## 原型链\_\_proto\_\_、prototype属性和constructor属性

1、那么\_\_proto\_\_是什么？

每个对象都会在其内部初始化一个属性，就是\_\_proto\_\_，当我们访问一个对象的属性 时，如果这个对象内部不存在这个属性，那么他就会去\_\_proto\_\_里找这个属性，这个\_\_proto\_\_又会有自己的\_\_proto\_\_，于是就这样 一直找下去，也就是我们平时所说的原型链的概念。

1. prototype

我们创建的每个函数都有一个 prototype (原型)属性,这个属性是一个指针,指向一个对象,而这个对象的用途是包含可以由特定类型的所有实例共享的属性和方法。如果按照字面意思来理解,那么 prototype 就是通过调用构造函数而创建的那个对象实例的原型对象。使用原型对象的好处是可以让所有对象实例共享它所包含的属性和方法。换句话说,不必在构造函数中定义对象实例的信息,而是

可以将这些信息直接添加到原型对象中。

3、constructor

无论什么时候,只要创建了一个新函数,就会根据一组特定的规则为该函数创建一个 prototype属性,这个属性指向函数的原型对象。在默认情况下,所有原型对象都会自动获得一个 constructor(构造函数)属性,这个属性包含一个指向 prototype 属性所在函数的指针。就拿前面的例子来说,Person.prototype. constructor 指向 Person 。而通过这个构造函数,我们还可继续为原型对象添加其他属性和方法。创建了自定义的构造函数之后,其原型对象默认只会取得 constructor 属性;至于其他方法,则都是从 Object 继承而来的。当调用构造函数创建一个新实例后,该实例的内部将包含一个指针(内部属性),指向构造函数的原型对象。ECMA-262 第 5 版中管这个指针叫 [[Prototype]] 。虽然在脚本中没有标准的方式访问 [[Prototype]] ,但 Firefox、Safari 和 Chrome 在每个对象上都支持一个属性\_\_proto\_\_ ;而在其他实现中,这个属性对脚本则是完全不可见的。不过,要明确的真正重要的一点就是,这个连接存在于实例与构造函数的原型对象之间,而不是存在于实例与构造函数之间。

对象通过constructor属性可以，查看到该对象的构造函数；

1. 读取属性

每当代码读取某个对象的某个属性时,都会执行一次搜索,目标是具有给定名字的属性。搜索首先从对象实例本身开始。如果在实例中找到了具有给定名字的属性,则返回该属性的值;如果没有找到,

则继续搜索指针指向的原型对象,在原型对象中查找具有给定名字的属性。如果在原型对象中找到了这个属性,则返回该属性的值。也就是说,在我们调用 person1.sayName() 的时候,会先后执行两次搜

索。首先,解析器会问:“实例 person1 有 sayName 属性吗?”答:

“没有。”然后,它继续搜索,再问:“ person1 的原型有 sayName 属性吗?”答:“有。“于是,它就读取那个保存在原型对象中的函数。当我们调用 person2.sayName() 时,将会重现相同的搜索过程,得到相同的结果。而这正是多个对象实例共享原型所保存的属性和方法的基本原理。

虽然可以通过对象实例访问保存在原型中的值,但却不能通过对象实例重写原型中的值。如果我们在实例中添加了一个属性,而该属性与实例原型中的一个属性同名,那我们就在实例中创建该属性,该

属性将会屏蔽原型中的那个属性。