**对象：**

**创建一个新对象**

**Var obj = new Object()**

**Var obj={name=”” , age=18 , ….} ; //注意逗号，最后不加。属性段独占一行**

**Var obj={name=”” , age=18 ,text{这里面也可以创建对象作为他的属性} ….}**

**里面的属性名是字符串**

**属性：**在对象保存的值

**添加属性**：对象.属性名=属性值（obj.name=”wxx”）

**读取对象属性**：对象.属性名（obj.name）console.log(obj.name)

如果对象没有这个属性，就会返回undefined

**修改对象属性**：对象.属性名=新值（obj.name=”qlb”）

**删除一个属性**：delect 对象.属性名（delect obj.name）

对象名字任何都可以

如果要使用特殊的属性名，不能采用.的方式来操作

需要使用另一种方式：

**语法：对象["属性名"] = 属性值（obj["nihao"] = "你好";）**

读取时也需要采用这种方式

允许接收特殊名字

不知道属性名是什么的时候可以用

**in 运算符** \* 通过该运算符可以检查一个对象中是否含有指定的属性  
 \* 如果有则返回true，没有则返回false  
 \* **语法：**  
 \* "属性名" in 对象  
 \*/  
//检查obj中是否含有test2属性//console.log("test2" in obj);

**for in**

语法：for….in

for(var key in obj){

console.log(key); **obj函数里面有多少个属性，里面就循环多少次，依次打印出属性名字,不是值**

console.log(wukong.[key]);**//依次打出属性值。只能用[ ],不能用.原点，**

console.log(key +’:’+ wukong.[key]));**//这个打印就是属性段属性名：属性值**

}

**基本数据类型和引用类型的区别**

|  |  |  |
| --- | --- | --- |
|  | 基本类型 | 引用类型（对象） |
| 存储方式 | 基本数据类型的值直接在栈内存中存储 | 保存到堆内存  是否相同时只管地址 不管内容，==与===都是比较两个对象是地址 |
| 值与值之间的影响 | 值与值之间是独立存在，修改一个变量不会影响其他的变量  **如：**  var a = 123; var b = a; a++; | 每创建一个新的对象，就会在堆内存中开辟出一个新的空间，而变量保存的是对象的内存地址（对象的引用），如果两个变量保存的是同一个对象引用，当一个通过一个变量修改属性时，另一个也会受到影响  **如：**  var obj = new Object(); obj.name = "孙悟空"; var obj2 = obj; //修改obj的name属性 obj.name = "猪八戒";  那两个都会变，因为他们的地址是一样的，值也是一样的 |
|  | 当比较两个基本数据类型的值时，就是比较值。  **如：**  var c = 10; var d = 10; console.log(c == d);  返回true | 而比较两个引用数据类型时，它是比较的对象的内存地址，  如果两个对象是一摸一样的，但是地址不同，它也会返回false  **如：**  var obj3 = new Object(); var obj4 = new Object(); obj3.name = "沙和尚"; obj4.name = "沙和尚";  两个值虽然相同但这是两个对象的值，就不一样了 |

**函数 function**

- 函数也是一个对象  
 - 函数中可以封装一些功能（代码），在需要时可以执行这些功能（代码）

- 函数中可以保存一些代码在需要的时候调用  
 - 使用typeof检查一个函数对象时，会返回function

**函数包括四大部分：**

* 函数的声明
* 函数的参数列表
* 函数的返回值 没有写都会返回undefined
* 函数的逻辑要求表达

**创建一个函数：**使用 函数声明 （函数命名要用小驼峰命名法）  
 语法：  
 function 函数名([形参1,形参2...形参N]){  
 语句...  
 }

function name(形参，形参){

**调用函数**

语法： 函数对象()

【name();】

**形式参数：**

可以在函数的()中来指定一个或多个形参（**形式参数**）  
多个形参之间使用,隔开，声明形参就相当于在函数内部声明了对应的变量 但是并不赋值  
如：  
function sum(a,b){  
 console.log("a = "+a);  
 console.log("b = "+b);  
 console.log(a+b);  
}

* 调用函数时解析器不会检查实参的类型,
* 所以要注意，是否有可能会接收到非法的参数，如果有可能则需要对参数进行类型的检查  
  函数的实参可以是任意的数据类型  
  //sum(123,"hello");  
  //sum(true , false);
* 多余实参不会被赋值，不会检查实参的数量  
   如果实参的数量少于形参的数量，则没有对应实参的形参将是undefined  
  //sum(123,456,"hello",true,null);  
  sum(123);

**return** 可以使用 return 来设置函数的返回值  
 语法：  
 return 值  
return后的值将会会作为函数的执行结果返回，  
如果return语句后不跟任何值就相当于返回一个undefine

如果函数中不写return，则也会返回undefined  
return后可以跟任意类型的值**eg：**  
function sum(a , b , c){  
 var d = a + b + c;  
 return d;}吧  
**调用函数：**  
变量result的值就是函数的执行结果  
函数返回什么result的值就是什么  
var result = sum(4,7,8);//用中间的值  
console.log("result = "+result);

实参可以是任意的数据类型，也可以是一个对象  
当我们的参数过多时，可以将参数封装到一个对象中，然后通过对象传递

**eg:**

function ren(o)  
 {alert('我是'+o.name+'今年'+o.age+'了') }

**//定义函数内容，用o.来定义函数的位置,方便** var obj = {name:'wxx', age:'148'};**//定义一个对象的属性**  
 ren(obj);**//在函数里面放入对象的属性并执行函数**

**立即执行函数**  
\* 函数定义完，立即被调用，这种函数叫做立即执行函数  
\* 立即执行函数往往只会执行一次

**eg：**

(function(a,b){  
 console.log("a = "+a);  
 console.log("b = "+b);  
})(123,456);//定义完马上在后面用

自执行函数;IIFE

(function(a,b,){

})(123,456)

**方法**：（就是函数作为了属性）

\* 函 数也可以称为对象的属性，  
\* 如果一个函数作为一个对象的属性保存，  
\* 那么我们称这个函数是这个对象的方法  
\* 调用这个函数就说调用对象的方法（method）  
\* 但是它只是名称上的区别没有其他的区别

作用域：

全局作用域:

直接在script标签中的JS代码，在全局作用域

全局作用域是在页面打开时创建的，在页面关闭时销毁

在全局作用域中有一个全局对象window

**在全局作用域中，**

创建的变量都会作为window对象的属性保存

创建的函数都会作为winow对象的方法

function fun

**变量的声明提前**。

使用var关键字声明的变量，会在所有代码执行之前被声明

**是提升 var num，不是提升nun=0，所以等于没有初始化**

但是如果声明变量时不适用var关键字，则变量不会被声明提前

**函数的声明提前**

* 使用function 函数（）{ }可提升也可以呼出、、、

它会在所有代码执行之前就创建，所以可以在函数.声明前调用函数

* 使用函数表达式创建函数 var fun2=function（）{}不可先调

不会被声明提前，所以不能在声明之前就调用，提升var fun2

全局作用域中的变量都是全局变量，在页面任意位置都能调用到

**函数作用域:**

* 调用函数时创建函数作用域。函数执行完毕以后销毁
* 每调用一次函数就会调用一次新的函数函数作用域
* 现在自身函数找，找不到 往上级找，知道找到全局作用域
* 想访问全局变量可使用window.来调用 window.a
* 在函数作用域中，使用var都会在所有代码执行时声明
* 函数声明也一样
  + 没有使用var变量 就会变成全局变量

**例子1：**

var a;//没有初始化

console.log(a); //所以返回值是undefined

a = 9;

console.log(a);//上面既定义了，又赋值了

**例子2：**

console.log(a);//使用的时候再上面找不到a所以undefined

var a = 9;

**例子3：**

**var fun2 = function(){console.log("我是 fun2 函数");}; fun2();**

**可直接调用**

**例子4：**

**fun2();**

**var fun2 = function(){console.log("我是 fun2 函数");};**

**这个会返回undefined，因为函数提升是提升var fun2**

**this和arguments**

调用函数时，浏览器每次都会传递进两个隐含的参数

第一个是this 另外一个是arguments

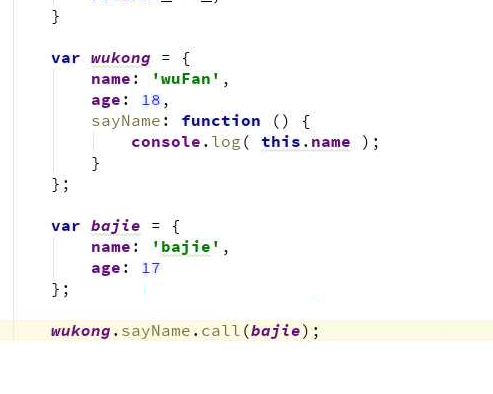
arguments

**this：**

指向函数的调用者，没有指向就是window，同一功能用一个函数就好。

//在调用函数时。浏览器每次都会传入两个隐含的参数

**arguments：调用函数时实参的数，成一个数组**  
 //第一个函数的上下文对象是this  
 //第二个是 封装实参的对象arguments  
 //arguments是一盒类数组的对象，它可以引用索引来操作数据  
 //获得长度，在调用函数时，我们传入的实参是在argument中保存的  
 //即使不定义形参，也可以通过arguments来调用实参  
 // argumenta[0]第一个参数  
 // callee对应的是当前正在执行的函数对象  
 function fun() {  
 console.log(arguments.callee);**//arguments长度是实参的数量**  
 } **// argumenta[0]第一个参数**  
 fun( 'hello','ture'); **// callee对应返回的是当前正在执行的整个 函数对象**



**工厂方法创建对象**

//工厂方法创建对象  
function people(name,age,grender) {  
 **//创建新的对象** var obj= new Object();  
  **//添加属性和方法** obj.name = name;  
 obj.age = age;  
 obj.grender = grender;  
 obj.sayName =function () {  
 alter('hhh');  
 };  
 return obj**;//一定要记得返回值**}  
var obj1 = people('ss',18,'男');**//创建一个新的对象**  
console.log(obj1);**//打印对象**

缺点：创建的对象都是object，导致无法区分

**构造函数** 不同的是构造函数习惯上大驼峰**首字母大写**  
 构造函数和普通函数的区别就是调用方式的不同  
 普通函数是直接调用，而构造函数需要使用**new关键字来调**

**构造函数的执行流程：**  
 1.立刻创建一个新的对象  
 2.将新建的对象设置为函数中this,在构造函数中可以使用this来引用新建的对象  
 3.逐行执行函数中的代码  
 4.将新建的对象作为返回值返回

* 使用同一个构造函数创建的对象，我们称为一类对象，也将一个构造函数称为一个类。
* 我们将通过一个构造函数创建的对象，称为是该类的实例  
  **this的情况：**  
   1.当以函数的形式调用时，this是window  
   2.当以方法的形式调用时，谁调用方法this就是谁  
   3.当以构造函数的形式调用时，this就是新创建的那个对象

function person(name,age) {  
 this.name = name;**//this表示这个函数** this.age = age;  
 this.sayName=function Name() {  
 alert('dd')  
 }}**//这里要用new来调用**  
 var per = new person('sen',18);**//这里的per是person的事例** console.log(per);

**instanceof**

使用instanceof可以检查一个**对象**是否是**一个类**的实例  
 **语法：**  
 对象 instanceof 构造函数  
 如果是，则返回true，否则返回false

**console.log(per instanceof Person)**

**原型对象**：

我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype  
\* 这个属性对应着一个对象，这个对象就是我们所谓的**原型对象**  
\* 如果函数作为普通函数调用prototype没有任何作用  
\* 当函数以**构造函数**的形式调用时，它所创建的对象中都会有一个隐含的属性，  
\* 指向该构造函数的原型对象，我们可以通过**\_\_proto\_\_**来访问该属性

原型对象就相当于一个公共的区域，所有同一个类的实例都可以访问到这个原型对象，  
\* 我们可以将对象中共有的内容，统一设置到原型对象中。

\* 以后我们创建构造函数时，可以将这些对象共有的属性和方法，统一添加到构造函数的原型对象中，  
\* 这样不用分别为每一个对象添加，也不会影响到全局作用域，就可以使每个对象都具有这些属性和方法了

**//向MyClass的原型中添加属性a**MyClass.prototype.a = 123;

**//向MyClass的原型中添加一个方法**  
MyClass.prototype.sayHello = function(){  
 alert("hello");  
};

**in**

用in检查对象中是否含有某个属性时，如果对象中没有但是原型中 ，也会返回true console.log("name" in mc);

**hasOwnProperty**  
 使用对象的hasOwnProperty()来检查对象**自身**中是否含有该属性  
 该方法只有当对象**自身中含有属性时，才会返回true** console.log(mc.hasOwnProperty("age"));  
 console.log(mc.hasOwnProperty("hasOwnProperty"));

**原型对象也是对象，所以它也有原型，** 用一个对象的属性或方法时，会现在自身中寻找，  
 自身中如果有，则直接使用，  
 如果没有则去原型对象中寻找，如果原型对象中有，则使用，  
 如果没有则去原型的原型中寻找,直到找到Object对象的原型， Object对象的原型没有原型，如果在Object原型中依然没有找到，则返回undefined

存储一些共享数据

检查属性是否存在

方法两种：

in关键词

hasOwnProperty关键词

instanceof 检查这个对象是不是这个类的实例

**Date 对象**

在js中使用Date来表示一个时间

var d = new Date();

console

**创建一个Date对像**

如果直接用构造函数构造一个Date对象，则会封装当前代码执行的时间var d = new Date();

**创建一个使指定的时间对象**

需要在构造函数中传递一个表示时间的字符串作为参数

var d2 = new Date(‘12/03/2016 11:10:30’);

console.log

**方法：**

**getdate()**;获取当前日期对象是本月的几日  
**getDay()**;获取一周中的周几，0表示周日  
**getMonth()**获取月份，0是一月，11是12月

**getFullYear()**获取年  
var d4 = new Date();  
var date4 = d.getFullYear();  
alert('day'+date4 );  
//获取当前对象的时间戳，时间戳从格林威治标准时间从1970年1月1日

开始到现 在的毫秒数开始  
//到当前日期所花费非毫秒数（1秒=1000毫秒）

**gettime的应用：**

计算两年的时间差=后面与1970年和开始于1970年的差

var gettimes = prompt("请输入你要查询的日期：");//输入用户要查询的世界  
var d1990= new Date("1/1/1990");//先定义1990年的时间  
var dsj = new Date(gettimes);//定义用户定义的时间到1970年相距的时间  
var day = ( dsj.getTime() - d1990.getTime())/1000/60/60/24;

//用后面的时间-开始的时间，然后再把秒化成天数  
// 在这里相减再化成天数，可消除东八区与格林的时间差的时间。

//获取当前的时间戳  
time=Data.now();

**Math**  
Math是 一个普通对象，不是一个构造函数属于工具类，不用创建对象

封装了数学运算的属性和方法  
**Math.PI**是3.1415926....  
**Math.abs(-1)**可以用来计算一个数的绝对值  
**Math.ceil(1.4)**可以对一个数进行向上取整，只要小数点后面有书就直接是上一个整数  
**Math.floor(1.4)**可以对一个数进行向下取整，去掉小数点  
**Math.round()**进行四舍五入取整数  
**Math.random（）**可以生成0-1之间的随机数，可用for  
 生成0-x的随机数Math.round(Math.random（）\*x就可以0-x 都有  
 生成x-y之间 随机数 Math.round(Math.random（）\*（y-x）+x）  
**max（）**可以获取多个数中的最大值

**min（）**可以获取多个数中的最大值

**String的方法**

//在底层字符串是以字符数组的形式保存的  
**var str = 'hello at ni ';**  
console.log(str[1]);//会出第一个字母  
//**charAt**可以返回字符串中指定位置的字符，根据索引获得指定字符  
 str1 = 'HELLO';  
 var r = str1.charAt(0);//也返回字符串的第一个字母  
 alert('r' + r);  
 r2 = str.**charCodeAt**(6)//返回字符的编码  
 r3 = String.**fromCharCode**(72);//通过String的构造函数调用的，根据字符编码获得字符，写成16进制  
 r4 = str.**indexOf**('h')('h',2);//返回h 所在的的位置号码，也就是索引，没找到返回—1  
 r5 = str.**indexOf('**h',2);//从第二个位置开始查找  
 r6 = str.**lastIndexOf**();//从后往前找  
 r7 = str.**slice**(1);//可以从字符串中截取指定的内容，  
 // 第一个是开始位置，第二个是结束位置的索引  
 //省略第二个参数，就是整串全部要  
 r7 = str.**substrin**g(0,2);//第一个开始位置截取u、索引之前（不包括）  
 //与slice不同的是，不能接收负数传负数则默认使用0；会自动调整参数位置，会自动交换位置，从小到大排  
 r8 = str.**substr**(0,1);//用于截取字符串，第一个参数是开始位置，第二个参数是截取长度  
 r9 = str.**split(“=”)** ;//拆分数组，根据字符串拆分数组，第一个数组就是第一个逗号，按照=分成两边  
 //如果传递一个空串，把单词全部变成一个一个的  
 r10 = str.**toUpperCase()**;//将字符串转换成大写，都是不影响原字符  
 r11 = str.**toLowerCase();**//将字符串转换成小写，不影响原来

**正则表达式**：

用于定义一些字符串的规则，检查一个字符串是否符合规则，获取字符串符合规则的内容提取出来  
  **语法：**  
 var 变量=new RegExp('正则表达式'，'匹配模式');  
 字面量：var 变量=/正则表达式/匹配模式  
 使用构造函数更加灵活  
 使用| 用竖线表示或。  
 使用test()的方法可用来检查一个字符是否符合正则表达式  
 如果是就是true，不是就返回false；  
 严格区分大小写的  
 在构造函数中可传递一个匹配模式作为第二个参数

* **创建一个正则表达式检查一个字符串是否有字母。[]的内容也是或的关系**
* [ab]==a|b;
* [a-z]任意大小写
* [A-Z]任意大写字母
* [A-z]任意字母
* [^ab]除了ab以外的数
* [0-9]任意数字，加^表示除了

**split（）**可以将一个字符串拆分为一个数组  
可以传递一个正则表达式作为参数，就会根据表达式来拆分  
根据任意字母来拆分字符串拆分  
var result = str.split（/[A-z]/）;任意字母  
  **search（）** 可以搜索字符串是否含有指定内容  
 如果搜索到指定内容则返回第一次出现的索引，没有就返回-1  
 它可以接受一个正则表达式作为参数，然后会根据正则表达式去检索字符串 var result = str.search（/[A-z]/）  
  **match（）** 可以根据正则表达式，从一个字符中将符合条件的内容提取出来  
 默认情况下我们match只会找到第一个符合条件的内容，找到就返回  
 我们可以就设置正则表达式为全局匹配模式，这样就会匹配到所有的内容  
 可以为一个正则表达式设置多个匹配模式，且顺序无所谓。  
 match()会将匹配到的内容封装到一个数组中返回，即使只查询到一个结果

str = "1a2a3a4a5e6f7A8B9C";  
result = str.match(/[a-z]/ig);  
//console.log(result[2]);

**replace()**  
 可以将字符串中指定内容替换成新的内容  
 参数：  
 被替换的内容可以接收一个正则表达式作为参数  
 新的内容  
 默认只会替换第一个  
 var r = str.replace(/[a-z]/gi,'000')后面也可以为空，作用是删除第一个参数