## Javascript面向对象编程设计

ECMA-262把对象定义为：“无需属性的集合，其属性可以包含基本的值、对象或者函数。”严格来说，这就相当于说对象是一组没有特定顺序的值。对象的每个属性或者方法都有一个名字，而每个名字都映射到一个值。正因为这样（以及将要讨论的原因），我们可以把ECMAScript的对象想象成散列表：无非就是一组名值对，其中值可以是数据或者函数。

一、构造函数

每个对象都是基于一个引用类型创建的，这个引用类型可以是一个原生的类型，也可以是开发人员定义的类型。

function Person(name,age,job){

this.name= name;

this.age=age;

this.job=job;

this.sayName = function (){

Alert(this.name);

}

}

Var person1 = new Person('Nicholas',29,'Software Engineer');

Var person2 = new Person('Greg',27,'Doctor');

要创建Person的新实例，必须使用new操作符。以这种方法调用构造函数实际上会经历四个步骤：

1. 创建一个新的对象；
2. 将构造函数的作用域赋给新对象（因此this就指向了这个新的对象）；
3. 执行构造函数中的代码（为这个新的对象添加属性）；
4. 返回新的对象。

在前面例子的最后, person1 和 person2 分别保存着 Person 的一个不同的实例。这两个对象都

有一个 constructor (构造函数)属性,该属性指向 Person ,如下所示。

alert(person1.constructor == Person);//true

alert(person2.constructor == Person);//true

对象的 constructor 属性最初是用来标识对象类型的。但是,提到检测对象类型,还是 instanceof 操作符要更可靠一些。我们在这个例子中创建的所有对象既是 Object 的实例,同时也是 Person的实例,这一点通过 instanceof 操作符可以得到验证。

alert(person1 instanceof Object);//true

alert(person1 instanceof Person); //true

alert(person2 instanceof Object);//true

alert(person2 instanceof Person);//true

创建自定义的构造函数意味着将来可以将它的实例标识为一种特定的类型;而这正是构造函数模式胜过工厂模式的地方。在这个例子中, person1 和 person2 之所以同时是 Object 的实例,是因为所。有对象均继承自 Object ；

1. 原型链

javaScript中的每个函数中都有一个指向某一对象的prototype属性。该函数被new操作符调用时会创建并返回一个对象，并且该对象中会有一个指向其原型对象的秘密链接。通过该秘密链接（在某些环境中，该链接名为\_proto\_)，就可以在新建的对象中调用相关原型对象的方法和属性。

1. 理解原型对象

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。

以前面使用 Person 构造函数和 Person.prototype 创建实例的代码为例,图 6-1 展示了各个对

象之间的关系。

展示了 Person 构造函数、 Person 的原型属性以及 Person 现有的两个实例之间的关系。

Person.prototype 指向了原型对象,

而 Person.prototype.constructor 又指回了 Person 。

在此,

原型对象中除了包含 constructor 属性之外,还包括后来添加的其他属性。 Person 的每个实例——

person1 和 person2 都包含一个内部属性,该属性仅仅指向了 Person.prototype ;换句话说,它们

与构造函数没有直接的关系。此外,要格外注意的是,虽然这两个实例都不包含属性和方法,但我们却

图灵社区会员 StinkBC(StinkBC@gmail.com) 专享 尊重版权6.2 创建对象

149

可以调用 person1.sayName() 。这是通过查找对象属性的过程来实现的。

虽然在所有实现中都无法访问到 [[Prototype]] ,但可以通过 isPrototypeOf() 方法来确定对象之

间是否存在这种关系。从本质上讲,如果 [[Prototype]] 指向调用 isPrototypeOf() 方法的对象

,那么这个方法就返回 true ,如下所示:

( Person.prototype )

alert(Person.prototype.isPrototypeOf(person1));

alert(Person.prototype.isPrototypeOf(person2));

1

2

//true

//true

这里,我们用原型对象的 isPrototypeOf() 方法测试了 person1 和 person2 。因为它们内部都

有一个指向 Person.prototype 的指针,因此都返回了 true 。