第4章　使用Sencha Cmd创建应用程序

如果对 Sencha Cmd 不熟悉或没有了解过，一定会很诧异为什么要用这个东西来创建应用程序，使用它又有什么好处呢？本章将会解答这些疑问，并介绍如何使用它来创建应用程序，以及应用程序创建后如何开始进行开发。

4.1　概　述

从 Ext JS 4.1.1a 开始，为了配合 Ext JS 和 Sencha Touch 的开发，特意设计了 Sencha Cmd 这个跨平台的命令行工具。在此之前，这样的工具也是有的，名字为 SDK Tools，但是它的作用和意义对开发来说显得并不突出。从 Ext JS 4.1.1a 和 Sencha Touch 2.1 之后，Sencha Cmd 的作用和意义就显得很突出了，它的主要作用包括以下两点。

- 编译脚本：使用 Sencha Cmd 创建的应用程序，编写完成后，可通过 Sencha Cmd 的生成（build）功能，将零散的应用程序脚本打包成为一个下载包，而且可以把没有使用到的组件清理出去。这样做的好处是显而易见的。如果不使用该功能，就必须在页面中包含 ext-all.js 文件，下载全部的 Ext JS 组件，由于该文件比较大，会造成用户体验上的问题。在使用 ext-all.js 文件的同时，还需要使用动态加载功能，在需要的时候向服务器请求必要的脚本文件，在界面比较复杂、脚本文件比较多的情况下，就会发送大量的请求，这不单加重了服务器的负担，还严重影响了用户体验。使用生成功能后，则可将所有应用程序内的脚本打包为一个文件发布，这对于部署和用户体验都是有极大好处的。
- 编译主题：这是 Sencha Cmd 的作用体现得最突出的一点。一直以来，Ext JS 最突出的问题就是主题的缺乏，而主题的修改也不方便。从 Ext JS 4 开始，引入了 Compass 和 Sass，这使得创建和修改主题更加便利，但是，要编译主题并兼顾自定义样式，还是一件很麻烦的事，使用 Sencha Cmd 则使事情变得简单了。在应用程序的开发过程中，就可以修改主题以及对样式进行定义，然后使用生成功能，在打包脚本的同时打包主题和样式。在 Ext JS 6 中引用了新的 SASS 编译器——Fashion，不再需要使用 Compass 和 Sass 了。

4.2 安装 Sencha Cmd

4.2.1 运行环境配置

在编写本书时，笔者使用的 Sencha Cmd 版本为 6.2.1.29。如果你的版本太低，请使用 sencha upgrade 命令升级 Cmd。如果升级到最新的版本出现生成错误，就去官方网站寻找 6.2.1.29 版本。笔者在编写本书的时候，碰到过由于 Sencha Cmd 版本不同而造成生成出现错误的情况。

由于新版的 Sencha Cmd 已经内置 JRE 安装、不再需要 Compass，因而不再需要先安装 JRE 和 Ruby，无疑减少了 JRE 版本和 Ruby 版本对 Sencha Cmd 的影响，减少了配置错误。

Sencha Cmd 的下载地址是 https://www.sencha.com/products/extjs/cmd-download/，根据机器所使用的平台下载相应的安装文件就可以了。这里下载的是基于 Windows 的 64 位版本，下载的文件是 SenchaCmd-6.2.1-windows-64bit.zip。

4.2.2 安装 Sencha Cmd

将 SenchaCmd-6.2.1-windows-64bit.zip 文件解压，会看到 SenchaCmd-6.2.1.29-windows-64bit.exe 文件，双击该文件将会看到如图 4-1 所示的安装窗口。

图 4-1 Sencha Cmd 的安装窗口

单击 Next 按钮将会看到如图 4-2 所示的许可协议窗口。

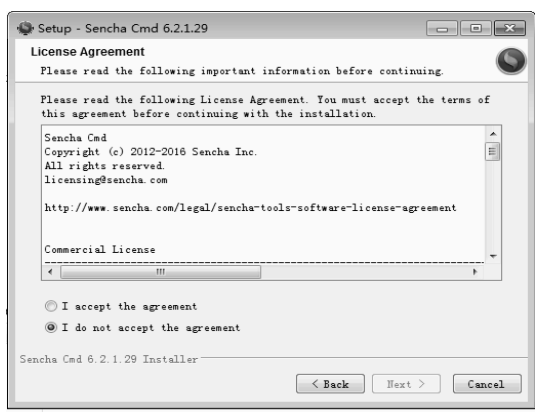

图 4-2 许可协议窗口

选择 I accept the agreement（我接受许可）之后，单击 Next 按钮，会看到如图 4-3 所示的设置安装目录窗口。

在设置安装目录窗口，可将安装目录修改为适合自己的目录，这里是安装在 D:\Program Files (x86)\Sencha\Cmd 目录的。目录修改完成后，单击 Next 按钮，会看到如图 4-4 所示的选择组件窗口。

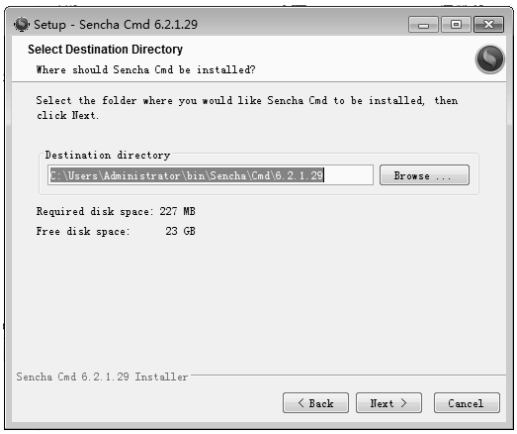

图 4-3　设置安装目录窗口

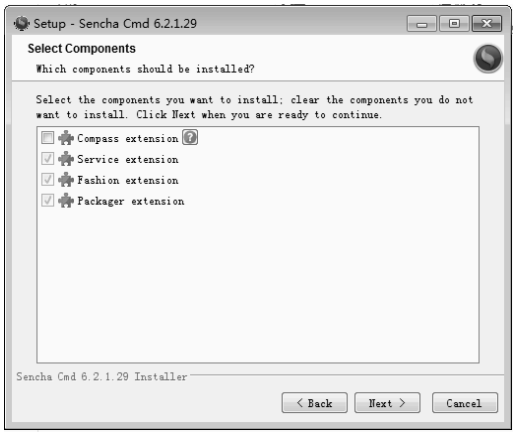

图 4-4　选择组件窗口

在选择组件窗口中，可看到 Compass 扩展，如果还需要维护 Ext JS 4 或 5 的项目，可以选择安装，在这里就不安装了。单击 Next 按钮，进入如图 4-5 所示的选择附加任务窗口。

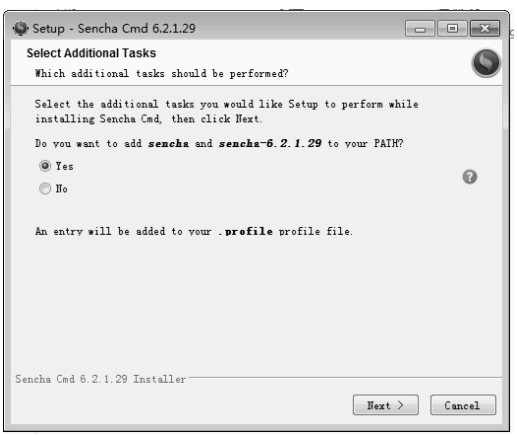

图 4-5　选择附加任务窗口

在选择附加任务窗口中，主要的任务是将 sencha 命令添加到全局路径中。这样，在打开命令提示符窗口后，就不需要输入完整路径来运行命令了。这里最好是选择 Yes，让安装程序自己去添加全局路径。单击 Next 按钮后就进入安装了。

安装结束后，单击 Finish 按钮完成安装。

4.2.3　验证安装

在 Sencha Cmd 安装以后，打开一个命令提示符窗口，并在窗口内输入 sencha 命令来验证安装

是否成功。如果输入 sencha 命令后，在窗口内输出如代码清单 4-1 所示的信息，就说明安装已经成功。

代码清单 4-1　验证 Sencha Cmd 的安装

```
    Sencha Cmd v6.2.1.29
    Sencha Cmd provides several categories of commands and some global switches.
In
    most cases, the first step is to generate an application based on a Sencha SDK
    such as Ext JS or Sencha Touch:

       sencha -sdk /path/to/sdk generate app MyApp /path/to/myapp

    Sencha Cmd supports Ext JS 4.1.1a and higher and Sencha Touch 2.1 and higher.

    To get help on commands use the help command:

       sencha help generate app

    For more information on using Sencha Cmd, consult the guides found here:

    http://docs.sencha.com/cmd/


    Options
      * --beta, -be - Enable beta package repositories
      * --cwd, -cw - Sets the directory from which commands should execute
      * --debug, -d - Sets log level to higher verbosity
      * --info, -i - Sets log level to default
      * --nologo, -n - Suppress the initial Sencha Cmd version display
      * --plain, -pl - enables plain logging output (no highlighting)
      * --quiet, -q - Sets log level to warnings and errors only
      * --sdk-path, -sd - The location of the SDK to use for non-app commands
      * --strict, -st - Treats warnings as errors, exiting with error if any warnings
are present
      * --time, -ti - Display the execution time after executing all commands

    Categories
      * app - Perform various application build processes
      * compile - Compile sources to produce concatenated output and metadata
      * cordova - Quick init Support for Cordova
      * diag - Perform diagnostic operations on Sencha Cmd
      * framework - Commands to manage frameworks in the current workspace
      * fs - Utility commands to work with files
      * generate - Generates models, controllers, etc. or an entire application
      * manager - Commands for interacting with Sencha Web Application Manager.
      * manifest - Extract class metadata
      * package - Manages local and remote packages
      * phonegap - Quick init support for PhoneGap
      * repository - Manage local repository and remote repository connections
      * template - Commands for working with templates
      * theme - Commands for low-level operations on themes
      * web - Manages a simple HTTP file server
      * workspace - Commands to perform actions on the current workspace

    Commands
      * ant - Invoke Ant with helpful properties back to Sencha Cmd
```

```
      * audit - Search from the current folder for Sencha frameworks and report their
license
      * build - Builds a project from a legacy JSB3 file.
      * config - Load a properties file or sets a configuration property
      * help - Displays help for commands
      * js - Executes arbitrary JavaScript file(s)
      * switch - Manage the active Sencha Cmd version
      * upgrade - Upgrades Sencha Cmd
      * which - Displays the path to the current version of Sencha Cmd
```

从代码清单4-1可以看到，现在使用的Cmd版本是6.2.1.29。在版本号后面，紧接着列出的sencha语句是用来创建应用程序的 sencha 语句。下面的另一句 sencha 语句是用来查看帮助信息的。

紧接着列出了 Options、Categories 和 Commands 三种命令参数。其中，Options 是命令的可选参数（如-sdk），用来指定 SDK 的路径；Categories 是命令的类别（如 app 类别），配合相应的命令，可用来生成应用程序；Commands 是具体的命令（如 build），和 app 配合，可用来生成应用程序。

4.2.4 语法

Sencha Cmd 的基本语法规则如下：

```
    sencha [category] [command] [options...] [arguments...]
```

虽然一般语法是这样，但也不是完全这样，例如用来创建应用程序的 options 项-sdk，就必须在 category 项之前。总之，用法比较灵活，也没统一规定，使用的时候需要注意。

4.3 创建应用程序

4.3.1 创建应用程序前要考虑的问题

在 Ext JS 4 中，使用 Cmd 来创建应用程序，只有一种模式；而在 Ext JS 6 中，将 Sencha Touch 合并到 Ext JS 后，就衍生出了通用应用程序、经典应用程序和现代应用程序三种模式。经典应用程序就是之前的 Ext JS 应用程序。现代应用程序指的就是之前的 Sencha Touch 应用程序。通用应用程序，顾名思义就是兼有经典应用程序和现代应用程序的应用程序。

通用应用程序的目的很明确，就是打通桌面与移动设备之间的鸿沟，开发一套系统就能适用于所有平台，应用程序会根据设备来选择是运行经典应用程序还是现代应用程序。

为什么要这么麻烦呢？为什么不能通过响应式布局来实现不同设备显示不同的视图，而要硬生生地将它们区分为经典应用程序和现代应用程序呢？要将 Ext JS 和 Sencha Touch 完全合并在一起，还有许多开发工作要做，是一项非常费时的浩大工程，譬如核心的融合、组件的融合以及主题的融合，都非一朝一夕能完成的，因而，采取渐进式的方式去逐步推进是最有效的办法。Ext JS 6 就是这种渐进式开发的第一个版本，先把核心部分融合在一起，以满足目前市场的需求。

Ext JS 6 的这种半融合模式，对开发人员来说，具有相当大的挑战性。因为，为了区分经典模式视图和现代模式视图，Ext JS 6 在应用程序目录架构上做了一些修改。之前版本的应用程序目录

架构只有一个 app 文件夹来存放应用程序的类文件。在新的架构下，添加了 calssic 和 modern 两个文件夹，分别用来存放经典应用程序的类文件和现代应用程序的类文件。这样的划分虽然很清晰，但带来一个问题，Ext JS 的自动加载机制根据类名找不到具体的文件位置了。造成这个问题的主要原因是自动加载机制会将命名空间默认指向 app 文件夹。当然，你也可以自己修改这个目录，但无论你怎么修改，也不可能将命名空间同时指向 app、classic 和 modern 这三个文件夹，从而造成无法加载类文件的错误。为了解决这个问题，Sencha 采用了在 bootstrap.json 文件中，使用指定路径的方式来指定类的路径，这也就是每次添加新的类后都需要执行一次 sencha app build，或使用 sencha app watch 来监控应用程序，自动生成项目的原因。这项操作的主要作用是更新 bottstrap.json 文件中类文件的路径。

在使用 Ext JS 6 初期，一般习惯从创建一个通用应用程序开始，不过在开发过程中，由于类的路径问题，最后还是会放弃。笔者所做的大部分项目都只是应用于桌面平台的，因而在后期，基本都只创建经典应用程序，中间会有些现代应用程序，也只是根据需要创建现代应用程序，不再使用通用应用程序进行开发，个中原因主要是路径问题。

以上说了那么多，主要目的是提醒大家，在选择 Ext JS 6 的开发模式上，需要谨慎应对。在不需要使用到通用应用程序的时候，就不要去使用通用应用程序的开发模式，正如本书的示例，不需要考虑现代应用程序这种模式，就只创建经典模式的应用程序。

4.3.2 开始创建应用程序

如果不清楚如何创建经典应用程序，可以在命令提示符窗口输入以下命令来获取帮助：

```
    sencha help generate app
```

在显示的帮助信息中，会看到以下信息：

```
    For Ext JS 6, by default, this application will be a Universal Application.
    To override this and select a particular toolkit, you can use either of
    these variations:

    sencha generate app -ext -classic MyAppName ./MyAppPath
    sencha generate app -ext -modern MyAppName ./MyAppPath
```

通过帮助信息的说明，可以知道，在默认情况下，创建的是通用应用程序，如果要指定工具包，可以使用 classic 或 modern 参数。

下面来创建本书系统所需要使用到的经典模式应用程序。打开一个命令提示符窗口，使用 cd 命令切换到合适的文件夹，这里是在 D:\workspace 文件夹下执行操作的。在 workspace 文件夹下，已经把 Ext JS 6 的框架文件解压到 ext6 文件夹。在命令提示符窗口中输入以下命令来创建应用程序：

```
    sencha -sdk d:\workspace\ext6 generate app -classic SimpleCMS
d:\workspace\SimpleCMS
```

在以上命令中，sdk 参数用来指定框架文件的位置。在 app 后面的 SimpleCMS 表示的是应用

程序的名称，而在框架中，SimpleCMS 也将是应用程序的命名空间，自定义组件的类名将以 SimpleCMS 为起始名称。最后的“d:\workspace\SimpleCMS”表示创建的应用程序将输出到 workspace\SimpleCMS 文件夹下。

命令运行后，如果创建应用程序成功，可在命令提示符窗口看到代码清单 4-2 所示的输出。如果出现了红色的“[ERR]”信息，就说明应用程序已经创建失败了，需要根据“[ERR]”后的信息去找出失败的原因。

代码清单 4-2　创建应用程序时的输出

```
    Sencha Cmd v6.2.1.29
    [INF] Copying framework to d:\workspace\SimpleCMS\ext
    [INF] Using GPL version of Ext JS version 6.2.0.981 from
d:\workspace\SimpleCMS\ext.
    [INF] The implications of using GPL version can be found here
(http://www.sencha.com/products/extjs/licensing).
    [INF] Processing Build Descriptor : default
    [INF] Loading app json manifest...
    [INF] Appending content to d:\workspace\SimpleCMS\bootstrap.js
    [INF] Writing content to d:\workspace\SimpleCMS\bootstrap.json
    [INF] merging 237 input resources into
d:\workspace\SimpleCMS\build\development\SimpleCMS\resources
    [INF] merged 237 resources into
d:\workspace\SimpleCMS\build\development\SimpleCMS\resources
    [INF] merging 17 input resources into
d:\workspace\SimpleCMS\build\development\SimpleCMS
    [INF] merged 17 resources into
d:\workspace\SimpleCMS\build\development\SimpleCMS
    [INF] Writing content to d:\workspace\SimpleCMS\sass\example\bootstrap.json
    [INF] Writing content to d:\workspace\SimpleCMS\sass\example\bootstrap.js
    [INF] writing sass content to
d:\workspace\SimpleCMS\build\temp\development\SimpleCMS\sass\SimpleCMS-all.scs
s.tmp
    [INF] appending sass content to
d:\workspace\SimpleCMS\build\temp\development\SimpleCMS\sass\SimpleCMS-all.scs
s.tmp
    [INF] appending sass content to
d:\workspace\SimpleCMS\build\temp\development\SimpleCMS\sass\SimpleCMS-all.scs
s.tmp
    [INF] writing sass content to
d:\workspace\SimpleCMS\build\temp\development\SimpleCMS\sass\config.rb
    [INF] Writing content to
d:\workspace\SimpleCMS\build\development\SimpleCMS\app.json
    [LOG] Building
d:/workspace/SimpleCMS/build/temp/development/SimpleCMS/sass/SimpleCMS-all.scs
s
    [WRN] @theme-background-image: Theme image not found:
images/menu/default-menubar-group-checked.png
    Exiting with code 0
    [INF] Appending content to d:\workspace\SimpleCMS\bootstrap.js
    [INF] Writing content to d:\workspace\SimpleCMS\bootstrap.json
```

从代码清单 4-2 中可以看到，sencha 命令首先做的工作是把框架文件复制到 d:\workspace\SimpleCMS\ext 文件夹，然后创建 bootstrap.js 和 bootstrap.json 文件。接下来的代码是

使用“sencha app build --development”命令生成了一次应用程序的输出代码，该过程的作用是创建应用程序调试所需的样式文件，把示例代码的类文件的路径写入 bootstrap.json 文件。在输出中有一个标记为“[WRN]”的警告，如果不是强迫症患者，不用介意，这个对应用程序没有影响，估计是框架自身的臭虫。

应用程序创建后，可以把文件夹放到 Web 服务器下来测试刚生成的应用程序。如果一时找不到 IIS 或其他 Web 服务器，也可以使用 Sencha Cmd 的 Web 服务器功能来测试。在命令提示符窗口中输入以下命令：

```
sencha fs web -port 8000 start -map d:\workspace\SimpleCMS
```

在命令中，port 选项的作用是定义访问端口，这里将使用 8000 作为端口；map 选项用来指定 Web 的访问目录。

命令执行后，会在命令提示符窗口看到以下输出：

```
Sencha Cmd v6.2.1.29
[INF] Starting server on port : 8000
[INF] Mapping http://localhost:8000/ to d:\Workspace\SimpleCMS...
[INF] Server started at port : 8000
```

从输出中可以看到，Web 服务已经启动，启动的端口为 8000，地址 http://localhost:8000/已经映射到 d:\Workspace\SimpleCMS 文件夹了。现在可以通过 http://localhost:8000/来访问应用程序了。

在浏览器中打开 http://localhost:8000/，会看到如图 4-6 所示的页面效果。从图中可以看到，Ext JS 6 应用程序默认使用了现在比较流行的管理系统风格，而主题是海卫一主题。

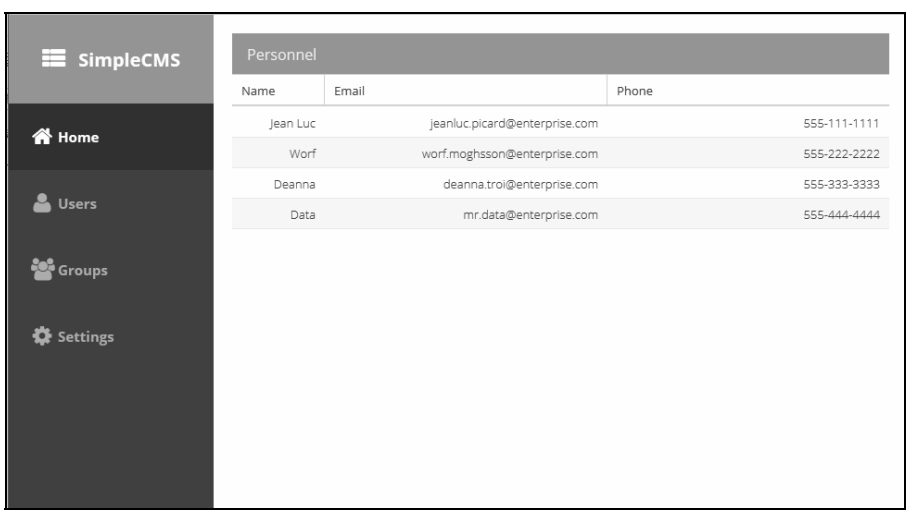

图 4-6　SimpleCMS 应用程序的默认效果

4.4 应用程序的结构

4.4.1 目录结构

应用程序创建后的目录结构如图 4-7 所示，以下是这些目录和文件的说明。

- .sencha 文件夹：在该文件夹下有 app 和 workspace 两个文件夹，主要包含一些生成应用程序所需的配置文件。一般情况下，不需要修改.sencha 文件夹下的任何文件。
- app 文件夹：这是后续开发将要使用到的文件夹。在该文件夹下有 model、store 和 view 三个文件夹，分别用来存放应用程序的控制器、模型、存储和视图文件。在 app 文件夹下，有一个 Application.js 文件，该文件是用来加载视图、控制器和存储的，在后续开发中随着视图、控制器和存储的添加，需要不断修改该文件。在 app\view 文件夹下，已经生成了 Viewport.js 和 Main.js 两个主界面视图，后续的开发将从修改这两个文件开始。
- ext 文件夹：Ext JS 的框架文件所在的文件夹。
- overrides 文件夹：当需要重写框架的类时，可存放在这里。
- packages 文件夹：用来放置打包后的文件。具体可参阅 API 的指南部分 Sencha Cmd 节点下关于 Sencha Cmd Packages 的介绍。
- resources 文件夹：存放应用程序的资源文件。
- sass 文件夹：存放自定义的主题或样式文件。
- app.js：应用程序的启动文件。
- app.json：应用程序的配置文件。
- bootstrap.css：样式引导文件。代替直接加载 ext-all.css 或其他编译好的样式文件。该文件会在生成应用程序时被修改。
- bootstrap.js：脚本引导文件，用于加载应用程序所需的脚本和样式文件。
- bootstrap.json：用来保存应用程序中脚本和样式文件的路径，为 bootstrap.js 脚本和样式文件提供加载地址。
- build.xml：生成应用程序时需要使用到的配置文件。
- index.html：应用程序启动时需要使用到的页面文件。由于后续使用 Asp.NET MVC 4 来进行开发，因此需要将该文件内的内容转移到首页文件中。
- workspace.json：一个大的项目可能是由几个小项目组成的，而这些小项目会有一些共享代码，这时候就需要用到工作空间（workspace）来解决共享代码的问题，本文件就是用来定义共享空间的。在本书中不需要使用这个，就不深入研究了。

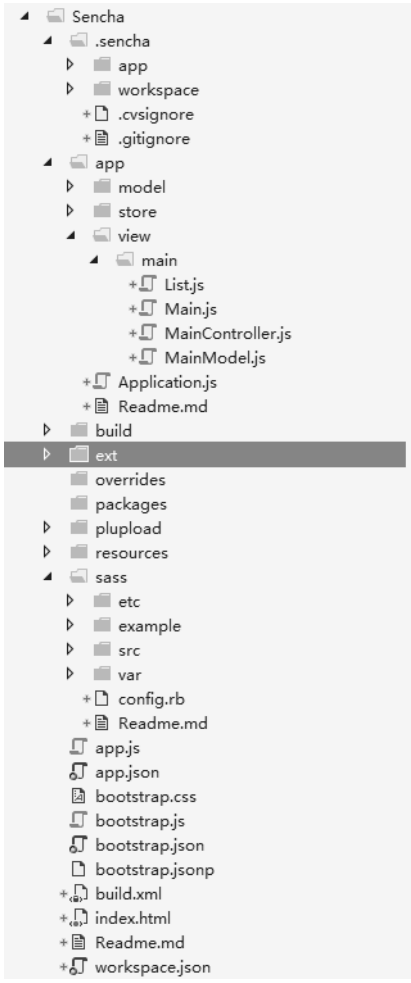

图 4-7　应用程序的目录结构

4.4.2　index.html

文件 index.html 的代码如代码清单 4-3 所示。从代码中可以看到，在 index.html 文件中只是加载了 bootstrap.js 文件，并没有加载其他脚本文件，其余脚本和样式文件的加载全靠 bootstrap.js 文件来实现。bootstrap.js 是通过 bootstrap.json 文件来获取加载的脚本和样式文件的。在开发通用应用程序的时候，了解这个很重要，这也是为什么开发通用应用程序需要经常生成的原因，需要不断地更新 bootstrap.json，不然就会出现找不到类的错误。开发经典模式或者现代模式的应用程序就没有这个麻烦了，因为框架在找不到类的时候，会根据默认机制去 app 文件夹下找文件，不会出现找不到类的情况。当然，写错了类名还是会找不到类的。

代码清单 4-3　index.html

```
<!DOCTYPE HTML>
<html manifest="">
<head>
```

```
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=10, user-scalable=yes">

        <title>SimpleCMS</title>

        <!--
        <script type="text/javascript">
            var Ext = Ext || {}; // Ext namespace won't be defined yet...

            // This function is called by the Microloader after it has performed basic
            // device detection. The results are provided in the "tags" object. You
            // can use these tags here or even add custom tags. These can be used
            // by platform filters in your manifest or by platformConfig expressions
            // in your app.
            Ext.beforeLoad = function (tags) {
                var s = location.search,  // the query string (ex "?foo=1&bar")
                    profile;

                // For testing look for "?classic" or "?modern" in the URL to override
                // device detection default.
                //
                if (s.match(/\bclassic\b/)) {
                    profile = 'classic';
                }
                else if (s.match(/\bmodern\b/)) {
                    profile = 'modern';
                }
                else {
                    profile = tags.desktop ? 'classic' : 'modern';
                    //profile = tags.phone ? 'modern' : 'classic';
                }

                Ext.manifest = profile; // this name must match a build profile name

                // This function is called once the manifest is available but before
                // any data is pulled from it.
                //
                //return function (manifest) {
                    // peek at / modify the manifest object
                //};
            };
        </script>
        -->

        <!-- The line below must be kept intact for Sencha Cmd to build your
application -->
        <script id="microloader" data-app="53f4957a-885b-457f-98a2-0314927f7a39"
type="text/javascript" src="bootstrap.js"></script>

    </head>
    <body></body>
    </html>
```

4.4.3 bootstrap.css

在 bootstrap.css 文件内只有一句代码：

```
@import 'build/development/SimpleCMS/resources/SimpleCMS-all.css';
```

从代码可以看到， 它引用了 build 目录下的样式文件，也就是在创建应用程序时，执行的那次生成应用程序所创建的样式文件，这也解释了为什么要在创建应用程序时需要执行一次生成应用程序操作。

这样做的目的，还是为了适应通用应用程序的需要。因为经典模式的主题和现代模式的主题包是两个不同的包，再加上可能还需要分别为两种模式自定义一些样式，如何去加载这些样式是一个问题。Sencha 认为最好的解决办法就是生成一次应用程序，把所需的样式都打包到一个样式文件中，这样就不需要考虑加载多少文件的问题了，只需要加载一个样式文件就行了。

新的样式引用方式，对于单独使用经典模式进行开发也有了新的要求，就是在引用一个组件的时候，可能该组件的样式还没打包进样式文件，这时就需要生成一次应用程序，将该组件的样式打包进样式文件。

注 意

当发现视图中的某个组件，如日期选择字段，在显示时有问题的时候，就需要考虑是否已经在视图的 requires 中引用了该组件，并使用 sencha app build 命令生成一次应用程序，以便将组件的样式打包到应用程序的样式文件，从而使组件能正常显示。

4.4.4 bootstrap.js

文件 bootstrap.js 的主要功能是用来加载应用程序的脚本和样式，如果在开发通用应用程序时经常找不到类文件，除了重新生成应用程序外，还可以通过修改该文件来实现。由于文件内容很多，在此就不多解释了，自己有兴趣的话可以研究一下。

4.4.5 application.js

文件 application.js 的代码如代码清单 4-4 所示，该文件与 Ext JS 4 的相比，没了 controls 和 models 配置项，主要原因是 Ext JS 6 的视图控制器是跟随视图创建的，不需要在这里预加载，而模型在存储中引用就行了，也不需要预加载。配置项 stores 的作用是预加载一些共享的存储。

在该文件中，还多了一个 onAppUpdate 方法，该方法的作用是在应用程序的脚本文件发生改变的时候，通知用户更新应用程序。在这里需要做的是修改一下里面的提示信息。

文件中的 launch 方法会在页面加载完成后执行，类似于 jQuery 的 ready 方法和 Ext JS 的 onReady 方法。在该方法内，可以做一些预处理。

代码清单 4-4 application.js

```
Ext.define('SimpleCMS.Application', {
    extend: 'Ext.app.Application',
```

```
        name: 'SimpleCMS',

        stores: [
            // TODO: add global / shared stores here
        ],

        launch: function () {
            // TODO - Launch the application
        },

        onAppUpdate: function () {
            Ext.Msg.confirm('Application Update', 'This application has an update,
reload?',
                function (choice) {
                    if (choice === 'yes') {
                        window.location.reload();
                    }
                }
            );
        }
    });
```

4.4.6 app.js

文件 app.js 的内容如代码清单 4-5 所示。代码在 requires 中引用了主视图，并在配置项 mainView 中将主视图定义为 SimpleCMS.view.main.Main，这样在应用程序启动时就会创建一个 SimpleCMS.view.main.Main 的实例，并渲染到页面中。

代码清单 4-5　app.js

```
    Ext.application({
        name: 'SimpleCMS',

        extend: 'SimpleCMS.Application',

        requires: [
            'SimpleCMS.view.main.Main'
        ],

        mainView: 'SimpleCMS.view.main.Main'

    });
```

4.4.7 SimpleCMS.view.main.Main

类 SimpleCMS.view.main.Main 的内容如代码清单 4-6 所示。从代码可以看到，主视图是从 Ext.tab.Panel 扩展的，也就是说主视图是一个标签页，在 items 中定义了 Home、Users、Groups 和 Settings 四个标签。

在配置项 requires 中，引用了插件 Ext.plugin.Viewport，该插件的作用就是将当前视图作为视区（Viewport），整个页面的顶层组件就是当前视图。

在配置项 controller 中，定义了视图的视图控制器，该视图控制器的别名为 main，在代码清单

4-8 中可以看到它的定义“controller.main”。

在配置项 viewModel 中，定义了视图的视图模型，该视图模型的别名为 main，在代码清单 4-9 中可以看到它的定义“alias: 'viewmodel.main'”。

配置项 controller 和 viewModel 在以后的开发中会经常使用到，因而这个要熟悉它是怎么使用的。

从图 4-6 中所看到的标签页与 Ext JS 示例中看到的标签页的样式不同，这是因为在这里使用了自定义的组件样式 navigation（配置项 ui）。这个组件的样式可以在 sass\src\view\main 文件夹下找到，在 4.5 节会讲述自定义样式这个问题。

在视图中，还使用了响应式布局，这个是在 responsiveConfig 配置项中定义的。在定义中，tall 的意思是当宽度小于高度的时候，标签（headerPosition）将显示在顶部（top），wide 的意思是当宽度大于高度的时候，标签将显示在左边（left）。这个大家可以通过浏览器的调试功能来查看，具体步骤是按 F12 键打开 Web 开发者工具，然后在右上角找到响应式设计模式，单击后就可切换不同的分辨率了。

在视图中，还使用到了最简单的数据绑定功能，包括视图的标题（title）和子组件的内容（html）。

代码清单 4-6　SimpleCMS.view.main.Main

```
Ext.define('SimpleCMS.view.main.Main', {
    extend: 'Ext.tab.Panel',
    xtype: 'app-main',

    requires: [
        'Ext.plugin.Viewport',
        'Ext.window.MessageBox',

        'SimpleCMS.view.main.MainController',
        'SimpleCMS.view.main.MainModel',
        'SimpleCMS.view.main.List'
    ],

    controller: 'main',
    viewModel: 'main',

    ui: 'navigation',

    tabBarHeaderPosition: 1,
    titleRotation: 0,
    tabRotation: 0,

    header: {
        layout: {
            align: 'stretchmax'
        },
        title: {
            bind: {
                text: '{name}'
            },
            flex: 0
        },
        iconCls: 'fa-th-list'
    },
```

```
        tabBar: {
            flex: 1,
            layout: {
                align: 'stretch',
                overflowHandler: 'none'
            }
        },

        responsiveConfig: {
            tall: {
                headerPosition: 'top'
            },
            wide: {
                headerPosition: 'left'
            }
        },

        defaults: {
            bodyPadding: 20,
            tabConfig: {
                plugins: 'responsive',
                responsiveConfig: {
                    wide: {
                        iconAlign: 'left',
                        textAlign: 'left'
                    },
                    tall: {
                        iconAlign: 'top',
                        textAlign: 'center',
                        width: 120
                    }
                }
            }
        },

        items: [{
            title: 'Home',
            iconCls: 'fa-home',
            // The following grid shares a store with the classic version's grid as
well!
            items: [{
                xtype: 'mainlist'
            }]
        }, {
            title: 'Users',
            iconCls: 'fa-user',
            bind: {
                html: '{loremIpsum}'
            }
        }, {
            title: 'Groups',
            iconCls: 'fa-users',
            bind: {
                html: '{loremIpsum}'
            }
        }, {
            title: 'Settings',
```

```
            iconCls: 'fa-cog',
            bind: {
                html: '{loremIpsum}'
            }
        }]
    });
```

4.4.8 SimpleCMS.view.main.List

SimpleCMS.view.main.List 类是在 Home 标签页中所显示的内容，它的代码如代码清单 4-7 所示。从代码中可以看到，这是一个网格（扩展自 Ext.grid.Panel），显示了 Name 、Email 和 Phone 等三列数据。

在这里，要特别注意的是 listeners 配置项。在配置里，为网格的 select 事件绑定了 onItemSelected 方法，但是在类定义中并没有看到该方法，说明该方法的定义不是在自身的视图控制器中，就是在父类的视图控制器中。

代码清单 4-7 SimpleCMS.view.main.List

```
Ext.define('SimpleCMS.view.main.List', {
    extend: 'Ext.grid.Panel',
    xtype: 'mainlist',

    requires: [
        'SimpleCMS.store.Personnel'
    ],

    title: 'Personnel',

    store: {
        type: 'personnel'
    },

    columns: [
        { text: 'Name',  dataIndex: 'name' },
        { text: 'Email', dataIndex: 'email', flex: 1 },
        { text: 'Phone', dataIndex: 'phone', flex: 1 }
    ],

    listeners: {
        select: 'onItemSelected'
    }
});
```

4.4.9 SimpleCMS.view.main.MainController

视图控制器 SimpleCMS.view.main.MainController 的代码如代码清单 4-8 所示。在代码中，只定义了网格的 select 事件对应的 onItemSelected 方法，以及确认对话框的回调函数 onConfirm。

代码清单 4-8 SimpleCMS.view.main.MainController

```
Ext.define('SimpleCMS.view.main.MainController', {
    extend: 'Ext.app.ViewController',

    alias: 'controller.main',
```

```
        onItemSelected: function (sender, record) {
            Ext.Msg.confirm('Confirm', 'Are you sure?', 'onConfirm', this);
        },

        onConfirm: function (choice) {
            if (choice === 'yes') {
                //
            }
        }
    });
```

4.4.10 SimpleCMS.view.main.MainModel

视图模型 SimpleCMS.view.main.MainModel 的代码如代码清单 4-9 所示。在代码中，定义了两个数据对象 name 和 loremIpsum，分别用来作为视图的标题和子组件的显示内容。

代码清单 4-9 SimpleCMS.view.main. MainModel

```
    Ext.define('SimpleCMS.view.main.MainModel', {
        extend: 'Ext.app.ViewModel',

        alias: 'viewmodel.main',

        data: {
            name: 'SimpleCMS',

            loremIpsum: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.'
        }

        //TODO - add data, formulas and/or methods to support your view
    });
```

4.4.11 app.json

文件 app.json 的主要功能是为生成工作做一些配置，常用的配置包括首页文件的位置、所需包的设置、主题设置和附加脚本设置等。

1. 首页设置

要修改首页位置，可以在文件中寻找 indexHtmlPath 属性，它的值就是首页的路径和文件名。

2. 所需包的设置

在 Ext JS 6 中，类似图形与图表、扩展与字体图标等部件是以包的形式添加到应用程序中的。如果没有添加这些包，就使用不了包里所包含的组件。在 QQ 群中，时常会有人问在应用程序中的图表不显示、没有找到类文件等问题，大多数情况都是由于没有添加所需包造成的。

要修改所需包的设置，可在文件中寻找 requires 属性，默认的代码如下：

```
    "requires": [
          "font-awesome"
       ],
```

从以上代码可以看到，默认情况下，已经把字体图标包包含在应用程序里了。如果使用的是海卫一主题，千万别删除这个包，因为海卫一主题的许多组件图标使用的都是字体图标，如果把这个包去了，图标就不显示了。

如果想添加图形和图表包，在 requires 中添加“charts”就行了。要添加扩展组件包，可添加“ux”。

在 Ext JS 6 中，本地化文件也是以包的形式提供的，包的引用只为“locale”。如果没有配置具体的本地化包，生成时会选择默认的包，其实是不使用任何本地化包。为了添加所需的语言包，还需要使用 locale 属性来指定单一的本地化包，或使用 locales 属性来指定多个本地化包，譬如，locale 的值是“zh_CN”，则使用简体中文作为本地化包。如果 locales 值是“["zh_CN","zh_TW"]”，则使用简体中文和繁体中文作为本地化包。当设置的是 locales 时，如果在生成时没有指定本地化包，则会把数组定义中的第一个值作为打包时的本地化包，如“["zh_CN","zh_TW"]”，默认会把简体中文作为本地化包。在生成时，可通过选项“—locale”来指定打包时的本地化包，如要指定繁体中文作为本地化包，则生成的命令如下：

```
sencha app build -l zh_TW
```

一个包含了字体图标、简体中文的本地化包、扩展组件包和图形与图表包的配置代码如下：

```
"requires": [
    "font-awesome",
    "locale",
    "ux",
    "charts"
  ],
  "locale": "zh_CN",
```

3. 主题设置

由于 Ext JS 6 使用的是生成后的样式文件，因此不能像 Ext JS 4 那样直接在页面中指定主题的样式文件，只能通过配置文件来指定主题包。

要修改主题，可在文件中查找 theme 属性。它的默认值是 theme-triton，如果要修改为 Ext JS 4 常用的海王星主题，可以将值修改为 theme-neptune。

如果想知道经典模式有哪些主题可用，可以进入 ext\classic 文件夹，以 theme 开头的文件夹就是可用的主题。

4. 附加脚本设置

在 Ext JS 4，如果要添加第三方库，一般的处理就是通过 SCRIPT 标记来添加。而在 Ext JS 6，多了一个选项，可以在 app.json 中添加。在 app.json 中添加的好处是不需要自己编写脚本来加载第三方库，通过 bootstrap.json 文件可自动加载第三方库。如果想更进一步，可以把第三方库也打包到生成的脚本文件中。

要添加额外的脚本，可在文件中寻找 js 属性，默认的代码如下：

```
"js": [
        {
            "path": "${framework.dir}/build/ext-all-rtl-debug.js"
        },
        {
            "path": "app.js",
            "bundle": true
        }
    ],
```

在代码中，已经包含了 ext-all-rtl-debug.js 和 app.js 两个脚本文件，其中的 ext-all-rtl-debug.js 是调试时使用到的框架包，而 app.js 则是应用程序的入口。

大家可能注意到了，在 app.js 下，有一个 bundle 关键字，它的值为 true 表示该文件为连接类的容器。要理解连接类的容器，需要清楚的一点是，JavaScript 是一个按文档流顺序执行的语言，也就是谁先加载的谁先执行。对于 Ext JS 这样存在多重继承的框架，根据规则，父类肯定要先加载，然后才能去加载子类、子类的子类，不然就会出现找不到父类的错误。还有一种情况就是在视图中使用了某个组件，如果视图先加载并创建了实例，而子组件的类文件还没有加载，也会出现找不到类的错误。因而，在 Ext JS 中，类之间哪个先加载，哪个后加载，是有顺序的。为了解决这个类加载的先后顺序问题，就促成了类似于 C#的 using 语句，类似于 Java 的 import 语句这样的 requires 配置项。在 requires 中引用的类都是要先加载的，然后才能加载视图。虽然有了 requires，但是这个 requires 该从什么地方开始算呢？这时候就需要一个入口容器来解决这个问题。在整个应用程序中，app.js 是整个应用程序的入口，因而把它作为连接器的容器是最适合不过了。bundle 关键字就是用来解决这个问题的。

要把第三方库加载到应用程序中，只要在 js 的数组内添加一个配置对象就行了。在配置对象内，需要使用 path 属性来指定第三方库的路径，这个可参考 ext-all-rtl-debug.js 的定义。一般情况，如果没有特殊要求，建议将第三方库放在 resource 文件夹内，而不是放在其他文件夹内。这样的好处是，便于在 app.json 中指定路径。在发布时，也不需要额外复制第三方库的文件，因为在复制 resource 文件夹时已经把第三方库也一起复制了。

如果要将第三方库打包到最终的应用程序脚本文件中，可以添加 includeInBundle 属性，并设置为 true。不过，在 6.2 这个版本中，第三方库不能打包在 Ext 框架的顶部，只能打包在应用程序的后面，这就存在问题了，使用到这些第三方库的组件时会出错，因为找不到对象。如果使用的是 6.2 版本，建议还是不要打包了。

注 意

不知道什么原因，Ext JS 6.2 默认使用 ext-all-rtl-debug.js 文件作为框架文件，而且该文件默认是右对齐的，从而让不少开发人员有点摸不着头脑。要解决这个问题其实很简单，将 ext-all-rtl-debug.js 修改为 ext-all-debug.js 就行了。

4.5 自定义样式

在创建的初始应用程序中，自定义了标签页的样式，这个是如何实现的呢？

首先需要了解清楚的是目录结构。在应用程序的文件夹中，有一个名为 sass 的文件夹，该文件夹就是用来存放自定义的样式文件的。在该文件夹下有两个自定义样式文件经常会使用到的文件夹 var 和 src。其中，var 文件夹用来存放自定义的 CSS 变量，src 文件夹用来存放自定义的样式文件。要查找某个组件有什么 CSS 变量，可参阅该组件的 API 文档中的 theme variables 节或 theme mixins 节。

无论是 src 文件夹还是 var 文件，都需要遵循一个规则，就是与 app\view 文件夹中视图的目录结构相同且文件名（不包含扩展名）相同。例如初始应用程序中，要为主视图自定义样式，就需要在 sass\src\view\main 文件夹中创建 Main.scss 文件，并在该文件中定义样式；如果要为样式定义 CSS 变量，则需要在 sass\var\view\main 文件夹中创建 Main.scss 文件，并在该文件中定义 CSS 变量。

接下来需要了解的是如何为组件定义样式。打开 sass\src\view\main\Main.scss 文件，会看到如代码清单 4-10 所示的代码。

代码清单 4-10　sass\src\view\main\Main.scss

```
/**
 * Generates a set of style rules for the "navigation" tab UI.
 */
@include extjs-tab-panel-ui(
    $ui: 'navigation',
    $ui-tab-background-color: transparent,
    $ui-tab-background-color-over: #505050,
    $ui-tab-background-color-active: #303030,
    $ui-tab-background-gradient: 'none',
    $ui-tab-background-gradient-over: 'none',
    $ui-tab-background-gradient-active: 'none',
    $ui-tab-color: #acacac,
    $ui-tab-color-over: #c4c4c4,
    $ui-tab-color-active: #fff,
    $ui-tab-glyph-color: #acacac,
    $ui-tab-glyph-color-over: #c4c4c4,
    $ui-tab-glyph-color-active: #fff,
    $ui-tab-glyph-opacity: 1,
    $ui-tab-border-radius: 0,
    $ui-tab-border-width: 0,
    $ui-tab-inner-border-width: 0,
    $ui-tab-padding: 24px,
    $ui-tab-margin: 0,
    $ui-tab-font-size: 15px,
    $ui-tab-font-size-over: 15px,
    $ui-tab-font-size-active: 15px,
    $ui-tab-line-height: 19px,
    $ui-tab-font-weight: bold,
    $ui-tab-font-weight-over: bold,
    $ui-tab-font-weight-active: bold,
    $ui-tab-icon-width: 24px,
    $ui-tab-icon-height: 24px,
    $ui-tab-icon-spacing: 5px,
    $ui-bar-background-color: #404040,
```

```
    $ui-bar-background-gradient: 'none',
    $ui-bar-padding: 0,
    $ui-strip-height: 0
);

/**
 * Generates a set of style rules for the "navigation" panel UI.
 */
@include extjs-panel-ui(
    $ui: 'navigation',
    $ui-header-color: #fff,
    $ui-header-glyph-color: #fff,
    $ui-header-glyph-opacity: 1,
    $ui-header-font-size: 20px,
    $ui-header-line-height: 24px,
    $ui-header-font-weight: bold,
    $ui-header-icon-height: 24px,
    $ui-header-icon-width: 24px,
    $ui-header-icon-spacing: 15px,
    $ui-header-background-color: $base-color,
    $ui-header-padding: 0,
    $ui-header-text-margin: 36px,
    $ui-header-noborder-adjust: false
);

.x-title-icon-navigation {
    font-family: FontAwesome;
    color: #fff;
    font-size: 24px;
    line-height: 24px;
}

.x-tab-icon-el-navigation {
    font-family: FontAwesome;
    color: #acacac;

    .x-tab-over & {
        color: #c4c4c4;
    }

    .x-tab-active & {
        color: #fff;
    }
}

.x-panel-header-title-navigation > .x-title-text-navigation:after {
    top: 30px;
    right: -24px;
}
```

在代码中，在第一个代码段内包含了 extjs-tab-panel-ui 这个变量，在标签面板（Ext.tab.Panel）的 API 文档中，在 theme mixins 中可找到它的说明。根据说明，可以知道$ui 是用来指定 ui 的名字的，当前的 ui 名字为 navigation，也就是说，在组件中，将配置项 ui 的值指定为 navigation 就可以使用这个自定义的 ui 了。至于其他定义，基本上是标签页的具体样式定义了。

在第二个代码段定义的是 extjs-panel-ui，也就是说，这里定义的是面板（Ext.panel.Panel）的样

式，在面板的 theme mixins 中可找到它的说明。

第三和第四个代码段的代码我们应该很熟悉了，是常用的 CSS 样式定义语句。

在第三个代码段中定义了标签的图标，在这里使用了字体图标，字体的颜色为 acacac。将鼠标移动到标签上时，字体颜色将转换为 c4c4c4。当标签为当前活动标签时，字体颜色将转换为 fff。

在第四个代码段中，调整了标题的位置。

以上是自定义样式的一些基本原则，也可以不必完全遵守。例如，可以在父视图中为子视图定义样式，但必须满足一个条件，就是编译器能够根据视图找到样式并打包进样式文件，如创建了一个没有视图与之对应的样式文件，那么编译器就不会把该样式文件的样式编译进样式文件中。

有些自定义样式可能会在多个视图中使用，但并不需要每个视图都定义一次，因为这些样式只要定义一次，并编译进样式文件，这个样式就可重复使用。视图与样式文件对应的目的是为了便于维护，便于找到视图所对应的样式。不过，按照这个道理，如果不嫌粘贴复制麻烦的话，每个视图都定义一次似乎更有利于查找样式。

4.6 生成应用程序

要生成应用程序很简单，在命令提示符窗口中，先进入到应用程序的根文件夹，然后输入以下命令即可：

```
sencha app build
```

在生成过程中，如果没有出现任何“[ERR]”开头的信息，说明生成成功了。生成成功后，可在 build\production\SimpleCMS 文件夹中找到生成后的文件。

如果在使用生成后的文件调试时出现“c is not a constructor return new c(a[0])”这样的错误，说明在某个视图中少引用了某个类或某个类的类名写错了，这时候，最简单直接的方法是使用以下命令生成一个测试版本来调试错误：

```
sencha app build --testing
```

以上命令会生成类似于 ext-all-debug.js 这样的，没有将脚本压缩的应用程序，非常便于调试应用程序。这时再调试应用程序，就可知道错误发生在哪里了。

4.7 关于乱码

使用 Sencha Cmd 创建的文件，不知道为什么不是以 UTF-8 编码格式保存的，当文件带有中文的时候，就会出现乱码。要解决这个问题不难，把文件另存为 UTF-8 编码格式的文件就行了。在 Visual Studio 中的操作是，在主菜单中选择“文件”→“另存为”，打开“另存文件为”对话框。在对话框中单击“保存”按钮的下三角按钮，在下拉列表中选择编码保存。这时会提示是否替换文件，选择“是”之后，会弹出高级保存选项对话框。在高级保存选项对话框中，在编码下拉选项中选择“Unicode（UTF-8 无签名）-代码页 65001”，最后单击“确定”按钮结束操作。

如果还是不行，就尝试在脚本文件的顶部添加以下代码来声明该文件是 UTF-8 编码格式的文件：

```
//@charset UTF-8
```

如果以上方法都不能解决编码问题，其实在与 QQ 群友的互动中也碰到这种情况，那就很可能是系统问题了，因为群友在其他机器上尝试是没有问题的。

4.8 小 结

本章主要讲述了如何使用 Sencha Cmd 来创建和生成应用程序。虽然整个过程对于新手来说有点困难，毕竟要安装的东西比较多，而且是文本模式操作；但是，这对于开发来说却是相当有帮助的，尤其是生成后的应用程序，既可以移除不需要的组件，以减少发布包的大小，还可以减少后续的动态加载，让应用程序的效率更高。

虽然 Sencha Cmd 挺方便的，但是当发生错误的时候，也是很让人头疼的，尤其是跟踪和查找错误比较困难。因而，在使用之前，需要有心理准备，尤其是在 Sencha Cmd 版本与 Ext JS 版本对应不上的时候，出现的错误可以说是根本解决不了，而且也是解决问题的盲点。造成这个问题的主要原因是版本不匹配的时候并不会有任何提示信息，只能自己去互联网寻找答案。为了解决这个问题，很多时候是不厌其烦地去测试，而并不会考虑到是版本不对的问题。

在下一章，将进入真正的开发阶段，开始使用 Visual Studio 2017 来创建简单的 CMS 系统。