Backbone.js 入门教程

作者：胡阳（the5fire）

blog:http://www.the5fire.net

关于作者更多信息：
http://www.the5fire.net/about

制作日期：2012-04-19

版权声明
本电子读物所登载文章,著作权均归作者本人持有。
除特别声明外,
本电子读物之内容采用如下 CC (Creative Commons) 协议授权:
署名-非商业性使用-相同方式共享。

目录

写在前面的话

这一系列的文章写了这么久，也算是告一段落了，为了方便大家查看，制作成 PDF 格式的放到网上，待有兴趣学习 backbone.js 的同学参考。

第一次写完一系列的东西，以前有过很多写系列文章的冲动，不过都是写了一段时间就因为一些事放下了，如：设计模式，还有 tomcat 源码。

其实这一系列文章的主要目的还是让初学 backbone.js 的人，能够快速的把它用到项目上。写 backbone 的原因是，在我搜索查找关于它的学习资料时，发现中文的资料比较少也比较散，虽然看到有网友在博客上说打算写，但毕竟只是打算，因此 ，我就一边学习 ，一边把里面的东西大概的梳理了一下，写成文章。

把这些东西写出来，不是说明我有多厉害，而只是表现我渴望学习、提高、分享。对于这些东西我未必完全掌握，但是我尽量把我知道的东西写出来，分享出去。

任何一个人都是从菜鸟慢慢成长起来的，而你成长过程中的所有经历恰恰又是下一代或者说你后面菜鸟所渴望知道的，同时也是对你以后成长大有帮助的。所以不管你觉得自己有多菜，你都应该把你学到的，思考的东西写下来，哪怕只有一点。

所有的文章中会有很多不足的地方，你如果发现错误，欢迎到对应的博文链接上拍砖。每篇文章都给了链接，方便大家快速跳转到网页。

最后，写上一句话，以显示我的文学水平（表拍砖）：没有开始，怎么会有成长;没有总结，怎么会有收获;没有分享，怎么会有升华。

1、初识 backbone.js

作者：胡阳

本文链接:http://www.the5fire.net/touch-backbone-js.html

backbone，英文意思是：勇气， 脊骨，但是在程序里面，尤其是在 backbone 后面加上后缀 js 之后，它就变成了一个框架，一个 js 库。

backbone.js，不知道作者是以什么样的目的来对其命名的，可能是希望这个库会成为 web 端开发中脊梁骨。

好了，八卦完了开始正题。

backbone.js 提供了一套 web 开发的框架，通过 Models 进行 key-value 绑定及 custom 事件处理,通过 Collections 提供一套丰富的 API 用于枚举功能,通过 Views 来进行事件处理及与现有的 Application 通过 RESTful JSON 接口进行交互.它是基于 jquery 和 underscore 的一个 js 框架。

整体上来说，backbone.js 是一个 web 端 javascript 的 mvc 框架，算得上是重量级的框架。它能让你像写 java 代码一些写 js 代码，定义类，类的属性以及方法。更重要的是它能够优雅的把原本无逻辑的 javascript 代码进行组织，并且提供数据和逻辑相互分离的方法，减少代码开发过程中的数据和逻辑混乱。

通过 backbone，你可以把你的数据当作 Models，通过 Models 你可以创建数据，进行数据验证，销毁或者保存到服务器上。当界面上的操作引起 model 中属性的变化时，model 会触发 change 的事件;那些用来显示 model 状态的 views 会接受到 model 触发 change 的消息，进而发出对应的响应，并且重新渲染新的数据到界面。在一个完整的 backbone 应用中，你不需要写那些胶水代码来从 DOM 中通过特殊的 id 来获取节点，或者手工的更新 HTML 页面，因为在 model 发生变化时，views 会很简单的进行自我更新。

上面是一个简单的介绍，关于 backbone 我看完他的介绍和简单的教程之后，第一印象是它为前端开发制定了一套自己的规则，在这个规则下，我们可以像使用 django 组织 python 代码一样的组织 js 代码，它很优雅，能够使前端和 server 的交互变得简单。

在查 backbone 资料的时候，发现没有很系统的中文入门资料和更多的实例，所以我打算自己边学边写，争取能让大家通过一系列文章能快速的用上 backbone.js。

关于 backbone 的更多介绍参看这个：

http://documentcloud.github.com/backbone/

http://backbonetutorials.com/

backbone 的应用范围：

它既然是一个重量级的框架，那就不是随便什么地方都能用的，不然就会出现杀鸡用牛刀，费力

不讨好的结果。那么适用在哪些地方呢？

根据我对 backbone.js 功能的理解，如果单个网页上有非常复杂的业务逻辑，那么用它很合适，它可以很容易的操作 dom 和组织 js 代码。

豆瓣的阿尔法城是一个极好的例子。当然，除了我自己分析的应用范围之外，在 backbone 的文档上看到了很多使用它的外国站点，有很多，说明 backbone 还是很易用的。

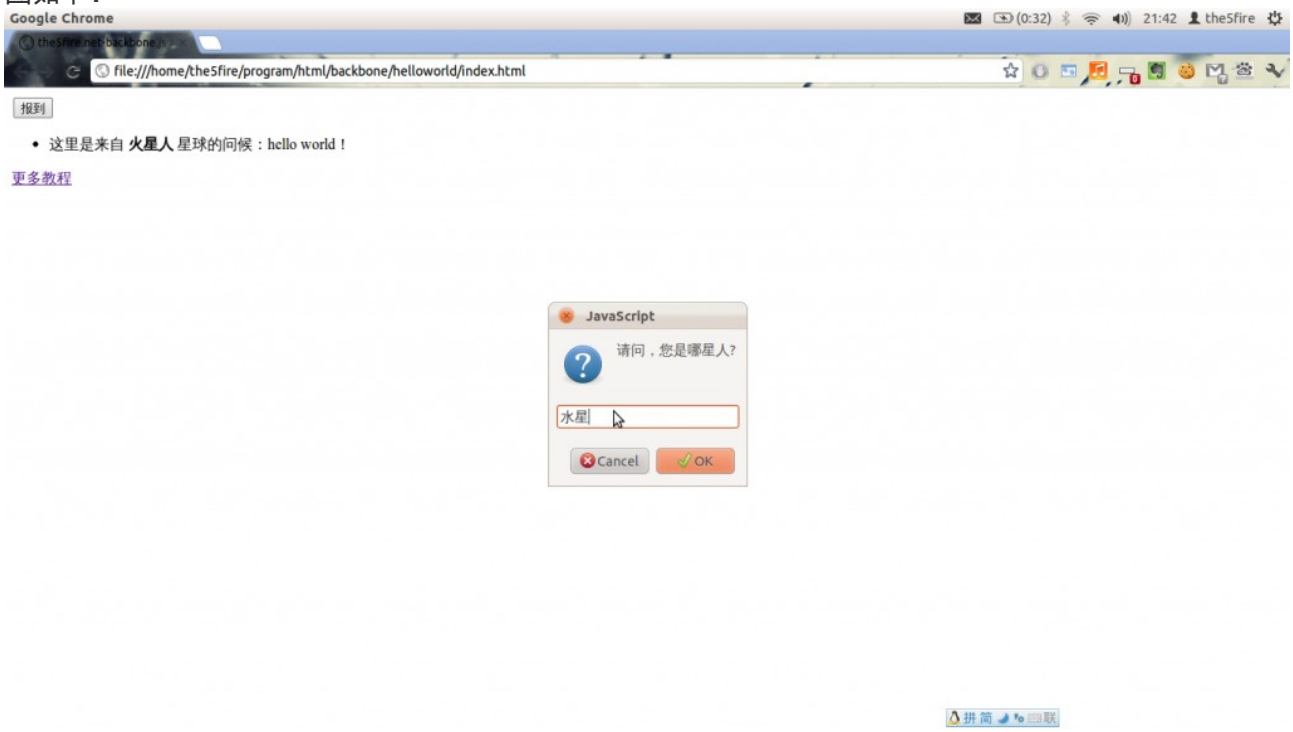
2、通过 helloworld 来认识下 backbone

作者：胡阳

本文链接:http://www.the5fire.net/2-helloworld-backbone.html

先来说一下这个 helloworld 的功能：

在页面上有一个报道的按钮，点击弹出输入框，输入内容，确认，最后内容会加到页面上。页面图如下：



下面来看代码：

```
<!DOCTYPE html>
<html>
<head>
 <title>the5fire.net-backbone.js-Hello World</title>
</head>
```

```
<body>
<buttonid="check">报到</button>
<ulid="world-list">

</ul>
<ahref="http://www.the5fire.net">更多教程</a>
</body>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
<scriptsrc="http://documentcloud.github.com/underscore/underscore-min.js"></script>
<scriptsrc="http://documentcloud.github.com/backbone/backbone-min.js"></script>
<script>
(function ($) {
 World = Backbone.Model.extend({
  //创建一个 World 的对象，拥有 name 属性
  name: null
 });

 Worlds = Backbone.Collection.extend({
  //World 对象的集合
  initialize: function (models, options) {
   this.bind("add", options.view.addOneWorld);
  }
 });

 AppView = Backbone.View.extend({
  el: $("body"),
  initialize: function () {
   //构造函数，实例化一个 World 集合类，并且以字典方式传入 AppView 的对象
   this.worlds = new Worlds(null, { view : this })
  },
  events: {
   "click #check": "checkIn", //事件绑定，绑定 Dom 中 id 为 check 的元素
  },
  checkIn: function () {
   var world_name = prompt("请问，您是哪星人?");
   if(world_name == "") world_name = '未知';
   var world = new World({ name: world_name });
   this.worlds.add(world);
```

```
    },
    addOneWorld: function(model) {
      $("#world-list").append("<li>这里是来自<b>" + model.get('name') + "</b>星球的问候：hello world！
</li>");
    }
  });
  //实例化 AppView
  var appview = new AppView;
})(jQuery);
</script>
</html>
```

我认为代码是直观的，这里面涉及到 backbone 的三个部分，view、model、collection，以后都会提到，这里只要了解，model 代表一个数据模型，collection 是模型的一个集合，而 view 是用来处理页面以及简单的页面逻辑的。

3、 backbone 中的 model 实例

作者：胡阳

本文链接:http://www.the5fire.net/3-backbone-model.html

关于 backbone，最基础的一个东西就是 model，这个东西就像是后端开发中的数据库映射那个 model 一样，也是数据对象的模型，并且应该是和后端的 model 有相同的属性（仅是需要通过前端来操作的属性）。

下面就从实例来一步一步的带大家来了解 backbone 的 model 到底是什么样的一个东西。

首先定义一个 html 的页面：

```
<!DOCTYPE html>
<html>
<head>
 <title>the5fire-backbone-model</title>
</head>
<body>
</body>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
<scriptsrc="http://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-min.js"></script>
<scriptsrc="http://ajax.cdnjs.com/ajax/libs/backbone.js/0.3.3/backbone-min.js"></script>
```

```
<script>
(function ($) {
/**
*此处填充代码
**/
})(jQuery);
</script>
</html>
```

下面的代码需要填到这个 html 的 script 标签中的 function 中。

1、最简单的一个对象

```
Man=Backbone.Model.extend({
    initialize:function(){
     alert('Hey, you create me!');
   }
 });
varman=newMan;
```

这个就很简单了，在 helloworld 里面也有了一个 model 的展现，不定义了属性，这里是一个 初始化时的方法，或者称之为构造函数

2、对象赋值的两种方法

第一种，直接定义，设置默认值。

```
Man=Backbone.Model.extend({
    initialize:function(){
     alert('Hey, you create me!');
   },
    defaults:{
     name:'张三',
      age:'38'
   }
 });

 varman=newMan;
 alert(man.get('name'));
```

第二种，赋值时定义

```
Man=Backbone.Model.extend({
    initialize:function(){
```

```
    alert('Hey, you create me!');
  }
});
 man.set({name:'the5fire',age:'10'});
alert(man.get('name'));
```

取值的时候都是用 get。

3、对象中的方法

```
Man=Backbone.Model.extend({
   initialize: function(){
     alert('Hey, you create me!');
  },
   defaults:{
     name:'张三',
     age:'38'
  },
   aboutMe: function(){
    return'我叫'+ this.get('name')+',今年'+ this.get('age')+'岁';
  }
});
 var man=newMan;
 alert(man.aboutMe());
```

4、监听对象中属性的变化

```
Man=Backbone.Model.extend({
   initialize:function(){
    alert('Hey, you create me!');
    //初始化时绑定监听
    this.bind("change:name",function(){
      varname=this.get("name");
      alert("你改变了 name 属性为："+name);
    });
  },
   defaults:{
    name:'张三',
     age:'38'
  },
   aboutMe:function(){
    return'我叫'+this.get('name')+',今年'+this.get('age')+'岁';
```

```
  }
});
varman=newMan;
 man.set({name:'the5fire'})//触发绑定的 change 事件，alert。
```

5、为对象添加验证规则，以及错误提示

```
Man=Backbone.Model.extend({
   initialize:function(){
    alert('Hey, you create me!');
    //初始化时绑定监听
    this.bind("change:name",function(){
      varname=this.get("name");
      alert("你改变了 name 属性为："+name);
    });
    this.bind("error",function(model,error){
      alert(error);
    });
   },
    defaults:{
     name:'张三',
      age:'38'
   },
    validate:function(attributes){
     if(attributes.name==''){
      return"name 不能为空！";
     }
   },
    aboutMe:function(){
     return'我叫'+this.get('name')+',今年'+this.get('age')+'岁';
   }
});
varman=newMan;
 man.set({name:''});//根据验证规则，弹出错误提示。
```

6、对象的获取和保存，需要服务器端支持才能测试。

首先需要为对象定义一个 url 属性，调用 save 方法时会 post 对象的所有属性到 server 端。

```
Man=Backbone.Model.extend({
   url:'/save/',
   initialize:function(){
```

```
    alert('Hey, you create me!');
    //初始化时绑定监听
    this.bind("change:name",function(){
      varname=this.get("name");
      alert("你改变了 name 属性为："+name);
    });
    this.bind("error",function(model,error){
      alert(error);
    });
  },
  defaults:{
    name:'张三',
     age:'38'
  },
  validate:function(attributes){
    if(attributes.name==''){
      return"name 不能为空！";
    }
  },
  aboutMe:function(){
    return'我叫'+this.get('name')+',今年'+this.get('age')+'岁';
  }
});
varman=newMan;;
 man.set({name:'the5fire'});
 man.save();//会发送 POST 到模型对应的 url，数据格式为 json{"name":"the5fire","age":38}
//然后接着就是从服务器端获取数据使用方法 fetch([options])
varman1=newMan;
//第一种情况，如果直接使用 fetch 方法，那么他会发送 get 请求到你 model 的 url 中，
  //你在服务器端可以通过判断是 get 还是 post 来进行对应的操作。
 man1.fetch();
//第二种情况，在 fetch 中加入参数，如下：
 man1.fetch({url:'/getmans/'});
  //这样，就会发送 get 请求到/getmans/这个 url 中，
  //服务器返回的结果样式应该是对应的 json 格式数据，同 save 时 POST 过去的格式。

//不过接受服务器端返回的数据方法是这样的：
 man1.fetch({url:'/getmans/',success:function(model,response){
```

```
    alert('success');
    //model为获取到的数据
    alert(model.get('name'));
  },error:function(){
    //当返回格式不正确或者是非json数据时，会执行此方法
    alert('error');
  }});
```

注：上述代码仅仅均为可正常执行的代码，不过关于服务器端的实例在后面会有。

这里还要补充一点，就是关于服务器的异步操作都是通过 Backbone.sync 这个方法来完成的，调用这个方法的时候会自动的传递一个参数过去，根据参数向服务器端发送对应的请求。比如你 save，backbone 会判断你的这个对象是不是新的，如果是新创建的则参数为 create，如果是已存在的对象只是进行了改变，那么参数就为 update，如果你调用 fetch 方法，那参数就是 read，如果是 destory，那么参数就是 delete。也就是所谓的 CRUD (“create”, “read”, “update”, or “delete”)，而这四种参数对应的请求类型为 POST，GET，PUT，DELETE。你可以在服务器根据这个 request 类型，来做出相应的 CRUD 操作。

PS：忘了解释关于 url 和 urlRoot 的事情了，如果你设置了 url，那么你的 CRUD 都会发送对应请求到这个 url 上，但是这样又一个问题，就是 delete 请求，发送了请求，但是却没有发送任何数据，那么你在服务器端就不知道应该删除哪个对象（记录），所以这里又一个 urlRoot 的概念，你设置了 urlRoot 之后，你发送 PUT 和 DELETE 请求的时候，其请求的 url 地址就是：/baseurl/[model.id]，这样你就可以在服务器端通过对 url 后面值的提取更新或者删除对应的对象（记录）。

有一个例外需要提醒下：

如果 model 和 collection 一起使用，那么你定义的在 collection 中的 url 将取代 model 所定义的 urlRoot，但是 model 中的 urlRoot 也必须存在。

关于这个 Backbone.sync 以后可能会说到，不过目前先以简单入门为主。

4、backbone 的 collection 实例

作者：胡阳

本文链接:http://www.the5fire.net/4-backbone-collection.html

collection 是 model 对象的一个有序的集合，概念理解起来十分简单，在通过几个例子来看一下，会觉得更简单。

1、关于 book 和 bookshelf 的例子

```
Book=Backbone.Model.extend({

default:{

title:'default'

},

initialize:function(){

//alert('Hey, you create me!');

}

});

BookShelf=Backbone.Collection.extend({

model:Book

});

varbook1=newBook({title:'book1'});

varbook2=newBook({title:'book2'});

varbook3=newBook({title:'book3'});

//var bookShelf = new BookShelf([book1, book2, book3]); //注意这里面是数组,或者使用 add

varbookShelf=newBookShelf;

bookShelf.add(book1);

bookShelf.add(book2);

bookShelf.add(book3);
```

```
bookShelf.remove(book3);


//基于 underscore 这个 js 库，还可以使用 each 的方法获取 collection 中的数据

bookShelf.each(function(book){

alert(book.get('title'));

});
```

很简单，不解释

2、使用 fetch 从服务器端获取数据

首先要在上面的的 Bookshelf 中定义 url，注意 collection 中并没有 urlRoot 这个属性。或者你直接在 fetch 方法中定义 url 的值，如下：

```
bookShelf.fetch({url:'/getbooks/',success:function(collection,response){

collection.each(function(book){

alert(book.get('title'));

});

},error:function(){

alert('error');

}});
```

其中也定义了两个接受返回值的方法，具体含义我想很容易理解，返回正确格式的数据，就会调用 success 方法，错误格式的数据就会调用 error 方法，当然 error 方法也看添加和 success 方法一样的形参。

对应的 BookShelf 的返回格式如下：[{'title':'book1'},{'title':'book2'}.....]

3、reset 方法

这个方法的时候是要和上面的 fetch 进行配合的，collection 在 fetch 到数据之后，会调用 reset 方法，所以你需要在 collection 中定义 reset 方法或者是绑定 reset 方法。这里使用绑定演示：

```
bookShelf.bind('reset',showAllBooks);

showAllBooks=function(){

bookShelf.each(function(book){

  //将 book 数据渲染到页面。

});

}
```

绑定的步骤要在 fetch 之前进行。

下面给出关于 collection 的完整代码，需要服务器端支持才行，服务器端的搭建在后面会写到。

```
<!DOCTYPE html>
<html>
<head>
 <title>the5fire-backbone-collection</title>
</head>
<body>
</body>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
<scriptsrc="http://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-min.js"></script>
<scriptsrc="http://ajax.cdnjs.com/ajax/libs/backbone.js/0.3.3/backbone-min.js"></script>
<script>
(function ($) {
 //collection 是一个简单的 models 的有序集合
 //1、一个简单的例子

 Book = Backbone.Model.extend({
   default : {
     title:'default'
   },
   initialize: function(){
     //alert('Hey, you create me!');
   }
 });
 BookShelf = Backbone.Collection.extend({
```

```
  model : Book
});

var book1 = new Book({title : 'book1'});
var book2 = new Book({title : 'book2'});
var book3 = new Book({title : 'book3'});

//var bookShelf = new BookShelf([book1, book2, book3]); //注意这里面是数组,或者使用 add
var bookShelf = new BookShelf;
bookShelf.add(book1);
bookShelf.add(book2);
bookShelf.add(book3);
bookShelf.remove(book3);
/*
for(var i=0; i<bookShelf.models.length; i++){
  alert(bookShelf.models[i].get('title'));
}
*/
//基于 underscore 这个 js 库，还可以使用 each 的方法获取 collection 中的数据
bookShelf.each(function(book){
  alert(book.get('title'));
});

//2、使用 fetch 从服务器端获取数据,使用 reset 渲染
bookShelf.bind('reset', showAllBooks);
bookShelf.fetch({url:'/getbooks/',success:function(collection,response){
  collection.each(function(book){
    alert(book.get('title'));
  });
},error:function(){
  alert('error');
}});
showAllBooks=function(){
  bookShelf.each(function(book){
    //将 book 数据渲染到页面。
  });
}
//上述代码仅仅均为可正常执行的代码，不过关于服务器端的实例在后面会有。
```

```
})(jQuery);
</script>
</html>
```

5、backbone 中的 Router 实例

作者：胡阳

本文链接:http://www.the5fire.net/5-backbone-router.html

关于这个 router 的使用，我现在依然是心存疑惑的。每点击一次这样的链接 action 会触发一个事件，但是 url 也会改变，这样刷性的话，岂不是会自动触发事件。或者这个东西只是用在单个页面的网站上，或者移动设备网站上，或者是我还不会用。

大概解释下 Router :
Backbone 中的 router，见名知意，router 有路由的意思，显然这里是要控制 url 的。

Backbone.Router 会把你连接中的#标签当做是 url 路径

即便我心存疑惑，依然是要写几个例子测试一下的。毕竟实践才能解惑。

1、一个简单的例子

```
varAppRouter=Backbone.Router.extend({

routes:{

"*actions":"defaultRoute"

},

defaultRoute:function(actions){

alert(actions);

}

});
```

```
varapp_router=newAppRouter;

Backbone.history.start();
```

需要通过调用 Backbone.history.start()方法来初始化这个 Router，这个 Router 的使用很像是 django 的 urlconf 文件提供的功能，如果你懂得 django 的话。

在页面上需要有这样的 a 标签：testActions

2、这个 routes 映射要怎么传参数呢

看下面例子，立马你就知道了

```
varAppRouter=Backbone.Router.extend({

 routes:{

  "/posts/:id":"getPost",

  "*actions":"defaultRoute"

 },

 getPost:function(id){

  alert(id);

 },

 defaultRoute:function(actions){

  alert(actions);

 }

});

varapp_router=newAppRouter;

Backbone.history.start();
```

对应的页面上应该有一个超链接：Post 120

从上面已经可以看到匹配#标签之后内容的方法，有两种：一种是用“:”来把#后面的对应的位置作为参数；还有一种是“*”，它可以匹配所有的 url，下面再来演练一下。

```
varAppRouter=Backbone.Router.extend({

 routes:{

  "/posts/:id":"getPost",

  "/download/*path":"downloadFile",//对应的链接为<a
href="#/download/user/images/hey.gif">download gif</a>

  "/:route/:action":"loadView", //对应的链接为<a href="#/dashboard/graph">Load Route/Action
View</a>

  "*actions":"defaultRoute"

 },

 getPost:function(id){

  alert(id);

 },

 defaultRoute:function(actions){

  alert(actions);

 },

 downloadFile:function(path){

       alert(path);// user/images/hey.gif

     },

      loadView:function(route,action){
```

```
        alert(route+"_"+action);// dashboard_graph

    }

});

varapp_router=newAppRouter;

Backbone.history.start();
```

总结，router 的使用看起来能够去除通过对 dom 节点的绑定来触发事件的那种繁琐，但唯一让我觉得不爽的就是点击之后如果再刷新，就会重新执行一遍对应的方法，因为 url 已经变了。或许这个也是可以解决的问题，只是我还没有发现。

另外，在其他的模块中（指：model,view,collection），也可以通过使用 routes:{}来根据链接触发函数。

下面给出完整的代码，注释自己去掉试验：

```
<!DOCTYPE html>
<html>
<head>
 <title>the5fire-backbone-router</title>
</head>
<body>
 <ahref="#/posts/120">Post 120</a>
<ahref="#/download/user/images/hey.gif">download gif</a>
<ahref="#/dashboard/graph">Load Route/Action View</a>
</body>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
 <scriptsrc="http://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-min.js"></script>
 <scriptsrc="http://documentcloud.github.com/backbone/backbone-min.js"></script>
 <script>
 (function ($) {
    //Backbone 中的 router，见名知意，router 有路由的意思，显然这里是要控制 url 的。
    //Backbone.Router 会把你连接中的#标签当做是 url 路径
    /**
    //1、来看一个简单的例子
   var AppRouter = Backbone.Router.extend({
     routes: {
       "*actions" : "defaultRoute"
```

```
    },
    defaultRoute : function(actions){
      alert(actions);
    }
  });

  var app_router = new AppRouter;

  Backbone.history.start();
```

//2、既然是对 url 进行匹配那么它应该不仅仅只是简单的静态匹配，应该具有传递参数的功能，所以下面再来一个动态的 router 的例子.

```
  var AppRouter = Backbone.Router.extend({
    routes: {
      "/posts/:id" : "getPost",
      "*actions" : "defaultRoute"
    },
    getPost: function(id) {
      alert(id);
    },
    defaultRoute : function(actions){
      alert(actions);
    }
  });

  var app_router = new AppRouter;

  Backbone.history.start();
  **/
```

//从上面已经可以看到匹配#标签之后内容的方法，有两种：一种是用":"来把#后面的对应的位置作为参数；还有一种是"*"，它可以匹配所有的 url，下面再来演练一下。

```
  var AppRouter = Backbone.Router.extend({
    routes: {
      "/posts/:id" : "getPost",
      "/download/*path": "downloadFile",
//上面对应的链接为<ahref="#/download/user/images/hey.gif">download gif</a>
      "/:route/:action": "loadView",
```

```
//上面对应的链接为<ahref="#/dashboard/graph">Load Route/Action View</a>
        "*actions" : "defaultRoute"
      },
      getPost: function(id) {
        alert(id);
      },
      defaultRoute : function(actions){
        alert(actions);
      },
      downloadFile: function( path ){
        alert(path); // user/images/hey.gif
      },
      loadView: function( route, action ){
        alert(route + "_" + action); // dashboard_graph
      }
    });

    var app_router = new AppRouter;

    Backbone.history.start();

  })(jQuery);
 </script>

</html>
```

6、 backbone 中的 view 实例

作者：胡阳

本文链接:http://www.the5fire.net/6-backbone-view.html

Backbone 的 view 是用来显示你的 model 中的数据到页面的，同时它也可用来监听 DOM 上的事件然后做出响应。

先要给出一个页面的大体代码，下面的所有试验代码都要放到这里面：

```
<!DOCTYPE html>
<html>
<head>
 <title>the5fire-backbone-view</title>
</head>
<body>
 <divid="search_container"></div>

 <scripttype="text/template"id="search_template">
  <label><%=search_label %></label>
  <inputtype="text"id="search_input"/>
  <inputtype="button"id="search_button"value="Search"/>
 </script>
</body>
<scriptsrc="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
<scriptsrc="http://ajax.cdnjs.com/ajax/libs/underscore.js/1.1.4/underscore-min.js"></script>
<scriptsrc="http://ajax.cdnjs.com/ajax/libs/backbone.js/0.3.3/backbone-min.js"></script>
<script>
(function ($) {
    //此处添加下面的试验代码
})(jQuery);
</script>
</html>
```

1、一个简单的 view

```
SearchView=Backbone.View.extend({

 initialize:function(){

  alert('init a SearchView');

 }

});

varsearchView=newSearchView();
```

是不是觉得很没有技术含量，所有的模块定义都一样。

2、 el 属性

这个属性用来引用 DOM 中的一些元素，每一个 Backbone 的 view 都会有这么个属性，如果没有显示声明，Backbone 会默认的构造一个，表示一个空的 div 元素

```
SearchView=Backbone.View.extend({

 initialize:function(){

  alert('init a SearchView');

 }

});

varsearchView=newSearchView({el:$("#search_container")});
```

接着来看这个 el 的应用，首先注意标签中的这个标签，这是我们定义的一个模板。

```
SearchView=Backbone.View.extend({

 initialize:function(){

  //this.render();

 },

 render:function(){

  //使用 underscore 这个库，来编译模板

  vartemplate=_.template($("#search_template").html(),{});

  //加载模板到对应的 el 属性中

  this.el.html(template);

 }

});

varsearchView=newSearchView({el:$("#search_container")});
```

```
searchView.render();//这个 reander 的方法可以放到 view 的构造函数中
```

运行页面之后，会发现 script 模板中的 html 代码已经添加到了我们定义的 div 中。

3、再来看对 DOM 中元素事件的绑定，很简单

```
SearchView=Backbone.View.extend({

 initialize:function(){

  this.render();

 },

 render:function(){

  //使用 underscore 这个库，来编译模板

  vartemplate=_.template($("#search_template").html(),{});

  //加载模板到对应的 el 属性中

  this.el.html(template);

 },

 events:{//就是在这里绑定的

  'click input[type=button]':'doSearch'//定义类型为 button 的 input 标签的点击事件，触发函数 doSearch

 },

 doSearch:function(event){

  alert("search for "+$("#search_input").val());

 }

});
```

```
varsearchView=newSearchView({el:$("#search_container")});
```

自己运行下，是不是很简单，比写$(“input[type=button]“).bind(‘click’,function(){})好看多了吧。

4、view 中的模板

如果你用过 django 模板的话，你应该会想到前面提到的模板和 django 模板是不是有同样的功能，既然是模板，那就应该能传入数据。

没错了，这个和 django 的使用一样，可以在模板中定义变量，然后通过字典的方式传递进去

注意 script 模板的变化

```
SearchView=Backbone.View.extend({

 initialize:function(){

  this.render();

 },

 render:function(){

  //使用 underscore 这个库，来编译模板

  vartemplate=_.template($("#search_template").html(),{search_label:"the5fire search"});

  //加载模板到对应的 el 属性中

  this.el.html(template);

 },

 events:{//就是在这里绑定的

  'click input[type=button]':'doSearch'//定义类型为 button 的 input 标签的点击事件，触发函数 doSearch

 },

 doSearch:function(event){
```

```
  alert("search for "+$("#search_input").val());
 }
});
varsearchView=newSearchView({el:$("#search_container")});
```

再次运行，有木有觉得这个东西对 dom 的操作还是很好玩的。别激动，再来稍微扩展一下。

对于实际应用来说，页面数据的变化需要同步到服务器端，最理想的方法，只是回传变化的数据就 ok，然后修改页面上对应的数据，而不是刷新页面。

```
SearchView=Backbone.View.extend({
 initialize:function(){
  this.render('the5fire');
 },
 render:function(search_label){
  //使用 underscore 这个库，来编译模板
  vartemplate=_.template($("#search_template").html(),{search_label:search_label});
  //加载模板到对应的 el 属性中
  this.el.html(template);
 },
 events:{//就是在这里绑定的
  'click input[type=button]':'doChange'
 },
```

```
doChange:function(event){
  //通过 model 发送数据到服务器
  this.render('the5fire'+$("#search_input").val());
}
});
varsearchView=newSearchView({el:$("#search_container")});
```

这是一个比较牵强的例子，但是如果加上 model 的使用，效果就会好很多，通过 view 和 model 可以使得业务和数据真正的分离。

总之，view 的主要应用就是绑定事件，处理业务，渲染页面。

7、backbone 实例 todos 分析（一）

作者：胡阳

本文链接:http://www.the5fire.net/7-backbone-todos-1.html

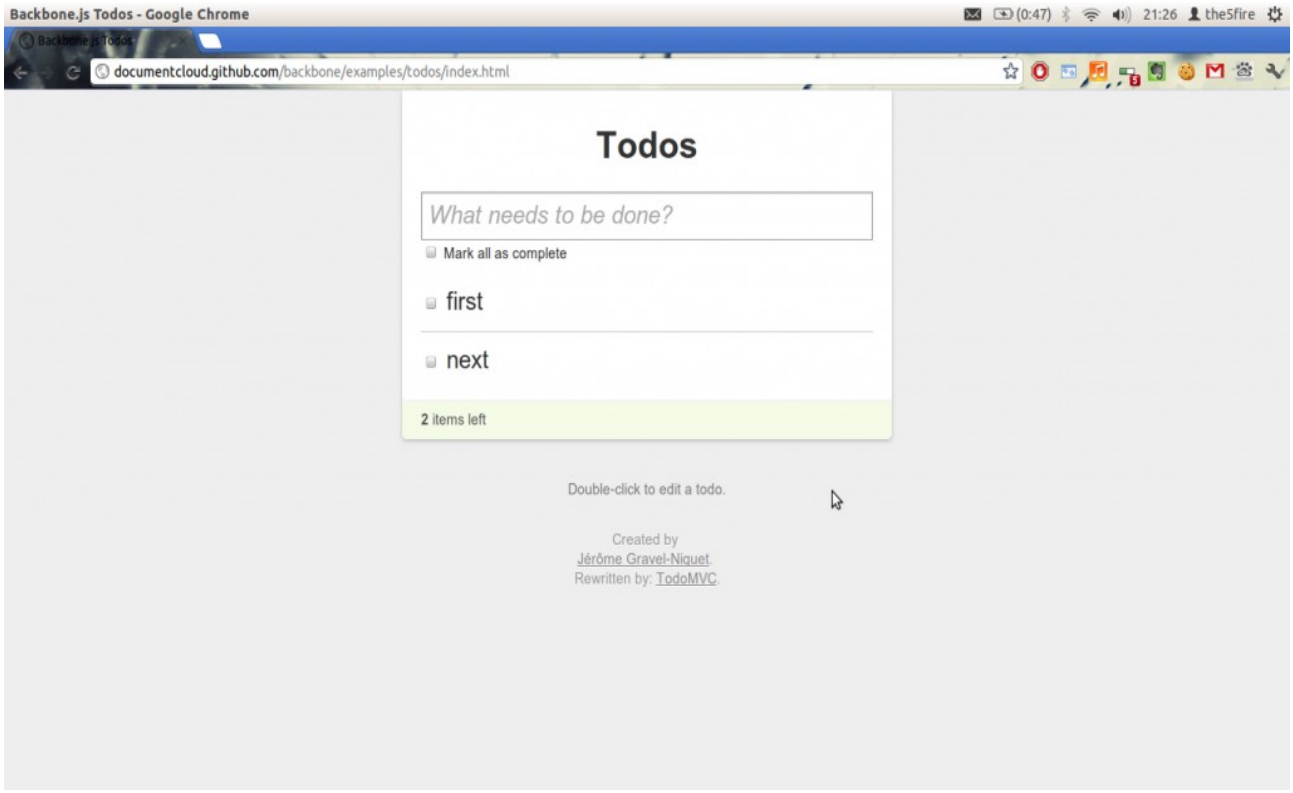
经过前面的几篇文章，backbone 中的 model，collection，router，view，都简单的讲了一下，我觉得看完这几篇文章，你应该达到的水平，或者说我要达到的目的就是：已经能够在自己的 web 项目或者是平时的练习中用的上 backbone 了。

其实对于一个 web 开发老手来说，基本上看完前面的内容，你已经可以把 backbone 的使用和自己的开发经验结合起来进行应用了，要想更进一步的话需要去看 backbone 的官方文档，或者去看官方实例。

这里我就 backbone 官网上的实例 todos 进行下分析，毕竟人家自己的东西，自己写出来应该能够把 backbone 的特性发挥的淋漓尽致，并且代码应该也是足够优秀的，不然也会放出来让大家参考。

好了，废话这么多，下面开始正题。Todos 的代码这里下载：
https://github.com/documentcloud/backbone/

首先应该来看下功能，先看截图：



从这个界面我们可以总结出来，这个 Todos 有哪些功能：

1、添加任务。

2、修改任务（包括内容，状态）。

3、删除任务。

4、任务完成情况统计。

总体上就这四项功能。

这个项目仅仅是在 web 端运行的，没有服务器进行支持，所以项目中使用了一个叫做 backbone-localstorage 的 js 库，用来把数据存储到前端。

因为 backbone 为 mvc 模式，根据对这种模式的使用经验，我们应该从分析其数据模型开始。当然，你也可以从其他地方入手。

这里我们显然是可以看到它的源代码的，所以直接来看其 model 层，：

```
/**
 *基本的 Todo 模型，属性为：content,order,done。
 **/
```

```
varTodo=Backbone.Model.extend({
// 设置默认的属性
defaults:{
content:"empty todo...",
done:false
},
//确保每一个 content 都不为空
initialize:function(){
if(!this.get("content")){
this.set({"content":this.defaults.content});
}
},
// 将一个任务的完成状态置为逆状态
toggle:function(){
this.save({done:!this.get("done")});
},

//从 localStorage 中删除一个条目
clear:function(){
```

```
this.destroy();

}

});
```

这段代码是很好理解的，不过我依然是画蛇添足的加上了一些注释。这个 Todo 显然就是对应页面上的每一个任务条目。那么显然应该有一个 collection 来统治（管理）所有的任务，所以再来看 collection：

```
/**

*Todo 的一个集合，数据通过 localStorage 存储在本地。

**/

varTodoList=Backbone.Collection.extend({

// 设置 Collection 的模型为 Todo

model:Todo,
//存储到本地，以 todos-backbone 命名的空间中
localStorage:newStore("todos-backbone"),

//获取所有已经完成的任务数组

done:function(){

returnthis.filter(function(todo){returntodo.get('done');});

},



//获取任务列表中未完成的任务数组

remaining:function(){

returnthis.without.apply(this,this.done());
```

```
},


//获得下一个任务的排序序号，通过数据库中的记录数加 1 实现。
nextOrder:function(){
if(!this.length)return 1;
returnthis.last().get('order')+1;
},

//Backbone 内置函数，根据 todo 对象的加入顺序进行排列
comparator:function(todo){
returntodo.get('order');
}
});
```

collection 的主要功能有以下几个：

1、按序存放 Todo 对象；

2、获取完成的任务数目；

3、获取未完成的任务数目；

4、获取下一个要插入数据的序号。

这里面有三个新的函数需要解释下：

第一个是 comparator，这是 backbone 的内置函数，起作用就是 collection 中数据的排序依据。文档参考这里：http://documentcloud.github.com/backbone/#Collection-comparator

第二个是获取完成任务数目时调用的 this.filter 这个函数，它是 underscore 的内置函数，作用是遍历当前对象，然后过滤出对象中指定内容为 True 的对象，并将这些对象放到数组中返回。参

考文档：http://documentcloud.github.com/underscore/#filter

第三个是获取未完成任务数据是调用的 this.without.apply(this, this.done())这个函数，without 也是 underscore 里面的函数。而后面的那个 apply 是 javascript 的内置函数，作用是把当前的上下文传入到函数中。这段代码的意思其实就是从 this（也就是 collection 中），排除已经完成的任务（this.done()）,返回数组。参考：http://stackoverflow.com/questions/9137398/backbone-js-todo-collection-what-exactly-is-happening-in-this-return-stateme

这篇文章先分析到这里，下篇文章继续分析。

8、 backbone 实例 todos 分析(二)view 的应用

作者：胡阳

本文链接:http://www.the5fire.net/8-backbone-todos-2.html

在上一篇文章中我们把 todos 这个实例的数据模型进行了简单的分析， 有关于数据模型的操作也都知道了。接着我们来看剩下的两个 view 的模型，以及它们对页面的操作。

首先要分析下，这个俩 view 是用来干嘛的。按照自己的想法，一个页面上的操作，直接用一个 view 来搞定不就行了吗，为何要用两个呢？

我觉得这就是新手和老手的主要区别之一，喜欢在一个方法里面搞定一切，随着时间的推移，再逐渐重构，让代码变得灵活可扩展。但既然我们拿到一个成熟的代码，就应该吸取其中的精华。

我觉得这里面的精华就是，将数据的展示和对数据的操作进行分离，也就是现在代码里面 TodoView 和 AppView。前者的作用是展示数据模型中的数据到界面，并对数据本身进行管理;后者是对整体的一个控制，如所有数据的显示（调用 TodoView），添加一个任务、统计多少完成任务等。

有了上面的分析，让我们来一起看下代码：

```
// 首先是创建一个全局的 Todo 的 collection 对象

varTodos=newTodoList;

// 先来看 TodoView，作用是控制任务列表
varTodoView=Backbone.View.extend({

 //下面这个标签的作用是，把 template 模板中获取到的 html 代码放到这标签中。
```

```
tagName:"li",

// 获取一个任务条目的模板
template:_.template($('#item-template').html()),

// 为每一个任务条目绑定事件
events:{
"click .check"      :"toggleDone",
"dblclick label.todo-content":"edit",
"click span.todo-destroy":"clear",
"keypress .todo-input"  :"updateOnEnter",
"blur .todo-input"    :"close"
},

//在初始化设置了 todoview 和 todo 的以一对一引用，这里我们可以把 todoview 看作是 todo 在界面的映射。
initialize:function(){
 _.bindAll(this,'render','close','remove');
this.model.bind('change',this.render);
this.model.bind('destroy',this.remove);//这个 remove 是 view 的中的方法，用来清除页面中的 dom
},

// 渲染 todo 中的数据到 item-template 中，然后返回对自己的引用 this
render:function(){
 $(this.el).html(this.template(this.model.toJSON()));
this.input=this.$('.todo-input');
returnthis;
},

// 控制任务完成或者未完成
toggleDone:function(){
this.model.toggle();
},

// 修改任务条目的样式
edit:function(){
 $(this.el).addClass("editing");
this.input.focus();
},
```

```
// 关闭编辑界面，并把修改内容同步到界面
close:function(){
 this.model.save({content:this.input.val()});//会触发 change 事件
  $(this.el).removeClass("editing");
},

// 按下回车之后，关闭编辑界面
 updateOnEnter:function(e){
 if(e.keyCode==13)this.close();
},

// 移除对应条目，以及对应的数据对象
 clear:function(){
 this.model.clear();
}
});

//再来看 AppView，功能是显示所有任务列表，显示整体的列表状态（如：完成多少，未完成多少）
//以及任务的添加。主要是整体上的一个控制
varAppView=Backbone.View.extend({

//绑定页面上主要的 DOM 节点
 el:$("#todoapp"),

// 在底部显示的统计数据模板
 statsTemplate:_.template($('#stats-template').html()),

// 绑定 dom 节点上的事件
 events:{
 "keypress #new-todo":"createOnEnter",
 "keyup #new-todo": "showTooltip",
 "click .todo-clear a":"clearCompleted",
 "click .mark-all-done":"toggleAllComplete"
},

//在初始化过程中，绑定事件到 Todos 上，当任务列表改变时会触发对应的事件。最后把存在 localStorage 中
的数据取出来。
```

```
initialize:function(){
//下面这个是 underscore 库中的方法，用来绑定方法到目前的这个对象中，是为了在以后运行环境中调用当前对象的时候能够找到对象中的这些方法。
 _.bindAll(this,'addOne','addAll','render','toggleAllComplete');
this.input=this.$("#new-todo");
this.allCheckbox=this.$(".mark-all-done")[0];
 Todos.bind('add', this.addOne);
 Todos.bind('reset',this.addAll);
 Todos.bind('all', this.render);

 Todos.fetch();
},

// 更改当前任务列表的状态
render:function(){
vardone=Todos.done().length;
varremaining=Todos.remaining().length;

this.$('#todo-stats').html(this.statsTemplate({
  total:  Todos.length,
  done:   done,
  remaining:remaining
}));

//根据剩余多少未完成确定标记全部完成的 checkbox 的显示
this.allCheckbox.checked=!remaining;
},

//添加一个任务到页面 id 为 todo-list 的 div/ul 中
addOne:function(todo){
varview=newTodoView({model:todo});
this.$("#todo-list").append(view.render().el);
},

// 把 Todos 中的所有数据渲染到页面,页面加载的时候用到
addAll:function(){
 Todos.each(this.addOne);
},
```

```
//生成一个新 Todo 的所有属性的字典
newAttributes:function(){
return{
  content:this.input.val(),
  order: Todos.nextOrder(),
  done: false
};
},

//创建一个任务的方法，使用 backbone.collection 的 create 方法。将数据保存到 localStorage,这是一个 html5 的 js 库。需要浏览器支持 html5 才能用。
createOnEnter:function(e){
if(e.keyCode!=13)return;
 Todos.create(this.newAttributes());//创建一个对象之后会在 backbone 中动态调用 Todos 的 add 方法，该方法已绑定 addOne。
this.input.val('');
},

// 去掉所有已经完成的任务
clearCompleted:function(){
 _.each(Todos.done(),function(todo){todo.clear();});
returnfalse;
},

//用户输入新任务的时候提示，延时 1 秒钟
//处理逻辑是：首先获取隐藏的提示节点的引用，然后获取用户输入的值，
//先判断是否有设置显示的延时，如果有则删除，然后再次设置，因为这个事件是按键的 keyup 时发生的，所以该方法会被连续调用。
showTooltip:function(e){
vartooltip=this.$(".ui-tooltip-top");
varval=this.input.val();
 tooltip.fadeOut();
if(this.tooltipTimeout)clearTimeout(this.tooltipTimeout);
if(val==''||val==this.input.attr('placeholder'))return;
varshow=function(){tooltip.show().fadeIn();};
this.tooltipTimeout=_.delay(show,1000);
},
```

```
//处理页面点击标记全部完成按钮
//处理逻辑：如果标记全部按钮已选，则所有都完成，如果未选，则所有的都未完成。
 toggleAllComplete:function(){
 vardone=this.allCheckbox.checked;
  Todos.each(function(todo){todo.save({'done':done});});
}
});
```

通过上面的代码，以及其中的注释，我们应该认识了其中的各个函数的作用。但是有一点没有说到的是 template 这个东西。

在前几篇的 view 介绍中我们已经认识过了简单的模板使用，以及变量参数的传递：

```
<script type="text/template"id="search_template">

  <label><%=search_label%></label>

  <input type="text"id="search_input"/>

  <input type="button"id="search_button"value="Search"/>

</script>
```

既然能定义变量，那么就能使用语法，如同 django 模板，那来看下带有语法的模板，也是上面的两个 view 用到的模板，我想这个是很好理解的。

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

 <divclass="todo <%= done ? 'done' : '' %>">

  <divclass="display">

   <inputclass="check"type="checkbox"<%=done?'checked="checked"':''%>/>

   <labelclass="todo-content"><%=content%></label>

   <spanclass="todo-destroy"></span>

  </div>

  <divclass="edit">
```

```
  <inputclass="todo-input"type="text"value="<%= content %>"/>
  </div>
 </div>
</script>

<script type="text/template"id="stats-template">
 <%if(total){%>
  <spanclass="todo-count">
   <spanclass="number"><%=remaining%></span>
   <spanclass="word"><%=remaining==1?'item':'items'%></span>left.
  </span>
 <%}%>
 <%if(done){%>
  <spanclass="todo-clear">
   <a href="#">
    Clear<spanclass="number-done"><%=done%></span>
    completed<spanclass="word-done"><%=done==1?'item':'items'%></span>
   </a>
  </span>
```

```
<%}%>

</script>
```

简单的语法，上面的那个对应 TodoView。

这一篇文章就先到此为止，文章中我们了解到在 todos 这个实例中，view 的使用，以及具体的 TodoView 和 AppView 中各个函数的作用，这意味着所有的肉和菜都已经放到你碗里了，下面就是如何吃下去的问题了。

下一篇我们一起来学习 todos 的整个流程。

9、backbone 实例 todos 分析（三）总结

作者：胡阳

本文链接:http://www.the5fire.net/9-backbone-todos-3.html

在前两篇文章中，我们已经对这个 todos 的功能、数据模型以及各个模块的实现细节进行了分析，这篇文章我们要对前面的分析进行一个整合。前面我们说过，有了肉和菜，剩下的就是要怎么吃。我个人倾向于菜和肉一起吃，这样不会觉得腻

首先让我们来回顾一下我们分析的流程：先对页面功能进行了分析，然后又分析了数据模型，最后又对 view 的功能和代码进行了详解。你是不是觉得这个分析里面少了点什么？没错了，就知道经验丰富的你已经看出来了，这里面少了对于流程的分析。

所以从我的分析中可以看的出来，我是先对各个原材料进行分析，然后再整体的分析（当然前提是我是理解流程的），这并不是分析代码的唯一方法，有时我也会采用跟着流程分析代码的方法。当然还有很多其他的分析方法，大家都是自己的套路嘛。

下面简单的说说流程分析的方法。记得多年前在学 vb 的时候，分析一个完整项目代码的时候，习惯从程序的入口点开始分析。虽然 web 网站和桌面软件的实现不同，但是大致思路是一样的（同时也有网站即软件的说法，在 RESTful 架构中）。所以我们要先找到网站的入口点所在。

和桌面应用项目的分析一样，网站的入口点就在于网页加载的时候。对于 todos，自然就是加载所有的任务。所以对应着我们就可以发现这段代码

首先是对 AppView 的一个实例化：

```
var App=newAppView;
```

实例化，自然就会调用构造函数：

//在初始化过程中，绑定事件到 Todos 上，当任务列表改变时会触发对应的事件。最后把存在 localStorage 中的数据取出来。

```
initialize: function(){
 //下面这个是 underscore 库中的方法，用来绑定方法到目前的这个对象中，是为了在以后运行环境中调用当前对象的时候能够找到对象中的这些方法。
  _.bindAll(this,'addOne','addAll','render','toggleAllComplete');
  this.input=this.$("#new-todo");
  this.allCheckbox=this.$(".mark-all-done")[0];
  Todos.bind('add',  this.addOne);
  Todos.bind('reset', this.addAll);
  Todos.bind('all',  this.render);

  Todos.fetch();
},
```

注意其中的 Todos.fetch()方法，前面说过，这个项目是在客户端保存数据，所以使用 fetch 方法并不会发送请求到服务器。另外在前面关于 collection 的单独讲解中我们也知道了 collection 中调用 fetch 方法之后就会触发 reset 这个方法。所以现在流程走向 reset——>addAll 这个方法。

来看 addAll 这个方法：

//添加一个任务到页面 id 为 todo-list 的 div/ul 中

```
addOne: function(todo){
 var view=newTodoView({model: todo});
 this.$("#todo-list").append(view.render().el);
},

// 把 Todos 中的所有数据渲染到页面,页面加载的时候用到
addAll: function(){
 Todos.each(this.addOne);
},
```

在 addAll 中调用 addOne 方法，关于 Todos.each 很好理解，就是语法糖（简化的 for 循环），到此，加载页面的整个流程也就完成了。关于 addOne 方法的细节下面介绍。

然后再来看添加任务的流程，一个良好的代码命名风格始终是让人满心欢喜的。因为很显然，添加一个任务，自然就是 addOne,其实你看 events 中的绑定也能知道，先看一下绑定：

// 绑定 dom 节点上的事件

```
events:{
"keypress #new-todo": "createOnEnter",
```

```
 "keyup #new-todo":  "showTooltip",
 "click .todo-clear a":"clearCompleted",
 "click .mark-all-done":"toggleAllComplete"
},
```

这里并没有 addOne 方法的绑定，但是却有 createOnEnter，语意其实一样的，另外这里其实要说下关于 showTooltip 这个方法，在任务输入框中，按键弹起的时候执行这个方法，具体代码有详细的注释了，这里不多介绍。

来看主线，createOnEnter 这个方法：

//创建一个任务的方法，使用 backbone.collection 的 create 方法。将数据保存到 localStorage,这是一个 html5 的 js 库。需要浏览器支持 html5 才能用。

```
 createOnEnter: function(e){
 if(e.keyCode!=13)return;
  Todos.create(this.newAttributes());//创建一个对象之后会在 backbone 中动态调用 Todos 的 add 方法，该方法已绑定 addOne。
  this.input.val('');
},
```

注释已写明，Todos.create 会调用 addOne 这个方法。由此顺理成章的来到 addOne 里面：

```
//添加一个任务到页面 id 为 todo-list 的 div/ul 中
 addOne: function(todo){
  var view=newTodoView({model: todo});
  this.$("#todo-list").append(view.render().el);
},
```

在里面实例化了一个 TodoView 类，前面我们说过，这个类是主管各个任务的显示的。具体代码就不细说了。

有了添加再来看更新，关于单个任务的操作，我们直接找 TodoView 就 ok 了。所以直接找到

```
// The DOM events specific to an item.
 events:{
 "click .check"      :"toggleDone",
 "dblclick label.todo-content":"edit",
 "click span.todo-destroy" :"clear",
 "keypress .todo-input"  :"updateOnEnter",
 "blur .todo-input"    :"close"
},
```

其中的 edit 事件的绑定就是更新的一个开头，而 updateOnEnter 就是更新的具体动作。所以只要搞清楚这俩方法的作用一切就明了了。这里同样不用细说。

在往后还有删除一条记录以及清楚已有记录的功能，根据上面的分析过程，我想大家都很容易的去'顺藤模瓜'。

关于 Todos 的分析到此就算完成了，我注释过的整个代码在 github 上：https://github.com/the5fire/the5fire-todos，供大家参考。

在下一篇文章中我们将一起来学习通过 django 来搭建 web 服务器，以及简单的数据库。

10、django 开发环境搭建及使用

作者：胡阳

本文链接:http://www.the5fire.net/10-django-dev-env.html

django 是基于 python 的一个框架，因此在此之前要先安装 python 环境，关于 python 环境的搭建，前面已写过，参考这里：http://www.the5fire.net/python-env.html。有了 python 环境接着就要开始安装 django 了。

在正式开始之前要先介绍一下 django 的开发环境包括哪些东西：

- 1、django 的安装
- 2、mysql 安装【非必须，本篇会提及】
- 3、apache 安装【非必须】

另外主要是为了和大家一起使用 django 来作为 todos 的服务器端，因此还简单的介绍下如何使用 django 来开发。

正式开始，安装 django 可谓相当简单，到这：http://www.djangoproject.com/download/1.3.1/tarball/ 我自己用的 django 版本是 1.3.1 的，大家可以自行选择。

下载下来之后，通过命令行进入 Django-1.3.1 目录中，执行：python setup.py install 即可（对于 ubuntu 用户可能需要加 sudo）。在命令行中输入 django-admin 看有木有提示，如果没有你需要到你对应的 C:/python2.7/Scripts 中看看有没有一个 django-admin.py 文件，如果有， 你需要把这个目录放到系统环境变量中。

再来看 mysql 的安装，【对于 todos 这个项目不会用到 mysql】

对于 windows 用户安装相对简单些，先现在 mysql 数据库安装包，然后下载对应的 MySQL-python-1.2.3.win32-py2.7 安装文件，先安装 mysql，然后安装对应的 mysql-python 文件。

对于 ubuntu 用户需要以下操作：

1、安装 mysql
sudo apt-get install mysql-server

sudo apt-get install libmysqld-dev
sudo apt-get instlal libmysqlclient-dev

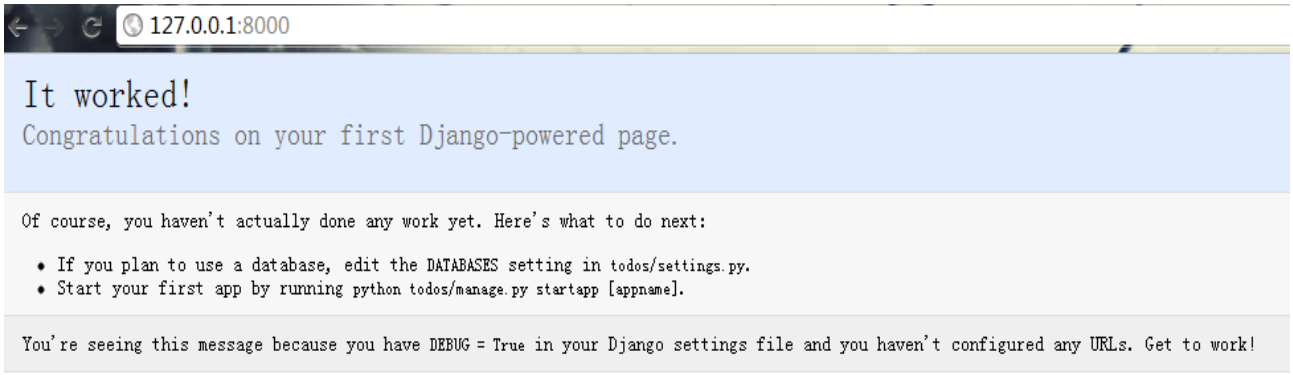
sudo apt-get install libmysqld-dev
sudo apt-get install libmysqlclient-dev
sudo apt-get install libmysqlclient-dev

2、配置 mysql 和 python
sudo apt-get install python-mysqldb

也可以参考我以前的一篇文章：http://www.the5fire.net/python-opt-mysql.html

至此，环境的搭建就完成了，简单的运用一下。

首先随便找一个地方创建一个应用，命令：django-admin.py startproject todos，然后你会发现已经有了一个 todos 文件夹，在你执行命令的文件中。这就是一个 todos 工程了， 切换到 todos 中，执行 python manage.py runserver，命令行会输出 django 版本，以及服务器 url。你在浏览器中访问就看到了。如下页面：



这篇文章大概介绍了 django 的安装和 mysql 的安装，以及 django 的简单使用。下一篇就开始把前面分析过的 todos 修改为一个需要同后台 web 服务器交互的一个程序。

11、 backbone 实例 todos 扩展+web 服务器

作者：胡阳

本文链接:http://www.the5fire.net/11-backbone-todos-djangowebserver.html

在第 7 节的时候，我们对 backbone 的功能进行了分析，建立了 web 端的 model。在本节中我们将对原先的 todos 进行扩展，使其能够将数据存到 server 端的数据库中。这里我们使用的是 django+sqlite 来进行实现。

现在我们应该对应着建立 server 端的 model。不过在此之前，为了方便不熟悉 django 的童鞋，简单的写下开发过程：

1、创建工程

根据上一篇中介绍的 django 的环境安装和使用，创建一个工程:django-admin.py startproject todos，然后在 cd 到 todos 文件夹中：python manage.py startapp todo，创建一个应用（称作模块也行）。

2、配置文件

在 todos 根目录的 settings 中，主要是数据配置：

```
DATABASES = {

‘default’: {
‘ENGINE’: ‘django.db.backends.sqlite3’, # Add ‘postgresql_psycopg2’, ‘postgresql’,
‘mysql’, ‘sqlite3’ or ‘oracle’.
‘NAME’: ‘D:/mytodos’, # Or path to database file if using sqlite3.
‘USER’: ”, # Not used with sqlite3.
‘PASSWORD’: ”, # Not used with sqlite3.
‘HOST’: ”, # Set to empty string for localhost. Not used with sqlite3.
‘PORT’: ”, # Set to empty string for default. Not used with sqlite3.
}
}
```

完整的配置最后贴出来供大家参考。

有了上面的一个铺垫，开始创建 model。

打开 todo 文件夹中的 models.py 文件，写入以下代码：

```
fromdjango.dbimportmodels


classTodo(models.Model):
    content=models.CharField(max_length=128)
```

```
    done=models.CharField(max_length=1,default='N')#Y表示完成 N表示未完成
    order=models.IntegerField(blank=True)
```

然后再来创建 views 代码，关于 django 的 mvc 模式这里不介绍，大家跟着操作进行。在 todo 下新建一个 views_todos.py 文件。

这个 views_todos 文件是用来操作数据库的所有代码所在。关于数据库的操作，**其实就是 CRUD（create 增加，request 查询，update 更新，delete 删除）**，在 django 的基础上，很好写。

这里是全部代码：

```
#coding=utf-8

'''
  author:huyang
  date: 2012-3-26
  blog:http://the5fire.net
'''
frommodelsimportTodo
fromdjango.httpimportHttpResponse
fromdjango.shortcutsimportrender_to_response
fromdjango.utilsimportsimplejson

'''
public
@desc 加载 todo 首页
@param
@return templates
'''
defindex(request):
 returnrender_to_response('todo/todos.html',{})

'''
public
@desc 控制创建和读取方法的一个跳转
@param
@return
'''
defcontrol_cr(request):
 ifrequest.method=='POST':
```

```
    returncreate(request)
  elifrequest.method=='GET':
    returngetAll(request)
  else:
    returnHttpResponse('
access deny

')

'''
public
@desc 控制更新和删除方法的一个跳转
@param url 中的 todo 对象 id
@return
'''
defcontrol_ud(request,todo_id):
  ifrequest.method=='PUT':
    returnupdate(request,todo_id)
  elifrequest.method=='DELETE':
    returndelete(request,todo_id)
  else:
    returnHttpResponse('
access deny

')
'''
protect
@desc 获取所有的 todo 对象，并转为 json 格式，返回
@param
@return json 格式的 todo 列表
'''
defgetAll(request):
  todos=Todo.objects.all()
  todo_dict=[]
  flag_dict={'Y':True,'N':False}
  fortodointodos:
    todo_dict.append({'id':todo.id,'content':todo.content,'done':flag_dict[todo.done],'order':todo.order})
  returnHttpResponse(simplejson.dumps(todo_dict),mimetype='application/json')
```

```
'''
protect
@desc 创建一个 todo 记录
@param POST 中的 json 格式 todo 对象
@return json 格式{'success':True/False}
'''
defcreate(request):
 req=simplejson.loads(request.raw_post_data)
 content=req['content']
 order=req['order']

 ifnotcontent:
  returnHttpResponse(simplejson.dumps({'success':False}),mimetype='application/json')
 todo=Todo()
 todo.content=content
 todo.order=order
 todo.save()
 returnHttpResponse(simplejson.dumps({'success':True}),mimetype='application/json')

'''
protect
@desc 更新一条 todo 记录
@param POST 中的 json 格式 todo 对象
@return json 格式{'success':True/False}
'''
defupdate(request,todo_id):
 req=simplejson.loads(request.raw_post_data)
 content=req['content']
 done=req['done']
 order=req['order']
 flag_dict={True:'Y',False:'N'}
 todo=Todo.objects.get(id=todo_id)
 todo.content=content
 todo.done=flag_dict[done]
 todo.order=order
 todo.save()
 returnHttpResponse(simplejson.dumps({'success':True}),mimetype='application/json')
```

```
'''
protect
@desc 删除一条 todo 记录
@param url 中的 todo 对象 id
@return json 格式{'success':True/False}
'''
defdelete(request,todo_id):
 Todo.objects.get(id=todo_id).delete()
returnHttpResponse(simplejson.dumps({'success':True}),mimetype='application/json')
```

上面的代码中除了有 CRUD 代码之后，还有两个重要的函数：control_cr 和 control_ud，从名字很容易看出来，前者是控制**创建**和**查询**的，后者是控制**更新**和**删除**的。为什么这么写呢，其原因在于使用 backbone 在 web 端进行 CRUD 操作的时候，对应的 url 并不一样，因此我写了两个函数。

在 control_cr 中，根据 GET 和 POST 来判断是查询还是创建爱你，在 control_ud 中，根据 PUST 和 DELETE 来判断是更新还是删除。

上面代码中其他函数就不详解了，都是很简单的语句。

然后我们需要做的就是配置 url，在 todos 下面的那个 urls.py 文件中的配置如下：

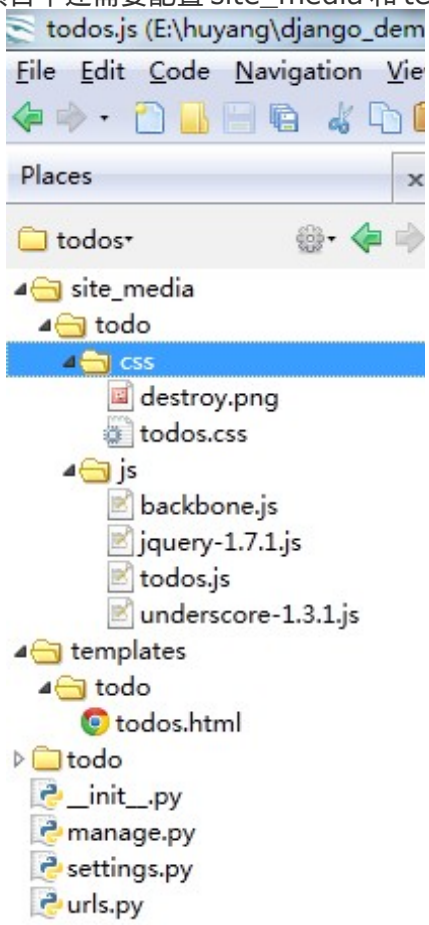
```
fromdjango.conf.urls.defaultsimportpatterns,include,url
importsettings

fromtodoimportviews_todos

urlpatterns=patterns('',

(r'^site_media/(?P.*)$','django.views.static.serve',{'document_root':
settings.STATIC_DOC_ROOT,'show_indexes':False}),

(r'^todo/control/$',views_todos.control_cr),
(r'^todo/control/(\d*)$',views_todos.control_ud),
#例如：http://127.0.0.1:8000/todo/control/1/ PUT 就是更新，DELETE 就是删除
(r'^',views_todos.index),
)
```

当然 web 端我们直接使用前面分析过的 todos 的，只需要修改一下其中的代码。

- 1、在 Todo 的模型中加入： urlRoot: '/todo/control/'

•2、在 collection TodoList 中加入：url: '/todo/control/'，并且去掉：localStorage: new Store("todos-backbone"),

这样就 ok 了。在 django 项目中还需要配置 site_media 和 templates 文件，结构如下：



我用的 Komodo Edit 这个 IDE 来开发的。你只要按照这样的结构来建立文件和文件夹就行了。

最后给出 settings 的所有代码：

```
# Django settings for testbackbone project.


DEBUG=True
TEMPLATE_DEBUG=DEBUG

ADMINS=(
 # ('Your Name', 'your_email@example.com'),
)

MANAGERS=ADMINS
```

```
DATABASES={
 'default':{
  'ENGINE':'django.db.backends.sqlite3',# Add 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or
'oracle'.
  'NAME':'D:/mytodos',# Or path to database file if using sqlite3.
  'USER':'',# Not used with sqlite3.
  'PASSWORD':'',# Not used with sqlite3.
  'HOST':'',# Set to empty string for localhost. Not used with sqlite3.
  'PORT':'',# Set to empty string for default. Not used with sqlite3.
 }
}

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# On Unix systems, a value of None will cause Django to use the same
# timezone as the operating system.
# If running in a Windows environment this must be set to the same as your
# system time zone.
TIME_ZONE='America/Chicago'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE='en-us'

SITE_ID=1

# If you set this to False, Django will make some optimizations so as not
# to load the internationalization machinery.
USE_I18N=True

# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale
USE_L10N=True

# Absolute filesystem path to the directory that will hold user-uploaded files.
# Example: "/home/media/media.lawrence.com/media/"
```

```
MEDIA_ROOT=''

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash.
# Examples: "http://media.lawrence.com/media/", "http://example.com/media/"
MEDIA_URL=''

# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/home/media/media.lawrence.com/static/"
STATIC_ROOT='./site_media/'

# URL prefix for static files.
# Example: "http://media.lawrence.com/static/"
STATIC_URL='/site_media/'


# URL prefix for admin static files -- CSS, JavaScript and images.
# Make sure to use a trailing slash.
# Examples: "http://foo.com/static/admin/", "/static/admin/".
ADMIN_MEDIA_PREFIX='/static/admin/'

# Additional locations of static files
STATICFILES_DIRS=(
 # Put strings here, like "/home/html/static" or "C:/www/django/static".
 # Always use forward slashes, even on Windows.
 # Don't forget to use absolute paths, not relative paths.
)

# List of finder classes that know how to find static files in
# various locations.
STATICFILES_FINDERS=(
 'django.contrib.staticfiles.finders.FileSystemFinder',
 'django.contrib.staticfiles.finders.AppDirectoriesFinder',
# 'django.contrib.staticfiles.finders.DefaultStorageFinder',
)
```

```
# Make this unique, and don't share it with anybody.
SECRET_KEY='q4%c$1t0@x0iaco8!8eacy5-g8t)z1549$s4049xf^2y2#!0ef'

# List of callables that know how to import templates from various sources.
TEMPLATE_LOADERS=(
 'django.template.loaders.filesystem.Loader',
 'django.template.loaders.app_directories.Loader',
 #  'django.template.loaders.eggs.Loader',
)

MIDDLEWARE_CLASSES=(
 'django.middleware.common.CommonMiddleware',
 'django.contrib.sessions.middleware.SessionMiddleware',
 #'django.middleware.csrf.CsrfViewMiddleware',
 'django.contrib.auth.middleware.AuthenticationMiddleware',
 'django.contrib.messages.middleware.MessageMiddleware',
)

ROOT_URLCONF='todos.urls'
importos
TEMPLATE_DIRS=(
 # Put strings here, like "/home/html/django_templates" or "C:/www/django/templates".
 # Always use forward slashes, even on Windows.
 # Don't forget to use absolute paths, not relative paths.
  os.path.join(os.path.dirname(__file__),'templates').replace('\','/'),
)

INSTALLED_APPS = (
  #'django.contrib.auth',
  #'django.contrib.contenttypes',
  #'django.contrib.sessions',
  ##'django.contrib.sites',
  #'django.contrib.messages',
  #'django.contrib.staticfiles',
  'todos.todo',
  # Uncomment the next line to enable the admin:
  #  'django.contrib.admin',
  # Uncomment the next line to enable admin documentation:
```

```
    # 'django.contrib.admindocs',
)

STATIC_DOC_ROOT = './site_media'

# A sample logging configuration. The only tangible logging
# performed by this configuration is to send an email to
# the site admins on every HTTP 500 error.
# See http://docs.djangoproject.com/en/dev/topics/logging for
# more details on how to customize your logging configuration.
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
    }
}
```

代码已经放到 github 上了，建议大家下载运行参考。
https://github.com/the5fire/the5fire-servertodos

12、backbone 实战：webchat（一）功能分析

作者：胡阳

本文链接:http://www.the5fire.net/12-backbone-webchat-1.html

在上一节中我们通过 django 搭建了 webserver 端，但是那毕竟是基于已有的 todos 代码来

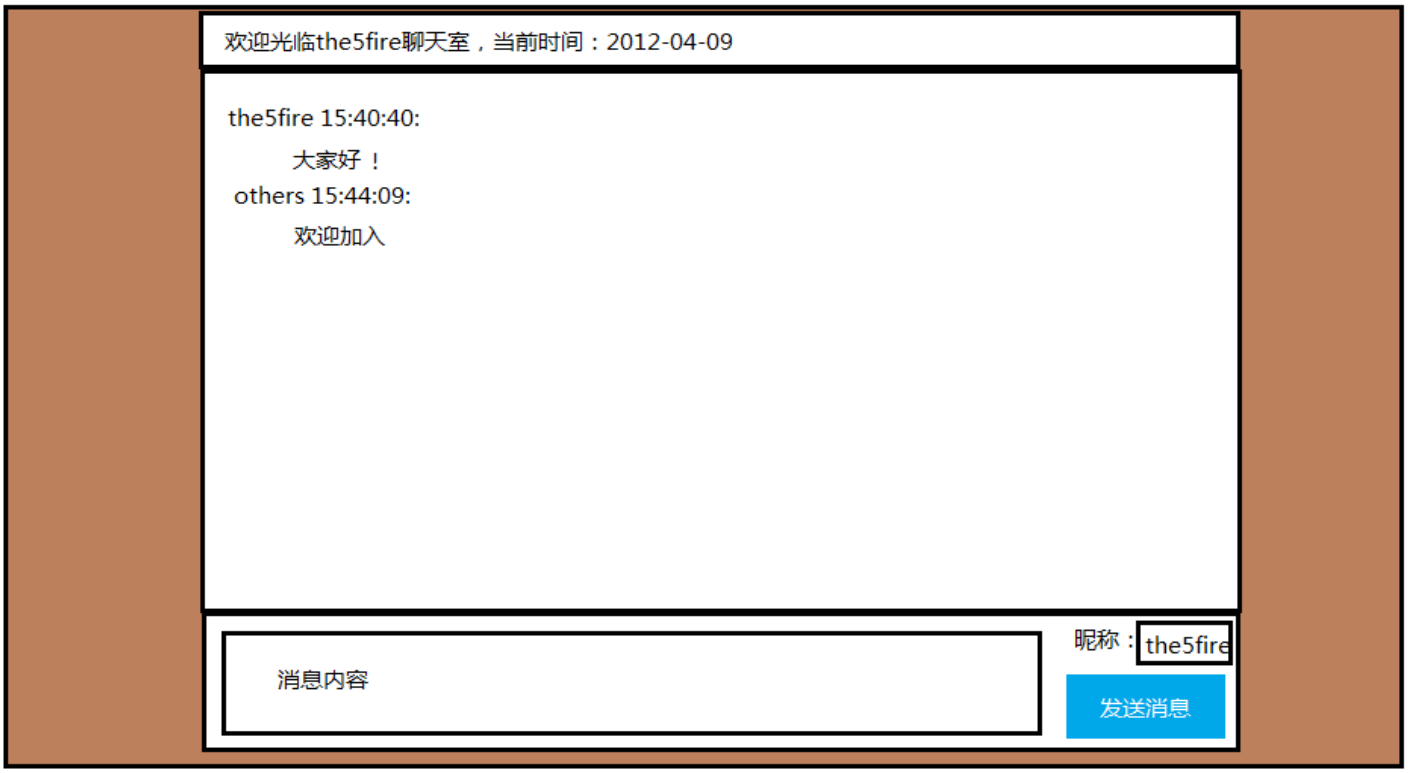
做的，总会觉得少了点什么。现在最后来从头开始做一个完整的实例，来体验一把 backbone 在开发过程中的使用。

这个实战项目我把它叫做 webchat（web 在线聊天室），使用技术就是 backbone+django+sqlite。在功能方面没有想的特别复杂，因为项目的目的就是让大家能够快速的用上 backbone。（不过经过前面的文章，我想应该已经能让你用上 backbone 了）

大概说下这个聊天室的功能，很简单，不用注册登录：

- 1、查看所有聊天记录
- 2、说话

功能有了，就来设计页面，用画图工具简单的设计一下：



页面也很简单，下一节我们来进行详细设计以及实现这个页面。

13、 backbone 实战： webchat（二）详细设计

作者：胡阳

本文链接:http://www.the5fire.net/13-backbone-webchat2.html

由上一篇的功能，我们可以概括出需要的实体类，模型如下：

```
        chat:
                id #主键
                content #消息
                username #昵称
                date #消息发送时间
```

这样的一个模型基本上已经满足了功能上的需求了。

再由上一篇中的那个页面设计，我们也进行了实现，代码就不解释了。

先是 html：

```
<!DOCTYPE html>
<html>
<head>
 <title>the5fire-WebChat</title>
 <linkhref="/site_media/chat/css/chat.css"media="all"rel="stylesheet"type="text/css"/>
 <scriptsrc="/site_media/chat/js/jquery-1.7.1.js"></script>
 <scriptsrc="/site_media/chat/js/underscore-1.3.1.js"></script>
 <scriptsrc="/site_media/chat/js/backbone.js"></script>
 <scriptsrc="/site_media/chat/js/chat.js"></script>
</head>
<body>
 <divclass="wrap">
  <divclass="main">
   <divclass="head">
    <span>欢迎光临 the5fire 聊天室，当前时间：<labelid="nowdate"></label></span>
   </div>
   <divclass="screen">
    <ulclass="chat_list">
     <li><divclass="msgtitle">the5fire 2012-04-10 23:16:00</div><p>大家好！</p></li>
     <li><divclass="msgtitle">other 2012-04-10 23:16:00</div><p>你好</p></li>
    </ul>
   </div>
   <divclass="send_message">
    <divclass="message">
     <textareaid="content"rows="4"></textarea>
    </div>
    <divclass="opt">
     <labelfor="nickname">昵称：</label><inputname="nickname"id="nickname"/><br/>
     <buttonid="send">发送消息</button>
```

```
    </div>
   </div>
  </div>
 </div>
</body>
<script>
 function show_time()
 {
   var today,hour,second,minute,year,month,date,time;

   today=new Date();

   year = today.getFullYear();
   month = today.getMonth()+1;
   date = today.getDate();
   hour = today.getHours();
   minute =today.getMinutes();
   second = today.getSeconds();
   if(minute<10)minute='0'+ minute;
  if(second <10)second='0'+ second;
  time=year +"-"+ month +"-"+ date +" "+ hour +":"+ minute +":"+ second;
  $("#nowdate").html(time);
 }
 setInterval(show_time,1000);
</script>
</html>
```

然后在是 CSS 代码：

```
/*
 author:the5fire
 blog:http://www.the5fire.net
 date:2012-04-09
*/
html{
 margin:0;
 padding:0;
}
body{
 margin:0;
```

```
  font-size:14px;
}
.wrap{
  background-color:#B26F4C;
  width:100%;
  height: 800px;
}


.main{
  width:50%;
  margin: auto;
  height: 700px;
  background-color:#fff;
}

.head{
  height: 40px;
  padding-top: 10px;
  border-bottom: 1px solid #000;
  font-size:20px;
}

.headspan{
  margin: auto;
  width: auto;
}

.screen{
  height:400px;
  width:auto;
  overflow-y: scroll;
  background:#CCCCCC;
  border: 2px solid #000;
}

.screen.msgtitle{
  color:blue;
```

```
}

.send_message{
 margin-top: 5px;
}

.send_message.message{
 width:60%;
float:left;
}

.send_message.messagetextarea{
 width:100%;
}

.send_message.opt{
 margin-right:10px;
 margin-top:10px;
float:right;
}
```

来看下界面：

界面设计和模型都有了，那么后台应该有哪些接口呢？

从功能上看也是很简单，只有两个：

- 1、说话（say），在此方法中，讲用户输入的内容保存到数据库。
- 2、获取所有聊天记录（getChatLog）,将数据库的内容全部提取出来。

这一节就到这里，下一节具体实现。

14、backbone 实战： webchat（三）web 端开发

作者：胡阳

本文链接:http://www.the5fire.net/14-backbone-webchat3.html

有了前面功能介绍以及整体详细设计 ，下面的开发就变得更有目的性了。

沿着上一篇文章的思路，我们先来把javascript 模板建立起来，模板用来取代上一篇中 html 代码里的：

```
<li>
<divclass="msgtitle">the5fire 2012-04-10 23:16:00</div>
<p>大家好！</p>
</li>
```

把它改成模板为：

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

<divclass="msgtitle">
 <%=username%><%=date%><a id="destroy">删除</a>
</div>
<p><%=content%></p>

</script>
```

其实模板的作用就是复用，里面多了一个删除的连接，主要是为了演示 backbone 的 DELETE 操作。

模板建立很容易，下面来建立页面端的实体类，这个更容易，因为上篇文章已经分析好了：

```
varChat=Backbone.Model.extend({

  urlRoot:'',

  defualts:{
    content:'',
    username:'',
    date:''
  },

  clear:function(){
   this.destroy();
  }
});
```

没有看到我上一篇插曲文章的同学可能觉得奇怪，为什么 urlRoot 为空？这里再次重复一下，当 model 和 collection 一起使用的时候，或者更确切的说是一个 model 属于某一个 collection

时，collection 的 url 将取代 mode 的 urlRoot，但是你的 urlRoot 还必须存在。

顺着思路，在来看 collection，其实简单的很，因为我这里的 collection 没有太多的动作要做：

```
varChatList=Backbone.Collection.extend({

  url:'/chat/',

  model:Chat

});
```

仅此而已，是不是很简单。

然后同以前我们分析的 todos 一样，我们也来建立一个管理单个 chat 界面的类,学以致用，就是**模仿–使用–发挥**：

```
varChatView=Backbone.View.extend({

  tagName:'li',

  template:_.template($('#item-template').html()),
  events:{
   'click #destroy':'clear'
  },

  initialize:function(){
    _.bindAll(this,'render','remove');
   this.model.bind('change',this.render);
   this.model.bind('destroy',this.remove);
  },

  render:function(){
    $(this.el).html(this.template(this.model.toJSON()));
   returnthis;
  },

  clear:function(){
   this.model.clear();
```

```
  }
});
```

代码不肖多说。

然后对应着，也要有一个整体的管理 view：

```
varAppView=Backbone.View.extend({
  el:$('.main'),

  events:{
   "click #send":"say"
  },

  initialize:function(){
    _.bindAll(this,'addOne','addAll');
   this.nickname=this.$('#nickname');
   this.textarea=this.$("#content");

    chatList.bind('add',this.addOne);
    chatList.bind('reset',this.addAll);
    chatList.fetch();
    setInterval(function(){
      chatList.fetch({add:true});
    },5000);
  },

  addOne:function(chat){
   //页面所有的数据都来源于 server端，如果不是 server端的数据，不应添加到页面上
   if(!chat.isNew()){
     varview=newChatView({model:chat});
     this.$(".chat_list").append(view.render().el);
      $('#screen').scrollTop($(".chat_list").height()+200);
    }
  },

  addAll:function(){
    chatList.each(this.addOne);
  },
```

```
  say:function(event){
    chatList.create(this.newAttributes());
    //为了满足 IE 和 FF 以及 chrome
    this.textarea.text('');
    this.textarea.val('');
    this.textarea.html('');

  },


  newAttributes:function(){

    varcontent=this.textarea.val();
    if(content==''){
      content=this.textarea.text();
    }

    return{
      content:content,
      username:this.nickname.val(),
      date:get_time()
    };
  }
});
```

其中有两个地方需要注意：

•1、 $('#screen').scrollTop($(".chat_list").height() + 200); 这个是为了让那个显示聊天信息的窗口滚动条始终处于最下方。

•2、

```
setInterval(function() {
chatList.fetch({add: true});

}, 5000);
```

这个的意思就是，每隔 5 秒就到到服务器取一下数据，里面的 add：true 参数表示，每次取回数据之后都在原有数据上累加。

剩下需要说的就是，不用忘了初始化 AppView，以及在 ChatView 定义的上方，实例化 ChatList：

```
varchatList=newChatList;

//ChatView定义上方

varappView=newAppView;
```

到这里 web 端的代码就构建完毕了，从上面的实现可以发现，web 端和 server 端的交互全部通过 collection 中定义的 url:'/chat/'来完成的。所有的 CRUD 操作通过 POST，GET，PUT，DELETE 来完成。

这篇文章就到此为止，有何疑问敬请留言。下一篇来构建服务器端。

15、 backbone 实战： webchat（四）server 端开发

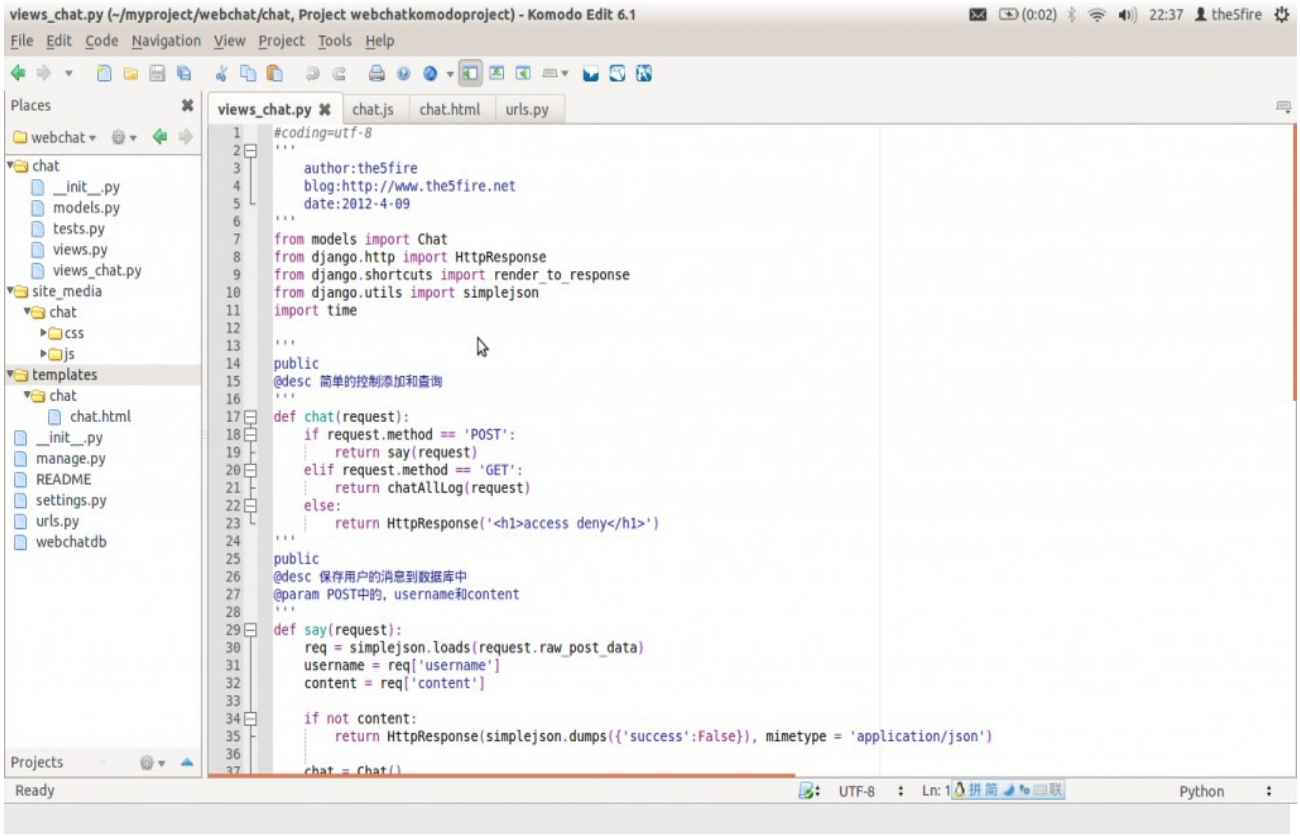
作者：胡阳

本文链接:http://www.the5fire.net/15-backbone-webchat4.html

上一篇把 web 端构建了一下，这一篇来构建服务器端。

关于 django 开发应用，相比大家都已经熟悉了，不熟悉的可以移步到这里：django 使用

webchat 的整体目录结构还是同 todos 一样，有图有真相：



先来配置数据库连接：

```
DATABASES={

'default':{

'ENGINE':'django.db.backends.sqlite3',# Add 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or 'oracle'.

'NAME':'webchatdb',        # Or path to database file if using sqlite3.

'USER':'',        # Not used with sqlite3.

'PASSWORD':'',        # Not used with sqlite3.

'HOST':'',        # Set to empty string for localhost. Not used with sqlite3.

'PORT':'',        # Set to empty string for default. Not used with sqlite3.

}
```

}

当然，像扩展 todos 时一样，settings.py 文件中还是有很多东西需要配置的。建议有兴趣的参考我放到 git 上的代码，最后给出链接。

有一点要提醒的是，因为这次用到了 session 的操作，所以在 INSTALL_APP 配置如下：

```
INSTALLED_APPS=(

'django.contrib.sessions',

'webchat.chat',

)
```

然后再来配置 urls.py，这个文件最先配置和最后配置都可以，其实我倒是觉得这个 urls 可以当作一个详细设计来用，定义好每一个后台需要提供函数，等后台来实现就行。不过很多时候并不是一开始就能很明确所需的功能的，就是想是我做这个 webchat 一样，尽管先前分析并扩展了 todos。

来看 urls.py 的配置：

```
#encoding=utf-8
'''
  author:the5fire
  blog:http://www.the5fire.net
  date:2012-4-6
'''
fromdjango.conf.urls.defaultsimportpatterns,include,url
fromdjango.views.generic.simpleimportdirect_to_template
importsettings
fromwebchat.chatimportviews_chat

urlpatterns=patterns('',
  (r'^site_media/(?P.*)$','django.views.static.serve',{'document_root':
settings.STATIC_DOC_ROOT,'show_indexes':False}),
  (r'^chat/$',views_chat.chat),
  (r'^chat/(\w+)$',views_chat.chatDelete),
  (r'^$',views_chat.loadpage),
)
)
```

两个主要的 url：chat 以及 chat/（\w+）,前者是让 CR 来访问的，后者是一个正则的 url，表示为 chat/[大于 1 位的任意字符]，用来让 UD 访问，因为这俩都是需要传 id 的。

urls.py 就是一个实现的指导，那么有了指导，来看对应实现吧，views_chat.py：

```
#coding=utf-8
'''
  author:the5fire
  blog:http://www.the5fire.net
  date:2012-4-09
'''
frommodelsimportChat
fromdjango.httpimportHttpResponse
fromdjango.shortcutsimportrender_to_response
fromdjango.utilsimportsimplejson
importtime

'''
public
@desc 页面载入或者刷新的时候，重置记录指针为 0
@return
'''

defloadpage(request):

 request.session['record_offset']=0
 returnrender_to_response('chat/chat.html',{})


'''

public

@desc 简单的控制添加和查询

'''

defchat(request):
 ifrequest.method=='POST':
```

```
    returnsay(request)
elifrequest.method=='GET':
    returnchatAllLog(request
else:
    returnHttpResponse('access deny')

'''

public

@desc 删除对应的记录

'''

defchatDelete(request,delete_id):
    Chat.objects.get(id=delete_id).delete()
    record_offset=request.session.get('record_offset')
    request.session['record_offset']=record_offset-1
    returnHttpResponse(simplejson.dumps({'success':True}),mimetype='application/json')

'''
public
@desc 保存用户的消息到数据库中
@param POST中的，username和content
'''

defsay(request):
    req=simplejson.loads(request.raw_post_data)
    username=req['username']
    content=req['content']

    ifnotcontent:
        returnHttpResponse(simplejson.dumps({'success':False}),mimetype='application/json')

    chat=Chat()
    chat.content=content
    chat.username=username
    chat.save()
```

```
returnHttpResponse(simplejson.dumps({'success':True}),mimetype='application/json')

'''
public
@desc 根据 session 中的 record_offset 的数值获取以该数值为起始的所有记录
@return 返回对应的对象的字典形式
'''

defchatAllLog(request):
 if'record_offset'inrequest.session:
    record_offset=request.session.get('record_offset')
 else:
    record_offset=0
    request.session['record_offset']=0
  chatList=Chat.objects.all()[record_offset:]
  chatlist_dict=[]
  request.session['record_offset']=len(chatList)+ record_offset
 forchatinchatList:
    chatlist_dict.append({'id':chat.id,'content':chat.content,
              'username':chat.username,
              'date':str(chat.date).split('.')[0]
              })
 returnHttpResponse(simplejson.dumps(chatlist_dict),mimetype='application/json')
```

里面的代码确实不多，只有五个函数，第一个 loadpage 自然就不用说了。那么剩下的四个呢？chat 函数，作用很明确嘛，就像在 urls.py 上说的一样，只负责根据 POST 和 GET 来执行对应操作。而 say 和 chatAllLog 就是增加和查询的函数。另外一个 chatDelete 很明显就是删除的操作。

在实现 server 端的时候其实是有一个疑问的。就是在查询的时候怎么返回已有的聊天记录，因为只有简单的一张表。

一开始的构思是 web 端每隔两三秒就 fetch 一下，把所有的数据都拿过去。但是这对于服务器和带宽来说开销太大，每次都查询和传递所有的数据确实显的太笨了。

那么怎么才能每次只返回最新插入数据库中的数据呢？

本来我想从 backbone 的 collection.fetch 这个函数上下手的，但是没找到我想象的那种“差异化查询“的东西。所以就想了一个比较笨的方法，就是使用 session 来记录每次取了多少数据，因为数据是累加的，所以只需要从对应的记录开始取就可以了。

所以就出现上面代码中没有提到的 session 操作。

具体来说有三个操作：

- 第一、每提取一次数据，都进行统计，统计出一共取了多少条数据到 web 端。
- 第二、每次删除一条记录，对应的减少 session 中数据的统计。
- 第三，每次刷新页面都要从 0 开始计数。

所以这样就出现了上述代码中关于 session 的操作部分。

好像遗忘了实体类的介绍，不过我觉得这个不重要，因为它和 web 端的是一样的。为了完整，粘贴到下面：

```
#coding=utf-8
fromdjango.dbimportmodels

classChat(models.Model):
 content=models.CharField(max_length=1024)
 username=models.CharField(max_length=1024)
 date=models.DateTimeField(auto_now_add=True)
```

到此为止，这个 webchat 已经构建完毕了，目前这个项目看起来还是很简单的，如果你感兴趣可以对其进行完善。关于 backbone 的实践到此也告一段落了。

想要告诉大家的是，一定要动手去做一个项目，即便是你跟着文章做了一遍。在我分析 todos 和写 webchat 时是两种截然不同的感受，分析 todos 时我觉得作者的代码很凝练，很优雅，我看着能理解，感觉能写出来，但是在写 webchat 的时候才发现，明白、理解不一定意味着你能写出来，所以想掌握、提高还是要多实践。

学习任何一门技术的过程都是：**模仿—使用—发挥**。

webchat 在 gihub:https://github.com/the5fire/webchat

16、总结的说

作者：胡阳

本文链接:http://www.the5fire.net/16-backbone-summary.html

到目前为止我个人感觉已经把 backbone.js 的基本使用说清楚了，如果有哪里不清楚的，大家提出来一起探讨。

从一开始写这系列文章到现在已经快一个月了，一开始接触到觉得这个很不错，但是中文资料太少了，所以就萌生了写一系列基础的文章，让其他人在学习这个框架的时候多些参考资料，这一系列文

章称不上教程，只算是笔记。能给其他学习者带来些助力是我最欣慰的事。

其实光研究这个框架花三天多的时间就够了，但是一写起文章来，就觉得要尽可能的详细，生怕误导别人，于是就战线越拉越长，到现在才算是告一段落。

说其对 backbone 的掌握，其实我自己依然是个初学者，停留在应用阶段，任何一个框架的诞生到流行，都离不开框架中所蕴涵的哲理，或者说是精髓。要掌握一个框架一定是从对框架的大量使用实践开始，不过对于有大量开发经验的人来说，直接品读其源码应该是最好的方式，我自己没有达到那样的水平，也没有尝试过，只是猜测。

总结的话也就这么多了，出发点只有两个，一是提高自己对 backbone 的掌握，二是分享些东西给大家。

下一篇写一些我用到的资源之后这一系列就算是完成了。

17、 backbone.js 相关资源

作者：胡阳

本文链接:http://www.the5fire.net/17-backbone-js-resource.html

首先自然就是 backbone.js 的官网：
http://documentcloud.github.com/backbone

然后是能让你大概认识 backbone.js 是什么以及怎么用的网站：
http://backbonetutorials.com/

另外还有几篇中文的博客也不错：
http://weakfi.iteye.com/blog/1391990

http://blog.csdn.net/soasme/article/details/6581029

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