

此处省略了三行文字。

### 6.1.3 原型模式

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，[它而这个对象的用途](#)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，如下面的例子所示：

```
function Person(){
}

Person.prototype.name = "Nicholas";
Person.prototype.age = 29;
Person.prototype.job = "Software Engineer";
Person.prototype.sayName = function(){
    alert(this.name);
};

var person1 = new Person();
person1.sayName(); // "Nicholas"

var person2 = new Person();
person2.sayName(); // "Nicholas"

alert(person1.sayName === person2.sayName); //true
```

在此，我们将 `sayName()`方法和所有属性直接添加到了 `Person` 的 `prototype` 属性中，构造函数变成了空函数。即使如此，也仍然可以通过调用构造函数来创建[一个新对象](#)，而且新对象还会具有相同的属性和方法。但与构造函数模式不同的是，新对象的这些属性和方法是由所有实例共享的。换句话说，`person1` 和 `person2` 访问的都是同一组属性和同一个 `sayName()`函数。要理解原型模式的工作原理，必须先理解 ECMAScript 中原型[对象](#)的性质。

#### 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 [prototype-属性原型对象](#) 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[在所在](#)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[原型对象原型属性](#)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[原型对象原型属性](#)。在很多实现中，这个内部属性的名字是 `__proto__`，而且通过脚本可以访问到（在 Firefox、Safari、Chrome 和 Flash 的 `ActionScript` 中，都可以通过脚本访问 `__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[原型对象原型属性](#)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。