作者：zhoudawn@163.com

**前言**

* ***WEB开发让人望而却步***

**眼花缭乱的HTML元素**

a、b、body、br、button、caption、div、dt、em、embed、fieldSet、font、form、frame、frameSet、head、hr、iframe、img、input、label、li、link、p、select、table、td、th、tr、u、ul 。。。

**根本不懂更不会用的CSS**

font、font-family、font-size、font-style、font-weight、text-align、text-decoration、text-indent、vertical-align、background-color、background-image、background-position、

background-repeat、color、border、float、margin、margin-bottom、margin-left、margin-right、margin-top、padding、padding-bottom、padding-left、padding-right、padding-top、height、left、overflow、position、right、top、visibility、width、z-index。。。

**最关键的**

**美工！ 美工！ 美工！**

|  |
| --- |
| **问题** 不懂HTML(5)、不懂CSS(3)、没有美工也能做WEB开发？  ExtJS 有我在，Web开发根本不算事。 |

* *ExtJS是什么*

目前世界上最先进和最强大的、支持多平台多设备的应用程序开发框架。

* *ExtJS发展简史*

1. YUI-Ext的作者Jack Slocum(杰克.斯洛克姆)打算对基于BSD协议的Yahoo User Interface (YUI)库进行自定义的扩展，但后来一度风头盖过其父辈YUI；
2. 在2006年底，YUI-Ext被简化为Ext，反映了它作为一个框架的成熟和独立。该公司成立于2007年初，Ext现在为双执照，使用LGPL和一个商业执照；
3. 在2007年4月1日，发布1.0正式版，现在的版本为6.0；
4. 从4.0开始引入了MVC/MVVM风格的应用架构和SCSS；
5. 从5.0开始，不再支持老旧浏览器，如IE5、6、7等；
6. 从6.0开始，Sencha Touch和ExtJS合二为一，标志着用户可以开发既能支持传统桌面、又支持平板或手机的应用。

# 开发准备

**下载ExtJS 6.0 SDK GPL正式版**

<https://www.sencha.com/legal/gpl/>

**下载ExtJS 6.0离线帮助文档**

经典版

<http://docs.sencha.com/downloads/extjs-docs-6.0.0-classic.zip>

现代版

<http://docs.sencha.com/downloads/extjs-docs-6.0.0-modern.zip>

**下载Sencha Tool**

<http://cdn.sencha.com/cmd/6.0.0.202/no-jre/SenchaCmd-6.0.0.202-windows-no_jre.zip>

（假设你的系统里已经安装了JRE7+，否则请下载带JRE版本的Sencha Tool）

**安装Sencha Tool**

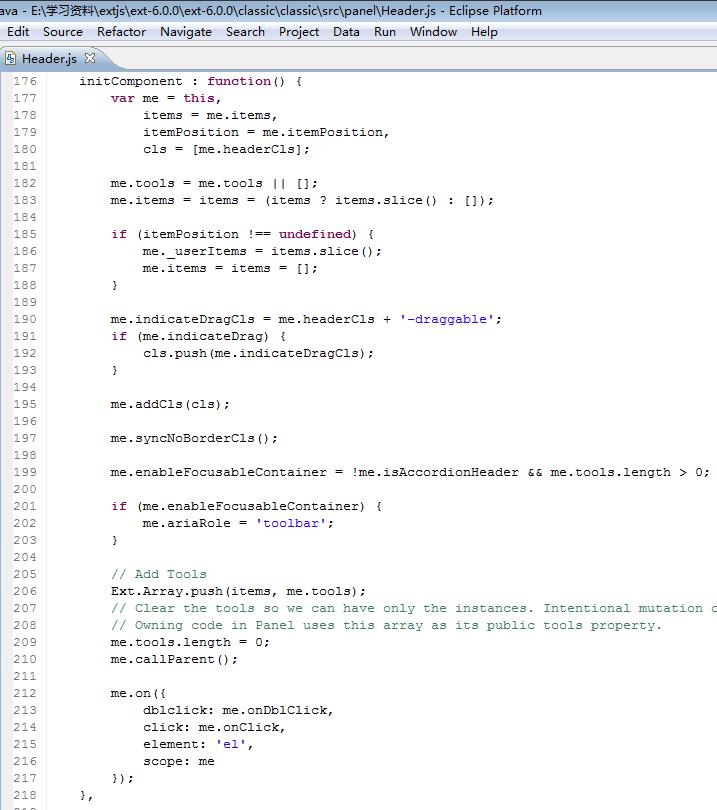
默认安装即可。

解压离线帮助文档extjs-docs-6.0.0-classic.zip

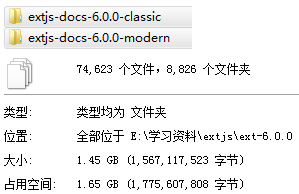
解压ext-6.0.0-gpl.zip

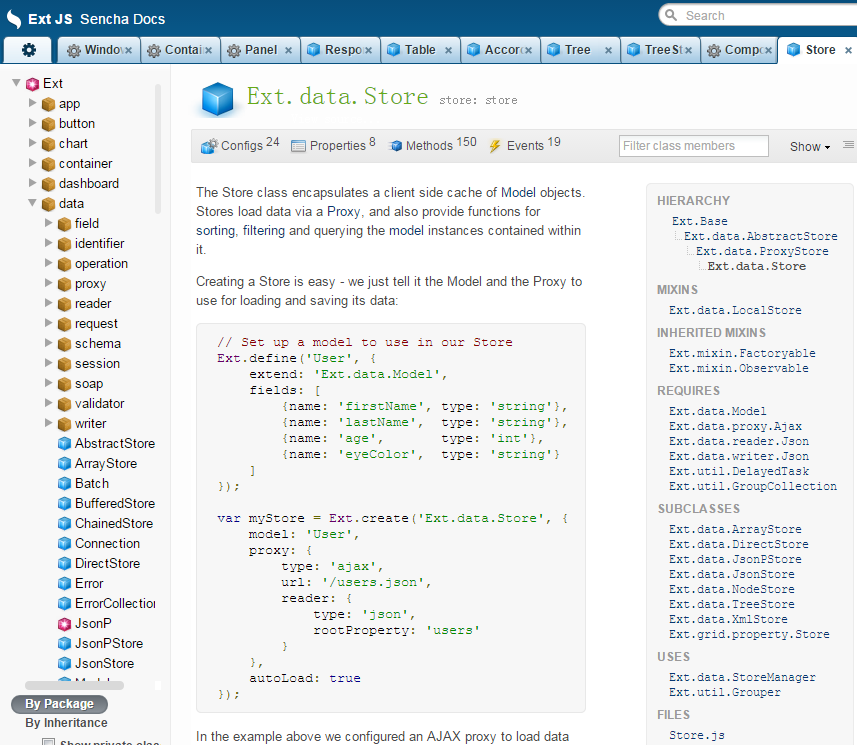
# 初识ExtJS

## 代码质量



## 帮助文档





## 商业支持

一套代码，两份许可（GPL和商业许可）。

## 强大的辅助开发工具

## 开发工作量

## 功能演示

详见ext-6.0.0\examples。

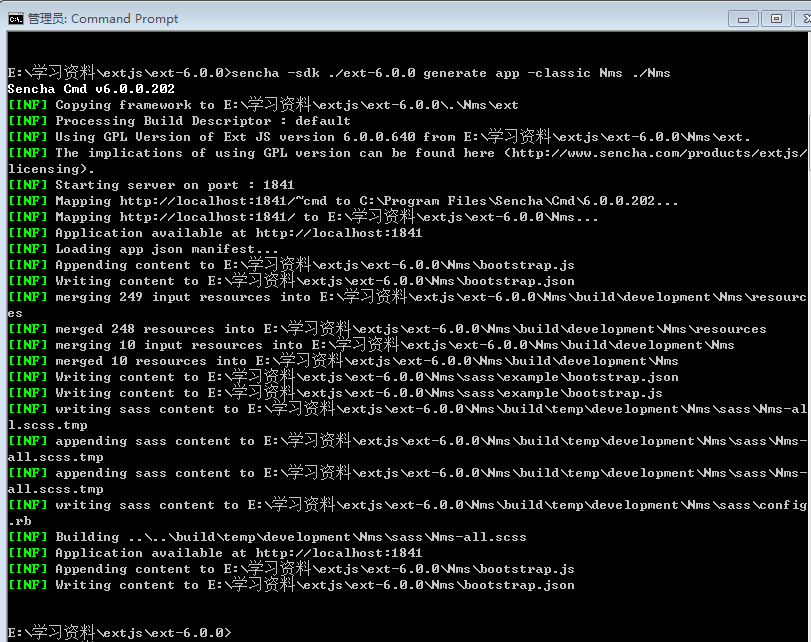
## ExtJS的优势总结

期待各位总结归纳。

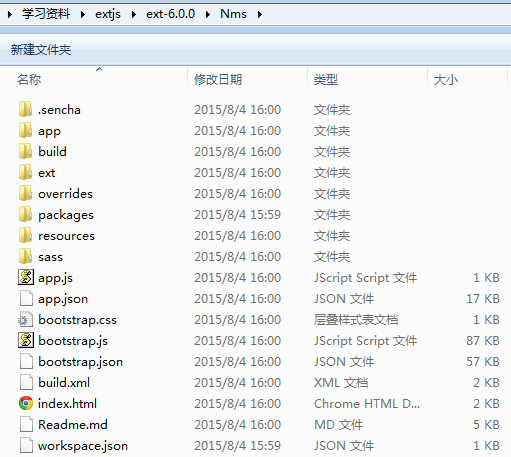
# 生成程序框架

**注意**：以下DOS命令均以管理员身份运行。

>sencha -sdk ./ext-6.0.0 generate app -classic Nms ./Nms



进入到Nms目录，看到生成的工程

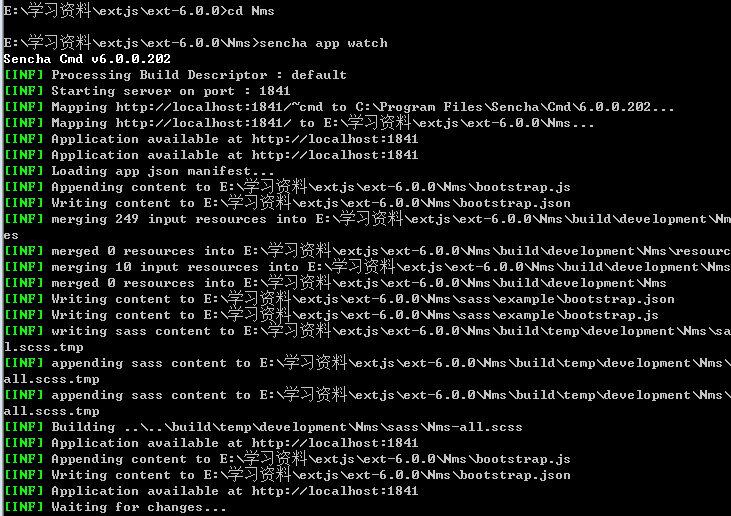


cd Nms

进入到该目录

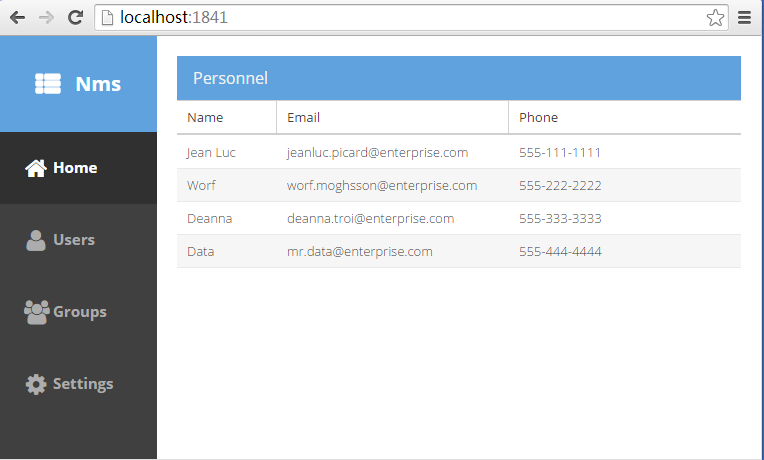
执行

sencha app watch



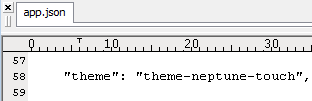
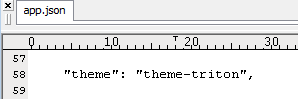
浏览器输入

<http://localhost:1841>



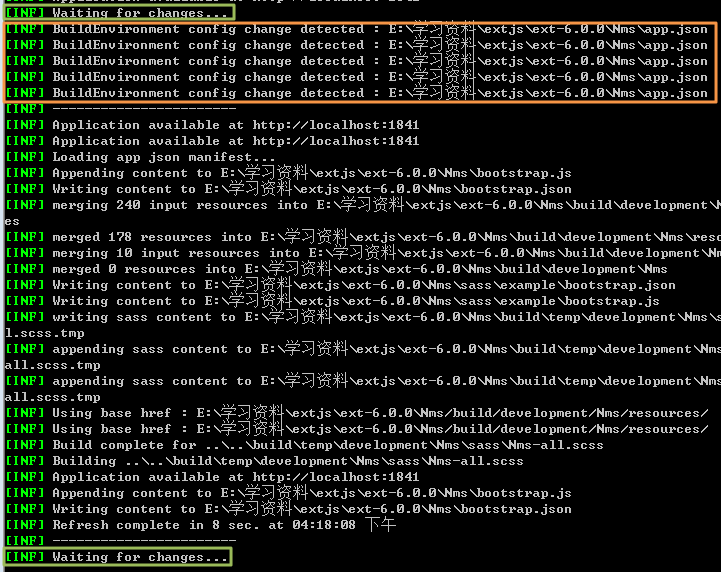
# 定制样式

## 修改主题



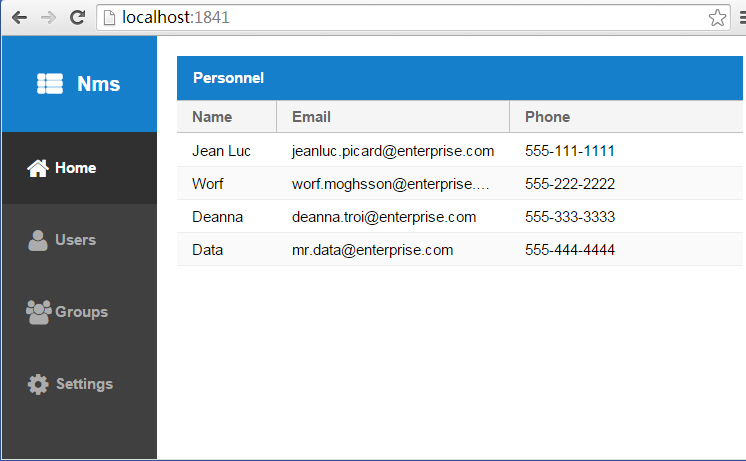
打开app.json，定位到"theme": "theme-triton"改为"theme": "**theme-neptune-touch**"并保存。

观察DOS窗口的变化。



可以看到，橙色标识的部分表示sencha cmd 工具监测到了文件被修改app.json被修改了，工具实时编译，完成后再次输出**Waiting for changes…**，继续监听文件是否有更改。

刷新浏览器，看到主题变化已经生效。

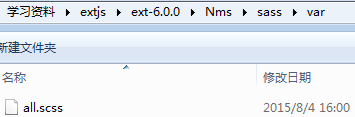


## 修改全局主题变量

为了能够实时看到修改后的效果，我们把浏览器地址增加一个参数

http://localhost:1841/**?platformTags=fashion:true**

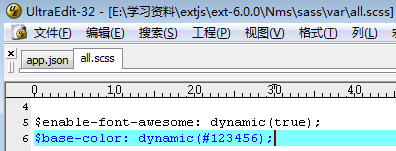
打开all.scss文件



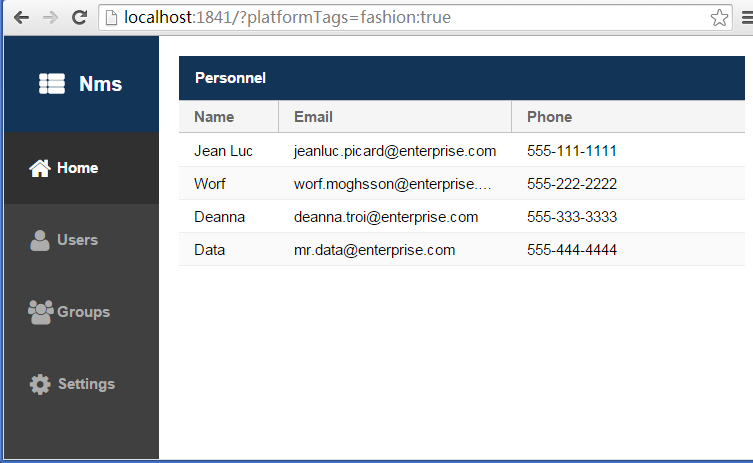
增加一行

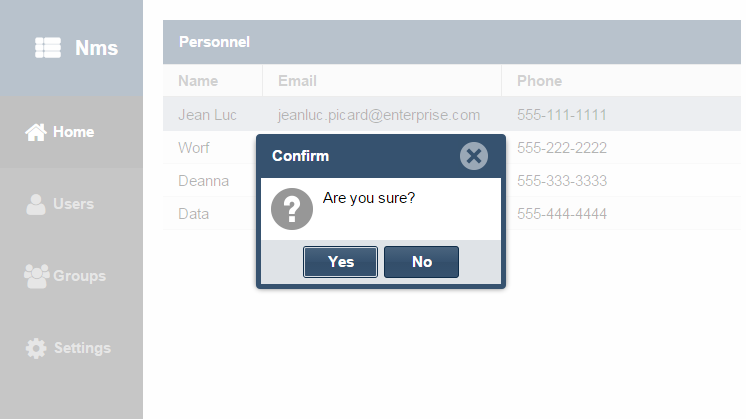
$base-color: dynamic(#123456);

如下图所示：



回到浏览器页面，效果如下：





可以看到，标题和按钮的颜色为我们设置的颜色值。

查看哪些全局的主题变量可以修改，需要参考Ext Api Document

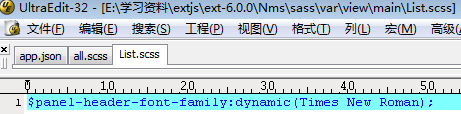


## 配置组件的变量

新建一个文件

Nms\sass\var\view\main\**List.scss**，该文件对应于代码

Nms\app\view\main\List.js，意味着我们将要修改视图Nms.view.main.List 的样式。



设置panel的背景

$panel-body-background-color : dynamic(#4682B4);

设置表头背景

$grid-header-background-color : dynamic(#F0FFF0);

设置行间隔颜色为透明

$grid-row-cell-border-color : dynamic(transparent);

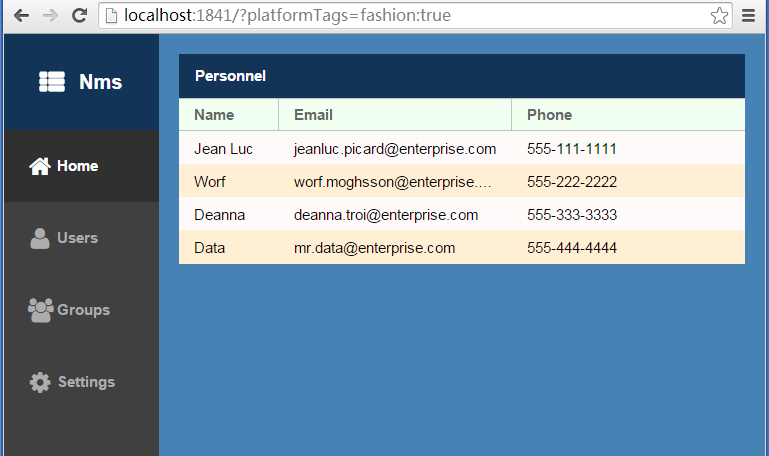
设置列表行背景

$grid-row-cell-background-color : dynamic(#FFFAFA);

设置列表行交替背景

$grid-row-cell-alt-background-color : dynamic(#FFEFD5);

最终效果如下



## 通过JS覆盖(override)主题

创建overrides\panel\Panel.js

加入如下代码

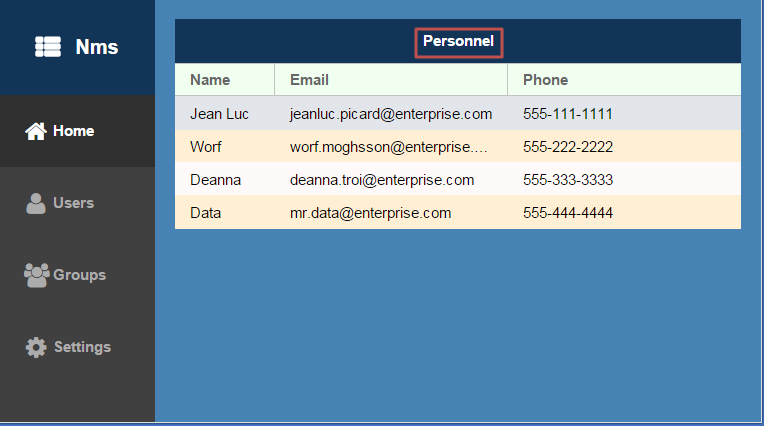
Ext.define('Nms.panel.Panel', {

override : 'Ext.panel.Panel',

titleAlign : 'center'

});

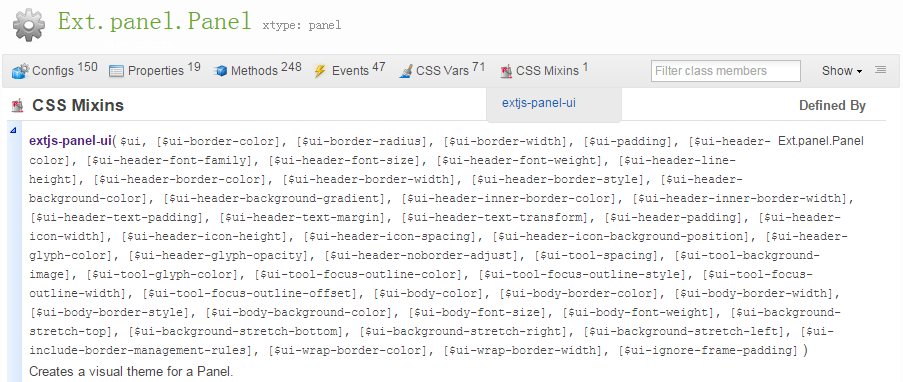
刷新页面，观察页面变化。



## 创建用户组件UI

每一个Ext Framework下的组件都拥有自己的UI设置参数，缺省下都是default，这个属性可以给予单独实体组件拥有自己的UI配参，可以让你在同一类型组件下给予不同的样式。

Ext为各种组件提供了很多SASS mixins混入，你可以用这些mixins来生成新的UI，这可以从API中找到它们。如



创建sass\src\panel\Panel.scss，增加

@include **extjs-panel-ui** (

$ui:'**highlight-framed**',

$ui-header-background-color:#FF6347,

$ui-border-color:#FF6347,

$ui-header-border-color:#FF6347,

$ui-body-border-color:#FF6347,

$ui-header-color:white

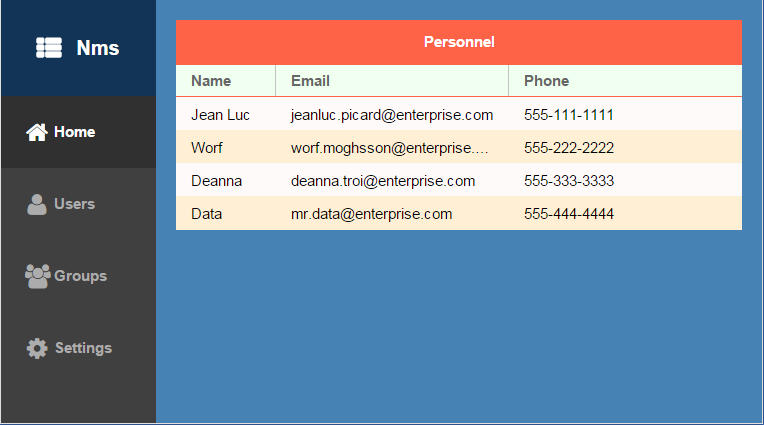
);

修改Nms\app\view\main\List.js代码，增加

**ui : 'highlight',**

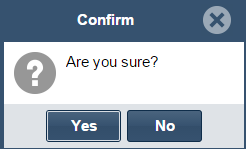
**frame : 'true'**,

两个配置，刷新页面，观察页面变化

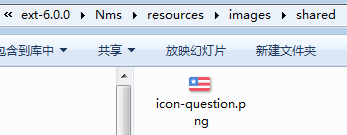


## 定制Image

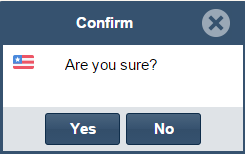
双击列表的某条记录，弹出如下对话框，



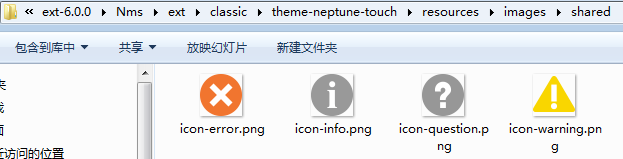
建立一个png文件



当再双击列表记录时，弹出的对话框图标变为了我们刚刚设置的图标。



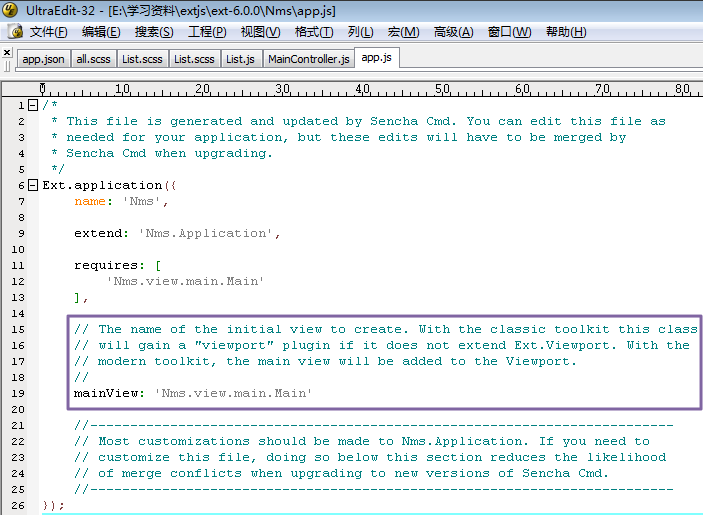
我们使用的主题是theme-neptune-touch，对比一下目录和文件名，就明白我们为什么要在Nms\resources\images\shared\下面创建同名icon-question.png图片了。



# 开发详解

## 理解框架代码

### 程序入口app.js



mainView配置项是用来创建的初始化视图，也就是说程序在初始化时会创建类为“Nms.view.main.Main”的实例。

注意，匿名类继承的是**Nms.Application**类。

### Nms.Application



stores存放的是全局或者共享的store，如果定义的store不在这里配置，则程序不会加载该store。

### Nms.view.main.Main

extend: 'Ext.tab.Panel', 继承自Ext.tab.Panel

controller: 'main', 使用名字为main的控制器，

Nms.view.main.MainController里有配置项alias: 'controller.**main**'

viewModel: 'main', 使用名字为main的视图模型

Nms.view.main.MainModel里有配置项alias: 'viewmodel.**main**'

ui: 'navigation', 见第二章介绍

header: {

layout: {

align: 'stretchmax'

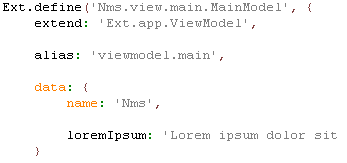
},

title: {

**bind**: {

text: '{**name**}' 绑定的是对应的viewModel里配置项data.name，见

Nms.view.main.MainModel



viewModel的好处是，当里面的数据发生变化时，bind到

viewModel的视图能立即更新

},

flex: 0

},

iconCls: 'fa-th-list'

},

responsiveConfig: {

tall: {

headerPosition: 'top' 当浏览器视图高度大于宽度时，tab显示在视图顶部

},

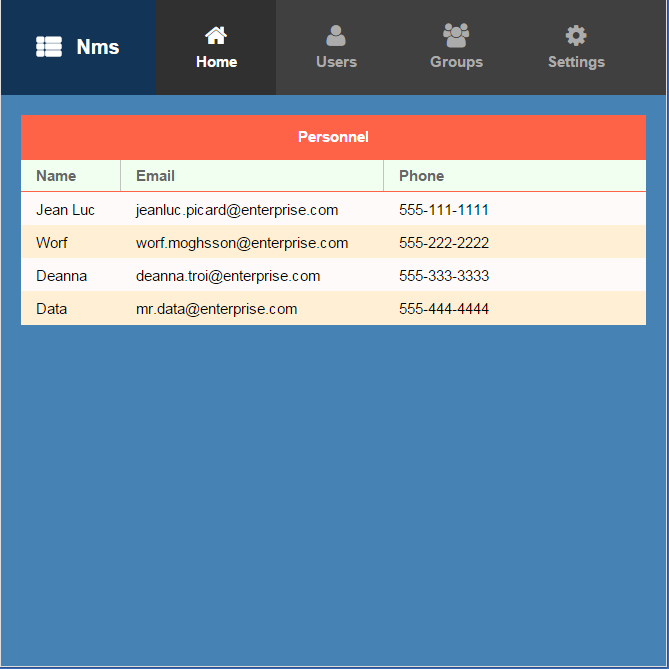
wide: {

headerPosition: 'left' 当宽度大于高度时，tab显示在视图左边

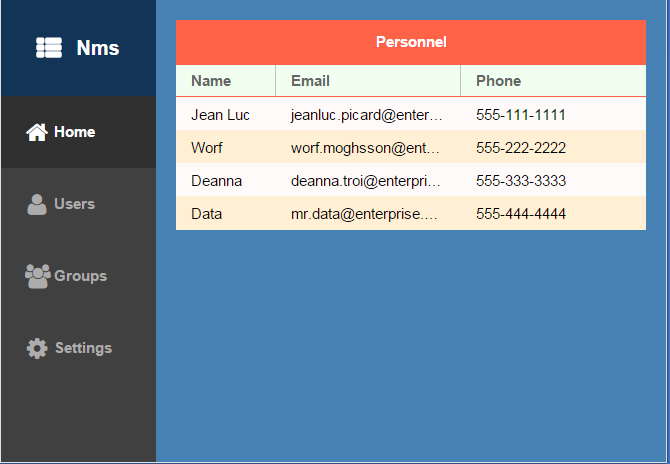
}

},

高度大于宽度效果



宽度大于高度效果



items: [{

title: 'Home',

iconCls: 'fa-home',

// The following grid shares a store with the classic version's grid as well!

items: [{

xtype: 'mainlist' Nms.view.main.List类的 **xtype: 'mainlist'**，是对类名的简写

}]

}, {

title: 'Users',

iconCls: 'fa-user',

bind: {

html: '{loremIpsum}' 见dataModel的data. **loremIpsum**

}

}, {

title: 'Groups',

iconCls: 'fa-users',

bind: {

html: '{loremIpsum}'

}

}, {

title: 'Settings',

iconCls: 'fa-cog',

bind: {

html: '{loremIpsum}'

}

}]

可以看出，配有4个Tab项。

### Nms.view.main.List

Ext.define('Nms.view.main.List', {

extend: 'Ext.grid.Panel',

xtype: 'mainlist',

ui : 'highlight',

frame : 'true',

requires: [

'Nms.store.Personnel'

],

title: 'Personnel',

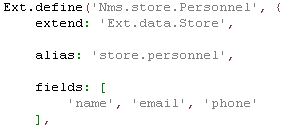
store: {

type: 'personnel' 使用的Nms.store.Personnel，其配置有alias: 'store.**personnel**',

},

columns: [ 有三列，每列要显示的值通过dataIndex指示

{ text: 'Name', dataIndex: 'name' }, dataIndex的配置值对应于

Nms.store.Personnel的fields里的项

{ text: 'Email', dataIndex: 'email', flex: 1 },

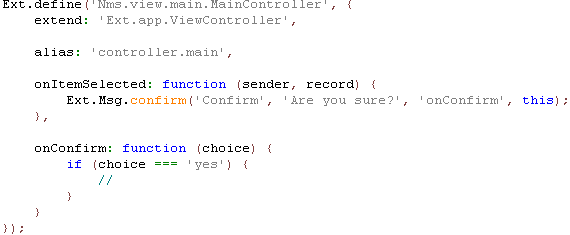
{ text: 'Phone', dataIndex: 'phone', flex: 1 }

],

listeners: {

select: 'onItemSelected' 注册事件处理函数，当select事件发生时，

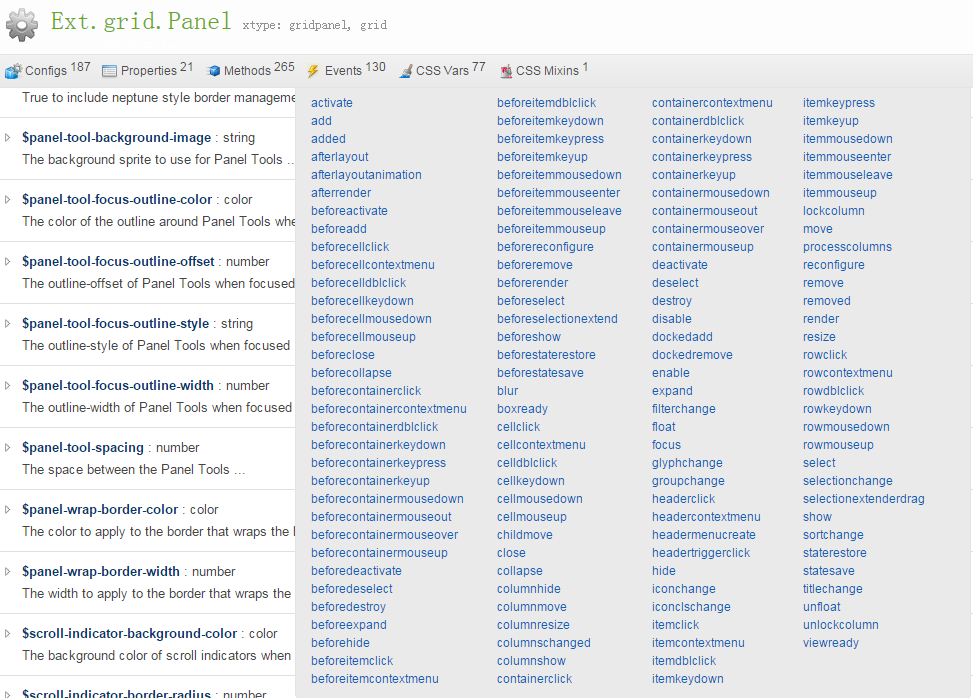
调用controller的'**onItemSelected**'函数



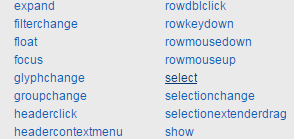
}

});

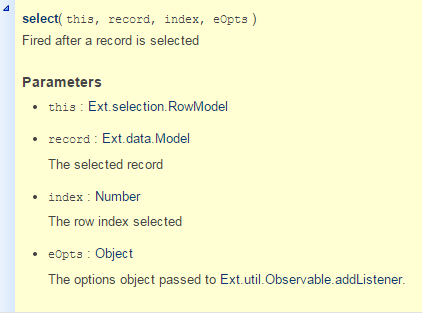
事件及其处理函数的参数，参考帮助文档。如



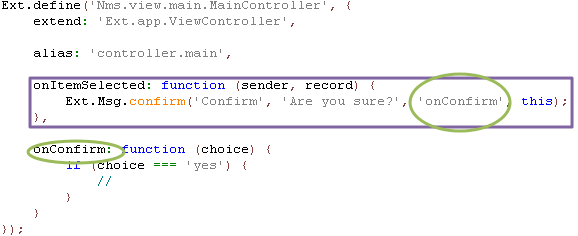
点击select



跳转到select事件处理的函数原型



### Nms.view.main.MainController



onItemSelected对应于Nms.view.main.List注册的回调处理函数。

### Nms.view.main.MainModel

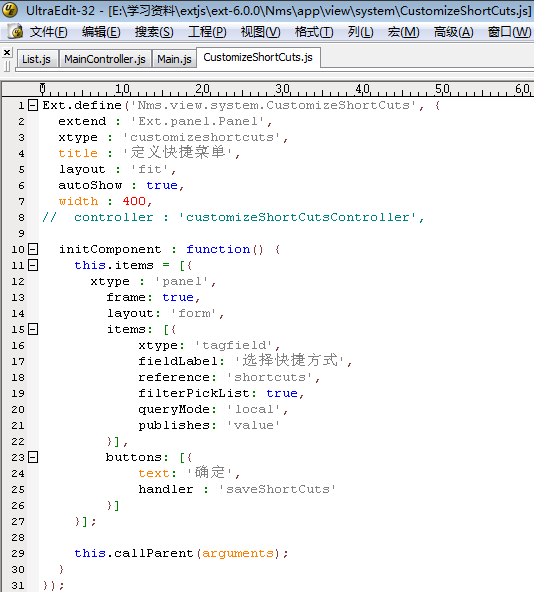
略。

## 开发实践

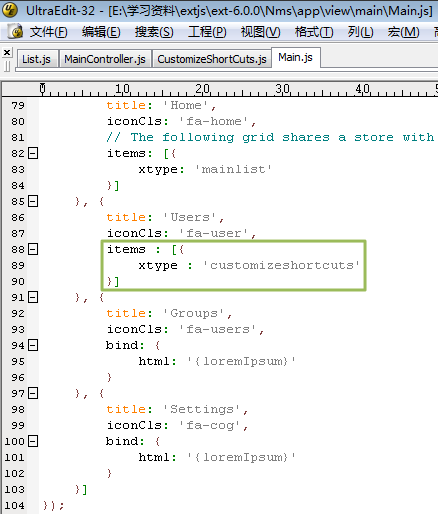
这一节，我们在原有程序上进行扩充，实现一个简单的页面。该页面用来定制快捷菜单。同时展示了在没有后台支撑的情况下，如何开发前台页面程序。

### 创建页面视图Nms.view.system.CustomizeShortCuts

创建一个js文件，如下图所示：



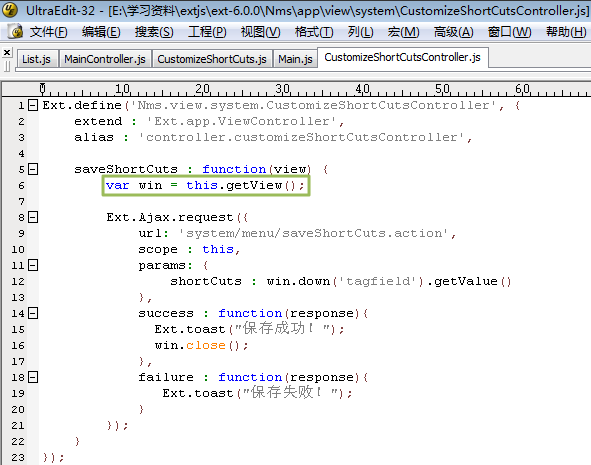
修改主视图，将第二个tab页面改为刚创建的视图。



刷新页面，看一下页面效果。

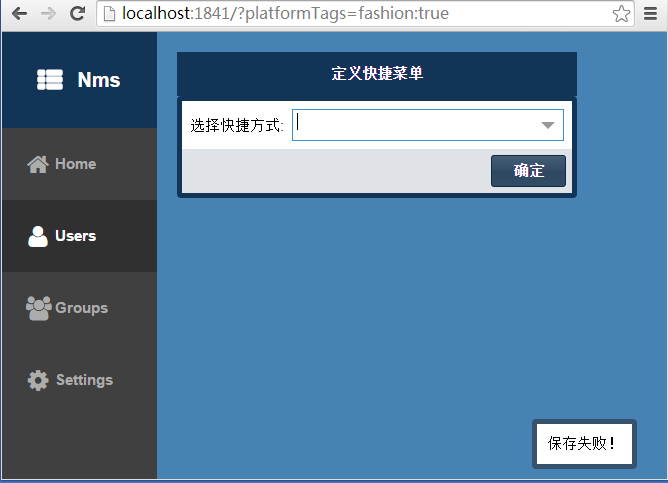


### 创建视图对应的控制器Nms.view.system.CustomizeShortCutsController



该控制器里有视图中【保存】按钮对应的事件处理函数。其中，getVeiw()函数获取的是注册到该控制器的视图，在这里指Nms.view.system.CustomizeShortCuts。我们需要注册该控制器到视图。在Nms.view.system.CustomizeShortCuts里放开注释的那一行。

刷新页面，看一下效果。



当点击确定按钮后，提示保存失败。

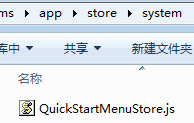
失败的原因在于，程序需要向url: 'system/menu/saveShortCuts.action'提交结果，而url对应的文件不存在，因此，我们在Nms\system下创建一个文件saveShortCuts.action内容为空。

再次运行，提示保存成功。

### 初始化视图Nms.view.system.CustomizeShortCuts的数据

我们注意到，视图的下拉框内容为空，因为我们没有为tagfield设置store。

### 创建store Nms.store.system.QuickStartMenuStore



内容如下：

Ext.define('Nms.store.system.QuickStartMenuStore', {

extend: 'Ext.data.Store',

model: 'Nms.model.system.MenuModel',

proxy: {

type: 'ajax',

api: {

read: 'system/menu/getMenusOfLeaf.action'

},

reader: {

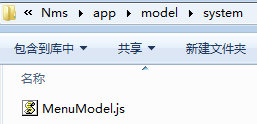
type: 'json'

}

}

});

### 创建model Nms.model.system.MenuModel



内容如下

Ext.define('Nms.model.system.MenuModel', {

extend : 'Ext.data.Model',

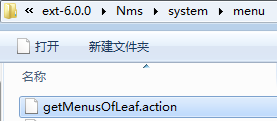
fields : [ 'id', 'text', 'cntType', 'icon', 'root', 'smallIcon', 'pId',

'pName', 'shortCut', 'quickStart', 'size', 'sort', 'mode' ]

});

### 创建模拟数据

我们的store设置了从read: '**system/menu/getMenusOfLeaf.action**'下面获取数据，因此，我们需要创建文件。



文件内容如下。

[ {

"id" :"5c919ee03772a0f50137731a772f9923",

"text" :"全网监视",

"root" :false,

"icon" :"resources/images/menustore/area.png",

"smallIcon" :"resources/images/smallmenustore/area.png",

"cntType" :"globaltopo",

"size" :-1,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731a772f9922",

"text" :"拓扑监测",

"root" :false,

"icon" :"resources/images/menustore/config.png",

"smallIcon" :"resources/images/smallmenustore/config.png",

"cntType" :"gistopo",

"size" :-1,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731a772f9924",

"text" :"物理拓扑",

"root" :false,

"icon" :"resources/images/menustore/topocfg.png",

"smallIcon" :"resources/images/smallmenustore/topocfg.png",

"cntType" :"topoconfiggraph",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731a772f9925",

"text" :"系统监视",

"root" :false,

"icon" :"resources/images/menustore/faultstatistic.png",

"smallIcon" :"resources/images/smallmenustore/faultstatistic.png",

"cntType" :"systemstatus",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731acaf12301",

"text" :"设备配置",

"root" :false,

"icon" :"resources/images/menustore/cmdterminal.png",

"smallIcon" :"resources/images/smallmenustore/cmdterminal.png",

"cntType" :"devframeconfigview",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731acaf2200",

"text" :"故障统计",

"root" :false,

"icon" :"resources/images/menustore/faulthistroy.png",

"smallIcon" :"resources/images/smallmenustore/faulthistroy.png",

"cntType" :"panel",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731acaf33300",

"text" :"任务管理",

"root" :false,

"icon" :"resources/images/menustore/buildstrategy.png",

"smallIcon" :"resources/images/smallmenustore/buildstrategy.png",

"cntType" :"panel",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731acaf32117",

"text" :"用户管理",

"root" :false,

"icon" :"resources/images/menustore/system.png",

"smallIcon" :"resources/images/smallmenustore/system.png",

"cntType" :"userlist",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731acaf32118",

"text" :"权限组管理",

"root" :false,

"icon" :"resources/images/menustore/usergroup.gif",

"smallIcon" :"resources/images/smallmenustore/usergroup.gif",

"cntType" :"usergroup",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731a772f9992",

"text" :"菜单管理",

"root" :false,

"icon" :"resources/images/menustore/menu.png",

"smallIcon" :"resources/images/smallmenustore/menu.png",

"cntType" :"menulist",

"size" :-1,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

"pname" :null,

"menu" :null

}, {

"id" :"5c919ee03772a0f50137731a772f9991",

"text" :"数据管理",

"root" :false,

"icon" :"resources/images/menustore/log.png",

"smallIcon" :"resources/images/smallmenustore/log.png",

"cntType" :"sysparam",

"size" :800,

"shortCut" :1,

"quickStart" :1,

"mode" :0,

"pid" :null,

"pName" :null,

"sort" :0,

"menuModel" :null,

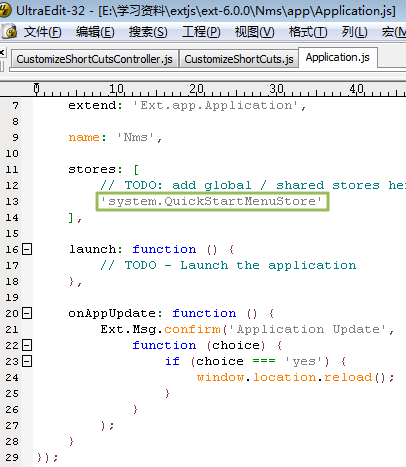
"pname" :null,

"menu" :null

} ]

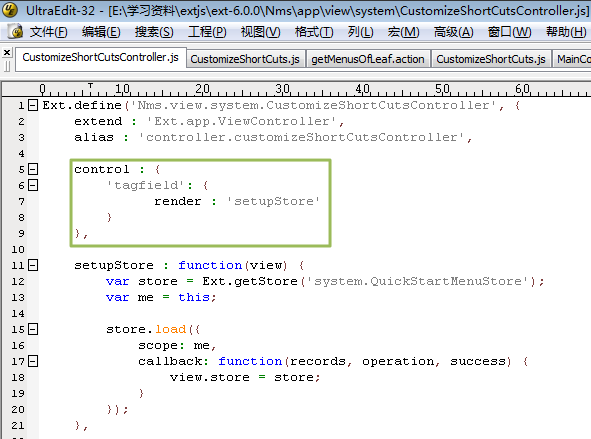
保存文件时，选择utf-8编码

### 配置全局store

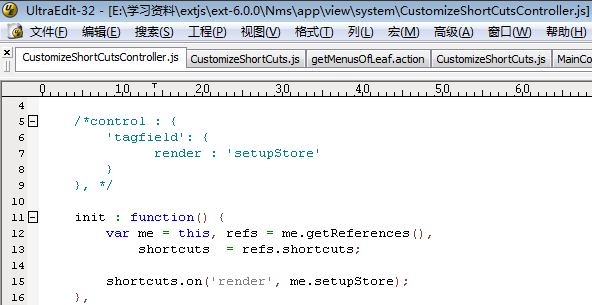


### 初始化视图的store

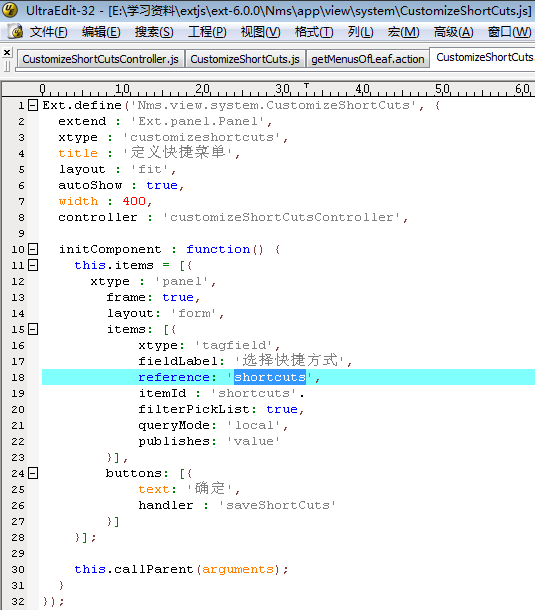
控制器的control配置项是另外一种注册事件响应函数的方式。



在init函数中使用视图的on函数注册事件响应函数



上述例子中，演示了viewController的getReferences方法的使用方式，可以快速定位指定控件。



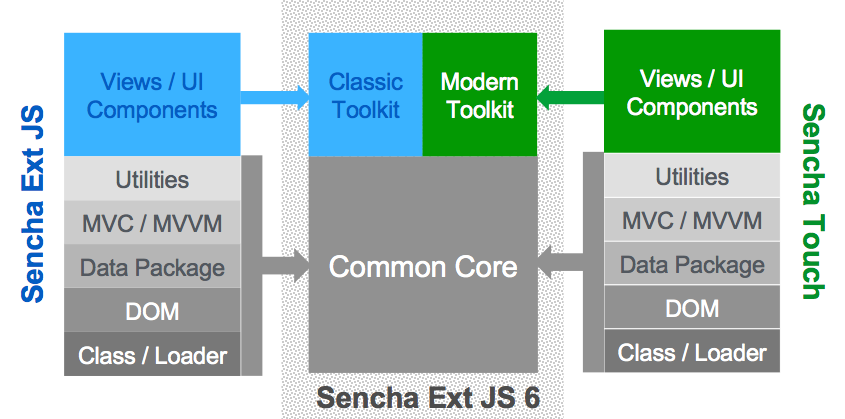
组件使用reference配置项进行配置。

## 设计通用应用程序

通用（universal）应用程序指的是既能够在桌面浏览器（**classic**）上使用，又能在平板或智能手机浏览器（**modern**）上使用的应用。这里强调的是**通用**，而不是仅能用于纯classic或纯modern的应用程序。

### 背景

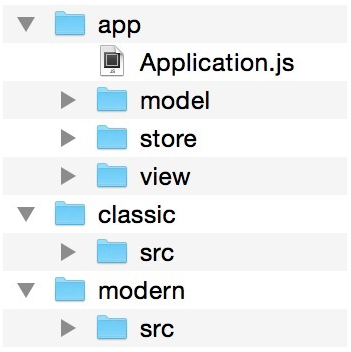
从6.0开始，Ext JS已经将Sencha Touch合二为一了。



Sencha已经将这些组件家族放入名为“classic”和“modern”的两个工具包（**Toolkit**）中。工具包classic中的是Ext JS组件集，而modern工具包中的是Sencha Touch 2组件集。每一个工具包的目标是去支持经典或现代的浏览器。Classic支持老旧的浏览器，如IE8，而modern则是针对现代浏览器，完全支持HTML5和CSS3。

通过设计，classic和modern工具包的目标是不同时代的浏览器。如果编写一个既保护classic和modern用户界面的应用程序，用户能看到到哪一个取决于运行环境。但两个工具包的视图组件不能混合使用。但这两个工具包都共享核心代码，如类系统、数据包以及视图-控制器-视图模型架构等等。

正因为如此，一个Ext JS 6通用应用程序可以将源代码划分为三个部分：classic、modern和共享。



### 创建通用App

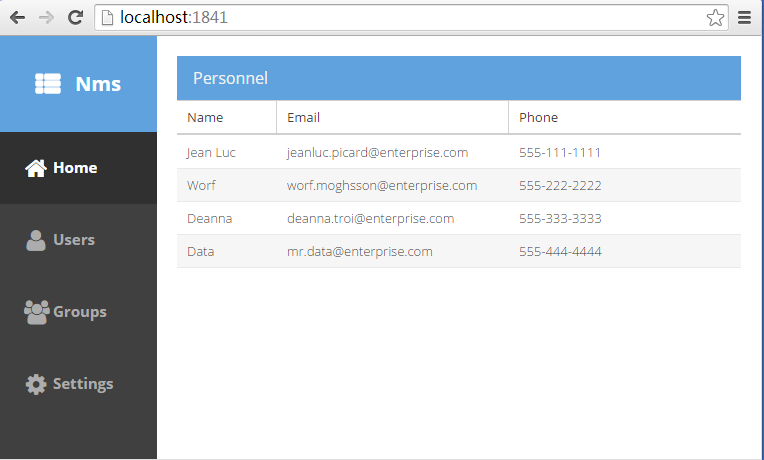
借助Sencha Cmd工具，我们创建一个通用应用，如App。命令如下：

sencha -sdk ./ext-6.0.0 generate app App ./App

进入到App目录，运行sencha app watch modern，浏览器输入

<http://localhost:1841>

可以看到工具为我们生成的桌面应用界面。

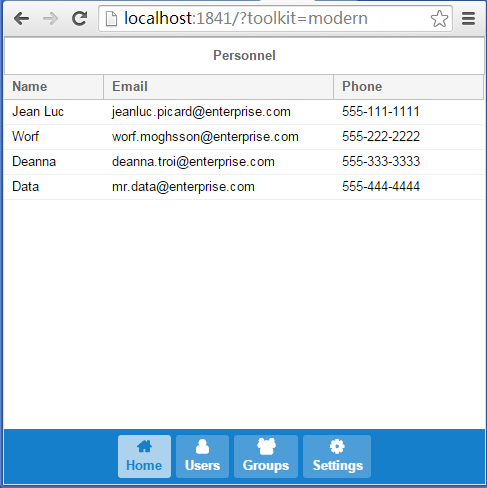


浏览器输入

[http://localhost:1841/**?toolkit=modern**](http://localhost:1841/?toolkit=modern)

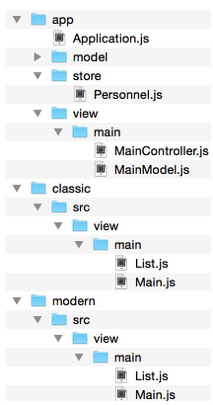
看一下在平板或手机上的运行效果。

点击不同的标签，查看页面切换效果。



### 共享代码

应用程序在app/store放至了一个共享的store类Personnel.js，该类还有一个别名。无论是classic还剩modern的视图都可以请求该类并使用别名来创建他们自己的实例。在app/view/main目录下还有一个共享的控制器MainController.js，以及在app/view/main目录下会有一个共享的视图模型MainModel.js。这些类都可以被classsic或modern文件夹的主视图请求。如下图所示。

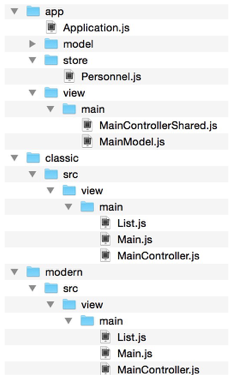


### 专用控制器

如果打开MainController.js，会看到只有一个onItemSelected方法用来在用户选择一个条目时显示一个告警。当在grid中单击一个条目时，该方法既可以运行在classic视图，也可以运行在modern视图。

如果希望考虑过在classic视图与modern视图中做不同的实现，比如在classic视图中使用Ext.Msg.alert()，而在modern视图中使用Ext.toast()。这可通过创建从共享控制器扩展的特定视图的控制器来实现。

要实现上面的描述，需要在app文件夹将共享的控制器改名为MainControllerShared。然后创建两个新的控制器，一个位于classic，而另一个位于modern。这两个都要扩展自共享的控制器。这样，classic的主视图将请求在classic中的控制器，而modern视图则请求在modern中的控制器。这些控制器中的每一个都有一个别名，且都扩展自共享控制器。然后，移除共享控制器中的方法和别名并实现特定视图的版本。目录结构如下图所示。



由于名称MyApp.view.main.MainController是相同的，但它们的位置在文件系统中已经从app文件夹修改为classic和modern文件夹，因此需要运行一下sencha app refresh。

修改后的各文件内容如下：

**app/view/main/MainControllerShared.js**

Ext.define('App.view.main.MainControllerShared', {

extend: 'Ext.app.ViewController'

});

**app/classic/src/view/main/MainControllerShared.js**

Ext.define('App.view.main.MainController', {

extend: 'App.view.main.MainControllerShared',

alias: 'controller.main',

onItemSelected: function (sender, record) {

Ext.Msg.alert('Star Trek', record.data.name);

}

});

**modern/src/view/main/MainControllerShared.js**

Ext.define('App.view.main.MainController', {

extend: 'App.view.main.MainControllerShared',

alias: 'controller.main',

onItemSelected: function (sender, record) {

Ext.toast(record.data.name);

}

});

说明：在这里不再有共享的控制器逻辑，后续有机会还是可以将共享的事件处理程序放回共享的控制器，或者可以实现一个initViewModel方法放置需要的逻辑。如果子类控制器需要initViewModel方法，要确保这些方法的第一句是this.callParent(arguments); 来确保父类的方法被执行。

### 专用ViewModel

首先看一下共享的视图模型，它包含了两个数据属性：name和loremlpsum。这些都会在两个视图中使用。下面替换classic视图Users标签页中gird的store，该store只使用于classic视图。

首先，将classic/src/view/main/Main.js中的第二个标签修改为以下代码：

{

iconCls: 'fa-user',

xtype: 'grid',

title: 'Beatles',

columns: [{

text: 'Name',

dataIndex: 'first',

flex: 1

}],

bind: {

**store: '{beatles}**'

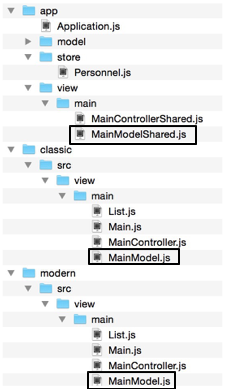
}

}

代码将创建一个绑定视图模型中名为“{beatles}”的store的grid，

该store目前尚不存在，需要在代码里创建。

类似上一节，我们同样抽象出一个共享的ViewModel，命名为MainViewModelShared，里面保留属性name和loremlpsum，因为这两个属性是classic和modern共用的。然后分别建立classic和modern专有的MainViewModel。如下图所示。



各文件代码如下：

**App/app/view/main/ MainModelShared.js**

Ext.define('App.view.main.MainModelShared', {

extend: 'Ext.app.ViewModel',

data: {

name: 'App',

loremIpsum: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit…..'

}

});

**classic/src/view/main/ MainModel.js**

Ext.define('App.view.main.MainModel', {

extend: 'App.view.main.MainModelShared',

alias: 'viewmodel.main',

stores: {

beatles: {

autoLoad: true,

data: [

{first: 'John'},

{first: 'Paul'},

{first: 'George'},

{first: 'Ringo'}

],

proxy: {type: 'memory'}

}

}

});

**modern/src/view/main/ MainModel.js**

Ext.define('App.view.main.MainModel', {

extend: 'App.view.main.MainModelShared',

alias: 'viewmodel.main'

});

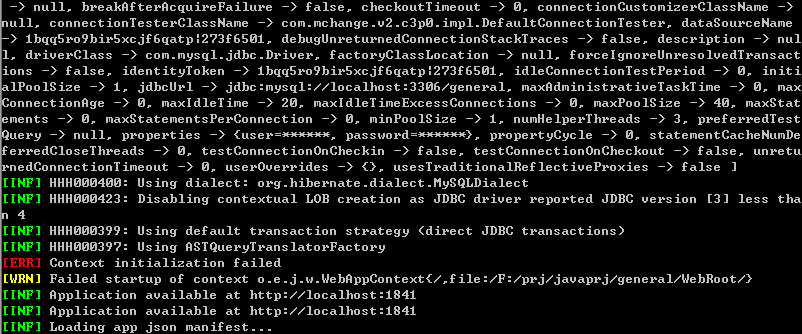
## 遇到的问题

这节讲述实际开发过程中遇到的问题，仅供参考。

### Context initialization failed

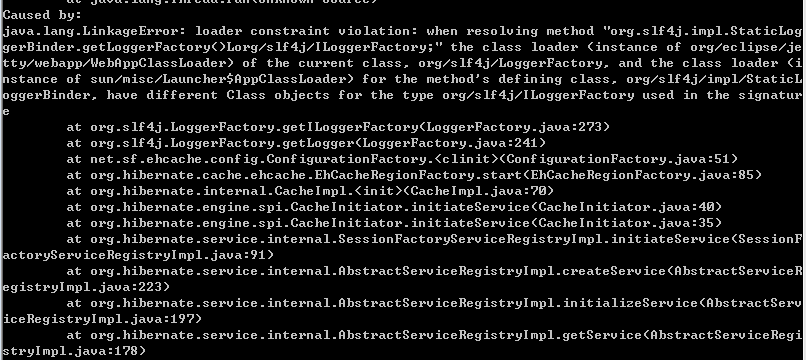
后台采用SpringMVC4+Hibernate4，并集成了用户管理等基础模块后，运行Sencha app watch

提示Context initialization failed。除此之外，没有任何信息可以参考，不知道程序哪里报错。



下面记录了问题定位过程。

1. 首先注释掉applicationContext.xml的所有内容、dispatcher-servlet.xml的所有内容，依旧报错；
2. 继续往上注释，把web.xml的spring相关的配置注释掉，虽然不再报“Context initialization failed”错误，但spring没有初始化，所以程序不能正常运行。
3. 下载jetty-distribution-9.2.13.v20150730，把程序放在webapps目录下，程序运行正常；
4. 下载Sencha Cmd使用的8.1.17版本jetty-distribution-8.1.17.v20150415，同样程序正常；
5. 尝试把使用Sencha\Cmd\6.0.0.202\lib下的jetty相关jetty-\* -8.1.7.v20120910.jar删掉并替换为jetty-\*-8.1.17.v20150415.jar，提示找不到类，虽然.jar里确实存在该类。如果把jetty-\*-8.1.17.v20150415.jar改名为jetty-\* -8.1.7.v20120910.jar，不再提示找不到类的错，（说明Sencha Cmd使用的jar文件绑定了文件名），但还是提示[Err]；
6. 由于Sencha Cmd使用的jar不能改名，如果拷贝其他jar文件到Sencha\Cmd\6.0.0.202\lib下，也不能解决缺少jar（假设可能缺少某个依赖库）的问题；
7. 把jetty-\*-8.1.17.v20150415.jar拷贝到Sencha\Cmd\6.0.0.202\jre\lib\ext目录下，再次运行sencha app watch，错误的详细信息被打印出来；



可以看出问题出在slf4j库上了。

我们把程序的WebRoot\WEB-INF\lib\slf4j-api-1.6.1.jar库删掉，然后再运行sencha app watch，程序运行正常；我们把Sencha\Cmd\6.0.0.202\jre\lib\ext下刚才拷贝过来的库删除，再运行sencha app watch，程序运行没问题；但如果我们发布到tomcat下，提示

我们把sencha cmd使用的slf4j-jdk14-1.6.6.jar 拷贝到WebRoot\WEB-INF\lib\下，并恢复WebRoot\WEB-INF\lib\slf4j-api-1.6.1.jar，Sencha Cmd和Tomcat下都正常。

## 其他

这节讲述上述没有涉及的部分。

### 本地化

在Ext JS 6中，Ext JS属于经典工具包，而本地化是包含在经典工具包中，因而在app.json中，要添加本地化包，必须在classic成员中添加，代码如下：

"classic": {

"js": [

{

// Remove this entry to individually load sources from the framework.

"path": "${framework.dir}/build/ext-all-rtl-debug.js"

}

],

// 以下是新增的添加本地化包的代码

"requires" : [

"locale"

],

"locale": "zh\_CN"

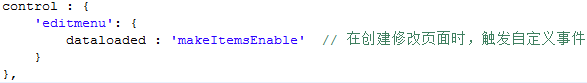
}

### 自定义事件

ExtJS6.0使用自定义事件时，不再要求首先使用addEventListener，



在控制器中，设置事件的处理函数



# 程序发布

管理员身份运行dos，进入开发目录Nms，执行

**sencha app build**

编译结束后，在build/production/Nms下生成目标程序。

编译过程中可能会出现错误。

sencha –debug app build native testing > c:\a.log