

**原型概念：**无论什么时候、只要创建一个新函数，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。默认情况下，所有原型对象都会自动获得一个constructor属性（ 构造函数） 属性，这个属性指向prototype属性所在的函数。

注：虽然在脚本没有标准的方式访问-proto-,但在Firfox,chrome,Safari在每个对象上都支持一个属性-proto-

列：var a=new Sub();

a.myname();

1 .a.\_proto\_=Sub.prototype //在创建实例过程中 a内部\_proto\_关联到sub.prototype，当a中无法 找 到myname时，它会通过委托在sub.prototype中查找

（委托意味着某些对象在找不到属性或方法时会把这个请求委托给另一个对象，禁止互相委托）

2.函数并不是一个构造函数，它就是一个普通函数，但是当你通过New调用这个函数时，这个函数就 变成了构造函数调用，实际上，new会戒指所有普通函数并用构造函数的形式调用它。

3 a.constructor===Sub a中并不存在一个属性construtor，.constrtor同样通过默认的\_proto\_被委托 到Sub

4 重写原型：如果创建一个新的sub.prototype则 新对象并不会获取.construtor属性，该属性不再指向sub，而是指向object

但是construtor是不可枚举，值可写并且可以修改，如果需要可以重写



也可用object.defineProperty(sub.prototype,"construtor",{})定义一个新属性 **原型链是是实现继承的主要方法**

面向类继承与面向对象继承有很大的区别：面向类是将父类的值复制到子类，而面向对象继承是通过创建实例与构造函数的关联，通过委托来获取属性和方法。

**创建关联的方法有：**

**object.create()**例：Bar.prototype=object.create(Sub.prototype) Bar.prototype的\_proto\_关联到Sub.prototype（缺点：需要创建一个新对象bar.prototype,丢弃默认的的，不能修改已有的默认对象

**object.setprototype( Bar.prototype,Sub.prototype )/**/es6提供的辅助函数 修改默认的 Bar.prototype

**如何检查实例a的委托对象（继承祖先）？**

方法一：alert(a instanceof Sub )//true (instanceof测试实例与原型链中出现过的构造函数）

alert（a instanceof object)//true

方法二：isPrototypeOf()(只要原型链中出现过的原型都可以说是a实例的原型

alert（object.prototype.isPrototypeOf(a));//true

alert（sub.prototype.isPrototypeOf(a));//true

**原型链概念：**每一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（construtor),而每个实例有包含一个指向原型对象的内部指针（\_proto\_)。那么，加入让一个原型对象等于另一个类型的实例，此时的原型将包含一个指向另一个原型的指针，相应的，另一个原型也包含着指向另一个原型的指针，加入另一个原型又是另一个构造函数的实例，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，这就是原型链。

**原型链继承：**

存在问题解释

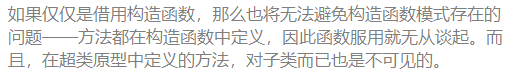
clipboard.png

sub.prototype中有colors引用类型值

**借用构造函数继承：**（解决了原型链继承的问题）

通过使用call在未来新要创建的实例环境下调用supertype，结果每一个实例都拥有自己的colors属性的副本，改动一个实例中的colors属性石另一个实例中的colors属性不变，也可传递参数

问题：



注：应该将方法写到原型中

**组合继承：**

clipboard.png

问题：调用两次构造函数

**原型式继承：**

就是通过上面介绍的方式创造关联还有var sub=object(object)sub将object作为原型

问题：出现原型链继承中引用类型值共享的问题。

**寄生式继承：**

问题 ：函数不能复用而效率降低