原型链

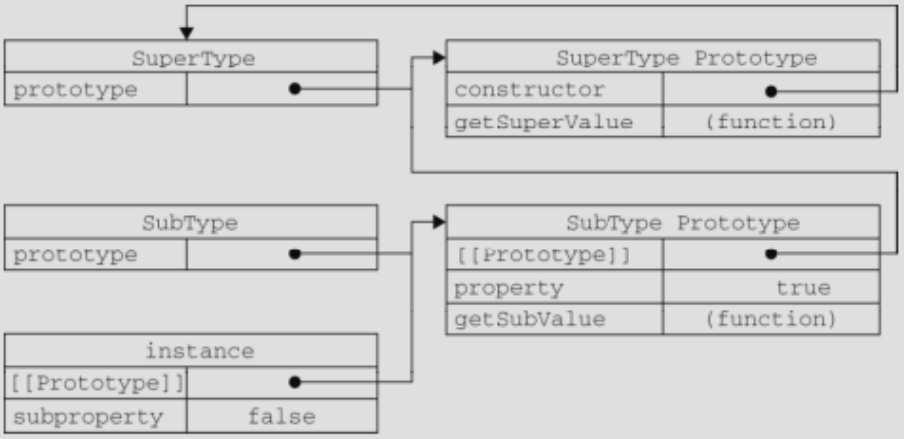
ECMAScript 中描述了**原型链**的概念，并将原型链作为实现继承的主要方法。其基本思想是利用原型让一个引用类型继承另一个引用类型的属性和方法。简单回顾一下构造函数、原型和实例的关系：每个构造函数都有一个原型对象，原型对象都包含一个指向构造函数的指针，而实例都包含一个指向原型对象的内部指针。那么，假如我们让原型对象等于另一个类型的实例，结果会怎么样呢？显然，此时的原型对象将包含一个指向另一个原型的指针，相应地，另一个原型中也包含着一个指向另一个构造函数的指针。假如另一个原型又是另一个类型的实例，那么上述关系依然成立，如此层层递进，就构成了实例与原型的链条。这就是所谓原型链的基本概念。

实现原型链有一种基本模式，其代码大致如下。

```
function SuperType(){
    this.property = true;
}
```

```
SuperType.prototype.getSuperValue = function(){
    return this.property;
};

function SubType(){
    this.subproperty = false;
}

//继承了 SuperType
SubType.prototype = new SuperType();

SubType.prototype.getSubValue = function (){
    return this.subproperty;
};

var instance = new SubType();
alert(instance.getSuperValue());      //true
```

以上代码定义了两个类型：SuperType 和 SubType。每个类型分别有一个属性和一个方法。它们的主要区别是 SubType 继承了 SuperType，而继承是通过创建 SuperType 的实例，并将该实例赋给 SubType.prototype 实现的。实现的本质是重写原型对象，代之以一个新类型的实例。换句话说，原来存在于 SuperType 的实例中的所有属性和方法，现在也存在于 SubType.prototype 中了。在确立了继承关系之后，我们给 SubType.prototype 添加了一个方法，这样就在继承了 SuperType 的属性和方法的基础上又添加了一个新方法。这个例子中的实例以及构造函数和原型之间的关系如图 6-4 所示。



在上面的代码中，我们没有使用 SubType 默认提供的原型，而是给它换了一个新原型；这个新原型就是 SuperType 的实例。于是，新原型不仅具有作为一个 SuperType 的实例所拥有的全部属性和方法，而且其内部还有一个指针，指向了 SuperType 的原型。最终结果就是这样的：instance 指向 SubType 的原型，SubType 的原型又指向 SuperType 的原型。getSuperValue()方法仍然还在 SuperType.prototype 中，但 property 则位于 SubType.prototype 中。这是因为 property 是一个实例属性，而 getSuperValue()则是一个原型方法。既然 SubType.prototype 现在是 SuperType 的实例，那么 property 当然就位于该实例中了。此外，要注意 instance.constructor 现在指向的是 SuperType，这是因为原来 SubType.prototype 中的 constructor 被重写了的缘故[①]。

通过实现原型链，本质上扩展了本章前面介绍的原型搜索机制。读者大概还记得，当以读取模式访问一个实例属性时，首先会在实例中搜索该属性。如果没有找到该属性，则会继续搜索实例的原型。在通过原型链实现继承的情况下，搜索过程就得以沿着原型链继续向上。就拿上面的例子来说，调用 instance.getSuperValue()会经历三个搜索步骤：1）搜索实例；2）搜索 SubType.prototype；3）搜索 SuperType.prototype，最后一步才会找到该方法。在找不到属性或方法的情况下，搜索过程总是要一环一环地前行到原型链末端才会停下来。