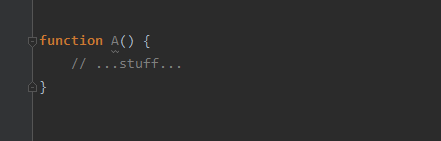
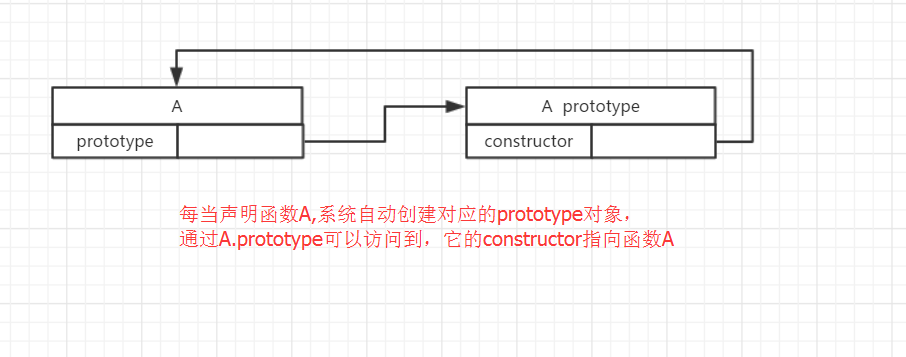
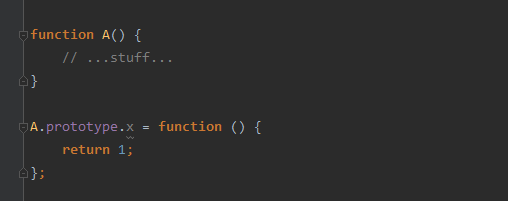
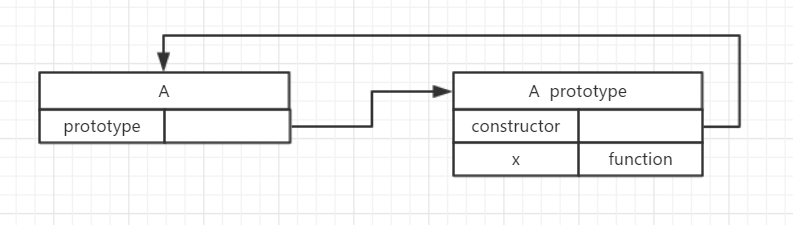
每声明一个函数，就会自动创建对应的prototype原型对象



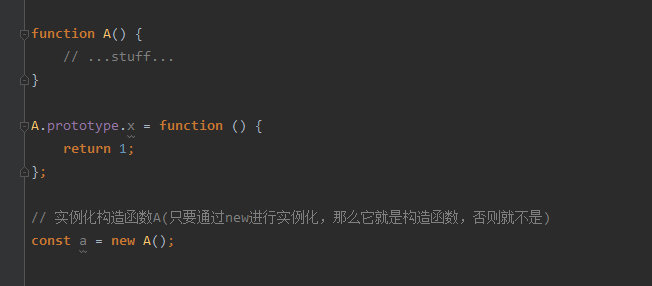


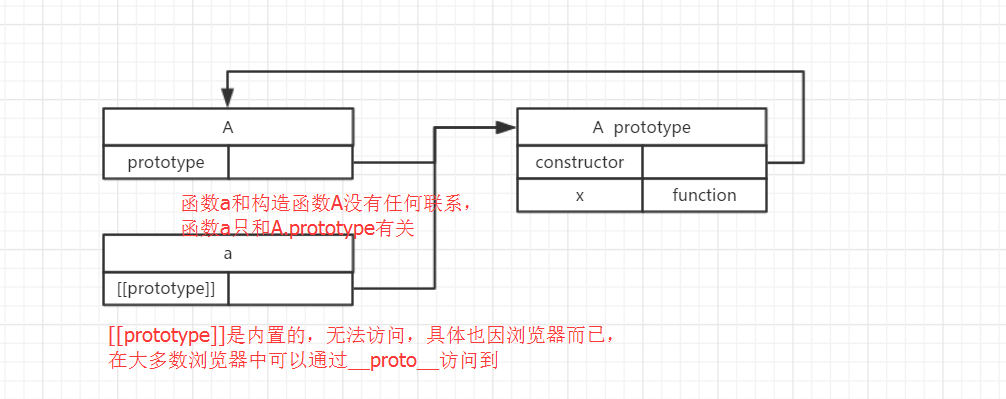
现在给A.prototype增加一个属性



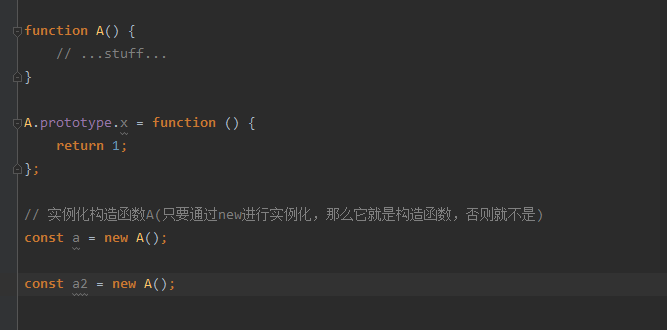


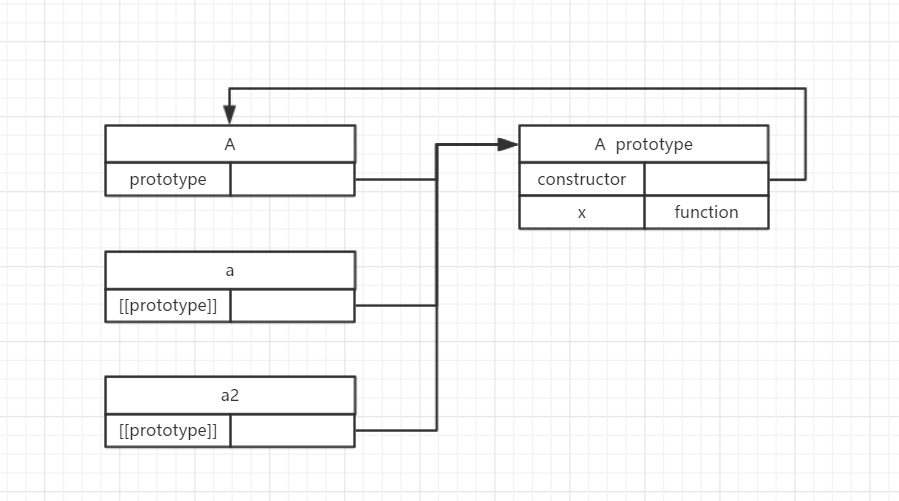
实例化构造函数A并且赋值给a，a同时可以得到A.prototype原型对象所有的属性。



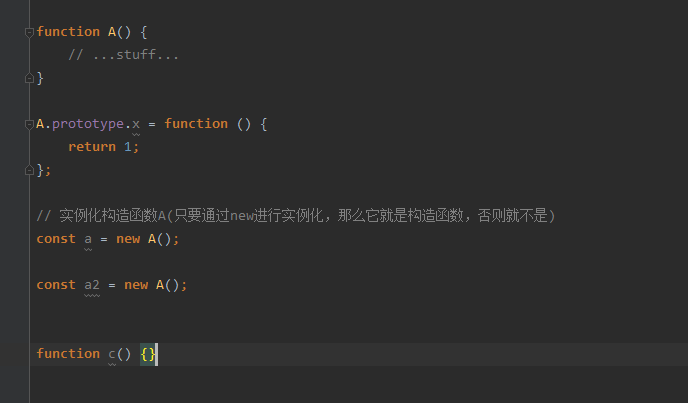


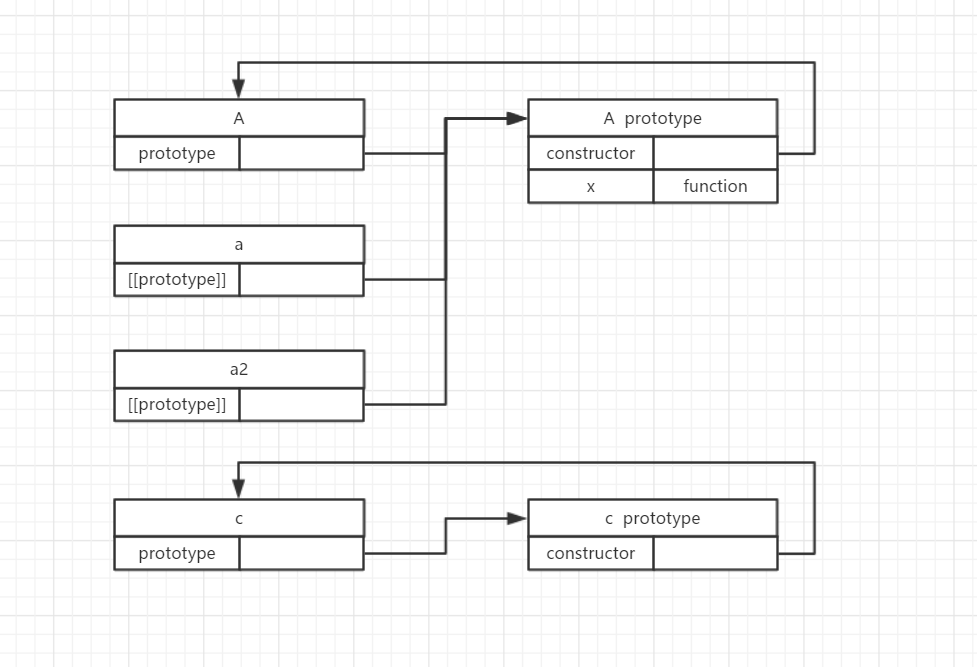
同理又创建函数a2,a2也和a一样共享原型对象A.prototype，都能访问到x属性对应的函数。





以上就是最基本的原型。

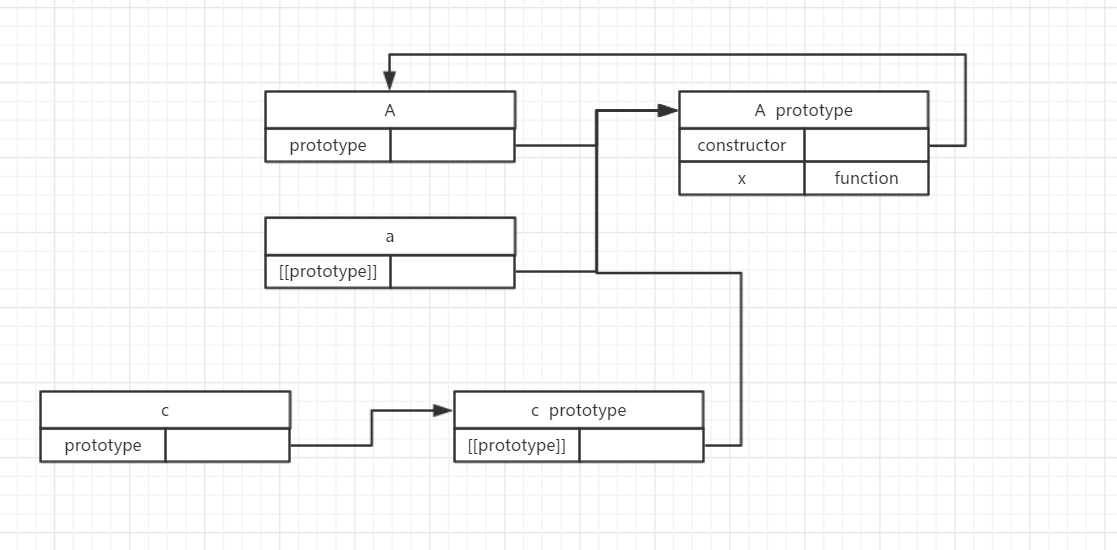
现在创建函数c,当然同时也对自动创建对应的原型对象c.prototype 



现在，必须明确一个误区，虽然A.prototype原型对象是自动创建的，但它除了系统自建之外本身和其它对象没有任何区别，可以被修改，赋值，甚至还可以实例化对应的构造函数。所以不要对prototype原型对象抱有任何“成见”。

那么完全可以改变c.prototype,更大胆地，类似实例化函数a一样，实例化c.prototype

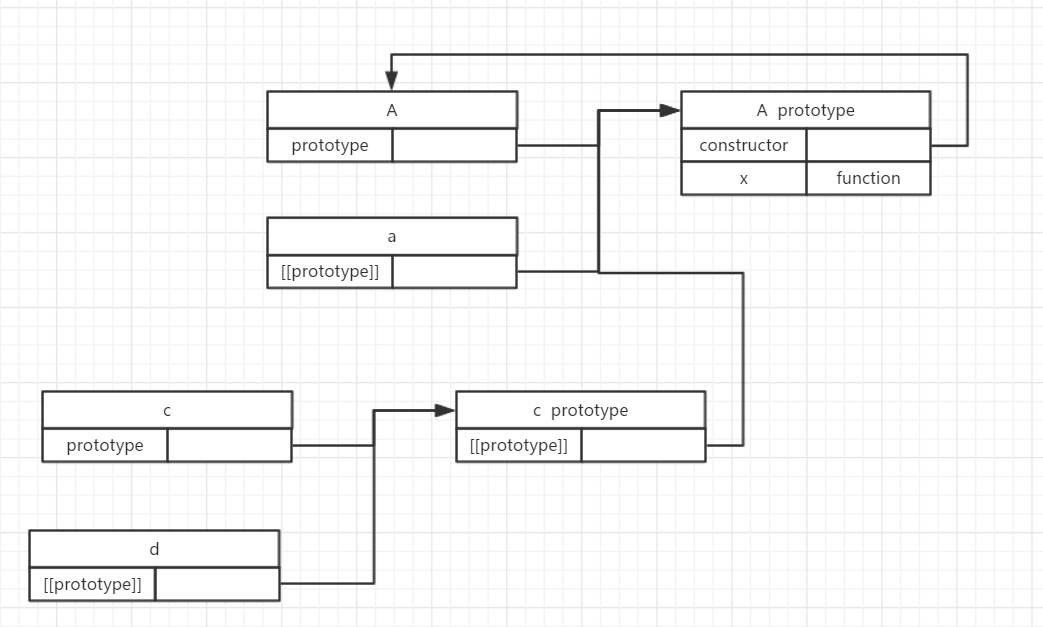




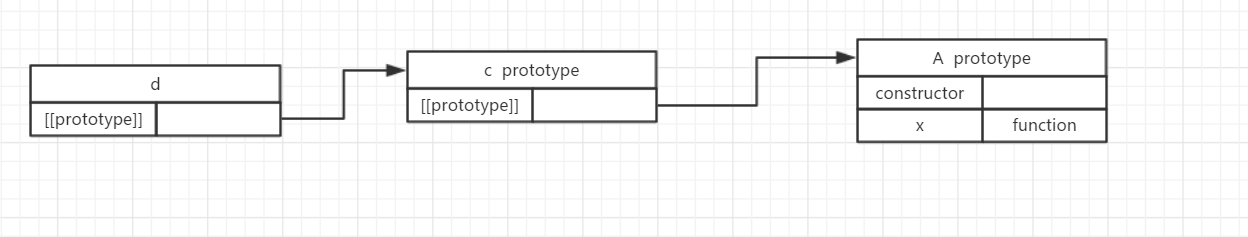
此时，c.prototype当然也能够访问到A.prototype上的x,

现在又new一个c，又会怎么样呢？





更简洁一点



d能够访问c.prototype,还能访问A.prototype。

最后,我们常用的Array这类构造函数除了系统自己内置之外，和上面定义的函数A没有任何区别，我们常用的，.toString,.push等等都分别继承自Object和Array，不过就是一次原型链的值搜索。

另外instanceof，比如 d instanceof A，它的本质含义就是左边变量的整个原型链上，是否存在右边变量的原型，d的原型链可以一一列举，c.prototype, A.prototype，当然还有Object.prototype。显然存在A.prototype，所以 d instanceof A为true;