目标

学习目标

- 能够使用js中的事件委托
- 能够使用事件对象获取鼠标的位置
- 能够完成跟随鼠标的天使案例
- 能够说出常用的3个键盘事件
- 能够知道如何获取当前键盘按下的是哪个键

- 能够知道浏览器的顶级对象window
- 能够使用window.onload事件
- 能够使用window.onresize事件
- 能够说出两种定时器的区别
- 能够使用location对象的href属性完成页面之间的跳转
- 能够使用location对象获取url中的参数部分

事件委托

事件冒泡本身的特性，会带来的坏处，也会带来的好处。

什么是事件委托

```
把事情委托给别人，代为处理。

事件委托也称为事件代理，在 jQuery 里面称为事件委派。

说白了就是，不给子元素注册事件，给父元素注册事件，把处理代码在父元素的事件中执行。
```

生活中的代理:

```
咱们班级有60多个同学,快递员那有60多个快递,
如果一个个的送,花费时间比较长,同时每个同学在领取的时候,也需要排队领取,
也花费时间比较长,该如何提高效率,节省时间呢?

解决方法: 快递员把60个快递,"委托"给班主任老师,班主任把这些快递放在办公室,
同学们下课自行领取即可.

优势:快递员省事,委托给班主任就可以走了. 同学们领取也方便,因为相信班主任.
```

js事件中的代理:

```
<ul>
    <li>知否知否，点我应有弹框在手！</li>
    <li>知否知否，点我应有弹框在手！</li>
    <li>知否知否，点我应有弹框在手！</li>
    <li>知否知否，点我应有弹框在手！</li>
    <li>知否知否，点我应有弹框在手！</li>
</ul>
```

```
8
9  点击每个li都会弹出对话框,以前需要给每个li都注册事件,是非常辛苦的,而且访问DOM的次数越多,
10  容易延迟整个页面的交互就绪时间
```

事件委托的原理

给父元素注册事件，利用事件冒泡，当子元素的事件触发，会冒泡到父元素，然后去控制相应的子元素。

事件委托的作用

- 我们只操作了一次 DOM ， 提高了程序的性能。
- 动态新创建的子元素，也拥有事件。

举例:

```
// 事件委托的核心原理：给父节点添加侦听器， 利用事件冒泡影响每一个子节点
// e.target 这个可以得到我们点击的对象
  <ul>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
        <li>知否知否，点我应有弹框在手！</li>
    </ul>
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <ul>
        <li>知否知否，点我应有弹框在手！1111</li>
        <li>知否知否，点我应有弹框在手！2222</li>
        <li>知否知否，点我应有弹框在手！33333</li>
        <li>知否知否，点我应有弹框在手！4444</li>
        <li>知否知否，点我应有弹框在手！55555</li>
    </ul>
    <script>
        // 我们实现每个li都可以点击,点击以后弹出当前li的标签内容

        // 以前的做法,我们会先获取所有的li标签,循环给每个li绑定事件
        /* var lis = document.querySelectorAll("ul li");
        for(var i=0;i<lis.length;i++){
            lis[i].onclick=function(){
                alert(this.innerHTML);
            }
        } */


        // 事件委托,就是不绑定事件在子元素,绑定事件在父元素上,借助事件冒泡的原理,触发子元素事件的时候,会冒泡到父元素事件
        var ul = document.querySelector("ul");
        ul.onclick=function(e){
            //  假设我点击了li 这里的this代表ul
            // 因为this是给谁绑定事件,就代表谁
            // console.log(  this );
            // console.log( this.innerHTML );


            // 如何获取事件对象
            e = e || window.event;
            // 使用事件对象的 e.target属性获取 触发事件目标对象
            // console.log( e.target );
            alert( e.target.innerHTML )
        }
    </script>
</body>
</html>
```

常用鼠标事件

鼠标事件	触发条件
onclick	鼠标点击左键触发
onmouseover	鼠标经过触发
onmouseout	鼠标离开触发
onfocus	获取鼠标焦点触发
onblur	失去鼠标焦点触发
onmousemove	鼠标移动触发

onmouseup	鼠标弹起触发
onmousedown	鼠标按下触发

案例:禁止选中文字和禁止右键菜单

禁止鼠标右键菜单　contextmenu主要控制应该何时显示上下文菜单,主要用于程序员取消默认的上下文菜单

```
document.addEventListener("contextmenu",function(e){
    e.preventDefault()
})
```

禁止鼠标选中内容(selectstart 开始选中)

```
document.addEventListener("selectstart",function(e){
    e.preventDefault()
})
```

举例:

> 我是一段不愿意分享的文字

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    我是一段不愿意分享的文字
    <script>
        // 禁止鼠标右键菜单      contextmenu主要控制应该何时显示上下文菜单,主要用于程序员取消默认的上下文菜单
        document.addEventListener("contextmenu",function(e){
            e = e || window.event;
            // 阻止默认行为
            e.preventDefault();
        })


        // 禁止鼠标选中内容( selectstart 开始选中)
        document.addEventListener("selectstart",function(e){
            e = e || window.event;
            // 阻止默认行为
            e.preventDefault();
        })
    </script>
</body>
</html>
```

鼠标事件对象

event 事件对象是事件相关的一系列信息的集合.

现阶段我们主要用鼠标事件对象**MouseEvent** 和 键盘事件对象**KeyBoardEvent**

鼠标事件对象	说明
e.clientX	返回鼠标相对于**浏览器窗口可视区**的X坐标
e.clientY	返回鼠标相对于**浏览器窗口可视区**的Y坐标
e.pageX	返回鼠标相对于**文档页面**的X坐标 IE9+支持
e.pageY	返回鼠标相对于**文档页面**的Y坐标 IE9+支持
e.x	返回鼠标相对于**文档页面**的X坐标 **IE9以下使用**
e.y	返回鼠标相对于**文档页面**的X坐标 **IE9以下使用**
e.screenX	返回鼠标相对于**电脑屏幕**的X坐标
e.screenY	返回鼠标相对于**电脑屏幕**的Y坐标

获取鼠标在页面的坐标

> 设置body高度3000,背景颜色为天蓝色

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        /* 设置body高度3000,背景颜色为天蓝色 */
        body{
            height: 3000px;
            width: 2500px;
            background:skyblue;
        }
    </style>
</head>
<body>
    <script>
        // 给document绑定鼠标单击事件
        // console.log( document );
        document.onclick=function(e){
            e = e || window.event;
            // 鼠标事件对象 说明
            // e.clientX 返回鼠标相对于浏览器窗口"可视区"的X坐标
            // e.clientY 返回鼠标相对于浏览器窗口"可视区"的Y坐标
            console.log("可视区坐标");
```

```
            console.log( e.clientX );
            console.log( e.clientY );
            console.log("");
            // // e.pageX 返回鼠标相对于"文档页面"的X坐标 IE9+支持
            // // e.pageY 返回鼠标相对于"文档页面"的Y坐标 IE9+支持
            console.log("文档页面坐标");
            console.log( e.pageX );
            console.log( e.pageY );
            console.log("");


            // e.x 返回鼠标相对于文档页面的X坐标 IE9以下使用
            // e.y 返回鼠标相对于文档页面的X坐标 IE9以下使用
            console.log("返回鼠标相对于文档页面的坐标 IE9以下使用");
            console.log( e.x );
            console.log( e.y );
            console.log("");


            console.log("获取鼠标相对与文档页面的坐标,兼容写法");
            console.log( e.pageX || e.x );
            console.log( e.pageY || e.y );
            console.log("");


            // e.screenX 返回鼠标相对于电脑屏幕的X坐标
            // e.screenY 返回鼠标相对于电脑屏幕的Y坐标
            console.log("鼠标相对于电脑屏幕坐标");
            console.log( e.screenX );
            console.log( e.screenY );
        }
    </script>
</body>
</html>
```

常用的键盘事件

键盘事件

键盘时间	触发条件
onkeyup	某个键盘按键被松开时触发
onkeydown	某个键盘按键被按下时触发
onkeypress	某个键盘按键被按下时触发 但是它不识别功能键 比如 ctrl shift 箭头等

注意:

1. **如果使用addEventListener不需要加on**
2. onkeypress和前面两个onkeyup跟onkeydown的区别是 **onkeypress不识别功能键,比如箭头,ctrl,shift 等**
3. 三个事件的执行顺序是: **keydown=>keypress=>keyup**

举例:

```
// 给document分别绑定三个键盘事件
// 1. keydown 按键按下的时候触发 能识别功能键 比如 ctrl shift 左右箭头啊
// 2. keypress 按键按下的时候触发 不能识别功能键 比如 ctrl shift 左右箭头啊
// 3. keyup 按键弹起的时候触发
// 4. 三个事件的执行顺序 keydown -- keypress -- keyup
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // onkeyup 某个键盘按键被松开时触发
        document.addEventListener("keyup",function(){
            console.log( "keyup某个键松开了");
        });
        // onkeypress 某个键盘按键被按下时触发 但是它"不识别功能键" 比如 ctrl shift 箭头等
        document.addEventListener("keypress",function(){
            console.log( "keypress某个键被按下了");
        });
        // onkeydown 某个键盘按键被按下时触发
        document.addEventListener("keydown",function(){
            console.log( "keydown某个键被按下了");
        });
        // 三个事件执行的顺序是 keydown=>keypress=>keyup
    </script>
</body>
</html>
```

键盘事件对象

键盘事件对象属性	说明
keyCode	返回该键的ASCII值

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符										ASCII 打印字符													
		0000					0001					0010		0011		0100		0101		0110		0111			
		0					1					2		3		4		5		6		7			
		十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	‼	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	▬	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	`Back space

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

注意:

1. **onkeydown和onkeyup不区分字母大小写,onkeypress区分字母大小写**
2. 在我们实际开发中,我们更多的使用`keydown`和`keyup`, 它能识别所有的键(包括功能键)
3. **keypress不识别功能键,但是keyCode属性能够区分大小写**,返回不同的ASCII值

不区分大小写说明

```
我们的keyup 和keydown事件不区分字母大小写 a 和 A 得到的都是65
我们的keypress 事件 区分字母大小写 a得到97          而A得到的是65
```

A=> 65

a=>97

键盘keydown了,keyCode是65
键盘keypress了,keyCode是65　　老师按下的是大写的A
键盘keyup了,keyCode是65

键盘keydown了,keyCode是65
键盘keypress了,keyCode是97　　老师按下的是小写的a
键盘keyup了,keyCode是65

举例: 使用keyCode属性判断用户按下哪个键

```
// 键盘事件对象中的keyCode属性可以得到相应键的ASCII码值
// 1. 我们的keyup 和keydown事件不区分字母大小写 a 和 A 得到的都是65
// 2. 我们的keypress 事件 区分字母大小写 a 97 和 A 得到的是65
// 我们可以利用keycode返回的ASCII码值来判断用户按下了那个键
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 键盘事件对象中有一个keyCode,可以记录触发键盘事件的时候的那个键的ASCII值
        // onkeydown 某个键盘按键被按下时触发
        document.addEventListener("keydown",function(e){
            e = e || window.event;
            console.log("键盘keydown了,keyCode是" + e.keyCode );
        });
        // onkeypress 某个键盘按键被按下时触发 但是它"不识别功能键" 比如 ctrl shift 箭头等
        document.addEventListener("keypress",function(e){
            e = e || window.event;
            console.log("键盘keypress了,keyCode是" + e.keyCode );
        });
        // onkeyup 某个键盘按键被松开时触发
        document.addEventListener("keyup",function(e){
            e = e || window.event;
            console.log("键盘keyup了,keyCode是" + e.keyCode );
        });
        // 注意: keypress获取的keyCode码是区分字母大小写的,而keydown和keyup是不大小写的
    </script>
</body>
</html>
```

案例:模拟京东按键输入内容

```
1 当我们按下 s 键， 光标就定位到搜索框（文本框获得焦点）
```

分析:

```
1 1. 核心思路:检测用户是否按下s键,如果按下s键,就把光标定位到搜索框里面
2 2. 使用键盘事件对象里面的keyCode判断用户按下的是否是s键
3 3. 搜索框获取焦点: 使用js里面的focus()方法
```

注意: 触发获得焦点事件，可以使用 元素对象.focus()

代码:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <input type="text"/>
    <script>
        // 当我们按下 s 键， 光标就定位到搜索框（文本框获得焦点）
        // 1. 核心思路:检测用户是否按下s键,如果按下s键,就把光标定位到搜索框里面
        // 2. 使用键盘事件对象里面的keyCode判断用户按下的是否是s键
        // 3. 搜索框获取焦点: 使用js里面的focus()方法


        // 获取文本框对象
        var input = document.querySelector("input");


        // 给document绑定键盘事件  keydown或者keypress(不识别功能)
        document.onkeydown=function(e){
            e = e || window.event;
            // 判断用户按下的键是否为s键  s的keyCode是83
            // console.log( e.keyCode );
            if(e.keyCode == 83 ){
                // 搜索框获取焦点: 使用js里面的 对象.focus() 方法
                input.focus();
            }
        }
    </script>
</body>
</html>
```

案例:跟随鼠标的天使

> 这个天使图片一直跟随鼠标移动

分析:

```
1. 鼠标不断的移动,使用的鼠标事件 mousemove
2. 在页面中移动,所以是给document注册事件
3. 图片要移动距离,而且不占位置,我们给图片加绝对定位即可
4. 核心原理:每次鼠标移动,我们都会获得最新的鼠标坐标,把这个x和y坐标作为图片的top和left值就可以移动图片了
```

```html
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        img{
            position: absolute;
        }
    </style>
</head>


<body>
    <img src="images/angel.gif"/>
    <script>
        // 因为图片要跟随鼠标整个网页中移动,所以我们给document绑定鼠标移动事件


        // 获取img对象
        var img = document.querySelector("img");


        // 鼠标移动事件
        document.onmousemove=function(e){
            e = e || window.event;
            // 获取鼠标在当前文档中的位置
            // console.log( e.pageX );
            // console.log( e.pageY );


            // e.pageX跟e.pageY只有IE9以及IE9以后的浏览器才可以使用
            // IE9之前的需要使用e.x  e.y
            var x = e.pageX || e.x;
            var y = e.pageY || e.y;


            // 把鼠标的坐标位置给img,注意:静态定位下,偏移量无效
            img.style.left = (x-40)+"px";
            img.style.top = (y-34)+"px";
        }

    </script>
</body>


</html>
```

案例:模拟京东快递单号查询

要求:当我们在文本框中输入内容时，文本框上面自动显示大字号的内容。

分析:

1. 快递单号输入内容时,上面的大号字体盒子(con)显示
2. 同时把快递单号里面的值(value)获取过来赋值给con盒子(innerText)做为内容
3. 如果快递单号里面内容为空,则隐藏大号字体盒子(con)盒子
4. 注意:keydown和keypress在文本框里面的特点:他们两个事件触发的时候,文字还没落入文本框中
5. keyup事件触发的时候,文字已经落入文本框里面了
6. 当我们失去焦点,就隐藏这个con盒子
7. 当我们获取焦点,并且文本框内容不为空,就显示这个con盒子

代码:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        .search {
            position: relative;
            width: 178px;
            margin: 100px;
        }

        .con {
            display: none;
            position: absolute;
            top: -40px;
            width: 171px;
            border: 1px solid rgba(0, 0, 0, .2);
            box-shadow: 0 2px 4px rgba(0, 0, 0, .2);
```

```
            box-shadow: 0 2px 4px rgba(0, 0, 0, .2);
            padding: 5px 0;
            font-size: 18px;
            line-height: 20px;
            color: #333;
        }

        .con::before {
            content: '';
            width: 0;
            height: 0;
            position: absolute;
            top: 28px;
            left: 18px;
            border: 8px solid #000;
            border-style: solid dashed dashed;
            border-color: #fff transparent transparent;
        }
    </style>
</head>


<body>
    <div class="search">
        <div class="con"></div>
        <input type="text" placeholder="请输入您的快递单号" class="jd">
    </div>
    <script>
        // 获取input对象
        var jd = document.querySelector(".jd");
        // 获取.con内容层
        var con = document.querySelector(".con");

        // input获取焦点的时候,并且内容不为空,显示.con内容框
        jd.onfocus = function(){
            if( jd.value!="" ){
                con.style.display = "block";
            }
        }
        // input失去焦点的时候,隐藏.con内容框
        jd.onblur = function(){
            con.style.display = "none";
        }
        // input输入内容的时候,显示.con内容框,并且把输入内容放进.con的标签内容中;当input中内容为空时候,隐藏.con内容层
        jd.onkeyup = function(){
            con.style.display = "block";
            con.innerHTML = jd.value;


            if(jd.value == ""){
                con.style.display = "none";
            }
        }


    </script>
</body>


</html>
```

BOM

什么是BOM

BOM（Browser Object Model）即浏览器对象模型，它提供了独立于内容而与浏览器窗口进行交互的对象，其核心对象是 window。

BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

BOM 缺乏标准，JavaScript 语法的标准化组织是 ECMA，DOM 的标准化组织是 W3C，BOM 最初是Netscape 浏览器标准的一部分。

DOM

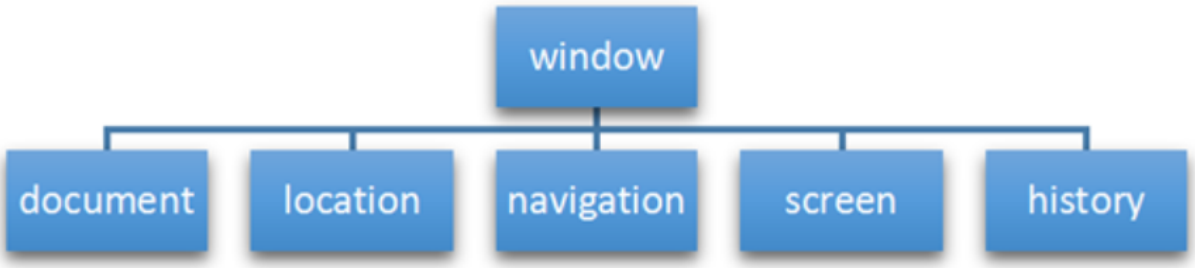
- 文档对象模型
- DOM 就是把「文档」当做一个「对象」来看待
- DOM 的顶级对象是 document
- DOM 主要学习的是操作页面元素
- DOM 是 W3C 标准规范

BOM

- 浏览器对象模型
- 把「浏览器」当做一个「对象」来看待
- BOM 的顶级对象是 window
- BOM 学习的是浏览器窗口交互的一些对象
- BOM 是浏览器厂商在各自浏览器上定义的，兼容性较差

BOM的构成

BOM 比 DOM 更大，它包含 DOM。



顶级对象window

window对象是浏览器的顶级对象,它具有双重角色.

1. 它是JS访问浏览器窗口的一个接口.
2. 它是一个全局对象. 定义在全局作用域中的变量、函数都会变成window对象的属性和方法
3. 在调用的时候可以省略window, 前面学习alert(),prompt()都属于window对象的方法
4. **注意:window下有一个特殊属性window.name**

举例:

```
使用window调用定义的函数和变量
console.dir(window)
window.name特殊属性
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 顶级对象window
        // window对象是浏览器的顶级对象,它具有双重角色.
        // 它是JS访问浏览器窗口的一个接口.
        // 它是一个全局对象. 定义在全局作用域中的变量、函数都会变成window对象的属性和方法
        // 在调用的时候可以省略window, 前面学习alert(),prompt()都属于window对象的方法
        // 注意:window下有一个特殊属性window.name


        // 定义一个变量
        var a = 10;
        console.log( a );
        // 因为window是一个全局对象,在全局作用域下定义的变量和函数,都会变成window对象的属性和方法
        console.log( window.a );
        console.log("");


        // 定义函数
        function fn(){
            console.log("hello");
            // 普通函数里面的this代表window对象
            console.log( this );
        }
        // 调用函数
        fn();
        window.fn();


        // 只要是window对象的方法和属性,就可以省略"window." 比如window.alert()=>alert()
        // 弹窗
        // window.alert("我是弹窗内容");
        // alert("我也是弹窗内容")
        console.log("");


        // window下有一个特殊属性window.name 是window自带的属性,window.name的类型是字符串
        console.log( window );
        console.log( typeof( window.name ) );
        console.log("");

        // 定义变量的时候,请谨慎使用name作为变量名,因为如果使用name变量名以后,不管值是什么,都会变成字符串类型
        var name = 123456;
        console.log( name );
        console.log( typeof(name) );


    </script>
</body>
</html>
```

window对象的属性

属性	功能
window.innerWidth	获取浏览器内部的宽度(不包含浏览器的左右两边边框)
window.innerHeight	获取浏览器内部的高度(不包含浏览器的菜单栏和地址栏以及上下的边框)
window.outerWidth	获取整个浏览器的宽度

window.outerHeight	获取整个浏览器的高度

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // window对象的属性
        // window.innerWidth 获取浏览器内部的宽度(不包含浏览器的左右两边边框)
        console.log("浏览器内部的宽度" + window.innerWidth);

        // window.innerHeight 获取浏览器内部的高度(不包含浏览器的菜单栏和地址栏以及上下的边框)
        console.log("浏览器内部的高度" + window.innerHeight);

        // window.outerWidth 获取整个浏览器的宽度
        console.log("整个浏览器的宽度" + window.outerWidth);

        // window.outerHeight 获取整个浏览器的高度
        console.log("整个浏览器的高度" + window.outerHeight);
    </script>
</body>
</html>
```

window弹出框方法

方法	**功能**
window.alert(内容)	向浏览器窗口中弹出一个警告提示框
window.prompt(内容)	向浏览器窗口中弹出一个用户输入对话框
window.confirm(内容)	向浏览器窗口中弹出一个**用户确认框** 如果点击了确定按钮会返回一个true如果点击了取消按钮会返回一个false

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // window.alert("内容");  弹窗显示内容
        window.alert("我是弹窗内容");


        // window.prompt("提示内容","默认值")  用户输入框  如果用户点击了取消,得到null;不管用户输入的是什么,最后的数据类型都是string
        var str = window.prompt("请输入你的账号","zhangsan");
        console.log( str );
        console.log( typeof str );

        // window.confirm("提示信息")  用户确认框  点击确定返回true,点击取消返回false
        console.log( window.confirm("你确定要删除我吗~") );
    </script>
</body>
</html>
```

window对象的常见事件

页面（窗口）加载事件（两种）

第1种 常用,要记住

```
window.onload = function(){}
或者
window.addEventListener("load",function(){});
```

window.onload 是窗口 (页面) 加载事件，**当文档内容完全加载完成**会触发该事件(包括图像、脚本文件、CSS 文件等), 就调用的处理函数。

注意:

1. 有了window.onload就可以把JS代码写在页面元素的上方,因为onload是等页面内容全部加载完毕,再去执行处理函数的
2. window.onload传统注册事件方式 只能写一次，如果有多个,会以最后一个window.onload为准
3. 如果使用addEventListener方法注册事件,则没有限制

第2种

```
document.addEventListener("DOMContentLoaded",function(){})
```

DOMContentLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等等。IE9以上才支持！！！

DOMContentLoaded 加载速度比 load更快一些

如果页面的图片很多的话, 从用户访问到onload触发可能需要较长的时间, 交互效果就不能实现，必然影响用户的体验，此时用 DOMContentLoaded 事件比较

合适。

举例:

```
多个window.onload同时存在
多个addEventListener添加load事件
document.addEventListener("DOMContentLoaded",function(){})
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        // 页面（窗口）加载事件（两种）
        // 第1种    常用,要记住
        /* window.onload = function(){}
        或者
        window.addEventListener("load",function(){}); */


        /* window.onload 是窗口 （页面）加载事件，当文档内容完全加载完成会触发该事件(包括图像、脚本文件、CSS 文件等), 就调用的处理函数。
        注意:
        有了window.onload就可以把JS代码写在页面元素的上方,因为onload是等页面内容全部加载完毕,再去执行处理函数的
        window.onload传统注册事件方式 只能写一次 ， 如果有多个,会以最后一个window.onload为准
        如果使用addEventListener方法注册事件,则没有限制 */



        // 如果直接在标签的上面使用js获取对象,容易获取失败得到null
        // 但是我们可以把获取对象的代码,放在window.onload事件中,给window对象绑定onload事件
        // window.onload代表页面加载完成时候,才执行函数里面的代码

        // 传统的注册事件方式
        /* window.onload = function(){
            var div = document.querySelector("div");
            console.log( div ) ;
            console.log( "页面加载完成了111111111" );
        }
        window.onload = function(){
            console.log( "页面加载完成了222222222" );
        } */


        // 监听方式注册事件
        window.addEventListener("load",function(){
            var div = document.querySelector("div");
            console.log( div ) ;
            console.log( "页面加载完成了111111111" );
        });


        window.addEventListener("load",function(){
            console.log( "页面加载完成了222222222" );
        });


        window.addEventListener("load",function(){
            console.log( "页面加载完成了3333333" );
        });


         /* 第2种
        document.addEventListener("DOMContentLoaded",function(){}) */


        /* DOMContentLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等等。IE9以上才支持！！！
        DOMContentLoaded 加载速度比 load 更快一些
        如果页面的图片很多的话，从用户访问到onload触发可能需要较长的时间，交互效果就不能实现，必然影响用户的体验，此时用 DOMContentLoaded 事件比较合适。 */
        document.addEventListener("DOMContentLoaded",function(){
            console.log("DOMContentLoaded 加载速度比 load 更快一些,因为我只要加载完成DOM我就执行,而load需要等到所有的东西加载完成才执行");
        })


        // 好消息,后面这种DOMContentLoaded几乎不用用到,常用的就是window.onload事件
    </script>
</head>
<body>
    <div>div内容</div>
</body>
</html>
```

调整窗口大小事件

```
window.onresize = function(){}
或者
window.addEventListener("resize",function(){})
```

window.onresize 是调整窗口大小加载事件，当触发时就调用的处理函数。

注意:

1. 只要窗口大小发生像素变化，就会触发这个事件。
2. 我们经常利用这个事件完成响应式布局。 **window.innerWidth** 获取当前屏幕的宽度

举例:

```
页面中有一个div,当window.innerWidth小于等于800的时候就隐藏,否则显示
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        div{
            width: 200px;
            height: 200px;
            background: pink;
        }
    </style>
</head>
<body>
    <!-- 页面中有一个div,当window.innerWidth小于等于800的时候就隐藏,否则显示 -->
    <div>div的内容</div>


    <script>
        // 调整窗口大小事件,会在浏览器的窗口大小变化的时候触发事件
        /* window.onresize = function(){}
        或者
        window.addEventListener("resize",function(){}) */


        var div = document.querySelector("div");
        window.onresize=function(){
            console.log("当前页面可视区宽度为:" + window.innerWidth );
            if(window.innerWidth <= 800){// 当window.innerWidth小于等于800的时候就隐藏div
                div.style.display = "none";
            }else{
                div.style.display = "block";
            }
        }
        // window.onresize 是调整窗口大小加载事件， 当触发时就调用的处理函数。
        // 注意:
        // 只要窗口大小发生像素变化，就会触发这个事件。
        // 我们经常利用这个事件完成响应式布局。 window.innerWidth  获取当前屏幕的宽度
    </script>
</body>
</html>
```

延时器与定时器

window 对象给我们提供了 2 个非常好用的方法，延时器和定时器。

- 延时器 setTimeout()
- 定时器 setInterval()

延时器 了解

方法名	功能
window.setTimeout(code,MilliSec) 参数说明:code:要执行的代码 代码也可以是一个函数名 MilliSec:时间 单位是毫秒	**当指定的时间MilliSec到了以后就执行一次code 只会执行一次code**

语法:

```
window.setTimeout(调用函数,[延迟的毫秒数]);
setTimeout()这个调用函数我们也称为回调函数 callback
```

注意:

1. window可以省略
2. 这个**回调函数**可以直接写**匿名函数** 或者写 **函数名** 还可以采取字符串 **"函数名()"** 三种形式 第三种不推荐
3. 延迟的毫秒数省略默认是0,如果写,必须是毫秒
4. 因为延时器可能会有很多,所以setTimeout的返回值是一个**标识符**,一般我们会用一个变量把这个标识符保存起来,方便后期清除延时器使用

回调函数补充说明:

```
普通函数是按照代码顺序直接调用。

简单理解: 回调，就是回头调用的意思。上一件事干完，再回头再调用这个函数。
例如:定时器中的调用函数，事件处理函数，也是回调函数。

以前我们讲的 element.onclick = function(){} 或者 element.addEventLister
er("click", fn); 里面的 函数也是回调函数。
```

举例:

```
// 实现3秒以后 才会在浏览器中弹出一个hello
// 1.回调函数写的是匿名函数
// 2.回调函数写的是函数名
// 3.回调函数写的是字符串"函数名()"
```

案例:要求实现每隔3秒钟在浏览器控制台输出一个hello

```
延时器 默认只能执行一次code

我们可以在 code 中再次调用 setTimeout()  实现每隔多久时间调用一次函数
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 延时器一般是多久以后,只执行一次代码


        /*
        延时器语法 window.setTimeout(回调函数,时间 )
        延时器语法 window.setTimeout(第一个参数,第二个参数 )
        第一个参数可以放 匿名函数 或者 有名函数名 或者 "函数名()" 代码要执行的代码块
        第二个参数时间 单位是毫秒数  1000毫秒等于1秒


        window.setTimeout()中的window可以省略,写成setTimeout()
        */


        // 实现3秒以后 才会在浏览器中弹出一个hello
        // 1.回调函数写的是匿名函数
        window.setTimeout(function(){
            alert("hello");
        },3000);

        // 2.回调函数写的是函数名
        /* function fn(){
            alert("hello222222");
        }
        window.setTimeout(fn,3000); */

        // 3.回调函数写的是字符串"函数名()"  不推荐这样写
        /* function fn2(){
            alert("33333333333");
        }
        window.setTimeout( "fn2()", 3000); */
    </script>
</body>
</html>
```

清除延时器　了解

方法名	功能
window.clearTimeout(timer)参数说明: timer:要清除的延时器的返回值	清除指定的延时器

```
window.clearTimeout()方法用于取消先前通过调用setTimeout()建立的延时器
```

案例: 实现一个开始按钮点击以后可以每隔3秒在控制台输出一个hello,再实现一个停止按钮,点击停止输出

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button>开始</button>
    <button>停止</button>


    <script>
        // 实现一个开始按钮点击以后可以每隔3秒在控制台输出一个hello,再实现一个停止按钮,点击停止输出

        // 定义一个全局变量保存延时器的返回值
        var timer = null;


        var btns = document.querySelectorAll("button");
        // 实现一个开始按钮点击以后可以每隔3秒在控制台输出一个hello
        btns[0].onclick = function(){
            // console.log("hello");


            // 延时器到一定时间以后,只执行一次代码
            /* setTimeout(function(){
                console.log("hello");
            },3000) */


            // setTimeout延时器会返回一个值,记录当前延时器
            timer = setTimeout( sayHello ,3000 );
        }


        // 如果想要延时器不断执行,我们可以在函数里面再次调用延时器
        function sayHello(){
            console.log("hello");
            timer = setTimeout( sayHello ,3000 );
        }


        // 一个停止按钮,点击停止输出,其实就是清除延时器
        btns[1].onclick = function(){
            // window.clearTimeout(timer)参数说明: timer:要清除的延时器的返回值 清除指定的延时器
            window.clearTimeout( timer );
        }
    </script>
</body>
</html>
```

定时器 重点

方法名	功能

window.setInterval (code,MilliSec)参数说明：code：要执行的代码 代码可以是一个函数MilliSec：时间 毫秒	**每隔指定的时间就执行code 无限次执行code**

语法:

```
window.setInterval( 回调函数,[间隔的毫秒数] );
setInterval()方法会每隔这个毫秒数的时间,就去调用一次回调函数
```

注意:

1. window可以省略
2. 这个回调函数可以直接写**匿名函数** 或者写**函数名** 还可以采取**字符串"函数名()"** 三种形式 第三种不推荐
3. 延迟的毫秒数省略默认是0,如果写,必须是毫秒,表示每隔多少毫秒就自动调用这个函数
4. 因为定时器可能会有很多,所以setInterval的返回值是一个**标识符**,一般我们会用一个变量把这个标识符保存起来,方便后期清除定时器使用
5. 第一次执行也是间隔毫秒数之后执行,之后每隔毫秒数就执行一次

举例:使用setInterval这个方法来使用每隔3秒在控制台里面打印hello

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // 定时器是指每隔多久以后调用一次代码块,不断调用  而setTimeout是多久以后,只调用一次代码


        // 使用window.setInterval这个方法来使用每隔3秒在控制台里面打印hello


        // 语法
        // window.setInterval(回调函数,时间)
        // 1.回调函数可以是匿名函数
        window.setInterval(function(){
            console.log("hello");
        },3000);


        // 2.回调函数可以是有名函数名
        /* function fn(){
            console.log("hello");
        }
        window.setInterval( fn, 3000); */


        // 3.回调函数可以是字符串 "函数名()"   不推荐
        /* function fn2(){
            console.log("hi~");
        }
        window.setInterval( "fn2()", 3000); */
    </script>
</body>
</html>
```

清除定时器 重点

方法名	功能
window.clearInterval (timer)参数说明：timer：要清除定时器的返回值	清除指定的定时器

```
window.clearInterval()方法用于取消先前通过调用setInterval()建立的定时器
```

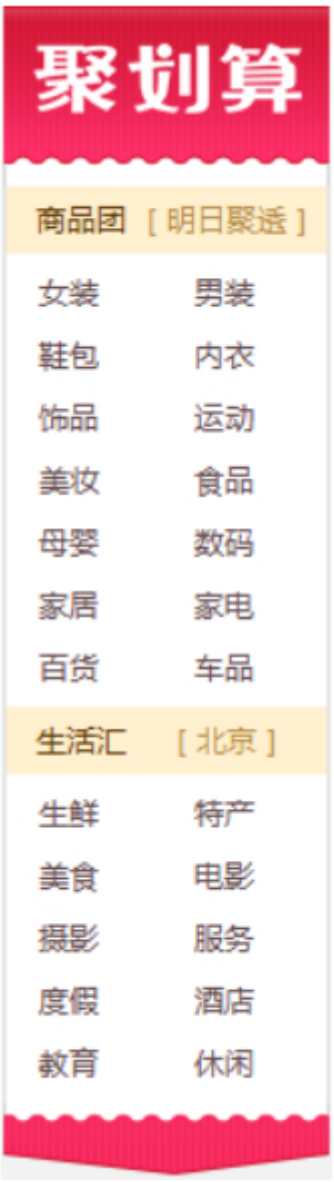
举例:使用setInterval这个方法来使用每隔3秒在控制台里面打印hello,接着实现一个停止按钮,点击以后清除定时器,停止输出hello

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button>开始</button>
    <button>停止</button>


    <script>
        // 实现一个开始按钮点击以后可以每隔3秒在控制台输出一个hello,再实现一个停止按钮,点击停止输出

        // 定义一个全局变量
        var timer = null;


        var btns = document.querySelectorAll("button");
        // 实现一个开始按钮点击以后可以每隔3秒在控制台输出一个hello
        btns[0].onclick = function(){
            // 定时器
            timer = setInterval(function(){
                console.log("hello")
            },3000);
        }


        // 停止按钮,清除定时器
        btns[1].onclick = function(){
            // window.clearInterval(timer) timer开启定时器的时候,得到的返回值
            window.clearInterval( timer );
        }
    </script>
</body>
</html>
```

小结:

```
什么时候情况下要用到延时器，什么情况下需要用到定时器？
只需要"执行一次"的我们就使用"延时器" 比如:头部广告几秒后消失
"无限次执行"的功能我们就"定时器"    比如:轮播图
定时器相对来说会用的比较多
```

案例:5秒后关闭广告



分析

```
1. 核心思路:5秒之后,就把这个广告隐藏起来
2. 使用延时器setTimeout
```

代码:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>


<body>
    <img src="images/ad.jpg" alt="" class="ad">
</body>
<script>
    // 获取对象
    var img = document.querySelector("img");
    // 开启延时器,5000毫秒以后隐藏img图片
    setTimeout(function(){
        img.style.display = "none";
    },5000)
</script>
</html>
```

案例:发送短信倒计时

要求:

```
1 点击按钮后,该按钮60秒之内不能再次点击,防止重复发送短信
```

分析:

```
1 1. 按钮点击之后,按钮会被禁用    设置button的disabled属性为true
2 2. 同时按钮里面的内容会变化, button标签里面的内容是通过innerHTML修改
3 3. 里面的秒数是有变化的,因此需要用到定时器
4 4. 定义一个变量,在定时器里面,不断递减
5 5. 如果变量为 0 说明时间到了 ,我们就需要停止定时器,并且复原按钮初始可以点击的状
态
```

代码:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
```

```
    <title>Document</title>
</head>


<body>
    手机号码:  <input type="text"> <button>获取验证码</button>
    <script>
        // 获取button按钮对象
        var btn = document.querySelector("button");
        // 定义秒数
        var seconds = 60;
        // 定义变量,保存定时器返回值
        var timer = null;


        btn.onclick=function(){
            alert("发送验证码成功");
            // 禁用按钮
            btn.disabled = true;
            // 开启定时器
            timer = setInterval(function(){
                // 注意:先判断清除定时器的情况,再执行正常的代码
                // 当秒数为0的时候,我们就清除定时器
                if( seconds == 0){
                    clearInterval( timer );
                    // 取消禁用状态
                    btn.disabled = false;
                    // 设置按钮内容
                    btn.innerHTML = "获取验证码";
                    // 设置秒数回60
                    seconds = 60;
                }else{
                    btn.innerHTML = seconds + "秒重新获取";
                    seconds--;
                }


            },1000)
        }
    </script>
</body>


</html>
```

this指向问题

this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，一般情况下this的最终指向的是那个调用它的对象。

现阶段，我们先了解一下几个this指向

1. 全局作用域或者普通函数中this指向全局对象window（注意延时器跟定时器里面的this指向window）
2. 对象里面的方法,谁调用, this就指向谁
3. 在事件处理 函数中,this指向绑定事件的那个对象
4. 构造函数中this指向构造函数的实例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
    // this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，一般情况下this的最终指向的是那个调用它的对象。
    // 现阶段，我们先了解一下几个this指向
    // 全局作用域或者普通函数中this指向全局对象window（注意延时器跟定时器里面的this指向window）
    console.log( this );
    console.log("");


    function fn(){
        console.log( this );
    }
    console.log( this );


    // 延时器里面指向window
    setTimeout(function(){
        console.log( this );
    },3000);


    // 定时器里面this也是指向window
    setInterval(function(){
        console.log( this );
    },3000);


    // 对象里面的方法,谁调用, this就指向谁
    // 构造函数中this指向构造函数的实例
    // 在事件处理 函数中,this指向绑定事件的那个对象
    </script>
</body>
</html>
```

location对象

什么是 location 对象

window对象给我们提供了一个location属性用于获取或设置窗体的URL网址,并且可以用于解析URL.因为这个属性返回的是一个对象,所以我们讲这个属性也称为location对象

URL介绍 了解

统一资源定位符(Uniform Resource Locaotr,URL) 是互联网上标准资源的地址.互联网上的每个文件都有一个唯一的URL,它包含的信息指出文件的位置以及浏

览器应该怎么处理它.

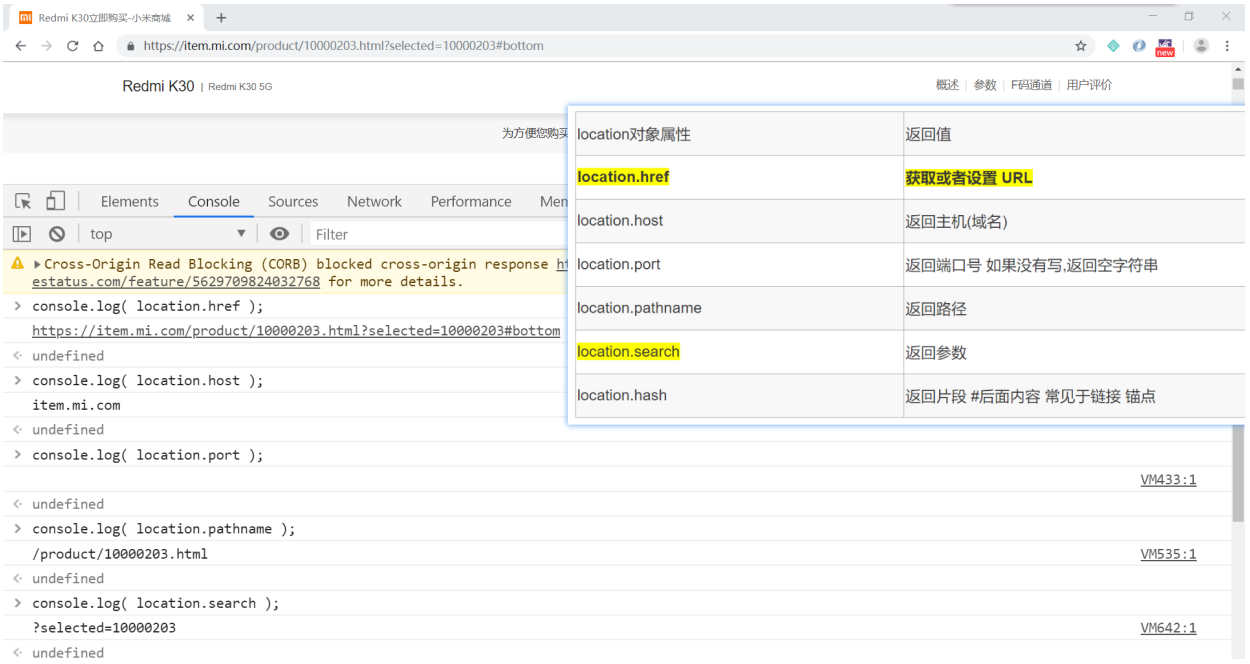
URL的一般语法格式如下:

```
protocol://host[:port]/path/[?query]#fragment

比如:
https://item.mi.com/product/10000203.html?selected=10000203#bottom
```

组成	说明
protocol	通信协议 常见有http,https,ftp等
host	主机(域名) 比如:www.mi.com
port	端口号 可选,省略时会使用默认端口,比如http的默认端口为80
path	路径 由零个或者多个"/"符号隔开的字符串,一般用来表示主机上的一个目录或者文件地址
query	参数 以键值对的形式 通过&符号分隔开 get请求提交数据就是这种方式
fragment	片段 #后面内容 常见于链接 锚点

location 对象的属性

location对象属性	返回值
location.href	**获取或者设置 URL**
location.host	返回主机(域名)
location.port	返回端口号 如果没有写,返回空字符串
location.pathname	返回路径
location.search	返回参数
location.hash	返回片段 #后面内容 常见于链接 锚点



location对象属性	返回值
location.href	获取或者设置 URL
location.host	返回主机(域名)
location.port	返回端口号 如果没有写,返回空字符串
location.pathname	返回路径
location.search	返回参数
location.hash	返回片段 #后面内容 常见于链接 锚点

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        div{
            width: 100px;
            height: 100px;
            background: pink;
        }
    </style>
</head>
<body>
    <button>按钮</button>
    <div>div的内容</div>


    <script>
        // 咱们以前实现网页的跳转,都需要通过a链接
        // 使用js实现点击按钮,跳转到百度页面
        var btn = document.querySelector("button");
        btn.onclick = function(){
            // location.href可以获取当前浏览器的url也可以设置当前浏览器url,url叫统一资源定位符,俗称网址

            // 获取url  但是url中的中文字符会进行url转码
            // console.log( location.href );


            // 设置url
            location.href = "https://www.baidu.com";
        }


        var div = document.querySelector("div");
        div.onclick = function(){
            location.href = "https://www.qq.com";
        }
    </script>
</body>
</html>
```

location对象的常见方法

location对象方法	返回值
location.assign(网址)	跟href一样,可以跳转页面(也称为重定向页面) **记录历史,可以实现后退功能**
location.replace(网址)	**替换当前页面,不记录浏览历史，所以不可以实现后退功能**
location.reload()	重新加载页面,相当于点了刷新,或者按了f5 如果参数为**true**,表示强制刷新ctrl+f5

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button>assign跳转页面,记录历史</button>
    <button>replace替换当前页面,不记录历史</button>
    <button>reload刷新页面</button>


    <script>
        var btns = document.querySelectorAll("button");
        // location对象的常见方法
        // location.assign(网址) 跟location.href一样,可以跳转页面(也称为重定向页面) 记录历史,可以实现后退功能   location.href是属性,location.assign()是方法
        btns[0].onclick = function(){
            location.assign("https://www.baidu.com");
        }
        // location.replace(网址) 替换当前页面,不记录浏览历史，所以不可以实现后退功能
        btns[1].onclick = function(){
            location.replace("https://www.qq.com");
        }
        // location.reload() 重新加载页面,相当于点了刷新,或者按了f5 如果参数为true,表示强制刷新ctrl+f5
        btns[2].onclick = function(){
            location.reload(true);
        }
    </script>
</body>
</html>
```

今日总结

xmind还是要做

今日作业

代码作业有8个,亿点都不多,加油,同学们~