目录

作业讲评

```
<!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>
        // 写一个函数 生成一个1~10之间的随机整数，每次调用函数都返回第一次生成的随机数
        function getRandom(){
            // 生成1~10之间的随机整数
            var num = Math.floor( Math.random()*10+1);

            // 使用闭包,延迟num变量的使用范围
            return function(){
                console.log( num );
            }
        }
        var f = getRandom();
        f();
        f();
        f();
        f();
        f();
        f();
    </script>
</body>
</html>
```

```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>点击按钮,段落内容字体大小变20px</button>
    <button>点击按钮,段落内容字体大小变40px</button>
    <button>点击按钮,段落内容字体大小变60px</button>
    <p>我是段落内容</p>


    <script>
        // 点击按钮改变页面上的文字大小
        var btns = document.querySelectorAll("button");
        var p = document.querySelector("p");


        // 方式一:使用自定义属性
        /* for(var i = 0;i < btns.length; i++){
            btns[i].setAttribute("data-size", (i+1)*20   )


            btns[i].onclick = function(){
                var size = this.getAttribute("data-size");
                p.style.fontSize = size + "px";
            }
        } */


        // 方式二:使用闭包
        for(var i = 0;i < btns.length; i++){
            // 借助立即调用函数
            (function(index){
                // 再立即调用函数里面给每个按钮绑定鼠标单击事件
                btns[i].onclick = function(){
                    // 按钮点击的事件函数中使用的index变量就是立即调用函数的index变量 ,立即调用函数是小闭包
                    // console.log( index );
                    p.style.fontSize = (index+1)*20 + "px";
                }
            })(i)
        }
    </script>
</body>
</html>
```

```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>赞(1)</button>
    <button>赞(1)</button>
    <button>赞(1)</button>
    <button>赞(1)</button>


    <script>
        // 获取所有按钮对象
        var btns = document.querySelectorAll("button");


        // 定义闭包函数
        function zan(){
            var num = 1;
            return function(){
                num++;
                // console.log( this ); 因为这个zan函数后面会给每个按钮使用,所以点击了哪个按钮,这个this就代表哪个按钮
                this.innerHTML = "赞("+num+")";
            }
        }


        // 定义点赞数初始化的值
        for(var i=0;i<btns.length;i++){
            // btns[i].onclick = zan;  // 无效


            // 返回值是调用函数以后,return后面返回内容
            btns[i].onclick = zan();
        }
    </script>
</body>
</html>
```

学习目标

- 能够说出闭包的作用
- 能够使用递归
- 能够说出深拷贝和浅拷贝的区别

立即调用函数传递参数问题

```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>
        (function(a , b){
            // 其中 a跟b是形参
            console.log( a );
            console.log( b );
            console.log( a+b );
        })(10,20);
        // 立即调用函数是在后面的那个括号传递实参的
    </script>
</body>
</html>
```

闭包应用

案例: 3秒钟之后,打印所有li元素的标签内容

```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 class="fruit">
        <li>苹果</li>
        <li>香蕉</li>
        <li>葡萄</li>
        <li>火龙果</li>
        <li>桔子</li>
    </ul>
    <script>
        // 3秒钟之后,打印所有li元素的标签内容
        // 获取所有li标签
        var lis = document.querySelectorAll("li");



        // 第一种方法:用延时器包裹for循环
        /* setTimeout(function(){
            for(var i=0;i<lis.length;i++){
                console.log( lis[i].innerHTML );
            }
        },3000) */


        // 第二个方法:在for循环里面使用延时器  也是使用闭包
        for(var i=0;i<lis.length;i++){
            // 用一个立即函数包裹延时器
            (function(index){
                setTimeout(function(){
                    console.log( lis[index].innerHTML );
                },3000)
            })(i)
        }
    </script>
</body>
</html>
```

案例: 闭包应用-计算打车价格

> 打车起步价13(3公里内), 之后每多一公里增加 5块钱. 用户输入公里数就可以计算打车价格 如果有拥堵情况,总价格多收取10块钱拥堵费

```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>
        // 闭包应用-计算打车价格
        // 打车起步价13(3公里内), 之后每多一公里增加 5块钱. 用户输入公里数就可以计算打车价格    如果有拥堵情况,总价格多收取10块钱拥堵费

        var cars = (function(){
            // 起步价
            var start = 13;
            // 总价
            var total = 0;


            // 返回了一个对象
            return {
                price:function(num){// num表示用户输入的公里数
                    if(num <= 3){
                        // 3公里内,总价就是起步价
                        total = start;
                        return total;
                    }else{
                        // 打车起步价13(3公里内), 之后每多一公里增加 5块钱
                        total = start + (num - 3)*5;
                        return total;
                    }
                },
                yd:function(flag){// flag的值如果是true就表示拥堵,需要额外加10块拥堵费
                    if( flag ){
                        total = total + 10;
                        return total;
                    }else{
                        return total;
                    }
                }
            }
        })();


        console.log( cars );// cars是一个对象,因为cars的值是一个立即调用函数,函数里面返回了一个对象
        console.log( cars.price(1) );// 13
        console.log( cars.price(2) );// 13
        console.log( cars.price(3) );// 13
        console.log( cars.price(5) );// 23
        console.log( cars.yd(true) );// 33
    </script>
</body>
</html>
```

闭包思考题

```js
// 思考题1:
var name = "The Window";
var object = {
    name: "My Object",
    getNameFunc: function() {
        return function() {
            return this.name;
        };
    }
};
console.log(object.getNameFunc()());
```

```
// 思考题2:
var name = "The Window";
var object = {
   name: "My Object",
   getNameFunc: function() {
       var that = this;
       return function() {
           return that.name;
       };
   }
};
console.log(object.getNameFunc()());
```

```
// 思考题3:
console.log('start');
setTimeout(function () {
 console.log('你猜我输出在哪');
}, 0);
console.log('end');
for (var i = 0; i < 10; i++) {
 console.log(i);
}
```

```
// 思考题4:
console.log('start');
for (var i = 0; i < 3; i++) {
 (function (index) {
    setTimeout(function () {
      console.log(index);
   }, 0);
 })(i);
}
console.log('end');
```

闭包总结

闭包是什么?

闭包是一个函数(一个作用域可以访问另外一个函数中的局部变量)

闭包的作用是什么?

延伸了变量的作用范围

闭包其他相关资料 可以上百度自行查询

比如: https://blog.csdn.net/weixin_43586120/article/details/89456183

递归

什么是递归

递归: 如果一个函数在内部可以调用其本身，那么这个函数就是递归函数。

简单理解:**函数内部自己调用自己, 这个函数就是递归函数**

注意: 递归函数的作用和循环效果一样，由于递归很容易发生“栈溢出”错误（stack overflow），所以必须要加退出条件return。

举例:

```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>
        // 递归：如果一个函数在内部可以调用其本身，那么这个函数就是递归函数。
        // 递归就是函数在函数体中再次调用自身
        // 递归容易发生栈溢出,因为每次调用函数都需要开启一个空间,最后空间不够用的时候,就会发现栈溢出  可以使用return关键字来结束递归

        // 目标:利用递归输出6句hello


        // 定义一个变量保存现在输出了多少次
        var count = 1;
        function fn(){
            if( count > 6 ){
                // 终止递归 return可以结束函数的执行
                return ;
            }
            console.log( "hello" );
            // 输出一次hello,就让count变量自加1一次
            count++;


            fn();
        }


        // 分析
        // 第一次输出hello以后  count = 2
        // 第二次输出hello以后  count = 3
        // 第三次输出hello以后  count = 4
        // 第四次输出hello以后  count = 5
        // 第五次输出hello以后  count = 6
        // 第六次输出hello以后  count = 7


        fn();
    </script>
</body>
</html>
```

递归的目的

小明的妈妈要出差 在家留下了一块方形蛋糕 每天只能吃3/4 的蛋糕

小明每天都吃掉3/4的蛋糕,到了第3天撑不住了
打电话问老妈,我是不是你捡来的 老妈说,你只要听话,每天都有蛋糕吃

小明每天做事情的步骤都一样 把蛋糕四等分 吃掉其中三份 但是蛋糕越来越小了

递归是用于解决特殊的问题,这类问题需要具备以下的特点 大问题可以拆分成小问题
小问题可以继续拆分为小小问题 无论问题规模的大小 解决方式都一样

递归的用法

递归输出10~1

第一步:封装一个能解决单一问题的函数(类似循环体)

第二步:在这个函数中调用当前函数 调整问题的规模(类似循环中的变量更新)

第三步:在函数中设置递归出口 如果没有递归出口 递归调用无法结束(类似循环中的判断条件)

```
<!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>
        // 递归输出10~1
        // 第一步:封装一个能解决单一问题的函数(类似循环体)


        /* var num = 10;
        function fn(){
            console.log( num );
        }
        fn(); */


        // 第二步:在这个函数中调用当前函数 调整问题的规模(类似循环中的变量更新)
        var num = 10;
        function fn(){
            // 第三步:在函数中设置递归出口 如果没有递归出口 递归调用无法结束(类似循环中的判断条件)
            if(num == 0){
                return ;
            }


            console.log( num );
            // 调整规模
            num--;
            fn();
        }


        fn();
    </script>
</body>
</html>
```

利用递归求1~n的阶乘

利用递归函数求1~n的阶乘 1 * 2 * 3 * 4...n

```
<script>
    // 利用递归函数求1~n的阶乘 1 * 2 * 3 * 4...n
    // 比如n = 5    1*2*3*4*5     5*4*3*2*1
    // 比如n = 4    1*2*3*4       4*3*2*1
    // 比如n = 3    1*2*3         3*2*1
    // ...


    function fn(num){
        if( num == 1 ){
            return 1;
        }
        return num*fn(num-1);
    }
    console.log( fn( 2 ) );
    console.log( fn( 5 ) );
</script>
```

利用递归求斐波那契数列

```
// 利用递归函数求斐波那契数列(兔子序列) 1、1、2、3、5、8、13、21...
// 我们只需要知道用户输入的n 的前面两项(n-1)和 (n-2)就可以计算出n 对应的序列值
// 用户输入一个数字 n 就可以求出 这个数字对应的兔子序列值
```

```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>
        // 利用递归求斐波那契数列
        // 利用递归函数求斐波那契数列(兔子序列) 1、1、2、3、5、8、13、21...
        // 我们只需要知道用户输入的n 的前面两项(n-1)和 (n-2)就可以计算出n 对应的序列值
        // 用户输入一个数字 n 就可以求出 这个数字对应的兔子序列值

        function fb( n ){
            if( n == 1 || n == 2){
                return 1;
            }

            return fb(n-1)+fb(n-2);
            /* return fb(6-1)+fb(6-2);
            return fb(5)+fb(4);
            return fb(4)+fb(3) + fb(4);
            return fb(3)+fb(2)+fb(3) + fb(4);
            return fb(2)+fb(1)+fb(2)+fb(3) + fb(4);
            return fb(2)+fb(1)+fb(2)+fb(2)+fb(1) + fb(4);
            return fb(2)+fb(1)+fb(2)+fb(2)+fb(1) + fb(3)+fb(2);
            return fb(2)+fb(1)+fb(2)+fb(2)+fb(1) + fb(2)+fb(1)+fb(2);
            return 8; */
        }
        console.log( fb(1) );
        console.log( fb(2) );
        console.log( fb(6) );
    </script>
</body>
</html>
```

利用递归遍历数据

准备数据:

```
var data = [{
 id: 1,
 name: '家电',
 goods: [{
 id: 11,
 gname: '冰箱',
 goods: [{
 id: 111,
 gname: '海尔'
  }, {
```

```
    id: 112,
    gname: '美的'
    },

    ]

    }, {
    id: 12,
    gname: '洗衣机'
    }]
}, {
    id: 2,
    name: '服饰'
}];
```

我们想要做输入id号,就可以输出查找到的数据对象

```
<!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>
        // 我们想要做输入id号, 就可以输出查找到的数据对象
        var data = [{
            id: 1,
            name: '家电',
            goods: [{
                id: 11,
                gname: '冰箱',
                goods: [{
                        id: 111,
                        gname: '海尔'
                    }, {
                        id: 112,
                        gname: '美的'
                    },
                ]


            }, {
```

```
            id: 12,
            gname: '洗衣机'
        }]
    }, {
        id: 2,
        name: '服饰'
    }];


    // console.log( data );


    // 遍历数组可以使用forEach
    /* data.forEach(function( item ){
        // item就是数组中每个元素
        console.log( item );
    }) */

    // console.log("");


    // 封装一个函数,根据用户传入的id号,输出找到的对象,json参数代表需要查询的数据
    function getId(json , id ){
        json.forEach(function( item ){
            // item就是数组中每个元素
            // console.log( item );


            // 如果想得到对象中属性的值 通过 对象.属性名 或者 对象[属性名]
            // console.log( item.name );
            // console.log( "" );


            if( id == item.id ){// 判断用户传入的id号是不是等于item中的id属性值
                console.log( item );
            }else{// 如果外层找不到想要的,我们可以尝试去内层找

                // 如果item里面存着goods数组 并且goods数组的元素还需要大于0
                if( item.goods && item.goods.length > 0){
                    // 递归调用getId函数
                    getId(  item.goods  ,  id );
                }

            }
        })
    }

    // console.log( data );


    // 这题就是使用递归的原理,一层一层循环对应id号的对象
    getId(data,1);
    getId(data,2);
    getId(data,11);
    getId(data,12);
    getId(data,111);
    getId(data,112);
</script>
```

```
</body>


</html>
```

我们想要输入id号,就可以返回对应的数据对象

```
<!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>
        var data = [{
            id: 1,
            name: '家电',
            goods: [{
                id: 11,
                gname: '冰箱',
                goods: [{
                        id: 111,
                        gname: '海尔'
                    }, {
                        id: 112,
                        gname: '美的'
                    },
                ]


            }, {
                id: 12,
                gname: '洗衣机'
            }]
        }, {
            id: 2,
            name: '服饰'
        }];

        // 我们想要输入id号,就可以返回对应的数据对象


        // 封装一个函数,根据用户传入的id号,"返回"找到的对象,json参数代表需要查询的数据
        function getId(json , id ){
            // 先定义一个空对象
            var obj = {};
```

```
            json.forEach(function( item ){
                if( id == item.id ){// 判断用户传入的id号是不是等于item中的id属性值
                    // console.log( item );
                    obj = item;
                    return item;
                }else{// 如果外层找不到想要的,我们可以尝试去内层找
                    // 如果item里面存着goods数组 并且goods数组的元素还需要大于0
                    if( item.goods && item.goods.length > 0){
                        // 递归调用getId函数
                        obj = getId(  item.goods  ,  id );
                    }
                }
            })


            // 返回obj对象
            return obj;
        }


        console.log( getId(data,1) );
        console.log( getId(data,2) );
        console.log( getId(data,11) );
        console.log( getId(data,12) );
        console.log( getId(data,111) );
        console.log( getId(data,112) );
    </script>
</body>


</html>
```

严格模式

什么是严格模式

JavaScript 除了提供正常模式外，还提供了严格模式（strict mode）。ES5 的严格模式是采用具有限制性 JavaScript变体的一种方式，即在严格的条件下运行 JS 代码。

严格模式在**IE10 以上版本的浏览器中才会被支持**，旧版本浏览器中会被忽略。

严格模式对正常的 JavaScript 语义做了一些更改：

1.消除了 Javascript 语法的一些不合理、不严谨之处，减少了一些怪异行为。

2.消除代码运行的一些不安全之处，保证代码运行的安全。

3.提高编译器效率，增加运行速度。

4.禁用了在 ECMAScript 的未来版本中可能会定义的一些语法，为未来新版本的 Javascript 做好铺垫。比如一些保留字如：class,enum,export, extends, import,

super 不能做变量名

开启严格模式

严格模式可以应用到整个脚本或个别函数中。因此在使用时，我们可以将严格模式分为为脚本开启严格模式和为函数开启严格模式两种情况。

为脚本开启严格模式

为整个脚本文件开启严格模式，需要在所有语句之前放一个特定语句"use strict";（或'use strict';）。

```
<script>
"use strict";
console.log("这是严格模式。");
</script>
```

```
<!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>
        // 为脚本开启严格模式
        // 为整个脚本文件开启严格模式，需要在所有语句之前放一个特定语句"use strict";(或'use strict';)。
        // 我们的脚本写正在script标签中或者.js文件中;
        // 只需要在script标签或者.js文件中所有语句之前放一个特定语句"use strict"
        "use strict";

        a = 10;// 报错,因为严格模式下,需要使用使用关键字var或者const或者let声明变量,不能省略
        console.log( a );
    </script>
</body>
</html>
```

因为"use strict"加了引号，所以老版本的浏览器会把它当作一行普通字符串而忽略。

有的 script 脚本是严格模式，有的 script 脚本是正常模式，这样不利于文件合并，所以可以将整个脚本文件放在一个立即执行的匿名函数之中。这样独立创建一个作用域而不影响其他script 脚本文件。

```
<script>
  (function (){
       "use strict";
       var num = 10;
       function fn() {}
  })();
</script>
```

为函数开启严格模式

要给某个函数开启严格模式，需要把"use strict"; (或 'use strict';) 声明放在函数体所有语句之前。

```
function fn(){
"use strict";
return "这是严格模式";
}
//当前fn函数开启了严格模式
```

将 "use strict" 放在函数体的第一行，则整个函数以 "严格模式" 运行。

```
<!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>
        // 为函数开启严格模式
        // 要给某个函数开启严格模式，需要把"use strict"; (或 'use strict'; ) 声明放在函数体所有语句之前。
        // 将 "use strict" 放在函数体的第一行，则整个函数以 "严格模式" 运行。


        // 正常模式下,变量不加var声明也不会报错
        // 但是严格模式下,变量不加var声明就会报错



        function fn1(){
            a = 10;
            console.log( a );
        }
        fn1();

        function fn2(){
            // 为fn2开启严格模式,也就是只fn2里面的代码才会使用严格模式运行
            "use strict";
            b = 20;
            console.log( b );
        }
        fn2();
    </script>
</body>
</html>
```

严格模式中的变化

严格模式对 Javascript 的语法和行为，都做了一些改变

变量规定

1. 在正常模式中，如果一个变量没有声明就赋值，默认是全局变量。严格模式禁止这种用法，变量都必须先用var声明，然后再使用变量。
2. 严禁删除已经声明变量。例如，delete x; 语法是错误的。

```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>
        // 开启严格模式
        "use strict";


        // 变量规定
        // 在正常模式中，如果一个变量没有声明就赋值，默认是全局变量。
        // 严格模式禁止这种用法，变量都必须先用var声明，然后再使用变量。
        // a = 10;
        // console.log( a );// a is not defined


        // 严禁删除已经声明变量。例如，delete x; 语法是错误的。
        var obj = {
            uname : "张三",
            age : 23
        }
        console.log( obj );


        // 使用delete删除obj中的uname属性
        // delete用法可以自行百度,https://www.cnblogs.com/SRH151219/p/10420819.html
        // delete一般只用于删除对象的属性,很少用于删除变量
        delete obj.uname;

        console.log( obj );
        console.log("");


        // delete还可以删除没有使用var声明的全局变量,但是严格模式下,会报错
        b = 20;
        console.log( b );// b is not defined
        delete b;// 报错  Delete of an unqualified identifier in strict mode.
        console.log( b );
    </script>
</body>
</html>
```

严格模式下this指向问题

1. 以前在全局作用域函数中的 this 指向 window 对象
2. **严格模式下全局作用域中函数中的 this 是 undefined**
3. 以前构造函数时不加 new也可以 调用,当普通函数，this 指向全局对象

4. 严格模式下,如果 构造函数不加new调用, this 指向的是undefined 如果给他赋值则 会报错

5. new实例化的构造函数,this指向创建对象的实例

6. 定时器、延时器里面的this还是指向window

7. 事件方法、对象的方法还是指向调用者

```
<!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>按钮</button>


    <script>
        // "use strict";
        // 以前在全局作用域函数中的 this 指向 window 对象
        // 严格模式下全局作用域中函数中的 this 是 "undefined"

        function fn(){
            console.log("全局作用域函数中");
            console.log( this );
            console.log("");
        }
        fn();


        // 以前构造函数时不加 new也可以 调用构造函数， 构造函数会被当做普通函数，this 指向全局对象
        // 严格模式下,如果 构造函数不加new调用, this 指向的是undefined 如果给他赋值则 会报错
        // new实例化的构造函数,this指向创建对象的实例
        function Star(uname,age){
            this.uname = uname;
            this.age = age;
            console.log( this );
        }
        var ldh = new Star("刘德华",18);
        // var zxy = Star("张学友",19); // 严格模式下,不能省略new关键字


        // 定时器、延时器里面的this还是指向window
        /* setInterval(function(){
            console.log("定时器中的this");
            console.log( this );
        },1000); */

        /* setTimeout(function(){
            console.log("延时器中的this");
            console.log( this );
        },3000); */


        // 事件方法、对象的方法还是指向调用者
        var obj = {
            uname:"张三",
            sayHello:function(){
```

```
            sayHello:function(){
                console.log("对象里面的方法,this指向");
                console.log( this );
            }
        }
        obj.sayHello();


        var btn = document.querySelector("button");
        btn.onclick =  function(){
            console.log("事件方法中的this");
            console.log( this );
        }
    </script>
</body>
</html>
```

函数变化

严格模式下,函数定义的时候不能有相同名的形参

```
<!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>
        // 开启严格模式
        "use strict";


        // 严格模式下,函数定义的时候不能有相同名的形参
        function getSum( a , a ){
            console.log( a + a );
        }


        getSum( 10 , 20);
    </script>
</body>
</html>
```

更多严格模式要求参考 https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode

今日总结

xmind作业要做

今日作业

1. 把今天上课讲的递归遍历数组代码最少写一遍

2. 使用面向对象完成贪吃蛇游戏

```
贪吃蛇里面有什么功能?
1. 使用js显示地图
2. 食物可以随机在地上任意一个位置出现
3. 在地图上显示一条蛇
4. 蛇可以自动移动
5. 可以通过wsad键控制蛇移动的方向  默认蛇是向右运动的
6. 蛇可以吃到食物,吃到食物以后,蛇变长,并且食物会在新的位置重新出现
7. 蛇不可以跑出地图,跑出地图外游戏结束
8. 蛇自残(蛇头如果吃到蛇身体)的话,也算游戏结束

 贪吃蛇游戏中需要哪些对象
 地图对象
 食物对象
 蛇对象
```