**原型和原型链**

## 一、原型模式

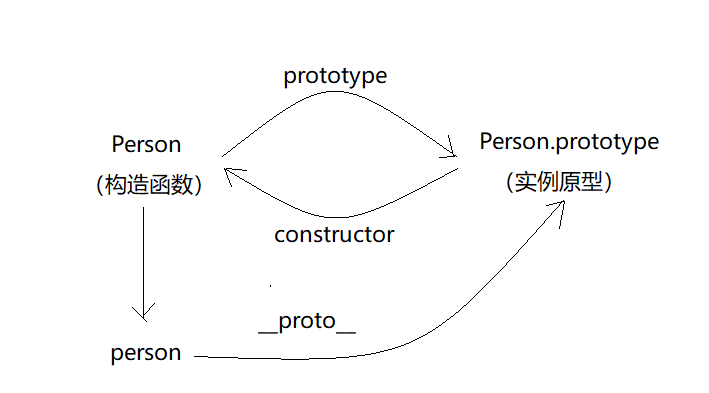
我们创建的每一个函数都有一个prototype(原型)属性，这个属性指向的是通过调用构造函数来创建出来的**对象实例**的**原型对象**，这个原型对象可以让所有对象实例**共享**它所包含的属性和方法。

|  |
| --- |
| function Person () {    }  Person.prototype.name = "xiao";  Person.prototype.sayName = function () {  alert('this.name')  }  var person1 = new Person();  var person2 = new Person();  person1.sayName() // "xiao"  console.log(person1.name == person2.name) // "true" |

上面的例子当中我们创建了一个构造函数Person，并通过它的prototype属性在它的**原型对象**中定义了name属性并赋值，然后通过调用构造函数Person实例化出来两个对象实例，通过打印出来的值我们可以得知，person1和person2共享了原型对象中的属性和方法。

### 构造函数，原型对象和对象实例的关系

我们知道每个函数都有一个prototype属性指向函数的原型对象。在默认情况下，所有原型对象都有一个constructor(构造函数)属性，这个属性指向了prototype属性所在的函数，比如前面的例子中,Person.prototype.constructor就指向Person。  
 另外，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\_\_porto\_\_属性（仅在Firefox、Safari、Chrome中支持）,这个属性指向的就是构造函数的原型对象。由此我们可以得出以下图示的结论：



通过代码来验证：

|  |
| --- |
| # 实例和原型对象之间的关系  console.log(person.\_\_proto\_\_ == Person.prototype) // true  # 也可以通过isPrototypeOf()和ES5中的Object.getPrototypeOf()方法来判断  console.log(Person.prototype.isPrototypeOf(person1)) // true  console.log(Object.getPrototypeOf(person) === Person.prototype) // true  # 原型对象和构造函数的关系  console.log(Person.prototype.constructor == Person) // true |

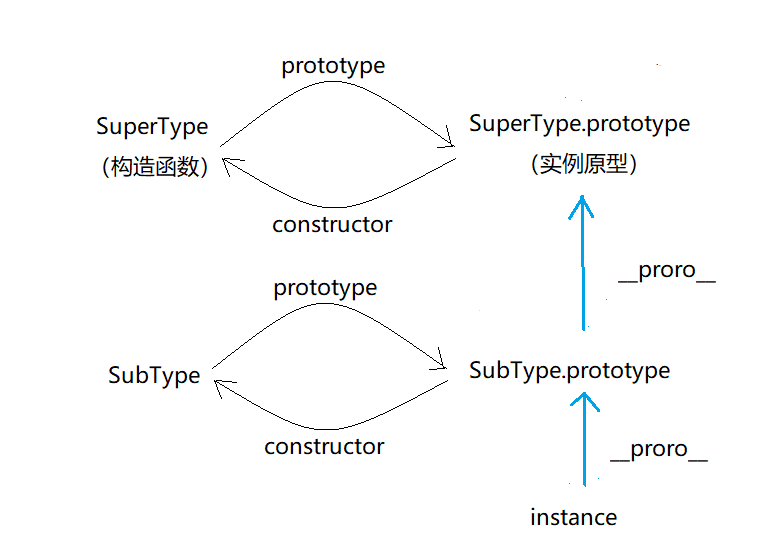
## 二、原型链

通过前面我们对构造函数，对象实例和原型对象三者关系的描述可知，实例都包含了指向原型对象的内部指针。  
那么假如现在我们有两个构造函数**A**跟**B**，我们让构造函数**A**的原型对象等于构造函数**B**的实例，根据前面的推论，这个时候**A**的原型对象就包含指向B的原型对象的指针，再假如又有一个构造函数**C**，让**A**的原型对象等于**C**的实例，上述关系依旧成立，以此类推便形成了实例与原型的链条，即原型链，它主要作为JS中实现继承的主要方法。

**原型链的基本实现**

|  |
| --- |
| function SuperType() {  this.property = true;  }  SuperType.prototype.getSuperValue = function() {  return this.property;  }  # 继承了SuperType  SubType.prototype = new SuperType();  SubType.prototype.getSubValue = function() {  return this.subproperty;  }  var instance = new SubType();  console.log(instance.SuperValue()); // true |

在上面的代码中，我们没有使用SubType默认的原型，而是将SuperType的实例赋给它，重写了SubType的原型对象；这样一来SubType.prototype的内部便具有一个指向SuperType原型的指针，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。  
 instance同理，还要注意的是由于SubType的原型指向了SuperType的原型，而SuperType的原型的constructor属性指向的是SuperType构造函数，那么instance.constructor也就指向了SuperType



**原型搜索机制**

当访问一个实例属性或方法时，在通过原型链实现继承的情况下，首先会在实例中搜索该属性，在没有找到属性或方法时，便会沿着原型链继续往上搜索，直到原型链末端才会停下来。  
这里还有一个重要的点，事实上所有引用类型默认都继承了Object，而这个继承也是通过原型链实现的，也就是说，所有函数的默认原型都是Object的实例，这也是所有自定义类型都会继承toString()、valueOf()等默认方法的根本原因。

**Object.prototype的原型**

既然所有类型默认都继承了Object，那么Object.prototype又指向哪里呢，答案是null，我们可以通过下面的代码打印试试看:

console.log(Object.prototype.\_\_proto\_\_ === null) // true

null即没有值，也就是说属性或方法的查找到Object.prototype就结束了。

这个时候，我们就可以看到完整的原型链关系图如下：

