在 Javascript 中，万物皆对象，但对象也有区别，大致可以分为两类，即：普通对象（Object）和函数对象（Function）。

一般而言，通过 new Function 产生的对象是函数对象，其他对象都是普通对象。

举例说明：

```
function f1(){
    //todo
}
var f2 = function(){
    //todo
};
var f3 = new Function('x','console.log(x)');

var o1 = {};
var o2 = new Object();
var o3 = new f1();

console.log(
    typeof f1,//function
    typeof f2,//function
    typeof f3,//function
    typeof o1,//object
    typeof o2,//object
    typeof o3 //object
);
>> function function function object object object
```

f1 属于函数的声明，最常见的函数定义方式，f2 实际上是一个匿名函数，把这个匿名函数赋值给了 f2,属于函数表达式，f3 不常见，但也是一种函数对象。

Function 是 JS 自带的对象,f1,f2 在创建的时候,JS 会自动通过 new Function()的方式来构建这些对象，因此，这三个对象都是通过 new Function()创建的。

在 Javascript 中创建对象有两种方式：对象字面量和使用 new 表达式，o1 和 o2 的创建恰好对应了这两种方式,重点讲一下 o3, 如果用 Java 和 C#的思路来理解的话，o3 是 f1 的实例对象，o3 和 f1 是同一类型，至少我以前这么认为，其实不然...

那么怎么理解呢？很简单，看 o3 是不是通过 new Function 产生的, 显然不是，既然不是函数对象，那就是普通对象 。

通过对函数对象和普通对象的简单理解之后，我们再来了解一下 Javascript 中的原型和原型链：

在 JS 中，每当创建一个函数对象 f1 时，该对象中都会内置一些属性，其中包括 prototype 和__proto__， prototype 即原型对象，它记录着 f1 的一些属性和方法。

需要注意的是，prototype 对 f1 是不可见的，也就是说，f1 不会查找 prototype 中的属性和方法。

```
function f(){}
f.prototype.foo = "abc";
console.log(f.foo); //undefined
```

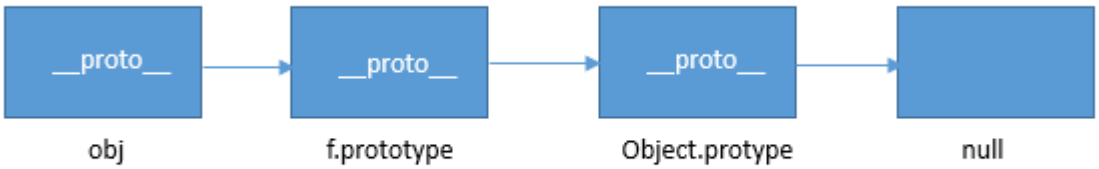
那么，prototype 有什么用呢？ 其实 prototype 的主要作用就是继承。 通俗一点讲，prototype 中定义的属性和方法都是留给自己的“后代”用的，因此，子类完全可以访问 prototype 中的属性和方法。

想要知道 f1 是如何把 prototype 留给“后代”，我们需要了解一下 JS 中的原型链，此时，JS 中的 __proto__ 入场了，这哥们长的很奇特，隐藏的也很深，以致于你经常见不到它，但它在普通对象和函数对象中都存在， 它的作用就是保存父类的 prototype 对象，JS 在通过 new 表达式创建一个对象的时候，通常会把父类的 prototype 赋值给新对象的__proto__属性，这样，就形成了一代代传承...

```
function f(){}
f.prototype.foo = "abc";
var obj = new f();
console.log(obj.foo); //abc
```

现在我们知道，obj 中__proto__保存的是 f 的 prototype， 那么 f 的 prototype 中的__proto__中保存的是什么呢？看下图：

如图所示，f.prototype 的__proto__中保存的是 Object.prototype，Object.prototype 对象中也有 __proto__，而从输出结果看，Object.prototype.__proto__ 是 null，表示 obj 对象原型链的终结。如下图所示：



obj对象的原型链

obj 对象拥有这样一个原型链以后，当 obj.foo 执行时，obj 会先查找自身是否有该属性，但不会查找自己的 prototype,当找不到 foo 时，obj 就沿着原型链依次去查找...

在上面的例子中，我们在 f 的 prototype 上定义了 foo 属性，这时 obj 就会在原型链上找到这个属性并执行。

最后，用几句话总结一下本文中涉及到的重点：

1. 原型链的形成真正是靠__proto__ 而非 prototype,当 JS 引擎执行对象的方法时，先查找对象本身是否存在该方法，如果不存在，会在原型链上查找，但不会查找自身的 prototype。
2. 一个对象的__proto__记录着自己的原型链，决定了自身的数据类型，改变__proto__就等于改变对象的数据类型。
3. 函数的 prototype 不属于自身的原型链，它是子类创建的核心，决定了子类的数据类型，是连接子类原型链的桥梁。
4. 在原型对象上定义方法和属性的目的是为了被子类继承和使用。

finish!

update at 2015-12-07 09:34:00

update at 2015-12-08 13:16:19