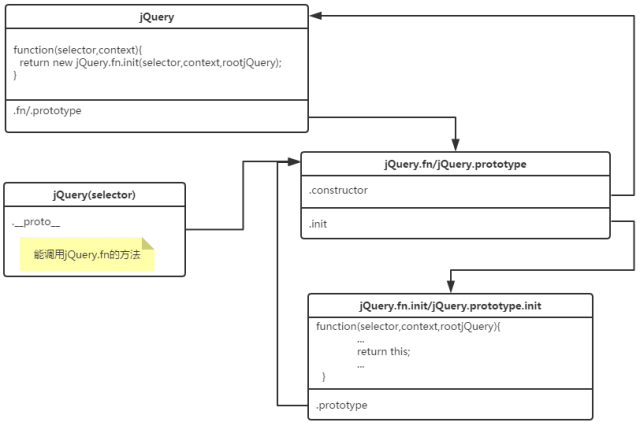
IMG_256

jQuery结构初体验

IMG_257

    jQuery集合了很多js语法的精华，下面就一起读一下jQuery，挖掘一些有用的技术和原理。

    先看一下jQuery的整体结构（为了方便，后面使用$代替jQuery）。



    从这个结构可以看出，每执行一次jQuery方法就新生成了一个实例$(selector)，同时$.fn上的所有方法都可以通过实例来访问，这样的好处就是多个实例公用相同的方法但是又互不影响；而且一但在$.fn上拓展一个属性，则所有的实例都拥有了这个属性。

    jQuery利用了js的两个知识点实现了上述的设计模式。

    实例化运算符new和对象原型prototype。

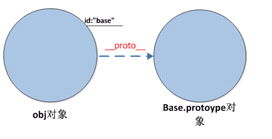
IMG_259

运算符new的深度理解

IMG_260

    这是做前端开发的一定深刻理解的基础知识之一。以网上的一个通用例子

var obj = new Base();



    new操作符具体干了什么呢?其实很简单，就干了三件事情。

var obj = {};  
obj.\_\_proto\_\_ = Base.prototype;  
Base.call(obj);

    第一行，创建了一个空对象obj；

    第二行，将这个空对象的\_\_proto\_\_成员指向了Base函数的prototype成员对象；

    第三行，将Base函数的this指针替换成obj，然后再调用Base函数，所以Base函数里面的this指的就是obj。

    注意：如果Base的返回值是对象的话则返回该对象，否则返回Base的实例

    如果Base函数如下

function Base(){

    this.age = 28;

}

    实例化对象obj将拥有属性age，并且obj.age的值为28。

    如果Base函数如下

function Base(){

    this.age = 28;

    return {}

}

    实例化对象obj是一个空对象，obj.age的值是undefined。

IMG_262

原型prototype的深度理解

IMG_263

    js有一个说法——一切皆对象。js的数据类型主要分为两种：简单数据类型（undefined/null/string/number/boolean）,复杂数据类型（对象object）。而且其中null理解为空对象。除了简单数据类型以外，其他的复杂的数据类型都是继承自原始对象，所以才说复杂类型只有一种——对象。

    这里定义几个概念，后文会用到。这里把通常意义上的对象叫做对象；而像函数、数组这类像对象一样拥有对象属性的特殊对象叫类对象；对象和类对象统称为泛对象。

    在所有泛对象当中，函数是比较特殊的一种，只有函数拥有原型对象prototype(通过funcName.prototype访问funcName函数的原型对象)。所有泛对象都有一个隐藏的属性\_\_proto\_\_，称之为继承对象。继承对象实际就是泛对象的构造函数的原型对象，即obj.\_\_proto\_\_ = obj的构造函数.prototype。

({}).\_\_proto\_\_ === Object.prototype;//true

([]).\_\_proto\_\_ === Array.prototype;//true

(function(){}).\_\_proto\_\_ === Function.prototype;//true

    要明白，所有的泛对象实际上都是相应的构造函数的实例。从深度理解new的原理同样可以理解上面的结果。

**函数的原型对象（fn.prototype）**

    这里需要理解原型对象是怎么来的。原型对象的值实际上就是在函数创建的时候,创建了一个它的实例对象并赋值给它的prototype。可以说函数的原型对象是函数的一个无传参的实例。

    过程如下（以函数fn为例）

var temp = new fn();  
fn.prototype = temp;

    先前说道所有对象都继承至原始对象。必须提的是所有函数的原型对象的继承对象就是原始对象,即fn.prototype.\_\_proto\_\_为原始对象。这其中比较特别的是Object函数，Object的原型对象就是原始对象，即Object.prototype。

var f1 = new Function();

var f2 = Function();

var fn3 = function(){}  
  
f1.prototype.\_\_proto\_\_ === Object.prototype;//true   f2.prototype.\_\_proto\_\_ === Object.prototype;//true   f2.prototype.\_\_proto\_\_ === Object.prototype;//true  
Number.prototype.\_\_proto\_\_ === Object.prototype;//true   Boolean.prototype.\_\_proto\_\_ === Object.prototype;//true

Object.\_\_proto\_\_;//null

**对象的继承属性——继承对象（\_\_proto\_\_）**

    实际上js没有继承这个概念，但是\_\_proto\_\_却起到了类似继承的作用。所有的对象起源都是一个空对象，我们把这个空对象叫做原始对象。所有的对象通过\_\_proto\_\_回溯最终都会指向（所谓的指向类似C中的指针，这个原始对象是唯一的，整个内存中只会存在一个原始对象）这个原始对象。

    用下面的例子佐证

var o = new Object();

var f = new Function();

var a = new Array();

o.\_\_proto\_\_ === Object.prototype;//true

f.\_\_proto\_\_.\_\_proto\_\_ === Object.prototype;//true

a.\_\_proto\_\_.\_\_proto\_\_ === Object.prototype;//true

Object.\_\_proto\_\_.\_\_proto\_\_ === Object.prototype;//true

Function.\_\_proto\_\_.\_\_proto\_\_ === Object.prototype;//true

    原始对象的\_\_proto\_\_为null。

    有一个比较特殊的点。Function是函数实例的构造函数。所有函数都是Function的实例，也就是说函数都继承自原始函数，Function本身也不例外。

//Function比较特殊，他的原型对象也就是原始函数

Object.\_\_proto\_\_ === Function.prototype ;//true  
Function.\_\_proto\_\_ === Function.prototype;//true  
  
//而原始函数又继承自原始对象  
Function.prototype.\_\_proto\_\_ === Object.prototype;

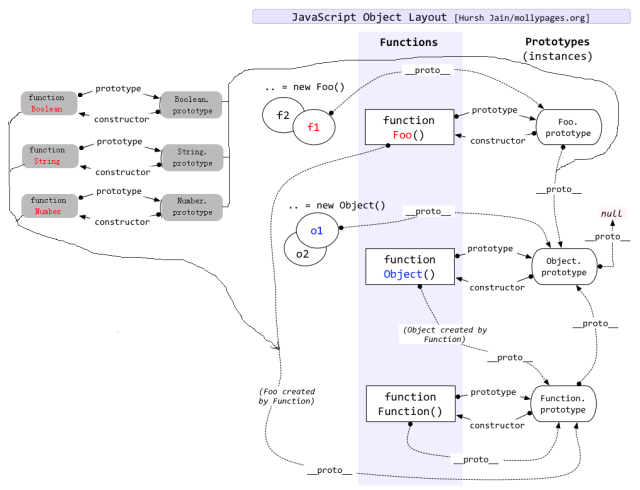
    所以往往用Function.prototype表示原始函数。

IMG_264

原型链

IMG_265

    总结一下原型对象和继承对象。套用国外网友总结的一张图。



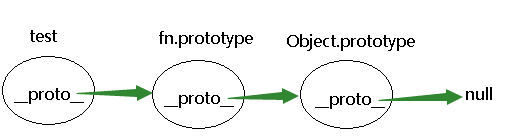
    这里面的关键点有三个，上面都已经提到了。

1. 原始对象Object.prototype。
2. 函数的原型对象是函数的一个无传参实例
3. 原始函数Function.prototype。

    在使用new方法实例化函数的时候得到的新对象的\_\_proto\_\_属性会指向函数的原型对象，而函数的原型对象又继承至原始对象。所以呈现以下结构

function fn(){};

var test = new fn();



    把这个有\_\_proto\_\_串起来的直到Object.prototype.\_\_proto\_\_为null的链叫做原型链。原型链实际上就是js中数据继承的继承链。

IMG_268

总结

IMG_269

    当认真学习了new和原型相关的知识以后就会发现jQuery巧妙的利用他们实现了jQuery以下的特征。

* 执行一次$(selector)生成一个新的实例，保障了独立的作用域
* 绑定到$.fn上的属性所有实例都可以访问，jQuery便捷的插件扩展机制实际就是在$.fn添加方法。

**本文作者:**  预知子（农金圈研发团队)，热爱技术的前端攻城狮。