中文帮助文档

http://www.csser.com/tools/backbone/backbone.js.html

[backbone.js 初探](http://weakfi.iteye.com/blog/1391990)

**什么是backbone**

backbone不是脊椎骨，而是帮助开发重量级的javascript应用的框架。

主要提供了3个东西：1、models(模型) 2、collections(集合) 3、views(视图)

backbone.js文件本身很小，压缩后只有5.3KB，作为一个框架级别的核心JS文件，这个数字很可怕。

除此之外，这个JS还必须依赖于另一个JS文件：underscore.js(包含许多工具方法，集合操作，js模板等等)。

**简介**

用Backbone.Model表示应用中所有数据，models中的数据可以创建、校验、销毁和保存到服务端。

当models中值被改变时自动触发一个"change"事件、所有用于展示models数据的views都会侦听到这个事件，然后进行重新渲染。

Backbone.Collection和我们平时接触的JAVA集合类相似，具有增加元素，删除元素，获取长度，排序，比较等一系列工具方法，说白了就是一个保存models的集合类。

Backbone.View中可以绑定dom el和客户端事件。页面中的html就是通过views的render方法渲染出来的，当新建一个view的时候通过要传进一个model作为数据，例如：

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. var view = new EmployeeView({model:employee});

var view = new EmployeeView({model:employee});

 也就是说model就是以这种方式和view进行关联的。

**特点**

创建models或者views的语法：extends，相当于类继承

models的创建，销毁，校验等一系列改变都会触发相应的事件

**示例**

**需求：**用backbone.js和jquery实现一个可编辑的员工信息表格。

**功能：**1、录入员工信息。2、删除员工信息。3、双击表格可对员工信息进行修改。4、能对员工信息进行有效性校验。5、能对员工信息进行持久化。

**设计：**

用Employee类（继承自Backbone.Model）表示员工信息，包含ID、姓名、性别、年龄和职位字段。

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. window.Employee = Backbone.Model.extend({
2. // 模型值校验
3. validate:function(attrs){
4. for(var key in attrs){
5. if(attrs[key] == ''){
6. return key + "不能为空";
7. }
8. if(key == 'age' && isNaN(attrs.age)){
9. return "年龄必须是数字";
10. }
11. }
12. }
13. });

window.Employee = Backbone.Model.extend({

// 模型值校验

validate:function(attrs){

for(var key in attrs){

if(attrs[key] == ''){

return key + "不能为空";

}

if(key == 'age' && isNaN(attrs.age)){

return "年龄必须是数字";

}

}

}

});

声明Employee类之后就可以新增一个Employee的示例对象了：

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. var employee = new Employee();

var employee = new Employee();

Employee类中不必声明ID、姓名等业务字段。当需要给employee设置这些信息时候，只需要调用

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. employee.set({'id':1,'name':'Jason'});

employee.set({'id':1,'name':'Jason'});

当然，如果需要对employee的信息进行校验，需要给Employee类配置一个validate方法，这个方法的参数attrs就是set进去的json数据。这样，当employee里面的数据每次发生改变的时候都会先调用这个validate方法。

Model类定义好之后就可以开始定义集合类了，在集合类里面可以对里面的每个Model进行增加，删除等一系列操作，还可以调用fetch方法从server端获取集合的初始值。

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. window.EmployeeList = Backbone.Collection.extend({
2. model : Employee,
3. // 持久化到本地数据库
4. localStorage: new Store("employees"),
6. });
7. window.Employees = new EmployeeList();

window.EmployeeList = Backbone.Collection.extend({

model : Employee,

// 持久化到本地数据库

localStorage: new Store("employees"),

});

window.Employees = new EmployeeList();

设置 localStorage属性后Employees里面的数据自动会同步保存到本地数据库里面，每当调用Employees.fetch()后又会从localStorage里面恢复数据。

View类主要负责一切和界面相关的工作，比如绑定html模板，绑定界面元素的事件，初始的渲染，模型值改变后的重新渲染和界面元素的销毁等：

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. window.EmployeeView = Backbone.View.extend({
2. tagName : 'tr',
3. template : \_.template($('#item-template').html()),
4. events : {
5. "dblclick td" : "edit",
6. "blur input,select" : "close",
7. "click .del" : "clear",
8. },
9. initialize : function(){
10. // 每次更新模型后重新渲染
11. this.model.bind('change', this.render, this);
12. // 每次删除模型之后自动移除UI
13. this.model.bind('destroy', this.remove, this);
14. },
15. setText : function(){
16. var model = this.model;
17. this.input = $(this.el).find('input,select');
18. this.input.each(function(){
19. var input = $(this);
20. input.val(model.get(input.attr("name")));
21. });
22. },
23. close: function(e) {
24. var input = $(e.currentTarget);
25. var obj = {};
26. obj[input.attr('name')] = input.val();
27. this.model.save(obj);
28. $(e.currentTarget).parent().parent().removeClass("editing");
29. },
30. edit : function(e){
31. // 给td加上editing样式
32. $(e.currentTarget).addClass('editing').find('input,select').focus();
33. },
34. render: function() {
35. $(this.el).html(this.template(this.model.toJSON()));
36. // 把每个单元格的值赋予隐藏的输入框
37. this.setText();
38. return this;
39. },
40. remove: function() {
41. $(this.el).remove();
42. },
43. clear: function() {
44. this.model.destroy();
45. }
46. });

window.EmployeeView = Backbone.View.extend({

tagName : 'tr',

template : \_.template($('#item-template').html()),

events : {

"dblclick td" : "edit",

"blur input,select" : "close",

"click .del" : "clear",

},

initialize : function(){

// 每次更新模型后重新渲染

this.model.bind('change', this.render, this);

// 每次删除模型之后自动移除UI

this.model.bind('destroy', this.remove, this);

},

setText : function(){

var model = this.model;

this.input = $(this.el).find('input,select');

this.input.each(function(){

var input = $(this);

input.val(model.get(input.attr("name")));

});

},

close: function(e) {

var input = $(e.currentTarget);

var obj = {};

obj[input.attr('name')] = input.val();

this.model.save(obj);

$(e.currentTarget).parent().parent().removeClass("editing");

},

edit : function(e){

// 给td加上editing样式

$(e.currentTarget).addClass('editing').find('input,select').focus();

},

render: function() {

$(this.el).html(this.template(this.model.toJSON()));

// 把每个单元格的值赋予隐藏的输入框

this.setText();

return this;

},

remove: function() {

$(this.el).remove();

},

clear: function() {

this.model.destroy();

}

});

 这个类里面的代码比较多，但主要和界面的渲染有关。一个EmployeeView对象对应table里面的一个tr元素。每次new一个EmployeeView对象的时候都会先调用initialize方法，这个方法里面绑定的事件确保了tr元素对应的model值每次发生改变或者被删除时都会同步到界面。也就是说当每次操作界面对数据进行修改后都是先把当前的变更保存到view绑定的model对象里面，然后model里面的事件机制会自动触发一个"change"事件对界面进行修改。

template中使用的方法\_.template($('#item-template').html())是前面提到的underscore.js中提供一个工具方法，可以通过界面的HTML模板和一个JSON生成动态的HTML，说白了就是把JSON里面的值填充到HTML模板中对应的占位符里面去，牛X的是HTML模板里面支持一些常用的逻辑表达式如if,else,foreach等：

Html代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<script** type="text/template" id="item-template"**>**
2. **<td><**%= eid %**></td>**
3. **<td** class="username"**>**
4. **<div** class="display"**><**%= username %**></div>**
5. **<div** class="edit"**><input** class="username" name="username"**></input></div>**
6. **</td>**
7. **<td** class="sex"**>**
8. **<div** class="display"**><**%= sex=="1" ? "女":"男" %**></div>**
9. **<div** class="edit"**>**
10. **<select** name="sex" class="sex" style="width:45px"**>**
11. **<option** value="0"**>**男**</option><option** value="1"**>**女**</option>**
12. **</select>**
13. **</div>**
14. **</td>**
15. **<td** class="age"**>**
16. **<div** class="display"**><**%= age %**></div>**
17. **<div** class="edit"**>**
18. **<input** class="age" name="age"**></input>**
19. **</div>**
20. **</td>**
21. **<td** class="position"**>**
22. **<div** class="display"**><**%= position %**></div>**
23. **<div** class="edit"**>**
24. **<input** class="position" name="position"**></input>**
25. **</div>**
26. **</td>**
27. **<td>**
28. **<a** href="#" class="del"**>**删除**</a>**
29. **</td>**
30. **</script>**

<script type="text/template" id="item-template">

<td><%= eid %></td>

<td class="username">

<div class="display"><%= username %></div>

<div class="edit"><input class="username" name="username"></input></div>

</td>

<td class="sex">

<div class="display"><%= sex=="1" ? "女":"男" %></div>

<div class="edit">

<select name="sex" class="sex" style="width:45px">

<option value="0">男</option><option value="1">女</option>

</select>

</div>

</td>

<td class="age">

<div class="display"><%= age %></div>

<div class="edit">

<input class="age" name="age"></input>

</div>

</td>

<td class="position">

<div class="display"><%= position %></div>

<div class="edit">

<input class="position" name="position"></input>

</div>

</td>

<td>

<a href="#" class="del">删除</a>

</td>

</script>

setText方法主要负责把model里面的数据设置到每个tr里面的隐藏输入域里面。

close方法被绑定到了input和select元素的blur事件中。当用户对单元格数据进行修改后都会把鼠标点击到界面其他地方然后输入框会自动隐藏并且把修改的数据显示在表格上面。close方法首先从当前被编辑的元素中拿到最新值，然后封装成一个对象，调用model的save方法后首先执行model的validate方法，如果校验通过则保存到本地存储并触发"change"事件。

最后还需要一个主界面View，这个View主要绑定了界面中的录入表单的“增加”按钮事件，Employees的相关事件以及页面初始化时从本地存储中恢复数据：

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. window.AppView = Backbone.View.extend({
2. el : $("#app"),
3. events : {
4. "click .#add-btn" : "createOnEnter"
5. },
6. // 绑定collection的相关事件
7. initialize: function() {
8. Employees.bind('add', this.addOne, this);
9. // 调用fetch的时候触发reset
10. Employees.bind('reset', this.addAll, this);
11. Employees.fetch();
12. },
13. createOnEnter : function(e) {
14. var employee = new Employee();
15. var attr = {};
16. $('#emp-form input,#emp-form select').each(function(){
17. var input = $(this);
18. attr[input.attr('name')] = input.val();
19. });
20. employee.bind('error',function(model,error){
21. alert(error);
22. });
23. // set方法中会自动调用model的validate方法进行校验，如果不通过则返回false
24. if(employee.set(attr)){
25. Employees.create(employee);
26. }
27. },
28. addOne : function(employee){
29. employee.set({"eid":employee.get("eid")||Employees.length});
30. employee.bind('error',function(model,error){
31. alert(error);
32. });
33. var view = new EmployeeView({model:employee});
34. $(".emp-table tbody").append(view.render().el);
35. },
36. addAll : function(){
37. Employees.each(this.addOne);
38. }
39. });

window.AppView = Backbone.View.extend({

el : $("#app"),

events : {

"click .#add-btn" : "createOnEnter"

},

// 绑定collection的相关事件

initialize: function() {

Employees.bind('add', this.addOne, this);

// 调用fetch的时候触发reset

Employees.bind('reset', this.addAll, this);

Employees.fetch();

},

createOnEnter : function(e) {

var employee = new Employee();

var attr = {};

$('#emp-form input,#emp-form select').each(function(){

var input = $(this);

attr[input.attr('name')] = input.val();

});

employee.bind('error',function(model,error){

alert(error);

});

// set方法中会自动调用model的validate方法进行校验，如果不通过则返回false

if(employee.set(attr)){

Employees.create(employee);

}

},

addOne : function(employee){

employee.set({"eid":employee.get("eid")||Employees.length});

employee.bind('error',function(model,error){

alert(error);

});

var view = new EmployeeView({model:employee});

$(".emp-table tbody").append(view.render().el);

},

addAll : function(){

Employees.each(this.addOne);

}

});

 initialize方法中绑定了Employees的add和reset事件，也就是说每当往Employees中添加一个model的时候都会调用AppView的addOne方法，这个方法主要绑定了model的error事件以及把EmployeeView生成的html插入到界面中的合适位置。

OK，万事俱备，只欠启动，整个应用的初始化方法就是AppView的initialize方法，因此只需要新建一个AppView就可以了：

Js代码 [复制代码](http://weakfi.iteye.com/blog/1391990) [[收藏代码http://weakfi.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. window.App = new AppView();

window.App = new AppView();

整个示例的JS代码很少，加上注释只有100行左右，感兴趣的可以下载看看。由于示例使用到了本地存储，所以不要用IE运行示例，你懂的

# 入门教程

## 第一章

作者: **阿伦** 日期: 2012 年 3 月 24 日 分类: [Javascript](http://lison.sinaapp.com/?cat=6) 标签: [Backbone.js](http://lison.sinaapp.com/?tag=backbone-js), [Javascript](http://lison.sinaapp.com/?tag=javascript), [Tutorial](http://lison.sinaapp.com/?tag=tutorial), [前端开发](http://lison.sinaapp.com/?tag=%e5%89%8d%e7%ab%af%e5%bc%80%e5%8f%91), [结构化](http://lison.sinaapp.com/?tag=%e7%bb%93%e6%9e%84%e5%8c%96)

**Backbone.js中的Model类简介**

本文适于对面向对象的JavaScript开发感兴趣的前端开发人员

纵观目前互联网上关于MVC的定义，各种说法不尽相同，笔者也很难给出一个准确的定义。在Backbone.js中，作者给除了Model在Backbone.js中所代表的含义：

Models are the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.  
Model在任何JavaScript应用程序中都算得上是核心模块，包括需要频繁存取的数据以及大量围绕数据展开的会话、校验、计算属性以及访问控制。

**既然这个章节是介绍Model，那我们就先创建Backbone.js的Model：**

1. Person = Backbone.Model.extend({
2. initialize: function(){
3. alert("Welcome to this world");
4. }
5. });
6. var person = new Person;

显而易见，当我们使用这里定义的Person去创建新的对象时会立即调用initialize()函数（initialize()在Model、 Collection、View中都是这样工作的）。虽然，initialize()函数并不是必须要定义的函数，但在实际的开发过程中，你会渐渐发现它 的神奇之用处。

**设置属性：**

我们可以在初始化Model对象的时候传入我们想设置的属性：

1. Person = Backbone.Model.extend({
2. initialize: function(){
3. alert("Welcome to this world");
4. }
5. });
6. var person = new Person({ name: "Thomas", age: 67 });
7. delete person;
8. // or we can set afterwards, these operations are equivelent
9. var person = new Person();
10. person.set({ name: "Thomas", age: 67 })

和在创建时传入需要设置的参数一样，Model.set()同样可以实现对对象的属性进行设置。当我们对对象设置了属性，那么对象中将会含有一个属性集，我们可以使用相应的方法去读取其中的值。

**获取属性：**

Model.get()方法可以让我们方便的获得对象的属性：

1. Person = Backbone.Model.extend({
2. initialize: function(){
3. alert("Welcome to this world");
4. }
5. });
6. var person = new Person({ name: "Thomas", age: 67, children: ['Ryan'] });
7. var age = person.get("age"); // 67
8. var name = person.get("name"); // "Thomas"
9. var children = person.get("children"); // ['Ryan']

**设置默认属性：**

有些时候需要对Model设置默认值。Backbone.js提供了default属性用于对Model对象的默认值设置：

1. Person = Backbone.Model.extend({
2. defaults: {
3. name: 'Fetus',
4. age: 0,
5. children: []
6. },
7. initialize: function(){
8. alert("Welcome to this world");
9. }
10. });
11. var person = new Person({ name: "Thomas", age: 67, children: ['Ryan'] });
12. var age = person.get("age"); // 67
13. var name = person.get("name"); // "Thomas"
14. var children = person.get("children"); // ['Ryan']

**操作Model的属性：**

我们可以在 Model添加任意数量的自定义方法。默认情况下，这些自定义添加的方法是public方法，能够在闭包外面被调用。

1. Person = Backbone.Model.extend({
2. defaults: {
3. name: 'Fetus',
4. age: 0,
5. children: []
6. },
7. initialize: function(){
8. alert("Welcome to this world");
9. },
10. adopt: function( newChildsName ) {
11. var children\_array = this.get("children");
12. children\_array.push( newChildsName );
13. this.set({ children: children\_array });
14. }
15. });
16. var person = new Person({ name: "Thomas", age: 67, children: ['Ryan'] });
17. person.adopt('John Resig');
18. var children = person.get("children"); // ['Ryan', 'John Resig']

这样我们可以把一些频繁出现对对象内属性的操作以自定义方法的形式保存在Model中。

**监听Model对象的变化：**

下面所有讲解的是像Backbone.js这样的开发库的最重要的模块之一。Model的每个属性都可以有相应的监听函数，即当值发生变化时监听函 数就会被调用。在下面的代码中，在初始化函数中添加了对于Person的属性的监听函数。于是Person的对象的属性发生变化时，监听函数会立即被调 用。

1. Person = Backbone.Model.extend({
2. defaults: {
3. name: 'Fetus',
4. age: 0,
5. children: []
6. },
7. initialize: function(){
8. alert("Welcome to this world");
9. this.bind("change:name", function() {
10. var name = this.get("name"); // 'Stewie Griffin'
11. alert("Changed my name to " + name);
12. });
13. },
14. replaceNameAttr: function( name ) {
15. this.set({ name: name });
16. }
17. });
18. var person = new Person({ name: "Thomas", age: 67, children: ['Ryan'] });
19. person.replaceNameAttr('Stewie Griffin');
20. // This triggers a change and will alert()

在上面的例子里面，我们仅仅监听了Person的name属性，但我们也可以监听整个对象所有属性的变化。

**Model对象的获取、保存以及销毁：**

Model在进行获取、保存以及销毁的操作过程中，通常会以Collection的部分的形式出现。这一章的教程更大程度上是讲解单独的Model对象。所以，对于Model对象的上述操作将会在讲解Collection一章节中一并讲解。

## 第二章

**Backbone.js中的View类简介**

本文适于对面向对象的JavaScript开发感兴趣的前端开发人员

Backbone.js中的Views是用来呈现Web应用程序中的数据模型，并完成对于事件的监听以及对事件的响应。本文将不会赘述如何将 Model、Collection绑定在View上，将主要描述View本身的功能特点以及如何在View中配合使用JavaScript模版（本文将主 要介绍Underscore.js’s 中的\_.template）。

这里我们将会使用jQuery1.5作为基础函数库对DOM节点进行操作。当然，我们也可以使用其他同来的JavaScript基础函数库，譬如 MooTools或是Sizzle，但在Backbone.js的官方文档中推荐使用jQuery。另外一方面，Backbone.View中的事件相关 的功能函数无法在没有jQuery的情况下工作。

接下来我们会开发一个搜索框作为示例（[DEMO](http://jsfiddle.net/thomas/C9wew/6)）：

1. SearchView = Backbone.View.extend({
2. initialize: function(){
3. alert("Alerts suck.");
4. }
5. });
7. // initialize()会在View被初始化时立即执行
8. // 换句话说，这里的initialize()可以被看成构造函数
9. var search\_view = new SearchView;

**“el”属性：**

“el”属性用于存储浏览器中原生态的DOM对象。每个Backbone的View对象都会具有el属性，如果不存在则Backbone.js会自动生成一个空的Div元素赋值给el。

这里我们将View的“el”属性设置为#search\_container值，就相当于将Backbone.View设置为了这个DOM元素的操控者。

1. <div id="search\_container"></div>
3. <script type="text/javascript">
4. SearchView = Backbone.View.extend({
5. initialize: function(){
6. alert("Alerts suck.");
7. }
8. });
10. var search\_view = new SearchView({ el: $("#search\_container") });
11. </script>

说明：所有在View内的事件触发都必须在View的el所指向的元素上。

**加载模版**

Backbone.js是基于Underscore.js，而Underscore里面包含了微型模版组建。（[Underscore.js文档](http://documentcloud.github.com/underscore/)）

下面我们编写“render()”，当View被初始化的时候会自动调用“render()”。然后，在render()中将我们模版中的内容加载到View的“el”元素中去。

1. <div id="search\_container"></div>
3. <script type="text/javascript">
4. SearchView = Backbone.View.extend({
5. initialize: function(){
6. this.render();
7. },
8. render: function(){
9. // Compile the template using underscore
10. var template = \_.template( $("#search\_template").html(), {} );
11. // Load the compiled HTML into the Backbone "el"
12. this.el.html( template );
13. }
14. });
16. var search\_view = new SearchView({ el: $("#search\_container") });
17. </script>
19. <script type="text/template" id="search\_template">
20. <label>Search</label>
21. <input type="text" id="search\_input" />
22. <input type="button" id="search\_button" value="Search" />
23. </script>

说明：把模版们都存储在一个文件里面，并将文件放在CDN上。使得这些模版文件能够被用户缓存，减少不必要的网络流量。

**监听事件**

如果需要在我们的View中添加事件绑定，我们就需要在Backbone.View中添加events属性。另外，我们需要强调事时间绑定只适用鱼View的el元素的子元素。下面我们对我们例子中的按钮添加事件绑定。

1. <div id="search\_container"></div>
3. <script type="text/javascript">
4. SearchView = Backbone.View.extend({
5. initialize: function(){
6. this.render();
7. },
8. render: function(){
9. var template = \_.template( $("#search\_template").html(), {} );
10. this.el.html( template );
11. },
12. events: {
13. "click input[type=button]": "doSearch"
14. },
15. doSearch: function( event ){
16. //按钮被点击后，你可以使用event.currentTarget获得被点击的元素
17. alert( "Search for " + $("#search\_input").val() );
18. }
19. });
21. var search\_view = new SearchView({ el: $("#search\_container") });
22. </script>
24. <script type="text/template" id="search\_template">
25. <label>Search</label>
26. <input type="text" id="search\_input" />
27. <input type="button" id="search\_button" value="Search" />
28. </script>

**技巧**

关于在模版中使用变量

1. <div id="search\_container"></div>
3. <script type="text/javascript">
4. SearchView = Backbone.View.extend({
5. initialize: function(){
6. this.render();
7. },
8. render: function(){
9. // 设定传输给模版的变量
10. var variables = { search\_label: "My Search" };
11. // 用underscore中\_.template()渲染模版以及变量
12. var template = \_.template( $("#search\_template").html(), variables );
13. // 将渲染好的内容设置到View的el元素中
14. this.el.html( template );
15. },
16. events: {
17. "click input[type=button]": "doSearch"
18. },
19. doSearch: function( event ){
20. // 按钮被点击后，你可以使用event.currentTarget获得被点击的元素
21. alert( "Search for " + $("#search\_input").val() );
22. }
23. });
25. var search\_view = new SearchView({ el: $("#search\_container") });
26. </script>
28. <script type="text/template" id="search\_template">
29. <!-- 用<%= %>调用传输给模版的变量 -->
30. <label><%= search\_label %></label>
31. <input type="text" id="search\_input" />
32. <input type="button" id="search\_button" value="Search" />
33. </script>

原文链接

## 第三章

**ackbone.js中的Router类简介**

*本文适于对面向对象的JavaScript开发感兴趣的前端开发人员*

Backbone中Router类实现了用hash标签对页面实现路由控制。在传统的MVC模式下并不包含这里的Router所 实现的功能，如果你已经阅读了“View类简介”你就会发现View完成了本应由Router完成的功能（从传统MVC的角度看）。尽管如 此，Backbone中的Router类对于需要实现URL路由或是历史记录功能的Web应用程序仍然是很重要的部分。

我们开发Router时，通常会在里面包含至少一个Route以及用于完成映射的函数。在下面的例子里面我们定的route在每次访问时都会被调用。

另外说明一点：Router所能够处理的是URL中#后的所有内容。换句话说，基于Router的Web应用程序里，所有的链接都会以 “#/action”或“#action”结尾。（在#后面添加/让URL更利于阅读，例如，http://example.com/#/user /help）

1. <script>
2. var AppRouter = Backbone.Router.extend({
3. routes: {
4. "\*actions": "defaultRoute" // 对应于http://example.com/#anything-here
5. },
6. defaultRoute: function( actions ){
7. // alert所有Route的action
8. alert( actions );
9. }
10. });
11. // 初始化Router
12. var app\_router = new AppRouter;
13. // 开启Backbone的历史功能
14. Backbone.history.start();
16. </script>
17. [Activate route](#action)
18. [Activate another route](#/route/action)
19. \_Notice the change in the url\_

请注意：先前的Backbone 0.5版本（2011年7月1日发布）中的Router被称为Controller。为了避免混淆，Backbone的开发人员将名称改为了 Router。因此，如果你发现你在使用的Backbone是0.5版本之前的那那就需要用Backbone.Controller.extend({ \*\* })；

**动态路由**

大多现存的框架都能够支持你定义包含静态内容以及动态参数的路由。例如你需要通过文章id获取到文章的链接，很常见的例子就是“http://example.com/#/posts/12”。当链接被访问时，你希望能够获取到id值。下面就用程序实现了我们的要求：

1. <script>
2. var AppRouter = Backbone.Router.extend({
3. routes: {
4. "/posts/:id": "getPost",
5. "\*actions": "defaultRoute" // Backbone会按照从上到下的顺序对Route进行匹配
6. },
7. getPost: function( id ) {
8. // 在Route中定义的变量，可以传入到Route对应的处理函数
9. alert( "Get post number " + id );
10. },
11. defaultRoute: function( actions ){
12. alert( actions );
13. }
14. });
15. // 初始化Router
16. var app\_router = new AppRouter;
17. // 开启Backbone的历史功能
18. Backbone.history.start();
20. </script>
21. [Post 120](#/posts/120)
22. [Post 130](#/posts/130)
23. \_Notice the change in the url\_

**动态路由（”:params”和”\*splats”）**

在Backbone的Route中，我们有两种方式传递Route中的变量。第一种：“:params”用于匹配URL中用“/”隔开的一部分；第 二种：“\*splats”用于匹配从前面的字串的下一个字符开始直至URL结尾。换句话说，用“\*splat”匹配到的变量总是URL中最后一个变量。

任何在Route中定义的变量最终都会以变量的形式传递到用户所绑定函数。例如，“/:route/:action”会传递两个变量给处理这一URL的函数（“route”和“action”）。

1. routes: {
2. "/posts/:id": "getPost",
3. // <a href="http://example.com/#/posts/121">例子</a>
5. "/download/\*path": "downloadFile",
6. // <a href="http://example.com/#/download/user/images/hey.gif">下载</a>
8. "/:route/:action": "loadView",
9. // <a href="http://example.com/#/dashboard/graph">加载内容</a>
10. },
11. getPost: function( id ){
12. alert(id); // 121
13. },
14. downloadFile: function( path ){
15. alert(path); // user/images/hey.gif
16. },
17. loadView: function( route, action ){
18. alert(route + "\_" + action); // dashboard\_graph
19. }

Route是非常有用组件，但不可滥用。如果你在使用Route的同时考虑网站的SEO，可以在Google上搜索“google seo hashbangs”

## 第四章

**ackbone.js中的Collection类简介**

*本文适于对面向对象的JavaScript开发感兴趣的前端开发人员*

简单地说，Backbone的Collection就是一组Model的集合。下面是几个使用场景：

* Model: Student, Collection: ClassStudents
* Model: Todo Item, Collection: Todo List
* Model: Animals, Collection: Zoo

声明一点，在Collection中只能存放单一类型的Model，但是同一种类型的Model可以在不同的Collection中出现：

* Model: Student, Collection: Gym Class
* Model: Student, Collection: Art Class
* Model: Student, Collection: English Class

下面是一个常见的例子：

1. var Song = Backbone.Model.extend({
2. initialize: function(){
3. console.log("Music is the answer");
4. }
5. });
7. var Album = Backbone.Collection.extend({
8. model: Song
9. });

**Collection的创建**

下面我们将演示如何完成创建：

1. var Song = Backbone.Model.extend({
2. defaults: {
3. name: "Not specified",
4. artist: "Not specified"
5. },
6. initialize: function(){
7. console.log("Music is the answer");
8. }
9. });
11. var Album = Backbone.Collection.extend({
12. model: Song
13. });
15. var song1 = new Song({ name: "How Bizarre", artist: "OMC" });
16. var song2 = new Song({ name: "Sexual Healing", artist: "Marvin Gaye" });
17. var song3 = new Song({ name: "Talk It Over In Bed", artist: "OMC" });
19. var myAlbum = new Album([ song1, song2, song3]);
20. console.log( myAlbum.models ); // [song1, song2, song3]

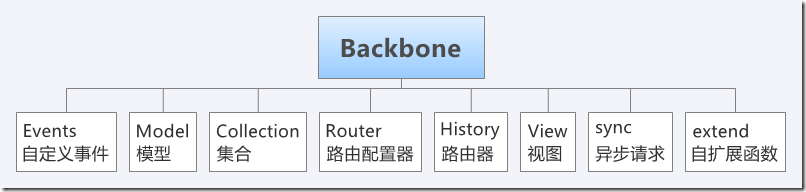
## 学习网站

## <http://www.cnblogs.com/nuysoft/archive/2012/03/19/2404274.html>

声明：本文为原创文章，如需转载，请注明来源并保留原文链接。

[Backbone0.9.1源码分析分析系列](http://www.cnblogs.com/nuysoft/archive/2012/03/14/2395260.html)

[jQuery1.6.1源码分析系列](http://www.cnblogs.com/nuysoft/archive/2011/11/14/2248023.html)

[](http://images.cnblogs.com/cnblogs_com/nuysoft/201203/201203180243287136.png)

# 参考资料

<http://documentcloud.github.com/backbone/>   
<http://www.csser.com/tools/backbone/backbone.js.html>

# 官网介绍

Backbone通过提供模型Models、集合Collection、视图Veiew赋予了Web应用程序分层结构。通过以下方式实现分层结构：   
λ 模型Model绑定键值数据和自定义事件；   
λ 集合Colection是模型的有序或无序集合，带有丰富的可枚举API；   
λ 视图View声明事件监听函数；   
λ 将模型、集合、视图与服务端的RESTful JSON接口连接。

# 自调用匿名函数

整个Backbone的源码用一个自调用匿名函数包裹，可以通过闭包特性引用变量（例如 previousBackbone、slice、splice等），同时避免污染全局命名空间。   
整体结构如下，还是很清晰的：

1: (function() {

2: Backbone.Events // 自定义事件

3: Backbone.Model // 模型构造函数和原型扩展

4: Backbone.Collection // 集合构造函数和原型扩展

5: Backbone.Router // 路由配置器构造函数和原型扩展

6: Backbone.History // 路由器构造函数和原型扩展

7: Backbone.View // 视图构造函数和原型扩展

8: Backbone.sync // 异步请求工具方法

9: var extend = function (protoProps, classProps) { ... } // 自扩展函数

10: Backbone.Model.extend = Backbone.Collection.extend = Backbone.Router.extend = Backbone.View.extend = extend; // 自扩展方法

11: }).call(this);

# 依赖库

Backbone必须依赖于 Underscore.js，DOM操作和AJAX请求依赖于第三方jQuery/Zepto/ender之一，也可以通过 Backbone.setDomLibrary( lib ) 设置其他的第三方库。

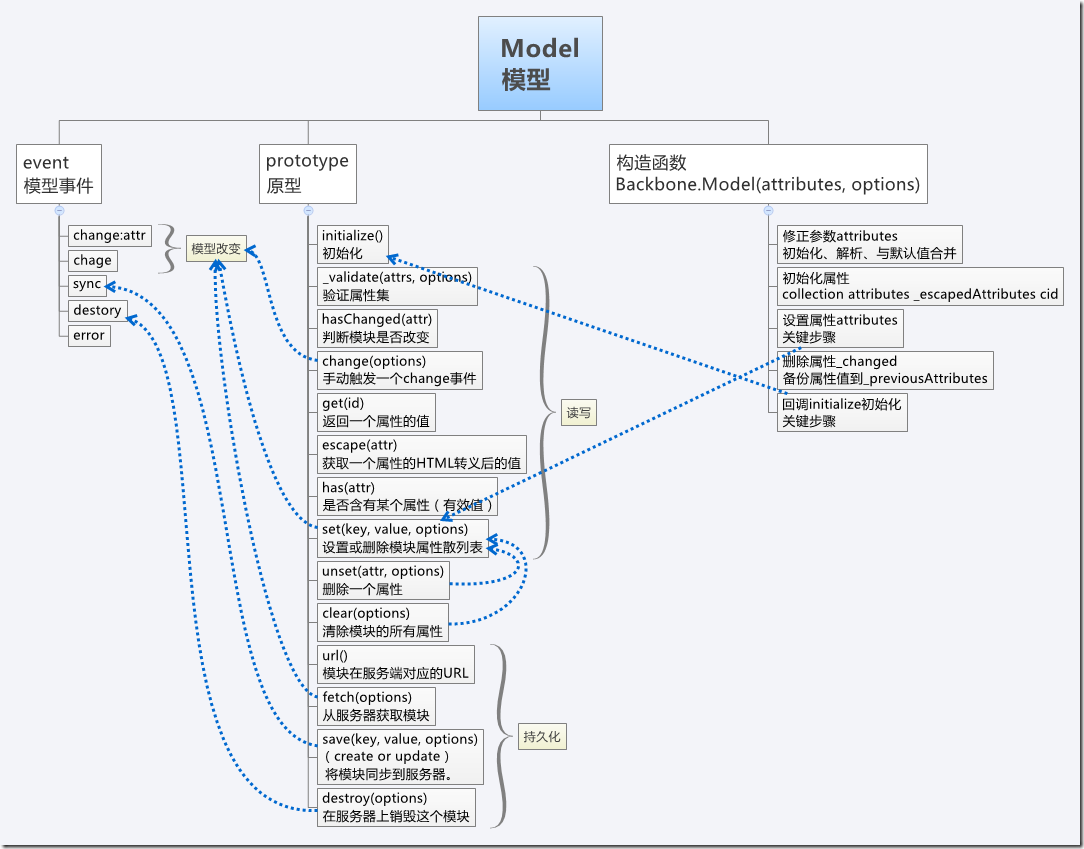
# 自定义事件模块 Backbone.Events

可以和任意对象合体（将方法赋值到其他对象或原型上），合体后的对象可以自定义事件，提供三个方法来绑定、移除、触发自定义事件。

[](http://images.cnblogs.com/cnblogs_com/nuysoft/201203/201203180243289154.png)

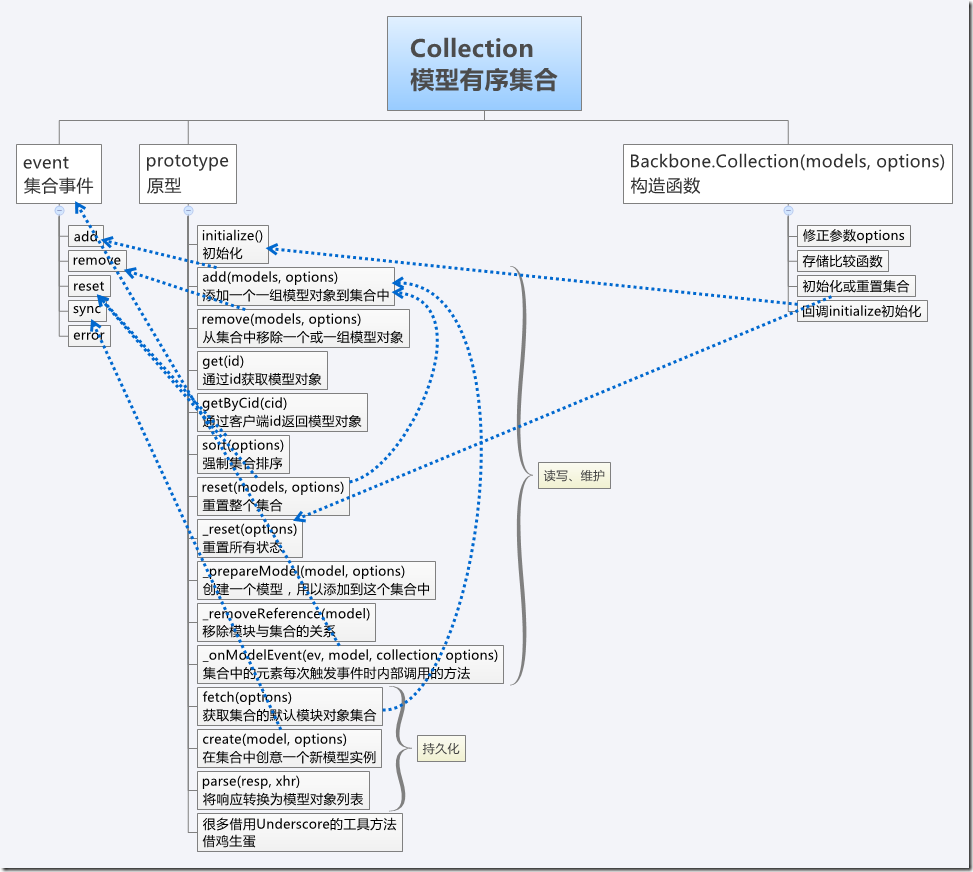
# 模型 Backbone.Model

是 JavaScript 应用程序的核心，包含业务数据和对业务数据的读写和持久化，模型的主要方法包括读写和持久化。

[](http://images.cnblogs.com/cnblogs_com/nuysoft/201203/201203180243294237.png)

# 集合 Backbone.Collection

集合是模型的有序集合，可以在集合上绑定 "change" 事件，当集合中的任何模型发生变化时收到通知，集合也可以监听 "add" 和 “remove" 事件， 从服务器获取数据，并能使用 Underscore.js 提供的全套方法。   
为了方便，在集合中的模型上触发的任何事件都会在集合上直接触发。这样就可以监听集合中模型的指定属性的变化。 例如：Documents.on("change:selected", ...)   
集合的主要方法包括读写、维护和持久化。

[](http://images.cnblogs.com/cnblogs_com/nuysoft/201203/201203180243308239.png)

# 路由配置器 Backbone.Router

Web应用程序通常需要为重要页面提供可链接、收藏、可分享的 URL。 直到最近， 锚文片段（hash #page）可以被用来提供这种固定链接；同时随着 History API 的到来，锚文现在可以用于处理标准 URLs （/page）。

Backbone.Router 为客户端页面路由提供了许多方法，并能连接到指定的动作（actions）和事件（events）。对于不支持 History API 的旧浏览器，路由器提供了优雅的回调函数并可以透明的进行 URL 片段的转换。

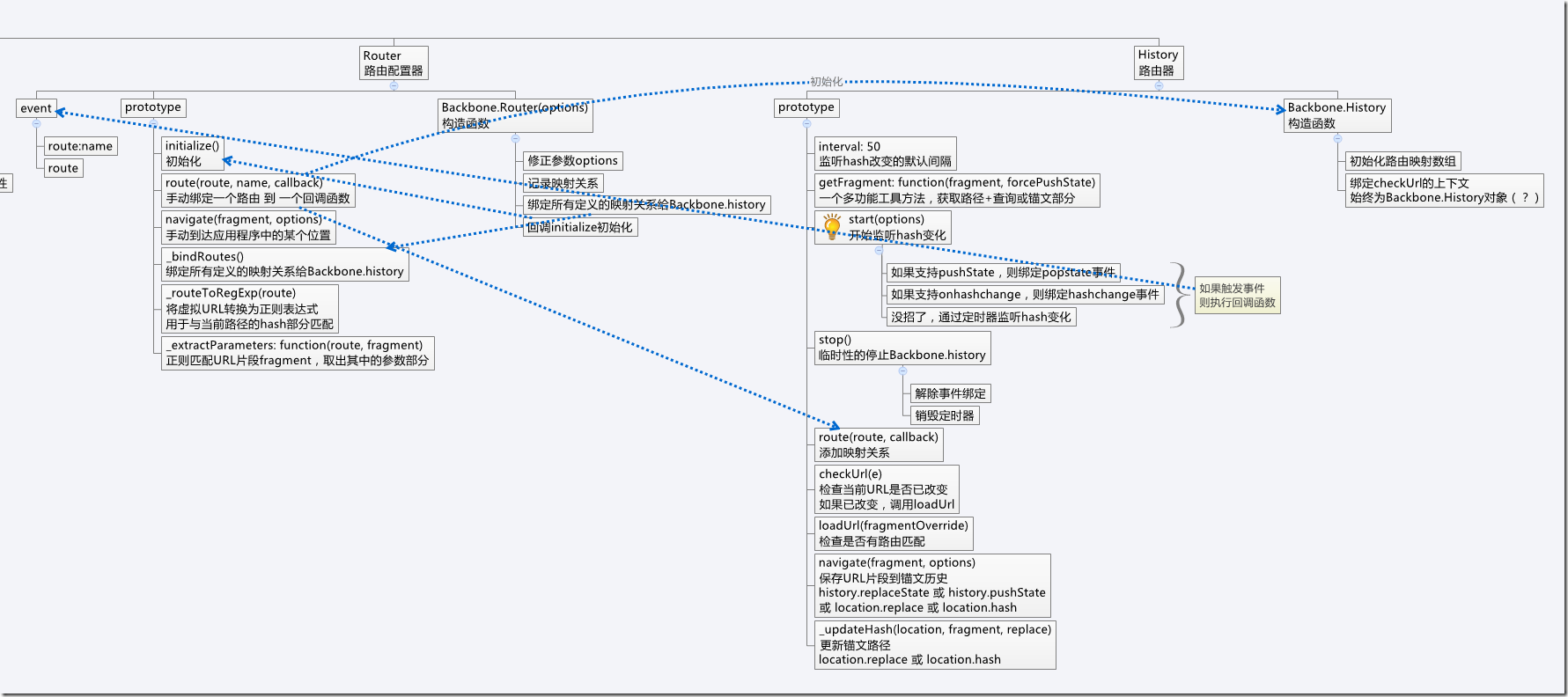
页面加载期间，当应用程序已经创建了所有的路由表，需要调用 Backbone.history.start()，或 Backbone.history.start({pushState : true}) 确保路由初始 URL。

## 

# 路由器 Backbone.History

作为全局路由器，用于处理 hashchange 或 pushState事件，匹配适合的路由表，并触发回调函数。   
如果使用带有路由表的路由器，会自动创建一个History对象，此时不要再创建一个History对象，而是使用 Backbone.history。   
Backbone 会自动判断浏览器对 pushState 的支持，以做内部的选择。 不支持 pushState 的浏览器将会继续使用基于锚点的 URL 片段。

**关于官网疑惑：**如果兼容 pushState 的浏览器访问了某个 URL 锚点，将会被透明的转换为真实的 URL。 注意使用真实的 URLs 需要 web 服务器支持直接渲染那些页面，因此后端程序也需要做修改。   
浏览器发起HTTP请求时并不会把锚文部分放在请求头中，查看 http rfc2612 也没有发现相关的说明，保留疑问。   
<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

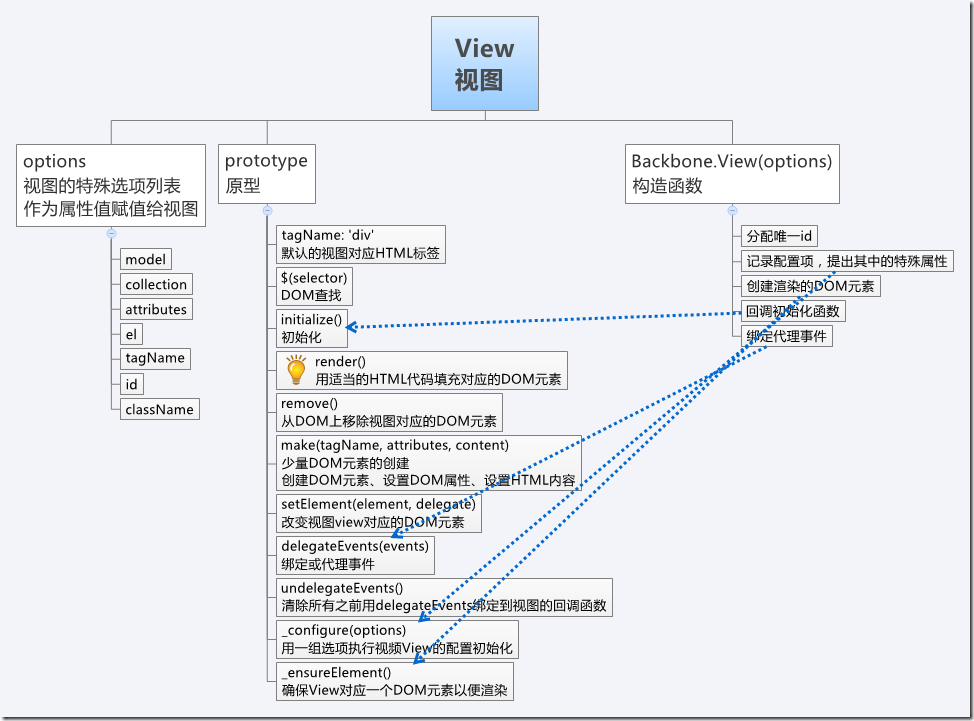
[](http://images.cnblogs.com/cnblogs_com/nuysoft/201203/201203180247367116.png)

# 视图 Backbone.View

视图的使用相当方便，不需要判断任何HTML、CSS，可以任意JavaScript模板引擎集成。

通用的做法是，将界面组织成基于模型的诗句，当模型改变时视图立即更新，而不需要重画整个页面。

不再需要纠结于JSON对象、查找DOM元素、手动更新HTML，只需把视图 render 方法绑定到模型的 change 事件，模型数据会立即显示到UI上。

[](http://images.cnblogs.com/cnblogs_com/nuysoft/201203/201203182310341300.png)

# 扩展方法 extend

模型、集合、视图、路由器都一个extend方法，用于扩展原型属性和静态属性，创建自定义的视图、集合、视图、路由器类。

分类: [Backb](http://www.cnblogs.com/nuysoft/category/362937.html)

# JSMVC职责划分

**M 模型**

* 业务模型：业务逻辑、流程、状态、规则
* （核心）数据模型：业务数据、数据校验、增删改查（AJAX）

**V 视图**

* （核心）视图：定义、管理、配置
* 模板：定义、配置、管理
* 组件：定义、配置、管理
* （核心）用户事件配置、管理
* 用户输入校验、配置、管理

**C 控制器/分发器**

* （核心）事件分发、模型分发、视图分发
* 不做数据处理、业务处理，即业务无关
* 扩展：权限控制、异常处理等
* C是JSMVC框架的核心，实现集中式配置和管理，可以有多个控制器

**工具库**

* 主要是异步请求、DOM操作，可以依赖于jQuery等

# JSMVC实现探讨

MVC的模型Model、视图View、控制器Control三个相互独立又相互联系，C作为其中的桥梁。MVC作为由来已久的成熟开发模型，已经有经典的实现可控参考，在浏览器和JS这个特定应用场景中，我们做如下探讨：

**M模型**是**自包含**的，可以**嵌套包含**，不会主动引用视图和控制器；可以是简单的JSON对象/数组，也可以用**组合模式Composite**实现嵌套包含；

**V视图**是**嵌套包含**的，可以用组合Composite实现；视图需要引用模型（**M-V**），一个视图引用一个多个视图，视图会收到模型的通知并自动更新，可以用**观察者模式Observer**实现；视图需要响应用户的交互，使用**浏览器事件模型**；

**C控制器**作为MVC框架关注的核心，采取**集中配置**的策略（**V-C**）；可以有多个控制器（**C+C**），用**策略模式Strategy**实现；hash事件驱动需要用到浏览器事件模型。

可以看到M模型和C控制器之间没有关联，一般我认为是不需要的；但是在有的框架中实现了M模型和C控制器事件的关联（比如公司在用的Magix），也是一种实践，可以参考。

上边的论述解析MVC在JS中定位，但是理解JSMVC如何运行的关键在于对事件驱动的理解。

# 事件驱动

在浏览器中JSMVC有**三种事件驱动方式**：

1. **hash驱动**，通过popstate/hashchange事件驱动控制器（前边有过介绍，后续结合Backbone的Router和History详细讲解它原理、实现、技巧）

2. **DOM事件**，用来驱动视图（这个我们已经很熟悉了）

3. **模型事件**（业务模型事件和数据模型事件），用来驱动模型和模型结合（这个后续结合Backbone的Model讲解）

今天就写到这里，下节介绍Backbone的架构、能做什么、不能做什么、以及还能扩展什么。

# 代码混乱

前端攻城师是个新兴的职业，一直以来前端开发面临着以下的问题：

* 数据模型、业务模型不清晰
* 代码代码复用率低，很多时候是Ctrl-C/V
* 代码可读性和可维护性低
* 难以满足需求的变化，特别是前端需求的频繁变化

jQuery在一定程度上改善了这种状态，解决了原生JS代码写起来繁琐的问题，但jQuery毕竟是一个以DOM查找和操作为主的库，这样的定位使得jQuery的大型前端项目中只能作为底层库使用，应用代码依然是杂乱无章，不得不依赖于攻城师的技术水准和职业素养。

而MVC模型通过细分模型、视图、控制器的职责，约定开发方式，让代码开发和管理变得条理、清晰。JS应用程序的本质是事件驱动，而MVC适合于事 件驱动的场景，两者不谋而合。MVC在大部分语言的框架中都有实现，JavaScript虽然在这方面起步较晚，但随着前端和JS越来越被认可和重 视，JSMVC框架的出现成为必然。

Backbone正是一个优秀的JSMVC框架。

# 传说中的单页面应用程序OPOA

在JS半死不活的年代，面临着随时被Flash吃掉的恐惧和各种市场唱衰，以及大公司富客户端产品的竞争，像Sun的JavaFX、微软的Sliveright，AJAX的异步请求和动态加载，可以说是力挽狂澜，有着拥立之功。

在AJAX被发掘之后，编写类似桌面程序的单页面应用程序OPOA“似乎”成为了可能，但是随后发现不能支持浏览器历史的前进、后退、收藏，使得OPOA成了各种DEMO秀，一直没有流行起来。

直到对URL hash的挖掘（location.hash），OPOA才变得可行。hash虽然不会在请求时传给服务器，但是在浏览器中，改变hash却可以前进、后退、收藏（在低版本的IE中可以通过iframe实现）。

hash驱动正是Backbone事件驱动模型中一种，在某些应用中可能是最重要的一种。

Backbone还有模型事件驱动和视图事件驱动， Backbone也不仅仅应用在OPOA中，在后边的章节将逐步介绍。

# 还不够完美

还有模块依赖、动态载入、模板等等需要去研究和学习，这些概念和框架已经不是什么新鲜事务了，但是如何将各种专注的库、框架、工具集成，变成完整、可靠、稳定、高效开发、可以成熟应用的系统解决方案，是前端领域需要继续研究和实践的课题。

我也是刚开始接触和学习，有感于最近这些知识受到的冲击，罗哩罗嗦的写了这些。下边开始分析我对MVC在浏览器中应用的认识。

# MVC的优点和缺点

MVC可以简化重构、解耦合、提高代码复用、适应变化、易读、较少维护代码、提高可维护性，因为是结构化模块化开发，还可以实现代码自动生成。但是MVC也会导致复杂度上升、运行效率下降。

真的解耦了吗？只是尽可能的解耦，事实上也不可能完全解耦。模型与视图的关系在实际应用中的关系可能更加复杂，视图也不仅仅负责渲染、交互，还可能需要模板引擎、数据解析、适配等等。

为了更好的分析JSMVC的实现，我们先对MVC在JS中的职责进行梳理。请看下节 [Backbone源码分析系列之JSMVC](http://www.cnblogs.com/nuysoft/archive/2012/03/14/2395272.html)。

### [让我们用Backbone.js来写一个HelloWorld程序。](http://blog.csdn.net/soasme/article/details/6581029)

2011-07-02 22:57 18061人阅读 [评论](http://blog.csdn.net/soasme/article/details/6581029#comments)(13) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/soasme/article/details/6581029#report)

[backbone.js](http://blog.csdn.net/tag/details.html?tag=backbone.js)[underscore](http://blog.csdn.net/tag/details.html?tag=underscore)[function](http://blog.csdn.net/tag/details.html?tag=function)[collections](http://blog.csdn.net/tag/details.html?tag=collections)[html](http://blog.csdn.net/tag/details.html?tag=html)[jquery](http://blog.csdn.net/tag/details.html?tag=jquery)

新建一个api.php文件，内容：

**[php]** [view plaincopyprint?](http://blog.csdn.net/soasme/article/details/6581029)

1. header('Content-Type: application/json; charset=utf-8');
2. die(json\_encode(array('name'=>'tom')));

新建一个index.html文件。(backbone基于jquery、underscore，我们使用Mustache来做模板解析，当然用其他的haml、jade，或者underscore里面的模板也都是可以)

内容：

**[html]** [view plaincopyprint?](http://blog.csdn.net/soasme/article/details/6581029)

1. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2. <HTML>
3. <HEAD>
4. <TITLE> New Document </TITLE>
5. <script type="text/javascript" src="./jquery.min.js"></script>
6. <script type="text/javascript" src="./underscore.min.js"></script>
7. <script type="text/javascript" src="./backbone.min.js"></script>
8. <script type="text/javascript" src="./mustache.min.js"></script>
9. <script type="text/javascript" src="./custom.js"></script>
10. </HEAD>
11. <BODY>
12. <p><script id="hello-container-template" type="text/template"></p><p><div>{{name}} says: {{message}} </div></p><p></script></p>
13. </BODY>
14. </HTML>

新建一个custom.js文件，内容：

**[html]** [view plaincopyprint?](http://blog.csdn.net/soasme/article/details/6581029)

1. // 这是一个管理着 视图/控制/模型 的全局类
2. var App = {
3. Models: {},
4. Views: {},
5. Controllers: {},
6. Collections: {},
7. initialize: function() {
8. new App.Controllers.Routes();
9. Backbone.history.start() // 要驱动所有的Backbone程序，Backbone.history.start()是必须的。
10. }
11. };
12. App.Models.Hello = Backbone.Model.extend({
13. url: function() {
14. return '/api.php'; // 获得数据的后台地址。
15. },
16. initialize: function() {
17. this.set({'message':'hello world'}); // 前端定义一个message字段，name字段由后端提供。
18. }
19. });
20. App.Views.Hello = Backbone.View.extend({
21. el: $("body"),
22. template: $("#<span style="font-family: monospace; white-space: pre; ">hello-container-template</span>").html(),
23. initialize: function(options){
24. this.options = options;
25. this.bind('change', this.render);
26. this.model = this.options.model;
27. },
28. render: function(){ // render方法，目标只有两个：填充this.el，返回this以便链式操作。
29. $(this.el).html(Mustache.to\_html($(this.el).template,this.model.toJSON()) );
30. return this
31. }
32. });
33. App.Controllers.Routes = Backbone.Controller.extend({
34. routes: {
35. "!/hello" : "hello",//使用#!/hello驱动路由
36. },
37. hello : function() {
38. //新建一个模型，模型向后端请求更新内容成功后根据模型渲染新页面
39. var helloModel = new App.Models.Hello;
40. helloModel.fetch({
41. success: function(model){
42. var helloView = new App.Views.Hello({model: model});
43. helloView.trigger('change');
44. }
45. })
46. }});
47. App.initialize();