[](http://blog.jobbole.com/)

[](http://blog.jobbole.com/99322/?utm_source=blog.jobbole.com&utm_medium=topBanner&utm_content=2016.4.5)

* [**首页**](http://blog.jobbole.com/)
* [**最新文章**](http://blog.jobbole.com/all-posts/)
* [**经典回顾**](http://blog.jobbole.com/tag/featuredpost/)
* [**开发**](http://blog.jobbole.com/category/programmer/)
* [**设计**](http://design.jobbole.com/)
* [**极客**](http://blog.jobbole.com/category/geeks/)
* [**IT技术**](http://blog.jobbole.com/category/it-tech/)
* [**业界**](http://top.jobbole.com/category/news/)
* [**职场**](http://blog.jobbole.com/category/career/)
* [**创业**](http://blog.jobbole.com/category/startup/)
* [**访谈**](http://blog.jobbole.com/category/interview/)
* [**在国外**](http://blog.jobbole.com/category/overseas/)



[伯乐在线](http://www.jobbole.com/) > [首页](http://blog.jobbole.com/) > [所有文章](http://blog.jobbole.com/all-posts/) > [IT技术](http://blog.jobbole.com/category/it-tech/) > JavaScript原型和继承

JavaScript原型和继承

2013/04/16 · [IT技术](http://blog.jobbole.com/category/it-tech/), [Web前端](http://blog.jobbole.com/category/webfront/), [开发](http://blog.jobbole.com/category/programmer/) · [5 评论](http://blog.jobbole.com/19795/#article-comment)· [Javascript](http://blog.jobbole.com/tag/javascript/)

分享到：**41**

来源：[网易邮箱](http://ntesmailfetc.blog.163.com/blog/static/2062870612012418102658176/)

前几天看了《[再谈js面向对象编程](http://blog.jobbole.com/13881/)》，当时就请教哈大神，发现文章有的地方可能会造成误导（或者说和ECMA有出入），后来自己翻一翻ECMA，总算找到“标准”的理解……

本文适合初学者，特别是对构造函数、原型和原型链概念比较模糊的，大牛请路过，好了，让我们一步步来看看 Javascript原型（链）到底有多神秘……

**一、函数创建过程**

在了解原型链之前我们先来看看一个函数在创建过程中做了哪些事情，举一个空函数的例子：

JavaScript



|  |  |
| --- | --- |
| 1 | function A() {}; |

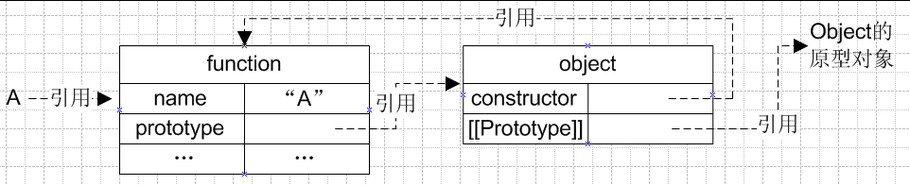
当我们在代码里面声明这么一个空函数，js解析的本质是（肤浅理解有待深入）：

1、创建一个对象（有constructor属性及[[Prototype]]属性），根据ECMA，其中[[Prototype]]属性不可见、不可枚举

2、创建一个函数（有name、prototype属性），再通过prototype属性 *引用* 刚才创建的对象

3、创建变量A，同时把函数的 *引用* 赋值给变量A

如下图所示：

[](http://jbcdn2.b0.upaiyun.com/2012/05/JavaScript-prototypes-and-inheritance.jpg)

（注意图中都是“ *引用* ”类型）

**每个函数的创建都经历上述过程。**

**二、构造函数**

那么什么是构造函数呢？

按照ECMA的定义

*Constructor* is a function that *creates* and *initializes* the newly created object.

构造函数是用来新建同时初始化一个新对象的函数。

什么样的函数可以用来创建同时初始化新对象呢？答案是：任何一个函数，包括空函数。

所以，结论是：**任何一个函数都可以是构造函数。**

**三、原型**

根据前面空函数的创建图示，我们知道每个函数在创建的时候都自动添加了prototype属性，这就是函数的**原型**，从图中可知其实质就是对一个对象的引用（这个对象暂且取名原型对象）。

我们可以对函数的原型对象进行操作，和普通的对象无异！一起来证实一下。

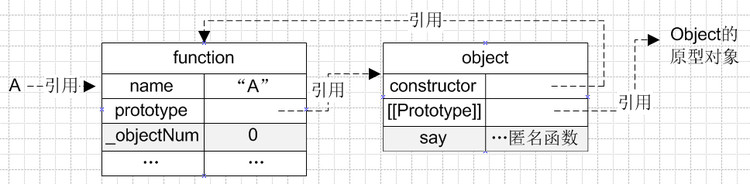
围绕刚才创建的空函数，这次给空函数增加一些代码：

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | function A() {  this.width = 10;  this.data = [1,2,3];  this.key = "this is A";  }  A.\_objectNum = 0;//定义A的属性  A.prototype.say = function(){//给A的原型对象添加属性  alert("hello world")  } |

第7~9行代码就是给函数的原型对象增加一个say属性并引用一个匿名函数，根据“函数创建”过程，图解如下：

[](http://jbcdn2.b0.upaiyun.com/2012/05/JavaScript-prototypes-and-inheritance2.jpg)

（灰色背景就是在空函数基础上增加的属性）

简单说**原型就是函数的一个属性，在函数的创建过程中由js编译器自动添加**。

**那么原型有什么用呢？**

先了解下new运算符，如下：

JavaScript



|  |  |
| --- | --- |
| 1  2 | var a1 = new A;  var a2 = new A; |

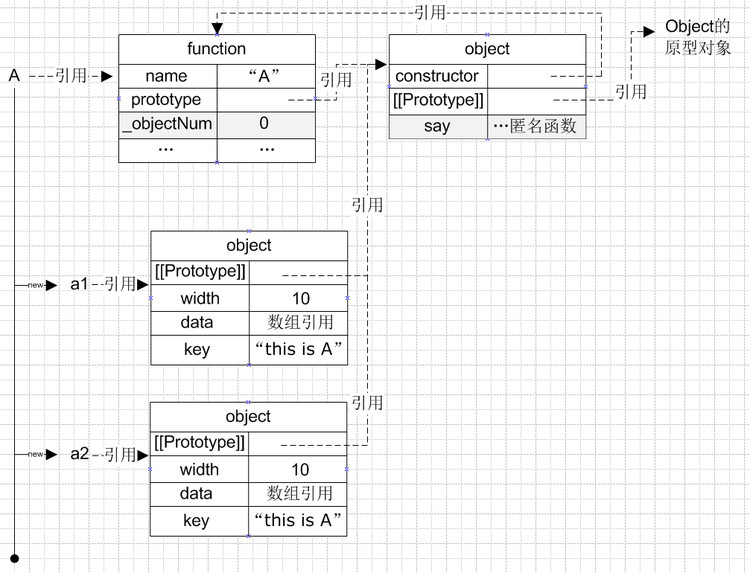
这是通过构造函数来创建对象的方式，那么创建对象为什么要这样创建而不是直接var a1 = {};呢？这就涉及new的具体步骤了，这里的new操作可以分成三步(以a1的创建为例)：

1、新建一个对象并赋值给变量a1：var a1 = {};

2、把这个对象的[[Prototype]]属性指向函数A的原型对象：a1.[[Prototype]] = A.prototype

3、调用函数A，同时把this指向1中创建的对象a1，对对象进行初始化：A.apply(a1,arguments)

其结构图示如下：

[](http://jbcdn2.b0.upaiyun.com/2012/05/JavaScript-prototypes-and-inheritance3.jpg)

从图中看到，无论是对象a1还是a2，都有一个属性保存了对函数A的原型对象的引用，对于这些对象来说，一些公用的方法可以在函数的原型中找到，节省了内存空间。

**四、原型链**

了解了new运算符以及原型的作用之后，一起来看看什么是[[Prototype]]？以及对象如何沿着这个引用来进行属性的查找？

在js的世界里，每个对象默认都有一个[[Prototype]]属性，其保存着的地址就构成了对象的原型链，它是由js编译器在对象 *被创建* 的时候自动添加的，其取值由new运算符的右侧参数决定：当我们var object1 = {};的时候，object1的[[Prototype]]就指向Object构造函数的原型对象，因为var object1 = {};实质上等于var object = new Object();（原因可参照上述对new A的分析过程）。

对象在查找某个属性的时候，会首先遍历自身的属性，如果没有则会继续查找[[Prototype]]引用的对象，如果再没有则继续查找[[Prototype]].[[Prototype]]引用的对象，依次类推，直到[[Prototype]].….[[Prototype]]为undefined（Object的[[Prototype]]就是undefined）

如上图所示：

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //我们想要获取a1.fGetName    alert(a1.fGetName);//输出undefined    //1、遍历a1对象本身    //结果a1对象本身没有fGetName属性    //2、找到a1的[[Prototype]]，也就是其对应的对象A.prototype，同时进行遍历    //结果A.prototype也没有这个属性    //3、找到A.prototype对象的[[Prototype]]，指向其对应的对象Object.prototype    //结果Object.prototype也没有fGetName    //4、试图寻找Object.prototype的[[Prototype]]属性，结果返回undefined，这就是a1.fGetName的值 |

简单说就是**通过对象的[[Prototype]]保存对另一个对象的引用，通过这个引用往上进行属性的查找，这就是原型链。**

**五、继承**

有了原型链的概念，就可以进行继承。

JavaScript



|  |  |
| --- | --- |
| 1 | function B() {}; |

这个时候产生了B的原型B.prototype

原型本身就是一个Object对象，我们可以看看里面放着哪些数据

B.prototype 实际上就是 {constructor : B , [[Prototype]] : Object.prototype}

因为prototype本身是一个Object对象的实例，所以其原型链指向的是Object的原型

JavaScript



|  |  |
| --- | --- |
| 1  2 | B.prototype = A.prototype;//相当于把B的prototype指向了A的prototype；这样只是继承了A的prototype方法，A中的自定义方法则不继承    B.prototype.thisisb = "this is constructor B";//这样也会改变a的prototype |

但是我们只想把B的原型链指向A，如何实现？

第一种是通过改变原型链引用地址

JavaScript



|  |  |
| --- | --- |
| 1 | B.prototype.\_\_proto\_\_ = A.prototype; |

ECMA中并没有\_\_proto\_\_这个方法，这个是ff、chrome等js解释器添加的，等同于EMCA的[[Prototype]]，这不是标准方法，那么如何运用标准方法呢？

我们知道new操作的时候，实际上只是把实例对象的原型链指向了构造函数的prototype地址块，那么我们可以这样操作

JavaScript



|  |  |
| --- | --- |
| 1 | B.prototype = new A(); |

这样产生的结果是：

产生一个A的实例，同时赋值给B的原型，也即B.prototype 相当于对象 {width :10 , data : [1,2,3] , key : "this is A" , [[Prototype]] : A.prototype}

这样就把A的原型通过B.prototype.[[Prototype]]这个对象属性保存起来，构成了原型的链接

但是注意，这样B产生的对象的构造函数发生了改变，因为在B中没有constructor属性，只能从原型链找到A.prototype，读出constructor:A

JavaScript



|  |  |
| --- | --- |
| 1  2 | var b = new B;    console.log(b.constructor);//output A |

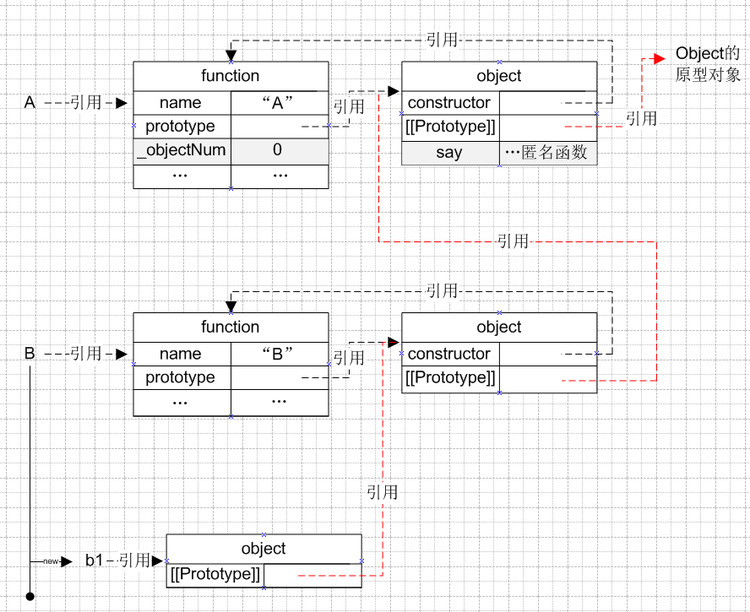
所以我们还要人为设回B本身

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | B.prototype.constructor = B;    //现在B的原型就变成了{width :10 , data : [1,2,3] , key : "this is A" , [[Prototype]] : A.prototype , constructor : B}    console.log(b.constructor);//output B    //同时B直接通过原型继承了A的自定义属性width和name    console.log(b.data);//output [1,2,3]    //这样的坏处就是    b.data.push(4);//直接改变了prototype的data数组（引用）    var c = new B;    alert(c.data);//output [1,2,3,4]    //其实我们想要的只是原型链，A的自定义属性我们想在B中进行定义（而不是在prototype）    //该如何进行继承？    //既然我们不想要A中自定义的属性，那么可以想办法把其过滤掉    //可以新建一个空函数    function F(){}    //把空函数的原型指向构造函数A的原型    F.prototype = A.prototype;    //这个时候再通过new操作把B.prototype的原型链指向F的原型    B.prototype = new F;    //这个时候B的原型变成了{[[Prototype]] : F.prototype}    //这里F.prototype其实只是一个地址的引用    //但是由B创建的实例其constructor指向了A，所以这里要显示设置一下B.prototype的constructor属性    B.prototype.constructor = B;    //这个时候B的原型变成了{constructor : B , [[Prototype]] : F.prototype}    //这样就实现了B对A的原型继承 |

图示如下，其中红色部分代表原型链：

[](http://jbcdn2.b0.upaiyun.com/2012/05/JavaScript-prototypes-and-inheritance4.jpg)

作为初学者浅陋的理解，本文目的在于更具象地去理解js的面向对象，疏漏之处请指正。