



js的class真的仅仅是一颗语法糖吗?



丁盛

转弯曲

关注

6 人赞同了该文章

前言：本人高龄，想重拾n年前一个被耽搁的梦想：用js开发一款web编辑器，核心功能定位于能支持图形符号的插入、连接、公式输入。结果掉入了js早就为我挖好的prototye原型继承机制大坑之中，几经折腾，本以为阮一峰大师已为我打通任督二脉，于是兴致勃勃重入coding世界，恰道是好事多磨.....，于是有了这篇文章，也许会成系列文章，以铭记js带给我的这些伤痛...

先看看我开发中遇到的一个场景：A对应于一个图形符号的基类，C是A的派生类---某个具体的图形符号类（比如圆）。图形符号的connector作为起止点，用于画符号间的连接线，另有一个函数负责向C的实例对象中加入connector。我写下了类似的代码：

```
function A(){
  this.connectors=[];
  return this;
}
function C(name){
  this.name=name;
  return this;
}
C.prototype=new A();
function f(owner,cn){
  owner.connectors.push(cn);
  return this;
}
c1=new C('c1');
c2=new C('c2');
f(c1,1);
f(c1,2);
f(c2,3);
f(c2,4);
f(c2,5);
console.log("c1.connectors:")
c1.connectors.forEach(function(t){console.log(t)})//会在控制台输出1, 2, 3, 4, 5
console.log("c2.connectors:")
c2.connectors.forEach(function(t){console.log(t)})//会在控制台输出1, 2, 3, 4, 5
```

但是，你会发现c1,c2对象的connectors中的内容完全相同，都是1, 2, 3, 4, 5，这不是我想要的结果，按C++继承机制，我想要的结果是c1的connectors的内容为1, 2；c2的connector的内容为3, 4, 5。高手们可能已经看出问题之所在，这是原型继承的坑！语句C.prototype=new A()采用是原型继承，C函数的所有实例访问的都是同一个A的实例中的connectors，所以你得不到C++继承机制的结果！

知乎

方式:

```
class A{
  constructor(){
    this.connectors=[];
    return this;
  }
}
class C extends A{
  constructor(name){
    super();
    this.name=name;
    return this;
  }
}
function f(owner,cn){
  owner.connectors.push(cn);
  return this;
}
c1=new C('c1');
c2=new C('c2');
f(c1,1);
f(c1,2);
f(c2,3);
f(c2,4);
f(c2,5);
console.log("c1.connectors:")
c1.connectors.forEach(function(t){console.log(t)})//会在控制台输出1, 2
console.log("c2.connectors:")
c2.connectors.forEach(function(t){console.log(t)})//会在控制台输出3, 4, 5
```

也就意味着在class继承方式, c1,c2中的connectors是两个不同的对象, 而这正是我想要的结果, 所以es6的class不仅仅是一颗语法糖, 对初学者可大大减小js面向对象编程的学习成本。

当然, 要不用class而实现同样的效果, 大神早已找到了方法, 那就是apply调用, 但是, 如下代码中, 你还能直观看出C与A的继承关系吗?

```
function A(){
  this.connectors=[];
  return this;
}
function C(name){
  A.apply(this, arguments);
  this.name=name;
  return this;
}
function f(owner,cn){
  owner.connectors.push(cn);
  return this;
}
c1=new C('c1');
c2=new C('c2');
f(c1,1);
f(c1,2);
f(c2,3);
f(c2,4);
f(c2,5);
console.log("c1.connectors:")
c1.connectors.forEach(function(t){console.log(t)})//会在控制台输出1, 2
console.log("c2.connectors:")
c2.connectors.forEach(function(t){console.log(t)})//会在控制台输出3,4, 5
```

看来es6引入class的面向对象编程方式不仅仅是安慰婴儿的语法糖, 从降低学习成本、提高代码的可读性、可维护性都有现实意义, 遗憾的是, 我们不得不站在巨人的肩膀上, 不得不面对一坨坨充满prototype继承的遗留代码, 你不得不用prototype去继承遗产, 你不得不学习prototype。! 那么, 可不可以用class 去继承 prototype 的遗产呢? 答案是肯定的, 这也就是本文要讲的内容, 也是ES6

知乎

```
function A(){

  this.connectors=[];

}

/**class C: a wrapper of function A*****/
class Wrapper {
  constructor(){
    A.apply(this,arguments)
    return this;
  }
}

class C extends Wrapper{
  constructor(name){
    super()
    this.name=name;
    return this;
  }
}

function f(owner,cn){
  owner.connectors.push(cn);
  return this;
}

c1=new C('c1');
c2=new C('c2');
f(c1,1);
f(c1,2);
f(c2,3);
f(c2,4);
f(c2,5);
console.log("c1.connectors:")
c1.connectors.forEach(function(t){console.log(t)})//会在控制台输出1, 2
console.log("c2.connectors:")
c2.connectors.forEach(function(t){console.log(t)})//会在控制台输出3,4, 5
```

重点：通过Wrapper类对遗留代码中的作为基类的某个函数（比如A）的封装，可以完全用ES6的class语法来创建A的派生类，Wrapper类阻断了函数式继承中prototype的无限传播。

也许在你的项目组中，将来只需要1~2个搞清prototype的程序员去封装遗留代码，其他程序员可以抛开prototype，用class去就创作一个新世界！

编辑于 2019-12-16 01:51

[JavaScript 编程](#) [JavaScript权威指南（书籍）](#)



欢迎参与讨论

3 条评论

默认 最新



Kero

实际上就是语法糖☺️ 参照MDN Web文档
2020-04-26

回复 1



丁盛 作者

算写完了，欢迎交流
2019-12-16

回复 喜欢



丁盛 作者

匆匆写完，尚有

推荐阅读



Op-Id	Op-Id	Op-Id	Description
14	14	14	14
15	15	15	15
16	16	16	16

从 V8 源码理解 Javascript 函数是一等公民

徐鹏跃



【自然语言处理】收藏！使用 Python 的4种句嵌入技术

字七 发表于【自然语言...



如何在 JavaScript 中运行 C / C++ 语言？【WebAssembl...

邵励治



JS语法入门1

For955