一直想写一些 extjs4 MVC 的东西，但由于自己的英文水平足够媲美小学 5 年纪的学生，所以在找了一些比我强一点的网友+机器翻译，总结出了以下这篇文章。但个人强烈建议去看英文原版（点击进入）。看完本文后，如有任何错误，欢迎来信或者留言指正(QQ:301109552)。

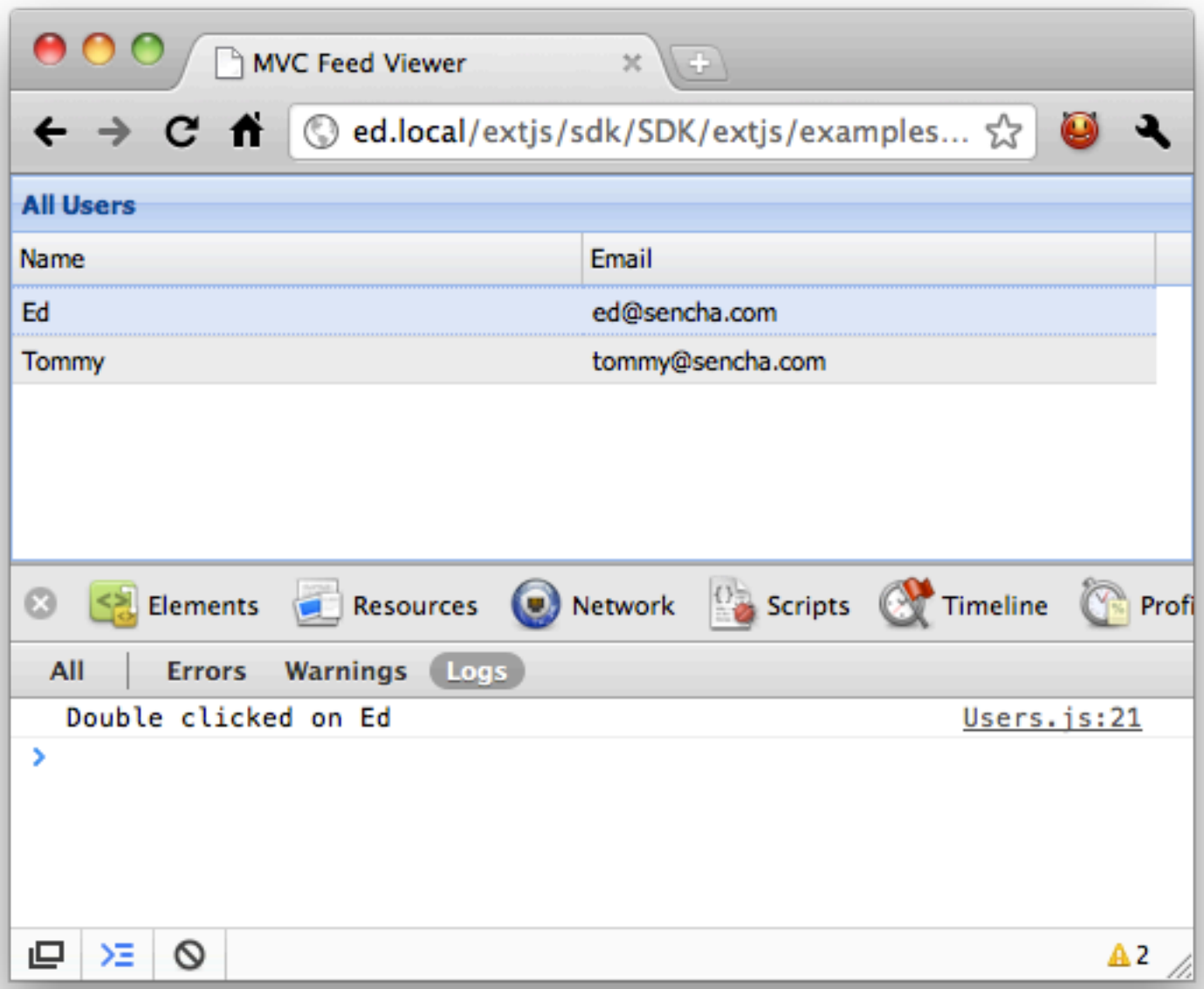
如果对此还不算了解，请查看上一节：extjs4 API 文档阅读（二）——MVC 构架(上)

控制网格

要注意的是,onPanelRendered 功能仍然是被调用的。这是因为 gird 匹配'viewport > panel'。
然后我们添加一个监听，当我们双击 grid 中的行，就可以编辑用户。

```
1.  Ext.define('AM.controller.Users', {
2.  extend: 'Ext.app.Controller',
3.  views: ['user.List'],
4.  init: function() {
5.          this.control({
6.                  'userlist': {
7.                  itemdblclick: this.editUser
8.                  }
9.          });
10. },
11. editUser: function(grid, record) {
12.         console.log('Double clicked on ' +record.get('name'));
13. }
14. });
```

这里，我们修改了 ComponentQuery 的选择（'userlist'）和事件的名称（'itemdblclick'）和处理函数（'editUser'）。



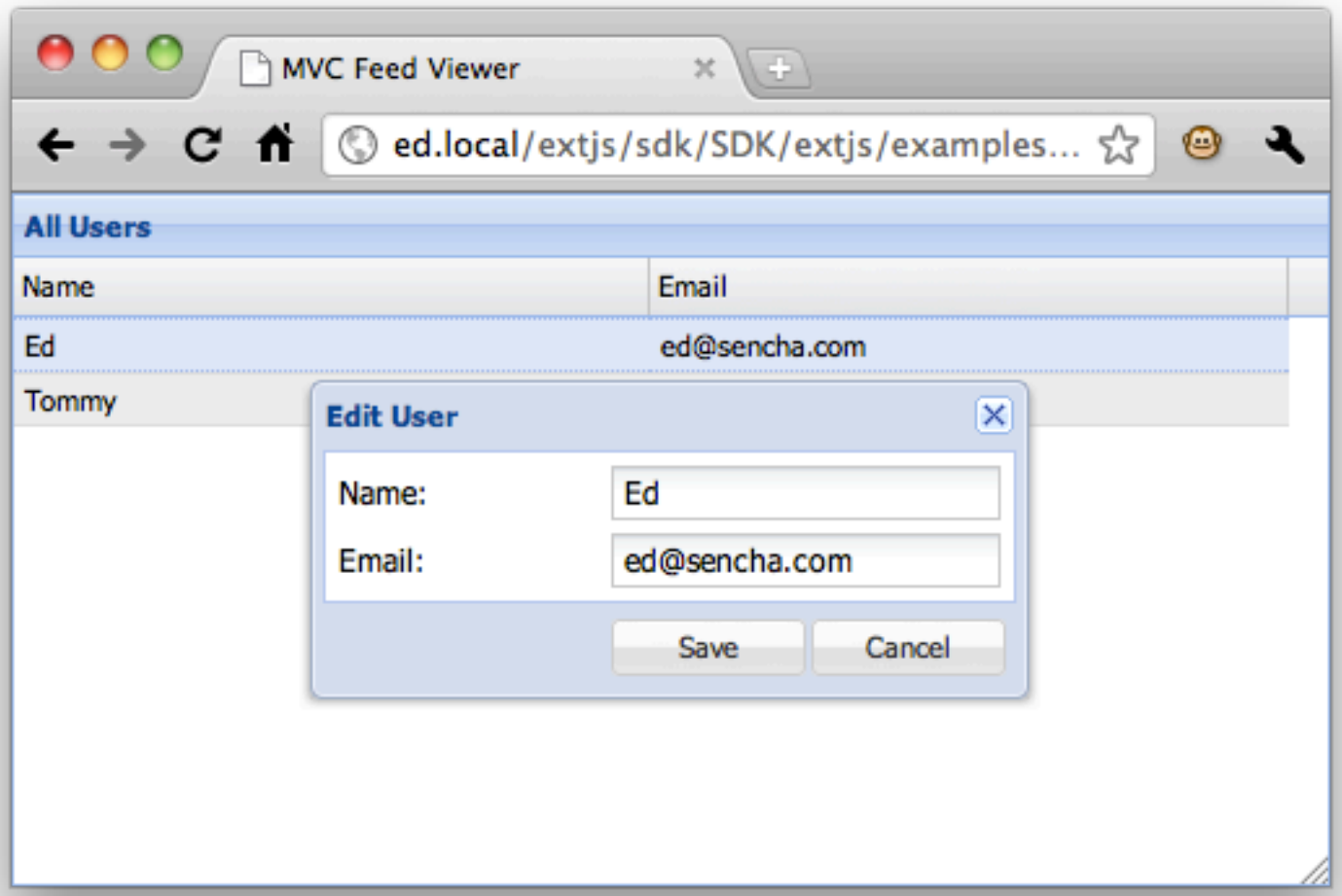
如果要实现真正编辑用户，那么我们需要一个真正用于用户编辑 window，接着，创建一个 JS 文件。其路径是：app/view/user/Edit.js，代码是这样的：

```
Ext.define('AM.view.user.Edit', {
extend: 'Ext.window.Window',
    alias : 'widget.useredit',
title : 'Edit User',
layout: 'fit',
    autoShow: true,
initComponent: function() {
        this.items = [{
            xtype: 'form',
            items: [{
                xtype: 'textfield',
                    name :'name',
                fieldLabel: 'Name'
              },
            {
                xtype: 'textfield',
                name : 'email',
                fieldLabel: 'Email'
                }]
        }];
        this.buttons = [{
            text: 'Save',
          action: 'save'
        },
          {
          text: 'Cancel',
          scope: this,
            handler: this.close
       }];
        this.callParent(arguments);
    }
});
```

我们定义了一个子类，继承 Ext.window.Window，然后使用 initComponent 创建了一个表单和两个按钮，表单中，两个字段分别装载用户名和电子邮件。接下来，我们修改视图控制器，使其可以载入用户数据。

```
Ext.define('AM.controller.Users', {
extend: 'Ext.app.Controller',
views: [
        'user.List',
    'user.Edit'
],
init: ...
    editUser: function(grid, record) {
        var view = Ext.widget('useredit');
            view.down('form').loadRecord(record);
}
});
```

首先，我们创建的视图，使用便捷的方法 Ext.widget，这是相当于 Ext.create（'widget.useredit“）。然后我们利 用 ComponentQuery 快速获取编辑用户的形式引用。在 Ext JS4 的每个组件有一个 down 函数，可以使用它快速的找到任何他的子组件。当双击 Grid 中的行，我们可以看到如下图所示：



创建 Model 和 Store

```
1. Ext.define('AM.store.Users', {
2. extend: 'Ext.data.Store',
3.     fields: ['name', 'email'],
4. data: [
5.         {name: 'Ed',    email: 'ed@sencha.com'},
6.         {name: 'Tommy', email: 'tommy@sencha.com'}
7. ]
8. });
```

接下来修改两个文件：

```
1. Ext.define('AM.controller.Users', {
2. extend: 'Ext.app.Controller',
3.     stores: [       'Users'    ],
4.     ...
5. });
```

修改 app/view/user/List.js，使其引用 Users

```
1. Ext.define('AM.view.user.List' ,{
2. extend: 'Ext.grid.Panel',
3.     alias : 'widget.userlist',
4.     //we no longer define the Users store in the `initComponent` method
5.     store: 'Users',
6. ...
7. });
```

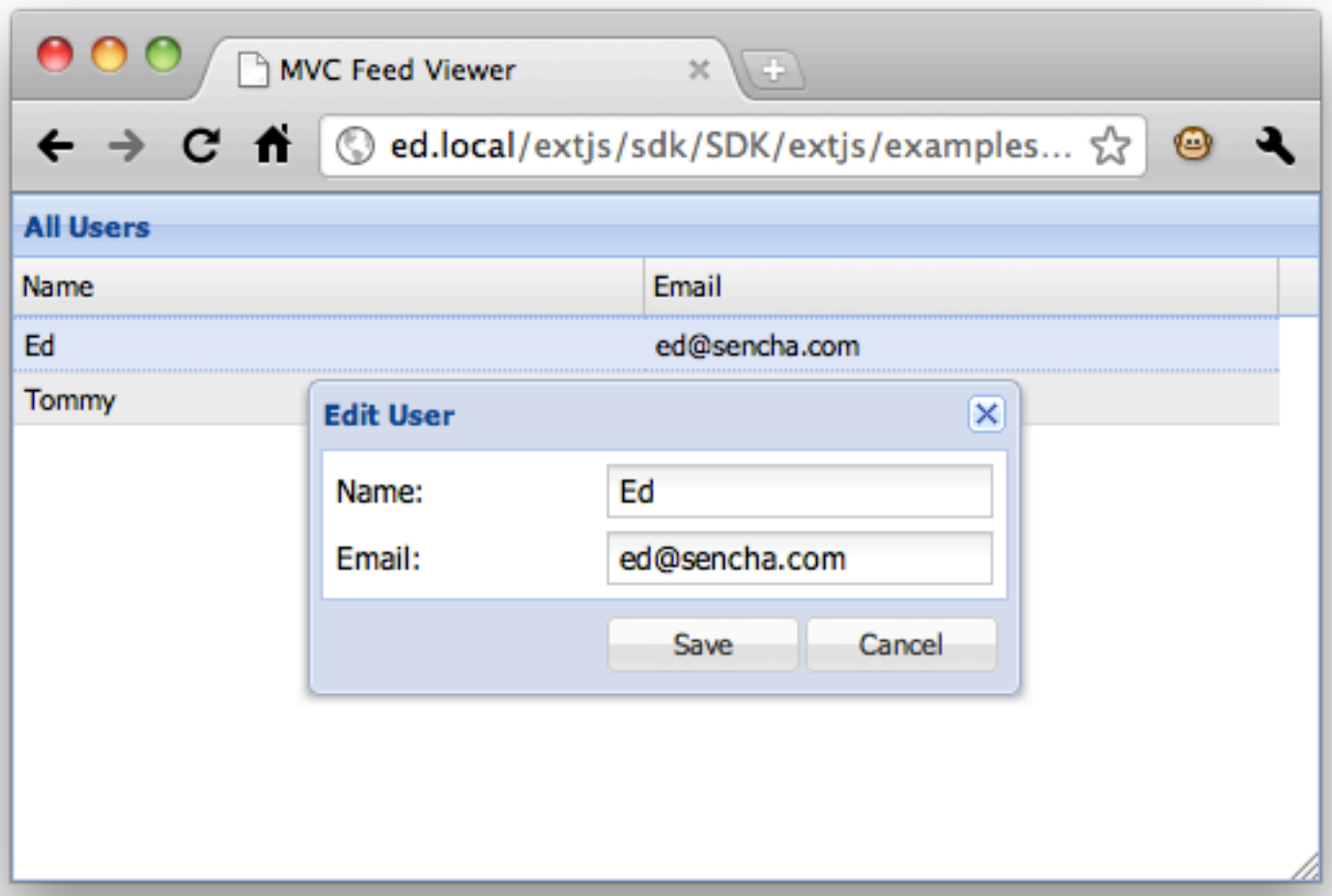
现在，我们定义的用户控制器已经能顺利的加载数据了，到目前，我们所定义的 Store 已经足够使用了，但是 extjs4 提供了一个强大的 Ext.data.Model 类，不如，我们利用它来重构下我们 Store，创建 app/model/User.js

```
1.  Ext.define('AM.model.User', {
2.  extend: 'Ext.data.Model',
3.      fields: ['name', 'email']
4.  });
```

创建好模型之后，我们将他引入用户控制：

```
1.  //the Users controller will make sure that the User model is included on the page and available to our app
2.  Ext.define('AM.controller.Users', {
3.  extend: 'Ext.app.Controller',
4.      stores: ['Users'],
5.  models: ['User'],
6.  ...
7.  });
8.  // we now reference the Model instead of defining fields inline
9.  Ext.define('AM.store.Users', {
10. extend: 'Ext.data.Store',
11.     model: 'AM.model.User',
12. data: [
13.         {name: 'Ed',    email: 'ed@sencha.com'},
14.         {name: 'Tommy', email: 'tommy@sencha.com'}
15. ]
16. });
```

完成上面的代码后，刷新页面，看到的结果和以前的一样。



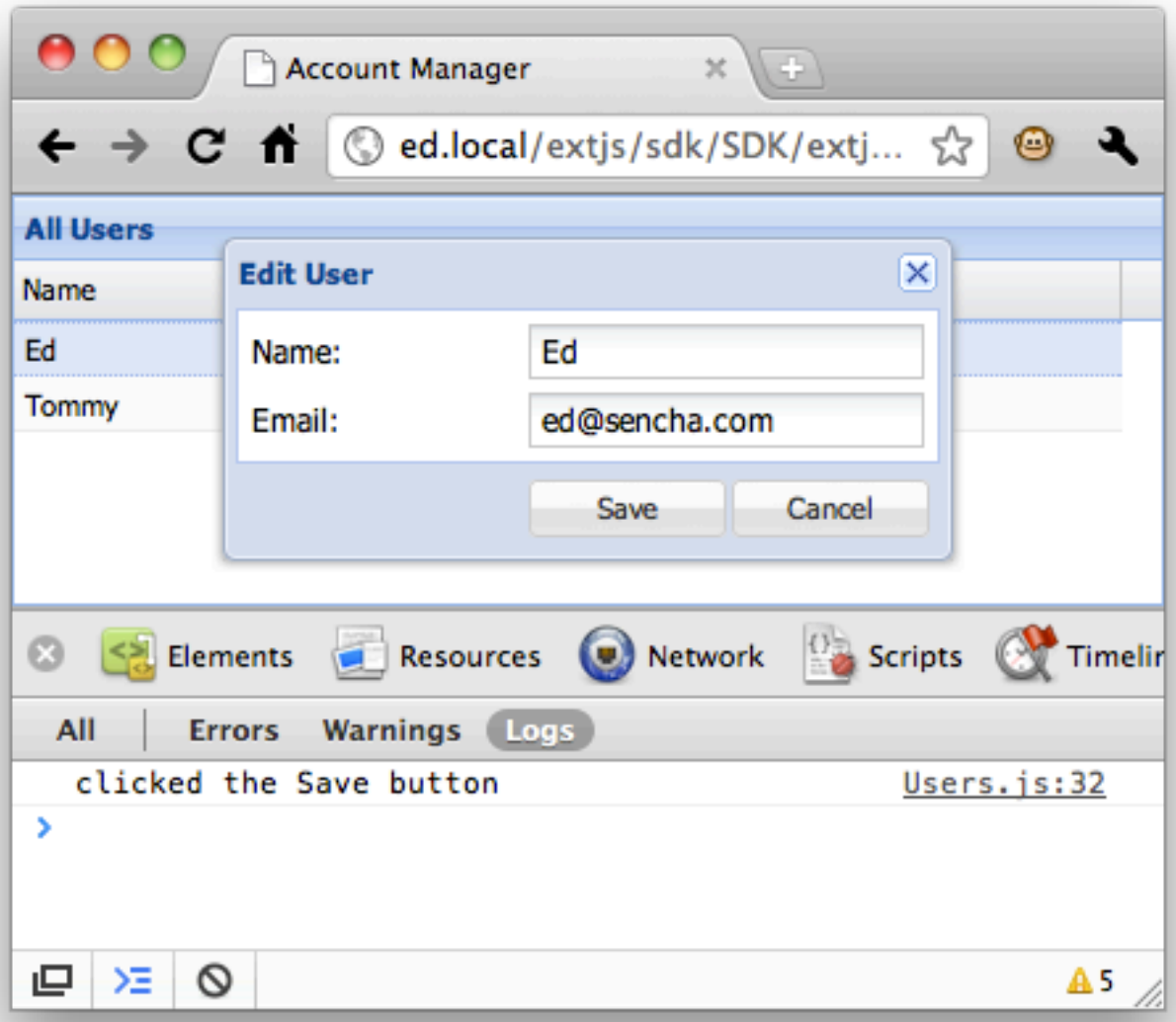
保存数据

现在双击 Grid 中的行，会弹出编辑用户的 window，实现 Save 来保存用户数据，我们需要修改 init 函数。

```
1.  Ext.define('AM.controller.Users', {
```

```
2.  init: function() {
3.          this.control({
4.              'viewport > userlist': {
5.                  itemdblclick: this.editUser
6.          },
7.              'useredit button[action=save]': {
8.                  click: this.updateUser
9.              }
10.       });
11. },
12. updateUser: function(button) {
13.         console.log('clicked the Save button');
14. }
15. });
16.
```

在 this.control 中，我们增加了一个选择项，'useredit button[action=save]'，当 ComponentQuery 找到符合的组件（button 按钮，并且 action 动作为 save），给他 增加一个方法 click，事件为 updateUser。如图：

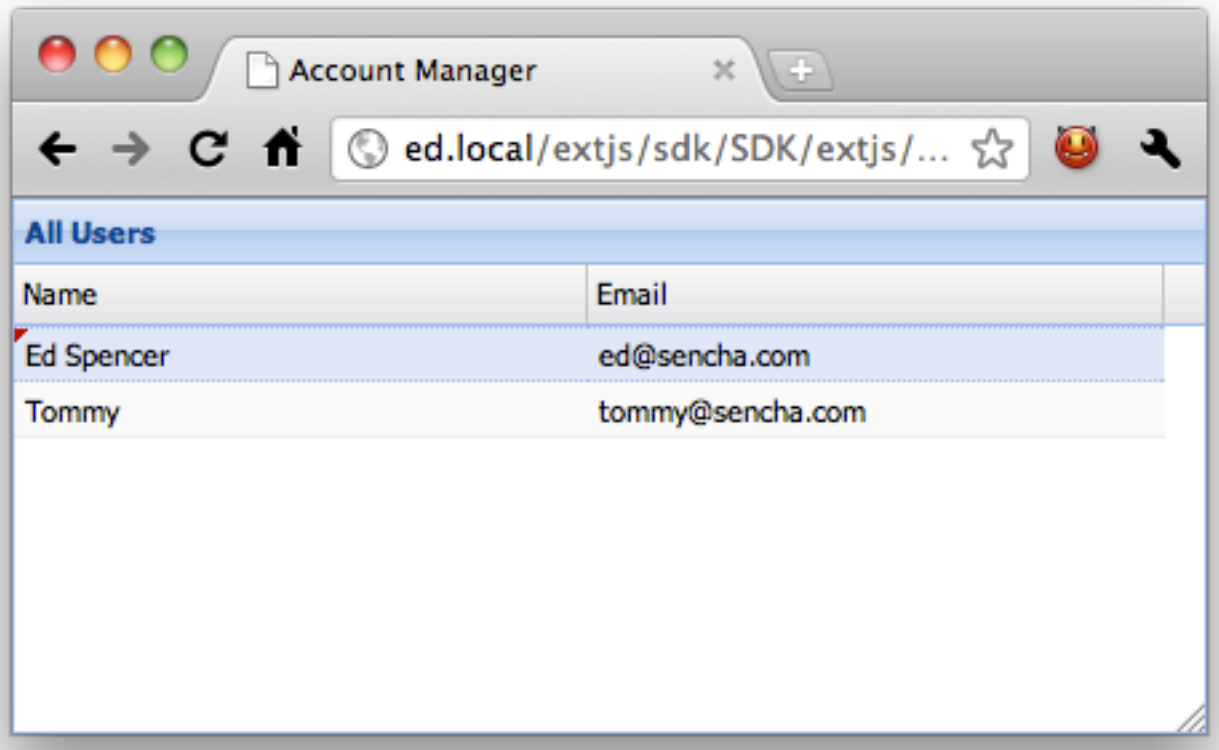


上图中，可以看到正确的 Click 事件，在 updateUser 函数中，需要一个正式的逻辑，来完成用户数据的更新。

```
1.  updateUser: function(button) {
2.  var win    = button.up('window'),
3.          form   = win.down('form'),
4.      record = form.getRecord(),
5.          values =form.getValues();
6.      record.set(values);
7.     win.close();
8.  }
```

这里，当我们点击 Save 按钮，我们将按钮本身传入函数 updateUser，这时，我们使用 button.up('window')来获取用户 window 的引用，然后使用 win.down('form')来获取表单的引用，然后获取表单的记录，获取表单中的值，再设置记录为新的值，最后关闭 window。

修改数据后点击 Save，更新完成。



实际应用过程中，我们需要将数据保存到服务器，所以，需要修改文件，达到想要的目的。

将数据保存到服务器

这很容易做到。使用 AJAX 来实现即可。

```
1.  Ext.define('AM.store.Users', {
2.      extend: 'Ext.data.Store',
3.          model: 'AM.model.User',
4.      autoLoad: true,
5.      proxy: {
6.          type: 'ajax',
7.              url: 'data/users.json',
8.          reader: {
9.              type: 'json',
10.             root: 'users',
11.             successProperty: 'success'
12.     }
13.     }
14. });
```

在 AM.store.Users，移除 data，用一个代理（proxy）取代它，用代理的方式来加载和保存数据。在 extjs4 中，代理的方式有 AJAX, JSON-P 和 HTML5 localStorage，这里使用 AJAX 代理。数据从 data/users.json 中得到。

我们还用 reader 来解析数据，还指定了 successProperty 配置。具体请查看 Json Reader，把以前的数据复制到 users.json 中。得到如下形式：

```
1.  {   success: true,   users: [
2.          {id: 1, name: 'Ed',   email: 'ed@sencha.com'},
3.          {id: 2, name: 'Tommy', email: 'tommy@sencha.com'}
4.   ]}
```

到这里，唯一的变化是将 Stroe 的 autoLoad 设置为了 true，这要求 Stroe 的代理要自动加载数据，当刷新页面，将得到和之前一样的效果。

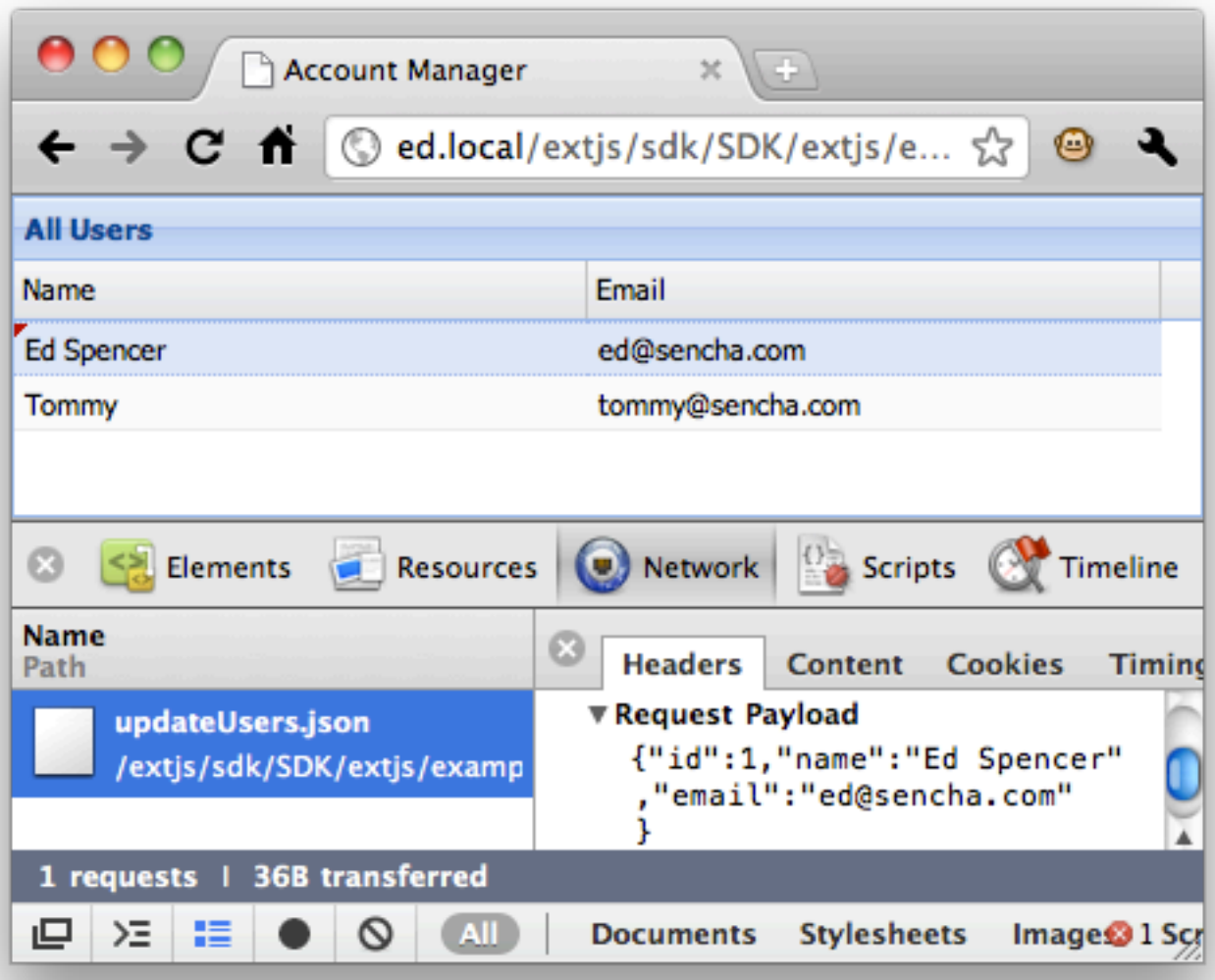
最后一件事情，是将修改的数据发送到服务器，对于本例，服务端只用了一个静态的 JSON，所以我们不会看到有任何变化，但至少可以确定这样做是可行的，相信服务端处理数据能力，大家都应该有。本例用，做个小小的变化，即新的代理中，使用 api 来更新一个新的 URL。

```
1.  proxy: {
2.      type: 'ajax',
3.      api: {
4.          read: 'data/users.json',
5.              update: 'data/updateUsers.json'
6.      },
7.     reader: {
8.          type: 'json',
9.              root: 'users',
10.         successProperty: 'success'
11. }
12. }
13.
```

再来看看是如何运行的，我们还在读取 users.json 中的数据，而任何更新都将发送到 updateUsers.json，在 updateUsers.json 中，创建一个虚拟的回应，从而让我们知道事件确实已经发生。updateUsers.json 只包含了 {"success": true}。而我们唯一要做的事情是服务的同步编辑。在 updateUser 函数增加这样一个功能。

```
1.  updateUser: function(button) {
2.          var win    = button.up('window'),
3.          form   = win.down('form'),
4.          record = form.getRecord(),
5.          values = form.getValues();
6.  record.set(values);
7.      win.close();
8.          this.getUsersStore().sync();
9.  }
```

现在，运行这个完整的例子，当双击 grid 中的某一样并进行编辑，点击 Save 按钮后，得到正确的请求和回应。



至此，整个 MVC 模式全部完成，下一节，将讨论控制器（Controller）和模式（patterns），而这些，可以使应用代码更少，更容易维护