本地和推送通知编程指南

原著：Apple Inc.

翻译：謝業蘭 【老狼】

联系：xyl.layne@gmail.com

鸣谢：有米移动广告平台

CocoaChina 社区

目录

关于本地通知和推送通知

本地通知和推送通知是没有运行在前台的应用程序可以让它们的用户获得相关消息通知的方式。消息通知可能是一条消息，即将发生的日历事件，或远程服务器的新数据。当被操作系统显示时，本地通知和推送通知看起来一样。它们可以显示一个警告信息或在应用程序的图标上面显示一个徽标。它们也可以在警告窗或徽标显示时播放一段声音。

推送通知是在 iOS 3.0 和 Mac OS X v7.0 之后引入的。本地通知是在 iOS 4.0 之后引入的。它们都不支持 Mac OS X。

当用户被通知相应的应用程序有消息，事件，或其他数据时，他们可以启动该应用程序并查看详情。他们也可以选择忽略通知，此时应用程序没有被激活。

注意：*推送通知和本地通知与广播通知（NSNotificationCenter）或键-值观察通知没有相关性。*

概览

本地通知和推送通知有几个重要的方面你需要了解的。

本地通知和推送通知所解决的问题

任何时候前台都只能有一个应用程序处于活跃状态。许多应用程序运行在一个基于时间或相关的环境里面，此时用户感兴趣的事件可能在应用没有在前台的时候发生。本地通知和推送通知允许这些应用程序来通知它们的用户告知这些事件的发生。

相关章节：*本地通知和推送通知的深度*

本地通知和推送通知起源不同

本地通知和推送通知为不同的需求而设计的。本地通知是本地 iPhone、iPad、或 iPod touch 上面的应用发起的。相反推送通知（又称远程通知）是从其他设备上面到达的。它来自一个远程设备——应用程序的提供者——并在有新的消息需要查看或新的数据需要下载的时候被推送到本地设备上面的应用（通过苹果的推送通知服务——Apple Push Notification service）。

相关章节：*本地通知和推送通知的深度*

调度一个本地通知，注册一个推送通知，并处理它们

为了让 iOS 在之后发送一个本地通知，应用程序需要创建一个 UILocalNotification 对象，并给它设置发送的时间和日期，指定细节，并调度它。为了接收一个推送通知，应用程序必须注册接收通知并把它从操作系统获得的一个设备令牌环传递给它的通知提供者。

当操作系统发送一个本地通知（只有 iOS 支持）或推送通知（iOS 或 Mac OS X 都支持）时，如果此时应用程序没有运行在前台，它将会显示通知（警告窗、图标徽标、声音）。如果显示一个通知警告窗并且用户轻击或单击动作按钮（或移动动作滑块），相应的应用程序将会加载启动并调用本地通知对象或远程通知载体传递的相应方法。如果通知到达的时候应用程序正运行在前台，应用程序的委托接收一个本地或推送通知。

相关章节：*调度、注册、和处理通知*

苹果推送通知服务(APNs)是推送通知的网关

苹果推送通知服务（APNs）负责把通知消息传递到已经有应用注册监听这些通知的设备上面。每个设备和 APNs 建立一个认证和加密的 IP 连接，并接收这一持久连接的通知。当监听到有需要发送到这些客户端设备的通知到来时，通知提供者和 APNs 通过一个持久和安全的通道建立连接。当新的数据到达时，提供者准备并通过这条和 APNs 之间的通道把通知发送到设备，即把通知推送到目标设备上面。

相关章节：*苹果推送通知服务（APNs）*

为推送通知获取授权

为了给提供者这边开发和配置推送通知，你必须从开发者中心（即苹果官网 Dev Center）取得 SSL 授权证书。每个授权证书限制必须对应一个单独的应用，由应用的 Bundle ID 标识。而且该授权证书也被限制用于以下两个环境之一，沙箱环境（用于开发和测试）和生产环境。这些环境都拥有它自己的 IP 地址，而且需要它们自己的授权证书。你必须同时获得这些环境的的配置文件（即 Provisioning profiles）。

相关章节：*配置和开发*

提供者和APNs之间通过二进制接口通信

二进制接口是异步的，而且它使用 TCP 方式通过 sock 连接把二进制内容的推送通知发送给 APNs。沙箱和生产环境都有自己独立的接口，每个都有它自己的地址和端口。对于每个接口，你需要使用 TLS(或 SSL)和已经拿到的 SSL 授权证书来建立一条到 APNs 的安全通信通道。提供者把推送通知打包并通过该通道发送给 APNs。

APNs 包含了一个反馈服务，它负责维护每个应用的传递通知失败的设备列表（即 APNs 当前无法把推送通知传递到这些设备上面的对应的应用）。提供者应该周期性的连接到反馈服务来查看推送失败的设备以便它可以把之前失败的通知重复发送过去。

相关章节：*苹果推送服务（APNs）、提供者和 APNs 通信*

先决条件

对于本地通知和推送通知的客户端方面的实现，你应该对 iOS 的应用开发很熟悉。

对于推送的提供者（即服务器端）方面的实现，你应该具备 TLS/SSL 和 sock 知识有所了解。

其他参考资料

以下相关资源也包含了部分本地通知和推送通知实现的额外介绍。

- UILocalNotification,UIAppliation 和 UIApplicationDelegate 的参考文档介绍了本地通知和推送通知的 iOS 客户端相关 API。
- NSApplication 和 NSApplicationDelegate Protocol 的参考文档介绍了推送通知在 Mac OS X 上面的相关 API。
- 安全预览（Security Overview）描述了iOS及Mac OS X平台上面使用到的安全证书和相关技术。
- RFC 5246介绍了TLS协议的标准。

数据提供者和苹果推送服务(APNs)之间的安全通信需要对 TLS(Transport Layer Security--传输层安全)或它的前身 SSL(Secure Sockets Layer--安全套接字层)有所了解。请参阅这些加密协议有关详细描述的更多信息。

第一章　本地和推送通知深度

本地通知和推送通知的主要目的是让一个应用程序可以在一个应用程序没有运行在前台的时候通知它的用户它有重要的信息要告知它们，比如一条消息或一个到期的约会等。本地通知和推送通知的主要区别很简单：

- 本地通知由同一设备上面的应用自己调度和传递。

  本地通知只在 iOS 上面可用。

- 推送通知（又称远程通知）由应用的远程服务器（即提供者）发送给苹果推送服务（APNs），它负责把推送通知发送安装了该应用的设备上。

  推送通知在 iOS 和 Mac OS X v10.7(Lion)之后都支持。

  以下各部分介绍本地通知和推送通知的相同点并比较它们的不同点。

注意：*关于 iOS 上面的推送和本地通知的使用说明，参阅 iOS 人机交互指南（iOS Human Interface Guidelines）的"启用推送通知"部分。*

1.1　推送和本地通知显示效果一样

从用户角度来看，推送通知和本地通知显示相同的内容。因为它们的目的都是为了在应用当前没有运行在前台的时候告知一个应用的使用者他会感兴趣的信息。

让我们假设你正使用你的 iPhone 来拨打电话，网上冲浪，听音乐。你有一个国际象棋的应用安装在你的 iPhone 上面，你决定和你遥远的朋友进行一场象棋游戏。你走了第一步（充分考虑游戏提供者），然后你退出了客户端程序来查收邮件。此时你的朋友等待你走棋。象棋应用提供者已经知道该走步，并知道当前象棋应用在你的设备上面不在活跃，它会发送一个推送通知到苹果推送服务（APNs）。

很快你的设备，或者确切说是的设备的操作系统通过 Wi-Fi 或蜂窝网络连接到 APNs 接收到该推送通知。因为当前象棋应用没有运行，iOS 会显示一个提升窗类似图 1-1 所示。该消息包含了应用的名称，一个短消息，和两个按钮（在该例中）:Close 和 View。右边的按钮又称为动作按钮，它默认的标题是"View"。应用可以自定义该动作按钮的标题，而且可以国际化该按钮标题和消息内容来适应你用户的偏好语言。

Figure 1-1 A notification alert

如果单击 View 按钮，将会加载启动象棋应用，并连接到游戏提供者，下载最新的数据和调整界面来显示当前到你朋友走棋。（点击 Close 按钮会隐藏提示窗）

Mac OS X 注意：*当前在 Mac OS X 上面非运行的应用的可用推送消息类型仅为徽标。换而言之，如果应用当前没有运行它只会在 Dock 里面相应的应用打上徽标。如果用户没有把应用的图标显示在 Dock 上面，系统会自动的把应用图标插入到 Dock 里面以便它可以给它打上徽标（当程序再次退出的时候会从 Dock 里面自动删除）。运行的应用程序可能核查其他类型的通知载体（提示和声音），并相应的处理它们。*

让我们考虑其他要求类型的应用，有一个应用管理 To-Do 列表的，而且列表的每个条目都有一个该条目需要完成的截止日期和时间。用户可以让应用在过期日期之前的特点时间通知他们。为了使之更高效，该应用调用一个本地通知到该日期和时间。以其显示一个提示消息，此刻该应用旋转指定一个特定的徽标数字（1）。在合适的时间，iOS 会在应用图标的右上角显示该徽标数字，如图 1-2 所示那样。

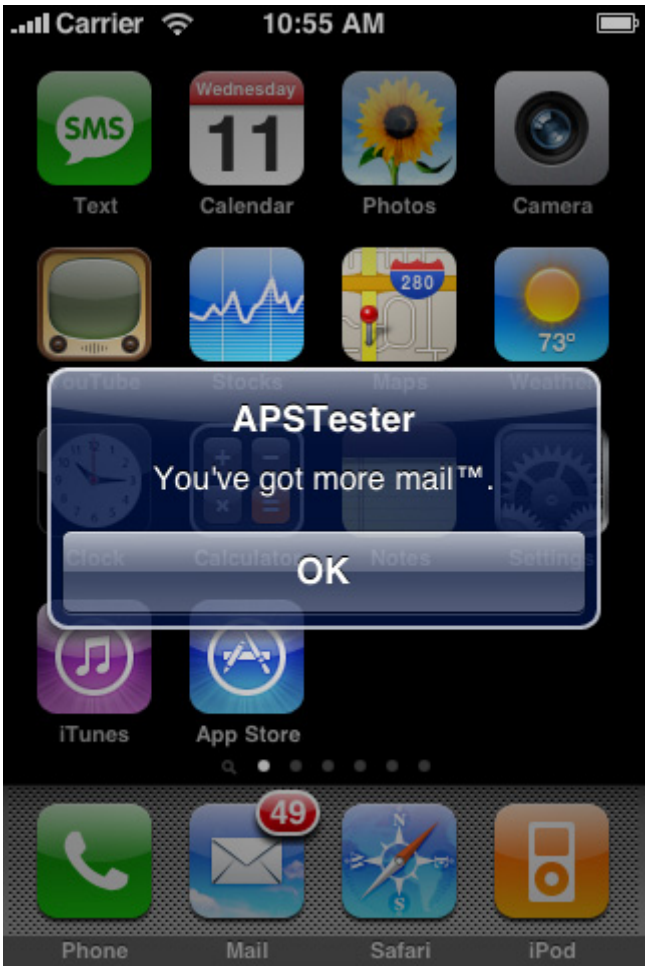
对于本地通知和推送通知，徽标数字可以由应用程序自己指定为任意数字，比如即将发生的日历事件的数量或下载数据项的数量或未读（但已经下载）的邮件数量。用户看到该徽标并点击应用图标，或在 Mac OS X 上面点击 Dock 里面的应用图标来启动该应用，它会显示 To-Do 项或用户感兴趣的其他任何内容。

Figure 1-2 An application icon with a badge number (iOS)

在 iOS 上面，一个应用指定提示信息或徽标数字时也可以指定一个声音文件。该声音文件应该包含一个短的，有特色的声音。在 iOS 显示提示窗或图标上显示徽标的时候，它会播放该声音文件的声音来告知通知已经到达。

通知提示消息可以只包含一个按钮。在后面的例子中，动作按钮被取消，如图 1-3 所示那样。用户只能隐藏该提示窗。

Figure 1-3 A notification alert message with the action button suppressed



操作系统传递一个本地通知或推送通知给应用程序，无论应用程序此时是否运行在前台。如果在通知到达的时候应用程序正在运行，没有提示窗显示或图标徽标或声音播放，即使此时设备是锁屏的（iOS 上）。相反，应用的委托会被告知该通知以便其直接处理该通知。（“调度、注册、和处理通知”部分会详细的介绍这些情景）

iPhone,iPad,和 iPod touch 设备的用户可以控制设备或设备上面安装的应用是否接收推送通知（设置 > 通知）。他们也可以选择性的启用或禁用某个特定应用推送通知的某些类型通知（比如图标徽标，提示消息，和声音）。你可以在设置里面的通知中心设置这些限制。UIKit 框架提供检测一个给定应用的用户偏好的编程接口。

1.2 更多关于本地通知

本地通知（仅在 iOS 上面支持）非常适合基于时间的行为，包括简单的日历事件或 To-Do 列表应用。本地通知对 iOS 允许的特定时间内运行在后台的应用程序也非

常有帮助。比如，应用程序依赖于服务器的消息或数据可以在后台运行的时候从服务器端获取到期项目。如果一个消息已经准备好被查看或一个更新已经准备好下载，它们可以立即显示一个本地通知告知用户。

本地通知是 UILocalNotification 的一个实例，包含了 3 个通用的属性：

- **调度时间** 你必须指定操作系统传递通知的日期和时间，及通常认为的开始时间。你可以需要按照特定的时区时间来指定日期以便操作系统可以在用户旅行的时候调整相应的开始时间。你也可以要求操作系统以特定的间隔来重新调度该通知（每周，每月等等）。
- **通知类型** 该类别包含了提示信息，动作按钮的标题，应用程序图标的徽标数字，和要播放的声音等。
- **自定义数据** 本地通知可以包含一个自定义数据的字典。

“调度本地通知”部分介绍了这些属性的编程细节。一旦应用程序创建了本地通知对象，它让操作系统来调度该通知或立即显示。

设备上面的每个应用程序最大调度的本地通知数为 64 个。超出最大限制时操作系统丢弃超出部分的通知。通常一个重复的通知可以作为一个通知来对待。

1.3 更多关于推送通知

iOS 应用或 Mac OS X 应用通常是 C/S（Client/Server）模式的一部分。客户端部分的应用程序通常安装来手持设备或电脑之上。服务器端的应用通常包含一个主函数来提供数据给它的许多客户端程序（即通常认为的提供者）。客户端程序偶尔连接到它的提供者并下载它等待的数据。Email 或社交网络应用就是典型的 C/S 模式。

但是当提供者有新的数据让客户端下载时，如果此时客户端应用没有连接服务器提供者，即使此时客户端应用是在设备或电脑上是运行在前台的，那会将会怎么样呢？客户端如何知道有新的数据待下载？推送通知出发点即是要解决这些问题。一个推送通知即是服务器提供者要传递给客户端设备操作系统的一个短消息。反过来操作系统通知设备上面的应用程序有新的数据待下载或有消息要查看等等。如果用户启用了该特性（iOS 上）而且应用程序也注册正常，通知会被传递到操作系统，然后分发给应用程序。苹果推送服务（APNs）即是实现推送特性的最佳方式。

推送通知服务器和桌面系统的后台应用很类似，但是它没有增加额外的消耗。对

于一个当前没有正在运行的应用程序，或者在 iOS 上的话即是没有运行在前台的应用程序，通知是间接发生的。操作系统接收一个应用程序的推送通知并提醒给用户。一旦提醒完成，用户可能选择加载启动应用程序，然后应用程序会从提供者上面下载最新的数据。如果一个应用程序在推送通知到来的时候正在运行，应用程序可以选择直接处理通知。

iOS 注意:*从 iOS 4.0 开始，应用程序可以进入后台模式，但是有特定的期限限制。任一时刻前台都只能运行一个应用程序。*

就像它的名字一样，苹果推送服务（APNs）使用推送来描述传递通知到目标设备或电脑之上的过程。推送（Push）设计和下拉（Pull）设计不同，下拉设计立即收到通知，此时操作系统被动的监听更新而不是积极的主动轮询。推送设计让下拉设计遇到可扩展性问题和消息及时传播的问题成为可能。APNs 实现推送通知时使用了持久的 IP 连接。

大部分推送通知都是由有一个载体构成：一个包含了定义 APNs 指定如何通知用户的属性列表。出于性能方面的考虑，载体应该尽可能小。尽管你可能为这些载体自定义属性，但是你不应该使用远程通知机制来传递数据，因为推送通知无法保证这些自定义属性可以被传输。关于载体的更多信息，参阅“通知载体”。

APNs 引用了它从提供者那接收到的最后一条给设备上面的应用的通知。所以如果一个设备或电脑在线但是还没有接收到之前的通知时，APNs 把保存的通知推送给它。运行 iOS 的设备通过 Wi-Fi 或蜂窝网络接收推送通知。运行 Mac OS X 的电脑通过 Wi-Fi 和以太网接收推送通知。

重要：*在 iOS 上，Wi-Fi 仅在蜂窝网络无法连通或设备是 iPod touch 时才会用于推送通知。对于一些通过 Wi-Fi 接收推送通知的设备，设备的显示器必须是开着的(也就是它不能处于休眠状态)，或者必须是插入的。相反，在 iPad 上当设备休眠的时候它仍然保持则 Wi-Fi 连接，所以仍然可以传递推送通知。无线 Wi-Fi 唤醒任何传入主机的流量。*

把远程通知特性添加到你的应用要求你从 iOS 的或 Mac OS X 的 Dev Center 获得合法的授权证书，并且编写客户端和服务器提供者上面的必要代码。“配置和开发”部分解析了证书授权和设置步骤，且“提供者和苹果推送服务（APNs）之间的通信”部分和”调度、注册、和处理通知”部分描述了具体的实现。

苹果推送服务（APNs）一直监听提供者以防非法行为，或寻找突然的尖峰活动，

快速连接-断开的周期，和类似的活动。当苹果检查到这些行为的时候，它会尝试通知提供者，并且如果这些行为一直持续的话，它可能把提供者的证书放入吊销列表并拒绝进一步的连接。任何持续的非法或有问题的行为都有可能造成提供者访问 APNs 的中断。

第二章　调度、注册、和处理通知

本章描述了 iPhone、iPad、或 iPod touch 应用部分应该处理调度本地通知，注册远程通知，和处理本地及远程通知的相关任务。因为和推送通知相关的客户端 API 作为远程通知的部分，所以本章术语中把推送通知替换远程通知。

2.1　准备好自定义的警告声音

对于 iOS 上面的远程通知，你可以在显示某个应用的本地或远程通知时指定一个自定义的声音来播放。但是声音文件必须在客户端程序的主目录下面（即 main bundle）。

因为自定义的警告声音由 iOS 的声音系统设备播放，所以它必须是以下任一格式的音频文件：

- Linear PCM
- MA4(IMA/ADPCM)
- μLaw
- aLaw

你可以把音频文件打包为 aiff、wav、或 caf 文件。然后在 Xcode 里面把声音文件添加到你的工程里面作为程序目录的一个非本地化的资源。

你可以使用 afconvert 工具来转换音频文件。比如，为了把一个 16-位的 Linear PCM 的系统声音 Submarine.aiff 转换为 IMA4 的 CAF 音频文件，你可以在终端使用如下的命令：

```
afconvert /System/Library/Sounds/Submarine.aiff ~/Desktop/sub.caf -d ima4 -f caff -v
```

你可以通过在 QuickTime Player 里面打开一个声音文件并选择 Movie 选项的 Show Movie Inspector 来检查一个声音文件以确定它的数据格式。

自定义的声音文件播放时长必须在 30 秒以内。如果一个自定义的声音文件播放超过 30 秒的限制，那将会被系统的声音替换。

2.2 调度本地通知

在 iOS 上面创建和调度本地通知需要你完成下面几个简单的步骤：

1. 创建并初始化一个 UILocalNotification 对象。
2. 设置系统应该处理该通知的日期和时间，即 fireDate 属性。

   如果你设置了本地的 NSTimeZone 的 timeZone 属性，当设备跨越不同的时区（或被重置时）系统将会自动调整触发时间（系统计算一个给定的日历或者日期值时，时区影响日期的属性--即天、月、小时、年、和分钟等）。你也可以循环的调度一个通知（每天，每周，每月等等）。
3. 配置通知的实体内容：警告、图标数字、和声音等
   a) 警告有一个消息的属性（alertBody 属性）和动作按钮或滑块的标题（alertAction）;这两个属性的字符串都可以国际化为用户当前的偏好语言。
   b) 你可以通过 applicationIconBadgeNumber 属性来设置应用程序图片上面显示的数字。
   c) 你可以给 soundName 属性赋值一个当前应用主目录下本地化的自定义声音文件名。为了过去系统默认的声音文件，给该属性赋值 UILocalNotificationDefaultSoundName。声音应该始终伴随着出现警告消息或图标数字，否则他们不应该被播放。
4. 可选的，你可以通过 userInfo 属性来放入自定义的数据到通知里面。userInfo 的键和值必须是属性列表对象。
5. 调度本地通知以便处理。

   你可以通过调用 UIApplication 的方法 scheduleLocalNotification:来调度一个本地通知。程序通过使用 UILocalNotification 对象指定的触发日期来调度它。另外，你可以通过调用 presentLocalNotificationNow：方法来立即显示一个通知。

列表 2-1 显示了创建和调度一个本地通知来告知用户未来的一个任务列表。有几点需要注意的。对于 alrtBody 和 alertAction 属性，它从主目录的本地化字符串里面获取用户偏好语言的相应字符（通过 NSLocalizedString）。它同时把任务列表的相关名词赋值到 userInfo 属性里面。

Listing 2-1 Creating, configuring, and scheduling a local notification

```
- (void)scheduleNotificationWithItem:(ToDoItem *)item interval:(int)minutesBefore {

    NSCalendar *calendar = [NSCalendar autoupdatingCurrentCalendar];

    NSDateComponents *dateComps = [[NSDateComponents alloc] init];

    [dateComps setDay:item.day];

    [dateComps setMonth:item.month];

    [dateComps setYear:item.year];

    [dateComps setHour:item.hour];

    [dateComps setMinute:item.minute];

    NSDate *itemDate = [calendar dateFromComponents:dateComps];

    [dateComps release];



    UILocalNotification *localNotif = [[UILocalNotification alloc] init];

    if (localNotif == nil)

        return;

    localNotif.fireDate = [itemDate addTimeInterval:-(minutesBefore*60)];

    localNotif.timeZone = [NSTimeZone defaultTimeZone];



    localNotif.alertBody = [NSString stringWithFormat:NSLocalizedString(@"%@ in %i minutes.",
nil),

         item.eventName, minutesBefore];

    localNotif.alertAction = NSLocalizedString(@"View Details", nil);



    localNotif.soundName = UILocalNotificationDefaultSoundName;

    localNotif.applicationIconBadgeNumber = 1;



    NSDictionary *infoDict = [NSDictionary dictionaryWithObject:item.eventName
forKey:ToDoItemKey];

    localNotif.userInfo = infoDict;



    [[UIApplication sharedApplication] scheduleLocalNotification:localNotif];

    [localNotif release];

}
```

你可以通过调用程序对象的 cancelLocalNotification:方法来取消一个指定的已经调度的通知，甚至你可以通过调用 cancelAllLocalNotifications 方法来取消所有已调度的通知。这两个方法都可以隐藏当前已经显示的通知警告窗口。

当应用程序运行在后台的时候，一些用户感兴趣的消息，数据，或其他项抵达时，此时本地通知就会非常有帮助。在这些情况下，它们应该通过使用 UIApplication 的方法 presentLocalNotificationNow:来立即显示通知（iOS 给定有限的时间让应用程序在后台执行）。列表 2-2 列举了如何操作：

Listing 2-2 Presenting a local notification immediately while running in the background

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    NSLog(@"Application entered background state.");
    // bgTask is instance variable
    NSAssert(self->bgTask == UIInvalidBackgroundTask, nil);

    bgTask = [application beginBackgroundTaskWithExpirationHandler: ^{
        dispatch async(dispatch get main queue(), ^{
            [application endBackgroundTask:self->bgTask];
            self->bgTask = UIInvalidBackgroundTask;
        });
    }];

    dispatch async(dispatch get main queue(), ^{
        while ([application backgroundTimeRemaining] > 1.0) {
            NSString *friend = [self checkForIncomingChat];
            if (friend) {
                UILocalNotification *localNotif = [[UILocalNotification alloc] init];
                if (localNotif) {
                    localNotif.alertBody = [NSString stringWithFormat:
                        NSLocalizedString(@"%@ has a message for you.", nil), friend];
                    localNotif.alertAction = NSLocalizedString(@"Read Message", nil);
                    localNotif.soundName = @"alarmsound.caf";
                    localNotif.applicationIconBadgeNumber = 1;
                    [application presentLocalNotificationNow:localNotif];
```

```
                [localNotif release];
                friend = nil;
                break;
            }
        }
    }
    [application endBackgroundTask:self->bgTask];
    self->bgTask = UIInvalidBackgroundTask;
});

}
```

2.3 注册远程通知

一个应用程序必须在设备（iOS 设备或 Mac 电脑）上面注册了苹果推送通知服务才能接收来自程序提供者的远程通知。

注册过程包含以下三个步骤：

1. 程序调用 registerForRemoteNotificationTypes:方法。
2. 委托实现 application:didRegisterForRemoteNotificationsWithDeviceToken:方法来接收设备令牌。
3. 把设备令牌作为非对象（二进制值）传递给程序提供者。

注意：*除非特别声明，否则本章所有声明的方法都来着 UIApplication 和 NSApplication，而对于委托，是 NSApplicationDelegate 协议和 UIApplicationDelegate。*

在“令牌的生成和扩散”部分的图 3-3 将会列举说明应用程序、设备、APNs、和消息提供者之间的关系。

应用程序应该在每次它启动的时候注册，并把最新获得的设备令牌发送给它的提供者。它通过调用 registerForRemoteNoificationTypes:方法来揭开注册的过程。此方法的参数需要一个 UIRemoteNotificationType（在 Mac OS X 上为 NSRemoteNotificationType）位掩码，它指定应用程序希望得到的初始通知类型，比如图标数字、声音、但不包含警告信息等。在 iOS 上，之后用户可以在应用设置的通知中心修改已启用的通知类型。在 iOS 和 Mac OS X 上面，你可以通过调用

enableRemoteNotificationTypes 方法了获得当前可用的通知类型。如果这些通知类型的任何一个不被启用，那么操作系统将不会标记图标数字，显示警告信息，或播放警告声音等，即使在通知负载里面已经指定了相应的内容。

Mac OS X 注意：*因为对于非运行的应用程序唯一可用的通知类型是图标数字，所以可以简单的把 NSRemoteNotificationTypeBadge 作为 registerForRemoteNotificationTypes:的参数。*

如果注册成功，APNs 会返回一个设备令牌给设备，iOS 把令牌通过应用的委托方法 application:didRegisterForRemoteNotificationsWithDeviceToken:返回。应用程序应该连接到它的提供者并把令牌作为二进制传递给它。如果在获取令牌的时候发生任何异常，操作系统将会通过调委托的 application:didFailToRegisterForRemoteNotificationsWithError:方法来通知应用程序。该方法的参数 NSError 清晰的描述了错误的原因。例如，错误的一个可能是证书文件里面的 aps-environment 值。你应该把错误视为一个短暂的状态，而不是试图解析它。（详细可以参阅“创建并安装证书”部分）。

iOS 注意：*如果设备的蜂窝连接或 Wi-Fi 连接都不可用，则 application:didRegisterForRemoteNotificationsWithDeviceToken:方法或 application:didFailToRegisterForRemoteNotificationsWithError:方法都不会被调用。在 Wi-Fi 环境下，有时候发生在设备无法连接到APNs 的端口5223。如果发生这样情况的话，用户可以在 iPhone,iPad 等设备上面连接其他没有阻塞该端口的 Wi-Fi 网络，等待直到蜂窝数据服务可用。在任何一种情况下，连接成功的话，委托的任一方法将会被调用。*

通过程序每次启动的时候请求最新哦设备令牌并把令牌发送给它的提供者，你可以确保提供者拥有当前设备的可用令牌。如果用户恢复设备或电脑到不是为它创建的备份（比如，用户移动数据到一个新的设备或电脑），那么他必须起码启动并加载一次应用程序以便它可以再次接收远程通知。如果用户恢复备份数据到一个新的设备或电脑，或重新安装操作系统，设备的令牌都会改变。 此外，不要缓存一个设备令牌并把它发送给提供者，要记住总是在你需要设备令牌的时候总是从操作系统获取最新的令牌。如果你的应用之前已经注册了通知，那么调用 registerForRemoteNotificationTypes:的时候会立即返回设备的令牌给委托，而不会造成任何的额外负载。

列表 2-3 给出了一个如何在 iOS 应用程序上面注册远程通知的简单例子。该示例

代码和 Mac OS X 应用上面的类似。（SendProviderDeviceToken 是一个由客户端创建的假想方法，在它里面连接提供者，并把设备令牌发生给提供者）。

Listing 2-3 Registering for remote notifications

```
- (void)applicationDidFinishLaunching:(UIApplication *)app {

   // other setup tasks here....

    [[UIApplication sharedApplication]
registerForRemoteNotificationTypes:(UIRemoteNotificationTypeBadge |
UIRemoteNotificationTypeSound)];

}



// Delegation methods

- (void)application:(UIApplication *)app
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)devToken {

    const void *devTokenBytes = [devToken bytes];

    self.registered = YES;

    [self sendProviderDeviceToken:devTokenBytes]; // custom method

}



- (void)application:(UIApplication *)app
didFailToRegisterForRemoteNotificationsWithError:(NSError *)err {

    NSLog(@"Error in registration. Error: %@", err);

}
```

2.4 处理本地和远程通知

让我们检查一下当操作系统提供一个本地通知或远程通知给一个应用程序的时候可能出现的情况。

- 无论应用是否运行在前台，通知都会被提供。

  在这种情况下，系统展示通知，显示一个警告，在图标上面显示数字，可能播放一个声音。

- 通知展示的结果是，用户点击警告窗口的动作按钮或点击应用程序的图标。

  如果运行 iOS 的设备应用程序的图标被点击，该应用程序会调用相同的方法，但是不会提供任何关于通知的信息。如果运行 Mac OS X 的设备的应用程序图标被点击，应用会调用委托的 applicationDidFinishLaunching:方法，在该委

托里面可以获取远程通知的负载。

iOS 注意：*应用的委托可以实现 applicationDidFinishLaunching:方法，而不是 application:didFinishLaunchingWithOptions:，但是强烈不建议这样做。后一种方法允许应用程序接收和它启动相关的信息，这些信息不仅仅包括通知。*

- 当应用运行在前台的时候，通知被提交。

  应用程序调用它委托的 application:didReceiveRemoteNotification:方法（对于远程通知而言），或者 application:didReceiveLocalNotification:方法（对于本地通知而言），并传递通知的负载或本地通知对象给方法的参数。

注意：*本部分指出的委托方法在名称里面包含"RemoteNotification"字样的都是在 NSApplicationDelegate 协议和 UIApplicationDelegate 里面定义的。*

应用程序可以使用传递进来的远程通知负载，或者在 iOS 上面的 UILocalNotification 对象来帮助设置上下文处理和通知相关的项目。理想情况下，任何时候每个平台上面的委托处理由远程和本地通知传递的所有信息。

- 对于 Mac OS X,它应该通过 NSApplicationDelegate 协议，并实现 applicationDidFnishLaunching:方法和 application:didReceiveRemoteNotification:方法。
- 对于 iOS,它必须通过 UIApplicationDelegate，并实现 application:didFinishLaunchingWithOptions:方法和 application:didReceiveRemoteNotification:或 application:didReceiveLocalNotification:方法。

iOS 注意：*在 iOS 上，你可以通过确定应用程序的状态来确定是否是应用程序由于用户点击了动作按钮而启动，或通知被提交给已经运行的应用。通过委托实现 application:didReceiveRemoteNotification:或 application:didReceiveLocalNotification:方法，获取应用程序的 applicationState 属性值并判定它。如果值为 UIApplicationStateInactive,表明用户单击了动作按钮。如果值为 UIApplicationStateActive,表明应用收到此通知的时候已经运行在前台。*

列表 2-4 的的 iOS 应用程序的委托事项了 application:didFinishLaunchingWithOptions:方法来处理本地通知。它使用键 UIApplicationLaunchOptionsLocalNotificationKey 从加载选项里面的字典获取了一个相关的 UILocalNotification 对象。从 UILocalNotification 对象的 userInfo 字典里面，

它获取任何项目（这是通知的目的），并使用它来设置程序的初始化上下文。在该示例中，作为处理通知的一部分，你应用正确的重置应用程序图标的数字，或者如果没有额外的任务时把它移除。

Listing 2-4 Handling a local notification when an application is launched

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

    UILocalNotification *localNotif =

        [launchOptions objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];

    if (localNotif) {

        NSString *itemName = [localNotif.userInfo objectForKey:ToDoItemKey];

        [viewController displayItem:itemName];  // custom method

        application.applicationIconBadgeNumber =
localNotif.applicationIconBadgeNumber-1;

    }

    [window addSubview:viewController.view];

    [window makeKeyAndVisible];

    return YES;

}
```

实现对远程通知的处理和本地通知类似，唯一的区别是你使用一个在每个平台上面已经定义的常量来来访问远程通知的负载。

- 在 iOS 平台上，委托在它的 application:didFinishLaunchingWithOptions:方法里面使用 UIApplicationLaunchOptionsRemoteNotificationKey 来从加载选项字典里面获取远程通知的负载。
- 在 Mac OS X 平台上面，委托在它的 applicationDidFinishLaunching:方法的实现里面，使用 NSApplicationLaunchRemoteNotificationKey 来访问 NSNotification 对象的 userInfo 字典里面的负载字典。

负载本身是一个 NSDictionary 对象，包含了通知的元素--警告语，标记数字，声音等等。它同样也包含了自定义数据，程序可以使用这些自定义数据来设置用户界面的初始化情况。参阅“通知负载”部分来了解更多远程通知负载的介绍。

重要：*你不应该在通知负载里面定义自定义属性来传递自定义数据或任何其他敏感数据。远程通知不保证一定被提交。正确使用自定义负载属性的一个示例是使用一个字符串来识别邮箱消息被下载到那个邮箱客户端。应用程序可以把该字符串纳入到它的下载用户界面。另外一个自定义负*

载属性的示例是一个时间戳标识一个提供者第一次发送通知的时间。客户端程序可以使用该值来估计这个通知已经多久了。

当在 application:didFinishLaunchingWithOptions:或 applicationDidFinishLaunching:里面处理远程通知的时候，应用程序的委托可能执行一个额外的任务。仅在程序启动后，委托应该连接到它的提供者并获取等待的数据。列表 2-5 给出了一个该处理的一个大纲。

Listing 2-5 Downloading data from a provider

```
- (void)application:(UIApplication *)app didFinishLaunchingWithOptions:(NSDictionary
*)opts {

    // check launchOptions for notification payload and custom data, set UI context

    [self startDownloadingDataFromProvider];  // custom method

    app.applicationIconBadgeNumber = 0;

    // other setup tasks here....

}
```

注意：*客户端应用程序应该是一和它的提供者之间采用异步的方式或独立的线程。*

列表 2-6 的代码显示了 application:didReceiveLocalNotification:方法的实现，它被运行在前台的应用程序调用。这里应用程序执行和列表 2-4 相同的工作。这次它可以直接访问 UILocalNotification 对象因为该对象是方法的参数之一。

Listing 2-6 Handling a local notification when an application is already running

```
- (void)application:(UIApplication *)app
didReceiveLocalNotification:(UILocalNotification *)notif {

    NSString *itemName = [notif.userInfo objectForKey:ToDoItemKey]

    [viewController displayItem:itemName];  // custom method

    application.applicationIconBadgeNumber =
notification.applicationIconBadgeNumber-1;

}
```

如果你想让你的应用程序在运行在前台的时候也能接收系统提交的远程通知信息，那么应用程序委托应该实现 application:didReceiveRemoteNotification:方法。委托应用开始下载等待数据，消息，或其他项目的过程，这些完成之后，它应该移除程序图标的数字（如果你的应用程序经常检测它的提供者来获取新的数据，那么没有必要实现该方法）。该方法的第二参数是通知的负载，你不应该使用它包含的任何自定义属性来修改你应用的当前上下文。

虽然 Mac OS X 上面非前台的应用程序唯一支持的通知类型是标记，但是委托可

以实现 application:didReceiveRemoteNotification:来检查通知负载以获取其他通知类型，并正确的处理它们（比如显示警告窗，播放声音等）。

iOS 注意：*如果用户在接收到通知显示很短的时间内解锁设备，那么操作系统将会自动的触发和警告窗相关的动作（该行为和短信及日历提醒一致）。它对于远程通知相关的动作不会产生破坏性的效果很重要。用户应该始终对于破坏应用程序上下文的存储数据的结果作出选择。*

2.5 传递当前用户偏好语言给提供者（远程通知相关）

如果应用程序没有使用 aps 里面的 loc-key 和 loc-args 属性来让客户端获取本地化的警告信息，提供者则需要本地化它放入通知负载的警告信息。然而为了实现该目的，提供者需要知道用户设备的当前偏好语言（用户通过设置 General > International > Language 及通用 > 国际化 > 语言）。客户端程序需要把用户的偏好语言的缩写标识符发生给它的提供者。可以规范化的 IETF BCP 47 语言标识符，比如”en”或”fr”。

注意：*关于 loc-key 和 loc-args 属性和客户端本地化消息的更多信息，参阅“**通知负载**”部分。*

列表 2-7 举例说明了如何获取当前所选择的偏好语言和与提供者通讯。在 iOS 平台上，NSLocale 的 preferredLanguages 返回的数组包含一个对象：一个 NSString 对象封装了偏好语言的语言标识码。UTF8String 把字符串对象转换为 UTF8 编码的 C 字符串。

Listing 2-7 Getting the current supported language and sending it to the provider

```
NSString *preferredLang = [[NSLocale preferredLanguages] objectAtIndex:0];
const char *langStr = [preferredLang UTF8String];
[self sendProviderCurrentLanguage:langStr]; // custom method
}
```

应用程序可能需要每次用户在修改当前本地里面的偏好语言时把最新偏好语言发送给它的提供者。为了实现该目的，你可以监听 NSCurrentLocaleDidChangeNotification，并在你的通知处理方法里面获取最新偏好语言的标识码，并发生给你的提供者。

如果偏好语言不被当前应用程序所支持，那个提供者应用使用广泛的后备语言的本地化消息，比如英语或西班牙语。

第三章　苹果推送通知服务

苹果推送通知服务（简称 APNs）是推送通知特性的核心。它是设备（比如 iPhone、iPad、iPod touch 等）传播信息的一种强大且高效的途径。每个设备和服务器建立一个认证服务和加密的 IP 连接，并持续接收通知。如果一个应用的通知在应用出于非运行状态的时候抵达，那么设备将会提示用户该应用有等待的新数据。

应用开发者（简称“提供者”）在他们的应用服务端组织通知。当提供者有新来的数据要提供给他们的应用的时候，它们会通过持久和安全的通道和 APNs 建立连接。当给应用的新数据抵达时，提供者通过和 APNs 建立的通道准备并发送一个通知，该通知会发送到目标设备上面。

APNs 除了是一个简单且有效的高容量的运输服务外，它还包含了一个为存储-转发质量的默认的质量服务组件。参阅“服务质量”部分获取更多信息。

“提供者和苹果推送通知服务间的通信”和“调度、注册、和处理通知”部分讨论了提供者和 iOS 应用需要分别实现的要求。

3.1　推送通知和它的路径

苹果推送通知服务把提供者提供的通知传输并路由到给定的目标设备。一个推送通知主要由两个主要部分构成：**设备令牌**和**负载**。设备令牌类似一个电话号码。它主要包含了可以让 APNs 定位到目标设备那个已安装的应用的信息。APNs 也用它来验证通知的路由信息。负载主要是一个 JSON 定义的属性列表，它指定了目标设备上面的应用如何显示一个警告提示。

注意：*关于设备令牌的更多信息，请参阅“安全架构”部分。关于通知负载的更深入了解，参阅“通知负载”部分。*

远程通知的数据流是单向性的。提供者组织通知包内容包含目标设备应用的令牌和负载。提供者把通知发送给 APNs，而由 APNs 把通知推送到目标设备上面。

当提供者和 APNs 验证的时候，它会把将要提高数据的目标设备的应用的主题提供给 APNs。该主题当前是目标 iOS 设备上面应用的标识符（bundle identifier）。

Figure 3-1　A push notification from a provider to a client application

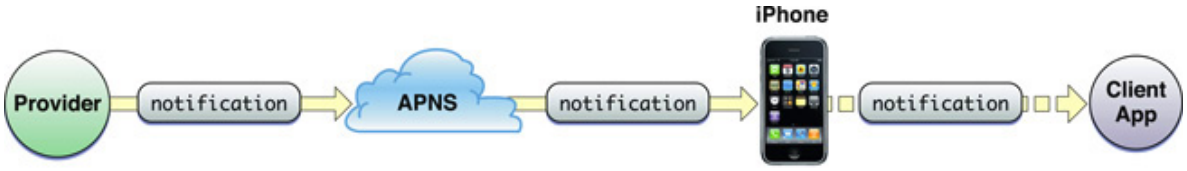
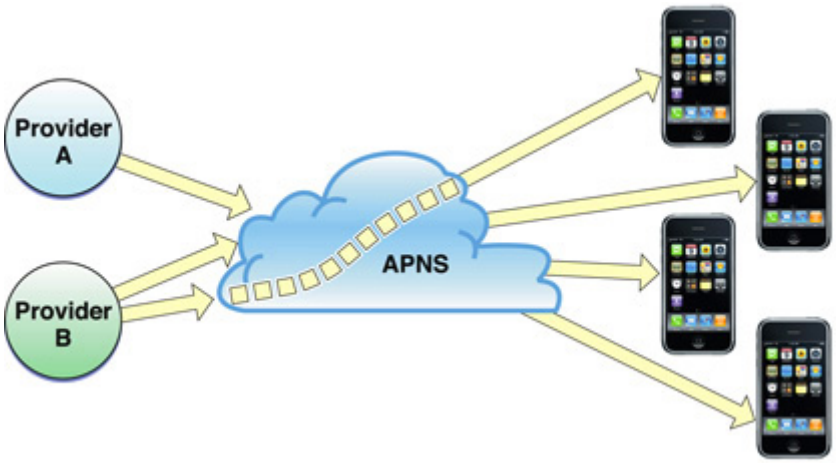


图 3-1 高度概括了 APNs 在提供者和设备间扮演的角色。APNs 面向设备和面向提供者之间建立了多个连接节点。在面向提供者方面，这些建立称为网关（gateway）。通常包含很多提供者，每个提供者都通过这些网关和 APNs 建立一个或多个持久安全的连接。而且这些提供者通过 APNs 发送通知到许多已经安装了它客户端应用的设备上面。图 3-2 描述了该过程。

Figure 3-2 Push notifications from multiple providers to multiple devices



3.2 反馈服务

有些时候 APNs 试图多次把通知推送给目标设备的应用程序，但是设备可能没有目标应用所以可能重复拒绝推送。这通常发生在用户已经卸载了应用的情况下。这个时候，APNs 通过和提供者建立的连接反馈这个情况。反馈服务维护了一个设备列表，该列表列出了那些设备经常而且重复的拒绝推送通知。提供者应用获取该设备列表，并停止发送通知到这些设备上面。关于该服务的更多信息，参阅“反馈服务”部分。

3.3 服务质量

苹果推送通知服务包含了一个默认的服务质量（QoS）组件，它扮演了存储和转发的功能。如果 APNs 试图把推送通知发送到一个离线的设备时，QoS 把该通知存储起来。它为每台设备的每个应用只保留一份推送通知：即从应用的提供者哪里获得的最新的一个通知。当理想的设备恢复上线时，QoS 把存储的通知发送到目标设备上面。

QoS 为通知保留一段有限的时间，过期则把它删除掉。

3.4 安全架构

为了在提供者和设备之间建立稳定的通信，苹果推送通知服务必须公开某些入口节点。但为了安全起见，它还必须规范这些入口节点的访问。为了实现该目的，APNs 要求提供者、设备、和它们之间的通信拥有两个不同的信任级别。这就是通常的**连接信任**和**令牌信任机制**。

连接信任确保一方面 APNs 连接和已经同意接收推送通知的提供者之间的验证。从设备连接的另一方面考虑，APNs 必须验证和设备之间的连接是合法的。

在 APNs 已经和入口节点建立稳定的信任连接后，它必须确保传达通知的合法节点。为了实现该目的，它必须验证消息传输的路由路径。仅且是目标的设备才会接收该通知。

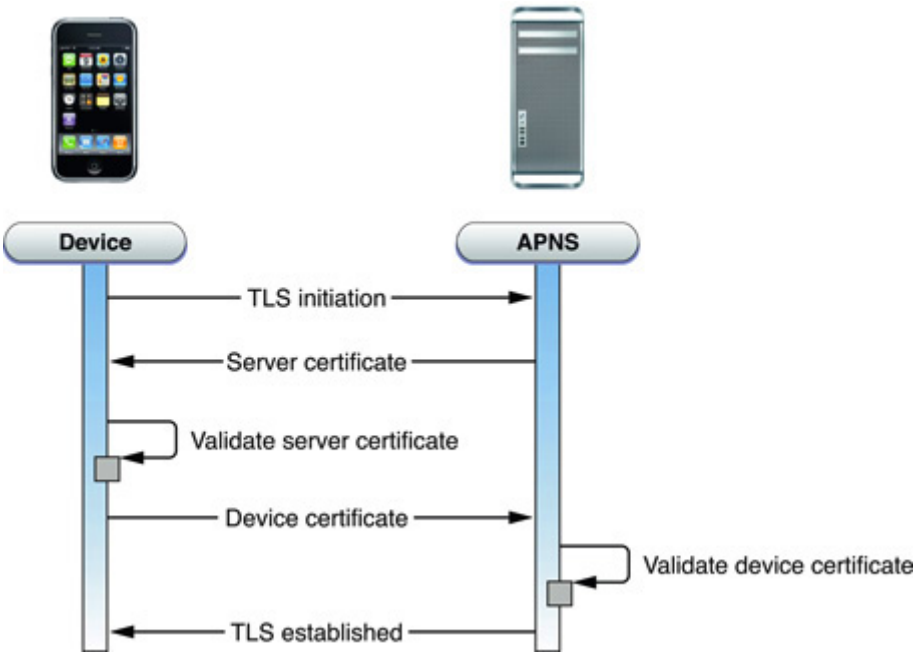
APNs 通过设备令牌确保消息的路由是正确的，即**令牌信任机制**。一个设备令牌是一个不透明的设备标识符，它是 APNs 在和设备第一次连接时候产生的。设备和它的提供者之间共享该设备令牌。此后该设备令牌一直伴随的每个从提供者那发出的通知。这是建立合法的路由路径的信任基础（从隐喻的意义考虑，它和电话号码具有类似的功能，即标识一个通信的目标）。

注意：*设备令牌和 UIDevice 的属性 uniqueIdentifier 返回的 UDID 不同。*

以下各节讨论了连接信任所需条件和建立稳定信任连接的四个过程。

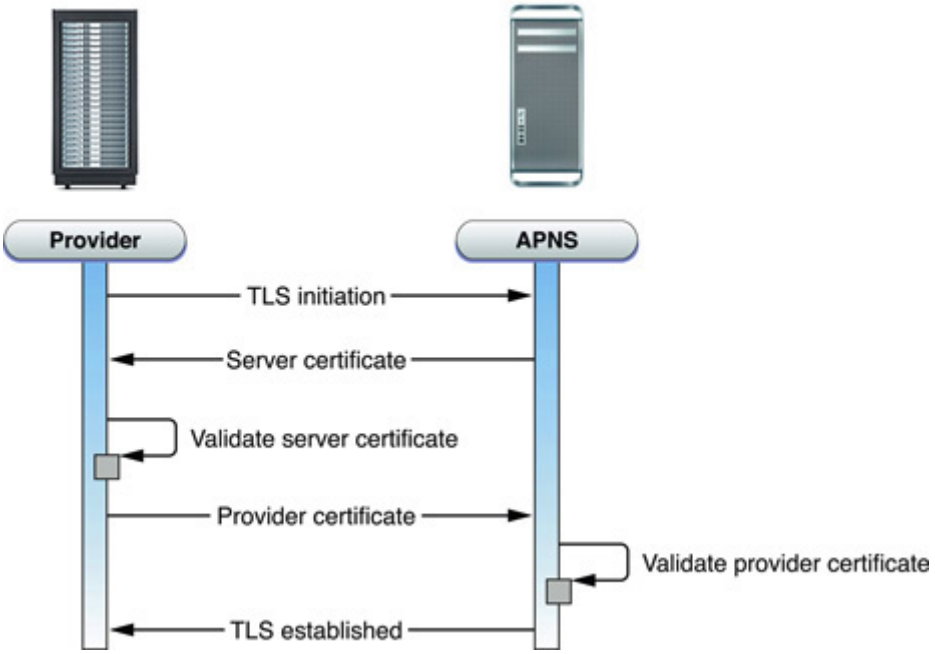
3.4.1 服务器-设备的信任连接

APNs 通过 TLS 的点对点认证方式来建立稳定的连接（注意 iOS 会自己处理该信任连接，你自己不要实现任何东西）。在此过程中，一个设备和 APNs 初始化一个 TLS 连接，然后 APNs 返回自己的服务器证书。设备验证该证书并把它的设备证书发送给 APNs，然后 APNs 验证该证书。



3.4.2 提供者-服务器的信任连接

提供者和 APNs 之间的信任连接同样通过 TLS 的点对点认证来建立。连接的建立过程和”服务器-设备的信任连接”部分描述的类似。提供者初始化一个 TLS 连接，从 APNs 获取服务器的证书，并验证该证书。然后提供者把自己的证书发送给 APNs，APNs 验证收到的证书。一旦建立过程完成，一个安全稳定的 TLS 连接即形成。

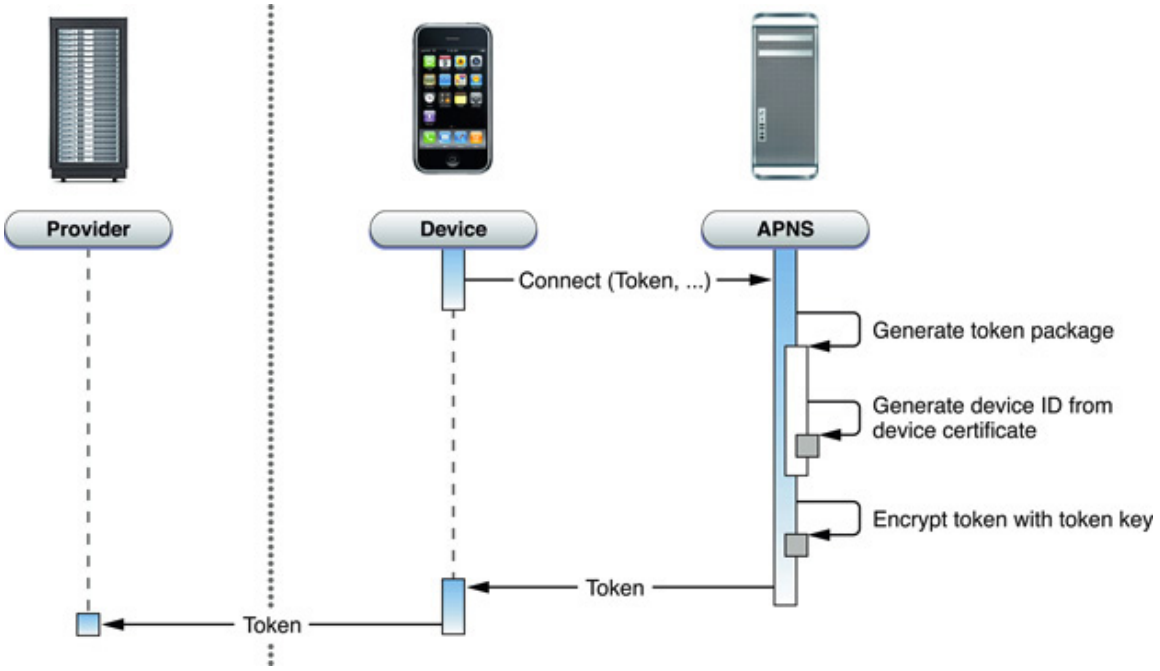


需要注意的是提供者建立的连接只能把推送通知发送给指定的应用程序，而该应用程序由证书里面指定的标识符（Bundle ID）指定。APNs 同样维护了一个失效证书

清单。如果某个提供者的证书在该清单里面，APNs 可能撤销提供者的信任（即拒绝连接）。

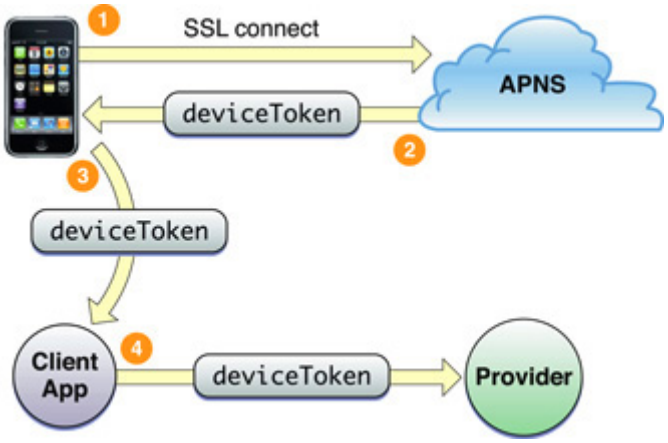
3.4.3 令牌的生成和扩散

基于 iOS 的应用必须注册来接收推送通知。这通常在应用被安装到设备后完成（该过程在“调度、注册、处理通知”部分介绍）。iOS 接收到应用的注册请求，然后连接到 APNs，并提交该请求。APNs 利用包含在设备唯一证书的信息来生成令牌。设备令牌包含了设备的标识符。然后它使用令牌键来加密设备令牌并把它返回给设备。



设备把设备令牌以 NSData 的对象返回给发送请求的应用。然后应用必须把该设置令牌以二进制或十六进制的格式发送给它的提供者。图 3-3 同样举例说明了令牌生成和扩散的顺序，但是还额外显示了客户端应用提供给它提供者设备令牌的过程。

Figure 3-3 Sharing the device token

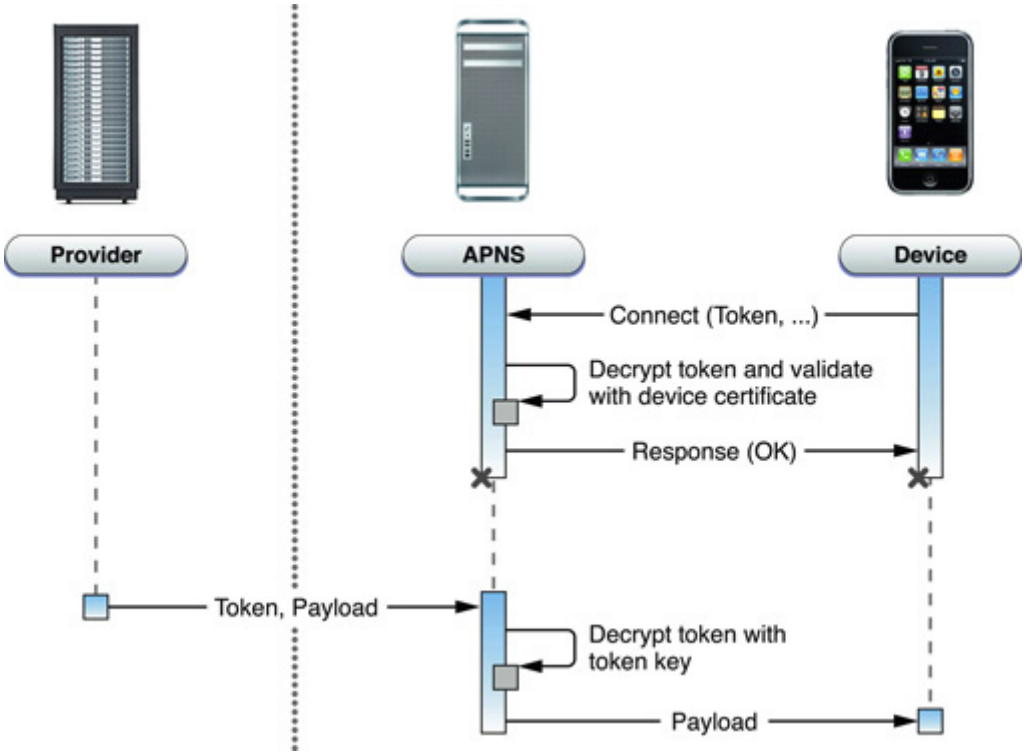


令牌信任的形式确保仅且只有 APNs 可以生成令牌，而且保证设备提交给它的令牌和它之前为该设备生成的令牌相同。

3.4.4 令牌信任（通知）

在 iOS 从 APNs 获取了设备令牌后，如“令牌生成和扩散”部分描述的，它必须在每次和 APNs 连接的时候提供该设备令牌。APNs 解码该令牌并验证该令牌是否是当前连接设备产生的。为了验证该设备令牌，APNs 确保令牌包含的设备标识符和设备证书里面的设备标识符匹配。

提供者每次发送给 APNs 用于提供给设备的通知必须附带它从设备的应用程序里面获取的设备令牌。APNs 使用令牌键解密该令牌，然后确定该通知合法。然后它是由包含在设备令牌的设备 ID 来确定通知的目标设备。



3.4.5 组件信任

为了支持 APNs 的安全模式，提供者和设备必须具备合法的证书，证书颁发机构（CA），或令牌。

- **提供者**：每个提供者请求一个唯一的提供者证书和与 APNs 建立连接的私有密钥。该证书由苹果颁发，必须标识有提供者发表的特定主题。该主题就是客户端应用的 Bundle ID。对于每个通知，提供者必须提供标识目标设备的设备令牌给 APNs。提供者可以选择性的验证它用于和 APNs 服务器建立连接的证书。
- **设备**：iOS 使用已经和它建立连接的 APNs 提供的公共服务器证书来验证服务。它包含了一个唯一的私有键和证书，iOS 使用它们来验证服务并建立稳定的 TLS 连接。它在设备活跃的时候获取设备证书和键并把它们存储到钥匙串里面。iOS 同样保存特定的设备令牌，该设备令牌是在服务连接的过程中收到的。每个已经注册的客户端应用负责把该设备令牌提交给它的提供者。

APNs 服务器用户拥有必要的证书，CA 证书，和加密密钥（公有和私有）来验证连接并标识提供者和设备。

3.5 通知负载

每个推送通知都携带自己的负载。负载指定了如何通知用户有等待的数据需要下载到客户端。负载运行的最大值是 256 字节。苹果推送通知服务拒绝任何负载超过该限制的通知。请记住提交通知只是“尽力而为”，而不一定推送完成。

对于每个推送通知，提供者必须自己构建一个严格遵守 RFC 4627 的 JSON 字典。该字典必须包含一个键名为 aps 的字典。而 aps 的字典包含一个或多个属性，这些属性指定如下动作：

- 显示给用户的警告信息
- 应用程序图标显示的数字
- 播放的声音

注意：*尽管你可以在一个独立的通知里面组合警告信息、图标数字、和声音，但是你应该考虑推送通知的人机界面影响。比如，用户可能会频繁的警告提示声音会比较恼人，特别是需要下载的数据没有临界。*

如果通知抵达的时候目标应用程序没有运行，警告信息、声音、或数字值简化被播放或显示。如果当前应用程序正在运行，iOS 会把通知作为 NSdictionary 对象提交给应用程序的委托。该字典包含了 Cocoa 正确的属性列表对象（还有 NSNull）。

提供者可以在苹果保留的 aps 空间之外指定自定义的负载值。自定义值必须使用 JSON 结构和原始类型：字典（对象）、数组、字符串、数字、和布尔类型。你不应该包含自定义信息作为自定义负载数据。相反，为此你需要设置上下文（为了用户界面）或内部指标。比如，自定义负载值可以是一个即时通信客户端应用使用的会话标识符或一个用来标识通知发送的时间戳。任何与警告信息有关的行为都不应该具有破坏性，比如删除设备上面的数据。

重要：*应用推送得不到保证，所以你不应该依赖远程通知的负载来提供关键的数据给客户端应用。而且绝不要在负载里面包含任何敏感的信息。你应该仅用它来通知用户有新的数据可用。*

表 3-1 列出了 aps 负载的键和预期的值。

Table 3-1 lists the keys and expected values of the `aps` payload.

Key	Value type	Comment
`alert`	string or dictionary	If this property is included, iOS displays a standard alert. You may specify a string as the value of `alert` or a dictionary as its value. If you specify a string, it becomes the message text of an alert with two buttons: Close and View. If the user taps View, the

		application is launched. Alternatively, you can specify a dictionary as the value of `alert`. See Table 3-2 for descriptions of the keys of this dictionary.
`badge`	number	The number to display as the badge of the application icon. If this property is absent, any badge number currently shown is removed.
`sound`	string	The name of a sound file in the application bundle. The sound in this file is played as an alert. If the sound file doesn't exist or `default` is specified as the value, the default alert sound is played. The audio must be in one of the audio data formats that are compatible with system sounds; see"Preparing Custom Alert Sounds" for details.

Table 3-2 Child properties of the `alert` property

Key	Value type	Comment
`body`	string	The text of the alert message.
`action-loc-key`	string or`null`	If a string is specified, displays an alert with two buttons, whose behavior is described in Table 3-1. However, iOS uses the string as a key to get a localized string in the current localization to use for the right button's title instead of "View". If the value is `null`, the system displays an alert with a single OK button that simply dismisses the alert when tapped. See"Localized Formatted Strings" for more information.
`loc-key`	string	A key to an alert-message string in a `Localizable.strings` file for the current localization (which is set by the user's language preference). The key string can be formatted with `%@` and `%n$@` specifiers to take the variables specified in `loc-args`. See "Localized Formatted Strings" for more information.
`loc-args`	array of strings	Variable string values to appear in place of the format specifiers in `loc-key`. See "Localized Formatted Strings" for more information.
`launch-image`	string	The filename of an image file in the application bundle; it may include the extension or omit it. The image is used as the launch image when users tap the action button or move the action slider. If this property is not specified, the system either uses the previous snapshot,uses the image identified by the `UILaunchImageFile` key in the application's `Info.plist` file, or falls back to `Default.png`. This property was added in iOS 4.0.

注意：*如果你想让 iPhone、iPad、或 iPod touch 设备显示一个警告消息的时候包含两个按钮（即关闭按钮-Close 和查看按钮-View），那么指定一个字符串作为 alert 的直接值。不要给 alert 属性指定一个只包含 body 属性的字典值（即选择{'aps':{'alert':"message..."}}，而不要这样{'aps':{'alert'::{'body':"message..."}}}）。*

3.5.1 本地化格式的字符串

你可以以两种方式来显示本地化的警告消息。一种方式是可以在服务器端把通知本地化。为此，你必须知道当前设备所选择的偏好语言（参阅" 把当前偏好语言发送给提供者"部分）。另一种方式是客户端应用实现对警告语言的本地化工作。提供者指

定推送通知的 aps 字典里面的 loc-key 和 loc-args 属性。当设备接收到推送通知时（假设当前应用程序没有运行），它使用这些 aps 字典的属性来查找当前语言的本地化格式对应的字符串，并把它们显示给用户。

以下详细介绍第二种方式如何工作。

iOS 应用可以国际化它所支持的语言的资源，比如图片、声音、和文字，国际化工作收集这些资源并把它们以两部分名称放到主目录下的子文件夹：语言代码和.lproj 扩展的文件（比如 fr.lproj）。编程显示的本地化字符串放在一个名为 Localizable.strings 文件。该文件里面的每个条目包含了一个键和本地化的字符串值。字符串可以包含规范化格式的变量值。当应用程序请求一个特定资源--即本地化字符串--它将会获得和当前所选择的偏好语言相匹配的本地化资源。比如，如果偏好语言为法语，警告消息的字符串将会从 Localizable.strings 里面的 fr.lproj 字典里面获取本地化的字符串（iOS 通过使用 NSLocalizedString 来创建一个本地化字符串）。

注意：*当 action-loc-key 的属性是一个字符串的时候它同样适用该形式。该字符串是当前所选择偏好语言在 Localizable.strings 里面的键。iOS 使用该键来过去警告信息窗口右边按钮的标题（即“action”按钮）。*

为了让你对上面所介绍的有一个更详细的了解，让我们举一个例子。提供者指定如下的字典作为 alert 属性值。

```
"alert" : { "loc-key" : "GAME PLAY REQUEST FORMAT", "loc-args" : [ "Jenna", "Frank"] },
```

当设备接收到该通知时，它使用”GAME_PLAY_REQUEST_FORMAT”作为键在 Localizable.strings 文件里面当前偏好语言的.lproj 字典里面查找匹配的字符串。假设当前 Localizable.strings 条目包含以下信息：

```
"GAME PLAY REQUEST FORMAT" = "%@ and %@ have invited you to play Monopoly";
```

设备将会显示一个警告信息为“Jenna and Frank have invited you to play Monopoly”。

除了%@指定的格式，你可以使用%n$@格式来指定子字符串变量的位置。n 代表了 loc-args 数组里面变量的下标（从 1 开始算起）。（同样可以使用%%来指定一个百分号（%））。 所以如果 Localizable.strings 的条目为：

```
"GAME PLAY REQUEST FORMAT" = "%2$@ and %1$@ have invited you to play Monopoly";
```

设备将会显示一个警告信息为“Frank and Jenna have invited you to play Monopoly”。

关于通知负载使用 loc-key 和 loc-args 属性的更完整的例子，参阅“JSON 负载例子”部分的最后一个例子。为了了解更多关于 iOS 上面的国际化工作，参阅 iOS App Programming Guid 的“高级 App 提示”部分。关于国际化的通用信息，参阅“Internationalization Programming Topics”。而 String Programming Guide 的”字符串对象格式”部分介绍了字符串格式。

注意：*你应该只在你需要的时候使用 loc-key 和 loc-args 属性--通常在 alert 字典里面。这些属性的值，尤其是当它们是一个长字符串的时候，可能会使用更多的带宽从而造成性能影响。大部分应用程序都不需要使用到这些属性，因为它们的字符串通常在服务器端就已经本地化了（即隐式“本地化”）。*

3.5.2 JSON负载示例

以下通知负载的示例举例说明了表 3-1 所列举的属性。名为”acme”的属性是一个自定义负载数据的例子。该示例包含了可读性的空白符和换行符。为了提高性能，提供者应该忽略空白符和换行符。

示例 1：以下负载包含哦一个简单的 aps 字典，采用默认的警告按钮的提示信息（关闭按钮和查看按钮）。它使用字符串而不是字典作为 alert 的值。该负载同样包含了一个自定义的属性数组。

```
{
    "aps" : { "alert" : "Message received from Bob" },
    "acme2" : [ "bang",  "whiz" ]
}
```

示例 2：该示例的负载包含了一个 aps 的字典，指定设备显示一个警告消息并在左边包含一个关闭按钮和右边显示一个本地化的”action”按钮。在该例中，”PLAY”被作为键使用来从 Localizable.strings 文件里面当前偏好语言的字典里面获取对应的“Play”的字符串。aps 字典同样要求应用程序的图标显示数字 5。

```
 {
       "aps" : {
                  "alert" : {
                            "body" : "Bob wants to play poker",
                            "action-loc-key" : "PLAY"
                            },
                 "badge" : 5,
               },
      "acme1" : "bar",
      "acme2" : [ "bang",  "whiz" ]
}
```

示例 3：该示例的负载指定设备应用显示一个警告信息并包含关闭按钮和查看按钮。同时它要求应用程序的图标显示数字 9，并在通知显示的时候播放主目录厦门的 bingbong.aiff 音频文件。

```
{
    "aps" : {
        "alert" : "You got your emails.",
        "badge" : 9,
        "sound" : "bingbong.aiff"
    },
    "acme1" : "bar",
    "acme2" : 42
}
```

示例 4：该示例的负载主要关注的是使用 alert 字典里面的 loc-key 和 loc-args 子属性来从应用程序的主目录下面获取一个本地化的字符串并根据 loc-args 选择合适的子变量位于正确的位置。它同样指定了一个自定义声音文件 chime，并包含了一个自定义属性。

```
{
    "aps" : {
        "alert" : { "loc-key" : "GAME PLAY REQUEST FORMAT", "loc-args" : [ "Jenna",
"Frank"] },
        "sound" : "chime",
    },
    "acme" : "foo"
}
```

示例 5：下面的示例显示了一个空的 aps 字典，因为 badge 属性被隐藏了，所以当前应用程序图标的任何数字都会被移除。而自定义属性 acme2 是一个包含两个整形的数组。

```
{
    "aps" : {
    },
    "acme2" : [ 5,  8 ]
}
```

记住，为了更好的性能，你应该在把字符串添加到通知里面之前尽可能的去除空白字符和换行字符。

第四章　配置和开发

4.1　沙箱和产品环境

为了开发并展开客户端/服务器应用程序的各方面，你必须从 Dev Center 里面获取合适的 SSL 证书。每个证书被限制匹配单独一个应用程序，并且由应用程序的 Bundle ID 标识。每个证书同时只能是两个开发环境中的一种，根据不同的开发环境设置不同的 IP 地址。

- **沙箱环境（Sandbox）**：沙箱环境主要用于初始开发环境，并测试提供者的应用。它提供了和产品环境一样的测试服务，尽管它拥有数量比较少的服务器单元。沙箱环境同时还充当了虚拟设备，并内嵌模拟终端到终端的测试。

  可以通过 gateway.sandbox.push.apple.com 的 TCP 端口 2195 访问沙箱环境地址。
- **产品环境（Production）**：当你编译提应用的发布版本的时候，请改用产品环境。使用产品环境的应用必须严格符合苹果的可靠性标准。

  可以通过 gateway.push.apple.com 的 TCP 端口 2195 来访问产品环境。

你必须为沙箱环境和产品环境获取独立的 SSL 证书。你获取的证书通过由推送通知接受的应用程序的标示符来标记。标标示符通常为应用程序的 Bundle ID。当为某个环境创建一个配置证书的文件时，相应的要求会被自动的添加到配置文件里面，包含了推送通知指定的权利和<aps-environment>。这两个证书通常称为 Development 和 Distribution。而 Distribution 证书通常用于你把应用上传到 App Store 的时候。

Mac OS X 注意：*Mac OS X 证书的权利是 com.apple.developer.aps-environment。*

你可以在 Xcode 里面通过选中的代码签名证书来确定你当前所使用的环境。如果你看到字样”iPhone Developer:Firstname Lastname”的出现在证书的名称里面，那么你当前正处于沙箱环境下。如果你看到一个”iPhone Distribution: Companyname”的字样出现在证书名称里面，那么你当前正使用产品环境。通常最好的办法是你在 Xcode 中创建一个发现版本的配置以帮助你进一步的区别环境。

尽管 SSL 证书不包含再配置证书里面，但是出于相关 SSL 证书和应用唯一标示符的原因<aps-environment>同样被添加到配置证书里面。

4.2 配置过程

在任何 iOS 应用开发里面，开发人员总是在团队里面扮演某一角色：团队经理，团队管理员，和团队成员。iPhone 开发证书和配置文件由不同角色完成。通常团队经理是团队里面唯一可以创建 Development SSL 证书（沙箱环境）和 Distribution SSL 证书（产品环境）的人。而团队经理和团队管理员都可以创建 Development 和 Distribution 的配置文件。团队成员只能下载并安装证书和配置文件（！这里要区别好证书和配置文件的概念：证书，即 Certificates，它主要是使用 Access Key Chain 来生成的一个签名密钥。而配置文件，即 Provisioning profiles，它主要是根据不同的 App 来生成的针对性的代码签名文件。二者区别是一般一个开发者账户只能有一个 Certificates（而 Development 一个，Distribtuion 一个），但是可以有多个 Provisioning profiles 文件，且这些 Provisioning profiles 通常创建的时候都使用同一个 Certrificates）。以下部分详细介绍该过程。

注意：*iOS Provisioning Portal给所有iOS开发者提供了一个使用文档和一系列的视频来介绍证书和配置文件的创建过程。以下部分侧重在APNs相关的证书和配置文件的创建过程。为了访问Portal，iOS开发者应该打开iOS Dev Center(*http://developer.apple.com/devcenter/ios*)，登陆，并点击访问iOS Provisioning Portal主页（即右上角的链接）。*

4.2.1 创建SSL证书和密钥

在 iOS Dev Center 的 iOS Provisioning Portal 里面，团队经理选中相应 APNs 的 AppID。他需要完成以下步骤来创建 SSL 证书：

1. 点击窗口左边侧边栏的 App IDs。

   将会跳转到一个显示当前合法应用 IDs 的页面。每个应用的 ID 前面包含了它的 Bundle ID，而在 Bundle ID 之前包含了苹果生成的十位字符串。团队管理员必须输入 Bundle ID。对于每个证书，它必须匹配某一特定的 Bundle ID，你不能使用“wildcart”的应用 ID。即要使用推送通知的应用的 Bundle ID 不能使用“*”来创建。

2. 找到需要创建 SSL 证书的应用 ID(和 Development Provisioning Profile 相关)，并单击 Configure 按钮。

   你必须确保该 App ID 下面的苹果推送通知服务栏下面显示“可用”状态才能为

该应用配置 APNs 证书。

3. 在 App ID 配置页面，检查已经勾选了“Enable for Apple Push Notification service”复选框，并点击下面的“Configure”按钮。

   单击该按钮将会启动 APNs 助理，它会通过一系列步骤来引导完成操作。

4. 第一步要求你启动 Keychain Access 应用并生成一个 Certificate Signing Request(CSR)文件。

   下面的介绍来自证书助理。当你完成生成 CSR 后，点击 Keychain Access 上面 Continue 来返回到 APNs Assistant。

   当你创建一个 CSR 时，Keychain Access 生成一个私有和公有的加密密钥对。而私有的密钥会默认被放入你的 Login keychain 里面。而公有的密钥包含 CSR 会发送到配置服务器。当配置服务器把证书发回给你的时候，证书里面的其中一个条目将是公有密钥。

5. 在 Submit Certificate Signing Request 面板上，单击 Choose File。导航到你之前创建的 CSR 文件的地方并选中它。

6. 单击 Generate 按钮。

   在显示 Generate Your Certificate 面板的同时，Assistant 会配置并生成你的客户端 SSL 证书。如果生成成功，它将会显示信息“Your APNs Certificate has been generated.”。单击 Continue 来进入下一步操作。

7. 在下一个面板，单击 Download Now 按钮来下载证书文件到本地文件夹。找到文件下载的地方并双击证书文件（该证书文件包含一个.cer 的扩展名）来把它安装到你的 keychain 里面。当完成后，单击 APNs Assistant 上面的 Done 按钮。双击加载启动 Keychain Access。确保你已经把刚才的证书安装到了你用于开发的电脑的 login keychain 里面。在 Keychain Access 里面，确保你的证书使用的 ID 匹配你应用的 Bundle ID。APNs 的 SSL 证书应该安装到你的通知服务上面。

当你返回到 iOS Dev Center Portal 的 Configure App ID 页面完成这些步骤后，你的证书应该会变成绿色，并且显示“Enabled”。

为了给产品环境创建一个证书，重复上述步骤，但记住选中产品证书的应用 ID。

4.2.2 创建并安装配置证书

团队经理或团队管理器接下来必须创建在服务器用于远程通知开发的配置证书（Development 或 Distribution）。配置文件就是一个集合，它囊括了和应用相关的开发者和开发团队验证过的设备并使用这些设备来测试应用程序。配置文件包含了证书、设备标示符、应用的 Bundle ID、和所有权利，包括<aps-environment>。所有团队程序需要在运行并测试他们应用代码的设备上面安装该配置文件。

注意：*关于创建配置文件的详细解析参考编程用户指南。*

为了下载并安装配置文件，团队程序必须完成以下步骤：

1. 进入 iOS Dev Center 的 Provisioning portal 页面。
2. 创建一个新的配置文件，并包含你注册用于 APNs 的 App ID。
3. 在你下载这个新的配置文件之前修改任何已存在的配置文件。

   你必须修改配置文件的一些细微部分（比如切换选项）来生成一个新的配置文件。如果配置文件并没有“受损（dirtied）”，你不应该给以该原始配置文件任何推送的权利。
4. 找到文件的下载目录，把该配置文件（通常是一个.mobileprovision 扩展文件）拖拉到 Xcode 或 iTunes 应用程序的图标上面。

   可选的，你也可以把配置文件复制到~/Library/MobileDevice/Provisioning Profiles 目录。如果当前目录不存在则生成一个新目录。
5. 验证该配置文件的权利是否正确。为此，使用文本编辑器打开.mobileprovision 文件。该文件的内容是一个 XML 的文本。查看在 aps-environment 键的位置的字典值。对于一个开发模式的配置文件，该值应该是 development；而对应发布模式的配置文件，该字符值应该是 production。
6. 在 Xcode 的 Organizer 窗口，查看 Provisioning Profiles 部分，并确认证书已经安装到你的设备上面。

   当你编译该工程的时候，二进制文件现在使用证书的私有密钥签名。

4.2.3 安装SSL证书和密钥到你的服务器上面

你必须安装 SSL 发布证书和你之前获取的私有加密密钥到需要运行提供者代码的服务器上，从该服务器连接到 APNs 的沙箱或产品环境。为此，需要完成以下步骤：

1. 打开 Keychain Access 实体并单击左边面板的 My Certificates 类别。
2. 找到你将要安装的证书，并打开相应内容。

   你将会看到证书和私有密钥。
3. 选中证书和密钥，选中 File > Export Items，并把它们导出个人信息交换文件（.p12）。
4. 如果服务器采用 Buby 或 Perl 语言来实现的话，那么它们更容易处理个人信息交换格式的文件。为了把证书转换为该格式，需要完成以下步骤：
   a) 在 Keychain Access 里面，选中相应证书并选择 File > Export Items。选择个人信息交换格式选项（.p12）。选择一个保存地址，并单击 Save 按钮。
   b) 加载终端应用，并输入如下的命令：

      openssl pkcs12 -in *CertificateName*.p12 -out *CertificateName.*pem -nodes
5. 拷贝.pem 证书文件到新的电脑并安装它到合适的地方。

第五章　提供者与APNs间通信

本章描述了提供者用于和 APNs 通信的接口，并讨论提供者需要完成的一些功能。

5.1　一般提供者要求

提供者和 APNs 之间的通信采用二进制接口。该提供者接口是一个高速、高质量的接口。它使用 TCP 套接字的数据流来联接二进制内容。该二进制接口是异步的。

产品环境下的二进制接口为 gateway.push.apple.com，端口号为 2195。而沙箱环境下的二进制接口为 gateway.sandbox.push.apple.com，端口号为 2195。你可以同时建立多个稳定链接到同样网关或多个网关实例。

对于每个接口，你必须使用 TLS(或 SSL)的安全通信通道。建立通信的过程需要使用到 iOS Provisioning Portal 提供的 SSL 证书（参阅“配置和开发环境-Provisioning and Development”）。为了建立稳定的可靠提供者标示符，你必须把该证书通过点对点的认证方式传递给 APNs 服务器。

注意：*为了和 APNs 建立稳定的 TLS 会话，一个 Entrust Secure CA 根证书必须安装到提供者的服务器之上。如果服务器运行 Mac OS X，那么该证书已经在 keychain 里面了。在其他系统，证书可能是不可用的。你可以从 Entrust SSL Certificates 网站下载该证书。*

你同样应该保持一份连接多个通知的 APNs 的链接。APNs 会把多次并重复尝试建立的连接是为 DOS 攻击服务而拒绝建立链接。一旦发生异常，APNs 关闭发生异常的连接。

作为提供者，你负责完成以下和推送通过相关的事情：

- 你必须组织并构造通知负载（参阅“通知负载”部分）。
- 你负责提供显示在应用程序图标上面的数字。
- 你应该经常的连接到反馈服务器，并获取多次尝试建立连接而失败的设备列表。然后停止把通知发送到这些应用的相关设备上面。参阅“反馈服务”获取更多详细介绍。

如果你试图让你的推送通知支持多种语言，但又不想使用负载的 aps 字典里面的 loc-key 和 loc-args 属性来获取本地化的警告字符串，那么你必须在服务器端根据客户端的偏好语言本地化该提示信息。为此，你需要确定当前客户端所选中的偏好语言。

“调度、注册、和处理通知”部分描述了如何获得该信息。参阅“通知负载”部分来查看和 loc-key 和 loc-args 相关的属性。

5.2 二进制接口和通知格式

二进制接口占用了一个 TCP 套接字来发送自然格式的二进制内容数据流。为了达到最佳的性能，你必须在同一个接口批量化的传输多个通知内容，或者显式使用 TCP/IP Nagle 算法。

该接口接受两种格式的通知包，一种是简单格式，而另外一种是增强格式，增强格式解决了简单格式的地址问题。

- **通知过期** APNs 拥有一个存储-转发特性，它保持最近要发送给应用特定设备的通知。如果该设备离线一段时间，那么 APNs 将会在设备下次在线的时候才把通知发送给该设备。对于简单格式，通知将会被无条件的发送给设备而不考虑通知相关性问题。换言之，该通知可以随着时间的推移而变为“过时”了。增强格式包含了一个过期值指示该通知的有效期。当有效时间过期时，APNs 把存储-转发里面的相应通知删除掉。
- **错误响应** 在简单格式下，如果你发送的通知是畸形的话（比如，有效负载超过规定的限制），APNs 将会切断连接。它不会给出拒绝通知的原因。增强格式可以让提供者给通知打上任意的标记符。如果发送异常，APNs 将会返回该通知的标记和一个相关的错误码。该响应可以让提供者定位问题并修正畸形的通知。

  对于大部分提供者，我们建议采用增强格式的通知。

因为增强格式的通知很多地方和简单格式相同，所以首先让我们看看简单格式的通知。图 5-1 显示简单格式：

Figure 5-1 Simple notification format

第一个字节代表命令值 0。设备令牌的长度和负载长度必须采用网络字节序（即大端）。此外，你还应该把设备令牌编码为二进制格式。负载必须不超过 256 字节，

且不能为空结尾。

列表 5-1 给出了一个示例函数，该函数通过二进制接口采用简单的格式发送通知给 APNs。该示例假设已经和 gateway.push.apple.com（或 gateway.sandbox.push.apple.com）建立了 SSL 连接，并完成了端到端的验证。

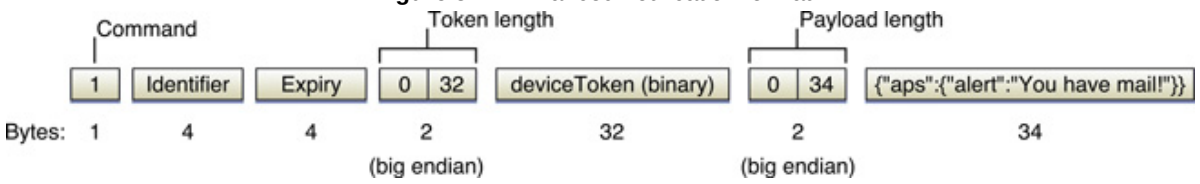
Listing 5-1 Sending a notification in the simple format via the binary interface

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff, size_t
payloadLength)

{

    bool rtn = false;

    if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)

    {

        uint8_t command = 0; /* command number */

        char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint16_t) +

            DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];

        /* message format is, |COMMAND|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */

        char *binaryMessagePt = binaryMessageBuff;

        uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);

        uint16_t networkOrderPayloadLength = htons(payloadLength);


        /* command */

        *binaryMessagePt++ = command;


       /* token length network order */

        memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16_t));

        binaryMessagePt += sizeof(uint16_t);


        /* device token */

        memcpy(binaryMessagePt, deviceTokenBinary, DEVICE_BINARY_SIZE);

        binaryMessagePt += DEVICE_BINARY_SIZE;


        /* payload length network order */

        memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16_t));

        binaryMessagePt += sizeof(uint16_t);
```

```
        /* payload */
        memcpy(binaryMessagePt, payloadBuff, payloadLength);
        binaryMessagePt += payloadLength;
        if (SSL_write(sslPtr, binaryMessageBuff, (binaryMessagePt - binaryMessageBuff)) >
0)
            rtn = true;
    }
    return rtn;
}
```

图 5-2 显示了增强格式的通知数据包。该格式下，如果 APNs 接收到一个非法的命令，那么它在关闭连接前会发送错误响应。

Figure 5-2 Enhanced notification format

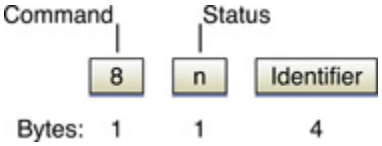


增强格式通知的第一个字节是命令为 1 的值。该格式新增了两个新的字段，即 identifier 和 expiry 值，他们都是四个字节长度（其他部分和简单格式的一样）。

- **Identifier** -- 一个可以标识通知的任意值。如果 APNs 无法正常解析通知的时候，它返回的错误信息里面包含该通知的标示符。
- **Expiry** -- 一个符合 UNIX 纪元日期秒数（UTC），它标记了通知何时失效且可以丢弃掉。expiry 值采用网络序（即大端）。如果 expiry 的值为正值，APNs 起码尝试发送该通知一次。你可以指定该值为 0 或负数来要求 APNs 不要存储该通知。

如果你发送了一个通知，而 APNs 检测到该通知是畸形的获取存在其他错误，那么 APNs 将会在关闭连接之前返回一个错误信息（如果没有错误发送，APNs 不会返回任何信息）。图 5-3 描述了错误响应信息。

Figure 5-3 Format of error-response packet



该数据包包含了一个命令值 8，后面跟着是状态码和通知标示符，而该通知标示符是由提供者在发送通知的时候指定的任意值。表 5-1 列出了所有可能的状态码和它

们表示的信息。

Table 5-1 Codes in error-response packet

Status code	Description
0	No errors encountered
1	Processing error
2	Missing device token
3	Missing topic
4	Missing payload
5	Invalid token size
6	Invalid topic size
7	Invalid payload size
8	Invalid token
255	None (unknown)

列表 5-2 修改列表 5-1 的代码来发送 APNs 前构建一个增强格式的通知。类似之前的示例，它假设已经和 gateway.push.apple.com(或 gateway.sandbox.push.apple.com)建立了稳定的 SSL 连接，并完成了端到端的验证。

Listing 5-2 Sending a notification in the enhanced format via the binary interface

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff, size_t
payloadLength)

{

  bool rtn = false;

  if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)

  {

     uint8_t command = 1; /* command number */

     char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint32_t) + sizeof(uint32_t) +
sizeof(uint16_t) +

        DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];

     /* message format is, |COMMAND|ID|EXPIRY|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */

     char *binaryMessagePt = binaryMessageBuff;

     uint32_t whicheverOrderIWantToGetBackInAErrorResponse_ID = 1234;

     uint32_t networkOrderExpiryEpochUTC = htonl(time(NULL)+86400); // expire message
if not delivered in 1 day

     uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);
```

```
     uint16 t networkOrderPayloadLength = htons(payloadLength);


     /* command */

     *binaryMessagePt++ = command;


    /* provider preference ordered ID */

    memcpy(binaryMessagePt, &whicheverOrderIWantToGetBackInAErrorResponse ID,
sizeof(uint32 t));

    binaryMessagePt += sizeof(uint32 t);


    /* expiry date network order */

    memcpy(binaryMessagePt, &networkOrderExpiryEpochUTC, sizeof(uint32 t));

    binaryMessagePt += sizeof(uint32 t);


    /* token length network order */

     memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16 t));

     binaryMessagePt += sizeof(uint16 t);


     /* device token */

     memcpy(binaryMessagePt, deviceTokenBinary, DEVICE BINARY SIZE);

     binaryMessagePt += DEVICE BINARY SIZE;


     /* payload length network order */

     memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16 t));

     binaryMessagePt += sizeof(uint16 t);


     /* payload */

     memcpy(binaryMessagePt, payloadBuff, payloadLength);

     binaryMessagePt += payloadLength;

     if (SSL write(sslPtr, binaryMessageBuff, (binaryMessagePt - binaryMessageBuff)) >
0)

         rtn = true;

  }
```

```
    return rtn;
}
```

需要注意的是同一设备的同一应用在产品环境和开发环境下的设备令牌是不一样的。

5.3 反馈服务

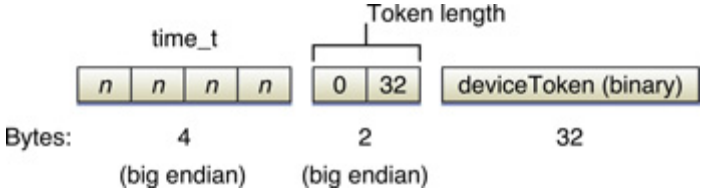
如果一个提供者试图把一个通知发送到某一个设备的应用，而该应用已经不存在该设备（通常被用户卸载），该设备将会反馈给苹果推送通知服务（APNs）。这通常发送在用户把设备的应用卸载了的情况。如果设备报告传递通知失败的信息给 APNs，那么 APNs 需要以某种方式来通知提供者以便他们停止发送通知到该设备的应用上。这样可以减少不必要的流量，并提供系统性能。

为此 APNs 包含了一个反馈服务，由 APNs 持续的更新一个传递通知失败的设备列表。列表里面的设备由二进制的设备令牌标识别。提供者应该周期性的查询该反馈接口并获取他们应用的该设备列表。然后验证那些设备已经从它应用里面失效了，提供者应该停止再发送通知到这些设备上面。

访问反馈服务接口和访问用于发送通知的二进制接口类似。你可以通过 feedback.push.apple.com 的端口号 2196 来访问产品环境的反馈服务。和发送通知的二进制接口类似，你必须使用 TLS(或 SSL)来和 APNs 建立文档的安全通信渠道。用于建立安全通信连接的 SSL 证书和用于建立推送通知通信连接的 SSL 证书一样。为了标识一个可信的提供者，你必须把该证书通过端对端验证的连接发送给 APNs。

一旦连接建立，传输立即进行。你不需要发送任何的命令。你需要读取由反馈服务器写出的字节流直到没有任何数据为止。收到的数据是按照如下的格式组织的内容：

Figure 5-4 Binary format of a feedback tuple



Timestamp	A timestamp (as a four-byte `time_t` value) indicating when the APNs determined that the application no longer exists on the device. This value, which is in network order, represents the seconds since 1970, anchored to UTC. You should use the timestamp to determine if the application on the device re-registered with

	your service since the moment the device token was recorded on the feedback service. If it hasn't, you should cease sending push notifications to the device.
Token length	The length of the device token as a two-byte integer value in network order.
Device token	The device token in binary format.

注意：APNs 时刻监听提供者频繁的查询反馈接口请求，以便提供者可以停止发送通知到这些不存在设备的应用上面。

结束语

说到 iOS 上面的通知，我们肯定想到的是 Push 通知了。因为现在很多应用基本都会有服务器端的 Push 服务，这样一来可以在用户没有启动应用的时候告知用户他们感兴趣的事儿，二来也是很重要的一点：当用户很久没有打开应用的时候，我们可以 Push 提醒一下用户，以便增加应用的活跃度。而本人对后一种做法持保留意见。

自从 iOS 5 出来，苹果的 Push 服务可以说相当 Perfect 了。苹果从 Android 里面学到很多易用的特性，而且摒弃了之前 Push 打断用户行为的流氓性。但是本人比较担心 2012 有可能是 Push 泛滥的一年。当大家有事没事都 Push 给用户一些无用的信息，那 Push 在用户眼里就已经失去价值了。所以还是希望你在开发你应用的时候慎用 Push 服务。

补充一下，本篇翻译的基础是旧版的 iOS push 特性，也就是你在文章里面看到的很多示例和介绍图片都还没涉及到最新的 Push 样式。但是因为本篇所述的都是 Push 的原理和通信过程，样式篇幅反而不大。所以你若通读了全篇，基本就了解 Push 过程了。而且 Push 的 Payload 是一个 Json 格式，在 iOS 5 没有改变。所以本篇也有一种“以不变应万变”的味道。

顺便给大家推荐一个新手必备的Push的网站：**http://www.easyapns.com/**

最后，本文在翻译过程中发现很多地方直译成中文比较晦涩，所以采用了意译的方式，这不可避免的造成有一些地方可能和原文有一定的出入，所以如果你阅读的时候发现有任何的错误都可以给我发邮件:xyl.layne@gmail.com。

最后可以关注我微博大家一起沟通交流学习。

微博地址：**http://weibo.com/u/1826448972**

推荐资源

- **核心动画编程指南【Core Animation Programming Guide】**

下载地址:
http://www.cocoachina.com/bbs/read.php?tid=84461

- **多线程编程指南【Threading Programming Guide】**

下载地址:
http://www.cocoachina.com/bbs/read.php?tid=87592

- **Blocks 编程要点【Blocks Programming Topics】**

下载地址:
http://www.cocoachina.com/bbs/read.php?tid=87593

- **Instruments 用户指南【Instruments User Guide】**

下载地址:
http://www.cocoachina.com/bbs/read.php?tid=92026