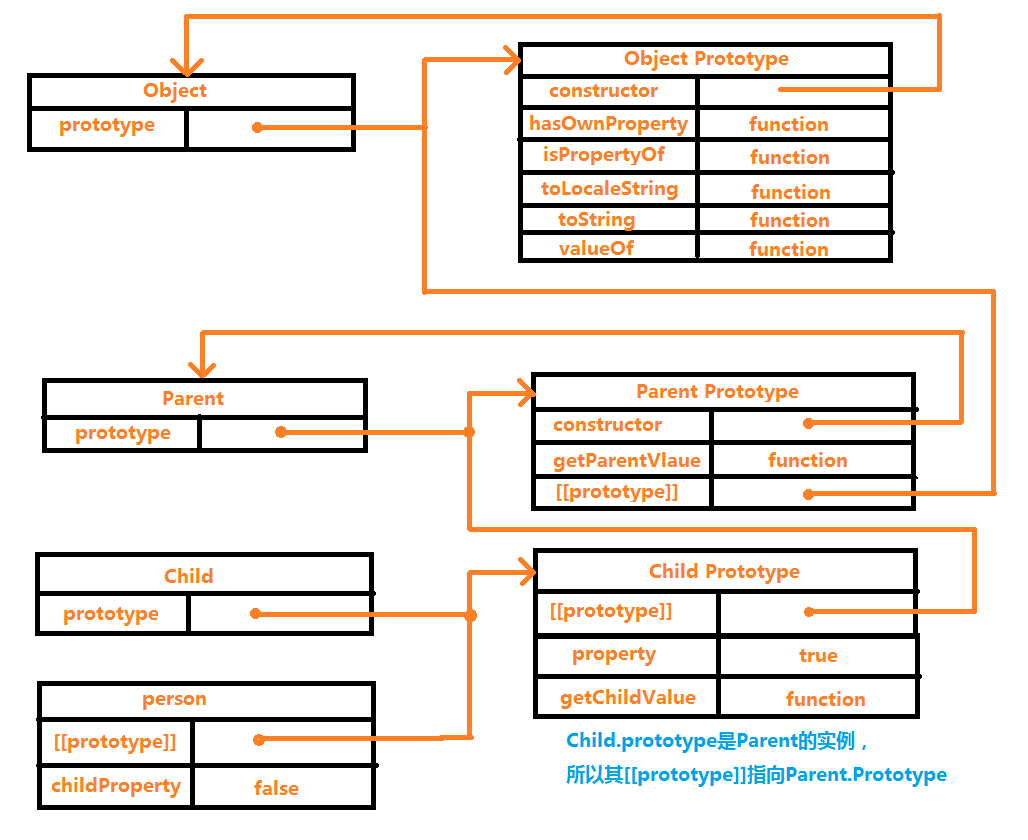
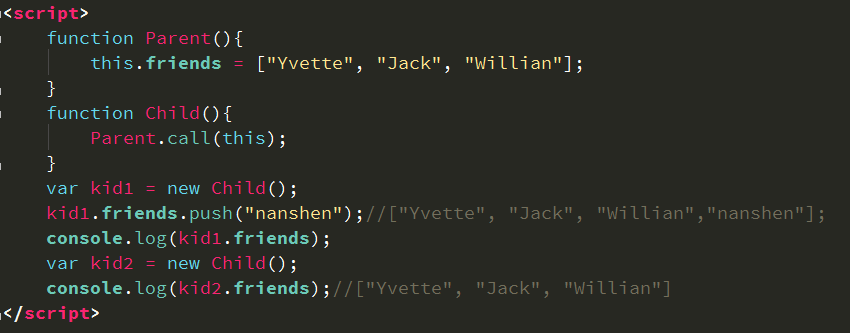
1. 基于原型链的继承  
   核心思想：将父类的实例作为子类的原型  
   代码：  
   function B(){}  
   B.prototype=new A();  
   B.prototype只生成了一份实例，所以的都继承至这一个对象。  
     
   有一个很大的缺点：原型对象的引用属性是所有实例对象共享的。  
   另一个很大的问题：在创建子类型的时候，无法向父类的构造函数传参。  
   很少使用。Vue组件data是函数的原理也在此。
2.   
   借用构造函数  
   实例 instanceOf Parent //false  
   实例 instanceOf Child //true  
   这种方法可以解决无法向父构造函数传参的问题。  
   问题在于：1.实例并不是父类的实例，只是子类的实例。2.只能继承父类的实例和方法，不能继承原型属性和方法（父类的prototype）。3.无法实现函数复用，每个子类都有父类函数的副本，影响性能。  
   很少使用。
3.   
   组合继承：  
   综合上述两种，在子类的构造函数中结用父类的构造函数（call），实现父类实例不会共享，也可以向父类构造函数传参。在外面设置，子类的prototype为父类的实例，同时别忘了设置子类的constructor指向自己。  
   缺点：调用了两次父类的构造函数，生成了两份实例，只不过子类实例将子类原型上拿分屏蔽了，多消耗了一点内存。

4.寄生组合继承：  
通过寄生方法，砍掉父类的实例属性。这样，在调用两次父类构造函数的时候，就不会初始化两次实例方法和属性。  
function Cat(name){

Animal.call(this);

this.name = name || 'Tom';

}

(function(){

// 创建一个没有实例方法的类

var Super = function(){};

Super.prototype = Animal.prototype;

//将实例作为子类的原型

Cat.prototype = new Super();

})();  
**感谢 @[bluedrink](http://home.cnblogs.com/u/1291442/" \t "_blank) 提醒，该实现没有修复constructor。**

1. InstanceOf只检查实例是不是从prototype继承来的，所以构造函数借用只是借用函数初始化，不会认为是父类的实例。这时候继承不了父类函数的prototype。  
   原型链继承虽然子类的prototype变成父类的实例，子类实例化的时候，依然继承它，除此之外，父类的实例又属于父类。但是，此时子类的constructor不再是子类函数。