JS笔记

JS的定义

JavaScript 是互联网上最流行的脚本语言，这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

前端分为三部分

HTML(结构层)
CSS(样式层)
javascript（行为层）

```
BOM浏览器对象模型
    console.log()
    alert()
    prompt()
    confirm()
DOM
    document.write()
ECMAScript(核心语法规则)
    变量----var 变量 = 数据;
```

document.write(“hello wordl”);
这条代码的作用就是向页面中输入（ ）之内的东西。

< script src=””“> < /script>
当一个script中有src属性的时候，不论是否有src的属性值，该代码之间不能再书写代码。

var welcome=”啦啦啦”
以上代码的含义：在内存中开辟一个空间，该空间的名字是welcome，该空间保存了一个值，就是“啦啦啦”，以上代码只是变量的定义，以后再单独遇见welcome，就是在访问，使用变量的值。

JS的书写位置

js可以卸载script标签中，js文件中。

推荐书写位置：head标签的末尾，这种情况对应的是 引入外部文件，而且外部文件的代码必须放在onload事件中。
body标签的末位，这种情况对应的是将js代码放在script标签中。

变量的命名规则：

1、整体由数字，字母，_，$组成
2、不可以以数字开头
3、变量名称不可以是关键字和保留字。
4、驼峰法命名：首写字母小写，之后的每一个单词首字母大写。

常量

> 定义一个常量
> const PI=3.14
> 常量的命名规则：
> 1、所有字母大写
> 2、每个字母用_分开

数据类型

> 1、在js中，数据类型一共有6种类型，可以划分为两大类：值类型（基本数据类型，原始值），引用类型（会在函数，数组和对象中使用）
> 2、任何基本数据类型都有包装类型（除了underfined，null）。字符串的包装类型是String，数字的包装类型是Number，布尔值的包装类型是Boolean

值类型：string（字符串），number（数值），boolean（布尔值），undefined（undefined），null（null），Symbol(第一无二的符号)

引用类型 object：函数，数组，对象
判定不同的数据类型：console.log(typeof str)
判定的方法有两种: typeof?变量 typeof(变量)

数据类型转换

字符串转数字

```
字符串转数字：
方式一、parseInt()**
```

```
该方法用于将字符串转换为数字，第一个参数是被转换的字符串
转换规则：
1、忽略字符串开头的空格
2、能转几个数字就转换几个数字
3、一旦开始转换之后，遇见的第一个非数字字符就停止
4、如果第一个就是非数字字符，则立刻停止并返还NAN
**方式二、parseFloat()**
1、忽略字符串开头的空格
2、能转几个数字就转换几个数字
3、一旦开始转换之后，遇见的第一个非小点数字字符就停止
4、如果第一个就是非数字字符，则立刻停止并返还NAN

**方式三、Number()**
转换规则：能够全部转换为数字的才能转换，只要有任何一个非数字字符，就转换失败，返还NaN，忽略开头和结尾的空白

**方式4：、   +**
转换规则：等同于Number
```

转成字符串

```
方式1、string(num)
方式2、num.toString()
方式3、num + " "
方式4、num.toFixed()
```

进制

定义二进制

var b = 0b10101010;

定义八进制

var e=07777;

注意：八进制有一个问题，如果你不小心超过了8也不会报错，但是会转为16进制。

定义十进制

var t=123124124125;

定义十六进制

var s=0xfff;

运算符

JS中的运算符:数学运算符，赋值运算符，逻辑运算符，比较运算符，三元运算符，位运算符

数学运算符：+ - * / %

```
比较运算符：  >  <    >=    <=      !=      ===    !=
比较运算符返还一个布尔值作为结果
==是相等的意思，不比较数据类型
            ===是绝对等于，比较数据类型
```

逻辑运算符有三个： &&逻辑与 ||逻辑或 ！逻辑非
&&逻辑与：有假就假，都真才真
||逻辑或：有真就真，都假才假
逻辑非：真的变假，假的变真

能表示假的数据：0 ” ” NaN undefined null fales
`false转换为数字是0，true转化为数字为1，undefined转化为数字是NAN`

```
false == undefined
结果为假，因为==会将两边向数字转换，false转为0，undefined为NAN。
```

10+12+11-11+10

提示框

浏览器自带了一些提示框
alert() 警告框
prompt() 提示框
confirm() 确认框
javascript 是一个单线程的语言

三元运算符

三元运算符也叫作三目运算符，是if else 的一个变种写法
例子：num%2 === 0? console.log(”偶数“) : console.log(“奇数”);

Switch

```
switch(条件){
      case 预测结果1:
      break;
      case 预测结果2:
      break;
      case 预测结果3:
      break;
      default: 当所有case都不符合执行
      break;
}
```

do while

执行顺序：第一次先执行do语句块内容，然后再判定while条件

do{

}

while(条件){

}

for 循环

语法规则:

for (初始值; 终止条件;改变初始值){

循环体

}

for (var i = 0;i<10;i++) {

}

执行规则：

对应编号：初始值=》 1

终止条件=》2

循环体=》3

改变初始值=》4

第一次执行：1 2 3 4

往后的每一次执行：2 3 4

函数

函数的特点：
函数分两步：1、定义 2、调用
1、第一种定义的语法: function 函数名称(形参列表){函数体}
2、第二种定义语法 var 函数名 = function (){}

3、调用方式： 函数名() 当调用函数的时候，会执行函数体内的代码

4、函数的特点：1、封装 2、复用 3、调用的时机

5、函数内部的变量，在函数运行结束之后会被收回

函数可以在任何时候调用
函数声明式可以在任何位置调用
函数表达式只能在定义函数之后使用

函数声明式:function fun(){}
函数表达式: var fun = function(){}

6、函数也是一个对象，里边有很多内置的方法和属性。
fun.name 为函数名
fun.length 为函数形参的长度

> 函数的形参和实参的对应关系
> 一一对应
> 如果形参的个数比实参多，会导致有些形状接收不到数据，值是undefiend
> 如果形参的个数比实参少，会没有形参接收

`函数自执行：(函数)()`

arguments

arguments 是函数内部一个默认的**对象**，专门用来接收实参。

return

return作用：
1、向函数外部返回数据，必须有变量接收返回的结果
2、终值函数执行
tips：return必须写在函数中

break

break ：用于中断循环
continue：跳过本次循环

作用域

作用域指的是变量起作用的范围
作用域的划分：目前我们学习的是ESS。只有函数能分割作用域。
全局作用域：script 标签之间。多个script标签之间属于同一个全局作用域。所以不同标签之间的变量是可以访问到的。但是因为script加载顺序的不同，后边的script可以访问前边script标签定义的变量，但是前面的script标签无法访问后边script标签定义的标量。

作用域分全局作用域和局部作用域，局部作用域可以访问全局作用域的内容，但是全局作用域不能访问局部作用域的内容。

当访问变量时，会开始进行变量的查找，查找时会先找当前作用域中，看是否有该变量。如果有，就使用，如果没有，就向上级作用域中查找，依次循环直到找到该变量或者到了全局作用域还没有就报错。

```
var a = 10;
function fun() {
    console.log(a);
    var b = 11;

}
fun();

// 全局作用域中 访问局部作用域中的内容 访问不到
console.log(b);
```

交换两个变量的值

```
var a = 10;
var b = 11;
[a,b] = [b,a]
```

复制变量的值

```
// 因为我们目前只学习了值类型数据类型
var n = 10;
// 复制一份n的值给m
var m = n;
// m = 10 n = 10
// 此时将两者中的任何一个修改 不会影响到另一个
console.log(n, m);
n = 11;
console.log(n, m);
```

声明变量执行过程

```
// 请说出以下代码的执行过程
var a = 10; // 开辟了一个内存空间 该空间的名字是a 存放的值是10
a = 11; // 将10丢弃 把11放进去
```

变量的声明提升

```
// JS引擎执行代码的顺序：
// 1 加载代码
    // 1.1 通篇阅览（语法错误在此阶段被检查出）
    // 1.2 声明提升
        // 指的是将所有的变量和函数 统统提升到最开头
// 2 执行代码
// 变量的声明提升
console.log(a); // 这里是可以使用的，但是没有值， 因为“声明”提升了 “赋值”没有提升
var a = 10; // 声明与赋值两个行为发生了 因为声明提升了 所以这里的主要作用就是赋值
console.log(a);// 这里是有值的
```

变量和函数的提升行为会发生在每一个作用域中，提升只会提升到本作用域的头部

局部作用域中，如果不用var定义变量，会注册到全局变量

事件

事件流程

1. + 事件流程：规定了当一系列子父元素之间拥有同类型事件，子元素触发了事件时，父元素以及祖先元素的同类型事件的执行顺序

- 元素.事件属性 = 函数 这种绑定方式，叫做DOM0级事件绑定方式，它只能够绑定到冒泡阶段
- 事件冒泡: 当点击时 事件从最精确的元素 一层一层往上触发 直到最顶层元素 这个过程叫做事件冒泡
- 事件捕获: 当点击时 事件从最顶层元素 一层一层的往下触发 直到最精确元素 这个过程叫做事件捕获
  - 最精确元素: 鼠标点到谁 谁就是最精确元素
  - 最顶层元素:
    - 高级浏览器 最顶层元素是window
    - IE中 最顶层元素是 document

> 注：当元素是最精确元素时，不区分它身上的捕获和冒泡，而是按照绑定顺序执行。

事件绑定方式

- DOM0级
  - 绑定
    - 元素.on事件类型 = 事件函数
    - 只能够绑定一个事件 因为它是对属性进行赋值
  - 移除
    - 元素.on事件类型 = null

> 我们都知道，一个对象的属性只能够保存一个值。 如果对一个对象属性进行多次赋值，后面赋值的属性会替换掉前面的属性

- DOM2级
  - 绑定
    - dom.addEventListener(type, handler, boolean)
      - type: 事件类型字符串 不带on
      - handler: 事件处理函数
      - boolean: 布尔值 决定绑定到捕获阶段还是冒泡阶段 默认是false false表示冒泡
  - 结论: 可以通过addEventListener方法进行多个事件函数的绑定 执行时是按照代码的书写顺序执行 因为代码的书写顺序决定了绑定顺序 如果有DOM0级与DOM2级同时存在，依旧按照绑定顺序执行

- 移除事件||删除事件
  - document.removeEventListener(type, handler, boolean);
    - type: 事件类型字符串 不带on
    - handler: 事件处理函数 一定要保证函数地址是绑定的那个
    - boolean: 布尔值 决定移除的是捕获阶段还是冒泡阶段 默认是false false表示冒泡
  - 结论: 第二个参数是要移除的函数 函数是引用类型 引用类型的比较的是地址 所以一定要保证 **移除的函数是当初绑定的那个函数本身**

删除，移除DOM0的绑定事件语句为：元素.事件=null

- IE中的高级绑定方式(非DOM2级)

```
// DOM2级事件绑定方式只能够在高级浏览器中使用  IE8及以下不支持
// 但是IE中有自己的高级事件绑定方式

// dom0级 不论在什么浏览器下 通用



// IE8及以下 可以使用 attachEvent 方法
document.attachEvent("onclick", function() {
    console.log(2)
})
document.onclick = function() {
    console.log(1);
```

- 绑定方式:
  - dom.attachEvent(type, handler);
    - type: 事件类型字符串 带on
    - handler: 事件处理函数
    - 没有第三个参数 意味着不能够绑定到捕获阶段
      特点: 可以对同一个元素绑定多个同类型事件的处理函数 执行起来是倒着执行 先绑定的后执行 后绑定的先执行 如果DOM0级与高级绑定方式同时存在，先执行DOM0级。再逆序执行高级绑定的函数。
- 移除方式:
  - dom.detachEvent(type, handler);
    - type: 事件类型字符串 带on
    - handler: 事件处理函数 要注意函数的地址问题

处理绑定时候的兼容性问题

```
// 定义一个函数 能够通过该函数绑定事件 而且该函数可以在任何浏览器下使用
function bindEvent(dom, type, handler) {
    if (dom.addEventListener) {
        console.log("高级浏览器")
        dom.addEventListener(type, handler, false);
    } else if (dom.attachEvent) {
        console.log("高级IE浏览器")
        dom.attachEvent("on" + type, handler);
    } else {
        console.log("其它浏览器")
        var t = "on" + type;
        dom[t] = handler;
    }
}


bindEvent(box, "mousedown", function() {
    console.log(1)
});
```

事件对象

event是浏览器收集到的触发时间时的信息组成的对象。它里面包含着鼠标的各种信息。比如鼠标位置，热键信息，触发事件的元素，绑定事件的元素。

- 热键信息 altkey 表示事件触发时键盘的alt是否按下。
- 热键信息 ctrlkey ctrl键是否按下
- 热键信息 shiftkey 键盘的shift是否被按下。
- 位置信息 offsetx offsety 鼠标在当前元素中的位置 该组属性会被子元素影响。
- 位置信息 e.clientY e.clientY鼠标在视口中的位置，
- 位置信息 鼠标在页面中的位置 e.pageX e.pageY(IE中没有)
- 位置信息 鼠标在屏幕中的位置 e.screenX e.screnn Y
- 目标元素 e.target 触发事件的元素(又叫做最精确元素 在IE中 e.srcElement)
- 目标元素 e.currentTarget 绑定事件的元素

> 事件对象本身也是具备兼容性问题的 如果是高级浏览器 浏览器会主动传递该参数 但是如果是IE8及以下 浏览器没有传递到函数中 而是放在了window.event属性上。
> 解决办法：
> var e = e || window.event;

停止冒泡

通过事件对象停止冒泡。
高级浏览器中，e.stopPropagation()

IE浏览器中， e.cancelBubble = true

阻止默认行为

e.preventDefault() 高级浏览器可用。
e.returnValue = false 如果IE8及以下，用该句。
如果是DOM0级绑定方式 则可以使用**return false;**

快捷属性

clientWidth 指的是盒模型中内容宽+左右padding
clientHeight 指的是盒模型中 内容高+上下padding .(不算border)
offsetWidth 指得是盒模型中除了margin的内容（算border）
offsetHeight 指的是盒模型中除了margin的内容
clientLeft 指的是左边框的宽度
clientTop 指的是上边框的宽度
offsetParent 属性指向 定位父元素
offsetLeft,offsetTop 从自身的边框外，到定位父元素的边框内的距离。（这两个属性有兼容性问题，独独在IE8的版本中）

鼠标事件

- 分为鼠标事件、键盘事件、表单事件、其他事件
  - 1、onclick 鼠标点击
  - 2、ondblclick 鼠标双击
  - 3、onmousedown 按钮被鼠标按下
  - 4、onmouseup 鼠标松开
  - 5、onmouseover 鼠标移到某区域触发
  - 6、onmousemove鼠标移动时候触发
  - 7、onmouseout 鼠标离开某区域触发
  - 8、onkeypress 当键盘上的某个键被按下并且释放时触发的事件.[注意:页面内必须有被聚焦的对象]
  - 10、onscroll 当视口展示的页面内容发生变化时候，会触发该事件，如果该事件有事件处理程序，会执行事件处理程序
  - 11、onKeyDown 当键盘上某个按键被按下时触发的事件[注意:页面内必须有被聚焦的对象]
  - 12、onKeyUp 当键盘上某个按键被按放开时触发的事件[注意:页面内必须有被聚焦的对象]
  - 13、onkeypress 当有字符输入的时候
  - 14、onresize 当视口尺寸发生变化的时候会触发该事件，如果该事件有事件处理程序，会执行事件处理程序
  - 15、onmouseenter onmouseenter 事件类似于 onmouseover 事件。 唯一的区别是 onmouseenter 事件不支持冒泡 。
  - 16、onmouseleave 不支持冒泡

- 17、onselect 当文本被选中
- 18、oncontextmenu 右键点击的菜单栏。
- 19、 onmousewheel 当鼠标滚轮滚动(火狐不支持，对应事件为DOMMouseScroll,该事件只能通过DOM2级添加)
- 20、 onsubmit 表单事件
- 21、oninput 表单事件，当输入的时候
- 22 、onchange 表单事件，当表单改变的时候
- 23、onfocus 获取焦点的时候
- 24、onblur 失去焦点的时候
- 25、onanimationend
- 26、onanimationstart
- 27、ontransitionend 过渡结束事件。没有开始的。
- 28、touchstart 移动端事件 当手指触摸屏幕时候触发，即使已经有一个手指放在屏幕上也会触发。
- 29、touchmove 移动端事件 当手指在屏幕上滑动的时候连续地触发。在这个事件发生期间，调用preventDefault()事件可以阻止滚动。
- 30、touchend 当手指从屏幕上离开的时候触发
- 31、touchcancel 移动端事件 事件被中断

`拓展，高级浏览器的scrollTop绑定在documentElement，IE绑定在body中`

禁止浏览器事件

```
//禁用拖拽
document.ondragstart = function () {
return false;
};
/**
```

- 禁用右键菜单
  */
  document.oncontextmenu = function () {
  event.returnValue = false;
  };
  /**
- 禁用选中功能
  */
  document.onselectstart = function () {
  event.returnValue = false;
  };
  /**
- 禁用复制功能
  */

```
document.oncopy = function () {
event.returnValue = false;
};
/**
```

- 禁用鼠标的左右键

- param {Object} e

```
*/
document.onmousedown = function () {
if (event.which == 1) {//鼠标左键
return false;
}

if (event.which == 3) {//鼠标右键
return false;
}
};
```

/**

- 获取键盘上的输入值

```
*/
document.onkeydown = function () {
console.info(event.which);
if (event.which == 13) {
console.info("回车键");
}
};
```

禁止右键
oncontextmenu="return false"

禁止复制和剪切：
oncopy="return false;"
oncut="return false;"

禁止复制
onselectstart="return false"

禁止图片拖动
ondragstart="return false"

视口属性

- window.innerWidth 视口的宽度
- window.innerHeight 视口的高度

视口：指的是浏览器提供的展示页面的部分

严格模式

ECMAScript5 中新增了“严格模式”，目的是为了保证程序的健壮性，效率。
每一次新版本的发布都会对之前的版本进行一定的修改。
“use strict”; 这是开启严格模式的方式，如果浏览器支持严格模式会开启严格模式，如果乱了你去不支持严格模式则也只是定义了一个 字符串而已。

严格模式分为两种：全局严格模式和局部严格模式

全局严格模式的开启方法：在全局作用域的所有代码前面书写”use strict” 保证他是第一条语句

局部严格模式的开启:在局部作用域的所有代码前面书写”use strict” 保证他是第一条语句

> 严格模式的规则：
> 1、变量必须被定义，必须var
> 2、不允许使用八进制
> 3、arguments.callee 函数自己。
> 4、函数的参数不允许重名
> 5、严格模式下的arguments和参数变量是分开的，一个变化不会导致另一个变化。
> 非严格模式下，arguments和参数变量是绑定在一起的

```
3 arguments.callee
function demo() {
    console.log(arguments.callee)
}
demo();

console.log(demo);
```

```
// 4 函数的参数不允许重名
function demo(a, a, b) {
    console.log(a, b);
}

demo(1, 2, 2);
```

常用的转义字符:

- \b 退格
- \n 换行
- \r 回车
- \t 制表符，横向跳格
- \’ 单引号
- \” 双引号
- \ 反斜杠
- \f 走纸换页

字符串方法总结

- **字符串方法**
  - **toUpperCase()** 让字符串字母变为大写
  - **toLowerCase()** 让字符串字母表位小写
  - **charAt()** 根据下标获取该位置的字符。
  - **charCodeAt()** 获取该下标字符的unicode编码
  - **concat()** 【重要】 字符串拼接，是字符串方法
  - **slice(strat,end)** 【非常重要】 截取字符串，两个参数，第一个是截取开始位置，第二个是结束位置，下标从0开始。当只传一个参数的时候，从参数的位置截取到最后，可以传入负值，负值从尾部开始计算，返回成员。
  - **subString(start,end)** 和slice用法相同，不能传入负数
  - **substr(start,length)** 第一个传截取的开始位置，第二个截取长度。可以传入负的参数。
  - **replace(“被替换的字符”，“要替换的字符”)** 此方法用另一个值替换在字符串中的指定值，对大小写敏感 eg：str = “Please visit Microsoft!”;var n = str.replace(“Microsoft”, “W3School”);
  - **split()** 用于以指定字符串将字符串分割成数组。split(“”) 可以将字符串每个字符分割成数组中的成员。 split(“,”) 以逗号为基准进行分割
  - **trim** 删除字符串两端的空白符，IE8及以下版本不支持。
  - **indexof()** indexof会返回检索字符串的第一个字符在被检索字符串中的下标数字，如果搜索不到，返回负一。可以接受两个参数，第一个参数是检索的字符串，第二个参数是检索开始的位置。
  - **lastIndexOf()** 会返回指定文本在字符串中的最后一次出现的索引位置。从字符串的尾巴开始检索
  - **search()** 不接受第二个参数
  - **match()** match 匹配的结果是一个数组，该数组的成员只有一项，就是匹配到的内容。如果匹配不到，返还null。数组有一些属性；
    - index 检索内容在被检索内容的下标
    - input 被检索数组的内容
    - length 数组长度
- startsWith() 该方法用于判定字符串是否是以另一个字符开头，返回布尔值。它还接收

第二个参数，表示判定开始的位置。

- endsWith() 该方法用于判定字符串是否以另一个字符串结尾，返回布尔值。他还接受第二个参数，表示判定结束的位置。
- includes() 该方法用于判定字符串是否包含另一个字符串，返回布尔值。 它也接受第二个参数，参数是判定的开始位置。
- repeat() 该方法用于将字符串复制多次，参数就是被复制的

数组

数组是JS中的引用类型之一，用于存放一组数据。引用类型又叫做复杂类型，或复合类型

值类型保存在内存中的栈内存中，比如var x = 5；会把这句全部保存在栈内存中

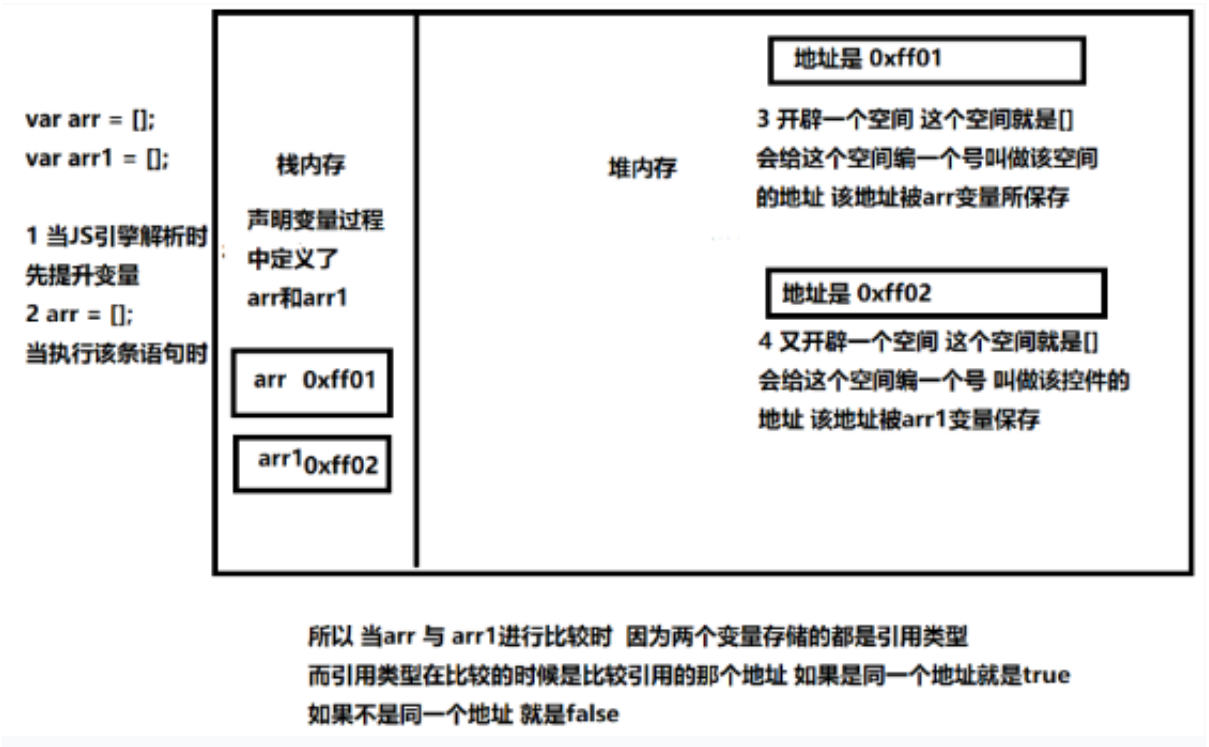
字面量（直接量）定义数组
var arr = []

构造函数定义数组。
var arr1 = new Array();
var arr2 = Array();

Array是数组的构造函数。

数组的初始化bug：当构造函数式定义数组是，如果参数是数字，并且只有一个，此时会当做数组长度来传入

length属性
【可读，可写】
赋值成员： arr[x] = 123;
读取成员：arr[x]



数组的方法

- `push()` 向数组的末尾添加一项，返回值为数组的长度。 会改变原数组
- `pop()` 向数组的末位移除一项，并返回被移除的数值。 会改变原数组
- `unshift()` 向数组的头部添加一项，返回值为数组长度。 会改变原数组
- `shift()` 从数组的头部移除一项，返回被移除的数值。 会改变原数组
- `concat()` 接受任意个参数，每一个参数都会被拼接起来，拼接到一个新数组中并返回，如果参数是数组，会将数组拆开，只拆一层。 不会改变原数组 。
- `slice()` 截取数组中的一段，它也 不会改变原数组，第一个参数是截取开始的位置，第二个是截取结束的位置 。
  + `join()` 把每个数组中的”,”改成join中的参数，并返回一个字符串。它也 不会改变原数组 。
- `sort()` 用于内部数组的排序，有时也可以进行对象的排序。 会改变原数组 eg：

```
1.  var arr = [1,4,2,6,2,6,7]
2. arr.sort(function(a,b){
3.      return a-b;    升序
4.      return b-a;     降序
5.      })
```

- `indexOf()` 数组查询 不会改变原数组
- `splice()` 会改变原数组
  - 第一个参数：操作位置
  - 第二个参数：删除项
  - 第三个及以后：新增成员
- `reverse()` 反转显示数组 ， 会改变原数组

- `forEach()` 1、这是ES5新增了的forEach方法，用来循环数组。第一个参数是函数，该函数参数有三个形参，分别表示：数组成员，数组下标，原数组。 该函数参数会执行多次。2、forEach方法的循环次数只跟数组的调用forEach时的长度有关。如果数组在循环中发生了长度的改变，不会增加循环次数。3、forEach方法只会循环有效的数组成员4、空的成员不会循环，不能使用break关键字，continue关键字跳出循环。5、第二个参数是函数执行时的this，如果没有传递默认为underfined，this指向window。

  有一种设计模式，叫做迭代器模式。它提供一种方式，能在不暴露目标数据存储方式的情况下，遍历出数据的各个成员。

```
// ES5新增了forEach方法
arr.forEach(function(value, index, arr) {
    console.log(a)
})
```

```
+ **`map()`**  和forEach() 语法完全相同，会多次执行函数，并返回一个新数组。新函数是以原数组遍历映射出来的。
```

- `filter()` 过滤器。返回一个数组，数组中的成员是否则条件的原数组的值。

```
var arr = [80, 90, 99, 95, 50, 78, 66, 68, 3];
// 过滤里面大于等于90分的
var arr1 = arr.filter(function(value) {
    return value >= 90;
})
console.log(arr1);
```

- `some()` some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。some() 方法会依次执行数组的每个元素。如果有一个元素满足条件，则表达式返回true，剩余的元素不会再执行检测。
  如果没有满足条件的元素，则返回false。

```
var arr = [80, 90, 100, 99, 95, 50, 78, 66, 68, 3];

var result = arr.some(function(value) {
    console.log(value)
    return value === 100;
})
console.log(result);
```

- `every()` some方法是有一个符合就为真，every方法是全部符合才为真。every方法接

受一个函数作为参数，参数函数有三个参数，分别是：成员，下标，数组。该参数需要返回一个布尔值作为结果，如果布尔值为真继续循环，如果布尔值为假停止循环并且every返回结果为假。如果到了循环结束都为真才返回真。

- `fill()`

```
var arr = [80, 90, 100, 99, 95, 50, 78, 66, 68, 3];
arr.fill(1); // 将数组从头至尾都设置为1
console.log(arr);
arr.fill(2, 5); // 从下标5的位置开始往后每一个都设置为2
console.log(arr);
arr.fill(3, 6, 8); // 从下标6的位置开始到下标8(不包含)的位置结束都设置为3
console.log(arr);
```

- `reduce()` reduce方法用于循环数组，接受一个函数作为参数，该函数有四个参数，分别是：函数上一次的返回值，当前循环到的成员，下标，原数组。 注意：第一次因为没有上一次函数的返回值，所以第一次函数的第一个参数是数组的第一项。reduce方法的返回值是最后一个函数执行完毕的返回值。

```
var x = [95,67,78,65,43,98,96,67,56]
var result = x.reduce(function(a,b){
    return a+b;
})
//result的值为数组所有的值相加的和
```

- `reduceRight()` reduceRight与reduce方法唯一不同的就是遍历顺序不同，reduce从左到右，reduceRight从右到左。
- `isArray()` 用于判定一个数据是否是数组。返回布尔值。

```
var arr = []
console.log(Array.isArray(arr))  //true
```

- 以下是ES6新增的方法，有两个静态方法，三个普通方法。静态方法是函数本身调用的方法，也就是Array调用的方法。普通方法指的是数组实例（就是字面量定义的数组=>[]）调用的方法。
- `Array.of ()` 该方法用于定义数组。该方法解决了Array在接受参数时的BUG。原来的BUG是当参数为N时，会创建长度为N的空数组。【静态方法】
- `Array.from()` 该方法用于将类数组对象（伪数组对象）转为数组。在ES6之前，可以通过[].slice.call(arr)方法转为数组。【静态方法】
- `find()` find方法用于模糊查询数组中的内容。find方法接受一个函数作为参数，该函数有三个形参：成员，索引，原数组。该函数要返回一个布尔值，如果为真则停止循环并返回当前成员作为find方法的返回值。如果为假则继续循环，如果到了最后依旧没有返回真，则find方法没有返回值，item变量就是undefined。【普通方法】

```
var arr = ["张三", "李四", "王五", "赵六", "罗七", "刘八"];
find方法用于模糊查询数组中的内容
想查询数组中的包含"六"这个字的成员是谁
find方法接收一个函数作为参数，该函数有三个形参：成员、索引、原数组 该函数要返回一个布尔值，
如果为真则停止循环并返回当前成员作为find方法的返回值 如果为假则继续循环。如果到了最后依旧没有返回真
则find方法没有返回值，item变量就是undefined
var item = arr.find(function(value, index, arr) {
    console.log(value);
    return value.includes("六");
});
```

- `findIndex` 该方法的用法与find方法一致，返回值是否he条件的那个下标。如果没有符合条件的，最终返回-1。【普通方法】
- `copyWithin` 该方法用于数组的内部复制，第一个参数表示替换的开始位置 第二个参数表示复制的开始位置 第三个参数表示复制的结束位置【普通方法】

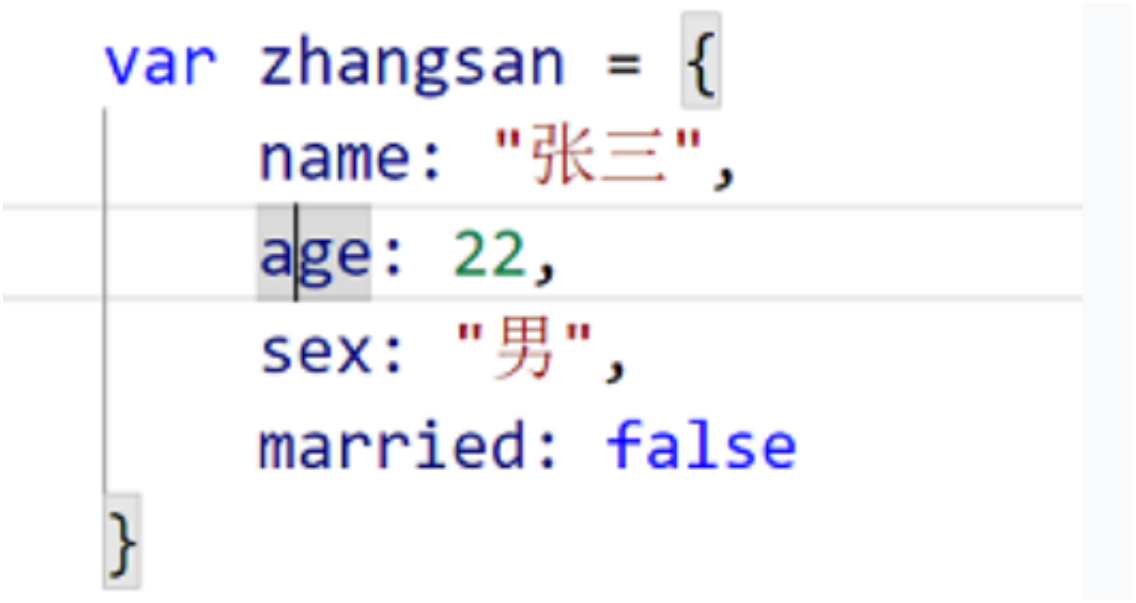
```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];

arr.copyWithin(4, 3, 6);// 从下标3的位置开始复制，到下标6的位置结束(不包含6) 将复制到的内容 从
下标4的位置开始替换
```

对象

对象是属性的无序集合

定义方法：
1、字面量（直接量） var a = {}
2、构造函数var a = new Object()

对象的定义规则：
1、{}表示定义一个对象。
2、对象由属性组成
3、属性由属性名和属性值组成
4、属性名和属性值之间使用 ： 分隔
5、每一组属性之间使用 ，分隔
6、属性名可以随意书写，但是推荐符合变量命名规范
7、属性值可以是任意的数据类型。对象中可以有对象
8、当对象属性加了""之后，可以不符合变量命名规则
操作对象：



语法1：console.log(zhangsan[“age”])
语法2：console.log(zhangsan.age)
增加属性，修改属性
zhangsan[“age”] = 123; 修改属性值
zhangsan.length = 333; 新增/修改属性

删除属性
delete zhangsan.length; 会返回一个操作结果，true或false 。不论有没有该属性都会返回true，除非该对象被冻结。

冻结对象
object.freeze(zhangsan);冻结对象，冻结后不能操作对象，此时操作属性会返回true。

密封对象
object.seal 密封对象

循环对象
for(var i in zhangsan){
console.log(i) 只会读取张三的 i 属性
console.log(zhangsan[i]); for in循环中，当参数是i的时候，只能使用方括号语法读取对应的值
}

> `for in 循环和for循环的区别`
> for in循环不会执行空项的循环

直接取出对象的各个属性

对象由内置的属性方法：Object.keys()；该方法返回一个数组，数组内包括对象内可美剧的属性以及方法名称。数组中的属性名的顺序和for in循环的顺序一致。 `var keys =`

`Object.keys(对象名)`

对象方法

- Object.is() 用于比较两者是否全等。【静态方法】

```
console.log(NaN === NaN);  // false
console.log(Object.is(NaN, NaN)); // true
console.log(0 === -0); // true
console.log(Object.is(0, -0)); // false
```

- Object.assign(接受的对象，复制对象1，复制对象2，复制对象3，……) 用于对象属性的复制（浅复制），如果有相同的属性，后边的参数会覆盖前边的属性。

Math

`math是js定义的一个对象，内置了各种数学计算函数。`

Math.abs() 该方法获取一个数的绝对值。

** 向下取整数。整数会变小。3.9变为3 ，-3.9变为-4

Math.ceil() 向上取整

Math.floor(x) 向下取整

Math.round() 四舍五入。

Math.max() 取传入的参数的最大值。

Math.min() 取传入参数的最小值

Math.random() 取0—1之间的随机数，可能有0，不可能有1。 从a到b的随机取值公式：

Math.floor((Math.random()*(y-x+1))+x)

Math.pow(2,3) 求2的三次方

深复制和浅复制

浅复制只复制值类型，深复制也可以复制引用类型

递归

`递归：递，传递 归，回归，专门指函数的一种高级使用方式，函数内部调用函数自己。没有停止条件的递归就是死循环，要用递归，先写停止条件`

```
var i = 0;
function demo() {
    console.log(1);
    i++;
    if (i == 9) {
        return;
    }
    demo();
}
demo();
```

Date

Date() 是一个函数，返回值是一串字符串
new Date() 返回值是一个对象。
var d = new Date();

```
var d7 = new Date(2019, 0, 2, 14, 23, 44, 777);
console.log(d7);
// 总结：任何一个数值越界都会导致进位
```

下边的方法都是Date对象的方法，所以使用时需要先new
Date对象的方法、

- getFullYear() 获取年
- getMonth() 获取月
- getDate() 获取日
- getHours() 获取时
- getMinutes() 获取分
- getSeconds() 获取秒
- getMilliseconds() 获取秒毫秒
- getDay() 获取星期几 0表示星期日，1-6表示星期一~星期六
    - getTime() 获取1970年1月1日零点至今的毫秒值
      该方法除了getDay没有对应的set方法外，其他都有

定时器

- setInterval() 每间隔多久就执行一次代码，除非手动清除该定时器否则不会停止。

- setTimeout() 只会在间隔时间之后执行一次就结束，之后不会再执行。
  都接受两个参数，第一个是函数或者字符串（不推荐使用字符串），第二个是毫秒数字

> 清除定时器 clearInterval() 可以接受一个参数，参数为要结束的定时器返回的句柄值。

操作元素的类名

```
<style>
    * {
        margin: 0;
        padding: 0;
    }
    .box {
        width: 100px;
        height: 100px;
        background-color: red;
    }
    div {
        height: 50px;
        background-color: blue;
    }
</style>
<body>
<div style="" id="box" class="abc" >你好</div>
<script>
    // 操作元素的类名
    // 因为class是JS中的关键字 需要避讳一下 使用className来代替
    console.log(box.className);
```

```
22.     // setTimeout(function() {
23.     //      box.className = "box";
24.     // }, 1000);
25.     box.onclick = function() {
26.         box.className = "box";
27.     }
28.
29.
30.
31.     // 操作样式
32.     box.style.lineHeight = 20;
33.     box.style["line-height"] = 1;
34.     box.style.backgroundColor = "rgba(244,
  23, 56, .7)";
35.     dom.setAttribute('style','background-c
  olor:blue;border:1px solid black;')
36.     dom.style.cssText="background-color:bl
  ue;left:200px"
37.
38.     // 设置内部文本
39.     box.innerHTML = "大吉大利,今晚吃鸡";
40. </script>
```

函数中的this

this是函数中特有的内容，任何一个**this**必须是一个引用类型。

全局中的this是window，局部作用域中的this到底是谁，定义时无法确定，只有调用时候才能确定。

函数中的this指向谁，根据一个原则确定：谁调用就指向谁，如果调用者不明确，将会指向window。

确定调用者的方式：点前面是谁，调用者就是谁。

在JS中，函数，数组都是对象，因为它们是引用类型，所以身上可以有属性，每一个属性都天生拥有好多属性和方法。都可以使用for in循环

call的作用：

- 立刻执行这个函数
- 将函数中的this指向改变成call所接受的第一个参数。
- 原函数所需的参数要从call的第二个参数开始传递。

apply的作用

和call的区别，传递参数的时候，不是从第二个参数位置往后传递，而是将所有的参数存入数组中，再将数组放在第二个参数位置。

bind

bind是ES5中新增的改变this指向的方法。

1、返回值是一个函数。

2、当新的函数是执行的时候，里面的this是bind的第一个参数。原函数依旧遵循谁调用就是谁的规则。

3、可以预定义参数。

```
// 特点一：返回值是函数
// 函数中改变this指向的方法现在一共有3个： call、 apply、 bind
// var result = demo.bind(document.body)
// // bind方法返回的是一个新的函数
// console.log(result)
// // 当新的函数执行的时候 里面的this是bind的第一个参数
// result();
// // 原函数依旧遵循：“谁调用 就是谁”的规则
// demo();

// 特点二：可以预定义参数
var result = demo.bind(document.body, 1, 2)
// 最终输出的arguments => [1, 2, 3, 4, 5]
result(3, 4, 5);
```

4、无法再使用call和apply改变里面的this

```
// 特点三：无法再使用call改变里面的this
result.apply(document)
```

- BOM：是Browser object model 可以简单理解为window 包含：BOM，历史记录，地址栏内容，setTnterval,setTimeout,关闭页面，alert，prompt，confirm
  - getComputedStyle() 获得元素的内部样式。当使用dom的div.style.width只能获得元素的内联样式。使用它必须加window `有兼容性问题，IE8及之前浏览器不能用，可以用currentStyle`
- DOM：是Document Object Model 简单理解为document; 所有与文档相关的操作都需要通过DOM来完成
- DOM元素，指的是标签通过document获取之后在JS中的身份

window

- location 浏览器的地址栏
- history 浏览历史
  - history.back() 后退
  - history.forward() 前进
  - history.go() 参数为正前进N步，参数为负后退N步。参数为0一直刷新。
- navigator 当前浏览器信息。可以检测到用户的浏览器 window.navigator.userAgent可以判断浏览器型号

DOM

DOM：Document Object Model
所有与页面相关的操作都要通过DOM
DOM树：浏览器会把所有的标签抽象成一颗树状结构。叫做DOM
我们可以根据document对象，获取元素，创建元素，追加元素，删除元素

> 获取元素的方式
> 1、根据ID获取元素（没有兼容问题）
> var box = document.getElementById(“id”)
> 2、根据标签名获取元素集合（没有兼容问题）
> var box = document.getElementsByTagName(“div”)
> 3、根据name属性获取元素集合document.getElementByName()
> 4、根据类名获取元素集合
> document.getElementClassName()
> 5、根据选择器获取一个元素
> document.querySelector()
> 6、根据选择器获取多个元素的集合
> document.querySelectorAll()

节点

节点：就像一颗树的每一个分叉部位就是节点。 我们整个文档由12种节点组成。元素只是节点的一种。

- 节点名称 name.nodeName
- 节点值 name.nodeValue
- 节点类型-节点名称-节点值
  - 1、元素-标签名大写字符串- null
  - 2、属性 -属性名- 属性值
  - 3、文本-#text- 文本字符串
  - 8、注释-#comment- 注释的字符串
  - 9、文档（document）-#document- null

```
var list = document.getElementById("list");
    // list本身就是节点所以它拥有节点所拥有的属性
```

节点关系：节点关系就两种关系，父子关系，兄弟关系。

父找子

```
// 获取ul
var ul = document.querySelector("ul");
// 1 父找子
var childNodes = ul.childNodes; // 找到的是所有的子节点集合
var children = ul.children; // 找到的是所有的子元素节点集合
var firstChild = ul.firstChild; // 找的是大儿子
var firstElementChild = ul.firstElementChild; // 找到的是第一个元素儿子
var lastChild = ul.lastChild; // 找到的是小儿子
var lastElementChild = ul.lastElementChild; // 找到的是最后一个元素儿子
```

子找父
console.log(ul.parentNode)
兄弟

```
1.  ul.nextSibling 下一个兄弟节点
2.  ul.nextElementSibling   下一个元素兄弟节点
3.  ul.previousSibling 前一个兄弟节点
4.  ul.previousElementSibling       前一个元素兄弟节点
5.
6.
7.  **以上属性，都必须具备具体的目标元素才能找到，如果没有返回Null**
```

元素操作：创建，追加，移除，克隆，替换

- 创建
  var div = document.createElement(“div”)
  创建了一个孤儿节点。
  document.body.appendChild(div)
  把孤儿节点追加到dom树上
- **两种移除方法**
  - 自身移除 div.remove()
  - 被父元素移除 document.body.removeChild(div)
- **克隆** p.cloneNode() 只克隆元素自身，不克隆子元素。 可以接受一个参数，参数为true时候，可以克隆子元素。
- **替换** document.body.replaceChild(p1,p)
- **插入**
  - 插入到前 insertBefore(p1,p)
    第一个参数是要插入的元素，第二个是被插入的元素

回调函数

通常指的是异步代码完成之后执行的函数。

同步代码：指的是代码严格按照书写顺序从上到下执行。基本所有的代码都是同步代码。因为JS是单线程的语言。
异步代码：指的是代码不按照顺序严格执行。上一条代码还没有执行完毕，先执行下一条代码，通常会出现在堵塞和耗时行为中。

内存泄漏

指的是当元素或变量被删除或者改变指向时，对应的内存空间无法被回收利用。

垃圾回收机制

根据浏览器的不同分为两种：

> 第一种：引用计数 每当一个内存空间被一个变量保存，计数+1，每当一个变量不再保存该地址，则计数减一，当为0时，应当被清除。该种方式存在一个问题：循环引用，比如a保存b，b保存a，此时a和b的计数永远不会为0
> 第二种：标记清除（现在一般是这个）标记清除有一个很重要的概念，就是根。根可以理解为全局变量和DOM树。和DOM数没有关系的节点将会被回收。

事件委托模式

将子元素的事件委托给不可能被删除的祖先元素。 但是因为祖先元素范围比较大，所以为了实现“点击原来的元素才执行事件”的效果，我们就必须分辨用户点击的元素是谁。

委托模式的三个特点：
1、减少事件数量
2、预言未来元素
3、防止内存泄漏

ES6

ES6的正式名称：ECMAScript2015
我们之前学的都是ES5、ES3.1等
ES6提供了新的关键字，新的语法，新的数据类型，新的数据结构，异步编程，新的作用域，类的定义

块级作用域

ES6中规定：{}之间也会分割作用域，叫做块级作用域：{}之间就是块级作用域。需要和let关键字配合。

let关键字

let是ES6中的关键字。

- let声明的变量，遵循块级作用域的规则。
- let声明的变量，在同一个作用域中命名不能重复。
- let声明的变量，没有声明提升
- let声明的全局变量，不会挂载在window上。

- let声明的循环变量，会在每一个函数中保持循环时的值。

const常量

- 它的前四个特点与let一致。不能作为循环变量。
- 命名特点：通常全部大写，多个单词之间使用下划线去分割。
- 定义时，必须要给定初始值，定义之后，就再也不能使用等号去修改内容。
- 通常常量是用来保存值类型的数据。但如果保存了引用类型数据，可以通过点语法和方括号语法添加属性，访问属性。

定义对象的简化写法

1、ES6中，当对象的属性名和属性值一样时，可以只写一个。
2、定义对象时属性名部分可以使用[]语法。此时方括号创建了一个JS执行的环境，注意一定要用方括号全包起来
eg：var obj = {
[“a”+”b”]:”123”
}
3、定义方法时
var obj = {
say(){console.log(“我是obj的方法”)}
}

解构

ES6中的解构对象

```
var {username, age, sex, width} = xiaoliu;
   console.log(username, age, sex, width);
   // 语法规则： var | let {变量1, 变量2, 变量3, …… 变量4} = 对象
   // var 和 let 是声明的关键字 {} 在这里只表示解构的目标是对象的语法 不是块级作用域 里面不能写语句、表达式
   // 变量与对象的属性名要保持一致
```

解构数组

```
var arr = [0, 1, 2, 3, 4]
  var [a, b, c, d, e, f] = arr;
  console.log(a, b, c, d, e, f);
```

```
3.
```

```
// 层级要对应
var arr = [[0, 1], 2, 3, 4, [[[5, 6, 7]]]];
var [[a, b], c, d, e, [[[f, g, h]]]] = arr;
console.log(a, b, c, d, e, f, g, h);
```

ES6新增的多行字符串

```
1.方法: var str = `
2.     <div>${a}</div>
3.     <div>${b}</div>
4.`
```

&{}是一个固定语法，只能用在多行字符串中``,{}内是一个JS执行环境，可以调用任何JS的方法。

ES6中新增了一种函数的新的定义方式 箭头函数。

匿名定义() => {}
()表示形参列表
=>箭头
{}函数体。
表达式定义。
var arrow = () =>{
console.log(“我是一个箭头函数”)
}

与其他函数的区别

- 只有函数表达式，没有函数声明
- 函数中的arguments不见了，被移除了。可以使用ES6中的拓展运算符…来收集参数。

```
// var assign = (target, ...arr) => {
//     // arguments在箭头函数中已经没有了 使用...运算符代替
//     // arr是一个数组，里面装的是剩余的所有参数
//     arr.forEach(function(value) {
//         for (var i in value) {
//             target[i] = value[i];
//         }
//     });
// }
// assign(obj, obj1, obj3, obj2);
```

- 箭头函数中的this，一旦确定，将再也不会发成变化。箭头函数的this遵守的规则：定义箭头函数时所在的作用域中的this
- 箭头函数不能当做构造函数使用，会报错。因为ES6不允许，因为ES6中实现了class关键字

箭头函数的省略写法。
1、当形参有且只有一个是，可以省略()
2、当有函数体有且只有一条代码时，并且还是返回值时，可以省略{}和return

函数的参数默认值

注意：这不是箭头函数特有的，而是ES6对所有函数增加的特点。
函数的参数默认值要定义在形参后面，通过赋值运算符设置。如果函数调用的时候，传递了对应参数，就是用传的值，如果没有传递，就使用默认值。

```
function fun(a,b=0){
   return a+b;
}
var result = fun(1)
console.log(result);//result=1;
```

事件队列

JS有一个事件队列机制。我们的JS是单线程，也就意味着同一时间只能够做一件事情。
JS在执行代码的时候有两个区域，一个是执行栈，一个是队列（分宏任务队列，微任务队列，现在只考虑宏任务队列）浏览器在加载页面过程中，要执行代码，此时会遇到一个一个的script标签，每遇见一个script标签都会立即执行它，再执行下一个。执行的时候会把所有代码放入执行栈。一条一条执行。在执行过程中，如果遇到了异步代码，此时会执行掉它（将函数交给浏览器），继续往下执行，会先把执行栈内的代码执行了（执行之后处于空闲

状态），才会去任务队列中查看是否有后续的代码需要执行。在这个过程中，如果定时器的时间到，浏览器会将函数交给队列，在队列中排队，不直接放入执行栈。
不仅仅只有setTimeout和setInterval 会产生异步代码，事件处理函数也会。

拓展运算符

…拓展运算符必须出现在最后，且它是一个收集参数的**数组**

- 有三个作用
  - 1、在函数中收集参数

```
/ function fun(a, ...b) {
/     console.log(a, b);
/ }
/ fun(1, 2, 3)
/ fun(1, 2, 3, 4)
```

  - 2、解构数组时可以使用

```
var arr = [1, 2, 3, 4, 5];
var [a, b, ...c] = arr;
```

  - 3、传递参数时

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 11];
// Math.max.apply(null, arr);
// 以上写法等价于下面代码
// Math.max(1, 2, 3, 4, 5, 6, 7, 8, 9, 11);

// 现在有了...语法 就可以
Math.max(...arr);
```

新增的数据类型—Symbol

Symbol是ES6中新增的数据类型，该数据类型属于值类型，它表示独一无二的符号。定义方式为：

```
1 var a = Symbol();
2 console.log(typeof a);    此时a的数据类型为Symbol
```

```
// 定义一个Symbol数据的方式是通过 Symbol()
var a = Symbol("woshiyigea");
// console.log(typeof a);
var b = Symbol("woshiyigea");
// console.log(a === b);
// 该数据类型主要用于解决对象的属性不能重复的问题
var obj = {
    c: 1
}


/* 设置为属性 */
obj[a] = "123";
obj[b] = "321";

// console.log(obj[a])
// console.log(obj[b])


/* symbol属性默认不可遍历 */
for (var i in obj) {
    console.log(i)
}
```

该数据类型主要用于解决对象的属性不能重复的问题。symbol属性默认不可遍历

Proxy类

它可以对指定对象的访问操作和修改操作进行拦截。

```
var star = {
    name: "蔡徐坤",
    age: 42,
    sex: '男',
    hobby: '打篮球'
}

var proxy = new Proxy(star, {
    get: function (star, propName) {
        // console.log(propName)
        if (propName === "age") {
            return 22;
        }
        return star[propName];
    },
    set: function (star, propName, propValue) {
        // console.log(arguments)
        console.log("想要修改属性,得经过我");
        if (propName === "hobby") {
            return;
        }
        star[propName] = propValue;
    }
});
```

通过代理类生成一个star的代理对象，之后通过代理对象进行目标对象的属性访问，会先出发get方法，得到的内容以get函数的返回值为准。通过代理对象进行目标对象的属性修改，会先出发set方法，set方法如果允许修改才修改成功。如果不允许，将不会修改。

set是ES6中新增的数据结构

set是ES6中新增的数据结构，他可以认为是一个不可重复的数组。

```
var arr = [1, 1, 2, 2, 3, 4, 5, 5]
var set = new Set(arr);
console.log(...set);
```

此时输出为1,2,3,4,5。注意，该结构和数组还有不同，不是下标对应成员，而是成员对应成员。

也可以用来数组去重

```
// 数组去重的方法之一
var arr1 = [...new Set(arr)];
console.log(arr1);
```

- Set对象具备方法。
  - set.add() add方法用于添加一个新的成员到集合中，但是不能是已经存在的成员。
  - set.delete() delete方法用于删除一个成员
  - set.has() 查询成员有没有某项，返回布尔值
  - set.clear() 清空成员
  - set.forEach()
    -
```
set.forEach(function(value, index, set) {
    console.log(value, index, set);
});
```

Map 超对象

Map可以认为是一个超对象，普通对象的key必须是一个字符串，value可以是任何内容。
Map 的key可以是任何内容，value可以是任何内容。

- map下的方法
  - set() 向map超对象添加成员,第一个参数为key，第二个为value
  - get() 访问成员，接受一个参数，参数是key
  - delete() 删除成员，接受一个参数，参数是key
  - forEach(function(value,index,map){})
  - claer() 清除所有成员
  - has() 查找是否有该成员

面向对象

不成文的规定：在定义函数时，向定义的是普通函数时，首字母不要不要大写。如果要定义的是构造函数，首字母要大写。

构造函数

构造函数与普通函数的区别
1、构造函 因为它想要去构造成一个对象。构造函数就是普通函数的一种，调用方式不一样而已，默认会返回个对象，所以构造函数中，不要出现return。
2、构造函数原则上不允许出现return。如果出现了分两种情况。1：返回值是值类型，返回值无效。2：返回值是引用类型，会生效，new出来的对象会被返回值中的引用类型覆盖了。

> 构造函数指的是用来创建实例的函数,是一种规则。构造函数是指用来创建对象的函数。

> 实例指的是调用完构造函数之后得到的那个对象
> 当调用构造函数的时候，会有：1、在内存中开辟一个新空间 2、将函数中的this与该空间绑定。3、执行函数体中的代码 4、返回这个空间的地址

`箭头函数不能当做构造函数使用，会报错。因为ES6不允许，因为ES6中实现了class关键字`

原型

原型:prototype
原型是每一个函数都有的属性，对于普通的函数来说该属性没有作用，对于构造函数来说，该属性用来复用方法。
原型的特点：构造函数上的每一个实例都可以调用原型上的内容
原型存在的目的就是让构造函数的每一个实例都复用一套方法。一般我们将方法放在原型上。

原型上天生有一个属性，constructor 指向 构造函数。

`for in循环可以循环原型上的内容`
原型赋值有两种方式：
方式1：替换 指的是使用新的对象来替换原来的对象。 `people.prototype = {}`
缺点：会导致原来的原型丢失，原来的原型上的内容就没了，constructor没有了。补救措施：在手动补回一个constructor属性。
有点：写的代码少
方式2：添加 指的是不替换的情况下，给原来的原型添加属性。
`people.prototype.sayHello = function(){}`
优点：不会导致原来的原型丢失。
确定：代码稍长

ES6中的类

`ES5之前的类叫做其实就是构造函数，但是ES6开始后，我们可以通过class关键字，创建类。`

```
// ES5及之前的类(其实叫做构造函数)
 function Animal1() {
 }

// ES6中的类
 class Animal2 {
 }
 var animal1 = new Animal1();
```

```
var animal2 = new Animal2();
console.log(animal1)
console.log(animal2)
```

类的属性

ES6中类定义属性，要定义在constructor函数内，该函数是每一个类都有的，这就是每个类的构造函数。

```
// ES5及之前的类(其实叫做构造函数)
function Animal1(name, age, sex) {
    this.name = name;
    this.age = age;
    this.sex = sex;
}

// ES6中的类
class Animal2 {
    /* ES6中定义属性，要定义在constructor函数内，该函数是每一个类都有的，这就是每个类的构造函数 *
    constructor(name, age, sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }
}
var animal1 = new Animal1("小白", 11, "雌");
var animal2 = new Animal2("小黑", 22, "雄");
console.log(animal1)
console.log(animal2)
```

类中定义方法

ES6中的类定义方法要写在类体中

```
class Animal2 {
    /* ES6中定义属性，要定义在constructor函数内，该函数是每-
    constructor(name, age, sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }
    /* ES6中的类定义方法要写在类体中 */
    sayHello() {
        console.log("我是" + this.name);
    }
}
```

类中定义存取器

```
class Dog {
    constructor(name, type) {
        this.name = name;
        this.type = type;
    }
    /* 定义取器 */
    get getName() {
        console.log("我是获取name的存取器方法");
        return this.name;
    }
    /* 定义存器 */
    set setName(value) {
        console.log("我是设置name的存取器方法");
        this.name = value;
        dogName.innerHTML = this.name;
    }
}
```

正则表达式

正则表达式是由特殊字符组成的一组规则表达式.用于检测字符串是否否he某种规则。
正则的定义：
方式一： 字面量定义 /正则表达式/标识符
方式二: 构造函数定义。 new RegExp(表达式，修饰符)

```
var str = "23";
var str1 = "13";
var reg = /\d\d/;
console.log(reg.test(str));
```

规则：
特殊字符：

- \s表示所有的空白符。 \S表示所有的非空格
- \d表示有所的数字 \D表示所有的非数字
- \w 表示数字，字母，下划线 \W表示除了数字，字母，下划线之外。
- \b 表示单词分界。

元字符：

- . 除了回车和换行所有的字符。
- () 表示分组 (com|cn|org)表示匹配com或者cn或org
- [] 表示范围 [a-d]匹配abcd中的任何一个字符。
- {} 表示数量

- | 表示或者
- ^ 在正则的最左侧出现表示开头，在[]中出现表示取反。
- $ 在正则的最末尾出现，表示结尾
- ？ 表示0个或1个
- + 表示一个或者多个（至少一个）
  - {m} 固定长度
  - {m,n} 最少m个，最多不限制
  - {m,}最少m个，最多不限制。
- * 表示0个或多个（任意个）
- {}有三种方法。

正则表达式中，一个字符和一个特殊字符只能匹配字符串中的一个字符。
{}是数量规则，属于正则表达式的元字符，作用是规定数量，只对它前面的一个规则生效。
每一个特殊字符或者普通字符都是一个，如果被()包裹，则被()包裹起来的规则作为一个。

标识符（修饰符）
标识符有g，m，i

- g 全局
- m 多行匹配
- i 忽略大小写

replace方法与正则结合实现过滤关键字
方法一：

```
var str = "asdfasdfSMasdfasdfsmasdfSmsadfsMwe"
   var reg = /sm/gi
   var str1 = str.replace(reg,"**")
   console.log(str1)
```

方法二：

```
var str = "absfhc123dxe"
var reg = /\w/g
var obj = {
    a:"北",
    b:"京",
    c:"欢",
    d:"迎",
    e:"你"
}
var str1 =  str.replace(reg,function(match,index,str){
        console.log(match,index,str)
        if(obj[match]){
            return obj[match]
        }
        else{
            return match;
        }

})
console.log(str1)  北京sfh欢123迎x你
```

方法三：

```
//将一段字符串替换成驼峰命名法,并且将 - 后的一个或两个字母替进行大写
var str = "border-left,border-right,border-top,border-b"
var reg = /-(\w\w?)/gi
var str1 = str.replace(reg,function(match,$1,index,str){
    console.log(match,$1,index,str)
     return $1.toUpperCase()
})
console.log(str1)
```

match方法与正则结合使用进行查找

```
// match方法与正则结合使用进行查找
var str = "ASDFsdfASDsGFDGxcvSsGSs"
var str1 = str.match(/[a-z]/g)
console.log(str1)  // ["s", "d", "f", "s", "x", "c", "v", "s", "s"]
```

match方法配合正则会找到所有符合规则的成员并返回

search方法与正则结合使用进行查找

```
//search方法与正则结合使用进行查找
var str = "ASDFsdfASDsGFDGxcvSsGSs"
var str1 = str.search(/[a-z]/g)
console.log(str1)  //4
```

search方法配合正则只会找到第一个符合规则的成员的下标并返回

正则方法

> exec()是正则表达式中的方法，用于寻找字符串中对应的正则匹配到的字符串的位置，匹配的结果是一个数组,与字符串match方法类似。匹配不到返回null
> test() 检索字符串中指定的值。返回 true 或 false
> toString() 返回正则表达式规则的字符串。

```
let reg = /\d/g;
reg.toString() // 输出 /\d/g;
```

正则表达式中，表示特殊含义的：() [] {} | ^ $. * /
如果想要在正则表达式中，表示元字符本身的含义，需要用转义字符。

```
var reg = /^{+$/; // 该正则的含义：  以{开头 以{结尾 中间都是连续的{ 数量必须至少一个
```

```
//用正则表达式规定用户输入qq号。开头必须是数字，规定数字为5——11位之内，必须以com结尾
var reg = /^\d{5,11}@{1}(qq)\.{1}(com)$/
var str = "1050462990@qq.com"
reg.test(str)  结果为真
```

安全类

```
var Cat = function(){
    if(this == window){  //当this指向是window时候说明没new
        return new Cat()
    }
    this.name = "可可"
}
var cat = new Cat()
var cat1 = Cat()
console.log(cat)
console.log(cat1)
```

安全类的思路：如果外部调用是通过new调用 则正常执行 如果外部调用不是通过new调用 当前函数就是一个普通函数调用 没有返回值 则主动返回一个实例

instanceof

判定该对象是不是构造函数的实例
ES6中提供了一个关键字，instanceof
用法： 对象 instanceof 构造函数
返回一个布尔值。

instanceof的检测机制是检测原型链 只要在原型链上有 就为真

```
console.log([] instanceof Array); // true
console.log({} instanceof Object); // true
console.log(function() {} instanceof Function); // true

console.log(1 instanceof Number); // false
console.log("" instanceof String); // false
```

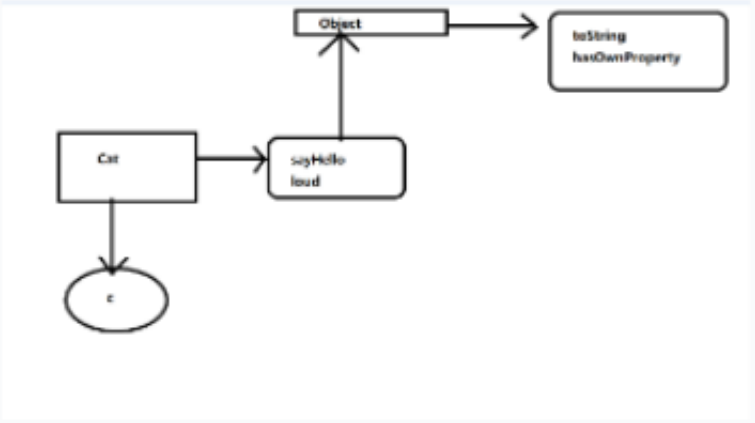
hasOwnProperty

hasOwnProperty是ES5中新增的一个对象方法，用来判定属性是否是自身的属性。

```
function Cat() {
    this.name = "喵咪";
    this.age = 13;
    this.color = "blue";
}
Cat.prototype.sayHello = function() {
    console.log("hello");
}
Cat.prototype.loud = function() {
    console.log("我们一起学猫叫");
}
var c = new Cat();
console.log(c);

for (var i in c) {
   console.log("c有没有" + i + "这个属性呢？" +  (c.hasOwnProperty(i) ? "有" : "没有") )
}
// for in循环可以循环原型上的内容
// 有些情况下我们不希望循环原型上的内容，所以需要判定
// hasOwnProperty方法 可以判定属性是否是自身属性
console.log(Cat.prototype);  //原型下的方法不是自己的属性
```

原型链

如果我们定义一个构造函数并初始化时，将得到一个对象。（任何函数都有一个prototype属性，值是对象，是object实例），当访问该对象时，如果该对象自身没有该属性，将会找构造函数的原型（找object的实施），这种机制叫做原型查找。等于我们访问了object的实例。又会触发新的查找。如果该对象上没有方法，将会查找 Object.prototype。这种连环查找就叫做原型链查找。当在某一环查找到时，将会停止。最终会找到Object.prototype。这里就是原型链的终点，如果这里还找不到，返回undifined



原型链关系

1、所有对象的构造函数是Object
2、所有函数的构造函数是Function
3、函数也是对象

```
function Dog() {

}
var d = new Dog();
/* 此时有三个内容:
    d 是Dog的实例
    Dog是d的构造函数
    Dog.prototype 是 Dog的原型
    Dog.prototype 是 d的原型对象
    d.__proto__ 是Dog.prototype
    注:  __proto__ 是浏览器添加的属性 为了访问方便
    不是ECMAScript的范围内 所以尽量不要使用 要用就用Dog.prototype
*/
```

继承

继承是父类与子类之间的一种关系。主要目的是让自雷拥有父类的属性的方法的过程。ES6之前使用原型来模拟继承。

```
function People(name, age, sex) {
    this.name = name;
    this.age = age;
    this.sex = sex;
}
People.prototype.run = function() {
    console.log("跑")
}
var p = new People("小红", 12, "女");
p.run();
p.cry();
function Student(name, age, sex, grade) {
}
// Student.prototype = People.prototype;
// 直接将父类和子类的原型统一 不好，因为子类添加方法的时候会影响到父类
// 问题：什么东西能够访问父类的原型，又在改变自身的时候不会影响到父类的原型
// 答案：父类的实例，既能够访问到父类的原型，又在改变自身的时候不影响父类的原型。
Student.prototype = new People();
Student.prototype.constructor = Student;  //将构造器补回来
Student.prototype.doHomeWork = function() {
    console.log("做作业");
}
// p.doHomeWork(); //访问不到了，因为p的原型链中没有doHomeWork方法。
var s = new Student("小明", 13, "男", 6);
s.run();
s.cry();
s.doHomeWork();
// 现在继承了方法
```

以上为继承方法。

```
function People(name, age, sex) {
    this.name = name;
    this.age = age;
    this.sex = sex;
}
function Student(name, age, sex, grade) {
    // 这就是继承属性了
    People.call(this, name, age, sex);
    this.grade = grade;
}

var s = new Student("小明", 13, "男", 6);
var p = new People();

console.log(s instanceof Student); // true
console.log(s instanceof People); // false
```

以上为继承属性。

继承方法的原理是改变了原型，叫做原型式继承。继承属性叫做构造函数式继承。组合起来就是第三种继承方法：组合式继承。

构造函数式继承只是为了复用代码，原型继承才是真的继承。

寄生式继承

```
/* 父类 */
function People(name, age, sex) {
    this.name = name;
    this.age = age;
    this.sex = sex;
}
People.prototype.cry = function () {
    console.log("哭")
}
/* 子类 */
function Student(name, age, sex, grade)
{
    // 构造函数式继承 只是为了复用代码
    People.call(this, name, age, sex);
    this.grade = grade;
}
// 原型式继承在继承的时候因为指向的是父类的实例 而父类的实例身上是有属性的 会导致有很多无用的属性出现在子类的原型上
// 所以想办法去掉这些属性
// 方式1: 一条一条的删除，但是太麻烦了 案例也不做了
// 方式2: 换一个构造函数
function F() {}
F.prototype = People.prototype;
Student.prototype = new F();
Student.prototype.constructor = Student;
Student.prototype.learn = function() {
```

```
25.     console.log("学习");
26.   }
27.
28.   var s = new Student("小明", 12, "男", 5);
29.
30.   // 检测s是否是Student的实例
31.   console.log(s instanceof Student);  //true
32.   // 检测s是否是People的实例
33.   console.log(s instanceof People);   //true
```

- 寄生式函数

```
/* 寄生式继承函数 */
function extend(SuperClass, SubClass) {
    var F = function() {}
    F.prototype = SuperClass.prototype;
    SubClass.prototype = new F();
    SubClass.prototype.constructor = SubClass;
}
```

- ES6中的继承

```
// ES6定义子类
class HSQ extends Dog {
    constructor(name, age, sex) {
        // super表示超类、也就是父类的构造函数
        super(name, age);
        this.sex = sex;
    }
}
```

关键字是extends。他会继承所有的东西，包括属性，方法，存取器。

JS第二月

服务器

简单来说服务器就是用来提供服务器的一台计算机。该计算机与普通计算机没什么本质区

别，性能，安全性有所提高。

服务器也分软件服务器和硬件服务器。
硬件服务器指的就是看的见摸得着的计算机。
软件服务器指的就是服务器程序。
它们运行在硬件服务器上。

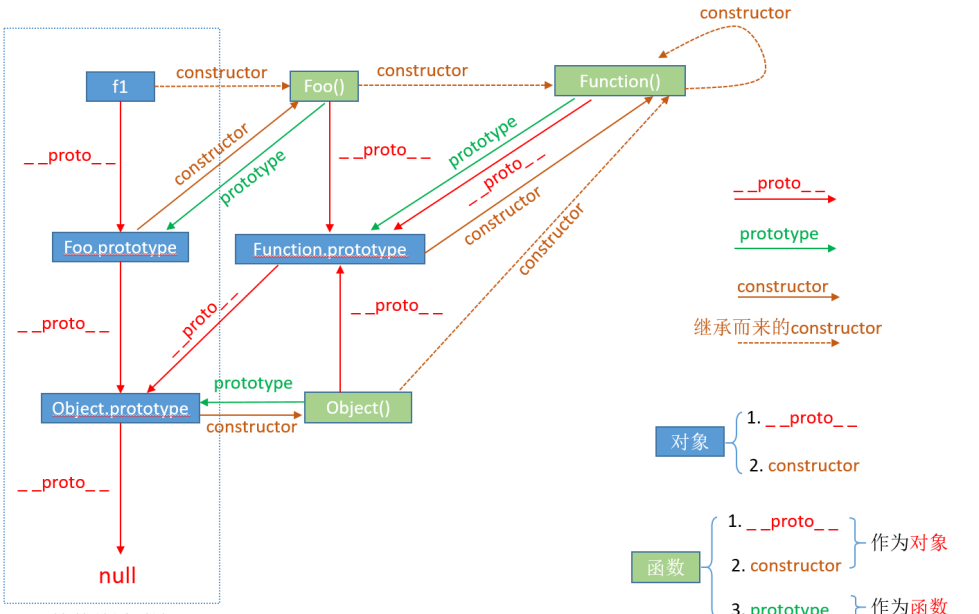
浏览器。

与服务器队形，也分软硬件。硬件指的就是计算机。软件指的是一阶段学习过的各种浏览器：chrome，firefox，ie，opera，safari

__proto__,constructor,prototype

1、__proto__和constructor属性是对象所独有的
2、prototype属性是函数独有的，因为函数也是一种对象，所以函数也有__proto__和constructor属性；
3、**proto**属性的作用就是当访问一个对象的属性时，如果该对象内部不存在这个属性，那么就会去它的**proto**属性所指向的那个对象（父对象）里找，一直找，直到**proto**属性的终点null，再往上找就相当于在null上取值，会报错。通过**proto**属性将对象连接起来的这条链路即我们所谓的原型链。
4、prototype属性的作用就是让该函数所实例化的对象们都可以找到公用的属性和方法，即f1.**proto** === Foo.prototype。
5、constructor属性的含义就是指向该对象的构造函数，所有函数（此时看成对象了）最终的构造函数都指向Function。
6、__proto__是对象独有的，它都是由一个对象指向一个对象，即指向他们的原型对象（也可以理解为父对象）。
7、__prototype__是函数独有的，它是从一个函数指向一个对象，它的含义是函数的原型对象。它的作用就是包含可以由特定类型的所有实例共享的属性和方法，也就是让该函数所实例化的对象们都可以找到公用的属性和方法。任何函数在创建的时候，其实会默认同时创建该函数的prototype对象。

```
function Foo() { ...};
let f1 = new Foo();
```



这就是原型链