一．变量分为原始值和引用值(由值决定类型)

1> 原始值 存在stack(栈)里面, 修改值为值的重新赋值

(1)Number(数字类型)

(2)String(字符类型)

(3)Boolean(true/false)

(4)undefined(没有定义)

(5)null(无意义空 占个位置)

2>引用值 大部分放在heap(堆)里面, 修改值为地址执行不同, 会影响指向相同的值。

(1)array(数组)

(2)Object(对象)

(3)function(方法)

(4).......date regExp

1. 布尔值为false的情况(除了这些boolean都为true)
2. undefined(未定义，找不到值时出现)
3. null(代表空值)
4. false(布尔值的false，字符串"false"布尔值为true)
5. 0(数字0，字符串"0"布尔值为true)
6. NaN(无法计算结果时出现，表示"非数值"；但是typeof NaN==="number")
7. ""(双引号)或''(单引号) (空字符串，中间有空格时也是true)

7>实例 -->

打印出false 或者 使用Boolean(a)测试

三．与( || )或( && )非( ! )问题

1>或( && )从第一个往后依次判断是否位假, 是假则返回其值, 若全为真则返回最后一

个值 -->

打印出的结果为3

1. 与( || )从第一个往后依次判断是否位真, 是真则返回其值, 若全为假则返回最后一个

值 -->

打印出的结果为1

3>非( ! )表达式转化成布尔值取反

注意: !!就是取反再取反, 没有发生改变

四．typeof详细讲解(检验值类型)

1>typeof()返回值的六种类型

(1)number

(2)string

(3)boolean

(4)object(null数组对象都是返回这个)

(5)undefind

(6)function

2>typeof的显示类型转换(注意其大小写)

(1)Number(mix)返回值为mix(数字原样返回) 返回类型为number类型

注意：若为字符返回NaN

(2)parseInt(string, [radix])返回解析string成十进制整数展示 返回类型为数字类型

##详细说明

string是radix为基底的数, 解析成10进制的数

会去string符合radix格式的数, 若后面字符不符合则会取符合字符之前的字符

若所有不符合则会返回NaN

##radix参数说明

--选填, 表示要解析的基数, 该值介于2~36之间

--省略该值则其值为0, 数字以10为基来解析

--以”0x”或”0X”, 则以16为基数

--若改参数小于0或者大于36, parseInt()返回值为NaN -->

parseInt(0010, 2) => NaN(string0开头都会); parsint(“0010”, 2) => 2

(3)parsFloat(str)返回值为str(数字原样返回) 返回类型为number类型

注意：若有字符则去掉字符, 数字原样放回, 注意和Number区分

(4)String(str)返回值为str(原样返回) 返回类型为string类型

(5)str.toString(radix)返回值为str十进制数成radix的数, 返回类型为string类型

##详细说明

若str为数字则会把str当成十进制的数解析成radix为基底的数

若str为字符串则会原样返回

##radix参数说明

--可选。 规定表示数字的基数, 2~36之间取值

--省略参数则使用基数为10

--若参数是10以外的其他值, 则ECMAScript标准允许实现返回任意值 -->

0010.toString(2) => 1000; “0010”.toString(2) => 0010

(6)boolean(str)返回值为str的true/false, 返回类型为boolean类型

3>显示类型转换常用方法

(1)其他转字符串用toString()

(2)字符串转数字用parseInt()

4>隐式类型转化

(1)isNaN(a)a转化成number与NaN进行对比 返回类型为boolean

(2)++/-- +/-(一元正负)转化成Number在相加, 返回其值 返回类型类为number

(3)+ 转化成string在相加, 返回其值 返回类型为string

注意：两边有一个是字符串才转化

(4)-\*/% 转化成Number在相乘, 相减.. 返回类型为number

注意：若有字符串则会返回NaN

(5)&&||！数字优先原则, 返回值为true/false 返回类型为boolean

(6)<><=>= 转化成数字优先原则, 返回值为true/false 返回类型为boolean

(7)== !== 转化数字优先原则, 值为true或者false 返回类型为boolean

数字优先原则(全为数字返回数字)

5>隐式类型转化注意点

(1)undefind和null不大于等于小于零

(2)undefind=null是true,

(3)NaN=NaN是false

(4)===和！==不发生隐示类型转换

1. 函数

1>函数声明

(1)function () {}(匿名函数, 一般立即执行函数这样声明)

(2)function test() {} 执行函数直接test()

注意：函数声明开发规范写明命名函数需是小头封式(theFirstName)

2>函数表达(使用一个变量来接收这个函数)

(1)var test = function test () {} (命名函数表达式) test无意义

(2)var test = function abc () {} (匿名函数表达式) 常用

注意：两者唯一的区别就是查看test.name属性时写了属性数abc没写则是test

3>实参和形参

(1)实例演示 -->

function sun(a, b) { c = a + b console.log(c) } sun(1, 2)

a, b是形参 1, 2是实参

(2)实参长度和形参长度

形参长度 --> arguments.length

实参长度 -->函数名.length

(3)映射关系(形参和实参对应时才有) -->

function sum(a, b) { b = 2; console.log(arguments[1]);} sum(1);

打印出undefined, 打印不出2

sum(1)改为sum(1, 随便什么参数) 打印出2

4>this

(1)this指向问题

##函数的预编译过程this执向window

new test();new完之后会预编译第一时候this = Object.creat(test.prototype)

##全局作用域this指向window

##call/aplay可以改变this指向。

##obj.fun(); fun()里面的this指向obj(谁调用这个方法this指向谁) -->

var obj = { a : function () { console.log(this.name) }, name : "abc" }

obj.a(); 这个this指向obj

(2)this小知识总结

##函数的连续调用return this(函数不屑return默认的值是undefined)

##call的用法 --> .call(需要指向的函数, 参数)

##apply的用法 --> .apply(需要指向的函数, 数组(参数的集合))

注意：只能是数组

##bind的用法 -->函数.bind(obj), this永远执行obj, this被绑死了

5>return

(1)函数有return时结束 函数没写return在函数的最后系统会隐示的自动加上return.

(2)能返回值(如函数对象等)。(用这个时候一般用一个变量接收这个值)

1. js执行的三部曲(预编译)

1>语法解析(通篇扫描下有没有低级的语法错误 比如少了分号没有)

2>预编译(重点)

(1)预解析的详情讲解过程(函数整体提升, 变量声明提升)

##创建AO对象(Active Object)

##找形参和变量声名，将变量和形参名作为AO属性名，值为undefind

##将实参和形参相统一(实参值赋值给形参)

##在函数体里面找函数声明，赋值予函数体

(2)函数整体提升, 变量声明提升详细说明

##函数是整个函数提升上去 -->

function test() { console.log(“a”) }整个函数都提升上去了包括console.log(“a”)

##变量只是变量名提升上去了而变量值没提升上去 -->

var a = 3只是将a提上去了, 3并没有提上去, 3后面一行一行执行赋值

3>解释执行(解析一行执行一行)

1. 函数的作用域(里面的可以访问外面的, 外面的不能访问里面的, 同级的也可以访问)

1>变量的作用域类型

(1)全局作用域(window, GO)

##全局上的任何变量都是window所有

##暗示全局变量

未经声明的变量赋值, 此变量全局对象所有(window), 称暗示全局变量 -->

在函数内无声明直接赋值(a=1)a为全局变量, 注意函数需执行了才算

(2)局部作用域(AO)

##在函数内声明的函数, 且函数执行了

(3)变量生命周期

全局 : 准备执行全局代码前产生, 当页面刷新/关闭页面时死亡

函数 : 调用函数时产生, 函数执行完时死亡

1. [[scope]]属性(重点作用域链, 闭包)

1>概念

每个JavaScript函数都是一个对象，对象中有些属性我们可以访问， 有些不可以，这些属性仅供javaScript引擎存取,[[scope]]就是其中一个[[scope]]指的就是我们所说的作用域，其中存储了运行期上下文的集合。

2>执行期上下文(AO, GO)

[[scope]]中所存储执行期上下文对象的集合，这个集合呈链式连接，我们把这种链式连接叫作用域链

3>作用域链形成时间

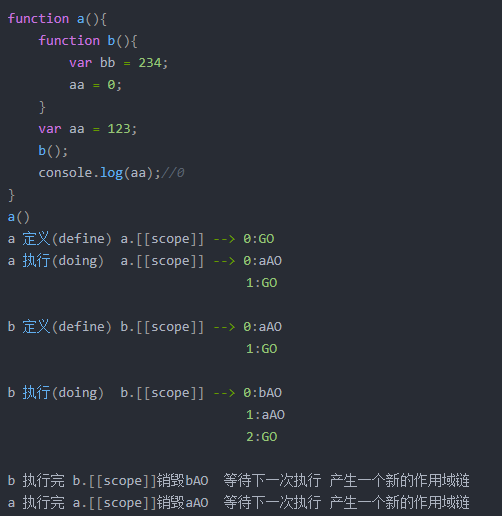
作用域链形成时间为编译期间, 且不能改变

(作用域链关系不能改变从而出现子函数一定能访问父函数的变量现象, 即闭包)

4>运行期上下文(函数执行)

当函数执行时，会创建一个称之为 执行期上下文 (AO对象） 的内部对象。一个执行期上下文定义了一个函数执行时的环境，函数每次执行时对应的执行上下文都是独一无二的，所以多次调用一个函数会导致创建多个执行上下文，当函数执行完毕，他所产生的执行上下文被销毁。

5>作用域链实例详解演示 -->



##理解

每次函数执行产生AO, 往[[scope]]的第0位塞,

把改函数定义时[[scope]]的所有[[scope]]属性往后退一位(站在巨人肩膀上)

6>闭包

(1)基本概念

闭包就是里面的函数被保释到了外部(在销毁之前保存到了外部)

(2)形成条件

##函数嵌套(不需要一定式父子嵌套关系)

##函数的局部变量被保存到了外部

(3)闭包的私有化(留一个接口return里面的函数) -->



1. 立即执行行数(与普通函数唯一的区别是执行后马上释放)

1>写法

(1)( function (形参) {} (实参) )

(2)( function (形参) {}) (实参)

注意：function () {}是匿名函数

2>一般用法

(1)return出一个值, 使用一个变量接收

3>注意事项：

(1)只有表达式能被执行符号执行, 且被执行符号执行的函数会被永久放弃 -->

##var test = function () { console.log('a'); } () 输出a

##function test() { console.log('a');}() 输出不了a, test输出不了函数

(2)()里面是表达式 (function f() {})里面的函数变成了函数表达

1. 对象

1>构建对象的方式(常用, 解决了下面所毛病)

(1)自定义构造函数创建对象

##function 大头峰式命名 () {} var 变量(小写) = new 前面声明的函数() -->



##new对象时候详细讲解



(2)调用系统构造函数创建对象(无法识别是属于具体的哪个对象)

##var 变量名 = new Object() //用个变量接收叫实例化对象 -->



(3)工厂模式创建对象(能批量生产对象,且能识别类型，但繁琐) -->

##function 变量() { var obj = new Object(); 添加属性方法在此添加 return obj;}

var 变量接收 = 变量() (函数执行); (接收变量和变量可以不同) -->



注意:return obj一定要写不然打印出的对象是undefined

(4)通过字面量对象创建对象var obj = {}

(5)Object.create()创建对象

##语法

Object.create(proto[,propertiesObject])

-- proto: 对象, 要继承的原型为null值时

--propertiesObject: 对象, 可选参数, 为新创建的对象指定属性对象。

注意：第一个参数为null时, 是唯一一个继承终端不是Object.prototype

2>循环对象的方法

(1)for prop(随便取, 其值为对象的key) in 对象

3>对象的增删改查

(1)增 对象.东西(里面没有的东西) = "值" -->

mrZhu.age = "21" 则在1的例子里面加上了 age : 21,属性。

(2)查 对象.东西(里面有的东西) -->

mrZhu.sex 里面就会显示male

(3)改 对象.东西(里面有的东西) = "值" -->

mrZhu.smoke = "1年" 对象里面smock就会显示1年。

(4)删 delete 对象.属性(里面有的属性) -->

delete mrZhu.sex 对象里面就没有sex属性了。

4>构造函数的内部原理

(1)在函数题前面隐示的加上this = { }

(2)执行this.xxx = xxx

(3)隐示的返回this -->



注意：只有new Person()之后才发生构造函数的这三步，否则就是正常函数的执行。

5>对象小知识总结

(1)当一个变量没经过声名就使用会报错，但当一个属性没有使用会输出undefined

(2)调用属性的两种方式

##obj.属性 (使用obj.属性里面会隐示的转化下obj[name])

##obj['属性'] 应用(实现调用方法数字几就出几的名字)-->



1. 原型以及原型链

1>原型

(1)型是function对象的一个属性，它定义了构造函数制造出了对象的共有祖先。

(2)原型即prototype，函数刚出生原型就定义好了。它类似一个空对象。

(3)函数会继承原型上的属性和方法(可以利用其, 提取出共有属性)。

注意：若自己身上有原型上一模一样的属性时出现自己身上的属性。

2>原型链

对象通过自己的\_\_proto\_\_找父对象的prototype, 在通过父对象原型的\_\_proto\_\_找祖父对象的prototype, ....这样呈链式结构即原型链。

2>原型的增删改查(四个差不多, 以增为例子) -->

Car.prototype.height = 1400;

function Car(color) { this.color = color } var car = new Car(）；

car.prototype.height = 1000;

注意：添加需要.prototype

3>constructor(构造器), prototype和 \_\_proto\_\_

(1)constructor(构造器)

##每个具有原型的对象都会自动获得constructor属性, 对象的constructor属性返回创建该对象的函数的引用。

##静态属性和静态方法都挂载在其身上(实例不能访问的)

(2)\_\_proto\_\_

##当调用构造函数创建实例时，该实例内部将包含一个指向构造函数原型对象的指针，在大部分浏览器中用\_\_proto\_\_标识；(继承就是上它指向的原型找, 一般顶层为Object)

(3)prototype

##只要创建一个任意新函数，就会生成一个prototype属性，该属性指向函数的原

型对象；

##实例属性和方法都挂在其身上(实例能访问, 自己也能访问)

(4)三者之间的联系

##person(实例化函数).\_\_proto\_\_ == = Person(构造函数).prototype

##person(实例化函数).constructor === Person(构造函数)

##Person(构造函数).prototype.constructor === Person(构造函数)

##Person(构造函数).prototype === new Object(父实例化对象)

(5)总结

##\_\_proto\_\_和constructor是对象独有的属性

(在js中对象是由函数构成的所以有这两个属性)

##prototype是函数独有的属性

(函数也是对象的一种，是并集关系不是父子集关系)

##hasOwnProperty(属性的字符串形式), 判断是不是原型上的属性

##in 使用形式 -->"属性名" in obj

注意：in判断原型上的属性和hasOwnProperty一样, 但不同的是它会把自己的原型也算上来

4>构造函数内部原理 -->



5>原型小知识总结

(1)object.prototype是所有原型链的终端除了Object.create(null)创建的对象

(2)不能通过原型链增删改查, 只能自己修改自己原型

(3)Object.creat( 对象或者null )

##对象的话原型就指向这个对象

##null对象不最终继承Object.prototype除了这个全是最终继承Object.prototype

(4)人为设的\_\_proto\_\_不会继承(写了Object.creat(null)截断了继承对象，然后人为设个\_\_proto\_\_指向是不会继承的)

(5)constructor是Object()就是原型链的顶端

(6)因为原始值undefind和null不是对象，也没有原型链所以不可以经过包装类所以不可以添加属性和方法

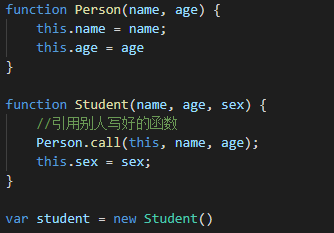
1. 继承发展发展史
2. 传统模式 --- 原型链

缺点：过多的继承了没用的属性, 全部都会继承

2>借用构造函数(开发时常用, 不能算标准的继承)

缺点：每次构造函数都要多走一个函数, 原理是方法借用, 原型是自己

(1)实例 -->



(2)使用说明

##静态属性

--构造函数上创建的属性 -->Person.name = “张三”

##实例属性

--构造函数里面的属性 -->Person (name) { this.name = name //这个是实例属性}

##静态方法(实例不能访问)

--在构造函数上创建的 -->Person.say = function () {}

--若需要然实例访问可以在构造函数的原型上定义称之为实例方法 -->

Person.prototype.say = function () {} 这样实例也能访问这个方法

##实例方法

在实例对象上创建的 -->person.say = function () {}

3>共享原型

(1)实例 -->



(2)共享原型继承封装成方法(Target继承Oringin的) -->



缺点：不能修改自己的原型, 修改自己的原型会影响父级。

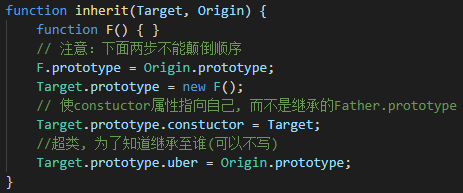
4>圣杯模式

原理：创建一个中间函数, 通过new 中间函数连接两个函数

(1)实例 -->



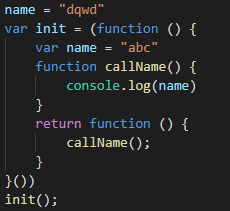
(2)封装方法 -->



1. 命令空间(闭包应用, 模块化开发, 防止污染全局变量)

1>目的：管理变量, 防止污染全局变量(模块化开发)

2>实例 -->

 打印出abc, 不会污染全局变量

1. 包装类

1>包装类基本概念

(1)普通变量不能直接调用属性和方法, 而对象可以直接调用属性和方法

这些普通类型的变量会默认会调用包装

包装类: 普通类型的变量经过包装可以直接调用属性和方法 -->



(2)基本包装类型：数字(number), 字符串(string), 布尔(boolean)

2>包装类详解

加属性方法时候内部会隐式的添加, 执行完毕之后会销毁 -->

var num = 4; num.len = 3 new Number(num).len(隐式的, 执行后销毁)

注意：字符串赋length属性不会影响length数值, 数组会截断数组 -->

var str = “abcd” str.length = 2

访问str.length值为4(包装类的length属性) str = “abcd”

var arr = [1, 2, 3, 4] arr.length = 2

访问arr.length = 2 arr = [1, 2]

3>包装类小知识总结

(1)原始值不能有属性和方法的为什么能调用？

经过包装类(不能加属性但又能访问属性)

1. 字符(都是静态方法)
2. string.indexOf(str, [start])

(1)参数

##str为需要检索的字符

##start需是整数。规定在字符串中开始检索的位置。它的合法取值是 0 到 string.length - 1。如省略该参数，则将从字符串的首字符开始检索。

(2)返回值

##返回值为其在string中的索引位, number类型

##如果要检索的字符串值没有出现，则该方法返回 -1,

(3)实例 -->

 返回值为0

注意区分：lastIndexOf()从后先前找，获取的索引仍然是从左往右数

1. string.charAt(index)

(1)参数

##表示字符串中某个位置的数字，即字符在字符串中的下标。

(2)返回值

##返回值为string索引位为index的字符, string类型

##如果参数 index 不在 0 与 string.length 之间，该方法将返回一个空字符串

(3)实例 -->

 返回值为h

注意: 超过字符串长度(length-1)返回空字符串,空格也占一个字符索引位。

3>string.split(separator, [howmany] )

(1)使用说明

将字符串拆分成数据, 注意区分join方法

(2)实例 -->

 返回值为[“a”, “b”]

4>.slice(“从第几位”, “到第几位”)返回提取后的字符串(第几位都是索引)

 返回值为h

5>.substring(“开始的索引”, “结束的索引”), 返回截取后的字符串，不包含结束的索引字符串, 不能为负值

 返回值为he

6>.substr(“开始的位置”, [“个数”])返回截取后的新字符串

 返回值为el

7>.trim(); 去除字符串两端的空格



返回值为 

8>.toLocaleLowerCase(); 把字符串转为小写的 和toLocalecase();一模一样

 返回值为hello world

9>.toUpperCase(); 把字符转为大写

 返回值为HELLO WORLD

10>.replace(“old string”, “nwe string”)方法 把old string替换成nwe string

 返回值为olllll

注意：不用正则只替换一次

11>String.fromCharCode(ussc码, ussc码, ...)返回参数对应的USSC码对应的字符串

 返回值为ABC

1. 字符串小知识

(1)字符串string和字符串对象(new String())

-->

 输出还是hello 无法改变(只可以读，不可写)

-->

栈指向了新的堆

输出的结果是”小苏好邪恶”

1. 数组

1>构建数组的方式

(1)var arr = [];

(2)var arr = new Array();

注意：这个括号和(1)构建的对象[]添加值一样唯一的区别就是只传一个值这个代表数组长度。

2>数组小知识

(1)当给数组第十位赋值时会将数组撑长为11 -->

arr[10] = abc 打印出 [undefined\*10, abc], 且length值为11

(2)数组除了可以用for循环遍历外还可以使用for item in arr(数组是特殊的对象)

3>数组方法

(1)可以改变原数组的方法

##.push(); 把值追加贷数组中(最后面), 返回值为追加后数组的长度 -->



数组为 打印arr为 5

##.pop()把删除数组中最后一个元素的值, 返回值为删除的这个值



数组为 打印arr为400

##.shift()删除数组中第一个元素, 返回值为删除的这个值



数组为 打印arr为 100

##.unshift(值)向数组中的第一个元素前插入值, 返回的是插入后数组的长度



数组为 打印arr为5

##.sort(function () {})排序不稳定, 可传入一个比较函数可以进行排序

function () { 排序规则 } 升序原理(return a-b;)降序(return b-a)

注意：1形参一定要写两个值

2看返回值

1)当返回值为负数,那么前面的数放在前面 (return 1和return -1)

2)当为正数,那么后面的数在前面

3)为0,不动

##.reverse()数组反向



数组为 打印arr为[25, 16, 4, 1]

##.splice(“开始”, “要删除的个数”, [“截取后添加的元素”])截取切在截取位置添加元素



数组为 打印arr为4

注意：截0位(在数组中间添加元素)

(2)不改变原数组(一般用个函数接收不然没意义)

##.concat()连接两个数组



数组为 打印arr为原数组arr

##.join("东西"按东西样式连接(注意需要时字符串形式))



数组为 打印arr为原数组arr

##.split("东西"按照东西样式拆分数组)



数组为 打印arr为

##.slice(“开始”, “结束“);截取数组从开始到结束(不包括结束那个元素)



数组为 打印arr为原数组arr

4>其他数组方法

(1)forEach(function (value, index, arr) {}) 循环数组(相当于一个for循环)

##第一个参数为元素的值

##第二个为元素的索引

##第三个参数为原数组

(2)map(function (value, index, arr) {})

##参数和forEach一样

##数组中每个元素都会执行这个函数, 执行计算后重新放到一个新数组返回出来





forEach和map区别：forEach不会返回值, map会返回一个新数组

(3)every(function (ele, index, arr) {})

##函数的第一个参数代表元素的值

##第二个代表元素的索引

##第三个是原数组(谁调用的就是谁一般用不到第三个参数)

##数组中的每个元素都会执行这个函数, 当所有元素符合条件则返回true -->





(4)some(function () {})

##函数的第一个参数代表元素的值

##第二个代表元素的索引

##第三个是原数组(谁调用的就是谁一般用不到第三个参数)

##数组中的每个元素都会执行这个函数, 当一个元素符合条件则返回true

every和some区别：every所有元素符合条件返回true, 而some只需一个元素满足

(5)filter(function () {})

##函数的第一个参数为元素的值

##第二个元素的索引

##第三个参数为遍历的数组(原数组)

##返回符合设定条件的值成数组(新数组)形式 数组中的每个元素都会执行这个函数





(6)indexOf(string)

##返回的是string在素组中的索引位, 没有这个元素值则返回-1

##不能检测NaN, 检测NaN需要使用includes方法

 返回值为0

(2)Array.isArray(数组) 判断是否是数组 -->





1. 类数组

1>是类数组的条件

(1)属性要为索引(数字)属性

(2)必须要有length属性(必要条件),做好加上push属性

2>类数组注意事项

(1)类数组长得像数组但又不能用数组的方法 -->

arguments是个类数组,不能用push数据进数组里面会报错

(2)强行让类数组调用push方法,则会根据length属性值的位置进行扩充 -->



打印出

##其原理就是push方法的封装 -->



1. 区分数组[]和对象{}的几种方法

1>.obj.constructor打印function Object(){[native code]}

2>.[] instanceof Array 打印ture ([]对象\_\_poto\_\_上有没有Array的prototype)

(1)实例 -->

A instanceof B(为Array或者Object) 检测A是否属于B





注意：数组是特殊的对象上例子中写 [] instanceof Object 返回值为true

3>Object.prototype.toString.call([]); 打印出'[object array]'

注意：var obj = {} (直接在控制台上打印{}不可以会报错)

1. Data对象

1>不需要先new Date的方法(静态方法)

(1).new Date() 获取当前时间 -->





(2).new Date(string) 传入字符串参数的时间 8:00 -->





注意:需要是字符串，获取的是时间类型的值

(4)计算时间 -->





注意：是数字类型 为了计算，时间值和时间值不可计算可以用其计算时间



2>需先new Date的方法(常用)(动态方法)

(1).getFullYear()获取年的方法用 -->





注意区分：.getYear()方法不常用(1900到现在的年数)

(2).getMonth() 获取月份方法 -->





注意：获取的月份从0开始要 +1才是当前的月份 例子中真实是7月

(3).getDay- 获取星期几 -->





(4).getDate() 获取日期 -->





(5).获取小时,分钟，秒



(6)其他方法

##日期









##小时分钟秒









##毫秒数





注意：和Date.now()区别

-- Date.now和new Date是同级， 而value需先new Date在使用

1. Math内置对象(大部分用来处理数字)
2. 为静态方法, 方法查看js文档百度搜索MDN

2>Math不是一个构造函数不像Object -->

(1)不能使用new Math实例化一个方法(报错)

(2)Math.PI();(PI方法在里面已经定义好了)

3>封装一个Math.max()方法-->



1. 其他常见方法

1>JSON.stringfy方法和JSON.parse方法(不是相互转化关系)

json.stringfy()将对象、数组转换成字符串；json.parse()将字符串转成json对象。

2>eval(string)

##把字符串string当函数来执行, 严格模式(“use strict”)下有自己单独的作用域。

##requireJS原理把依赖的模块解析成text/html, 即动态的字符串,

在用eval执行(动态执行函数)