js高级总结:

**day03:函数,重overload ,匿名函数, 作用域和作用域链 闭包**

**函数:function的本质是new Function**

**一.函数**

**1.创建函数**

1.1.function 函数名(形参){函数体,返回值return}**注意:函数会整体被声明提前 (同名函数会替换)**

1.2.v**ar 函数名=function(){函数体,return返回值}** 注意**函数名var 函数名被声明提前,=function(){}留在原位置(常用可以避免声明提前)**

1.3. 函数名=new Function("形参变量1","形参变量",...,"函数体; return 返回值");

**js函数的本质: 函数也是一个对象**

**其实function 只是 new Function()的简写而已！**

**function等效于new Function()**

**2.声明提前**

Js引擎会先将var声明的变量和function声明的函数，提前到当前作用域的顶部集中创建

**2.1 var 声明提前**

var a=1ex:var a 声明提前 值留在原地1

ex: var a

a=1

**2.2 函数声明提前**

赋值(=)留在原地！

=前的声明部分被提前，而函数定义留在原地。

函数其实也是一个对象，只不过内部存的不是数据，而是代码段而已

函数名仅仅是一个普通的变量。函数名通过保存函数对象的地址，来引用这函数对象

**2.3强调:**

引用类型存在多个值,在内存中只会保存变量的地址 如果使用同一个数组/对象 内存会复制一个地址新的变量,因为地址是指向同一个地址 所以修改任何一个都会影响所有的变量的值

原始类型:内存只可以存储一个值,所以只有重新赋值才会改变变量的值 另外要赋值个另外一个变量,内存会复制一个副本赋值给新德变量,不影响原始的值

**二.重载overload**

js不支持重载的效果(多个相同名的函数会覆盖为最后一个,声明提前)

解决: js中是通过一个特殊arguments对象来变通实现重载效果 arguments属于类数组对象,有一个length属性但不能使用数组家的函数

相同: 1. 下标i, 2. length, 3. 可用for循环遍历

arguments: { 0: 值1, 1:值2,... , length:元素个数 }

不同: 类数组对象不是数组家孩子！无权使用数组家任何函数！

使用:不要定义函数里不要定义形参,直接使用 arguments.length

ex:手机支付 function(){if(if(arguments.length==0){}else if(.....

**三.匿名函数以及匿名函数自调**

作用:防止全局污染

function(){} 匿名函数

强调:几乎所有回调函数都用匿名函数:

(function(){})();匿名函数自调

强调: 匿名函数自调前后的语句必须都加";"，不能偷懒!

**四.作用域(scope)和作用域链(scope chain)**

作用域:

在函数内var出来的变量，是局部变量

形参变量也会成为局部变量

1.全局作用域对象 属于全局变量 优点:可重用 缺点:造成全局污染

全局作用域存储在window中

2.函数作用域对象 属于局部变量/形参也是局部变量 优点:不会造成全局污染,缺:不可重用

2.1强调:先局部后全局

2.2调用函数时,会创建临时作用域对象 储存 函数体内的局部变量,以及形参,

2.3调用函数时,临时作用域对象会把地址给 给离他最近的"好友列表" 之后每次都会先找临时函数作用域对象,没有再找window, 两者都没找到才会报错

**五.闭包**

闭包形成的原因: 外层函数调用后，外层函数的作用域对象被返回出来的内层函数引用着，无法释放！就行了闭包对象!

本质:外层函数的作用域对象被内层函数应用着无法释放，这样的作用域对象就称为闭包对象

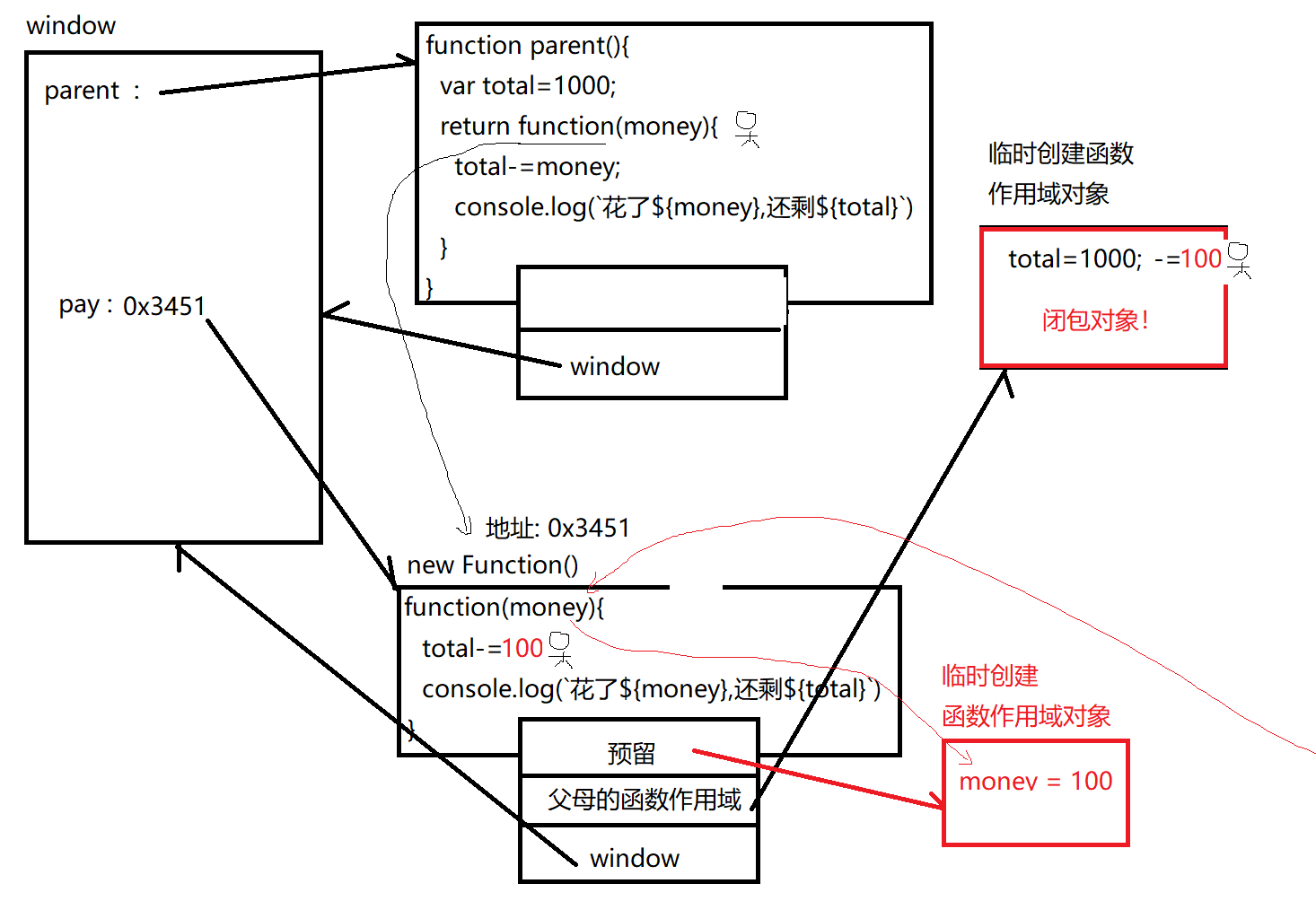
闭包3步:

(1). 用外层函数包裹要保护的变量和内层函数

(2). 外层函数将内层函数return到外部

(3). 外部的使用者需要调用外层函数，才能获得返回出来的内层函数对象。

(4).闭包有三个好友列表,最近的是临时创建的函数作用域,父母的作用域,window



**day04总结:面向对象 封装 继承 多态**

**封装:**

**1.创建对象的方式:**

1.var 变量名={属性名:属性值,属性名:方法function(){ 使用this访问当前属性}} ex方法内 this.属性名 this指向点前的对象 对象里没有作用域,只有方法才有

2.用new Object()创建

3.都能用下标方式访问对象中的成员属性，且都能用.简写下标访问:

lilei["sage"] 可简写为 lilei.sage

lilei.sage 会被自动翻译为 lilei["sage"]

强调: 如果要访问的属性名不是固定的，而是来自于其他变量或需要动态拼接，则~~不能用.~~，只能用[]，还不能加""

for(var key in lilei){

//key变量接住的值每一轮都在变化: sname sage intr className

console.log(`${key} : ${lilei[key]}`)

~~//console.log(`${key} : ${lilei["key"]}`) //变量不能加""~~

~~//console.log(`${key} : ${lilei.key}`) //.key会被自动翻译为["key"]~~

}

3.). 关联数组可通过强行赋值方式随时添加新元素，对象也可通过强行赋值方式随时添加新属性！而不会报错！

3.2). 访问关联数组中一个不存在的位置，不会报错，而是返回undefined。

同样，访问对象中一个不存在的属性，也不会报错！也返回undefined

3.3). 都能用for in遍历每个成员！

**2.浅克隆/深克隆**

浅克隆:

定义一个克隆函数 浅克隆只能克隆当前的属性不能克隆子对象里面的内容 即属性:{}

//判断传参是是否为空

//判断传参时是否为对象

//创建一个变量,判断传参时是否为数组 若是则输出[] 否则{}

//遍历传参的对象,并且赋值个新德对象

深克隆:

同上

多一步:在遍历oldObj时重新调用deepClone()遍历每一个属性是否存在子对象 ex:deepClone(oldObj[key])

看实例

**3.构造函数:**

何时:创建多个相同结构的对象 都用构造函数

负责反复创建该类型的子对象

1). 定义一个构造函数来描述一类对象统一的结构

i. function 类型名(形参1, 形参2,...){

this.属性名=形参1;

this.属性名=形参2;

~~this.方法名=function(){~~

~~... this.属性名 ...~~

~~//构造函数的方法定义在继承上,否则在函数体内创建方法会造成每次调用都要在内存里创建一个方法导致浪费内存~~

~~}~~

}

ii. 强调:

① 希望将来出现在新对象中的每个属性，都要用"this."定义

② 将来的新对象规定有几个属性，构造函数上就要定义几个形参变量，用来在将来创建新对象时，接受新对象的具体属性值。

2). 反复用new 调用构造函数，反复创建相同类型的多个对象

i. var 新对象=new 类型名(属性值1, 属性值2,...)

ii. 结果:

① new先创建一个空对象

②调用构造函数时的实参值会传给构造函数的形参变量

③构造函数中规定的所有属性，都会被添加到新对象内

④ new会返回新创建的对象，保存到=左边的变量中

如何访问对象中的成员: 通过.操作符访问

(1). 访问保存在一个对象中的属性值: 对象名.属性名

(2). 调用保存在一个对象中的方法: 对象名.方法名()

**继承:**

1. 问题: 构造函数只能重用代码结构，但是浪费内存。

(1). 因为凡是放在构造函数中的方法定义，每创建一个新对象，都要重复创建这个方法对象的副本！——浪费内存！

(2). 所以: 今后构造函数中不应该包含方法的定义！构造函数中应该只包含属性结构的定义。

(3). 所有对象都要使用的公共的方法定义在哪儿呢？——继承

**2.概述:**

2. 什么是继承: 父对象中的成员，子对象无需重复创建就可直接使用

3. 为什么: 既可重用代码，又可节约内存！

4. 何时: 只要同一类型多个对象都需要公共的方法定义时，就要用继承实现

5.负责存储该类型的子对象共用的方法

3.创建 构造函数的类型名.prototype.方法名=function(){}

(1)其实定义每个构造函数时都会附赠一个父对象——原型对象(prototype)

new会自动设置 新子对象.\_ \_proto\_ \_=构造函数.prototype

孩子 的 爹 是 妈妈 的 老公

(2). 将所有子对象公共的方法定义添加到共同的父对象原型对象中，只能通过强行赋值的方式:

构造函数.prototype.公共的方法=function(){ ... this.属性名 ...}

调用时，.前是谁！this就临时指谁！

所以，原型对象中公共函数里的this指将来调用这个函数的某一个子对象

3.自有属性和共有属性:

(1). 自有属性: 保存在子对象内部，归某个子对象独有的属性

(2). 共有属性: 保存在原型对象中，归多个子对象共有的属性

(3). 获取属性值时: 两种属性都可用"子对象.属性名"方式访问

(4). 修改属性值时:

a. 自有属性: 可以用"子对象.属性名=新值"方式修改

b. 共有属性: 必须用"原型对象.属性名=新值"方式修改

~~不能用"子对象.属性名=新值"方式修改~~

如果硬要用子对象修改共有属性的值，结果会为这个子对象隐式添加一个自有的同名属性。从此，在这一个属性的使用上，这个子对象和其他子对象，分道扬镳！再无关系！

4.原型对象为某种类型的所有子对象添加新共有方法(内置原型对象的方法11种)

a. 其实ES标准中已经有11种内置类型/对象:

String Number Boolean ——包装类型(看小程序中视频)

Array Date RegExp Math(不能new)

Error

Function Object

global(全局作用域对象，在浏览器中被window代替，不能new)

b. 除Math和global之外的9种内置类型都包含构造函数和原型对象, 比如:

1). Array其实是一个function Array(){ ... }构造函数

所以创建数组时可以: var arr=new Array()

其实[] 是 new Array()的简写

2). Array也一定有一个附赠的原型对象: Array.prototype

其中保存着所有数组共用的方法: .sort() .reverse() .push() .slice() ... ...

c. 如果数组家缺一个求和的函数sum()，就自己定义一个sum()函数放到数组的原型对象中

Array.prototype.sum=function(){

... ...

}

d. 结果: 所有数组家子对象都可享用这个原型对象中的sum函数了

e. 其实今后想了解一个类型中还有哪些好用的函数辅助我们开发，都可查看该类型的原型对象:

比如: String.prototype、Array.prototype等

5. 原型链

(1). 什么是原型链: 由多级父对象逐级继承形成的链式结果

(2). 保存着一个对象可用的所有属性，包括自有属性和共有属性

(3). 控制着属性的使用顺序: 先自有,再共有,然后再延原型链向上找！

总结: 目前为止this的用法: 4种:

1. obj.fun() fun()中的this->.前的obj对象

2. new Fun() Fun()中的this->new创建的新对象

3. fun() 或 (function(){ ... })() 中的this->window

4. Student.prototype.intr=function(){ ... this.sname ...} 中this->将来调用这个函数的某个子对象，比如lilei或hmm。

总结: 面向对象:

1. 封装:

(1). 只创建一个对象: {}

(2). 反复创建多个相同结构的对象: 构造函数

2. 继承:

(1). 今后所有子对象共有的方法和属性值，不要定义在构造函数中，而应该添加到原型对象中: 构造函数.prototype.共有方法=function(){ ...this.属性名... }