尼克陈 Lv4
2021年03月01日 阅读 10191
已关注

面不面试的，你都得懂原型和原型链

> 新蜂商城开源仓库（内涵 Vue 2.x 和 Vue 3.x 的 H5 商城开源代码，带服务端 API 接口）：github.com/newbee-ltd

> Vue 3.x + Vant 3.x + Vue-Router 4.x 高仿微信记账本开源地址（带服务端 API 接口）：github.com/Nick930826/...

前言

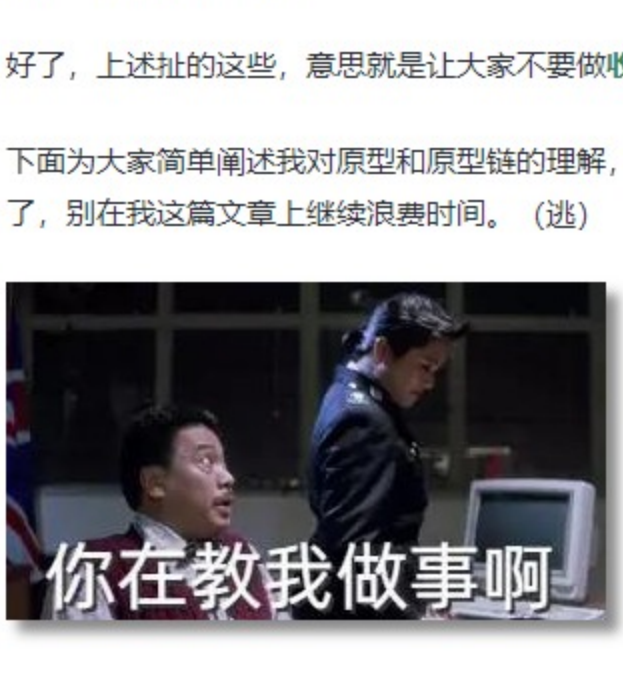
不要为了面试而去背题，匆匆忙忙的，不仅学不进去，背完了几天后马上会忘记。

艾宾浩斯遗忘曲线

你可能会说，“没办法，这不是为了能找份工作嘛！”。我想说的是，“那你没开始找工作的时候，咋不好好学习呢。”

好了，上述扯的这些，意思就是让大家不要做收藏家，不要把好文收藏了，就放在收藏夹里吃灰！

下面为大家简单阐述我对原型和原型链的理解，若是觉得有说的不对的地方，还望直接把页面关闭了，别在我这篇文章上继续浪费时间。（逃）



四个规则

我们先来了解下面引用类型的四个规则：

1、引用类型，都具有对象特性，即可自由扩展属性。

2、引用类型，都有一个隐式原型 `__proto__` 属性，属性值是一个普通的对象。

3、引用类型，隐式原型 `__proto__` 的属性值指向它的构造函数的显式原型 `prototype` 属性值。

4、当你试图得到一个对象的某个属性时，如果这个对象本身没有这个属性，那么它会去它的隐式原型 `__proto__`（也就是它的构造函数的显式原型 `prototype`）中寻找。

> 引用类型：Object、Array、Function、Date、RegExp。这里我姑且称 **proto** 为隐式原型，没有官方中文叫法，大家都瞎叫居多。

下面我们逐一验证上面几个规则，就会慢慢地理解原型和原型链。

规则一

引用类型，都具有对象特性，即可自由扩展属性：

```javascript
const obj = {}
const arr = []
const fn = function () {}

obj.a = 1
arr.a = 1
fn.a = 1

console.log(obj.a) // 1
console.log(arr.a) // 1
console.log(fn.a) // 1
```

> 这个规则应该比较好理解，Date 和 RegExp 也一样，就不赘述了。

规则二

引用类型，都有一个隐式原型 `__proto__` 属性，属性值是一个普通的对象：

```javascript
const obj = {};
const arr = [];
const fn = function() {}

console.log('obj.__proto__', obj.__proto__);
console.log('arr.__proto__', arr.__proto__);
console.log('fn.__proto__', fn.__proto__);
```

```
const obj = {};
const arr = [];
const fn = function() {}

console.log('obj.__proto__', obj.__proto__);
console.log('arr.__proto__', arr.__proto__);
console.log('fn.__proto__', fn.__proto__);
obj.__proto__ ▸{constructor: f, __defineGetter__: f, __defineSetter__: f, hasOwnProperty: f, __lookupGetter__: f, …}
arr.__proto__ ▸[constructor: f, concat: f, copyWithin: f, fill: f, find: f, …]
fn.__proto__ f () { [native code] }
```

规则三

引用类型，隐式原型 `__proto__` 的属性值指向它的构造函数的显式原型 `prototype` 属性值：

```javascript
const obj = {};
const arr = [];
const fn = function() {}

obj.__proto__ == Object.prototype // true
arr.__proto__ === Array.prototype // true
fn.__proto__ == Function.prototype // true
```

规则四

当你试图得到一个对象的某个属性时，如果这个对象本身没有这个属性，那么它会去它的隐式原型 `__proto__`（也就是它的构造函数的显式原型 `prototype`）中寻找：

```javascript
const obj = { a:1 }
obj.toString
// ƒ toString() { [native code] }
```

首先，`obj` 对象并没有 `toString` 属性，之所以能获取到 `toString` 属性，是遵循了第四条规则，从它的构造函数 `Object` 的 `prototype` 里去获取。

一个特例

我试图想推翻上面的规则，看下面这段代码：

```javascript
function Person(name) {
  this.name = name
  return this // 其实这行可以不写，默认返回 this 对象
}

var nick = new Person("nick")
nick.toString
// ƒ toString() { [native code] }
```

按理说，`nick` 是 `Person` 构造函数生成的实例，而 `Person` 的 `prototype` 并没有 `toString` 方法，那么为什么，`nick` 能获取到 `toString` 方法？

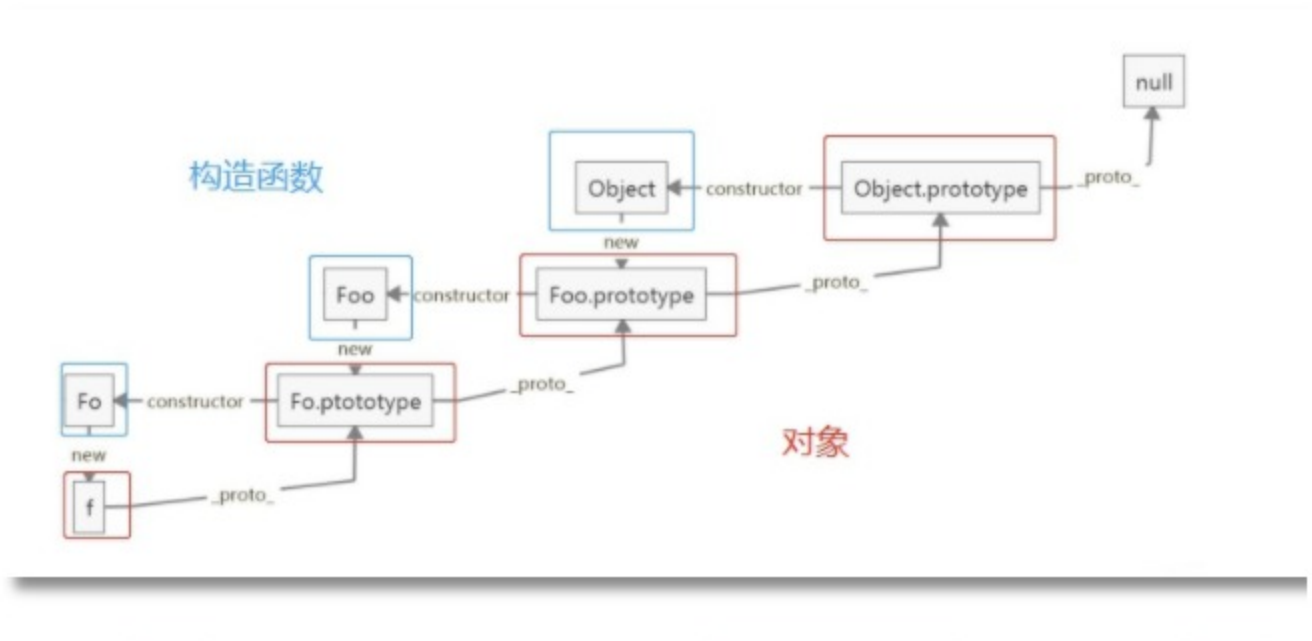
这里就引出 `原型链` 的概念了，`nick` 实例先从自身出发检讨自己，发现并没有 `toString` 方法。找不到，就往上走，找 `Person` 构造函数的 `prototype` 属性，还是没找到。构造函数的 `prototype` 也是一个对象嘛，那对象的构造函数是 `Object`，所以就找到了 `Object.prototype` 下的 `toString` 方法。

```
> function Person(name) {
    this.name = name
    return this // 其实这行可以不写，默认返回 this 对象
  }

  var nick = new Person("nick")

  console.log(Person.prototype)
  console.log(Person.prototype.__proto__ === Object.prototype)
  ▸{constructor: f}
  true
<· undefined
```

> 上述寻找的过程就形成了原型链的概念，我理解的原型链就是这样一个过程。也不知道哪个人说过一句，JavaScript 里万物皆对象。从上述情况来看，好像是这么个理。🤔

一张图片

用图片描述原型链：



最后一个 `null`，设计上是为了避免死循环而设置的，`Object.prototype` 的隐式原型指向 `null`。

一个方法

`instanceof` 运算符用于测试构造函数的 `prototype` 属性是否出现在对象原型链中的任何位置。`instanceof` 的简易手写版，如下所示：

```javascript
// 变量R的原型 存在于 变量L的原型链上
function instance_of (L, R) {
  // 验证如果为基本数据类型，就直接返回 false
  const baseType = ['string', 'number', 'boolean', 'undefined', 'symbol']
  if(baseType.includes(typeof(L))) { return false }

  let RP = R.prototype;  // 取 R 的显示原型
  L = L.__proto__; // 取 L 的隐式原型
  while (true) {
    if (L === null) { // 找到最顶层
      return false;
    }
    if (L === RP) { // 严格相等
      return true;
    }
    L = L.__proto__;  // 没找到继续向上一层原型链查找
  }
}
```

我们再来看下面这段代码：

```javascript
function Foo(name) {
  this.name = name;
}
var f = new Foo('nick')

f instanceof Foo // true
f instanceof Object // true
```

上述代码判断流程大致如下：

1、`f instanceof Foo`：`f` 的隐式原型 `__proto__` 和 `Foo.prototype`，是相等的，所以返回 `true`。

2、`f instanceof Object`：`f` 的隐式原型 `__proto__`，和 `Object.prototype` 不等，所以继续往上走。`f` 的隐式原型 `__proto__` 指向 `Foo.prototype`，所以继续用 `Foo.prototype.__proto__` 去对比 `Object.prototype`，这会儿就相等了，因为 `Foo.prototype` 就是一个普通的对象。

> 再一次验证万物皆对象。。。。

总结

通过四个特性、一个例子、一张图片、一个方法，大家应该对原型和原型链的关系有了大概的认知。我的认知就是，原型链就是一个过程，原型是原型链这个过程中的一个单位，贯穿整个原型链。就好像你要是看完了不点个赞，我可以顺着网线找到你。

文章分类 前端　文章标签 前端

尼克陈 Lv4 全栈开发 @ 掘金 God Father
发布了 17 篇专栏 · 获得点赞 4,327 · 获得阅读 104,188
已关注

输入评论...

用户8490546675641
nice哦
6天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
感谢评论！
6天前 回复

很帅很愁人 全村工程师 @ 下王庄村委员会
网线已拔
7天前 回复

用户1996224108937
努力逛掘金的第一天..
13天前 1 回复

永恒的爱
谢谢作者，对原型和原型链有新的了解
15天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
感谢评论
15天前 回复

CookiePool Lv2 FE @首席摸鱼官
这种文章看着很舒服，没废话，图文并茂，思路清晰！向楼主学习
15天前 2 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
感谢支持
15天前 回复

了不起的王小一
看起来__proto__和prototype的作用是一样哇的啊，干嘛搞两个呢
17天前 1 回复

白养
原型链好比继承+迭代，他可以让一个对象，继承多种多样的属性，方法。
而__proto__和prototype，就好比人的dna。__proto__指向父母，prototype代表父母自己。
两者的区别就在于此。
15天前 4 回复

摸鱼酱 Lv3 高级摸鱼工程师 @ 深圳
给我的第一个粉丝三连
17天前 回复

掘金酱 Lv4 ♥首席客服君 @ 掘金
Hi，掘金酱邀请你参加「刷题打卡」任务挑战，完成 10 篇题解文章就可以赢得大奖，掘友们冲啊！🎁 sourl.co
18天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
别闹了，这都三月了，我还没收到一月的作者榜奖励。
18天前 回复

宇哥来学前端 前端工程师
通俗易懂，没有多余废话 看完两个字：舒服
19天前 1 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
看完你的评论，两个字：干脆
19天前 回复

Zeong 摸鱼工程师切图仔 @ 摸鱼大队...
图文并茂，讲解清晰！点个赞
19天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
感谢支持
19天前 回复

maxlxq 小前端开发 @ HZFE
点赞
20天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
😘 点赞请动动你的发财手 [图片]
20天前 回复

夕阳染红颜 入门打工人
```
function Person(name) {
 this.name = name
 return this // 其实这行可以不写，默认返回 this 对象
}
```
这里不是默认返回undefined吗?
20天前 回复

_newbie 前端
看你这是函数调用，还是实例化
20天前 3 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
1、return 的是五种简单数据类型：String，Number，Boolean，Null，Undefined。
这种情况下，忽视 return 值，依然返回 this 对象。
2、return 的是 Object。
这种情况下，不再返回 this 对象，而是返回 return 语句的返回值。
综上所述，默认不写 return 的情况下，构造函数实例化，就是返回 this 上下文。
20天前 3 回复

づLiao﹤ 前端开发工程师
var nick = new Person("nick") 关键在这里。
19天前 1 回复

我不是菜鸟 切图仔
距离忘记还剩10秒钟，请多做笔记
20天前 回复

Round22503 搬砖
原本我是知道的可过了一段时间我又忘了 😂
20天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
这不又记起来了嘛
20天前 回复

gqz Lv2 前端菜鸡
网线已拔
20天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
你确实是个人才
20天前 回复

徐四 Lv2 要来一杯嘛？！我超甜~！耶咻...
果然还得是我友哥，简单明了就阐述了一座大山 ⛰️😝
21天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
来啦，老四。
21天前 回复

盖聂真的喜欢端木蓉
懂了，收藏了
21天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
嗯嗯，感谢支持！
21天前 回复

tianxingyu web前端
真是棒啊 我的尼克陈 😘
21天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
嗯嗯，感谢支持～～
21天前 回复

shahow
针不戳
21天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
感谢支持～
21天前 回复

前端五连鞭 前端开发
你在教我做事
21天前 回复

尼克陈 Lv4 (作者) 全栈开发 @ 掘金 God Fa...
感谢支持
21天前 回复

关于作者
尼克陈 Lv4
全栈开发 @ 掘金 G...
获得点赞 4,327
文章被阅读 104,188

目录