构建 NetworkExtension 应用（二）

NetworkExtension

15 Nov 2017

1 Comment

前言

之前介绍了关于科学上网的一些知识，这章会先介绍下 NetworkExtension，以及相关的一些 iOS 平台的开源项目。最后再开始我们自己的项目。

实际上，我们自己的 NetworkExtension 应用，其实就是扮演 SS-Local 的角色。

NetworkExtension 相关

NetworkExtension是苹果提供的用于配置 VPN 和定制、扩展核心网络功能的框架。NE 框架提供了可用于定制、扩展 iOS 和 MacOS 系统的核心网络功能的 API。

Network Extension 最早出现在 iOS 8，不过那个版本不支持虚拟网卡，只能简单调用 iOS 系统自带的 IPSec 和 IKEv2 协议的 VPN。在 iOS 9 中，开发者可以用 NETunnelProvider 扩展核心网络层，从而实现非标准化的私密VPN技术。最重要的两个类是

`NETunnelProviderManager` 和 `NEPacketTunnelProvider`。

Potatso 便是使用 NE 框架实现了 Shadowsocks 代理，遗憾的是由于种种原因作者删除了开源代码。GitHub 上有不少人维护了其分支，但也都更新很慢，最近发现的一个可运行版本是这个，但我之前升级了 Xcode 9，所以也要进行一系列改动。最后终于改出一个可在 Xcode 9 上编译运行的版本，但是也并没有改动的很完美。大家凑合学习吧。

NEKit 相关

通过 Potatso 学习 Network Extension，对于初学者来说不太友好，毕竟项目很久不维护了。还有个更简单的方案，这要多亏了 NEKit 框架。NEKit 甚至可以不依赖 Network Extension framework（当然我们构建的项目是需要的）。有个 demo 可以看下。

初始化项目

新建项目

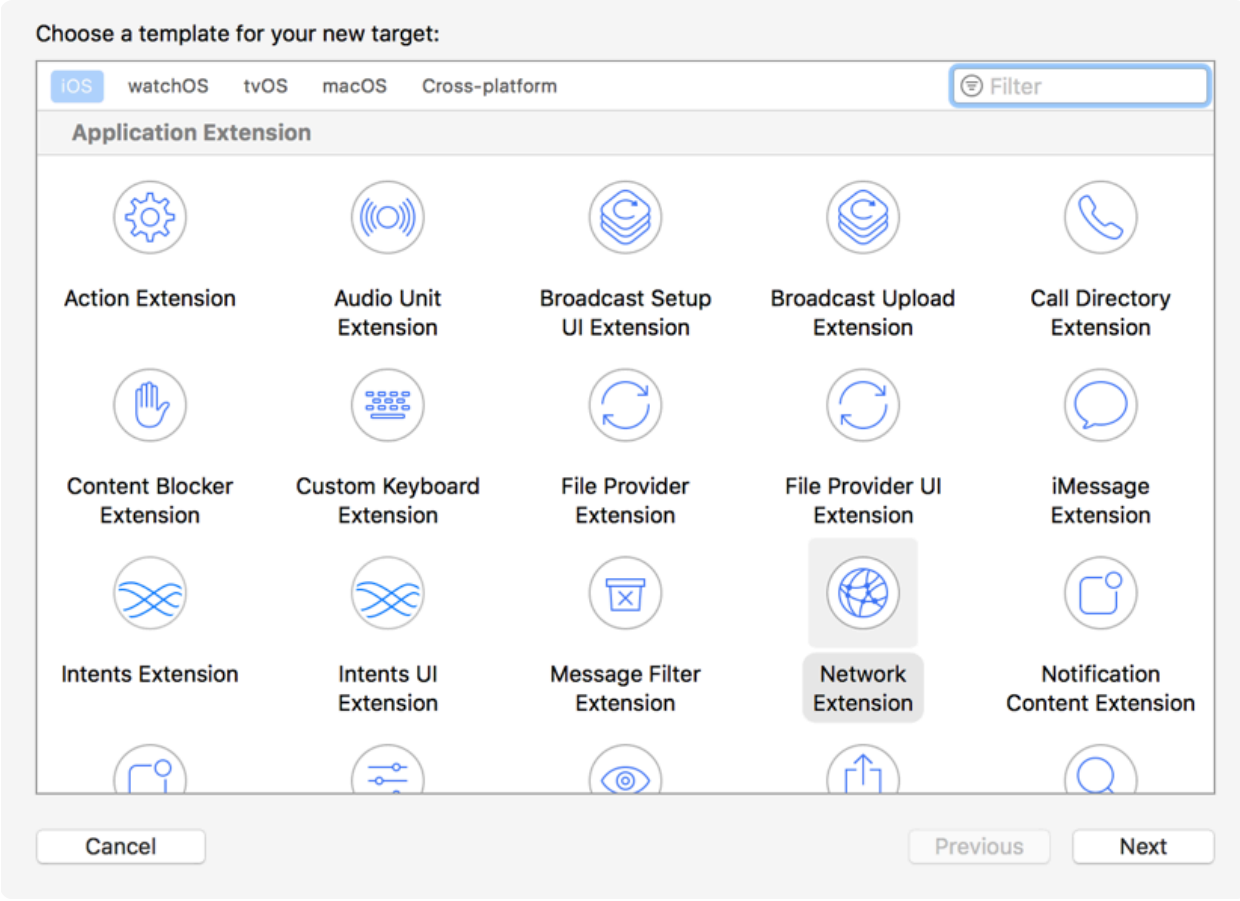
建立普通 Swift 项目 QLadder(此项目后来也将作为我们 iOS 部门内部翻墙用的客户端，所以用了企业证书)。

项目支持的最低 iOS 版本是 9.3，因为之前我改过一次 9.0，会有问题。

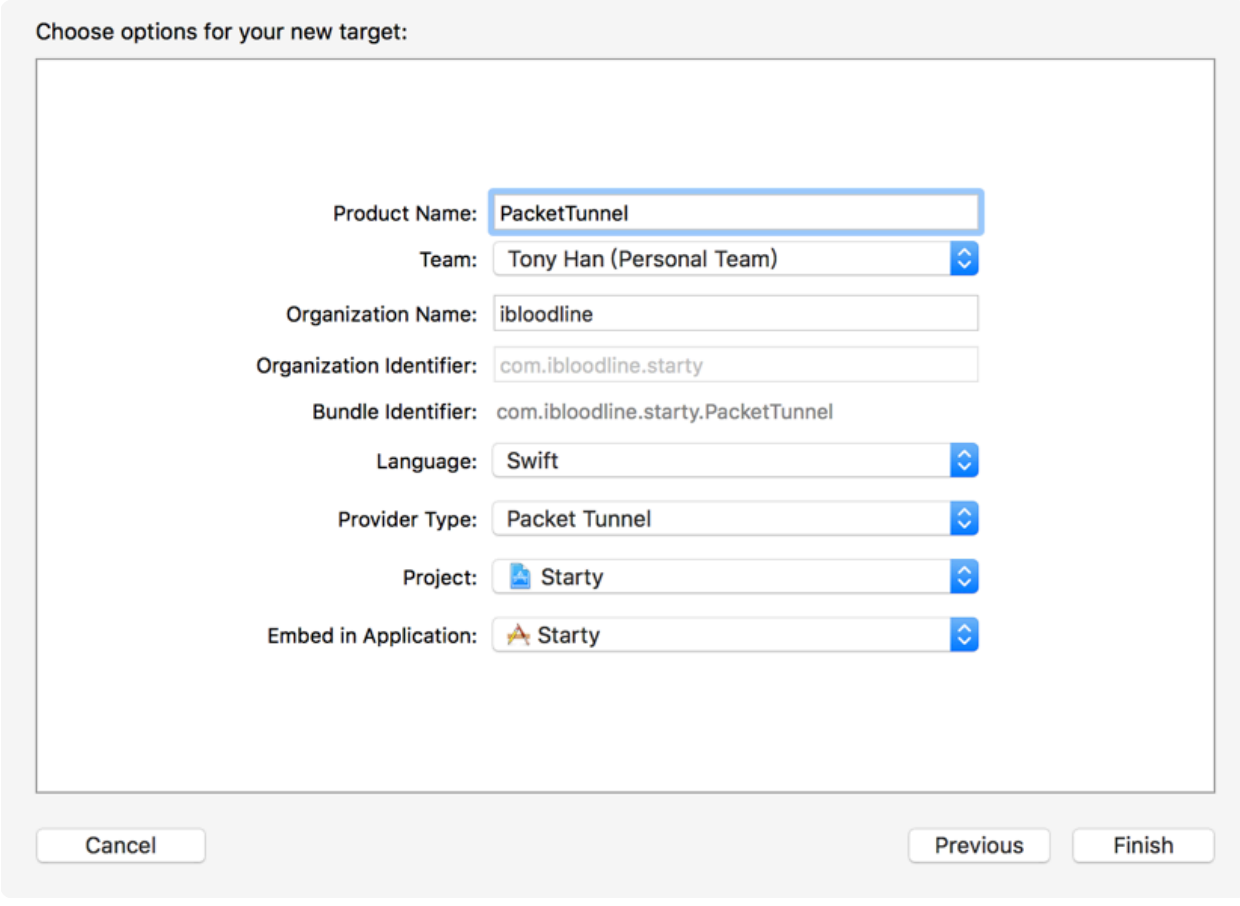
还有，Network Extension 无法在模拟器上调试。同时，你得有开发者账号，用来申请相关 Capabilities。

新建 PacketTunnel

新建 Target，选择 Network Extension。



然后选择 Provider Type 为 PacketTunnel。



申请 entitlements

如果 containing app 要与 extension 共享数据，则必须要

开启 App Groups。

Personal VPN 和 Network Extensions（App Proxy、Content Filter、Packet Tunnel）也当然要开启。

第三方框架

NEKit 推荐将项目拖入工程，或者使用 Carthage 集成。

其他的第三方框架使用 Pod：

- SwiftColor
- CocoaLumberjack/Swift
- Alamofire

代码

`NETunnelProviderManager`

上面提到了两个核心类，其中一个是，`NETunnelProviderManager`，它和 vpn 设置是一一对应关系。如果 App 有两个 vpn 设置，我们通过代码就能得到两个 `NETunnelProviderManager` 实例。我们要通过代码，对 `NETunnelProviderManager` 做四种操作。

1. 创建 vpn 配置

```
fileprivate func createProviderManager() -> NETunnelPr
    let manager = NETunnelProviderManager()
    manager.protocolConfiguration = NETunnelProviderPr
    return manager
}
```

1. 保存 vpn 配置

```
manager.saveToPreferences {
    if let error = $0 {
        complete(nil, error)
    } else {
        manager.loadFromPreferences(completionHandler:
            if let error = error {
                complete(nil, error)
            } else {
                complete(manager, nil)
            }
        })
    }
}
```

这段代码执行时会请求用户的授权，允许之后会添加一份 vpn 的配置。

1. 开启和关闭 vpn

```
fileprivate func startVPNWithOptions(_ options: [Strin
    // Load provider
    loadAndCreateProviderManager { (manager, error) ->
        if let error = error {
            complete?(nil, error)
        } else {
            guard let manager = manager else {
                complete?(nil, ManagerError.invalidPro
                return
            }
            if manager.connection.status == .disconnec
                do {
                    try manager.connection.startVPNTun
                    self.addVPNStatusObserver()
                    complete?(manager, nil)
                }catch {
                    complete?(nil, error)
                }
            } else {
                self.addVPNStatusObserver()
                complete?(manager, nil)
```

```
            }
        }
    }
}

public func stopVPN() {
    // Stop provider
    loadProviderManager {
        guard let manager = $0 else {
            return
        }
        manager.connection.stopVPNTunnel()
    }
}
```

1. 监听并更新 vpn 状态

```
/// 添加vpn的状态的监听
func addVPNStatusObserver() {
    guard !observerDidAdd else {
        return
    }
    loadProviderManager {
        if let manager = $0 {
            self.observerDidAdd = true
            NotificationCenter.default.addObserver(fo
                self.updateVPNStatus(manager)
            })
        }
    }
}

/// 更新vpn的连接状态
///
/// - Parameter manager: NEVPNManager
func updateVPNStatus(_ manager: NEVPNManager) {

    switch manager.connection.status {
    case .connected:
        self.vpnStatus = .on
    case .connecting, .reasserting:
```

```
            self.vpnStatus = .connecting
        case .disconnecting:
            self.vpnStatus = .disconnecting
        case .disconnected, .invalid:
            self.vpnStatus = .off
        }
    }
```

`NEPacketTunnelProvider`

`NEPacketTunnelProvider`，是真正的 vpn 核心代码。项目中 `PacketTunnelProvider` 是其子类，并且以下两个方法必须实现。

```
@available(iOS 9.0, *)
open func startTunnel(options: [String : NSObject]? =

@available(iOS 9.0, *)
open func stopTunnel(with reason: NEProviderStopReason
```

当 App 里的 `NETunnelProviderManager` 对象调用 `startVPNWithOptions` 时，控制流程就跳到 Extension 里的 `startTunnel` 方法。

`startTunnel` 有两个参数：options 是个字典，从 App 里传过来，具体有啥信息完全由开发者自己定义。completionHandler 是个闭包回调，由系统提供。我们可以将其保存到某个变量，待 vpn 启动完成时主动调用。

`stopTunnel` 也有两个参数：`reason` 表示 vpn 被关闭的理由。iOS 预先定义了一组 `NEProviderStopReason` 常数，但实际应用时，`reason` 基本不用。`completionHandler` 的用法同 `startTunnel` 第二个参数一样，不再重复。

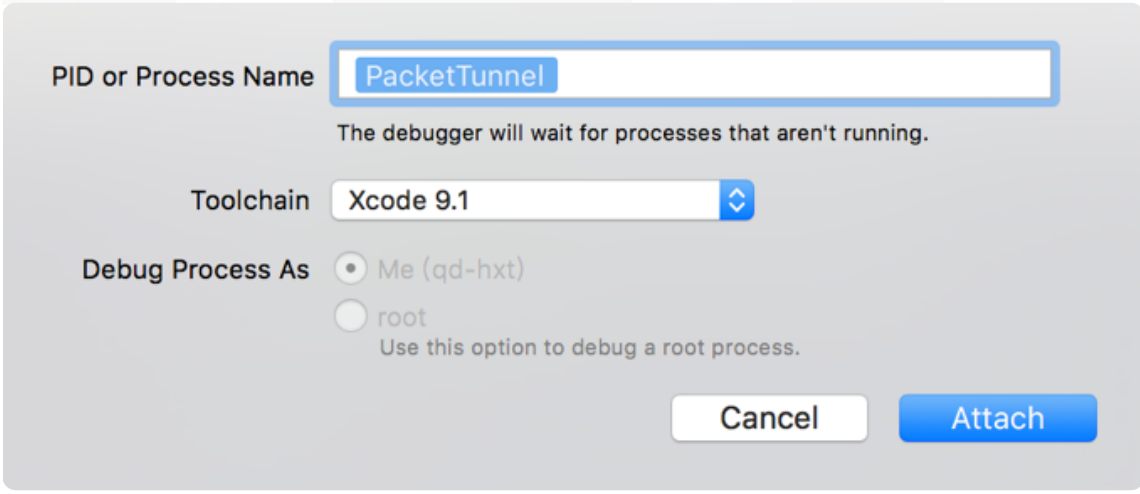
这里的代码基本是参考自 Potatso 、Specht 以及

RabbitVpnDemo。

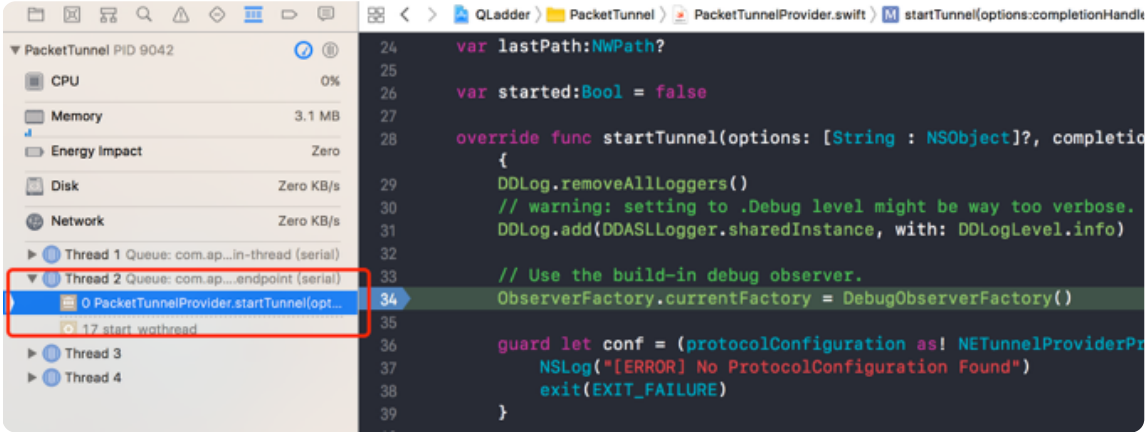
调试 Network Extension

调试 App 的代码很简单，但是如何调试 Extension 中的代码呢？在已经完成 PacketTunnel 的代码情况下：

1. 构建并运行应用。
2. 停止运行。
3. Xcode 菜单中 `Debug -> attach to process by PID or name`，填入 `PacketTunnel`，然后 `Attach`。



4. 在手机上运行（不要通过 Xcode）应用，点击连接的时候，进入了断点。



代码：

文章中的代码都可以从我的GitHub `QLadder` 找到。