实验 1

使用 Event Kit 框架创建日历和提醒

实验结构

- 使用 Event Kit 框架创建日历和提醒

实验目标

本实验结束后，你将能够：

- 使用 Event Kit 框架在日历中创建事件
- 使用 Event Kit 框架从日历中删除事件
- 使用 Event Kit 框架创建提醒应用

导论

事件基本上也就是提醒和日历项。iOS 日历和提醒应用所使用的数据库叫作 **Calendar database**，也被称作 **Event Store**。

你可以通过在项目中添加 **Event Kit 框架**访问用户的日历和提醒信息。Event Kit 包含两部分：

- EventKitUI 框架
- EventKit 框架

EventKitUI 框架提供操控事件的用户界面元素。

实验：使用 EVENT KIT 框架创建日历和提醒

背景

Event Kit 框架让你的应用能够取回用户已有的日历和提醒数据。它还让你能够创建应用，为日历和提醒创建新事件。本实验中，你将使用 `EKEventStore` 类来为用户日历数据库创建事件。

你需要执行下面这些任务：

1. 访问日历数据
2. 使用 EventKitUI 框架创建事件
3. 为日历事件创建一个提醒

实验准备

要执行这些任务，你需要有：

- 带有 Xcode 的 Mac OS
- 苹果开发者帐户

实验推荐解决方案

1. 打开 Xcode 并选择 **Create a new Xcode project** 选项，如图 1 所示：

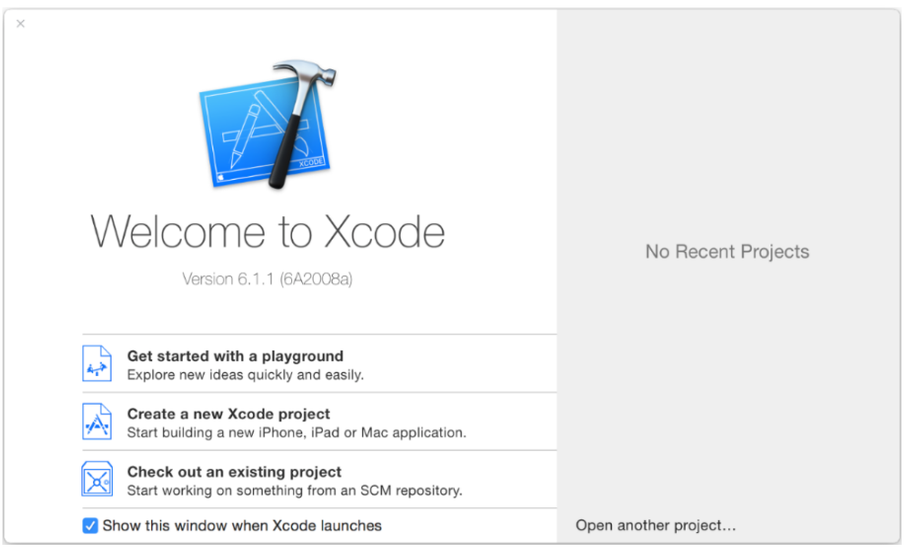

图 1:选择创建新 Xcode 项目

2. 选择 **Single View Application** 模板，并点 **Next**，如图 2 所示：

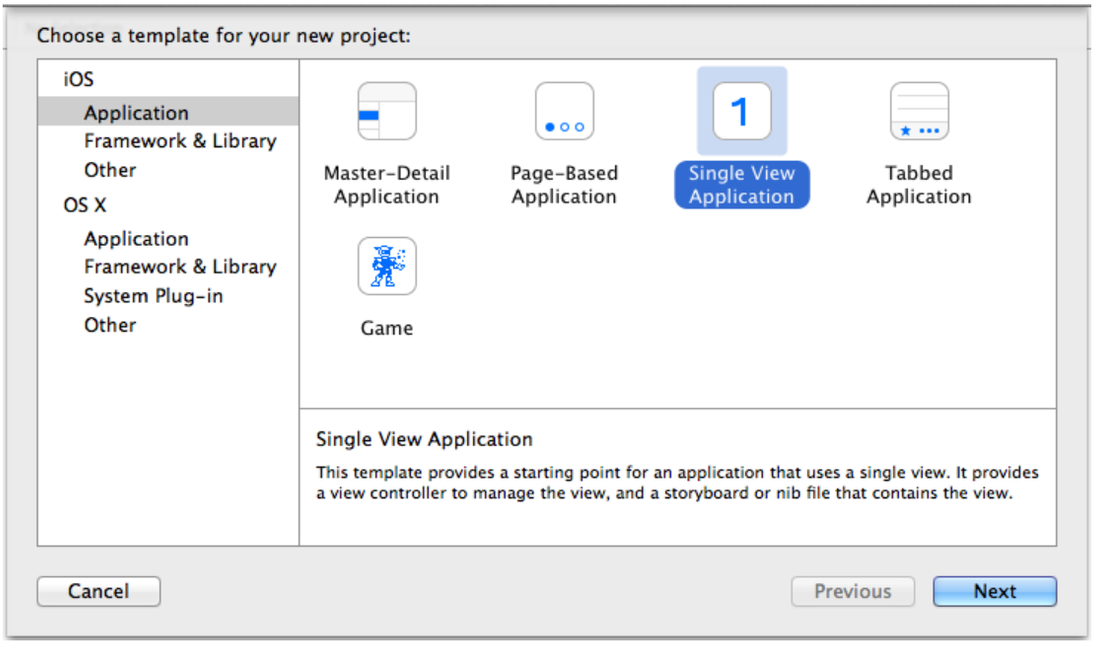

图 2:选择 Single View Application 模板

3. 将项目命名为 `MyEvents`。**Language** 下拉列表中选择 **Swift**，如图 3 所示。点 **Next** 推进到下一个界面。

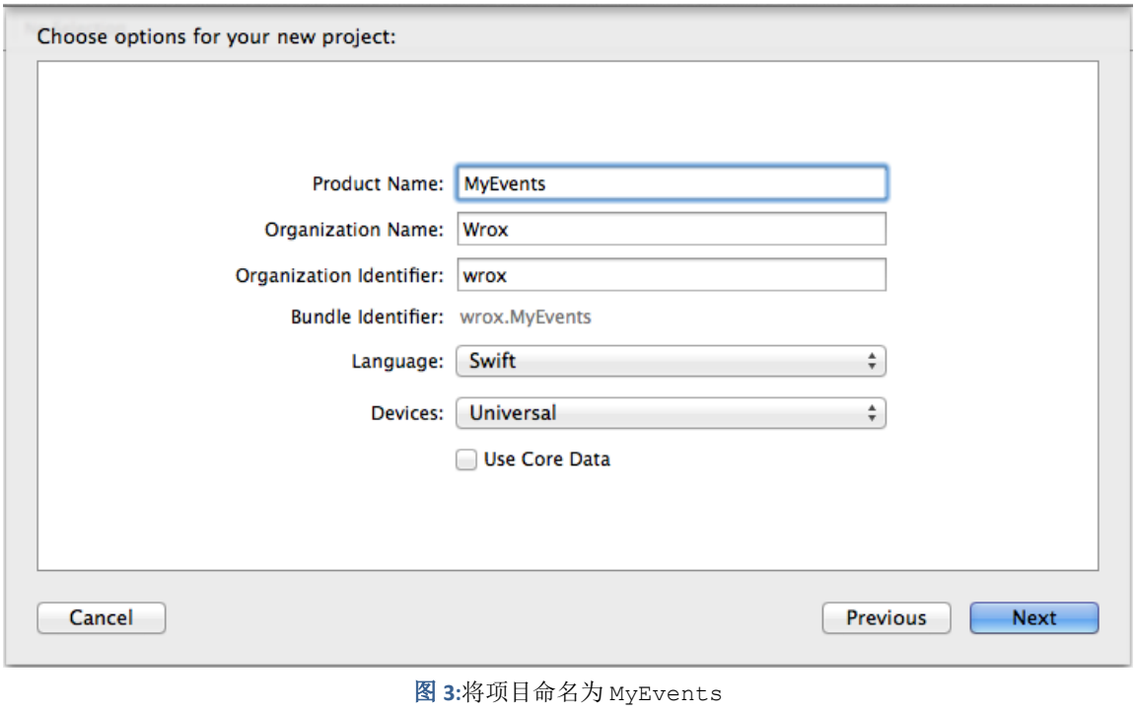

图 3:将项目命名为 `MyEvents`

4. 在下一个页面上，在对应位置选择 `MyEvents` 项目，然后点 **Create**。这会在 Xcode 中打开 Swift 项目，如图 4 所示：

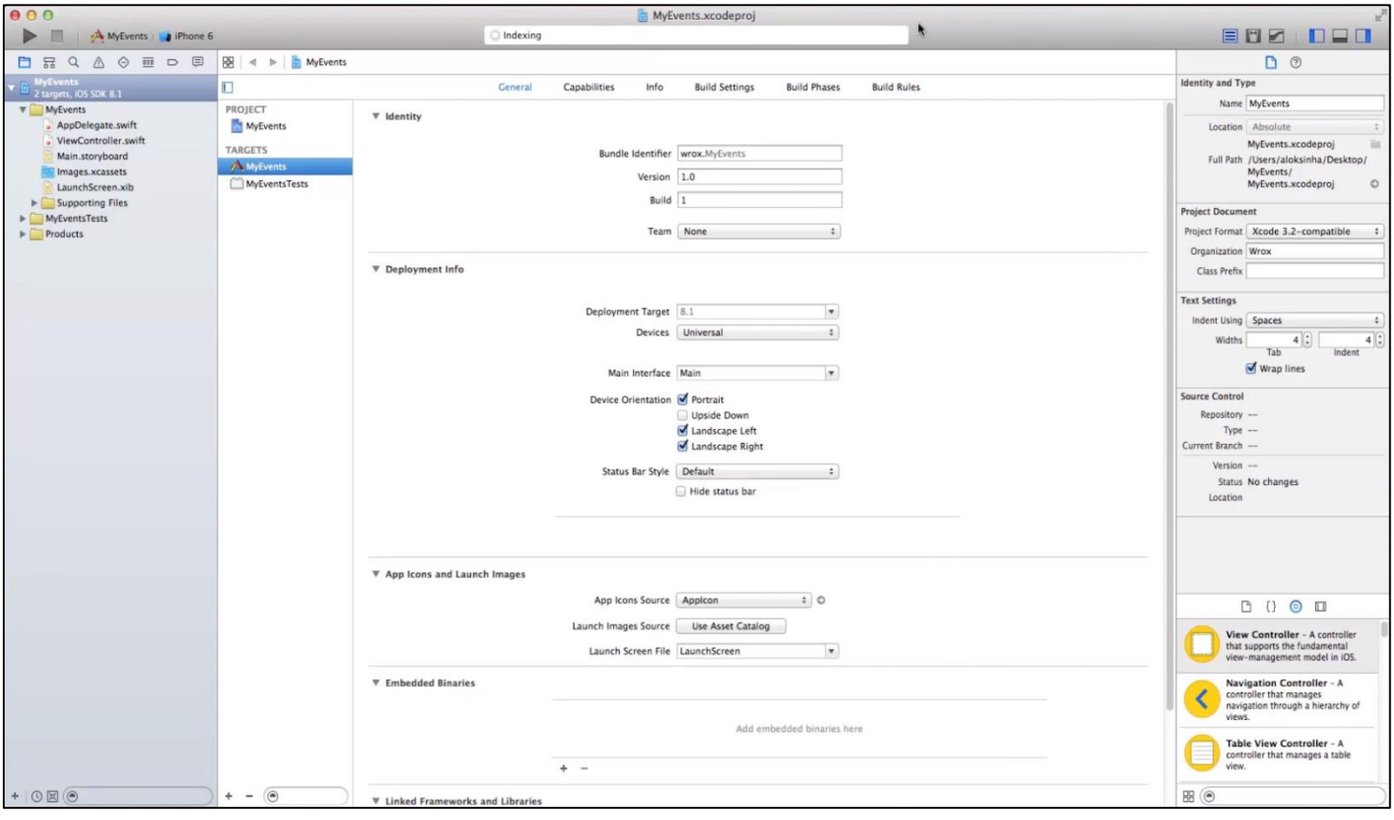

图 4:Swift 中的 MyEvents 项目

5. 下面，打开 ViewController.swift 并导入 EventKitUI，如下所示：

```
importUIKit
importEventKit
importEventKitUI
@UIApplicationMain
classAppDelegate: UIResponder, UIApplicationDelegate {
```

6. **苹果 iOS 不允许任何应用访问其个人数据库**。要访问用户的 Event Store，你需要从 iOS 获得权限。你需要一个用户界面来请求用户授权你的应用访问数据库的特定部分，例如日历。要获得访问权限，你需要使用 EKEventStore 类的 authorizationStatusForEntityType 类方法，并传递参数 EKEntityTypeEvent，以获得用户授权。在 viewController.swift 文件中使用如下代码：

```
classViewController: UIViewController {
    overridefuncviewDidLoad() {
        super.viewDidLoad()

        leteventStore = EKEventStore()

        switchEKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent) {

        case .Authorized:
            readEventsFromCalendar(eventStore)
```

```
        case .Denied:
            displayAccessDenied()

        case .NotDetermined:
            eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion:
        {[weak self](granted: Bool, error: NSError!) -> Void in

                if granted {
                    self!.insertEventIntoStore(eventStore)

                } else {
                    self!.displayAccessDenied()
            }
                })
        case .Restricted:
            displayAccessRestricted()

        }
    }
```

7. 添加代码之后，你需要用户界面来显示内容。为此，在 AppDelegate.swift 文件中的 didFinishLaunchingWithOptionslaunchOptions:方法中添加如下代码：

```
@UIApplicationMain
classAppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
    func application(application: UIApplication, didFinishLaunchingWithOptionslaunchOptions:
[NSObject: AnyObject]?) ->Bool {
    window = UIWindow(frame: UIScreen.mainScreen().bounds)
    if let Display = window
    {
        Display.backgroundColor = UIColor.whiteColor()
        Display.rootViewController = ViewController()
        Display.makeKeyAndVisible()
}
    return true
}
```

8. 下面运行模拟器，做法是点 **Project** 菜单中的 **Run**，或是使用 Command+Run 路径。你会得到如图 5 所示的输出结果。图 6 显示了 MyEvents 应用如何请求用户许可来访问集中式日历数据库。点 **OK** 来允许访问。

图 5:MyEvents 应用的启动界面

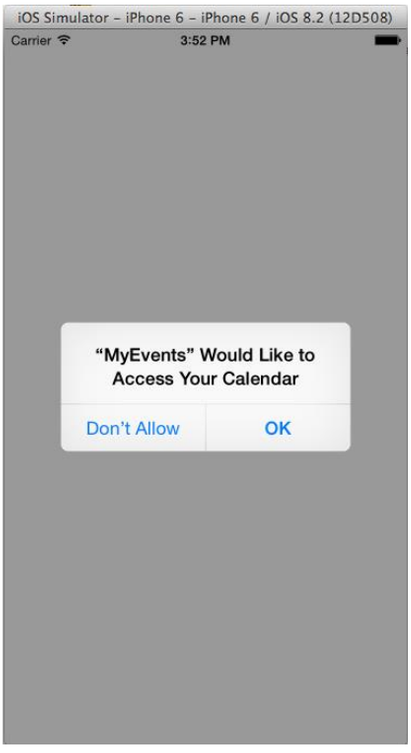

图 6:点 OK 来允许访问

9. 有了权限之后，你可以从 **Settings** 中改变 MyEvents 应用的配置，如图 7 和 8 所示：

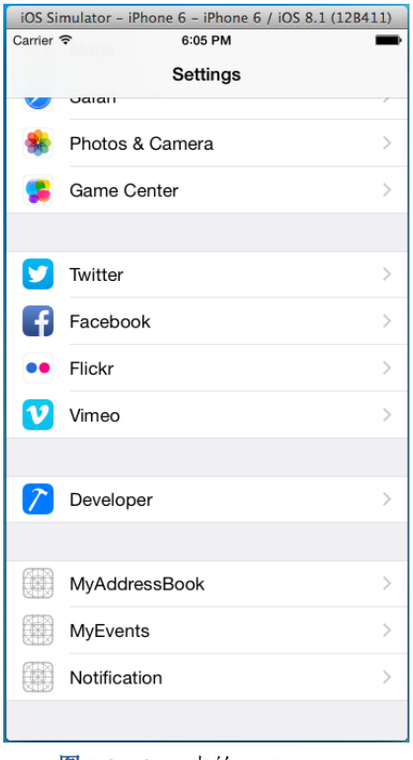

图 7:Settings 中的 MyEvents

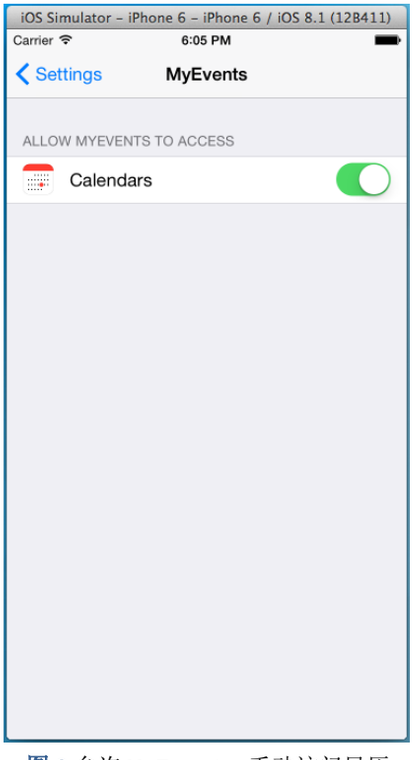

图 8:允许 MyEvents 手动访问日历

10. 访问日历数据库后，从 Event Store 读取数据。为此，使用 calendarsForEntityType:实例方法来遍历日历对象。如下面代码所示：

```
funcreadEventsFromCalendar(EventStore: EKEventStore){

    letCalType = ["Local","Exchange","Birthday"]
    letCalData = EventStore.calendarsForEntityType(EKEntityTypeEvent) as [EKCalendar]
    forcal in CalData{

        println("Calender Title = \(cal.title)")
        println("Calendar type = \(CalType[Int(cal.type.value)])")

        let color = UIColor(CGColor: cal.CGColor)
        println("Calendar color = \(color)")

        ifcal.allowsContentModifications == false {
            println("Calendercan not be modified")
}
}
```

11. 要在 Event Store 中添加事件，使用 sourceInEventStore:sourceType:sourceTitle:方法。该方法被用于查找特定类型和标题的事件源。在 ViewController.swift 中添加如下代码，用以查找本地日历和添加日历事件。

```
funcinsertEventIntoStore(store: EKEventStore){

    leteventSource = sourceInEventStore(store,
    type: EKSourceTypeLocal,
    title: "MyEvent title")

    ifeventSource == nil{
        println("Calender not configured with device")
        return
}

    letcal = calWithTitle("Calendar",
    type: EKCalendarTypeLocal,
    source: eventSource!,
    eventType: EKEntityTypeEvent)

    ifcal == nil{
        println("Calender not found.")
        return
    }

    letstartDate = NSDate()    //Start event from now
    letendDate = startDate.dateByAddingTimeInterval(0 * 60 * 60)    //event will end after
60 min, 60 sec

    ifAddEvent("MyEvent",
```

```
        startDate: startDate,
        endDate: endDate,
        inCalendar: cal!,
        inEventStore: store,
        notes: ""){
            println("Event created Successfully.")
    } else {
        println("Failed to create the event.")
        return
    }

}
```

12. 在上述代码中，你使用 `EKSourceTypeLocal` 来查找日历类型。找到日历并添加事件后，添加如下代码，用于创建 `NewEvent`，并将新事件创建到日历 Event Store。如下面代码所示：

```
funcAddEvent(title: String, startDate: NSDate, endDate: NSDate, inCalendar: EKCalendar,
inEventStore: EKEventStore, notes: String) ->Bool{
   /* If a calendar does not allow modification of its contents, then we cannot insert an
event into it */
   ifinCalendar.allowsContentModifications == false
   {
      println("Calender does not allow modification")
      return false
   }
   //event created
   varEventSet = EKEvent(eventStore: inEventStore)
   EventSet.calendar = inCalendar
   EventSet.title = title      //set title
   EventSet.notes = notes      //set note
   EventSet.startDate = startDate  //set start date
   EventSet.endDate = endDate      //set end date
   varerror:NSError?
   // save event into the calender
   letEventSetResult = inEventStore.saveEvent(EventSet, span: EKSpanThisEvent, error: &error)
   ifEventSetResult == false
   {
      if let CalError = error
      {
         println("An error occurred \(CalError)")
      }
   }
   returnEventSetResult
}
```

13. 再次运行模拟器。输出启动界面如图 9 所示。这里，你的 `MyEvent` 被创建了。

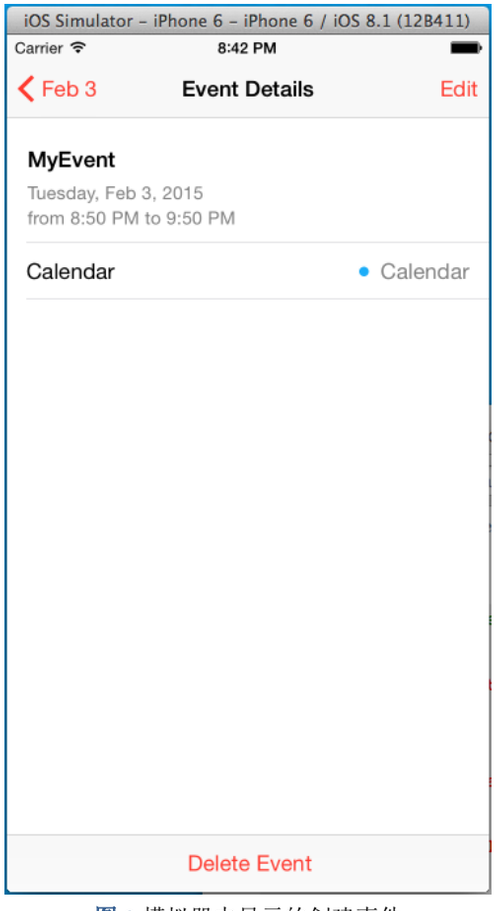

图 9:模拟器中显示的创建事件

14. 要添加提醒给日历事件，你可以添加如下代码到 ViewController.swift 文件：

```
funcEventAlarm(store: EKEventStore, cal: EKCalendar){
    letstartDate = NSDate(timeIntervalSinceNow: 60.0)  // start alarm from now
    letendDate = startDate.dateByAddingTimeInterval(20.0)
    letEventAlarm = EKEvent(eventStore: store)
    EventAlarm.calendar = cal
    EventAlarm.startDate = startDate
    EventAlarm.endDate = endDate
    let alarm = EKAlarm(relativeOffset: -5.0)   //End Alarm 5 sec before event
    EventAlarm.title = "MyEvent Alarm"
    EventAlarm.addAlarm(alarm)
    varerror:NSError?
    ifstore.saveEvent(EventAlarm, span: EKSpanThisEvent, error: &error){
        println("Event saved with alarm")
    } else if let theError = error{
        println("Error : Failt to save event = \(theError)")
    }
}
```

15. 再次运行模拟器，你会得到输出启动画面，提醒会设置成功，如图 10 所示：

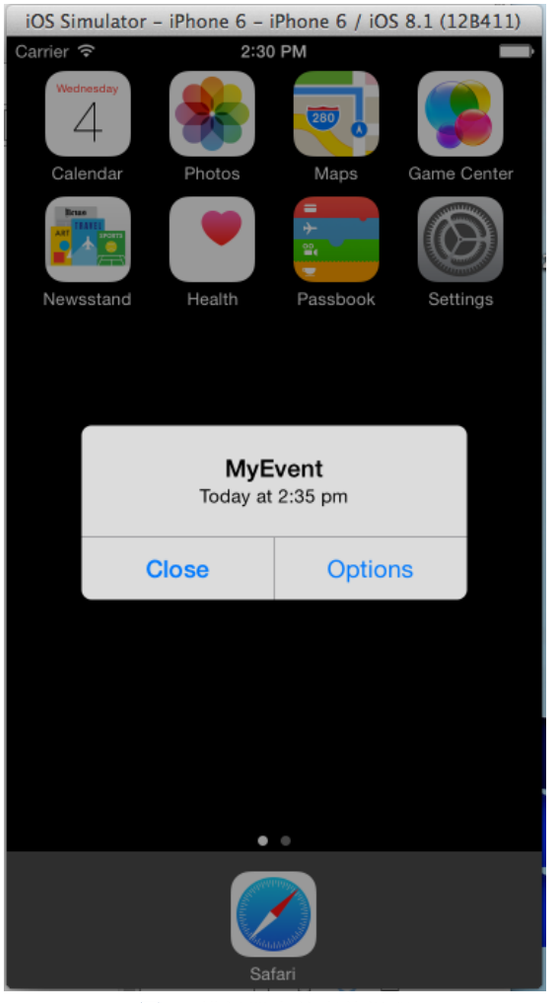

图 10:模拟器中的事件提醒

16. 要从日历中删除提醒，使用事件 EKEvent 和 removeAlarm 实例方法。要从日历删除事件，使用 removeEvent:span:commit:error 实例方法。
17. 使用 span 参数来确定要删除的事件数，是在日历中删除当前事件还是所有事件。要删除特定事件，使用带 removeEvent 参数的 EKSpanThisEvent，并提供值来指定 EKSpanThisEvent。

源码

AppDelegate.swift

```
   importUIKit
   importEventKit
   importEventKitUI

   @UIApplicationMain
   classAppDelegate: UIResponder, UIApplicationDelegate {

      var window: UIWindow?
      func application(application: UIApplication, didFinishLaunchingWithOptionslaunchOptions:
[NSObject: AnyObject]?) ->Bool {
```

```
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        if let Display = window
        {
              Display.backgroundColor = UIColor.whiteColor()
              Display.rootViewController = ViewController()
              Display.makeKeyAndVisible()
        }
              return true
    }
```

ViewController.swift

```
    importUIKit
    importEventKit
    importEventKitUI

    classViewController: UIViewController {



    overridefuncviewDidLoad() {
        super.viewDidLoad()

        leteventStore = EKEventStore()

        switchEKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent) {

              case .Authorized:
              readEventsFromCalendar(eventStore)

              case .Denied:
              displayAccessDenied()

              case .NotDetermined:
              eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion:
                    {[weak self](granted: Bool, error: NSError!) -> Void in

                    if granted {
                          self!.insertEventIntoStore(eventStore)

    } else {
                          self!.displayAccessDenied()
    }
    })
              case .Restricted:
                    displayAccessRestricted()
    }
         }
```

```
funcdisplayAccessDenied(){
   println("Access to the event store is denied.")
}

funcdisplayAccessRestricted(){
   println("Access To Event store is restricted.")
}

funcreadEventsFromCalendar(EventStore: EKEventStore){

   letCalType = ["Local","Exchange","Birthday"]
   letCalData = EventStore.calendarsForEntityType(EKEntityTypeEvent) as [EKCalendar]

   forcal in CalData{

         println("Calendar Title = \(cal.title)")
         println("Calendar type = \(CalType[Int(cal.type.value)])")

         let color = UIColor(CGColor: cal.CGColor)
         println("Calendar color = \(color)")

         ifcal.allowsContentModifications == false {
               println("Calendar cannot be modified")
     }
   }
}
overridefuncdidReceiveMemoryWarning() {
   super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

funcsourceInEventStore(
   eventStore: EKEventStore,
   type: EKSourceType,
   title: String) ->EKSource?{

   for source in eventStore.sources() as [EKSource]{
         ifsource.sourceType.value == type.value&&
   source.title.caseInsensitiveCompare(title) == NSComparisonResult.OrderedSame{
               return source
}
   }

   return nil
}
```

```
   funccalWithTitle(
      title: String,
      type: EKCalendarType,
      source: EKSource,
      eventType: EKEntityType) ->EKCalendar?{

      for calendar in source.calendarsForEntityType(eventType).allObjects
      as [EKCalendar]{
            ifcalendar.title.caseInsensitiveCompare(title) ==
                  NSComparisonResult.OrderedSame&&
                  calendar.type.value == type.value{
                  return calendar
            }
   }

      return nil
   }

   funcAddEvent(title: String, startDate: NSDate, endDate: NSDate, inCalendar: EKCalendar,
inEventStore: EKEventStore, notes: String) ->Bool{

      ifinCalendar.allowsContentModifications == false
   {
            println("Calendar does not allow modification")
            return false
     }
      //event created
      varEventSet = EKEvent(eventStore: inEventStore)
      EventSet.calendar = inCalendar
      EventSet.title = title      //set title
      EventSet.notes = notes      //set note
      EventSet.startDate = startDate  //set start date
      EventSet.endDate = endDate      //set end date

      varerror:NSError?
   // save event into the calendar
      letEventSetResult = inEventStore.saveEvent(EventSet, span: EKSpanThisEvent, error:
&error)

      ifEventSetResult == false
      {
            if let CalError = error
            {
                  println("An error occurred \(CalError)")
            }
      }

      returnEventSetResult
```

```
    }

    funcinsertEventIntoStore(store: EKEventStore){

        leteventSource = sourceInEventStore(store,
            type: EKSourceTypeLocal,
            title: "MyEvent title")

        ifeventSource == nil{
            println("Cloud Calendar not configured with device")
            return
        }

        letcal = calWithTitle("Calendar",
            type: EKCalendarTypeLocal,
            source: eventSource!,
            eventType: EKEntityTypeEvent)

        ifcal == nil{
            println("Calendar not found.")
            return
        }

        letstartDate = NSDate()    //Start event from now
        letendDate = startDate.dateByAddingTimeInterval(0 * 60 * 60)    //event will end after
60 min, 60 sec

        ifAddEvent("MyEvent",
            startDate: startDate,
            endDate: endDate,
            inCalendar: cal!,
            inEventStore: store,
            notes: ""){
            println("Event created Successfully.")
        } else {
            println("Failed to create the event.")
            return
        }

}

    funcEventAlarm(store: EKEventStore, cal: EKCalendar){

        letstartDate = NSDate(timeIntervalSinceNow: 60.0)  // start alarm from now

        letendDate = startDate.dateByAddingTimeInterval(20.0)

        letEventAlarm = EKEvent(eventStore: store)
```

```
        EventAlarm.calendar = cal
        EventAlarm.startDate = startDate
        EventAlarm.endDate = endDate

        let alarm = EKAlarm(relativeOffset: -5.0)    //End Alarm 5 sec before event

        EventAlarm.title = "MyEvent Alarm"
        EventAlarm.addAlarm(alarm)

        varerror:NSError?
        ifstore.saveEvent(EventAlarm, span: EKSpanThisEvent, error: &error){
              println("Event saved with alarm")
    } else if let theError = error{
              println("Error : Failed to save event = \(theError)")
        }

    }
    }
```