day16 ES5及ES6回顾及补充、运动讲解

回顾

ES5新增

严格模式

- 声明必须有关键词 （var）
- this不能指向全局变量
- arguments的实参不同步
- 函数声明只能处在上下文中
- 禁止使用八进制
- 函数名不允许相同

数组新增的高阶函数

forEach 、map 、filter、reduce、reduceRight 、every 、some

更改this指向的方法

bind 、apply 、call

对象中的getter和setter

数组的辅助方法

Array.isArray indexOf lastIndexOf ...

字符串新增

trim

序列化 反序列化

JSON.stringify JSON.pase

ES6新增

块级作用域

let 、const （常量）

字符串新增方法

includes 是否包含 、startsWith 是否开头 、endsWith 是否结尾 、repeat 平铺

数组新增

Array.of 、Array.from 、find、findIndex、fill、includes、flat （数组扁平化方法）

数组扁平化就是只将一个多维数组变成一个一维数组 [[1],[2],[3]] ==> [1,2,3]

对象简写

对象中的属性简写 （属性名和属性值的变量名一致的情况下） 对象中的函数简写 （省略对应function）

函数增强

函数中的默认参数 （在形参中使用=号指定）

箭头函数

- 没有this和arguments
- 没有原型 不能被new
- 只有一个参数可以省略（）
- 只有一行代码可以省略 {}
- 只有一行代码 且省略{} 可以省略return

解构赋值

数组解构根据对应的顺序来 var [a] = [1,2,3] a提取的是1

对象解构根据对应的属性名来 var {name} = {age:18,name:'jack'} name提取的是jack

扩展运算符 ...

打开对应的对象或者数组将里面内容暴露出来（合并）

作为修饰形参时 可以传入不定个数的参数 使用对应的形参来接收（自动封装为数组）

新增基础值类型

BIgInt （大的整型）、Symbol （独一无二的值（机器码）用于对象的属性）

generator函数 （function*）

用于解决异步问题 将异步代码同步执行 （yield 开启执行块 next方法来进入这个执行块执行 done 是否完成）

promise （es7）

用于解决异步问题 将异步代码同步执行 （then 、catch方法来进行相关操作（三种状态 等待 成功 错误））

模块化

import 导入

export 导出

增强的循环

for of （数组 实现了迭代器） 、 for in （遍历对象的）

字符串模板

ES6新增内容补充

class 类

```
//class修饰的内容是一个类 这个类是用于产生对象的
//类 （描述的模板） 与 对象 （产生的实例）
//从类变成对象的过程叫做构造    使用new关键词来构造 new关键词调用的函数被称为构造函数
//构造函数的首字母大写
//类名首字母大写
class Person {
    //构造函数 在类中的构造函数的名字就是类名
```

```
    constructor() {
        this.name = "张三" //这个地方的this指向对应的构造的对象
        //person.name = '张三'
    }
}
//构造实例
let person = new Person()
console.log(person)
```

static 修饰静态函数及属性

```
class Person {
    //构造函数 在类中的构造函数的名字就是类名
    constructor() {
    }
    static sayHello() {
        console.log('hello')
    }
    static max = 10
}
//类中的关键词 static 静态的 静态方法使用类名直接点 （静态函数优先加载）
Person.sayHello()
console.log(Person.max)
```

extends 关键词用于继承

```
//extends 继承 用于类与类之间的继承  猴子 -- 猩猩 --- 类人猿 --- 人
//继承只能继承非私有的内容
class Student extends Person{
    //继承的类中的构造函数 如果要用到this 必须先调用的super （父类构造函数）
    constructor(){
        super() //指向父类的构造函数 ===> this.name = "张三" //student.name = '张三'
        this.age = 18
        //student.age = 18
    }
}
let student = new Student()
console.log(student);
```

Set

set是一个存储数据的集合（伪数组）他不允许重复的数据（基础数据的去重）（他的值就是key）

构建

```
var set = new Set()
```

属性

size 长度

方法

- add 添加
- delete 删除
- clear 清空

- keys 、values 、entries
- forEach 遍历

```
var set = new Set([1,3,1,2,3]) //iterable 迭代器 （传入的是伪数组或者是数组）
console.log(set)
//keys 获取所有的key  values 获取所有value  entries 获取所有的键值对
console.log(set.keys())
console.log(set.values())
console.log(set.entries())
//增删改查
//添加 add push set
set.add(3)
set.add(4)
console.log(set.size) //长度获取
//删除
set.delete(4)
//删除后再添加
//清空
set.clear()
console.log(set)
//has 是否存在
console.log(set.has(1));
//forEach 根据数组的forEach一致
set.forEach(function(v,i,arr){
    console.log(v,i,arr)
})
```

weakSet 专门存在对象的set

```
//专门存储对象的set （并不能完成对象去重）
let obj = {name:'jack'}
let weakSet = new WeakSet([obj,obj,{name:'jack'}])
console.log(weakSet)
```

Map

map是一种特殊的数据结构 他是由key：value构成的，key不允许重复，value允许重复。

构造方法

```
let map = new Map()
```

属性

size 获取长度

方法

- set 设置
- get 获取
- delete 删除
- clear 清空
- keys values entries
- forEach

```
//通过传入二维数组来构建
```

```
let map = new Map([[1,'a'],[2,'b'],[3,'c'],[4,'d']])
console.log(map)
//map是通过key：value进行存储的
//添加方法 set
//同时也是修改 相同的key的值会被覆盖
map.set('name','jack')
map.set('name','rose')
map.set('name','tom')
//获取
console.log(map.get('name')) //通过key获取value
//删除
map.delete('name') //通过key来删除
console.log(map.get('name')) //通过key获取value
//清空方法
map.clear()
map.set('age',18)
//keys value  entries
console.log(map.keys())
console.log(map.values())
console.log(map.entries())
//forEach
map.forEach(function(v,k,map){ //v值 key 键  遍历的map
    console.log(v,k,map);
})
//has 判断是否存在对应的key
console.log(map.has('age'))
```

WeakMap

```
//weakMap 专门存储对象的
let weakMap = new WeakMap()
console.log(weakMap)
```

运动（上）

运动（动画）的概念，就是在一定时间内对应的位置的移动或者是样式的变化（就是css3里面的动画*）（操作dom 超出多次的重绘和回流）

运动的要素

- 迭代量 （匀速运动（迭代量不变）缓冲运动（迭代量要越来越小））
- 终止条件 （当前的位置到达目标位置 清除定时器）
- 当前位置（当前位置 = 上一个位置+迭代量）
- 时间 （定时器）

匀速运动

示例

控制一个盒子从左移动到右 到达500的时候停止运动

初始位置为0、 目标位置为500、 迭代量是10

```
<div></div>
  <script>
    //获取元素div
```

```
    var div = document.querySelector('div')
    //已知条件 初始位置为0、 目标位置为500、 迭代量是10
    var current = 0
    var target = 500
    var step = 10
    //设置定时器
    var timer = setInterval(function () {
      //在定时器内 控制对应的位置的变化
      //当前位置发生变化
      current += step
      //设置给对应的div
      div.style.left = current + 'px'
      //当到达目标位置的时候终止对应的定时器
      if(current == target){
        clearInterval(timer)
      }
    },20)
  </script>
</body>
```

封装匀速运动

```
//获取样式的方法
function getStyle(ele) {
    return getComputedStyle ? getComputedStyle(ele) : ele.currentStyle
}
//封装一个匀速运动的方法 传入元素对象 目标对象 left:300
function uniform(element, target,time=20) {
    //清除上一次的影响
    clearInterval(element.timer) //element.timer保证定时器唯一
    //设置定时器
    element.timer = setInterval(function () {
        var flag = true //判断是否全部到达目标位置
        //遍历target里面key
        for (var key in target) {
            //获取当前位置
            var current = parseInt(getStyle(element)[key])
            //当前位置如果小于目标位置 那么对应的step值是正的 否则是负
            var step = current < target[key] ? 10 : -10
            //在定时器内 控制对应的位置的变化
            //当前位置发生变化
            //设置给对应的div
            element.style[key] = current + step + 'px'
            //判断是否到达目标位置
            if (current + step != target[key]) {
                flag = false
            }
        }
        // 当到达目标位置的时候终止对应的定时器
        if(flag){
            clearInterval(element.timer)
        }
    }, time)
}
```

匀速的问题存在于对应的变化量是一样的导致的对应的时间不一致

缓冲运动

迭代量 （（目标位置-当前位置）/10）

示例

缓冲运动的透明度变化 （将对应的步长和对应的current先乘以100 再除100的操作）

```
<div></div>
<script>
    //控制div的透明度变化
    //获取div
    var div = document.querySelector('div')
    //读取当前值
    var current = 0.75
    //读取目标值
    var target = 0.5
    //设置定时器
    var timer = setInterval(function () {
        //在定时器内容设置对应的step
        var step = (target - current)*100 / 10 // 0.01 - 0.1
        //小于0向下取整  大于0向上取整 除以100 为了保证他取值范围在  0.01 - 0.1
        step = step<0?Math.floor(step):Math.ceil(step)
        //将步长和对应的当前的opacity先乘以100 再除100
        current = parseInt(current*100+step)/100
        //在定时器内设置对应的样式
        div.style.opacity = current
        //到达位置清除对应的定时器
        if (current == target) {
            clearInterval(timer)
        }
    }, 1000)
</script>
```

缓冲运动的封装

```
//运动的元素  目标对象
function bufferAnimation(element, targetObj) {
    //清除上一次的定时器
    clearInterval(element.timer)
    //设置当前定时器
    element.timer = setInterval(() => {
        //获取当前的target目标值
        var flag = true //记录当前是否全部完成
        //遍历target对象
        for (var key in targetObj) {
            //获取当前的目标值
            var target = targetObj[key]
            //获取当前的位置
            var current = parseFloat(getStyle(element)[key])
            //设置步长 (目标值-当前值)/10
            var step = target > current ? Math.ceil((target - current) / 10) :
Math.floor((target - current) / 10)
            //传入key 为 left top width height... 传统的需要带上px为单位
            if (key == 'left' || key == 'top' || key == 'width' || key ==
'height') {
                //设置对应的位置
```

```
                current =  current + step + 'px'
            }
            //如果传入的是opacity 没有单位 以及变化量要乘以100 再除100
            else if (key == 'opacity') {
                //小于0向下取整  大于0向上取整 除以100 为了保证他取值范围在  0.01 - 0.1
                step = target < current ? Math.floor((target - current) * 100 /
10) : Math.ceil((target - current) * 100 / 10)
                //将步长和对应的当前的opacity先乘以100 再除100
                current = parseInt(current * 100 + step) / 100
            }
            //如果传入的color 或者是 zIndex 直接设置
            else {
                //直接设置
                current = target
            }
            //将对应的样式设置
            element.style[key] = current
            //判断是否完成
            if(parseFloat(getStyle(element)[key]) != target){
                flag = false
            }
        }
        //判断是否全部完成
        if(flag){
            clearInterval(element.timer)
        }
    }, 20)
}
//获取当前样式的方法
function getStyle(ele) {
    return getComputedStyle ? getComputedStyle(ele) : ele.currentStyle
}
```

链式运动

概述：上一个完成继续下一个 像链子一样衔接运行

- 主要是通过对应的回调函数来完成对应的链式运动操作

```
// 运动 利用定时器来让对应的元素位置发生变化 由于定时器是异步 我不知道他什么时候执行完
function fn(callback){
    setTimeout(() => {
        console.log(1)
        //将第二个定时器的代码放入到第一个定时器里面最后执行
        setTimeout(() => {
            console.log(2)
            console.log(3)
            callback()//将函数或者代码放入到对应的异步代码里面的最后执行 就可以将我们的代码
同步执行 回调函数 最后调用的函数
        }, 2000)
    }, 2000)
}
```