iOS 5 Programming Cookbook

第十四章
日期，日程表和事件

版本 1.0

翻译时间：2012-08-13

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun dymx101
Jimmylianf BeyondVincent

DevDiv 校对： laigb kyelup

DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问 DevDiv 移动开发论坛 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3、xiebaochun、dymx101、jimmylianf 和 BeyondVincent 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

更多学习资料

我们也为大家准备了 iOS6 新特征的相关内容，请参考下面的帖子：

iOS6 新特征：参考资料和示例汇总------持续更新中

目录

前言

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_前言

第 1 章　基础入门

参考帖子

[DevDiv翻译]iOS 5 Programming Cookbook翻译_第一章_基础入门

第 2 章　使用控制器和视图

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(上)

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(下)

第 3 章　构造和使用 Table View

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第三章_构造和使用 Table View

第 4 章 Storyboards

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第四章_Storyboards

第 5 章　并发

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第五章_并发

第 6 章　定位核心与地图

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第六章_核心定位与地图

第 7 章　实现手势识别的功能

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第七章_实现手势识别

第 8 章　网络，JSON，XML 以及 Twitter

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第八章_网络，JSON，XML 以及 Twitter

第 9 章　音频和视频

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第九章_音频和视频

第 10 章 通讯录

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译 第十章 通讯录

第 11 章 照相机和图片库

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十一章_照相机和图片库

第 12 章 多任务

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十二章_多任务

第 13 章 Core Data

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 中文翻译_第十三章 Core Data

第 14 章 日期，日程表和事件

14.0. 概述

Event Kit 和 Event Kit UI 框架允许 iOS 开发者访问 iOS 设备上的 Calendar 数据库。你可以使用 Event Kit 框架插入、读取、和编辑事件。Event Kit 框架允许你展示 SDK 内建的 GUI 元素，允许用户手动操作 Calendar 数据库。在这一章，我们首先会关注 Event Kit 框架，然后学习 Event Kit UI 框架。

使用 Event Kit 框架，程序员可以在用户不知情的情况下修改其 Calendar 数据库。但是，这不是一个很好的行为。实际上，Apple 禁止程序员这样做，并要求我们在程序可能改变 Calendar 数据库时，总是告知用户。下面这段话引用自 Apple：

如果你的程序编辑了用户的 Calendar 数据库，在此之前，必须得到用户的确认。程序永远不要在没有得到用户特定的指令的情况下修改 Calendar 数据库。

iOS 本身带有一个内建的 Calendar 应用。Calendar 应用可以和不同类型的日程表一块工作，比如本地、CalDAV 等等。在这一章中，我们也会和不同类型的日程表打交道。为了确保你已做好准备，来运行本章一小节的代码，请创建一个 Google 账号并将其关联到 Google Calendar。要开始这件事，请打开 http://www.google.com/ calendar/。

打开上面的网址后，创建一个 Google 账号。在你完成后，将 Google Calendar 按以下步骤添加到你的 iOS 设备上：

(译者注：因为之后的截图都为英文，所以菜单和按钮等名字不做翻译，我假设你 iOS 系统的语言为英文。)

1.转到你的 iOS 设备的主屏幕。
2.进入“Settings”。
3.选择“Mail”，“Contacts”，“Calendars”。
4.选择“Add Accounts…”
5.选择“Other”。
6.在 Calendar 组下面，选择"Add CalDAV Account"按钮。
7.在 CalDAV 屏幕，输入下面的信息（如图 14-1）：
    a.Server 栏填写 www.google.com
    b.User Name 栏填写你的 Google 用户名
    c.Password 栏填写你的 Google 密码
    d.Description 栏填写描述信息
8.点击 Next

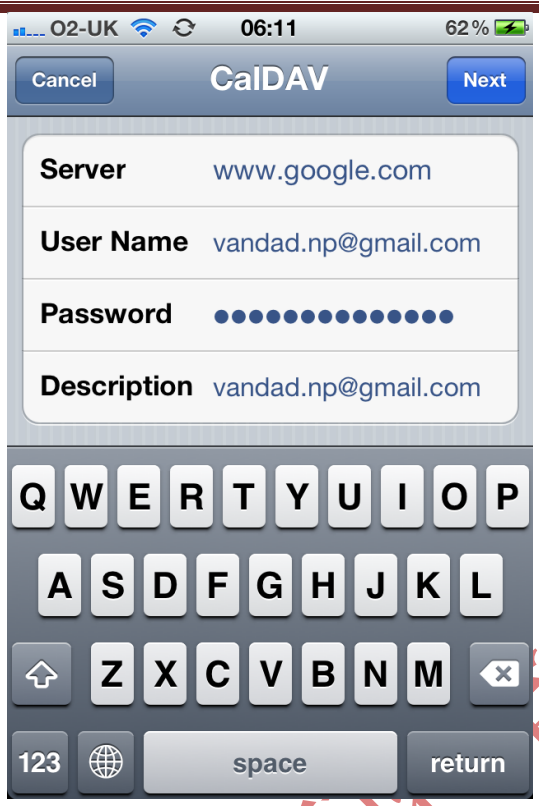

图 14-1. 向 iOS 设备添加一个 Google 账号

一旦你添加了新的 Google 账号，并且它能访问你的 IOS 设备的 Calendar 以后，你就可以看到，这个日程表出现在 Calendar 应用程序的日程表列表里面了，如图 14-2 所示。

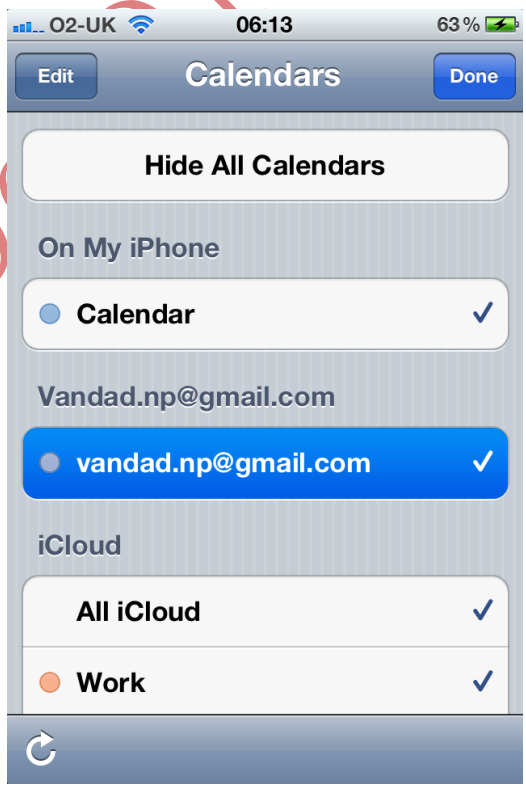

图 14-2. 新的 Google Calendar 被添加到了 iOS 设备的日程表列表中

CalDAV 是个协议，它允许访问被 Google Calendar 所使用的标准 Calendar 格式，这个格

式也被 iOS 所支持。关于 CalDAV 的更多信息，请参考 RFC 4791，地址为 http://tools.ietf.org/html/rfc4791。

本章中几乎所有章节都显示了运用一个 CalDAV 日程表的 Event Kit 和 Event Kit UI 框架的用法。在阅读他们之前，请花点时间，遵照这里提供的说明来创建一个链接到你的 iOS 设备的 CalDAV 日程表。下面是一些使用 CalDAV 日程表的好处：

1. 它容易设置
2. 它可以在不同平台间被共享，所以，对 CalDAV 日程表本地实例的改变---会被 iOS 自动的---反映到 CalDAV 服务器。这会让你更好的理解日程表如何在 iOS 中工作，你只要在线检查你的 Google Calendar，就可以确定本地修改已经被反映到那里。

3. 用 CalDAV 日程表，你可以为一个事件添加参与者，在 14.6 节中解释这一点。
4.你可以为你的 Mac 上的 iCal 添加一个 CalDAV 日程表，对你的 iOS 设备也适用，这样就能跨越所有机器进行事件同步。

为了运行这章的样例代码，你必须为你的程序添加 Event Kit 框架，有时还需添加 Event Kit UI 框架，步骤如下：
1. 点击 XCode 中你的工程的图标
2.选择你想要添加框架的那个目标
3.从屏幕的顶端选择 Build Phase
4.展开 Build Phase 中的 Link Binary with Libraries，点击左下角的“+”号按钮。
5.选择 Event Kit 和 Event Kit UI 框架并点击 Add

Mac OS X 上的 iOS 模拟器不能模拟 iOS 设备上的 Calendar 应用。为了测试本章的这些方法，你必须在真机上运行调试你的程序。本章所有的例子已经在 iPhone4 和 iPad2 上测试过。

在本章的大部分示例代码中，我们会关注手动阅读和操作日程表中的事件。如果你想使用 iOS 内建的能力来允许用户快速访问他们的日程表事件，请参考第 14.9 和 14.10 小节。

14.1. 获取日程列表

14.1.1. 问题

在尝试向用户设备上的日程列表中插入新事件之前，你想获得可用的日程表列表。

14.1.2. 方案

访问 EKEventStore 类实例的 calendars 数组属性。每一个日程表都是 EKCalendar 类型：

```
- (BOOL) application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

   EKEventStore *eventStore = [[EKEventStore alloc] init];

    /* These are the calendar types an iOS Device can have. Please note that the "type" property of an object of type
    EKCalendar
    is of type EKCalendarType. The values in the "CalendarTypes"
    array reflect the exact same values in the EKCalendarType
    enumeration, but as NSString values */
    NSArray *calendarTypes = [[NSArray alloc] initWithObjects:
                                    @"Local",
                                    @"CalDAV",
                                    @"Exchange",
                                    @"Subscription",
                                    @"Birthday", nil];

   /* Go through the calendars one by one */
   NSUInteger counter = 1;
   for (EKCalendar *thisCalendar in eventStore.calendars){

   /* The title of the calendar */
   NSLog(@"Calendar %lu Title = %@",
            (unsigned long)counter, thisCalendar.title);

    /* The type of the calendar */
    NSLog(@"Calendar %lu Type = %@", (unsigned long)counter,
             [calendarTypes objectAtIndex:thisCalendar.type]);

     /* The color that is associated with the calendar */ NSLog(@"Calendar %lu Color = %@",
             (unsigned long)counter,
             [UIColor colorWithCGColor:thisCalendar.CGColor]);

     /* And whether the calendar can be modified or not */
      if ([thisCalendar allowsContentModifications]){
           NSLog(@"Calendar %lu can be modified.", (unsigned long)counter);
           } else {
           NSLog(@"Calendar %lu cannot be modified.",
       }
            (unsigned long)counter);
          counter++; }
      self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];

     self.window.backgroundColor = [UIColor whiteColor];
     [self.window makeKeyAndVisible];
     return YES; }
```

在有 6 个日程表的 iOS 设备上运行这段代码（图 14-2）将会把类似下面的结果，打印到控制台窗口：

```
Calendar 1 Title = Birthdays
Calendar 1 Type = Birthday
Calendar 1 Color = UIDeviceRGBColorSpace 0.509804 0.584314 0.686275 1 Calendar 1 cannot be modified.
```

```
Calendar 2 Title = Calendar
Calendar 2 Type = Local
Calendar 2 Color = UIDeviceRGBColorSpace 0.054902 0.380392 0.72549 1 2 can be modified.
Calendar 3 Title = Calendar
Calendar 3 Type = CalDAV
Calendar 3 Color = UIDeviceRGBColorSpace 0.054902 0.380392 0.72549 1 3 can be modified.
Calendar 4 Title = Home
Calendar 4 Type = CalDAV
Calendar 4 Color = UIDeviceRGBColorSpace 0.443137 0.101961 0.462745 1 4 can be modified.
Calendar 5 Title = Work
Calendar 5 Type = CalDAV
Calendar 5 Color = UIDeviceRGBColorSpace 0.964706 0.309804 0 1
Calendar 5 can be modified.
Calendar 6 Title = vandad.np@gmail.com
Calendar 6 Type = CalDAV
Calendar 6 Color = UIDeviceRGBColorSpace 0.160784 0.321569 0.639216 1 6 can be modified.
```

14.1.3. 讨论

通过分配和初始化一个 EKEventStore 类型对象，你可以访问 iOS 设备上不同类型的日程表。iOS 支持如 CalDAV 和 Exchange 格式之类的常用日程表格式。EKEventStore 实例的 canlendars 属性是一个 NSArray 类型，并包含 iOS 设备上的日程表的数组。这个数组中的每个对象都是一个 EKCalendar 对象，且每个日程表都有一些属性，例如（某个属性）允许我们决定是否可以插入新事件到日程表中。

如我们将在 14.2 节中看到的，一个日程表对象仅在 allowsContentModifications 属性为 YES 时允许编辑。

你可以使用 UIColor 类的 colorWithCGColor:实例方法，来从 CGColorRef 获得 UIColor 类型。

14.1.4. 参考

14.2 节

14.2. 为日程表添加事件

14.2.1. 问题

你可能希望能在用户的日程表中创建新的事件。

14.2.2. 方案

找到你想插入新事件的日程表（请参考 14.1 小节）。用 EKEvent 的 eventWithEventStore:类方法，创建一个 EKEvent 类型的对象，然后用 EKEventStore 的 saveEvent:span:error: 实例方法将事件保存到用户的日程表中：

```
- (BOOL)       createEventWithTitle:(NSString *)paramTitle
                          startDate:(NSDate *)paramStartDate
                            endDate:(NSDate *)paramEndDate
                 inCalendarWithTitle:(NSString *)paramCalendarTitle
                 inCalendarWithType:(EKCalendarType)paramCalendarType
                              notes:(NSString *)paramNotes{

    BOOL result = NO;

    EKEventStore *eventStore = [[EKEventStore alloc] init];

    /* Are there any calendars available to the event store? */
    if ([eventStore.calendars count] == 0){
        NSLog(@"No calendars are found.");
        return NO;
    }

    EKCalendar *targetCalendar = nil;

    /* Try to find the calendar that the user asked for */
    for (EKCalendar *thisCalendar in eventStore.calendars){
        if ([thisCalendar.title isEqualToString:paramCalendarTitle] &&
            thisCalendar.type == paramCalendarType){
            targetCalendar = thisCalendar;
            break;
        }
    }

    /* Make sure we found the calendar that we were asked to find */
     if (targetCalendar == nil){
         NSLog(@"Could not find the requested calendar.");
         return NO;
        }

     /* If a calendar does not allow modification of its contents then we cannot insert an event into it */
     if (targetCalendar.allowsContentModifications == NO){
          NSLog(@"The selected calendar does not allow modifications.");
          return NO; }

     /* Create an event */
     EKEvent *event = [EKEvent eventWithEventStore:eventStore];
      event.calendar = targetCalendar;

     /* Set the properties of the event such as its title, start date/time, end date/time, etc. */
     event.title = paramTitle;
    event.notes = paramNotes; event.startDate = paramStartDate; event.endDate = paramEndDate;

    /* Finally, save the event into the calendar */
     NSError *saveError = nil;
     result = [eventStore saveEvent:event
     span:EKSpanThisEvent error:&saveError];
     if (result == NO){
```

```
        NSLog(@"An error occurred = %@", saveError);
        }
        return result;
    }
```

你可以用我们刚刚实现的方法往用户的日程表中插入新的事件：

```
- (BOOL) application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

/* The event starts from today, right now */
NSDate *startDate = [NSDate date];

/* And the event ends this time tomorrow.
  24 hours, 60 minutes per hour and 60 seconds per minute hence 24 * 60 * 60 */
  NSDate *endDate = [startDate
                     dateByAddingTimeInterval:24 * 60 * 60];

/* Create the new event */
BOOL createdSuccessfully = [self createEventWithTitle:@"My event"
                                    startDate:startDate endDate:endDate
                                        inCalendarWithTitle:@"Calendar"
          inCalendarWithType:EKCalendarTypeLocal notes:nil];

if (createdSuccessfully){
NSLog(@"Successfully created the event.");
} else {
    NSLog(@"Failed to create the event.");
}

self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor]; [self.window makeKeyAndVisible];
return YES;
}
```

14.2.3. 讨论

要想通过编码，在 iOS 设备上创建一个新的事件，我们必须：

1. 分配和初始化一个 EKEventStore 实例

2.找到我们想要将事件保存到的那个日程表（请参考 14.1）.我们必须通过检查目标日程表对象的 allowsContentModifications 属性是否为 YES，来确定该日程表支持编辑。如果它不支持编辑，你就必须选择另一个日程表，或者放弃保存事件。

3.找到目标日程表之后，使用 EKEvent 的 eventWithEventStore:类方法创建一个 EKEvent 类型的事件。

4.设置这个新的事件的属性，比如它的 title, startDate,和 endDate。

5.使用 EKEvent 实例的 calendars 属性，将你的事件和第 2 步找到的日程表关联起来。

6.当你完成事件属性的设置之后，使用 EKEvetnStore 的 saveEvent:span:error: 实例方法将事件加到日程表中。这个方法的返回值（是一个 BOOL 值）表明了这个事件是否成功的加入到了 Calendar 数据库中。如果操作失败，传入到这个方法的 error 参数中的 NSError 对象将包含在插入这个事件时，系统中发生的错误信息。

7.释放你在第 1 步分配的事件仓库（event store）对象。

如果你尝试不指定目标日程表插入一个事件，或者向不可编辑的日程表插入事件，EKEventStore 的 saveEvent:span:error: 实例方法将类似于下面的错误而失败：

Error Domain=EKErrorDomain Code=1 "No calendar has been set." UserInfo=0x15d860 {NSLocalizedDescription=No calendar has been set.}

在 iOS 设备上运行我们的代码，我们将看到在 Canlendar 数据库中创建了一个事件，如图 14-3 所示。

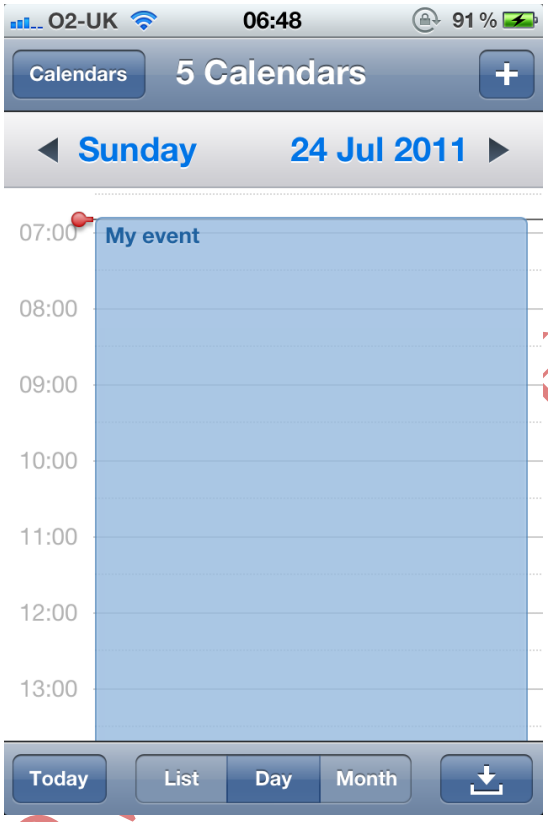

图 14-3. 编码方式添加一个事件到日程表

iOS 会在线对日程表进行同步。这些日程表可能是 Exchange,CalDAV，或者其他常用格式。在 iOS 设备上创建一个 CalDAV 日程表，将在服务器上创建相同的事件。在 Calendar 数据库和服务器进行同步后，对服务器的修改也会反映到 Canlendar 数据库。

14.2.4. 参考

14.1 小节

14.3. 访问日程表的内容

14.3.1. 问题

你想从 iOS 设备上的 EKCalendar 类型的日程表中，获得 EKEvent 类型的事件。

14.3.2. 方案

采用以下步骤：

1.实例化一个 EKEventStore 类型的对象。

2.使用这个事件仓库(event store)的 calendars 属性，找到你想读取的那个日程表。

3.决定你想在日程表中开始搜索以及结束搜索的时间和日期。

4.将日程表对象和第 3 步中确定的两个日期作为参数，传入 EKEventStore 的 predicateForEventsWithStartDate:endDate:calendars:实例方法中。

5.将第 4 步创建的断言（predicate）传入到 EKEventStore 的 eventsMatchingPredicate:实力方法中。这个方法的结果是一个 EKEvent 对象数组，这个数组中的事件都发生在指定日程表中的给定日期之间。

下面的代码说明了这些步骤：

```
- (EKCalendar *) calDAVCalendarWithTitleContaining :(NSString *)paramDescription{

EKCalendar *result = nil;

EKEventStore *eventStore = [[EKEventStore alloc] init];

for (EKCalendar *thisCalendar in eventStore.calendars){ if (thisCalendar.type == EKCalendarTypeCalDAV){
   if ([thisCalendar.title
      rangeOfString:paramDescription].location != NSNotFound){
         return thisCalendar;
            }
      }
}
 return result;
}

- (void) readEvents{

   /* Find a calendar to base our search on */ EKCalendar *targetCalendar =
   [self calDAVCalendarWithTitleContaining:@"gmail.com"];

   /* If we could not find a CalDAV calendar that we were looking for, then we will abort the operation */
   if (targetCalendar == nil){
   NSLog(@"No CalDAV calendars were found."); return;
   }

   /* We have to pass an array of calendars to the event store to search */

   NSArray *targetCalendars = [[NSArray alloc] initWithObjects: targetCalendar, nil];

   /* Instantiate the event store */
   EKEventStore *eventStore = [[EKEventStore alloc] init];

   /* The start date will be today */
     NSDate *startDate = [NSDate date];
```

```
    /* The end date will be 1 day from today */
    NSDate *endDate = [startDate dateByAddingTimeInterval:24 * 60 * 60];
    /* Create the predicate that we can later pass to the event store in order to fetch the events */
      NSPredicate *searchPredicate =
      [eventStore predicateForEventsWithStartDate:startDate
                                endDate:endDate calendars:targetCalendars];

    /* Make sure we succeeded in creating the predicate */ if (searchPredicate == nil){
    NSLog(@"Could not create the search predicate.");
    return;
      }

    /* Fetch all the events that fall between the starting and the ending dates */
    NSArray *events = [eventStore eventsMatchingPredicate:searchPredicate];

      /* Go through all the events and print their information out to the console */
      if (events != nil){

          NSUInteger counter = 1;
          for (EKEvent *event in events){

              NSLog(@"Event %lu Start Date = %@", (unsigned long)counter,
                                                    event.startDate);
              NSLog(@"Event %lu End Date = %@", (unsigned long)counter,
                                                    event.endDate);
              NSLog(@"Event %lu Title = %@",
                (unsigned long)counter, event.title);

          counter++;
          }

       } else {
         NSLog(@"The array of events for this start/end time is nil.");
      }
  }
  - (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    [self readEvents];
  self.window = [[UIWindow alloc] initWithFrame:
                    [[UIScreen mainScreen] bounds]];
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];
  return YES;
  }
```

如果我们的 iOS 设备设置有 6 个日程表（其中有一个是 Google CalDAV 日程表），当我们在这个设备上运行以上代码时，我们就能得到运行的当天以及第二天之间的事件了，如图 14-2 所示。

iOS 设备上的 Calendar 应用以图 14-4 中的格式显示 s 事件。请记住，除非你创建了 Google Calendar 事件，就像我做的那样，时间和日期也要完全一样，否则你看不到类似的结果。但是，如果你决定在其他的日程表中创建事件，比如本地日程表，日期设定也不一样，你就需要确保修改了 event 的断言中的起止日期，并且修改你要进行搜索的日程表。更多信息，请参考本节的“讨论”部分。

图 14-4. iOS 设备上的 Calendar 应用

14.3.3. 讨论

本章的概述中曾提到，iOS 设备可以使用不同类型的日程表进行配置，如 CalDAV,Exchange 等等。每个可被 Event Kit 框架访问的日程表，都是以 EKCalendar 对象包装的，这个对象可以被 EKEventStore 实例的 calendars 数组属性访问。你能用不同的方式从日程表中获得事件，但是最简单的方法是，在事件仓库内，创建和执行一个日期和时间的特定格式说明，叫做断言（predicate）。

使用 EKEventStore 的 predicateForEventsWithStartDate:endDate:calendars:实例方法，可以创建一个使用在 Event Kit 框架中的 NSPredicate 类型的断言。这个方法的参数是：

predicateForEventsWithStartDate

要获取的事件的开始日期和时间

endDate

要获取的事件的截止日期和时间

calendars

要在开始和截止日期之间搜索事件的那个日程表数组

14.3.4. 参考

14.1 小节

14.4. 从日程表删除事件

14.4.1. 问题

你想要能从用户的日程表中删除一个指定的事件或者一组事件。

14.4.2. 方案

使用 EKEventStore 的 removeEvent:span:commit:error:实例方法。

14.4.3. 讨论

EKEventStore 的 removeEvent:span:commit:error:实例方法可以删除一个事件的实例，或者一个重复事件的所有实例。关于重复事件的更多信息，参见 14.5 小节。在本小节，我们只删除日程表中事件的一个实例，而不删除同一事件的其他实例。

这个方法接受的参数为：

removeEvent
这是要从日程表中删除的 EKEvetn 实例

span
这个参数告诉事件仓库，在日程表中，我们是想只删除这一个事件呢，还是它的所有实例。如果只删除当前的事件，就为 removeEvent 参数指定 EKSpanThisEvent。如果想从日程表中删除同一事件的所有实例，就要为 removeEvent 参数传入 EKSpanFutureEvents。

commit
一个布尔值，告诉事件仓库，修改是否必须立即保存到远程/本地日程表。

error
这个参数在方法返回 NO 时，向被传入的 NSError 对象的引用中写入错误信息。

为了演示，让我们使用 14.2 节中实现的事件创建方法。我们可以做的是，在我们的 Google CalDAV 日程表中创建完一个事件之后，再尝试将他从事件仓库中删除：

```
- (BOOL) createEventWithTitle:(NSString *)paramTitle
                    startDate:(NSDate *)paramStartDate
                      endDate:(NSDate *)paramEndDate
          inCalendarWithTitle:(NSString *)paramCalendarTitle
          inCalendarWithType:(EKCalendarType)paramCalendarType
```

```
                                notes:(NSString *)paramNotes{

BOOL result = NO;

EKEventStore *eventStore = [[EKEventStore alloc] init];

/* Are there any calendars available to the event store? */
 if ([eventStore.calendars count] == 0){
   NSLog(@"No calendars are found.");
   return NO;
   }

EKCalendar *targetCalendar = nil;

/* Try to find the calendar that the user asked for */
for (EKCalendar *thisCalendar in eventStore.calendars){
   if ([thisCalendar.title isEqualToString:paramCalendarTitle] &&
        thisCalendar.type == paramCalendarType){
      targetCalendar = thisCalendar;
      break;
       }
}

/* Make sure we found the calendar that we were asked to find */ if (targetCalendar == nil){
NSLog(@"Could not find the requested calendar.");
return NO; }

/* If a calendar does not allow modification of its contents then we cannot insert an event into it */
  if (targetCalendar.allowsContentModifications == NO){
  NSLog(@"The selected calendar does not allow modifications.");
   return NO;
}

/* Create an event */
EKEvent *event = [EKEvent eventWithEventStore:eventStore];
event.calendar = targetCalendar;
/* Set the properties of the event such as its title,
     start date/time, end date/time, etc. */ event.title = paramTitle;
   event.notes = paramNotes;
   event.startDate = paramStartDate;
   event.endDate = paramEndDate;

   /* Finally, save the event into the calendar */
   NSError *saveError = nil;

   result = [eventStore saveEvent:event
                                span:EKSpanThisEvent
                                error:&saveError];
     if (result == NO){
        NSLog(@"An error occurred = %@", saveError);
     }
     Return result;
       }

 - (BOOL) removeEventWithTitle:(NSString *)paramTitle
                          startDate:(NSDate *)paramStartDate
                            endDate:(NSDate *)paramEndDate
                 inCalendarWithTitle:(NSString *)paramCalendarTitle
```

```
                    inCalendarWithType:(EKCalendarType)paramCalendarType
                                 notes:(NSString *)paramNotes{

  BOOL result = NO;

  EKEventStore *eventStore = [[EKEventStore alloc] init];
/* Are there any calendars available to the event store? */
   if ([eventStore.calendars count] == 0){
   NSLog(@"No calendars are found.");
   return NO;
}

  EKCalendar *targetCalendar = nil;

/* Try to find the calendar that the user asked for */
   for (EKCalendar *thisCalendar in eventStore.calendars){
      if ([thisCalendar.title isEqualToString:paramCalendarTitle] &&
           thisCalendar.type == paramCalendarType){
         targetCalendar = thisCalendar;
         break;
         }
   }

   /* Make sure we found the calendar that we were asked to find */
      if (targetCalendar == nil){
         NSLog(@"Could not find the requested calendar.");
         return NO; }

   /* If a calendar does not allow modification of its contents then we cannot insert an event into it */
      if (targetCalendar.allowsContentModifications == NO){
        NSLog(@"The selected calendar does not allow modifications.");
         return NO;
     }

  NSArray *calendars = [[NSArray alloc] initWithObjects:targetCalendar, nil];
NSPredicate *predicate =
                          [eventStore predicateForEventsWithStartDate:paramStartDate
                           endDate:paramEndDate calendars:calendars];

  /* Get all the events that match the parameters */
  NSArray *events = [eventStore eventsMatchingPredicate:predicate];
     if ([events count] > 0){

  /* Delete them all */
  for (EKEvent *event in events){ NSError *removeError = nil;

  /* Do not commit here, we will commit in batch after we have removed all the events that matched our criteria */
    if ([eventStore removeEvent:event span:EKSpanThisEvent
                         commit:NO error:&removeError] == NO){
        NSLog(@"Failed to remove event %@ with error = %@", event,
                                                          removeError);
                                                           }
     }

    NSError *commitError = nil;
    if ([eventStore commit:&commitError]){
                        result = YES;
                        } else {
    NSLog(@"Failed to commit the event store."); }
```

```
    } else {
    NSLog(@"No events matched your input.");
      }
    return result;
    }

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  NSDate *startDate = [NSDate date]; /* Now */

  const NSTimeInterval NSOneHour = 60 * 60; /* 60 minutes, each 60 seconds */
  NSDate *endDate = [startDate dateByAddingTimeInterval:NSOneHour];

  BOOL createdSuccessfully = [self createEventWithTitle:@"Shopping"
                                               startDate:startDate
                                                 endDate:endDate
                                    inCalendarWithTitle:@"vandad.np@gmail.com"
                                    inCalendarWithType:EKCalendarTypeCalDAV
                                                 notes:@"Get bread"];

    if (createdSuccessfully){
    NSLog(@"Successfully created the event.");

     BOOL removedSuccessfully =
      [self removeEventWithTitle:@"Shopping"
                       startDate:startDate
                       endDate:endDate
              inCalendarWithTitle:@"vandad.np@gmail.com"
              inCalendarWithType:EKCalendarTypeCalDAV
                           notes:@"Get bread"];

          if (removedSuccessfully){
            NSLog(@"Successfully removed the event.");
            } else {
            NSLog(@"Failed to remove the event.");
            }
            } else {
              NSLog(@"Failed to create the event.");
              }

          self.window = [[UIWindow alloc] initWithFrame:
                         [[UIScreen mainScreen] bounds]];
          self.window.backgroundColor = [UIColor whiteColor];
            [self.window makeKeyAndVisible];
          return YES;
      }
```

在本例中，我们没有将每个事件的删除行为逐个提交。我们对于 removeEvent:span:commit:error: 方法的 commit 参数，只是简单的设置为 NO，而在我们完成之后，再对于事件仓库的 commit:方法进行显式的调用。原因是，我们不想在每次删除之后就进行一次提交。那会成为性能负担。我们可以一直继续删除我们需要删除的事件，然后一次性的提交这些操作。

14.4.4. 参考

14.1 小节；14.3 小节

14.5. 向日程表添加重复事件

14.5.1. 问题

你想向日程表添加一个重复事件

14.5.2. 方案

下面是创建重复发生事件的步骤。本例中，我们创建一个事件，它发生在一整年中每个月份的同一天：

1. 创建一个 EKEventStore 的实例
2. 在事件仓库的 calendars 数组中找到一个可编辑的日程表（更多信息，参见 14.1）。
3. 创建一个 EKEvent 类型的对象（更多信息，参见 14.2）。
4. 为事件设置适当的值，比如 startDate 和 endDate（更多信息，参见 14.2）。
5. 实例化一个 NSDate 对象，包含事件重复结束的确切日期，这个日期是从今天开始的一年之后。
6. 使用 EKRecurrenceEnd 的 recurrenceEndWithEndDate:类方法，传入你在第 5 步创建的 NSDate，来创建一个 EKRecurrenceEnd 类型的对象。
7. 使用 EKRecurrenceRule 类的 initRecurrenceWithFrequency:interval:end: 方法分配和初始化一个 EKRecurrenceRule 类型的对象，将第 6 步创建的重复结束日期作为 end 参数传入。更多关于这个方法的信息，请参照本节的“讨论”部分。
8. 将第 7 步创建的重复事件，对第 3 步创建的 EKEvent 对象的 recurringRule 属性赋值。
9. 对这个事件（在第 3 步创建）调用 saveEvent:span:error:实例方法，将其作为 saveEvent 参数，并将 EKSpanFutureEvents 赋值给 span 参数。这会为我们创建重复事件。

下面的代码实现了上述步骤：

```
- (void) createRecurringEventInLocalCalendar{

    /* Step 1: And now the event store */
    EKEventStore *eventStore = [[EKEventStore alloc] init];

   /* Step 2: Find the first local calendar that is modifiable */ EKCalendar *targetCalendar = nil;

    for (EKCalendar *thisCalendar in eventStore.calendars){
    if (thisCalendar.type == EKCalendarTypeLocal &&
       [thisCalendar allowsContentModifications]){
     targetCalendar = thisCalendar;
          }
    }
```

```
  /* The target calendar wasn't found? */
  if (targetCalendar == nil){
       NSLog(@"The target calendar is nil.");
        return;
    }

    /* Step 3: Create an event */
    EKEvent *event = [EKEvent eventWithEventStore:eventStore];

  /* Step 4: Create an event that happens today and happens
                every month for a year from now */

    NSDate *eventStartDate = [NSDate date];

  /* Step 5: The event's end date is one hour from the moment it is created */
  NSTimeInterval NSOneHour = 1 * 60 * 60;
  NSDate *eventEndDate = [eventStartDate dateByAddingTimeInterval:NSOneHour];

    /* Assign the required properties, especially
    the target calendar */
    event.calendar = targetCalendar;
    event.title = @"My Event";
    event.startDate = eventStartDate;
    event.endDate = eventEndDate;

    /* The end date of the recurring rule is one year from now */
      NSTimeInterval NSOneYear = 365 * 24 * 60 * 60;
      NSDate *oneYearFromNow = [eventStartDate dateByAddingTimeInterval:NSOneYear];

    /* Step 6: Create an Event Kit date from this date */ EKRecurrenceEnd *recurringEnd =
    [EKRecurrenceEnd recurrenceEndWithEndDate:oneYearFromNow];

    /* Step 7: And the recurring rule. This event happens every
    month (EKRecurrenceFrequencyMonthly), once a month (interval:1)
    and the recurring rule ends a year from now (end:RecurringEnd) */

    EKRecurrenceRule *recurringRule =
      [[EKRecurrenceRule alloc]
        initRecurrenceWithFrequency:EKRecurrenceFrequencyMonthly
          interval:1
        end:recurringEnd];

  /* Step 8: Set the recurring rule for the event */
  event.recurrenceRules = [[NSArray alloc] initWithObjects:recurringRule, nil];

    NSError *saveError = nil;

    /* Step 9: Save the event */
    if ([eventStore saveEvent:event
                         span:EKSpanFutureEvents
                        error:&saveError]){
    NSLog(@"Successfully created the recurring event."); } else {
    NSLog(@"Failed to create the recurring event %@", saveError);
    }
}
```

14.5.3. 讨论

一个重复事件是发生多次的事件。我们可以像创建普通事件一样创建重复事件。参考 14.2 节了解如何向 Canlendar 数据库中插入普通事件。重复事件和普通事件之间的唯一区别是，重复事件应用了重复规则。重复规则告诉 Event Kit 框架，这个事件在将来如何发生。

通过调用 initRecurrenceWithFrequency:interval:end: 方法初始化一个 EKRecurrenceRule 对象，我们创建一个重复规则，此方法的参数为：

initRecurrenceWithFrequency

指定你希望事件重复发生的频率，是每天 (EKRecurrenceFre quencyDaily)，每周 (EKRecurrenceFrequencyWeekly)，每月 (EKRecurrenceFre quencyMonthly)，或是每年 (EKRecurrenceFrequencyYearly)。

interval

一个大于 0 的值，用于指定每次发生的开始到结束之间的时间间隔。举个例子，如果你想创建一个每周重复的事件，可以指定 EKRecurrenceFrequencyWeekly 值，使用 1 作为 interval 参数。如果你想要这个事件每隔一周发生一次，则可以指定 EKRecurrenceFrequencyWeekly 值，使用 2 作为 interval 参数。

end

一个 EKRecurrenceEnd 类型的日期，指定了在特定日程表中，重复事件结束的日期。这个参数和事件结束日期（EKevent 的 endDate 属性）不一样。事件结束日期指定了在日程表中特定事件于何时结束，而 initRecurrenceWithFrequency:interval:end: 方法中的 end 参数指定了在数据库中该事件最后一次发生的时间。

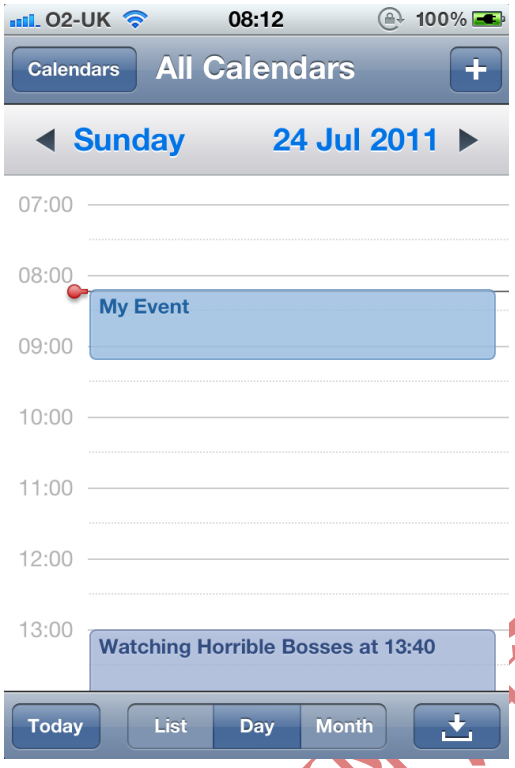

图 14-5 描绘了在设备上的 Calendar 应用中，重复事件如何发生。

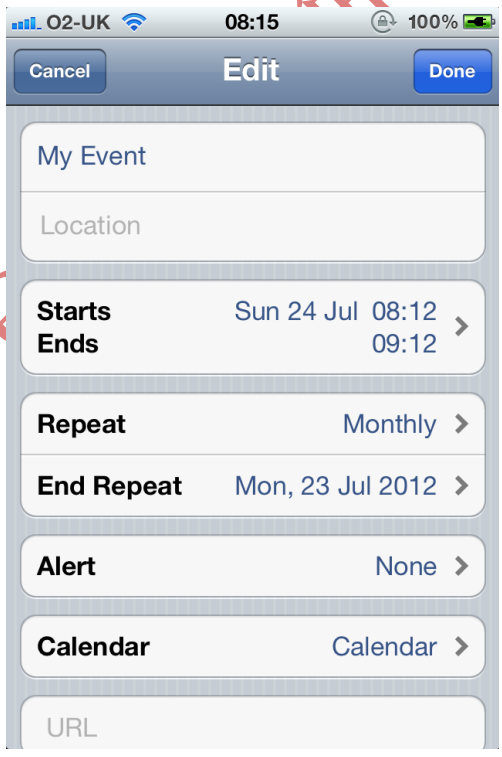

图 14-5. Calendar 应用显式了我们创建的名为 My Event 的重复事件。

通过在 iOS 设备上的 Calendar 应用上编辑这个事件（参见图 14-6），你可以看到，这个事件真的是一个每月发生的重复事件，在我们创建事件的同一天发生，为期一年。

图 14-6. 在 iOS 设备上的 Canlendar 应用中编辑一个重复事件。

14.5.4. 参考

14.2 小节

14.6. 获取一个事件的参与者

14.6.1. 问题

你想获取指定事件的参与者列表。

14.6.2. 方案

使用 EKEvent 的 attendendees 属性。这个属性是一个 NSArray 类型，并包含 EKParticipant 的对象。

下面的示例代码将获取今天（不管是哪一天）发生的所有事件，并将包括事件的参与者在内的有用的事件信息输出到窗口：

```
- (EKCalendar *) calDAVCalendarWithTitleContaining
                    :(NSString *)paramDescription{

  EKCalendar *result = nil;

  EKEventStore *eventStore = [[EKEventStore alloc] init];

  for (EKCalendar *thisCalendar in eventStore.calendars){
   if (thisCalendar.type == EKCalendarTypeCalDAV){
      if ([thisCalendar.title
        rangeOfString:paramDescription].location != NSNotFound){
       return thisCalendar;
              }
          }
     }
      return result;
}

 - (void) enumerateTodayEvents{
 /* Find a calendar to base our search on */
 EKCalendar *targetCalendar =
  [self calDAVCalendarWithTitleContaining:@"vandad.np@gmail.com"];

  /* If we could not find a CalDAV calendar that
     we were looking for, then we will abort the operation */
   if (targetCalendar == nil){
      NSLog(@"No CalDAV calendars were found."); return;
      }
```

```
  /* We have to pass an array of calendars to the event store to search */
  NSArray *targetCalendars = [[NSArray alloc] initWithObjects:targetCalendar, nil];
/* Instantiate the event store */
  EKEventStore *eventStore = [[EKEventStore alloc] init];

  /* Construct the starting date for today */
    NSDate *startDate = [NSDate date];

  /* The end date will be 1a day from now */
  NSTimeInterval NSOneDay = 1 * 24 * 60 * 60;
NSDate *endDate = [startDate dateByAddingTimeInterval:NSOneDay];

/* Create the predicate that we can later pass to the event store in order to fetch the events */
NSPredicate *searchPredicate =
[eventStore predicateForEventsWithStartDate:startDate
                                    endDate:endDate calendars:
                                    targetCalendars];

/* Make sure we succeeded in creating the predicate */
  if (searchPredicate == nil){
  NSLog(@"Could not create the search predicate.");
    return;
      }

  /* Fetch all the events that fall between the starting and the ending dates */
  NSArray *events = [eventStore eventsMatchingPredicate:searchPredicate];

  /* Array of NSString equivalents of the values in the EKParticipantRole enumeration */
  NSArray *attendeeRole = [NSArray arrayWithObjects:
@"Unknown", @"Required",
@"Optional", @"Chair",
@"Non Participant", nil];

/* Array of NSString equivalents of the values in the EKParticipantStatus enumeration */
NSArray *attendeeStatus = [NSArray arrayWithObjects: @"Unknown",
@"Pending",
@"Accepted",
@"Declined",
@"Tentative",
@"Delegated",
@"Completed",
@"In Process", nil];

/* Array of NSString equivalents of the values in the EKParticipantType enumeration */
NSArray *attendeeType = [NSArray arrayWithObjects: @"Unknown",
@"Person",
@"Room",
@"Resource",
@"Group", nil];

/* Go through all the events and print their information out to the console */
if (events != nil){
NSUInteger eventCounter = 0;
for (EKEvent *thisEvent in events){
      eventCounter++;

NSLog(@"Event %lu Start Date = %@", (unsigned long)eventCounter, thisEvent.startDate);
NSLog(@"Event %lu End Date = %@", (unsigned long)eventCounter, thisEvent.endDate);
```

```
    NSLog(@"Event %lu Title = %@", (unsigned long)eventCounter, thisEvent.title);
    if (thisEvent.attendees == nil || [thisEvent.attendees count] == 0){
      NSLog(@"Event %lu has no attendees", (unsigned long)eventCounter);
      continue; }

      NSUInteger attendeeCounter = 1;
      for (EKParticipant *participant in thisEvent.attendees){
      NSLog(@"Event %lu Attendee %lu Name = %@", (unsigned long)eventCounter,
      (unsigned long)attendeeCounter, participant.name);
      NSLog(@"Event %lu Attendee %lu Role = %
              @", (unsigned long)eventCounter, (unsigned long)attendeeCounter, [attendeeRole objectAtIndex:
                  participant.participantRole]);
      NSLog(@"Event %lu Attendee %lu Status = %@", (unsigned long)eventCounter,
      (unsigned long)attendeeCounter, [attendeeStatus objectAtIndex:
      participant.participantStatus])

      NSLog(@"Event %lu Attendee %lu Type = %@",
      (unsigned long)eventCounter,
      (unsigned long)attendeeCounter,
        [attendeeType objectAtIndex:
    participant.participantType]);

    NSLog(@"Event %lu Attendee %lu URL = %@",
       (unsigned long)eventCounter,
       (unsigned long)attendeeCounter,
       participant.URL);
          attendeeCounter++;
       }

       } /* for (EKEvent *Event in Events){ */
    } else {
      NSLog(@"The array of events is nil.");
    }
}
 - (BOOL) application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    [self enumerateTodayEvents];

    self.window = [[UIWindow alloc] initWithFrame:
                        [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
      [self.window makeKeyAndVisible];
    return YES;
}
```

我们在 iOS 设备上设置了一个名为 vandad.np@gmail.com 的 CalDAV 日程表，当在这个设备上运行上面的代码（关于 CalDAV 日程表以及如何在 iOS 设备上设置一个 CalDAV 日程表的更多信息，参见本章概述），我们在控制台窗口得到了类似的结果。

14.6.3. 讨论

不同类型的日程表，比如 CalDAV，可以在事件中包含参与者。虽然 iOS 不允许向设备的日程表中添加参与者，但它允许用户在服务器上向日程表添加参与者。比如，你可以使用 Google Calendar 这么做。

一旦用户为一个事件添加了参与者，你口能使用 EKEvent 的 attendees 属性，访问 EKParticipant 类型的参与者对象了。每个参与者都有如下属性：

name
这是参与者的名字。如果你只为添加到事件的人指定了 EMail 地址，那么这个字段将是这个 EMail 地址。

URL
这通常是参与者的"mailto"URL。

participantRole
这是事件中参与者的角色。可以从 EKParticipantRole 枚举中选取不同的值应用到本属性。

participantStatus
这个参数告诉我们，参与者是否接受了事件请求。这个属性还可以取其他的值，这些值都在 EKParticipantStatus 枚举中指定。

participantType
这个参数是 EKParticipantType 类型，EKParticipantType 是一个枚举，他的名字表明，它指定了参与者的类型，比如是组(EKParticipantTypeGroup)还是个人(EKParticipantTypePerson)。

14.6.4. 参考

14.2 小节；14.3 小节

14.7. 为日程表添加闹铃

14.7.1. 问题

你想在日程表中为事件添加闹铃。

14.7.2. 方案

使用 EKAlarm 类的 alarmWithRelativeOffset:类方法，可创建一个 EKAlarm 实例。使用 EKEvent 类的 addAlarm:实例方法，将闹铃添加到事件中，如下：

```
- (EKCalendar *) getFirstModifiableLocalCalendar{

EKCalendar *result = nil;

EKEventStore *eventStore = [[EKEventStore alloc] init];
```

```
for (EKCalendar *thisCalendar in eventStore.calendars){
   if (thisCalendar.type == EKCalendarTypeLocal &&
         [thisCalendar allowsContentModifications]){ return thisCalendar;
        }
     }
      return result;

      }

- (void) addAlarmToCalendar{

  EKCalendar *targetCalendar =
   [self getFirstModifiableLocalCalendar];
    If (targetCalendar == nil){
    NSLog(@"Could not find the target calendar."); return;
    }

  EKEventStore *eventStore = [[EKEventStore alloc] init];

  /* The event starts 60 seconds from now */
  NSDate *startDate = [NSDate dateWithTimeIntervalSinceNow:60.0f];

  /* And end the event 20 seconds after its start date */
  NSDate *endDate = [startDate dateByAddingTimeInterval:20.0f];

  EKEvent *eventWithAlarm = [EKEvent eventWithEventStore:eventStore];
   eventWithAlarm.calendar = targetCalendar;
   eventWithAlarm.startDate = startDate;
   eventWithAlarm.endDate = endDate;

/* The alarm goes off 2 seconds before the event happens */
     EKAlarm *alarm = [EKAlarm alarmWithRelativeOffset:-2.0f];
eventWithAlarm.title = @"Event with Alarm";
                          [eventWithAlarm addAlarm:alarm];
  NSError *saveError = nil;
   if ([eventStore saveEvent:eventWithAlarm
                       span:EKSpanThisEvent
                       error:&saveError]){
    NSLog(@"Saved an event that fires 60 seconds from now."); } else {
    NSLog(@"Failed to save the event. Error = %@", saveError);
         }
    }


  - (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  [self addAlarmToCalendar];
   self.window = [[UIWindow alloc] initWithFrame:
                  [[UIScreen mainScreen] bounds]];
   self.window.backgroundColor = [UIColor whiteColor];
                                  [self.window makeKeyAndVisible];
   return YES;
}
```

14.7.3. 讨论

一个 EKEvent 类型的事件可以有多个闹铃。只要使用 EKAlarm 的

alarmWithAbsoluteDate: 或者 alarmWithRelativeOffset: 类方法，就可以创建闹铃。前一个方法需要一个绝对的日期和时间（你可以使用 CFAbsoluteTimeGet Current 函数来得到当前的绝对时间），而后一个方法需要知道，闹铃触发的时间相对于事件发生日期之间的秒数。例如，如果事件安排到今天早上 6:00 开始，我们就创建一个闹铃并设置相对偏移为-60（以秒作为单位），那么我们的闹铃将在同一天的 5：59 触发。偏移量只能设置 0 或者负值。正值将被 iOS 自动改为 0。闹铃一旦被触发，iOS 会将这个闹铃显示给用户，如图 14-7 所示。

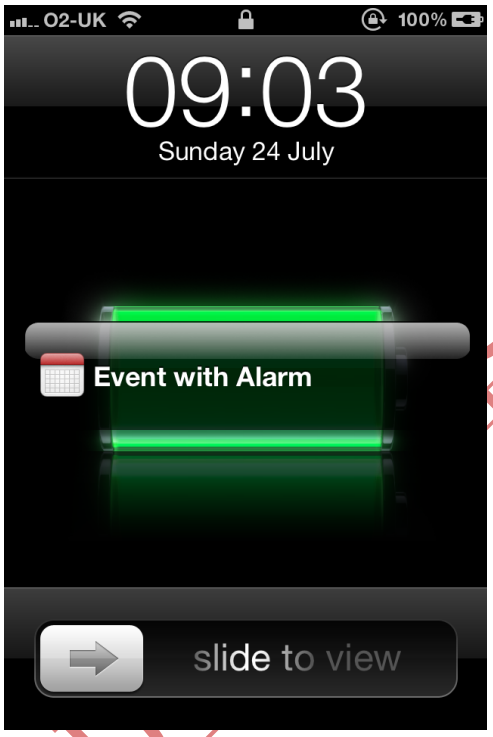

图 14-7. 当闹铃被触发时，iOS 在屏幕上显示一个警告。

你可以使用 EKEent 的 removeAlarm: 实例方法将一个闹铃从相关的事件实例上移除。

14.7.4. 参考

14.1 小节

14.8. 处理事件变化通知

14.8.1. 问题

你想在用户修改了 Calendar 数据库的时候，在你的应用程序中得到通知。

14.8.2. 方案

注册 EKEventStoreChangedNotification 通知：

```
- (EKCalendar *)
   calDAVCalendarWithTitleContaining:(NSString *)paramDescription
                             inEventStore:(EKEventStore *)paramEventStore{
       EKCalendar *result = nil;

   for (EKCalendar *thisCalendar in paramEventStore.calendars){
      if (thisCalendar.type == EKCalendarTypeCalDAV){
       if ([thisCalendar.title
          rangeOfString:paramDescription].location != NSNotFound){
           return thisCalendar;
               }
          }
     }
  return result;

  }

  - (void) eventsChanged:(NSNotification *)paramNotification{

   NSMutableArray *invalidatedEvents = [[NSMutableArray alloc] init];

   NSLog(@"Refreshing array of events...");

  for (EKEvent *event in self.eventsForOneYear){
       if ([event refresh] == NO){
          [invalidatedEvents addObject:event];
          }
     }
     if ([invalidatedEvents count] > 0){
     [self.eventsForOneYear removeObjectsInArray:invalidatedEvents];
     }
  }

- (BOOL) application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

   self.eventStore = [[EKEventStore alloc] init];

  EKCalendar *calendar =
   [self calDAVCalendarWithTitleContaining:@"vandad.np@gmail.com"
   inEventStore:self.eventStore]; NSTimeInterval NSOneYear = 1 * 365 * 24 * 60 * 60;
   NSDate *startDate = [NSDate date];

   NSDate *endDate = [startDate dateByAddingTimeInterval:NSOneYear];
   NSArray *calendars = [[NSArray alloc] initWithObjects:calendar, nil];
   NSPredicate *predicate =
   [self.eventStore predicateForEventsWithStartDate:startDate
                                             endDate:endDate
                                           calendars:calendars];

   NSArray *events = [self.eventStore eventsMatchingPredicate:predicate];

   self.eventsForOneYear = [[NSMutableArray alloc] initWithArray:events];
```

```
    [[NSNotificationCenter defaultCenter]
    addObserver:self
    selector:@selector(eventsChanged:)
    name:EKEventStoreChangedNotification object:nil];
    self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

14.8.3. 讨论

在 iOS 上多线程是可能的。想像你从 EKEventStore 得到一组事件，将他们存入数组，你允许你的用户操作他们（编辑、添加、或删除）。用户可从你的应用程序中切换到 Calendar 应用程序，然后删除她想从你的程序中删除的相同事件。这样的一系列活动将产生一个 EKEventStoreChanged Notification 通知，你可以选择接收它。

即使你的程序在前台，EKEventStoreChangedNotification 通知还是会被发给它（仅当你订购了这个通知时）。因此，根据你的程序是在后台还是前台，你必须区别的对待这个通知。下面是一些需要考虑的问题：

1. 如果你的程序在前台时，你收到了 EKEventStoreChangedNotification 通知，最好能实现一种机制，来确定对于事件仓库的修改是来源于你自己的程序还是程序外的某个地方。如果他们来自程序外部，你必须确信保留了仓库中事件的最新版本，而不是旧的版本。如果因为任何原因，你从事件仓库中拷贝了一个事件并在某处保存了这个拷贝，你就必须调用 EKEvent 类型事件的 refresh 实例事件。如果此方法返回结果为 YES，你就可以在内存中保留此对象，如果返回值为 NO，你就必须处置该对象，因为在你的程序之外有人将他删除了，或者因为某种原因事件失效了。

2.如果你的程序在后台，这时你收到了 EKEventStoreChangedNotification 通知，根据 APPLE 的文档，你的程序不应该尝试做任何 GUI 相关的处理，实际上，应该尽可能不做处理。那样的情况下，你必须避免对程序的 GUI 添加新画面，或者做出任何修改。

3. 如果你的程序在后台得到 EKEventStoreChangedNotification 通知，你必须在程序中将它记录下来（可能保存在一个 BOOL 类型的属性中），当程序回到前台之后再做处理。一般来说，如果你在后台收到了关于事件变化的通知，那么在程序返回前台时，你应该获得所有保存在程序中的事件。

EKEventStoreChangedNotification 事件仓库通知不支持联合（Coalescing）。也就是说，你可以在 Calendar 数据库中，获得同一个事件变化的多次通知。何时重新获取你保留的事件，由你来决定。

14.9. 显示事件视图控制器

14.9.1. 问题

你想使用 iOS 内建的视图控制器来显示 Calendar 数据库中的某个事件的属性。

14.9.2. 方案

创建一个 EKEventViewController 类的实例，将其推入到导航控制器，或者在另一个视图控制器之上，作为模态视图控制器展现。

14.9.3. 讨论

iOS 设备的用户已经熟悉了 Calendar 应用的界面。当他们选择一个事件，他们可以看到事件的属性并可能被允许编辑此事件。为了使用 iOS 内建的 SDK 事件视图控制器，来展示一个视图，我们可以创建一个 EKEventViewController 实例，并为其事件属性分配一个 EKEvent 类型的事件。然后，我们可以将事件视图控制器推入到导航控制器，并让 iOS 接收处理。

我们想在 iOS 设备上的任何可用的日程表上查找一个事件（或任何事件），从一年前直到现在。我们将使用 EKEventViewController 来将该事件展示给用户。下面是我们的视图控制器的.h 文件：

```
#import <UIKit/UIKit.h>
#import <EventKit/EventKit.h> #import <EventKitUI/EventKitUI.h>

@interface Presenting_Event_View_ControllersViewController
        : UIViewController <EKEventViewDelegate>

@property (nonatomic, strong) EKEventStore *eventStore;

@end
```

我们应该在我们的视图控制器的实现中，同步 eventStore 属性：

```
#import "Presenting_Event_View_ControllersViewController.h"

@implementation Presenting_Event_View_ControllersViewController

@synthesize eventStore;
...
```

现在我们继续编写视图控制器的 viewDidLoad 方法，从一年以前到现在，我们在设备上的日程表中找到的第一个事件时，显示 EKEventViewController 实例：

```
- (void)viewDidLoad{
  [super viewDidLoad];
```

```
self.eventStore = [[EKEventStore alloc] init];
NSTimeInterval NSOneYear = 1 * 365 * 24.0f * 60.0f * 60.0f;
NSDate *startDate = [[NSDate date] dateByAddingTimeInterval:-NSOneYear];
NSDate *endDate = [NSDate date];

NSPredicate *predicate =
[self.eventStore predicateForEventsWithStartDate:startDate
                                         endDate:endDate
                                       calendars:self.eventStore.calendars];

NSArray *events = [self.eventStore eventsMatchingPredicate:predicate];

if ([events count] > 0){
      EKEvent *event = [events objectAtIndex:0];
      EKEventViewController *controller = [[EKEventViewController alloc] init];
      controller.event = event;
      controller.allowsEditing = NO;
      controller.allowsCalendarPreview = YES;
      controller.delegate = self;

      [self.navigationController pushViewController:controller
       animated:YES];
       }
   }
- (void)viewDidUnload{ [super viewDidUnload];
self.eventStore = nil;
}
```

最后，但不是最不重要的一点是，如你在代码中所见，我们成为了事件视图控制器的委托对象，所以让我们确定在需要时处理了委托方法：

```
- (void)eventViewController:(EKEventViewController *)controller
         didCompleteWithAction:(EKEventViewAction)action{

     switch (action){

        case EKEventViewActionDeleted:{
           NSLog(@"User deleted the event.");
           break;
         }
           case EKEventViewActionDone:{
                NSLog(@"User finished viewing the event.");
                  break;
           }
           case EKEventViewActionResponded:{
              NSLog(@"User responsed to the invitation in the event.");
              break;
              }
      }
}
```

一旦我们在 iOS 设备上运行这个程序，我们可以看到内建的视图控制器显示了我们找到的事件内容（参见图 14-8）。

图 14-8. iOS 内建的事件视图控制器

下面是我们可以用于修改视图控制器实例行为的不同属性：

allowsEditing

如果这个属性为 YES，事件视图控制器的导航控制器上会出现 Edit 按钮，允许用户编辑该事件。这种情况只会在可编辑的日程表上发生，并且仅限于用户在设备上创建的事件。例如，你在 WEB 上使用 Google Calendar 创建了一个事件，之后这个事件出现在你的 iOS 设备上，此时你是不能编辑的。

allowsCalendarPreview

如果这个属性被设置为 YES，并且用户在看的事件是一个邀请，用户将可以选择在日程表上查看这个事件，这个日程表上还有被安排到同一天的其他事件。

event

这个属性必须在事件视图控制器展示之前设置。它将成为事件视图控制器显示给用户的那个事件。

当你推入一个事件视图控制器，Back 按钮将默认显示“Back”标题，所以你不需要手动改变它。但是，如果你决定改变 Back 按钮，你可以通过为导航物件（navigation item）的 backBarButtonItem 属性设置一个新的 UIBarButtonItem 对象来实现。在我们的示例代码中，在将事件视图控制器推入之前，我们可以编辑 pushController:方法，来给我们的视图控制器一个自定义额 Back 按钮。

```
- (void)viewDidLoad{
  [super viewDidLoad];

    self.eventStore = [[EKEventStore alloc] init];

     NSTimeInterval NSOneYear = 1 * 365 * 24.0f * 60.0f * 60.0f;
```

```
NSDate *startDate = [[NSDate date] dateByAddingTimeInterval:-NSOneYear];
NSDate *endDate = [NSDate date];

NSPredicate *predicate =
[self.eventStore predicateForEventsWithStartDate:startDate
                                         EndDate:endDate
                                         calendars:self.eventStore.calendars];

NSArray *events = [self.eventStore eventsMatchingPredicate:predicate];

if ([events count] > 0){
   EKEvent *event = [events objectAtIndex:0];
   EKEventViewController *controller = [[EKEventViewController alloc] init];
   controller.event = event;
   controller.allowsEditing = YES;
   controller.allowsCalendarPreview = YES;
   controller.delegate = self;

   self.navigationItem.backBarButtonItem =
   [[UIBarButtonItem alloc] initWithTitle:@"Go Back"
                                    style:UIBarButtonItemStylePlain
                                   target:nil action:nil];

    [self.navigationController pushViewController:controller
                                         animated:YES];
   }
}
```

这个修改的结果显示在图 14-9 中（请记住在这个例子中，事件视图控制器是可编辑的）。

图 14-9. 一个可编辑的视图控制器，带有可编辑状态和一个自定义的返回按钮

14.9.4. 参考

14.10 小节

14.10. 显示事件编辑视图控制器

14.10.1. 问题.

你想使用 SDK 内建的视图控制器，让用户从程序内的 Calendar 数据库中编辑（插入、删除和编辑）事件。

14.10.2. 方案

实例化一个 EKEventEditViewController 类型的对象，再使用 UINavigationControler 的 presentModalViewController:animated:实例方法将其显示于导航控制器上。

14.10.3. 讨论

EKEventEditViewController 类的实例允许我们向用户展示一个事件编辑控制器。这个视图控制器，根据我们对它的设置，可以允许用户编辑一个事件或者创建一个新事件。如果你希望这个视图控制器用于编辑事件，可以对此实例的 event 属性设置一个事件对象。如果你想让用户往系统中插入一个事件，可将 event 参数设为 nil。

EKEventEditViewController 实例的 editViewDelegate 属性是一个可接收委托消息的对象，委托消息从视图控制器发送出来，告诉程序员用户所采取的行动。你的委托对象必须处理的最重要的委托消息之一（也是必须实现的委托选择器），是 eventEditViewController:didCompleteWithAction:方法。每当用户取消了视图控制器时，这个委托方法会被调用，用户取消的方式由 didCompleteWithAction 参数说明。这个参数可以取下面的一些值：

EKEventEditViewActionCanceled

用户点击了视图控制器的 Cancel 按钮。

EKEventEditViewActionSaved

用户向 Calendar 数据库中保存（添加或编辑）了一个事件。

EKEventEditViewActionDeleted

用户从 Calendar 数据库删除了一个事件。

如果你将事件编辑视图控制器作为模态视图控制器显示，请确保在收到这个委托消息后

取消。

现在让我们继续来定义我们的视图控制器：

```
#import <UIKit/UIKit.h>
#import <EventKit/EventKit.h> #import <EventKitUI/EventKitUI.h>
@interface Presenting_Event_Edit_View_ControllersViewController
        : UIViewController <EKEventEditViewDelegate>

@property (nonatomic, strong) EKEventStore *eventStore;

@end
```

下一步，在我们的视图控制器的实现文件中，同步 eventStore 属性：

```
#import "Presenting_Event_Edit_View_ControllersViewController.h"
@implementation Presenting_Event_Edit_View_ControllersViewController
@synthesize eventStore;

...
```

现在让我们试着从一年以前查找第一个事件（不管它会是什么事件），然后显示编辑视图控制器来让用户进行编辑：

```
- (void)eventEditViewController:(EKEventEditViewController *)controller
      didCompleteWithAction:(EKEventEditViewAction)action{

  switch (action){

      case EKEventEditViewActionCanceled:{
      NSLog(@"Cancelled");
      break;
      }
      case EKEventEditViewActionSaved:{
      NSLog(@"Saved");
      break;
      }
      case EKEventEditViewActionDeleted:{
      NSLog(@"Deleted");
      break;
      }
  }
}

- (void)viewDidLoad{
  [super viewDidLoad];

  self.eventStore = [[EKEventStore alloc] init];

    NSTimeInterval NSOneYear = 1 * 365 * 24.0f * 60.0f * 60.0f;
    NSDate *startDate = [[NSDate date] dateByAddingTimeInterval:-NSOneYear];
    NSDate *endDate = [NSDate date];

    NSPredicate *predicate =
    [self.eventStore predicateForEventsWithStartDate:startDate
                                             endDate:endDate
```

```
                                                calendars:self.eventStore.calendars];

    NSArray *events = [self.eventStore eventsMatchingPredicate:predicate];

    if ([events count] > 0){
        EKEvent *event = [events objectAtIndex:0];

        EKEventEditViewController *controller =
        [[EKEventEditViewController alloc] init];

      controller.event = event;
      controller.editViewDelegate = self;

      [self.navigationController presentModalViewController:controller
                                                   animated:YES];
        }
      }

  - (void)viewDidUnload{
    [super viewDidUnload];
    self.eventStore = nil;
}
```

根据我们在设备上找到的事件，用户将看到类似于图 14-10 所示的画面：

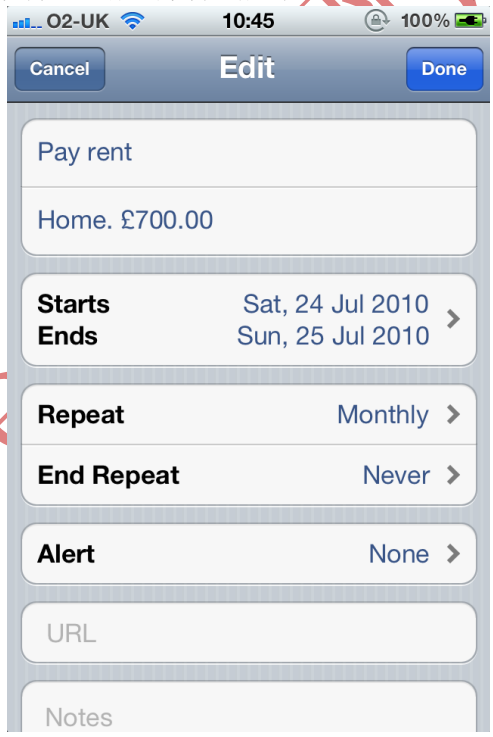

图 14-10. 显示了一个事件的编辑事件控制器

14.10.4. 参考

14.9 小节

点击这里访问：DevDiv.com 移动开发论坛