JavaScript深入之继承的多种方式和优缺点

冴羽 Lv6

2017年05月12日 02:54 · 阅读 15169

> JavaScript深入系列第十五篇，讲解JavaScript各种继承方式和优缺点。

写在前面

本文讲解JavaScript各种继承方式和优缺点。

但是注意：

这篇文章更像是笔记，哎，再让我感叹一句：《JavaScript高级程序设计》写得真是太好了！

1.原型链继承

复制代码

```
function Parent () {
    this.name = 'kevin';
}

Parent.prototype.getName = function () {
    console.log(this.name);
}

function Child () {

}

Child.prototype = new Parent();

var child1 = new Child();
```

问题：

1.引用类型的属性被所有实例共享，举个例子：

复制代码

```
function Parent () {
    this.names = ['kevin', 'daisy'];
}

function Child () {

}

Child.prototype = new Parent();

var child1 = new Child();

child1.names.push('yayu');

console.log(child1.names); // ["kevin", "daisy", "yayu"]

var child2 = new Child();

console.log(child2.names); // ["kevin", "daisy", "yayu"]
```

2.在创建 Child 的实例时，不能向Parent传参

2.借用构造函数(经典继承)

复制代码

```
function Parent () {
    this.names = ['kevin', 'daisy'];
}

function Child () {
    Parent.call(this);
}

var child1 = new Child();

child1.names.push('yayu');

console.log(child1.names); // ["kevin", "daisy", "yayu"]

var child2 = new Child();

console.log(child2.names); // ["kevin", "daisy"]
```

优点：

1.避免了引用类型的属性被所有实例共享

2.可以在 Child 中向 Parent 传参

举个例子：

复制代码

```
function Parent (name) {
```

```
}

function Child (name) {
    Parent.call(this, name);
}

var child1 = new Child('kevin');

console.log(child1.name); // kevin

var child2 = new Child('daisy');

console.log(child2.name); // daisy
```

缺点：

方法都在构造函数中定义，每次创建实例都会创建一遍方法。

3.组合继承

原型链继承和经典继承双剑合璧。

复制代码

```
function Parent (name) {
    this.name = name;
    this.colors = ['red', 'blue', 'green'];
}
```

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

function Child (name, age) {

    Parent.call(this, name);

    this.age = age;

}

Child.prototype = new Parent();

var child1 = new Child('kevin', '18');

child1.colors.push('black');

console.log(child1.name); // kevin
console.log(child1.age); // 18
console.log(child1.colors); // ["red", "blue", "green", "black"]

var child2 = new Child('daisy', '20');

console.log(child2.name); // daisy
console.log(child2.age); // 20
console.log(child2.colors); // ["red", "blue", "green"]
```

优点：融合原型链继承和构造函数的优点，是 JavaScript 中最常用的继承模式。

```
function createObj(o) {
    function F(){}
    F.prototype = o;
    return new F();
}
```

就是 ES5 Object.create 的模拟实现，将传入的对象作为创建的对象的原型。

缺点：

包含引用类型的属性值始终都会共享相应的值，这点跟原型链继承一样。

```
var person = {
    name: 'kevin',
    friends: ['daisy', 'kelly']
}

var person1 = createObj(person);
var person2 = createObj(person);

person1.name = 'person1';
console.log(person2.name); // kevin

person1.firends.push('taylor');
console.log(person2.friends); // ["daisy", "kelly", "taylor"]
```

注意：修改 `person1.name` 的值， `person2.name` 的值并未发生改变，并不是因为 `person1` 和 `person2` 有独立的 name 值，而是因为 `person1.name = 'person1'`，给 `person1` 添加了 name 值，并非修改了原型上的 name 值。

5. 寄生式继承

创建一个仅用于封装继承过程的函数，该函数在内部以某种形式来做增强对象，最后返回对象。

复制代码

```
function createObj (o) {
    var clone = object.create(o);
    clone.sayName = function () {
        console.log('hi');
    }
    return clone;
}
```

缺点：跟借用构造函数模式一样，每次创建对象都会创建一遍方法。

6. 寄生组合式继承

为了方便大家阅读，在这里重复一下组合继承的代码：

复制代码

```
function Parent (name) {
    this.name = name;
```

```
}

Parent.prototype.getName = function () {
    console.log(this.name)
}

function Child (name, age) {
    Parent.call(this, name);
    this.age = age;
}

Child.prototype = new Parent();

var child1 = new Child('kevin', '18');

console.log(child1)
```

组合继承最大的缺点是会调用两次父构造函数。

一次是设置子类型实例的原型的时候：

复制代码

```
Child.prototype = new Parent();
```

一次在创建子类型实例的时候：

复制代码

```
var child1 = new Child('kevin', '18');
```

回想下 new 的模拟实现，其实在这句中，我们会执行：

```
Parent.call(this, name);
```

复制代码

在这里，我们又会调用了一次 Parent 构造函数。

所以，在这个例子中，如果我们打印 child1 对象，我们会发现 Child.prototype 和 child1 都有一个属性为 `colors`，属性值为 `['red', 'blue', 'green']`。

那么我们该如何精益求精，避免这一次重复调用呢?

如果我们不使用 Child.prototype = new Parent() ，而是间接的让 Child.prototype 访问到 Parent.prototype 呢?

看看如何实现：

```
function Parent (name) {
    this.name = name;
    this.colors = ['red', 'blue', 'green'];
}

Parent.prototype.getName = function () {
    console.log(this.name)
}

function Child (name, age) {
```

复制代码

```
}

// 关键的三步
var F = function () {};

F.prototype = Parent.prototype;

Child.prototype = new F();


var child1 = new Child('kevin', '18');

console.log(child1);
```

最后我们封装一下这个继承方法：

复制代码

```
function object(o) {
    function F() {}
    F.prototype = o;
    return new F();
}

function prototype(child, parent) {
    var prototype = object(parent.prototype);
    prototype.constructor = child;
    child.prototype = prototype;
}

// 当我们使用的时候：
```

引用《JavaScript高级程序设计》中对寄生组合式继承的夸赞就是：

这种方式的高效率体现它只调用了一次 Parent 构造函数，并且因此避免了在 Parent.prototype 上面创建不必要的、多余的属性。与此同时，原型链还能保持不变；因此，还能够正常使用 instanceof 和 isPrototypeOf。开发人员普遍认为寄生组合式继承是引用类型最理想的继承范式。

相关链接

《JavaScript深入之从原型到原型链》

《JavaScript深入之call和apply的模拟实现》

《JavaScript深入之new的模拟实现》

《JavaScript深入之创建对象》

深入系列

JavaScript深入系列目录地址：github.com/mqyqingfeng...。

JavaScript深入系列预计写十五篇左右，旨在帮大家捋顺JavaScript底层知识，重点讲解如原型、作用域、执行上下文、变量对象、this、闭包、按值传递、call、apply、bind、new、继承等难点概念。

分类： 前端 标签： JavaScript 前端

文章被收录于专栏：

JavaScript 深入系列

旨在帮大家捋顺 JavaScript 底层知识，重点讲解如原型、作用域、执行上下文、变量对象、this、闭包、按值传递、call...

关注专栏

评论

输入评论（Enter换行，Ctrl + Enter发送）

热门评论

哈娄 前端开发工程师 @ 天狮... 2年前

在组合式寄生继承的方式中，为什么还要创造一个F的匿名函数，直接Child.prototype=Parent.prototype不可以吗?

点赞 3

almostly 2年前

这样Child的实例的构造函数的constructor指向的是Parent，应该是Child,所以不推荐这样

点赞 回复

倔强的小码农 Lv1 2年前

当我们想给 Child 的prototype里面添加共享属性或者方法时，如果其 prototype 指向的是 Parent 的 prototype，那么在 Child 的 prototype 里添加的属性和方法也会反映在 Parent 的 prototype 里面，这明显是不合理的，这样做的后果是当我们只想使用 Parent 时，也能看见 Child 往里面扔的方法和属性。所以需要每个构造函数都需要持有自己专用的prototype对象

4 回复

查看更多回复

259 37 收藏

chuck Lv1 前端开发 @ 南京 4年前

楼主，你终于更新完了，我可以总体看一遍了 多谢多谢

3 1

全部评论 37

最新 最热

秦淮酒家 前端 8月前

第3小结组合继承, 有些文章会在`Child.prototype = new Parent();`后加一行`Child.prototype.constructor = Child` 用来修正Child的Constructor的指向, 为什么这里没有呀?

1 回复

yokodak 1年前

"这种方式的高效率体现它只调用了一次 Parent 构造函数，并且因此避免了在 Parent.prototype 上面创建不必要的、多余的属性。"
这里不应该是避免了在Child.prototype上创建不必要的属性么，因为调用Child.prototype = new Parent();并没有在Parent.prototype上创建属性，而是在Child.prototype添加了多余的colors以及name属性

1 回复

Cheri49721 前端工程师 1年前

最后一个封装继承不到构造函数里的属性啊

点赞 1

Cheri49721 1年前

259 37 收藏

点赞 回复

三杯秋 1年前

注意：修改person1.name的值，person2.name的值并未发生改变，并不是因为person1和person2有独立的 name 值，而是因为person1.name = 'person1'，给person1添加了 name 值，并非修改了原型上的 name 值。
请问一下：person1 访问 name值是“kevin”，person1.__proto__ = { name: "kevin", friends: [...] }。person1 设置 name 值为啥不是person1的__proto__里面的name呢？按照变量访问的规则，不是到上一层去找吗，既然 person1 的 __proto__ 里面找得到，那设置却不是这个。最后 person1 却是这样的 person1 {name: "新的值"} ，person1.__proto__ {name: "kevin", friends: [.....] } 。

展开

点赞 2

chengxi_24 1年前

person1.name = 'person1'是给person1这个对象本身添加name属性并赋值为'person1'

点赞 回复

用户1472546078933 1月前

找不到才去上层找，person1就有name就无需去原型里找

点赞 回复

CatWatermelon Lv1 共产主义接班人 2年前

function createObj(o){
let obj = {}

259 37 收藏

return obj
}
请问原型式继承和上面这个差别在哪里

点赞 回复

提莫队长同志884... 前端开发工程师 @ 软通 2年前

😘

点赞 回复

哈娄 前端开发工程师 @ 天狮... 2年前

在组合式寄生继承的方式中，为什么还要创造一个F的匿名函数，直接Child.prototype=Parent.prototype不可以吗？

点赞 3

almostly 2年前

这样Child的实例的构造函数的constructor指向的是Parent，应该是Child,所以不推荐这样

点赞 回复

倔强的小码农 Lv1 2年前

当我们想给 Child 的prototype里面添加共享属性或者方法时，如果其 prototype 指向的是 Parent 的 prototype，那么在 Child 的 prototype 里添加的属性和方法也会反映在 Parent 的 prototype 里面，这明显是不合理的，这样做的后果是当我们只想使用 Parent 时，也能看见 Child 往里面扔的方法和属性。所以需要每个构造函数都需要持有自己专用的prototype对象

4 回复

259 37 收藏

独孤玉辉 web前端 @ 前端劝退师 2年前

感觉看得有点吃力 😳

点赞 回复

天罗 Lv1 2年前

借用构造函数有一个缺点，Parent 的原型不会继承。

点赞 回复

氢氟酸 2年前

运行原型式继承的例子时，一直报错Cannot read property 'push' of undefined，后来仔细一看才发现，属性是friends 不是firends，可能是楼主手快了 😂

点赞 回复

大静 web前端 2年前

借用构造函数 还有一个缺点

点赞 回复

尤小左 Lv2 低级前端开发工程师 @ ... 3年前

问下啊，你上面的例子，组合继承 和 寄生组合继承感觉没区别啊

点赞 1

259 37 收藏

银银 3年前

楼主好赞

点赞 回复

君君菇凉 Lv1 web前端工程师 3年前

注意：修改person1.name的值，person2.name的值并未发生改变，并不是因为person1和person2有独立的 name 值，而是因为person1.name = 'person1'，给person1添加了 name 值，并非修改了原型上的 name 值。
起初 没有懂这句话 ，后来把代码放到js 执行 ，懂了 ，所以还是不能太懒了 作者这一系列的看着好难 ，可能是自己的基础不够。 给作者一个大大的赞

点赞 1

静怡师太 前端开发工程师 4年前

楼主，辛苦了！

点赞 回复

KofW 一个前端小菜鸟 @ 还在... 4年前

不自主的看向评论区

1 1

武安君丿白起 4年前

相当厉害

点赞 回复

259 37 收藏

一念成魔，一念成佛，爱恨就在一瞬间

点赞 回复

APPStore 4年前

你一直在用表情敷衍我，伤透了我的心，我恨你

点赞 1

冴羽 Lv6 （作者） 公众号@yayujs @ 淘宝 4年前

吓得我键盘都掉了...∑(O_O；)

1 回复

查看全部 37 条回复

相关推荐

摸鱼的春哥 | 2月前 | 前端 · JavaScript

2022，前端的天🌧要怎么变?

6.8w 664 250

HighClassLickDog | 4月前 | 前端 · JavaScript

259 37 收藏

6.4w 459 171

Nordon | 5月前 | 前端 · JavaScript

JS 6 种继承方式及优缺点

551 13 4

sunshine小小倩 | 4年前 | JavaScript · 前端

this、apply、call、bind

12.2w 3147 250

前端胖头鱼 | 5月前 | JavaScript · Vue.js · 前端

就因为JSON.stringify，我的年终奖差点打水漂了

7.6w 1067 233

Sunshine_Lin | 7月前 | 前端 · JavaScript · Vue.js

15张图，20分钟吃透Diff算法核心原理，我说的！！！

5.7w 2232 246

echo_dc | 1年前 | JavaScript

javascript的6种继承方式

572 6 1

大帅老猿 | 9月前 | 前端 · JavaScript · HTML

5.9w 1420 240

浪里行舟 | 3年前 | JavaScript · 前端

九种跨域方式实现原理（完整版）

13.6w 2612 107

Sunshine_Lin | 5月前 | 前端 · JavaScript · ECMAScript 6

「万字总结」熬夜总结50个JS的高级知识点，全都会你就是神！！！

7.5w 2178 107

程序员依扬 | 2年前 | 面试 · 前端

【1 月最新】前端 100 问：能搞懂 80% 的请把简历给我

52.7w 9279 308

Karon_ | 3年前 | JavaScript

JS 总结之原型继承的几种方式

4106 67 7

老姚 | 2年前 | JavaScript · 前端

你未必知道的CSS知识点（第二季）

6.4w 2482 196

大帅老猿 | 10月前 | 前端 · JavaScript

259 37 收藏

10.8w 2907 576

前端阿飞 | 4月前 | 前端 · JavaScript

10个常见的前端手写功能，你全都会吗?

8.9w 2222 178

非优秀程序员 | 4月前 | 前端 · JavaScript

如何用 CSS 中写出超级美丽的阴影效果

31.2w 235 12

阿里南京技术专刊 | 3年前 | Angular.js · Git · JavaScript

优雅的提交你的 Git Commit Message

9.6w 1213 39

CUGGZ | 5月前 | 前端 · JavaScript · 程序员

33个非常实用的JavaScript一行代码，建议收藏!

7.3w 1730 59

杭州程序员张张 | 3年前 | Vue.js · JavaScript · 前端

Vue.js中 watch 的高级用法

10.2w 923 43

CRPER | 4年前 | Vue.js · JavaScript · 前端

259 37 收藏