第 12 章

选择和查询应用

12.1 组件简介

12.1.1 日期选择组件

许多应用程序都需要用户输入日期或时间，日期选择组件也即 DatePicker 组件，能在界面创建一个显示日期和时间的卷轴，用户卷动卷轴即可选取日期和时间。

DatePicker 组件常用属性如表 12-1 所示。

表 12-1

属性名称	说明
Mode	设置日期和时间显示模式，默认值为 Date and Time，表示日期和时间都会显示
Locale	设置地区，默认值为 Default，以英文显示
Interval	设置显示项目的时间间隔，范围是 1 ～ 30min，默认值为 1min
Date	设置选取日期，默认值为当前日期
Constraints /Minimum Date	设置最小选取日期，须先选中，再在下方输入日期和时间；若未选则无法输入
Constraints /Maximum Date	设置最大选取日期，须先选中，再在下方输入日期和时间；若未选则无法输入
Timer	设置倒计时的时间，单位为秒

在属性面板设置属性固然方便，但却有许多缺点。首先，每次通过拖动创建组件时，都要一一设置各种属性，有时可能会发生遗漏。其次，阅读别人的程序时，不可能检查每个组件的每一个属性，因而降低了程序的可读性。最后，有些属性并未完全列在属性面板中，这些属性必须通过程序来设置。

下面重点说明 Mode 属性和 Locale 属性。

1. Mode 属性

设置 DatePicker 组件 Mode 属性的语法为：

```
DatePicker 组件 .datePickerMode=UIDatePickerMode.模式
```

“模式”有四种值：DateAndTime、Date、Time 以及 CountDownTimer。

例如，设置名称为“日期选择器”项目的 DatePickerMode 组件的 Mode 属性为：

```
DateAndTime:
datePicker.datePickerMode=UIDatePickerMode.DateAndTime
```

2. Locale 属性

DatePicker 组件显示的文字会随着 Locale 属性设置值的不同而异，程序的设置语法为：

```
DatePicker 组件 .locale=NSLocale(localeIdentifier:" 地区代码 ")
```

不同语言其“地区代码”不同，较常用的是：en 为英文、zh_TW 为繁体中文、zh_CN 为简体中文。

例如，设置名称为“日期和时间选择器”项目的 DatePickerMode 组件使用简体中文显示，代码为： datePicker.locale=NSLocale(localeIdentifier:"zh_CN")。

12.1.2 表视图组件

Table View 组件称为表视图组件，它以一行一行的方式显示数据，每一行数据称为一个单元格（cell），每一个单元格可以包含一个或多个不同类型的组件。

Table View 继承自 View 类，简单的做法是从组件区拖动 Table View 组件到 View 中，就可以加入一个 Table View 组件。有时也可以由程序代码自动生成。

索引表格项目需要用到的代理方法如表 12-2 所示。

表 12-2

代理方法	方法说明
numberOfSectionsInTableView(_:)	设置表格视图中章节的数量，默认值为 1。如果需要添加多个章节，只返回一个更大的整数即可
tableView(_:numberOfRowsInSection:)	设置指定章节中单元格行的数量
tableView(_:titleForHeaderInSection:)	设置章节的标题文字，返回结果为字符串。若返回结果为 nil，则章节没有标题
sectionIndexTitlesForTableView(_:)	设置在表格右侧显示的索引序列的内容，返回结果为一个字符串数组
tableView(_:cellForRowAtIndexPath:)	初始化和复用单元格

12.2 日期和时间选择器的设计

设计步骤如下。

步骤 1：创建工程项目——日期和时间选择器

创建过程类似图 11-21，项目名为“日期和时间选择器”。

步骤 2：设计用户界面

用户界面在模拟器上显示界面如图 12-1 所示。

界面元素由一个 Date Picker 组件和 4 个标签组件组成，如图 12-2 所示。

步骤 3：编写程序

在图 12-3 所示的项目文件浏览区双击 ViewController.swift 文件，打开代码框架，编写如下代码：

图 12-1

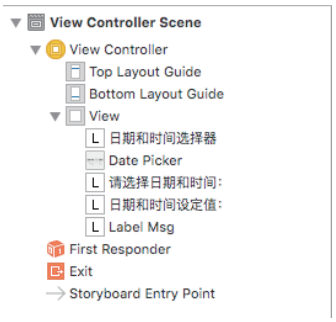

图 12-2

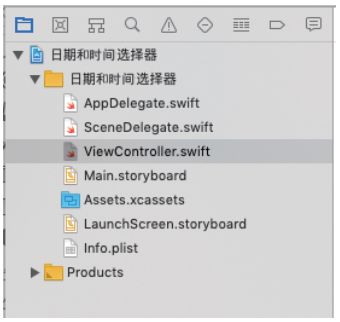

图 12-3

```
import UIKit

class ViewController: UIViewController {

    var dateFormatter=DateFormatter()                    //声明日期时间格式变量

    @IBOutlet var datePicker:UIDatePicker!               //日期时间组件连接
    @IBOutlet var labelMsg:UILabel!                      //显示信息的标签组件连接

    //用户选取日期就更新显示
    @IBAction func dataChange(sender:UIDatePicker)
    {
        labelMsg.text=dateFormatter.string(from: datePicker.date)
    }

    override func viewDidLoad()
    {
        super.viewDidLoad()
        //Do any additional setup after loading the view, typically
          from a nib
          datePicker.datePickerMode=UIDatePicker.Mode.dateAndTime
        //显示模式
        datePicker.locale=NSLocale(localeIdentifier:"zh_CN") as Locale
        //用简体中文显示
        datePicker.date=NSDate() as Date
        dateFormatter.dateFormat=" 公元 y 年 M 月 d 日 hh 点 mm 分 ss 秒 "
        //显示格式
        labelMsg.text=dateFormatter.string(from: datePicker.date)
```

```
        }

    }
```

代码分析：

用 var dateFormatter=DateFormatter() 声明日期时间格式变量，此变量在 dataChange() 和 viewDidLoad() 中都会使用，故在此声明为全局变量；

用 dataChange() 方法控制当用户选取日期和时间时立即在标签组件中按日期时间组件设置的格式更新显示；

用 datePicker.datePickerMode=UIDatePicker Mode.DateAndTime 设置组件的显示模式为日期和时间；

用 datePicker.locale=NSLocale(localeIdentifier:"zh_CN") 设置组件的显示地区语言为简体中文；

用 datePicker.date=NSDate() 设置组件显示的为当前的日期和时间；

用 dateFormatter.dateFormat="公元 y 年 M 月 d 日 hh 点 mm 分 ss 秒" 设置组件的显示格式；

用 labelMsg.text=dateFormatter.string (datePicker.date) 在标签组件中显示程序开始执行时的日期和时间。

步骤 4：完成插座和动作的关联，实现界面组件元素与程序代码的连接

用右键把 View Controller 按钮拖曳至 datePicker 组件上释放，在弹出的如图 12-4 所示的快捷菜单中选中日期时间组件的对象名 datePicker。接着用右键把 datePicker 组件起拖曳至 View Controller 按钮，在弹出的快捷菜单中分别选中相应按钮组件对应的方法名 dataChange。

再用右键把 View Controller 按钮拖曳至下方的标签组件上释放，在弹出的如图 12-5 所示的快捷菜单中选中标签组件的对象名 labelMsg。

完成插座和动作的关联之后，要及时按 Command+S 组合键保存。

最后右击 View Controller 按钮，在弹出的如图 12-6 所示的快捷菜单中可以查看全部连接信息。

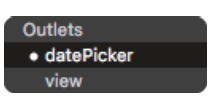

图 12-4

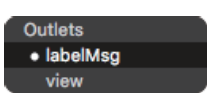

图 12-5

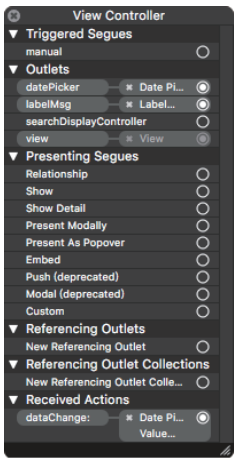

图 12-6

步骤 5：运行应用程序

按 Command+R 组合键后，用户界面在仿真器上显示如图 12-7 所示，显示的是系统当前的日期和时间。

在选取新的日期和时间之后，显示的是更新的日期和时间，如图 12-8 所示。

图 12-7

图 12-8

至此，“日期和时间选择器”项目全部设计完毕。

12.3 数据查询器的设计

数据查询器的设计总窗口如图 12-9 所示。

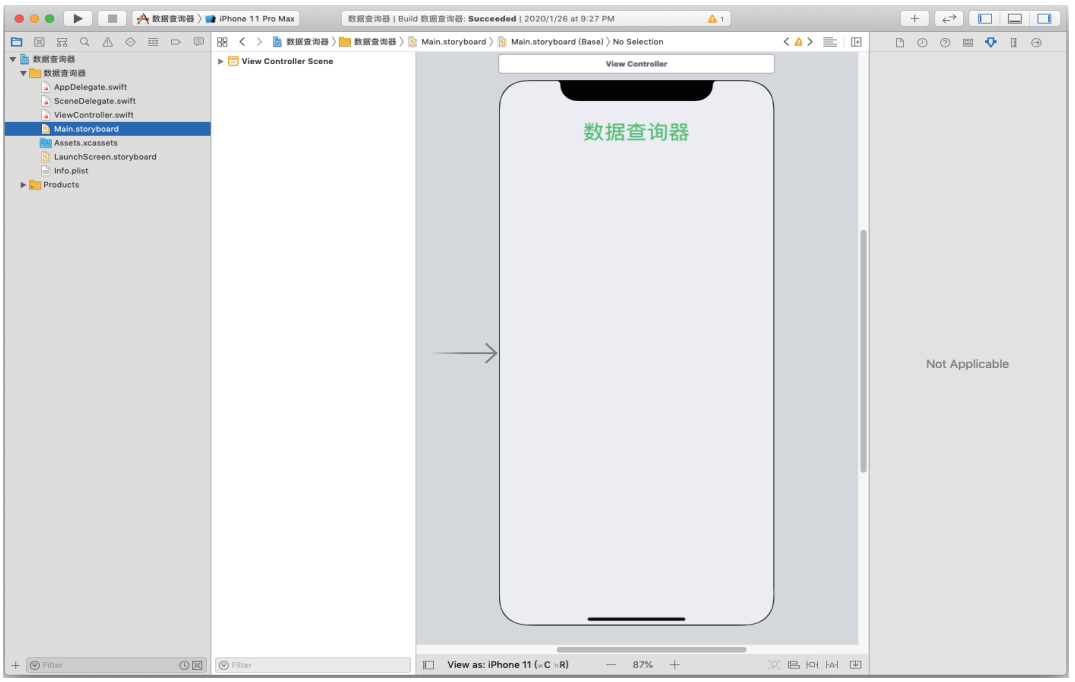

图 12-9

设计步骤如下。

步骤 1：创建工程项目——数据查询器

创建过程类似图 11-21，项目名为“数据查询器”。

步骤 2：设计用户界面

该项目全部是由程序代码运行生成的，因而是一个空的界面，其中只有一个标志项目名称的静态标签，如图 12-10 所示。

步骤 3：编写程序

在图 12-11 所示的项目文件浏览区双击 ViewController.swift 文件，打开代码框架，编写如下代码：

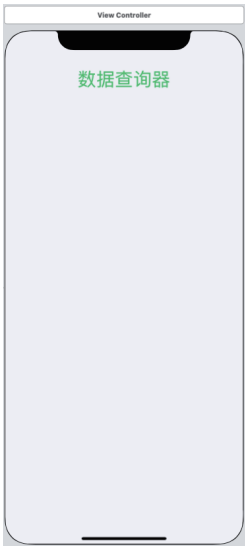

图 12-10

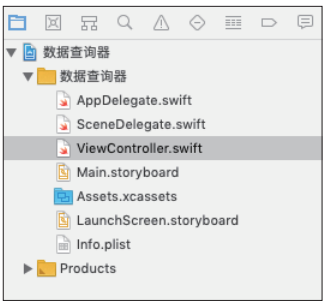

图 12-11

```
import UIKit

class ViewController: UIViewController,UITableViewDataSource
{

    var countries:Dictionary<String,[String]>=
    ["Hh":[" 黄海 "," 男 "," 重庆 ","13667890123","QQ:678901234","hh@126.com"],
    "Lb":[" 李斌 "," 男 "," 上海 ","13612345678","QQ:123456789","lb@126.com"],
    "Lm":[" 李明 "," 男 "," 西安 ","13678901234","QQ:789012345","lm@126.com"],
    "Qx":[" 钱新 "," 男 "," 天津 ","13645678901","QQ:456789012","qx@126.com"],
    "Sl":[" 宋玲 "," 女 "," 北京 ","13634567890","QQ:345678901","sl@126.com"],
    "Sm":[" 孙梅 "," 女 "," 广州 ","13689012345","QQ:890123456","sm@126.com"],
    "Wy":[" 王英 "," 女 "," 武汉 ","13656789012","QQ:567890123","wy@126.com"],
    "Xl":[" 谢璐 "," 女 "," 南昌 ","13690123456","QQ:901234567","xl@126.com"],
    "Zq":[" 周青 "," 男 "," 南京 ","13623456789","QQ:234567890","zq@126.com"],
    "Zj":[" 赵佳 "," 女 "," 北京 ","13601234567","QQ:012345678","zj@126.com"]]
```

```
        var keys:[String]=[]

        override func viewDidLoad()
        {
            super.viewDidLoad()
            //Do any additional setup after loading the view

               keys=Array(countries.keys).sorted()

            let screenRect=UIScreen.main.bounds
            let tableRect=CGRect(x:0,y:20,width:screenRect.size.width,height:
screenRect.size.height-20)

            let tableView=UITableView(frame:tableRect)

            tableView.dataSource=self
            self.view.addSubview(tableView)
        }
        //返回键名数组的长度作为表格中章节的数目
        func numberOfSections(in tableView: UITableView) -> Int
        {
            return keys.count
        }

        //根据键名对应的键值，返回数组的长度作为指定章节的单元格数列数量
        func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int
        {
            let subCountries=countries[keys[section]]
            return(subCountries?.count)!
        }

        //返回键名数组的键名作为表格中章节的标题文字
        func tableView(_ tableView: UITableView, titleForHeaderInSection
section:Int) -> String? {
            return keys[section]
        }

        //返回键名数组作为索引序列的内容
        func sectionIndexTitles(for tableView: UITableView) -> [String]?
        {
            return keys
        }
```

```
        func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell
        {
            let identifier="reusedCell"
            var cell=tableView.dequeueReusableCell(withIdentifier: identifier)
            if(cell==nil)
            {
                 cell=UITableViewCell(style:UITableViewCell.CellStyle.default,
reuseIdentifier:identifier)
            }

            //根据 IndexPath 参数的章节值获得当前单元格所在的章节序号并组成数组
            let subCountries=countries[keys[(indexPath as NSIndexPath).section]]
            //根据 IndexPath 参数的 row 值获得当前单元格所在章节行号并完成初始化和设置
            cell?.textLabel?.text=subCountries![(indexPath as NSIndexPath).row]

            return cell!
        }
    }
```

代码分析：

该项目是设计一个带索引的表格型的通讯录，每条记录共有姓名、性别、通信地址、电话号码、QQ 号和电子邮箱六项，属于多种数据类型，所以使用了字典给表格对象提供数据源。字典对象的键作为 UITableView 的 Section（章节），字典对象的值（数组）作为 Section 中单元格的内容。

接着定义了一个数组对象 keys 用来存储按升序排列的键名序列，这个数组的长度将作为表格中章节的数目。用 keys=Array(countries.keys).sorted() 获得 countries 字典对象所有的键名，并转换为一个按升序排列的数组对象。数组对象的 sorted() 方法就是对数组进行升序排列用的。

用程序代码设计数据表格的范围为整个屏幕，显示内容的索引序列设置在数据表格的右侧。

步骤 4：完成插座和动作的关联，实现界面组件元素与程序代码的连接

此项目不需要，从略。

步骤 5：运行应用程序

按 Command+R 组合键后，用户界面在模拟器上显示如图 12-12 所示。

在右边中间索引序列条内单击 Qx 之后，在模拟器上显示如图 12-13 所示。

在右边中间索引序列条内单击 Wy 之后，在模拟器上显示如图 12-14 所示。

在右边中间索引序列条内单击 Zj 之后，在模拟器上显示如图 12-15 所示。

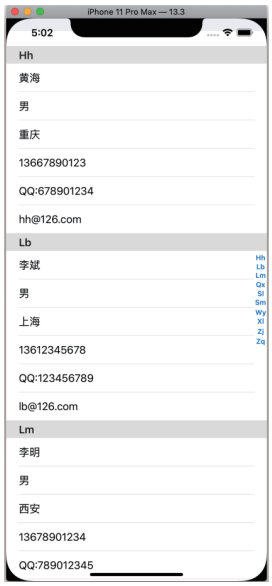

图 12-12

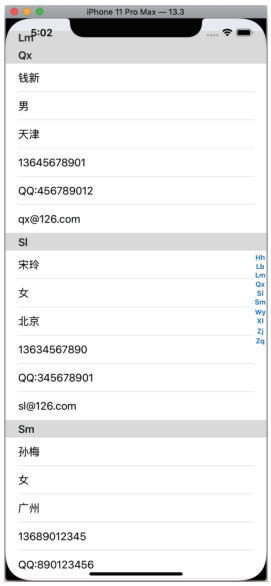

图 12-13

图 12-14

图 12-15

至此，“数据查询器”项目全部设计完毕。

第 13 章

图片应用

13.1 数字化图像

数码照片几乎都是采用压缩格式存放的，需要了解其存放格式和大小的转换和处理，这就涉及数字化图像的有关概念。

数字化图像可以分为图形和图像两种。图像是呈现给人们的一幅幅界面，一般是由图像输入设备捕获，以数字化的形式存储在计算机中的，例如照片、绘画等；图形则是由绘图工具绘制，由点、线、面、体和文字等图元构成的，例如图案、工程图样等。

13.1.1 图像的大小

数码照片是数字化图像，在计算机屏幕上是由若干个点（也称像素）构成的。通常，计算机中有一个显示存储器（也称帧存储器），它存放的是与屏幕上的像素一一对应的一个数据矩阵，即屏幕上有多少个像素，帧存储器中就有多少个元素。帧存储器中的每个元素都存放着屏幕上对应像素的颜色、亮度等信息。

决定数字图像质量的主要因素有分辨率和颜色深度两个指标。

分辨率是用水平方向的像素点和垂直方向的像素点的乘积来表示的，例如，水平方向有 1024 像素，垂直方向有 768 像素，分辨率就用 1024×768 表示。iPhone 3G 的分辨率为 480×320，iPhone 4 的分辨率为 960×640，iPhone 6G 的分辨率为 1024×768。

屏幕上每一像素的颜色信息都是用若干位二进制数据来表示的，这个数据就是图像的颜色深度，颜色深度反映了构成图像所用的颜色的总数。颜色深度与颜色总数的关系如表 13-1 所示。

表 13-1

颜色深度	颜色总数	图像名称
1	2	单色图像
4	16	16 色图像
8	256	256 色图像
16	65 536	16 位增强色
24	16 777 216	24 位真彩色

13.1.2 图像的格式

图像的基本格式是 BMP，其全称是 bitmap（即位图）。它采用一位映射的存储形式，存储构成图像的每一个点上的颜色、亮度等一些相关信息。位图适合描写风景、人物等内容，适用于表现含有大量细节的界面，其扩展名为 .bmp。软件的图像资源多数以 BMP 格式存储，多数图形图像软件都支持这种格式。一般作为资源使用的 .bmp 文件都是非压缩的，由于它能被大多数软件所接受，所以 BMP 格式又称为通用格式。存储一幅分辨率为640×480、24位真彩色的图像约需1MB的存储空间。iPhone的存储量都较小，无法采用这种格式。

JPEG 格式是由静态图像专家组制定的图像标准，其初衷是为了解决专业摄影人员高质量图片的存储问题。JPEG 文件的扩展名是 .jpg，其最大的特点就是采用很高的压缩比将图像用 JPEG 方法进行压缩，由于其利用了视觉特征，去除了人眼不敏感的冗余数据，因此尽管压缩比例到 1/10 甚至 1/20，图像质量并没有明显降低，所以 JPEG 是用最经济的存储空间得到较好图像质量的一种图像格式。这种格式的文件非常小，一般只有几万字节到一二十万字节，而色彩数可达到 24 位，因而在数码摄影中广泛采用。iPhone 的应用软件使用 UIImageView 类，它支持的图片格式除 JPG 外，还有 PNG 和 GIF 等。

13.2 组件介绍

13.2.1 图像视图组件

图像视图组件也即 Image View 组件，是 iPhone 应用中常用的控件之一。图像组件是用来把图像显示给用户看的，图像显示分为静态显示和动态显示，静态显示是指图像在程序运行过程中不会发生变化，它把要显示的图像文件直接加载到图像视图组件中；而动态图像则是指要显示的图像内容是变化的。作为简单入门，在这里只介绍静态图像显示。

图像视图组件在组件箱的位置如图 13-1 所示。

Image View 对应 UIImageView 类，其继承关系如下：NSObject → UIResponder → UIView → UIImageView。

框架位于 /Sytem/Library/Frameworks/UIKit.framework。

图像视图组件的属性包括 Image View 和 View 两个区域，如图 13-2 所示，其主要在 Image View 区域中的 Image 字段进行设置。

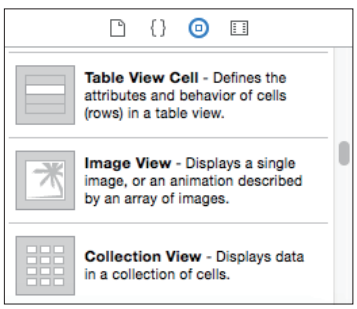

图 13-1

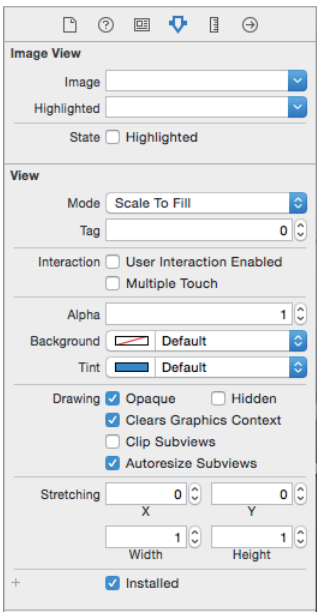

图 13-2

（1）Image View 区域。在 Image 字段右侧有一个方向朝下的箭头，单击这个下拉箭头，框内会显示可用的图像，当然这些图像都是已经添加到工程中的文件。选中想要的图像文件，完成这个操作之后，图像将自动显示在图像视图中。

（2）View 区域。

- Mode 字段：用于指定图像在视图内部的对齐方式，默认值为中间对齐（Center）。
- Alpha 字段：定义图像的透明度，也就是图像背后内容的可见度。如果 Alpha 值小于 1.0，iPhone 应用程序将以透明的方式显示该视图，这样，位于图像视图之后的控件都是可见的。如果图像视图背后没有可显示的内容，应将 Alpha 值设置为 1.0，这是因为系统在绘制透明视图时需要额外的资源开销。

13.2.2 开关组件

开关组件也即 Switch 组件，是很像开关的组件，可以在界面创建“开 / 关”按钮，其功能是用来控制布尔数据类型。开始执行时，Switch 组件呈“开”状态，单击组件后会改变为“关”状态，再单击又呈“开”状态。Switch 组件常用属性如表 13-2 所示。

表 13-2

属性名称	说明
State	组件状态，有 On 和 Off 两种，默认值为 On
On Tint	组件状态为 On 时左边圆形的颜色，默认为绿色
Thumb Tint	组件状态为 On 时右边圆形的颜色，默认为白色

Switch 组件大小是固定的，不允许用户变更。系统默认 Switch 组件的宽为 51 像素，高为 31 像素。

如果不喜欢组件原来的颜色，则可以使用 On Tint 和 Thumb Tint 属性更改组件左边和右边的颜色。

13.2.3 滑动器组件

滑动器组件也即 Slider 组件，是一个像滑动电阻器的组件，它是改变数值的工具。Slider 组件只有一个按钮，拖动按钮就可改变目前的属性值（在程序中称为 Value 属性）。Slider 组件常用属性如表 13-3 所示。

表 13-3

属性名称	说明
Value/Minimum	组件可设置的最小值，默认值为 0
Value/Maxmum	组件可设置的最大值，默认值为 1
Current	组件当前的设置值，默认值为 0.5

续表

属性名称	说明
Minimum Track Tint	组件按钮左边的颜色，默认为蓝色
Maxmum Track Tint	组件按钮右边的颜色，默认为灰色
Events/Continuous Updates	如果选中此项目，则在拖动按钮时组件值会随时更新；若未选中，则在拖动按钮时组件值不会更新，直到放开按钮时才更新。默认为选中

13.3 色彩的变化的设计

“色彩的变化”项目的设计总窗口如图 13-3 所示。

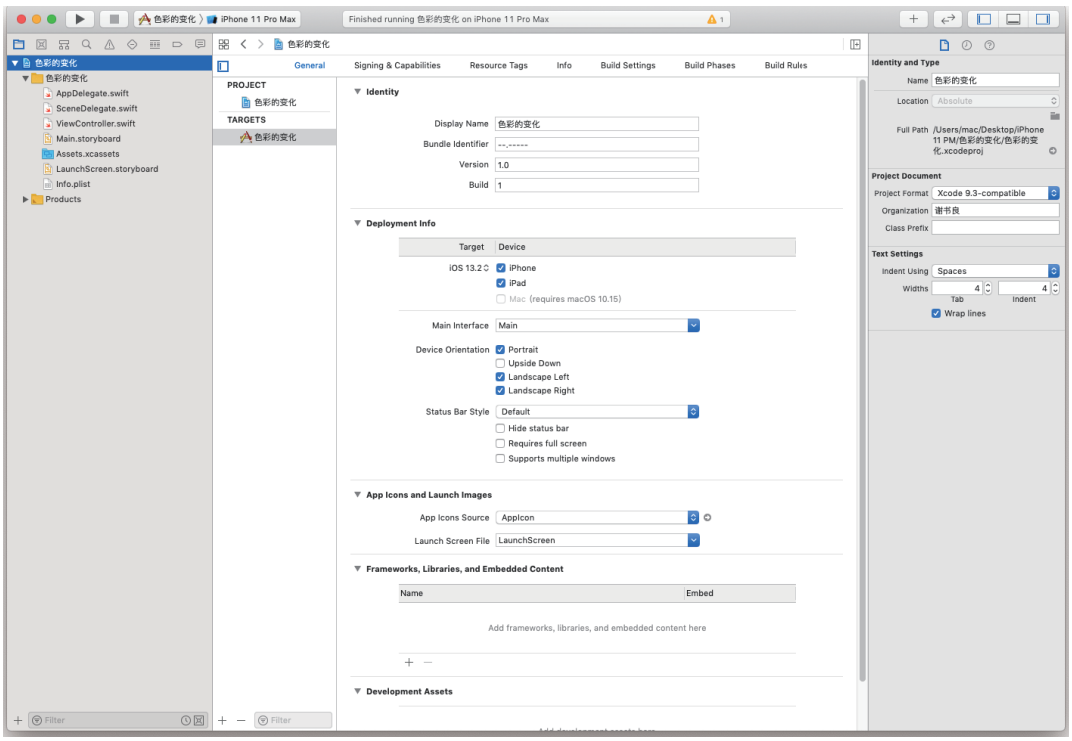

图 13-3

设计步骤如下。

步骤 1：创建工程项目——色彩的变化

创建过程类似图 11-21。项目名为“色彩的变化”。

步骤 2：设计用户界面

用户界面在模拟器上显示如图 13-4 所示。

界面之素由一个 Switch 组件和一个 Slider 组件组成，如图 13-5 所示。

步骤 3：编写程序

在图 13-6 所示的项目文件浏览区双击 ViewController.swift 文件。

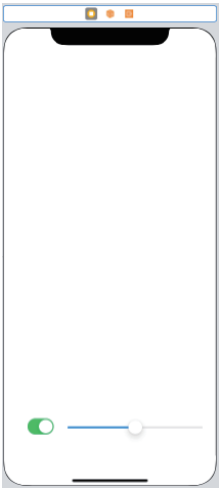

图 13-4

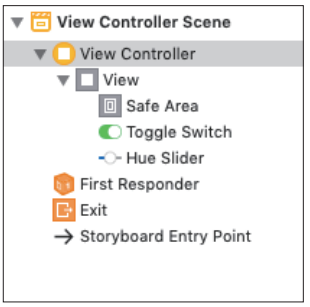

图 13-5

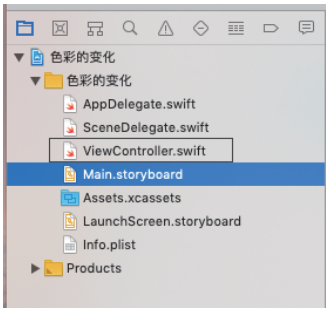

图 13-6

打开代码框架，编写如下代码：

```
import UIKit

class ViewController: UIViewController
{
    @IBOutlet weak var toggleSwitch: UISwitch!
    @IBOutlet weak var hueSlider: UISlider!

    let kOnOffToggle="onOff"
    let kHueSetting="hue"

    @IBAction func setBackgroundHueValue(_ sender: AnyObject?)
    {

        let userDefaults: UserDefaults = UserDefaults.standard

        userDefaults.set(toggleSwitch.isOn, forKey: kOnOffToggle)
        userDefaults.set(hueSlider.value, forKey: kHueSetting)
        userDefaults.synchronize()

        if toggleSwitch.isOn
        {
            view.backgroundColor=UIColor(hue: CGFloat(hueSlider.value),
                saturation: 0.75, brightness: 0.75, alpha: 1.0)
        }
        else
        {
            view.backgroundColor=UIColor.white
        }
```

```
    }


    override func viewDidLoad()
    {
        super.viewDidLoad()
        //Do any additional setup after loading the view, typically
        //from a nib.
        let userDefaults: UserDefaults = UserDefaults.standard
        hueSlider.value=userDefaults.float(forKey: kHueSetting)
        toggleSwitch.isOn=userDefaults.bool(forKey: kOnOffToggle)

        setBackgroundHueValue(nil)
    }

}
```

代码分析：

程序开始设置了 Switch 组件和 Slider 组件的对象变量 toggleSwitch 和 hueSlider，定义了两个常量 kOnOffToggle 和 kHueSetting，并赋予初始值。后面用了两个方法控制两个组件的装入和使用。

步骤 4：完成插座和动作的关联，实现界面组件元素与程序代码的连接

此项目需要建立界面组件元素与程序代码的连接，方法比较简单。首先用右键把 View Controller 按钮拖曳至 Switch 组件上释放，在弹出的快捷菜单中选中 toggleSwitch，如图 13-7 所示；其次用右键把 View Controller 按钮拖曳至 Slider 组件上释放，在弹出的快捷菜单中选中 hueSlider，如图 13-8 所示，至此完成界面组件元素与程序代码的连接。

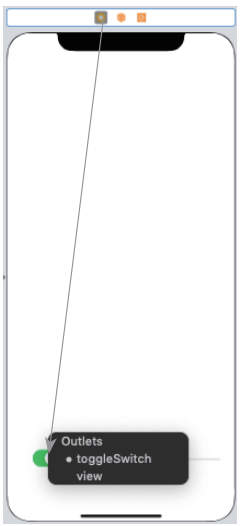

图 13-7

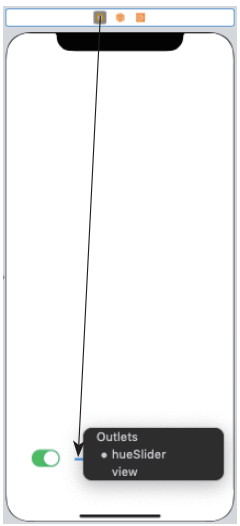

图 13-8

步骤 5：运行应用程序

按 Command+R 组合键后，用户界面在模拟器上显示如图 13-9 所示。

单击 Switch 组件后用户界面在模拟器上显示如图 13-10 所示。

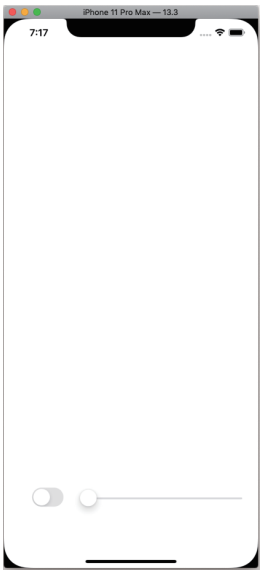

图 13-9

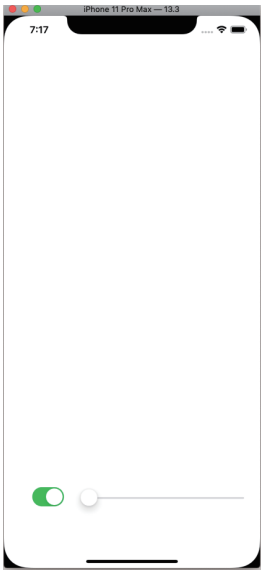

图 13-10

拖动 Slider 组件按钮后，颜色分别变化为黄色、绿色、蓝色、紫色、红色。

“色彩的变化”项目设计至此结束。

13.4 照片切换器的设计

设计步骤如下。

步骤 1：创建工程项目——照片切换器

创建过程类似图 11-21。项目名为“照片切换器”。

步骤 2：设计用户界面

用户界面在模拟器上显示如图 13-11 所示。

界面元素由一个 Image View 组件、两个 Button 组件和一个用于界面标题的 Label 组件构成，如图 13-12 所示。

步骤 3：编写程序

在图 13-13 所示的项目文件浏览区双击 ViewController.swift 文件。

图 13-11

图 13-12

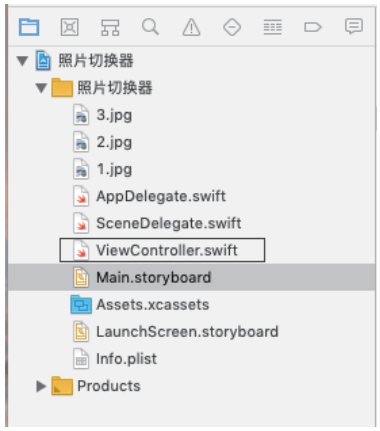

图 13-13

打开代码框架，编写如下代码：

```
import UIKit

class ViewController: UIViewController
{

    @IBOutlet var imagePhoto:UIImageView!


    @IBAction func pic1Click(sender:UIButton)
    {
        imagePhoto.image=UIImage(named:"2.jpg")
    }

    @IBAction func pic3Click(sender:UIButton)
    {
        imagePhoto.image=UIImage(named:"3.jpg")
    }

}
```

代码分析：

此程序代码比较简单，首先定义了一个图像类的对象变量 imagePhoto，然后在两个方法中完成按钮触发事件，实现两幅照片的切换。

步骤 4：完成插座和动作的关联，实现界面组件元素与程序代码的连接

用右键把 View Controller 按钮拖曳至图像视图组件上释放，在弹出的如图 13-14 所示的快捷菜单中选中图像视图组件的对象名 imagePhoto。

用右键分别把两个按钮拖曳至 View Controller 按钮上释放，在弹出的如图 13-15 和图 13-16 所示的快捷菜单中分别选中相应按钮组件对应的方法名 pic1Click: 和 pic3Click:。

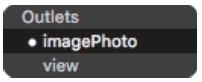

图 13-14

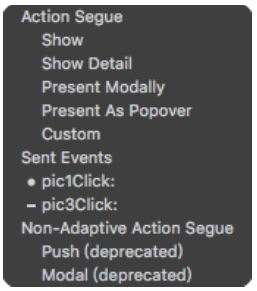

图 13-15

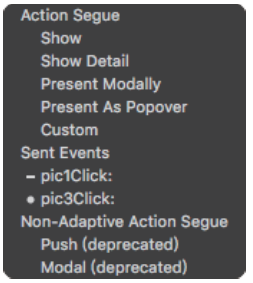

图 13-16

完成插座和动作的关联之后，要及时按 Command+S 组合键保存。

最后右击 View Controller 按钮，在弹出的如图 13-17 所示的快捷菜单中可以查看到全部连接信息。

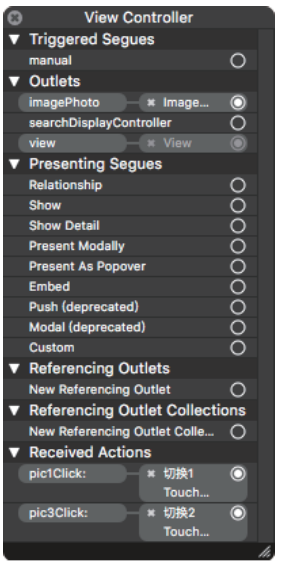

图 13-17

步骤 5：运行应用程序。

按 Command+R 组合键后，用户界面在仿真器上显示如图 13-18 所示，显示的是预先设置的一幅照片。

单击“切换 1”按钮，在仿真器上显示如图 13-19 所示的照片；单击“切换 2”按钮，在仿真器上显示如图 13-20 所示的照片。

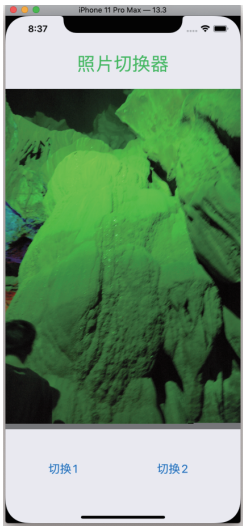

图 13-18

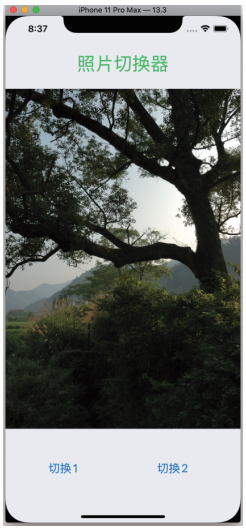

图 13-19

图 13-20

整个项目至此全部设计完毕。

13.5 照片浏览器的设计

设计步骤如下。

步骤 1：创建工程项目——照片浏览器

创建过程类似图 11-21。项目名为“照片浏览器”。

步骤 2：设计用户界面

用户界面在模拟器上显示如图 13-21 所示。

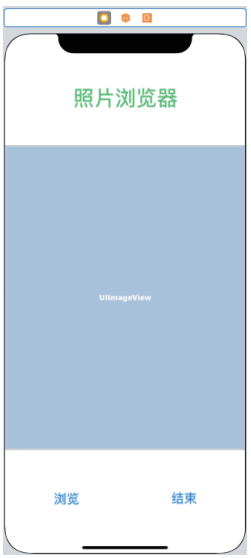

图 13-21

图 12-1

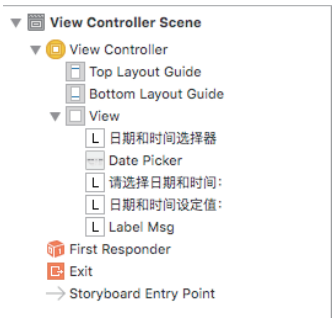

图 12-2

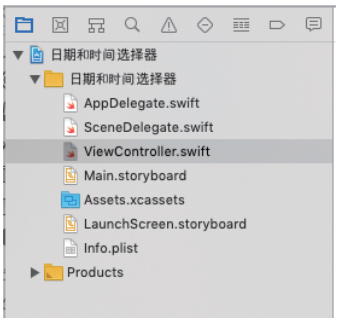

图 12-3

```
import UIKit

class ViewController: UIViewController {

    var dateFormatter=DateFormatter()                    //声明日期时间格式变量

    @IBOutlet var datePicker:UIDatePicker!               //日期时间组件连接
    @IBOutlet var labelMsg:UILabel!                      //显示信息的标签组件连接

    //用户选取日期就更新显示
    @IBAction func dataChange(sender:UIDatePicker)
    {
        labelMsg.text=dateFormatter.string(from: datePicker.date)
    }

    override func viewDidLoad()
    {
        super.viewDidLoad()
        //Do any additional setup after loading the view, typically
          from a nib
          datePicker.datePickerMode=UIDatePicker.Mode.dateAndTime
        //显示模式
        datePicker.locale=NSLocale(localeIdentifier:"zh_CN") as Locale
        //用简体中文显示
        datePicker.date=NSDate() as Date
        dateFormatter.dateFormat=" 公元 y 年 M 月 d 日 hh 点 mm 分 ss 秒 "
        //显示格式
        labelMsg.text=dateFormatter.string(from: datePicker.date)
```

```
        }

    }
```

代码分析：

用 var dateFormatter=DateFormatter() 声明日期时间格式变量，此变量在 dataChange() 和 viewDidLoad() 中都会使用，故在此声明为全局变量；

用 dataChange() 方法控制当用户选取日期和时间时立即在标签组件中按日期时间组件设置的格式更新显示；

用 datePicker.datePickerMode=UIDatePicker Mode.DateAndTime 设置组件的显示模式为日期和时间；

用 datePicker.locale=NSLocale(localeIdentifier:"zh_CN") 设置组件的显示地区语言为简体中文；

用 datePicker.date=NSDate() 设置组件显示的为当前的日期和时间；

用 dateFormatter.dateFormat="公元 y 年 M 月 d 日 hh 点 mm 分 ss 秒" 设置组件的显示格式；

用 labelMsg.text=dateFormatter.string (datePicker.date) 在标签组件中显示程序开始执行时的日期和时间。

步骤 4：完成插座和动作的关联，实现界面组件元素与程序代码的连接

用右键把 View Controller 按钮拖曳至 datePicker 组件上释放，在弹出的如图 12-4 所示的快捷菜单中选中日期时间组件的对象名 datePicker。接着用右键把 datePicker 组件起拖曳至 View Controller 按钮，在弹出的快捷菜单中分别选中相应按钮组件对应的方法名 dataChange。

再用右键把 View Controller 按钮拖曳至下方的标签组件上释放，在弹出的如图 12-5 所示的快捷菜单中选中标签组件的对象名 labelMsg。

完成插座和动作的关联之后，要及时按 Command+S 组合键保存。

最后右击 View Controller 按钮，在弹出的如图 12-6 所示的快捷菜单中可以查看全部连接信息。

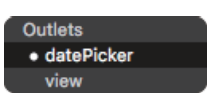

图 12-4

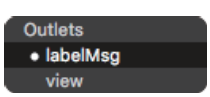

图 12-5

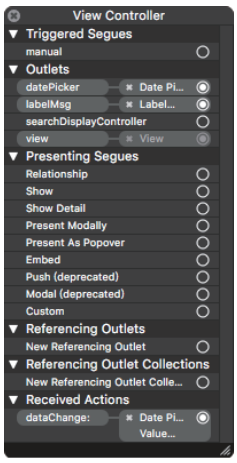

图 12-6

步骤 5：运行应用程序

按 Command+R 组合键后，用户界面在仿真器上显示如图 12-7 所示，显示的是系统当前的日期和时间。

在选取新的日期和时间之后，显示的是更新的日期和时间，如图 12-8 所示。

图 12-7

图 12-8

至此，“日期和时间选择器”项目全部设计完毕。

12.3 数据查询器的设计

数据查询器的设计总窗口如图 12-9 所示。

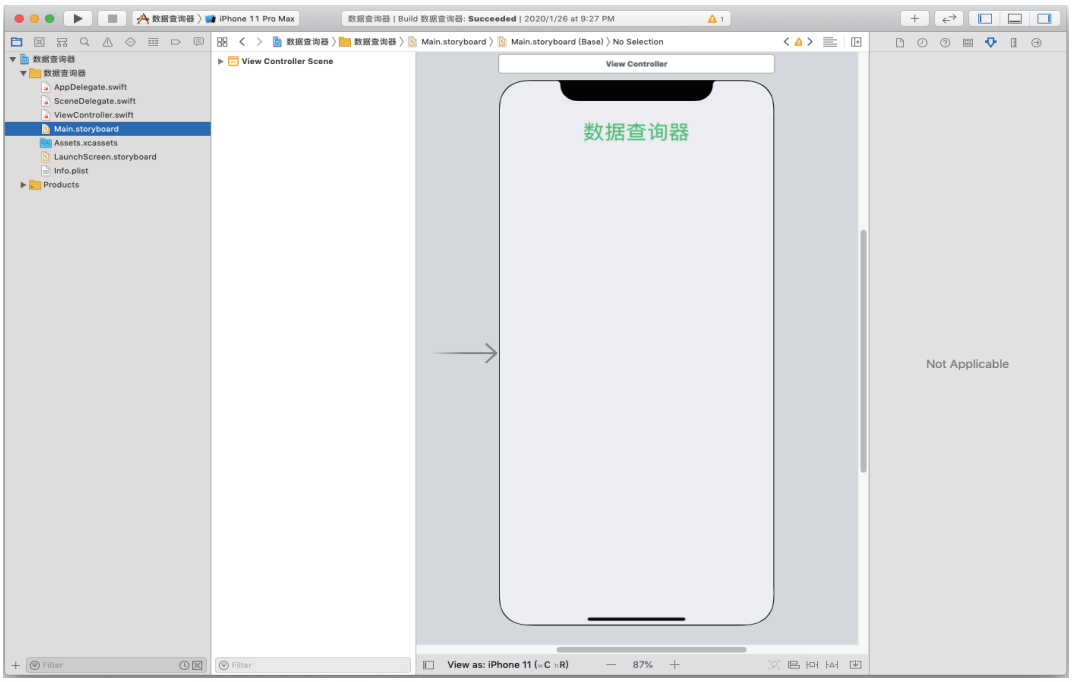

图 12-9

设计步骤如下。

步骤 1：创建工程项目——数据查询器

创建过程类似图 11-21，项目名为“数据查询器”。

步骤 2：设计用户界面

该项目全部是由程序代码运行生成的，因而是一个空的界面，其中只有一个标志项目名称的静态标签，如图 12-10 所示。

步骤 3：编写程序

在图 12-11 所示的项目文件浏览区双击 ViewController.swift 文件，打开代码框架，编写如下代码：

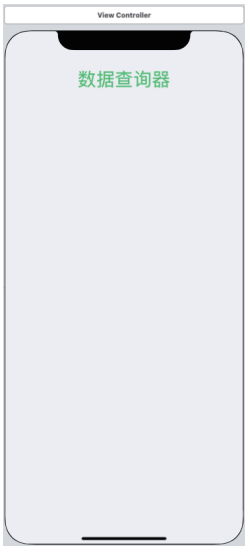

图 12-10

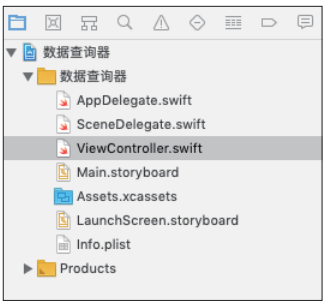

图 12-11

```
import UIKit

class ViewController: UIViewController,UITableViewDataSource
{

    var countries:Dictionary<String,[String]>=
    ["Hh":[" 黄海 "," 男 "," 重庆 ","13667890123","QQ:678901234","hh@126.com"],
    "Lb":[" 李斌 "," 男 "," 上海 ","13612345678","QQ:123456789","lb@126.com"],
    "Lm":[" 李明 "," 男 "," 西安 ","13678901234","QQ:789012345","lm@126.com"],
    "Qx":[" 钱新 "," 男 "," 天津 ","13645678901","QQ:456789012","qx@126.com"],
    "Sl":[" 宋玲 "," 女 "," 北京 ","13634567890","QQ:345678901","sl@126.com"],
    "Sm":[" 孙梅 "," 女 "," 广州 ","13689012345","QQ:890123456","sm@126.com"],
    "Wy":[" 王英 "," 女 "," 武汉 ","13656789012","QQ:567890123","wy@126.com"],
    "Xl":[" 谢璐 "," 女 "," 南昌 ","13690123456","QQ:901234567","xl@126.com"],
    "Zq":[" 周青 "," 男 "," 南京 ","13623456789","QQ:234567890","zq@126.com"],
    "Zj":[" 赵佳 "," 女 "," 北京 ","13601234567","QQ:012345678","zj@126.com"]]
```

```
        var keys:[String]=[]

        override func viewDidLoad()
        {
            super.viewDidLoad()
            //Do any additional setup after loading the view

               keys=Array(countries.keys).sorted()

            let screenRect=UIScreen.main.bounds
            let tableRect=CGRect(x:0,y:20,width:screenRect.size.width,height:
screenRect.size.height-20)

            let tableView=UITableView(frame:tableRect)

            tableView.dataSource=self
            self.view.addSubview(tableView)
        }
        //返回键名数组的长度作为表格中章节的数目
        func numberOfSections(in tableView: UITableView) -> Int
        {
            return keys.count
        }

        //根据键名对应的键值，返回数组的长度作为指定章节的单元格数列数量
        func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int
        {
            let subCountries=countries[keys[section]]
            return(subCountries?.count)!
        }

        //返回键名数组的键名作为表格中章节的标题文字
        func tableView(_ tableView: UITableView, titleForHeaderInSection
section:Int) -> String? {
            return keys[section]
        }

        //返回键名数组作为索引序列的内容
        func sectionIndexTitles(for tableView: UITableView) -> [String]?
        {
            return keys
        }
```

```
        func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell
        {
            let identifier="reusedCell"
            var cell=tableView.dequeueReusableCell(withIdentifier: identifier)
            if(cell==nil)
            {
                 cell=UITableViewCell(style:UITableViewCell.CellStyle.default,
reuseIdentifier:identifier)
            }

            //根据 IndexPath 参数的章节值获得当前单元格所在的章节序号并组成数组
            let subCountries=countries[keys[(indexPath as NSIndexPath).section]]
            //根据 IndexPath 参数的 row 值获得当前单元格所在章节行号并完成初始化和设置
            cell?.textLabel?.text=subCountries![(indexPath as NSIndexPath).row]

            return cell!
        }
    }
```

代码分析：

该项目是设计一个带索引的表格型的通讯录，每条记录共有姓名、性别、通信地址、电话号码、QQ 号和电子邮箱六项，属于多种数据类型，所以使用了字典给表格对象提供数据源。字典对象的键作为 UITableView 的 Section（章节），字典对象的值（数组）作为 Section 中单元格的内容。

接着定义了一个数组对象 keys 用来存储按升序排列的键名序列，这个数组的长度将作为表格中章节的数目。用 keys=Array(countries.keys).sorted() 获得 countries 字典对象所有的键名，并转换为一个按升序排列的数组对象。数组对象的 sorted() 方法就是对数组进行升序排列用的。

用程序代码设计数据表格的范围为整个屏幕，显示内容的索引序列设置在数据表格的右侧。

步骤 4：完成插座和动作的关联，实现界面组件元素与程序代码的连接

此项目不需要，从略。

步骤 5：运行应用程序

按 Command+R 组合键后，用户界面在模拟器上显示如图 12-12 所示。

在右边中间索引序列条内单击 Qx 之后，在模拟器上显示如图 12-13 所示。

在右边中间索引序列条内单击 Wy 之后，在模拟器上显示如图 12-14 所示。

在右边中间索引序列条内单击 Zj 之后，在模拟器上显示如图 12-15 所示。

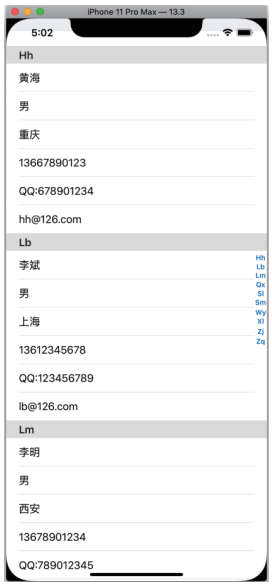

图 12-12

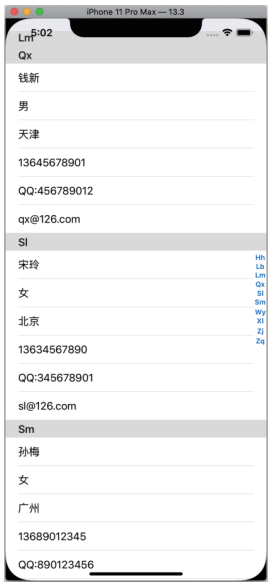

图 12-13

图 12-14

图 12-15

至此，“数据查询器”项目全部设计完毕。