js两周总结

js组成部分

- Ecmascript核心语法部分：变量，命名规范，数据类型，各种语句…
- Dom（document object model）部分：提供了操作Dom元素的属性和方法
- Bom（browser object model）部分：提供操作浏览器的属性和方法

js引入方式

- 行内式，容易被注入病毒代码，没有实现代码分离不推荐
- js内嵌式
  js有个常见的功能就是操作页面的元素，要求页面的元素必须存在，script要写在所有标签之后，body之前，或者/html标签之后
- js外链式
  一旦script里有src属性 script标签里不能让你写任何的js代码，写了也不会起作用

js调试的方式

- `alert()` 调试简单的数类型，一弹框显示
- `console.log（）` 控制台调试(可以直接写代码)
- `console.dir()` 将数据在控制台展示出来（把数据详细的打印出来 如打印对象类型）
- `document.write（ ）` 直接展示在页面中，会影响页面内容（不推荐）

获取到元素的方式

- 1.获取到div元素 （通过id名，类名，标记名等）
  - `document.getElementById("div1")`
  - `getElementsByTagName("ul")[0];` //拿到ul下的li `[0]`
  - `document` 整个文档
  - `.` 从属 例如 身体.手臂.小手.手指
  - `get` 获得
  - `Element` 一个元素
  - `By` 通过
  - `Id` id名

2.改变这个元素的特征（字体大小，背景颜色，内容等）

```
1、document.getElementById("div1").style.fontSize = "20px";
2、document.getElementById("div1").style.textDecoration= "line-throu
gh";
3、document.getElementById("div1").style.color="green";
4、document.getElementById("div1").innerHTML="<a href='http://www.zh
ufengpeixun.cn'>珠峰</a>";//优先选择innerHTML
5、document.getElementById("div1").innerText ="<a href='http://www.z
hufengpeixun.cn'>珠峰</a>";
6、绑定点击事件 当点击时操作什么事情
    document.getElementById("div1").onclick = function(){
        alert(11111);
    }
```

变量

- 变量的语法 var 变量名 = 值
- 变量作用：用来存储值 变化的量（可以存储不同的值）
- 用变量把查找的元素保存下来，变量就代表了获取到的元素
- 假如不需要再操作div1，可以再存储其他的值再利用变量
- 变量可以重复的利用

数据类型：

- 基本类型数据：数类型（`number`）,字符串（`string`）,布尔（`boolean`）,`null`和`undefined`
- 引用类型数据:对象类型
  - （`object`对象,`array`数组,`regExp`,`Date`日期）和函数类型(`function`)

区别

- 基本类型
  - 基本类型的值是不可变得
  - 基本类型的变量是存放在栈区的（栈区指内存里的栈内存）
- 引用数据类型
  - 引用类型的值是可变的
  - 引用类型的值是同时保存在栈内存和堆内存中的对象

- 引用类型的比较是引用的比较

检测变量的数据类型

- `typeof`检测变量(值)的数据类型
  - `console.log(typeof func);`typeof检测的结果是把检测类型的结果放在引用中
    `console.log(aaaa)` 若没有变量则报错`"aaaa is not defined"`
- `instanceof` 检测一个对象所属的类 (检测大范围中的小范围)
  - `console.log(a instanceof Array);` true 判断出是否属于数组这个细分类
    `console.log(reg instanceof RegExp)`

number类型

- 语法：var 变量 = sh
- `number`类型数据：**整数**，**小数**，`NaN`（not a number不是一个数）
  - 1、四则运算（+-*/%）失败时->NaN, `%`：表示两个数相除取余数
  - 2、其他数据类型转换成数类型->NaN（转换失败会显示NaN）
- 强制转换字符串时，只要有一个字符不是数字，则转换结果就是NaN

转换数类型方式

- `parseInt()`
  将字符串中整数部分提取出来从左往右一个一个字符的提取，若第一个字符不是数字，则结果为NaN
- `parseFloat()`
  将字符串中整数和小数部分提取出来从左往右一个一个字符的提取，若第一个字符不是数字，则结果为NaN
- `Number()` 或 `+()`
  把其他类型数据转成数值，可以用+号代替Number

Number() parseInt() parseFloat()

```
Number(null)
->0
Number(undefined)
->NaN
Number(true)
->1
Number(false)
->0
Number("")
```

```
 ->0
Number("10.5")
->10.5
Number("10.5a")
->NaN
Number("10a.5")
->NaN
Number([])
->0
Number({})
->NaN
```

判断一个数是否是有效数

- `isNaN()`
  - 第一步`Number()`强制转换为数类型 （**NaN或整数，小数**）
  - 第二步`isNaN（）`结果为false则是有效数，若结果为true则不是有效数

string字符串

- 语法：var string = “字符串”
- 字符串类型：将字符拿引号包起来
- `str.length`1.获取每一个字符和字符串的长度
  - 通过编号（索引，从0开始的）拿到每一个字符串`str[0] str[1]`
  - 任何字符串最后一个字符的索引为`str.length-1`
- 2.字符串中包引号 双包单 单包双 不可以双包双 单包单
- 3.如果双包双或单包单则需要转义字符\ \->\ \”->” \’->’ \t(制表符) \r\n 回车换行 将字符转义成本身的含义
- 4.变量和字符串的区别 不加引号的是变量
- 5.在字符串中`+`号表示拼接的意思

转换为字符串

- `String()`
  - String(x) // 将变量 x 转换为字符串并返回
    String(123) // 将数字 123 转换为字符串并返回
    String(100 + 23) // 将数字表达式转换为字符串并返回
- `Number 方法 toString()`也是有同样的效果。
  - x.toString()
    (123).toString()
    (100 + 23).toString()
- 日期转字符串

    - obj = new Date()
      obj.toString() // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

boolean布尔类型数据

- 布尔类型数据：`true`或`false`
    - 强制将其他类型转换为布尔类型
    - 1.只有`null,undefined,"",0,NaN`是`false`,其他的全部为`true`
    - 2.!取反`![] = !Boolean([]) = !true =false`
      `!![] = Boolean([]) = true`
      if(5){//条件判断中进行布尔类型转换`

`null`和`undefined`的区别

```
- `document.parentNode`应该能找到但是没有找到，结果`null`
- `document.body.parentnode` 语法错误，没有`parentnode`属性，从来没有存在过这个属性，结果为`undefined`
```

- `null`和`undefined` 空的意思
    - `null`
        - 1.现在没有但是以后会有 起了标识的作用 `var a = null;`
        - 2.应该有，但是没有找到
    - `undefined` 从来就没有过

object对象类型

- 对象类型的数据是多种数据类型的复合载体，对象中所有的属性相当于是对这个事物的描述,对象中多个属性组成，每个属性（键值对）之间以逗号隔开，每个属性由属性名和属性值组成，属性名和属性值之间以：隔开
- 语法：`var person = {姓名:"小明",性别:"男",身高:165,爱好:"敲代码",年龄:25}`

对象定义的方式

- 1.字面量的方式 `var obj = {}`
- 2.构造函数的方式 `var obj = new Object()`

增删改查

- 操作对象 增删改查 打点的方式或者`[]`的方式
  - 属性名是数字或者变量时，不能用打点的方式获得属性，只能用`[]`
- 1、给`obj`添加属性`width`
  `obj.width = "100px";`
  `obj["width"] = "100px";` []里的属性名要拿引号包起来
- 2、修改（保证属性是存在的，若不存在则表示添加）
  `obj.name = "lucy";`
  `obj["name"]="lucy";`
  `console.log(obj);`
- 3、查找 通过属性名获取对应的属性值
  `console.log(obj.age);` 18
  `console.log(obj["age"]);` 18
  `console.log(obj.aa);`输出`undefined`若属性不存在则结果为`undefined`
  `console.log(obj['aa'])`输出`undefined`
- 4、删除
  - 假删除
    `obj.height = null;`删除
    `obj["height"] = null;`
  - 真删除
    `delete obj["height"];`真删除
    `delete obj.height;`

ES6提供对象方法

- Object.is() 查看对应是否相等
  - Object.is({},{}) false
- assign() 合并对象
- {...obj1,...obj2} 合并对象

全局作用域

- 浏览器在拿到一个JS文件的时候，先开辟一个全局作用域 (globle scope)
- 浏览器自上而下执行JS代码（之前有一步 预解释/变量提升）
- 当遇到基本数据类型的时候
  - 1.直接在globle作用域内存里面储存字符串
  - 2.识别变量
  - 3.将变量a和储存的值联系起来
- 当遇到引用数据类型时
  - 1.先开辟一个私有的作用域空间，将内容以字符串形式存储
  - 2.将空间名赋给变量

基本类型存储过程

1、先开辟一个内存存储变量num
2、开辟一个内存存储10
3、把10赋给num，称为定义

对象类型存储过程

1、开辟一个堆内存，假设引用地址fff
2、将对象属性名和属性值存入这个堆内存
3、将引用地址fff赋值给变量obj，obj就指向了这个内存空间
注：引用类型的数据操作的是引用地址

栈内存和堆内存的区别

1、占内存存放基本类型的值，堆内存存放引用类型的值
2、堆：为成员分配和释放，由程序员自己申请、自己释放。否则发生内存泄露。典型为使用new申请的堆内容。
除了这两部分，还有一部分是：
3、 静态存储区：内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。它主要存放静态数据、全局数据和常量。

遍历对象

一次性把对象的属性名和属性值拿到
重复做一件事情 循环语句 for……in 循环

- `obj[key]` key是不固定的值，不能加引号(相当于索引)
- `arguments[i]` 获得所有索引（for循环函数中）
- `this` 当前选中的

a++

- `a++` 表示加1
  - `a++` 是先参加程序运行再加1 先执行表达式后再自增，表达式执行时使用的是a原值
- `++a` 表示加1
  - `++a` 先加1再参加程序执行 先自增再执行表达式，表达式执行时使用的是自增后的a

数组

- 数组使用中括号去描述, 每个值用逗号隔开，可以为空
- length 长度 长度是数组里面值的数量 包括空的占位
- index 索引 是从0开始的 逐1递加
- 数组里面的值是任意的

查询

- 不能用打点(newArr.1)这种方式去查询
  - newArr.0 增删改查都会报错，不能这么操作和对象一样
- 索引可以是String，因为在解释的时候会用Number(索引)解析一下

增加修改

- 数组修改或者添加可以是任意数据类型
- 数组可以不按顺序添加，中间没有的值会被空(empty)替代

各种等号=

- = 一个等号
  - a = 'b' // 基本数据类型赋值，直接替换
  - var obj = {} // xxxaaa
    obj = {} // xxxbbb 引用数据类型替换的是空间地址
- ==两个等号表示比较的意思，结果是一个布尔值
  - 两个空对象并不相等
  - 两个等号的对比， 在对比之前会把等号左右两个值转成相同的数据类型
- === 三个等号叫绝对的比较
  - 不会将两边的值进行数据类型转换
  - 如果数据类型不相等直接false
  - `var c = 1`
    `console.log(1 === c)` 和变量c代表的值去比较
    - 变量代表的是赋给它值，变量本身并不是数据类型。比较的是变量的值
- NaN 和自己本身是不相等

函数function

- 函数由于比较特殊，所以单独独立成一个函数类型
- 函数的作用：1.具有封装性（防止冲突和覆盖） 2.减少冗余代码，把实现相同功能的代

码都写在一个函数里，等下次需要实现这个功能时，只需要执行函数即可，不需要把这个功能再实现一遍

- 函数定义：
  function 函数名(){ //相当于把考勤制度制定好了
  // 函数体的内容（实现一个功能具体的逻辑）
  }
- 函数定义的步骤：
  - 1.开辟一个堆内存，假设引用地址FFF000;
  - 2.将函数体的内容以字符串的形式存在这个堆内存里
  - 3.将引用地址FFF000赋值给函数名sum,sum就代表了整个函数
- 函数的执行
  - 函数名() //相当于把考勤制度执行了
- 函数执行步骤：
  - 1.开辟一个封闭内存空间(私有作用域)
  - 2.将函数体内的字符串转换成JS代码，代码从上往下执行
  - 函数执行时每次开辟的内存空间相互之间没有任何关系
  - 实现求和的功能
    function sum(){
    var total = null;//起标识作用
    total = Math.sqrt(Math.pow(3,2)+Math.pow(4,2))
    console.log(total);
    }

  console.log(sum);
  sum();
  sum();

创建函数方式

函数名字要据有功能性（能一眼看出是做什么的）

- 函数的定义：function 函数名（型参）{函数体}
- 1、匿名函数 `function fn() {` `count = count + 10` `count = count / 2` `}` 达到加10除2 再加10除2
  - count:计数
- 2、 `function fn() { }`

执行函数

函数执行步骤：
1、开辟一个封闭的内存空间（私有作用域闭包）
2、将函数体内的字符串转换成js代码，代码从上往下执行

3、函数执行时每次开辟的内存空间，相互之间没有任何关系

- `fn1(实参)` 函数名加小括号 表示函数执行
- `fn1()` 函数名(被函数赋值的 变量) 加上小括号就是函数执行
- `function fn3(val1) { }` val 是方法执行时候穿进来的参数
    - 1. 参数可以是任意数量
    - 1. 方法执行时候传参也是任意数量
    - 1. 方法执行时传的参数和方法的参数是一一对应的。
- `fn3([1,2,3], 'asd')` 方法执行时小括号里面是参数（任意数据类型）
    - arguments 什么意思？

      > **传参最好是一个对象，需要的参数都用键值对的方式传给函数, 方便扩展**

参数

- 通过传参实现功能的差异
- 函数外面没法拿到函数内的值
- 通过 `return` 将函数内部的值暴露给外界,函数运行完后自动携带**返回值**
    - 1、return 不是一定要写，只有当函数外面想拿到函数里面的值时才需要
    - 2、若没有写return a的接收值是 `undefined`
    - 3、若写return 但return后面没有写值，则返回值是 `undefined`
    - 4、遇到return终止运行，return下面的代码终止执行

三元运算符 ternary operator

- 语法：条件 ？ 成立做的事 : 不成立做的事（相当于简单的if/else判断）

特殊情况

- 1、如果三元运算符中的某一部分不需要做任何处理，用null/undefined/void/0…站位即可
- 2、如果要执行多条语句，语句用小括号包起来，语句和语句之间用逗号隔开
    - 例：`num = 10;`
      `num >=10? (num++, num* = 10):null;`
    - 第一步也是用boolean(条件)
- 加 从左到右 逐一相加

switch case

> js中的一种判断方式

- 语法：

```
switch(n)
{
case 1:
  执行代码块 1
  break;
case 2:
  执行代码块 2
  break;
default:
  n 与 case 1 和 case 2 不同时执行的代码
}
```

- 工作原理：
  - 首先设置表达式 n（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 break 来阻止代码自动地向下一个 case 运行。
- switch case应用于变量（或者表达式等）在不同值情况下的不同操作，每一种case结束后都要加berak（结束整个判断）
- switch case中每一种case情况的比较都是基于“===”绝对相等来完成的（不同类型直接false）
- 不加break，后面的条件不管是否成立，都会被执行；利用此机制可以完成一些特殊的处理，
  - 例如num=10和num=5都要做同一种事，那么写在一起不用加break

判断语句if else if

- 语法：`if (判断条件) {}`
  - Boolean(判断条件)
  - 空数组的长度是0
  - `[,,].length == 2` 逗号是跟随前面的值
- `if else if`是一级一级的进行判断，如果当前判断成立就执行大括号里面的代码
  - 后面的判断就不再生效，也不会执行，直接跳过
  - 如果当前判断不成立，才会走下一个判断
  - 如果上面的判断都不成立，才会走最后的else

hasOwnProperty和in

- in操作符只要通过对象能访问到属性就返回true

- hasOwnProperty()只在属性存在于实例中时才返回true。

第三天

- 函数可以像对象一样通过打点或者其他方式 添加一个属性名和属性值
- 并且函数也可以通过打点等方式拿到这个属性值
- 每个函数都有一个自带的属性叫name 匿名函数的name属性的值是空字符串

创建函数的两种方式

- 1.创建函数 第一种方式创建函数 `function name(val) {}`
- 2.第二种方式创建函数 `var variable = function () {}`

函数的执行 参数（形参，实参）

- `name()`
  - 形参是个变量 代表传过去的值
  - 实参 真实传到函数里面的数据 是一个具体值
  - 函数执行的时候，如果实参是一个变量或者表达式，会先获得变量代表的值，或先执行表达式获取结果，再把结果传给形参
- 函数执行的时候，如果实参是一个变量或者是表达式,会先获取变量代表的值，或者先执行表达式获取最终的结果。再传给函数

arguments

- `arguments` 类数组 像一个数组可以通过 `[]` 加数字的方式拿到里面的值,但是没有数组有的方法,就是实参的集合,这里面可以不需要形参。
  - `arguments` 不管函数有没有形参， `arguments` 都是完整的 所有的 实参的 集合

匿名函数 自执行函数

- 创建方式
  - ```
    var fn4 = function () {
    创建变量，把函数赋值给变量
    }
    ```
  - `(function () {})()`
    - 第一步两个（）（）
    ```

        - 第二步在第一个小括号里放入 `function () {}`
        - 大括号里面写需要的代码
        - 第二个 ( ) 里面是实参
        - 切记这样的自执行函数前面的表达式必须有分号结束
        - ctrl+f 当前文件搜索
    - 第三种 !或~
        - 如果前面的代码块没有分号结束，可以用各种符号开头 例如：`% ^ &`
        - 如果前面代码已经用分号结束，就只能用 `!` 或 `~`

- 自己执行自己
- 正常函数，我们需要一个 `fnName()` 函数名加小括号的方式去执行函数

switch case

- `switch case` 是三个等号的绝对比较
  `switch (判断的值) {`
  `case 对比的值 :`
  `alert('比较成功后执行的代码')`
  `break;`
  `default:`
  `console.log()`
  `}`
    - 相当于 `if else if else` 里面的else`
    - 如果所有对比都不成立，就会执行default里面的代码块
    - 如果前面有对比成功的 就不会走到这里了

for循环

- 语法：`for (var i = 0; i < arr.length; i++) {`
  `console.log(arr[i])`
  `}`
  `
    1. var i = 0; 创建变量i 默认值设为0
    1. i < arr.length; (判断)
    1. 如果第二步是true 运行代码块 如果为false就结束循环
    1. i++ 意思就是i自增 i = i + 1

```
var arr = ['xiaobai', 'dabai', 'yuanxiao']
// console.log(arr[0], arr[1], arr[2])

for (var i = 0; i < arr.length; i++) {
```

```
    // var i = 0; 0 < 3; 代码执行; i++
    // i 等于 1;  1 < 3; 代码执行; i++
    // i 等于 2;  2 < 3; 代码执行; i++
    // i 等于 3;  3 < 3; 结束循环
    console.log(arr[i], i)
}
```

- 例子：一次循环 在偶数的时候打印出a 奇数的时候打印出b
  - `var arr1 = [1, 2, 3, 4]`
    `var flag = true`
    `for (var i = 0; i < arr1.length; i++) {`
    `if (flag) {`
    `console.log('a')`
    `flag = false`
    `} else {`
    `console.log('b')`
    `flag = true`
    `}`
    `}`

自定义属性添加的第一种方式（可以存储值以便后期使用）

```
var oDiv = document.getElementById("div1");
oDiv.className = "a1"; //固有属性设置时在结构中才能显示出来
oDiv["zhufeng"] = 0;//自定义属性在结构中不显示出来
//console.log(typeof oDiv);//object
console.dir(oDiv);
console.log(oDiv.zhufeng)
```

Break 和 Continue 语句

- break 语句用于跳出循环。
  - break 语句跳出循环后，会继续执行该循环之后的代码（如果有的话）：
  - break 语句（不带标签引用），只能用在循环或 switch 中。
- continue 用于跳过循环中的一个迭代。
  - continue 语句中断循环中的迭代，如果出现了指定的条件，然后继续循环中的下一个迭代。
  - continue 语句（带有或不带标签引用）只能用在循环中。

`Math.max`是一个函数`Math.max()`是函数执行，函数的结果是获取一个最大值

例子

隔行变色

```
*{
  margin:0;
  padding:0;
  }
ul{
  list-style: none;
  width: 500px;
  margin:0 auto;
}
ul li{
     border-bottom: 1px dashed #ccc;
     line-height: 40px;
     padding-left: 10px;
}
/*ul li:nth-child(2n){  !*偶数行*!
            background: orange;
        }
        ul li:nth-child(2n+1){!*奇数行*!
            background: pink;
        }*/
        ul li.bg0{
            background: orange;
        }
        ul li.bg1{
            background: palegreen;
        }
        ul li.cur{
            background: pink;
        }
<ul id="list">
    <li>1、HTML5、es6/es7一个也不能少</li>
    <li>2、实用至上，高效为王的DOM库（单例模式）</li>
    <li>3、比jQuery动画还精彩的CSS3+HTML5手机端效果应用</li>
    <li>4、使用观察者设计模式，从原理入手的事件库</li>
    <li>5、开发自己的移动端HTML5事件库</li>
```

```
    <li>6、模块化开发与webpack</li>
    <li>7、前端必杀技之：Node、Vue与React</li>
</ul>

    //利用对象的特征 给元素添加自定义属性
    var oUl = document.getElementById("list");
    var oLis = oUl.getElementsByTagName("li") ;//li元素的集合
    for(var i = 0;i<oLis.length;i++){
        oLis[i].className = i%2 ? "bg1":"bg0";
        //把现在的类名保存在自定义属性zhufeng
        oLis[i].zhufeng =  oLis[i].className;
        oLis[i].onmouseover = function(){//鼠标划过时做什么事情
            //事件绑定的函数里不可以出现i
            // i ->7 oLis[i]->undefined
            //this表示当前操作的元素
            this.className = "cur";
        };
        oLis[i].onmouseout = function(){//鼠标划出时做什么事情
            //还原成以前的类名  之前的类型保存在自定义属性zhufeng上
           // console.dir(this);
            this.className = this.zhufeng;
        }
    }
```

- Cannot set property 'className' of undefined 不能设置undefined的className属性
- 事件绑定函数里的i为什么会是7？ 在for循环时，事件绑定函数是定义阶段，函数体里的内容是字符串，for循环执行的速度非常快，在你没划过之前for循环已经执行结果，for循环结束i的值是7 后期再划过li时执行事件绑定函数，这函数里获取的i就是7

选项卡

```
        *{
            margin:0;
            padding:0
        }
        ul{
            list-style: none;
        }
        #tab{
            width: 500px;
            margin: 0 auto;
        }
        #tab ul{
            text-align: center;
```

```
        }
        #tab ul li{
            width: 120px;
            height: 40px;
            line-height: 40px;
            font-size: 16px;
            display: inline-block;
            border:1px solid palegreen;
        }
        #tab ul li.selectedLi{
            background:palegreen;
        }
        #tab div{
            height: 250px;
            line-height: 250px;
            background: palegreen;
            display: none;
            text-align: center;
        }
        #tab div.selectedDiv{
            display: block;
        }
    </style>
<body>
    <div id="tab">
        <ul>
            <li >新闻</li>
            <li>电影</li>
            <li >音乐</li>
        </ul>
        <div>新闻的内容</div>
        <div >电影的内容</div>
        <div >音乐的内容</div>
    </div>
</body>

    //点击的li添加选中的类名，相应div也添加选中的类名，其他的li和div应该移除类名
    //1.获取元素
    var oTab = document.getElementById("tab");
    var oLis = oTab.getElementsByTagName("li");
    var oDivs = oTab.getElementsByTagName("div");
    //2.给每个li绑定点击事件
    //3.处理点击时的逻辑  当前点击的li和对应的div添加类名，其他的元素的类名都移除
    function changeTab(n){
        //先把所有的li和div类名都去掉
        for(var i = 0;i<oLis.length;i++){
            oLis[i].className = "";
```

```
            oDivs[i].className = "";
        }
        //再给当前点击的li和div添加类名 这样就能保证只有一个是选中的
        oLis[n].className ="selectedLi";
        oDivs[n].className = "selectedDiv";
    }
    //通过for循环获取所有的li,并且给每个li绑定点击事件

    for(var i = 0;i<oLis.length;i++){
        //点击之前把每个li的索引保存在自定义属性zhufeng上
        oLis[i].zhufeng = i;
        oLis[i].onclick = function(){
            //console.log(i);//i指for循环结束后的i 也就是3
            //this 当前点击的元素 this.zhufeng存当前点击元素的索引
            changeTab(this.zhufeng) //小括号里面的实参指当前点击元素的索
引   实参不管是什么形式，都会找到对应存储的值 this.zhufeng的值作为实参传给形参
n
        }
    }
```

选项卡精简

```
<script>
    var oTab = document.getElementById("tab");
    var oLis = oTab.getElementsByTagName("li");
    var oDivs = oTab.getElementsByTagName("div");
    function changeTab(n){
        for(var i = 0;i<oLis.length;i++){
            oLis[i].className = "";
            oDivs[i].className = "";
        }
        oLis[n].className ="selectedLi";
        oDivs[n].className = "selectedDiv";
    }
    for(var i = 0;i<oLis.length;i++){
        oLis[i].zhufeng = i;
        oLis[i].onclick = function(){
            changeTab(this.zhufeng)
   }
}
```

数组去重

方法1

思路“1.拿出数组的每一项，分别跟后面所有项进行比较，若重复的则把后面重复的删除

```
var ary = [5,5,5,2,3,2,3,10];
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]){ //把索引为j的这一项删除
   ary.splice(j,1);//数组塌陷的
   j--;
    }
  }
}
console.log(ary);
```

方法2

第二种思路：借助下对象 把数组的每一项存入对象中，存入之前判断下，对象中是否有这一项，若有则说之前存过，这一项就是重复需要删除，若没有，则存入对象中方便下次判断

```
var ary = [5,5,5,2,3,2,3,10];
var obj = {};
for(var i = 0;i<ary.length;i++){
var cur = ary[i];
/*if(typeof obj[cur]=="undefined"){
    obj[cur] = cur;
    }else{
      ary.splice(i,1);
      i--;
    }*/
if(typeof obj[cur]==="undefined"){//说明对象中不存在，是第一次出现
       obj[cur]=cur;
       continue;
      }
        ary.splice(i,1);
        i--;
}
console.log(ary);
```

递归调用

- 递归调用：指方法自己调用自己 (重复执行方法里的功能)
  - 1.在方法内部调用自己的方法写在return后面

    - 2.设置边界条件，让递归调用停下来
- 将10以内的奇数想乘 1*3*5*7*9

```
function fn(n){
        if(n===1){
            return 1;
        }
        if(n%2==0){
            return fn(n-1);
        }else{
            return n*fn(n-2);
        }
  }
  console.log(fn(10));
```

- 100以内的数求和

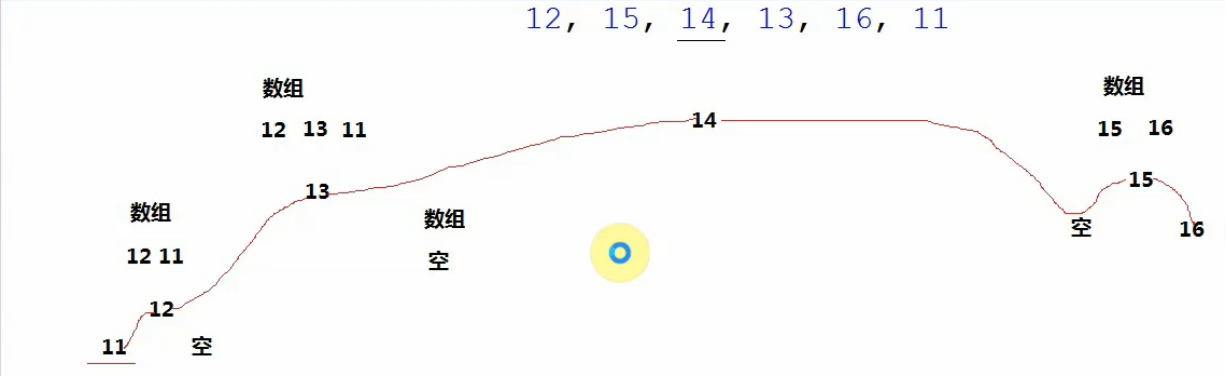
```
function fn(n){
    if(n==1){
       return 1;
        }
       return n+fn(n-1);
    }
fn(100);
```

冒泡排序

原理：两两比较，若前一项比后一项大，则他两交换位置，这样一轮下把最大的值排到最后

```
var ary = [7,5,3,2];
for(var i = 0;i<ary.length-1;i++){//轮数
        for(var j = 0;j<ary.length-1-i;j++){//次数
            //前一项 ary[j] 后一项 ary[j+1]
            if(ary[j]>ary[j+1]){
                var temp = ary[j];//把前一项的值保存在temp变量
                ary[j] = ary[j+1]; //把后一项的值赋给前一项
                ary[j+1] = temp; //把前一项的值赋值给后一项
            }
        }
    }
console.log(ary);
```

快速排序



- 思路：先取出数组的中间项，将数组的其他项跟中间项进行比较，若比中间项小则放在左手边，若比中间项大则放在右手边，左手边和右手边再重复上面的步骤，最后把所有的数合并在一起

```
    var ary1 = [33,2,5,12,18,9,10];
    function quickSort(ary){
处理边界
        if(ary.length<=1){
            return ary;
        }
先获取中间项索引
        var pointIndex =Math.floor(ary.length/2);
通过中间索引，把中间项从数组中删除
        var pointValue = ary.splice(pointIndex, 1)[0];
        var left = [];//左手边
        var right = [];//右手边
       for(var i = 0;i<ary.length;i++){
           if(ary[i]<=pointValue){
              left.push(ary[i]);
           }else{
              right.push(ary[i]);
           }
       }
       return quickSort(left).concat(pointValue,quickSort(right)) ;
    }
   quickSort(ary1);
```

js算法

- `Math.random()` 随机数
- `Math.max(7.25,7.30)` 比较两个数的最大值
- `Math.E` 常数

- `Math.PI` 圆周率
- `Math.SQRT2` 2 的平方根
- `Math.SQRT1_2` 1/2 的平方根
- `Math.LN2` 2 的自然对数
- `Math.LN10` 10 的自然对数
- `Math.LOG2E` 以 2 为底的 e 的对数
- `Math.LOG10E` 以 10 为底的 e 的对数
- `Math.round(4.7)` 四舍五入
- `Math.floor()` 向下取整
- `Math.parselnt` 向上取整

数组13种常用方法

第二阶段 扩展的数组的方法
filter,forEach,map,some,every,reduce,reduceRight,copyWithin,find,findIndex,fill,includes

- 1.push 往数组的末尾添加内容
  - 参数：添加的内容
  - 返回值：添加内容后数组的长度
  - var res = arr.push(6,7);
- 2.unshift 往数组的起始位置添加内容
  - 参数：添加的内容
  - 返回值：添加内容后数组的长度
- 3.pop 删除数组的最后一项
  - 参数：无
  - 返回值：删除的内容
  - var res = arr.pop();
- 4.shift 删除数组第一项
  - 参数：无
  - 返回值：删除的内容
- 5.splice(n) 从索引n开始删除到最后
  - 返回值:将删除的内容放在一个新数组中返回
  - var ary = [10,5,3,2,15];
    var res = ary.splice(2);
- splice(n,m) 从索引n开始删除m个
  - 返回值:将删除的内容放在一个新数组中返回
  - var ary = [10,5,3,2,15];
    var res = ary.splice(2,2)
- splice(n,m,x) 从索引n开始删除m个，用x的内容代替删除的内容 从第三个开始的参数用x表示(添加的内容)
  - var ary = [10,5,3,2,15];

var res = ary.splice(1,2,77,88,99);
console.log(res);//[5,3]
- var res = ary.splice(1,0,55,66); n=0表示在索引n之前添加内容

- 6.reverse 反向排列
  - 参数：无
  - 返回值：反向排列的数组
  - var ary = [10,5,3,2,15];
    var res = ary.reverse();

- 7.sort
  - 参数：可以不传或传函数的定义
  - var ary = [10,5,3,2,15];
    //var ary1 = [5,3,2,4,1];
    //var res = ary.sort(); //只能对10以内的数进行排序 (按照unicode编码（ASCII码）进行排列)
    var res = ary.sort(function(a,b){ //自己设置排序的方式是从大到小，还是从小到大
    return a-b //从小到大
    //return b-a //从大到小
    });
    以上方法数组会改变
    原有数组没有发生改变

    ```
    var ary = [10,5,3,2,15];
    ```

- 1.toString 将数组转换成字符串
  - var res = ary.toString();
    //console.log(res);//”10,5,3,2,15”
    //[].toString() ->””

- 2.join() 将数组按照指定的字符拼接成字符串
  - 参数：指定的字符或不传
  - var res = ary.join(“+”)
    console.log(res);//”10+5+3+2+15”
  - var res = ary.join(“”); //将数组的每一项靠在一起返回一个字符串
  - var res = ary.join();//不传参返回结果跟toString返回结果一样
    console.log(eval(res));//35 eval是全局下的方法，将字符串转换成JS代码执行

- 3.concat 合并数组的方法
  - 返回值：返回合并后的数组
  - var ary1 = [10,5];
    var ary2 = [20,3];
    var res = ary1.concat(ary2); //把ary2中的内容合并到ary1中
    console.log(res);//[10, 5, 20, 3]
  - var res = [].concat(ary1,ary2);//ary1和ary2合并到空数组中
    console.log(res)//[10, 5, 20, 3]
  - var res = ary1.concat(); //没有传参表示克隆

- 4.slice 截取数组中某些项
  - 参数：无或一个或两个
  - console.log(ary.slice()); //把原数组克隆一份
  - console.log(ary.slice(0)); //把原数组克隆一份
  - var res = ary.slice(2) //从索引2开始截取到最后
    console.log(res);//[3,2,15]
  - slice(n,m) 从索引n截取到索引m(包前不包后) n,m还可以是负数
  - var res = ary.slice(2,4);
    console.log(res)//[3,2]
  - console.log(ary.slice(-3, -1));//[3,2] 负数索引+数组的长度 倒数第三项到倒数第一项
- 5.indexOf和lastIndexOf 查找数组是否有这一项 若有则返回这一项的索引，若没有查找到则返回-1
  - indexOf 从左往右查找
  - lastIndexOf 从右往左查找
  - var ary = [10,5,2,5,13];
    console.log(ary.indexOf(7));//-1 没有7这一项，所以结果为-1
  - 扩展indexOf和lastIndexOf 还可以设置第二个参数(表示设置起始查找的位置)
    console.log(ary.indexOf(5, 2));//3 从左往右查找，从索引2位置开始查找

字符串常用方法

- 字符串排序：a.localeCompare(b)

> 掌握字符串常用的11个方法
> **根据索引查找字符**
> console.dir(String.prototype);//查看字符串中有哪些方法

- 1.charAt 根据索引查找对应的字符，若找不到则返回“”;
- 2.charCodeAt 根据索引返回对应字符的ASCII码值
  - var str = “abcABCDpfg”;
    var res = str.charCodeAt(3);
- 2.截取字符串的方法
  - 以下方法若只有一个参数表示从索引n截取到最后
    - substr(n,m) 从索引n开始截取m个
    - substring(n,m) 从索引n截取到索引m (包前不包后)
    - slice(n,m) 从索引n截取到索引m (包前不包后) 可以是负数索引
    - var str = “abcABCDpfg”;
      /*var res = str.substr(2,5);*/
      //var res = str.substring(-1,2); //负数转换成0
      //var res = str.substring(2,5); //cAB
      var res = str.slice(2,4);//cA

console.log(str.length);
var res = str.slice(-4,-2);//负数索引 = 负数索引+字符串的长度

- 3.split 将字符串按照指定的字符拆分成数组中的每一项
- var str = “2018-08-28”;
  var res = str.split(“-“);
  console.log(res);//[“2018”, “08”, “28”]
- var res = str.split(“”);将每个字符串拆开放入数组中console.log(res);// [“2”, “0”, “1”, “8”, “-“, “0”, “8”, “-“, “2”, “8”]
- var res = str.split();//将这个字符串放入数组中
  console.log(res);//[“2018-08-28”]

- 4.转换大小写的方法
  - toUpperCase 全部转换成大写
  - toLowerCase 全部转换成小写
  - var str = “abcDFG”;
    console.log(str.toUpperCase());//”ABCDFG”
    console.log(str.toLowerCase());//”abcdfg”
- 5.查找字符串是否有这个字符（通过字符串->索引）
  - indexOf 从左往右查找，找到则返回该字符索引，若找不到则返回-1
  - lastIndexOf 从右往左查找，找到则返回该字符索引，若找不到则返回-1
  - var str = “abcDFGabc”;
    console.log(str.indexOf(“b”));//1
    console.log(str.lastIndexOf(“b”));//7
    console.log(str.lastIndexOf(“k”));//-1
- 6.replace 替换字符串 返回值是替换后的结果
  - var str = “zhufeng2018zhufeng”;
    var res = str.replace(“zhufeng”,”珠峰”).replace(“zhufeng”,”珠峰”);
    var res = str.replace(/zhufeng/g,”珠峰”);
    console.log(res);
- //”2018-08-28 17:54:30” ->”2018年08月28日17时54分30秒”

  var date = “2018-08-28 17:54:30”;
  //先把字符串截取成日期部分和时间部分
  //日期部分通过-拆分成数组的每一项
  //时间部分通过：拆分成数组的每一项

  //1.先获取空格的索引
  var index = date.indexOf(” “);//10
  var str1 = date.slice(0,index);//”2018-08-28”
  var str2 = date.slice(index+1);//”17:54:30”
  var ary1 = str1.split(“-“);//[“2018”,”08”,”28”];
  var ary2 = str2.split(“:”);//[“17”,”54”,”30”]
  var resStr =
  ary1[0]+”年”+ary1[1]+”月”+ary1[2]+”日”+ary2[0]+”时”+ary2[1]+”分”+ary2[2]+”秒”;

```
console.log(resStr);//”2018年08月28日17时54分30秒”
```

DOM节点

浏览器渲染时，页面上的内容会渲染成有层次结构的节点，一个页面只有一个根节点`document`,根节点下根元素只有一个，就是html标签

文档->文档节点
文本->文本节点
注释->注释节点
标签->元素节点

nodeName(节点名称)nodeType(节点类型)|nodeValue(节点内容)

》	nodeName	nodeType	nodeValue
文档节点	`#document`	9	null
文本节点	`#text`	3	文本的内容（包括换行）
注释节点	`#comment`	8	注释的内容
元素节点	`大写的标记名`	1	null

DOM节点之间相互关系的属性

1. `childNodes` 所有的子节点(文本节点，元素节点，注释节点)
2. `children` 所有的子元素节点（IE8下把文本节点当成元素节点）
3. `firstChild` 第一个子节点
4. `firstElementChild` 第一个子元素节点（ie6~ie8不支持）
5. `lastChild` 最后第一个子节点
6. `lastElementChild` 最后一个子元素（ie6~ie8不支持）
7. `nextSibling` 相邻弟弟节点
8. `nextElementSibling` 相邻弟弟元素节点（ie6~ie8不支持）
9. `previousSibling` 相邻哥哥节点
10. `previousElementSibling` 相邻哥哥元素节点（ie6~ie8不支持）
11. `parentNode` 父元素节点

获取DOM元素

- id名 `document.getElementById(id名)`
- 标记名 `context.getElementsByTagName("li")`
- 类名 `context.getElementsByClassName("a")` ie6-ie8不支持

- name属性 `document.getElementsByName("")`

  > 在标准浏览器下对所有元素起作用 但在ie下只对表单元素起作用

- 选择器 `document.querySelectorAll()` 一组元素
  `document.querySelector()` 一个元素

  > 在移动端常用的方法

- 设置DOM元素的自定义属性
  - `ele.setAttribute(key,value);`
  - `ele.getAttribute(key);`
  -

获取浏览器屏幕宽高

```
"width": window.innerWidth,
"height": window.innerHeight
```

动态操作dom元素

- getAttributeNode() 方法：获取元素自定义属性（返回属性值）
- 创建dom节点 document.createElement（“div”）
- 创建文本节点 document.createTextNode（“珠峰”）
- 添加dom元素（添加到父节点内容的末尾位置） 父节点.appendChild（“oDiv”）
- 插入 父节点.insertBefore(newEle,oldEle)
- 替换 父节点.replaceChild(newEle,oleEle);
- 删除 父节点.removeChild(ele);
- 克隆 ele.cloneNode(true); true表示把所有的后代都克隆，不加true表示只克隆元素本身
- mainin。remove（ ）//可以删除自己

Math

- Math.sqrt() 开方
- Math.pow() 幂次方
- Math.abs() 绝对值
- Math.ceil() 向上取整
- Math.floor() 向下取正
- Math.round() 四舍五入

- Math.random() 取0~1的随机数
- Math.max() 求最大值
- Math.min() 求最小值

取0~10之间的随机整数 能取到0，取不到10

Math.floor(Math.random()*10)

取0~10之间的随机整数 不能取到0，取到10

Math.ceil(Math.random()*10)

取0~10之间的随机整数 能取到0，能取到10

Math.round(Math.random()*10)

2~62之间的随机整数 0~60 +2

Math.round(Math.random()*(62-2)+2);

n~m之间的随机整数 （ n<m ） 公式如下：

Math.round(Math.random()*(m-n)+n)

获取6个随机不重复整数

需求：从15~100之间随机的取6个不重复的整数放入数组中返回

- 1.取15~100之间的随机整数 Math.round(Math.random()*(100-15)+15)
- 2.如何取6次？ for循环取六次
- 3.解决不重复 存入数组之前判断下数组中是否有这一项，若没有则存入数组中，若有则重新取随机整数

```
function getRandom(){
        var ary = [];
        for(var i = 0;i<6;i++){
            var random = Math.round(Math.random()*(100-15)+15);
            if(ary.indexOf(random)===-1){//表示数组中没有这一项
         ary.push(random);
         continue;
            }
//问题：把重复项丢掉后，最终的数组中的成员就会少几项，就不是6个
             i--; //若重复的i的值不累加，拿这次取随机整数的机会就没有丢掉
        }
        return ary;
    }
```

```
    console.log(getRandom());

//循环的次数不确定，则推荐用while循环，若条件成立则一直执行循环体的内容，只有条件不成立时才退出while循环
function getRandom2(){
//while循环要找到条件判断是什么？
var ary = [];
while(ary.length<6){
     var random = Math.round(Math.random()*(100-15)+15);
     if(ary.indexOf(random)===-1){
        ary.push(random);
           }
       }
       return ary;
   }
console.log(getRandom2());
```

获取4位随机验证码

需求：取四个随机不重复的字符放入div1中

```
var str = "asdsadwersdv32twegrw3t2qevad2t4qegavd";
function getRandom(str){
//存入数组中，最后再把数组转换成字符串作为返回值
var ary = [];
while(ary.length<4){
//1.随机索引 ->随机的字符
var index = Math.floor(Math.random()*str.length)
var s = str.charAt(index);
if(ary.indexOf(s)===-1){
ary[ary.length] = s;//往数组中添加字符
  }
}
return  ary.join("");//相当于这个意思 ["a","b","c","d"]->"abcd" 数组的每一项靠一起返回一个字符串
}
var oDiv = document.getElementById("div1");
oDiv.innerHTML = getRandom(str)
```

动态操作dom元素

```
<div id="div2"></div>
```

```
<p>
<span>11111</span>
</p>
<script>
    var oDiv = document.createElement("div");
    oDiv.id = "div1";
   // oDiv.innerHTML = "珠峰培训";
    var oText = document.createTextNode("珠峰");
    oDiv.appendChild(oText);
    //oDiv.innerHTML = oText; //不对的 innerHTML的值必须是字符串
    //document.body.appendChild(oDiv);

   //插入
  /*  var oDiv2 = document.getElementById("div2");
    document.body.insertBefore(oDiv,oDiv2);*/

   //替换
    var oDiv2 = document.getElementById("div2");
    //document.body.replaceChild(oDiv,oDiv2);

    //删除
    document.body.removeChild(oDiv2);

    //克隆
    var oP = document.querySelector("p");
    var newP = oP.cloneNode(true);
    document.body.appendChild(newP);//克隆完后还需添加到页面才能显示
```

倒计时

- 定时器
  - window.setTimeout(方法，时间) 执行一次
  - window.setInterval(方法，时间) 如果不停止会一直执行
  - clearInterval(timer); 停止计时器

```
<div>距离下课还有<span>0时0分0秒</span></div>

需求：现在距离下课还有多少时多少分多少秒
var oSpan = document.querySelector("div>span");
function getComputed(){
   var now = new Date() ;//以现在时间为基准创建日期对象
   var target = new Date("2018/08/30 18:00:00");//以参数为基准创建日期
对象
//获取这个时间相差的毫秒数
//var time1 = target - now;
```

```
    var time = target.getTime() - now.getTime();
//1小时 = 60*60*1000
//1分钟 = 60*1000
//1秒 = 1000
//有多少小时数
    var hour = Math.floor(time/(60*60*1000));
//除去小时毫秒数剩下的毫秒数
    time = time - hour*(60*60*1000);
//有多少分钟数
    var minute = Math.floor(time/(60*1000));
//除去分钟毫秒数剩下的毫秒数
    time  =time - minute*(60*1000);
//有多少秒
    var second = Math.floor(time/1000);
    var str = hour+"时"+minute+"分"+second+"秒";
    oSpan.innerHTML = str;
     }
getComputed();
var timer = window.setInterval(getComputed,1000);

var oDiv  = document.querySelector("div");
oDiv.onclick = function(){
//若定时器是启动则让其停止
//若定时器是停止的则让其启动
console.log("点击刚开始",timer);
if(timer){//说明现在定时器是启动,要让其停止
     clearInterval(timer);
     timer = null;
}else{
timer = window.setInterval(getComputed,1000);//重新启动定时器
    }
console.log("点击结束后",timer);
}
```

求n到m之间的随机整数公式

Math.round(Math.random()*(m-n)+n)