学习方法

1. 建立逻辑思维
   1. 多分析
   2. 理解逻辑算法
   3. 多练、多敲、多读
2. 编程习惯
3. 大胆动手
4. 保持好奇好
5. 乐观面对错误

就业能力

1. 分析能力
2. 独立思考能力
3. 程序阅读能力
4. 程序排错能力
5. 独立编写程序能力
6. 团队协作能力
7. 学习能力
8. 善于总结的能力

专业技能

ES5及ES6

1. 严格模式
   1. 概念：所谓严格模式，从字面上就很好理解，即更严格的
   2. Demo:  
      function m1(){  
       max = 100;  
      }  
      m1();  
      alert(max); //由于max没有var 声明，因此max会变成全局变量  
      但是在严格模式下：  
      function m1(){  
       “use strict”;  
       max = 100;  
      }  
      m1();  
      alert(max);  
      报错：Uncaught ReferenceError:max is not defined  
      注：不要轻易在全局范围开头增加”use strict”声明  
       建议在作用域内部使用
2. Bind方法/this关键字(bind()方法：改变this的指向
   1. 举例：  
      btn.onclick = function(){  
       setTimeout(function(){  
       this.innerText = “已点击”;  
       }.bind(this),2000)  
      }  
      我们都知道定时器的匿名函数里不能写this，它会指向window但有了bind方法以后，我们可以随意设定this的指向
3. JSON.parse/JSON.stringify
   1. 我们从eval方法，过渡到了JSON.parse(),去除了eval方法注入代码的风险。
   2. JSON提供了两个方法
      1. JSON.parse，解析一个字符串，并返回对象
         1. 举例：  
            var obj = JSON.parse(‘{“id”:3,”value”:30}’);  
            obj.value; //30
         2. 注意：id/value都使用了双引号，否则解析不正确
      2. JSON.stringify,解析一个对象，并返回字符串
         1. 举例：  
            var obj = {id:3,value:30};  
            JSON.stringify(obj); //’{“id”:3,”value”:30}’
4. Let/const
   1. Let关键字：用来替代var的关键字
      1. 举例：  
         for(var i = 0; i < 5; i ++){  
          setTimeout(function(){  
          alert(i);  
          },1000);  
         }  
         你认为弹出的结果是多少？  
         结果：5 5 5 5 5  
         是不是跟你认为的0，1，2，3，4不太一样？？？  
         原因就在于，var i = 0,定义的不是局部变量，而是全局变量，这里不会形成闭包。（函数中嵌套函数，就形成闭包）
      2. 再举例：  
         for(var i = 0; i < 5; i ++){  
          for(var i = 0; i < 5; i ++){  
          console.log(i);  
          }  
         }  
         你认为打印的结果是多少？  
         结果：0 1 2 3 4  
         难到不是打印25次么？  
         原因还是因为，像for循环的大括号，是不会形成作用域的  
         那么如何在一个语句块中定义局部变量呢？  
         那就是使用let，不过目前大多数的ES6只允许在严格模式下使用  
         “use strict”  
         for(let i=0;i<3;i++){  
          setTimeout(function(){alert(i)},1000);  
         }  
         我们尝试一下if语句的大括号  
         “use strict”;  
         let a = 10;  
         if(window){  
          let a = 100;  
          console.log(a);  
         }  
         console.log(a);  
         结果: 100 10  
         从标准的ES6开始，我们有了块级作用域  
         例如：  
         function m(){  
          console.log(“11111111”);  
         }  
         if(flag > 5){  
          function m(){  
          console.log(“22222222”);  
          }  
         }  
         m();  
         在ES5中，函数m会在第二次定义时被覆盖。结果：22222222  
         在ES6中，则会出现11111111的结果。
   2. const
      1. 我们终于可以定义常量了！  
         const G = 9.8;
5. 变量解构
   1. destructuring解构
      1. 以前我们定义多个变量，可以这样写：  
         var x = 10,y = 20,z = 30;
      2. 现在我们可以这样写：  
         let[x,y,z] = [10,20,30];  
         let[x,[a,b],y] = [10,[15,18],20];  
         这并不是在定义数组，而是通过等号两边的结构匹配，进行赋值。
      3. 当然，如果你写的不够规范：  
         let [x,y] = [10,20,30]; //结果x = 10,y = 20  
         let[x,[a,b],y] = [10,[5],20]; //结果x = 10,a = 5,y = 20,b =undefined  
         浏览器并不会报错，依然赋值成功。
      4. 甚至可以使用对象来赋值  
         var {id,name,age} = {id:43,name:’zzl’,age:18}
   2. 解构赋值的用途
      1. 交换变量的值  
         [a,b] = [b,a]; //排序可能会常用到
      2. 函数返回多个值，这个功能比较有颠覆性  
         var [r1,r2,r3] = exec();  
         function exec(){  
          return [“结果a”,”结果b”,”结果c”];  
         }
      3. 函数的参数定义方式，不用再考虑顺序  
         function exec({time,speed,pos}){  
         }  
         执行函数：  
         exec({  
          pos : {x:20,y:55},  
          speed : 50,  
          time : 2000  
         })  
         默认值也可以顺便带上：  
         function exec({time = 2000,speed = 50,pos}){  
         }
6. 字符串扩展
   1. includes(),startsWith(),endsWith()  
      传统上，JavaScript只有indexOf方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6又提供了三种新方法。
      1. includes():返回布尔值，表示是否找到了参数字符串。  
         startsWith():返回布尔值，表示参数字符串是否在源字符串的头部。  
         endsWith():返回布尔值，表示参数字符串是否在源字符串的尾部。  
         var s = "Hello world!";  
         alert(s.startsWith("Hello")); //true  
         alert(s.endsWith("!")); //true  
         alert(s.includes("o")); //true
      2. 这三个方法都支持第二个参数，表示开始搜索的位置。  
         var s = "Hello world!";  
         alert(s.startsWith("world",6)); //true  
         alert(s.endsWith("Hello",5)); //true  
         alert(s.includes("Hello",6)); //false  
         上面代码表示，使用第二个参数n时，endsWith的行为与其他两个方法有所不同。它针对n前的字符，而其他两个方法针对从第n个位置直到字符串结束。
   2. repeat():返回一个新字符串，表示将原字符串重复n次。
      1. alert('x'.repeat(3)); //”xxx”  
         alert("hello".repeat(2)); //”hellohello”  
         alert("na".repeat(0)); //””
      2. 参数如果是小数，会被取整。  
         alert('na'.repeat(2.9)); //"nana"
      3. 如果repeat的参数是负数或者Infinity，会报错。  
         alert('na'.repeat(Infinity)); //RangeError  
         alert("na.repeat(-1)); //RangeError
      4. 但是，如果参数是0到-1之间的小数，则等同于0,这是因为会进行取整运算。0到-1之间的小数，取整以后等于-0，repeat视同为0。  
         alert(“na”.repeat(-0.9)); //””
      5. 参数NaN等同于0
      6. 如果repeat的参数是字符串，则会先转换成数字。
7. 正则扩展
   1. flags属性：ES6为正则表达式新增了flags属性，会返回正则表达式的修饰符。  
      alert(/abc/ig.source); //"abc"  
      //ES5的source属性：返回正则表达式的正文  
      alert(/abc/ig.flags); //"gi"  
      //ES6的flags属性：返回正则表达式的修饰符

第二节：应用

1. 掌握ES5内容

第三节

1. let\const关键字
2. =>函数
   1. 函数原来的写法  
      var test = function(x){  
       return x + 2;  
      }
   2. 使用箭头函数：  
      var test = x => x +2;  
      看起来很简单吧？省略了function,return关键字和大括号，使用方法跟以前一样没区别  
      test(5); //结果： 7
   3. 还有好处就是，自动绑定this  
      var obj = {  
       left : 200,  
       move : function(){  
       setTimeout(function(){  
       //this.left = 100; //以前这里不能写this  
       }，1000);  
       }  
      }  
      使用了箭头函数：  
      var obj = {  
       left : 200,  
       move : function(){  
       setTimeout( ()=>{  
       this.left = 100;  
       },1000);  
       }  
       }
   4. 当然也有一些缺陷
      1. 箭头函数是不能new 的，它的设计初衷就跟构造函数不太一样
      2. 箭头函数如果要返回一个JSON对象，必须用小括号包起来  
         var test = () =>({id:3,value=20})
      3. 不推荐使用：因为这样的设计对代码的可读性伤害太大了。（引用国际著名专家，敏捷开发创始人马丁.福勒的一句名言：任何一个傻瓜都能写出计算器可以理解的代码。惟有写出人类容易理解的代码，才是优秀的程序员。我们读代码的时间和写代码的时间比率是10：1，这意味着我们大部分时间都在阅读老代码，以便于之后新代码的编写。因为读代码占得比重太大了，因此我们希望在阅读代码的时候能够更加轻松，即便在编写代码的时候需要费点劲。）  
         省略掉一个function单词，并不能给开发效率提高多少，但牺牲的却是最基本的代码可读性。除了数学运算，我们几乎从来不用符号表示一些复杂的含义。
3. 解构赋值
4. 字符串扩展方法
5. symbol类型（第七种数据类型）
   1. var s1 = Symbol();  
       var s2 = Symbol();  
       var s3 = Symbo(“abc”);  
       var s4 = Symbol(“abc”);  
       s1 不等于s2 s3不等于s4  
      Symbol函数会生成一个唯一的值，可以理解为Symbol类型跟字符串是接近的，但每次生成唯一的值，也就是每次都不相等，至于它等于多少，并不重要。这对于一些字典变量，比较有用  
      const TYPE = {  
       SMALL : Symbol(),  
       MIDDLE :Symbol(),  
       LARGE : Symbol()  
      }  
      //以前我们可能会把SMALL、MIDDLE、LARGE赋值为数字或字符串，还要确保它们的值不能发生重复，但现在不用担心了。
6. Set和Map结构
   1. Set集合，本质上就是对数组的一种包装  
      例如：  
      let imgs = new Set();  
      imgs.add(1);  
      imgs.add(1);  
      imgs.add(5);  
      imgs.add(“5”);  
      imgs.add(new String(“abc”));  
      imgs.add(new String(“abc”));  
      for(let i of imgs){  
       document.write(i + “ “);  
      }  
      打印的结果：1 5 “5” “abc” “abc”  
      Set集合是默认去重复的，但前提是两个添加的元素严格相等，所以5和”5”不相等，两个new出来的字符串对象不相等
   2. 关于遍历的方法
      1. 由于Set集合本质上还是一个map,因此会有以下几种奇怪的遍历方法  
         var imgs = new Set([‘a’,’b’,’c’]);  
         //根据key遍历  
         for(let item of imgs.keys()){  
          console.log(item);  
         }  
         //a  
         //b  
         //c  
         //根据value遍历  
         for(let item of imgs.values()){  
          console.log(item);  
         }  
         //a  
         //b  
         //c  
         //根据key-value遍历  
         for(let item of imgs.entries()){  
          console.log(item);  
         }  
         //[‘a’,’a’]  
         //[‘b’,’b’]  
         //[‘c’,’c’]  
         //普通for…of循环（for…of跟for…in的区别很明显，就是直接取值，而不再取下标了）  
         for(let item of imgs){  
          console.log(item);  
         }  
         //a  
         //b  
         //c  
         SET集合没有提供下标方式的访问，因此只能使用for of来遍历。
      2. 下面展示了一种极为精巧的数组去重的方法  
         var array = [1,3,2,1,2,5,2];  
         var newarr = […new Set(array)];  
         解释：  
         Map集合，即映射  
         let map = new Map();  
         map.set(“S230”,”张三”);  
         map.set(“S231”,”李四”);  
         map.set(“S232”,”王五”);  
         获取某一个元素  
         map.get(“S232”); //王五  
         循环遍历，配合解构赋值  
         for(let [key,value] of map){  
          console.log(key,value);  
         }
7. Generators生成器函数
   1. 基本概念：Generator函数是ES6提供的一种异步编程解决方案，语法行为与传统函数完全不同。
      1. Generator函数有多种理解角度。从语法上，首先可以把它理解成，Generator函数是一个状态机，封装了多个内部状态。
      2. 执行Generator函数会返回一个遍历器对象，也就是说，Generator函数除了状态机，还是一个遍历器对象生成函数。返回的遍历器对象，可以依次遍历Generator函数内部的每一个状态。
      3. 形式上，Generator函数是一个普通函数，但是有两个特征。一：function关键字与函数名之间有一个星号；二：函数体内部使用yield语句，定义不同的内部状态（yield语句在英语的意思就是“产出”）。  
         function \* helloWorldGenerator(){  
          yield ‘hello’;  
          yield ‘world’;  
          return ‘ending’;  
         }  
         var hw = helloWorldGenerator();  
         上面代码定义了一个Generator函数helloWorldGenerator，它内部有两个yield语句“hello”和“world”，即该函数有三个状态：hello，world和return语句（结束执行）。  
         然后，Generator函数的调用方法与普通函数一样，也是在函数名后面加上一对圆括号。不同的是，调用Generator函数后，该函数并不执行，返回的也不是函数运行结果，而是一个指向内部状态的指针对象，也就是上面讲的遍历器对象（Iterator Object）。下一步，必须调用遍历器对象的next方法，使得指针移向下一个状态。也就是说，每次调用next方法，内部指针就从函数头部或上一次停下来的地方开始执行，直到遇到下一个yield语句（或return语句）为止。换言之，Generator函数是分段执行的，yield语句是暂停执行的标记，而next方法可以恢复执行。  
         hw.next() //{value: ‘hello’,done:false}  
         hw.next() //{value: ‘world’,done:false}  
         hw.next() //{value: ‘ending’,done:true}  
         hw.next() //{value: ‘undefined’,done:true}  
         上面代码一共调用了四次next方法。  
         第一次调用，Generator函数开始执行，直到遇到第一个yield语句为止。next方法返回一个对象，它的value属性就是当前yield语句的值hello，done属性的值false，表示遍历还没有结束。  
         第二次调用，Generator函数从上次yield语句停下的地方，一直执行到下一个yield语句。next方法返回的对象的value属性就是当前yield语句的值world，done属性的值false，表示遍历还没有结束。  
         第三次调用，Generator函数从上次yield语句停下的地方，一直执行到return语句（如果没有return语句，就执行到函数结束）。next方法返回的对象的value属性，就是紧跟在return语句后面的表达式的值（如果没有return语句，则value属性的值为undefined），done属性的值true，表示遍历已经结束。  
         第四次调用，此时Generator函数已经运行完毕，next方法返回对象的value属性为undefined，done属性为true。以后再调用next方法，返回的都是这个值。  
         总结一下，调用Generator函数，返回一个遍历器对象，代表Generator函数的内部指针。以后，每次调用遍历器对象的next方法，就会返回一个有着value和done两个属性的对象。value属性表示当前的内部状态的值，是yield语句后面那个表达式的值；done属性是一个布尔值，表示是否遍历结束。
      4. ES6没有规定，function关键字与函数名之间的星号，写在哪个位置。这导致下面的写法都能通过。  
         function \* foo(x,y){…}  
         function \*foo(x,y){…}  
         function\* foo(x,y){…}  
         function\*foo(x,y){…}  
         由于Generator函数仍然是普通函数，所以一般的写法是上面的第三种，即星号紧跟在function关键字后面。
8. 了解Class的写法
   1. Class保留字终于成了关键字
      1. 以前编写一个构造函数（类）  
         function Pad(color){  
          this.color = color;  
         }
      2. 现在的写法跟Java更接近了  
         class Iphone{  
          constructor(color,size){  
          this.color = color;  
          this.size = size;  
          }  
          playgame(){  
          ……  
          }  
          toString(){  
          return ‘这台手机的颜色是${this.color}屏幕大小是${this.size}’;  
          }  
         }  
         我们定义了一个类，名字叫Iphone  
         通过类生成一个实例：  
         var iphone = new Iphone(“白色”,5);  
         其中constructor被称之为构造方法，在我们new一个对象的时候，自动被调用  
         不过本质上，JS依然使用了原型来实现，也就是说，这不过是一个新的写法而已，跟以前的构造函数没有区别。
      3. 要注意的是，使用了class来定义类，必须先定义再使用  
         以前这样写没有问题：  
         new Person();  
         function Person(){  
         }  
         现在这样写报错：  
         new Person();  
         class Person{  
          constructor(){  
           
          }  
         }  
         甚至还可以定义一次性的类  
         let person = new class{  
          constructor(){  
            
          }  
         }();  
         这个类被定义出来，只能实例化一次，跟Java的匿名内部类很像
      4. 关于继承  
         class Son extends Father{  
          constructor(age){  
          super(); //这相当于调用了父类的构造方法，类似于传统写法Father.call(this) 基类（父类、超类）  
          //但是这个方法是必须在第一行先调用  
          this.age = age;  
          }  
         }

第四节：应用

1. 掌握class面向对象编程
2. 掌握ES6语法的灵活运用

第五节：综合应用

1. 封装拖拽效果
2. 封装常见兼容

周六练习

1. 掌握事件流的概念
2. 掌握兼容的解决方案
3. 掌握事件委托的应用场景
4. 鼠标跟随特效  
   
5. 扩展案例：滑动条拖拽  
   