### 根据cocosjs 记录的知识点

null 空 还有undefined

常量 const NUM=0;常量通常用大写

变量 var num=0;//var num1=10, num2=15;

var currentDay=new Date()//用new生成一个新对象

数组

Var arr=new Array(3)3是数组长度，通过arr.length可以获取长度

//转化

parseInt()和parseFloat()

//也有string用能

Var num=10

Num.string();

//强制转化类型

Boolean(vlaue) Number(value) String(value)

//运算

都一样 ，-- ++ -= +=

//控制运算 for if switch while do-while一样

数组

Var arrayOfints=[1,123,33,3]

函数

Function 函数名称（）

{

}

//嵌套函数

Fuction calculate(opr,a,b)  
{

Function add()

{

rturn a+b;

}

}

返回函数-----

3原型----------------

这个链接可以看一下

<https://www.cnblogs.com/Jabin/p/5804159.html>

}

//测试中用到是见console.time()和console.timeEnd()来进行测试。

 console.time('no declare');

console.timeEnd('no declare');//应该是说明用的我没去查找

///函数的理解

Function Ceshi ()

{

Ceshi2:function ()//这里函数里面

{

}，

Ceshi3:function ()//这里函数里面

{

}

}

Var Ceshi=function()//函数在后  
{

}

### 防护数据措施

assert(key)

assert(typeof(number)==="number")

assert(typeof(string)==="string")

assert(Array.isArray(array))

assert(typeof(object)==="object")

assert(typeof(bool)==="boolean")

#### 可变参数

function abc()

{

//js中有个变量arguments,可以访问所有传入的值

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

window.alert(arguments[i]);

}

}

看完这些在总结js和lua的不同

#### 初始化数据，字典

这个是js写法，

var imagePathes=

{

[1]:{

[0] : "games/xini/images/effectPizi/my\_card\_corner\_1.png",

[1] : "games/xini/images/effectPizi/my\_card\_corner\_1.png",

},

[2]:{

[0] : "games/xini/images/effectPizi/card\_corner\_s\_self.png",

[1] : "games/xini/images/effectPizi/card\_corner\_s\_right.png",

         },

};

数组的写法

var dataTable =

[

{ title : "人数",

labelTab : [ "4人", "3人", "2人" ],

tagTab : [ 4, 3, 2 ],

type : 1,

posY : 218,

countH : 3,

spaceH : 60,

line : true,

defaultS : 0

},

{

title : "局数",

labelTab : [ "8局", "16局", "32局"],

tagTab : [ 8, 16, 32 ],

type : 1,

posY : 288,

countH : 3,

spaceH : 60,

line : true,

defaultS : 1

}

]

这个是lua写法

local imagePathes=

{

[1]={

--- Down  = 0,Right   = 1, Up         = 2, Left   = 3,

[Enum.Chair.Down] = "games/xini/images/effectPizi/my\_card\_corner\_1.png",

[Enum.Chair.Right] = "games/xini/images/effectPizi/my\_card\_corner\_1.png",

[Enum.Chair.Up] = "games/xini/images/effectPizi/my\_card\_corner\_1.png",

[Enum.Chair.Left] = "games/xini/images/effectPizi/my\_card\_corner\_1.png",

},

[2]={

[Enum.Chair.Down] = "games/xini/images/effectPizi/card\_corner\_s\_self.png",

[Enum.Chair.Right] = "games/xini/images/effectPizi/card\_corner\_s\_right.png",

[Enum.Chair.Up] = "games/xini/images/effectPizi/card\_corner\_s\_self.png",

[Enum.Chair.Left] = "games/xini/images/effectPizi/card\_corner\_s\_left.png",

}

}

#### 数组添加

Var arrNum=[];

直接添加 arrNum.push(11);或者arrNum[arrNum.length]=11

#### Js时间获取

var i, t = Date.now()

console.log(Date.now() - t) // 250 （chrome浏览器）

#### js闭包的理解

//js的做作用域无非就是全局和局部变量（let模块内就是{}很很少用，知道就好），全局没发获取局部，所有闭包其实可以理解为给外层用的

函数内声明变量时，一定要使用var 如果没加那就是全局表量

Function fn()  
{

N=555;//全局

}

各种文献的闭包（closure）定义非常抽象，很难理解，我的理解闭包就是读取其他函数内部变量的函数

什么是闭包：当内部函数在定义它作用域的外面被引用就创建了该函数的闭包

#### Js的prototype和\_proto\_区别

prototype和\_\_proto\_\_都指向原型对象，**任意一个函数（包括构造函数）都有一个prototype属性**，指向该函数的原型对象，同样**任意一个构造函数实例化的对象**，**都有一个\_\_proto\_\_属性（\_\_proto\_\_并非标准属性，ECMA-262第5版将该属性或指针称为**

function a() {

//console.log("I'am a function.");

}

//b是实例化对象，a是构造函数

var b = new a();

console.log(b.\_\_proto\_\_ == a.prototype); //true  
console.log(Object.getPrototypeOf(b) == a.prototype); //true,标准方法返回的结果和上述一样

##### 区别

//注意：a作为构造函数时的prototype属性与a作为普通函数时的\_\_proto\_\_属性并不相等

console.log(a.prototype == a.\_\_proto\_\_);//false

//这个怎么理解呢，我们再看看如下代码

console.log(a.\_\_proto\_\_); //function (){}

console.log(a.\_\_proto\_\_ == Function.prototype);//true

//实际上，a作为一个普通函数调用的时候，它的构造函数就是内置对象Function了，所以它指向的原型对象，自然对应就是Function.prototype.

//其实这个和console.log(b.\_\_proto\_\_ == a.prototype)是一样的道理

//我们继续看如下代码，当a作为构造函数时，它的原型，和它的原型的原型都指向神马

console.log(a.prototype); //a{}

console.log(a.prototype.\_\_proto\_\_); //Object{}

//我们再看看a作为普通函数时，它原型的原型指向神马

console.log(a.\_\_proto\_\_.\_\_proto\_\_); //Object{}

//即有以下结果

console.log(a.\_\_proto\_\_.\_\_proto\_\_ == a.prototype.\_\_proto\_\_); //true  
  
//结论：[prototype属性可以给函数和对象添加可共享（继承）的方法、属性](http://www.ecma-international.org/ecma-262/6.0/index.html" \l "sec-terms-and-definitions-prototype" \t "_blank)，而\_\_proto\_\_是查找某函数或对象的原型链方式  
//因此，有人说原型链实际是通过\_\_proto\_\_属性链接起来的，也是有一定道理的

#### js和lua的create差别？？

在js中

Var NodeMgr=cc.Node.extend(

{

Ctor:function()  
{

}

})

NodeMgr.cretae=function()  
{

这里的create自己写 而lua不用

}

一、基础教程

js语法可以根据下面网址学习，可以忽略关于HTML章节（JS HTML DOM、JS浏览器DOM、JS库、JS实例）

<http://www.runoob.com/js/js-mistakes.html>

二、进阶教程

js怎么模拟类的实现、js可优化的性能、js哪些坑等

<http://bonsaiden.github.io/JavaScript-Garden/zh/>

三、js和lua区别（js语法基本和c++一致）

0.调用函数

Lua习惯使用“:”，如Sprite:create()

JS固定使用“.”，如Sprite.create()

1. 数组

local dataLua = { 1, 2, 3, 4, 5 }

var dataJS = [ 1, 2, 3, 4, 5 ]

dataLua[1] // Lua索引从1开始

dataJS[0] // JS索引从0开始

2. 字典

// 初始化方式

local dataDic = { “a”=1, “b”=2, “c”=3 } // Lua

var dataDic = { a:1, b:2, c:3 } // JS

// lua可以使用变量作为Key值，JS不能

local key = “MyKey”

local dataDic = { key = “value” }

print( dataDic[key] ) -> value

print( dataDic[“key”] ) -> nil

var key = “MyKey”

var dataDic = { key : “value” }

log ( dataDic[“key”] ) -> “value”

log ( dataDic[key] ) -> undefined

// Lua允许使用数组或字典或Object当键值，但JS不行

var obj1 = { a:1 }

var obj2 = { b:2 }

var dataDic = { }

dataDic[obj1] = 100

dataDic[obj2] = 200

log(dataDic[obj1]) –> 200

// Lua删除健值只要赋值为nil即可，js需要使用delete

var dic = { a:1, b:2, c:3 }

dic["b"] = undefined

for (var key in dic)

{

log(dic[key])

}

输出：1、undefined、3

delete dic["b"]

for (var key in dic)

{

log(dic[key])

}

输出：1、3

3. Lua支持函数多个返回值、JS只能返回一个值

4. js有null和undefined表示空值，undefined相当于Lua的nil，一般不使用null

5. Lua 数字0和字符串”” 表示为真，在js表示为false

local value = 0 // 同理value = “”

if value then

print(“It is True”)

else

print(“It is False”)

end

输出：It is True

var value = 0 // 同理value = “”

if (value)

{

print(“ It is True ”)

}

else

{

print(“It is False”)

}

输出：False

// false与0的比较

var cond = false

log(cond == 0) -> true

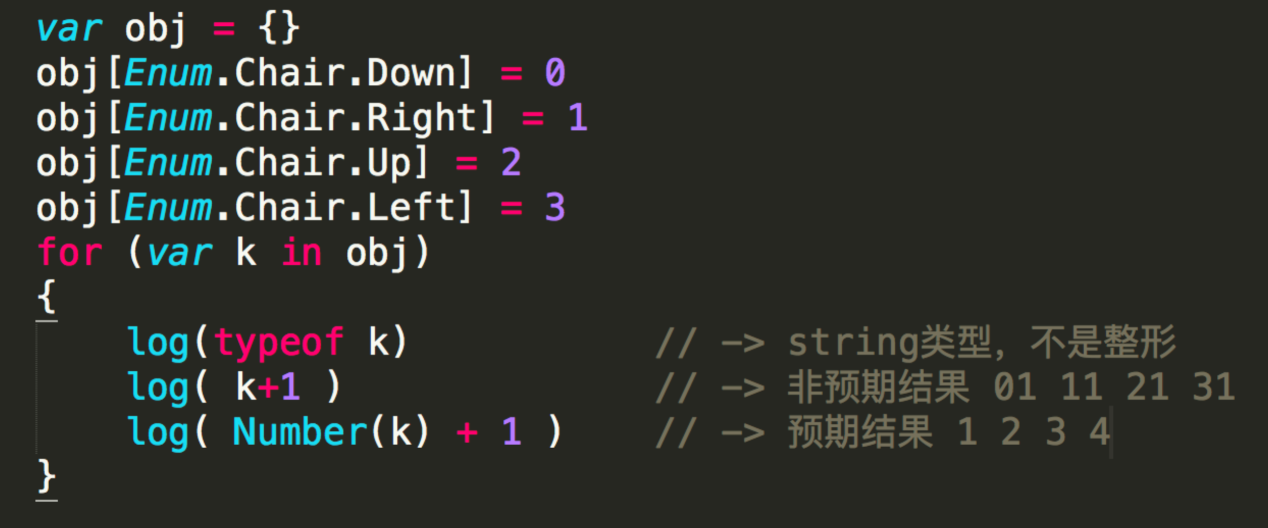
log(cond === 0) -> false

所以推荐使用 ”===” 替换 “==”

6. js的object数据类型遍历区别

Lua：key值可以是整形、字符串、对象等

JS：key值会强制转为字符串



# [javaScript链式调用原理以及加法实现](https://segmentfault.com/a/1190000008724608)

var labelBeiShuBg = Sprite.createWithPlist(undefined,"hgan\_res/other/bg\_level.png")//先生成来复制非labelBeiShuBg

.anchorEx(layout[1])// 返回值设置让回在返回值

.setCascadeOpacityEnabledEx(true)// 返回值设置让回在返回值

labelBeiShuBg.setVisible(false)// 这里没放回值导致undefine

## for in 的说明

for(var i = 0; i<4;i++)

{

var beiShu = []

beiShu.wZhuang = msg.ReadInt() // --庄

beiShu.wKaiKou = msg.ReadInt() //--开口

beiShu.wFaKuan = msg.ReadInt() //--罚款（未开口）

beiShu.wLaiZiGang = msg.ReadInt() // --赖子杠

beiShu.wHongZhongGangPiZiGang = msg.ReadInt() // --红中皮子杠

beiShu.wWinBeiShu = msg.ReadInt() // --赢家倍数

beiShu.wAnGang = msg.ReadInt() //--暗杠

for (var i in beiShu)

{

console.log((i + ": " + beiShu[i]));

}

beiShuInfo.push(beiShu)

}

结果是这样的，所以

wZhuang: 0

wKaiKou: 1

wFaKuan: 0

wLaiZiGang: 0

wHongZhongGangPiZiGang: 0

wWinBeiShu: 0

selfFanShu: 2

wMingGang: 1

wAnGang: 0

dZiMo: 0

wYingHu: 0

wYuanLai: 0

fFangChong: 0

llScores: -160

fengDingLeiXing: 0

bTotalFanShu: 2

##### js 中console.log()和console.dir()不同dir可以打出属性就是了

## 总结js中apply和call的方法

对于apply和call两者在作用上是相同的，但两者在参数上有区别的。  
对于第一个参数意义都一样，但对第二个参数：  
apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。  
如 func.call(func1,var1,var2,var3)对应的apply写法为：func.apply(func1,[var1,var2,var3])

同时使用apply的好处是可以直接将当前函数的arguments对象作为apply的第二个参数传入

### Js和lua一些数组不同

//lua

iTable:append(ruleData, roomInfo.logRules)

local ruleData = iTable:copy({}, roomInfo.rules)

[\\js](file:///\\\\js)

var ruleData = roomInfo.rules.slice()

ruleData.concat(roomInfo.logRules)

#### 3 数组的forEach方法

var dataNum=[1,2,3,55,3,6,77]

dataNum.forEach(function(num,index,arry)//参数1 值 参数2 index 参数3 arra

{

console.log("num=======",index)

})

# 端开发的45个经典技巧

# 这个很重要很好的知识

<https://blog.csdn.net/u013084331/article/details/51037111>

# bind中我遇到bind(cfg)这是在PKButtomGroup中遇到的

### 处理麻将点炮最后一张不要被排序

// 3n+2 的牌最后一张我不排序

for (var i =0;i< handCardes.length ;i++)

{

var canPop = handCardes[i].length>0 & handCardes[i].length % 3 == 2;

var lastSprite = canPop ? handCardes[i].pop() : undefined

handCardes[i].sort(function(v1, v2)

{

var Num1=v1

var Num2=v2

Num1 = Num1==laizi ? Num1-200 : Num1

Num2 = Num2==laizi ? Num2-200 : Num2

return Num1-Num2;

})

}