg*(a,b){  
 alert(a);*//1* a=2;  
 alert(arguments[0]);*//2* alert(arguments[1])*//undefined* b=3;  
 alert(arguments[1])*//undefined*}  
*funcArg*(1);

1. arguments对象中有一个非常有用的属性callee，callee返回此arguments对象所在函数的引用。在使用递归调用时建议使用arguments.callee调用函数本身。

**function** *funcArg*(n){  
 **if**(n===0){  
 **return** 0;  
 }  
 **return** n+arguments.callee(--n);  
}  
 alert(*funcArg*(3));

### display：none，visibility：hidden

display:none;元素不显示，不占用高度，原因：div默认为display：block;有高度

visibility：hidden；也不显示，但是占用高度，因为display:block

还是有效。

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**></**title**>  
 <**style**>  
 **div**{  
 **height**: 100**px**;  
 }  
 **#div1**{  
 **background-color**: **red**;  
 **display**: **none**;  
 }  
 **#div2**{  
 **background-color**: **goldenrod**;  
 **visibility**: **hidden**;  
  
 }  
 **#div3**{  
 **background-color**: **#808080**;  
 }  
 </**style**>  
  
</**head**>  
<**body**>  
<**div id="div1"**>hjkh</**div**>  
<**div id="div2"**></**div**>  
<**div id="div3"**></**div**>  
</**body**>  
</**html**>

### 首字母大写

**#div1**:**first-letter**{  
 **text-transform**:**capitalize**;

}

### 加法操作符

如果两个操作数都是数值，按常规的加法计算，然后根据下列柜子返回结果：

1. 如果一个操作数是NaN，则结果是NaN；
2. 如果是Infinity加Infinity，则结果是Infinity；
3. 如果是-Infinity加-Infinity，则结果是-Infinity；
4. 如果是Infinity加-Infinity，则结果是NaN；
5. 如果是+0加+0，则结果是+0；
6. 如果是-0加-0，则结果是-0；
7. 如果是+0加-0，则结果是+0；

不过，如果过一个操作数是字符串，那么要应用如下规则：

1. 如果两个操作数是字符串，则将第二个操作数与第一个操作数拼接起来；
2. 如果只有一个操作数是字符串，则将另一个操作数转换为字符串，然后再将两个字符串拼接起来。

如果有一个操作数是对象、数值或布尔值，则调用它们的toString()方法取得相应的字符串，在应用前面关于字符串的规则。对于undefined和null，则分别调用String()函数取得字符串“undefined”和“null”

### 正则表达式

防止过度匹配，将匹配变为惰性匹配，即是在贪婪模式后面加上一个“？”即可

\* –> \*?

+ –> +?

{n,} –> {n,}?

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**></**title**>  
  
</**head**>  
<**body**></**body**>  
<**script**>  
 **var *re1*** = /.\*bbb/g;*//贪婪* **var *re2*** = /.\*?bbb/g;*//惰性  
 // var re3 = /.\*+bbb/g;//支配性,javascript不支持，IE与所有最新的标准浏览器都报错* **console**.log(***re1***.exec(**"abbbaabbbaaabbbb1234"**)+**""**);  
 **console**.log(**"abbbaabbbaaabbbb1234"**.match(***re1***)+**""**);  
  
 **console**.log(***re2***.test(**"abbbaabbbaaabbbb1234"**)+**""**);  
 **console**.log(***re2***.exec(**"abbbaabbbaaabbbb1234"**)+**""**);  
 **console**.log(**"abbbaabbbaaabbbb1234bbb"**.match(***re2***)+**""**);  
</**script**>  
</**html**>

### scrollWidth/clientWidth/offsetWidth

scrollWidth：是对象的实际内容的宽，不包含边线宽度，会随着内容的多少改变。

clientWidth：对象可见高度，不包含滚动条等边线，会随着窗口的显示大小而改变。

offsetWidth：是对象的可见宽度，包滚动条等边线，会随窗口的显示大小改变；

### XMLHttpRequest post请求

首先Content-Type头部信息设置为application/x-www-form-urlencoded

### 关于DOM操作常用属性

## 面试题

### 请用javascript写一个Array数组的去重函数，数组为任意类型元素数组

<**script**>  
 **var *array***=[1,2,1,2,1,3,3,5];  
 *Array*.**prototype**.getSingle=**function**(){  
 **var** array=**this**.valueOf();  
 **var** newArray=[];  
 **for**(**var** len=array.**length**,i=0;i<len;i++){  
 **var** repeate=**false**;  
 **for**(**var** newArrayLen=newArray.**length**,j=0;j<newArrayLen;j++){  
 **if**(array[i]===newArray[j]){  
 repeate=**true**;  
 **break**;  
 }  
 }  
 !repeate&&newArray.push(array[i]);  
 }  
 **return** newArray;  
 }  
 alert(***array***.getSingle());  
</**script**>

### 判断字符串以字母开头，后面可以是数字，下划线，字母，长度为6-30s

var reg=/^[a-zA-Z]{1}\w{6,30}$/

<http://www.cnblogs.com/rubylouvre/archive/2010/03/09/1681222.html>

### 请给Array本地对象增加一个原型方法，他的用途是删除数组中重复的条目，并将新的数组返回

<**script**>  
*Array*.**prototype**.slectSingle=**function**(){  
 **var** array=**this**,temp=**new** *Array*(),isRepeat=**false**;  
 **for**(**var** len=array.**length**,i=0;i<len;i++){  
 isRepeat=**false**;  
 **for**(**var** j= 0;j<i;j++){  
 **if**(array[j]===array[i]){  
 isRepeat=**true**;  
 **break**;  
 }  
 }  
 !isRepeat&&temp.push(array[i]);  
 }  
 **return** temp;  
}  
**var *array***=[1,2,3,2,3,3,5,5,6];  
alert(***array***.slectSingle());  
</**script**>

### 用javascript实现：将页面里面所有className 包含test的A标签找出来，并且给他们添加黄色的背景

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**></**title**>  
  
</**head**>  
<**body**>  
<**div id="div1"**>  
 <**a href="#" class="fdsafweftest"** >fdsafweftest</**a**>  
 <**a href="#" class="fdtestsa"** >fdtestsa</**a**>  
 <**a href="#" class="fdsfds"**>fdsfds</**a**>  
 <**a href="#" class="test1fd"**>test1fd</**a**>  
 <**a href="#" class="c23est1"**>c23est1</**a**>  
</**div**>  
</**body**>  
<**script**>  
 **window**.onload=**function**(){  
 **var** a=**document**.getElementsByTagName(**'a'**);  
 **var** test=/test/;  
 **for**(**var** len= a.**length**,i=0;i<len;i++){  
 **if**(test.test(a[i].**className**)){  
 a[i].**style**.**backgroundColor**=**'yellow'**;  
 }  
 }  
 }  
</**script**>  
</**html**>

### 深度对象clone

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**></**title**>  
 <**style**>  
 **#div1**{  
 **background-color**: **red**;  
 **text-transform**:**capitalize**;  
  
 }  
 **#div2**{  
 **background-color**: **goldenrod**;  
 }  
 **#div3**{  
  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="div1"**>fdsafdsafd fdsafdsa</**div**>  
<**div id="div2"**></**div**>  
<**div id="div3"**></**div**>  
</**body**>  
<**script**>  
 **var** *isClass*=**function**(obj){  
 **if**(obj===**null**){  
 **return 'Null'**;  
 }**else if**(obj===**undefined**){  
 **return 'Undefined'**;  
 }**else** {  
 **return** *Object*.**prototype**.toString.call(obj).slice(8,-1);  
 }  
 }  
 **var** *deepClone*=**function**(obj){  
 **var** result,classType=*isClass*(obj);  
 **if**(classType===**'Object'**){  
 result={};  
 }**else if**(classType===**'Array'**){  
 result=[];  
 }**else**{  
 **return** obj;  
 }  
 **for**(key **in** obj){  
 **var** copy=obj[key];  
 **if**(*isClass*(copy)==**"Object"**){  
 result[key]=arguments.**callee**(copy);  
 }**else if**(*isClass*(copy)==**"Array"**){  
 result[key]=arguments.**callee**(copy);  
 }**else**{  
 result[key]=obj[key];  
 }  
 }  
 **return** result;  
 }  
 **var *array***=[1,2,3];  
 **var *copy***=*deepClone*(***array***);  
 ***copy***[2]=8;  
 alert(***copy***);  
</**script**>  
</**html**>

### 请举例使用callee属性实现函数的递归使用

<**script**>  
 **var** *sum*=**function**(num){  
 **if**(num<=1){  
 **return** num;  
 }  
 **return** num+arguments.**callee**(num-1);  
 }  
 alert(*sum*(10));  
</**script**>

### 实现一个遍历数组或对象里所有成员的迭代器。

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
  
  
</**head**>  
<**body**>  
<**canvas id="drawing" width="200" height="200"**>drawing something here</**canvas**>  
</**body**>  
<**script**>  
 **var** *each*=**function**(obj,callback){  
 **if**(obj **instanceof** *Array*){  
 **for**(**var** i=0;i<obj.**length**;i++){  
 callback.call(obj,obj[i],i+1);  
 }  
 }  
 **if**(obj **instanceof** *Object*){  
 **for**(**var** o **in** obj){  
 callback.call(obj,obj[o],o);  
 }  
 }  
 };  
 **var *array***=[1,2,3,4,4,5,6,6,7];  
 **var *obj***={  
 **a**:**"abc"**,  
 **b**:**"bc"**,  
 **c**:**"c"** };  
 *each*(***array***,**function**(item,index){  
 **console**.log(index+**" : "**+item);  
 });  
 *each*(***obj***,**function**(value,key){  
 **console**.log(key+**" : "**+value);  
 })  
</**script**>  
</**html**>

### Doctype? 严格模式和混杂模式-如何触发这两种模式，区分它们有何意义?

当浏览器厂商开始创建与标准兼容的浏览器时，他们希望确保向后兼容性。为了实现这一点，他们创建了两种呈现模式：标准模式和混杂模式（quirks mode）。在标准模式中，浏览器根据规范呈现页面；在混杂模式中，页面以一种比较宽松的向后兼容的方式显示。混杂模式通常模拟老式浏览器（比如Microsoft IE 4和Netscape Navigator 4）的行为以防止老站点无法工作。

差异：

对于这两种模式之间的差异，最显著的一个例子与Windows上IE专有的盒模型有关。在IE 6出现时，在标准模式中使用的是正确的盒模型，在混杂模式中使用的则是老式的专有盒模型。为了维持对IE 5和更低版本的向后兼容性，Opera 7和更高版本也在混杂模式中使用有缺点的IE盒模型。

呈现方面的其他差异比较小，而且是与特定浏览器相关的，包括对于十六进制颜色值不需要#号、假设CSS中没有指定单位的长度的单位是像素，以及在使用关键字时将字号增加一级。

如何触发这两种模式:

浏览器根据DOCTYPE（文档声明）是否存在以及使用那种DTD来选择要使用的呈现方式。如果XHTML和HTML文档保存形式完整的DOCTYPE，那么它一般以标准模式呈现。相反，如果文档的DOCTYPE不存在或者形式不正确则导致HTML和XHTML以混杂模式呈现。

### 行内元素有哪些？块级元素有哪些？CSS的盒模型？

行内元素：

行内元素不可以设置宽（width）和高（height），但可以跟其他元素位于同一行，行内元素一般不可以包含块级元素，行内元素的高度一般由元素内部的字体决定，元素的长度由内容的长度决定，常见的行内元素有a,em，strong。

块级元素：

块状元素排斥其他元素与其位于同一行，可以设定元素的宽（width）和高（height），块级元素一般是其他元素的容器，可容纳块级元素和行内元素。常见的块级元素有div, p ,h1~h6等。

代码如下:

.test{width:100px; height:100px;}

当然我们还可以通过样式display属性来改变元素的显示方式。当display的值设为block时，元素将以块级方式呈现；当display值设为inline时，元素将以行内形式呈现。所以我们可以给a标签应用以下样式：

代码如下:

.test{display:block; width:50px; height:50px;}

另外，如果我们即想让一个元素可以设置宽度高度，又想让它以行内形式显示，这时我们可以设置display的值为inline-block。如：

代码如下:

a{display:inline-block; width:100px; height:100px;}

### CSS 引入的方式有哪些? link 和@import 的区别是?

外部引用CSS中link与@import的区别

质差别

1. Link属于XHTML 标签，而@import完全是CSS提供的一种方式link标签出来可以加载CSS外还可以以做很多其他的事情，比如定义RSS，定义rel链接，import只能加载CSS了。
2. 加载顺序：当页面加载时，link引用的CSS会同时加载，而@import的CSS会等到页面全部被下载完再被加载。所以有时候浏览@import加载的CSS的页面是开始会没有样式；
3. 兼容性问题：@import是CSS2.1提出的，所有老的浏览器不支持，@import只有在IE5以上的才能识别，而link则不存在兼容问题；
4. 使用DOM控制样式时的差别
5. 当使用JavaScript控制DOM去改变样式的时候，只能使用link标签，因为@import不是DOM可以控制的；
6. @import可以在CSS中再次引入其他样式表。

### 前端页面有哪三层构成，分别是什么？作用是什么？

前端开发由结构层、表现层、行为层三个层次组成。

结构层：HTML或XHTML等标记语言创建了网页的结构层。结构层体现了网页文档中包含哪些标签，有哪些内容。

表现层：CSS样式创建了网页的表现层。表现层就是鼎业网页文档的外观视觉如何展现；

行为层：JavaScript创建了网页的行为层。行为层定义了网页中的元素具有怎样的行为动作。

### 你做的页面在哪些流览器测试过？这些浏览器的内核分别是什么?经常遇到的浏览器的兼容性有哪些？怎么会出现？解决方法是什么？

IE内核浏览器：360，傲游，搜狗，世界之窗，腾讯TT

非IE内核浏览器：firefox opera safari chrome

1.就是ie6双倍边距的问题，在使用了float的情况下，不管是向左还是向右都会出现双倍，最简单的解决方法就是用display:inline;加到css里面去。

2.文字本身的大小不兼容。同样是font-size:14px的宋体文字，在不同浏览器下占的空间是不一样的，ie下实际占高16px，下留白3px，ff下实际占高17px，上留白1px，下留白3px，opera下就更不一样了。解决方案：给文字设定 line-height 。确保所有文字都有默认的 line-height 值。这点很重要，在高度上我们不能容忍1px 的差异。

3. ff下容器高度限定，即容器定义了height之后，容器边框的外形就确定了，不会被内容撑大，而ie下是会被内容撑大，高度限定失效。所以不要轻易给容器定义height。

4. 还讨论内容撑破容器问题，横向上的。如果float 容器未定义宽度，ff下内容会尽可能撑开容器宽度，ie下则会优先考虑内容折行。故，内容可能撑破的浮动容器需要定义width。

5. 浮动的清除，ff下不清除浮动是不行的。

6. mirrormargin bug，当外层元素内有float元素时，外层元素如定义margin-top:14px，将自动生成margin-bottom:14px。padding也会出现类似问题，都是ie6下的特产，该类bug 出现的情况较为复杂，远不只这一种出现条件，还没系统整理。解决方案：外层元素设定border 或 设定float。

7. 吞吃现象，限于篇幅，我就不展开了。还是ie6，上下两个div，上面的div设置背景，却发现下面没有设置背景的div 也有了背景，这就是吞吃现象。对应上面的背景吞吃现象，还有滚动下边框缺失的现象。解决方案：使用zoom:1。这个zoom好象是专门为解决ie6 bug而生的。

8. 注释也能产生bug~~~“多出来的一只猪。”这是前人总结这个bug使用的文案，ie6的这个bug 下，大家会在页面看到猪字出现两遍，重复的内容量因注释的多少而变。解决方案：用“ picRotate start。

### 如何居中一个浮动元素?

思路：设置margin-left=50%；left=width/2;

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 **#d**{  
 **width**:200**px**;  
 **height**: 200**px**;  
 **float**: **left**;  
 **left**: -100**px**;  
 **position**: **relative**;  
 **margin-left**: 50%;  
 **background**: **red**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="d"**>  
fjdklsa  
</**div**>  
  
</**body**>  
</**html**>

### js 是什么，js 和html 的开发如何结合？

Js是JavaScript的简称，通过DOM把HTML分析为节点来进行相应的开发。

### 怎样添加、移除、移动、复制、创建和查找节点？

appendChild来添加节点，cloneNode复制节点，removeChild删除节点，createElement创建元素，getElementById、getElementsByTagName、getElementsByName查找节点。

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
  
</**head**>  
<**body**>  
<**div id="father"**></**div**>  
</**body**>  
<**script**>  
 **var *father***=**document**.getElementById(**'father'**);  
 *//添加一个子元素* **var *child1***=**document**.createElement(**'p'**);  
 **var *text1***=**document**.createTextNode(**'text11111'**);  
 ***child1***.appendChild(***text1***);  
 ***father***.appendChild(***child1***);  
 *//添加一个子元素* **var *child2***=**document**.createElement(**'p'**);  
 **var *text2***=**document**.createTextNode(**'text222'**);  
 ***child2***.appendChild(***text2***);  
 ***father***.appendChild(***child2***);  
 *//添加一个子元素* **var *child3***=**document**.createElement(**'p'**);  
 **var *text3***=**document**.createTextNode(**'text333'**);  
 ***child3***.appendChild(***text3***);  
 ***father***.appendChild(***child3***);  
 *//替换* **var *child4***=**document**.createElement(**'p'**);  
 ***child4***.appendChild(**document**.createTextNode(**'text44444'**));  
 ***child3***.**parentNode**.replaceChild(***child4***,***child3***)  
 *//访问第一个子节点* alert(**"firstChild:"**+***father***.**firstChild**.**innerHTML**);  
 *//访问最后一个子节点* alert(**"lastChild:"**+***father***.**lastChild**.**innerHTML**);  
 *//访问前一个节点* alert(**"previoustSibling of child3:"**+***child3***.**previousSibling**.**innerHTML**);  
 *//访问后一个节点* alert(**"nextSibling of child1:"**+***child3***.**nextSibling**.**innerHTML**);  
 *//替换节点*</**script**>  
</**html**>

### 怎样使用事件以及IE和DOM事件模型乊间存在哪些主要差别？

DOM0添加事件：

var btn=document.getElementById(“myBtn”);

btn.onclick=function(){

alert(“clicked”);

};

DOM2添加事件：

var btn=document.getElementById(“mybtn”);

Btn.addEventListener(“click”, onclick,false);

var onclick=function(){

alert(“clicked”);

};

IE添加事件

Var btn=document.getElementById(“mybtn”);

Btn.attachEvent(“click”, onclick,false);

var onclick=function(){

alert(“clicked”);

};

<**script**>  
 **var *EventUtil***={  
 addHandler:**function**(element,type,handler){  
 **if**(element.addEventListener){  
 element.addEventListener(type,handler,**false**);  
 }**else if**(element.attachEvent){  
 element.attachEvent(type,handler,**false**);  
 }**else**{  
 element[**"on"**+type]=handler;  
 }  
 },  
 removeHandler:**function**(element,type,handler){  
 **if**(element.removeEventListener){  
 element.removeEventListener(type,handler);  
 }**else if**(element.detachEvent) {  
 element.detachEvent(type, handler);  
 }**else**{  
 element[**"on"**+type]=**null**;  
 }  
 },  
 getEvent:**function**(event){  
 **return** event?event:**window**.**event**;  
 },  
 getTarget:**function**(event){  
 **return** event.**target**||event.**srcElement**;  
 },  
 preventDefualt:**function**(event){  
 **if**(event.preventDefualt){  
 event.preventDefualt();  
 }**else**{  
 event.**returnValue**=**false**;  
 }  
 },  
 stopPropagation:**function**(event){  
 **if**(event.stopPropagation){  
 event.stopPropagation()  
 }**else**{  
 event.**cancelBubble**=**true**;  
 }  
 }  
 }  
  
</**script**>

### 面向对象编程:b 怎么继承a？

原型链继承方式：

**function** *SuperType*(){  
 **this**.**superProp**=**true**;  
}  
*SuperType*.**prototype**.getValue=**function** (){  
 **return this**.**superProp**;  
}  
**function** *SubType*(){  
 **this**.**subProp**=**false**;  
}  
*SubType*.**prototype**=**new** *SuperType*();

基本思想：将子类的原型等于父类的一个实例，子类将拥有父类的所有属性及方法。

缺点：（1）所有的实例属性共享（包括引用对象）

（2）不能向超类的构造函数传递参数，确切的说是不能在不影响其他实例的情况下传递参数。

借用构造函数：

基本思想：在子类的构造函数内部调用父类构造函数。



优点：构造函数可以传参

问题：方法需定义在构造函数中，无重用性可言

组合继承：综合原型链和借用构造函数继承的优点，思路是使用原型链实现对原型属性和方法的继承，而通过构造函数来实现对实例属性的继承。

### 请编写一个JavaScript凼数 parseQueryString，它的用途是把URL参数解析为一个对象

var obj = parseQueryString(url);

alert(obj.key0) // 输出0

**function** *parseQueryString*(url){  
 **var** argString=url.split(**'?'**)[1].split(**'&'**);  
 **var** obj={};  
 argString.forEach(**function**(item,index,array){  
 **var** temp=item.split(**'='**);  
 obj[temp[0]]=temp[1];  
 });  
 **return** obj;  
};  
**var *obj***=*parseQueryString*(**"http:baidu.com?key0=1&key1=2"**);  
alert(***obj***.key0);

### ajax是什么? ajax的交互模型? 同步和异步的区别? 如何解决跨域问题?

AJAX=异步JavaScript和XML

通过后台与服务器进行少量数据交换，AJAX可以使网页实现异步更新。这意味着可以在不刷新。

交互流程：

1. 创建XMLHttpRequest对象
2. open打开url通道，并设置异步传输
3. send发送数据到服务器
4. 服务器接受数据并处理，处理完成后返回结果
5. 客户端接收服务器端返回

跨与请求资源问题解决：

通过XHR实现Ajax通信的主要限制来源于跨域安全策略。XHR对象只能访问与抱哈它的页面位于同一个域中的资源。这种安全策略可以预防某些恶意行为，但实现合理的跨域请求对开发某些浏览器应用程序也是至关重要的。

CORS（Cross-Origin Resource Sharing，跨域资源共享）是W3C的一个工作草案，定义了在必须访问跨域资源是，浏览器与服务器应该如何沟通。CORS背后的基本思想是使用自定义的THHP头部让浏览器与服务器进行沟通，从而决定请求或响应是否应该成功。

客户端在发送请求时，需要给它附加一个额外的Origin头部，其中包含请求页面的源信息，以便服务器根据这个头部信息来决定是否给予响应：

Origin：<http://www.nczonline.net>

如果服务器认为这个源可以接受，就在Access-Control-Allow-Origin头部中间发回相同的源信息。例如

Access-Control-Allow-Origin：<http://www.nczonline.net>

跨浏览器的CORS

**function** *createCORSRequest*(method,url){  
 **var** xhr=**new** *XMLHttpRequest*();  
 **if**(**"withCredential" in** xhr){  
 xhr=**new** XDomainRequest();  
 xhr.open(method,url);  
 }**else**{  
 xhr=**null**;  
 }  
 **return** xhr;  
}

### 前端解决跨域问题

注意

（1）如果是协议和端口造成的跨域问题“前台”是无能为力的

（2）在跨域问题上，域仅仅通过“URL的首部”来识别，而不会尝试判断一个IP地址对应着的两个域是否在同一个IP上。

跨域方法一：

document.domain+iframe实现跨域

思想：将在一个基础域上的通过设置document.domain相等以后，嵌入iframe，随意操作iframe里的内容。

方法二：动态创建script

思想：script不受同源策略的限制

方法三：location.hash+iframe

思想：a.html要访问b.html 可通过改变location.hash实现

（1）a.html首先创建一个隐藏的iframe，iframe的src指向cnblogs.com域下的b.html

(2)b.html相应后通过修改a.html的hash值来传递数据

（3）同时在a.html上加上定时器，隔一段时间来判断location.hash的值有没有变化一旦有变化则获取hash值

<**script**>  
 **function** *startRequest*(){  
 **var** ifr=**document**.createElememt(**'iframe'**);  
 ifr.style.**display**=**'none'**;  
 ifr.**src**=**'http://www.cnblogs.com/lab/cscript/b.html#paramdo'**;  
 **document**.**body**.appendChild(ifr);  
  
 }  
 **function** *checkHash*(){  
 **try**{  
 **var** data=**location**.**hash**?**location**.**hash**.substring(1):**''**;  
 **if**(**console**.log){  
 **console**.log(data)  
 }  
 }**catch**(e){  
  
 }  
 }  
 setInterval(*checkHash*,2000);  
  
 *//b.html* **switch** (**location**.**hash**){  
 **case '#paramdo'**:  
 *callback*();  
 **break**;  
 **case '#paramset'**:  
 *//do something* **break**;  
 }  
 **function** *callback*(){  
 **try**{  
 **parent**.**location**.**hash**=**'somedata'**;  
 }**catch** (e){  
 **var** ifrproxy=**document**.createElement(**'iframe'**);  
 ifrproxy.**style**.**display**=**'none'**;  
 ifrproxy.**src**=**'http://a.com/textcsript/c.html#somedata'**;  
 **document**.**body**.appendChild(ifrproxy);  
 }  
 }  
 *//c.html* **parent**.parent.**location**.**hash**=**self**.**location**.**hash**.substring(1);  
</**script**>

1. window.name 的美妙之处：name 值在不同的页面（甚至不同域名）加载后依旧存在，并且可以支持非常长的 name 值（2MB）
2. <!DOCTYPE **html**>  
   <**html**>  
   <**head**>  
    <**script**>  
    **function** *proxy*(url, func){  
    **var** isFirst = **true**,  
    ifr = **document**.createElement(**'iframe'**),  
    *loadFunc* = **function**(){  
    alert(**"excut"**);  
    **if**(isFirst){  
    ifr.**contentWindow**.**location** = **'cs1.html'**;  
    isFirst = **false**;  
    }**else**{  
    func(ifr.**contentWindow**.name);  
    ifr.**contentWindow**.close();  
    **document**.**body**.removeChild(ifr);  
    ifr.**src** = **''**;  
    ifr = **null**;  
    }  
    };  
     
    */\*ifr.src = url;\*/* ifr.**style**.**display** = **'none'**;  
    **if**(ifr.attachEvent) ifr.attachEvent(**'onload'**, *loadFunc*);  
    **else** ifr.**onload** = *loadFunc*;  
     
    **document**.**body**.appendChild(ifr);  
    }  
    </**script**>  
   </**head**>  
   <**body**>  
   <**script**>  
    *proxy*(**'http://www.baidu.com/'**, **function**(data){  
    alert(data);  
    });  
   </**script**>  
   </**body**>  
   </**html**>
3. postMessage
4. CORS(Cross-Origin Resource Sharing)

**function** *createCORSRequest*(method,url){  
 **var** xhr=**new** *XMLHttpRequest*()();  
 **if**(**"withCredentials" in** xhr){  
 xhr.open(method,url,**true**);  
 }**else if**(**typeof** XDomainRequest!=**"undefined"**){  
 **vxhr**=**new** XDomainRequest();  
 xhr.open();  
 }**else**{  
 xhr=**null**;  
 }  
 **return** xhr;  
}

1. http://www.cnblogs.com/JChen666/p/3399951.html

### 什么是闭包？下面这个ul，如何点击每一列的时候alert其index?

<ul id=”test”>

<li>这是第一条</li><li>这是第二条</li><li>这是第三条</li>

</ul>

闭包：闭包是指有权访问另外一个函数作用域中的对象的表达式，通常是函数。

闭包的作用：

1. 保护函数内部变量的安全。
2. 在内存中维持一个变量；
3. 通过保护变量的安全实现JS的安全实现JS私有属性和方法。

代码实现

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
</**head**>  
<**body**>  
<**ul id="test"**>  
 <**li**>这是第一条</**li**><**li**>这是第二条</**li**><**li**>这是第三条</**li**>  
</**ul**>  
</**body**>  
<**script**>  
 **var *test***=**document**.getElementById(**'test'**);  
 **var *children***=***test***.**childNodes**;  
 **for**(**var *i***= 0,***I***=***children***.**length**;***i***<***I***;***i***++){  
 (**function**(index){  
 ***children***[index].onclick=**function**(){  
 alert(**'这是第'**+index+**'行'**);  
 }  
 })(***i***);  
 }  
</**script**>  
</**html**>

或者

<**script**>  
 **var *test***=**document**.getElementById(**'test'**);  
 **var *children***=***test***.**childNodes**;  
 **for**(**var *i***= 0,***I***=***children***.**length**;***i***<***I***;***i***++){  
 ***children***[***i***].**index**=***i***;  
 ***children***[***i***].onclick=**function**(){alert(**this**.**index**)};  
 }  
</**script**>

### 如何让img标签在div里上下居中

(1)水平居中：div设置：text-align:center;

img设置：width:图片宽度; margin:0 auto;

垂直居中：div设置：position:relative;

img设置：position:absolute; top:50%; height:图片高度; margin-top:图片高度的一半;

(2)垂直居中

vertical-align定义：vertial-align属性设置元素的垂直对齐方式。该属性定义行内元素的基线相对于该元素所在行的基线垂直对齐。

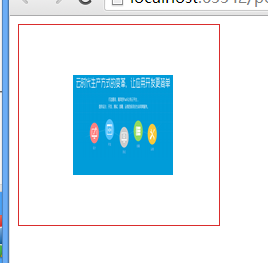
理解：（1）在表单元格中，这个属性会设置单元格框中的单元格内容的对齐方式。例如：如果给一个表格td加上一个vertical-align:middle的样式，表格里面的内容会垂直居中。

1. 该属性定义行内元素的基线相对于该元素所在行的基线垂直对齐，

理解：假设有两个行内元素a和b，a和b都是img，当a加了一个vertical-align:middle样式之后，b的底部就会对齐a的中间位置。

如果a和b都加了一个vertical-align:middle样式，那么就互相对齐了对方的中间位置，也就是它们在垂直方向上的中线对齐了

效果



代码

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 **#div1**{  
 **width**: 200**px**;  
 **height**: 200**px**;  
 **border**: 1**px solid #da2d2d**;  
 **text-align**: **center**;  
 }  
 **#div1 span**{  
 **height**: 100%;  
 **background-color**: **#009cd9**;  
 **display**: **inline-block**;  
 **vertical-align**: **middle**;  
 }  
 **#img1**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **vertical-align**:**middle**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="div1"**>  
 <**span**></**span**>  
 <**img id="img1" src="../images/roll1.png" title="图片" alt="图片"**>  
</**div**>  
</**body**>  
</**html**>

### 实现左侧规定宽200，右侧自适应宽度的布局

1. 思想：

将左侧绝对定位，后侧设置margin-left:200px

代码

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 .**common**{  
  
 **height**: 200**px**;  
 }  
 **#div1**{  
 **position**: **absolute**;  
 **width**: 200**px**;  
 **border**: 1**px solid #da2d2d**;  
 }  
 **#div2** {  
 **width**: 100%;  
 **margin-left**: 200**px**;  
 **border**: 1**px solid #002d9d**;  
 }  
  
 </**style**>  
</**head**>  
<**body**>  
<**div id="div1" class="common"**></**div**>  
<**div id="div2" class="common"**></**div**>  
</**body**>  
</**html**>

（2）思想：左侧float:left 后侧margin-left

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 .**common**{  
  
 **height**: 200**px**;  
 }  
 **#div1**{  
 **float**: **left**;  
 **width**: 200**px**;  
 **border**: 1**px solid #da2d2d**;  
 }  
 **#div2** {  
 **width**: 100%;  
 **margin-left**: 200**px**;  
 **border**: 1**px solid #002d9d**;  
 }  
  
 </**style**>  
</**head**>  
<**body**>  
<**div id="div1" class="common"**></**div**>  
<**div id="div2" class="common"**></**div**>  
</**body**>  
</**html**>

### 写一个浮动块拽动的程序

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
</**head**>  
<**style type="text/css"**>  
 **#container**{  
 **width**: 500**px**;  
 **height**: 200**px**;  
 **position**: **relative**;  
 **border**: 5**px solid red**;  
 }  
 **#dd**{  
 **width**: 50**px**;  
 **height**: 50**px**;  
 **background**: **#5ecc49**;  
 }  
</**style**>  
<**script**>  
 **window**.onload=**function**(){  
 **var** container=**document**.getElementById(**'container'**);  
 **var** dd=**document**.getElementById(**'dd'**);  
 **var** bodyWidth=container.**offsetWidth**,  
 bodyHeight=container.**offsetHeight**;  
 **var** maxX=bodyWidth-dd.**offsetWidth**-10;  
 **maxY**=bodyHeight-dd.**offsetHeight**-10;  
 **var** d=**new DragDrop**({  
 **target**:dd,  
 **area**:[0,maxX,0,**maxY**]  
 });  
 }  
</**script**>  
<**body**>  
<**div id="container"**>  
 <**div id="dd"**></**div**>  
</**div**>  
  
</**body**>  
<**script**>  
 **DragDrop**=**function**(window){  
 **var** doc=window.**document**;  
 **var** EventUtil={  
 addHandler:**function**(element,type,handler){  
 **if**(element.addEventListener){  
 element.addEventListener(type,handler,**false**);  
 }**else if**(element.attachEvent){  
 element.attachEvent(**'on'**+type,handler);  
 }**else**{  
 element[**'on'**+type]=handler;  
 }  
 },  
 removeHandler:**function**(element,type,handler){  
 **if**(element.removeEventListener){  
 element.removeEventListener(type,handler,**false**);  
 }**else if**(element.detachEvent){  
 element.detachEvent(**'on'**+type,handler);  
 }**else**{  
 element[**'on'**+type]=handler;  
 }  
 },  
 getEvent:**function**(event){  
 **return** event?event:window.**event**;  
 }  
 }  
 **var** config=**null**,diffX,diffY;  
 **return function**(opt){  
 **function** *DragDrop*(opt){  
 config=**new** *Config*(opt);  
 **var** ele=config.**target**;  
 EventUtil.addHandler(ele,**'mousedown'**,*mousedown*);  
 }  
 **function** *mousedown*(e){  
 **var** ele= config.**target**;  
 e=EventUtil.getEvent(e);  
 ele.**style**.**position**=**'absolute'**;  
 ele.**style**.**cursor**=**'move'**;  
 **if**(ele.setCapture){  
 EventUtil.addHandler(ele,**'losecapture'**,*mouseup*);  
 ele.setCapture();  
 e.**cancelBubble**=**true**;  
  
 }**else if**(window.captureEvents){  
 EventUtil.addHandler(window,**'blur'**,*mouseup*);  
 e.stopPropagation();  
 e.preventDefault();  
 }  
 diffX=e.**clientX**-ele.**offsetLeft**;  
 diffY= e.**clientY**-ele.**offsetTop**;  
 EventUtil.addHandler(doc,**'mousemove'**,*mousemove*);  
 EventUtil.addHandler(doc,**'mouseup'**,*mouseup*);  
 }  
 **function** *mousemove*(e){  
 e=EventUtil.getEvent(e);  
 **var** ele=config.**target**;  
 **var** moveX= e.**clientX**-diffX;  
 **var** moveY= e.**clientY**-diffY;  
 **var** minX=config.**area**[0];  
 **var** maxX=config.**area**[1];  
 **var** minY=config.**area**[2];  
 **var** maxY=config.**area**[3];  
 moveX<minX&&(moveX=minX);  
 moveX>maxX&&(moveX=maxX);  
 moveY<minY&&(moveY=minY);  
 moveY>maxY&&(moveY=maxY);  
 ele.**style**.**left**=moveX+**'px'**;  
 ele.**style**.**top**=moveY+**'px'**;  
 }  
 **function** *mouseup*(e){  
 **var** ele=config.**target**;  
 EventUtil.removeHandler(doc,**'mousemove'**,*mousemove*);  
 EventUtil.removeHandler(doc,**'mouseup'**,*mouseup*);  
 ele.**style**.**cursor** = **'defualt'**;  
 **if**(ele.releaseCapture){  
 EventUtil.removeHandler(ele,**'losecapture'**,*mouseup*);  
 el.releaseCapture();  
 }**else if**(window.captureEvents){  
 EventUtil.removeHandler(window,**'blur'**,*mouseup*);  
 }  
  
 }  
 **function** *Config*(opt){  
 **this**.**target**=opt.**target**;  
 **this**.**area**=opt.**area**;  
 }  
 **return new** *DragDrop*(opt);  
 }  
 }(**this**)  
</**script**>  
</**html**>

### 列丼hack的技巧

hack书写顺序：一般是将识别能力强的浏览器放在CSS的后边

### html5离线存储

### 写一个获取url参数的函数

方法一：字符串截断的方法

<**script**>  
 **function** *getParams*(url){  
 **var** keyValue={};  
 **var** start=url.indexOf(**'?'**);  
 **if**(start===-1){  
 **return null**;  
 }  
 **var** params=url.substring(start+1,url.**length**).split(**'&'**);  
 **for**(**var** i= 0,len=params.**length**;i<len;i++){  
 **var** kv=params[i].split(**'='**);  
 keyValue[kv[0]]=kv[1];  
 }  
 **return** keyValue;  
 }  
 **var *keyValue***=*getParams*(**'http://a.com?bb=123&cc=abc'**);  
 alert(***keyValue***[**'bb'**]);  
 alert(***keyValue***[**'cc'**]);  
</**script**>

方法二：正则表达式法，首先有一个正则表达式子模式的问题，正则表达式子模式用()扩起来，

*String*.prototype.getD=**function**(name){  
 **var** reg=**new** *RegExp*(**".\*[&|?]"**+name+**"=([^&]\*)&?.\*$"**,**"i"**);  
 reg.test(**this**);  
 **return** *RegExp*.**$1**;  
}

this的用法

var Pet=function()

{

this.msg="Please show me delicious food...";

this.shout = function()

{

console.log(this.msg);

}

this.waitAndShout = function()

{

//在这里写出隔2秒时间调用一次shout的代码。

}

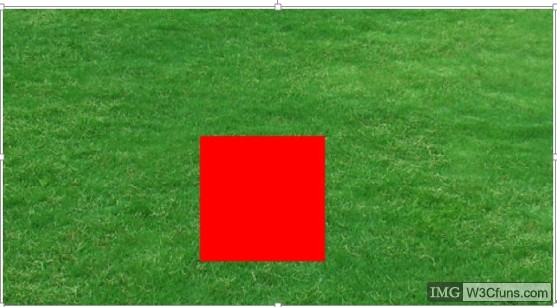
}

<**script**>  
 **var** *Pet*=**function**()  
 {  
 **this**.**msg**=**"Please show me delicious food..."**;  
 **this**.shout = **function**()  
 {  
 alert(**this**.**msg**);  
 }  
 **this**.waitAndShout = **function**()  
 {  
 **var** that=**this**;  
 setInterval(**function**(){that.shout()},2000);  
 }  
 }  
 **var *p***=**new** *Pet*();  
 ***p***.waitAndShout();  
</**script**>

### 使用原生JavaScript给下面列表中的结点绑定点击事件，点击时创建一个Object对象，兼容IE和标准浏览器。

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
  
</**head**>  
<**body**>  
<**ul id="nav"**>  
 <**li**><**a href="http://ju.taobao.com/tg/brand.htm"**>品牌团</**a**></**li**>  
 <**li**><**a href="http://ju.taobao.com/tg/point\_list.htm"**>整点聚</**a**></**li**>  
 <**li**><**a href="http://ju.taobao.com/jusp/jiazhuang/tp.htm"**>聚家装</**a**></**li**>  
 <**li**><**a href="http://ju.taobao.com/jusp/liangfan/tp.htm"**>量贩团</**a**></**li**>  
</**ul**>  
</**body**>  
<**script**>  
 **EventUtil** = {  
 addHandler: **function** (ele, type, handler) {  
 **if** (ele.addEventListener) {  
 ele.addEventListener(type, handler, **false**);  
 } **else if** (ele.attachEvent) {  
 ele.attachEvent(**'on'** + type, handler);  
 }**else**{  
 ele[**'on'**+type]=handler;  
 }  
 },  
 getEvent: **function** (event) {  
 **return** event? event : **window**.**event**;  
 },  
 getTarget: **function** (event) {  
 **if** (event.target) {  
 **return** event.target;  
 } **else** {  
 **return** event.**srcElement**;  
 }  
 **console**.log(event);  
 },  
 preventDefault: **function** (event) {  
 **if** (event.preventDefault) {  
 event.preventDefault();  
 } **else** {  
 event.**returnValue** = **false** }  
 }  
  
 }  
 **var *ul*** = **document**.getElementById(**'nav'**);  
 **var *lis*** = ***ul***.**children**;  
 **for** (**var *i*** = 0, ***len*** = ***lis***.**length**; ***i*** < ***len***; ***i***++) {  
 **var *a***=**function**(index){  
 **var** element=***lis***[index];  
 **EventUtil**.addHandler(element, **'click'**, **function** (event) {  
 event = **EventUtil**.getEvent(event);  
 **EventUtil**.preventDefault(event);  
 **var** ele = **EventUtil**.getTarget(event);  
 **console**.log(ele);  
 **var** obj = {  
 **index**: index,  
 **name**: ele.**innerHTML**,  
 **link**: ele.getAttribute(**"href"**)  
 };  
 alert(obj.**index**);  
 }, **false**);  
 }(***i***);  
 **console**.log(***lis***);  
 }  
</**script**>  
</**html**>

### 图片分析，请编写JS代码获取下图中“红框”的位置（“红框”的颜色为“FF0000”）；



<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style**>  
 **#father**{  
 **height**: 100%;  
 **width**: 100%;  
 **background-color**: **#808080**;  
 }  
 **#child**{  
 **margin**: **auto auto**;  
 **height**: 200**px**;  
 **width**: 200**px**;  
 **background-color**: **#ff0000**;  
 }  
 </**style**>  
  
</**head**>  
<**body**>  
<**div id="father"**>  
 <**div id="child"**></**div**>  
</**div**>  
</**body**>  
<**script**>  
 **var *child***=**document**.getElementById(**'child'**);  
 **function** *getElementBybgValue*(ele){  
 **console**.log(**window**.getComputedStyle(ele).**backgroundColor**);  
 **if**(**window**.getComputedStyle(ele).**backgroundColor**===**"rgb(255, 0, 0)"**){  
 **return** ele;  
 }  
 **if**(ele.children.length!==0){  
 **for**(**var** i= 0;i<ele.children.length;i++){  
 **var** e=*getElementBybgValue*(ele.children[i]);  
 **if**(e){  
 **return** e;  
 }**else**{  
 **continue**;  
 }  
 }  
 }  
 **return null**;  
 }  
 **var *div***=*getElementBybgValue*(**document**.**body**);  
 alert(***div***.**offsetLeft**);  
  
</**script**>  
</**html**>

### 获取某元素的地n层子节点

编写一个JavasSript函数，给定一个DOM节点node和一个正整数n, 返回node的所有第n代后代节点（直接子节点为第1代）  
function getDescendants(node, n) {  
    // return an Array  
}

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
</**head**>  
<**body**>  
<**div**><**ul**><**li**></**li**>  
 <**li**></**li**><**li**></**li**>  
  
</**ul**></**div**>  
<**div**><**div**>  
 <**p**>fd</**p**>  
 <**p**>fd</**p**>  
 <**p**>fd</**p**>  
</**div**></**div**>  
</**body**>  
<**script**>  
 **function** *getDescendants*(node,n){  
 **var** childs=node.children;  
 **if**(n===1){  
 **return** childs;  
 }**else**{  
 **for**(**var** i=1;i<n;i++){  
 childs=*getChilds*(childs);  
 }  
 }  
 **return** childs;  
 }  
 **function** *getChilds*(nodes){  
 **var** childs=[];  
 **for**(**var** i= 0,len=nodes.**length**;i<len;i++){  
 **var** temp=nodes[i].children;  
 **for**(**var** j=0;j<temp.length;j++){  
 childs.push(temp[j]);  
 }  
 }  
 **return** childs;  
 }  
 **var *child***=*getDescendants*(**document**.**body**,3);  
 **console**.log(***child***);  
  
</**script**>  
</**html**>

### 浮框拖动

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 **body**{  
 **-webkit-box-sizing**: **border-box**;  
 **-moz-box-sizing**: **border-box**;  
 **box-sizing**: **border-box**;  
 **font-size**: 16**px**;  
 }  
 **#dialogbg**{  
 **position**: **fixed**;  
 **left**: 0**px**;  
 **top**: 0**px**;  
 **width**: 100%;  
 **height**: 100%;  
 **z-index**:999;  
 **display**: **none**;  
 **filter**:**alpha**(**opacity**=50);  
 **background**:**#444444**;  
 **background**:**rgba**(44,44,44,0.4);  
  
 }  
 **#dialog**{  
 **width**: 300**px**;  
 **height**: 150**px**;  
 **box-sizing**: **inherit**;  
 **position**: **absolute**;  
 **display**: **none**;  
 **z-index**:1000;  
 }  
 **#title**{  
 **background-color**: **dodgerblue**;  
 **color**: **white**;  
 **height**: 30**px**;  
 **line-height**: 30**px**;  
 **padding**: 5**px**;  
 **text-align**: **left**;  
 **vertical-align**: **middle**;  
 **border-top-right-radius**: 5**px**;  
 **border-top-left-radius**: 5**px**;  
 **position**: **relative**;  
 }  
 .**inner**{  
 **padding**: 10**px**;  
 **background-color**: **#ffffff**;  
 **text-align**: **center**;  
 **height**:120**px**;  
 }  
 .**center**{  
 **text-align**: **center**;  
 **vertical-align**: **middle**;  
 **height**: 70**px**;  
 **line-height**: 70**px**;  
 }  
 **#namegroup**{  
 **display**: **none**;  
 }  
 **#button**{  
 **height**: 40**px**;  
 }  
 .**btn**{  
 **background-color**: **#0066cc**;  
 **height**: 30**px**;  
 **color**: **#ffffff**;  
 **border**: **none**;  
 **border-radius**: 5**px**;  
 **padding**: 5**px** 10**px**;  
 **border-radius**: 5**px**;  
 }  
 **#cancel**{  
 **margin-left**: 10**px**;  
 **display**: **none**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="memu"**>  
 <**a href="javascript:***msgalert***('徐xx要去上班啦')"**>alert</**a**>  
 <**a href="javascript:***msgalert***('LMM你确定去上班吗？',function(t){alert(t)})"**>confirm</**a**>  
 <**a href="javascript:***msgalert***('姓名：',function(t){alert(t)},'匿名')"**>prompt</**a**>  
</**div**>  
  
 *<!-- style="display: none"-->* <**div id="dialogbg"**></**div**>  
 <**div id="dialog"** >  
 <**div id="title"**>消息</**div**>  
 <**div class="inner"**>  
 <**div class="center" id="msg"**>  
 </**div**>  
 <**div class="center" id="namegroup"**>  
 <**label id="namelabel" for="name"**></**label**>  
 <**input type="text" id="name"**>  
 </**div**>  
 <**div id="button"**>  
 <**button class="btn" id="confirm"**>确定</**button**>  
 <**button class="btn" id="cancel"**>取消</**button**>  
 </**div**>  
 </**div**>  
</**div**>  
<**script**>  
 **var *EventUtil***={  
 addHandler:**function**(ele,type,handler){  
 **if**(ele.addEventListener){  
 ele.addEventListener(type,handler,**false**);  
 }**else if**(ele.attachEvent){  
 ele.attachEvent(**'on'**+type,handler)  
 }**else** {  
 ele[**"on"** + type] = handler;  
 }  
 },  
 removeHandler:**function**(ele,type,handler){  
 **if**(ele.removeEventListener){  
 ele.removeEventListener(type,handler,**false**);  
 }**else if**(ele.detachEvent){  
 ele.detachEvent(**'on'**+type,handler)  
 }**else** {  
 ele[**"on"** + type] = **null**;  
 }  
 },  
 getEvent:**function**(eve){  
 **return event**!==**null**?**event**:**window**.**event**;  
 },  
 getTarget:**function**(event){  
 **return** event.**target**||event.**srcElement**;  
 },  
 preventDefault:**function**(event){  
 **if**(event.preventDefault){  
 event.preventDefault();  
 }**else**{  
 event.**returnValue**=**false**;  
 }  
 },  
 stopPropagation:**function**(event){  
 **if**(event.stopPropagation){  
 event.stopPropagation()  
 }**else**{  
 event.**cancelBubble**=**true**;  
 }  
 }  
 }  
 **function** *msgalert*(msg,func,content){  
 */\* var text=document.createTextNode(msg);\*/* **var** dialogbg=**document**.getElementById(**'dialogbg'**);  
 **var** dialog=**document**.getElementById(**'dialog'**);  
 **var** msgDiv=**document**.getElementById(**'msg'**);  
 **var** confirmButton=**document**.getElementById(**'confirm'**);  
 **var** cancelButton=**document**.getElementById(**'cancel'**);  
 **var** name=**document**.getElementById(**'name'**);  
 **var** namegroup=**document**.getElementById(**'namegroup'**);  
 **var** namelabel=**document**.getElementById(**'namelabel'**);  
 **var** *show*=**function**(){  
 dialogbg.**style**.**display**=**'block'**;  
 dialog.**style**.**display**=**'block'**;  
 dialog.**style**.**left**=(dialogbg.**offsetWidth**-dialog.**offsetWidth**)/2+**'px'**;  
 dialog.**style**.**top**=(dialogbg.**offsetHeight**-dialog.**offsetHeight**)/2+**'px'**;  
 }  
 **var** *hide*=**function**(){  
 dialog.**style**.**display**=**'none'**;  
 dialogbg.**style**.**display**=**'none'**;  
 cancelButton.**style**.**display**=**'none'**;  
 namegroup.**style**.**display**=**'none'**;  
 msgDiv.**style**.**display**=**'block'**;  
 }  
 **if**(func&&content){  
 msgDiv.**style**.**display**=**'none'**;  
 namegroup.**style**.**display**=**'block'**;  
 cancelButton.**style**.**display**=**'inline-block'**;  
 namelabel.**innerHTML**=msg;  
 name.**value**=content;  
  
 confirmButton.onclick=**function**(){  
 **console**.log(name.**value**);  
 **var** nameValue=(name.**value**)?name.**value**:content;  
 func(nameValue);  
 *hide*();  
 };  
 cancelButton.onclick=**function**(){  
 func(content);  
 *hide*();  
 };  
 }**else if**(func){  
 msgDiv.**innerHTML**=msg;  
 cancelButton.**style**.**display**=**'inline-block'**;  
 confirmButton.onclick=**function**(){  
 func(**true**);  
 *hide*();  
 };  
 cancelButton.onclick=**function**(){  
 func(**false**);  
 *hide*();  
 };  
 }**else**{  
 msgDiv.**innerHTML**=msg;  
 confirmButton.onclick=**function**(){  
 *hide*();  
 };  
 }  
 *show*();  
 */\*添加拖动事件\*/  
 drag*(dialog,dialogbg);  
 */\*添加窗口大小变化事件\*/* ***EventUtil***.addHandler(**window**,**'resize'**,**function**(){  
 dialog.**style**.**left**=(dialogbg.**offsetWidth**-dialog.**offsetWidth**)/2+**'px'**;  
 dialog.**style**.**top**=(dialogbg.**offsetHeight**-dialog.**offsetHeight**)/2+**'px'**;  
 });  
  
 **function** *drag*(dialog,dialogbg){  
 **var** diffX=0,diffY= 0;  
 **var** doc=**window**.**document**;  
 **var** titleDiv=**window**.**document**.getElementById(**'title'**);  
 ***EventUtil***.addHandler(titleDiv,**'mousedown'**,*mouseDown*);  
 **function** *mouseDown*(e){  
 ***EventUtil***.stopPropagation(e);  
 ***EventUtil***.preventDefault(e);  
 e=***EventUtil***.getEvent(**event**);  
 dialog.**style**.**cursor**=**'move'**;  
 **if**(e.setCapture){  
 ***EventUtil***.addHandler(dialog,**'losecapture'**,*mouseUp*);  
 }**else**{  
 ***EventUtil***.addHandler(**window**,**'blur'**,*mouseUp*);  
 }  
 diffX= e.**clientX**-dialog.**offsetLeft**;  
 diffY= e.**clientY**-dialog.**offsetTop**;  
 ***EventUtil***.addHandler(doc,**'mousemove'**,*mouseMove*);  
 ***EventUtil***.addHandler(doc,**'mouseup'**,*mouseUp*);  
 }  
 **function** *mouseMove*(e){  
 **var** maxX=dialogbg.**offsetWidth**-dialog.**offsetWidth**,  
 maxY=dialogbg.**offsetHeight**-dialog.**offsetHeight**;  
 **var** area={  
 **minX**:0,  
 **maxX**:maxX,  
 **minY**:0,  
 **maxY**:maxY  
 }  
 **var** moveX,moveY;  
 moveX=(e.**clientX**-diffX);  
 moveY=(e.**clientY**-diffY);  
 (moveX<area.**minX**)&&(moveX=area.**minX**);  
 (moveX>area.**maxX**)&&(moveX=area.**maxX**);  
 (moveY<area.**minY**)&&(moveY=area.**minY**);  
 (moveY>area.**maxY**)&&(moveY=area.**maxY**);  
 dialog.**style**.**left**=moveX +**'px'**;  
 dialog.**style**.**top**=moveY+**'px'**;  
 }  
 **function** *mouseUp*(e){  
 dialog.**style**.**cursor**=**'default'**;  
 **if**(e.setCapture){  
 ***EventUtil***.addHandler(dialog,**'losecapture'**,*mouseUp*);  
 }**else**{  
 ***EventUtil***.addHandler(**window**,**'blur'**,*mouseUp*);  
 }  
 ***EventUtil***.removeHandler(doc,**'mousemove'**,*mouseMove*);  
 ***EventUtil***.removeHandler(doc,**'mouseup'**,*mouseUp*);  
 }  
 }  
  
 }  
</**script**>  
  
  
</**body**>  
</**html**>

# JQuery

## jQuery语法:

基础语法是：**$(selector).action()**

* 美元符号定义 jQuery
* 选择符（selector）“查询”和“查找” HTML 元素
* jQuery 的 action() 执行对元素的操作

这是为了防止文档在完全加载之前运行jQuery代码

$(document).ready(

function(){

});

### 元素选择器：

$(“p”)选取<p>元素

$(“p.intro”)选取所有class=”intro”的p元素

$(“p#demo”)选取所有id=”demo”的p元素

### 属性选择器：

$(“[href]”)选取所有带有href属性的元素。

$(“[href=’#’]”)选择所有带有href值等于”#”的元素。

$(“[href!=’#’]”)选择所有带有href值不等于”#”的元素。

$(“[href$=’.jpg’]”)选取所有href值以”.jpg”结尾的元素。

### jQuery CSS选择器

jQuery CSS选择器可用于改变html元素的CSS属性

$(“p”).css(“background-color”,”red”)

### jQuery事件

$(document).ready(function) 将函数绑定到文档的就绪事件（当文档完成加载时）

$(selector).click(function) 触发或将函数绑定到被选元素的点击事件

$(selector).dblclick(function) 触发或将函数绑定到被选元素的双击事件

$(selector).focus(function) 触发或将函数绑定到被选元素的获得焦点事件

$(selector).mouseover(function) 触发或将函数绑定到被选元素的鼠标悬停事件

### jQuery hide()和show()

语法

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

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

可选的speed参数规定隐藏/显示的速度，可以去以下值：“slow”、“fast”或毫秒

可选的callback参数是隐藏或显示完成后所执行的函数名称。

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**script type="text/javascript"**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#hide"**).click(**function**(){  
 *$*(**"p"**).hide(3000,**function**(){  
 *$*(**"#show"**).hide();  
 });  
 });  
 *$*(**"#show"**).click(**function**(){  
 *$*(**"p"**).show();  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**p id="p1"**>如果点击“隐藏”按钮，我就会消失。</**p**>  
<**button id="hide" type="button"**>隐藏</**button**>  
<**button id="show" type="button"**>显示</**button**>  
</**body**>  
</**html**>

### toggle()切换hide()和show()方法

显示被隐藏的元素，并隐藏已显示的元素

<script type="text/javascript">

$(document).ready(function(){

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

$("p").toggle();

});

})

</script>

</head>

<body>

<button type="button">切换</button>

<p>这是一个段落。</p>

<p>这是另一个段落。</p>

</body>

### 淡入淡出

（1）fadeIn()淡入

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

（2）fadeout()淡出$(selector).fadeOut(speed,callback);

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是 fading 完成后所执行的函数名称。

1. fadeToggle()淡入淡出切换

语法：$(selector).fadeToggle(speed,callback);

可选的speed参数规定效果时长。可以取值“slow”、“fast”或者毫秒

可选的callback参数表示fading完成后所执行的函数名称

如果元素已经淡出，则添加淡入效果

如果元素已经淡入，则添加淡出效果

1. fadeTo()允许渐变为给定的透明度（值介于0与1之间）

语法：$(selector).fadeTo(speed,opacity,callback);

必需的speed参数规定效果时长

必需的opacity参数将淡入淡出效果设置为给定的不透明度

可选的callback参数是该函数完成后所执行的函数名称。

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **margin-top**: 20**px**;  
 }  
 **#div1**{  
 **background-color**: **#5ecc49**;  
 }  
 **#div2**{  
 **background-color**: **#00edbc**;  
 }  
 **#div3**{  
 **background-color**: **#efed24**;  
 }  
 </**style**>  
 <**script type="text/javascript"**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#fadeIn"**).click(**function**(){  
 *$*(**"#div1"**).**fadeIn**();  
 *$*(**"#div2"**).**fadeIn**(2000);  
 *$*(**"#div3"**).**fadeIn**(3000);  
 });  
 *$*(**"#fadeOut"**).click(**function**(){  
 *$*(**"#div1"**).**fadeOut**();  
 *$*(**"#div2"**).**fadeOut**(2000);  
 *$*(**"#div3"**).**fadeOut**(3000);  
 });  
 *$*(**"#fadeOut"**).click(**function**(){  
 *$*(**"#div1"**).**fadeOut**();  
 *$*(**"#div2"**).**fadeOut**(2000);  
 *$*(**"#div3"**).**fadeOut**(3000);  
 });  
 *$*(**"#fadeToggle"**).click(**function**(){  
 *$*(**"#div1"**).**fadeToggle**();  
 *$*(**"#div2"**).**fadeToggle**(2000);  
 *$*(**"#div3"**).**fadeToggle**(3000);  
 });  
 *$*(**"#fadeTo"**).click(**function**(){  
 *$*(**"#div1"**).fadeTo(2000,0.3);  
 *$*(**"#div2"**).fadeTo(2000,0.5);  
 *$*(**"#div3"**).fadeTo(2000,0.7);  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="fadeIn"**>fadeIn</**button**>  
<**button id="fadeOut"**>fadeOut</**button**>  
<**button id="fadeToggle"**>fadeToggle</**button**>  
<**button id="fadeTo"**>fadeTo</**button**>  
<**div id="div1"**></**div**>  
<**div id="div2"**></**div**>  
<**div id="div3"**></**div**>  
</**body**>  
</**html**>

### 滑动

slideDown();

slideUp();

slideToggle();

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#slideDown"**).click(**function**(){  
 *$*(**"#div1"**).**slideDown**(2000);  
 });  
 *$*(**"#slideUp"**).click(**function**(){  
 *$*(**"#div1"**).**slideUp**(2000);  
 });  
 *$*(**"#slideToggle"**).click(**function**(){  
 *$*(**"#div1"**).**slideToggle**(2000);  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="slideDown"**>slideDown</**button**>  
<**button id="slideUp"**>slideUp</**button**>  
<**button id="slideToggle"**>slideToggle</**button**>  
<**div id="div1"**></**div**>  
  
</**body**>  
</**html**>

### 动画animate方法

用于创建自定义动画

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

必需的 params 参数定义形成动画的 CSS 属性。

可选的 speed 参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。

可选的 callback 参数是动画完成后所执行的函数名称。

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 **position**: **absolute**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#animate"**).click(**function**(){  
 *$*(**"#div1"**).animate({ **height**:**'+=150px'**,  
 **width**:**'+=150px'**,**left**:**'200px'** },3000);  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="animate"**>animate</**button**>  
<**div id="div1"**></**div**>  
</**body**>  
</**html**>

### stop()方法

$(selector).stop(stopAll,goToEnd);

可选的stopAll参数规定是否应该清楚动画队列。默认为false即立即停止活动的动画

可选的goToEnd参数规定是否立即完成当前动画。默认false

### 获得内容text()/html()以及val()

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 **position**: **absolute**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#btn1"**).click(**function**(){  
 alert(*$*(**"#test"**).text())  
 });  
 *$*(**"#btn2"**).click(**function**(){  
 alert(*$*(**"#test"**).html())  
 });  
 *$*(**"#btn3"**).click(**function**(){  
 alert(*$*(**"#input"**).val())  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**p id="test"**>这是段落中的<**b**>粗体</**b**>文本。</**p**>  
<**input id="input" type="text" value="这是value"**>  
<**button id="btn1"**>显示文本</**button**>  
<**button id="btn2"**>显示 HTML</**button**>  
<**button id="btn3"**>显示 val</**button**>  
</**body**>  
</**html**>

### 获取属性attr()

jQuery attr()方法用于获取属性

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

alert($("#w3s").attr("href"));

});

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 **position**: **absolute**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#btn1"**).click(**function**(){  
 *$*(**"#test"**).text(**"设置text"**);  
 });  
 *$*(**"#btn2"**).click(**function**(){  
 *$*(**"#test"**).html(**"<em>设置html</em>"**);  
 });  
 *$*(**"#btn3"**).click(**function**(){  
 *$*(**"#input"**).val(**"设置value"**);  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**p id="test"**>这是段落中的<**b**>粗体</**b**>文本。</**p**>  
<**input id="input" type="text" value="这是value"**>  
<**button id="btn1"**>显示文本</**button**>  
<**button id="btn2"**>显示 HTML</**button**>  
<**button id="btn3"**>显示 val</**button**>  
</**body**>  
</**html**>

**修改attr**

**回调函数由两个参数：被选元素列表中当前元素的下标，以及原始（旧的）值。然后以函数新值返回您希望使用的字符串。**

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 **position**: **absolute**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#btn1"**).click(**function**(){  
 *$*(**"a"**).attr(**"href"**,**function**(i,oldval){  
 **return** oldval+**"/new"**+i;  
 });  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**a href="baidu.com"**>baidu.com0</**a**>  
<**a href="baidu.com"**>baidu.com1</**a**>  
<**button id="btn1"**>更改属性</**button**>  
</**body**>  
</**html**>

### 添加元素

append()在被选元素的结尾插入内容

prepend()在被选元素的开头插入内容

after()在被选元素之后插入内容

before()在被选元素之前插入内容

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 **position**: **absolute**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 **var** p1=**"<p>Text1</p>p"**;  
 **var** p2=*$*(**"<p></p>"**).text(**"Text2."**);  
 **var** p3=**document**.createElement(**"p"**); *// 以 DOM 创建新元素* p3.**innerHTML**=**"Text."**;  
 *$*(**"#div1"**).append(p1,p2,p3)  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**div id="div1"**></**div**>  
</**body**>  
</**html**>

### 删除元素

remove()删除所选元素（及其1·93子元素）

$(“#div1”).remove();

empty()从被选元素中删除子元素

$(“div1”).empty();

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 **background-color**: **#009cd9**;  
 **position**: **absolute**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#remove"**).click(**function**(){  
 *$*(**"#div1"**).remove();  
 });  
 *$*(**"#empty"**).click(**function**(){  
 *$*(**"#div1"**).empty();  
 });  
  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="remove"**>remove</**button**>  
<**button id="empty"**>empty</**button**>  
<**div id="div1"**>  
 <**p**>this is p </**p**>  
 <**p**>this is p </**p**>  
 <**p**>this is p </**p**>  
 <**p**>this is p </**p**>  
</**div**>  
</**body**>  
</**html**>

### 获取设置CSS类

addClass()向被选元素添加一个或多个类

removeClass()从被选元素删除一个或多个类

toggleClass()对被选元素进行添加/删除的切换操作

css()设置或返回样式属性。

返回CSS属性：css(“propertyname”);

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **width**: 200**px**;  
 **height**: 100**px**;  
 }  
 .**back**{  
 **background-color**: **#ea4444**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#addClass"**).click(**function**(){  
 *$*(**"#div1"**).addClass(**"back"**);  
 });  
 *$*(**"#removeClass"**).click(**function**(){  
 *$*(**"#div1"**).removeClass(**"back"**);  
 });  
 *$*(**"#toggleClass"**).click(**function**(){  
 *$*(**"#div1"**).toggleClass(**"back"**);  
 });  
 *$*(**"#css"**).click(**function**(){  
 *$*(**"#div1"**).css({**"background-color"**:**"green"**,**"font-size"**:**"20px"**})  
 });  
 *$*(**"#css1"**).click(**function**(){  
 alert( *$*(**"#div1"**).css(**"background-color"**));  
 });  
  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="addClass"**>addClass</**button**>  
<**button id="removeClass"**>removeClass</**button**>  
<**button id="toggleClass"**>toggleClass</**button**>  
<**button id="css"**>css</**button**>  
<**button id="css1"**>css1</**button**>  
<**div id="div1"**>  
 <**p**>this is p </**p**>  
</**div**>  
</**body**>  
</**html**>

### 尺寸

width()方法设置或返回元素的宽度（不包括内边距、边框或外边距）

height()方法设置或返回元素的高度（不包括内边距、边框或外边距）

innerWidth()返回元素宽度（包括内边距）

innerHeight()返回元素高度（包括内边距）

outerWidth()返回元素宽度（包括内边距、边框）

outerHeight()返回元素高度（包括内边距、边框）

outerWidth(true)返回元素宽度（包括内边距、边框、外边距）

outerHeight(true)返回元素高度（包括内边距、边框、外边距）

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 */\*box-sizing:border-box;\*/* **width**: 300**px**;  
 **height**: 300**px**;  
 **padding**: 50**px**;  
 **border**: 20**px solid #808080**;  
 **margin**: 30**px**;  
 **display**: **table-cell**;  
 **vertical-align**: **middle**;  
 **text-align**: **center**;  
 }  
 .**back**{  
 **background-color**: **#ea4444**;  
 }  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#height"**).click(**function**(){  
 alert(*$*(**"#div1"**).height());  
 });  
 *$*(**"#innerHeight"**).click(**function**(){  
 alert(*$*(**"#div1"**).innerHeight());  
 });  
 *$*(**"#outerHeight"**).click(**function**(){  
 alert(*$*(**"#div1"**).outerHeight());  
 });  
 *$*(**"#outerHeightTrue"**).click(**function**(){  
 alert(*$*(**"#div1"**).outerHeight(**true**));  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="height"**>height</**button**>  
<**button id="innerHeight"**>innerHeight</**button**>  
<**button id="outerHeight"**>outerHeight</**button**>  
<**button id="outerHeightTrue"**>outerHeight-True</**button**>  
<**div id="div1"**>  
 width: 300px;<**br**>  
 height: 300px;<**br**>  
 padding: 50px;<**br**>  
 border: 20px solid #808080;<**br**>  
 margin: 30px;  
</**div**>  
</**body**>  
</**html**>

### 遍历

祖先：

parent()方法返回被选元素的直接父元素

parents()方法返回被选元素的所有祖先元素，一路向上知道节点的根元素<html>

parentsUntil()方法返回介于两个给定元素之间的所有祖先元素

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **height**: 200**px**;  
 **width**: 200**px**;  
 **border**: **solid** 1**px #808080**;  
 **text-align**: **center**;  
 **display**: **table-cell**;  
 **vertical-align**: **middle**;  
 }  
 **ul**{  
 **height**: 100**px**;  
 **width**: 100**px**;  
 **border**: **solid** 1**px #808080**;  
 **margin**: **auto**;  
 }  
 **li**{  
 **height**: 50**px**;  
 **width**: 50**px**;  
 **border**: **solid** 1**px #808080**;  
 }  
 **span**{  
 **height**: 20**px**;  
 **width**: 20**px**;  
 **border**: **solid** 1**px #808080**;  
 **text-align**: **center**;  
 }  
  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#parent"**).click(**function**(){  
 *$*(**"span"**).parent().css({**"border"**:**"1px solid red"**});  
 });  
 *$*(**"#parents"**).click(**function**(){  
 *$*(**"span"**).parents().css({**"border"**:**"1px solid blue"**});  
 });  
 *$*(**"#parentsUntil"**).click(**function**(){  
 *$*(**"span"**).parentsUntil(**"div"**).css({**"border"**:**"1px solid #aadd33"**});  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="parent"**>parent</**button**>  
<**button id="parents"**>parents</**button**>  
<**button id="parentsUntil"**>parentsUntil</**button**>  
<**div id="div1"**>  
<**ul**>祖父ul  
 <**li**>父亲li  
 <**span**>span</**span**>  
 </**li**>  
</**ul**>  
</**div**>  
</**body**>  
</**html**>

### 向下遍历DOM树

**children()返回被选元素的所有直接子元素**

$(“#div1”).children()返回div1的所有直接子元素

$(“#div1”).children(“p”)返回div1直接子元素中的p元素

**find()方法返回被选元素的后代元素，一路向下直到最后一个后代元素**

$(“#div1”).children(“\*”)返回div1元素的所有子元素

$(“#div1”).children(“p”)返回div1元素所有子元素中的p元素

### 水平遍历

**siblings()返回所选元素的所有同胞元素**

$(“h2”).siblings()返回h2的所有同胞元素

**next()返回下一个同胞元素**

$(“h2”).next()返回h2的下一个同胞元素

**nextAll()返回被选元素的所有跟随的同胞元素**

$(“h2”).nextAll()返回h2的所有跟随的同胞元素

**nextUntil()返回介于两个给定参数之间所有跟随的同胞元素**

$(“h2”).nextUntil(“p”)返回介于h2和p元素之间的所有跟随子元素

prev()返回上一个同胞元素

prevAll()返回被选元素的前面所有同胞元素

preveUntil()返回介于两个给定参数之间所有前面的同胞元素

### 元素过滤

first(),

last()

eq()

filter()

not()

### AJAX

jQuery load()方法是简单强大的AJAX方法

load()方法从服务器加载数据，并把返回的数据放入被选元素中

$(selector).load(URL,data,callback);

其中URL参数是规定希望加载的URL

可选的data参数规定与请求一同发送的查询字符串键/值对集合

可选的callback参数是load()方法完成后所执行的函数名称

回调函数参数如下：

responseText包含调用成功时的结果内容

statusTXT包含调用的状态

xhr包含XMLHttpRequest对象

<!DOCTYPE **html**>  
<**html**>  
<**head lang="zh"**>  
 <**script src="../lib/jquery-2.1.3.js"**></**script**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"** />  
<**style**>  
 **div**{  
 **height**: 200**px**;  
 **width**: 200**px**;  
 **border**: **solid** 1**px #808080**;  
 **text-align**: **center**;  
 **display**: **table-cell**;  
 **vertical-align**: **middle**;  
 }  
 **ul**{  
 **height**: 100**px**;  
 **width**: 100**px**;  
 **border**: **solid** 1**px #808080**;  
 **margin**: **auto**;  
 }  
 **li**{  
 **height**: 50**px**;  
 **width**: 50**px**;  
 **border**: **solid** 1**px #808080**;  
 }  
 **span**{  
 **height**: 20**px**;  
 **width**: 20**px**;  
 **border**: **solid** 1**px #808080**;  
 **text-align**: **center**;  
 }  
  
</**style**>  
 <**script**>  
 *$*(**document**).ready(**function**(){  
 *$*(**"#load"**).click(**function**(){  
 *$*(**"#div1"**).load(**"a.html"**)  
  
 });  
 });  
 </**script**>  
</**head**>  
<**body**>  
<**button id="load"**>load</**button**>  
<**div id="div1"**>  
</**div**>  
</**body**>  
</**html**>

# HTML

## 面试题

### 什么是语义化的HTML?有何意义？为什么要做到语义化。

根据内容的结构化选择合适的标签，便于开发者阅读和写出更优雅的代码的同时让浏览器的爬虫和机器很好地解析。

为什么要语义化：

（1）在没有CSS的情况下，页面也能很好地呈现内容结构

（2）用户体验：如title、alt用于解释名词或图片信息、label标签活用

（3）有利于SEO：和搜索引擎建立良好的沟通，有利于爬虫获取更多有利信息，爬虫依赖标签来确定上下文和各个关键字的权重；

（4）便于其他设备的解析（如屏幕阅读器、盲人阅读器、移动设备）根据标签语义来渲染页面；

（5）便于团队开发和维护，

### html5缓存机制

### 加载顺序

http://renyongjie668.blog.163.com/blog/static/1600531201097062789/

### html5标签拖拽

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**></**title**>  
 <**style**>  
 **#div1**{  
 **border**: 1**px solid #808080**;  
 **height**: 200**px**;  
 **width**: 200**px**;  
 }  
 **#img**{  
 **width**: 100**px**;  
 **height**: **auto**;  
 }  
 </**style**>  
  
</**head**>  
<**body**>  
<**div id="div1" ondragover="allowDrop(event)" ondrop="drop(event)"**>  
</**div**>  
<**img id="img" src="../images/roll1.png" draggable="true" ondragstart="drag(event)"**>  
</**body**>  
<**script**>  
 **var** *drag*=**function**(event){  
 event.dataTransfer.setData(**'Text'**,event.**target**.**id**);  
 }  
 **var** *allowDrop*=**function**(event){  
 event.preventDefault();  
 }  
 **var** *drop*= **function** (event) {  
 event.preventDefault();  
 **var** data=event.dataTransfer.getData(**'Text'**);  
 event.**target**.appendChild(**document**.getElementById(data));  
 }  
</**script**>  
</**html**>

# CSS

## 常用样式总结

### text-decoration

值：

overlinne:定义文本上一条线

line-through：定义穿过文本一条线

underline：定义文本下的一条线

blink：定义闪烁的文本

none：默认定义标准的文本

### 不换行

white-space:nowrap;

### text-overflow 属性规定当文本溢出包含元素时发生的事情。

text-overflow: clip|ellipsis|*string*;

clip：修剪文本

ellipsis：用省略号代替

string：使用特定的字符串来代替修剪的文本；

### 鼠标覆盖、焦点样式

.**navbar** .**search-query**:**hover**, .**navbar** .**search-query**:**focus**{  
 **background**:**#ffffff**;  
 **transition**:0.5**s** ;  
}

上述代码的意思是在鼠标覆盖或者处于焦点时将背景改为白色，时延为0.5s;

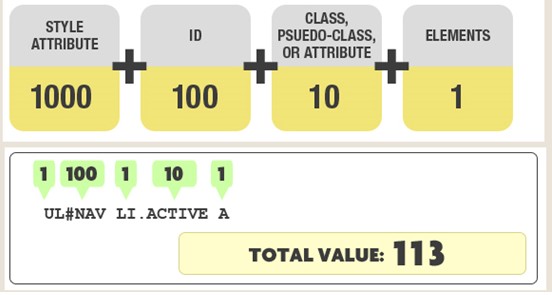
### transition

CSS3过渡效果，如下代码为background属性过渡

.**navbar** .**search-query**:**hover**, .**navbar** .**search-query**:**focus**{  
 **background**:**#ffffff**;  
 **transition**: **background** 0.5**s** ;  
 **-webkit-transition**: **background** 0.5**s** ;*/\*chrome和Safari\*/* **-moz-transition**: **background** 0.5**s** ;*/\* Firefox 4\*/* **-o-transition**: **background** 0.5**s**;*/\*Opera\*/*}

### 选择器优先级、权重

参考：<http://www.nowamagic.net/csszone/css_SeletorPriorityRules.php>



4个等级定义如下：

第一等：代表内联样式，如style=””，权值为1000；

第二等：代表ID选择器，如：#content，权值为100；

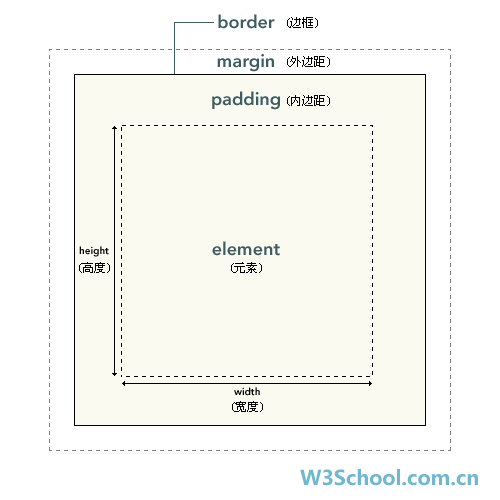
第三等：代表类，伪类和属性选择器，如.content，权值为10；

第四等：代表类型选择器和伪元素选择器，如div p，权值为1；

### 盒子模型

参考：<http://www.w3school.com.cn/css/css_boxmodel.asp>

CSS框模型（Box Model）规定了元素框处理元素内容、内边距、边框、外边距的方式。



box-sizing:content-box||border-box||inherit

1. content-box:此值为其默认值，让其元素维持W3C的标准Box Model，也就是元素的高度等于元素边框高度（border）+元素内边距（padding）+内容的高度（content width/height）
2. border-box：此值让元素维持IE传统的Box Model

/\*Content box\*/

Element {

-moz-box-sizing: content-box; /\*Firefox3.5+\*/

-webkit-box-sizing: content-box; /\*Safari3.2+\*/

-o-box-sizing: content-box; /\*Opera9.6\*/

-ms-box-sizing: content-box; /\*IE8\*/

box-sizing: content-box; /\*W3C标准(IE9+，Safari5.1+,Chrome10.0+,Opera10.6+都符合box-sizing的w3c标准语法)\*/

}

/\*Border box\*/

Element {

-moz-box-sizing: border-box; /\*Firefox3.5+\*/

-webkit-box-sizing: border-box; /\*Safari3.2+\*/

-o-box-sizing: border-box; /\*Opera9.6\*/

-ms-box-sizing: border-box; /\*IE8\*/

box-sizing: border-box; /\*W3C标准(IE9+，Safari5.1+,Chrome10.0+,Opera10.6+都符合box-sizing的w3c标准语法)\*/

}

### 动画

Internet Explorer 10、Firefox 以及 Opera 支持 @keyframes 规则和 animation 属性。

Chrome 和 Safari 需要前缀 -webkit-。

**注释：**Internet Explorer 9，以及更早的版本，不支持 @keyframe 规则或 animation 属性。

动画规则定义@keyframes

**@keyframes padding-bottom** {  
 **from**{**padding-bottom**: 8**px**}  
 **to** {**padding-bottom**: 5**px**}  
}  
**@-moz-keyframes padding-bottom** {  
 **from**{**padding-bottom**: 8**px**}  
 **to** {**padding-bottom**: 5**px**}  
 }  
**@-webkit-keyframes padding-bottom** {  
 **from**{**padding-bottom**: 8**px**}  
 **to** {**padding-bottom**: 5**px**}  
}  
**@-o-keyframes padding-bottom** {  
 **from**{**padding-bottom**: 8**px**}  
 **to** {**padding-bottom**: 5**px**}  
 }

代码意思：定义一个动画规则padding-bottom（规则名称），做的事情是一开始将元素CSS属性padding-bottom从8px变到5px

动画应用：animation

例如：

.**underline a**:**hover**{  
 **border-bottom**: **solid** 2**px #ffffff**;  
 **animation**: **padding-bottom** 0.3**s**;  
 **-o-animation**: **padding-bottom** 0.3**s**;*/\*Opera\*/* **-moz-animation**: **padding-bottom** 0.3**s**;*/\*Firefox\*/* **-webkit-animation**:**padding-bottom** 0.3**s**; */\*Safari chrome\*/*}

大意：对.uunderline 下的<a>鼠标放置时，执行padding-bottom规则，时长为0.3秒

### 伪元素:before :after

在一个元素的前边或者后边加上一个元素；

可用作图标、消除浮动。

.**clearfix**:**after**{  
 **content**: **''**;  
 **display**: **block**;  
 **clear**: **both**;  
}  
.**clearfix**{  
 **zoom**: 1;  
}

### Position样式

postion元素规定元素的定位类型。

可能的值：

|  |  |
| --- | --- |
| **值** | **描述** |
| absolute | 生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。  元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。 |
| fixed | 生成绝对定位的元素，相对于浏览器窗口进行定位。  元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。 |
| relative | 生成相对定位的元素，相对于其正常位置进行定位。  因此，"left:20" 会向元素的 LEFT 位置添加 20 像素。 |
| static | 默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。 |
| inherit | 规定应该从父元素继承 position 属性的值。 |

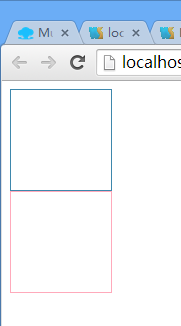
首先需要了解文档流的概念，浏览器页面显示元素横的竖的一个个接一个排称为文档流。

**static默认值，没有定位**

上代码说明问题：

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 **#div1**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #3e87ac**;  
 }  
 **#div2**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #ffaabb**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="div1"**></**div**>  
<**div id="div2"**></**div**>  
  
</**body**>  
</**html**>

没有给div块设置任何position样式



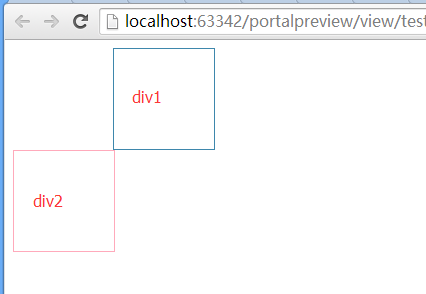
**relative样式**

生成相对定位的元素，相对于其正常位置进行定位。

给div添加

**position**: **relative**;  
**left**: 100**px**;

过后效果如下：

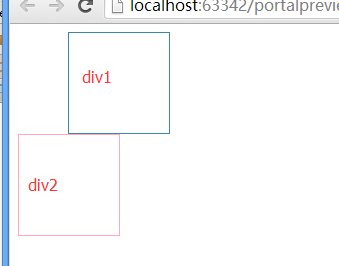


可以看到div1相对于正常的位置向右偏移了100px。

那么再给div1设置margin值呢

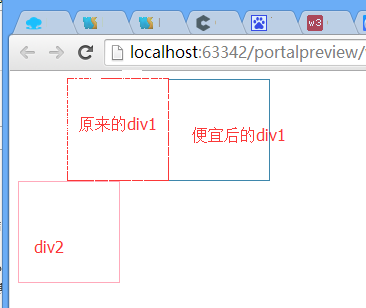
先给div1加个margin-left:50px

**#div1**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #3e87ac**;  
 **margin**:0 50**px**;  
}  
**#div2**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #ffaabb**;  
}



在此基础上设置div1

**position**: **relative**;  
**left**: 100**px**;



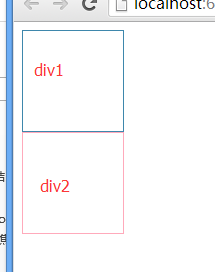
瞬间从原来的位置向右偏移了100px有没有。

总之一句话，relative就是基于它本来应该在的位置偏移XX值啦，那么它对于其他元素有没有影响呢，在来瞅瞅，基于刚刚的例子让div1向下偏移。

初始状态：

<**style type="text/css"**>  
 **#div1**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #3e87ac**;  
 }  
 **#div2**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #ffaabb**;  
 }  
</**style**>

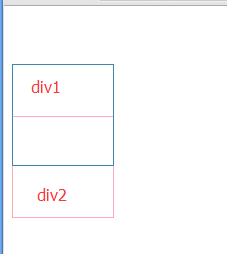
效果：



将div1向下偏移，添加样式

**position**: **relative**;  
**top**: 50**px**;

效果：



对div1相对于它本身应该的在的位置，向下偏移了100px，div2完全不受影响，还在原来的位置哦。

**absolute**

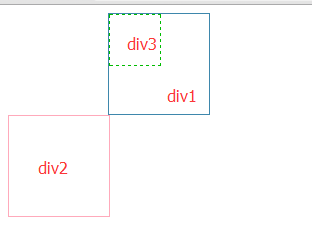
生成绝对定位的元素，**相对于 static 定位以外的第一个父元素进行定位。**

元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定

还是基于上述例子：先给div1定义一个relative好啦，再在div1中添加一个小的div3

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
 **#div1**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #3e87ac**;  
 **position**: **relative**;  
 **left**: 100**px**;  
 }  
 **#div2**{  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: 1**px solid #ffaabb**;  
 }  
 **#div3**{  
 **width**: 50**px**;  
 **height**: 50**px**;  
 **border**: 1**px dashed #00bb00**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="div1"**>  
 <**div id="div3"**></**div**>  
</**div**>  
<**div id="div2"**></**div**>  
  
</**body**>  
</**html**>

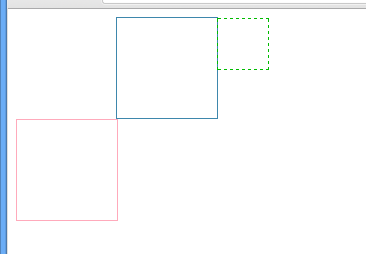
如图所示



此时给div3设置样式

**position**: **absolute**;  
**left**: 100**px**;

那么div3应该为相对于div1(**相对于 static 定位以外的第一个父元素进行定位,应该div1为relative定位)进行定位，**以div1为基准向右偏移了100px**。**



**fixed**

fixed其实就是absolute的一个特例而已，相对于body进行绝对定位。

### placeholder 与提示信息

placeholder属性提供可描述输入字段预期值得提示信息（hint）。

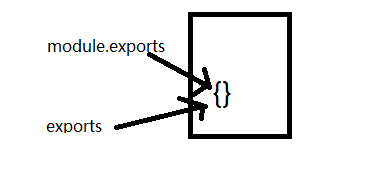
该提示会在输入字段为空时显示，并会在字段获得焦点时消失。

# Nodejs

## 疑难知识点

### module.exports与exports的区别与联系

require返回的事module.exports，而默认module.exports是个空对象，exports变量指向module.exports。所以如果你想复写exports，直接写exports=XXX肯定是不行的，因为这只是改变了变量exports的引用，并没有改变module.exports，所以只能写成module.exports=xxx；



## 常见库

### utility

功能：常见的加密解密功能

使用：var *utility*=require('utility');

**var** sha1Value=***utility***.sha1(q)

### superagent

功能：可以发起 get 或 post 请求

API：<http://visionmedia.github.io/superagent/>

### cheerio

功能：理解成一个 Node.js 版的 jquery，用来从网页中以 css selector 取数据，使用方式跟 jquery 一样一样的。

API：<https://github.com/cheeriojs/cheerio>

### evenproxy

功能：将串行等待变成并行等待，提升多异步协作场景下的执行效率

地址：<https://github.com/JacksonTian/eventproxy>

### async

<https://github.com/alsotang/async_demo>

### 测试should

测试框架 mocha : <http://mochajs.org/>

断言库 should : <https://github.com/tj/should.js>

测试率覆盖工具 istanbul : <https://github.com/gotwarlost/istanbul>

简单 Makefile 的编写 : <http://blog.csdn.net/haoel/article/details/2886>

## Nodejs文件操作

### 小文件拷贝

使用fs.readFileSync从源路径读取文件内容，并使用fs.writeFileSync将文件内容写入目标路径；

代码：

*/\*\*  
 \* Created by Thea on 2014/12/22.  
 \*/***var *fs***=require(**'fs'**);  
**function** *copy*(src,dst){  
 ***fs***.writeFileSync(dst,***fs***.readFileSync(src));  
}  
**function** *main*(argv){  
 *copy*(argv[0],argv[1]);  
 **console**.log(**"complete copy"**)  
}  
*main*(process.argv.slice(2));

其中：process是一个全局变量，可通过process.argv获得命令行参数，由于argv[0]固定等于Nodejs的绝对路径，argv[1]固定等于主模块的绝对路径，因此命令参数从argv[2]开始。

### 大文件拷贝

上边的程序拷贝一些小文件没有啥问题，但这种一次性把所有文件内容都读取到内存中，再一次性写入磁盘的方式不适合拷贝大文件，内存会爆仓。对于大文件，我们只能读一点写一点，直到完成拷贝。

代码：

*/\*\*  
 \* Created by Thea on 2014/12/22.  
 \*/***var *fs***=require(**'fs'**);  
**function** *copy*(src,dst){  
 ***fs***.createReadStream(src).pipe(***fs***.createWriteStream(dst))  
}  
**function** *main*(argv){  
 *copy*(argv[0],argv[1]);  
 **console**.log(**"copy big file complete"**);  
}  
*main*(process.argv.slice(2));

## Stream数据流操作

官方文档： <http://nodejs.org/api/stream.html>

## Path路径

官方文档： <http://nodejs.org/api/path.html>

遍历文件

取文件后缀path.extname(file);

## url

官方文档： <http://nodejs.org/api/url.html>

var url=require(‘url’);

url.url(‘http://assets.example.com/foo/??bar.js,baz.js’)

=foo/??bar.js,baz.js’

## process

引用：var chihild=require(‘child\_process’);

创建进程:

var child = child\_process.spawn('node', [ 'xxx.js' ]);

.spawn(exec, args, options)方法，该方法支持三个参数。第一个参数是执行文件路径，可以是执行文件的相对或绝对路径，也可以是根据PATH环境变量能找到的执行文件名。第二个参数中，数组中的每个成员都按顺序对应一个命令行参数。第三个参数可选，用于配置子进程的执行环境与行为。

进程间的通信：

/\* parent.js \*/

var child = child\_process.spawn('node', [ 'child.js' ]);

child.kill('SIGTERM');

/\* child.js \*/

process.on('SIGTERM', function () {

cleanUp();

process.exit(0);

});

守护子进程：

# 响应式网页设计

参照蒋宇捷的博客（http://blog.csdn.net/hfahe）

响应式网页设计（Responsive web deesign）是一种现代网页设计方法，基于CSS3的媒介查询（Media Query）特性使得网页适应不同设备，即根据设备的分辨率和缩放自动重新布局。

第一步：Meta标签

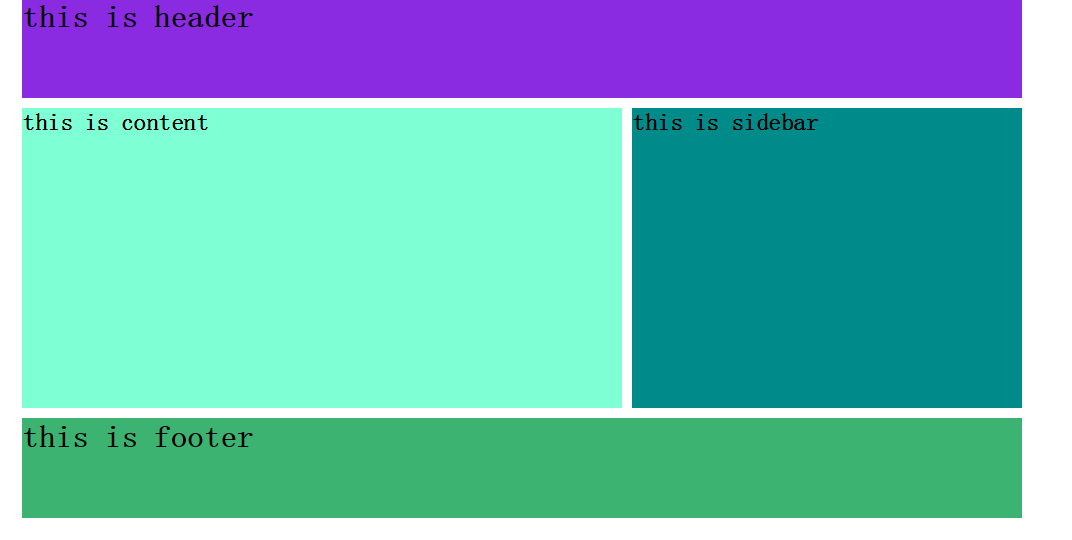
大多数移动浏览器将HTML页面方大为宽的师徒以符合屏幕分辨率。可以使用视图的meta标签进行重置。

**<meta** name="viewport" content="width=device-width, initial-scale=1.0"**>**

IE8或者更早的浏览器并不支持Media Query，可以使用media-queries.js或者respond.js来为IE添加media query支持。

**<script** src="http://css3-mediaqueries-js.googlecode.com/svn/trunk/css3-mediaqueries.js"**></script>**

第二步：HTML布局



代码：

<!DOCTYPE **html**>  
<**html**>  
<**head lang="en"**>  
 <**meta charset="UTF-8"**>  
 <**title**></**title**>  
 <**style type="text/css"**>  
  
 **body**,**h1**,**h2**{  
 **margin**: 0 0;  
 }  
 **#pagewrap**{  
 **margin**: 0 **auto**;  
 **width**: 1000**px**;  
 }  
 **#header**{  
 **width**: 100%;  
 **height**: 100**px**;  
 **margin-bottom**: 10**px**;  
 **background-color**: **blueviolet**;  
 }  
 **#content**{  
 **width**:600**px**;  
 **height**: 300**px**;  
 **margin-right**: 400**px**;  
 **background-color**: **aquamarine**;  
 }  
 **#sidebar**{  
 **width**: 390**px**;  
 **float**: **right**;  
 **height**: 300**px**;  
 **background-color**: **darkcyan**;  
 }  
 **#footer**{  
 **margin-top**: 10**px**;  
 **width**: 100%;  
 **height**: 100**px**;  
 **background-color**: **mediumseagreen**;  
 }  
 */\*for 900 or less,当浏览器宽度≤900px时应用此段代码\*/* **@media screen and** (**max-width**: 900**px**) {  
 **#pagewrap**{  
 **width**: 95%;  
  
 }  
 **#content**{  
 **width**: 63%;  
  
 }  
 **#sidebar**{  
 **width**: 36%;  
 }  
 }  
 */\*for 700 or less,当浏览器宽度≤700px时应用此段代码\*/* **@media screen and** (**max-width**: 700**px**) {  
 **#content**{  
 **margin-right**: **auto**;  
 **width**: 100%;  
 }  
 **#sidebar**{  
 **float**: **none**;  
 **width**: 100%;  
 **margin-bottom**: 10**px**;  
 }  
 }  
 */\*for 480 or less,当浏览器宽度≤480px时应用此段代码\*/* **@media screen and** (**max-width**: 480**px**) {  
 **h1**{  
 **font-size**: 24**px**;  
 }  
 **h2**{  
 **font-size**: 14**px**;  
 }  
 }  
 </**style**>  
 <**script src="../lib/css3-mediaqueries.js"**></**script**>  
</**head**>  
<**body**>  
  
<**div id="pagewrap"**>  
 <**div id="header"**><**h1**>this is header</**h1**></**div**>  
 <**div id="sidebar"**><**h2**>this is sidebar</**h2**></**div**>  
 <**div id="content"**><**h2**>this is content</**h2**></**div**>  
 <**div id="footer"**><**h1**>this is footer</**h1**></**div**>  
</**div**>  
</**body**>  
</**html**>