千锋教育
QIAN FENG EDUCATION

做真实的自己，用良心做教育

面向对象

H5教学部

目录

做真实的自己，用良心做教育

面向对象介绍

面向对象的编程思想

面向对象是利用对象进行编程的一种思想, 面向对象的编程思想是相对于面向过程的编程思想而言;

在面向对象的程序设计(英语:Object-oriented programming,缩写:OOP)中, 对象是一个由信息及对信息进行处理的描述所组成的整体, 是对现实世界的抽象. 在现实世界里我们所面对的事情都是对象, 如计算机、电视机、自行车等.

谈到面向对象的语言, 总会设计到对象的概念以及另一个概念: 类,一般的面向对象语言都通过类可以创建任意多个具有相同属性和方法的对象. 但是,ECMAScript没有类的概念，所以它的对象与其他基于类的语言中的对象有所不同.

ECMAScript有两种开发模式: 1.面向过程, 2.面向对象(OOP).

面向对象介绍

面向对象编程(OOP)的三个基本特征是：封装、继承、多态

封装：将属性和方法(数据和功能)封装在一起; //private,protect,public

继承：继承是指这样一种能力：它可以使用现有类的功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。

多态：允许让父类的指针分别指向不同的子类, 调用不同子类的同一个方法, 会有不同的执行效果(因为没有类, 所以多态在JS中运用较少)

面向对象介绍

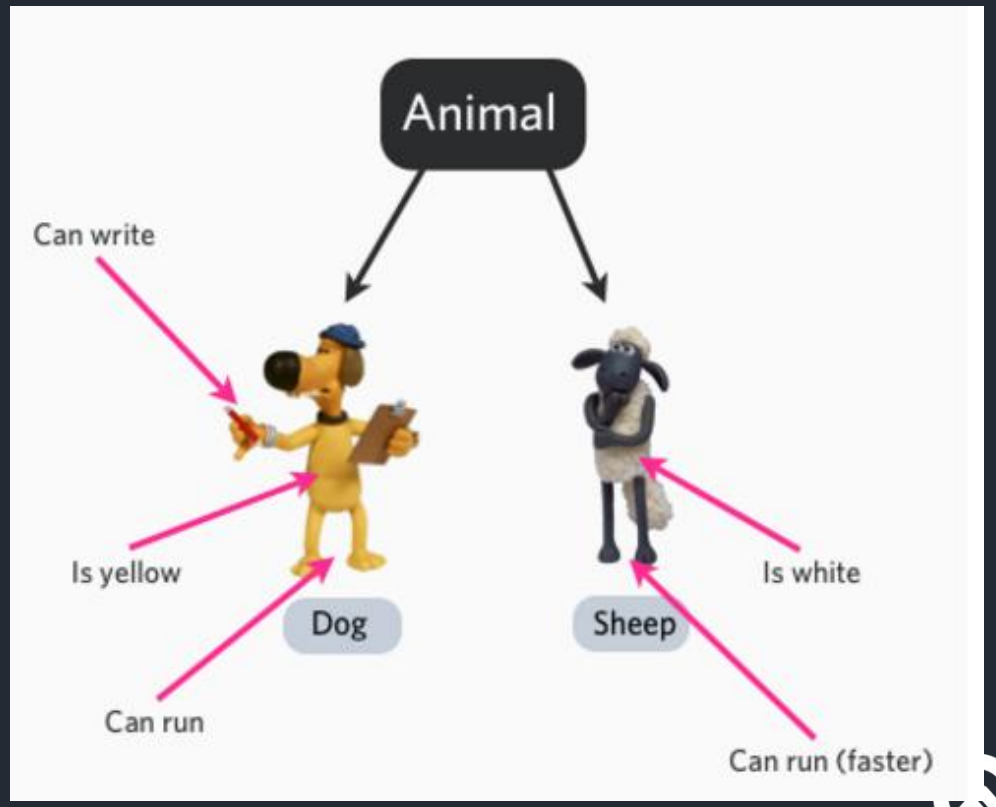
对象的组成

属性(对象的属性) ——变量：状态、静态的

方法(对象的行为) ——函数：过程、动态的

比如:

Animal(动物)是一个抽象类，我们可以具体到一只狗跟一只羊，而狗跟羊就是具体的对象，他们有颜色属性，可以写，可以跑等行为.



面向对象介绍

面向过程:

就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了。

面向对象:

是把构成问题的事务分解成各个对象，每个对象都有自己独立的属性和行为, 对象可以将整个问题事务进行分工, 不同的对象做不同的事情, 这种面向对象的编程思想由于更加贴近实际生活, 所以被计算机语言广泛应用。

面向对象介绍

例如: 五子棋游戏

面向过程的设计思路: 分析问题的步骤：1、开始游戏，2、黑子先走，3、绘制画面，4、判断输赢，5、轮到白子，6、绘制画面，7、判断输赢，8、返回步骤2到7、最后输出输赢的结果。把上面每个步骤用分别的函数来实现，问题就解决了.

而面向对象的设计则是从另外的思路来解决问题: 整个五子棋可以分为 1、黑白双方(这两方的行为是一模一样的)，2、棋盘系统(负责绘制画面)，3、规则系统(负责判定诸如犯规、输赢等). 第一类对象(玩家对象)负责接受用户输入，并告知第二类对象(棋盘对象)棋子布局的变化，棋盘对象接收到了棋子的变化就要负责在屏幕上面显示出这种变化，同时利用第三类对象(规则系统)来对棋局进行判定.

两种不同的思路都可以实现相同的功能, 但是面向对象的实现方式更加贴近人类生活(比如: 公司的各个部门, 学校的各个院校等). 在程序的编程中我们把项目中的每个不同的个体看成整体功能的每一个对象, 对象中再包含其属性和方法, 让对象对其属性和方法进行封装, 这样也更加利于后期扩展和维护. 现在几乎所有的语言都支持面向对象的编程思想.

面向对象介绍

示例:

小狗吃食（闻一闻smell、舔一舔lick、咬一咬bite）

分别采用面向过程和面向对象来分析

面向过程： 先闻一闻, 然后再舔一舔, 最后再咬一咬 (注重过程)

面向对象： 小狗是一个对象, 它可以闻一闻食物, 可以舔一舔食物, 可以咬一咬食物. (不注重过程, 注重对象)

面向对象介绍

面向过程的语言:

C语言等.

面向对象的语言:

C++, java, C#,

OC, JS, Swift等.

Jan 2017	Jan 2016	Change	Programming Language	Ratings	Change
1	1		Java	17.278%	-4.19%
2	2		C	9.349%	-6.69%
3	3		C++	6.301%	-0.61%
4	4		C#	4.039%	-0.67%
5	5		Python	3.465%	-0.39%
6	7	^	Visual Basic .NET	2.960%	+0.38%
7	8	^	JavaScript	2.850%	+0.29%
8	11	^	Perl	2.750%	+0.91%
9	9		Assembly language	2.701%	+0.61%
10	6	v	PHP	2.564%	-0.14%
11	12	^	Delphi/Object Pascal	2.561%	+0.78%
12	10	v	Ruby	2.546%	+0.50%
13	54	^^	Go	2.325%	+2.16%
14	14		Swift	1.932%	+0.57%
15	13	v	Visual Basic	1.912%	+0.23%
16	19	^	R	1.787%	+0.73%
17	26	^^	Dart	1.720%	+0.95%
18	18		Objective-C	1.617%	+0.54%
19	15	v	MATLAB	1.578%	+0.35%
20	20		PL/SQL	1.539%	+0.52%

面向对象介绍

创建对象

创建对象, 并添加对象所应该有的属性和方法

创建对象：var obj = new Object(); //也可以使用字面量的方式创建

添加属性：obj.name = 'zhangsan';

添加方法：obj.show = function(){

```
        console.log(this.name);
    }
```

工厂模式

为了解决多个类似对象声明的问题，我们可以使用一种叫做工厂模式的方法，这种方法就是为了解决实例化对象产生大量重复代码的问题

```
function createObject(name) {      //集中实例化的普通函数
    var obj = new Object();
    obj.name = name;
    obj.show = function(){
        alert(this.name)
    }
    return obj;
}
```

工厂模式解决了重复实例化的问题，但还有一个问题，那就是识别问题，因为根本无法搞清楚他们到底是哪个对象的实例。

```
console.log(typeof obj);  //object
console.log(obj instanceof Object); //true
```

构造函数

构造函数: 使用了new来创建对象的函数就是构造函数.

例如: 声明构造函数Person

```
function Person(name) {
    this.name = name;
    this.show = function(){
        console.log(this.name);
    }
}
var person = new Person( 'zhangsan' );      //new Person()即可
console.log(person instanceof Person);     //很清晰的识别他从属于 Box
```

构造函数和工厂模式的区别：

1. 构造函数方法没有显示的创建对象(在内部隐式调用了new Object())；
2. 直接将属性和方法赋值给 this 对象;
3. 没有 renturn 语句。

构造函数

构造函数和普通函数的区别:

构造函数与普通函数的唯一区别在于调用它们的方式, 任何函数, 只要通过new操作符来调用, 就可作为构造函数; 而任何函数, 若不通过new操作符调用, 就是普通函数。

例如:

```
var person = new Person('zhangsan');   //构造模式调用
Person('zhangsan');   //普通模式调用
```

类:

ECMAScript 没有类的概念, 我们可以把构造函数理解为就是类.

例如: 在下面的语句中, Person是类, person是对象; 实例化

```
var person = new Person( 'zhangsan' );
```

构造函数

构造函数中的方法：

```
var person1 = new Person(“张三”);  //传递一致
var person2 = new Person(“张三”);  //同上
console.log(person1.name == person2.name);  //true,属性的值相等
console.log(person1.show == person2.show);  //false,方法其实也是一种引用地址
```

构造函数的注意事项：

1, 函数名和实例化构造名相同且大写, (非强制，但这么写有助于区分构造函数和普通函数)；

2, 通过构造函数创建对象，必须使用 new 运算符。

构造函数

练习：创建以下构造函数(类)

Cat: 猫
属性：fur
方法：eat; miaow

GarfieldCat : 加菲猫
属性：fur; glasses
方法：eat; miaow; talk

TomCat : 汤姆猫
属性：fur; friend
方法：eat; miaow; catchMouse

构造函数

1, 匿名函数

匿名函数: 就是没有函数名字的函数

普通函数

```
function m1(){
    console.log("aa");
}
m1();
```

匿名函数

```
var aa = function(){
    console.log("aa");
}
aa();  //调用匿名函数, 这里的aa是函数名字
```

构造函数

函数中包含匿名函数

```
function bb(){
    return function(){
        console.log("cc");
    }
}
bb()();  //bb函数中的cc函数
```

构造函数

2, 匿名函数的自运行

声明函数的方式一: 普通函数

```
function m2(){
    console.log("m2");
}
m2();  //需要主动调用
```

声明函数方式二: 匿名函数的自运行， 不需要主动调用, 会直接执行

```
(function(){
    console.log("m3");
})();
```

构造函数

传参1

```
var m4 = (function(){
    return function bb(name){
        console.log(name);
    }
})();
m4( “lisi” );
```

传参2

```
var m5 = (function(a){
    return function(b){
        console.log(a+b);
    }
})(3)    //把3传给a
m5(4);  //把4传给b
```

闭包

1, 闭包的概念

闭包是这样一种机制: 函数嵌套函数, 内部函数可以引用外部函数的参数和变量,参数和变量不会被垃圾回收机制所收回.

这里涉及到几个概念:

1, 函数嵌套函数

2, 垃圾回收机制 (内存的自动释放),

3, 作用域(全局变量和局部变量)

闭包

2, 函数嵌套函数

```
function aa(){
    console.log("aa");
    function bb(){
        console.log("bb");
    }
}
aa();
bb(); //无法直接访问函数内部的函数
```

闭包

3, 垃圾回收机制

JS引擎会在一定的时间间隔来自动对内存进行回收(把内存释放)

JS垃圾回收机制有两种: 1, 标记清除, 2, 引用计数

1, 标记清除: js会对变量做一个标记Yes or No的标签以供js引擎来处理, 当变量在某个环境下被使用则标记为yes, 当超出该环境(可以理解为超出作用域)则标记为no, js引擎会在一定时间间隔来进行扫描, 会对有no标签的变量进行释放(将该变量所占的内存释放掉)

2, 引用计数: 对于js中引用类型的变量, 采用引用计数的内存回收机制, 当一个引用类型的变量赋值给另一个变量时, 引用计数会+1, 而当其中有一个变量不再等于值时, 引用计数会-1, 如果引用计数为0, 则js引擎会将其释放掉

闭包

4, 作用域(全局变量和局部变量)

```
//全局变量: a的作用域是全局, 内存不会被释放
var a = 1;
function m1(){
    a++;
    console.log(a);
}
m1(); //2
m1(); //3
```

闭包

```
//局部变量: b的作用域是函数m2, 调用完函数m2后就会被释放, 而在m2内部的局部变量无法被外部调用
function m2(){
    var b = 1;
    b++;
    console.log(b);
}
m2(); //2
m2(); //2
//console.log(b);
```

闭包

4, 闭包的写法

```
//函数嵌套函数
function aa(){
    var a = 1;
    function bb(){
        console.log(a);
    }
    return bb;  //返回函数bb
}
//console.log(a);   //无法直接访问a
var cc = aa();      //aa函数被执行了, 并返回了bb给cc
cc();     //可以打印出a, 说明a并没有被释放
```

闭包

前面的函数简化一下:

```
function aa(){
    var a = 5;
        return function(){
        a++;
        console.log("a=" + a);
    }
}
var cc = aa();  //aa函数被执行了, 并返回了内部的匿名函数给cc
cc();  //a=6
cc();  //a=7
cc();  //a=8
```

闭包

相比全局变量和局部变量, 闭包有两大特点：

1, 闭包拥有全局变量的不被释放的特点

2, 闭包拥有局部变量的无法被外部访问的特点

闭包的好处：

1. 可以让一个变量长期驻扎在内存当中不被释放

2. 避免全局变量的污染, 和全局变量不同, 闭包中的变量无法被外部使用

3. 私有成员的存在, 无法被外部调用, 只可以自己内部使用

闭包

5, 闭包的应用

例如：在循环中直接找到对应元素的索引

```
for (var i=0; i<aLi.length; i++) {
    (function(index){
        aLi[index].onclick = function(){
            console.log(index);
        }
    })(i);
}
```

闭包

或者:

```
//在for循环过程中将i传递到函数表达式中的index, index会成为一个局部变量,不会被释放,长期驻扎在内存中
for (var i=0; i<aLi.length; i++) {
        aLi[i].onclick = (function(index){
                return function(){
                        console.log(index);
                }
        })(i);
}
```

this

1, this关键字是什么

JS中的this到底指的是哪个对象?

先来看一段代码:

```
target.onclick = function() {
    console.log(this);
}
```

这里的this会打印出什么?

很明显, 这里的this指的是target本身

this

JS中的this到底指的是当前对象，那么这个当前对象指的又是什么？

this在不同的代码环境下代表的对象是不一样的，就像说话的语义一样:

如果有个人对你喊：开!

当你手里有一把枪, 你会开枪;

当你坐在车上, 你会开车;

当你站在阳台上, 你会开窗...

this

this关键字: 指的是此刻正在运行的函数所依附的对象

```
target.onclick = function() {
    console.log(this);
}
var obj = {};
obj.test = target.onclick;
target.onclick();  //结果是target对象
obj.test();  //结果是obj对象
```

this好比一句话, 出自不同人之口, 代表的人就不一样

A和B吵架； A对B说: “老子要砍死你!”， 这里的老子指A

B对A说: “老子要弄死你!”， 这里的老子指B

this

2, 定时器中的this

定时器setInterval()中的匿名函数是在被全局中的window对象调用的, 所以里面的this指的就是window对象, 而不是box

```
box.onclick = function() {
        console.log(this);      //box对象
        setInterval(function(){
                console.log(this); //window对象
        }, 2000);
        setTimeout(function(){
                console.log("setTimeout:" + this);  //window对象
        }, 3000);
}
```

this

3, 构造函数中的this

在构造函数中的this是使用new关键字创建的那个对象,

下面的this是box对象

```
function Box(name) {
    this.name = name;
    this.show = function(){
        console.log(this.name);
    }
}
var box = new Box('zhangsan');
```

this

4, 普通函数中的this

普通函数调用

```
function func() {
        console.log(this);
}
func();  //window对象
```

练习

1, 采用面向对象的思路实现拖拽功能

2, 采用工厂模式, 创建一个函数, 将拖拽功能封装在函数内

3, 采用构造函数, 创建一个构造函数, 将拖拽功能封装在函数内

练习

4，创建一个萤火虫Fireworm类, 其中包括一个节点属性ele和飞的方法fly(), 每隔1秒创建一个萤火虫对象, 显示在屏幕的随机位置,然后自动飞到屏幕的其他随机位置,飞完后再继续飞...

THANK YOU

千锋教育
QIAN FENG EDUCATION

做真实的自己，用良心做教育