# JavaScript基础入门

第10天课堂笔记（本课程共10天）

# 目录

[JavaScript基础入门 1](#_Toc476950339)

[目录 1](#_Toc476950340)

[一、异步和回调函数 2](#_Toc476950341)

[1.1 同步和异步 2](#_Toc476950342)

[1.2 回调函数 3](#_Toc476950343)

[二、apply和call语句初步 4](#_Toc476950344)

[三、缓冲 6](#_Toc476950345)

[四、setTimeout()和函数节流 8](#_Toc476950346)

[4.1 setTimeout()方法 8](#_Toc476950347)

[4.2 函数节流 8](#_Toc476950348)

[五、传统轮播和呼吸轮播复习 9](#_Toc476950349)

[5.1 传统轮播 9](#_Toc476950350)

[5.2 三位置传统轮播图 9](#_Toc476950351)

[5.3 呼吸轮播（交叉淡入淡出轮播） 11](#_Toc476950352)

[六、异形滚动 12](#_Toc476950353)

# 一、异步和回调函数

## 1.1 同步和异步

[JS同步编程方案思路](https://github.com/Gaubee/blog/issues/71) [JavaScript异步编程的4种方法阮一峰](http://www.ruanyifeng.com/blog/2012/12/asynchronous%EF%BC%BFjavascript.html) [同步异步Segmentfault](https://segmentfault.com/a/1190000004322358)

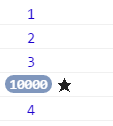
程序从上到下执行：

|  |
| --- |
| 1. console.log(1); 2. console.log(2); 3. console.log(3); 4. console.log(4); 5. console.log(5); |

同步synchronous

假如程序中有for循环，非常耗费时间，但是系统会用“同步”的方式运行：

|  |
| --- |
| 1. console.log(1); 2. console.log(2); 3. console.log(3); 4. for (var i = 0; i < 10000; i++) { 5. console.log("★"); 6. } 7. console.log(4); |



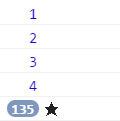
“同步”的意思：for循环很耗费时间，但是程序就是傻等，傻傻的等待10000个星星输出，然后输出4。

比如妈妈去接儿子的飞机，需要等很长时间，等待的时候就是傻等，不同时做别的事情。

异步Asynchronous

|  |
| --- |
| 1. console.log(1); 2. console.log(2); 3. console.log(3); 4. setInterval(function(){ 5. console.log("★"); 6. },1000); 7. console.log(4); |

输出4提前执行了，然后输出五角星



“异步”的意思：遇见了一个特别耗费时间的事情，程序不会傻等，而是先执行后面的语句。

比如妈妈去接儿子的飞机，需要等很长时间，但是妈妈同时逛逛商店什么的，不是傻等。

JS中的异步，需要异步语句：setInterval、setTimeout、Ajax、Node.js……等等

如果有异步语句了，那么一定是异步的。如果没有异步语句，那就不是异步的。

## 1.2 回调函数

[回调函数1](http://wiki.jikexueyuan.com/project/brief-talk-js/callback-function.html) [回调函数2](http://www.html-js.com/article/Sexy-Javascript-understand-the-callback-function-with-the-use-of-Javascript-in) [回调函数3](https://cnodejs.org/topic/564dd2881ba2ef107f854e0b) [回调函数4](http://blog.csdn.net/luoweifu/article/details/41466537)

异步的事情做完了，我们想继续做什么事儿，那此时怎么办呢？

回调函数： 异步的语句做完之后要做的事情

|  |
| --- |
| 1. var count = 0; 2. var timer = setInterval(function(){ 3. console.log("★"); 4. count++; 5. if(count == 300){ 6. clearInterval(timer); 7. **callback();** 8. } 9. },20); 10. **//回调函数** 11. **function callback(){** 12. **alert("全部星星输出完毕");** 13. **}** |

# 二、apply和call语句初步

[call](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/call) [apply](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) [bind](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/bind)

我们试图在回调函数中，用this表示oDiv对象，这样感觉爽。

|  |
| --- |
| 1. animate(oDiv,{"left":600},2000,function(){ 2. this.style.backgroundColor = "red"; 3. }); |

但是不行，回调函数中this不是oDiv。

所以我们现在要想一个办法，让callback运行，并且callback里面的this是oDiv。

|  |
| --- |
| 1. callback.call(elem); |

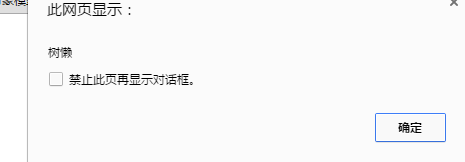
或者：

|  |
| --- |
| 1. callback.apply(elem); |

执行callback函数，并且让callback函数中的this关键字为elem。

例子：

|  |
| --- |
| 1. var obj2 = { 2. "name" : "树懒", 3. "age" : 16, 4. "sex" : "男" 5. } 6. function xianshixinxi(){ 7. alert(this.name); 8. } 9. //下面这个call语句的含义有两层： 10. //1. xianshixinxi函数将被调用 11. //2. 同时这个函数内部的this就是obj了 12. **xianshixinxi.call(obj2);** |



说白了apply、call功能是一样的，功能都是：让函数调用，并且给函数设置this是谁。

call就是英语呼叫的意思，apply是应用的意思。

现在说说区别：

|  |
| --- |
| 1. var obj = { 2. "name" : "考拉", 3. "age" : 18, 4. "jiehun" : false 5. } 6. function xianshixinxi(a,b,c){ 7. console.log(this.name); 8. console.log(a + b + c); 9. } 10. xianshixinxi.call(obj,2,3,4); 11. xianshixinxi.apply(obj,[2,3,4]); |

call和apply功能一样，就是让一个函数里面的this设置为一个对象。

拿上例来说，xianshixinxi函数里面的this就是obj对象。但是我们发现，xianshixinxi这个函数，有参数。那么我们现在又想设置this是谁，又想把参数往里传，此时就有区别了：

|  |
| --- |
| 1. fn.call(obj,参数1,参数2,参数3……); |

|  |
| --- |
| 1. fn.apply(obj,[参数1,参数2,参数3……]); |

call需要你用逗号罗列所有参数，但是apply是把所有参数写在数组里面。即使只有一个参数，也必须写在数组里面。

比如：

|  |
| --- |
| 1. function showInfo(n){ 2. for(var i = 1 ; i <= n ; i++){ 3. console.log(this.name); 4. } 5. } 6. showInfo.call(obj,**10**); 7. showInfo.apply(obj,**[10]**); |

一定不要记混，apply要的是数组，你想啊，ly、ly、ly长得像list，就是数组。

总结一下，这两个东西功能一致，就是把一个函数里面的this设置为某个对象。区别就是后面的参数的语法。

# 三、缓冲

一个盒子用3000毫秒时间，从100→700，不一定是匀速的。

时间精确、移动的变化量也精确，但是不一定是匀速的。

想象一下小时候升国旗，国歌55秒，旗杆10m。小孩子总能55秒准确的升到10m顶端，但是，你懂得。

到底是匀速的、还是先快后慢、先慢后快呢？这就是缓冲，英语叫做tween。

缓冲的实现，非常简单，就是我们的数学家给我们提供了非常多的缓冲算法，都很好用：

|  |
| --- |
| 1. function linear(t , b , c , d){ 2. return c \* t / d + b; 3. } 4. function easeIn(t,b,c,d){ 5. return c \* ( t /= d) \* t + b; 6. } 7. function easeOut(t,b,c,d){ 8. return -c \*(t/=d)\*(t-2) + b; 9. } |

他们的参数，都是t、b、c、d：

第一个参数t表示当前帧编号

第二个参数b表示起始位置

第三个参数c表示变化量

第四个参数d表示总帧数

函数的返回值，就是t这一帧，元素应该在的位置。

[Tween算法](http://www.cnblogs.com/bluedream2009/archive/2010/06/19/1760909.html)

函数的命名，需要会：

Quad二次的

Cubic三次的

Quart四次的

Quint五次的

随着次数的增加，曲线变陡：

|  |  |  |  |
| --- | --- | --- | --- |
| 2次 | 3次 | 4次 | 5次 |

后面跟着小词儿：easeIn、easeOut、easeInOut分别表示：

|  |  |  |
| --- | --- | --- |
| easeIn | easeOut | easeInOut |

将缓冲功能封装在我们的运动框架中，比如：

|  |
| --- |
| 1. aimate(oDiv,{“left”:600},3000,**”bounceEaseOut”**,function(){ 2. }); |

# 四、setTimeout()和函数节流

## 4.1 setTimeout()方法

[setTimeout](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/setTimeout) [clearTimeout](https://developer.mozilla.org/zh-CN/docs/Web/API/WindowTimers/clearTimeout)

setInterval是设置间隔器；

setTimeout是设置延时器。

|  |
| --- |
| 1. window.setTimeout(函数,时间); |

在指定时间之后，执行函数一次，仅仅执行1次。

同样的，它也是window对象的方法，可以不写window



在3000毫秒之后，控制台显示你好，仅仅显示1次：

|  |
| --- |
| 1. setTimeout(function(){ 2. console.log("你好"); 3. }, 3000); |

延时器也能被清除，当延时器没有执行的时候，就可以清除，清除不会触发函数。

|  |
| --- |
| 1. clearTimeout(); |

## 4.2 函数节流

[函数节流 前端社区](http://imweb.io/topic/577aa790ea7bb9b760c7adc3) [函数节流搜索场景](http://web.jobbole.com/88733/) [函数节流](http://wiki.jikexueyuan.com/project/brief-talk-js/function-throttling.html)

所谓的函数节流，就是我们希望一些函数不要连续的触发。甚至于规定，触发这个函数的最小间隔是多少时间。

这个就是函数节流。

方法1：

经典的函数节流模型：

|  |
| --- |
| 1. var lock = true; 2. input.onclick = function(){ 3. **if(!lock) return;** 4. **lock = false;** 5. **setTimeout(function(){** 6. **lock = true;** 7. **},1000);** 8. } |

方法2：

改变我们的运动框架，在运动框架里面加上一个逻辑：运动开始了，就给elem加上一个属性isanimated，表示是否在运动，改为true。然后运动停止之后，停表之后，把elem.isanimated设为false

|  |
| --- |
| 1. .onclick = function(){ 2. if(m\_unit.isanimate) return; //如果点击按钮的时候运动机构在动，那么return 3. } |

# 五、传统轮播和呼吸轮播复习

## 5.1 传统轮播

是一个a标签，但是我们不希望点击之后有任何默认的事情，默认事情通常是刷新当前页：

|  |
| --- |
| 1. <a href="javascript:;" class="leftBtn" id="leftBtn"></a> |

最最关键的就是右按钮、左按钮的策略：

右按钮，处理最后一张的时候，先拉后瞬移：

|  |
| --- |
| 1. rightBtn.onclick = function(){ 2. idx ++; 3. animate(m\_unit, {.......},500,function(){ 4. if(idx > 4){ 5. idx = 0; 6. m\_unit.style.left = “0px”; 7. } 8. }) 9. } |

左按钮，处理第一张的时候，先瞬移后拉动：

|  |
| --- |
| 1. leftBtn.onclick = function(){ 2. idx --; 3. if(idx < 0 ){ 4. idx = 4; 5. m\_unit.style.left = -560 \* 5 + "px"; 6. } 7. animate(m\_unit, {.....},500) 8. } |

## 5.2 三位置传统轮播图

页面效果，随着人们的认识深入，就产生更多的原理。

传统轮播的“火车法”的问题是，小圆点点击的时候，拉动的图片，跳跃性大。

比如现在是0号图，点击3号小圆点，就会看见1号、2号，停留在3号图。不是平滑过渡。

三位置传统轮播图，特点是，所有的li不是火车，而是都绝对定位。他们的left值只有三种可能：

-560 0 560



## 5.3 呼吸轮播（交叉淡入淡出轮播）

摞在一起，不管点击哪个按钮，老图淡出，新图淡入。点击小圆点，老图淡出，新图淡入。

我们讲一个bug，就是元素opacity:0了，也能够被点击。

解决办法就是，页面上有且又有一个li的display:block；

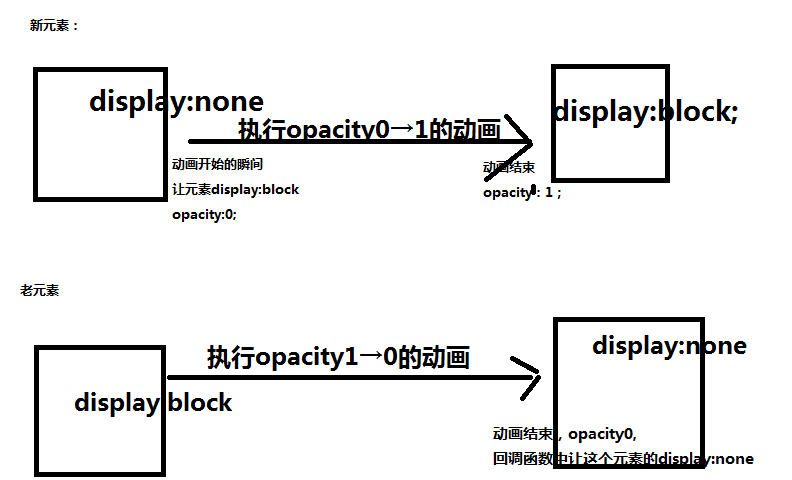
当点击右按钮的时候：

当前这个图片执行opacity1→0的动画，动画结束之后，让这个元素display:none;

新图片执行opacity0→1的动画，动画开始之前，瞬间让新图片display:block;然后再动画

一定要牢记：呼吸轮播动画的起点是display:block； 动画的终点是display:none;

但是JS是没法过渡display属性的，所以我们就往中间加上了opacity



# 六、异形滚动

每次老师吹哨的时候：

小明擦玻璃

小红拖地

小刚扫地

让小明改名叫小红

让小红改名叫小刚

让小刚改名叫小明

所以老师每次吹哨，这三个学生依次做不同的劳动。

先运动，然后改变身份。

lisArr = [lis.stateA,lis.stateB,lis.stateC,lis.stateD……]

点击一下按钮，我们永远让数组的第0项瞬移到6号位置，第1项动画变为0，第2项动画变为1……

lisArr.push(lisArr.shifi());

[lis.stateB,lis.stateC,lis.stateD……lis.stateA]