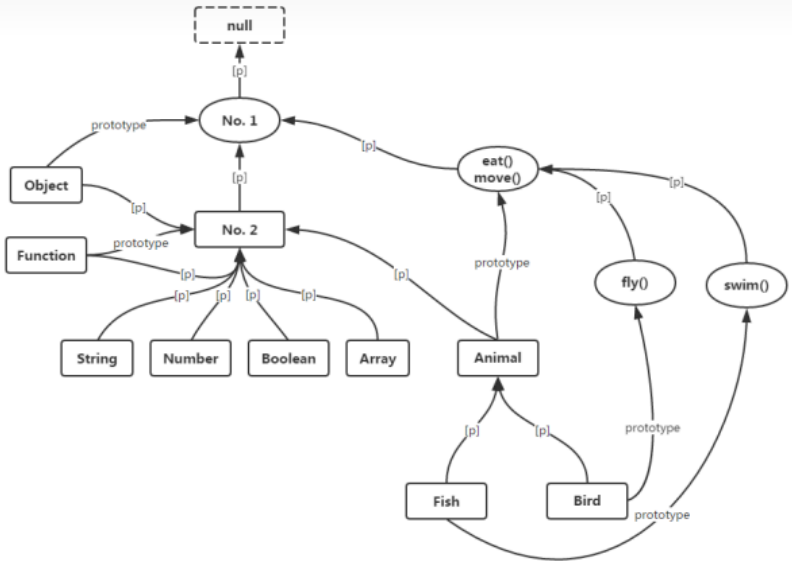
知乎 首发于 写代码的苏打饼



JavaScript 世界万物诞生记

manxisuo
物理专业程序员，五仁月饼、哆啦A梦、高木同学爱好者。
关注他

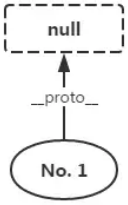
615 人赞同了该文章

一. 无中生有

起初，什么都没有。

造物主说：没有东西本身也是一种东西啊，于是就有了null：

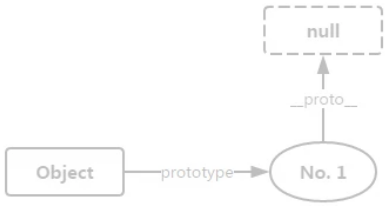
现在我们要造点儿东西出来。但是没有原料怎么办？
有一个声音说：不是有null嘛？
另一个声音说：可是null代表无啊。
造物主说：那就无中生有吧！
于是：



JavaScript中的1号对象产生了，不妨把它叫做No. 1。
这个No. 1对象可不得了，它是真正的万物始祖，它拥有的性质，是所有的对象都有的__proto__是什么呢？是

知乎 首发于 写代码的苏打饼

既然已经有了一个对象，剩下就好办了，因为一生二，二生三，三生万物嘛。
不过造物主很懒，他不想一个一个地亲手制造对象。于是他做了一台能够制造对象的机器：



他给这台机器起了一个名字：**Object。**
这台机器并不能凭空造出对象，它需要一个模板对象，按照这个模板对象来制造对象。很自然的，它把目前仅有的No. 1对象作为模板。图中的prototype就代表机器的**模板对象。**

机器如何启动呢？通过**new**命令。你对着机器喊一声：“new！”，对象就造出来了。

机器的产生，实现了对象的批量化自动化生产，解放了造物主的双手。于是造物主忙别的去了。
如果机器只是按照模板的样子，机械地复制出一模一样的对象，那就太笨了。
人类的后代在继承了父辈的性状的基础上，可以产生父辈没有的性状。同样地，机器在制造对象时，除了继承模板对象的属性外，还可以添加新的属性。这使得JavaScript世界越来越多样化。

比如说，有一天Object机器制造一个对象，它有一个特殊的属性，叫做flag，属性值是10。用图形表示是这样的：

写成代码就是：

```
var obj = new Object({ flag: 10 });
```

轰轰烈烈的造物运动开始了……

三. 更多制造对象的机器

一天天过去了，造物主来视察工作。看到Object制造出了好多好多对象，他非常高兴。
同时他还发现：根据“物以类聚”的原则，这些对象可以分成很多**类。**聪明的造物主想，我何不多造几台机器，让每一台机器专门负责制造某一类对象呢？于是，他动手造出了几台机器并给它们起了名字。它们分别是：

String：用来制造表示一段文本的对象。
Number：用来制造表示一个数字的对象。
Boolean：用来制造表示是与非的对象。
Array：用来制造有序队列对象。
Date：用来制造表示一个日期的对象。
Error：用来制造表示一个错误的对象。
……

多台机器齐开动，各司

于是，造物主基于No. 1对象，造出了一个No. 2对象，用它来表示所有机器的共同特征。换句话说，把它作为所有机器的原型对象。

（注：__proto__写起来太麻烦了，后面我们用[p]来代替）

当然了，和Object一样，这些机器也需要各自有一个模板对象，也就是它们的prototype属性指向的那个对象。显然它们的模板对象应该是继承自No. 1对象的，即

这张图显示了JavaScript世界中那些最基本的机器本身的原型链，以及它们的模板对象的原型链。不过看起来太复杂了，所以后面我们就不再把它们完整地画出来了。

四. 制造机器的机器

造物主高兴地想：这下可好了，我造出了Object机器，实现了对象制造的自动化。然后又造出了String、Number等机器，实现了特定类别的对象制造的自动化。但是，为啥总感觉似乎还缺点什么呢？

很快，万能的造物主就把它造了出来，并把它命名为**Function**。有了Function机器后，就可以实现自动化地制造机器了。

让我们来观察一下Function：
首先，Function是一台机器，所以它的原型对象也是No. 2对象。
其次，Function又是一台制造机器的机器，所以它的模板对象也是No. 2对象。
所以我们得到了Function的一个非常特别的性质：

```
Function.__proto__ === Function.prototype
```

哇，太奇妙了！

不要奇怪，这个性质不过是”Function是一台制造机器的机器“这个事实的必然结果。

于是JavaScript的世界的变成了下面的样子：

从这张图中，我们发现：所有的函数(包括Function)的原型都是No. 2对象，而同时Function.prototype也是No. 2对象。这说明了：

> 从逻辑上，我们可以认为所有机器(包括Function自己)都是由Function制造出来的。

同时，如果再仔细瞧瞧，你会发现：

> Object作为一个机器可以看做是有由Function制造出来的，而Function作为一个对象可以看做是由Object制造出来的。

这就是JavaScript世界的“鸡生蛋，蛋生鸡”问题。那么到底是谁生了谁呢？Whatever！

五. 让世界动起来

就像前面所说，机器用来制造某一类对象。正因如此，机器可以作为这类对象的标志，即面向对象语言中类(class)的概念。所以机器又被称为**构造函数**。在ES6引入class关键字之前，我们常常把构造函数叫做类。

然而，除了作为构造函数来制造对象外，函数通常还有另一个功能：**做一件事情**。正是有了这个功能，JavaScript的世界:

```
function Bird(color) {
    this.color = color;
}
```

然后，对着造鸟机说：“new！”，于是造鸟机发动起来，制造一个红色的鸟：

```
var redBird = new Bird('#FF0000');
```

如果现在我们想让鸟飞起来，该怎么办呢？我们需要再次用Function制造出一台机器，不过这台机器不是用来制造对象的，而是用来做事儿的，即“让鸟飞起来”这件事情：

```
// 这是一台通过晃动鸟的翅膀，让鸟飞起来的简陋的机器。
function makeBirdFly(bird) {
    shakeBirdWing(bird);
}
```

我们知道，让一台制造对象的机器发动，只需要对它喊“new”即可；那么怎样让一台做事情的机器发动呢？更简单，对它咳嗽一声就行了。咳咳咳，

```
makeBirdFly(redBird);
```

于是红鸟飞了起来，世界充满了生机。

从上面的Bird和makeBirdFly的定义可以看出：实际上，制造对象的机器和做事情的机器没什么明显区别，不同的只是它们的使用方式。在两种情况下，它们分别被叫做**构造函数**和**普通函数**。

说明1：function xxx语法可以看成new Function的等价形式。
说明2：用户自定义的函数通常既可以作为普通函数使用，又可以作为构造函数来制造对象。ES6新增的class语法定义的函数只能作为构造函数，ES6新增的=>语法定义的箭头函数只能作为普通函数。

六. 让世界立体起来

造物主对目前的世界还是不太满意，因为几乎所有的机器的模板对象都是No. 2，这使得JavaScript世界看起来有点扁。

于是造物主再次研究世界万物的分类问题。他发现有些对象会动、还会吃东西，于是把它们叫做动物，然后造了一台Animal机器来制造它们。他进一步发现，即使都是动物，也还是可以进一步分类，比如有些会飞、有些会游，他分别把它们叫做鸟类、鱼类。于是他想，我何不单独造几台机器，专门用来制造某一类动物呢。于是它造出了Bird、Fish等机器。

接下来，在选择这些机器的模板对象时碰到一个问题：如果还像之前那样直接复制一个No. 1对象作为Bird、Fish的模板，那么结果就是这样的：

这样可不好。首先没体现出鸟类、鱼类跟动物的关系，其次它们的模板对象存了重复的东西，这可是一种浪费啊。怎么办呢？简单，让Bird和Fish的模板对象继承自Animal的模板对象就好了，就是说

于是：

用同样的方法，造物主造出了一个立体得多的JavaScript世界。

然而这样还不够。虽然那些纯对象现在充满了层次感，但是那些机器对象之间的关系还是扁平的：

那又该怎么办呢？其实用类似的办法就行了：

为了更方便地做到这一点，造物主发明了class关键字。

七. 世界最终的样子

经过一番折腾，JavaScript世界发生了大变化。变得丰富多彩，同时变得很复杂。用一张图再也没法画出它的全貌，只能画出冰山一角：

JavaScript的世界还在不断进化中……

编辑于 2023-03-09 09:30 · IP 属地未知

「真诚赞赏，手留余香」

知乎 首发于 写代码的苏打饼

JavaScript

欢迎参与讨论

70 条评论 默认 最新

刘速
“Object作为一个机器可以看做是有由Function制造出来的，而Function作为一个对象可以看做是由Object制造出来的。”我质疑这句话。

Function对象是由Object制造出来的，那么就满足Function.__proto_===Object.prototype。但是Function.__proto===Function.prototype。说明Function是由它“自交”出来的。所以我觉得这句话应该改成：
“Object作为一个机器可以看做是有由Function制造出来的，而Function作为一个对象也是由Function机器制造出来的。然而Function的模板的模板是Object的模板。”
Function是机器(构造函数)的鼻祖，Object.prototype是模板(原型)的鼻祖。
2019-09-25 回复 14

易之
赞
2020-11-30 回复 1

杵臼之交
仁兄高见了啦！
2022-08-19 回复 喜欢

飞行家
已打赏，写的确实不错。不过觉得这文章我点受众不清晰。对于程序猿来说过于浅显，毕竟程序员更关注具体的问题比如某个架构原理或者某个库怎么用。可如果受众是普通吃瓜群众，那科普的深度又有点深了。
2016-10-23 回复 8

yicccc
受众是面试者
2018-09-11 回复 2

颜海镜 ‣ **yicccc**
更适合面试者的版本：颜海镜：如何回答面试中的JavaScript原型链问题
2021-03-24 回复 1

展开其他 3 条回复 》

小昭
很棒的文章！！！
2016-10-18 回复 4

想得美
对我这种菜鸟受益很大，感觉受众就是我这种的。
2016-10-24 回复 3

知乎用户
一切的开始，不应该是 undefined 吗?
2016-10-22 回复 2

jediego
undefined值是派生自null值的（高程3第3章-p24）
2016-10-25 回复 4

万物皆数 ‣ **jediego**
不对，是第26页
2018-04-04 回复 喜欢

展开其他 1 条[illegible]

知乎 首发于 写代码的苏打饼

莉莉丝造出生命，人类就是函数原型，函数制造各类函数，人类繁衍，亚当也是使徒，莉莉丝也是使徒，看，多么类似!

2019-09-17 回复 1

Sherrychen
对于英语专业的我来说，感觉你在讲人类简史和音标音素的产生。。。
2019-04-17 回复 1

邢sir
ES5，所谓的机器对象之间好象不能继承吧?
2016-11-14 回复 1

我在厕所看吃播
prototype 和 _prop_ 啥区别
2016-10-25 回复 1

manxisuo 作者 ‣ **三土**
不是一回事。
__proto__是所有对象（包括函数对象）都有的一个属性（当然只是逻辑上有这么个概念），当我们说“原型链”的时候，就是指对象通过这个属性互相连接而形成的链状结构。原型链也就是继承链。
prototype是只有函数（准确地说是构造函数）才有的一个属性，例如对于对于某个函数Fun。它的意义在于，当你用var obj = new Fun() 得到一个对象obj时，这个obj的原型就是F.prototype。即(new Fun()).__proto__ === Fun.prototype，见文中第4个图。

No. 1其实就是Object.prototype，No. 2其实就是Function.prototype。我只是为了强调这两个对象的重要性，故意这样说的。不太严格地说，前者就是一个空对象类似：{}，后者就是一个空函数，类似：fubction() {} 。
2016-10-28 回复 8

manxisuo 作者
前者是函数才有的属性，准确的说是构造函数才有的属性。后者是所有JavaScript对象（包括函数）都有的属性。前者是规范中定义的属性，JS运行时环境必须实现。后者不是一个标准属性，只有chrome的v8等部分引擎才支持。前者表示，在用某个构造函数实例化一个对象时，这个被实例化出来的对象的原型是谁。后者表示某个对象的原型。
2016-10-25 回复 2

展开其他 3 条回复 》

林志鹏
String Number Boolean Array这些对象不能继承，不满意
2016-10-21 回复 1

twilightwalker
可以继承
2016-11-01 回复 喜欢

点击查看全部评论 》

欢迎参与讨论

文章被以下专栏收录

写代码的苏打饼
没事写代码玩儿...

前端
记录前端开发的一切干货

推荐阅读

知乎 首发于 **写代码的苏打饼**

一张思维导图搞定 最近看了丰子恺先生的《美的情绪：西洋画派十二讲》，感觉自己以前杂七杂八了解的艺术画派都理清了，豆瓣评分9.1，即使是普通读者，也值得一读，而且通俗易懂。 多图预警！...

Mia chen

舒德干：进化论的进化简史 | 新译《物种起源》发布

知识分子 发表于知识分子

万物有灵且美

鸟川芥 发表于鸟川芥

高级动物，果然高级——对高级动物的“退乎之作”

自由与正义