Backbone源码解析

01 框架结构

02 Event

03 extend

04 Route与History

03 view model Collection的关联

06 sync

07 template

框架结构

```
(function( root, factory){
        //环境配置
        //针对AMD以及Node.js和common.js
}( this, function( root, Backbone,_,$){
        var Events  =   Backbone. Events;
        var Model =    Backbone.Model;
        var  Collection =   Backbone.Collection;
        var View =  Backbone.View;
        var Route =  Backbone.Route;
        var History =  Backbone.History
        Backbone.sync;
}))
```

Event—事件的定义

所谓事件即在某处定义一个方法体并且给予他一个名字，在需要时执行此该方法

Event—事件池

```
_events:{
        eventName1:[callback1,callback2]
        ,eventName2:[callback1,callback2]
        ,eventName3:[callback1,callback2]
}
```

Event—事件绑定的几种形式

1.多个事件绑定一个回调函数

```
this.model.on('change:titlechange:name',function(){
                    alert(1)
});
```

Event—事件绑定的几种形式

2.一次绑定多个事件，每个事件的回调不一致

```
this.model.on({
                "change:title":function(){alert(1)}
                ,"change:name":function(){alert(2)}
        })
```

```
this.model.on({
                "change:title":function(){alert(1)}
                ,"change:title":function(){alert(2)}
    })(错误写法)
```

Event—事件绑定的几种形式

正确写法：

```
this.model.on('change:title',function(){alert(1)});
this.model.on('change:title',function(){alert(2)});
```

不过很好有人需要这种形式吧，大都是将需要做的事情都写在一个回调里吧。 我好奇，就试了下

Event—事件绑定的几种形式

3.一次绑定一个

```
this.model.on('change:title',function(){});
```

extend

1、继承原则

需要继承的属性分别为：原型属性、静态属性、实例属性（其中静态属性是容易被忽略的）

extend

```
// by us to simply call the parent's constructor.
if (protoProps && _.has(protoProps, 'constructor')) {
    child = protoProps.constructor;
} else {
  child = function(){ return parent.apply(this, arguments); };
}

// Add static properties to the constructor function, if supplied.
_.extend(child, parent, staticProps);
// Set the prototype chain to inherit from `parent`, without calling
// `parent`'s constructor function.
var Surrogate = function(){ this.constructor = child; };
Surrogate.prototype = parent.prototype;
child.prototype = new Surrogate;

// Add prototype properties (instance properties) to the subclass,
// if supplied.
if (protoProps) _.extend(child.prototype, protoProps);
```

view model Collection

在model collection view 的构造函数中，除去各自不同的处理之外，都将event 以及各自的方法添加在prototype上

```
// Attach all inheritable methods to the Model prototype.
_.extend(Model.prototype, Events, {

  // A hash of attributes whose current and previous value differ
  changed: null,
```

Model

```
var Person = Backbone.Model.extend({
       defaults:{‘name’:'lily',
                      'age':22
                    }
      ,getName:function(){
             console.log(this.get('name'));
       }
 },{'staticAttr':true})
var person  = new Person({'name':'xgh'});

Person.staticAttr  //true
```

Model

1.设置cid
2.处理options
3.设置attributes 、changed
4.执行initialize

```
var Model = Backbone.Model = function(attributes, options) {
  var attrs = attributes || {};
  options || (options = {});
  this.cid = _.uniqueId('c');
  this.attributes = {};
  if (options.collection) this.collection = options.collection;
  if (options.parse) attrs = this.parse(attrs, options) || {};
  attrs = _.defaults({}, attrs, _.result(this, 'defaults'));
  this.set(attrs, options);
  this.changed = {};
  this.initialize.apply(this, arguments);
};
```

collection

```
var Person = Backbone.Model.extend({});
 var personModel = new Person();
var Persons = Backbone.Collection.extend({
                                model:Person
                    });
var person = new Persons([{'name':'a1'},{'name':'a2'}]);


//var person = new Persons([{'name':'a1'},{'name':'a2'}],{"model":Person});
```

collection

```
var Collection = Backbone.Collection = function(models, options) {
  options || (options = {});
  if (options.model) this.model = options.model;
  if (options.comparator !== void 0) this.comparator = options.comparator;
  this._reset();
  this.initialize.apply(this, arguments);
  if (models) this.reset(models, _.extend({silent: true}, options));
};

// Default options for `Collection#set`.
var setOptions = {add: true, remove: true, merge: true};
var addOptions = {add: true, remove: false};

// Define the Collection's inheritable methods.
_.extend(Collection.prototype, Events, {
```

1.设置cid
2.处理options
3.处理models
4.执行initialize

view

```
var View = Backbone.View.extend({
                    el : '#view'
                    ,events : {
                        'click #create' : 'createData',
                        'click #add' : 'changeFn',
                    }
                    ,createData : function() {
                      alert(1);
                    }
                    ,changeFn:function(){
                            alert(2);
                    }
        });
var view = new View();
 //view.setElement($('#view1'));
 //false  不会为新的$el绑定事件
 //true   会为新的$el绑定事件
```

view

1.设置cid
2.将options中的一些属性绑定在this上
3.确定el
4.绑定事件

```
var View = Backbone.View = function(options) {
  this.cid = _.uniqueId('view');
  options || (options = {});
  _.extend(this, _.pick(options, viewOptions));
  this._ensureElement();
  this.initialize.apply(this, arguments);
  this.delegateEvents();
};
```

view

3.delegateEvents undelegateEvents

```
delegateEvents: function(events) {
  if (!(events || (events = _.result(this, 'events')))) return this;
  this.undelegateEvents();
  for (var key in events) {
    var method = events[key];
    if (!_.isFunction(method)) method = this[events[key]];
    if (!method) continue;
    var match = key.match(delegateEventSplitter);
    var eventName = match[1], selector = match[2];
    method = _.bind(method, this);
    eventName += '.delegateEvents' + this.cid;
    if (selector === '') {
      this.$el.on(eventName, method);
    } else {
      this.$el.on(eventName, selector, method);
    }
  }
  return this;
},
undelegateEvents: function() {
  this.$el.off('.delegateEvents' + this.cid);
  return this;
},
```

view

3. setElement 修改el

```
setElement: function(element, delegate) {
  if (this.$el) this.undelegateEvents();
  this.$el = element instanceof Backbone.$ ? element : Backbone.$(element);
  this.el = this.$el[0];
  if (delegate !== false) this.delegateEvents();
  return this;
},
```

view model Collection的关联

1.Model和Collection的关联

1>、在Collection中有一个Model的属性，用来标志这个collection中的model是那种Model类型，如果不设置model则会默认为Backbone的Model

2>、 在我们

var test = new Backbone.Collection([‘name’:’qq’])

时，在Collection中判断传入的参数是否为Backbone.Model的实例，如果不是则转化

view model Collection的关联

2.Model、Collection与View的关联

_.extend(this, _.pick(options, viewOptions));

将model、collection作为view的属性存在

注：在new View时，传入的model，collection必须为标准的Backbone.Model、Backbone.Collection

view model Collection的关联

3、常写的View中的events和Backbone.Events是不同的。View中的Events是定义$el这段Html片段上的dom事件，通过事件代理的形式绑定在$el上。

Route与History

1、路由的原理

Backbone的路由 由Route于History配合完成

其中，Route用于定义路由规则，将url与Action联系起来（其实质也是事件的原理）

History用于监听url的变化，并且触发url对应的Action

Route与History

2、Route与History的联系

当new一个Route的时候，在Route内部会调用History的route方法。将定义好的路由规则一并传入

在History中监听hash的变化($(window.onhashchange()))，来执行不同的Action

Route与History

在Route中调用History的route方法

```
    Backbone.history.route(route, function(fragment) {
      var args = router._extractParameters(route, fragment);
      router.execute(callback, args);
      router.trigger.apply(router, ['route:' + name].concat(args));
      router.trigger('route', name, args);
      Backbone.history.trigger('route', router, name, args);
    });
    return this;
  },
```

Route与History

History中的Route,将route定义的路由规则存在在数据中

```
// Add a route to be tested when the fragment changes. Routes added
// may override previous routes.
route: function(route, callback) {
  this.handlers.unshift({route: route, callback: callback});
},
```

Route与History

当hash改变的时候执行对应的回调

```
// returns `false`.
loadUrl: function(fragment) {
  fragment = this.fragment = this.getFragment(fragment);
  return _.any(this.handlers, function(handler) {
    if (handler.route.test(fragment)) {
      handler.callback(fragment);
      return true;
    }
  });
},
```

Route与History

```
var AppRouter = Backbone.Router.extend({
    routes : {
        '' : 'main',
        'topic' : 'renderList',
        'topic/:id' : 'renderDetail',
        '*error' : 'renderError'
    },
    main : function() {
        console.log('应用入口方法');
    },
    renderList : function() {
        console.log('渲染列表方法');
    },
    renderDetail : function(id) {
        console.log('渲染详情方法, id为: ' + id);
    },
    renderError : function(error) {
        console.log('URL错误, 错误信息: ' + error);
    }
});
var router = new AppRouter();
Backbone.history.start();
```

Route与History

3、TalentJs路由

在TalentJs中通过hash来加载对应文件下的index-page-view

Route与History

```
loadPageView: function() {
    // redirect to page of entryPageId if history fragment is empty
    if(!Backbone.history.getFragment()){
        return Backbone.history.navigate(this.options.entryPageId, true)
    }
    var self = this;
    var fullPageViewPath = this.getPageViewPath();
    var rootPageViewPath = this.getPageViewPath(true);

    require([rootPageViewPath], function(RootPageView) {

        require([fullPageViewPath], function(PageView) {
            var pageView = new PageView({
                    "queryObject": self.getQueryObject(),
                    "fragments": self.getFragments()
                });

            /**
             * 如果PageView有相同的Layout，则局部刷新
             */
```

sync

在Backbone中提供了sync对象，用于和后端交互

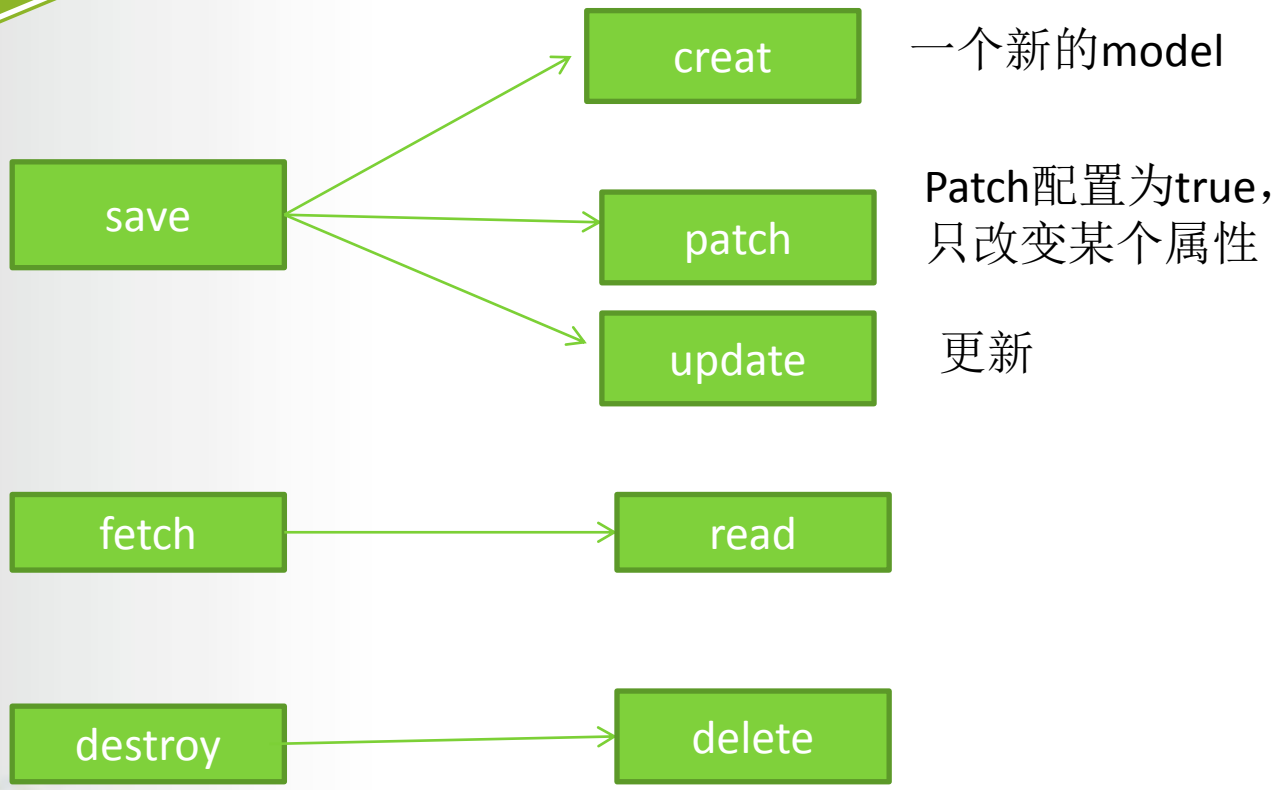
其背后利用Jquery的Ajax实现

在sync中在大量的组织Ajax需要的参数，例如：
URL、data、type、contentType　而返回一个xhr对象

sync

```
// Map from CRUD to HTTP for our default `Back
var methodMap = {
  'create': 'POST',
  'update': 'PUT',
  'patch':  'PATCH',
  'delete': 'DELETE',
  'read':   'GET'
};
```

sync



一个新的model

Patch配置为true，
只改变某个属性

更新

sync

1.emulateHTTP

如果后端接口不支持Backbone默认的REST架 构，可以将其设置为true 而将 PUT DELETE PATCH转化为POST

2.emulateJSON

用来配置contentType

默认为 application/json

设置为true则为：'application/x-www-form-urlencoded';

template

既然是模板，肯定最后生成的一段html，将model或者传入的值计算之后写生成html

在TalentJs中Marionette帮我们调用了template;如果不借助Marionette我们需要自己去显示的使用template计算模板

template

```
var compiled = _.template("<p>hello world</p>");
compiled();
"<p>hello world</p>"
```

```
var compiled = _.template("<p>hello: <%= text%></p>");
compiled({text: 'xgh'});
"<p>hello: xgh</p>"
```

template

```
if (!settings.variable) source = 'with(obj||{}){\n' + source + '}\n';

source = "var __t,__p='',__j=Array.prototype.join," +
  "print=function(){__p+=__j.call(arguments,'');};\n" +
  source + 'return __p;\n';

try {
  var render = new Function(settings.variable || 'obj', '_', source);
} catch (e) {
  e.source = source;
  throw e;
}

var template = function(data) {
  return render.call(this, data, _);
};


var argument = settings.variable || 'obj';
template.source = 'function(' + argument + '){\n' + source + '}';

return template;
```

template

```
dragq">\r\n\t<p class="lead">\r\n\t\t<strong>' +((__t = ( head )) == nu

ntent star_score"> ' +((__t = ( text )) == null ? '' : __t) +' </span>'


){ ;__p += '\r\n\t\t<span class=\'txt\'>' +((__t = ( text )) == null ?

ss=\'common_main select_part\'>\r\n\t\t<div class=\'common_tit\'>' +((

ader-region\'>\r\n\t\t<div class=\'quiz_head \'>\r\n\t\t\t<div class=\'
```