Local および Push Notification プログラミングガイド

Developer

目次

図、表、リスト

ローカル通知、リモート通知について

「ユーザ通知」には、ローカル通知とリモート通知の2種類があります（リモート通知のことをプッシュ通知とも言います）。いずれも、フォアグラウンドで動作していないアプリケーションが、ユーザに何らかの情報を伝えるための仕組みです。その情報は、メッセージ、間近のカレンダーイベント予定、またはリモートサーバ上の新しいデータの存在などの可能性があります。オペレーティングシステムによって表示される場合、ローカル通知とリモート通知の見た目やサウンドは同じです。それらの通知は、警告メッセージまたはアプリケーションアイコンのバッジを表示できます。警告や数字付きバッジを表示するときに、サウンドを再生することもできます。

アプリケーションからのメッセージ、イベント、あるいはそのほかのデータがあることがユーザに通知されると、ユーザはアプリケーションを起動してその詳細を表示できます。また、通知を無視することも選べますが、その場合、アプリケーションはアクティブになりません。

> **注意:** リモート通知やローカル通知に、通知のブロードキャスト（`NSNotificationCenter`）やキー値監視通知との関連は*ありません*。

はじめに

ローカル通知とリモート通知にはいくつか、知っておくべき重要な問題があります。

ローカル通知もリモート通知も、同じ課題に対処するための仕組みである

フォアグラウンドでは、1度に1つのアプリケーションしかアクティブになれません。多くのアプリケーションは、時間ベースまたは相互接続された環境で動作しているため、アプリケーションがフォアグラウンドにないときに、ユーザが関心を持っているイベントが発生する可能性があります。ローカル通知およびリモート通知により、アプリケーションは、これらのイベントが発生したことをユーザに通知できるようになります。

関連する章: “ローカル通知およびリモート通知の詳細”（11 ページ）

ローカル通知とリモート通知は発生源が異なる

ローカル通知とリモート通知の設計上のニーズは異なります。ローカル通知は、アプリケーション自身がスケジューリングし、送信します。リモート通知（あるいはプッシュ通知）は、アプリケーションが動作するデバイスやMac以外から届きます。リモート通知は、表示すべきメッセージやダウンロードすべきデータがあるときに、リモートサーバ（アプリケーションのプロバイダ）上で発生して、デバイス上のアプリケーションに（Apple Push Notificationサービス経由で）配信（プッシュ）されます。

関連する章: “ローカル通知およびリモート通知の詳細”（11 ページ）

アプリケーションは、ローカル/リモート通知の両方を登録、スケジューリング、操作する

システムが後でローカル通知を配信するようにしたい場合、アプリケーションは通知のタイプを登録し（iOS 8以降）、ローカル通知オブジェクトを（`UILocalNotification`または`NSUserNotification`で）生成し、配信する日時を割り当て、表示の詳細を指定した上で、配信するようスケジューリングします。アプリケーションがリモート通知を受け取るためには、通知のタイプを登録し、オペレーティングシステムから取得したデバイストークンをプロバイダに渡す必要があります。

オペレーティングシステムは、ローカル通知やリモート通知を配信した時点で、対象アプリケーションがフォアグラウンドで動作していなければ、警告、数字付きバッジアイコン、サウンドの形でユーザに伝えます。通知警告が提示され、ユーザがアクションボタンをタップまたはクリックすると（またはアクションスライダを動かすと）、アプリケーションが起動し、メソッドが呼び出され、ローカル通知オブジェクトまたはリモート通知ペイロードが渡されます。通知が配信されたときにアプリケーションがフォアグラウンドで実行中であれば、アプリケーションデリゲートはローカル通知またはリモート通知を受け取ります。

iOS 8以降、通知にカスタムアクションを組み込めるようになりました。また、ロケーションベースの通知を、ユーザが所定の場所に到着したときに送信できるようになりました。

関連する章: “ユーザ通知の登録、スケジューリング、対処”（16 ページ）

Apple Push Notificationサービスはリモート通知のゲートウェイとして機能する

Apple Push Notificationサービス（APNs）により、これらの通知を受け取るための登録を行ったアプリケーションを持つデバイスに、リモート通知が伝達されます。各デバイスは、このサービスとの間で認証済みの暗号化されたIP接続を確立し、この永続的な接続を介して通知を受信します。プロバイダは、対象となるクライアントアプリケーション宛ての受信データを監視している間、永続的でセキュアなチャネルを介してAPNsに接続します。アプリケーション宛ての新しいデータを受信すると、プロバイダは通知を作成してこのチャンネルを介してAPNsに送信します。APNsは、その通知をターゲットデバイスに配信（プッシュ）します。

関連する章: “Apple Push Notificationサービス”（38 ページ）

リモート通知のセキュリティ認証情報を取得する必要がある

リモート通知を行うためにアプリケーションのプロバイダ側を開発して配備するには、Member CenterからSSL証明書を取得する必要があります。証明書は、バンドルIDで識別される1つのアプリケーションに対して1つと限定されています。また、証明書は開発環境用と製品環境用のいずれかに限定されています。これらの環境は固有のIPアドレスを割り当てられており、固有の証明書を必要とします。また、これらの環境のそれぞれにプロビジョニングプロファイルを取得する必要もあります。

関連する章: “プロビジョニングおよび開発”（55 ページ）

プロバイダによるバイナリインターフェイス経由でのAPNsとの通信

バイナリインターフェイスは非同期であり、リモート通知をバイナリコンテンツとしてAPNsに送信するために、TCPソケットを介するストリーミング設計を用います。開発環境と製品環境には別々のインターフェイスがあり、それぞれ固有のアドレスとポートを持ちます。インターフェイスごとに、TLS（またはSSL）およびSSL証明書を使用して、セキュアな通信チャンネルを確立する必要があります。プロバイダは、それぞれの送信通知を作成し、このチャンネル経由でAPNsに送信します。

APNsには、配信に失敗したデバイス（つまり、APNsがデバイス上のアプリケーションへリモート通知を配信できなかった対象デバイス）のアプリケーションごとのリストを管理するフィードバックサービスがあります。プロバイダは、定期的にフィードバックサービスに接続し、配信が失敗したデバイスへのリモート通知の送信を停止できるように、失敗が続いているデバイスを確認するべきです。

関連する章: "Apple Push Notificationサービス"（38 ページ）、"プロバイダとApple Push Notificationサービスの通信"（57 ページ）

必要事項

『*iOS App Programming Guide*』ではiOSアプリケーションの大まかな開発手順を説明しています。

ローカル通知、およびリモート通知のクライアント側を実装するには、iOSのアプリケーション開発に精通していることが前提になります。プロバイダ側の実装には、TLS/SSLおよびストリーミングソケットに関する知識が役立ちます。

関連項目

この技術の背景となる情報が以下の資料に載っています。

- 『*App Distribution Quick Start*』では、APNsを有効にする前の手続きとして、Xcode上でチームプロビジョニングプロファイルを生成する手順を説明しています。
- 『*App Distribution Guide*』では、APNsの設定など、XcodeやMember Centerでおこなうさまざまなタスクの手順を説明しています。
- 『*Entitlement Key Reference*』には、アプリケーションがリモート通知を受け取るために必要なエンタイトルメントに関する説明があります。

次の情報は、ローカル通知およびリモート通知の理解および実装の参考になります。

- `UILocalNotification`、`UIApplication`、`UIApplicationDelegate`のリファレンス文書では、iOSで動作するクライアントアプリケーション用のローカル通知およびリモート通知に関するAPIについて説明しています。
- `NSApplication`および`NSApplicationDelegate Protocol`のリファレンス文書では、OS X上で動作するクライアントアプリケーション用のリモート通知APIについて説明しています。
- 『*Security Overview*』では、iOSおよびOS Xの両システムで使われているセキュリティのテクノロジーと手法について説明しています。
- RFC 5246はTLSプロトコルの標準です。

  データプロバイダとApple Push Notificationサービス間のセキュアな通信には、Transport Layer Security（TLS）、またはその前身のSecure Sockets Layer（SSL）に関する知識が必要です。詳細については、これらの暗号プロトコルのオンラインまたは印刷版解説書を参照してください。

OS X上でウェブサイトを閲覧しているユーザにプッシュ通知を送る方法については、『*Notification Programming Guide for Websites*』の“Configuring Safari Push Notifications”を参照してください。

ローカル通知およびリモート通知の詳細

ローカル通知およびリモート通知の目的の本質は、アプリケーションがフォアグラウンドで実行中でないときに、ユーザに対して何らかの情報（メッセージ、次の予約など）があることの通知をアプリケーションが行えるようにすることです。ローカル通知とリモート通知の間の本質的な相違はシンプルです。

- ローカル通知は、アプリケーションによってスケジューリングされ、同じデバイスに配信されます。
- リモート通知（プッシュ通知とも呼ばれる）は、サーバによってApple Push Notificationサービスに送信され、そこからデバイスに配信（プッシュ）されます。

ユーザにとってはローカル通知もリモート通知も同じに見える

ユーザのもとには次のような方法で通知が届きます。

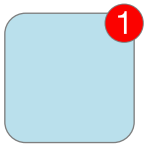
- 画面上の警告やバナー
- アプリケーションアイコン上のバッジ
- 警告、バナー、バッジに添えられたサウンド

ユーザにとってはローカル通知もリモート通知も、アプリケーションにとって着目するべき、何かが起こっていることを表します。

たとえばToDoリストを管理するアプリケーションを考えてみましょう。リストの各項目には、完了しなければならない日時があります。ユーザは、この期日に達するまで、特定の間隔で通知することをアプリケーションに要求できます。これを達成するため、アプリケーションはその日時にローカル通

知をスケジューリングします。アプリケーションは警告メッセージの代わりに、数字（1）付きバッジを指定します。予定の時間になると、iOSはサウンドを鳴らし、アプリケーションのアイコンの右上隅に数字付きバッジを表示します。その様子を図 1-1に示します。

図 1-1　数字付きバッジが表示されたアプリケーションアイコン（iOS）



ユーザはサウンドやバッジに気がつき、アプリケーションを起動してToDo項目を確認します。ユーザは、デバイスや、デバイスにインストールされているアプリケーションに対して、通知をどのように処理して欲しいか制御できます。アプリケーションごとにリモート通知のタイプ（アイコンのバッジ、警告メッセージ、警告音）を選択的に有効にしたり無効にしたりもできます。

アプリケーションにとっては、ローカル通知とリモート通知は違うものに見える

ローカル/リモート通知が届いたとき、アプリケーションが最前面にあれば、アプリケーションデリゲートの`application:didReceiveRemoteNotification:`メソッドまたは`application:didReceiveLocalNotification:`メソッドが呼び出されます。最前面にない、あるいはまったく動作していない場合は、アプリケーションデリゲートの`application:didFinishLaunchingWithOptions:`に渡されたオプション辞書の`UIApplicationLaunchOptionsLocalNotificationKey`キーまたは`UIApplicationLaunchOptionsRemoteNotificationKey`キーを調べて、通知の取り扱いを判断します。通知の取り扱いについて詳しくは、“ユーザ通知の登録、スケジューリング、対処”（16 ページ）を参照してください。

ローカル通知に関する追加情報

ローカル通知は、カレンダーアプリケーションやToDoリストアプリケーションを含む、時間ベースで動作するアプリケーションに最適です。iOSで許されている短い時間の間にバックグラウンドで動作するアプリケーションも、ローカル通知が役立つかもしれません。たとえば、メッセージやデータをサーバに依存するアプリケーションは、バックグラウンドでの動作中に、受信する項目をサーバにポーリングできます。表示すべきメッセージやダウンロードすべき更新があれば、必要に応じてデータを処理した後、ユーザにそのことを通知します。

ローカル通知は、次の3つの一般的な種類のプロパティを持つ、`UILocalNotification`または`NSUserNotification`のインスタンスです。

- **スケジューリング。**オペレーティングシステムから通知を配信する日時を指定する必要があります。これは、*配信日（fire date）*とも呼ばれます。ユーザが旅をするときなどにシステムが調整できるように、配信日を特定のタイムゾーンで修飾することができます。また、ある一定の間隔（毎週、毎月など）で、通知をスケジュールしなおすようにオペレーティングシステムにリクエストすることもできます。
- **通知のタイプ。**このプロパティには、警告メッセージ、デフォルトアクションボタンのタイトル、アプリケーションアイコンのバッジの数字、再生するサウンド、カスタムアクションのカテゴリ（iOS 8以降の場合）が含まれます。
- **カスタムデータ。**ローカル通知には、カスタムデータのユーザ情報辞書を含めることができます。

リスト 2-9 （30 ページ）では、プログラミングの面からこれらのプロパティの詳細を説明します。アプリケーションは、ローカル通知オブジェクトを作成したら、オペレーティングシステムに対してスケジューリングするか、すぐに提示するかのいずれかを行えます。

デバイス上の各アプリケーションは、64件までのローカル通知をスケジューリングできます。この制限を超えた場合、直近の64件を残して破棄するようになっています。同じ通知が繰り返し現れた場合、1回分の通知として扱われます。

リモート通知に関する追加情報

iOS/Macアプリケーションは、多くの場合、クライアント/サーバモデルに基づくより大きなアプリケーションの一部に過ぎません。クライアント側のアプリケーションはデバイスまたはコンピュータにインストールされます。一方、サーバ側のアプリケーションは、クライアントアプリケーションにデータを提供することが主な役割なので、「*プロバイダ*」と呼ばれます。クライアントアプリケーションは定期的にプロバイダに接続して、そのアプリケーション用の更新データをダウンロードします。電子メールアプリケーションやソーシャルネットワーキングアプリケーションは、このようなクライアントサーバモデルの例です。

しかし、ダウンロード可能な更新データがプロバイダにあるときに、アプリケーションがプロバイダに接続されていなかったり、デバイスまたはコンピュータ上で実行すらされていない場合はどうなるでしょうか？アプリケーションはどのようにして更新データのことを知るのでしょうか？リモート通知（あるいはプッシュ通知）がこの悩みを解決します。リモート通知はプロバイダがデバイスまたはコンピュータのオペレーティングシステムに配信した短いメッセージです。オペレーティングシステムはこれを受信すると、ダウンロード可能なデータ、表示可能なメッセージなどが存在することをクライアントアプリケーションのユーザに通知します。ユーザが（iOSで）この機能を有効にしていて、

アプリケーションが正しく登録されていれば、この通知はオペレーティングシステム（アプリケーションの場合もある）に配信されます。Apple Push Notificationサービス（略してAPNs）はリモート通知機能の基本テクノロジーです。

リモート通知は、デスクトップシステム上のバックグラウンドアプリケーションと同じ働きをしますが、オーバーヘッドが加わることはありません。現在フォアグラウンドで実行されていない（iOSの場合は動作していない）アプリケーションに対しては、間接的に通知が行われます。オペレーティングシステムは、アプリケーションの代わりにリモート通知を受け取って、ユーザに警告します。警告を受け、ユーザがアプリケーションの起動を選択すると、アプリケーションはプロバイダからデータをダウンロードします。通知を受信したときにアプリケーションが実行されている場合は、アプリケーションはその通知を直接処理することを選ぶこともできます。

Apple Push Notificationサービスは、デバイスやコンピュータにリモート通知を配信するためにリモート型の設計を使用しています。プッシュ型設計がその反対のプル型設計と異なるのは、通知の受け手が、能動的に更新情報をポーリングするのではなく、受動的に更新情報を検知する点です。プッシュ型設計によって、情報を広範囲かつタイムリーに配信することが可能になり、プル型設計につきもののスケーラビリティの問題もほとんどありません。APNsでは、リモート通知を実現するために永続的なIP接続を使用します。

リモート通知のほとんどは、ペイロード（ユーザへの通知方法を指定するAPNs定義のプロパティを含むプロパティリスト）で構成されています。パフォーマンスの理由から、ペイロードは意図的に小さくしています。ペイロードにカスタムプロパティを定義することもできますが、リモート通知による配信は保証されていないため、データ送信の目的でリモート通知メカニズムを使用してはいけません。ペイロードの詳細については“通知ペイロード”（46 ページ）を参照してください。

APNsは、デバイスまたはコンピュータ上のアプリケーション宛てにプロバイダから受信した最後の通知を保持しています。したがって、デバイスがオンラインになったときに、その通知をまだ受信していない場合、APNsは保存されている通知をデバイスに配信します。iOSが動作するデバイスは、Wi-Fi接続、携帯電話接続の両方でリモート通知を受け取ります。OS Xが動作するコンピュータは、Wi-Fi接続、Ethernet接続の両方でリモート通知を受け取ります。

Important: iOSで、Wi-Fi接続がリモート通知に使用されるのは、携帯電話接続が存在しない場合、またはデバイスがiPod touchの場合だけです。デバイスによっては、Wi-Fi経由で通知を受信するには、デバイスのディスプレイがオンになっている（つまり、スリープ状態でない）か、デバイスがプラグに接続されていなければなりません。これに対して、iPadでは、スリープ状態でもWi-Fiアクセスポイントと結ばれているので、リモート通知の配信が可能です。Wi-Fi無線により、すべての受信トラフィックに対して、ホストプロセッサのスリープが解除されます。

過度に通知を送信するとデバイスの電池寿命に悪影響があります。デバイスはネットワークにアクセスして通知を受信しなければならないからです。

アプリケーションにリモート通知機能を追加するには、iOS Developer Program、Mac Developer Programとも、Member Centerから適切な証明書を取得し、クライアント側とプロバイダ側の各アプリケーションに必要なコードを記述する必要があります。“プロビジョニングおよび開発”（55 ページ）では、プロビジョニングとセットアップの手順について説明します。また、“プロバイダとApple Push Notificationサービスの通信”（57 ページ）および“ユーザ通知の登録、スケジューリング、対処”（16 ページ）では、実装の詳細について説明します。

ユーザ通知の登録、スケジューリング、対処

ローカル通知やリモート通知を登録、スケジューリングし、また、これが届いたときに対処するため、iOS/OS Xアプリケーションが実行するべきタスクがいくつかあります。

iOSで通知のタイプを登録する

iOS 8以降、ローカル/リモート通知を利用するアプリケーションは、配送する通知のタイプを登録しなければならなくなりました。システムには、アプリケーションが表示する通知のタイプを、ユーザが制限できるようにする機能があります。ある通知のタイプがアプリケーション側で有効になっていなければ、通知ペイロードにそのタイプが指定されていても、システムがアイコンにバッジを添え、警告メッセージを表示し、警告サウンドを鳴らすことはありません。

iOSでは、`UIApplication`の`registerUserNotificationSettings:`メソッドで通知のタイプを登録します。通知のタイプは、通知を受け取ったときにアプリケーションが表示するUI要素、すなわち、アイコンに添えるバッジ、鳴らすサウンド、表示する警告に影響します。通知のタイプを何も登録しなければ、システムはリモート通知を黙って、すなわち画面に何も表示することなく、アプリケーションにプッシュします。リスト 2-1に通知のタイプを登録する方法を示します。

リスト 2-1　通知のタイプを登録するコード例

```
UIUserNotificationType types = UIUserNotificationTypeBadge |
             UIUserNotificationTypeSound | UIUserNotificationTypeAlert;

UIUserNotificationSettings *mySettings =
            [UIUserNotificationSettings settingsForTypes:types categories:nil];

[[UIApplication sharedApplication] registerUserNotificationSettings:mySettings];
```

リスト 2-1には`categories`という引数が使われています。カテゴリには、通知とともに表示するアクションをグループ化する働きがあります。詳しくは、“通知アクションを扱う（iOS）”（27 ページ）で説明します。

`registerUserNotificationSettings:`メソッドを初めて呼び出した時点で、iOSはダイアログを表示して、アプリケーションが登録したタイプの通知を画面に表示してよいか、許可を求めます。ユーザが応答すると、iOSは非同期に、`UIApplicationDelegate`オブジェクトの`application:didRegisterUserNotificationSettings:`メソッドを、コールバックとして呼び出します。その際、ユーザが許可した通知のタイプを表す`UIUserNotificationType`オブジェクトを、引数として渡します。

ユーザはいつでも「Settings」アプリケーションで、通知に関する設定を変更できます。アプリケーションから`registerUserNotificationSettings:`を呼び出せば、当該アプリケーションの設定画面がすぐに「Settings」アプリケーションに追加されます。ユーザは、通知の有効/無効を切り替えるほか、その表示位置や表示方法も修正できます。ユーザは初期設定をいつでも変更できるので、通知を表示する都度、`currentUserNotificationSettings`で設定を確認してください。

iOS 8以降、`registerUserNotificationSettings:`メソッドはローカル通知だけでなくリモート通知にも適用されるようになりました。アプリケーションが表示するリモート通知のタイプもこれで設定するようになったので、`registerForRemoteNotificationTypes:`は非推奨になっています。リモート通知について詳しくは、“リモート通知を登録する”（20 ページ）を参照してください。

ローカル通知をスケジューリングする

iOSでは、`UILocalNotification`オブジェクトを生成し、`UIApplication`の`scheduleLocalNotification:`メソッドで配送のスケジューリングをします。OS Xの場合は、（配送時刻も保持する）`NSUserNotification`オブジェクトを生成し、適切に配送する処理は`NSUserNotificationCenter`がおこないます（OS Xアプリケーションには`NSUserNotificationCenterDelegate`プロトコルを実装して、`NSUserNotificationCenter`オブジェクトのデフォルトの挙動をカスタマイズすることも可能）。

iOSでローカル通知を生成し、スケジューリングする手順を以下に示します。

1. iOS 8以降の場合、“iOSで通知のタイプを登録する”（16 ページ）で説明したように、通知のタイプを登録します（OS Xや旧iOSではリモート通知の場合のみ必要）。登録済みであれば、ユーザが許可した通知のタイプを`currentUserNotificationSettings`で確認できます。
2. `UILocalNotification`オブジェクトを割り当て、初期化します。
3. オペレーティングシステムが通知を配信しなければならない日時を設定します。これは、`fireDate`プロパティです。

   `timeZone`プロパティを現在のロケールの`NSTimeZone`オブジェクトに設定した場合、システムは、デバイスが異なるタイムゾーンに渡って移動したときに、自動的に配信日を調整します。（タイムゾーンは、指定されたカレンダーと日付の値をシステムが計算する、日付コンポーネント（日、月、時、年、分）の値に影響します）。

通知の配信が繰り返し行われるように（毎日、毎週、毎月など）スケジューリングすることもできます。

4. 警告、アイコンに添えるバッジ、サウンドを適切に設定して、ユーザの好みに合った形で通知を配送できるようにします（各タイプがどのような場合に適しているか、については"Notification Center"を参照）。
    - 警告には、メッセージを表すプロパティ（`alertBody`）、アクションボタンやスライダのタイトルを表すプロパティ（`alertAction`）があります。ユーザの言語設定に応じてローカライズした文字列を指定してください。通知をApple Watchに表示できる場合は、値を`alertTitle`プロパティに設定してください。
    - アイコン上にバッジとして数字を添える場合、その数字を`applicationIconBadgeNumber`プロパティで指定します。
    - サウンドを再生する場合、そのサウンドを`soundName`プロパティに割り当ててください。アプリケーションのメインバンドル（またはデータコンテナ）内にあるローカライズされていないカスタムサウンドのファイル名か、またはデフォルトのシステムサウンドを表す`UILocalNotificationDefaultSoundName`を指定します。サウンドは単独で使わず、警告メッセージまたはバッジを併用してください。
5. 必要ならば`userInfo`プロパティを使って、通知にカスタムデータを添えることができます。たとえば、CloudKitのレコードが変化した旨の通知には、当該レコードのIDを添えて、ハンドラがレコードを取得、更新できるようにするとよいでしょう。
6. iOS 8以降、必要ならば、ローカル通知にカスタムアクションを組み込んで、ユーザが実行できるようにすることも可能です（"通知アクションを扱う（iOS）"（27 ページ）を参照）。
7. ローカル通知の配信をスケジューリングします。

    ローカル通知のスケジュール設定は`scheduleLocalNotification:`メソッドでおこないます。アプリケーションは、`UILocalNotification`オブジェクトに指定されている配信日（fire date）を配信のタイミングとして使用します。あるいは、`presentLocalNotificationNow:`メソッドを呼び出して、すぐに通知を提示することもできます。

リスト2-2のメソッド例では、架空のToDoリストアプリケーションに設定されているToDo項目の期日が間近であることをユーザに知らせるために、通知の作成とスケジューリングを行っています。これについて、いくつかの注意すべき点を示します。`alertBody`、`alertAction`、`alertTitle`の各プロパティに関しては、ユーザによって設定されている言語環境に対応して、メインバンドル（`NSLocalizedString`マクロを介して）からローカライズされた文字列がフェッチされます。また、`userInfo`プロパティに割り当てられている辞書に、関連するToDo項目の名前も追加されます。

リスト 2-2　ローカル通知の作成、設定、スケジューリング

```
- (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];

    UILocalNotification *localNotif = [[UILocalNotification alloc] init];
    if (localNotif == nil)
        return;
    localNotif.fireDate = [itemDate
dateByAddingTimeIntervalInterval:-(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.alertTitle = NSLocalizedString(@"Item Due", nil);

    localNotif.soundName = UILocalNotificationDefaultSoundName;
    localNotif.applicationIconBadgeNumber = 1;

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

    [[UIApplication sharedApplication] scheduleLocalNotification:localNotif];
}
```

スケジューリングされた特定の通知をキャンセルするには、アプリケーションオブジェクトの`cancelLocalNotification:`を呼び出します。また、スケジューリングされたすべての通知をキャンセルするには、`cancelAllLocalNotifications`を呼び出します。これらのメソッドはともに、現在表示されている通知警告もプログラム上で閉じます。たとえば備忘通知は、ユーザがもう必要ないと思えば、いつでもキャンセルできるようにするべきでしょう。

ローカル通知が有用なケースとしては、アプリケーションがバックグラウンドで実行されているときに、ユーザが関心を持つようなメッセージ、データ、そのほかの項目を受信した場合が考えられます。その場合は、`UIApplication`の`presentLocalNotificationNow:`メソッドを使用して、すぐに通知を提示すべきです（iOSでは、アプリケーションがバックグラウンドで動作できる時間は限られています）。

OS Xの場合、ローカル通知を生成し、配送のスケジューリングをする、リスト 2-3のようなコードを記述してください。なお、アプリケーションが最前面にあれば、OS Xはローカル通知を配送しません。さらにOS Xのユーザは、「システム環境設定(System Preferences)」で、受け取る通知の設定を変更できます。

リスト 2-3　ローカル通知を生成、スケジューリングするコード例（OS X）

```
//Create a new local notification
NSUserNotification *notification = [[NSUserNotification alloc] init];
//Set the title of the notification
notification.title = @"My Title";
//Set the text of the notification
notification.informativeText = @"My Text";
//Schedule the notification to be delivered 20 seconds after execution
notification.deliveryDate = [NSDate dateWithTimeIntervalSinceNow:20];

//Get the default notification center and schedule delivery
[[NSUserNotificationCenter defaultUserNotificationCenter]
scheduleNotification:notification];
```

リモート通知を登録する

アプリケーションのプッシュプロバイダから送信されたリモート通知を受信するため、アプリケーションはiOS用のApple Push Notificationサービス（APNs）に登録する必要があります。iOS 8以降、登録は次の4段階でおこなうようになりました。

1. アプリケーションが対処できる通知のタイプを、registerUserNotificationSettings:で登録する。
2. APNs経由でプッシュ通知を受け取る旨を、アプリケーションのregisterForRemoteNotificationsメソッドで登録する。
3. 登録に成功した場合、サーバからアプリケーションデリゲートに返されるデバイストークンを保存しておく。失敗した場合も適切に対処する。
4. デバイストークンをアプリケーションのプッシュプロバイダに転送する。

（iOS 7では、手順1および2の代わりに、UIApplicationのregisterForRemoteNotificationTypes:メソッドで登録します。また、OS Xの場合は、NSApplicationのregisterForRemoteNotificationTypes:メソッドを実行します。）登録の過程で実行されるアクションを、図 3-3 （44 ページ）の"トークンの生成と共有" （43 ページ）に示します。

デバイストークンは変化することがあるので、アプリケーションは起動の都度、登録し直す必要があります。

登録に成功すると、APNsはデバイスにデバイストークンを返します。iOSは、application:didRegisterForRemoteNotificationsWithDeviceToken:メソッドでこのトークンをアプリケーションデリゲートに渡します。アプリケーションはこのトークンを、バイナリ形式にエンコードし、プロバイダに渡します。トークンを取得する際に問題が発生した場合、オペレーティングシステムはapplication:didFailToRegisterForRemoteNotificationsWithError:メソッド（OS Xの場合はapplication:didFailToRegisterForRemoteNotificationsWithError:メソッド）を呼び出してデリゲートに知らせます。このメソッドに渡されるNSErrorオブジェクトには、エラーの原因が明確に記述されています。たとえば、エラーは、プロビジョニングプロファイルのaps-environment値の誤りであるかもしれませんエラーは一時的な状態として見るだけにし、パースしようとはしないでください

> **iOSにおける注意事項:** 携帯電話接続またはWi-Fi接続が使用できない場合、application:didRegisterForRemoteNotificationsWithDeviceToken:メソッドもapplication:didFailToRegisterForRemoteNotificationsWithError:メソッドも呼び出されません。Wi-Fi接続では、これは、デバイスがポート5223経由で接続できない場合に発生することがあります。これが発生した場合、ユーザはこのポートがブロックされていないほかのWi-Fiネットワークに移動します。iPhoneまたはiPadでは、携帯電話データサービスが使用可能になるまで待機します。どちらの場合も、接続に成功すれば、いずれかのデリゲーションメソッドが呼び出されます。

アプリケーションが起動するたびに、デバイストークンを要求してそれをプロバイダに渡すことで、プロバイダが最新のデバイストークンを持つことを保証できます。これを怠ると、ユーザのデバイスに通知が届かないこともありえます。バックアップの作成元となったものとは異なるデバイスやコン

ピュータにバックアップを復元した場合（たとえば、新しいデバイスやコンピュータにデータをインポートした場合）、再び通知を受信するために、少なくとも一度はアプリケーションを起動する必要があります。バックアップデータを新しいデバイスやコンピュータに復元した場合や、オペレーティングシステムを再インストールした場合は、デバイストークンが変更されます。さらに、デバイストークンをキャッシュしてプロバイダに渡してはいけません。常に、デバイストークンは必要になったときにその都度システムから取得します。アプリケーションが以前に登録されていれば、`registerForRemoteNotifications`を呼び出すと、オペレーティングシステムは、余分なオーバヘッドを生じさせることなく、ただちにデバイストークンをデリゲートに渡します。さらに、デバイストークンが変化すれば、いつでもデリゲートメソッドが呼び出されうることに注意してください。アプリケーションの登録/再登録時だけではありません。

リスト 2-4は、iOSアプリケーションにおけるリモート通知の登録方法を示した簡単な例ですMacアプリケーションでも同様です。

リスト 2-4　リモート通知を登録するコード例

```
- (void)applicationDidFinishLaunching:(UIApplication *)app {
   // other setup tasks here....
    UIUserNotificationType types = UIUserNotificationTypeBadge |
                UIUserNotificationTypeSound | UIUserNotificationTypeAlert;

    UIUserNotificationSettings *mySettings =
               [UIUserNotificationSettings settingsForTypes:types categories:nil];

   [[UIApplication sharedApplication] registerUserNotificationSettings:mySettings];
    [[UIApplication sharedApplication] registerForRemoteNotifications];
}

// 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);
}
```

`application:didFailToRegisterForRemoteNotificationsWithError:`の実装では、エラーオブジェクトを適切に処理するとともに、リモート通知が届くことを前提とした処理は排除してください。届くはずのない通知のために、無駄な処理をするのは望ましくありません。「賢く」機能を落とすことが肝腎です。

ローカル通知、リモート通知に対処する

システムがアプリケーションにローカル通知やリモート通知を配信したときに起こりうる、さまざまな状況を考えてみましょう。

アプリケーションがフォアグラウンドで動作中でないときに、通知が配信される。システムはこの場合、警告を表示し、アイコンにバッジを添え、サウンドを鳴らす、という形でユーザに知らせるとともに、おそらくアクションボタンをいくつか表示してタップできるようにします。

iOS 8で、通知に応じてユーザがカスタムアクションボタンをタップする。この場合iOSは、`application:handleActionWithIdentifier:forRemoteNotification:completionHandler:`または`application:handleActionWithIdentifier:forLocalNotification:completionHandler:`を呼び出します。どちらも引数としてアクションのIDが渡されるので、どのボタンがタップされたか判断できます。さらに、リモート/ローカル通知オブジェクトも渡されるので、アクションの処理に必要な情報を取得できます。

ユーザが、警告のデフォルトボタンをタップし、あるいはアプリケーションアイコンをタップ（またはクリック）する。（iOSが動作するデバイス上で）デフォルトのアクションボタンがタップされると、システムはアプリケーションを起動し、アプリケーションはデリゲートの`application:didFinishLaunchingWithOptions:`メソッドを呼び出します。その際、（リモート通知ならば）通知ペイロード、または（ローカル通知ならば）ローカル通知オブジェクトを引数として渡します。`application:didFinishLaunchingWithOptions:`は、通知に対処するコードの記述場所として最適とは言えませんが、この時点でペイロードを取得しておけば、ハンドラメソッドが呼び出される前に、更新処理を開始する機会が得られます。

リモート通知の場合、システムは`application:didReceiveRemoteNotification:fetchCompletionHandler:`も呼び出します。

OS Xが稼働するコンピュータ上でアプリケーションアイコンがクリックされた場合、アプリケーションはデリゲートの`applicationDidFinishLaunching:`メソッドを呼び出し、デリゲートはこのメソッド内でリモート通知ペイロードを取得できます。iOSが稼働するデバイス上で、アプリケーションアイコンがタップされた場合、アプリケーションは同じメソッドを呼び出しますが、通知に関する情報は渡しません。

アプリケーションがフォアグラウンドで動作中に、通知が配信される。アプリケーションは`UIApplicationDelegate`の`application:didReceiveLocalNotification:`メソッドまたは`application:didReceiveRemoteNotification:fetchCompletionHandler:`メソッドを呼び出します（`application:didReceiveRemoteNotification:fetchCompletionHandler:`が実装されていなければ`application:didReceiveRemoteNotification:`を呼び出し）。OS Xの場合は`application:didReceiveRemoteNotification:`メソッドを呼び出します。

アプリケーションは、渡されたリモート通知ペイロード、または、iOSでは`UILocalNotification`オブジェクトを利用して、通知に関連する項目を処理するコンテキストを設定できます。理論上、デリゲートは各プラットフォームで、あらゆる状況でリモート通知やローカル通知を配信できるよう、次のような処理をします。

- OS Xの場合、デリゲートは`NSApplicationDelegate Protocol`プロトコルに対応し、`application:didReceiveRemoteNotification:`メソッドを実装していなければなりません。
- iOSの場合、デリゲートは`UIApplicationDelegate`プロトコルに対応し、`application:didReceiveRemoteNotification:fetchCompletionHandler:`メソッドまたは`application:didReceiveLocalNotification:`メソッドを実装していなければなりません。通知アクションに対処するためには、`application:handleActionWithIdentifier:forLocalNotification:completionHandler:`メソッドまたは`application:handleActionWithIdentifier:forRemoteNotification:completionHandler:`メソッドを実装する必要があります。

リスト 2-5では、iOSアプリケーション用のデリゲートに`application:didFinishLaunchingWithOptions:`メソッドを実装して、ローカル通知を処理します。`UIApplicationLaunchOptionsLocalNotificationKey`キーを使用して、起動オプション辞書から、関連する`UILocalNotification`オブジェクトを取得します。`UILocalNotification`オブジェクトの`userInfo`辞書から、通知の理由となるToDo項目にアクセスして、アプリケーションの初期コンテキストの設定に使用します。この例に示すように、通知を処理する一環として、アプリケーションアイコンのバッジの数字を適切にリセットしてください（未処理の項目がなければバッジを消去します）。

リスト 2-5　ローカル通知に対処するコード例（通知に応じてアプリケーションが起動された場合）

```
- (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
        app.applicationIconBadgeNumber = localNotif.applicationIconBadgeNumber-1;
    }
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
    return YES;
}
```

リモート通知の場合の実装も、各プラットフォームで特に宣言された定数を通知ペイロードにアクセスするキーとして使用することを除けば、これと似ています。

- iOSの場合、デリゲートは`application:didFinishLaunchingWithOptions:`メソッドの実装において、`UIApplicationLaunchOptionsRemoteNotificationKey`キーを、起動オプション辞書からペイロードにアクセスするためのキーとして使用します。
- OS Xの場合、デリゲートは`applicationDidFinishLaunching:`メソッドの実装において、`NSApplicationLaunchRemoteNotificationKey`キーを、メソッドに渡された`NSNotification`オブジェクトの`userInfo`辞書からペイロード辞書にアクセスするためのキーとして使用します。

ペイロード自体は、通知の要素（警告メッセージ、バッジの数字、サウンドなど）を含む、`NSDictionary`オブジェクトです。ペイロードには、アプリケーションが当初のユーザインターフェイスをセットアップするときにそのコンテキストの設定に使用できるカスタムデータも含めることができます。Remote Notificationペイロードの詳細については、“通知ペイロード”（46 ページ）を参照してください。

Important: リモート通知は確実に配送されるという保証がないので、重要なデータや、他の手段では取得できないデータの配送に、通知ペイロードを使わないでください。

カスタムペイロードプロパティの適切な使いかたの一例は、メッセージがクライアントにダウンロード済みの電子メールアカウントを表す文字列です。アプリケーションは、この文字列をダウンロードインターフェイスに組み込むことができます。カスタムペイロードプロパティのもう1つの例は、プロバイダが最初に通知を送信した日時を表すタイムスタンプです。クライアントアプリケーションは、この値を使用してその通知がどのくらい古いかを測定できます。

通知ハンドラメソッドでリモート通知に対処する際、アプリケーションデリゲートは重要な追加タスクを実行しているかも知れません。アプリケーションの起動直後、デリゲートはそのプロバイダに接続して、待機中のデータをフェッチする必要があります。

注意: クライアントアプリケーションは、非同期または二次スレッドで常にプロバイダと通信する必要があります。

リスト 2-6に、アプリケーションがフォアグラウンドで動作中の場合に呼び出される、`application:didReceiveLocalNotification:`メソッドの実装を示します。ここでのアプリケーションデリゲートは、リスト 2-5で行ったのと同じ作業を行います。オブジェクトがメソッドの引数であるため、ここでは`UILocalNotification`オブジェクトに直接アクセスできます。

リスト 2-6 ローカル通知に対処するコード例（アプリケーションがすでに動作中の場合）

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

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

    [viewController displayItem:itemName];  // custom method

    app.applicationIconBadgeNumber = notification.applicationIconBadgeNumber - 1;

}
```

アプリケーションがフォアグラウンドで動作しているときにシステムから配信されたリモート通知をキャッチさせたい場合には、アプリケーションデリゲートは`application:didReceiveRemoteNotification:fetchCompletionHandler:`メソッドを実装する必要があります。デリゲートは、待機中のデータ、メッセージ、またはほかの項目をダウンロードする手順を開始し、終了後に、アプリケーションアイコンからバッジを消去すべきです このメソッドの2番目のパラメータで渡された辞書は通知ペイロードです。そこに含まれているカスタムプロパティをアプリケーションの現在のコンテキストを変更するために使用すべきではありません。

通知アクションを扱う（iOS）

OS Xや旧iOS（iOS 8より前）では、ユーザ通知にデフォルトアクションを1つ設定することしかできませんでした。iOS 8以降は、カスタムアクションを追加できるようになっています。ロック画面、バナー、通知センターには、2つのアクションを表示できます。モーダル型の警告画面には、ユーザが「Options」ボタンをタップしたとき、最大4つのアクションが現れるようにすることができます。アプリケーションで通知アクションを扱うためには、当該アクションを登録すること、ローカル通知をスケジューリングしまたはリモート通知をプッシュすること、さらに、ユーザが選択したアクションに対処することが必要です。

通知アクションを登録する

アプリケーションが通知アクションを扱うためには、必要な数のアクションを定義し、これをカテゴリとしてグループ化し、アプリケーションの共有`UIApplication`インスタンスに登録する必要があります。

通知アクションを定義するためには、まず通知アクションクラス（通常は`UIMutableUserNotificationAction`）のインスタンスを生成、初期化しなければなりません。続いて、アプリケーションがアクションを処理する際、ハンドラに渡されるIDを定義します。また、アクションボタンに表示する、ローカライズした文字列も定義します。次に、アクションの起動モードを、ユーザの作業に割り込む（関与を求める）必要があればフォアグラウンド、そうでなければバックグラウンドと設定します。最後に、アクションが「破壊的」か否かを宣言します。破壊的であるとすれば、ボタンは赤で表示され、実行に先だってパスコードを入力するよう求められるようになります。リスト 2-7に以上の手順を実装したコード例を示します。

リスト 2-7　通知アクションを定義するコード例

```
UIMutableUserNotificationAction *acceptAction =
            [[UIMutableUserNotificationAction alloc] init];

// Define an ID string to be passed back to your app when you handle the action
acceptAction.identifier = @"ACCEPT_IDENTIFIER";

// Localized string displayed in the action button
acceptAction.title = @"Accept";

// If you need to show UI, choose foreground
acceptAction.activationMode = UIUserNotificationActivationModeBackground;
```

```
// Destructive actions display in red
acceptAction.destructive = NO;

// Set whether the action requires the user to authenticate
acceptAction.authenticationRequired = NO;
```

`activationMode`プロパティは、ユーザが通知に応答したとき、iOSがアプリケーションをフォアグラウンドで起動するか、バックグラウンドで起動するかを表します。`UIUserNotificationActivationModeBackground`の場合、アプリケーションは低い優先度で起動されることになります。`destructive`プロパティが`NO`であれば、アクションのボタンは青で表示されます。`YES`ならば赤になります。アクションの`authenticationRequired`プロパティを`YES`と設定すれば、ユーザが通知に応答したときにデバイスがロックされていた場合、パスコードを入力しなければ当該アクションを選択できないようになります。しかし、ここでパスコードを入力してもロックが解除されるわけではないので、ファイルにアクセスする必要があれば、当該ファイルを適切なデータ保護クラスにしておかなければなりません。`activationMode`プロパティの値が`UIUserNotificationActivationModeForeground`であれば、`authenticationRequired`プロパティは、実際の値にかかわらず`YES`であるものとして処理します。

たとえば「カレンダー」アプリケーションのアクションを考えてみましょう。「Accept」アクションは、「Accept」ボタンがタップされた後、ユーザとのやり取りを要しないので、`activationMode`はバックグラウンドで構いません。また、「Accept」アクションは破壊的でないので、通知画面やロック画面に表示されるボタンは赤になりません。招待を受け入れても害はあまりないので、認証も必要ありません。別の例を挙げてみましょう。「メール(Mail)」アプリケーションでメッセージを消去する「Trash」アクションは、ボタンがタップされた後、やはりユーザとのやり取りは不要であり、バックグラウンドで実行できます。しかしこれは破壊的なので、`destructive`プロパティは`YES`であり、他人に勝手に消去されてしまわないよう認証する必要があります。一方、「Reply」アクションはユーザとのやり取りが必要になるので、起動モードはフォアグラウンドでなければなりません。破壊的ではありませんが、`authenticationRequired`プロパティの値にかかわらず、フォアグラウンドでのアクションには常に認証を要するので、ユーザはデバイスのロックを解除する必要があります。

必要なアクションをすべて定義した後、それぞれを「カテゴリ」分けします。これには、通知のタイプに、関連するアクション群を関連づける働きがあります。たとえば「Invite」カテゴリには、「Accept」、「Maybe」、「Decline」などのアクションが入るでしょう。あるいは、「New mail」カテゴリには「Mark as Read」や「Trash」、「Tagged」カテゴリには「Like」、「Comment」、「Untag」といったアクションを入れることが考えられます。ユーザのデバイスに配送するローカル/リモート通知を組み立てるときに、その通知画面に表示するべきアクションを集めたカテゴリを指定します。iOSは通知を表示する際、カテゴリ情報にもとづいて、警告画面に表示するボタンを決め、ユーザが選択したアクションをアプリケーションに通知します。

いくつかのアクションをカテゴリとしてまとめるためには、通知カテゴリクラス（通常は`UIMutableUserNotificationCategory`）のインスタンスを生成、初期化します。続いて、カテゴリのIDを定義します。これを、ローカル通知や、リモート通知のプッシュペイロードに設定することになります。

次に、該当するアクションをこのカテゴリに追加し、アクションコンテキストを設定します。ユーザ通知アクションのコンテキストには、4つまでのアクションに対応した「デフォルト」コンテキストと、2つのアクションを表示できる「最小」コンテキストの2種類があります。コンテキストは、通知に応じて表示されるUI部品に関係します。ロック画面には2つのアクションを表示する余地しかないので、「最小」コンテキストが適用されます。一方、モーダル警告には4つとも表示できるので、「デフォルト」コンテキストになります。リスト 2-8に、以上の手順を実装したコード例を示します。

リスト 2-8　アクションをカテゴリにまとめるコード例

```
// First create the category
UIMutableUserNotificationCategory *inviteCategory =
        [[UIMutableUserNotificationCategory alloc] init];

// Identifier to include in your push payload and local notification
inviteCategory.identifier = @"INVITE_CATEGORY";

// Add the actions to the category and set the action context
[inviteCategory setActions:@[acceptAction, maybeAction, declineAction]
    forContext:UIUserNotificationActionContextDefault];

// Set the actions to present in a minimal context
[inviteCategory setActions:@[acceptAction, declineAction]
    forContext:UIUserNotificationActionContextMinimal];
```

リスト 2-8 （29 ページ）に2箇所現れる`setActions:forContext:`メッセージは、アクションを「デフォルト」コンテキストに正しい順序で表示すること、重要なアクションを「最小」コンテキストに表示することを、それぞれ表します。すなわち、モーダル警告には「Accept」、「Maybe」、「Decline」が（この順序で）表示され、ロック画面には「Accept」および「Decline」が表示されます。2つめの`setActions:forContext:`がなかったとすれば、ロック画面には「デフォルト」コンテキストのボタンのうち最初の2つ、すなわち「Accept」と「Maybe」だけが表示されます。

通知アクションのカテゴリを定義した後、これを登録する必要があります。これは、カテゴリをすべて集合としてまとめ、ユーザ通知設定に与えた上で、共有アプリケーションインスタンスに登録する、という手順でおこないます。リスト 2-9にそのコード例を示します。

リスト 2-9　通知カテゴリを登録するコード例

```
NSSet *categories = [NSSet setWithObjects:inviteCategory, alarmCategory, ...

UIUserNotificationSettings *settings =
       [UIUserNotificationSettings settingsForTypes:types categories:categories];

[[UIApplication sharedApplication] registerUserNotificationSettings:settings];
```

`UIUserNotificationSettings`クラスの`settingsForTypes:categories:`メソッドは、リスト 2-1（16 ページ）に示したものと同じ（ただし`categories`引数には`nil`を渡した）であり、やはり通知設定をアプリケーションインスタンスに登録する働きがあります。前回は通知のタイプでしたが、今回は通知カテゴリが、アプリケーションの通知設定として組み込まれるのです。

カスタムアクションを組み込んだリモート通知をプッシュし、またはローカル通知をスケジューリングする

定義し、カテゴリ分けし、登録した通知アクションを表示するためには、リモート通知をプッシュするか、またはローカル通知をスケジューリングする必要があります。リモート通知の場合、プッシュペイロードにカテゴリのIDを設定します（リスト 2-10を参照）。カテゴリの処理は、iOSアプリケーションとプッシュ通知サーバが連携しておこないます。プッシュサーバは、ユーザに向けて通知を送信する際、適切な値のカテゴリキーを通知のペイロードに追加できます。iOSは、カテゴリキーが設定されたプッシュ通知があると、アプリケーションに登録されているカテゴリを検索します。合致するものが見つかると、通知画面に該当するアクションを表示します。

iOS 8では、プッシュペイロード長が256バイトから2キロバイトに拡張されました。Remote Notificationペイロードの詳細については、“通知ペイロード”（46 ページ）を参照してください。

リスト 2-10　プッシュペイロードにカテゴリIDを設定するコード例

```
{
    "aps" :  {
        "alert" : "You're invited!",
        "category" : "INVITE_CATEGORY"
    }
}
```

ローカル通知の場合、通常どおり通知を生成した後、提示するアクションのカテゴリを設定し、最後に通常どおり通知をスケジューリングします（リスト 2-11を参照）。

リスト 2-11　ローカル通知に用いるアクションのカテゴリを定義するコード例

```
UILocalNotification *notification = [[UILocalNotification alloc] init];
. . .
notification.category = @"INVITE_CATEGORY";
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

通知アクションに対処する

アプリケーションがフォアグラウンドで動作していない場合、ユーザが通知をスワイプまたはタップしたときにデフォルトのアクションを起こすため、iOSはフォアグラウンドでアプリケーションを起動し、`UIApplicationDelegate`の`application:didFinishLaunchingWithOptions:`メソッドを呼び出します。その際、`options`辞書のローカル通知またはリモート通知を渡します。リモート通知の場合、システムは`application:didReceiveRemoteNotification:fetchCompletionHandler:`メソッドも呼び出します。

アプリケーションがすでにフォアグラウンド状態であれば、iOSは通知を表示しません。代わりに、デフォルトアクションに対処するため、`UIApplicationDelegate`の`application:didReceiveLocalNotification:`メソッドまたは`application:didReceiveRemoteNotification:fetchCompletionHandler:`メソッドを呼び出します（`application:didReceiveRemoteNotification:fetchCompletionHandler:`が実装されていなければ`application:didReceiveRemoteNotification:`を呼び出し）。

最後に、iOS 8で追加されたカスタムアクションに対処するため、アプリケーションデリゲートに、`application:handleActionWithIdentifier:forRemoteNotification:completionHandler:`メソッドまたは`application:handleActionWithIdentifier:forLocalNotification:completionHandler:`メソッドの、少なくとも一方を実装しなければなりません。いずれの場合もアクションIDが渡されるので、どのアクションがタップされたか判断できます。また、通知（リモートまたはローカル）も渡されるので、アクションの処理に必要な情報を取得できます。最後に、システムから渡される完了ハンドラを、アクションの処理の最後に呼び出さなければなりません。リスト 2-12に、独自に定義したアクションハンドラメソッドを呼び出す形で実装した例を示します。

リスト 2-12　カスタム通知アクションに対処するコード例

```
- (void)application:(UIApplication *) application
              handleActionWithIdentifier: (NSString *) identifier
          // either forLocalNotification: (NSDictionary *) notification or
                  forRemoteNotification: (NSDictionary *) notification
                      completionHandler: (void (^)()) completionHandler {
```

```
    if ([identifier isEqualToString: @"ACCEPT_IDENTIFIER"]) {
        [self handleAcceptActionWithNotification:notification];
    }

    // Must be called when finished
    completionHandler();
}
```

ロケーションベースの通知を使う

iOS 8以降、地理上の特定の場所に到達したとき、ユーザに通知を送ることができるようになりました。この機能はCore Locationを利用しており、`UILocalNotification`クラスに追加された、簡単なAPIを使って実装されています。Core Locationの「地域」オブジェクトを定義し、これに通知を関連づけて、ユーザがこの地域の付近に来たとき、この地域に入ったとき、この地域から出たときに発火する（送信される）ようにします。ユーザがこの地域に入った初回のみ通知することも、アプリケーションにとって意味があれば継続的に通知を送るよう設定することも可能です。

ロケーションベースの通知を登録する

ロケーションベースの通知をスケジューリングするためには、あらかじめCore Locationに登録しておく必要があります。これは、`CLLocationManager`のインスタンスを生成し、そのデリゲートとしてアプリケーションを設定する、という方法でおこないます。デリゲートには、ユーザが居場所の追跡を許可しているか否か、アプリケーションに伝えるためのコールバックがあり、必要に応じて呼び出されます。最後に、場所マネージャのインスタンスに`requestWhenInUseAuthorization`メッセージを送信します（リスト2-13を参照）。アプリケーションがこのメソッドを最初に呼び出した時点で、居場所の追跡を許可するか否かユーザに訊ねる、警告画面が現れます。この画面には、「居場所の追跡機能を有効にすれば、現在どこにいるか、友だちも知ることができるようになります」など、アプリケーション側が用意した説明文も表示されます。*この説明文は、場所サービスを利用するためには必須です。* アプリケーションはこの文字列を、`Info.plist`ファイルの`NSLocationWhenInUseUsageDescription`キーに定義します。別の言語ロケールでアプリケーションが動作する場合、`Info.plist`の文字列ファイルで、適切にローカライズしてください。ユーザが許可すれば、アプリケーションがフォアグラウンドで動作している間、居場所を追跡できるようになります。

リスト 2-13　ユーザの居場所を追跡する許可を求める（認証を得る）コード例

```
CLLocationManager *locMan = [[CLLocationManager alloc] init];

// Set a delegate that receives callbacks that specify if your app is allowed to
track the user's location

locMan.delegate = self;


// Request authorization to track the user’s location and enable location-based
notifications

[locMan requestWhenInUseAuthorization];
```

ロケーションベースの通知による警告画面は、アプリケーションがバックグラウンド状態や中断状態であっても現れることがあります。しかし、アプリケーションのコールバックが呼び出されるのは、ユーザがこの警告に応答した後で、しかも居場所の情報に対するアクセスが許可されている場合に限ります。

Core Locationのコールバックに対処する

起動の際に、認証状態（ロケーションベースの通知を許可するか否か）を確認し、状態情報を保存しておかなければなりません。Core Locationマネージャから呼び出されるデリゲートコールバックとしては、まず、認証状態が変わった旨を通知する`locationManager:didChangeAuthorizationStatus:`があります。最初に、コールバックに引数として渡された状態（status）が`kCLAuthorizationStatusAuthorizedWhenInUse`であるかどうか調べてください（リスト 2-14を参照）。これは、ユーザの居場所を追跡してよいことを表します。その上で、ロケーションベースの通知のスケジューリングを始めます。

リスト 2-14　Core Locationから呼び出される認証コールバックのコード例

```
- (void)locationManager:(CLLocationManager *)manager
                didChangeAuthorizationStatus:(CLAuthorizationStatus)status {


    // Check status to see if the app is authorized

    BOOL canUseLocationNotifications = (status ==
kCLAuthorizationStatusAuthorizedWhenInUse);


    if (canUseLocationNotifications) {

        [self startShowingLocationNotifications]; // Custom method defined below

    }

}
```

リスト2-15に、ユーザがある地域に入ったときに送信されるよう、通知をスケジューリングするコード例を示します。まず実行しなければならないのは、所定の日時に送信されるローカル通知の場合と同様、`UILocalNotification`のインスタンスを生成し、そのタイプ（この場合は警告）を定義することです。

リスト 2-15 ロケーションベースの通知をスケジューリングするコード例

```
- (void)startShowingNotifications {

    UILocalNotification *locNotification = [[UILocalNotification alloc] init];
    locNotification.alertBody = @"You have arrived!";
    locNotification.regionTriggersOnce = YES;

    locNotification.region = [[CLCircularRegion alloc]
                        initWithCenter:LOC_COORDINATE
                                radius:LOC_RADIUS
                            identifier:LOC_IDENTIFIER];

    [[UIApplication sharedApplication] scheduleLocalNotification:locNotification];
}
```

リスト2-15で定義した地域にユーザが入ると、アプリケーションがフォアグラウンドで動作していなければ、「You have arrived!」という警告が現れます。次の行では、この通知が1度だけ、ユーザが当該地域に入ったか、または当該地域から出たときに送信される旨を指定しています。実際にはこれはデフォルトの挙動なので、明示的に`YES`を指定しなくても構いません。状況により、この挙動を変えたければ`NO`を指定してください。

次に、`CLCircularRegion`のインスタンスを生成し、`UILocalNotification`のインスタンスのregionプロパティに設定します。この場合、アプリケーションが定義した位置座標と半径を与えて、この円内にユーザが入ったとき、通知が送信されるようにしています。この例ではプロパティとして`CLCircularRegion`を与えていますが、`CLBeaconRegion`その他、`CLRegion`の各種サブクラスも指定できます。

最後に、他のローカル通知の場合と同様に、この通知オブジェクトを引数として`UIApplication`の共有インスタンスの`scheduleLocalNotification:`メソッドを呼び出します。

ロケーションベースの通知に対処する

リスト 2-15 （34 ページ）で定義した地域にユーザが入ったとき、アプリケーションが中断状態であれば、「You have arrived!」という警告が現れます。アプリケーションはこのローカル通知を、アプリケーションデリゲートに定義したコールバックメソッドである`application:didFinishLaunchingWithOptions:`で処理します。また、ユーザがこの地域に入ったとき、アプリケーションがフォアグラウンド状態であれば、アプリケーションデリゲートの`application:didReceiveLocalNotification:`がコールバックとして呼び出されます。

ロケーションベースの通知に対処する処理ロジックは、`application:didFinishLaunchingWithOptions:`メソッドでも`application:didReceiveLocalNotification:`メソッドでもほとんど同じです。どちらも引数として`UILocalNotification`のインスタンスが渡され、これには`region`というプロパティがあります。その値が`nil`でなければ、この通知はロケーションベースなので、それに応じた処理をすることになります。リスト 2-16に示した例では、アプリケーションデリゲートの`tellFriendsUserArrivedAtRegion:`というメソッド（その中身は省略）を呼び出しています。

リスト 2-16 ロケーションベースの通知に対処するコード例

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

    CLRegion *region = notification.region;

    if (region) {
            [self tellFriendsUserArrivedAtRegion:region];
    }
}
```

最後に、ユーザがCore Locationを無効にしていれば、`application:didReceiveLocalNotification:`メソッドは呼び出されないことに注意してください。ユーザはいつでも、「設定(Settings)」アプリケーションの「Privacy」＞「Location Services」以下で無効にすることができます。

カスタム警告音の準備

iOSにおけるリモート通知に関してデベロッパは、iOSがアプリケーションにローカル通知またはリモート通知を提示したときに再生するカスタム警告音を指定できます。このサウンドファイルは、クライアントアプリケーションのメインバンドル内か、アプリケーションのデータコンテナの`Library/Sounds`フォルダ内にあるものを指定できます。

カスタム警告音はiOSのシステムサウンド機能によって再生されるため、次のいずれかのオーディオデータフォーマットでなければなりません。

- リニアPCM
- MA4（IMA/ADPCM）
- μLaw
- aLaw

このオーディオデータは`aiff`、`wav`、または`caf`ファイルとして保存できます。次に、Xcodeで、このサウンドファイルをアプリケーションバンドルの非ローカライズリソースとしてプロジェクトに追加するか、データコンテナの`Library/Sounds`フォルダに追加します。

`afconvert`ツールを使ってサウンドを変換することもできます。たとえば、16ビットリニアPCMのシステムサウンドの`Submarine.aiff`をIMA4オーディオのCAFファイルに変換するには、ターミナルアプリケーションで次のコマンドを使用します。

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

QuickTime Playerでサウンドを開き、「ウインドウ」メニューから「ムービーインスペクタを表示」を選ぶと、そのサウンドのデータフォーマットを判別できます。

カスタムのサウンドの再生時間は30秒未満でなければなりません。カスタムのサウンドがこの制限を超える場合、代わりにデフォルトのシステムサウンドが再生されます。

プロバイダに現在の言語設定を渡す（リモート通知）

アプリケーションが、ローカライズされた警告メッセージをクライアント側で取得するために、`aps`辞書の`loc-key`プロパティと`loc-args`プロパティを使用しない場合は、プロバイダが、通知ペイロードに含める警告メッセージのテキストをローカライズする必要があります。ただしそれには、デバイスのユーザが設定言語として選択している言語を検出する必要があります（ユーザはこれを、「設定

(Settings)」アプリケーションの「一般(General)」＞「言語環境(International)」＞「言語(Language)」ビューで設定します）。クライアントアプリケーションは設定言語の識別子をプロバイダに送信する必要があります。これは、標準化されているIETF BCP 47の言語識別子（「en」、「fr」など）です。

> **注意:** `loc-key`プロパティと`loc-args`プロパティ、およびクライアント側のメッセージローカライズの詳細については、“通知ペイロード” （46 ページ）を参照してください。

リスト2-17は、現在選択されている言語を取得して、それをプロバイダに送信する手法を示しています。iOSでは、`NSLocale`の`preferredLanguages`プロパティで返される配列に1つのオブジェクト（選択されている言語を識別する言語コードをカプセル化した`NSString`オブジェクト）が含まれています。`UTF8String`は、この文字列オブジェクトをUTF8エンコードのC文字列に変換します。

リスト 2-17 現在サポートされている言語を取得してプロバイダに送信する

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

アプリケーションは、ユーザが現在のロケールを何か変更するたびに設定言語をプロバイダに送信することもできます。それには、`NSCurrentLocaleDidChangeNotification`という名前の通知を検知し、通知処理メソッド内で、設定言語の識別コードを取得してそれをプロバイダに送信します。

設定言語がアプリケーションでサポートしていない言語の場合、プロバイダは広く普及している代替言語（英語、スペイン語など）にメッセージテキストをローカライズしなければなりません。

Apple Push Notificationサービス

Apple Push Notificationサービス（略してAPNs）はリモート通知機能の中核です。これは、iOS/OS Xデバイスに情報を配信するための堅牢で非常に効率的なサービスです。各デバイスは、このサービスとの間で認証済みの暗号化されたIP接続を確立し、この永続的な接続を介して通知を受信します。アプリケーションが実行されていないときに通知が届いた場合、デバイスはそのアプリケーションに更新データがあることをユーザに警告します。

ソフトウェアのデベロッパ（「プロバイダ」）は、サーバソフトウェア内で通知を作成します。プロバイダは、対象となるクライアントアプリケーション宛ての受信データを監視している間、永続的でセキュアなチャネルを介してAPNsに接続します。アプリケーション宛ての新しいデータを受信すると、プロバイダは通知を作成してこのチャンネルを介してAPNsに送信します。APNsは、その通知をターゲットデバイスに配信（プッシュ）します。

APNsは単純でありながら効率的で高い処理能力を持つ配信サービスです。さらに、APNsには蓄積交換機能を提供するデフォルトのQuality-of-Serviceコンポーネントが含まれています。詳細については、“Quality of Service”（40 ページ）を参照してください。

“プロバイダとApple Push Notificationサービスの通信”（57 ページ）および“ユーザ通知の登録、スケジューリング、対処”（16 ページ）では、プロバイダとiOSアプリケーションに固有の実装要件について、それぞれ説明しています。

リモート通知とその経路

Apple Push Notificationサービスは、特定のプロバイダから特定のデバイスにリモート通知をルーティングします。通知は、デバイストークンとペイロードという2つの主要なデータ部分から構成された短いメッセージです。デバイストークンは電話番号にたとえることができます。そこには、クライアントアプリケーションがインストールされているデバイスをAPNsが見つけるために必要な情報が含まれています。また、APNsはこれを使用して通知のルーティングを認証します。ペイロードは、デバイス上のアプリケーションのユーザに警告する方法を指定するための、JSONで定義されたプロパティリストです。

> **注意:** デバイストークンの詳細については、“セキュリティアーキテクチャ” （41 ページ）を参照してください。通知ペイロードの詳細については、“通知ペイロード” （46 ページ）を参照してください。

リモート通知のデータは1方向に流れます。プロバイダは、クライアントアプリケーションのデバイストークンとペイロードを含む通知パッケージを作成します。プロバイダは、その通知をAPNsに送信します。そして、APNsがその通知をデバイスに配信（プッシュ）します。

プロバイダは、自分自身をAPNsに対して認証する際、データの提供先となるアプリケーションを識別するトピックをAPNsサーバに送信します。このトピックは、現在のところターゲットアプリケーションのバンドル識別子です。

図 3-1 プロバイダからクライアントアプリケーションにリモート通知をプッシュする様子

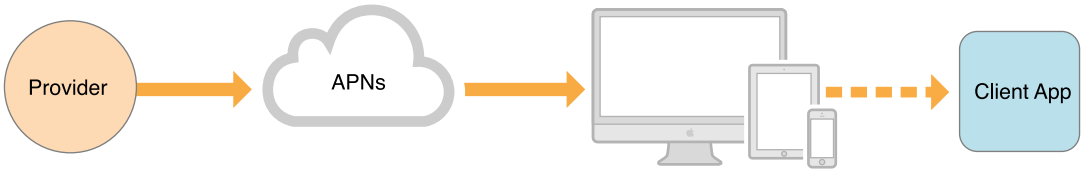
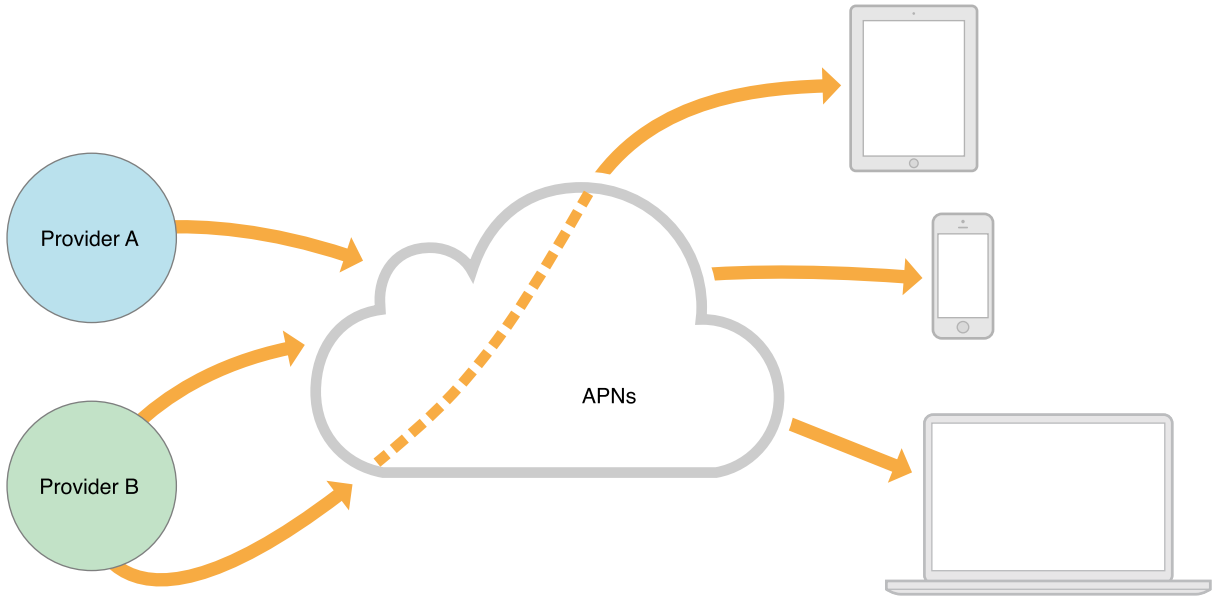


図 3-1は、APNsによってプロバイダとデバイスの間に実現される仮想ネットワークを非常に単純化した図です。ASPsのデバイス側とプロバイダ側には、共に複数の接続ポイントがあります。そして、プロバイダ側の接続ポイントをゲートウェイと呼びます。通常は、複数のプロバイダが存在し、それぞ

れがこれらのゲートウェイを介して、APNsとの間に1つ以上の永続的でセキュアな接続を確立します。そして、これらのプロバイダは、APNsを経由してクライアントアプリケーションがインストールされている複数のデバイスに通知を送信します。図 3-2は、上よりも少し現実的な図です。

図 3-2 複数のプロバイダから複数のデバイスにリモート通知をプッシュする様子



フィードバックサービスはプロバイダに、配送できなかった通知に関する情報（ターゲットアプリケーションが当該デバイスから削除されていたのが原因、など）を渡します。詳しくは“フィードバックサービス”（61 ページ）を参照してください。

Quality of Service

Apple Push Notificationサービスには、蓄積交換機能を実行するデフォルトのQuality of Service（QoS）コンポーネントが含まれています。

APNsが通知を配信しようとしたときにデバイスがオフラインだった場合は、通知を一定時間保存しておき、オンラインになってから配送するようになっています。

ただし保存されるのは、アプリケーションごとに、直近の通知だけです。オフラインの間に複数の通知が送られた場合、古い方から順に破棄されてしまいます。この動作を通知の**併合**と言います。

長時間にわたってオフラインになると、保存していた通知も破棄されてしまいます。

セキュリティアーキテクチャ

プロバイダとデバイス間の通信を可能にするには、Apple Push Notificationサービスが、それらに対して特定のエントリポイントを公開しなければなりません。ただしセキュリティを保証するために、これらのエントリポイントへのアクセスを規制する必要もあります。この目的のためにAPNsは、プロバイダ、デバイス、およびそれらの通信に対して、2つの異なる信頼レベルをリクエストします。これらは、接続信頼とトークンによる信頼と呼ばれます。

接続信頼は、プロバイダ側のAPNs接続が、通知を配信することをAppleから許可されている認定済みのプロバイダとの接続であることを保証します。デバイス側の接続では、APNsはその接続が正当なデバイスとの接続であることを検証しなければなりません。

エントリポイントでの信頼を確立したら、APNsは正当なエンドポイントだけに通知を配信することを保証しなければなりません。それには、この接続を通してメッセージが送信されるルートを検証して、意図した通知先であるデバイスだけが通知を受信するようにしなければなりません。

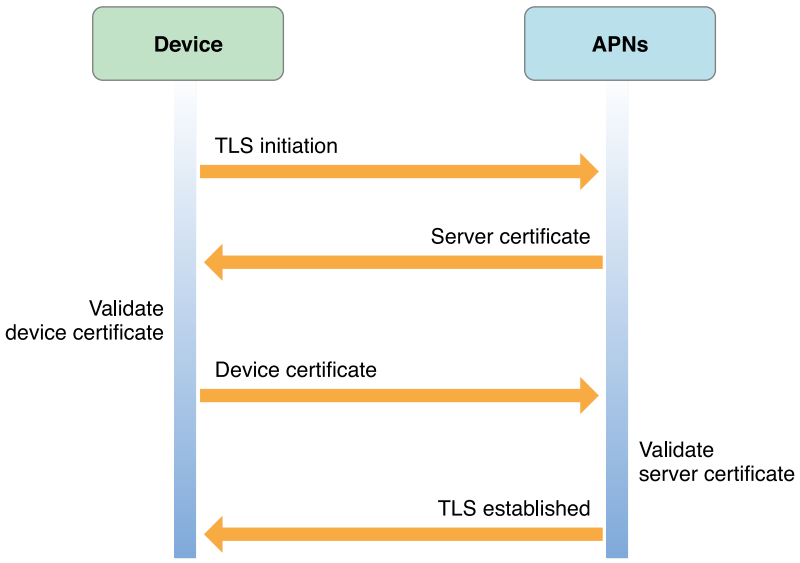
APNsでは、デバイストークンを利用して正確なメッセージルーティングを保証すること（つまり、**トークンによる信頼**）が可能になっています。デバイストークンは、APNsが初めてデバイスに接続したときに、APNsからそのデバイスに渡される不透過なデバイス識別子です。デバイスは、このデバイストークンをプロバイダと共有します。その後は、このトークンがプロバイダからのすべての通知に添付されます。これが、特定の通知のルーティングが正当であることを保証するための基礎になります

> **注意:** デバイストークンは、`UIDevice`の`identifierForVendor`プロパティや`uniqueIdentifier`プロパティの値である、デバイスUDIDとは違います。`ASIdentifierManager`の`advertisingIdentifier`プロパティなど、これに類するものとも別物です。

この後のセクションでは、接続信頼とトークンによる信頼に必要なコンポーネントと、信頼を確立するための4つの手続きについて説明します。

サービスーデバイス間の接続信頼

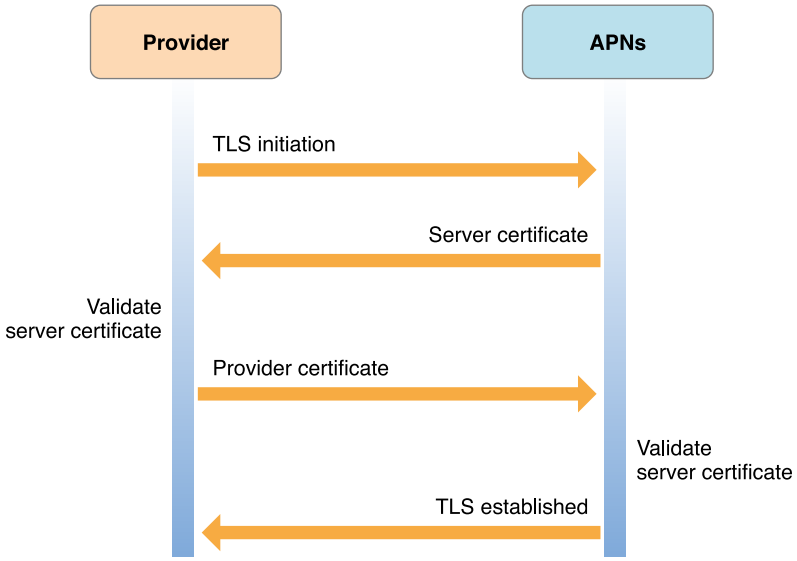
APNsは、TLSのピアツーピア認証を利用して、接続してきたデバイスの認証を行います（接続信頼のこのステージはシステムによって処理されるので、デベロッパが自身で実装する必要はありません）。この手続きの中で、デバイスはAPNsとのTLS接続を開始し、APNsはサーバ証明書を返します。デバイスはこのサーバ証明書を検証して、APNsにデバイス証明書を送信します。APNsはそのデバイス証明書を検証します。



プロバイダーサービス間の接続信頼

プロバイダとAPNs間の接続信頼も、TLSのピアツーピア認証を利用して確立されます。手続きも“サービスーデバイス間の接続信頼” （42 ページ）で説明したものと同様ですプロバイダはTLS接続を開始し、APNsからサーバ証明書を取得します。そしてそのサーバ証明書を検証します。次に、プロバイダ

はAPNsにプロバイダ証明書を送信します。今度は、APNsがそのプロバイダ証明書を検証します。この手続きが完了したら、セキュアなTLS接続が確立されます。これでAPNsは、この接続が正当なプロバイダによって確立したことを保証できます。

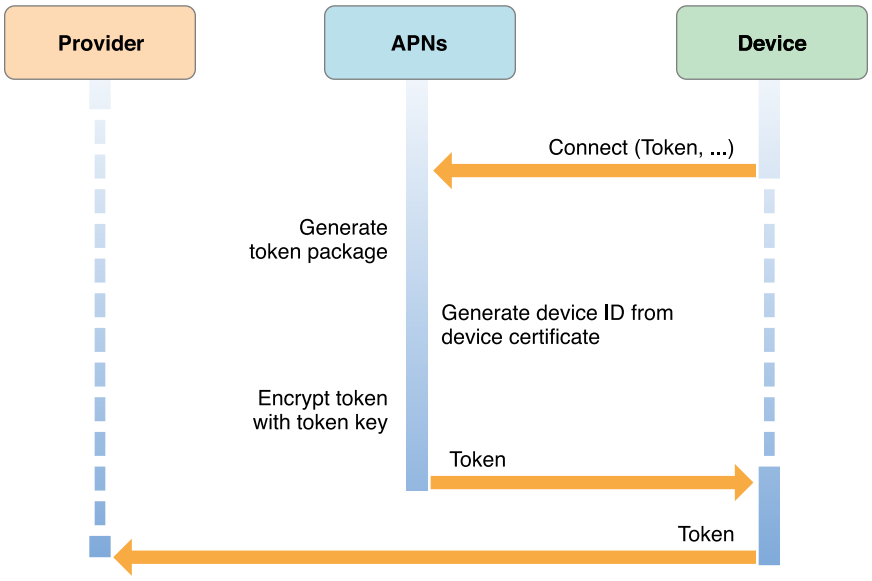


このプロバイダ接続は、証明書に規定されたトピック（バンドルID）によって識別される、ある特定のアプリケーションへの配信に対してのみ有効です。また、APNsは証明書失効リストを管理しています。プロバイダの証明書がこのリストに存在する場合、APNsはプロバイダの信頼を無効にする（つまり、接続を拒否する）こともあります。

トークンの生成と共有

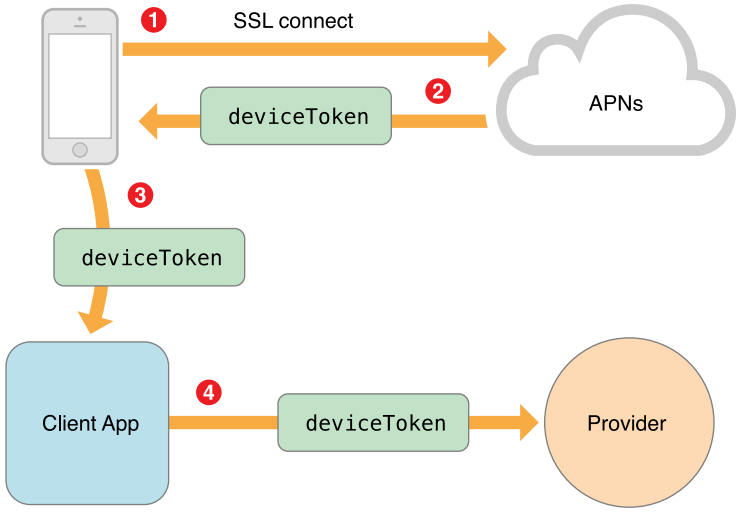
アプリケーションは、リモート通知を受信するための**登録**をしなければなりません。通常、これは、アプリケーションがデバイスにインストールされた直後に行われます（この手続きについては“ユーザ通知の登録、スケジューリング、対処”（16 ページ）で説明しています）。システムは、アプリケーションからの登録リクエストを受け取ると、APNsに接続してそのリクエストを転送します。APNs

は、一意のデバイス証明書に含まれている情報を使用してデバイストークンを生成します。このデバイストークンには、デバイスの識別子が含まれています。次に、トークンキーを利用してデバイストークンを暗号化してデバイスに返します。



デバイスは、このデバイストークンの要求を出してきたアプリケーションに`NSData`オブジェクトとして返します。アプリケーションはこのデバイストークンをバイナリ形式または16進形式でプロバイダに渡さなければなりません。図 3-3は、トークンの生成と共有の順番を示しています。さらに、プロバイダにデバイストークンを供給する際のクライアントアプリケーションの役割も示しています。

図 3-3　デバイストークンの共有

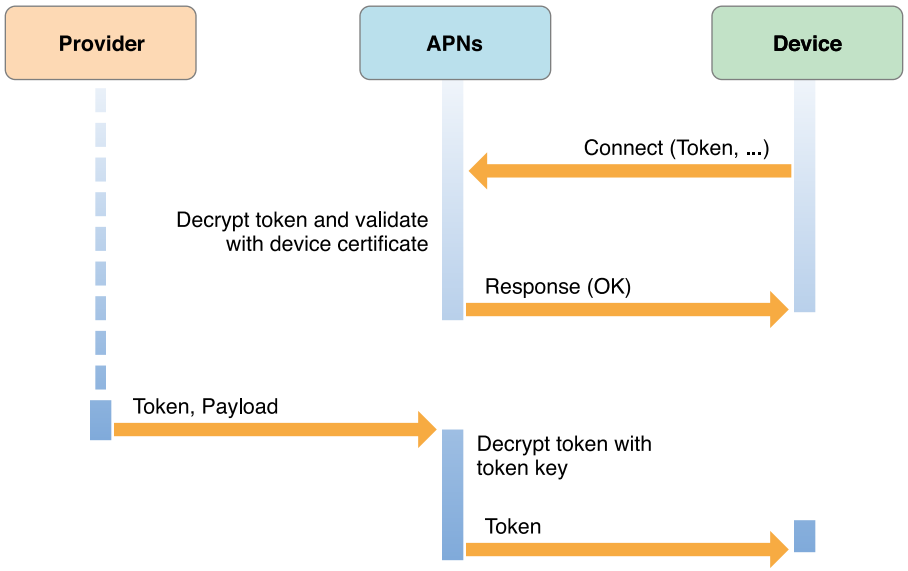


このような形態のトークンによる信頼フェーズによって、後で認証に使われるトークンを生成するのはAPNsだけであることが保証されます。また、APNsは、デバイスから渡されたトークンが、その特定のデバイスのために（そのデバイスのためだけに）以前用意したものと同じトークンであることを自ら確認できます。

トークンによる信頼（通知）

システムは、APNsからデバイストークンを取得した後は（“トークンの生成と共有” （43 ページ）で説明）、APNsに接続するたびにそのトークンをAPNsに渡さなければなりません。APNsはデバイストークンを解読し、そのトークンが接続してきたデバイス用に生成されたものかどうかを検証します。検証のために、APNsは、トークンに含まれているデバイス識別子と、デバイス証明書のデバイス識別子が一致するかどうかを確認します。

デバイスへの配信のためにプロバイダがAPNsに送信するすべての通知には、そのデバイス上のアプリケーションから取得したデバイストークンが添付されていなければなりません。APNsはトークンキーを使用してトークンを解読し、通知が正当であることを確認します。次に、デバイストークンに含まれているデバイスIDを使用して、通知の配信先となるデバイスを判別します。



信頼に関連するコンポーネント

APNsのセキュリティモデルをサポートするために、プロバイダとデバイスは、特定の証明書、認証局（CA）証明書、またはトークンを所有していなければなりません。

- **プロバイダ**：各プロバイダには、一意のプロバイダ証明書と、APNsとの接続を検証するために使用する秘密暗号鍵が必要になります。この証明書はAppleから提供されますが、プロバイダが発行する特定のトピック（クライアントアプリケーションのバンドルID）を識別できなければなり

ません。通知のたびに、プロバイダはターゲットデバイスを識別するためのデバイストークンをAPNsに渡さなければなりません。プロバイダは、APNsサーバから提供された公開サーバ証明書を使用して、接続先のサービスを検証することもできます。

- **デバイス**：システムは、APNsから渡された公開サーバ証明書を使用して、接続先のサービスを認証します。iOSは、一意の秘密鍵と証明書を持っており、これを使用してサービスへの認証を行いTLS接続を確立します。iOSは、デバイスの起動中にこの秘密鍵とデバイス証明書を取得して、キーチェーンに格納します。また、システムは、特定のデバイストークンも保持しています。これは、サービス接続の処理中に受け取ります。登録済みの各クライアントアプリケーションは、このトークンをコンテンツプロバイダに送信する役割を担っています。

APNsサーバも、プロバイダとデバイスの接続とIDを検証するために、必要な証明書、CA証明書、および暗号鍵（秘密鍵と公開鍵）を持っています。

通知ペイロード

各リモート通知にはペイロードがあります。ここには、システムがユーザに警告する方法に関する情報や、別途渡したカスタムデータなどが収容されます。通知ペイロードに許される最大サイズは2キロバイトです。Apple Push Notificationサービスでは、この上限を超える通知は拒否されます。

通知ごとにJSON辞書オブジェクト（RFC 4627に定義）を生成します。この辞書には、キー`aps`で識別されるもう1つの辞書が含まれている必要があります。`aps`辞書には、次のユーザ通知の種類を指定する1つ以上のプロパティが含まれています。

- ユーザに表示する警告メッセージ
- アプリケーションアイコンに付けるバッジの数字
- 警告音

`aps`辞書には`content-available`プロパティも設定できます。 `content-available`プロパティの値が1であれば、リモート通知は「無言の」通知として振る舞います。無言の通知が届くと、iOSはアプリケーションをバックグラウンドで「起こし」て、サーバから新規データを取得したり、バックグラウンドで情報を処理したりできるようにします。ユーザは、この時点では新しい情報や変化した情報を何も知らされませんが、次にアプリケーションを開いた時点で知ることができます。

無言のリモート通知に対処できるようにするためには、`remote-notification`という値を、`Info.plist`ファイルの`UIBackgroundModes`配列に追加する必要があります。この配列について詳しくは、“UIBackgroundModes”を参照してください。

通知が届いたときにターゲットアプリケーションが実行されていない場合は、警告メッセージ、警告音、またはバッジが再生または表示されます。アプリケーションが実行されている場合は、システムがその通知を`NSDictionary`オブジェクトとしてアプリケーションデリゲートに渡します。この辞書には、それに対応するCocoaプロパティリストオブジェクト（`NSNull`も含む）が含まれています。

プロバイダは、Appleによって予約されている`aps`名前空間の外部に、カスタムペイロード値を定義することもできます。カスタム値にはJSONの構造体および基本型（辞書（オブジェクト）、配列、文字列、数字、Boolean）を使用しなければなりません。カスタムペイロードデータとして顧客情報（その他重要なデータ）を含めるべきではありません。代わりに、（ユーザインターフェイスの）コンテキストや内部基準の設定などの目的にカスタムペイロードデータを使用します。たとえば、カスタムペイロード値を、インスタントメッセージのクライアントアプリケーションで使用する会話識別子にしたり、プロバイダが通知を送信した日時を識別するタイムスタンプにできます。警告メッセージに関連付けられているアクションは破壊的（たとえば、デバイス上のデータを削除するなど）であるべきではありません。

Important: 通知の配信は「ベストエフォート型」で、保証はありません。データをアプリケーションに配送することを意図しているのでなければ、新しいデータが利用できることをユーザに*通知する*ためだけに利用すべきです。

表 3-1は、`aps`ペイロードのキーと値のリストです。

表 3-1 `aps`辞書のキーと値

キー	値の型	説明
`alert`	文字列または辞書	このプロパティが含まれていると、システムは標準的な警告を表示します。`alert`の値として、文字列または辞書を指定できます。文字列を指定すると、その文字列は2つのボタン（「Close」と「View」）が付いた警告のメッセージテキストになります。ユーザが「View」をタップすると、アプリケーションが起動します。 `alert`の値として辞書を指定することもできます。この辞書のキーの説明は、表 3-2 （48 ページ）を参照してください。
`badge`	数値	アプリケーションアイコンのバッジとして表示する数字です。 このプロパティが存在しない場合、バッジは変化しません。バッジを削除したい場合は、このプロパティの値を`0`としてください。

キー	値の型	説明
sound	文字列	アプリケーションバンドル内またはアプリケーションのデータコンテナの「Library/Sounds」フォルダ内にあるサウンドファイルの名前です。このファイル内のサウンドが、警告音として再生されます。サウンドファイルが存在しないか、値としてdefaultが指定されている場合は、デフォルトの警告音が再生されます。このオーディオは、システムサウンドと互換性のあるオーディオデータフォーマットのいずれかでなければなりません。詳細については"カスタム警告音の準備"（36 ページ）を参照してください。
content-available	数値	このキーの値が1であれば、新しいコンテンツがあることを表します。このキーと値を設定すると、アプリケーションがバックグラウンドで起動され、あるいは再開したとき、application: didReceiveRemoteNotification: fetchCompletionHandler:が呼び出されるようになります。 （Newsstandアプリケーションは、このキーが指定されたプッシュ通知を、24時間ごとに少なくとも1回受け取れます。）
category	文字列	カスタムアクションを定義するために作成したUIMutableUserNotificationCategoryオブジェクトのidentifierプロパティを表す文字列を指定します。カスタムアクションの使い方の詳細について、"通知アクションを扱う（iOS）"（27 ページ）を参照してください。

表 3-2に、alert辞書のキーと、想定される値を示します。

表 3-2 alertプロパティの子プロパティ

キー	値の型	説明
title	文字列	通知の目的を表す短い文字列。Apple Watchはこの文字列を、通知インターフェイスの一部として表示します。実際に表示されるのはごく短時間なので、ひと目で理解できるように記述しなければなりません。このキーはiOS 8.2で追加されました。
body	文字列	警告メッセージのテキストです。

キー	値の型	説明
`title-loc-key`	文字列または`null`	現在のローカライズに対応する、`Localizable.strings`ファイル内のタイトル文字列のキーです。このキー文字列を`%@`指定子や`%n$@`指定子を使用して書式化すると、`title-loc-args`配列で指定した変数に置き換えることができます。詳細については"ローカライズ形式の文字列"（50 ページ）を参照してください。このキーはiOS 8.2で追加されました。
`title-loc-args`	文字列の配列、または`null`	`title-loc-key`内の書式指定子の代わりに表示する変数文字列値です。詳細については"ローカライズ形式の文字列"（50 ページ）を参照してください。このキーはiOS 8.2で追加されました。
`action-loc-key`	文字列または`null`	文字列が指定されていれば、システムは「Close」および「View」というボタンがついた警告画面を表示します。この文字列をキーとして使用して現在のローカライズからローカライズ文字列を取得し、「View」の代わりに右ボタンのタイトルとして使用します。詳細については"ローカライズ形式の文字列"（50 ページ）を参照してください。
`loc-key`	文字列	現在のローカライズ（ユーザの言語環境によって設定される）に対応する`Localizable.strings`ファイル内の警告メッセージ文字列のキーです。このキー文字列を`%@`指定子や`%n$@`指定子を使用して書式化すると、`loc-args`配列で指定した変数に置き換えることができます。詳細については"ローカライズ形式の文字列"（50ページ）を参照してください。
`loc-args`	文字列の配列	`loc-key`内の書式指定子の代わりに表示する変数文字列値です。詳細については"ローカライズ形式の文字列"（50 ページ）を参照してください。
`launch-image`	文字列	アプリケーションバンドル内の画像ファイルのファイル名です。拡張子を含んでも省略してもかまいません。画像は、ユーザがアクションボタンをタップするか、アクションスライダを動かしたときの起動画像として使われます。このプロパティが指定されていない場合、システムは以前のスナップショットを使用するか、アプリケーションの`Info.plist`ファイルの`UILaunchImageFile`キーで指定された画像を使用するか、`Default.png`にフォールバックします。 このプロパティはiOS 4.0で追加されました。

注意: 「Close」ボタンと「View」ボタンの両方を持つ警告内にメッセージテキストをそのまま表示する場合は、直接alertの値として文字列を指定します。辞書がbodyプロパティしか持たない場合は、alertの値として辞書を指定しないでください。

ローカライズ形式の文字列

ローカライズされた警告メッセージを表示するには2つの方法があります。

- 1つは、通知を作成するサーバでテキストをローカライズする方法です。それには、デバイスで現在選択されている言語設定をサーバが検出しなければなりません（"プロバイダに現在の言語設定を渡す（リモート通知）" （36 ページ）を参照）。
- クライアントアプリケーションは、ローカリゼーションごとに翻訳した警告メッセージ文字列を、バンドルに保存しておけます。プロバイダは、通知ペイロードのaps辞書に、loc-keyキーとloc-argsキーを含めます（タイトル文字列については、aps辞書に、title-loc-keyキーとtitle-loc-argsキーを含めます）。（アプリケーションが実行されていない場合）デバイスは通知を受信すると、これらのaps辞書プロパティを使用して、現在の言語にローカライズされた文字列を検索し書式化します。そして、それをユーザに表示します。

ここでは、2番目の方法が動作する仕組みをもう少し詳しく説明します。

アプリケーションは、画像、サウンド、テキストなどのリソースをサポート対象の各言語用に国際化できます。国際化によって、これらのリソースは集められて、2つの部分（言語コードと.lprojという拡張子）から構成される名前（たとえば、fr.lproj）のバンドルのサブディレクトリに配置されます。プログラムで表示されるローカライズ文字列は、Localizable.stringsというファイルに保存されています。このファイルの各エントリにはキーとローカライズ文字列値が含まれています。この文字列には、変数値に置き換え可能な書式指定子を含めることができます。アプリケーションが特定のリソース（たとえば、ローカライズ文字列）をリクエストすると、ユーザによって現在選択されている言語にローカライズされたリソースを取得します。たとえば、設定言語がフランス語の場合、警告メッセージ用の文字列の値はアプリケーションバンドルのfr.lprojディレクトリ内のLocalizable.stringsから取得されます（アプリケーションは、NSLocalizedStringマクロを利用してこのリクエストを出します）。

> **注意:** action-loc-keyプロパティの値が文字列の場合も、この一般的なパターンに従います。この文字列は、現在選択されている言語のローカライズディレクトリ内のLocalizable.stringsへのキーです。iOSは、このキーを使用して警告メッセージの右側のボタン（“アクション”ボタン）のタイトルを取得します。

わかりやすいように例を考えてみましょう。プロバイダは、警告プロパティの値として次のような辞書を指定しています。

```
"alert" : {
    "loc-key" : "GAME_PLAY_REQUEST_FORMAT",
"loc-args" : [ "Jenna", "Frank"]
}
```

デバイスはこの通知を受信すると、"GAME_PLAY_REQUEST_FORMAT"をキーとして使用し、現在の言語のLocalizable.stringsディレクトリ内の.lprojファイルから、それに関連付けられた文字列値を検索します。現在のローカライズに次のようなLocalizable.stringsエントリがあるとします。

```
"GAME_PLAY_REQUEST_FORMAT" = "%@ and %@ have invited you to play Monopoly";
```

デバイスは、「Jenna and Frank have invited you to play Monopoly.」というメッセージを含む警告を表示します。

書式指定子%@のほかに、%*n*$@書式指定子を使用して、位置を指定して文字列変数を置き換えることができます。*n*は、loc-args内の置換対象の配列値のインデックス（1から始まる）です（パーセント記号（%）を表す%%指定子もあります）。たとえば、Localizable.stringsのエントリが次のようになっているとします。

```
"GAME_PLAY_REQUEST_FORMAT" = "%2$@ and %1$@ have invited you to play Monopoly";
```

デバイスは、「Frank and Jenna have invited you to play Monopoly.」というメッセージを含む警告を表示します。

loc-keyプロパティとloc-argプロパティを使用した通知ペイロードの完全な例については、“JSONペイロードの例”を参照してください。国際化の詳細については、『*Internationalization and Localization Guide*』を参照してください。文字列の書式化については、『*String Programming Guide*』の“Formatting String Objects”で解説されています。

> **注意:** loc-keyプロパティとloc-argsプロパティ（および、たいていのalert辞書）は、本当に必要な場合にだけ使用してください。これらのプロパティの値は、特に長い文字列の場合、パフォーマンスが向上するどころか帯域幅を使い果たすおそれがあります。多くのアプリケーションでは、これらのプロパティは必要ありません。メッセージ文字列はユーザから与えられるからです。

JSONペイロードの例

通知のペイロード部分に関する次の例は、表3-1で示したプロパティの実際の使い方を示しています。キー名に「acme」を含むプロパティは、カスタムペイロードデータの例です。

> **注意:** この例は、読みやすいよう、空白や改行を適宜入れています。実際には空白と改行を削除してペイロード長を短縮することにより、ネットワーク処理性能を改善できます。

例1.次のペイロードには、デフォルトの警告ボタン（「Close」と「View」）付きの警告メッセージに対応した簡単でお勧めの形式のaps辞書が含まれています。この例では、alertの値として、辞書ではなく文字列を使用しています。このペイロードにはカスタム配列プロパティも含まれています。

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

例2.この例のペイロードでは、aps辞書を使用して、左側には「Close」ボタン、右側の"アクション"ボタンにはローカライズされたタイトルが表示される警告メッセージをデバイスにリクエストしています。ここでは、「Play」と同じ意味のローカライズ文字列を取得するために、現在選択されている言語のLocalizable.stringsファイルのキーとして、「PLAY」を使用しています。また、このaps辞書は、アプリケーションアイコンに付けるバッジに5と表示するようにリクエストしています。Apple Watchの場合、タイトルキーは、新規リクエストがある旨をユーザに警告します。

```
{
    "aps" : {
        "alert" : {
            "title" : "Game Request",
            "body" : "Bob wants to play poker",
            "action-loc-key" : "PLAY"
```

```
        },
        "badge" : 5,
    },
    "acme1" : "bar",
    "acme2" : [ "bang",  "whiz" ]
}
```

例3.この例のペイロードでは、デバイスが「Close」と「View」の両方のボタンが付いた警告メッセージを表示するよう指定しています。また、アプリケーションアイコンに9というバッジを付け、通知の配信時にバンドルされている警告音を鳴らすこともリクエストしています。

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

例4.この例では、アプリケーションバンドルからローカライズ形式の文字列を取得し、適切な場所にある変数文字列値（`loc-args`）に置き換えるために、`alert`辞書の子プロパティ`loc-key`と`loc-args`を使用しています。また、カスタムサウンドを指定していますし、カスタムプロパティも含まれています。

```
{
    "aps" : {
        "alert" : {
            "loc-key" : "GAME_PLAY_REQUEST_FORMAT",
            "loc-args" : [ "Jenna", "Frank"]
        },
        "sound" : "chime.aiff"
    },
    "acme" : "foo"
}
```

例5.この例のペイロードは`category`キーを使って、通知アクションのグループを指定しています（カテゴリの詳細については、“通知アクションを扱う（iOS）”（27 ページ）を参照してください）。

```
{
   "aps" : {
      "category" : "NEW_MESSAGE_CATEGORY"
      "alert" : {
         "body" : "Acme message received from Johnny Appleseed",
         "action-loc-key" : "VIEW",
      },
      "badge" : 3,
      "sound" : "chime.aiff"
   },
   "acme-account" : "jane.appleseed@apple.com",
   "acme-message" : "message123456"
}
```

プロビジョニングおよび開発

開発環境と製品環境

クライアントサーバアプリケーションのプロバイダ側を開発してデプロイするには、Member CenterからSSL証明書を取得する必要があります。1つの証明書は、バンドルIDで識別される1つのアプリケーションに限定されています。また、証明書はそれぞれ、次の2つの開発環境のいずれか1つに限定されています。それぞれの開発環境は固有のホスト名を持っています。

- **開発環境**：開発環境は、プロバイダアプリケーションの初版の開発とテストに使用されます。サーバユニットの数は少ないものの、製品環境と同じサービスセットが提供されます。開発環境は、シミュレートされたエンドツーエンドテストを可能にする仮想デバイスとしての役割も果たします。

  開発環境には、`gateway.sandbox.push.apple.com`のアウトバウンドTCPポート2195からアクセスできます。

- **製品環境**：製品版のプロバイダアプリケーションをビルドするときは製品環境を使用します。製品環境を使用するアプリケーションは、Appleの信頼性要件を満たしている必要があります。

  製品環境には、`gateway.push.apple.com`のアウトバウンドTCPポート2195からアクセスできます。

開発環境用と製品環境用に、別々の証明書を取得する必要があります。この証明書はリモート通知を受信するアプリケーションの識別子に関連付けられます。この識別子にはアプリケーションのバンドルIDが含まれています。これらの環境のいずれかのプロビジョニングプロファイルを作成すると、必要な資格が自動的にこのプロファイルに追加されます。これには、リモート通知に固有の資格（`<aps-environment>`）も含まれます。プロビジョニングプロファイルには、開発用と配布用の2種類があります。配布用プロビジョニングプロファイルは、アプリケーションをApp Storeに投稿するための必要条件です。

> **OS Xに関する注意:** OS Xプロビジョニングプロファイルの資格は`com.apple.developer.aps-environment`で、プラットフォームに範囲を限定します。

どの環境で作業をしているかは、Xcodeでコード署名IDを選択すると判別できます。「iPhone Developer: *Firstname Lastname*」という証明書／プロビジョニングプロファイルのペアが表示された場合は、開発環境にいます。「iPhone Distribution: *Companyname*」という証明書／プロビジョニングプロファイルのペアが表示された場合は、製品環境にいます。この2つの環境を区別しやすくするために、Xcodeで配布用リリース構成を作成するとよいでしょう。

SSL証明書がプロビジョニングプロファイルに含まれていなくても、この証明書と特定のアプリケーションIDが関連付けられているため、`<aps-environment>`がプロファイルに追加されます。その結果、この資格がアプリケーションに組み込まれて、リモート通知の受信が可能になります。

プロビジョニングの手順

Apple Push Notificationサービス（APNs）が利用できるのは、iOS App StoreまたはMac App Storeを通して配布するアプリケーションと、エンタープライズアプリケーションに限ります。アプリケーションサービスを使うためには、プロビジョニングおよびコード署名が必要です。企業開発者の場合、この設定の大部分は、チームエージェントまたはチーム管理者だけが実施できるようになっています。

開発中のアプリケーションにコード署名を施し、プロビジョニングする手順については、『*App Distribution Quick Start*』を参照してください。APNsを有効にする手順は、『*App Distribution Guide*』の“Configuring Push Notifications” in *App Distribution Guide* に載っています。

しかし、APNsの全機能を有効にするためには、Member Centerでクライアント SSL証明書を作成しなければなりません（“Creating Push Notification Client SSL Certificates” in *App Distribution Guide* を参照）。アプリケーションの開発およびテスト時は、開発用クライアントSSL証明書を作成してください。クライアントSSL証明書に秘密鍵で署名するための署名IDが、キーチェーンに保存されます。クライアントSSL署名IDをエクスポートし、サーバにインストールする手順は、“Installing Client SSL Certificates” in *App Distribution Guide* に説明してあります。

実稼働用クライアントSSL証明書を作成する用意ができたら、“Creating Push Notification Client SSL Certificates” in *App Distribution Guide* に説明してある手順で作成しますが、「Development」ではなく「Production」以下のSSL証明書を選択する点が違います。実稼働用クライアントSSL証明書は、アプリケーションをストアに登録する前に作成しておかなければなりません。

APNsを利用するようアプリケーションを設定すると、iOSアプリケーションをベータテスト用にエクスポートし、あるいはストアに登録する際、Xcodeは自動的に、必要な配布用プロビジョニングプロファイルを生成します。

プロバイダとApple Push Notificationサービスの通信

この章では、Apple Push Notificationサービス（APNs）との通信のためにプロバイダが使用するインターフェイスについて説明し、プロバイダに期待されるいくつかの機能について解説します。

プロバイダの一般的な要件

プロバイダは、バイナリインターフェイスを介してApple Push Notificationサービスと通信します。このインターフェイスは、プロバイダ用の高速で大容量のインターフェイスです。このインターフェイスは、TCPソケットを介したストリーミング設計をバイナリコンテンツと組み合わせて使用しています。このバイナリインターフェイスは非同期です。

製品環境のバイナリインターフェイスは`gateway.push.apple.com`のポート2195から利用できます。開発環境のバイナリインターフェイスは`gateway.sandbox.push.apple.com`のポート2195から利用できます。

インターフェイスごとに、TLS（またはSSL）を使用してセキュアな通信チャンネルを確立します。この接続に必要なSSL証明書は、Member Centerから取得します（詳細については、“プロビジョニングおよび開発”（55 ページ）を参照してください）。認証済みのプロバイダであることを証明するため、接続時にピアツーピア認証を使用してこの証明書をAPNsに渡してください。

> **注意:** APNsとTLSセッションを確立するには、Entrust Secure CAルート証明書をプロバイダのサーバ上にインストールする必要があります。サーバがOS Xを実行している場合、このルート証明書はすでにキーチェーンに存在します。その他のシステム上では、証明書が存在しない可能性があります。この証明書はEntrust SSL Certificateのウェブサイトからダウンロードできます。

プロバイダは、リモート通知の次の側面を担当します。

- 通知ペイロードを作成する必要があります（“通知ペイロード”（46 ページ）を参照）。
- アプリケーションアイコンに表示するバッジの数字を提供する必要があります。
- 定期的にフィードバックサービスに接続して、何度も配信失敗が報告されているデバイスの最新リストを取得してください。そして、これらのアプリケーションに関連付けられているデバイスへの通知の送信を停止します。詳細については、“フィードバックサービス”（61 ページ）を参照してください。

複数の言語で通知メッセージをサポートする場合に、ローカライズされた警告文字列をクライアント側で取得するためにapsペイロード辞書のloc-keyプロパティとloc-argsプロパティを使用しないときは、サーバ側で警告メッセージのテキストをローカライズする必要があります。それには、クライアントアプリケーションから現在の言語環境を検出する必要があります。“ユーザ通知の登録、スケジューリング、対処”（16 ページ）では、この情報を取得するためのアプローチを示します。“通知ペイロード”（46 ページ）プロパティの詳細については、loc-key「通知ペイロード」loc-argsを参照してください。

接続管理に関するベストプラクティス

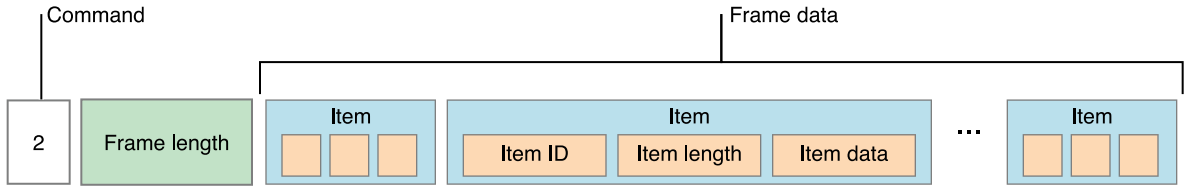
同じゲートウェイまたは複数のゲートウェイインスタンスに、複数の接続を確立することもできます。多数のリモート通知を送信する必要があれば、異なるゲートウェイを通る、いくつかの接続に分割してください。単一の接続に比べて性能を改善できます。リモート通知の送信だけでなく、APNsによる配送も高速になります。

APNsとの接続は、通知の都度切断せず、開いたままにしてください。APNsは、接続と切断がすばやく何度も繰り返される場合、それをDoS（サービス運用妨害：denial-of-service）攻撃と見なします。一定時間以上アイドル状態になると分かっている場合を除き、接続したままにしてください。たとえば、通知が日に1度だけであれば、その都度の接続で構いません。

バイナリインターフェイスの形式と通知形式

バイナリインターフェイスは、本質的にストリーミング方式のバイナリコンテンツにはプレーンなTCPソケットを採用します。パフォーマンスを最適化するために、TCP/IPのNagleアルゴリズムを使用するか、またはインターフェイスを通して明示的に複数の通知を1回の転送で一括送信してください。通知の形式を図 5-1に示します。

図 5-1 通知の形式



注意: データはすべてネットワークバイト順序（ビッグエンディアン）にしてください。

通知形式の最上位レベルは、順に次のようになっています。

フィールド名	長さ	説明
Command	1バイト	番号2に固定。
Frame length	4バイト	フレームデータ（Frame data）の長さ
Frame data	可変長	本体。一連の項目を構造化して収容。

フレームデータは、いくつかの項目を順に並べたものです。各項目は次のような構成になります。

フィールド名	長さ	説明
Item ID	1バイト	アイテムのID。たとえばペイロードの項目番号は2。
Item data length	2バイト	項目データの長さ。
Item data	可変長	アイテムの値。

アイテムとそのIDは次の通りです。

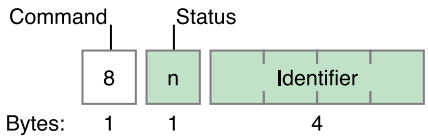
アイテムID	アイテム名	長さ	データ
1	デバイストークン	32バイト	バイナリ形式のデバイストークン（デバイスが登録したもの）。
2	ペイロード	可変長、2キロバイト以下	JSON形式のペイロード。 ペイロードはNULLで終了してはいけません。
3	通知の識別番号	4バイト	この通知を識別する何らかの値。エラーが発生したとき、サーバに報告するために使います。
4	有効期限	4バイト	秒（UTC）で表わされた固定のUNIXエポック日です。通知が無効になり破棄できるようになるタイミングを示します。 0以外の値であれば、APNsは通知をいったん保存し、少なくとも1度は配送しようとします。0であれば、通知は即座に無効になり、APNsが通知を保存することはなくなります。

アイテムID	アイテム名	長さ	データ
5	優先度	1バイト	通知の優先度次のいずれかの値を指定します。 • `10`：プッシュメッセージをすぐに送信します。 その結果、デバイスには、アラート、サウンド、バッジのいずれかが現れます。`content-available`キーのみのプッシュ通知に対してこの優先度を指定すればエラーになります。 • `5`：送信先デバイスが節電モードのときにプッシュメッセージを送信します。

送信した通知をAPNsが受け付けた場合、何も返されません。

形式が誤っているなど通知に問題があれば、APNsはエラーレスポンスパケットを返した後、切断してしまいます。それ以降に同じ接続を介して送信した内容は破棄されるので、再送を要します。図 5-2にエラーレスポンスパケットの形式を示します。

図 5-2 エラーレスポンスパケットの形式



パケットのコマンド値は8であり、その後に1バイトの状態コードと、件の通知の識別子が続きます。表 5-1に可能なステータスコードとその意味を示します。

表 5-1 エラーレスポンスパケット内のコード

状態コード	説明
0	エラーなし
1	処理エラー
2	デバイストークン欠如
3	トピック欠如

状態コード	説明
4	ペイロード欠如
5	無効なトークンサイズ
6	無効なトピックサイズ
7	無効なペイロードサイズ
8	無効なトークン
10	シャットダウン
255	なし（不明）

状態コードが10の場合、APNsサーバが（たとえば保守のため）接続を切ったことになります。エラーレスポンス中の通知識別子は、正常に送信できた直近の通知を表します。それ以降に送信した通知は破棄されているので、再送を要します。この場合、同じ接続は使用せず、新たに接続を開いてください。

本番環境のデバイストークンと開発環境のデバイストークンは同じ値ではないことに注意してください。

フィードバックサービス

Apple Push Notificationサービスにはフィードバックサービスという、正常に処理できなかったリモート通知に関する情報を返す仕組みがあります。リモート通知を配送できなかった原因が、配送先アプリケーションがデバイス上で動作していなかったためであった場合、フィードバックサービスは当該デバイスのトークンをリストに追加します。リモート通知が配送前に期限切れになった場合、配送に失敗したことにはならないので、フィードバックサービスの対象外です。この情報に基づき、配送に失敗すると予測されるリモート通知は初めから送信しないようにすれば、不要なメッセージによるオーバーヘッドを削減し、システム全体の処理性能を改善できます。

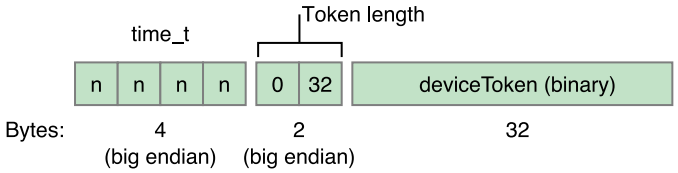
毎日定期的にフィードバックサーバに問い合わせて、デバイストークンのリストを取得してください。タイムスタンプをもとに、フィードバックエントリが生成されて以降、該当するデバイストークンが登録されないようにします。登録されていない各デバイスについて、通知の送信を停めてください。APNsは、プロバイダがフィードバックサービスをチェックして、デバイス上に存在しないアプリケーションへのリモート通知の送信を停止する作業をこまめに行っているかどうかを監視します。

> **注意:** フィードバックサーバは、プッシュトピック別にリストを管理します。アプリケーションが複数ある場合、それぞれについて、対応する証明書を用いて、フィードバックサービスに接続しなければ、フィードバックをすべて取得することはできません。

フィードバックサービスへのアクセスは、リモート通知の送信と同様のバイナリインターフェイスを介して行われます。製品環境のフィードバックサービスには`feedback.push.apple.com`のポート2196からアクセスします。開発環境のフィードバックサービスには`feedback.sandbox.push.apple.com`のポート2196からアクセスします。リモート通知のバイナリインターフェイスと同様に、TLS（またはSSL）を使用してセキュアな通信チャンネルを確立する必要があります。通知の送信に使っているのと同じSSL証明書で、フィードバックサービスにも接続できます。認証済みのプロバイダであることを証明するため、接続時にピアツーピア認証を使用してこの証明書をAPNsに渡してください。

接続すると、ただちに転送が始まります。APNsには一切コマンドを送信する必要はありません。読み込むべきデータがなくなるまで、フィードバックサービスによって書き込まれたストリームを読み込みます。受信したデータは次の形式を持つタプルです。

図 5-3 フィードバックタプルのバイナリ形式



Timestamp	デバイス上にアプリケーションがもう存在しないとAPNsが判断した日時を表すタイムスタンプ（4バイトの`time_t`値）。この値（ネットワークバイト順になっている）は、1970年1月1日12:00からの秒数（UTCに基づく）を表します。
トークン長	デバイストークンの長さ（ネットワークバイト順による2バイトの整数値）。
デバイストークン	バイナリ形式のデバイストークン。

フィードバックサービスが保持しているリストは、読み取り後、クリアされてしまいます。フィードバックサービスに接続したときに得られる情報は、前回接続した時点以降、配送に失敗した通知に関するものだけです。

従前の形式について

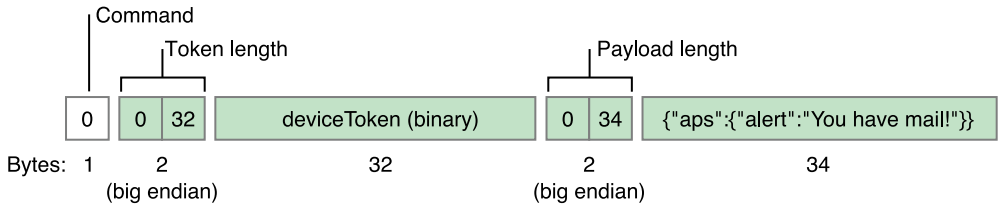
新規に開発する場合は、最新の形式でAPNsに接続するようにしてください（“バイナリインターフェイスの形式と通知形式” （58 ページ）を参照）。

以下の形式には優先度を示すフィールドがありません。優先度は10と看做されます。

通知の単純形式

図 A-1にこの形式を示します。

図 A-1　単純形式の通知



旧形式の最初のバイトはコマンド値で0（ゼロ）です。他のフィールドは拡張形式と同じです。リスト A-1に、単純形式の通知を使い、バイナリインターフェイスを介してAPNsへリモート通知を送信する関数の例を示します。この例は、事前にgateway.push.apple.com（または gateway.sandbox.push.apple.com）にSSL接続してピア交換認証を行っていることを前提としています。

リスト A-1　バイナリインターフェイスを通しての単純形式の通知の送信

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff,
size_t payloadLength)
{
    bool rtn = false;
    if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)
    {
        uint8_t command = 0; /* コマンド番号 */
        char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint16_t) +
```

```
            DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];
        /* メッセージ形式は |COMMAND|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */
        char *binaryMessagePt = binaryMessageBuff;
        uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);
        uint16_t networkOrderPayloadLength = htons(payloadLength);

        /* コマンド */
        *binaryMessagePt++ = command;

       /* トークン長（ネットワークバイト順序） */
        memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

        /* デバイストークン */
        memcpy(binaryMessagePt, deviceTokenBinary, DEVICE_BINARY_SIZE);
        binaryMessagePt += DEVICE_BINARY_SIZE;

        /* ペイロード長（ネットワークバイト順序） */
        memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

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

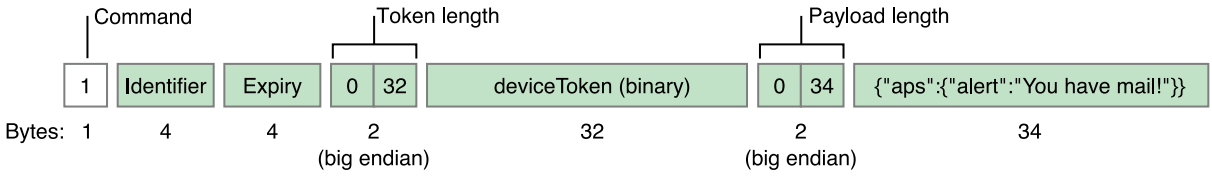
拡張形式の通知

拡張形式は、単純形式に比べていくつか改善が施されています。

- **エラーレスポンス**。単純形式では、何らかの誤った形式の（たとえば指定の限度を超えたペイロードなど）通知パケットを送った場合、APNsは接続を切断することでレスポンスします。なぜ通知が拒否されたか、理由は一切表示されません。拡張形式では、任意の識別子をつけてプロバイダに通知を区別させます。エラーがあれば、APNsはエラーコードと識別子を結びつけるパケットを返します。このレスポンスにより、プロバイダは誤った形式の通知を特定し、修正することができます。
- **通知の期限**。APNsには、デバイス上のアプリケーションに送信された最新の通知を保持する蓄積交換機能があります。配信時にデバイスがオフラインの場合、APNsはデバイスが次にオンラインになった時に通知を配信します。単純形式では、その通知が適切かどうかに関わらず通知が配信されます。つまり、時間の経過とともに通知は"古く"なっている場合があります。拡張形式では、通知が有効な期間を示す期限の値が含まれます。APNsは、この期間を経過すると蓄積交換内の通知を破棄します。

図 A-2に通知パケットの形式を示します。

図 A-2　拡張形式の通知



最初のバイトはコマンド値で1です。以降のフィールドは次のようになっています。

- **識別子**。この通知を識別する任意の値です。APNsが通知を解釈できない場合、同じ識別子がエラーレスポンスパケット内に返されます。
- **期限**。秒（UTC）で表わされた固定のUNIXエポック日です。通知が無効になり破棄できるようになるタイミングを示します。期限の値はネットワークバイト順序（ビッグエンディアン）です。期限の値が0以外であれば、APNsは少なくとも一度は通知を配信しようとします。0であれば、APNsは通知をまったく保持しないようになります。
- **トークン長**。デバイストークンの長さをネットワークバイト順序（ビッグエンディアン）で表します。
- **デバイストークン**。バイナリ形式のデバイストークン。
- **ペイロード長**。ペイロード長をネットワークバイト順序（ビッグエンディアン）で表します。ペイロードは256バイトを超えたり、ヌルで終了したりしては*いけません*。
- **ペイロード**。通知ペイロード。

リスト A-2では、拡張形式でリモート通知を生成し、APNsに送信しています。事前に gateway.push.apple.com（またはgateway.sandbox.push.apple.com）にSSL接続してピア交換認証を行っていることを前提としています。

リスト A-2　バイナリインターフェイスを通しての拡張形式の通知の送信

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff,
size_t payloadLength)
{
  bool rtn = false;
  if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)
  {
      uint8_t command = 1; /* コマンド番号 */
     char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint32_t) + sizeof(uint32_t)
 + sizeof(uint16_t) +
          DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];
      /* メッセージ形式は |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);

      /* コマンド */
      *binaryMessagePt++ = command;

     /* プロバイダ指定の順序のID */
     memcpy(binaryMessagePt, &whicheverOrderIWantToGetBackInAErrorResponse_ID,
sizeof(uint32_t));
     binaryMessagePt += sizeof(uint32_t);

     /* 期限日（ネットワークバイト順序） */
     memcpy(binaryMessagePt, &networkOrderExpiryEpochUTC, sizeof(uint32_t));
     binaryMessagePt += sizeof(uint32_t);

     /* トークン長（ネットワークバイト順序） */
```

```
        memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

        /* デバイストークン */
        memcpy(binaryMessagePt, deviceTokenBinary, DEVICE_BINARY_SIZE);
        binaryMessagePt += DEVICE_BINARY_SIZE;

        /* ペイロード長（ネットワークバイト順序） */
        memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

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

書類の改訂履歴

この表は「*Local および Push Notification プログラミングガイド*」の改訂履歴です。

日付	メモ
2015-09-16	カスタム通知サウンドの保存可能な場所について明確にし、さらに細かな訂正を加えました。
2015-03-09	「Settings」アプリケーションで、ユーザが通知設定を変更できる旨の情報を追加しました。
2014-10-31	細かな訂正を行いました。
2014-10-16	カスタム通知アクションおよびロケーションベースの通知に関する情報を追加しました。OS Xには未対応の通知タイプがある旨の記述を削除しました。現在では、あらゆるタイプの通知を配送できるようになりました。 題名を『*Local および Push Notification プログラミングガイド*』から変更しました。
2014-09-17	content-availableキーに関する注記を追加しました。カスタム通知アクションおよびロケーションベースの通知に関する情報を追加しました。 題名を『*Local および Push Notification プログラミングガイド*』から変更しました。
2014-02-11	content-availableキーに関する注記を追加しました。
2013-09-18	プッシュ通知の優先度設定に関する説明を追加しました。

日付	メモ
2013-04-23	「プロバイダとApple Push Notificationサービスの通信」の章を改訂しました。他の章についても細かな変更を施してあります。 “接続管理に関するベストプラクティス” （58 ページ） の節を追加しました。エラーコード「10」を表 5-1 （60 ページ） に追加しました。“フィードバックサービス” （61 ページ） の説明を詳しくしました。旧プロトコルに関する説明を付録に移動しました。
2011-08-09	OS Xデスクトップクライアントでのプッシュ通知の実装に関する情報を追加しました。iOS向けおよびOS X向けのガイドを統合しました。
2010-08-03	ユーザが通知警告のアクションボタンをタップすることでアプリケーションを起動したかどうかを判別する方法について説明しました。
2010-07-08	「iPhone OS」という記述を「iOS」に変更しました。
2010-05-27	iOS 4.0で導入されたローカル通知機能についての記述を更新、再編しました。APNsに送信するプッシュ通知の新しい形式についても説明しています。
2010-01-28	細かな訂正を数多く行いました。
2009-08-14	細かな訂正を行い、Cocoaの概念に関する短い記事へのリンクを作成しました。
2009-05-22	Wi-Fiおよび登録頻度についての注記を追加し、開発環境用のゲートウェイアドレスを追加しました。いくつかの説明を明確化し、拡張しました。
2009-03-15	プロバイダがApple Push Notificationサービスを使用してクライアントアプリケーションにPush Notificationを送信する仕組みについて説明した文書の初版。

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

Apple Japan
〒106-6140 東京都港区六本木 6
丁目10番1号 六本木ヒルズ
http://www.apple.com/jp

Offline copy. Trademarks go here.