"类"设计模式和"原型"设计模式——"复制"和"委托"的差异

掘金安东尼 2021-05-21 1,912 阅读8分钟 关注

小引

JavaScript 技能持有者一定有问过这个问题：

> JavaScript 是面向对象语言吗？

你期望得到的答案应该为："是" 或 "不是"。

但是可惜，你得不到这样简单的答案！

你大概了解一通之后，你会被告知：

> JavaScript 不是纯粹的面向对象语言！

wtf！为什么是不纯粹？能不能纯粹一点？！我们喜欢纯粹，不喜欢混沌！

.....

实际上，死扣定义真的没太必要。定义背后的故事才是最重要的！

看完本篇，你就会明白这种"混沌"是什么、来自何处，以及去往何方！！

撰文不易，多多鼓励。点赞再看，养成习惯。👍👍👍

"类"设计模式

1. 所谓封装，即把客观事物封装成抽象的类。

2. 所谓继承，即子类继承父类的能力。

3. 所谓多态，即子类可以用更特殊的行为重写所继承父类的通用行为。

其中，"类"的概念最最关键！**【类】描述了一种代码的组织结构形式，它是软件中对真实世界中问题领域的建模方法。**

举个例子：

就好比我们现实中修房子，你要修一个"写字楼"、或者一个"居民楼"、或者一个"商场"，你就得分别找到修"写字楼"、"居民楼"、"商场"的【设计蓝图】。

但是设计蓝图只是一个建筑计划，并不是真正的建筑。要想在真实世界实现这个建筑，就得由建筑工人将设计蓝图的各类特性（比如长宽高、功能）【复制】到现实世界来。

这里的【设计蓝图】就是【类】，【复制】的过程就是【实例化】，【实例】就是【对象】。

类的内部通常有一个同名的构造方法，我们设想下，它的伪代码就可能是这样的：

scss 复制代码

```scss
class Mall { // "商场"类

    Mall( num ){ // 同名构造方法
        garage = num // 地下车库数量
    }

    shop( goods ) { // 买东西
       output( "We can buy: ", goods )
    }

}

// 构造函数大多需要用 new 来调，这样语言引擎才知道你想要构造一个新的类实例。
vanke = new Mall(1) // vanke 有 1 个地下车库

vanke.shop("KFC") // "We can buy: KFC"

```

java 是典型的面向对象语言。基于"类"，我们再通过以下一段 java 代码来看看对**继承**和**多态**的理解。

scala 复制代码

```scala
public abstract class Animal{ // 抽象类
     abstract void sound();
}
```

```
        sound("咯咯咯");
    }
}
public class Duck extends Animal{
    public void sound(){
        sound("嘎嘎嘎");
    }
}

public static void main(String args[]){
  Aninal chicken = new Chicken();
  Animal duck = new Duck();
  chicken.sound(); //咯咯咯
  duck.sound();  //嘎嘎嘎
}
```

鸡和鸭都属于动物分类，都可以发出叫声（继承），但是它们却可以发出不同的叫声（多态），很容易理解。

继承可以使子类获得父类的全部功能； 多态可以使程序有良好的扩展；

回想下：在 JS 中，我们可能会怎样写：

javascript 复制代码

```
var Duck = function () {};
var Chicken = function () {};
var makeSound = function ( animal ) {
    if( animal instanceof Duck){
        console.log("嘎嘎嘎");
    }else if( animal instanceof Chicken){
        console.log("咯咯咯");
    }
};
makeSound(new Duck());
makeSound(new Chicken());
```

这里既没用到继承，也没用到多态。这样【写判断】是代码"不清爽"的罪魁祸首！

- 此处留一个疑问，如果不用判断，还可以怎么写？

在 vue2 中，我们可能会这么写：

scss 复制代码

```
export default {
  data() {
      return {

```

```
            this.Chicken()
            this.Duck()
        },
        methods:{
            funtion AnimalSound(sound){
                console.log("叫声:" + sound)
            },
            funtion Chicken(){
                this.AnimalSound("咯咯咯")
            },
            funtion Duck(){
                this.AnimalSound("嘎嘎嘎")
            }
        }
    }
```

像这种函数嵌套调用是很常见的。没有看到继承，也没有看到多态，甚至都没有看到最根本的"类"？！

（实际上，每个函数都是一个 Function 对象。按照最开始定义所述，**对象是类的实例**，所以也是能在函数中看到"类"的！）

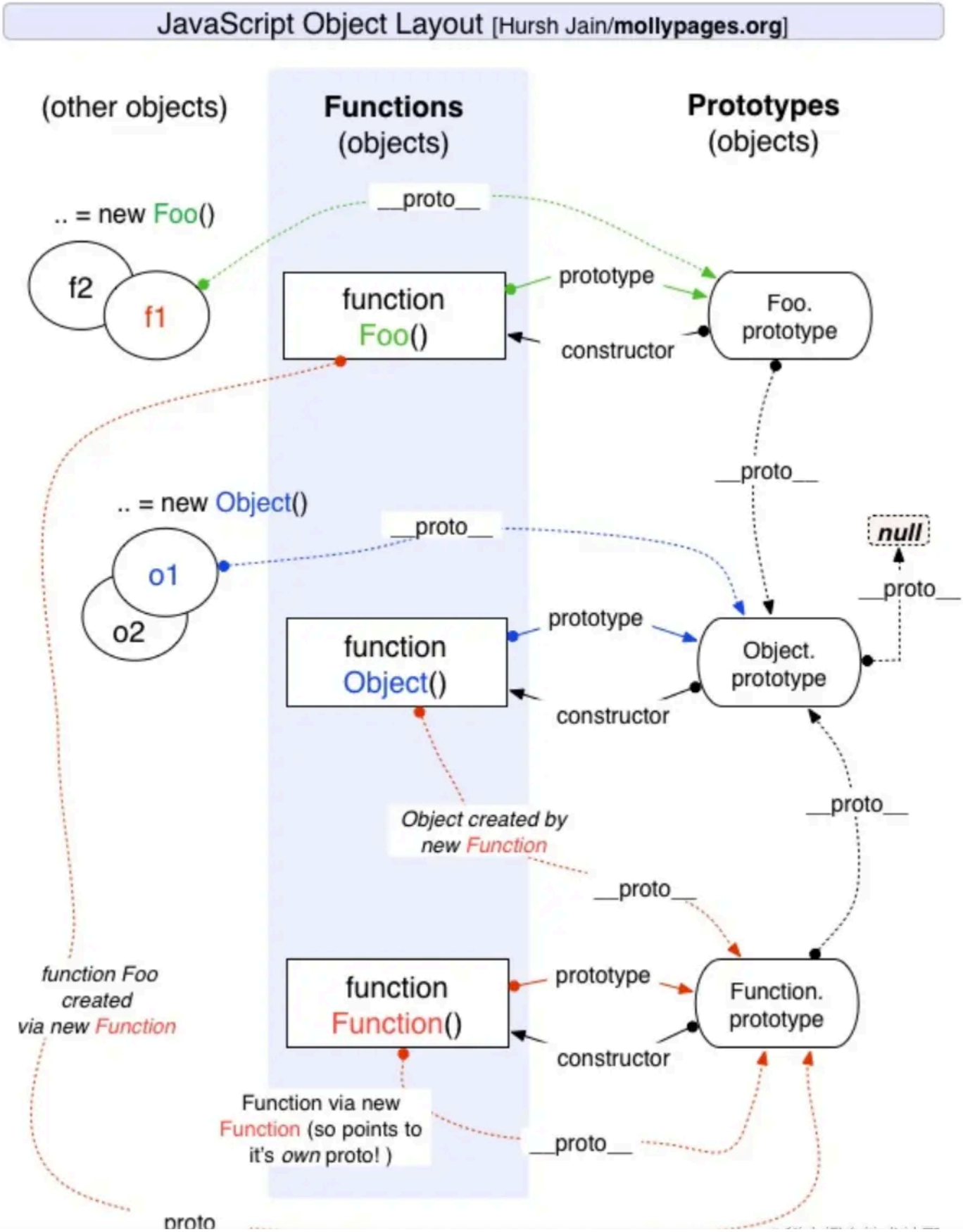
在 JavaScript 中，**函数成了第一等公民！** 函数似乎什么都能做！它可以返回一个对象，可以赋值给一个变量，可以作为数组项，可以作为对象的一个属性.....

但这明显不是"类的设计模式"吧！

"类的设计模式" 意味着对【设计蓝图】的【复制】，在 JS 各种函数调用的场景下基本看不到它的痕迹。

"原型"设计模式

一图看懂原型链?


JavaScript Object Layout [Hursh Jain/mollypages.org]
(other objects)
Functions (objects)
Prototypes (objects)
.. = new Foo()
f2
f1
__proto__
function Foo()
prototype
constructor
Foo. prototype
__proto__
.. = new Object()
o1
o2
__proto__
null
__proto__
function Object()
prototype
constructor
Object. prototype
__proto__
Object created by new Function
__proto__
function Foo created via new Function
function Function()
prototype
constructor
Function. prototype
Function via new Function (so points to it's own proto!)
__proto__
proto

看不懂？没关系，记住这两句话再来看:

1. **一个对象的显示原型的构造函数指向对象本身**（很熟悉有没有？在本文哪里见过？）
2. **一个对象的隐式原型指向构造这个对象的函数的显示原型。**

原来，JS 不是通过在类里面写同名构造函数的方式来进一步实现的实例化，它的构造函数在原型上！这种更加奇特的代码服用机制有异于经典类的代码复用体系。

这里再附一个经典问题？JS new 操作会发生什么?

会是像类那样进行复制吗?

答案是否定的!

JS 访问一个对象的属性或方法的时候，先在对象本身中查找，如果找不到，则到原型中查找，如果还是找不到，则进一步在原型的原型中查找，一直到原型链的最末端。复制不是它所做的，这种查找的方式才是！对象之间的关系更像是一种委托关系，就像找东西，你在我这找不到？就到有委托关系的其它人那里找找看，再找不到，就到委托委托关系的人那里找.....直至尽头，最后还找不到，指向 null。

> 所以：JavaScript 和面向对象的语言不同，它并没有类来作为对象的抽象模式或者设计蓝图。JavaScript 中只有对象，对象直接定义自己的行为。对象之间的关系是委托关系，这是一种极其强大的设计模式。**在你的脑海中对象并不是按照父类到子类的关系垂直组织的，而是通过任意方向的委托关联并排组织的!**

不过你也可以通过这种委托的关系来模拟经典的面向对象体系：类、继承、多态。但"类"设计模式只是一种可选的设计模式，你可以模拟，也可以不模拟!

现实是 ES6 class 给我们模拟了:

kotlin 复制代码

```kotlin
class Widget {
    constructor(width,height) {
        this.width = width || 50;
        this.height = height || 50;
        this.$elem = null;
    }
    render($where){
        if (this.$elem) {
            this.$elem.css( {
                width: this.width + "px",
                height: this.height + "px"
            }).appendTo( $where );
```

```
}
class Button extends Widget {
    constructor(width,height,label) {
        super( width, height );
        this.label = label || "Default";
        this.$elem = $( "<button>" ).text( this.label );
    }
    render($where) {
        super.render( $where );
        this.$elem.click( this.onClick.bind( this ) );
    }
    onClick(evt) {
        console.log( "Button '" + this.label + "' clicked!" );
    }
}
```

看起来，非常不错，很清晰！

没有 .prototype 显示原型复杂的写法，也无需设置 .proto 隐式原型。还似乎用 extends 、super 实现了继承和多态。

然而，这只是语法糖的陷阱！JS 没有类，没有复制，它的机制是"委托"。

class 并不会像传统面向类的语言一样在申明时作静态复制的行为，如果你有意或者无意修改了父类，那子类也会收到影响。

举例：

javascript 复制代码

```
class C {
 constructor() {
    this.num = Math.random();
 }
 rand() {
    console.log( "Random: " + this.num );
 }
}
var c1 = new C();
c1.rand(); // "Random: 0.4324299..."
C.prototype.rand = function() {
    console.log( "Random: " + Math.round( this.num * 1000 ));
};
var c2 = new C();
c2.rand(); // "Random: 867"
c1.rand(); // "Random: 432" —噢!
```

ES6 class 混淆了"类设计模式"和"原型设计模式"。它最大的问题在于，它的语 法有时会让你认为，定义了一个 class 后，

> 总地来说，ES6 的 class 想伪装成一种很好的语法问题的解决方案，但是实际上却让问题更难解决而且让 JavaScript 更加难以理解。 —— 《你不知道的 JavaScript》

小结

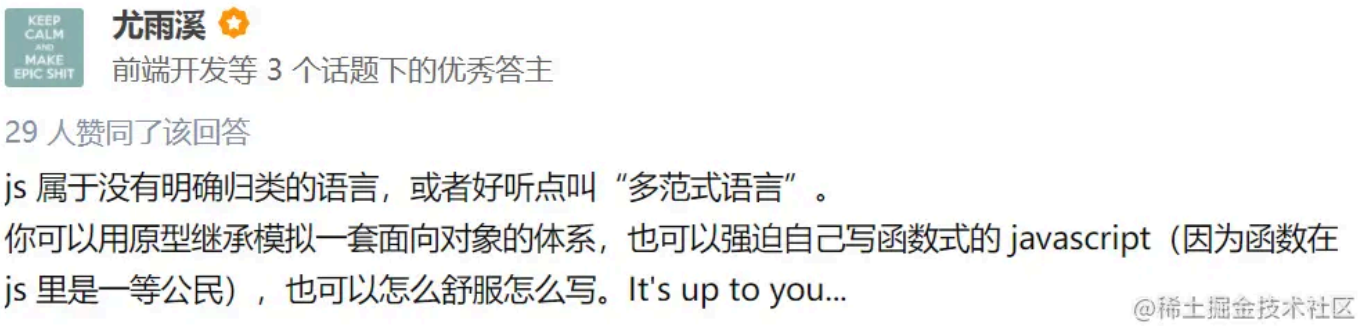
- "类设计模式"的构造函数挂在同名的类里面，类的继承意味着复制，多态意味着复制 + 自定义。
- "原型设计模式"的构造函数挂在原型上，原型的查找是一种自下而上的委托关系。
- "类设计模式"的类定义之后就不支持修改。
- "原型设计模式"讲究的是一种动态性，任何对象的定义都可以修改，这和 JavaScript 作为脚本语言所需的动态十分契合！

你可以用"原型设计模式"来模拟"类设计模式"，但是这大概率是得不偿失的。

最后，如果再被问道：JavaScript 是面向对象语言吗？

如果这篇文章看懂了，就可以围绕："类设计模式"和"原型设计模式"来吹了。

如果本文没有看懂，就把下面的标答背下来吧.....



关注公众号《掘金安东尼》，持续输出ing！！！

最近有点疲于写业务代码 .....谁能支支招?

参考文献

- 命名函数表达式探秘
- 函数式和面向对象编程有什么区别?
- tutorials/js.mp

标签： JavaScript 前端

本文收录于以下专栏

JavaScript【掘金安东尼】 专栏目录

JavaScript 核心，我更关心"如何使用闭包"以及"如何充分利用异步"！

369 订阅 · 116 篇文章

订阅

上一篇 medium 五万赞好文-《我永远不懂 JS ...

下一篇 XDM，JS如何函数式编程？看这就够了...

评论 1

平等表达，友善交流

0 / 1000 发送

最热 最新

AnneSuzy LV.1 前端工程师 @secret

感觉像个病毒一样~~!! 颇有种株连九族的感觉

1年前 点赞 评论

目录 收起

黑白手绘线稿图变3D彩色粒子，带你用Three.js Shader一步步实现（上）
139阅读 · 2点赞

AIGC学习分享（二）：openAI（API接口）它来了！
299阅读 · 25点赞

5分钟！搭建自己专属的AI应用！
213阅读 · 20点赞

高德地图导航咋知道还有几秒变绿灯？
284阅读 · 1点赞

COZE智能应用专属卡片的配置详细教程，它来了！
320阅读 · 15点赞

为你推荐

推荐 10 个很"哇塞"的Web"资源"给前端工友，收藏等于学会~
掘金安东尼 2年前 75k 3.8k 239 前端 设计 前端框架

Vue(v2.6.11)万行源码生啃，就硬刚！
掘金安东尼 3年前 60k 1.7k 177 Vue.js

🍩 想要白嫖正则是吧？这一次给你个够！
掘金安东尼 1年前 56k 1.6k 204 前端 JavaScript 面试

感谢 compose 函数，让我的代码屎山 💩 逐渐美丽了起来~
掘金安东尼 2年前 63k 981 175 JavaScript 前端 面试

👳 面试官：工作两年了，这么简单的算法题你都不会？
掘金安东尼 1年前 66k 482 128 前端 JavaScript 面试

正则什么的，你让我写，我会难受，你让我用，真香！
掘金安东尼 1年前 48k 751 65 前端 JavaScript 正则表...

推荐 5 个你大概率没见过的免费 API ，一键获取数据！
掘金安东尼 1年前 41k 502 36 前端 API JavaScript

1234 再来一次，继续分享新 10 个"哇塞"的 web 资源，收藏等于学会~
掘金安东尼 2年前 26k 1.2k 69 前端 设计 前端框架

为什么我更推荐 Notion AI 胜于 ChatGPT？

把收藏力拉满，前端 50 个优质 Web 在线资源~

掘金安东尼 2年前 35k 509 35 前端 JavaScript CSS

免费用 GPT4，为啥这些网站的聚合做的这么好？！

掘金安东尼 1年前 33k 343 88 人工智能 ChatGPT GitHub

Vue 虚拟列表，纵享丝滑【实践篇】

掘金安东尼 3年前 31k 303 36 Vue.js

VS Code settings.json 10 个高（装）阶（杯）配置！

掘金安东尼 2年前 29k 179 19 前端 JavaScript Visual S...

万字年中总结，共勉

掘金安东尼 1年前 25k 237 134 前端 JavaScript 面试

写出干净的 JavaScript 5 个小技巧

掘金安东尼 1年前 20k 328 60 前端 JavaScript 面试