## Day01

### ## 1. 内存,数据, 变量三者之间的关系

内存是可以存储不同数据的容器

变量与内存对应, 变量名是内存的标识名称, 变量值是内存中保存的数据

### ## 2. 关于引用变量赋值的2个问题

2个引用变量指向同一个对象, 通过一个引用变量修改对象内部数据, 另一个引用变量也看得见

2个引用变量指向同一个对象,让一个引用变量指向另一个对象, 另一个引用变量还是指向原来的对象

### ## 3. 如何判断函数是否是回调函数

你定义的

你没有直接调用

但它最终执行了

### ## 4. 说说对象,数组, 函数的区别

对象: 一个可以存储多个无序数据的容器

数组: 一个特别的对象, 存储的数据是有序的

函数: 一个特别的对象, 包含可执行的代码(可以执行)

### ## 5. 如何判断函数中的this

直接调用: xxx() : window

new调用: new xxx() : 新创建的对象

对象调用: obj.xxx() : obj

call/apply调用: xxx.call(obj) : obj

var p1 = {id: 1, name: 'Tom'}

var p2 = {id: 2, name: 'Tom2'}

var p3 = {id: 3, name: 'Tom'}

var obj = {}

obj[p1.id] = p1

obj[p2.id] = p2

obj[p3.id] = p3

var arr = []

arr.push(p1)

arr.push(p2)

arr.push(p3)

var id = 3

arr.forEach (function (p){if(p.id===id){}})

obj[id]

## Day02

### ## 1. git的6个基本操作

1). 创建本地仓库

创建.gitignore并配置忽略

git init

git add \*

git commit -m "xxx"

2). 创建远程仓库

New Repository

指定名称

创建

3). 将本地仓库推送到远程仓库

git remote add origin https://github.com/zxfjd3g/xxx.git 关联远程仓库

git push origin master

4). 如果本地有更新, 推送到远程

git add \*

git commit -m "xxx"

git push origin master

5). 如果远程有更新, 拉取到本地

git pull origin master

6). 克隆远程仓库到本地

git clone https://github.com/zxfjd3g/xxx.git

### ## 2. 说说原型与原型链

1). 所有函数都有prototype属性(显式原型), 属性值指向原型对象

2). 所有实例对象都有\_\_proto\_\_属性(隐式原型), 其值就就等于构造函数的prototytpe属性值, 也就是也指向原型对象

3). 多个对象通过\_\_proto\_\_属性就形成了链就是原型链, 在查找对象的某个属性时: 先在对象自身上找, 如果找不到就会沿着原型链查找

### ## 3. 说说变量提升和函数提升

1). 变量提升: 在用var声明变量的语句之前就可以访问到变量 ==> var声明变量部分被提前执行了

2). 函数提升: 在function函数声明语句之前就可以调用函数 ==> 函数声明语句被提前执行了

3). 原因: JS引擎在执行代码时做了预处理/解析的工作

### ## 4. 说说作用域与作用域链

1). 作用域: 变量可见的一块特定代码区域, 包括全局作用域和函数作用域, 作用域在编写代码时就确定了

2). 作用域链: 由内向外的多个嵌套作用域形成的链, 在查找变量时, 会沿着作用域链查找

### ## 5. 画内存详图

function Foo () {}

var f1 = new Foo()

var f2 = new Foo()

var o1 = {}

var o2 = {}

var obj1 = {m: 1}

var obj2 = {n: 2}

obj1.xxx = obj2

var parent = {name: 'tom'}

var child = {name: 'jack'}

child.parent = parent

var child2 = {name: 'bob'}

child2.parent = parent

child2.parent.name

parent.children = []

parent.children.push(child)

parent.children.push(child2)

parent.children.forEach(function (child) {child.name})

a.b

## Day03

### ## 1. 产生闭包的条件与什么是闭包

1). 产生闭包的条件

\* 函数嵌套

\* 内部函数引用了外部函数的数据(变量/函数)

\* 执行外部函数

2). 什么是闭包

闭包本质是内部函数中的一个容器, 这个容器中包含引用的变量

### ## 2. 编写程序演示闭包

function fn1() {

var a = 3

function fn2() {

a++

console.log(a)

}

return fn2

}

var f = fn1()

f() // 4

f() // 5

### ## 3. 对象创建模式(4种)

1). Object构造函数模式

var obj = {}

obj.name = 'Tom'

obj.setName = function(name){this.name=name}

2). 对象字面量模式

var obj = {

name : 'Tom',

setName : function(name){this.name = name}

}

3). 构造函数模式

function Person(name, age) {

this.name = name

this.age = age

this.setName = function(name){this.name=name}

}

new Person('tom', 12)

4). 构造函数+原型的组合模式

function Person(name, age) {

this.name = name

this.age = age

}

Person.prototype.setName = function(name){this.name=name}

new Person('tom', 12)

### ## 4. 编码实现组合继承模式

function Parent(xxx){this.xxx = xxx}

Parent.prototype.test = function(){}

function Child(xxx,yyy){

Parent.call(this, xxx) //借用构造函数 this.Parent(xxx)

}

Child.prototype = new Parent() //得到test()

Child.proptotype.constructor = Child

Child.prototype.test2 = function (){}

var child = new Child() //child.xxx为'a', 也有test()

### ## 5. 下面代码的输出

for (var i = 0; i < 3; i++) {

setTimeout(function() {

console.log(i);

}, 0);

console.log(i);

} // 0 1 2 3 3 3

for (var i = 0; i < 5; i++) {

(function(i){

setTimeout(function (){

console.log(i);

},1000);

})(i);

} // 0 1 2 3 4