[**通过backbone的extend函数又想了一下JS的继承**](http://www.cnblogs.com/sc-xx/archive/2012/07/27/2611034.html)

var Tempt = Backbone.Model/.../.extend({.A...}{..B..}),

下面多处会应用上面的代码

使用backbone的时候，不免的是各种extend，不明真相永远是痛苦的...

痛苦是需要解决的，所以就extend得想了一些backbone的extend.

在backbone源码的第1303行出现了传说中的extend

Model.extend = Collection.extend = Router.extend = View.extend = extend;

根据高中和初中语文老师说的一样，解药往往在毒药的旁边，就想射雕里面过儿寻找绝情草...

对就在backbone的1298行，有下面的代码

var extend = function(protoProps, classProps) {

return inherits(this, protoProps, classProps);

};

学东西，我们就应该有顺藤摸瓜的精神，现在遇到的是关于inherits的理解，也就是今晚重点要理解的一个知识点，正好买的那本一块钱一页的书，也刚好看到这里，结合着书上的一起

理解。

废话不多说了，先看看backbone里面是怎样来实现inherits的

[复制代码](javascript:void(0);)

1 var inherits = function(parent, protoProps, staticProps) {

2 var child;

3

4 // The constructor function for the new subclass is either defined by you

5 // (the "constructor" property in your `extend` definition), or defaulted

6 // by us to simply call the parent's constructor.

7 if (protoProps && protoProps.hasOwnProperty('constructor')) {

8 child = protoProps.constructor;

9 } else {

10 child = function(){ parent.apply(this, arguments); };

11 }

12

13 // Inherit class (static) properties from parent.

14 \_.extend(child, parent);

15

16 // Set the prototype chain to inherit from `parent`, without calling

17 // `parent`'s constructor function.

18 ctor.prototype = parent.prototype;

19 child.prototype = new ctor();

20

21 // Add prototype properties (instance properties) to the subclass,

22 // if supplied.

23 if (protoProps) \_.extend(child.prototype, protoProps);

24

25 // Add static properties to the constructor function, if supplied.

26 if (staticProps) \_.extend(child, staticProps);

27

28 // Correctly set child's `prototype.constructor`.

29 child.prototype.constructor = child;

30

31 // Set a convenience property in case the parent's prototype is needed later.

32 child.\_\_super\_\_ = parent.prototype;

33

34 return child;

35 };

[复制代码](javascript:void(0);)

经典的东西一般都不多，加上注释一个35行，看看它的实现吧！

首先看它的参数部分

var inherits = function(parent, protoProps, staticProps)

其中有三个参赛分别是parent和protoProps，和staticProps，从字面上来理解，这三个参书分别代表父类，实例属性，静态属性。

要理解这个函数，就必须知道三个形参对应的实参是什么，所以首先从调用它的地方看起

var extend = function(protoProps, classProps) {

return inherits(this, protoProps, classProps);

};

// Set up inheritance for the model, collection, and view.

Model.extend = Collection.extend = Router.extend = View.extend = extend;

我们一般调用extend的时候都是通过var Tempt = Backbone.Model/.../.extend({.A...}{..B..}),所以实际上是在执行 Model.extend = function (protoProps, classProps){},然后在这个extend method中又调用并返回了 inherits(this, protoProps, classProps),根据JS精粹中关于函数5种invocation来确定this的方法，这里extend是作为Model等的method来调用的，所以这时的this应该是Model等,所以inherits形参对应的实参应该是,parent 对应的是Model/Collection/Router/View,protoProps对应的是extend({...A...}{...B...})中的A,classProps对应的是

B。带着这样的参数，我们继续理解inherits函数。

接着是代码的正文部分

[复制代码](javascript:void(0);)

// The constructor function for the new subclass is either defined by you

// (the "constructor" property in your `extend` definition), or defaulted

// by us to simply call the parent's constructor.

if (protoProps && protoProps.hasOwnProperty('constructor')) {

child = protoProps.constructor;

} else {

child = function(){ parent.apply(this, arguments); };

}

[复制代码](javascript:void(0);)

通过英文的注释我们不难看出这个代码片段完成的功能是给子类提供constructor属性，在这里如果在extend({...A...}{...B...})中的A提供了一个constructor字段的话，这时子类的

constructor属性将会是这个这个属性对应的函数，如果没有那么子类的constructor将会和父类的一样，在这里需要复习的一个概念是child = function () {}其实是要在new的时候才会

调用的，所以我们先假设我们设置的字段里面没有constructor，所以当我们通过var temptA = new Tempt()的时候，调用的是function () {Model.apply(temptA, arguments)},

测试见<http://jsfiddle.net/xiaoxiong_1990/tErsT/> 当我们console出来的时候temptA已经拥有了一些属性的，而这些属性也正是

[复制代码](javascript:void(0);)

var Model = Backbone.Model = function(attributes, options) {

var defaults;

attributes || (attributes = {});

if (options && options.collection) this.collection = options.collection;

if (options && options.parse) attributes = this.parse(attributes);

if (defaults = getValue(this, 'defaults')) {

attributes = \_.extend({}, defaults, attributes);

}

this.attributes = {};

this.\_escapedAttributes = {};

this.cid = \_.uniqueId('c');

this.changed = {};

this.\_silent = {};

this.\_pending = {};

this.set(attributes, {silent: true});

// Reset change tracking.

this.changed = {};

this.\_silent = {};

this.\_pending = {};

this.\_previousAttributes = \_.clone(this.attributes);

this.initialize.apply(this, arguments);

};

[复制代码](javascript:void(0);)

所定义的。

在这里个人有了第一个理论：

（1）当子类没有提供特定的constructor的时候，应该调用父类的。

接下来接着理解inherts

// Inherit class (static) properties from parent.

\_.extend(child, parent);

上面虽然说只有一句代码，但是包含的功能却不少，通过函数的调用我们可以轻易的看出，这里实际上是调用了underscore中的extend的方法，又是extend

下面是underscore从第690行开始的

[复制代码](javascript:void(0);)

// Extend a given object with all the properties in passed-in object(s).

\_.extend = function(obj) {

each(slice.call(arguments, 1), function(source) {

for (var prop in source) {

obj[prop] = source[prop];

}

});

return obj;

};

[复制代码](javascript:void(0);)

然后又的extend自己了，遇到的第一个问题就是关于each方法的理解，给一个测试例子<http://jsfiddle.net/a6Rx4/> (这个例子为引用别人的)

[复制代码](javascript:void(0);)

var someOtherArray = ["name","patrick","d","w"];

\_.each([1, 2, 3], function(num) {

// In here, "this" refers to the same Array as "someOtherArray"

alert( this[num] ); // num is the value from the array being iterated

// so this[num] gets the item at the "num" index of

// someOtherArray.

}, someOtherArray);

[复制代码](javascript:void(0);)

这样通过上面两个代码片段，我们就不难理解\_.extend(child, parent)究竟做了什么事，我们还是把实参带入函数来说吧

在\_.extend = function (obj){...}中，obj 就是child,然后通过slice.call(arguments, 1),这样来获得child后面的函数，在这里就是parent，作为function (source) {...}中的source

然后通过for...in遍历parent中的property并将其复制给child，这样就完成了从parent复制静态属性到child的过程。

接着是继承prototype的属性，在backbone中采用了DG提出来的继承方式(<http://javascript.crockford.com/prototypal.html>)，即通过proxy来实现继承，这样就避免了在classical继承中出现的parent的constructor被使用两次带来

的效率问题和在原型链中再次继承this的属性。

// Set the prototype chain to inherit from `parent`, without calling

// `parent`'s constructor function.

ctor.prototype = parent.prototype;

child.prototype = new ctor();

在上面的例子中ctor其实是一个空的函数，它充当了parent和child的proxy.ctor的prototype属性指向了parent的prototype属性。child的prototype属性是ctor这个空函数的一个实例。

通过这种方式就仅仅继承了parent的prototype属性。

下面的代码就分别完成了给新的类指定实例属性和静态属性的功能。

[复制代码](javascript:void(0);)

// Add prototype properties (instance properties) to the subclass,

// if supplied.

if (protoProps) \_.extend(child.prototype, protoProps);

// Add static properties to the constructor function, if supplied.

if (staticProps) \_.extend(child, staticProps);

// Correctly set child's `prototype.constructor`.

child.prototype.constructor = child;

// Set a convenience property in case the parent's prototype is needed later.

child.\_\_super\_\_ = parent.prototype;

[复制代码](javascript:void(0);)

其实上面的继承也是JS的一种比较典型的继承方式.....