事件冒泡：

事件捕获->事件触发->事件冒泡

DOM事件流：同时支持两种事件模型：捕获型事件和冒泡型事件，但是，捕获型事件先发生。两种事件流会触及DOM中的所有对象，从document对象开始，也在document对象结束。

addEventListener(event,fn,useCapture)方法，基中第3个参数useCapture是一个Boolean值，用来设置事件是在事件捕获时执行，还是事件冒泡时执行。false表示冒泡，默认。

(IE)用attachEvent(没有最后那个参数，因为没有捕获阶段)方法，此方法没有相关设置，不过IE的事件模型默认是在事件冒泡时执行的.

有些事件是没有冒泡过程的：load、unload、focus、blur

原生js事件绑定的写法：

```
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;
    }
},
```

冒泡的应用：事件代理。target就是你点击的对象，点击以后会从点击对象开始向上传播，一旦上级有绑定事件，就会触发事件。

```
function getEventTag(a){
    var e = a||window.event;
     return e.target || e.srcElement;
}
function editCell(e){
    var target = getEventTag(e);
    if(target.tagName.toLowerCase() === 'td'){
        var _v = target.textContent || target.innerText;
        alert(_v);
    }
}
function bind(elem,type,fn){
    if(elem.attachEvent){
        elem.attachEvent('on' + type,fn);
    }
    if(elem.addEventListener){
        elem.addEventListener(type,fn,false);
    }
}
// function fi(e){
//  editCell(e);
// }
bind($("test"),"click",editCell);
```

mouseover和mouseenter

mouseover（mouseout）：当光标进入该元素或者该元素的子元素就会触发事件。原因是，这个事件会冒泡。

mouseenter（mouseleave）：只有当光标进入该元素才会触发。

mouseenter事件在鼠标进入某个元素，或第一次进入这个元素的某个子元素时触发。一旦触发后，在mouseleave之前，鼠标在这个元素的子元素上触发mouseenter事件都不会触发这个元素的mouseenter事件。即：一旦进入，在子元素间的mouseenter不算是在本元素上的mouseenter。

而mouseover事件是必然冒泡的，一旦子元素mouseover了，本元素必然mouseover（除非子元素上禁止冒泡了）。

阻止事件冒泡：

```
if (e.stopPropagation) {
    e.stopPropagation();
}else{
    window.event.cancelBubble=true;
}
```

补充：取消事件冒泡同时阻止当前节点上的事件处理程序被调用：
e.stopImmediatePropagation()
取消事件默认行为：e.preventDefault()
IE取消事件默认行为：returnValue()

事件触发：
eventTarget.dispatchEvent(type)
fireEvent(e)

原生ajax和jquery ajax：

1、原生ajax：

原生ajax基本可以分为三步：创建、open、send。

ajax对象有两个属性：readyState、status、responseText、responseXML

readyState有五种状态：

0 (未初始化)：(XMLHttpRequest)对象已经创建，但还没有调用open()方法；

1 (载入)：已经调用open() 方法，但尚未发送请求；

2 (载入完成)： 请求已经发送完成；

3 (交互)：可以接收到部分响应数据；

4 (完成)：已经接收到了全部数据，并且连接已经关闭。

status实际是一种辅状态判断，只是status更多是服务器方的状态判断。关于status，由于它的状态有几十种，我只列出平时常用的几种：

100——客户必须继续发出请求

101——客户要求服务器根据请求转换HTTP协议版本

200——成功

201——提示知道新文件的URL

300——请求的资源可在多处得到

301——（删除请求数据？）永久移动，所请求的文档在别的地方；文档新的URL会在定位响应头信息中给出。浏览器会自动连接到新的URL。

302—— 临时移动，定位头信息中所给的URL应被理解为临时交换地址而不是永久的

304—— 自从上次请求后，网页未改变，服务器不会返回网页内容，可以使用浏览器缓存

404——没有发现文件、查询或URI

500——服务器产生内部错误

responseText：作为响应主题被返回的文本。

responseXML：如果响应的内容类型是“text/XML”或者“application/XML”，则这个属性中包含着响应数据。

第一步：创建。

```
function createXHR(){
        if (typeof XMLHttpRequest != "undefined") {//非IE6
            return new XMLHttpRequest();
        }else if (typeof ActiveXObject != "undefined") {//IE6极其以下
            if (typeof arguments.callee.activeXString != "string") {
                var versions = [ "MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0", "MSXML2.XMLH
                    i, len;
                for (i = 0,len = versions.length; i < len; i++) {
                    try{
                        new ActiveXObject(versions[i]);
                        arguments.callee.activeXString = versions[i];
                        break;
                    } catch(ex){
                        //
                    }
                }
            }
            return new ActiveXObject(arguments.callee.activeXString);
        }else {
            throw new Error("No XHR object available.");
        }
    }
```

new XMLHttpRequest()　非IE6

new ActiveXObject(‘Microsoft.XMLHTTP’)　IE6

javascript高级程序中：

```
function createXHR(){
    if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
    }else if(window.ActiveXObject){
        return new ActiveXObject('Microsoft.XMLHTTP');
    }else{
        throw new Error('No XHR object available.')
    }

}
```

现在大家基本上都用更简化的方式，也可以满足大家的需求。

第二步：open（请求类型，URL，是否异步）

1、get

open(“GET”,”/xx/xx/xx?xx=xx&xx=xx”,true)

2、post

open(“POST”,”/xx/xx”,true)

第三步：send

1、get

send(null)

2、post

sent(data).

下面看完整例子

readystatechange事件是用于异步的。如果同步的话，只要sent以后，执行readystatechange对应的函数中的内容即可。

```
xhr = createXHR;
xhr.onreadystatechange = function(){
    if (xhr.readyState == 4) {
        if (xhr.status >= 200 && xhr.status < 300 || xhr.status == 304) {
            console.log(xhr.responseText);
        }
    }
}
xhr.open("GET",'/test/hello?name=Swing&age=20');
xhr.send(null);
```

GET 请求方式是通过URL参数将数据提交到服务器的，POST则是通过将数据作为 send 的参数提交到服务器；

POST 请求中，在发送数据之前，要设置表单提交的内容类型；

```
xhr = createXHR;
xhr.onreadystatechange = function(){
    if (xhr.readyState == 4) {
        if (xhr.status >= 200 && xhr.status < 300 || xhr.status == 304) {
            console.log(xhr.responseText);
        }
    }
}
xhr.open("POST",'/test/hello',true);
xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xhr.send(data);
```

data：可以是键值对比如：”age=20&name=Swing”；提交表单的时候，需要手动序列化表单，serialize(form)；或者直接使用FormData(form)，可能低版本的浏览器不支持，使用FormData的时候，不需要setRequestHeader。

get和post有字符编码的错误的时候，应该用encodeURIComponent()对key和

value进行编码

```
xhr = createXHR;
xhr.onreadystatechange = function(){
    if (xhr.readyState == 4) {
        if (xhr.status >= 200 && xhr.status < 300 || xhr.status == 304) {
            console.log(xhr.responseText);
        }
    }
}
xhr.open("POST",'/test/hello',true);
xhr.send(new FormData(form));
```

encodeURI()：用于整个 URI 的编码，不会对本身属于 URI 的特殊字符进行编码，如冒号、正斜杠、问号和井号；其对应的解码函数 decodeURI()；

encodeURIComponent()：用于对 URI 中的某一部分进行编码，会对它发现的任何非标准字符进行编码；其对应的解码函数 decodeURIComponent()；

```
url += encodeURLComponent(name) + '=' + encodeURLComponent(value);
```

jquery的ajax：

```
<script type="text/javascript">
$('#list li').click ( function () {
        $.ajax({
            type:'GET',
            data:'',
            url:"getDetails.php?LiName="+this.id,
            success:function(data){
                $('#inf').html(data);
            },
            dataType:'text',
            error:function (){
                alert("失败！");
            }
        })
    });
</script>
```

dataType：

"xml": 返回 XML 文档，可用 jQuery 处理。

"html": 返回纯文本 HTML 信息；包含的 script 标签会在插入 dom 时执行。

"script": 返回纯文本 JavaScript 代码。不会自动缓存结果。除非设置了 "cache" 参数。注意：在远程请求时(不在同一个域下)，所有 POST 请求都将转为 GET

请求。（因为将使用 DOM 的 script标签来加载）

"json": 返回 JSON 数据 。

"jsonp": JSONP 格式。使用 JSONP 形式调用函数时，如 "myurl?callback=?"

jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。

"text": 返回纯文本字符串

简写方式：$.get(url,{data},success function,datatype)

$.post(url,{data},success function,datatype)

```
//$.get("请求url","发送的数据对象","成功回调","返回数据类型");
$.get("test.cgi",{ name: "John", time: "2pm" },
   function(data){
      alert("Data Loaded: " + data);
},'json');
```

```
//完整实例如：（表单html结构不在写）
    $("form").on("submit",function(){
        var url = this.action;   //可以直接取到表单的action
        var formData = $(this).serialize();
        $.post(url,formData,
            function(data){
                //返回成功，可以做一个其他事情
                console.log(data);
            },'json');

        //阻止表单默认提交行为
        return false
    })
```

jquery ajax 经常用到的一个函数：

$('form').serialize()，当需要提交表单的时候，对表单进行序列化。上图

新版可以写成连写方式，链式调用：.done().fail()

```
$.get("ajax.php").done(function() {
    //延迟成功
    alert("ok");
}).fail(function(){
    //延迟失败;
    alert("$.get failed!");
});
```

ajax不能跨域，需要借助其他技术

首先什么是跨域，简单地理解就是因为JavaScript同源策略的限制，a.com 域名下的js无法操作b.com或是c.a.com域名下的对象。

1、CORS协议（由后端服务器实现CROS接口）

2、IFRAME

3、CSST：通过读取 CSS3 content 属性获取传送内容。

4、Comet

5、图像Ping：img.onload=img.onerror=function()

6、WebSocket

7、JSONP：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。

var url = “http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998&callback=flightHandler";

script.setAttribute('src', url);

于是服务器会返回给flightHandler一个json格式的data。

```
></title>
t type="text/javascript">
航班信息查询结果后的回调函数
ightHandler = function(data){
ert('你查询的航班结果是：票价 ' + data.price + ' 元，' + '余票 ' + data.tickets + ' 张。');

jsonp服务的url地址（不管是什么类型的地址，最终生成的返回值都是一段javascript代码）
l = "http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998&callback=flightHandler";
script标签，设置其属性
ript = document.createElement('script');
.setAttribute('src', url);
cript标签加入head，此时调用开始
nt.getElementsByTagName('head')[0].appendChild(script);
pt>
```

jquery的jsonp跨域：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Untitled Page</title>
     <script type="text/javascript" src=jquery.min.js"></script>
     <script type="text/javascript">
    jQuery(document).ready(function(){
       $.ajax({
            type: "get",
            async: false,
            url: "http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998",
            dataType: "jsonp",
            jsonp: "callback",//传递给请求处理程序或页面的，用以获得jsonp回调函数名的参数名(一般默认为:callback)
            jsonpCallback:"flightHandler",//自定义的jsonp回调函数名称，默认为jQuery自动生成的随机函数名，也可以写"?"，jQuery会自动为
你处理数据
            success: function(json){
                alert('您查询到航班信息：票价：  ' + json.price + ' 元，余票：  ' + json.tickets + ' 张。');
            },
            error: function(){
                alert('fail');
            }
        });
    });
    </script>
    </head>
 <body>
 </body>
</html>
```

```
OCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3
nsitional.dtd">
tml xmlns="http://www.w3.org/1999/xhtml" >
ead>
  <title>Untitled Page</title>
   <script type="text/javascript" src=jquery.min.js"></script>
   <script type="text/javascript">
  jQuery(document).ready(function(){
     $.ajax({
          type: "get",
          async: false,
          url: "http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998
          dataType: "jsonp",
          jsonp: "callback",//传递给请求处理程序或页面的，用以获得jsonp回调函数名的
          jsonpCallback:"flightHandler",//自定义的jsonp回调函数名称，默认为jQue
为你处理数据
          success: function(json){
              alert('您查询到航班信息：票价：  ' + json.price + ' 元，余票：  ' +
          },
          error: function(){
              alert('fail');
          }
```

为什么我这次没有写flightHandler这个函数呢？而且竟然也运行成功了！哈哈，这就是jQuery的功劳了，jquery在处理jsonp类型的ajax时（还是忍不住吐槽，虽然jquery也把jsonp归入了ajax，但其实它们真的不是一回事儿），自动帮你生成回调函数并把数据取出来供success属性方法来调用，

setTimeout和setInterval的相互实现：

```
// setTimeout实现setInterval，使用setTimeout的链式操作
i = 0;
setTimeout(function(){
    console.log(i);
    i++;

    if (i < 10) {
        setTimeout(arguments.callee,1000);
    }

},1000)

// setInterval实现setTimeout
var loop = setInterval(function(){
    console.log('hello');
    clearInterval(loop);
},1000)
```

```
// 先实现如何使用setTimeout每隔一秒打印递加的数字
for (var i = 0; i < 10; i++) {
    setTimeout(function(num){
        return function (){
            console.log(num);
        }
    }(i),1000 * i)
}
// 实现如何使用setInterval每隔一秒打印递加的数字
i = 0;
loop = setInterval(function(){
    console.log(i);
    i ++ ;
    if (i == 10) {
        clearInterval(loop);
    }
},1000)
```

bind、apply、call的区别：

apply和call的作用相同，改变某个函数执行的上下文环境，也就是可以改变函数内this的指向值。（可以用于继承）。区别是传入的参数方法不同。

call()方法中后面的所有参数都是用逗号表示，apply()方法中所有参数都是用数组表示。如果参数数量确定，可以使用call方法，如果数量不确定，则使用apply方法。

bind返回新的函数，但是不执行，apply/call会立即执行该函数。 bind()方法的第

一个参数作为 this，传入 bind() 方法的第二个以及以后的参数加上绑定函数运行时本身的参数按照顺序作为原函数的参数来调用原函数。

对数组和字符串的操作：

split：将字符串拆分为数组，stringObject.split(separator,howmany)

join：用于把数组中的所有元素放入一个字符串。arrayObject.join(separator)

slice：可从已有的数组中返回选定的元素。arrayObject.slice(start,end)，不改变原数组

splice：向/从数组中添加/删除项目，然后返回被删除的项目。该方法会改变原始数组。arrayObject.splice(index,howmany,item1,.....,itemX)

函数柯里化：

先介绍一下函数绑定：

```
function bind(fn,context){
    return function(){
        fn.apply(context,arguments);
    }
}
```

```
function curry(fn){
    var args = Array.prototype.slice.call(arguments,1);
    return function(){
        var innerargs = Array.prototype.slice.call(arguments);
        var finalargs = args.concat(innerargs);
        return fn.apply(null,finalargs);
    }
}
```

如同函数绑定，使用闭包返回函数，区别就在于当函数被调用的时候，返回的函数还需要设置一些
传入的参数。

f()()()()，经常会考的题目。也是运用了函数柯里化

```
function add(a){
    function toadd(b){
        a = a + b;
        return toadd;
    }
    toadd.toString = toadd.valueOf = function(){return a;}
    return toadd;
}
```

正则：

\w：任何一个字母数字字符(大小写均可)或下划线字符(等价于[a-zA-Z0-9_]),只能匹配单个字符。

\W: 任何一个非字母数字或非下划线字符。

\s：任何一个空白字符(等价于[\f\n\r\t\v])。

\S：任何一个非空白字符(等价于[^\f\n\r\t\v])。

\d：任何从0到9的数字。

{}大括号，它的最后一种用法是给出一个最小的重复次数(但不必给出一个最大值).比如说,{3,}表示至少重复3次，与之等价的说法就是”必须重复3次或更多次”。

懒惰型匹配法: *? （它是*的懒惰型版本)。+?， {n,}?

如利用cat可以匹配cat,但是也可以匹配category,如果我们只想匹配cat,则只需要加上\bcat\b即可。(b为boundary(边界)的意思),如果不匹配一个单词的边界，可以使用\B.

.*表示匹配的任意字符(.的零次或多次重复出现)

(?m) 单行匹配

^...$ 匹配的开始和结束符号

\s* 匹配零个或多个空白字符

// 匹配注释符号

.* 匹配注释符号后的任意字符

基本的元字符

.	匹配任意单个字符
|	逻辑或操作符
[]	匹配字符集合中的一个字符
[^]	对字符集合求非
-	定义一个区间（例如A-Z）
\	对下一个字符转义

匹配模式

(?m)	分行匹配模式

数量元字符

*	匹配前一个字符(子表达式)的零次或多次重复
*?	*的懒惰型版本
+	匹配前一个字符(子表达式)的一次或多次重复
+?	+的懒惰型版本
?	匹配的前一个字符(子表达式)零次或一次重复
{n}	匹配前一个字符(子表达式)的n次重复
{m,n}	匹配前一个字符(子表达式)至少m次其至多n次重复
{n,}	匹配前一个字符(子表达式)n次或更多次重复
{n,}?	{n,}的懒惰型版本

特殊字符元字符

[\b]	退格字符
\c	匹配一个控制字符
\d	匹配任意数字字符
\D	\d的反义
\f	换页符
\n	换行符
\r	回车符
\s	匹配一个空白字符
\S	\s的反义
\t	制表符(Tab字符)
\v	垂直制表符
\w	匹配任意字母数字字符或下划线字符
\W	\w的反义
\x	匹配一个十六进制数字
\o	匹配一个八进制数字

位置元字符

^	匹配字符串的开头
\A	匹配字符串的开头
$	匹配字符串的结束
\Z	匹配字符串的结束
\<	匹配单词的开头
\>	匹配单词的结束
\b	匹配单词边界（开头和结束）
\B	\b的反义

()	定义一个子表达式
\1	匹配第一个子表达式,\2代表第二个子表达式,以此类推
?=	向前查找
?<=	向后查找
?!	负向前查找
?<!	负向后查找
?()	条件(if then)
?()|	条件(if then else)

\E	结束\L或者\U的转换
\l	把下一个字符转换为小写
\L	把后面的字符转换为小写，直到遇见\E为止
\u	把下一个字符转换为大写
\U	把后面的字符转换为大写，知道遇见\E为止

正则对象的方法：

1、test()

一个字符串是否匹配正则表达式，返回布尔值。

2、exec()

检索字符串与正则表达式匹配的值。返回一个数组，其中存放匹配的结果。如果未找到返回null。

3、compile()

在脚本执行过程中编译正则表达式，也可以改变已有的表达式。

match：match() 方法可在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。

该方法类似 indexOf() 和 lastIndexOf()，但是它返回指定的值，而不是字符串的位置。它返回的结果跟exec方法很像，也是一个数组，其中存放匹配的结果，编号为0的位置存放的是整个正则的匹配，后面存放的是小括号的匹配结果（子式）。

ES6当中y修饰符：

y修饰符，叫做粘连修饰符，也是全局匹配，下次匹配也是从上次匹配成功的下一位开始。不同的是，g只要保证剩余的位置中存在匹配就行，而y必须从剩余的第一位开始匹配。

例子：

```
r1 = /a+/g
r2 = /a+/y
r3 = /^a+/g
r4 = /^a/g;
s1 = 'aaa-aa-a'
```

```
s2 = '-aaa-aa-a'
s3 = 'aaaaa'
    console.log(r1.exec(s1));//'aaa'
    console.log(r1.exec(s1));//'aa'
    console.log(r1.exec(s1));//'a'

    console.log(r2.exec(s1));//'aaa'
    console.log(r2.exec(s1));//null
    console.log(r2.exec(s1));//'aaa'

    console.log(r3.exec(s1));//'aaa'
    console.log(r3.exec(s1));//null
    console.log(r3.exec(s1));//'aaa'  又从头开始

    console.log(r1.exec(s2));//'aaa'
    console.log(r1.exec(s2));//'aa'
    console.log(r1.exec(s2));//'a'

    console.log(r2.exec(s2));//null
    console.log(r2.exec(s2));//null
    console.log(r2.exec(s2));//null

    console.log(r3.exec(s2));//null
    console.log(r3.exec(s2));//null
    console.log(r3.exec(s2));//null

    console.log(r4.exec(s3));//a
    console.log(r4.exec(s3));//null   因为不是从开头开始的，所以返回了null
    console.log(r4.exec(s3));//a
```

^表达式

y修饰符的作用基本跟/^/类似，不同的是^表示匹配输入字符串的开始位置，如果当前位置不是字符串的开始位置就匹配不成功。

在replace当中，如果只用y的话，不会全局匹配，要同时使用yg才会是全局且粘

连匹配。

例如：

```
console.log('aaxa'.replace(/a/gy,'b'));//bbxa
console.log('aaxa'.replace(/a/g,'b'));//bbxb
console.log('aaxa'.replace(/a/y,'b'));//baxa
```

[]和|的区别：

1、[]表示的是匹配里面的字符，|可以不是字符。所以对于不是字符的情况，例如\b（表示单词的分界，字与空格间的位置）就不能放在[]当中

2、[]是一个集合，里面的东西没有顺序，所以对于那些以什么为开头的情况不能放在[]当中。

常见的一些正则：

1、匹配JavaScript的注释行：(?m)^\s*/.*$

2、匹配IP：(((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))\.){3}((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))

3、匹配回溯，哪些字符是重复拼写的：[ ]+(\w+)[ ]+\1。[ ]+匹配一个或多个空格，\w+匹配一个或多个字母数字字符，[ ]+匹配随后的空格。\w+是括在括号里面的，它是一个子表达式。这个子表达式只是把整个模式的一部分单独划分出来以便在后面引用，这个模式的最后一部分是\1,这是一个回溯引用，而它引用的正是前面划分出来的那个子表达式。

4、日期各式：var reg = /^(\d{1,4})(-|\/)(\d{1,2})\2(\d{1,2})$/;

5、密码强度：大小写字母和数字的组合，不能使用特殊字符，长度在8-10之间。

/^(?=.*[0-9]+.*)(?=.*[a-z]+.*)(?=.*[A-Z].*)(?!.*[^a-zA-Z0-9].*).{8,10}$/

?=向前查找，不会被正则表达式引擎返回，只是代表字符串是否满足了这些规则。不像其他正则是从前到后进行匹配。(?:XXX)取消捕获组。

6、字符串仅是中文：

^[\u4e00-\u9fa5]{0,}$

7、数字、字母、下划线组成：

^\w+$

8、email：

^[a-zA-Z0-9]+([._\\-]*[a-zA-Z0-9])*@([a-zA-Z0-9]+.)[a-z0-9]+$

正则的向前查找和向后查找：

向前查找：查找出现在被匹配文本之后的字符，但不消费那个字符，位于匹配文本之后。(?=)

向后查找：查找出现在被匹配文本之前的字符,但不消费他，位于匹配文本之前。(?<=)

例如:(?<=<[tT][iI][lL][eE]>).*(?=</[tT][iI][lL][eE]>) 可以匹配到<title></title>之间的文本.

负向前查找和负向后查找：?!=和?<!=

js不支持正向后查找和负向后查找

9、价格千分位格式化：（并对小数四舍五入）

```
function formatNum(num){
3     return (num.toFixed(2) + '').replace(/\d{1,3}(?=(\d{3})+(\.\d*)?$)/g, '$&,');
4 }
```

$1、$2、...、$99	与 regexp 中的第 1 到第 99 个子表达式相匹配的文本。
$&	与 regexp 相匹配的子串。
$`	位于匹配子串左侧的文本。
$’	位于匹配子串右侧的文本。
$$	直接量符号。

其实，"小括号包含的表达式所匹配到的字符串" 不仅是在匹配结束后才可以使用，在匹配过程中也可以使用。表达式后边的部分，可以引用前面 "括号内的子匹配已经匹配到的字符串"。引用方法是 "/" 加上一个数字。"/1" 引用第1对括号内匹配到的字符串，"/2" 引用第2对括号内匹配到的字符串……以此类推，如果一对括号内包含另一对括号，则外层的括号先排序号。换句话说，哪一对的左括号 "(" 在前，那这一对就先排序号。好像应该是\1.

jquery中高度和宽度：

alert($(window).height()); //浏览器当前窗口可视区域高度

alert($(document).height()); //浏览器当前窗口文档的高度

alert($(document.body).height());//浏览器当前窗口文档body的高度
alert($(document.body).outerHeight(true));//浏览器当前窗口文档body的总高度 包括border padding margin
alert($(window).width());//浏览器当前窗口可视区域宽度
alert($(document).width());//浏览器当前窗口文档对象宽度
alert($(document.body).width());//浏览器当前窗口文档body的高度
alert($(document.body).outerWidth(true));//浏览器当前窗口文档body的总宽度 包括border padding margin
1)、页面内容大于视口（浏览器窗口）时：$(document).height() = $("body").height() > $(window).height();
2)、页面内容小于视口时：$(document).height() = $(window).height() > $("body").height();

字符串转换：
字符串转换成数值：
parseInt()和parseFloat().
parseInt('124bll')//124
parseInt('bluue')//NaN
parseInt()方法有基模式，可以把任何进制的字符串转换成整数
parseInt('AF',16)//175
parseInt('010',10)//10;parseInt('010')//8
parseFloat()必须以十进制形势表示浮点数，没有基模式。
parseFloat('23.45.5')//23.45
强制类型转换：
Number('12.34.5')//NaN
Number('5.5')//5.5
数值转换成字符串：
string类的toString方法：参数表示转换成多少进制的字符串
10.toString()//'10'
10.toString(16)//'a'
强制类型转换
Boolean(10)//true
String(10)//'10'

字符转ascii码

str="A";

code = str.charCodeAt();

ascii码砖字符：

String.fromCharCode(code);

escape方法： 该方法返回对一个字符串编码后的结果字符串。对字符串进行编码，这样就可以在所有的计算机上读取该字符串。该方法不会对 ASCII 字母和数字进行编码，也不会对下面这些 ASCII 标点符号进行编码： * @ - _ + . / 。其他所有的字符都会被转义序列替换。

浏览器中的一些位置：

1、clientX和clientY（客户区坐标）

鼠标事件都是在浏览器视口中的特定位置上发生的。这个位置信息保存在事件对象的clientX和clientY属性中。所有浏览器都支持，表示事件发生时鼠标指针在视口中的水平和垂直坐标。（视口代表不包括滚动部分）

2、pageX和pageY（页面坐标）

表示事件发生的时候，鼠标光标在页面中的位置，坐标是从页面本身而非视口的左边和顶边计算的。（包括滚动部分）

IE8及更早版本不支持。可通过客户区坐标和滚动信息计算，运用

document.body.scrollLeft(Top)(混杂模式)或者

document.documentElement.scrollLeft(top)(标准模式)

if (! event.pageX)

pageX = event.clientX + (document.body.scrollLeft ||

document.documentElement.scrollLeft)

3、screenX和screenY（屏幕坐标）

事件发生的时候，鼠标相对于屏幕的位置。

Expires / Cache-Control / Last-Modified / If-Modified-Since / ETag / If-None-Match 的区别以及使用详解

当服务器发出响应的时候，可以通过两种方式来告诉客户端缓存请求：

第一种是Expires，比如：

Expires: Sun, 16 Oct 2016 05:43:02 GMT
在此日期之前，客户端都会认为缓存是有效的。
不过Expires有缺点，比如说，服务端和客户端的时间设置可能不同，这就会使缓存的失效可能并不能精确的按服务器的预期进行。
第二种是Cache-Control，比如：
Cache-Control: max-age=315360000
这里声明的是一个相对的秒数，表示从现在起，315360000秒内缓存都是有效的，这样就避免了服务端和客户端时间不一致的问题。
但是Cache-Control是HTTP1.1才有的，不适用与HTTP1.0，而Expires既适用于HTTP1.0，也适用于HTTP1.1，所 以说在大多数情况下同时发送这两个头会是一个更好的选择，当客户端两种头都能解析的时候，会优先使用Cache-Control。
ETag是响应头，If-None-Match是请求头。Last-Modified / If-Modified-Since的主要缺点就是它只能精确到秒的级别，一旦在一秒的时间里出现了多次修改，那么Last-Modified / If-Modified-Since是无法体现的。相比较，ETag / If-None-Match没有使用时间作为判断标准，而是使用一个特征串。Etag把Web组件的特征串告诉客户端，客户端在下次请求此Web组件的时 候，会把上次服务端响应的特征串作为If-None-Match的值发送给服务端，服务端可以通过这个值来判断是否需要从重新发送，如果不需要，就简单的 发送一个304状态码，客户端将从缓存里直接读取所需的Web组件。

cookie跨域：
服务器在设置Cookie之前要加一行：header('P3P: CP="CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA div COM NAV OTC NOI DSP COR”');

cookie的操作：
这里document.cookie =“xx=xx”，每次都相当于是在后面追加。当键重复的时候，会替代原有的。
document.cookie=c_name+ "=" +escape(value)+ ((expiredays==null) ? "" : “;expires=“+exdate.toGMTString())，这条语句设置了c_name这个cookie要被存放的时间。

```
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=")
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1
    c_end=document.cookie.indexOf(";",c_start)
    if (c_end==-1) c_end=document.cookie.length
    return unescape(document.cookie.substring(c_start,c_end))
    }
  }
return ""
}

function setCookie(c_name,value,expiredays)
{
var exdate=new Date()
exdate.setDate(exdate.getDate()+expiredays)
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString())
}

function checkCookie()
{
username=getCookie('username')
if (username!=null && username!="")
  {alert('Welcome again '+username+'!')}
else
  {
  username=prompt('Please enter your name:',"")
  if (username!=null && username!="")
    {
    setCookie('username',username,365)
    }
  }
}
```

为了删除一个cookie，可以将其过期时间设定为一个过去的时间。

```
//获取当前时间
var date=new Date();
//将date设置为过去的时间
date.setTime(date.getTime()-10000);
//将userId这个cookie删除
document.cookie="userId=828; expires="+date.toGMTString();
```

默认情况下，如果在某个页面创建了一个cookie，那么该页面所在目录中的其他页面也可以访问该cookie。如果这个目录下还有子目录，则在子目录中也可以访问。

为了控制cookie可以访问的目录，需要使用path参数设置cookie，语法如下：

document.cookie="name=value; path=cookieDir”;

```
document.cookie="userId=320; path=/";
```

只能用这个方法一次设置或更新一个cookie。打印的时候，只会显示cookie的名字。 以下可选的cookie属性值跟在键值对后，定义cookie的设定/更新，跟着一个分号以作分隔：

- ;path=*path* (例如 '/', '/mydir') 如果没有定义，默认为当前文档位置的路径。
- ;domain=*domain* (例如 'example.com'， '.example.com' (包括所有子域名), 'subdomain.example.com') 如果没有定义，默认为当前文档位置的路径的域名部分。
- ;max-age=*max-age-in-seconds* (例如一年为60*60*24*365)
- ;expires=*date-in-GMTString-format* 如果没有定义，cookie会在对话结束时过期
  - 这个值的格式参见Date.toUTCString()
- ;secure (cookie只通过https协议传输)

cookie的值字符串可以用encodeURIComponent()来保证它不包含任何逗号、分号或空格(cookie值中禁止使用这些值).

指定可访问cookie的主机名

和路径类似，主机名是指同一个域下的不同主机，例如：www.google.com和gmail.google.com就是两个不同的主机名。默认情况下，一个主机中创建的cookie在另一个主机下是不能被访问的，但可以通过domain参数来实现对其的控制，其语法格式为：

document.cookie="name=value; domain=cookieDomain";

以google为例，要实现跨主机访问，可以写为：

```
document.cookie="name=value;domain=.google.com";
```

这样，所有google.com下的主机都可以访问该cookie。

cookie的属性：

Expires – 过期时间。指定cookie的生命期。具体是值是过期日期。如果想让cookie的存在期限超过当前浏览器会话时间，就必须使用这个属性。当过了到期日期时，浏览器就可以删除cookie文件，没有任何影响。

Path – 路径。指定与cookie关联的WEB页。值可以是一个目录，或者是一个路径。如果/head/index.html 建立了一个cookie，那么在/head/目录里的所有页面，以及该目录下面任何子目录里的页面都可以访问这个cookie。这就是说，在/head/stories/articles 里的任何页面都可以访问/head/index.html建立的

cookie。但是，如果/zdnn/ 需要访问/head/index.html设置的cookes，该怎么办?这时，我们要把cookies的path属性设置成“/”。在指定路径的时候，凡是来自同一服务器，URL里有相同路径的所有WEB页面都可以共享cookies。现在看另一个例子：如果想让 /head/filters/ 和/head/stories/共享cookies，就要把path设成“/head”。

Domain – 域。指定关联的WEB服务器或域。值是域名，比如goaler.com。这是对path路径属性的一个延伸。如果我们想让dev.mycompany.com 能够访问bbs.mycompany.com设置的cookies，该怎么办? 我们可以把domain属性设置成“mycompany.com”，并把path属性设置成“/”。FYI：不能把cookies域属性设置成与设置它的服务器的所在域不同的值。

Secure – 安全。指定cookie的值通过网络如何在用户和WEB服务器之间传递。这个属性的值或者是“secure”，或者为空。缺省情况下，该属性为空，也就是使用不安全的HTTP连接传递数据。如果一个 cookie 标记为secure，那么，它与WEB服务器之间就通过HTTPS或者其它安全协议传递数据。不过，设置了secure属性不代表其他人不能看到你机器本地保存的cookie。换句话说，把cookie设置为secure，只保证cookie与WEB服务器之间的数据传输过程加密，而保存在本地的cookie文件并不加密。如果想让本地cookie也加密，得自己加密数据。

HttpOnly:告知浏览器不允许通过document.cookie去更改Cookie值

在服务器端用nodeJS来获取cookie的内容：

console.log(request.headers.cookie);//服务器端获取cookie

在服务器端用nodeJS来设置cookie的内容：

response.setHeader('Set-Cookie',['a=000','t=111','w=222']);//服务器端设置cookie，用[]可以同时设置多个，否则一次只能设置一个，设置的cookie在浏览器端可以通过document.cookie获取到

response.setHeader('P3P','CP=CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR')//允许跨域的设置

跨域：通过在www.taobao.com 的server端提供一个获取当前域下所有cookie的php的请求地址，然后该php获取到cookie之后将期并成 js 代码，也就是以上第二个截图所看到的。然后再在 tmall 采用 jsonp 的方式跨域加载该 js 代码，从而实现 cookie 的跨域访问。

sessionStorage和localStorage

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。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。而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。

它们的用法是（key,value）或只有key

CommonJS是一个规范，Prototype也是一个框架。

标签中嵌入JS的方法：

1、iframe：<iframe src=”javascript: alert(1)”></iframe>

页面加载的时候会触发

2、img：

onerror事件，当图片不存在时，将触发

3、expression：IE下<s style=”top:expression(alert(1))”></s>

IE5及其以后版本支持在CSS中使用expression，用来把CSS属性和Javascript表达式关联起来，这里的CSS属性可以是元素固有的属性，也可以是自定义属性。就是说CSS属性后面可以是一段Javascript表达式，CSS属性的值等于Javascript表达式计算的结果。 在表达式中可以直接引用元素自身的属性和方法，也可以使用其他浏览器对象。这个表达式就好像是在这个元素的一个成员函数中一样。

4、onclick等事件：

事件里面可以直接使用javascript的函数：

```
<body>

<!--    <iframe src="javascript:hello('Swing')"></iframe>这样是错误的
 -->
    <iframe src="javascript:alert(2)"></iframe>
    <div onclick="hello('Swing')">hello</div>
</body>
    <script type="text/javascript">
        function hello(name) {
            alert("hello:" + name);
        }
    </script>
</html>
```

IE66和Firefox：

1、火狐浏览器中，非float的div前面有同一父级的float的div，此div若有背景图，要使用clear：both，才能显示背景图，而IE6.0中不用使用clear：both

2、在[text-decoration:underline]的属性下，IE6.0显示的下划线会比FireFox低一点。在FireFox中，部分笔画会在下划线的下面1个象素左右。

3、innerText IE支持，FIREFOX不支持，在Firefox 下使用 textContent。Firefox 不支持 DOM 对象的 outerHTML、innerText、outerText 属性

4、setAttribute(‘class', 'styleClass') FIREFOX支持，IE不支持， E6 IE7 IE8(Q)中无法通过

"Element.setAttribute("class", "AttributeValue")" 设置元素的 class 属性，而需要使用

"Element.setAttribute("className", "AttributeValue")"

IE并不是不支持setAttribute这个函数,而是不支持用setAttribute设置某些属性,例如对象属性、集合属性、事件属性，也就是说用setAttribute设置style和onclick这些属性在IE中是行不通的。为达到兼容各种浏览器的效果，可以用点符号法来设置Element的对象属性、集合属性和事件属性。

5、createElement在IE和Firefox下都可以，①document.createElement ie7以及以前有些问题，但是可以使用。d = document.createElement(“div”),d.id = “div1”

但是ie7以及以前是这么用的：（其它浏览器不支持这种做法）var d =document.createElement（" "）;其中原因有：无法动态设置iframe的name；把button的动态type= rest，表单重设不了。

6、IE6.0的div的内嵌div可以把父级的高度撑大，而FireFox不可以，要自己设置高度。

7、当设置为三列布局时，IE6.0的float宽度不能达到100%，而FireFox可以。当设置为两列布局时，两种浏览器都可以。
8、IE下，可以使用getAttribute获取自定义属性，也可以使用获取常规属性的方式获取自定义属性；firefox下只能使用getAttribute获取自定义属性

==引起的类型转换过程：
一、首先看双等号前后有没有NaN，如果存在NaN，一律返回false。
二、再看双等号前后有没有布尔，有布尔就将布尔转换为数字。（false是0，true是1）
三、接着看双等号前后有没有字符串，有三种情况：
1、对方是对象，对象使用toString()或者valueOf()进行转换；
2、对方是数字，字符串转数字；（前面已经举例）
3、对方是字符串，直接比较；
4、其他返回false
四、如果是数字，对方是对象，对象取valueOf()或者toString()进行比较，其他一律返回false
1、[].toString() = [].valueOf() = false
2、{}.toString() = [object Object]，{}.valueOf() = Object {}
五、null, undefined不会进行类型转换，但它们俩相等
[1,2] == true //false
2 == true //false
console.log([2,3].toString()) //“2,3”
除了下面这些，其余的转换为布尔值都为true
Boolean(undefined) // false
Boolean(null) // false
Boolean(0) // false
Boolean(NaN) // false
Boolean('') // false
Number(undefined) //NaN
Number(null) //0
0 == undefined //false
0 == null //false

undefined == null　//true　undefined值派生自null，所以返回true

false == null //false

false == undefined //false

false == 0 //true

false == ‘’ //true

==类型转换跟Number()、Boolean()等类型转换不一样，if是Boolean()类型转换

- **ajax请求和ajax事件：**

  ajax请求：

  $.post(url)是ajax请求；

  ajax事件：

  ajaxComplete(callback)

  ajaxError(callback)

  ajaxSend(callback)

  ajaxStart(callback)

  ajaxStop(callback)

  ajaxSuccess(callback)

- **jquery选择器**

  :contains选择器，选取包含指定字符串的元素，字符串也可以是文本

  $("p:contains(is)")

:input选择器，选取表单元素　　$(":input")

select得到选择的文本和值

```
<form name="a">
<select name="a" size="1" id="obj">
<option value="a">1</option>
<option value="b">2</option>
<option value="c">3</option>
</select>
</form>

console.log(obj.options[obj.selectedIndex].text); //1
console.log(obj.options[obj.selectedIndex].value); //a
console.log(obj.value); //a 直接可以得到被选中的option的值，传给后端
```

禁用表单元素：

Readonly只针对input(text/password)和textarea有效，而disabled对于所有的表单元素有效，包括select,radio,checkbox,button等。

javascript内部对象：

History 对象包含用户（在浏览器窗口中）访问过的 URL
Location 对象包含有关当前 URL 的信息
Window 对象表示浏览器中打开的窗口
Navigator 对象包含有关浏览器的信息

闭包的概念：

闭包是指有权访问另一个函数作用域中的变量的函数

闭包主要涉及到js的几个其他的特性:作用域链,垃圾(内存)回收机制,函数嵌套,等等.

自己函数的变量，父级函数变量，全局变量
闭包：突破作用域链

在程序语言中，闭包以很自然的形式，把我们的目的和我们的目的所涉及的资源全给自动打包在一起让人来使用。

一般来说，js的垃圾回收机制会在函数执行完毕的时候，把函数体内定义的变量都销毁回收。在js函数体内定义的变量在函数体外是无法访问的。闭包最大的用处就是，一个是可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。闭包就是能够读取其他函数内部变量的函数。就是将函数内部和函数外部连接起来的一座桥梁。

js解释器在遇到函数定义的时候,会自动把函数和他可能使用的变量(包括本地变量和父级和祖先级函数的变量(自由变量))一起保存起来。也就是构建一个闭包,这些变量将不会被内存回收器所回收,只有当内部的函数不可能被调用以后(例如被删除了，或者没有了指针),才会销毁这个闭包,而没有任何一个闭包引用的变量才会被下一次内存回收启动时所回收.

但外部函数返回后，其执行环境的作用域链会被销毁，但它的活动对象仍会留在内存中。内部函数可以继续访问外部函数中的变量。

闭包只能取得包含函数中任何变量的最后一个值。这是因为闭包所保存的是整个变量对象，而不是某个特殊的变量。

闭包容易引起内存泄漏。例如，如果闭包的作用域链中保存着一个HTML元素，就意味着该元素无法被销毁。P184.应该将所要的值保存在一个变量中，并将HTML对象设置为null，这样就能解除对DOM对象的引用。

http协议：

URL基本格式：

schema://host[:port#]/path/.../[;url-params][?query-string][#anchor]

scheme 指定低层使用的协议(例如：http, https, ftp)

host HTTP服务器的IP地址或者域名

port# HTTP服务器的默认端口是80，这种情况下端口号可以省略。如果使用了别的端口，必须指明，例如 http://www.cnblogs.com:8080/

path 访问资源的路径

url-params

query-string 发送给http服务器的数据

anchor- 锚

URL例子：

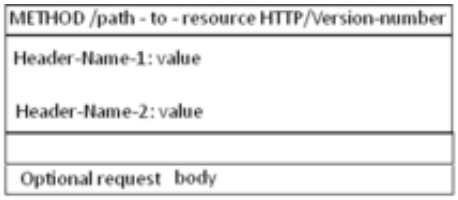
http://www.mywebsite.com/sj/test;id=8079?name=sviergn&x=true#stuff

http协议是无状态的：

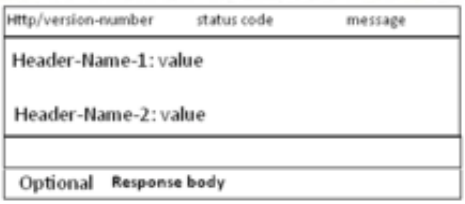
http协议是无状态的，同一个客户端的这次请求和上次请求是没有对应关系，对http服务器来说，它并不知道这两个请求来自同一个客户端。 为了解决这个问题， Web程序引入了Cookie机制来维护状态。

http消息结构：

Request 消息分为3部分，第一部分叫请求行， 第二部分叫http header, 第三部分是body. header和body之间有个空行， 结构如下图：



- 第一行中的Method表示请求方法，比如"POST"，"GET"， Path-to-resoure表示请求的资源， Http/version-number 表示HTTP协议的版本号
当使用的是"GET" 方法的时候， body是为空的。
和Request消息的结构基本一样。 同样也分为三部分，第一部分叫request line, 第二部分叫request header，第三部分是body. header和body之间也有个空行，结构如下图：



http与服务器交互的方法：

最基本的有4种：分别是GET,POST,PUT,DELETE。 一个URL地址用于描述一个网络上的资源，而HTTP中的GET, POST, PUT, DELETE就对应着对这个资源的查，改，增，删4个操作。

1、OPTIONS：

这个方法可使服务器传回该资源所支持的所有HTTP请求方法。用'*'来代替资源名称，向Web服务器发送OPTIONS请求，可以测试服务器功能是否正常运作。

2、HEAD：

与GET方法一样，都是向服务器发出指定资源的请求。只不过服务器将不传回资源的本文部分。它的好处在于，使用这个方法可以在不必传输全部内容的情况下，就可以获取其中“关于该资源的信息”（元信息或称元数据）。

3、GET：

向指定的资源发出“显示”请求。使用GET方法应该只用在读取数据，而不应当被用于产生“副作用”的操作中，例如在Web Application中。其中一个原因是GET可能会被网络蜘蛛等随意访问。参见安全方法

4、POST：

向指定资源提交数据，请求服务器进行处理（例如提交表单或者上传文件）。数据被包含在请求本文中。这个请求可能会创建新的资源或修改现有资源，或二者皆有。

5、PUT：

向指定资源位置上传其最新内容。

6、DELETE：

请求服务器删除Request-URI所标识的资源。

7、TRACE：

回显服务器收到的请求，主要用于测试或诊断。

8、CONNECT：

HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。通常用于SSL加密服务器的链接（经由非加密的HTTP代理服务器）。

我们看看GET和POST的区别

1. GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的Body中.

2. GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制.

3. GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。

4. GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

状态码：（ajax status）

1XX 提示信息 - 表示请求已被成功接收，继续处理

2XX 成功 - 表示请求已被成功接收，理解，接受

3XX 重定向 - 要完成请求必须进行更进一步的处理

4XX 客户端错误 - 请求有语法错误或请求无法实现

5XX 服务器端错误 - 服务器未能实现合法的请求

常见的状态码：

100——客户必须继续发出请求

101——客户要求服务器根据请求转换HTTP协议版本

200——成功

201——提示知道新文件的URL

204——请求成功，但响应的实体中不含实体的主体部分

206——范围请求，返回资源的一部分；如果服务器无法满足范围请求，则返回200

300——请求的资源可在多处得到

301——（删除请求数据？）永久重定向，所请求的文档在别的地方；文档新的URL会在定位响应头信息中给出。浏览器会自动连接到新的URL。

302—— 临时重定向，定位头信息中所给的URL应被理解为临时交换地址而不是永久的

303——与302差不多，但应该用GET方法

304—— 自从上次请求后，网页未改变，服务器不会返回网页内容，可以使用浏览器缓存

400—— Bad Request 客户端请求与语法错误，不能被服务器所理解

401—— unauthorized 表示客户端在授权头信息中没有有效的身份信息时访问受到密码保护的页面。

403—— Forbidden 服务器收到请求，但是拒绝提供服务。这个状态经常会由于服务器上的损坏文件或目录许可而引起。

404——没有发现文件、查询或URl

500——服务器产生内部错误

503—— Server Unavailable 服务器当前不能处理客户端的请求，一段时间后可能恢复正常，处于超负荷或者停机维护

request和response的头域：

request：

Cache 头域

If-Modified-Since

作用： 把浏览器端缓存页面的最后修改时间发送到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行对比。如果时间一致，那么返回304，客户端就直接使用本地缓存文件。如果时间不一致，就会返回200和新的文件内容。客户端接到之后，会丢弃旧文件，把新文件缓存起来，并显示在浏览器中。

If-None-Match

作用: If-None-Match和ETag一起工作，工作原理是在HTTP Response中添加ETag信息。 当用户再次请求该资源时，将在HTTP Request 中加入If-None-Match信息(ETag的值)。如果服务器验证资源的ETag没有改变（该资源没有更新），将返回一个304状态告诉客户端使用本地缓存文件。否则将返回200状态和新的资源和Etag. 使用这样的机制将提高网站的性能

Pragma

作用： 防止页面被缓存， 在HTTP/1.1版本中，它和Cache-Control:no-cache作用一模一样

Cache-Control

作用: 这个是非常重要的规则。 这个用来指定Response-Request遵循的缓存机制。各个指令含义如下

Cache-Control:Public 可以被任何缓存所缓存（）

Cache-Control:Private 内容只缓存到私有缓存中

Cache-Control:no-cache 所有内容都不会被缓存

Client 头域

Accept

作用： 浏览器端可以接受的媒体类型，

例如： Accept: text/html 代表浏览器可以接受服务器回发的类型为 text/html 也就是我们常说的html文档，

如果服务器无法返回text/html类型的数据，服务器应该返回一个406错误(non acceptable)

通配符 * 代表任意类型

例如 Accept: */* 代表浏览器可以处理所有类型，(一般浏览器发给服务器都是发这个)

Accept-Encoding：

作用： 浏览器申明自己接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate），（注意：这不是只字符编码）；

Accept-Language

作用： 浏览器申明自己接收的语言。

语言跟字符集的区别：中文是语言，中文有多种字符集，比如big5，gb2312，gbk等等；

User-Agent

作用：告诉HTTP服务器， 客户端使用的操作系统和浏览器的名称和版本.

Accept-Charset

作用：浏览器申明自己接收的字符集，这就是本文前面介绍的各种字符集和字符编码，如gb2312，utf-8（通常我们说Charset包括了相应的字符编码方案）

Cookie/Login 头域

Cookie:

作用： 最重要的header, 将cookie的值发送给HTTP 服务器

Entity头域

Content-Length

作用：发送给HTTP服务器数据的长度。

Content-Type

作用：

例如：Content-Type: application/x-www-form-urlencoded

Miscellaneous 头域

Referer:

作用： 提供了Request的上下文信息的服务器，告诉服务器我是从哪个链接

过来的，比如从我主页上链接到一个朋友那里，他的服务器就能够从HTTP Referer中统计出每天有多少用户点击我主页上的链接访问他的网站。

Transport 头域

Connection

例如： Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接

例如： Connection: close 代表一个Request完成后，客户端和服务器之间用于传输HTTP数据的TCP连接会关闭， 当客户端再次发送Request，需要重新建立TCP连接。

Host（发送请求时，该报头域是必需的）

作用: 请求报头域主要用于指定被请求资源的Internet主机和端口号，它通常从HTTP URL中提取出来的

HTTP Response header

Cache头域

Date

作用: 生成消息的具体时间和日期

例如： Date: Sat, 11 Feb 2012 11:35:14 GMT

Expires

作用: 浏览器会在指定过期时间内使用本地缓存

Vary

作用:

例如: Vary: Accept-Encoding

Cookie/Login 头域

P3P

作用: 用于跨域设置Cookie, 这样可以解决iframe跨域访问cookie的问题

例如: P3P: CP=CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR

Set-Cookie

作用： 非常重要的header, 用于把cookie 发送到客户端浏览器， 每一个写入cookie都会生成一个Set-Cookie.

Entity头域

ETag

作用：和If-None-Match 配合使用。

Last-Modified:

作用： 用于指示资源的最后修改日期和时间。（实例请看上节的If-Modified-Since的实例）

Content-Type

作用：WEB服务器告诉浏览器自己响应的对象的类型和字符集，

Miscellaneous 头域

Server:

作用：指明HTTP服务器的软件信息

X-AspNet-Version:

作用：如果网站是用ASP.NET开发的，这个header用来表示ASP.NET的版本

X-Powered-By:

作用：表示网站是用什么技术开发的

例如： X-Powered-By: ASP.NET

Transport头域

Connection

例如： Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接

例如： Connection: close 代表一个Request完成后，客户端和服务器之间用于传输HTTP数据的TCP连接会关闭， 当客户端再次发送Request，需要重新建立TCP连接。

Location头域

Location

作用： 用于重定向一个新的位置， 包含新的URL地址

HTTP协议是无状态的和Connection: keep-alive的区别

无状态是指协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。从另一方面讲，打开一个服务器上的网页和你之前打开这个服务器上的网页之间没有任何联系。

HTTP是一个无状态的面向连接的协议，无状态不代表HTTP不能保持TCP连接，更不能代表HTTP使用的是UDP协议（无连接）。

从HTTP/1.1起，默认都开启了Keep-Alive，保持连接特性，简单地说，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连

接。

Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。

Expires / Cache-Control / Last-Modified / If-Modified-Since / ETag / If-None-Match 的区别以及使用详解

当服务器发出响应的时候，可以通过两种方式来告诉客户端缓存请求：

第一种是Expires，比如：

Expires: Sun, 16 Oct 2016 05:43:02 GMT

在此日期之前，客户端都会认为缓存是有效的。

不过Expires有缺点，比如说，服务端和客户端的时间设置可能不同，这就会使缓存的失效可能并不能精确的按服务器的预期进行。

第二种是Cache-Control，比如：

Cache-Control: max-age=315360000

这里声明的是一个相对的秒数，表示从现在起，315360000秒内缓存都是有效的，这样就避免了服务端和客户端时间不一致的问题。

但是Cache-Control是HTTP1.1才有的，不适用与HTTP1.0，而Expires既适用于HTTP1.0，也适用于HTTP1.1，所 以说在大多数情况下同时发送这两个头会是一个更好的选择，当客户端两种头都能解析的时候，会优先使用Cache-Control。

ETag是响应头，If-None-Match是请求头。Last-Modified / If-Modified-Since的主要缺点就是它只能精确到秒的级别，一旦在一秒的时间里出现了多次修改，那么Last-Modified / If-Modified-Since是无法体现的。相比较，ETag / If-None-Match没有使用时间作为判断标准，而是使用一个特征串。Etag把Web组件的特征串告诉客户端，客户端在下次请求此Web组件的时 候，会把上次服务端响应的特征串作为If-None-Match的值发送给服务端，服务端可以通过这个值来判断是否需要从重新发送，如果不需要，就简单的 发送一个304状态码，客户端将从缓存里直接读取所需的Web组件。

http缓存机制

Expires：响应　一个绝对时间，在这个时间之前都有效，

Cache-Control：　请求/响应　相对值

可以取：Public：可以被任何缓存所缓存

Private：内容只缓存到私有缓存中

no-cache：不建议使用本地缓存，其仍然会缓存数据到本地

max-age ： 相对时间，在这个相对时间内都有效，此时相当于Expires = 当前时间+max-age

no-store：不会在客户端缓存任何响应数据

no-cache相当于max-age = 0，立即向浏览器请求

注意：1、Cache-Control 中指定的缓存过期策略优先级高于Expires，当它们同时存在的时候，后者会被覆盖掉。

2、缓存数据标记为已过期只是告诉客户端不能再直接从本地读取缓存了，需要再发一次请求到服务器去确认，并不等同于本地缓存数据从此就没用了，有些情况下即使过期了还是会被再次用到，具体下面会讲到。

Last-Modified：响应　资源的最后修改日期和时间

If-Modified-Since 请求　浏览器缓存页面的最后修改时间

这种组合只能精确到秒

Tag　响应

If-None-Match 请求　etag的值

在没有提供任何浏览器缓存过期策略的情况下，浏览器遵循一个启发式缓存过期策略：

根据响应头中2个时间字段 Date 和 Last-Modified 之间的时间差值，取其值的10%作为缓存时间周期，即作为max-age的值

当时间超过了expires或者本地时间+max-age的时间，浏览器会再次请求服务端，并携带上 Last-Modified 指定的时间或者If-None-Match去服务器对比：

- a）**对比失败**：服务器返回200并重发数据，客户端接收到数据后展示，并刷新本地缓存。
- b）**对比成功**：服务器返回304且不重发数据，客户端收到304状态码后从本地读取缓存数据。以下为模拟此种情况下请求后的抓包情况：

数据类型识别

typeof：可以有括号也可以没有。可以识别标准类型（Null除外,object，一般是认为逻辑上认为null值表示一个空对象指针，所以用typeof的时候，会返

回"object"。所以一般都会把null当做空对象的占位符。设计时留下的问题），不能识别具体对象类型（Function除外，Function可以识别）。返回的是小写的，比如typeof ‘ss’＝’string’。其他内置对象类型自定义等都会被识别为object。

Object.prototype.toString.call()：可以识别标准类型以及内置对象类型（如date），不能识别自定义类型,只能识别出是object。小写。

Object.prototype.toString.call(arr) === '[object Array]'

constructor：可以识别标准类型（Undefined和Null除外，出错），可以识别内置对象类型，可以识别自定义类型。大写。例如：[].constructor;

instanceof：可以识别内置对象类型，不能识别原始类型（如Number，String），可以识别自定义类型和父类型。大写。[1,2] instanceof Object // true.

在跨 frame 对象构建的场景下会失效

对于String、Number、Boolean，如果用new的方式来创建，例如a = new Boolean(false)，这时候a就不能用typeof来判断，用typeof会判断成object，应该用 a instanceof Boolean。a = Boolean(1)，这样的还是不是对象，可以用typeof。

Object的 方法：

Object.keys(obj)：

参数为包含属性和方法的对象

返回对象的可枚举属性和方法的名称

打开新窗口的方法；
open() 方法可以查找一个已经存在或者新建的浏览器窗口。
语法：
window.open([URL], [窗口名称], [参数字符串])
参数说明:
URL：可选参数，在窗口中要显示网页的网址或路径。如果省略这个参数，或者它的值是空字符串，那么窗口就不显示任何文档。
窗口名称：可选参数，被打开窗口的名称。
1.该名称由字母、数字和下划线字符组成。
2."_top"、"_blank"、"_selft"具有特殊意义的名称。
_blank：在新窗口显示目标网页
_self：在当前窗口显示目标网页
_top：框架网页中在上部窗口中显示目标网页
3.相同 name 的窗口只能创建一个，要想创建多个窗口则 name 不能相同。
4.name 不能包含有空格。
参数字符串：可选参数，设置窗口参数，各参数用逗号隔开。

javascript中字符串连接时用Array.join()替换 string += “xx"，换来几十倍的速度提升。
eg： [“abc","def"].join("")

网页链接应该有http开头，a标签中应该是http开头的连接。

var obj=Object.create(null)　第一个参数就是表示新创建的对象的原型。
obj他的原型是null，所以obj是没有prototype属性的
jquery选择器：
只有两个标签选择器之间需要有空格

*	$("*")	所有元素
#id	$("#lastname")	id="lastname" 的元素
.class	$(".intro")	所有 class="intro" 的元素
element	$("p")	所有 <p> 元素
.class.class	$(".intro.demo")	所有 class="intro" 且 class="demo" 的元素

:first	$("p:first")	第一个 <p> 元素
:last	$("p:last")	最后一个 <p> 元素
:even	$("tr:even")	所有偶数 <tr> 元素
:odd	$("tr:odd")	所有奇数 <tr> 元素

:eq(index)	$("ul li:eq(3)")	列表中的第四个元素（index 从 0 开始）
:gt(no)	$("ul li:gt(3)")	列出 index 大于 3 的元素
:lt(no)	$("ul li:lt(3)")	列出 index 小于 3 的元素
:not(selector)	$("input:not(:empty)")	所有不为空的 input 元素

:header	$(":header")	所有标题元素 <h1> - <h6>
:animated		所有动画元素

:contains(text)	$(":contains('W3School')")	包含指定字符串的所有元素
:empty	$(":empty")	无子（元素）节点的所有元素
:hidden	$("p:hidden")	所有隐藏的 <p> 元素
:visible	$("table:visible")	所有可见的表格

s1,s2,s3	$("th,td,.intro")	所有带有匹配选择的元素

[attribute]	$("[href]")	所有带有 href 属性的元素
[attribute=value]	$("[href='#']")	所有 href 属性的值等于 "#" 的元素
[attribute!=value]	$("[href!='#']")	所有 href 属性的值不等于 "#" 的元素
[attribute$=value]	$("[href$='.jpg']")	所有 href 属性的值包含以 ".jpg" 结尾的元素

:enabled	$(":enabled")	所有激活的 input 元素
:disabled	$(":disabled")	所有禁用的 input 元素
:selected	$(":selected")	所有被选取的 input 元素
:checked	$(":checked")	所有被选中的 input 元素

:input	$(":input")	所有 <input> 元素
:text	$(":text")	所有 type="text" 的 <input> 元素
:password	$(":password")	所有 type="password" 的 <input> 元素
:radio	$(":radio")	所有 type="radio" 的 <input> 元素
:checkbox	$(":checkbox")	所有 type="checkbox" 的 <input> 元素
:submit	$(":submit")	所有 type="submit" 的 <input> 元素
:reset	$(":reset")	所有 type="reset" 的 <input> 元素
:button	$(":button")	所有 type="button" 的 <input> 元素
:image	$(":image")	所有 type="image" 的 <input> 元素
:file	$(":file")	所有 type="file" 的 <input> 元素

```
var foo = {n:1};
(function(foo){                 //形参foo同实参foo一样指向同一片内存空间，这个空间
    var foo;                    //优先级低于形参，无效。
    console.log(foo.n);         //输出1
    foo.n = 3;                  //形参与实参foo指向的内存空间里的n的值被改为3
    foo = {n:2};                //形参foo指向了新的内存空间，里面n的值为2.
    console.log(foo.n);         //输出新的内存空间的n的值
})(foo);
console.log(foo.n);             //实参foo的指向还是原来的内存空间，里面的n的值为3.
```

变量提升和形参

函数体内定义的变量优先级低于形参，所以第一次输出1。foo.n=3，改变了形参，而形参又是引用变量，会改变全局中的foo。foo={n:2}，形参foo指向了另一个变量空间，里面的值是2，第二次输出2。第三次的全局在前面已经被改成了3，所以第三次输出3。

从+new Array(15)得出的一些类型转换：

+new Array(1) //0

+new Array(0) //0

+new Array(2) //NaN

+[] //0

+[1] //1

+[1,2] //NaN

+{} //NaN

+(XX) ——> Number(XX)，所以+new Array(15) ——> Number(new Array(15)),所以先执行valueOf，发现valueOf返回的是自己，所以继续执行toString——>Number(new Array(15).toString()) ——>Number(“,,,,,,,,,,,,,,”) ——>NaN.

+[1,2]——>Number([1,2])——>valueOf()返回自身[1,2]——>Number([1,2].toString())——>Number(“1,2”)——>NaN

+new Array(1)——>Number(new Array(1))——>valueOf()返回自身[]—>Number(new Array().toString())——>Number(“ ”)——>0

+[1]——>Number([1])——>valueOf()返回自身[1]——>Number([1].toString())——>Number(“1”)——>1

{}.toString() = [object Object]

{}.valueOf() = Object {}

所以Number({}) = NaN

false == null //false

false == undefined //false

转成Boolean类型和==不一样，有些可以转成Boolean类型，但是不能用

false== 来判断！！！

void

使用方法：void(表达式) = void 表达式

void是一元运算符，它出现在操作数之前，操作数可以是任意类型，操作数会照常计算，但忽略计算结果并返回undefined。由于void会忽略操作数的值，因此在操作数具有副作用的时候使用void来让程序更具语义

无论void后的表达式是什么，void操作符都会返回undefined。但void后面一定有表达式，否则会报错。

void的使用场景：

1.替代undefined：

由于undefined在js中并不是表达式，undefined可能会作为变量被赋值，所以可以用void 0 来替换undefined

2、让a标签不会跳转。页面上有些a标签并不希望它跳转，而是触发一些交互操作。这时候如果href里面不写URL，就会刷新当前页面。在href里面写上href=“javascript:void(0)，可以避免。（在a标签的href里面写#页面不会跳转，也不会刷新，而是到页面 中的某个锚点。）

3、如果我们要生成一个空的src的image，最好的方式似乎也是src='javascript:void(0)'

4、阻止默认事件：跟2一样的原理

//一般写法

```
<a href="http://example.com"
onclick="f();return false;">文字</a>
```

使用void运算符可以取代上面写法

```
<a href="javascript:void(f())">文字</a>
```

5、在URL中可以写带有副作用的表达式，而void则让浏览器不必显示这个表达式的计算结果。例如，经常在HTML代码中的<a>标签里使用void运算符：

```
<a href="javascript:void window.open();">打开一个新窗口</a>
```

href、URL、src：

href和src是有区别的，而且是不能相互替换的。我们在可替换的元素上使用src，然而把href用于在涉及的文档和外部资源之间建立一个关系。

href:

href是Hypertext Reference的缩写，表示超文本引用。用来建立当前元素和文档之间的链接。

href (Hypertext Reference)指定网络资源的位置，从而在当前元素或者当前文档和由当前属性定义的需要的锚点或资源之间定义一个链接或者关系。当我们写下：

<link href="style.css" rel="stylesheet" />

浏览器明白当前资源是一个样式表，页面解析不会暂停（由于浏览器需要样式规则去画或者渲染页面，渲染过程可能会被被暂停）。

src:

src是source的缩写，src的内容是页面必不可少的一部分，是引入。src指向的内容会嵌入到文档中当前标签所在的位置。

src会暂停浏览器加载，直到src对应的内容执行完毕。

src (Source)属性仅仅 嵌入当前资源到当前文档元素定义的位置。当浏览器找到<script src="script.js"></script>

在浏览器下载，编译，执行这个文件之前页面的加载和处理会被暂停。这个过程与把js文件放到<script>标签里类似。这也是建议把JS文件放到底部加载的原因。当然，img标签页与此类似。浏览器暂停加载直到提取和加载图像。

href：

<link>

<a>

src：

<img>　不会阻塞页面的渲染?

<script> 会阻塞页面的渲染

<iframe>

url:

background:url(“”)

JS处理小数（四舍五入、取整）

1、parseInt() 丢弃小数部分，保留整数部分

2、Math.ceil() 向上取整

3、Math.round() 四舍五入

4、Math.floor() 向下取整

5、().toFixed() 保留几位小数四舍五入

例如：var a = 2.985;a.toFixed(2) //1.99

跨域补充：

只要协议 、域名 、端口有任何一个不同, 都被当作是不同的域。

1、CORS

CORS（Corss-Origin Resource Sharing,跨资源共享），基本思想是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应的成功或失败。

即给请求附加一个额外的Origin头部，其中包含请求页面的源信息（协议、域名和端口），以便服务器根据这个头部决定是否给予响应。

2、document.domain

将页面的document.domain设置为相同的值，页面间可以互相访问对方的JavaScript对象。

注意：

不能将值设置为URL中不包含的域；
松散的域名不能再设置为紧绷的域名。
3、图像Ping
var img=new Image();
img.onload=img.onerror=function(){
... ...
}
img.src="url?name=value";
请求数据通过查询字符串的形式发送，响应可以是任意内容，通常是像素图或204响应。
图像Ping最常用于跟踪用户点击页面或动态广告曝光次数。
缺点：
只能发送GET请求；
无法访问服务器的响应文本，只能用于浏览器与服务器间的单向通信。
4、Jsonp
var script=document.createElement("script");
script.src="url?callback=handleResponse";
document.body.insertBefore(script,document.body.firstChild);
JSONP由两部分组成：回调函数和数据
回调函数是接收到响应时应该在页面中调用的函数，其名字一般在请求中指定。
数据是传入回调函数中的JSON数据。
优点：
能够直接访问响应文本，可用于浏览器与服务器间的双向通信。
缺点：
JSONP从其他域中加载代码执行，其他域可能不安全；
难以确定JSONP请求是否失败。
5、Comet
Comet可实现服务器向浏览器推送数据。
Comet是实现方式：长轮询和流
短轮询即浏览器定时向服务器发送请求，看有没有数据更新。
长轮询即浏览器向服务器发送一个请求，然后服务器一直保持连接打开，直到有数据可发送。发送完数据后，浏览器关闭连接，随即又向服务器发起一个新请求。其优点是所有浏览器都支持，使用XHR对象和setTimeout()即可实现。
流即浏览器向服务器发送一个请求，而服务器保持连接打开，然后周期性地向浏

览器发送数据，页面的整个生命周期内只使用一个HTTP连接。

6、WebSocket

WebSocket可在一个单独的持久连接上提供全双工、双向通信。

WebSocket使用自定义协议，未加密的连接时ws://；加密的链接是wss://。

```
var webSocket=new WebSocket("ws://");
webSocket.send(message);
webSocket.onmessage=function(event){
var data=event.data;
... ....
}
```

注意：

必须给WebSocket构造函数传入绝对URL；

WebSocket可以打开任何站点的连接，是否会与某个域中的页面通信，完全取决于服务器；

WebSocket只能发送纯文本数据，对于复杂的数据结构，在发送之前必须进行序列化JSON.stringify(message))。

优点：

在客户端和服务器之间发送非常少的数据，减少字节开销。

7、window.name

在应用页面（a.com/app.html）中创建一个iframe，把其src指向数据页面（b.com/data.html）。

数据页面会把数据附加到这个iframe的window.name上

在应用页面（a.com/app.html）中监听iframe的onload事件，在此事件中设置这个iframe的src指向本地域的代理文件（代理文件和应用页面在同一域下，所以可以相互通信）

总结起来即：iframe的src属性由外域转向本地域，跨域数据即由iframe的window.name从外域传递到本地域。这个就巧妙地绕过了浏览器的跨域访问限制，但同时它又是安全操作。

跨域和cookie跨域：

跨域的时候，HTTP请求和响应中，不会带cookie。要进行cookie的跨域，可以用jsonp。首先，B域中有获取本地cookie的页面，然后A域通过jsonp请求B域的页面，得到返回的cookie

关于Javascript中数字的部分知识总结：

1.Javascript中，由于其变量内容不同，变量被分为基本数据类型变量和引用数据类型变量。基本类型变量用八字节内存，存储基本数据类型(数值、布尔值、null和未定义)的值，引用类型变量则只保存对对象、数组和函数等引用类型的值的引用(即内存地址)。

2. JS中的数字是不分类型的，也就是没有byte/int/float/double等的差异。

ajax的事件是：

ajaxComplete(callback)

ajaxError(callback)

ajaxSend(callback)

ajaxStart(callback)

ajaxStop(callback)

ajaxSuccess(callback)

session和cookie：

cookie机制采用的是在客户端保持状态的方案，而Session机制采用的是在服务器端保持状态的方案。

Cookie机制：

就是当服务器对访问它的用户生成了一个Session的同时，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie，保存在客户端，里面记录着用户当前的信息。当用户再次访问服务器时，浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所

在的位置也就是对应的Cookie文件。 若存在，则把该cookie附在请求资源的HTTP请求头上发送给服务器。

如果不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，只要关闭浏览器窗口，cookie就消失了。这种生命期为浏览器会话期的 cookie被称为会话cookie。会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。如果设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。

Session机制：

当用户访问到一个服务器，服务器就要为该用户创建一个SESSION，在创建这个SESSION的时候，服务器首先检查这个用户发来的请求里是否包含了一个SESSIONID，如果包含了一个SESSIONID则说明之前该用户已经登陆过并为此用户创建过SESSION，那服务器就按照这个SESSIONID把这个SESSION在服务器的内存中查找出来（如果查找不到，就有可能为他新创建一个），如果客户端请求里不包含有SESSIONID，则为该客户端创建一个SESSION并生成一个与此SESSION相关的SESSIONID。这个SESSIONID是唯一的、不重复的、不容易找到规律的字符串，这个SESSIONID将被在本次响应中返回到客户端保存，而保存这个SESSIONID的正是COOKIE，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。

可以有其他机制在COOKIE被禁止时仍然能够把Session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把Session id直接附加在URL路径的后面一种是作为URL路径的附加信息,表现形式为：

http://
..../xxx;jSession=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764；

另一种是作为查询字符串附加在URL后面，表现形式为：

http://...../xxx?
jSession=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764

还有一种就是表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把Session id传递回服务器。

事实上，除非程序通知服务器删除Session，否则Session会被服务器一直保留，直到Session的失效时间到了自动删除。程序一般都是在用户做注销时删除Session。一般Session机制都使用cookie来保存Session id，而一旦关闭IE浏览器，Session id就不存在了，再连接服务器时找不到原来的Session了.如果服务

器设置的cookie被保存到硬盘上，或者使用某种手段改写浏览器发出的 HTTP请求头，把原来的Session id发送给服务器，则再次打开浏览器仍然能够找到原来的Session。恰恰是由于关闭浏览器不会导致Session被删除，迫使服务器为seesion设置了一个失效时间，当距离客户端上一次使用Session的时间超过这个失效时间时，服务器就可以认为客户端已经停止了活动，才会把Session删除以节省存储空间。
一般情况下，Session都是存储在内存里，当服务器进程被停止或者重启的时候，内存里的Session也会被清空，如果设置了Session的持久化特性，服务器就会把Session保存到硬盘上，当服务器进程重新启动或这些信息将能够被再次使用。

具体来说cookie机制采用的是在客户端保持状态的方案。它是在用户端的会话状态的存贮机制，他需要用户打开客户端的cookie支持。cookie的作用就是为了解决HTTP协议无状态的缺陷所作的努力.而Session机制采用的是一种在客户端与服务器之间保持状态的解决方案。
由于采用服务器端保持状态的方案在客户端也需要保存一个标识，所以Session机制可能需要借助于cookie机制来达到保存标识的目的。而Session提供了方便管理全局变量的方式。
正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie。

localStorage和sessionStorage：
html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。
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。
而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

  localStorage和sessionStorage都具有相同的**操作方法**，例如setItem、getItem和removeItem等。
  setItem存储value
  用途：将value存储到key字段
  用法：.setItem(key, value)
  代码示例：
  sessionStorage.setItem("key", "value"); localStorage.setItem("site", “js8.in");

  getItem获取value

  用途：获取指定key本地存储的值
  用法：.getItem(key)
  代码示例：

  var value = sessionStorage.getItem("key"); var site =
  localStorage.getItem(“site");

  removeItem删除key

  用途：删除指定key本地存储的值
  用法：.removeItem(key)
  代码示例：

  sessionStorage.removeItem("key"); localStorage.removeItem("site");

clear清除所有的key/value

用途：清除所有的key/value
用法：.clear()
代码示例：

sessionStorage.clear(); localStorage.clear();

其他操作方法：点操作和[]

localStorage和sessionStorage的key和length**属性实现遍历**

sessionStorage和localStorage提供的key()和length可以方便的实现存储的数据遍历，例如下面的代码：
var storage = window.localStorage; for (var i=0, len = storage.length; i < len; i++){ var key = storage.key(i); var value = storage.getItem(key); console.log(key + "=" + value); }

storage还提供了storage事件，当键值改变或者clear的时候，就可以触发storage事件，如下面的代码就添加了一个storage事件改变的监听：
if(window.addEventListener){ window.addEventListener("storage",handle_storage,false); }else if(window.attachEvent){ window.attachEvent("onstorage",handle_storage); } function handle_storage(e){ if(!e){e=window.event;} }
storage事件对象的具体属性如下表：

Property	Type	Description
key	String	The named key that was added, removed, or moddified
oldValue	Any	The previous value(now overwritten), or null if a new item was added
newValue	Any	The new value, or null if an item was added
url/uri	String	The page that called the method that triggered this change

DOMContentLoaded事件和window.onload=function(){}和$(document).ready(function(){})的区别：

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。

```
window.addEventListener("load", function() {
  // ...代码...
}, false);
document.addEventListener("DOMContentLoaded", function() {
  // ...代码...
}, false);
```

DOMContentLoaded仅当DOM加载完成就触发，不包括样式表、图片、flash等。

$(document).ready()是DOM结构绘制完毕后就执行，不必等到加载完毕，就是用DOMContentLoaded事件实现的。

2.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 (貌似是最后那个)

$(document).ready()可以同时编写多个，并且都可以得到执行

3.简化写法

document.onDOMContentLoaded = function(){}

window.onload没有简化写法

$(document).ready(function(){})可以简写成$(function(){});

jquery寻找子元素

$('#wrapper').children(); //（只沿着 DOM 树向下遍历单一层级）查询直接的子元素。而不管子元素的子元素。

$('#wrapper').html(); //返回的是dom结构。而不是集合

$('#wrapper').contents(); 获得匹配元素集合中每个元素的子节点，包括文本和注释节点。跟children不同的是在结果 jQuery 对象中包含了文本节点以及 HTML 元素（包括注释节点）

$('#wrapper').find("all"); //find() 方法允许我们在 DOM 树中搜索这些元素的后代，与children不同的是，可以沿着DOM树向下遍历好几层。（没有all这个元素）

js中的几个迭代方法：

1、filter：

对数组进行迭代，返回一个满足要求的新数组

如果要对对象中的值或者键进行过滤，可以采用下面的方法：

```
var data = {a: 1, b: 2, c: 3, d: 4};
console.log(Object.keys(data));//得到对象的键组成的数组
data1 = Object.keys(data).filter(function(x) { return data[x] > 2;})
console.log(data1);//符合要求的键组成的数组
data2 = Object.values(data).filter(function(x){return x > 2;})
console.log(data2);//符合要求的值组成的数组
```

Object.values()和Object.keys()可以得到对象的键值或者键名，返回的是数组。

2、sort：

对数组进行排序，当回调函数返回的为1表示两个值需要交换位置

如果是对象数组，要根据多个属性值来进行排序，例如先根据b排序，如果b一样再根据a排序：

```
var arr = [
  { a : 2, b : 3.2},
  { a : 3, b : 1.2},
  { a : 4, b : 2.2},
  { a : 6, b : 1.2},
```

```
    { a : 5, b : 3.2}
]
arr.sort(function (x, y) {
    if (x.b > y.b) {
        return 1;
    } else if (x.b === y.b) {
        return x.a > y.a ? 1 : -1;
    } else if (x.b < y.b) {
        return -1;
    }
})
```

页面总的各种宽高：
screen.height/screen.width：屏幕的宽高
outerHight/outerWidth：浏览器窗口大小
innerHeight/innerHeight：（HTML的）视图区大小
在IE、Firefox、Safari、Opera、Chrome中，
document.documentElement.clientWidth和
document.documentElement.clientHeight，可以得到页面视口大小。在IE6中，这些属性必须在标准模式下才有效；如果是混杂模式，就必须通过**document.body.clientWidth**和document.body.clientHeight取得。而对于混杂模式下的Chrome，两种方式都有取得视口大小。
窗口中文档显示区域的高度，不包括菜单栏、工具栏等部分。该属性可读可写。
IE不支持该属性，IE中body元素的clientHeight属性与该属性相同。
document.body.clientHeight/document.body.clientWidth：（body的）可见视口，inner的大小减去滚动条？
要获取页面视口大小可以通过如下顺序获取：**window.innerWidth---->document.documentElement.clientWidth--->document.body.clientWidth**
窗口缩放
window.resizeTo(x,y)
window.resizeBy(x,y)

pageXOffset/pageYOffset：向右/向下滚动的距离。
整数只读属性，表示文档向右滚动过的像素数。
IE不支持该属性，使用body元素的scrollLeft属性替代。

窗口位置：
screenLeft/screenTop：窗口相对于屏幕的位置（IE、Safari、Opera、Chrome）
screenX/screenY:窗口相对于屏幕的位置（Firefox、Chrome、Safari，Opera也能取值，但表现不一样，不建议）
screenLeft/screenTop相对于IE、Opera：屏幕到页面可见区域
screenY / screenTop相对于Chrome：屏幕到整个浏览器
当页面有外边距：Firefox、Safari、Chrome：使用screenX/screenY返回相同的值
IE、Opera：给出页边距相对于屏幕边界的精确坐标
窗口移动：
window:moveTo(x,y)
window.moveBy(x,y)

事件的三种宽高：
screenX:鼠标位置相对于用户屏幕水平偏移量，而screenY也就是垂直方向的，此时的参照点也就是原点是屏幕的左上角。
clientX:跟screenX相比就是将参照点改成了浏览器内容区域的左上角，该参照点会随之滚动条的移动而移动。
pageX：参照点也是浏览器内容区域的左上角，但它不会随着滚动条而变动

元素的大小：
偏移量：
offsetHeight：元素在垂直方向上占用的空间大小
offsetWidth
offsetLeft：元素的左外边框至包含元素的左内边框之间的像素
offsetTop
客户区大小：内容和内边框
element.clientHeight
element.clientWidth
滚动大小：
scrollHeight:包含滚动内容的高度，指的是元素自身带有滚动条时，总的高度
scrollWidth

scrollLeft：隐藏在内容区域左侧的像素数，指的是元素自身带有滚动条时

scrollTop

getBoundingClientRect()获得四个角的坐标

"JavaScript内部，所有数字都是以64位浮点数形式储存，即使整数也是如此。"

浏览器的检测：

1、navigator对象

navigator.userAgent："Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"；

navigator.appVersion："5.0 (Macintosh; Intel Mac OS X 10_11_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"；

navigator.appName："Netscape"。

2、不同浏览器的特征：

addEventListener等

<noscript></noscript>：当浏览器不支持脚本，或者浏览器支持脚本但脚本被禁用的时候，该元素中的内容才会被显示出来。

typeof a，a未定义，可以返回undefined。

History对象的属性和方法：

go()　　加载history列表中的某个具体页面；

back()　回到浏览器载入历史URL地址列表的当前URL的前一个URL；
forward() 转到浏览器载入历史URL地址列表的当前URL的下一个URL；
length　保存历史URL地址列表的长度信息。

浏览器存储技术：
IndexdDB：IndexdDB 是 HTML5 的本地存储，把一些数据存储到浏览器（客户端）中，当与网络断开时，可以从浏览器中读取数据，用来做一些离线应用。
IndexedDB 是一种低级API，用于客户端存储大量结构化数据(包括, 文件/blobs)。该API使用索引来实现对该数据的高性能搜索。虽然 Web Storage 对于存储较少量的数据很有用，但对于存储更大量的结构化数据来说，这种方法不太有用。IndexedDB提供了一个解决方案。
Cookie： 通过在客户端 (浏览器) 记录信息确定用户身份，最大为 4 kb 。
Session： 是服务器端使用的一种记录客户端状态的机制 。
localStorage： 也是 HTML5 的本地存储，将数据保存在客户端中（一般是永久的）。
userData：IE浏览器可以使用userData来存储数据，容量可达到640K，这种方案是很可靠的，不需要安装额外的插件。缺点：它仅在IE下有效。
浏览器传输数据：
url：get方式向后端传输数据，不超过2k
post：不限制传输大小

sort()不生成副本

contentWindow：可以取得子窗口的window 对象
contentDocument：可以取得子窗口的document对象

抛出错误与try-catch：
应用程序的较低层次中抛出错误，这个层次并不影响当前执行的代码，错误得不到解决。
在多个应用程序内部使用或者在应用程序内部多个地方使用，在抛出错误时尽可能详尽，并捕获处理这些错误。
捕获的时候，应该只捕获那些确切知道如何处理的错误；抛出的时候，应该提供具体原因。

requestAnimationFrame：轮播

document.bgColor：文档背景颜色

IE事件和DOM事件的不同

1、实行顺序：DOM以添加的顺序执行；IE相反

2、参数：DOM3个，IE两个

3、事件有没有on

4、this：DOM的this指向currentTarget；IE的this根据制定的方式确定

5、event和target的获取方式

push和unshift方法，返回数组的长度

AMD, CMD, CommonJS和UMD

CommonJS

CommonJS是服务器端模块的规范，Node.js采用了这个规范。

根据CommonJS规范，一个单独的文件就是一个模块。加载模块使用require方法，该方法读取一个文件并执行，最后返回文件内部的exports对象。

AMD和RequireJS

CMD和SeaJS

- 对于依赖的模块AMD是提前执行，CMD是延迟执行。不过RequireJS从2.0开始，也改成可以延迟执行（根据写法不同，处理方式不通过）。
- CMD推崇依赖就近，AMD推崇依赖前置。

UMD

UMD是AMD和CommonJS的糅合

UMD先判断是否支持Node.js的模块（exports）是否存在，存在则使用Node.js模块模式。

在判断是否支持AMD（define是否存在），存在则使用AMD方式加载模块。

Flash

Flash提供了ExternalInterface接口与JavaScript通信，ExternalInterface有两个方法，call和addCallback，call的作用是让Flash调用js里的方法，addCallback是用来注册flash函数让js调用。

javascript执行：

head 部分中的脚本: 需调用才执行的脚本或事件触发执行的脚本放在HTML的head部分中。当你把脚本放在head部分中时，可以保证脚本在任何调用之前被加载。

body 部分中的脚本: 当页面被加载时立即执行的脚本放在HTML的body部分。放在body部分的脚本通常被用来生成页面的内容。

1：将JavaScript标识放置<Head>... </Head>在头部之间，使之在主页和其余部分代码之前预先装载，从而可使代码的功能更强大； 比如对*.js文件的提前调用。 也就是说把代码放在<head>区在页面载入的时候，就同时载入了代码，你在<body>区调用时就不需要再载入代码了，速度就提高了，这种区别在小程序上是看不出的，当运行很大很复杂的程序时，就可以看出了。

当然也可以将JavaScript标识放置在<Body>... </Body>主体之间以实现某些部分动态地创建文档。 这里比如制作鼠标跟随事件，肯定只有当页面加载后再进行对鼠标坐标的计算。或者是filter滤镜与javascript的联合使用产生的图片淡入淡出效果

2：放入html的head,是页面加载前就运行，放入body中，则加载后才运行javascript的代码~~~所以head里面的先执行。

判断是不是Array：

1、arr instanceof Array　//在跨frame的时候会失效

2、Object.prototype.toString.call(arr) === '[Object Array]' 比第一种准确

3、Array.isArray(arr)

4、arr.constructor() === Array

setTimeout()不是全局函数，全局函数有：

函数	描述
decodeURI()	解码某个编码的 URI。
decodeURIComponent()	解码一个编码的 URI 组件。
encodeURI()	把字符串编码为 URI。
encodeURIComponent()	把字符串编码为 URI 组件。
escape()	对字符串进行编码。
eval()	计算 JavaScript 字符串，并把它作为脚本代码来执行。
getClass()	返回一个 JavaObject 的 JavaClass。
isFinite()	检查某个值是否为有穷大的数。
isNaN()	检查某个值是否是数字。
Number()	把对象的值转换为数字。
parseFloat()	解析一个字符串并返回一个浮点数。
parseInt()	解析一个字符串并返回一个整数。
String()	把对象的值转换为字符串。
unescape()	对由 escape() 编码的字符串进行解码。

短语元素：

	把文本定义为强调的内容。
	把文本定义为语气更强的强调的内容。
<dfn>	定义一个定义项目。
<code>	定义计算机代码文本。
<samp>	定义样本文本。
<kbd>	定义键盘文本。它表示文本是从键盘上键入的。它经常用在与计算机相关的文档或手册中。
<var>	定义变量。您可以将此标签与 <pre> 及 <code> 标签配合使用。
<cite>	定义引用。可使用该标签对参考文献的引用进行定义，比如书籍或杂志的标题。