学习笔记

452. 用最少数量的箭引爆气球

题目

在二维空间中有许多球形的气球。对于每个气球，提供的输入是水平方向上，气球直径的开始和结束坐标。由于它是水平的，所以纵坐标并不重要，因此只要知道开始和结束的横坐标就足够了。开始坐标总是小于结束坐标。

一支弓箭可以沿着 x 轴从不同点完全垂直地射出。在坐标 x 处射出一支箭，若有一个气球的直径的开始和结束坐标为 xstart，xend， 且满足 xstart ≤ x ≤ xend，则该气球会被引爆。可以射出的弓箭的数量没有限制。 弓箭一旦被射出之后，可以无限地前进。我们想找到使得所有气球全部被引爆，所需的弓箭的最小数量。

给你一个数组 points ，其中 points [i] = [xstart,xend] ，返回引爆所有气球所必须射出的最小弓箭数。

输入输出

```
输入：points = [[10,16],[2,8],[1,6],[7,12]]
输出：2
```

```
输入：points = [[1,2],[3,4],[5,6],[7,8]]
输出：4
```

```
输入：points = [[1,2],[2,3],[3,4],[4,5]]
输出：2
```

```
输入：points = [[1,2]]
输出：1
```

```
输入：points = [[2,3],[2,3]]
输出：1
```

思路

与435题重叠区间思路一致，先对一维数组的第二个值排序，不同的是，排序完成后，要求的是不重叠的区域，count++的位置要有所改变，且当[1,2],[2,3]时也是相交的。

代码：

```
var findMinArrowShots = function(points) {
    let count = 1; //计算不重叠区域数量
    /* 对二维数组points中的每一个一维数组的第二位升序排序 */
    for(let i = 1; i <= points.length-1; i++){
        for(let j= 1;j <= points.length-i;j++){
            let temp
            if(points[j][1] < points[j-1][1]){
                temp = points[j-1];
```

```
                points[j-1] = points[j];
                points[j] = temp;
            }
        }
    }
    let max_end = points[0][1]; //当前判断是否重叠的最大end坐标
    /* 计算不重复区间数 */
    for(let i = 1; i < points.length; i++){
        if(points[i][0] <= max_end){
            continue
        }else{
            count++;
            max_end = points[i][1];
        }
    }
    return count
}
```

js高级编程第十章——函数

重写或重载

重写：在方法名，参数列表，返回类型(除过子类中方法的返回值是父类中方法返回值的子类时)都相同的情况下，对方法体进行修改或重写

重载：同名的方法如果有不同的参数列表（参数类型不同、参数个数不同甚至是参数顺序不同）则视为重载。同时，重载对返回类型没有要求，可以相同也可以不同

JS没有重载

重名的函数，后者覆盖前者。

参数

ES5.1之前，可以用undefined来检测是否有参数，ES6可以显示定义默认参数。

```
function makeKing(name='ytt',age=18){
    return `$(name) $(age)`
}
```

传输组ES5使用apply将数组拆分

```
console.log(getSum.apply(null,values));
```

ES6有简单写法：

```
console.log(getSum(...values))
console.log(getSum(-1,...values,5))//前后都能传别的参数
console.log(getSum(...values,[5,6,7]))//也可传别的数组
```

可以吧传的参数合并为一个数组：

```
function getSum(...values){
    return values
}
console.log(getSum(1,2,3))
```

这种把参数收集为数组的方式，智能写成最后一个参数，如果前面有命名参数可以收集到，如果写在它后面则取不到，箭头函数不支持arguments，但支持参数收集。参数收集和arguments互不影响。

函数声明提升：函数声明会先被js引擎读取，在运行函数。所以声明不是一定要放在执行上下文之前。

```
sum(10,10)
function sum(num1,num2){
}
```

但是如果函数声明改成等价的函数表达式则不行。

函数内部

ES5新增arguments,this，ES6新增new.target

arguments

类数组对象，包含调用函数的传入的所有参数。

this

标砖函数,this指向调用函数的上下文对象

```
window.color = 'red'
let o = {
    color:'blue'
};
function sayColor(){
    console.log(this.color)
}
sayColor();//'red'全局上下文调用，this指向window
o.saycolor = saycolor;
o.saycolor()//'blue'把sayColor赋值给o之后再调用，this指向o
```

箭头函数this指向箭头函数上下文。

```
window.color = 'red'
let o = {
    color:'blue'
};
let sayColor = () => console.log(this.color)
sayColor();//'red'
o.saycolor = saycolor;
o.saycolor()//'red'
```

箭头函数是在window上下文中定义的。

实践回调，this指向并非想要的对象时，可以使用箭头函数解决。

```
function King(){
    this.Name = "ytt"
    setTimeout(()=>console.log(this.name),1000)
}
function Queen(){
    this.name = "zhu"
    setTimeout(function(){console.log(this.name);},1000);
}
new King()    //ytt
new Queen()   //undefined
```

new.target

ES6新增检查是否是new关键字调用的，如果不是，则值是undefined，如果是new关键字调用的，则mew.target指向被调用的构造函数。

```
function King(){
    if(!new.target){
        throw 'must use new!!'
    }
    console.log('right')
}
new King();
King;
```

函数属性及方法

1. 每个函数有两个属性：length，prototype。

   length是函数参数的个数。

   prototype保存引用类型实力的地方。

2. 函数有两个方法apply()和call()

   apply()接受两个参数，被调用的函数体内的this和一个参数数组，数组可以是Array实例也可以是arguments。

   ```
   function sum(num1,num2){
       return sum.apply(this,arguments)
   }
   function sum2(num1,num2){
       return sum.apply(this,[num1,num2])
   }
   ```

   call()也是用作传参，但形式不同，第一个参数this，剩下的参数要数个传递

   ```
   function sum(num1,num2){
       return sum.call(this,num1,num2);
   }
   ```