学习目标：

> 能够说出两种定时器的区别

1.2.5. 定时器（两种）

window 对象给我们提供了 2 个非常好用的方法-定时器。

- setTimeout()
- setInterval()

setTimeout() 炸弹定时器

开启定时器

```
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：

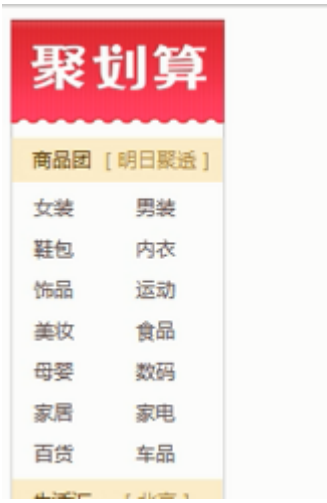
1. window 可以省略。
2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串 '函数名()'三种形式。第三种不推荐
3. 延迟的毫秒数省略默认是 0，如果写，必须是毫秒。
4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

> 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.addEventListener("click", fn); 里面的 函数也是回调函数。

```
    <script>
        // 回调函数是一个匿名函数
         setTimeout(function() {
             console.log('时间到了');

         }, 2000);
        function callback() {
            console.log('爆炸了');
        }
        // 回调函数是一个有名函数
        var timer1 = setTimeout(callback, 3000);
        var timer2 = setTimeout(callback, 5000);
    </script>
```

案例：5秒后关闭广告



案例分析

① 核心思路：5秒之后，就把这个广告隐藏起来

② 用定时器setTimeout

```
<body>
    <img src="images/ad.jpg" alt="" class="ad">
    <script>
        // 获取要操作的元素
        var ad = document.querySelector('.ad');
        // 开启定时器
        setTimeout(function() {
            ad.style.display = 'none';
        }, 5000);
    </script>
</body>
```

停止定时器

```
window.clearTimeout(timeoutID)
```

clearTimeout()方法取消了先前通过调用 setTimeout() 建立的定时器。

注意:

1. window 可以省略。

2. 里面的参数就是定时器的标识符。

```
    <button>点击停止定时器</button>
    <script>
        var btn = document.querySelector('button');
        // 开启定时器
        var timer = setTimeout(function() {
            console.log('爆炸了');
        }, 5000);
        // 给按钮注册单击事件
        btn.addEventListener('click', function() {
            // 停止定时器
            clearTimeout(timer);
        })
    </script>
```

setInterval() 闹钟定时器

开启定时器

```
window.setInterval(回调函数, [间隔的毫秒数]);
```

setInterval() 方法重复调用一个函数，每隔这个时间，就去调用一次回调函数。

注意:

1. window 可以省略。

2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串 '函数名()' 三种形式。

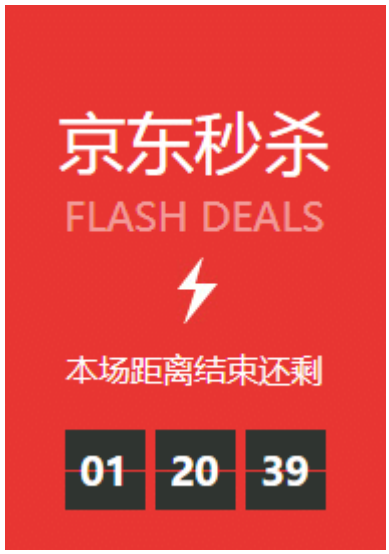
3. 间隔的毫秒数省略默认是 0，如果写，必须是毫秒，表示每隔多少毫秒就自动调用这个函数。

4.因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

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。

```
<script>
    // 1. setInterval
    setInterval(function() {
        console.log('继续输出');
    }, 1000);
</script>
```

案例：倒计时



案例分析

① 这个倒计时是不断变化的，因此需要定时器来自动变化（setInterval）
② 三个黑色盒子里面分别存放时分秒
③ 三个黑色盒子利用innerHTML 放入计算的小时分钟秒数
④ 第一次执行也是间隔毫秒数，因此刚刷新页面会有空白
⑤ 最好采取封装函数的方式，这样可以先调用一次这个函数，防止刚开始刷新页面有空白问题

```
<div>
    <span class="hour">1</span>
    <span class="minute">2</span>
    <span class="second">3</span>
</div>
<script>
    // 1. 获取元素（时分秒盒子）
    var hour = document.querySelector('.hour'); // 小时的黑色盒子
    var minute = document.querySelector('.minute'); // 分钟的黑色盒子
    var second = document.querySelector('.second'); // 秒数的黑色盒子
    var inputTime = +new Date('2019-5-1 18:00:00'); // 返回的是用户输入时间总的毫秒数

    countDown(); // 我们先调用一次这个函数，防止第一次刷新页面有空白

    // 2. 开启定时器
    setInterval(countDown, 1000);
```

```
        function countDown() {
            var nowTime = +new Date(); // 返回的是当前时间总的毫秒数
            var times = (inputTime - nowTime) / 1000; // times是剩余时间总的秒数
            var h = parseInt(times / 60 / 60 % 24); //时
            h = h < 10 ? '0' + h : h;
            hour.innerHTML = h; // 把剩余的小时给 小时黑色盒子
            var m = parseInt(times / 60 % 60); // 分
            m = m < 10 ? '0' + m : m;
            minute.innerHTML = m;
            var s = parseInt(times % 60); // 当前的秒
            s = s < 10 ? '0' + s : s;
            second.innerHTML = s;
        }
    </script>
```

停止定时器

```
window.clearInterval(intervalID);
```

clearInterval()方法取消了先前通过调用 setInterval()建立的定时器。

注意：

1. window 可以省略。

2. 里面的参数就是定时器的标识符。

案例：发送短信倒计时

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。

案例分析

① 按钮点击之后，会禁用 disabled 为true

② 同时按钮里面的内容会变化， 注意 button 里面的内容通过 innerHTML修改

③ 里面秒数是有变化的，因此需要用到定时器

④ 定义一个变量，在定时器里面，不断递减

⑤ 如果变量为0 说明到了时间，我们需要停止定时器，并且复原按钮初始状态。

```
    手机号码： <input type="number"> <button>发送</button>
    <script>
        var btn = document.querySelector('button');
        // 全局变量，定义剩下的秒数
        var time = 3;
        // 注册单击事件
```

```
        btn.addEventListener('click', function() {
            // 禁用按钮
            btn.disabled = true;
            // 开启定时器
            var timer = setInterval(function() {
                // 判断剩余秒数
                if (time == 0) {
                    // 清除定时器和复原按钮
                    clearInterval(timer);
                    btn.disabled = false;
                    btn.innerHTML = '发送';
                } else {
                    btn.innerHTML = '还剩下' + time + '秒';
                    time--;
                }
            }, 1000);
        });
    </script>
```

1.2.6. 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（注意定时器里面的this指向window）
2. 方法调用中谁调用this指向谁
3. 构造函数中this指向构造函数的实例

```
    <button>点击</button>
    <script>
        // this 指向问题 一般情况下this的最终指向的是那个调用它的对象
        // 1. 全局作用域或者普通函数中this指向全局对象window（ 注意定时器里面的this指向window）
        console.log(this);
        function fn() {
            console.log(this);
        }
        window.fn();
        window.setTimeout(function() {
            console.log(this);
        }, 1000);
        // 2. 方法调用中谁调用this指向谁
        var o = {
            sayHi: function() {
                console.log(this); // this指向的是 o 这个对象
            }
        }
        o.sayHi();
        var btn = document.querySelector('button');
        btn.addEventListener('click', function() {
```

```
            console.log(this); // 事件处理函数中的this指向的是btn这个按钮对象
        })
    // 3. 构造函数中this指向构造函数的实例
    function Fun() {
        console.log(this); // this 指向的是fun 实例对象
    }
    var fun = new Fun();
</script>
```