1.JS免费课第二周周末笔记

1、实战案例 - 倒计时

```
//=> HTML
<!-- 京东倒计时抢购 -->
<div class="time">
    <a href="#">
        京东秒杀<br/>距离本场活动结束的时间还有: <span class="timeBox" id="time
Box">00:10:00</span>
    </a>
</div>

//=> JS
/* 倒计时抢购*/
function computed() {
    var timeBox = document.getElementById('timeBox');

    var curTime = new Date(); // 电脑当前的时间
    var targetTime = new Date('2018/02/03 16:30'); //我们自己给定的活动结束的
时间;
    var areaTime = targetTime - curTime;
    //console.log(areaTime);  //150621 毫秒数

    if (areaTime < 0) {
        timeBox.innerHTML = '活动已经结束啦~';
        window.clearInterval(timer);
        return;
    }
    /*得到的毫秒数 算出小时*/
    var hour = Math.floor(areaTime / (1000 * 60 * 60));

    /*从剩余的毫秒数中 算出分钟*/
    areaTime -= hour * 1000 * 60 * 60;
    /*减去小时占的毫秒数 剩下再去算分钟占的毫秒数*/
    var minutes = Math.floor(areaTime / (1000 * 60));

    /*从剩余的毫秒数中 算出秒*/
    areaTime -= minutes * 1000 * 60;
    var seconds = Math.floor(areaTime / 1000);

    /*补0的操作 只要小于10 就在前面补一个0*/
    hour < 10 ? hour = '0' + hour : hour;
    minutes < 10 ? minutes = '0' + minutes : minutes;
    seconds < 10 ? seconds = '0' + seconds : seconds;

    timeBox.innerHTML = hour + ':' + minutes + ':' + seconds;
}
computed();
var timer = window.setInterval(computed, 1000); //每隔1s执行一次函数
```

2、实战案例 - 数组去重

1、方案一：双循环遍历法

> 遍历数组中的每一项，拿每一项和它后面的项依次比较，如果相同了，则把相同的这一项在原来数组中删除即可

```
var ary = [1, 2, 2, 2, 3, 2, 1, 2, 3, 2, 2, 3, 4, 3, 2, 2, 3];

//=>ary.length-1: 最后一项的后面没有内容了,我们不需要再比较
for (var i = 0; i < ary.length - 1; i++) {
    var cur = ary[i]; //=>当前遍历的这一项(索引i)

    //=>把拿出的这一项和后面的每一项进行比较
    //=>i+1: 把当前项和它后面项比较,当前项索引是i,后一项索引是i+1
    for (var j = i + 1; j < ary.length; j++) {
        //ary[j]:拿出来作比较的那一项
        if (cur === ary[j]) {
            //=>本次拿出来作比较的这一项和当前项相同，我们需要在原有数组中把作比较的这一项删除掉(作比较这一项的索引是j)
            ary.splice(j, 1);
        }
    }
}
console.log(ary);
```

> 数组塌陷问题 ：我们使用splice删除数组中的某一项后,删除这一项后面的每一项索引都要向前进一位(在原有索引上减一)，此时如果我们j++，循环操作的值累加了，我们通过最新j获取的元素不是紧挨删除这一项的元素，而是跳过一项获取的元素；

```
// 写法一:
for (var i = 0; i < ary.length - 1; i++) {
    var cur = ary[i];
    for (var j = i + 1; j < ary.length; j++) {
        if (cur === ary[j]) {
            ary.splice(j, 1);
            j--;//=>先让j--，然后再j++，相当于没加没减，此时j还是原有索引，再获取的时候就是删除这一项后面紧挨着的这一项
        }
    }
}

// 写法二:
for (var i = 0; i < ary.length - 1; i++) {
    var cur = ary[i];
    for (var j = i + 1; j < ary.length;) {
        cur === ary[j] ? ary.splice(j, 1) : j++;
    }
}
```

2、方案二：indexOf处理

> 利用indexOf来验证当前数组中是否包含某一项，包含把当前项删除掉（不兼容IE6~8）
> `ary.splice(i, 1)删除` 使用splice会导致后面的索引向前进一位,如果后面有很多项,消耗的性能很大；
> 解决思路：我们把最后一项拿过来替换当前要删除的这一项,然后再把最后一项删除；

```
var ary = [1, 2, 2, 2, 3, 2, 1, 2, 3, 2, 2, 3, 4, 3, 2, 2, 3];
for (var i = 0; i < ary.length; i++) {
    var cur = ary[i];//=>当前项
    var curNextAry = ary.slice(i + 1);//=>把当前项后面的那些值以一个新数组返回，我们需要比较的就是后面的这些项对应的新数组
    if (curNextAry.indexOf(cur) > -1) {
        //=>后面项组成的数组中包含当前这一项（当前这一项是重复的），我们把当前这一项删除掉即可
       // ary.splice(i, 1);  //=> 消耗性能
        ary[i] = ary[ary.length - 1];
        ary.length--;
        i--;
    }
}
console.log(ary);
```

3、方案三：对象键值对处理

> 遍历数组中的每一项，把每一项作为新对象的属性名和属性值存储起来，例如：当前项1，对象中存储的 {1:1}
>
> 在每一次向对象中存储之前，首先看一下原有对象中是否包含了这个属性（ typeof obj[xxx]==='undefined' 说明当前对象中没有xxx这个属性 ）,如果已经存在这个属性说明数组中的当前项是重复的（1-在原有数组中删除这一项 2-不再向对象中存储这个结果），如果不存在（把当前项作为对象的属性名和属性值存储进去即可）

```
var ary = [1, 2, 2, 2, 3, 2, 1, 2, 3, 2, 2, 3, 4, 3, 2, 2, 3];
var obj = {};
for (var i = 0; i < ary.length; i++) {
    var cur = ary[i];
    if (typeof obj[cur] !== 'undefined') {
        ary[i] = ary[ary.length - 1];
        ary.length--;
        i--;
        continue;
    }
    obj[cur] = cur;
}
```

4、在数组原型上扩展方法 - myUnique

```
Array.prototype.myUnique = function myUnique() {
    var obj = {};
    for (var i = 0; i < this.length; i++) {
        var item = this[i];
        if (typeof obj[item] !== 'undefined') {
            this[i] = this[this.length - 1];
            this.length--;
            i--;
            continue;
        }
        obj[item] = item;
    }
    obj = null;
    return this;
};
var ary = [12, 23, 13, 23, 12, 12, 12, 13, 23, 2, 31, 14];
console.log(ary.myUnique().sort(function (a, b) {
    return a - b;
}));
```

3、实战案例 - 算法类

1）冒泡排序

> 原理：让数组中的当前项和后一项进行比较，如果当前项大于后一项，我们让两者交换位置（小->）；

```
// 注释的写法的规范写法:

// bubble：冒泡排序
//  @parameter
//    ary：[array]需要实现排序的数组
//  @return
//    [array]排序后的数组(升序)
// by team on 2018/02/02

function bubble(ary) {
    //->外层循环控制的是比较的轮数
    for (var i = 0; i < ary.length - 1; i++) {
        //->里层循环控制每一轮比较的次数
        for (var j = 0; j < ary.length - 1 - i; j++) {
            //ary[j]:当前本次拿出来这一项
            //ary[j+1]:当前项的后一项
            if (ary[j] > ary[j + 1]) {
                //->当前这一项比后一项还要大,我们让两者交换位置
                var temp = ary[j];
                ary[j] = ary[j + 1];
                ary[j + 1] = temp;
            }
        }
    }
    return ary;
}
var ary = [12, 13, 23, 14, 16, 11];
console.log(bubble(ary));
```

2）递归(函数自己调用自己)

```
function fn(num) {
    console.log(num);
    if(num ==0){
        return;
    }
    fn(num - 1);
}
fn(10);
```

```
//=>需求：1~10以内的所有偶数乘积
function fn(num) {
    if (num <= 1) {
        return 1;
    }
    if (num % 2 === 0) {
        return num * fn(num - 1);
    }
    return fn(num - 1);
}
var result = fn(10);
console.log(result);
```

面试题：1~100之间，把所有能被3并且能被5整除的获取到，然后累加求和

```
//=> 方案一:
var total = null;
for (var i = 0; i <= 100; i++) {
    if (i % 3 == 0 && i % 5 == 0) {
       total += i;
    }
}
console.log(total); //315

//=>方案二:
function fn(num) {
    if (num > 100) {
        return 0
    }
    if (num % 15 == 0) {
        return num + fn(num + 1);
    }
    return fn(num + 1)
}
var res = fn(1);
console.log(res);
```

3）快速排序

> 原理：先找中间项，把剩余项中的每一个值和中间项进行比较，比他小的放在左边（新数组），比他大的放在右边（新数组）；

```
function quick(ary) {
    //->如果传递进来的数组只有一项或者是空的,我们则不再继续取中间项拆分
    if (ary.length <= 1) {
        return ary;
    }

    //->获取中间项的索引：把中间项的值获取到，在原有数组中删除中间项
    var centerIndex = Math.floor(ary.length / 2),
        centerValue = ary.splice(centerIndex, 1)[0];//->splice返回的是个数组,数组中包含了删除的那个内容值，把内容值获取到;

    //->用剩余数组中的每一项和中间项进行比较，比中间项大的放在右边，比他小的放在左边（左右两边都是新数组）
    var aryLeft = [],
        aryRight = [];
    for (var i = 0; i < ary.length; i++) {
        var cur = ary[i];
        cur < centerValue ? aryLeft.push(cur) : aryRight.push(cur);
    }
    return quick(aryLeft).concat(centerValue, quick(aryRight));
}
console.log(quick([12, 15, 14, 13, 16, 11]));
```

4）插入排序

```
function insert(ary) {
    //->先抓一张牌(一般都抓第一张)
    var handAry = [];//->存储的是手里已经抓取的牌
    handAry.push(ary[0]);

    //->依次循环抓取后面的牌
    for (var i = 1; i < ary.length; i++) {
        var item = ary[i];//->本次新抓的这张牌

        //->拿新抓的牌和手里现有的牌比较
        for (var j = handAry.length - 1; j >= 0; j--) {
            //handAry[j]:当前比较的手里的这张牌
            //->新抓的牌比当前比较的这张牌大了,我们把新抓的牌放在它的后面
            if (item > handAry[j]) {
                handAry.splice(j + 1, 0, item);
                break;
            }
            if (j === 0) {
                //->新抓的牌是最小的,我们把新抓的牌放在最开始的位置
                handAry.unshift(item);
            }
        }
    }
    return handAry;
}
console.log(insert([1, 2, 3, 2, 1, 2, 3, 4, 5, 23, 34, 4]));
```