iOS 5 Programming Cookbook

第十六章
核心运动

版本 1.0

翻译时间：2012-08-27

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

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十四章_日期，日程表和事件

第 15 章 图形和动画

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十五章_图形和动画

第 16 章 核心运动

16.0. 介绍

iOS 设备，例如 iPhone 和 iPad 通常会配备加速度计硬件。一些 iOS 设备也可能会配备陀螺仪，例如 iPhone4 和 iPad2。iOS 应用程序中，尝试使用加速度计或者陀螺仪之前，必须先检测应用程序所运行的设备的传感器是否可用。16.1 和 16.2 小节包含了检测加速度计和陀螺仪的可用性相关技术的讲解。在 iPhone4 和 iPad2 中的陀螺仪可以检测六轴的运动。

我们举一个简单的例子，这样你就可以尝试找出陀螺仪的价值所在。当你稳稳地拿着终端，坐在一个电脑椅上，顺时针或者逆时针旋转时，终端的加速计是没办法在它的垂直轴检测到终端的旋转的。从地板或者地球的角度来说，设备正围绕垂直轴旋转，但它并没有围绕它自己的垂直轴（即终端的垂直轴心）旋转。因此，加速计此时并不会检测到任何的动作。

但是，装备有陀螺仪的 IOS 终端就可以检测到上面提到过的动作。这就让我们在检测终端运动时更加流畅。这点对于游戏而言尤其有用，游戏开发中，开发者不仅要知道由加速计提供的 x,y,z 轴的信息，而且还要得到终端 x,y,z 轴的变化与地球的相对关系，而这，正是陀螺仪为我们提供的。

程序员会用到 Core Motion 框架来访问加速计和陀螺仪的数据（要是有的话）。本章通篇都是使用这个 Core Motion 框架。通过下列步骤可以为你的项目添加这个框架：

1、在 Xcode 中点击你的项目图标。
2、选择你要添加核心动作框架的目标。
3、在屏幕顶部，选择 Build Phases 选项卡。
4、在 Build Phases 选项卡中找到并展开 Link Binary with Libraries box，然后在它左下角点击 + 按钮。
5、在框架列表中，选择 CoreMotion.framework 并点击 Add 按钮。（见图 16-1）

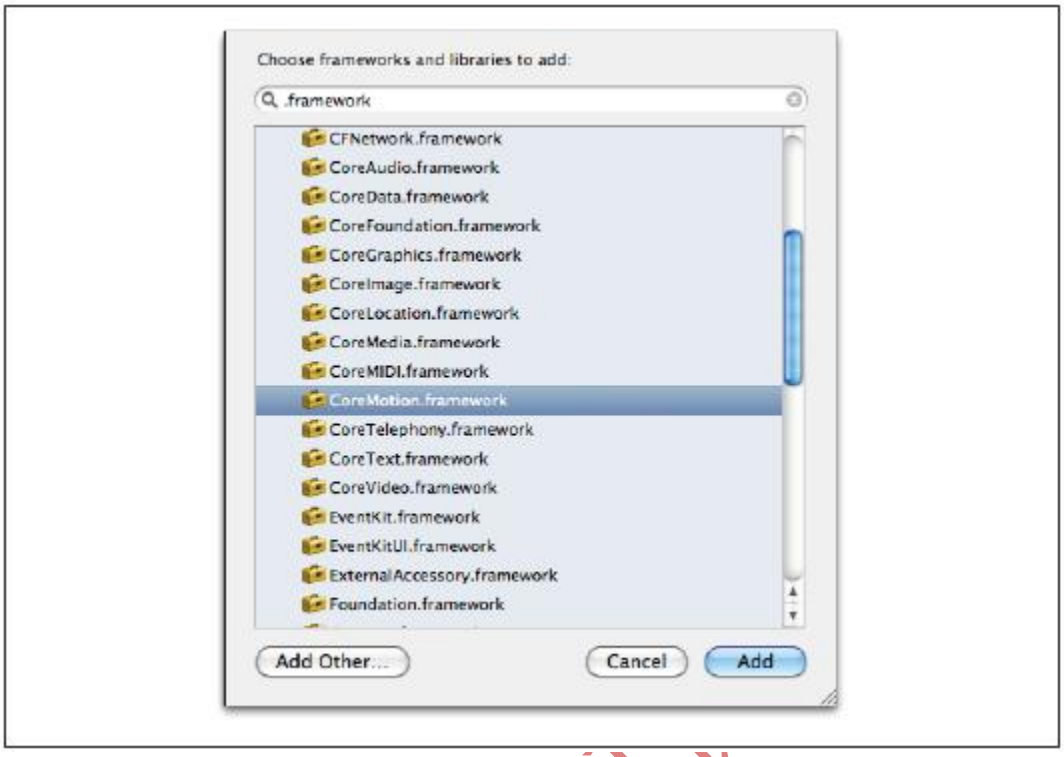

图 16-1 为目标添加 Core Motion 框架

iOS 模拟器是不能模拟加速计跟陀螺仪硬件的。但你可以通过 Hardware(硬件)→Shake Gesture(摇晃手势)来生成一个摇晃动作。(见图 16-2)

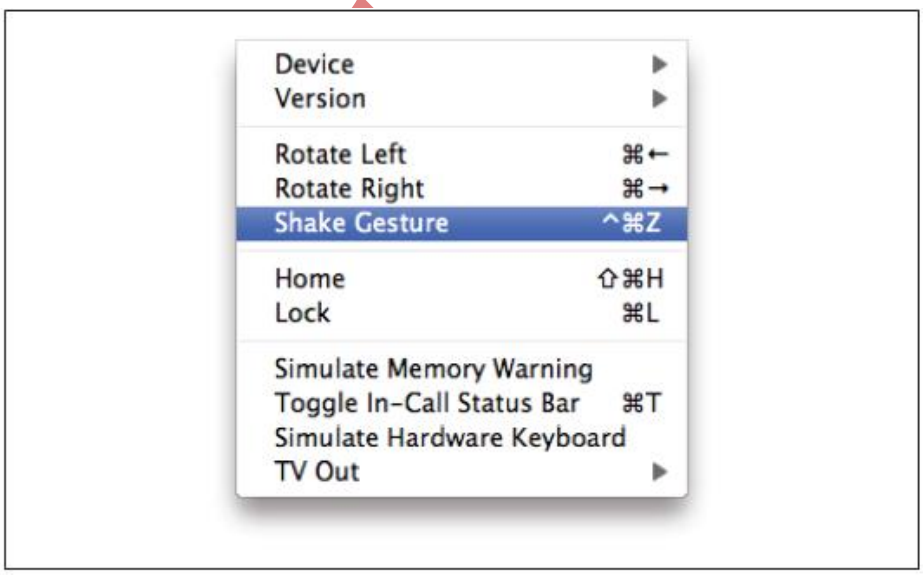

图 16-2 模拟器中的 Shake Gesture

16.1. 检测加速度计的可用性

16.1.1. 问题

在你的程序中，你想要检测加速计硬件是否可用。

16.1.2. 方案

用 CMMotionManager 的 isAccelerometerAvailable 方法可以检测加速计硬件，还可以用于检测加速计硬件是否正在向程序发送更新。

首先我们要确认我们导入了要求的头文件。

```
#import <UIKit/UIKit.h>
#import <CoreMotion/CoreMotion.h>
@interface Detecting_the_Availability_of_an_AccelerometerAppDelegate
: UIResponder <UIApplicationDelegate>
    @property (strong, nonatomic) UIWindow *window;
@end
```

然后我们就可以在我们应用的代理实现类中进行加速计的可用性检测了。

```
- (BOOL) application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
        CMMotionManager *motionManager = [[CMMotionManager alloc] init];
    if ([motionManager isAccelerometerAvailable]){
            NSLog(@"Accelerometer is available.");
    } else{
            NSLog(@"Accelerometer is not available.");
    }
    if ([motionManager isAccelerometerActive]){
            NSLog(@"Accelerometer is active.");
    } else {
            NSLog(@"Accelerometer is not active.");
    }
    self.window = [[UIWindow alloc] initWithFrame:
                    [[UIScreen mainScreen] bounds]];
    self.window.backgroundColor = [UIColor whiteColor];
                                    [self.window makeKeyAndVisible];
    return YES;
}
```

在运行你应用的 iOS 终端上的加速计硬件也许是可用的，但这并不意味着加速计硬件会向你的程序发送更新。要是加速计或者陀螺仪在给你的程序发送更新，我们称之为激活(*active*)，这就要求你定义一个代理对象，就像晚些你会看到的那样。

要是你在 iOS 模拟器中跑这段代码，你会在控制台窗口中看到类似下面的信息：

```
Accelerometer is not available.
Accelerometer is not active.
```

在 iPhone4 终端上跑相同的代码，你可能得到类似这样的信息：

```
Accelerometer is available.
Accelerometer is not active.
```

16.1.3. 讨论

一部 iOS 终端可能有一个内置的加速计。由于我们不能确定某个 iOS 终端是否装备有内置加速计，所以我们最好在用前检测一下可加速计是否可用。

要检测加速计硬件的可用性，你可以实例化一个 CMMotionManager 的对象并访问它的 isAccelerometerAvailable 方法。这个 BOOL 类型的方法在加速计可用时会返回 YES，反之返回 NO。

另外，你可以通过 CMMotionManager 的 isAccelerometerAvailable 方法来检测加速计是否在向你的程序发送更新（即是否是激活的）。你会在 16.3 小节中学习如何获取加速计的数据。

16.1.4. 参考

16.3 小节

16.2. 检测陀螺仪的可用性

16.2.1. 问题

想要知道运行你程序的当前 iOS 终端的陀螺仪是否可用。

16.2.2. 方案

用 CMMotionManager 的实例方法 isGyroAvailable 可以检测陀螺仪硬件的可用性。isGyroAvailable 方法还可以检测陀螺仪是否在向你的程序发送更新（即是否是激活的）。

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
        CMMotionManager *motionManager = [[CMMotionManager alloc] init];
        if ([motionManager isGyroAvailable]){
                NSLog(@"Gryro is available.");
        } else {
                NSLog(@"Gyro is not available.");
        }
        if ([motionManager isGyroActive]){
                NSLog(@"Gryo is active.");
        } else {
                NSLog(@"Gryo is not active.");
        }
        self.window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];
        self.window.backgroundColor = [UIColor whiteColor];
        [self.window makeKeyAndVisible];
        return YES;
}
```

iOS 模拟器没有陀螺仪的模拟。要是你在模拟器中跑这段代码，你会在控制台窗口中看到类似下面的信息：

```
Gyro is not available.
Gyro is not active.
```

16.2.3. 讨论

要是你打算发布一款会使用陀螺仪的应用，你要保证 iOS 终端在没有陀螺仪硬件时也可

以跑你的应用。比如，当你把陀螺仪作为你的游戏的一部分时，你要确保版本匹配的终端即便没有安装陀螺仪也可以玩这个游戏。

要作到这点，你就必须先实例化一个 CMMotionManager 的对象。然后访问（BOOL 类型的）isGyroAvailable 方法来获知在运行你的代码的终端上是否陀螺仪是否可用。你还可以用这个函数来查看陀螺仪是否在给你的应用发送一些更新。要了解更多相关信息，可以参阅 16.5 小节的内容。

16.2.4. 参考

16.5 小节

16.3. 获取加速器的数据

16.3.1. 问题

如果你想让 iOS 向你的应用发送加速计的数据。

16.3.2. 方案

使用 CMMotionManager 的实例方法：startAccelerometerUpdatesToQueue:withHandler: 。下面是使用 CMMotionManager 获取加速计更新的试图控制器的头文件：

```
#import <UIKit/UIKit.h>
#import <CoreMotion/CoreMotion.h>
        @interface Retrieving_Accelerometer_DataViewController : UIViewController
@property (nonatomic, strong) CMMotionManager *motionManager;
@end
```

下面我们将实现我们的视图控制器并使用 CMMotionManager 类的 startAccelerometerUpdatesToQueue:withHandler:方法：

```
#import "Retrieving_Accelerometer_DataViewController.h"
@implementation Retrieving_Accelerometer_DataViewController
    @synthesize motionManager;
        - (void)viewDidLoad{
    [super viewDidLoad];
    self.motionManager = [[CMMotionManager alloc] init];
    if ([self.motionManager isAccelerometerAvailable]){
            NSOperationQueue *queue = [[NSOperationQueue alloc] init];
              [self.motionManager startAccelerometerUpdatesToQueue:
queuewithHandler:^(CMAccelerometerData *accelerometerData, NSError *error) {
                        NSLog(@"X = %.04f, Y = %.04f, Z = %.04f",
                        accelerometerData.acceleration.x,
                        accelerometerData.acceleration.y,
                        accelerometerData.acceleration.z);
                }];
        } else {
            NSLog(@"Accelerometer is not available.");
        }
}
```

```
- (void)viewDidUnload{
        [super viewDidUnload];
        self.motionManager = nil;
}
- (BOOL)shouldAutorotateToInterfaceOrientation
:(UIInterfaceOrientation)interfaceOrientation{
        return YES;
}
@end
```

16.3.3. 讨论

加速计会报告三维数据（三个轴），即 iOS 会向你的程序报告 x, y, z 轴的值。这些值是封装在一个 CMAcceleration 构造器中的。

```
typedef struct {
        double x;
        double y;
        double z;
} CMAcceleration;
```

当你拿着终端，让终端的屏幕纵向对着你：

- x 轴会从终端的水平中心的左边跑到右边，值域范围从左到右为：-1 ~ +1
- y 轴会从终端的垂直中心的底部跑到顶部，值域范围从顶部到顶部为：-1 ~ +1
- z 轴会从终端的后面跑到朝向你的一面，值域范围从终端的后面到前面为：-1 ~ +1

理解加速计硬件报告的这些值的最好方法就是看些例子。下面就有一个例子：我们假设你拿着一台 iOS 终端，它的屏幕对着你，底部朝向地面，顶部指向上面。要是你可以向刚刚说的那样拿的很稳，没有任何偏移的话，你此时得到 x, y, z 轴的值会是(x:0.0, y:-1.0, z:0.0) 。现在你再上面姿势的基础上再做下下面的动作：

1. 让设备顺时针旋转 90 度。此刻你得到的值将是：(x:+1.0, y:0.0, z:0.0)
2. 让设备再顺时针旋转 90 度。现在终端的顶部一定是指向地面的。此刻你得到的值将是：(x:0.0, y: +1.0, z:0.0)
3. 让设备再顺时针旋转 90 度。现在终端的顶部一定是指向左边的。此刻你得到的值将是：(x: -1..0, y: 0.0, z:0.0)
4. 最后设备再顺时针旋转 90 度。现在终端的再次底部朝向地面，顶部指向上面。此刻你得到的值将是：(x:0.0, y:-1.0, z:0.0)

从这些值中，我们可以看到终端绕着 z 轴转时加速计报告 x 和 y 值改变了，z 值不变

我们来做另外一个实验。首先还是让终端的屏幕对着你，底部朝向地面，顶部指向上面。如你所知，此时得到 x, y, z 轴的值会是(x:0.0, y:-1.0, z:0.0) 。现在再做下下面的动作：

1. 让终端沿着 x 轴向后翻转 90 度，此时它的顶部会指向后面。换句话说，像把它放在桌子上并且面朝上那样拿着它。此刻你得到的值将是：(x:0..0, y: 0.0, z: -1.0)
2. 现在让终端沿着 x 轴再向后翻转 90 度，此时它的后面会朝向你，它的顶部指向地面，底部指向天空。此刻你得到的值将是：(x:0..0, y: 1.0, z:0.0)
3. 让终端沿着 x 轴再向后翻转 90 度，此时它的正面会朝向地面，它的反面指向天空，顶部指向你。此刻你得到的值将是：(x:0..0, y: 0.0, z: 1.0)
4. 最后，你再向同样的方向做一次，此时它的屏幕对着你，底部朝向地面，顶部指向天空。你得到的值会跟刚开始的时候是一样的。

因此，我们观察到当沿着它的 x 轴旋转时，y 轴跟 z 轴的值会改变，而 x 轴的则不变。

我鼓励你尝试一下第三种旋转的方式—绕着它的 y 轴（从顶部到底部）旋转—并观察 x, z 轴的值的变化。

要接收加速计的更新，你有两个选项：

1. CMMotionManager 的实例方法：startAccelerometerUpdatesToQueue:withHandler:
   这个方法会把加速计的更新信息传递给一个操作队列（一种 NSOperationQueue 队列）并需要有一些 Grand Central Dispatch(GCD)的关于块的基础知识。关于块的更多信息可以参阅第 5 章。
2. CMMotionManager 的实例方法：startAccelerometerUpdates:withHandler:
   一旦你调用了这个方法，加速计（如何可用的话）就会开始在动作管理对象中更新加速计数据。你需要设置你自己的线程来不断读取 CMMotionManager 类的（类型为 CMAccelerometerData 的）accelerometerData 属性的值。

在本节，你会用到第一个方法（块的方式）。我极力推荐在你开始本节内容前，先阅读第五章的内容。我们为 CMMotionManager 类的实例方法：startAccelerometerUpdatesToQueue:withHandler:提供的，块的类型必须是 CMAccelerometerHandler:类型的

```
typedef void (^CMAccelerometerHandler)
(CMAccelerometerData *accelerometerData, NSError *error);
```

换句话说，我们必须在块中接收两个参数。第一个参数必须是 CMAccelerometerData 类型的，并且第二个参数必须是 NSError 类型的，就像上面代码示例的实现这样。

16.3.4. 参考

第 16.1 小节

16.4. 检测 iOS 设备的摇晃

16.4.1. 问题

你如果想知道用户何时晃动了 iOS 终端

16.4.2. 方案

使用你的应用的窗口对象的 motionEnded:withEvent:方法

16.4.3. 讨论

当 iOS 终端捕获到一个动作时，你应用的窗口的 motionEnded:withEvent:方法就会被调用。下面是这个方法的简单实现：

```
- (void) motionEnded:(UIEventSubtype)motion
withEvent:(UIEvent *)event{
        /* Do something with the motion */
}
```

如你所见，这个 motion 参数的类型是 UIEventSubtype。UIEventSubtype 类型的其中一个值就是我们所感兴趣的 UIEventSubtypeMotionShake。只要我们检测到这个事件，我们就知道用户晃动了它的手机。为了得到我们应用的窗口，我们需要继承 UIWindow。具体步骤是：

1.在 Xcode 里，当你打开了你的项目后，选择 File→New File

2.从左边看，选择 iOS 主目录下的子目录 Cocoa Touch。

3.在右边的列表中，选择 Objective-C class 然后点击 Next(下一步)，就像图 16-3 所展示的。

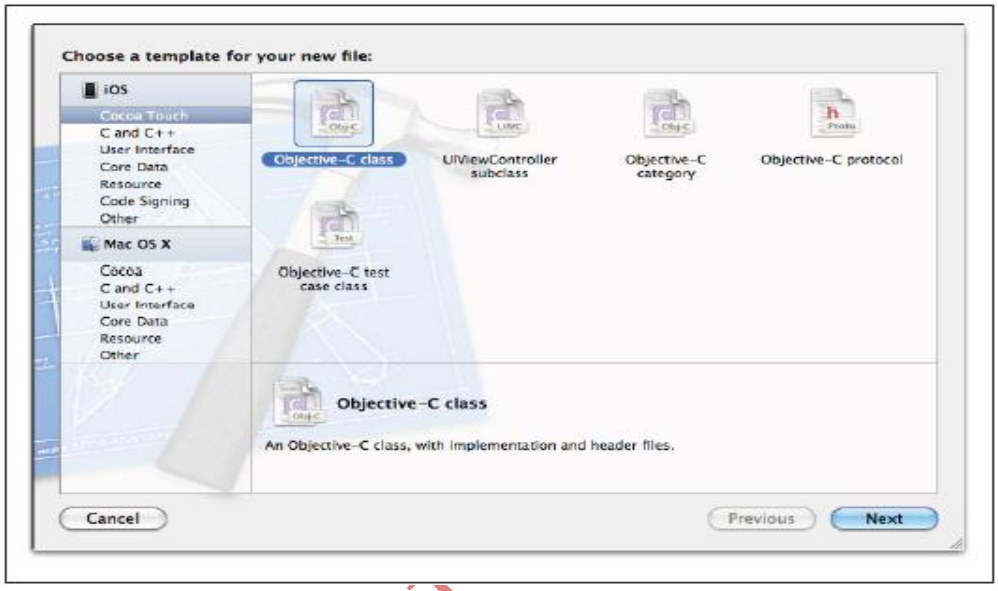

图 16-3 为我们的窗口创建一个新的 Objective-C class

4. 现在确保我们继承了 UIWindow 类然后点击 Next，就像图 16-4 所展示的。

图 16-4 继承 UIWindow

5. 在这个屏幕中，设置文件名为 MyWindow 然后点击 Save 按钮，就像图 16-5 所展示的。

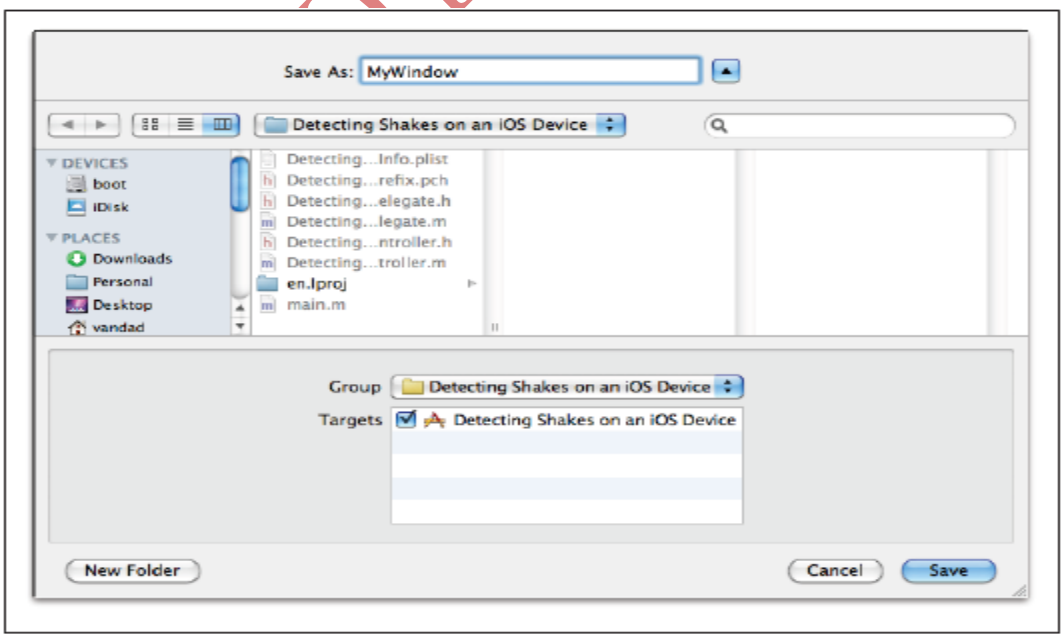

图 16-5 保存类文件

现在我们有了一个窗口类，打开我们应用的代理实现类，确保应用的代理的窗口对象是我们创建的 MyWindow 类的一个实例：

```
    #import "Detecting_Shakes_on_an_iOS_DeviceAppDelegate.h"
    #import "Detecting_Shakes_on_an_iOS_DeviceViewController.h"
#import "MyWindow.h"
@implementation Detecting_Shakes_on_an_iOS_DeviceAppDelegate
@synthesize window = _window;
@synthesize viewController = _viewController;
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
        self.window = [[MyWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];
        UIUserInterfaceIdiom device = [[UIDevice currentDevice] userInterfaceIdiom];
        if (device == UIUserInterfaceIdiomPhone) {
                self.viewController =
                [[Detecting_Shakes_on_an_iOS_DeviceViewController alloc]
                initWithNibName:@"Detecting_Shakes_on_an_iOS_DeviceViewController_iPhone"
                bundle:nil];
                //16.4 Detecting Shakes on an iOS Device | 829
         } else {
                self.viewController =
                [[Detecting_Shakes_on_an_iOS_DeviceViewController alloc]
                initWithNibName:@"Detecting_Shakes_on_an_iOS_DeviceViewController_iPad"
                bundle:nil];
         }
         self.window.rootViewController = self.viewController;
         [self.window makeKeyAndVisible];
         return YES;
}
```

要是你现在在 iOS 模拟器中模拟晃动事件（见 16.0 小节），你会看到我们的窗口会在控制台窗口打印语句：Detected a shake

16.4.4. 参考

16.0 小节

16.5. 获取陀螺仪的数据

16.5.1. 问题

如果你想让 iOS 向你的应用发送加速计的数据。

16.5.2. 方案

遵循下列步骤：

1. 判断 iOS 终端的陀螺仪硬件是否可用。请参阅 16.2 小节的指导。
2. 如果陀螺仪硬件可用，要想确保它已经在为你发送更新。请参阅 16.3 小节的指导。
3. 使用 CMMotionManager 类的实例方法 setGyroUpdateInterval:来设置每秒你想接收到的更新信息数。例如，要设置为 20 条更新信息每秒，那么值就设置为 1.0/2.0。
4. 调用 CMMotionManager 类的实例方法 startGyroUpdatesToQueue:withHandler:.这个队列对象可以简单作为主操作队列（就像我们晚点会看到的）并且处理块必须遵循 CMGyroHandler 的格式。

下面是上述步骤的代码实现：

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
        CMMotionManager *manager = [[CMMotionManager alloc] init];
        if ([manager isGyroAvailable]){
                if ([manager isGyroActive] == NO){
                        [manager setGyroUpdateInterval:1.0f / 40.0f];
                        NSOperationQueue *queue = [[NSOperationQueue alloc] init];
                        [manager
                        startGyroUpdatesToQueue:queue
                        withHandler:^(CMGyroData *gyroData, NSError *error) {
                                NSLog(@"Gyro Rotation x = %.04f", gyroData.rotationRate.x);
                                NSLog(@"Gyro Rotation y = %.04f", gyroData.rotationRate.y);
                                NSLog(@"Gyro Rotation z = %.04f", gyroData.rotationRate.z);
                        }];
                } else {
                        NSLog(@"Gyro is already active.");
                }
        } else {
                NSLog(@"Gyro isn't available.");
        }
        self.window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];
        self.window.backgroundColor = [UIColor whiteColor];
        [self.window makeKeyAndVisible];
        return YES;
}
```

16.5.3. 讨论

有了 CMMotionManager 类，应用开发者就可以尝试从 iOS 获取陀螺仪的更新信息。但你必须先确保陀螺仪硬件在运行你应用的 iOS 终端上是可用的（可以参阅 16.2 小节）。在这之后，你就可以调用 CMMotionManager 的实例方法 setGyroUpdateInterval:来设置你想要每秒从陀螺仪硬件接收到的更新信息数。例如，你想要每秒接收到 N 条更新信息，那么就设值为 1.0/N。

设置完更新间隔后，你就可以调用 CMMotionManager 的实例方法 startGyroUpdatesToQueue:withHandler:来为更新设置一个处理块了。要了解更多关于块的信息，请参阅第五章。你的块对象类型必须为 CMGyroHandler，接收两个参数：

gyroData

来自陀螺仪硬件的数据，包含在一个 CMGyroData 类型的对象中。你可以使用 CMGyroData 的 rotationRate 属性来访问数据的 x, y, z 值，它代表了 3 个欧拉角，即旋转，倾斜和平摇。你可以阅读飞行动力学来了解更多相关信息。

Error: 当陀螺仪在给我们发送更新数据时可能会产生一个 NSError 类型的错误。

要是你不想使用块对象，你必须调用 CMMotionManager 的实例方法 startGyroUpdates 来替代 startGyroUpdatesToQueue:withHandler:实例方法，并设置你自己的线程来持续读取你使用的 CMMotionManager 实例的，gyroData 属性发布的陀螺仪硬件更新信息。

16.5.4. 参考

16.2 小节