我对原型对象中this的一个懵懂错误认识

这篇文章发布于 2013年01月29日，星期二，18:00，归类于 JS实例。 阅读 108096 次, 今日 5 次 31 条评论

一、关于学习的吐槽

有时候，吐槽的东西比技术本身更受用！

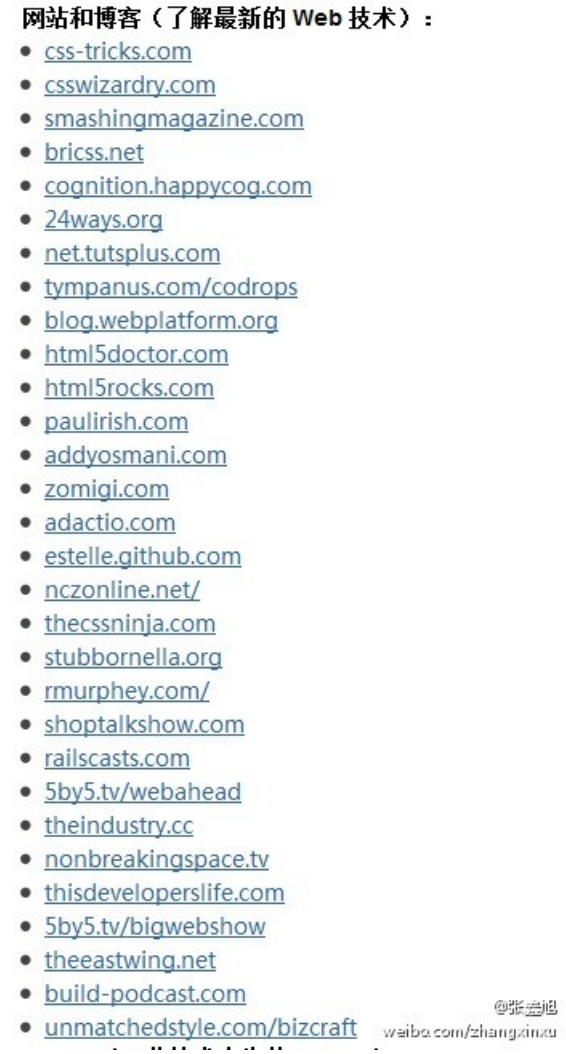
我的学习分三个大方向：一是云游四海，集百家之言；二是邯郸学步，依葫芦画瓢；三是磨刀霍霍向猪羊。

1. 云游四海，集百家之言

这也分为三个方向：一是国外有彩头的前端站点；二是微博上晃荡出来的A文B文；三是个人站点订阅以及一些UED团队订阅。

① 国外站点

如下截图，我留作备忘的，谁整理来着？不记得了，对不住这位小哥了，宣传你的机会给丢了~~



可惜1天只有24小时，除去拉屎吃饭睡觉觉、把妹聊天瞎闹闹，所剩时间有限的很，不可能把所有文章都

细读。因此，大部分文章快思维，知其大概；少部分文章很有兴趣，可以慢思维，这需要集中注意力，劳力伤神，也慎用。还有，英文很重要，前端技术文章颠来倒去就那些词，看多了，自然就速度了。

② 瞎晃悠
在一般的工作模式下，我隔三差五就去微博晃悠。一来是享受各种扑面而来的杂七杂八的信息；二来catch一些关键字、或nice文章，工具链接等。

看似晃悠，也能接触很多对工作学习有用的信息，所谓寓教于乐，似乎就有这种味道。例如最近折腾过的uglifyjs以及Opera回流测试工具都是在微博上瞎晃悠出来的。

其实上班闲晃没什么不好，只要有颗随时把学习放在心上的敏锐的心就好！

③ 订阅
唉，我现在订阅看得少了，因为……丫的更新太不勤快了，很多站点基本上就废掉了，还有的迁移到国外去了。估计是有家室了，给孩子换尿布太花时间了；或者因为新西兰奶粉太贵，赚外快去了！

一些团队UED博客，啧，很多也都废了，有点年把都不更新了，貌似盛大，搜狐之类就是这样。“可见一斑”这个词真是太那个应景了，淘宝UED估计是更新最勤快的，看人家公司的发展……

2. 邯郸学步，依葫芦画瓢
模仿没什么不好。当我们还是小白的时候，模仿是最好的学习方式，到了一定境界，自然有机会形成自己的风格。我的绘画，以及写作都是这么来的，从模仿开始。

话说这模仿的对象很重要。什么北大青鸟之流的都是毒害初学者的，因为这些人一开始“模仿”的东西就是坨褐色的冰淇淋。多去看看jQuery等优秀JS框架的书写，这个东西绝对靠谱。

然而，每天不学点东西，就想撞墙的学霸毕竟少数。如果你让你的下手：“最近没什么活，你去把jQuery的源码看看。”你觉得会发生什么？
拿我自己举例，如果让我去做这些事情，我是会download jQuery测试版js, 然后打开之；然后看到洋洋洒洒，大气磅礴的代码，我会立马感受到我在宇宙中的渺小；即使硬着头皮精读部分语句，也因“不是我不爱攀登，只因前山名叫珠穆朗玛”放弃之……

因此，纯粹去看jQuery源码是不实际的。如何做？寻找兴趣点，逐个击破之。比方说，我想知道一个DOM元素上的CSS值，咋办，咋实现？貌似用到getComputedStyle，具体怎么操作呢？此时，查看jQuery源码~，很赞的选择。千万不要去网上copy那些蛋疼菊紧害人的代码，虽然有功能，但是，书写啊命名啊等实在不敢恭维，当然，喜欢给自己挖坑的人随意。

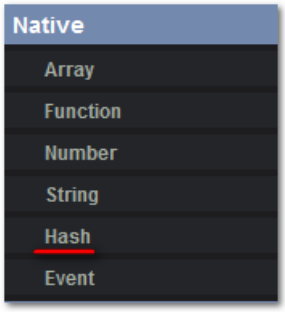
就我而言，我经常会有这样那样的idea, 我想实现这个，我想试试那个，于是，果断尝试之，遇到或没有遇到问题，我都会参考优秀的JS源代码，这让我学到了很多东西，因为这些毕竟是团队贡献的世界级的产品。

③ 磨刀霍霍向猪羊
拿刀上阵砍猪羊，指定就是实战，实际项目。这是另外的不可或缺的学习与历练，知识的巩固提升，感性认知的升华。然而，单纯的实践经验往往是狭隘的，单纯做项目、不利用业余时间学习的人，最后只会变成自我感觉良好，实际战斗力只有5的渣子。

二、我想构建一个Hash方法

我想构建一个 `Hash` 方法，什么时候有这个想法的呢？不记得了，可能是悠哉拉翔的时候，也可能是浑噩瞌睡的时候~~对自己的学习有什么用呢？我也不知道，只是觉得一番揣摩下来，应该会有所收获~~

Hash为何物?
在MooTools框架中， `Hash` 与 `Array` , `Number` , `Function` 等并成为原生类。



说穿了就是在原生`object`对象上的封装，MooTools的其他原生类都是在原生对象上做的扩展（这其实是糟糕的），唯独这个`Hash`，有封装，`object`还是那个纯洁的`object`. 原因可能是`object`为造物主转世，不好在他身上扎刺。//zxx: 猛然想起一个冷笑话——一个绣花针在公交车站伸手拦下了一只刺猬。

用法如下：

```
var obj = { a: 0, b: "1"}
    , myHash = new Hash(obj);
```

这个`myHash`保留了原生`object`的所有特性，同时有N多其他扩展，例如`each`方法（遍历），`hasValue`方法（检测是否有某值），`map`方法（都懂的），以及我比较喜欢的`toQueryString`方法（对象转为查询字符串）等。在实际处理的时候，很好用的。

我对实现没兴趣

这种想法不仅对用户如此，对于同行之间也是这样；人们总是乐于接受表面的、可以快速思维、不损耗精力的事情。因此，我是不会愚蠢到简单的代码秀，分享自己一点所学显然更有简单畅快之感。

三、这是个完整的故事

一个完整的故事会有起因，经过以及结果，我对对象字面量中`this`的认识也是如此。

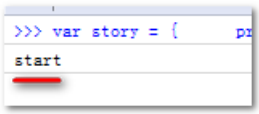
1. 事情的起因

平时接触的都是下面的故事：

```
var story = {
    progress: "unknown",
    start: function() {
        this.progress = "start";
    }
};
```

然后如下执行的时候，结果就是：

```
story.start();
console.log(story.progress);
```

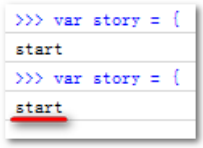


将故事中标红的`this`改成`story`，结果也是一样的：

```
var story = {
    progress: "unknown",
```

```
    start: function() {
        story.progress = "start";
    }
};

story.start();
console.log(story.progress);
```



因此，很自然而然的，在理解上，我就会将`this`等同于`story`，也就是`this`等于这个对象本身①

。

> //zxx: ① 实际上`this`并不能完全等同于`story`。如下例子：
>
> 结果就是”`unknown`“. //zxx: 此时window.progress为”start”;
>
> 而执行`story.progress = "start";`的结果则依然是：”`start`“.

2. 事情的经过

如下，很简单的几行原型相关的代码：

```
var Story = function() {};
Story.prototype = {
    progress: "unknown",
    start: function() {
        this.progress = "doing";
    }
};

var myStory = new Story();
myStory.start();
console.log(myStory.progress); // doing
```

结果是`doing`不难知道，关键在认识上。

`prototype`这东西有点像屌丝过滤器，不解释。

我刚接触原型概念的时候，被一堆解释搅得就像被抛弃在北京的雾霾天气里，不仅看不见方向，还胸闷！那种感觉就像是在步行街上遇到还没参加超女的张含韵，我不认识她，她也不认识我。

后来，生搬硬套，囫囵吞枣，也能用用：搞些扩展啊，继承啊什么的。然而多浮于表面，虽专门花了功夫去深入理透，但因大雾过于磅礴，依然理不清啊！这种感觉就像是张妹子参加超女红了，我认识她，她却不认识我，但是我仍能感受其表面的风采。

如果以上面这种状态理解为何结果为”`doing`“，可能自己那个未经深入思考的理解就是错误的。

我的错误理解

我以为，当执行了`myStory.start();`的时候，相当于这个：

```
progress: "unknown", → this.progress = "doing";
↓
progress: ~~"unknown"~~ "doing",
↓
myStory.progress = "doing"; √
```

图片示意就是：

骚年，不要开小差啊，上面的示意是错误的.错误的..错误的.....

错误的理解源于“故事的开始” – 普通对象字面量中的 `this` 这样理解很说得通的；以及对原型浮于表面的肤浅认知（有很多的感性成分）。

如何发现错误的?
所谓“实践出真知”，一点不假。如果整天睡在冻床上不知冷热，你则会————————变成小龙女！

我着手构建一个通过 `new` 构造的哈希方法(`Hash`)（本文part2），其中，有个很方法，叫做 `has` , 用来检测对象是否有某关键字（键值对中的键）（不包括原型中的）。其实本质上就是 `hasOwnProperty` 方法。

例如上面那个 `myStory` 实例，继续下面的检测代码：

```
console.log(myStory.hasOwnProperty("progress"));
console.log(myStory.hasOwnProperty("start"));
```

结果是什么呢？请选择：

- true, true
- true, false
- false, true
- false, false

按照我之前的错误认识，结果应该是两个 `false` . 而实际上，结果是 `true, false` .

显然，自己的理解是有错误的，而且现在看来，这种错误的理解是很低级与SB的。

错误在哪里?
一句话：*原型中的this不是指的原型对象，而是调用对象。*

换个通俗的解释。 `Story` 是个儿子模板， `Story.prototype` 是共同的爸爸。爸爸中的this并不是指的爸爸本人，而是未来某个new出来的儿子。

套上 `myStory` 相关代码解释就是：
`myStory` 是个复制的儿子，老爸中的this指的就是这个儿子，因此当 `myStory.start();` 执行后， `myStory` 多了个值为 `"doing"` 的 `progress` 属性。 老爸还是老爸那个样子。

```
this.progress = "doing"; → myStory.progress = "doing";
```

想想也是这样啊：老爸肯定是不能变的，否则，复制出来的其他儿子岂不都跟着基因突变，残疾弱智之类。何来继承！

三、故事的结果

就跟电影电视剧一样，需要一点波折才好看（韩剧那种揪心来揪心去的就算了）。故事的结果是什么呢？一个小白，冲破了一道门槛，前方之路立马变得平坦。哈哈，喜剧，鼓掌鼓掌  ！

所谓“平坦”指的是?

1. this的解释确实是那样

例如，对`String`原型扩展的一个过滤HTML代码方法：

```
String.prototype.stripHTML = function() {
    return this.replace(/<(?:.|\s)*?>/g, "");
};
```

显然，`this`指的是字符串本身的啦，怎么可能是原型呢！哈哈~~

2. 以前的一些迷糊开朗了

哦，原来jQuery中的继承以及包装器`this`也是这么个套路~~

原来，

```
var myArray = new Array();
```

也是这个套路，原本：

```
Array.prototype = { ... }
```

`Array`原型方法中的`this`实际上就是指的`myArray`啦，与上面的`myStory`一个套路。所以明白了，

```
Array.prototype.slice.call(arguments);
```

是个怎么回事了，`arguments`被`call`为`this`，指代为类似`myArray`的数组被返回，哈哈，明白啦，明白啦！

这就是我“邯郸学步”学出的一点东西。现在的我感觉就像是知道了张妹子其实就是个普通的妹子，不温不火，有时候生活可能比自己还苦逼，且圈子太复杂，很难遇到俺们这些根正苗红、品性纯良的好男人，唉，只能祝福了！

四、结语，略溅口水

巴拉巴拉这么多，一篇技术文章写出了散文的味道，“哈啾”，sorry，有点感冒了~

本文内容其实总结一下很简单。

1. 我是这么学习的……
2. 我最近用了其中一个方法学习……
3. 我用这个方法学习发现了我之前一个错误的认识……
4. 啊哈，我学到了东西。你呢?

我在海边撒贝壳，各种颜色，各种形状。于是，你可能喜欢这个，他可能喜欢那个，或者另外极品不喜欢贝壳。

JS资质有限，如果文中有什么理解错误的，务必指正，误人子弟可是大大的罪过啊，阿门阿门！

感谢阅读，欢迎交流。

我擦，忘记了一个重要的东西。上面所提到的折腾的Hash方法扩展，您有兴趣可以狠狠地点击这里查看：hash.js

有个各个方法的测试demo，您有兴趣可以狠狠地点击这里查看：hash.html
demo打开后，右键页面查看源代码，啦啦啦啦啦~~~

> 《CSS世界》签名版独家发售，包邮，可指定寄语，点击显示购买码

（本篇完） // 想要打赏？点击这里。有话要说？点击这里。

« 疑问：为什么要使用href="javascript:void(0);"?　　表单序列化、规则分离下的复杂表单计算 »

猜你喜欢

- jQuery诞生记-原理与机制
- ECMAScript 5(ES5)中bind方法、自定义及小拓展
- 翻译-不同CSS技术及其CSS性能
- 漫谈js自定义事件、DOM/伪DOM自定义事件
- HTML5 DOM元素类名相关操作API classList简介
- 代码精简 - 常见JavaScript代码缩写举例
- js面向数据编程(DOP)一点分享
- 翻译：ECMAScript 5.1简介
- ES5中新增的Array方法详细说明
- 我是如何理解"Another JavaScript quiz"中的题目

标签： prototype, this上下文, 原型, 对象字面量, 数组, 面向对象

发表评论（目前31条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. 君焰说道：

2016年12月23日 02:39

1.谁调用方法谁就是方法里面的this，否则是全局/
2.构造函数的this则是构造出来的那个方法。但这按第1点来又说不通，这就是我到this一直迷惑的地方

回复

2. 李阳说道：
2015年12月9日 10:39

this在ie下有兼容性吧？？小白一枚，勿喷

回复

3. 灰色毛巾说道：
2015年09月2日 09:14

一楼的解释瞬间感觉清晰了

回复

4. 有贝无患说道：
2015年07月23日 18:29

我认为this指的是函数的执行环境。
console.log(myStory.hasOwnProperty(“progress”));
console.log(myStory.hasOwnProperty(“start”));
为true, false是因为myStory.start();这行代码执行了，start的执行环境变成了myStory对象，而不是prototype。
下面代码就是两个false。

```
var Stroy = function(){};
Stroy.prototype = {
progress:”unkown”,
start:function(){
this.progress = “doing”;
}

};

Stroy.prototype.start();
var myStory = new Stroy();
//myStory.start();

console.log(myStory.progress);
console.log(myStory.hasOwnProperty(“progress”));
console.log(myStory.hasOwnProperty(“start”));
```

回复

悖论说道：

2015年10月9日 21:01

实例对象的执行环境是什么。。执行环境不是只分为全局执行环境和函数执行环境吗

回复

卡夫卡卡卡西说道：
2017年07月13日 10:50

```
var Stroy = function(){};
Stroy.prototype = {
progress:[”unkown”,”konw”],
```

```
add:function(val){
this.progress.push(val)
}

};

var myStory = new Stroy();
myStory.add("doing");
myStory.progress; //["unkown","know"]
myStory.hasOwnProperty("progress"); //false
myStory.__proto__.progress //["unkown","know"]
```

这个怎么解释

回复

5. **wengee**说道：

2013年09月15日 11:15

《套上myStory相关代码解释就是：
myStory是个复制的儿子，老爸中的this指的就是这个儿子，因此当myStory.start();执行后，myStory多了个值为"doing"的progress属性。老爸还是老爸那个样子。
this.progress = "doing"; → myStory.progress = "doing";
想想也是这样啊：老爸肯定是不能变的，否则，复制出来的其他儿子岂不都跟着基因突变，残疾弱智之类。何来继承！》

—xinxu的解释听通俗，风格！！。其实也可以这样解释：
1.实例对象会爬原型链，而函数原型中的定义语句肯定不会爬原型链；
2.即：myStroy.progress会先搜索自身，如无，再搜索原型；this.progress是定义语句，不会爬原型链。因此，当myStory调用start函数，走到this.progress时，直接查找自身，如无，在自身下创建一个。
3.实验验证：在myStory调用start前，console.log(myStory.hasOwnProperty("progress"));
//false；调用start后，console.log(myStory.hasOwnProperty("progress"));
//true

回复

> 卡夫卡卡卡西说道：
> 2017年07月13日 10:55
>
> 《this.progress是定义语句，不会爬原型链。》说的真好，感觉你这句话是关键，解决了我上面提到的困惑。
>
> 回复

6. **mcfog**说道：

2013年06月16日 17:07

this其实是一个隐藏参数，类似命令行里的"第零个参数"

建议看一下Function.prototype.{call, apply}，看一下[underscore.bind](http://underscorejs.org/#bind)

简单地说，call的第一个参数就是this

foo.bar.func(arg) => foo.bar.func.call(foo.bar, arg)
func(arg) => func.call(window, arg)

回复

7. **sprying**说道：

2013年04月19日 03:07

建议楼主去看js高级程序设计的原型章节
感觉楼主的知识体系还存在问题。
闭包可以看http://www.cn-cuckoo.com/2007/08/01/understand-javascript-closures-72.html#clSto

回复

8. **visual说道：**
2013年04月2日 14:28

作为构造函数用的时候this就是指这个对象； 作为方法调用，this貌似就是指这个上级对象

回复

9. **布道说道：**
2013年03月27日 10:33

我和 @marsxiang 理解一致 可以划分几个等级去理解 this 指向谁谁 有new 的时候 里面的this 一定指向new 出来的对象 没有情况 看调用情况 谁调用就指向谁 然就是直接调用 当然就指向window 还有就是定时器中 this 的指向要注意 再者就是函数绑定 时this的指向 另外 call 与apply 存在的情况this就需要重新考虑 call与apply中第一个参数 就是 里面的this 不管你写的是什么 空的话就是null

回复

10. **hezort说道：**
2013年03月25日 19:38

我有过一篇文章说的是同一个道理
http://blog.csdn.net/wangmlabc/article/details/8207811

回复

11. **marsxiang说道：**
2013年02月28日 14:56

随着函数使用场合的不同，this的值会发生变化。但是有一个总的原则，那就是this指的是，调用函数的那个对象。

回复

> **haha说道：**
> 2013年10月2日 18:00
>
> 你这话明显是错误的，判断this的指向，在不通过call等硬性改变的情况下，你只要判断这个函数是谁的属性方法就可以。
> 当你在一个函数里调用alert()其实等于调用window.alert()，所以alert()函数里如果出现this ,就代表window，
> 而当你通过var a=window.alert;的时候，当你调用a()其实相当于调用某个对象的属性方法obj.a()，所以这个时候a()里如果出现this，就代表obj了，
> 所以当你在一个函数里调用某个方法的时候，其实是沿着函数的作用域链在查找此方法属于谁，然后系统把作用域链里的某个对象作为隐形参数传给此方法。
> this，其实没有你们想的哪么复杂。
>
> 回复

12. **bennyrice说道：**
2013年02月20日 00:11

我也这样反复尝试过，一开始以为两个东西是一样的。问了很多人，也说不清楚，最后还是自己一点点测试才发现原来是不同的。。。~~~

回复

13. **hxors说道：**
2013年02月14日 09:15

js中的this都是动态的，随着程序执行的环境而改变的。

回复

14. **children117cl说道：**
2013年02月5日 16:15

“夸克白”受教了

回复

15. **rambo说道：**

2013年02月1日 11:01

还记得以前初学js的时候,老师为了方便,说了一句让我现在听到 挺内涵的一句话 ‘以后在对象中,需要调用自己的属性,不要使用this 尽量使用[对象名]来代替[this]’ 这句话很是内涵啊

回复

16. **jimvin说道：**

2013年01月31日 16:02

在未知調用方式的前提下，斷言prototype中的this指向實例是不正確的。以下代碼以供證明。

```
function ClassA(){}
ClassA.prototype.fn = function(){
this.name = ‘aaa’
}
var obj = {};
var instance = new ClassA();
instance.fn.call(obj);
alert(instance.name); // output undefined ; this 指向 obj（非實例instance）
alert(obj.name); // output aaa
ClassA.prototype.fn();
alert(name); // output aaa ; this 指向 window
var instance2 = new ClassA();
instance2.fn();
alert(instance.name); // output aaa ; this 指向 實例
```

回复

17. **Michael J.说道：**

2013年01月31日 15:47

学习了~今天还正好看了1.9的jQuery源码，
this的这个执行环境会变动，直接形成了两个编程上的技巧：

一个是Nicolas C. Zakas在Pro Javascript for web developer中提到的这种

```
var name = “The Window”;
var object = {
name : “My Object”,
getNameFunc : function(){
var that = this;
return function(){
return that.name;
};
}
};
```

用that保存this的方法。

还有一种方法就是jQuery.proxy啦。

回复

18. **vgche说道：**

2013年01月31日 10:10

以前在《javascript语言精粹》看过对this的详细讲解，在【第4章】中【4.3调用】一节，对函数的4种调用模式都有详尽说明，包括this的指向问题。
博主的技术文章还是这么欢乐哈^_^

回复

19. **nerd说道：**
2013年01月30日 23:52

1楼说得好

回复

20. **短笛说道：**
2013年01月30日 15:00

“prototype这东西有点像屌丝过滤器，不解释。”

这句话激励我一定把这篇文章弄懂！

回复

21. **Tai说道：**
2013年01月30日 11:18

博主的比喻实在是太强大了，本来明白的被绕晕了！一楼的解释很到位

回复

22. **小克说道：**
2013年01月30日 09:57

this 在Js里面是一个很神奇的存在，前端面试见过各种对this的考察。

学名是叫执行环境上下文？记不太清了……

不知道博主看到你书单上的那本猫头鹰没有，里面前两章讲继承啊this啊很有意思~

回复

23. **aaa说道：**
2013年01月30日 00:52

```
var story = {
progress: “unknown”,
start: function() {
this.progress = “start”;
}
};

var a = story.start;
a();
```

这样调用这方法的对象是window,因此window就有了progress属性被赋值为”start”了,一般修正this都是用个变量保存下来,于是这样玩

```
var story = {
progress: “unknown”,
start: (function() {
var that = this;
return function(){
that.progress = “start”;
}
})
};

var a = story.start();
a();
console.log(window.progress);
console.log(story.progress);
```

我记得尼古拉斯那本还是good parts 讲了这问题的,玩得还是不太好,见笑了

回复

张 鑫旭说道：
2013年01月30日 09:36

@aaa 嗯哪，学习了！

回复

bbb说道：
2013年09月20日 02:48

这不是闭包的概念吗?
var story = {
progress: “unknown”,
start: (function() {
var that = this;
return function(){
that.progress = “start”;
}
})
};

var a = story.start();
a();
当我们运行 alert(a);的时候，这时候，弹出的信息是
function (){
that.progress = “start”;
}
相当于
function a(){
story.progress = “start”;
}
因为在 story中我们定义了 var that = this;而this 就是 story对象，
所以
console.log(window.progress); //undefined
console.log(story.progress); //unknown
console.log(a.progress); //undefined

回复

24. **devan5说道：**
2013年01月29日 18:56

博主是想说明 由 prototype 所以误解了在 prototype 中的 this 指向的是实例而不是 prototype本身,继而发现了 实例(原型)属性读写”不平等”性

回复

张 鑫旭说道：
2013年01月29日 19:30

@devan5 恩？……恩~~没看到你评论的意思，貌似是吧~

回复

最新文章

» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

» 常见的CSS图形绘制合集 (193)

» 未来必热：SVG Sprite技术介绍 (120)
» 粉丝群第1期CSS小测点评与答疑 (115)
» HTML5终极备忘大全（图片版+文字版） (93)
» 让所有浏览器支持HTML5 video视频标签 (86)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (82)
» CSS3下的147个颜色名称及对应颜色值 (80)
» 视区相关单位vw, vh..简介以及可实际应用场景 (76)
» 写给自己看的display: flex布局教程 (76)
» 小tips: 纯CSS实现打字动画效果 (76)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- jQuery诞生记-原理与机制
- ECMAScript 5(ES5)中bind方法、自定义及小拓展
- 翻译-不同CSS技术及其CSS性能
- 漫谈js自定义事件、DOM/伪DOM自定义事件
- HTML5 DOM元素类名相关操作API classList简介
- 代码精简 - 常见JavaScript代码缩写举例
- js面向数据编程(DOP)一点分享
- 翻译：ECMAScript 5.1简介
- ES5中新增的Array方法详细说明
- 我是如何理解"Another JavaScript quiz"中的题目