第 3 篇

天气预报软件开发

——iOS 高级应用开发(高级控件与网络应用)

第 10 章　项目导学

10.1　项目简介

项目导学如图 10-1 所示。

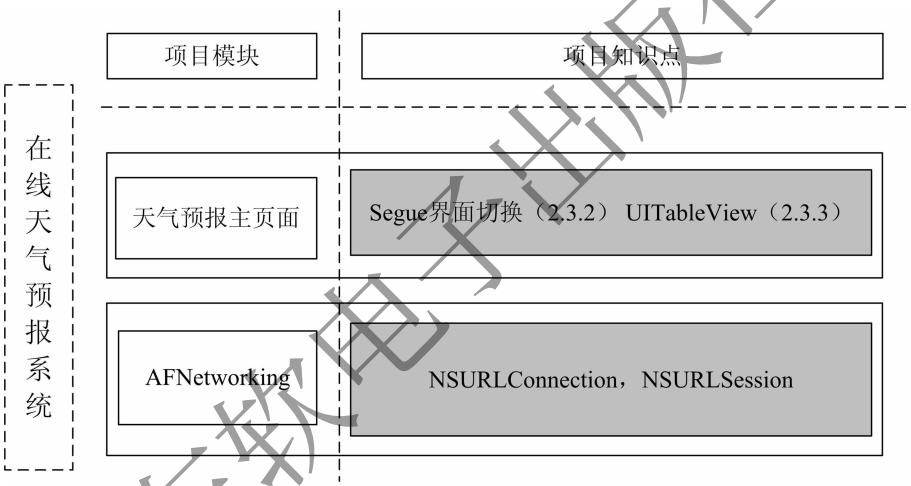

图 10-1　项目导学图

10.2　项目说明

本项目是基于 AFNetworking 开发的一款获得互联网天气信息的软件，一般来说天气信息软件也是手机软件中的必备软件之一。

天气预报软件主要让同学们掌握 ios 应用中网络编程的方法。

整个项目采用的是经典的 MVC 设计模式，MVC 即 Model（模型）View（视图）Controller（控制器）的缩写，它开发项目的时候层次清晰，可读性可维护性更强，有利于项目的扩展维护，对于大中型项目而言一个清晰的项目结构至关重要。

在天气预报主页面开发的时候，使用了 UITableView 控件，另外使用了 AFNetworking 获得网络信息。

10.3 项目目录结构的说明

目录结构如图 10-2 所示。

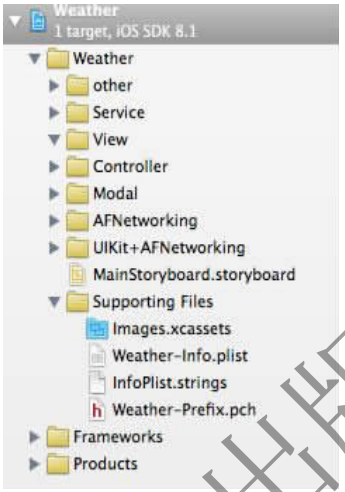

图 10-2 项目结构

其中主要目录的作用如表 10-1 所示。

表 10-1 主要目录

序号	目录	功能说明
1	Supporting Files	放置图片，媒体资源
2	Weather	项目源代码路径(在这个目录下区分 MVC)
3	Controller	放置控制器，这个目录下通过子目录新特性、首页等进行功能模块的区分。
4	Model	放置天气数据模型
5	View	放置视图(本项目没有使用)
6	Service	放置网络请求相关代码
7	Other	放置启动代码，以及工具类等

10.4 项目运行效果

项目运行效果如图 10-3 所示。

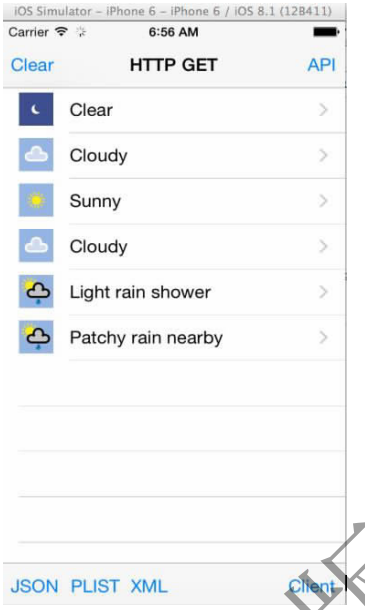

图 10-3　运行效果图

第 11 章　天气预报主页的实现

11.1　任务说明

天气预报主页主要是用来显示天气信息列表。

11.2　任务分析

天气信息列表采用 UITableView 进行显示，天气信息通过 AFNetworking 从网络获得。天气信息包括云量、湿度、pmm、压力、温度、风速、雨量、能见度等。

11.3　任务实施

11.3.1　AFNetworking 介绍

在 iOS 7 中，苹果推出了 NSURLSession 作为新的、优先考虑的连接网络的方法，而不是旧的 NSURLConnection API。使用 NSURLSession API 绝对是一个有效的方法，然而，有一个更加流行的第三方的网络框架 AFNetworking。

AFNetworking 的 2. x 版本是建立在 NSURLSession 基础之上的，所以你能得到 NSURLSession 的所有的优点的同时，你也能得到了很多额外的很酷的功能，例如序列化、可达性支持、UIKit 集成（如在 UIImageView 中异步再加载图片）以及更多的特性。

AFNetworking 也是一个最广泛使用的开源项目，在 GitHub 上超过 10000 星，2600 关注。

11.3.2　开始创建项目

首先我们开始一个没有 AFNetWorking 代码的项目，只是把 mainstoryboard. storyboard 打开，你会看到三个视图控制器，如图 11-1 所示。

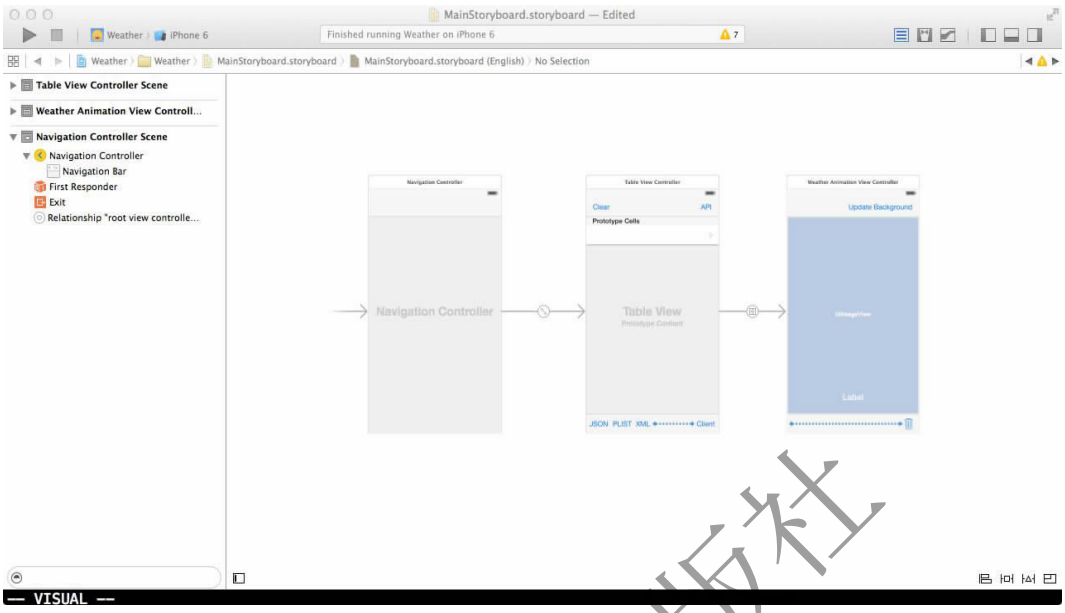

图 11-1　三个视图控制

从左到右，他们是：

(1)一个顶级导航控制器。

(2)一个表格视图控制器，将显示天气，每天一行。

(3)一个自定义的视图控制器(WeatherAnimationViewController)，当用户在一个表格视图单元格上点击时，将显示为一个单一的一天天气的详细情况。

Command＋R 编译和运行这个项目。你会看到用户界面出现，但什么都没有。这是因为应用程序需要的天气数据以及网络相关的代码还没被加到这个项目中。在本章的学习中我们会做这些事情。

11.3.3　把 AFNetworking 加入项目

在 GitHub 下载最新的版本的压缩包。当你解压缩文件，你会看到，它包括几个子文件夹和项目。它包括子文件夹 AFNetworking 和另一个文件夹叫 UIKit＋AFNetworking，如图 11-2 所示。

当在 xcode 中添加文件夹时出现添加文件夹选项对话框，确保【复制内容到目标群体的文件夹(如果需要)】和【为添加的任何文件夹创建组】这两项都被选中了，如图 11-3 所示。

要完成框架的整合，还需要在 weather－prefix.pch 中加入头文件，这个文件支持预编译，在文件中添加这行：

```
#import "AFNetworking.h";
```

添加 AFNetworking 到预编译的头，意味着框架将自动包括在所有的 AFNetworking 项目源文件。

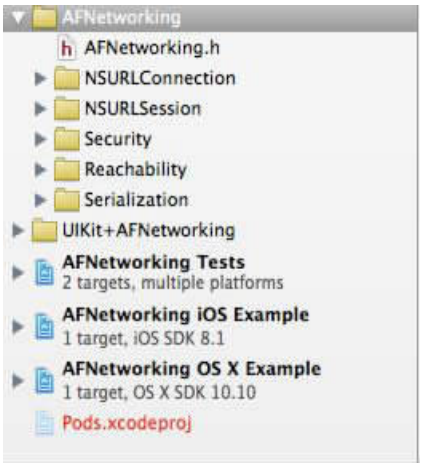

图 11-2　子文件夹示意

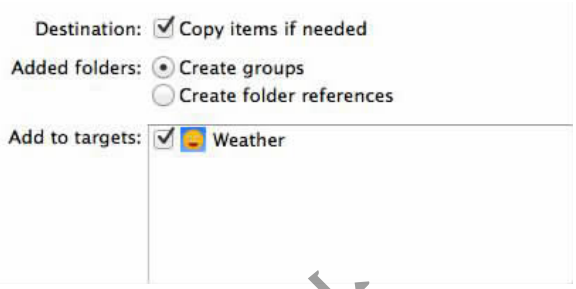

图 11-3　文件夹选项对话框

11.3.4　操作 JSON

AFNetworking 足够智能能够加载和处理结构化数据，以及普通的 HTTP 请求的网络框架。特别是，它支持 JSON，XML 和属性列表(plists)。你可以不通过 AFNetworking 下载一些 JSON 类型数据，比如使用 NSURLConnection，然后通过一个解析器(像内置 NSJSONserialization)来运行它，但这样做比较麻烦，AFNetworking 能做所有的一切。

Web 服务返回的天气数据的三种不同的格式-JSON，XML 和 plist。你可以通过使用这些 URL 返回的数据来观察。

http://www.raywenderlich.com/demos/weather_sample/weather.php?format=json

http://www.raywenderlich.com/demos/weather_sample/weather.php?format=xml

http://www.raywenderlich.com/demos/weather_sample/weather.php?format=plist（这个链接在有的浏览器中不支持）

JSON 数据格式

```
{
    "data": {
        "current_condition": [
            {
                "cloudcover": "16",
                "humidity": "59",
                "observation_time": "09:09 PM",
            }
          ]
        }
}
```

当用户点击天气预报主界面 JSON 按钮的时候，应用程序将加载从天气服务器进程得到的 JSON 数据。在 WTTablevViewController.m 文件中，发现 jsonTapped 方法名，用以下代码填充：

```
1.- (IBAction)jsonTapped:(id)sender
```

```
2.{
3.// 1
4.NSString * string = [NSString stringWithFormat:@"%@weather.php? format=json",BaseURLString];
5.NSURL * url = [NSURL URLWithString:string];
6.NSURLRequest * request = [NSURLRequest requestWithURL:url];
7.// 2
8.AFHTTPRequestOperation * operation = [[AFHTTPRequestOperation alloc] initWithRequest:request];
9.operation.responseSerializer = [AFJSONResponseSerializer serializer];
10. [operation setCompletionBlockWithSuccess: ^ (AFHTTPRequestOperation * operation, id
responseObject) {
11.// 3
12.self.weather = (NSDictionary * )responseObject;
13.self.title = @"JSON Retrieved";
14.[self.tableView reloadData];
15.} failure:^(AFHTTPRequestOperation * operation, NSError * error) {
16.// 4
17.UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Error Retrieving Weather"
18.message:[error localizedDescription]
19.delegate:nil
20.cancelButtonTitle:@"Ok"
21.otherButtonTitles:nil];
22.[alertView show];
23.}];
24.// 5
25.[operation start];
26.}
```

太棒了，这是你的第一段 AFNetworking 代码，因为这都是新的，以下会解释这一代码段。

(1)首先创建一个字符串，从基础 URL 字符串得到完整的 URL。这个字符是用于创建一个 NSURL 对象，这个 USURL 对象是用来创建一个 NSURLRequest 对象。

(2)AFHTTPRequestOperation 是处理在网络上的关于 HTTP 传输相关功能的类。如果你设置响应结果使用 JSON 类型，AFNetworking 将进行 JSON 解析。

(3)block 成功运行时，请求成功。JSON 序列化程序解析接收到的数据，并返回存储到字典，这些数据存储在天气属性。

(4)block 运行失败时——如网络不可用。如果这一切发生的时候，会显示一个简单的警告和错误消息。

(5)网络连接开始运行。

你可以看到，AFNetworking 是非常简单的使用。在短短的几行代码，你可以创建一个网络操作，下载并解析其响应。

现在天气数据存储在 self.weather 中，你需要显示它。找到 TableView numberOfRowsIn

Section:用以下方法取代它:

```
1. - ( NSInteger ) tableView: ( UITableView * ) tableView numberOfRowsInSection:
(NSInteger)section
2.{
3.     if(! self.weather)
4.     return 0;
5.     switch (section) {
6.         case 0: {
7.           return 1;
8.                   }
9.        case 1: {
10.NSArray * upcomingWeather = [self.weather upcomingWeather];
11.return [upcomingWeather count];
12.                  }
13.         default:
14.return 0;
15.}
16.}
```

表格视图有两部分:显示当前的天气和显示即将到来的天气。“等一下!”,你可能会想。这是什么样的[self.weather upcomingweather]? 如果是一个普通的老 self.weather 字典,如何知道未来天气?

为了轻松显示天气信息,我增加了两个字典分类。

```
1.NSDictionary+weather
2.NSDictionary+weather_package
```

这些类添加一些简单的方法,使它更容易访问数据元素。仍在 WTTableViewController.m 文件中,发现 TableView cellForRowAtIndexPath:用以下方法替代。

```
1.-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
2.{
3.static NSString *CellIdentifier = @"WeatherCell";
4.UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
5.forIndexPath:indexPath];
6.NSDictionary *daysWeather = nil;
7.switch (indexPath.section) {
8.     case 0: {
9.         daysWeather = [self.weather currentCondition];
10.         break;
11.               }
12.     case 1: {
13.NSArray *upcomingWeather = [self.weather upcomingWeather];
```

```
14. daysWeather = upcomingWeather[indexPath.row];
15.         break;
16.             }
17. default:
18. break;
19. cell.textLabel.text = [daysWeather weatherDescription];
20. // You will add code here later to customize the cell, but it's good for now.
21. return cell;
22. }
```

像上一个方法一样，这里也使用了字典分类。非常容易获得数据，当前天气数据是存储在字典中，而未来几天数据存储在数组中。现在编译运行这个项目，如图11-4所示。

以上，JSON 数据的操作就成功了。

11.3.5　操作 property Lists

property Lists 简称 plists 是苹果公司定义的一种和 xml 类似的数据格式，苹果公司用来存储一些用户设定信息，如图 11-5 所示。

图 11-4　JSON 操作成功示意图

```
<dict>
  <key>data</key>
  <dict>
    <key>current_condition</key>
      <array>
      <dict>
        <key>cloudcover</key>
        <string>16</string>
        <key>humidity</key>
        <string>59</string>
        ...
```

图 11-5　plists 数据格式示意图

上述数据格式表示：

1. 本词典用一个 key 被称为“data”，并且包含另外一本字典。

2. 被包含的字典仍然用一个 key 被称为“current_condition”，数据是一个数组。

3. 数组中包含键值对，比如 cloudcover＝16，humidity＝59。

Plists 版本的代码几乎和 JSON 版本代码一样，如下所示：

```
1. - (IBAction)plistTapped:(id)sender
2. {
```

```
NSString * string = [NSString stringWithFormat:@" % @weather.php? format=plist",
BaseURLString];
NSURL * url = [NSURL URLWithString:string];
NSURLRequest * request = [NSURLRequest requestWithURL:url];
AFHTTPRequestOperation * operation = [[AFHTTPRequestOperation alloc] initWithRequest:
request];
// Make sure to set the responseSerializer correctly
operation.responseSerializer = [AFPropertyListResponseSerializer serializer];
[operation setCompletionBlockWithSuccess:^(AFHTTPRequestOperation * operation, id
responseObject) {
self.weather = (NSDictionary * )responseObject;
self.title = @"PLIST Retrieved";
[self.tableView reloadData];
} failure:^(AFHTTPRequestOperation * operation, NSError * error) {
UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Error Retrieving
Weather"
message:[error localizedDescription]
delegate:nil
cancelButtonTitle:@"Ok"
otherButtonTitles:nil];
[alertView show];
}];
[operation start];
}
```

这是不错的:你的应用程序可以只是一个微小的更改代码接受 JSON 或者 plist 格式!编译和运行你的项目,试着敲 plist 按钮。你应该看到这样的事情,如图 11-6 所示。

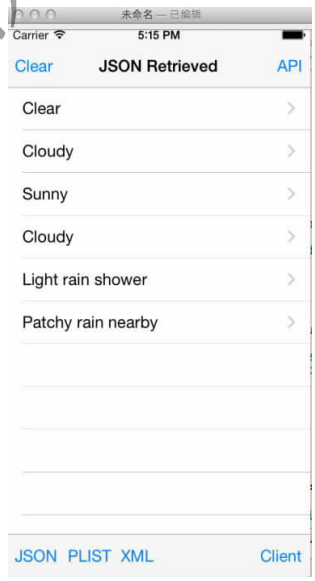

图 11-6　代码接受 JSON 或者 plist 格式效果

顶部导航栏中的按钮将清除标题栏的标题和 tableview 中的数据。

11.3.6　操作 XML

AFNetworking 能够为你解析处理 JSON 和 plists 类型数据，而使用 XML 则是更复杂一点的。这一次，你自己需要从 XML 中构建气象数据词典。幸运的是，iOS 提供了一些帮助，通过 NSXMLParser 类。仍在 WTTableViewController.m，发现 xmltapped 方法，用下面的方法，取代它的实现：

```
27.- (IBAction)xmlTapped:(id)sender
28.{
29.NSString *string = [NSString stringWithFormat:@"%@weather.php?format=xml",
30.BaseURLString];
31.NSURL *url = [NSURL URLWithString:string];
32.NSURLRequest *request = [NSURLRequest requestWithURL:url];
33.AFHTTPRequestOperation *operation = [[AFHTTPRequestOperation alloc] initWithRequest:
request];
34.// Make sure to set the responseSerializer correctly
35.operation.responseSerializer = [AFXMLParserResponseSerializer serializer];
36.[operation setCompletionBlockWithSuccess:^(AFHTTPRequestOperation *operation, id
responseObject){
37.NSXMLParser *XMLParser = (NSXMLParser *)responseObject;
38.[XMLParser setShouldProcessNamespaces:YES];
39.// Leave these commented for now (you first need to add the delegate methods)
40.// XMLParser.delegate = self;
41.// [XMLParser parse];
42.} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
43.UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Error Retrieving Weather"
44.message:[error localizedDescription]
45.delegate:nil
46.cancelButtonTitle:@"Ok"
47.otherButtonTitles:nil];
48.[alertView show];
49.}];
50.[operation start];
51.}
```

你应该很熟悉现在这些代码。最大的变化是，在成功的时候，block 中没有得到一个预处理好的 NSDictionary 对象。取而代之的是，得到 nsxmlparser 实例，在此实例中进行 xml 解析。

你需要实现一套为 nxmlparser 能够解析 XML 的委托方法。注意 XMLParser 的代理设置成这个实例，所以你将需要添加 nsxmlparser 代理方法到 wttableviewcontroller.m 中处理这些解析。

第一，更新 WTTableViewController.h 和改变类的声明，在顶部如下：

```
1.@interface WTTableViewController : UITableViewController<NSXMLParserDelegate>
```

这意味着该类将实现 NSXMLParserdelegate 协议。你要实现这些方法很容易，但首先你需要添加一些属性。这些属性在解析 XML 类型数据时使用。

```
1.@property(nonatomic, strong) NSMutableDictionary * currentDictionary; // current section being parsed
2.@property(nonatomic, strong) NSMutableDictionary * xmlWeather; // completed parsed xml response
3.@property(nonatomic, strong) NSString * elementName;
4.@property(nonatomic, strong) NSMutableString * outstring;
```

现在在 WTTableViewController.m 中实现如下方法。

```
1.- (void)parserDidStartDocument:(NSXMLParser * )parser
2.{
3.self.xmlWeather = [NSMutableDictionary dictionary];
4.}
```

这个方法在 XML 被解析的时候调用，产生一个新的字典实例，用来存储天气数据。

在上一个方法下再实现如下方法：

```
1.- (void) parser:(NSXMLParser * ) parser didStartElement:(NSString * ) elementName namespaceURI:(
2.NSString * )namespaceURI qualifiedName:(NSString * )qName attributes:(NSDictionary
3. * )attributeDict
4.{
5.self.elementName = qName;
6.if([qName isEqualToString:@"current_condition"] ||
7.[qName isEqualToString:@"weather"] ||
8.[qName isEqualToString:@"request"]) {
9.self.currentDictionary = [NSMutableDictionary dictionary];
10.}
11.self.outstring = [NSMutableString string];
12.}
```

解析器调用此方法时，它寻找一个新的元素的开始标记。当找到了一个新的元素，使用新的元素来更新 self.elementname。如果元素名称代表一种新的节点开始，还需要设置一个新的词典 self.currentdictionary，最后重置 outstring，得到新的 mutable string，来对应 XML 关联的元素。

下面实现如下代码：

```
1.- (void)parser:(NSXMLParser * )parser foundCharacters:(NSString * )string
2.{
3.if (! self.elementName)
4.return;
5.[self.outstring appendFormat:@"%@", string];
6.}
```

顾名思义，解析器调用此方法时，它在一个 XML 元素上发现新的字符串。你追加这个新的字符串到 outstring 上，这样他们就可以一次处理 XML，标记是封闭的。

接下来实现如下代码；

```
1.-(void)parser:(NSXMLParser * )parser didEndElement:(NSString * )elementName namespaceURI:(
```

```
2.NSString * )namespaceURI qualifiedName:(NSString * )qName
3.{
4.// 1
5.if ([qName isEqualToString:@"current_condition"] ||
6.[qName isEqualToString:@"request"]) {
7.self.xmlWeather[qName] = @[self.currentDictionary];
8.self.currentDictionary = nil;
9.}
10.// 2
11.else if ([qName isEqualToString:@"weather"]) {
12.// Initialize the list of weather items if it doesn't exist
13.NSMutableArray * array = self.xmlWeather[@"weather"] ?: [NSMutableArray array];
14.// Add the current weather object
15.[array addObject:self.currentDictionary];
16.// Set the new array to the "weather" key on xmlWeather dictionary
17.self.xmlWeather[@"weather"] = array;
18.self.currentDictionary = nil;
19.}
20.// 3
21.else if ([qName isEqualToString:@"value"]) {
22.// Ignore value tags, they only appear in the two conditions below
23.}
24.// 4
25.else if ([qName isEqualToString:@"weatherDesc"] ||
26.[qName isEqualToString:@"weatherIconUrl"]) {
27.NSDictionary * dictionary = @{@"value": self.outstring};
28.NSArray * array = @[dictionary];
29.self.currentDictionary[qName] = array;
30.}
31.// 5
32.else if (qName) {
33.self.currentDictionary[qName] = self.outstring;
34.}
35.self.elementName = nil;
36.}
```

当遇到结束标签时，这个方法被调用，之后检查特殊的标签：

1. 这个“current_condition”元素表明当天天气情况。把这直接加入 xmlweather 词典中。

2. “weather”元素意味着你有后续的天气信息。虽然当前天只有一天，可能会有后来的几天，所以你添加这个到天气信息的数组。

3. “value”标签只出现在其他的标签中，所以可以跳过它。

4. 这个“weatherdesc”元素和“weathericonurl”元素值需要存储在一个数组里面才可以被字典存储。这样，他们将完全匹配 JSON 和 plist 版本。

5.所有其他的元素可以像代码段 5 一样被存储。

现在实现最后的代理方法：

```
1.- (void) parserDidEndDocument:(NSXMLParser * )parser
2.{
3.self.weather = @{@"data": self.xmlWeather};
4.self.title = @"XML Retrieved";
5.[self.tableView reloadData];
6.}
```

解析器到达文件结束的时候调用此方法。在这一点上，该 xmlweather 字典，已经建设完成，所以表的视图可以被重载。

在另一个 NSDictionary 包装 xmlweather 似乎是多余的，但这使格式匹配正好与 JSON 和 plist 版本匹配。这样，所有三种数据格式可以显示相同的代码！

代理方法和属性都已经实现了，现在来实现点击 xmlTapped 方法：

```
1.- (IBAction)xmlTapped:(id)sender
2.{
3....
4.[operation setCompletionBlockWithSuccess:^(AFHTTPRequestOperation * operation, id
5.responseObject) {
6.NSXMLParser * XMLParser = (NSXMLParser * )responseObject;
7.[XMLParser setShouldProcessNamespaces:YES];
8.// These lines below were previously commented
9.XMLParser.delegate = self;
10.[XMLParser parse];
11....}];
12.}
```

现在编译运行这个项目，得到如图 11-7 所示。

图 11-7　运行效果示意图 1

11.3.7　天气图标

上边的 UI 看起来很沉闷，怎么能在表示图中显示气象信息的图标呢？我们看看之前的 JSON 格式，你会看到有关于天气图标的 URL。在每个表格单元中显示这些天气图标会给应用程序添加一些视觉兴趣。

AFNetworking 添加一个分类 UIImageView，可以异步加载图片，意味着用户界面将保持响应的同时，图标在后台下载。利用这一点，首先在 wttableviewcontroller.m 中添加分类头文件。

```
1. #import "UIImageView+AFNetworking.h"
```

找到 TableView cellForRowAtIndexPath 方法，把下面的代码粘贴在最后 return cell 之前。

```
1.cell.textLabel.text = [daysWeather weatherDescription];
2.NSURL *url = [NSURL URLWithString:daysWeather.weatherIconURL];
3.NSURLRequest *request = [NSURLRequest requestWithURL:url];
4.UIImage *placeholderImage = [UIImage imageNamed:@"placeholder"];
5.__weak UITableViewCell *weakCell = cell;
6.[cell.imageView setImageWithURLRequest:request
7.placeholderImage:placeholderImage
8.success:^(NSURLRequest *request, NSHTTPURLResponse
9.*response, UIImage *image) {
10.weakCell.imageView.image = image;
11.[weakCell setNeedsLayout];
12.} failure:nil];
```

UIImageView+AFNetworking 分类中提供 setimagewithurlrequest:和其他一些相关的方法。

成功与失败的 block 是可选的，但是如果你提供了一个成功的 block，你必须明确设置图像属性的图像视图（否则它不会被设置）。如果你不提供一个成功的 block，图像将被自动设置。

当单元格首先被创造出来，它的图标视图将显示图标占位符直到实际图标完成下载。

编译运行项目，如图 11-8 所示。

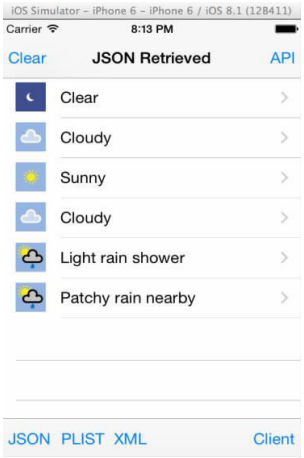

图 11-8　运行效果示意图 2

异步加载图像从未如此简单。

11.3.8 一种 RESTful 类

到目前为止你已经创建一个 AFHTTPRequestOperation 进行网络操作。另外，AFHTTPRequestOperationManager 和 AFHTTPSessionManager 旨在帮助你轻松互动一个单一的 Web 服务端点。

这些都允许你设置一个 URL 然后做几个请求到同一个终点。双方还可以监控连接的变化，编码参数，处理多形式的请求，队列的批处理操作，并帮助你执行 RESTful 的动词(GET，POST，PUT，DELETE)。“我应该使用哪一个?”，你可能会问。

如果你瞄准的是 iOS 7 及以上，使用 AFHTTPSessionManager，它内部创建和使用 NSURLSession 以及相关的对象。如果你瞄准的是 iOS 6 及以上，使用 AFHTTPRequestOperationManager，其功能类似于 AFHTTPSessionManager，但它内部使用 NSURLConnection 而不是 NSURLSession(iOS 6 不可用)。除此，这些类的功能非常相似。

在你的天气应用程序项目，你会用 AFHTTPSessionManager 进行一个 get 和 put 操作。

更新在 WTTableViewController.h 顶部的类声明：

```
1. @ interface WTTableViewController : UITableViewController < NSXMLParserDelegate, CLLocationManager Delegate,UIActionSheetDelegate>
```

在 WTTableViewController.m 中实现 clienttapped 方法：

```
1.(IBAction)clientTapped:(id)sender
2.{
3.UIActionSheet * actionSheet = [[UIActionSheet alloc] initWithTitle:@"AFHTTPSessionManager"
4.delegate:self
5.cancelButtonTitle:@"Cancel"
6.destructiveButtonTitle:nil
7.otherButtonTitles:@"HTTP GET",
8.@"HTTP POST", nil];
9.[actionSheet showFromBarButtonItem:sender animated:YES];
10.}
```

此方法创建并显示一个 action sheet，要求用户选择一个 GET 或 POST 选择。添加以下的类实现的方法，实施 actionsheet 委托方法：

```
1.－(void)actionSheet:(UIActionSheet * )actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex
2.{
3.if (buttonIndex == [actionSheet cancelButtonIndex]) {
4.// User pressed cancel －－ abort
5.return;
6.}
7.// 1
8.NSURL * baseURL = [NSURL URLWithString:BaseURLString];
```

```
9. NSDictionary * parameters = @{@"format": @"json"};
10. // 2
11. AFHTTPSessionManager * manager = [[AFHTTPSessionManager alloc]
12. initWithBaseURL:baseURL];
13. manager.responseSerializer = [AFJSONResponseSerializer serializer];
14. // 3
15. if (buttonIndex == 0) {
16. [manager GET:@"weather.php" parameters:parameters success:^(NSURLSessionDataTask
17. * task, id responseObject) {
18. self.weather = responseObject;
19. self.title = @"HTTP GET";
20. [self.tableView reloadData];
21. } failure:^(NSURLSessionDataTask * task, NSError * error) {
22. UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Error
23. Retrieving Weather"
24. message:[error localizedDescription]
25. delegate:nil
26. cancelButtonTitle:@"Ok"
27. otherButtonTitles:nil];
28. [alertView show];
29. }];
30. }
31. // 4
32. else if (buttonIndex == 1) {
33. [manager POST:@"weather.php" parameters:parameters success:^(NSURLSessionDataTask
34. * task, id responseObject) {
35. self.weather = responseObject;
36. self.title = @"HTTP POST";
37. [self.tableView reloadData];
38. } failure:^(NSURLSessionDataTask * task, NSError * error) {
39. UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Error Retrieving Weather"
40. message:[error local－izedDescription]
41. delegate:nil
42. cancelButtonTitle:@"Ok"
43. otherButtonTitles:nil];
44. [alertView show];
45. }];
46. }
47. }
```

这是上面发生了什么：

1. 你先设置 URL 地址和参数字典。

2. 然后你创造 AFHTTPSessionManager 的实例并设置其 responseserializer 到默认

JSON 序列化程序，类似以前的 JSON 的例子。

3. 如果用户按下按钮 HTTP GET，在 manager 中调用 GET 传递参数和往常一样的成功 block 以及失败的 block。

4. 你做同样的 POST 版本，在这个例子中，你是请求 JSON 响应，但是你可以很容易地按照以前的讨论完成其他两个格式。

编译和运行你的项目，点击 client 按钮，然后点击客户端的 HTTP GET 或 HTTP 后按钮启动，如图 11-9～图 11-11 所示。

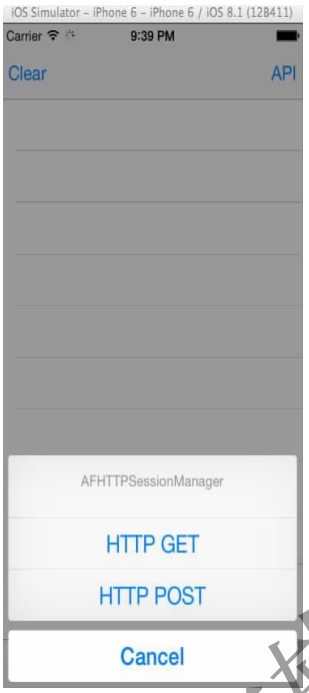

图 11-9　点击客户端的 HTTP GET 或 HTTP 后按钮启动效果图 1

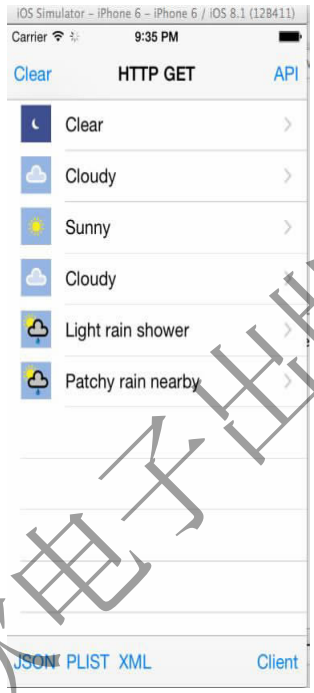

图 11-10　点击客户端的 HTTP GET 或 HTTP 后按钮启动效果图 2

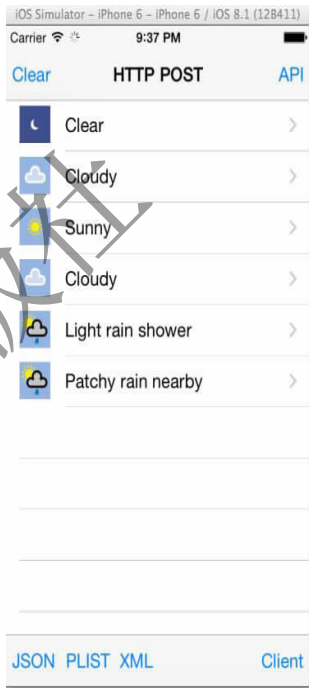

图 11-11　点击客户端的 HTTP GET 或 HTTP 后按钮启动效果图 3

到现在为止，你知道使用 AFHTTPSessionManager 的基本方法，但是还有一个更好的方式来使用它，这将使代码干净利落，下面我们来学习。

11.3.9　世界天气在线

你可以使用在线服务，首先你需要注册一个免费世界天气在线账户。别担心，这些会快速、容易地去做！当你注册时，你应该在你提供的邮件收到确认邮件，这将有一个链接，确认你的电子邮件地址（必填）。然后你需要通过“my account”页面请求一个免费的 API 密钥。把网页打开，拿到你的 API 密钥，因为你很快就需要它。

当你需要他们时，你已经能直接从 table view controller 创造出 AFHTTPRequestOperation 和 AFHTTPSessionManager，你的网络连接会应用在多个项目上。

AFHTTPSessionManager 有一切你连接 web 服务的 API。它会将你的网络通信的代

码从你的业务代码部分分离，并使你的网络通信的代码可重用整个项目。

你的项目目前没有 AFHTTPSessionManager 子类；让我们来解决这个问题。

首先，创建一个新类，叫它 WeatherHTTPClient 并且使它为 AFHTTPSessionManager 子类。

你想要的类来做三件事：执行 HTTP 请求更新气象数据时调用代理方法，使用用户的物理位置来获得准确的天气。

WeatherHTTPClient.h 中的代码实现如下：

```
#import "AFHTTPSessionManager.h"
@protocol WeatherHTTPClientDelegate;
@interface WeatherHTTPClient : AFHTTPSessionManager
@property (nonatomic, weak) id<WeatherHTTPClientDelegate>delegate;
+(WeatherHTTPClient *)sharedWeatherHTTPClient;
- (instancetype)initWithBaseURL:(NSURL *)url;
- (void)updateWeatherAtLocation:(CLLocation *)location forNumberOfDays:(NSUInteger)
number;
@end
@protocol WeatherHTTPClientDelegate <NSObject>
@optional
-(void)weatherHTTPClient:(WeatherHTTPClient *)client didUpdateWithWeather:(id)weather;
- (void) weatherHTTPClient: (WeatherHTTPClient *) client didFailWithError: (NSError *)
error;
@end
```

切换到 weatherhttpclient.m，添加以下的语句：

```
// Set this to your World Weather Online API Key
static NSString * const WorldWeatherOnlineAPIKey = @"PASTE YOUR API KEY HERE";
static NSString * const WorldWeatherOnlineURLString = @"http://api.worldweatheronline.
com/free/v1/";
```

确保使用你的实际世界天气在线 API 密钥取代@“PASTE YOUR KEY HERE”，实现如下方法：

```
static NSString * const WorldWeatherOnlineURLString = @"http://api.worldweatheronline.
com/free/v1/";
+(WeatherHTTPClient *)sharedWeatherHTTPClient
{
static WeatherHTTPClient *_sharedWeatherHTTPClient = nil;
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{
_sharedWeatherHTTPClient = [[self alloc] initWithBaseURL:[NSURL
URLWithString:WorldWeatherOnlineURLString]];
});
return _sharedWeatherHTTPClient;
}
```

```
13. - (instancetype)initWithBaseURL:(NSURL * )url
14. {
15. self = [super initWithBaseURL:url];
16. if (self) {
17. self.responseSerializer = [AFJSONResponseSerializer serializer];
18. self.requestSerializer = [AFJSONRequestSerializer serializer];
19. }
20. return self;
21. }
```

sharedWeatherHttpClient 方法的使用，确保共享对象只分配一次。使用基本 URL 初始化对象和设置请求以及从 Web 服务中得到期望的 JSON 格式响应。

```
1. - (void)updateWeatherAtLocation:(CLLocation * )location forNumberOfDays:(NSUInteger)
2. number
3. {
4. NSMutableDictionary * parameters = [NSMutableDictionary dictionary];
5. parameters[@"num_of_days"] = @(number);
6. parameters[@"q"] = [NSString stringWithFormat:@" % f, % f",location.coordinate.latitude,
7. location.coordinate.longitude];
8. parameters[@"format"] = @"json";
9. parameters[@"key"] = WorldWeatherOnlineAPIKey;
10.[self GET:@"weather.ashx" parameters:parameters success:^(NSURLSessionDataTask
11. * task, id responseObject) {
12. if ([self.delegate respondsToSelector:@selector(weatherHTTPClient:didUpdateWithWeather:)])
13. {
14.[self.delegate weatherHTTPClient:self didUpdateWithWeather:responseObject];
15. }
16. } failure:^(NSURLSessionDataTask * task, NSError * error) {
17. if ([self.delegate respondsToSelector:@selector(weatherHTTPClient:
18. didFailWithError:)]) {
19.[self.delegate weatherHTTPClient:self didFailWithError:error];
20. }
21. }];
22. }
```

此方法调用了世界天气在线服务，以获取一个特定位置的天气。一旦世界天气在线服务已经加载了气象数据，它需要一些通信方式返回给数据需求端。在此感谢 weatherhttpclientdelegate 协议及其委托的方法，在上面的块的运行成功与失败代码中可以通知控制器，天气已更新为一个给定的位置。这样，该控制器可以更新一下显示。

现在是时候把最后的拼在一起。weatherhttpclient 期望位置，还有一个定义的委托协议，所以你需要利用这些更新 wttableviewcontroller 类。

在 WTTableViewController.h 中

```
# import "WeatherHTTPClient.h"
```

```
1. #import "WeatherHTTPClient.h"
2. @interface WTTableViewController : UITableViewController < NSXMLParserDelegate,
CLLocationManagerDelegate,
3.UIActionSheetDelegate, WeatherHTTPClientDelegate>
```

添加了一个新的定位的属性：

```
1.@property (nonatomic, strong) CLLocationManager *locationManager;
```

在 WTTableViewController.m 的 viewDidLoad 方法底部添加以下代码：

```
1.self.locationManager = [[CLLocationManager alloc] init];
2.self.locationManager.delegate = self;
```

这两行代码，在视图加载时被调，确定用户的位置，当视图加载。通过回调函数通知地点位置。具体实现如下：

```
1. - (void) locationManager:(CLLocationManager *) manager didUpdateLocations:(NSArray
*)locations
2.{
3.// Last object contains the most recent location
4.CLLocation *newLocation = [locations lastObject];
5.// If the location is more than 5 minutes old, ignore it
6.if([newLocation.timestamp timeIntervalSinceNow] > 300)
7.return;
8.[self.locationManager stopUpdatingLocation];
9.WeatherHTTPClient *client = [WeatherHTTPClient sharedWeatherHTTPClient];
10.client.delegate = self;
11.[client updateWeatherAtLocation:newLocation forNumberOfDays:5];
12.}
```

现在，当有用户的行踪的更新时，你可以调用 weatherhttpclient 实例请求的当前位置的天气。记住，weatherhttpclient 有两个委托方法，你需要执行。

添加下面的两种方法的实现：

```
1. - (void)weatherHTTPClient:(WeatherHTTPClient *)client didUpdateWithWeather:(id)weather
2.{
3.self.weather = weather;
4.self.title = @"API Updated";
5.[self.tableView reloadData];
6.}
7. - (void)weatherHTTPClient:(WeatherHTTPClient *)client didFailWithError:(NSError *)error
8.{
9.UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Error Retrieving
10.Weather"
11.message:[NSString
12.stringWithFormat:@" % @",error]
13.delegate:nil
14.cancelButtonTitle:@"OK" otherButtonTitles:nil];
```

```
15.[alertView show];
16.}
```

当 WeatherHTTPClient 成功，你更新天气数据和重新加载表视图。万一网络请求错误，将显示一个错误信息。

下面实现 apitapped 方法：

```
1.- (IBAction)apiTapped:(id)sender
2.{
3.[self.locationManager startUpdatingLocation];
4.}
```

编译和运行你的项目，点击 API 按钮，启动 WeatherHTTPClient 请求，你应该看到如图 11-12 所示。

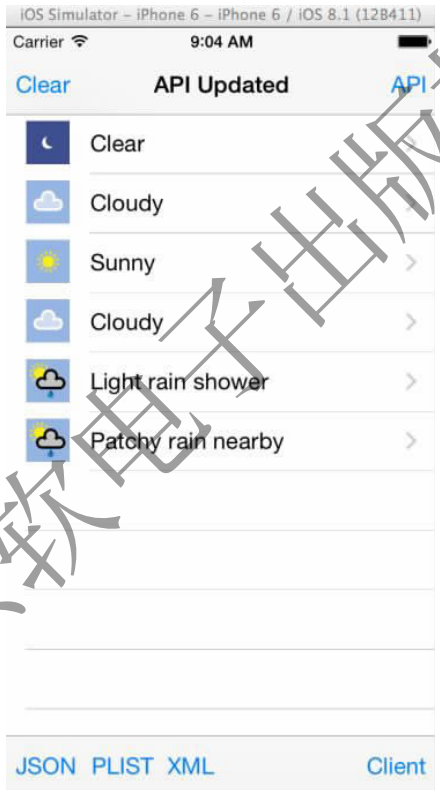

图 11-12　运行结果示意图