

实验 3



应用的技术分析

实验结构

- ▶▶ 为应用执行技术分析

实验目标

本实验结束后，你将能够：

- ▶▶ 在投入生产之前，对应用进行技术分析

导论

假设你为一个客户开发了一款应用，现在你要提供测试版供客户评估，看应用是否满足客户预期。你需要执行技术分析，确保应用能够按照预期那样工作。

实验：为应用执行技术分析

有研究表明，在智能手机上下载、安装并运行应用时，用户希望应用能够立刻平滑地运行。第一次使用时就崩溃的应用，多数情况下用户都会直接删除。此外，当应用使用大量存储空间或内存时，用户也有可能终止和删除应用。

背景

让应用执行差劲、不用户友好的主要有下面这些技术原因

- **应用崩溃**: 这是用户停止使用应用的最主要原因之一。你可以使用 **crash handler** 来避免应用崩溃。**crash handler** 能够在应用崩溃发生时发送消息给用户，也能让你收集到崩溃的日志信息。**crash handler** 会使用网络服务器发送用于分析的信息给你。
- **阻塞主线程**: 这一技术问题可能导致应用没有响应。不要在应用的主线程上执行长时间运行的繁重任务，因为 UIKit 要做所有那些工作。
- **内存泄露**: 应用在不正确管理内存分配时会出现内存泄露。结果会导致预计存在于内存中的对象无法被访问。
- **同步 HTTP 请求和流量消耗过大**: 很耗流量的应用会导致用户花费更多钱。用户会很不开心，会指责你的应用。为了避免这一问题，你可以使用苹果的样本应用 **Reachability**。该应用演示了如何使用 SystemConfiguration 框架监测设备的网络状态。
- **耗电**: 应用不能太耗电。一大很耗电的原因是调用了 CLLocationManager 类的 startUpdatingLocation 方法而不调用 stopUpdatingLocation 方法。

本实验中，你将为下面这些在应用中执行技术分析：

1. 解决应用崩溃
2. 解决主线程阻塞错误
3. 解决内存泄露
4. 解决同步 HTTP 请求及流量消耗过大
5. 解决耗电问题

实验准备

要执行这些任务，你需要有：

- 装有 Xcode 的 Mac
- 运行和测试应用的 iOS 设备

实验推荐解决方案

本实验涵盖了解决各种技术问题方法。并非所有这些方法对于每个应用都是必须的，具体要看应用中出现了哪类问题。不过，检验一下应用，让其在特定崩溃状况之前受到保护，是很有必要的。

任务 1 解决方案：选择一个项目管理软件来管理和跟踪 bug 和崩溃

- 1. 第一步是使用项目管理系统，它很好地集成有用于源码管理的崩溃处理系统。使用它是一个很健康的实践，确保 bug 能够被跟踪，团队也能很有效地管理源码。可用开源管理系统有不少。
- 2. 本实验中，你将使用 **Redmine**。你可以访问如下链接并下载软件：**www.redmine.org**，如图 1 所示。

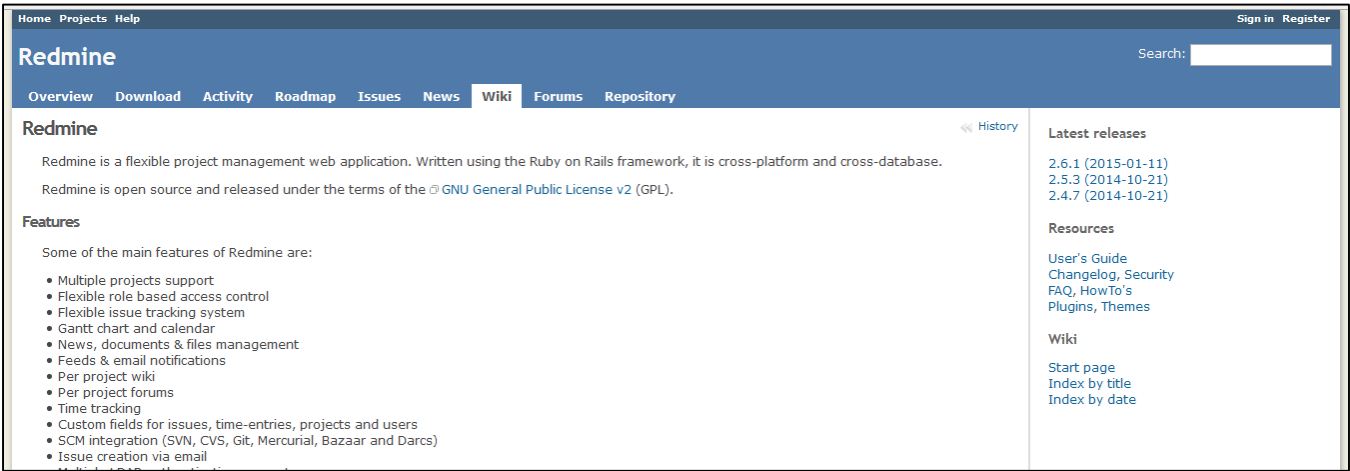


图 1: 下载 Redmine 的网站

- 3. 在 Redmine 上注册你的项目，这需要输入凭证，如图 2 所示：

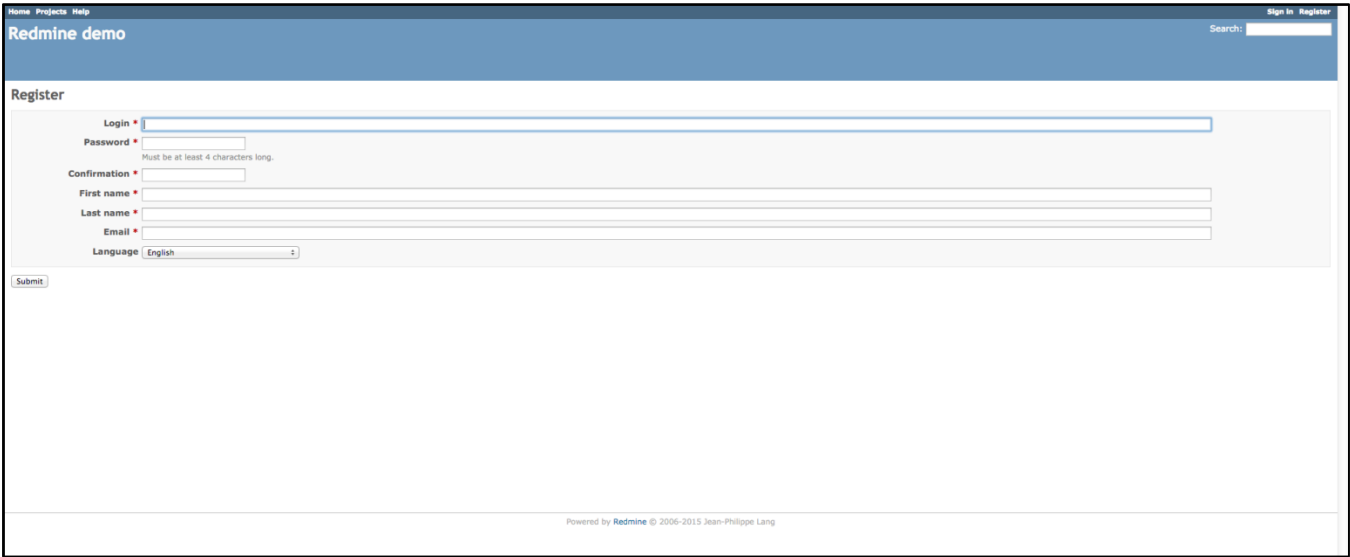


图 2: 在 Redmine 上注册项目

- 3. 在 Redmine 中创建一个新项目，如图 3 所示：

模块：使应用投入生产

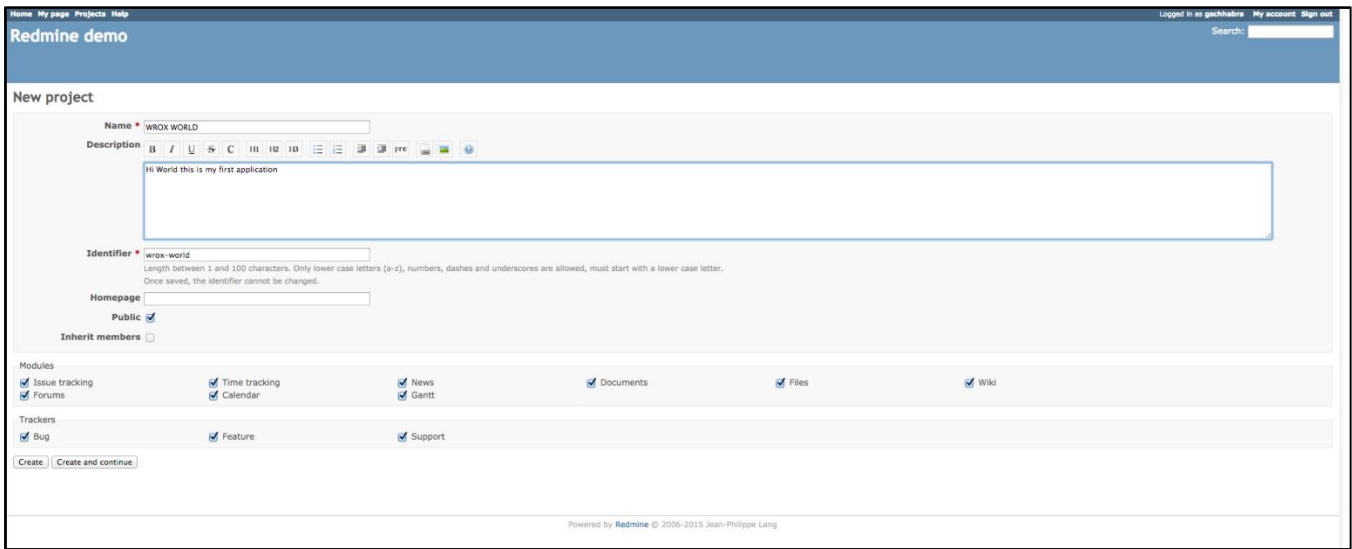


图 3: 在 Redmine 中添加项目

4. 创建项目时，你会得到如图 4 所示的界面，其中显示了项目的概况。

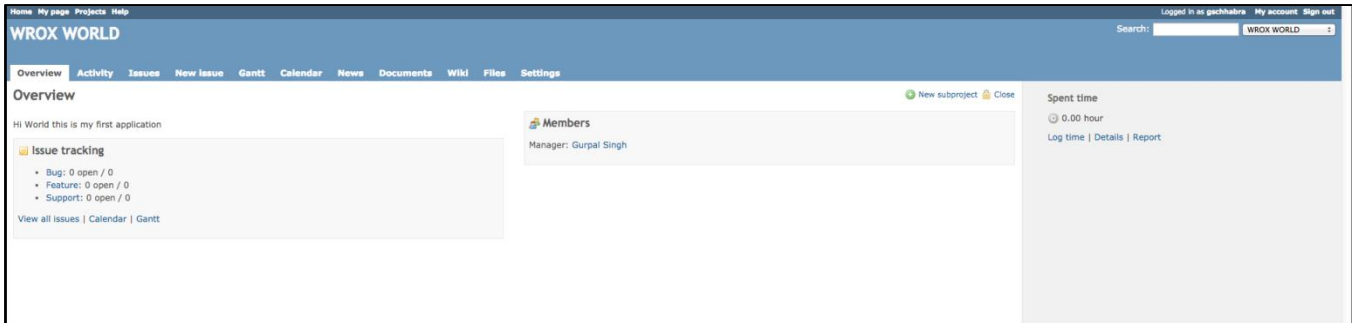


图 4: 空项目的概况

5. 在 **New Issue** 选项卡中，你可以汇报使用 bug 报告系统时所经历的问题，如图 5 所示：

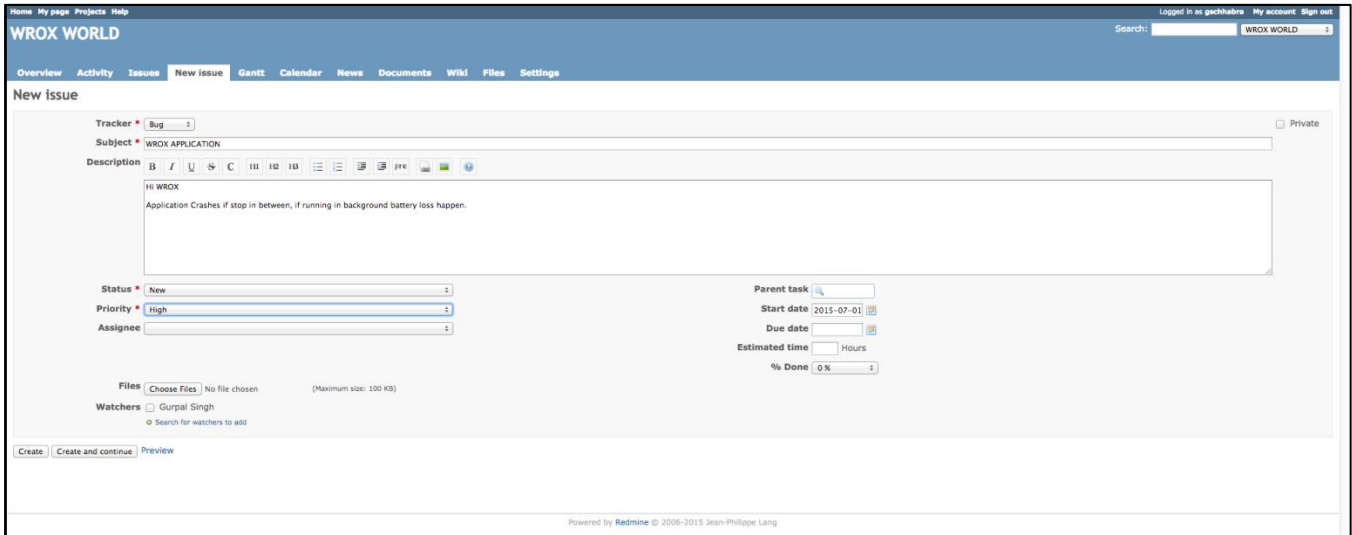


图 5: 使用 Redmine 网站来汇报 Bug

6. 在 **Wiki** 选项卡中，你需要创建一个 Wiki，通过它实现应用相关信息的分享，如图 6 所示：

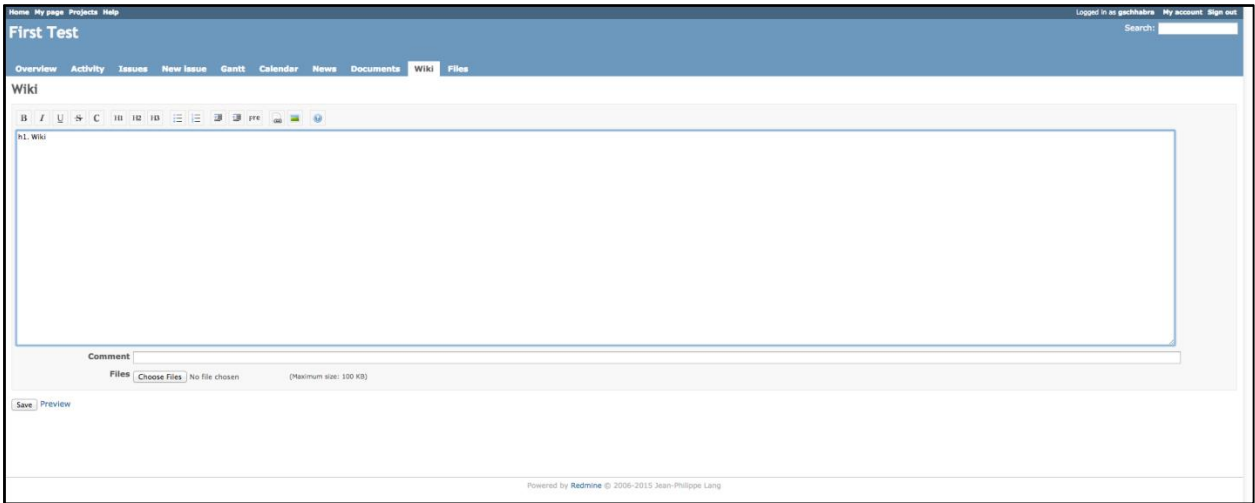


图 6: 创建项目 Wiki

7. 在 **Files** 选项卡，你可以上传整个开发团队中可以使用的的项目文件，如图 7 所示：

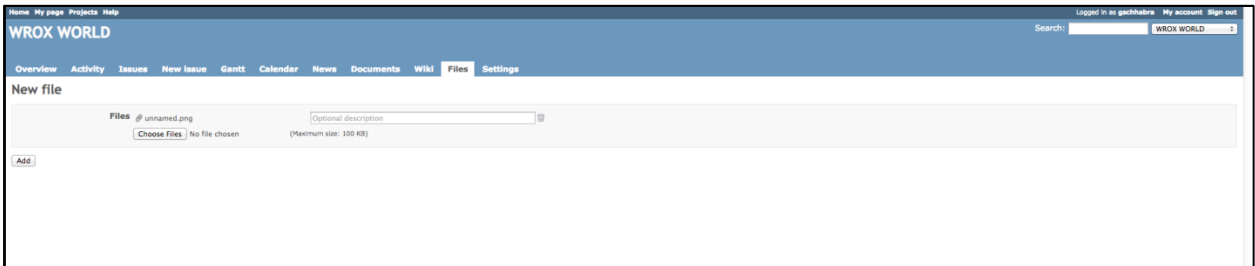


图 7: 在创建项目中添加文件

8. 在 **Calendar** 选项卡，你可以浏览日历来显示事件，这将能额外提供生产力。

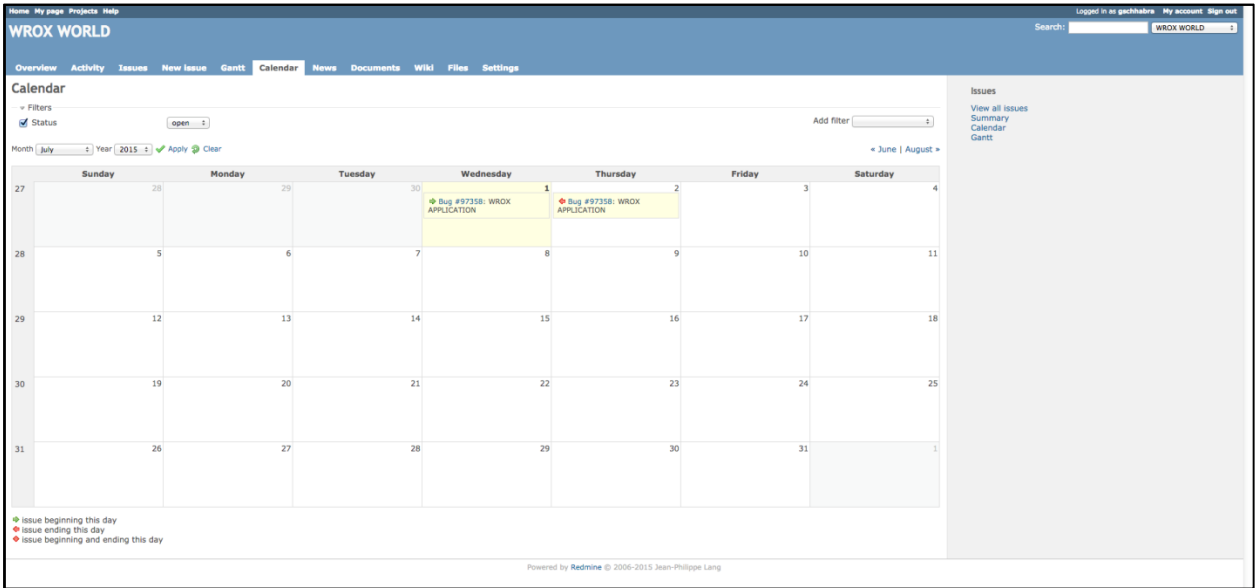


图 8: Redmine 中的日历特性

模块：使应用投入生产

9. 在 **Gantt** 选项卡中，你可以使用项目的 Gantt 图表，如图 9 所示：

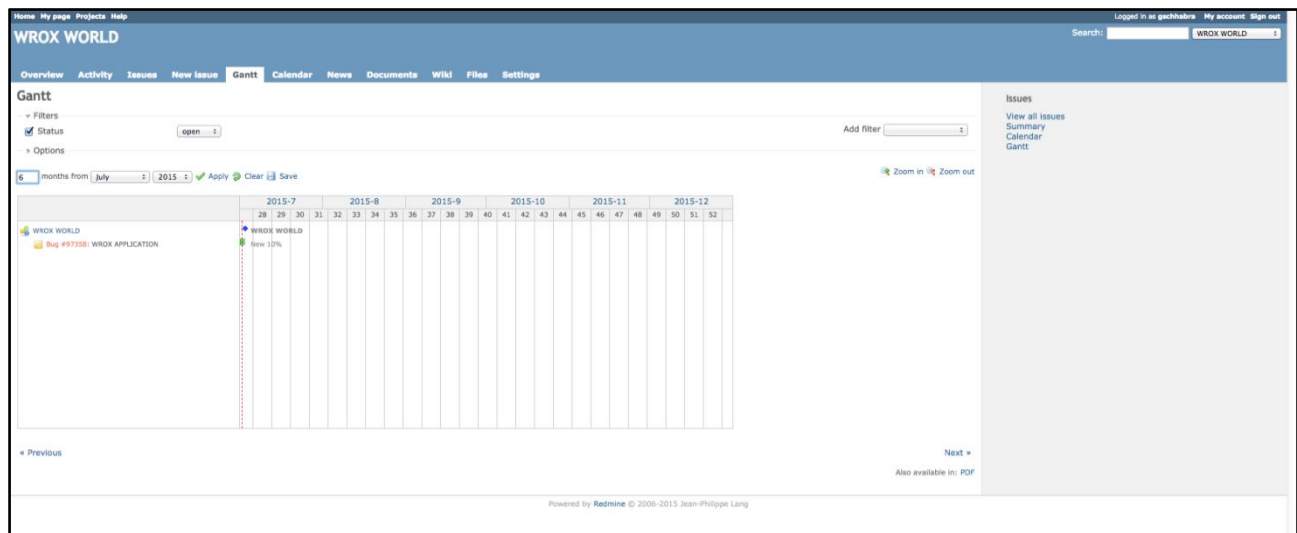


图 9: Redmine 中的 Gantt 特性

10. 在 **Issues** 选项卡，你可以汇报应用的问题，如图 10 所示。汇报的问题可以被整个开发团队看到。

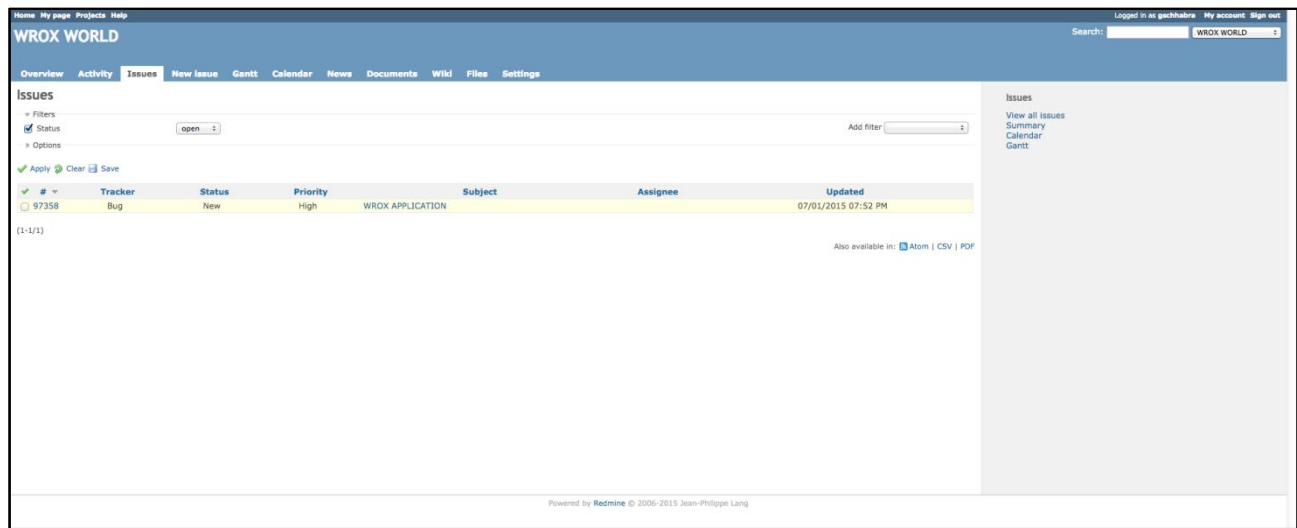


图 10: 项目中的问题

11. 最后，你可以浏览项目的概况，用于恰当的项目管理，如图 11 所示：

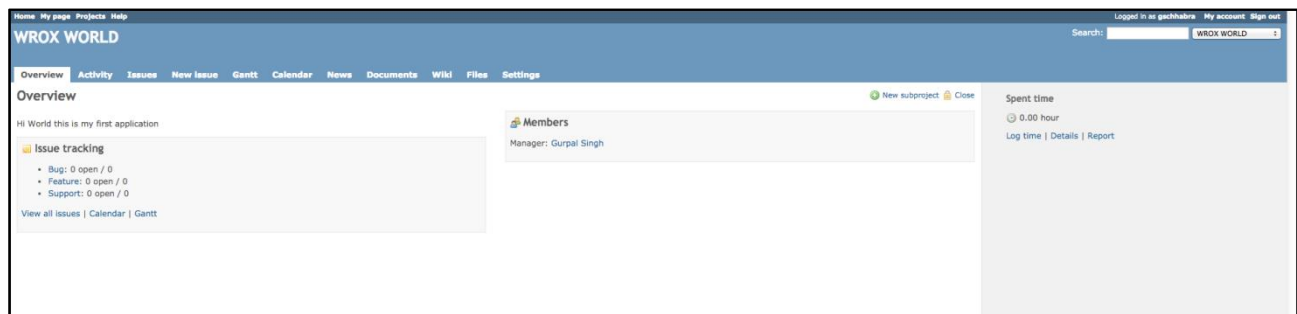


图 11: 项目概况

应用崩溃报告的创建位置是：~/Library/Logs/CrashReporter/MobileDevice/<设备名>。

你可以将收集的崩溃日志

你可以将收集的崩溃日志直接发送给 **Redmine**，做法是在系统中调用一个网络服务并创建一个崩溃类型的任务。如果你的应用在某位用户的设备上崩溃，你会立刻得知运行设备的信息、应用版本及崩溃日志。这将能够为你提供查找 **bug** 所需的所有信息，你可以依此来修正并发布新版应用。

任务 2 解决方案：阻塞主线程

分派队列被用于执行任何代码块，无论是同步还是异步。分派队列在这里被用于让主线程不被其它任务所阻塞从而无法执行，导致应用崩溃。

通过调用 `dispatch_get_global_queue` 函数创建一个 `dispatch_queue_t` 实例，这会返回一个特定优先等级的全局并发队列。调用 `dispatch_async` 函数来将代码块提交给队列。样本实现如下所示：

```
dispatch_queue_t bgQueue = dispatch_get_global_queue
    (DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0);
dispatch_async(bgQueue, ^{
    //perform your operation
});
```

除了调用 `dispatch_get_global_queue` 函数以外，你还可以通过调用 `dispatch_queue_create` 函数创建你自己的队列并传入队列的名称。

```
dispatch_queue_t bgQueue = dispatch_queue_create("net.yourdeveloper.app.queueName", nil);
dispatch_async(bgQueue, ^{
    //perform your operation
});
```

任务 3 解决方案：内存泄露

假设你正从另一位开发者手中接手一个项目。该项目的程序组织非常糟糕，而且很容易崩溃。由于应用功能复杂却编写糟糕，你需要花很多时间才能找到所有的内存泄露问题。你决定试用 **自动引用计数（ARC）** 并重新开发应用。使用 **ARC** 能够让你复制粘贴绝大部分代码，所有 **release** 和 **retain** 语句则需要删除。

ARC 为 **Objective-C** 对象和块实现自动内存管理，无需程序员手动插入 **retain** 和 **release**。它不提供循环收集器；用户必须明确管理对象的生命周期，手动或使用弱或不安全引用断开循环。当两个对象相互引用并被保留时，它会创建一个保留循环，因为两个对象都尝试相互保留，让释放不能成功。

要在代码中启用 **ARC**，到 **Edit -> Refactor -> Convert to Objective-C ARC**，如图 12 所示：

模块：使应用投入生产

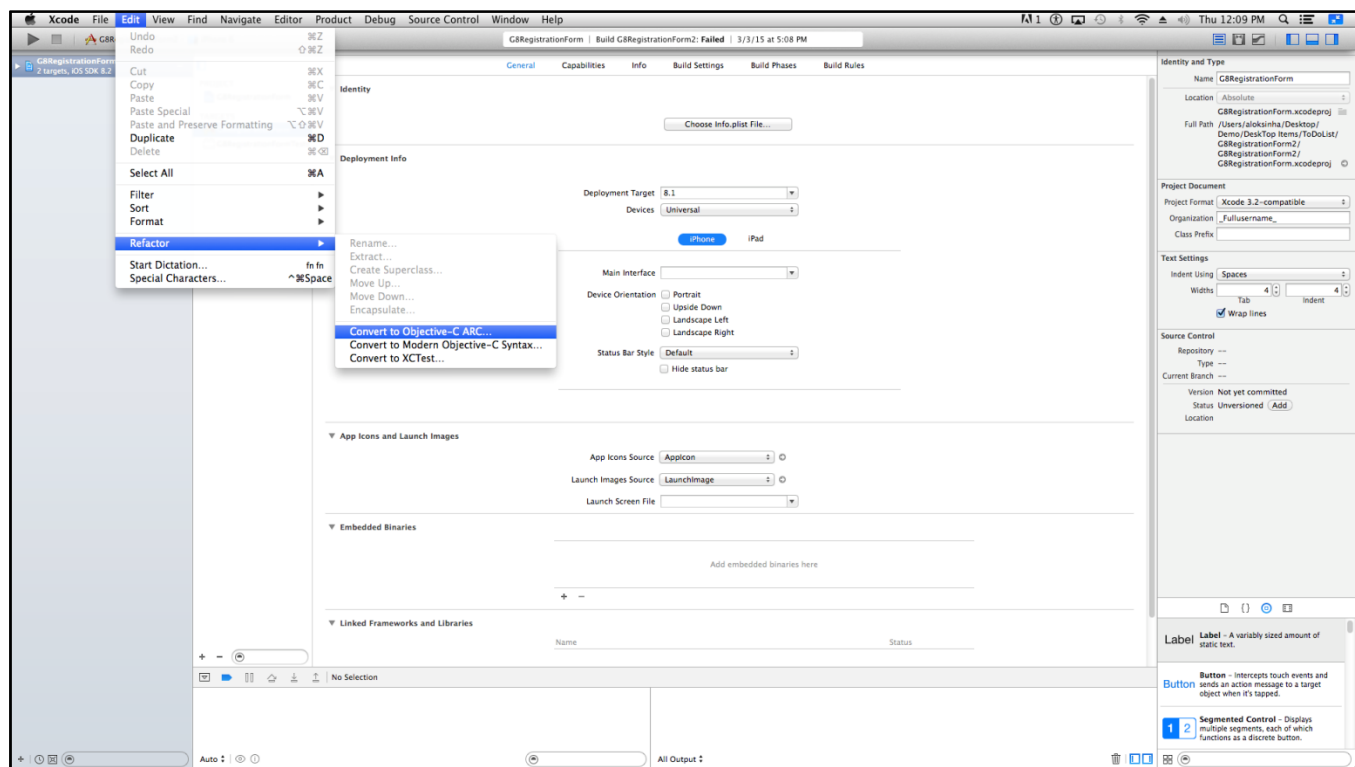


图 12: 在代码中启用 ARC 的做法

在 **Select Targets to Convert** 对话框中，点 **Check** 按钮看代码有没有被转换。黄色警告标志表示代码已经转换，如图 13 所示：

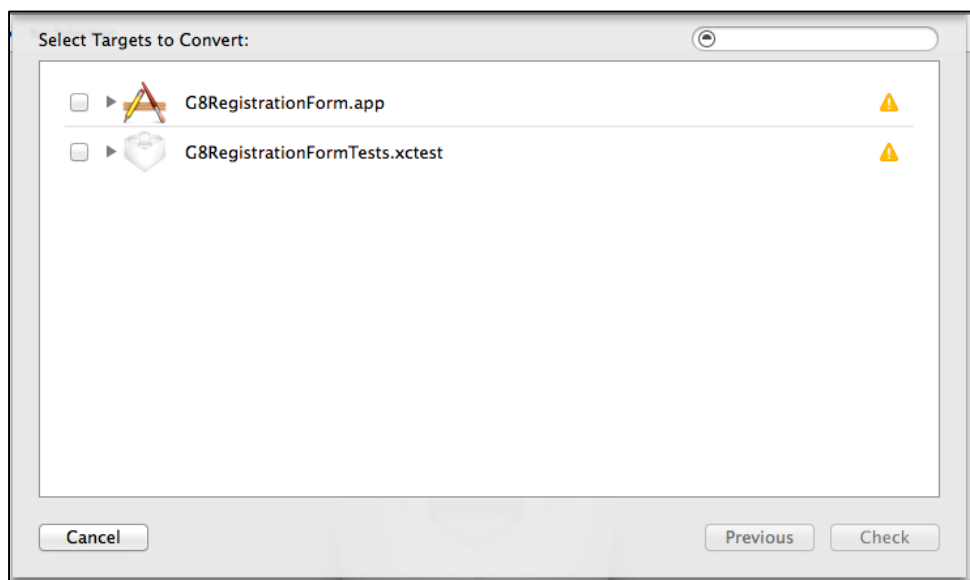


图 13: 检查代码有没有被转换

任务 4 解决方案：同步 HTTP 请求和流量消耗过大

应用同网络服务通信时需要小心。不要每次用户启动应用都全部重载所有数据集。有些接收到的远程信息可以使用 **Core Data** 存储起来。另一种很不错的改进是检查用户设备连的是 **Wi-Fi** 还是 **3G**、**4G** 等手机网络。

1. 苹果开发了一个样本应用，名为 **Reachability**，它演示了如何使用 `SystemConfiguration` 框架来监控设备网络状态。你可以直接从 <https://developer.apple.com/library/ios/samplecode/Reachability/Introduction/Intro.html> 下载 **Reachability** 应用：

Reachability 应用包含一个 `Reachability.h` 文件和一个 `Reachability.m` 文件，你可以在应用中免费使用它们。

注：`Reachability` 类同 `ARC` 不兼容，这就意味着如果你在一个 `ARC` 项目中使用这个类，你就必须在项目中为 `Reachability.m` 文件设置 `-fno -objc -arc` 编译器标志。

2. 打开 `YAppDelegate.h` 文件并导入 `Reachability` 头文件。创建一个 `Reachability` 类型的强属性，名为 `reachability`，以及一个公共方法，名为 `connectedViaWiFi`，如下所示：

```
#import <UIKit/UIKit.h>
#import "Reachability.h"
@class YDViewController;
@interface YAppDelegate : UIResponder<UIApplicationDelegate>
@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) YDViewController *viewController;
@property (strong, nonatomic) Reachability* reachability;
-(BOOL)connectedViaWiFi;
@end
```

3. 打开 `YAppDelegate.m` 文件，创建和初始化 `reachability` 实例。调用 `Reachability` 类的 `startNotifier` 方法。实现 `connectionViaWiFi` 方法，做法是返回 `currentReachabilityStatus` 等于 `reachability` 实例的 `ReachableViaWiFi` 常量。`YAppDelegate.m` 实现的主要部分如下所示：

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds];
    self.reachability = [Reachability reachabilityForInternetConnection];
    [self.reachability startNotifier];
    [self.reachability currentReachabilityStatus];

    // Override point for customization after application launch.
    self.viewController = [[YDViewController alloc]
initWithNibName:@"YDViewController" bundle:nil];
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}

-(BOOL)connectedViaWiFi
{
    return [self.reachability currentReachabilityStatus] == ReachableViaWiFi;
}
```

4. 使用界面生成器和辅助编辑器打开 `YDViewController.xib` 文件，创建一个简单用户界面，使用 `UILabel` 对象来显示网络连接状态，如图 14 所示：

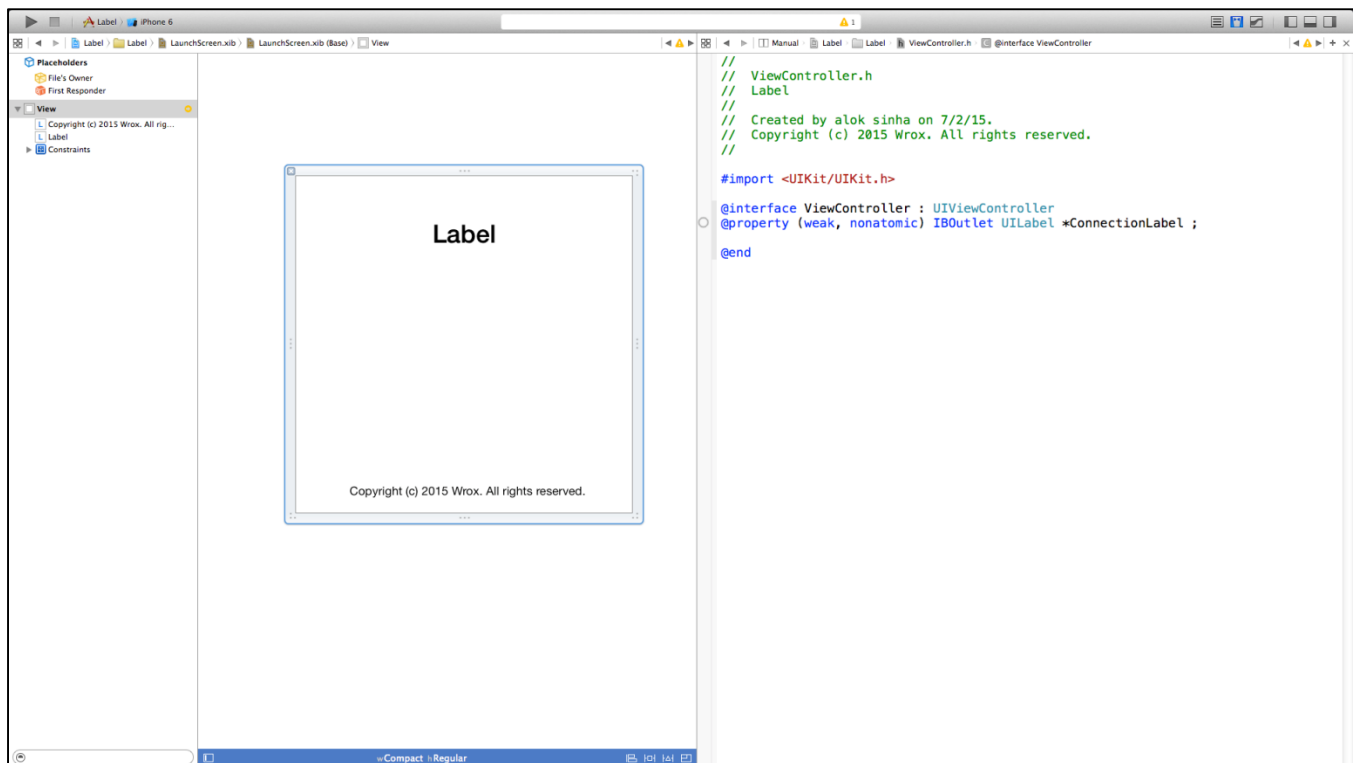


图 14: 创建一个用户界面并使用 UILabel 对象

5. 打开 YDViewController.m 文件并导入 YAppDelegate 头文件。在 viewDidLoad 方法中，创建一个 YAppDelegate 类的实例，命名为 appDelegate。网络状态变化时，它会广播一个通知，名为 kReachabilityChangedNotification。为这一通知创建一个观测器，以名为 networkStatusChanged 的方法为标的。
6. 要测试网络状态，你只需要调用 appDelegate 实例的 connectedViaWiFi 方法，并使用返回值来实现你的函数逻辑。完整 YDViewController.m 文件如下所示：

```
#import "YDViewController.h"
#import "YAppDelegate.h"
@interface YDViewController ()

@end

@implementation YDViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    YAppDelegate* appDelegate = (YAppDelegate *)[[UIApplication
    sharedApplication] delegate];

    [[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(networkStatusChanged:)
    name:kReachabilityChangedNotification
    object:appDelegate.reachability];
}
```

```

if ([appDelegateconnectedViaWiFi])
self.connectionLabel.text = @"Connected via WIFI";
else
self.connectionLabel.text = @"NOT Connected via WIFI";
}
- (void)networkStatusChanged:(NSNotification*)notification {
//you know the network status has changed so perform your action here
}
- (void)didReceiveMemoryWarning
{
[superdidReceiveMemoryWarning];
// Dispose of any resources that can be recreated.
}
@end

```

任务 5 解决方案：耗电

要在应用中执行耗电量的技术分析，你可以在 AppDelegate 中创建和初始化 CLLocationManager 实例。这是为了确定用户位置，以使用特定语言本地化你的应用。创建和初始化 CLLocationManager 类的实例，设置委托并在 application:didFinishLaunchingWithOptions:方法中调用 startUpdateLocation 方法。

1. 实现 locationManager:didUpdateToLocations:委托方法并在 CLLocation 属性中存储 [locations lastObject]值，这可以用于分析用途。打开 BatteryDrainer 项目并添加 CoreLocation 框架到项目。打开 YAppDelegate.h 文件并做出如下改变：

```

#import <UIKit/UIKit.h>
#import "Reachability.h"
#import <CoreLocation/CoreLocation.h>
@class YDViewController;

@interface YAppDelegate : UIResponder<UIApplicationDelegate, CLLocationManagerDelegate>
@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) YDViewController *viewController;
@property (nonatomic, strong) CLLocationManager* locmanager;
@property (nonatomic, strong) CLLocation *userlocation;
@property (strong, nonatomic) Reachability* reachability;
- (BOOL)connectedViaWiFi;
@end

```

2. 打开 DAppDelegate.m 文件并实现 CLLocationManager 逻辑，如下所示：

```

#import "YAppDelegate.h"
#import "YDViewController.h"
@implementation YAppDelegate
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
self.window = [[UIWindowalloc] initWithFrame:[UIScreen mainScreen] bounds]];
self.locmanager = [[CLLocationManageralloc] init];

```

```
self.locmanager.delegate=self;
[self.locmanagerstartUpdatingLocation];

self.reachability = [Reachability reachabilityForInternetConnection];
[self.reachabilitystartNotifier];
[self.reachabilitycurrentReachabilityStatus];

// Override point for customization after application launch.
self.viewController = [[YDViewControlleralloc]
initWithNibName:@"YDViewController" bundle:nil];
self.window.rootViewController = self.viewController;
[self.windowmakeKeyAndVisible];
return YES;
}

-(BOOL)connectedViaWiFi
{
return [self.reachabilitycurrentReachabilityStatus]== ReachableViaWiFi;
}

#pragma mark CoreLocation
- (void)locationManager:(CLLocationManager *)manager
didUpdateLocations:(NSArray *)locations
{
self.userlocation=[locations lastObject];
[self.locmanagerstopUpdatingLocation];
}
```