传智播客·前端与移动开发学院
http://web.itcast.cn

前端基本功—javascript 第六天

目录

1.1

1.2 复习

1. 节点　　网页是有很多的节点组成的　。

元素节点　指的是 ：　标签　　li　span

文本节点　　属性节点

父子兄弟　　父　　parentNode　　nextSibling

孩子　　childNodes　　nodeType == 1　来判断 是否是 元素节点

最喜欢用的　children　　只得到　　元素节点

1.获取节点属性　　getAttribute（“title”）

2.设置节点属性　　setAttribute (“class”,”one”)

3.删除节点属性　　removeAttribute(“title”);

4. 日期函数　　Date();

　声明:　var　date = new Date();

　使用:　得到现在的年分　　date.getFullYear();

　　月份:　date.getMonth();

　　日子;　date.getDate();

　　星期:　　date.getDay();

5. 定时器

　　定时器　不需要人工操作　　按照一定的时间进行某种动作。

　setInterval(“函数”,间隔时间)　　每隔 n 秒去执行一次函数

1.3 时钟案例

分两步进行的。

第一步： 要得到现在的 时 分 秒

但是这里面有一个小玄机 。

比如现在是 9 点整 时针指向 9 是没错的

但是如果现在是 9 点半 时针应该指向的是 9 到 10 之间 才对

所以，我们不但要得到现在的小时 ，还要得到 已经过去了多少分

```
ms = date.getMilliseconds(); // 现在的毫秒数

s = date.getSeconds() + ms / 1000;  //  得到秒 1.3 s

m = date.getMinutes() + s / 60;  //  得到的是分数  45.6 分钟
h = date.getHours() % 12 + m / 60 ;
```

旋转角度原理

秒针 一秒 走多少度呢 ？

一圈 360 ° 一共有 60 秒 每秒 6 °

分针　　一圈　360　　一圈走 60 次　　每次　6°　　每分钟 6°

时针　　一圈 360　　一共 12 个 表盘没有 24 小时　　每个小时 走 30°

完整代码：

```
<script>
  var hour = document.getElementById("hour");
  var minute = document.getElementById("minute");
  var second = document.getElementById("second");
    // 开始定时器
   var s = 0,m = 0, h = 0, ms = 0;
    setInterval(function() {
          // 内容就可以了
        var date = new Date();  // 得到最新的时间
         ms = date.getMilliseconds(); // 现在的毫秒数
         s = date.getSeconds() + ms / 1000;  //  得到秒 1.3 s
         m = date.getMinutes() + s / 60;  //  得到的是分数  45.6 分钟
         h = date.getHours() % 12 + m / 60 ;
         // console.log(h);
         // 旋转角度
        // 一圈  360 °     一共有 60 秒        每秒  6 °   现在是 s 秒
         second.style.WebkitTransform = "rotate("+ s*6 +"deg)";
                          //  变化              旋转     deg  度
         minute.style.WebkitTransform = "rotate("+ m*6 +"deg)";
         hour.style.WebkitTransform = "rotate("+ h*30 +"deg)";
         second.style.MozTransform = "rotate("+ s*6 +"deg)";
```

```
22                              //  变化                  旋转      deg  度
23              minute.style.MozTransform = "rotate("+ m*6 +"deg)";
24              hour.style.MozTransform = "rotate("+ h*30 +"deg)";
25
26          },30);
27  </script>
```

1.4 按钮不可用

button　不可以用　　disabled　不可用的意思

btn.disabled = “disabled”　　||　　btn.disabled = true;

灰色的

注意：

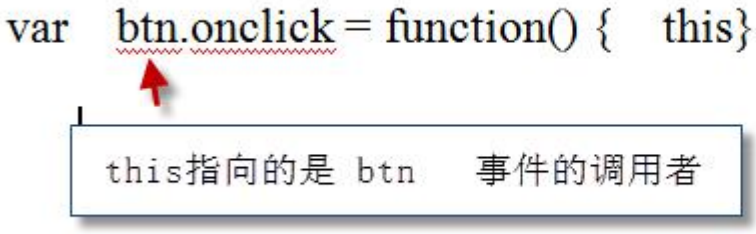
1. 因为 button 是个双标签　所以要更改他的值，　使用 innerHTML 的，不是 value。

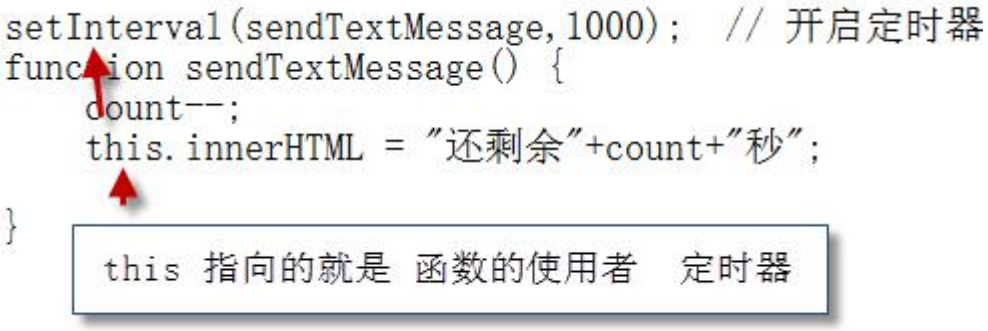
2. 关闭定时器　　clearInterval(定时器名称);　　定时器不再进行

1.5 this

this 指向的是 事件的调用者 ，或者是函数的使用者。

var btn.onclick = function() { this}





一般情况下，我们喜欢 var that = this;

```
var that = this;  // 把 btn 对象 给 that  var _this = this;
```

1.6 定时器之 setTimeout()

时间去哪儿了 类似于定时炸弹 。。

```
setTimeout(“函数”，时间 )
```

setInterval(fn,5000); 每隔 5 秒钟，就去执行函数 fn 一次

setTimeout(fn,5000); 5 秒钟之后，去执行 fn 函数， 只执行一次

1.6.1 深层次的看待定时器区别

setInterval 是排队执行的

假如 间隔时间是 1 秒， 而执行的程序的时间是 2 秒 上次还没执行完的代码会排队，上一次执行完下一次的就立即执行，这样实际执行的间隔时间为 2 秒

setTimeout 延迟时间为 1 秒执行，要执行的代码需要 2 秒来执行,那这段代码上一次与下一次的执行时间为 3 秒.

1.7 5 秒钟自动跳转页面

JS 页面跳转: window.location.href = ”http://www.itcast.cn” ; BOM

函数自己调用自己的过程 我们称之为 ： 递归调用 自残

```
var speed = 1000;
setTimeout(goIndexPage,speed);  //  1秒钟之后去执行  goIndexPage这个函
function goIndexPage() {
    count--;
    if(count < 0)
    {
        // 如果 count 小于 0 就到了时间了   我们应该跳转页面
         window.location.href = "http://www.baidu.com";
    }
    else
    {
        setTimeout(goIndexPage,speed);  //  递归调用  自己调用自己
    }
}
```

函数内部的定时器，又调用了本函数

但是这样用，一定要加一个退出 if 的条件，不然成为死循环了。

目的就是为了，模拟使用 settimeout 来实现 setinterval 的效果。

```
<script>
var demo = document.getElementById("demo");
var count = 5;
var speed = 1000;
setTimeout(goIndexPage,speed);  //  1 秒钟之后去执行  goIndexPage 这个函数
function goIndexPage() {
    count--;
    demo.innerHTML = "<a href='http://www.baidu.com'>本页面将在第"+count+"秒钟之后跳转页面</a>";
    if(count <= 0)
    {
        // 如果 count 小于 0 就到了时间了   我们应该跳转页面
         window.location.href = "http://www.baidu.com";
    }
    else
    {
        setTimeout(goIndexPage,speed);  //  递归调用  自己调用自己
    }
}
```

辞海 10 万字 2500 汉字 1000 次常用汉字

1.7.1　arguments 对象

function fn(a,b,c) {　console.log(a+b+c); alert(arguments.length;)}

fn(1,3,4,6);

arguments.length;　返回的是　实参的个数。

但是这个对象有讲究，他只在正在使用的函数内使用。

arguments.callee;

返回的是正在执行的函数。 也是在函数体内使用。 在使用函数递归调用时推荐使用 arguments.callee 代替函数名本身。

function fn() {　console.log(arguments.callee); }

这个 callee 就是 ：　function fn() {　console.log(arguments.callee); }

1.8　运算符

一元操作符 ++，　-- + -　　+5　-6

逻辑操作符 ! && ||

基本运算符 +, -, *, /, %

关系操作符 >, <, >=, <=, ===, ==, !=, !==

= 赋值　== 判断　=== 全等

条件操作符 （三元运算符） ? :

赋值运算符 +=, -=, *=, /=, %=

a+=5　　a= a + 5

逗号运算符 ,　　var　a=0,b=0;

1.8.1 运算符顺序

1　()

2　!、-、++、--　　(-10)　负号　正号

3 *、/、%

4 +、-　　10-5

5 <、<=、<、>=

6 ==、!=、===、!==、

7 &&

8 ||

9?:

10 =、+=、-=、*=、/=、%=　　赋值

1+2*3

1.8.2 几个面试题

1.　a&&b　　结果是什么？

如果 a 为假 ，则返回 a

如果 a 为真 ，则返回 b

var aa　=　0&&1;

alert(aa)　// 0

var bb =　1&&0;

alert(bb);　//0

var cc =　1&&10;

alert(cc);　// 10

2. a||b

如果 a 为假　则返回 b

如果 a 为真　则返回 a

```
console.log(0||1);   1
console.log(1||0);   1
console.log(1||5);   1
console.log(5||1);   5
```

var a = 1 && 2 && 3;

console.log(a);　3

var b = 0 && 1 && 2;

console.log(b);　0

var c = 1 && 0　&& 2;

console.log(c);　0

%=

a+=3

a = a % 3;

1.9 字符串对象常用方法

我们工作中经常进行字符串操作。

1.9.1 转换为字符串

1. + “” 2+ “” = “2” 2+”ab” = “2ab”

2. String() 转换为字符串

3. toString（基数） ; 基数就是进制

var txt = 10;

txt.toString(2) 二进制 1010

1.9.2 获取字符位置方法

charAt，获取相应位置字符（参数： 字符位置）

charCodeAt 获取相应位置字符 unicode 编码（参数： 字符位置）

var txt = “abcedf”;

比如， txt.charAt(4); 索引号一定是从 0 开始 返回的结果是 d

我们根据我们输入的 位数 返回相应的 字符 。

unicode 编码 是我们字符的字符的唯一表示 。

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		
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+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