JS深入浅出

. 原型对象
　在JavaScript 中，每当定义一个对象（函数）时候，对象中都会包含一些预定义的属性。
其中函数对象的一个属性就是原型对象 prototype。
注：普通对象没有prototype,但有__proto__属性。

原型对象其实就是普通对象（*Function.prototype除外,它是函数对象，*
但它很特殊，他没有prototype属性（前面说道函数对象都有prototype属性））

原型与原型链详解： http://www.108js.com/article/article1/10201.html?id=1092
console.log

!function
自执行函数表达式
在这种情况下，解析器在解析function关键字的时候，会将相应的代码解析成function表达式，
而不是function声明。
(function () { /* code */ } ()); // 推荐使用这个
(function () { /* code */ })(); // 但是这个也是可以用的
/ 由于括弧()和JS的&&，异或，逗号等操作符是在函数表达式和函数声明上消除歧义的
// 所以一旦解析器知道其中一个已经是表达式了，其它的也都默认为表达式了

Arguments是个类似数组但不是数组的对象，说他类似数组是因为其具备数组相同的访问性质及方式，
能够由arguments[n]来访问对应的单个参数的值，并拥有数组长度属性length。还有就是arguments对象存储的是实际 传递给函数的参数，
而不局限于函数声明所定义的参数列表，而且不能显式创建 arguments 对象。

6种数据类型：数字，字符串，布尔型，对象，空，未定义。除了对象之外都是原始类型
其中对象又包括：fucntion,Date,array

把变量转换成数字：num-0；
把变量转换成字符串：num+""
a==b
类型相同的话，同===
类型不同，尝试类型转换和比较

a===b
首先判断类型，类型不同返回false
NaN!=NaN
NEW Object !=new Object
因为对象是用引用比较

包装对象，就是当基本类型以对象的方式去使用时，
JavaScript会转换成对应的包装类型，相当于new一个对象，
内容和基本类型的内容一样，然后当操作完成再去访问的时候，
这个临时对象会被销毁，
然后再访问时候就是undefined

obj instanceof object 基于原型链
类型检测
typeof
适用基本类型及function检测，遇到null失效
instanceof
通过{}.toString拿到，适合内置对象和基元类型，遇到null和undefined失效（IE678等返回[object Object]）
Object.prototype.toString
适合自定义对象，也可以用来检测原生对象，在不同iframe和window间检测时失效
constructor 属性返回对创建此对象的数组函数的引用。

表达式
表达式是可以计算出值的任何程序单元
表达式是一种js短语，可使js解释器用来产生一个值
1原始表达式 2 数组，对象的初始化表达式
3函数表达式 4 属性访问表达式

5调用表达式 6对象创建表达式

运算符

一元 +num

二元 a+b

三元 c ? a:b

赋值 x+=1 比较a==b 算术 a-b

位 a|b 逻辑 exp1 && exp2

字符串 "a"+"b" 特殊运算符 delete

&&和||在JQuery源代码内尤为使用广泛，由于本人没有系统的学习js，所以只能粗略的自我理解出来，希望大家指点下。

粗略理解如下：

a() && b() :如果执行a()后返回true，则执行b()并返回b的值；如果执行a()后返回false，则整个表达式返回a()的值，b()不执行；

a() || b() :如果执行a()后返回true，则整个表达式返回a()的值，b()不执行；如果执行a()后返回false，则执行b()并返回b()的值；

&& 优先级高于 ||

特殊运算符小结：

一. ? var op = false ? 1:2 //op=2

二 , var op =(1,2,3,4) //op=4

三. delete var obj = {x:1};

obj.x; //1

delete obj.x;

obj.x; //undefined

configurable :false 无法delete 属性

默认情况下设置为true ,可以删除属性

四. in

window x = 1;

'x' in window //true

五 instanceof 判断对象类型

typeof 判断原始类型 , 函数类型

六 new

七 this

var op ={ func:function(){return this;}};

op.func() //op

8 void

void 0//undefined

void(0)//undefined

JavaScript程序由语句构成,语句遵循特殊的语法规则

例如 ：if语句 , while语句 , with语句

块block

语法:

{

语句1；

语句2；

。。。

语句n；

}

JavaScript中没有块作用域

try{

}catch{

}finally{

}

try中抛出的异常没有被同级的catch所捕获时，会被最近的外层catch所捕获。

try后面至少接一个catch 或者 finally，无论有没有异常最后都会执行finally。

for...in

遍历对象

1顺序不确定

2enumerable为false时不会出现

3 for in对象属性时原型链会受到影响

js中不建议使用with（原因）：

1.让JS引擎优化更难；
2.可读性差
3.可被变量定义代替
4.严格模式下被禁用
使用时可通过定义变量来取代with

严格模式是一种特殊的执行模式
它修复了部分语言的不足
提供更强的错误检查，并增强安全性
'use strict' 是javascript 的严格模式，他修复了javascript语言的不足，
不允许用with 所有变量必须声明, 赋值给为声明的变量报错，
而不是隐式创建全局变量。 eval中的代码不能创建eval所在作用域下的变量、函数。
而是为eval单独创建一个作用域，并在eval返回时丢弃。
函数中得特殊对象arguments是静态副本，而不像非严格模式那样，
修改arguments或修改参数变量会相互影响。
删除configurable=false的属性时报错，而不是忽略 禁止八进制字面量，
如010 (八进制的8) eval, arguments变为关键字，
不可作为变量名、函数名等 一般函数调用时(不是对象的方法调用，
也不使用apply/call/bind等修改this)this指向null，而不是全局对象。
若使用apply/call，当传入null或undefined时，
this将指向null或undefined，而不是全局对象。
试图修改不可写属性(writable=false)，
在不可扩展的对象上添加属性时报TypeError，
而不是忽略。 arguments.caller, arguments.callee被禁用
JS是向上兼容的

对象包含一系列属性，这些属性是**无序**的，每个属性都有一个**字符串key**和对应的value
例子 var obj={x:1 y:2}
obj.x;//1
obj.y;//2;

writable
enumerable
configurable
value
get/set
calss
proto
extensible

对象字面量可以嵌套
如：
```
var obj2={
x:1
y:2
o{
z:3,
n:4
}
};
```

所谓的prototype就是一条原型链，优先查找当前的对象上是否存在这个属性，如果不存在则查找上一层次的原型上是否存在。最终都会追溯到 object对象上，所有的函数的最终原型链都导向对象object，所以所有的function都具有tostring()方法。
函数声明时function foo(){}，默认会有原型prototype（对象）。
对象被创建时var obj = new foo(); obj也有原型，即构造器（foo函数体）原型（foo.prototype）,

该原型的原型为Object.prototype.最终指向null。

toString方法是Object.prototype上的方法，所以平常创建的对象都能调用toString方法，因为该方法在原型链上的末端。

obj.hasOwnProperty('z');//该方法判断z是否为obj对象自己的属性（非原型链上的属性）。
对象创建
```
varovj=Object.create({x:1});
obj.x//1
typeof obj.toString//"function"
obj.hasOwnProperty('x');//false

var ovj=Object.create(null);
```

obj.toString//undefined
读写对象属性
属性异常
删除属性
检查属性
枚举属性

for in遍历属性的话，是不确定的，有可能将原型链的东西也遍历出来

全局变量不能被删除 局部变量也是不能被删除 函数也是同理 无论全局函数还是函数内部的局部作用域的函数都是不可以被delete 但是隐式的
如
ohNo=1；
window。ohNo//1
delete ohNo//true

看对象是否可以被枚举
调用propertyIsEnumerable这个方法
如：

```
var cat = new Object;
cat.legs = 4;
cat.name = "Kitty";
 'legs' in cat; // true
'abc' in cat; // false
"toString" in cat; // true, inherited property!!!

cat.hasOwnProperty('legs'); // true
cat.hasOwnProperty('toString'); // false
 cat.propertyIsEnumerable('legs'); // true
cat.propertyIsEnumerable('toString'); // false
```