博客管理控制台

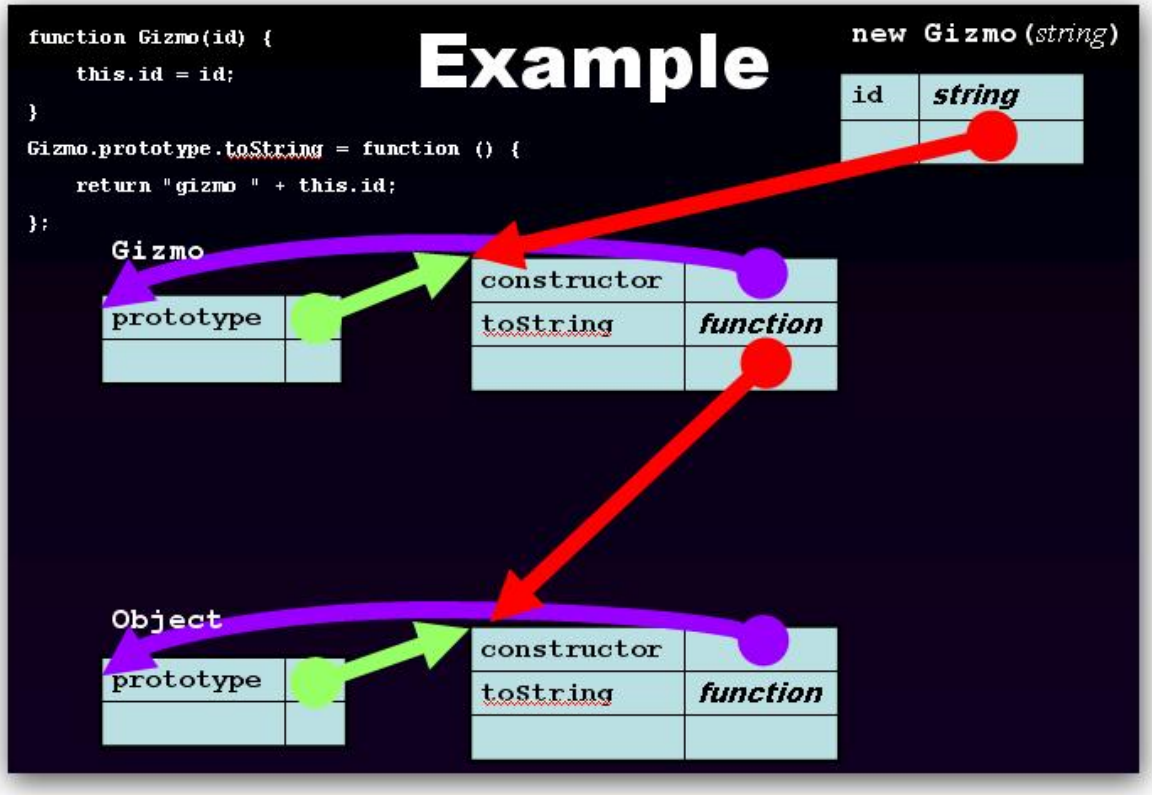
2009-07-26

浅析Javascript原型继承

关键字: 原型继承

JS没有提供所谓的类继承，据说在2.0中要加入这种继承方式，但是要所有浏览器都实现2.0的特性那肯定又得N多年。昨天看了crockford的一个视频，里面讲解了一下JS的继承方式，按照PPT里面说的，一共分了三类：Prototypal，pseudoclassical，Parasitic Inheritance。

下面主要介绍一下原型继承：**When a function object is created, it is given a prototype member which is an object containing a constructor member which is a reference to the function object.**



这里先区别一下什么是prototype属性，和constructor属性。也就是要区别什么是构造器，函数，对象实例。

其实在JS中构造器就是函数，函数就是构造器，对象实例就是通过var obj=new 函数();这种形式新建出来的实例。区别这些，在说prototype和constructor。从上面的英文中可以看出，prototype是个对象，里面定义了一个constructor，那么我们可以推论出，constructor是对象实例的属性！而不是函数（构造器）的属性。反过来，prototype是函数（构造器）的属性，而不是实例的属性！

```
//在下面这个代码示例中，MyObj是函数（构造器），obj是实例

function MyObj(id){
   this.id=id;
}

var obj=new MyObj(1);

alert(MyObj.constructor)   //本地代码
alert(obj.constructor)     //MyObj.toString()

alert(MyObj.prototype)     //[object Object]
alert(obj.prototype)       //undefined
```

我们可以看出MyObj是不具有JS意义下的constructor属性的，为什么这么说呢。alert(MyObj.constructor)这行代码还是有东西的：

这是因为MyObj是个函数，所以他的构造器就是本地的Function对象，也就是说MyObj是由Function构造出来的。但是这个对我们意义不大，因为这已经不再JS层面上了。所以这里可以认为MyObj不具有JS意义下的constrcutor属性。

alert(obj.prototype)通过这行我们可以看出，obj实例是不具有原型属性的。OK，现在区别清楚了这些，可以看原型继承了。如果不区别清楚这个，恐怕下面会看的很晕。

```
function Gizmo(id) {
   this.id = id;
}
Gizmo.prototype.toString = function () {
   return "gizmo " + this.id;
};

function Hoozit(id) {
   this.id = id;
}
Hoozit.prototype = new Gizmo();
Hoozit.prototype.test = function (id) {
   return this.id === id;
};
```

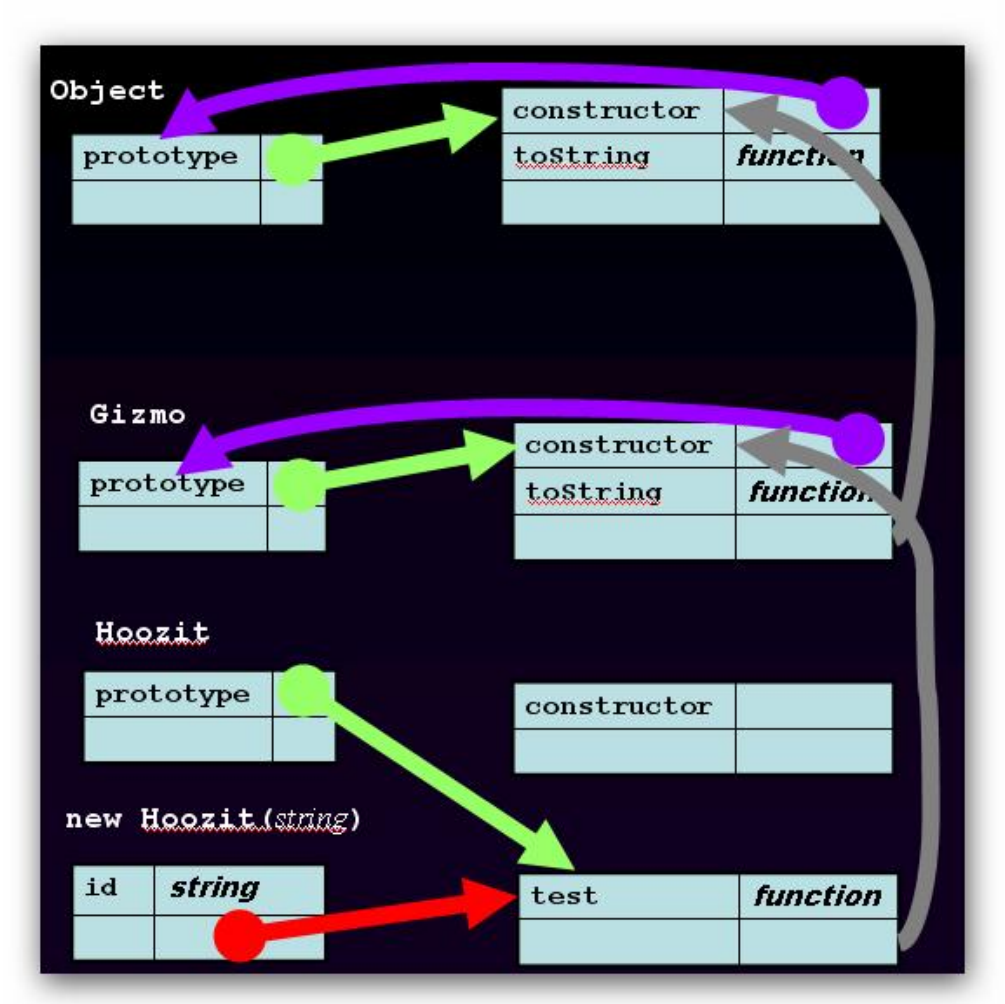
注意这行:Hoozit.prototype = new Gizmo();这行就是原型继承的核心代码。

还有要注意的是只有在new Gizmo()之后，才能添加test等其它方法，这个顺序不能倒过来！如果你先添加了test等方法，然后在new Gizmo()，那么原来添加的那些方法都将找不到了。具体原因，下面分析完了，也就清楚了。

```
Hoozit.prototype.test = function (id) {
   return this.id === id;
};

Hoozit.prototype = new Gizmo(2);

var h=new Hoozit();
alert(h.test(3));            //这里会报错！！
```



仔细看一下上面的图，这个就是原型继承的图示。左下角new Hoozit(stirng)代表的是新建的一个对象。为

了以下表述方便，我们管他叫objH1。最右边的灰色的箭头就是原型继承链。

根据文章开头的那段英文，我们知道每个函数都有一个原型，这个原型是个对象，并且对象里面包含一个constructor属性。其中Object，Gizmo，Hoozit就是函数，可以看出里面都有一个prototype项，而这个prototype又指向一个对象，这个对象里面又有一个constructor属性，constructor又指回自身。

```
alert(Gizmo.prototype.constructo===Gizmo) //true
```

但是这里有一个意外，我们发现Hoozit原型对象里面没有constructor属性，而这个函数的右边却有一个空的对象，里面包含了一个constructor属性？为什么呢？

这个问题会发生在原型继承过程中。主要就是因为Hoozit.prototype = new Gizmo();这句话引起的。这句话的意思就是新建了一个Gizmo对象并且赋给Hoozit的原型！那么，那么，仔细想想，是不是Hoozit原有的原型对象就被断开了呢？？没错，就是这样。所以那个有constructor属性的空对象再也访问不到了！

那现在又有一个疑问了，通过Hoozit.prototype = new Gizmo();这行代码之后，Hoozit.prototype.constructor指向哪里了呢？很简单，知道(new Gizmo()).constructor指向哪里吗？通过上面的图，可以清晰的看出来指向的是Gizmo函数。所以我们断定，现在的Hoozit.prototype.constructor也指向了那里！

```
alert(Hoozit.prototype.constructor===Gizmo); //true
```

上面这行代码验证了我们的猜测！OK，上面讲完了函数（构造器一边），然后我们再来说实例对象这边：每个实例对象都有一个constructor属性，并且指向构造器（函数）。而且每个new出来的实例都是某个原型constructor的实例：

```
var objG1=new Gizmo()
alert(objG1 instanceof Gizmo.prototype.constructor) //true
alert(Gizmo.prototype.constructor===objG1.constructor); //true
```

上面为什么不拿objH1举例呢，因为他的constructor已经不是他自己的了，而是Gizmo对象的，那么我们验证一下：

```
alert(objH1.constructor===objG1.constructor) //true
alert(objH1 instanceof Gizmo.prototype.constructor)          //true
```

看到了吗？其实这个问题也不算什么大问题，如果你要不使用instanceof检查父对象或者使用constructor进行原型回溯的话，这个问题可以不解决了。如果想解决这个问题怎么办呢？在Prototype框架的Class.create方法里面给出了一种方法，具体可以参考：http://blog.csdn.net/kittyjie/archive/2009/07/13/4345568.aspx

下面我简单说一下两种方法：

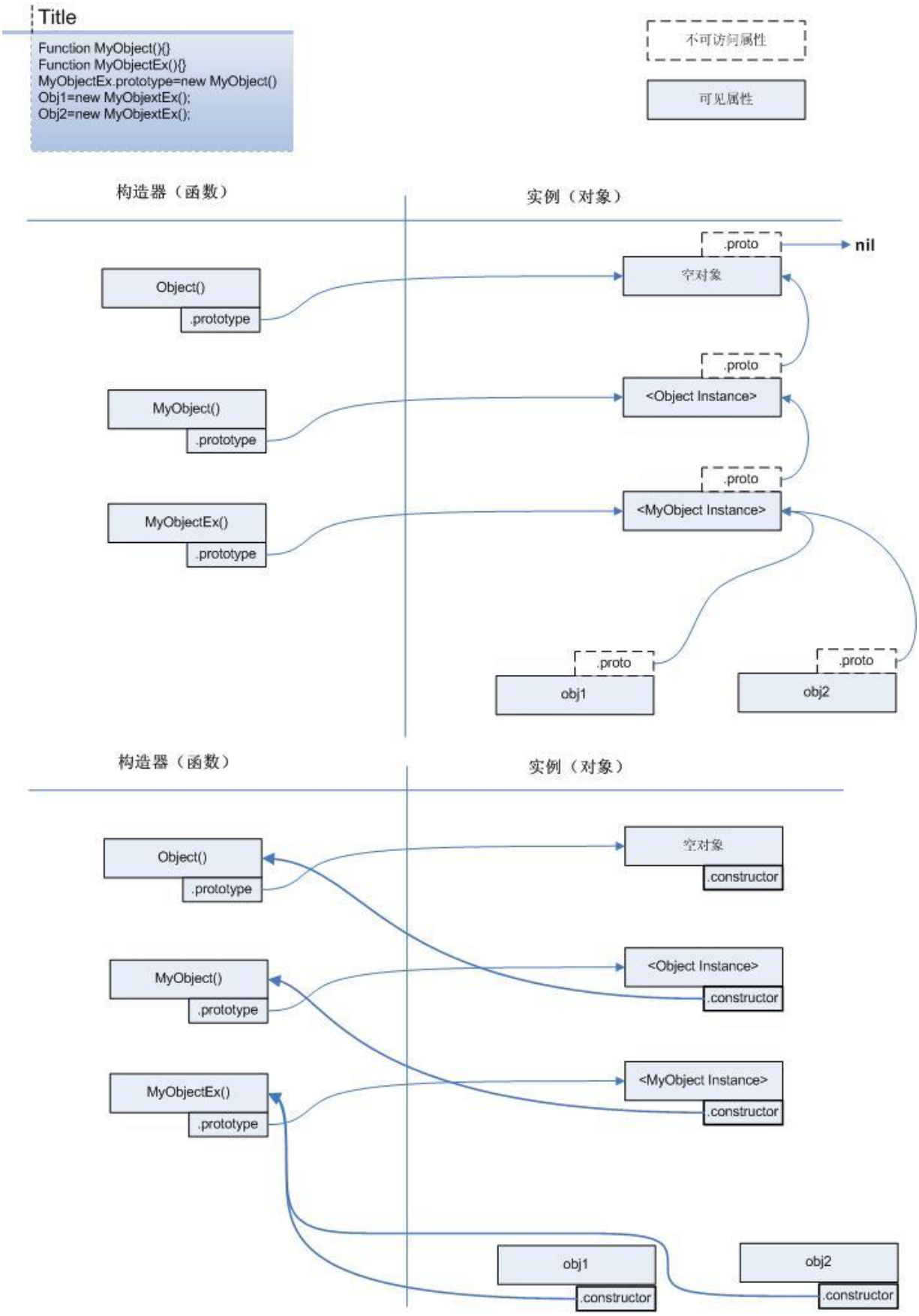
```
//方法一
//Prototype框架采用了此种方法
Hoozit.prototype = new Gizmo(2);
Hoozit.prototype.constructor = Hoozit;

//方法二
function Hoozit(id) {
   this.id = id;
   this.constructor=Hoozit;
}
```

//具体两种方法有什么区别，请参考《JAVASCRIPT语言精髓与编程实践》158~159页。

有兴趣的可以结合上面的图，想想这两种方法的Hoozit.prototype.constructor应该放在图的什么位置？想不明白的可以和我在交流。

下面看一下《JAVASCRIPT语言精髓和编程实践》书上的一张图（156~157页）：


Title
Function MyObject(){}
Function MyObjectEx(){}
MyObjectEx.prototype=new MyObject()
Obj1=new MyObjextEx();
Obj2=new MyObjextEx();
不可访问属性
可见属性
构造器（函数）
实例（对象）
.proto
nil
空对象
Object()
.prototype
.proto
<Object Instance>
MyObject()
.prototype
.proto
<MyObject Instance>
MyObjectEx()
.prototype
.proto
obj1
.proto
obj2
构造器（函数）
实例（对象）
Object()
.prototype
空对象
.constructor
<Object Instance>
MyObject()
.prototype
.constructor
<MyObject Instance>
MyObjectEx()
.prototype
.constructor
obj1
.constructor
obj2
.constructor

个人认为这张图下面的那个constructor属性的指向是不是有问题？书上面表达的意思肯定没有问题，但这张图画的很困惑。和我上面的解释不太一样？先不说这个了，大家自己研究研究吧。

下面我们再来说一下我给出的原型继承图中的右边灰色的箭头部分。从这部分能够看出继承的关系。所有未经过变动的函数（构造器）的原型，他们都会继承自Object对象。

```
alert(Gizmo.prototype instanceof Object.prototype.constructor);    //true
```

这样原型的继承关系就连接起来了。其实说白了，就是一个函数的prototype链向另一个函数实例，然后不断的这样进行下去，最上面的函数链接Object至对象实例，OK，所有函数就都连接起来了。

PS：

“还有要注意的是只有在new Gizmo()之后，才能添加test等其它方法，这个顺序不能倒过来！“ 这个问题是不是清楚了呢？

==========

下面看几个例子，说明几个问题：

```
function Gizmo(id) {
   this.id = id;
this.ask=function(){
alert("gizmo--ask:"+this.id);
}

function privateMethod(){
return "gizmo--privateMethod";
}

privateMethod2=function(){
return "gizmo--privateMethod2";
}
}
Gizmo.prototype.toString = function () {
   return "gizmo--toString:" + this.id;
};
Gizmo.prototype.id="gizmo3";

function Hoozit(id) {
   this.id = id;
}
Hoozit.prototype = new Gizmo("gizmo1");


var g=new Gizmo("gizmo2");
var h=new Hoozit("hoozit");
```

问题一：

```
h.ask=function(){
alert("h.ask");
}
h.ask();
delete h.ask;                    //"h.ask"
h.ask();                         //"gizmo--ask:hoozit"
delete h.id
h.ask();                         //"gizmo--ask:gizmo1"
delete Hoozit.prototype.id
h.ask();                         //"gizmo--ask:gizmo3"

/*
这里要说明的问题是:对象是如何找到属性和方法的?
第一步：先在实例对象上找ask方法，找到了，调用。第一个ask说明了这个问题
第二步：如果实例上没有ask方法，在自己的原型对象里面找ask方法，找到调用（没有给出这样的示例）
第三步：如果自己的原型中没有，回溯原型链，在父原型链中找ask方法，找到调用，第二个ask说明了这
个问题，这里会一直递归找到Object对象，如果还没找到，那就会报错了
*/

/*
再来看属性查找：
首先找实例对象上的属性，所以第二个ask输出"gizmo--ask:hoozit"，即id="hoozit"
然后找自己原型中的属性，删除掉h.id之后，找到原型上的属性，所以id="gizmo1"
接着递归原型链，找父对象原型中的属性，一直找到Object对象，所以删除掉Hoozit.prototype.id之后，
id="gizmo3"
*/
```

问题二：

```
Gizmo.prototype.question = function () {
   alert("gizmo--question:" + this.id);
};
h.question();

/*
方法可以随时添加，添加完之后就可以调用
*/
```

问题三:

```
Hoozit.prototype.toString = function () {
   return "hoozit--toString:" + this.id;
};
alert(h);
delete Hoozit.prototype.toString;
alert(h);

/*
这个问题和问题一有些重复，这里更清楚的看出，删除掉自己原型上的方法之后，就会找父原型中的方法

*/
```

问题四：

```
h.question.call(g);
alert(h.toString.call(g));

h.question.apply(g);
alert(h.toString.apply(g));

/*
可以利用apply和call方法把要调用的方法绑定到其它实例。通过结果可以看出上面那种方法调用输出的id是g对象的，而不是h对象的
*/
```

问题五：

```
alert(h.privateMethod());
alert(g.privateMethod2());

/*
上面的任意一个调用都会报错，也就是说通过显示命名函数或者匿名函数但是不加this的声明方式定义在函数之内的函数，是不能被外界访问的。这里一定注意第二种private方法声明，省略了this外面就访问不到了！
*/

/*
命名函数：(函数名字为func1)
function func1(){}

匿名函数：(注意这里，func1不是函数的名字，仅仅是个别名而已，可以通过func()来调用这个匿名函数)
func1=function(){}
*/
```

- 查看图片附件

kittyjie

博客管理

- 发表文章
- 分类管理
- 相册管理
- 收藏管理
- 评论管理
- 留言管理
- 做电子书

文章管理

- 全部博客 (1)
- 草稿箱 (0)

论坛文章管理

- 我的论坛帖子
- 我的隐藏帖子

其他管理选项

- 好友管理
- 链接管理
- 新闻管理
- 个人资料
- 博客设置
- 博客统计
- 博客导入

说点啥吧