位置情報対応プログラミングガイド

2011-10-12

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

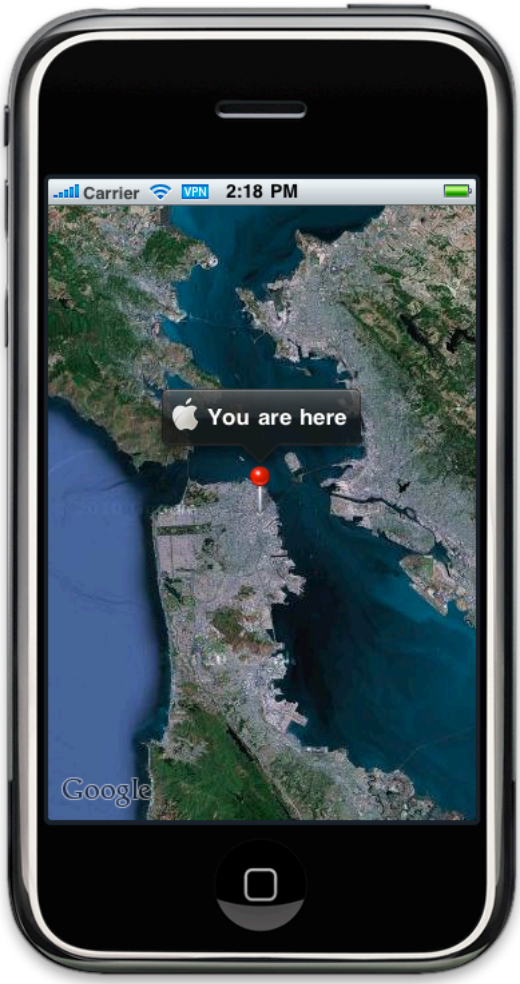
アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/

目次

図、表、リスト

アプリケーションを位置情報対応にする

アプリケーションに位置ベースの情報を取り入れることで、ユーザが周囲の世界とのつながりを保てる手段を提供できます。位置ベースの情報を実用目的で利用するのか（たとえばナビゲーション）、娯楽目的で利用するのかを問わず、こうした情報によりユーザ体験を全体的に向上させることができます。



iOSにおける位置ベースの情報は、位置情報サービスと地図の2つで構成されます。位置情報サービスはCore Locationフレームワークによって提供されます。このフレームワークは、ユーザの位置と方向に関する情報を得るためのObjective-Cのインターフェイスを備えています。地図はMap Kitフレームワークによって提供されます。このフレームワークは、「マップ(Maps)」アプリケーションと同じように、地図の表示と注釈をサポートします。

概要

地図および位置情報サービスは、ユーザとの対話操作を向上させる手段となります。地理データをアプリケーションに組み込むことにより、ユーザを周囲の環境に向けさせたり、周囲の人々とのつながりを保てるようにしたりできます。

アプリケーションに地理上のコンテキストを提供する位置情報サービス

位置情報サービスとは、すなわち機動性の源泉であり、どこにでも持っていけるデバイス上でアプリケーションが動作しているという事実を反映するものです。ユーザの地理的な位置を知ることで、提供する情報の質を上げることができ、時にはそれがアプリケーションの中核をなすこともあります。ナビゲーション機能を備えたアプリケーションは、位置情報サービスを利用してユーザの位置を観測し、更新情報を生成します。その他の多くのタイプのアプリケーションは、近くにいるユーザ同士を社会的につなぐ手段として位置情報を利用します。

関連する章： 「ユーザの位置の取得」（11 ページ）、「位置データのジオコーディング」（23 ページ）

ユーザの現在の向きを示すヘディング情報

ヘディングサービスは基本的な位置情報サービスを補うもので、デバイスが向いている方向に関する、より正確な情報を提供します。このテクノロジーの使用法としてすぐに思いつくのはコンパスを実装することですが、AR（拡張現実）、ゲーム、およびナビゲーションアプリケーションをサポートする目的にも使用されます。磁気センサー（正確なヘディング情報を得るためのハードウェア）のないデバイス上でも、ユーザの進路と速度に関する情報をアプリケーションで利用できます。

関連する章： 「方向関連のイベントの取得」（19 ページ）

ナビゲーションおよび地理的に関連のあるコンテンツの表示をサポートするマップ

地図は地理データを分かりやすい方法で視覚化する手段です。Map Kitフレームワークは、アプリケーションに組み込んで地理上の特定の地点に結び付けられた情報を表示できる標準的なビューを提供します。さらに、このフレームワークはカスタム情報を地図の前面に配置し、それを地図のほかの内容とともにスクロールさせる手段を提供します。

関連する章： 「地図の表示」（29 ページ）、「地図の注釈」（35 ページ）

この文書の使い方

個々のテクノロジーを使用するにあたり、この文書全体を読み通す必要はありません。Core Location フレームワークおよびMap Kitフレームワークによって提供されるサービスは独立しており、ほかのサービスに依存せずに使用できます。このため、各章の冒頭では、各テクノロジーを理解するために必要な用語と情報を紹介し、テクノロジーの使用方法に関するサンプルと、タスク関連の手順を取り上げています。「地図の注釈」（35 ページ）の章だけは、「地図の表示」（29 ページ）の章に掲載されている情報を前提に書かれています。

関連項目

Core Locationフレームワークのクラスの詳細については、『*Core Location Framework Reference*』を参照してください。

Map Kitフレームワークのクラスの情報については、『*Map Kit Framework Reference*』を参照してください。

ユーザの位置の取得

アプリケーションは、ソーシャルネットワーキングサービスから道案内サービスまで、さまざまな目的で位置情報を利用します。こうしたアプリケーションは、Core Locationフレームワークのクラスを使用して位置データを取得します。このフレームワークは、次に示すように、デバイスの現在位置の取得と観測が可能なサービスを提供しています。

- 大幅変更位置情報サービス。このサービスは、少ない電力消費で現在位置を取得し、その位置の変更を通知することができます（iOS 4.0以降）。
- 標準位置情報サービス。現在位置を取得するためのより詳細な設定が可能な方法を提供します。
- 領域観測。このサービスにより、あらかじめ定義された地域の境界線を横断したことを観測できます（iOS 4.0以降）。

Core Locationフレームワークの機能を使用するには、Xcodeプロジェクトでアプリケーションを`CoreLocation.framework`にリンクする必要があります。このフレームワークのクラスとヘッダにアクセスするには、関連するソースファイルの先頭に`#import <CoreLocation/CoreLocation.h>`ステートメントを追加します。

Core Locationフレームワークのクラスの全般的な情報については、『*Core Location Framework Reference*』を参照してください。

実行のために位置情報サービスを要求する

アプリケーションを正しく機能させるために位置情報サービスが必要な場合は、アプリケーションの`Info.plist`ファイルに`UIRequiredDeviceCapabilities`キーを追加する必要があります。このキーを使用して、アプリケーションの実行に位置情報サービスが必要であることを指定します。App Storeは、このキーの情報を使用して、リストに含まれている機能のないデバイスへのアプリケーションのダウンロードを防ぎます。

`UIRequiredDeviceCapabilities`の値は、アプリケーションが必要とする機能を示す文字列の配列です。位置情報サービスに関連する文字列は2つあります。

- 一般的な位置情報サービスを必要とする場合は、文字列`location-services`を指定します。
- GPSハードウェアでしか得られない測地精度を必要とする場合は、文字列`gps`を指定します。

重要： アプリケーションで位置情報サービスを利用するけれども、位置情報がなくても正常に動作する場合は、対応する文字列を`UIRequiredDeviceCapabilities`キーに指定しないでください。

`UIRequiredDeviceCapabilities`キーの詳細については、『*Information Property List Key Reference*』を参照してください。

ユーザの現在位置の取得

Core Locationフレームワークを使用すると、デバイスの現在位置を検出してその情報をアプリケーションで使用できます。このフレームワークは、デバイスに組み込まれている携帯電話、Wi-Fi、GPSなどのハードウェアから取得した情報を利用して、そのデバイスの位置を三角法で測ります。フレームワークは、その位置をアプリケーションのコードに報告し、サービスをどのように設定したかに応じて、新しいデータやより精度の高いデータを受信すると定期的に更新情報も提供します。

ユーザの現在位置を取得するには、次に2つのサービスがあります。

- **標準位置情報サービス**。設定可能な汎用のソリューションであり、すべてのバージョンのiOSでサポートされます。
- **大幅変更位置情報サービス**。携帯電話用の無線機能を備えたデバイスで利用できる低電力の位置情報サービスです。このサービスは、iOS 4.0以降でのみ利用でき、一時停止中または実行していないアプリケーションを作動させることもできます。

位置データの収集には、集中的に電力を消費します。内蔵無線の電源をオンにし、利用可能な携帯電話の基地局、Wi-Fiホットスポット、またはGPS衛星を照会する必要があるからです。これには数秒かかる場合があります。標準位置情報サービスを長時間実行したままにすると、デバイスのバッテリーが切れてしまう可能性があります（大幅変更位置情報サービスは、携帯電話の基地局の変更のみを観測することによってバッテリー消費を大幅に抑えますが、携帯電話の無線機能のあるデバイスでのみ動作します）。ほとんどのアプリケーションの場合、最初の位置を検出した後は、定期的に更新情報を取得すれば十分です。位置情報の定期的な更新が必ず必要な場合は、大幅変更位置情報サービスを可能な限り使用するべきです。そうしない場合は、バッテリー駆動時間への影響を極力抑えるように標準位置情報サービスのパラメータを設定します。

位置情報サービスの利用可否の確認

iOSベースのデバイスにはすべて、何らかの形で位置情報サービスをサポートする機能がありますが、状況によっては位置情報サービスを利用できないこともあります。

- ユーザが「設定(Settings)」アプリケーションで位置情報サービスを無効にしている。
- ユーザが特定のアプリケーションに対して位置情報サービスを拒否している。
- デバイスが機内モードになっていて、必要なハードウェアの電源を入れられない。

このような理由から、標準または大幅変更位置情報サービスを開始する前に、必ず`CLLocationManager`の`locationServicesEnabled`クラスメソッドを呼び出すべきです（iOS 3.xおよびそれ以前では、代わりに`locationServicesEnabled`プロパティの値を確認してください）。このクラスメソッドから`YES`が返された場合は、予定通り位置情報サービスを開始できます。`NO`が返された場合に位置情報サービスを開始しようとすると、システムはユーザに位置情報サービスを再度有効にするべきかどうか確認を求めてきます。位置情報サービスが意図的に無効になっている可能性が高いと考えると、ユーザがこの確認を歓迎しない可能性があります。

標準位置情報サービスの開始

標準位置情報サービスは、すべてのデバイス上で、すべてのバージョンのiOSで利用できるため、ユーザの現在位置を取得する最も一般的な方法です。このサービスを利用する前に、位置データの測地精度と、新しい位置を報告するまでに必要な移動距離を指定します。標準位置情報サービスは開始時に、指定されたパラメータに応じてどの無線機能を有効にするかを決めて、その後アプリケーションへ位置情報イベントを報告します。

標準位置情報サービスを使用するには、`CLLocationManager`クラスのインスタンスを作成し、その`desiredAccuracy`プロパティおよび`distanceFilter`プロパティを設定します。位置情報通知の受け取りを開始するには、そのオブジェクトにデリゲートを割り当て、`startUpdatingLocation`メソッドを呼び出します。位置データが利用可能になると、この位置情報マネージャは割り当てられているデリゲートオブジェクトに通知をします。位置情報の更新がすでに送信されていれば、次のイベントの送信を待たずに、最新の位置データを`CLLocationManager`オブジェクトから直接取得することもできます。

リスト1-1に、使用する位置情報マネージャを設定するメソッドの例を示します。このメソッドは、位置情報マネージャオブジェクトを後での使用に備えてメンバ変数にキャッシュするクラスの一部です（このクラスはまた、`CLLocationManagerDelegate`プロトコルにも準拠しているため、位置情報マネージャのデリゲートとしても機能します）。アプリケーションが厳密な位置データを必要としないため、ユーザのおおよその地域を報告し、ユーザが長距離（この例では0.5km）を移動したときにのみ通知するように位置情報サービスを設定しています。

リスト 1-1　標準位置情報サービスの開始

```
- (void)startStandardUpdates
{
    // このオブジェクトにまだ位置情報マネージャがなければ、
    // 位置情報マネージャを作成する
    if (nil == locationManager)
        locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;
    locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;

    // 新しいイベント用に、移動のしきい値を設定する
    locationManager.distanceFilter = 500;

    [locationManager startUpdatingLocation];
}
```

このサービスから位置情報の更新を受信するコードは、「サービスからの位置データの受信」（14 ページ）に示します。

大幅変更位置情報サービスの開始

iOS 4.0以降では、大幅変更位置情報サービスを使用して位置情報イベントを受信することができます。このサービスは電力を大幅に節約し、測地精度もほとんどのアプリケーションにとって十分適しています。デバイスの携帯電話用無線を利用してユーザの位置を特定し、その位置の変更を報告します。これによりシステムは、電力使用量をより積極的に管理することができます。このサービスは、新しい位置データを配信するために、現在一時停止中または実行していないアプリケーションを作動させることもできます。

大幅変更位置情報サービスを使用するには、リスト 1-2に示すように、CLLocationManagerクラスのインスタンスを作成し、デリゲートを割り当て、startMonitoringSignificantLocationChangesメソッドを呼び出します。位置データが利用可能になると、この位置情報マネージャは割り当てられているデリゲートオブジェクトに通知をします。位置情報の更新がすでに送信されていれば、次のイベントの送信を待たずに、最新の位置データをCLLocationManagerオブジェクトから直接取得することもできます。

リスト 1-2　大幅変更位置情報サービスの開始

```
- (void)startSignificantChangeUpdates
{
    // このオブジェクトにまだ位置情報マネージャがなければ、
    // 位置情報マネージャを作成する
    if (nil == locationManager)
        locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;
    [locationManager startMonitoringSignificantLocationChanges];
}
```

「サービスからの位置データの受信」（14 ページ）で説明しているように、位置データは標準位置情報サービスと同様にデリゲートオブジェクトに送信されます。

このサービスを実行したままにしてアプリケーションが一時停止または終了させられた場合、新しい位置データが到着したときに、サービスは自動的にアプリケーションを作動させます。作動させられたアプリケーションはバックグラウンドに回され、位置データを処理するための短い時間が与えられます。アプリケーションはバックグラウンドにあるため、行う処理を最小限に抑え、割り当てられた時間が経過するまで復帰しない可能性のあるタスク（ネットワークへの問い合わせなど）は避けます。さもなければアプリケーションは強制終了させられることがあります。

サービスからの位置データの受信

標準位置情報サービスを利用するか、大幅変更位置情報サービスを利用するかを問わず、位置情報イベントを受信する方法は同じです。新しいイベントがあると、位置情報マネージャはそれをデリゲートのlocationManager:didUpdateToLocation:fromLocation:メソッドに報告します。イベントの取得時にエラーが発生すると、位置情報マネージャはデリゲートのlocationManager:didFailWithError:メソッドを代わりに呼び出します。

リスト 1-3に、位置情報イベントを受信するデリゲートメソッドを示します。位置情報マネージャオブジェクトは、キャッシュされているイベントを返すことがあるため、受け取った位置情報イベントのタイムスタンプを確認することを推奨します（大まかな位置を取得するまで数秒を要する場合があります。したがって、古いデータは、最後に取得した位置を表す手段としてのみ役立ちます）。この例に示すメソッドは、ごく最近のイベントであれば十分に有効であるという想定のもと、15秒以上が経過しているイベントをすべて破棄します。ナビゲーションアプリケーションを実装している場合は、このしきい値を下げる必要があるかもしれません。

リスト 1-3　受信した位置情報イベントの処理

```
// CLLocationManagerDelegateプロトコルのデリゲートメソッド
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation
{
    // 比較的新しいイベントの場合は、節電のために更新をオフにする
```

```
    NSDate* eventDate = newLocation.timestamp;
    NSTimeInterval howRecent = [eventDate timeIntervalSinceNow];
    if (abs(howRecent) < 15.0)
    {
        NSLog(@"latitude %+.6f, longitude %+.6f\n",
                newLocation.coordinate.latitude,
                newLocation.coordinate.longitude);
    }
    // それ以外の場合は、このイベントをスキップして次のイベントを処理する
}
```

位置情報オブジェクトのタイムスタンプのほか、イベントを受け取るかどうかを判断する材料として、オブジェクトから報告された精度を使用することもできます。位置情報サービスは、より正確なデータを受け取ると、その精度の向上を反映した精度値を持つ追加イベントを返します。精度の低いイベントを破棄するということは、アプリケーションが、効果的に使用できないイベントに浪費する時間を減らせるということです。

形状に基づく領域観測

iOS 4.0以降では、アプリケーションは領域観測を使用して、ユーザが地理上の境界線を横断したときにそれを通知することができます。この機能を使用して、ユーザが特定の場所に近づいたときにアラートを発することができます。たとえば、特定のドライクリーニング店に近づいたら、アプリケーションでは、預けていてもう仕上がっている服を受け取るようにユーザに通知することができます。

アプリケーションに関連付けられた領域は、アプリケーションが実行していないときも含め、常に追跡されます。アプリケーションが実行していないときに、ある領域の境界線を横切ると、アプリケーションはそのイベントを処理するためにバックグラウンドで起動されます。同様に、イベントが発生したときにアプリケーションが一時停止中の場合、アプリケーションは作動させられて、イベントを処理するために短い時間が割り当てられます。

領域観測の利用可否の確認

特定の領域を観測しようとする前に、アプリケーションは、現在のデバイス上で領域観測がサポートされているかを確認するべきです。領域観測が利用できない理由はいくつか考えられます。

- デバイスが領域観測に必要なハードウェアを備えていない。
- ユーザが「設定(Settings)」アプリケーションで位置情報サービスを無効にしている。
- デバイスが機内モードになっていて、必要なハードウェアの電源を入れられない。

このような理由から、領域を観測する場合は、必ず`CLLocationManager`のクラスメソッドである`regionMonitoringAvailable`と`regionMonitoringEnabled`を呼び出すことが推奨されます。`regionMonitoringAvailable`メソッドを使用して、対象のハードウェアが領域観測をサポートしているかを確認できます。このメソッドから`NO`が返された場合、アプリケーションはそのデバイスで領域観測を使用することはできません。領域観測が利用可能であれば、`regionMonitoringEnabled`メソッドは、この機能が現在有効になっているかどうかを報告します。領域観測が利用可能であっ

ても、領域を観測しようとしたときに有効になっていなければ、システムはユーザに対して領域観測を再度有効にするかどうか確認を求めてきます。この機能が意図的に無効になっている可能性が高いと考えると、ユーザがこの確認を歓迎しない可能性があります。

観測する領域の定義

ある領域の観測を始めるには、領域を定義してそれをシステムに登録しなければなりません。領域は、`CLRegion`クラスを使って定義します。このクラスは現在、円形の領域の作成をサポートしています。作成する各領域には、地理上の地域を定義するデータと、一意の識別文字列の両方が含まれている必要があります（識別文字列は必須であり、アプリケーションが後で領域を特定するための唯一確実な方法です）。領域を登録するには、`CLLocationManager`オブジェクトの`startMonitoringForRegion:desiredAccuracy:`メソッドを呼び出します。

リスト 1-4に、円形のMap Kitオーバーレイに基づいて新しい領域を作成するメソッドの例を示します。オーバーレイの中心点と半径がこの領域の境界線を形成しますが、半径が大きすぎて観測できない場合は、半径が自動的に縮小されます。領域を登録した後は、領域オブジェクト自体は解放できます。Core Locationは領域に関連付けられたデータを保存しますが、通常、領域オブジェクト自体は保存しません。

リスト 1-4　Map Kitオーバーレイに基づいた領域の作成と登録

```
- (BOOL)registerRegionWithCircularOverlay:(MyCircle*)overlay
andIdentifier:(NSString*)identifier
{
   // サポートが利用できない、または無効な場合には領域を作成しない
   if ( ![CLLocationManager regionMonitoringAvailable] ||
        ![CLLocationManager regionMonitoringEnabled] )
      return NO;

   // 半径が大きすぎる場合、登録は自動的に失敗に終わるため、
   // 半径を最大値に固定
   CLLocationDegrees radius = overlay.radius;
   if (radius > self.locManager.maximumRegionMonitoringDistance)
      radius = self.locManager.maximumRegionMonitoringDistance;

   // 領域を作成し観測を開始する
   CLRegion* region = [[CLRegion alloc]
initCircularRegionWithCenter:overlay.coordinate
                        radius:radius identifier:identifier];
   [self.locManager startMonitoringForRegion:region
                    desiredAccuracy:kCLLocationAccuracyHundredMeters];

   [region release];
   return YES;
}
```

領域の観測は、登録直後に始まります。しかし、イベントをすぐに受信するものと想定しないでください。イベントは、境界線を横断したときにだけ生成されます。そのため、登録時にユーザの位置がすでに領域の内側にある場合、位置情報マネージャはイベントを生成しません。ユーザが領域の境界線を横断するまで待たないと、イベントが生成されてデリゲートに送信されることはありません。

観測する一連の領域を指定するときには常に慎重に検討する必要があります。領域は共有のシステムリソースであり、システム全体で利用可能な領域の総数は限られています。このため、Core Locationは、1つのアプリケーションで同時に観測できる領域の数を制限しています。このような制限を回避するには、ユーザの近隣の領域のみ登録することを検討してください。ユーザの位置が変わるのにつれて、距離の遠くなった領域を削除し、ユーザの進路上で近づいている領域を追加することができます。領域を登録しようとしたときに空きがないと、位置情報マネージャはデリゲートの`locationManager:monitoringDidFailForRegion:withError:`メソッドを呼び出して、エラーコード`kCLErrorRegionMonitoringFailure`を渡します。

領域の境界線横断イベントの処理

ユーザの現在位置が境界領域を超えるたびに、システムはアプリケーションに対して適切な領域イベントを生成します。アプリケーションが実行中の場合、これらのイベントは現在の位置情報マネージャオブジェクトのデリゲートに直接送信されます。アプリケーションが実行中でなければ、システムは、アプリケーションをバックグラウンドで起動してイベントに応答できるようにします。アプリケーションは、次のメソッドを実装して境界線の横断を処理することができます。

- `locationManager:didEnterRegion:`
- `locationManager:didExitRegion:`

システムは、境界線に加え、指定された緩衝距離を超えるまで境界線の横断を報告しません。領域に対して必要な緩衝距離は、領域の登録時に`startMonitoringForRegion:desiredAccuracy:`メソッドを使用して指定します。緩衝距離は、ユーザが境界線付近を移動しているときに、システムが大量の進入・退出イベントを立て続けに生成するのを防ぐための値です。

領域の境界線を横断したときの最も可能性の高い応答は、対象物に接近していることをユーザに知らせることです。アプリケーションがバックグラウンドで実行中の場合、ローカル通知を使用してユーザに知らせることができます。それ以外の場合は単にアラートを送信できます。

バックグラウンドでの位置情報イベントの取得

アプリケーションがフォアグラウンドにあるかバックグラウンドにあるかを問わず、位置情報の更新を受ける必要がある場合は、いくつかの選択肢があります。推奨される選択肢は、大幅変更位置情報サービスを使用して、アプリケーションを適時に作動させて新しいイベントを処理することです。しかし、アプリケーションで標準位置情報サービスを使用する必要がある場合は、バックグラウンド位置情報サービスが必要なアプリケーションとして宣言することができます。

バックグラウンド位置情報サービスの要求は、それがないとアプリケーションの動作能力が損なわれるという場合に限り行うべきです。さらに、バックグラウンド位置情報サービスを要求するアプリケーションは、これらのサービスにより、ユーザに具体的なメリットをもたらさなければなりません。たとえば、道案内アプリケーションであれば、ユーザの位置を追跡して次の曲がり角を曲がるタイミングを報告する必要があるため、バックグラウンド位置情報サービスの候補になり得るでしょう。

バックグラウンド位置情報アプリケーションを設定する手順については、『*iOS App Programming Guide*』の「Executing Code in the Background」で説明しています。

バッテリー電力節約のためのヒント

iOSベースデバイスの無線によるデータの送受信は、デバイス上のその他の操作に比べ、より多くの電力を必要とします。Core Locationはこれらの無線を利用してユーザの位置を特定するため、アプリケーションでの位置情報サービスの利用については慎重を期す必要があります。ほとんどのアプリケーションでは、位置情報サービスを実行させたままにしておく必要はないため、サービスをオフにするだけで電力の節約になります。

- **使用しない間は位置情報サービスをオフにする。**分かりきったことのように思えますが、繰り返します。曲がり角ごとに指示を出す道案内アプリケーションを除き、ほとんどのアプリケーションでは、位置情報サービスを実行したままにしておく必要はありません。位置情報サービスは、位置情報を取得するのに必要な間だけオンにして、その後はオフにしてください。ユーザが乗り物で移動中でない限り、現在位置は問題になるほど頻繁には変化しません。また、必要であれば後からいつでも、位置情報サービスを再開できます。

- **可能な場合は、標準位置情報サービスではなく、大幅変更位置情報サービスを使用する。**大幅変更位置情報サービスでは、電力を大幅に節約しながらサービスを動かしたままにしておけます。これは、ユーザの位置変更を追跡する必要があるけれども、標準位置情報サービスで提供されるような高度な測地精度は必要としないアプリケーションに強く推奨されます。

- **アプリケーションに支障をきたさない限り、測地精度にはより低い精度を使用する。**必要以上に高い測地精度を要求すると、Core Locationは追加のハードウェアに電力を投入し、不要な精度に対して電力を浪費することになります。アプリケーションにおいてユーザの位置を数メートルの範囲で本当に認識する必要がある場合を除き、`desiredAccuracy`プロパティに`kCLLocationAccuracyBest`や`kCLLocationAccuracyNearestTenMeters`の値を設定しないでください。また、`kCLLocationAccuracyThreeKilometers`の値を指定しても、位置情報サービスからより精度の高いデータが返されるのを避けることができません。Core Locationは、ほとんどの場合、Wi-Fiまたは携帯電話の信号に基づいて100メートル程度以内の測地精度で位置データを返すことができます。

- **一定期間を過ぎても測地精度が上がらない場合は位置情報イベントをオフにする。**アプリケーションの受信しているイベントの測地精度が望んでいるレベルでない場合、受信しているイベントの測地精度を調べて、時間の経過とともに向上しているか、だいたい同じレベルを維持しているのかを確認します。精度が向上していなければ、望んでいる測地精度が単純にその時点では得られていないということが考えられます。位置情報サービスをいったんオフにしておき、後で再度試すようにすれば、アプリケーションによる電力浪費を防ぐことができます。

方向関連のイベントの取得

Core Locationは、方向関連の情報を取得する方法を2とおりサポートしています。

- 磁力センサー付きのデバイスは、デバイスが指している方角（**ヘディング**ともいいます）を報告することができます。
- GPSハードウェア付きのデバイスは、デバイスが進行している方向（**コース**ともいいます）を報告することができます。

ヘディングとコースの情報は、同じ情報を示すものではないという点に留意してください。デバイスのヘディングは、真北または磁北を基準とするデバイスの実際の向きを示します。デバイスのコースが表すのは移動方向であり、デバイスの向きは考慮されません。アプリケーションに応じて、どちらか都合の良いほうを使うことも、両方を組み合わせて使うこともできます。たとえば、道案内アプリケーションであれば、ユーザの現在の速度に応じてコース情報とヘディング情報を切り替えることができます。歩行速度では、現在の環境に対してユーザを正しい方角に向けるためにヘディング情報が有効ですが、車の場合、コース情報により車が移動するおおよその方向が示されます。

方向関連のイベントのための要件の追加

アプリケーションを正しく機能させるために、何らかの方向関連情報を必要とする場合は、アプリケーションの`Info.plist`ファイルに`UIRequiredDeviceCapabilities`キーを追加する必要があります。このキーには、アプリケーションがiOSベースデバイスに対して要求する機能を示す文字列の配列が含まれます。App Storeはこの情報を利用して、ハードウェアの最小要件を満たしていないデバイスにユーザがアプリケーションをインストールしてしまうのを防ぎます。

方向関連のイベントに関しては、このキーに関連付けることのできる文字列が2つあります。

- `magnetometer`—アプリケーションでヘディング情報を必要とする場合はこの文字列を指定します。
- `gps`—アプリケーションでコース関連の情報を必要とする場合はこの文字列を指定します。

> **重要：** アプリケーションでヘディングまたはコースのイベントを使用するけれども、それらがなくてもアプリケーションが正常に動作する場合は、対応する文字列を`UIRequiredDeviceCapabilities`キーに指定しないでください。

また、どちらの場合も文字列`location-services`も配列に含めるべきです。`UIRequiredDeviceCapabilities`キーの詳細については、『*Information Property List Key Reference*』を参照してください。

方角（ヘディング）関連イベントの取得

ヘディングイベントは、磁力センサーを備えたデバイス上で実行するアプリケーションで利用できます。磁力センサーは地球から発生する近辺の磁場を測定し、デバイスの正確な向きを識別します。磁力センサーは局所的な磁場（オーディオスピーカー、モーター、その他の電子機器に含まれる固定磁石から発生する磁気など）の影響を受ける可能性がありますが、Core Locationフレームワークはデバイスとともに移動する磁場を除外するように十分に高性能にできています。

ヘディング値は、磁北または地図上の真北のどちらを基準にしても報告させることができます。磁北は、地球の表面上で、地球の磁場が発生する地点を表します。この位置は、真北を表す北極点と同じではありません。デバイスの位置にもよりますが、磁北は多くの用途に十分に適しています。ただし北極点に近づくほどこの値の有用性は低くなります。

ヘディングイベントを受信する手順は次のとおりです。

1. `CLLocationManager`オブジェクトを作成します。
2. `headingAvailable`クラスメソッドを呼び出すことで、ヘディングイベントが利用可能かを確認する（iPhone OS 3.xおよびそれ以前では、代わりに`headingAvailable`プロパティの値を確認してください）。
3. 位置情報マネージャオブジェクトにデリゲートオブジェクトを割り当てます。
4. 真北を基準にした値が必要な場合は、位置情報サービスを開始します。
5. `startUpdatingHeading`メソッドを呼び出してヘディングイベントの送信を開始します。

リスト2-1に、位置情報マネージャを設定し、ヘディングイベントの送信を開始するカスタムメソッドを示します。この場合、オブジェクトは現在の方角（ヘディング）をユーザに表示するView Controllerです。View Controllerは真北を基準にしたヘディング値を表示するため、ヘディングの更新に加えて位置情報の更新も開始します。このコードはiOS 4.0以降で動作します。

リスト 2-1　ヘディングイベントの送信を開始する

```
- (void)startHeadingEvents {
   if (!self.locManager) {
      CLLocationManager* theManager = [[[CLLocationManager alloc] init]
autorelease];

      // プロパティにオブジェクトを保持する。
      self.locManager = theManager;
      locManager.delegate = self;
   }

   // 真北を基準にしたヘディングを取得する位置情報サービスを開始する
   locManager.distanceFilter = 1000;
   locManager.desiredAccuracy = kCLLocationAccuracyKilometer;
   [locManager startUpdatingLocation];

   // ヘディングの更新を開始する
   if ([CLLocationManager headingAvailable]) {
      locManager.headingFilter = 5;
      [locManager startUpdatingHeading];
   }
```

```
}
```

デリゲートプロパティに割り当てるオブジェクトは、CLLocationManagerDelegateプロトコルに従わなければなりません。新しいヘディングイベントが届いたら、ロケーションマネージャオブジェクトはlocationManager:didUpdateHeading:メソッドを呼び出してそのイベントをアプリケーションに送信します。新しいイベントを受信するときは、リスト2-2に示すように、headingAccuracyプロパティをチェックして、今受信したデータが有効であることを確認する必要があります。さらに、真北を基準にしたヘディング値を使用する場合は、使用前に有効な値が含まれていることを確認する必要もあります。

リスト 2-2　ヘディングイベントを処理する

```
- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLHeading
 *)newHeading {
   if (newHeading.headingAccuracy < 0)
      return;

   // 有効な場合は真北を基準にしたヘディングを使用する
   CLLocationDirection  theHeading = ((newHeading.trueHeading > 0) ?
            newHeading.trueHeading : newHeading.magneticHeading);

   self.currentHeading = theHeading;
   [self updateHeadingDisplays];
}
```

ユーザの移動中にコース情報を取得する

GPSハードウェアを備えたデバイスは、デバイスの現在のコースと速度を示す情報を生成することができます。コース情報は、デバイスの移動方向を示すのに使用され、必ずしもデバイスそのものの向きを反映しません。そのため、ユーザが移動している最中にナビゲーション情報を提供するアプリケーションを主な対象としています。

実際のコースと速度の情報は、ユーザの位置の取得に使用するのと同じCLLocationオブジェクトを使用してアプリケーションに返されます。位置情報の更新を開始すると、Core Locationは、入手可能な場合はコースと速度の情報を自動的に提供します。Core Locationフレームワークは、受信した位置情報を使用して現在の移動方向を計算します。位置情報の更新を開始する方法の詳細については、「ユーザの位置の取得」（11 ページ）を参照してください。

位置データのジオコーディング

位置データは通常、地球上のある地点に対応する緯度と経度を表す一組の数値として返されます。このような座標値は、コード内で位置データを正確かつ簡単に指定する手段となりますが、ユーザにとっては直感的ではありません。ユーザには、グローバルな座標ではなく、番地、市区町村、州、県、国のように親しみのある情報を用いて示された位置のほうが理解しやすいでしょう。ユーザに親しみやすい方法で位置情報を表示する必要がある状況では、Geocoderオブジェクトを使用してその情報を取得することができます。

Geocoderオブジェクトについて

Geocoderオブジェクトは、緯度／経度と、ユーザに親しみのある**プレースマーク**、すなわち番地、市区町村、州、県、国などの一連のデータとの間で変換を行うネットワークサービスを使用します。**逆ジオコーディング**とは、緯度と経度をプレースマークに変換する処理です。**正ジオコーディング**とは、プレース名情報を緯度と経度の値に変換する処理です。逆ジオコーディングはどの版のiOSでも使えますが、正ジオコーディングはiOS 5.0以降でしか使えません。

Geocoderはネットワークサービスに依存するため、ジオコーディング要求が成功するためには有効なネットワーク接続が必要です。デバイスが機内モードにある場合や、ネットワークが現在設定されていない場合、Geocoderは必要なサービスに接続できないので、結果として適切なエラーを返す必要があります。ジオコーディング要求の作成時に適用すべき経験則をいくつか示します。

- 送信する逆ジオコーディング要求は、1つのユーザアクションに対して多くとも1つ程度にする。
- ユーザが、同じ場所のジオコーディングを含む複数のアクションを実行した場合は、アクションごとに個別の要求を開始するのではなく、最初のジオコーディング要求から返された結果を再利用する。
- 位置情報を自動的に更新する必要がある場合（ユーザが移動中のときなど）は、ユーザの位置が大幅に変わったときと、適度な時間が経過したあとにのみジオコーディング要求を再発行する。たとえば、一般的な状況では、ジオコーディング要求を1分間に2回以上送信するのは避けるべきです。
- ユーザが結果をすぐに確認しないときには、ジオコーディング要求を開始しない。たとえば、アプリケーションがバックグラウンドで動作しているときや、割り込みを受けて現在非アクティブ状態であるときは、要求を開始しないでください。

座標をプレース名情報に変換

注： iOS 5.0では、MKReverseGeocoderとMKReverseGeocoderDelegateは非推奨になっているので、新たにアプリケーションを開発する場合はCLGeocoderを使ってください。

iOSでは、逆ジオコーディングの処理に、CLGeocoderクラスやMKReverseGeocoderクラスが使えます。このうちでもCLGeocoderの方を推奨します。これはiOS 5.0以降で利用できます。しかし、旧版のiOSでも動作するようにしたい場合は、MKReverseGeocoderクラスを使ってください。

CLGeocoderでプレースマーク情報を取得

CLGeocoderクラスを使って逆ジオコーディング要求を開始するには、このクラスのインスタンスを作成し、reverseGeocodeLocation:completionHandler:メソッドまたはreverseGeocodeLocation:heading:completionHandler:メソッドを呼び出します。ジオコーダオブジェクトは逆ジオコーディング要求を非同期に開始し、結果を提供されたブロックオブジェクトに送ります。ブロックオブジェクトは、要求に対する結果が成功でも失敗でも実行されます。失敗であれば、その理由を示すエラーオブジェクトがブロックに渡されます。

注： 同じCLGeocoderオブジェクトを使って、ジオコーディング要求をいくつでも開始できますが、同じジオコーダに関して同時にアクティブになる要求は1つだけです。

「座標をプレース名情報に変換」に、地図上のある点を逆ジオコードする方法の例を示します。ジオコーディングに特有のコードは最初の数行だけで、ここでは必要に応じてジオコーダオブジェクトを割り当て、reverseGeocodeLocation:completionHandler:メソッドを呼び出して、逆ジオコーディング処理を起動しています（geocoder変数はジオコーダオブジェクトを格納するメンバ変数）。残りのコードはサンプルアプリケーション自身に特有のものです。この場合、サンプルアプリケーションはプレースマークを、独自の注釈オブジェクト（MapLocationクラスで定義される）とともに格納し、対応する注釈ビューの吹き出しにボタンを追加します。

リスト 3-1　CLGeocoderを用いた位置のジオコーディング

```
@implementation MyGeocoderViewController (CustomGeocodingAdditions)
- (void)geocodeLocation:(CLLocation*)location
forAnnotation:(MapLocation*)annotation
{
    if (!geocoder)
        geocoder = [[CLGeocoder alloc] init];

    [theGeocoder reverseGeocodeLocation:location completionHandler:
        ^(NSArray* placemarks, NSError* error){
            if ([placemarks count] > 0)
            {
                annotation.placemark = [placemarks objectAtIndex:0];

                // 注釈のビューに「More Info（詳細情報）」ボタンを追加する
                MKPinAnnotationView*  view = (MKPinAnnotationView*)[map
viewForAnnotation:annotation];
                if (view && (view.rightCalloutAccessoryView == nil))
                {
```

```
                    view.canShowCallout = YES;
                    view.rightCalloutAccessoryView = [UIButton
buttonWithType:UIButtonTypeDetailDisclosure];
                }
            }
    }];
}
@end
```

サンプルではこのようにブロックオブジェクトを使っているため、（注釈オブジェクトなどの）情報に容易にアクセスし、完了ハンドラの一部として使えるという利点があります。ブロックがなかったとすれば、データ変数を獲得するプロセスは、より複雑になっていたはずです。

逆ジオコーダからプレースマーク情報を取得する

iOS 4.1以前で動作するアプリケーションでは、逆ジオコーディング要求はMap Kitフレームワークの`MKReverseGeocoder`クラスを使って実行しなければなりません。このクラスは、1箇所のジオコーディングを行うのにデリゲートを用いる方式を使用します。つまり、`MKReverseGeocoder`クラスの1つのインスタンスを1回のみ使用できるということです。また、Googleの利用規約により、`MKReverseGeocoder`クラスをGoogleマップと組み合わせて使用する必要があります。

逆ジオコーディング要求を開始するには、`MKReverseGeocoder`クラスのインスタンスを作成し、適切なオブジェクトを`delegate`プロパティに割り当ててから、`start`メソッドを呼び出します。クエリが正常に終了すると、デリゲートの`reverseGeocoder:didFindPlacemark:`メソッドが呼び出されて、結果を含んだ`MKPlacemark`オブジェクトが渡されます。位置の逆ジオコーディングに問題が生じると、代わりに`reverseGeocoder:didFailWithError:`メソッドが呼び出されます。

リスト 3-2に、逆ジオコーディングの使用に必要なコードを示します。ジオコーディング処理が正常に完了すると、コードはプレースマーク情報を表示するためのボタンを注釈ビュー（Annotation View）の吹き出しに追加します。注釈は、自動的にデリゲートで利用できるわけではないため、カスタムの`annotationForCoordinate:`メソッドを使用してマップビューから適切な注釈オブジェクトを探しています。

リスト 3-2　`MKReverseGeocoder`を用いた位置のジオコーディング

```
@implementation MyGeocoderViewController (CustomGeocodingAdditions)
- (void)geocodeLocation:(CLLocation*)location
forAnnotation:(MapLocation*)annotation
{
    MKReverseGeocoder* theGeocoder = [[MKReverseGeocoder alloc]
initWithCoordinate:location.coordinate];

    theGeocoder.delegate = self;
    [theGeocoder start];
}

// デリゲートのメソッド群
- (void)reverseGeocoder:(MKReverseGeocoder*)geocoder
didFindPlacemark:(MKPlacemark*)place
{
    MapLocation*    theAnnotation = [map
annotationForCoordinate:place.coordinate];
    if (!theAnnotation)
        return;
```

```
    // プレースマークと注釈を関連付ける
     theAnnotation.placemark = place;

     // 注釈のビューに「More Info（詳細情報）」ボタンを追加する
     MKPinAnnotationView*  view = (MKPinAnnotationView*)[map
viewForAnnotation:annotation];
     if (view && (view.rightCalloutAccessoryView == nil))
     {
         view.canShowCallout = YES;
         view.rightCalloutAccessoryView = [UIButton
buttonWithType:UIButtonTypeDetailDisclosure];
     }
}

- (void)reverseGeocoder:(MKReverseGeocoder*)geocoder
didFailWithError:(NSError*)error
{
     NSLog(@"Could not retrieve the specified place information.\n");
}
@end

@implementation MKMapView (GeocoderAdditions)

- (MapLocation*)annotationForCoordinate:(CLLocationCoordinate2D)coord
{
     // マップビューの座標のリスト全体を反復処理し、
     // 指定された値と座標が正確に一致した
     // 最初の1つを返す
     id<MKAnnotation> theObj = nil;

     for (id obj in [self annotations])
     {
         if (([obj isKindOfClass:[MapLocation class]]))
         {
             MapLocation* anObj = (MapLocation*)obj;

             if ((anObj.coordinate.latitude == coord.latitude) &&
                 (anObj.coordinate.longitude == coord.longitude))
             {
                 theObj = anObj;
                 break;
             }
         }
     }

     return theObj;
}
@end
```

プレース名を座標に変換する

iOS 5.0以降、`CLGeocoder`クラスを使って、正ジオコーディング要求ができるようになりました。Address Book情報の辞書、または単なる文字列を使います。文字列ベースの要求に、専用の書式はありません。区切り文字は、あれば分かりやすくなりますが、なくても構いません。ジオコーダサーバは文字列を、大文字と小文字の区別がないものとして扱います。したがって、以下のどの文字列でも、何らかの結果が得られます。

- 「Apple Inc」
- 「1 Infinite Loop」
- 「1 Infinite Loop, Cupertino, CA USA」

正ジオコーダに与える情報が多いほど、よい結果が得られます。ジオコーダオブジェクトは、与えられた情報をパースし、合致が見つかれば、プレースマークオブジェクトをいくつか返します。返されるプレースマークオブジェクトの個数は、与えられた情報の具体性に大きく依存します。したがって、通り、市、県、国の情報が揃っていれば、通りと市の情報しかない場合に比べ、単一の住所を返す可能性が高くなります。ジオコーダに渡す完了ハンドラブロックは、したがって、次の例のように、複数のプレースマークが返された場合にも対処できるようでなければなりません。

```
[geocoder geocodeAddressString:@"1 Infinite Loop"
     completionHandler:^(NSArray* placemarks, NSError* error){
         for (CLPlacemark* aPlacemark in placemarks)
         {
             // プレースマークを処理
         }
}];
```

地図の表示

iOS 3.0で導入されたMap Kitフレームワークを利用すると、完全な機能を備えた地図インターフェイスをアプリケーションウインドウに埋め込むことができます。このフレームワークが提供する地図サポートには、「マップ(Maps)」アプリケーションに一般的に見られる多くの機能が含まれています。標準的なストリートレベルの地図情報、航空写真、またはこれら2つの組み合わせを表示できます。プログラムによって地図を拡大縮小したり、パンすることもできます。またこのフレームワークは、ユーザによる地図の拡大縮小やパンを可能にするタッチイベントを自動的にサポートします。地図にはカスタム情報を注釈として付けることもできます。

Map Kitフレームワークの機能を使用するには、Xcodeプロジェクトでアプリケーションを`MapKit.framework`にリンクする必要があります。このフレームワークのクラスとヘッダにアクセスするには、関連するソースファイルの先頭に`#import <MapKit/MapKit.h>`ステートメントを追加します。Map Kitフレームワークのクラスの全般的な情報については、『*Map Kit Framework Reference*』を参照してください。

重要： Map KitフレームワークはGoogleのサービスを使用して地図データを提供します。このフレームワークとそれに関連するインターフェイスを使用する場合は、Google Maps/Google Earth APIの利用規約に従わなければなりません。これらの利用規約は http://code.google.com/apis/maps/iphone/terms.html で参照できます。

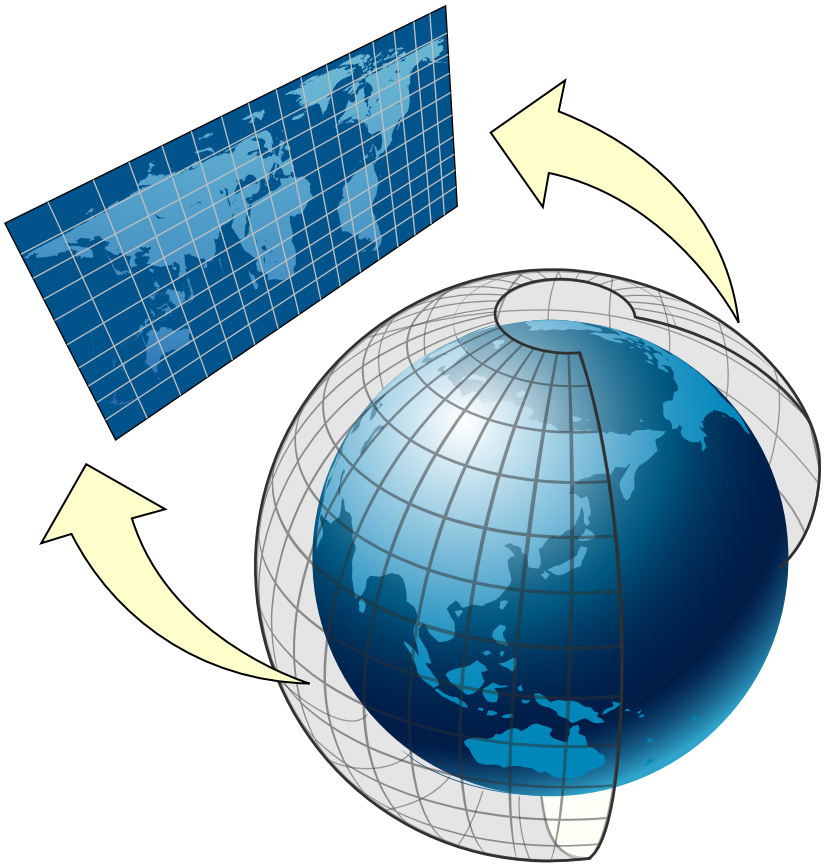
地図ジオメトリの理解

Map Viewには、球状のオブジェクト、すなわち地球を平面化した表現が含まれています。地図を効果的に使用するには、Map View上で点を指定する方法と、その点を地球の表面上の点に変換する方法をある程度理解しておく必要があります。地図の座標系を理解することは、オーバーレイのように地図上にカスタムコンテンツを重ねて配置する予定がある場合は特に重要です。

地図の座標系

Map Kitで使用される座標系を理解するには、地球の3次元の表面が2次元の地図にマッピングされる仕組みを理解すると役に立ちます。図 4-1は、地球の表面がどのように2次元平面にマッピングされるかを示します。

図 4-1　球状データから平面へのマッピング



Map Kitではメルカトル図法、すなわち図 4-1 （30 ページ）に示したような、特殊なタイプの円筒図法を使用しています。円筒図法では、球の座標が円筒の面にマッピングされ、その円筒が展開されて平面地図になります。このような投影図では、本来は極に収れんする経度線が、代わりに平行に並ぶため、赤道から離れるほど大陸は変形します。メルカトル図法の利点は、一般的なナビゲーションに有効な方法で地図の内容が拡大縮小される点です。具体的には、メルカトル図法では、地図上の任意の2点間に引いた直線から、地球の表面上での実際のナビゲーションで使用することのできる進路方向を求めることができます。Map Kitで用いられている投影法では、グリニッジ子午線を中心子午線として使用します。

地図上でデータ点を指定する方法は、そのデータ点の用途により異なります。Map Kitは、地図上のデータ点を指定するために基本的な3つの座標系をサポートしています。

- **地図座標**は、地球の球状表現上の緯度と経度です。地図座標は、地球上の場所を指定する基本的な方法です。個々の地図座標の値は、`CLLocationCoordinate2D`構造体を使用して指定します。領域を指定するには、`MKCoordinateSpan`構造体および`MKCoordinateRegion`構造体を使用します。

- **地図点**は、メルカトル図法の地図上のxおよびy値です。地図点は、必要な数学計算が平易であることから、多くの地図関連の計算において地図座標の代わりに使用されます。アプリケーションでは、地図上のカスタムオーバーレイの形状と位置を指定する際に、主に地図点を使用します。個々の地図点は、`MKMapPoint`構造体を使用して指定します。領域を指定するには、`MKMapSize`構造体および`MKMapRect`構造体を使用します。

- **点**は、UIViewオブジェクトの座標系に関連付けられた図形単位です。地図点と地図座標は、ビューにカスタムコンテンツを描画する前に点にマッピングされなければなりません。個々の点は、CGPoint構造体を使用して指定します。領域を指定するには、CGSize構造体およびCGRect構造体を使用します。

ほとんどの状況では、使用すべき座標系は、使用しているMap Kitインターフェイスによってあらかじめ決められます。実際のデータをファイルまたはアプリケーション内に保存することに関して言えば、位置データの保存には地図座標が正確かつポータブルであり最も適しています。また、Core Locationも位置の値を指定する際に地図座標を使用します。

座標系の変換

通常、地図上の点は緯度と経度の値を使用して指定しますが、異なる座標系との間で変換が必要な場合もあります。たとえば、通常はオーバーレイの形状を指定する際に地図点を使用します。表4-1に、異なる座標系への変換に使用する変換ルーチンの一覧を示します。これらの変換処理のほとんどは、点への変換または点からの変換を伴うため、ビューオブジェクトを必要とします。

表 4-1　地図座標系変換ルーチン

変換元	変換先	変換ルーチン
地図座標	点	convertCoordinate: toPointToView: (MKMapView) convertRegion: toRectToView: (MKMapView)
地図座標	地図点	MKMapPointForCoordinate
地図点	地図座標	MKCoordinateForMapPoint MKCoordinateRegionForMapRect
地図点	点	pointForMapPoint: (MKOverlayView) rectForMapRect: (MKOverlayView)
点	地図座標	convertPoint: toCoordinateFromView: (MKMapView) convertRect: toRegionFromView: (MKMapView)
点	地図点	mapPointForPoint: (MKOverlayView) mapRectForRect: (MKOverlayView)

ユーザインターフェイスへのMap Viewの追加

MKMapViewクラスは、アプリケーションの中で地図データを表現する自己完結型のインターフェイスです。このクラスは、地図データの表示、ユーザとの対話操作の管理、アプリケーションによって提供されたカスタムコンテンツのホスティングに対応しています。MKMapViewをサブクラス化することはできず、そのままアプリケーションのビュー階層に埋め込まなければなりません。また、地図にデリゲートオブジェクトを割り当てる必要もあります。Map Viewは、適切に応答する機会を得られるように、関連するすべての対話操作をデリゲートに報告します。

Map Viewは、プログラミングによって、あるいはInterface Builderを使用してアプリケーションに追加できます。

- Interface Builderを使用して地図を追加するには、Map Viewオブジェクトを該当するビューまたはウインドウにドラッグします。
- プログラミングによって地図を追加するには、`MKMapView`クラスのインスタンスを作成し、`initWithFrame:`メソッドでこれを初期化してからサブビューとしてビュー階層へ追加します。

Map Viewはビューであるため、ほかのビューを操作するときと同じ方法でこれを操作できます。たとえば、ビュー階層内でのサイズと位置の変更、自動サイズ変更動作の設定、サブビューの追加を行うことができます。ビューとは異なり、Map Viewで直接タッチイベントを処理することはありません。Map View自体は、地図関連のデータの表示、およびそのデータとのすべてのやり取りを処理する複雑なビュー階層のための不透過(opaque)型のコンテナです。Map Viewに追加したサブビューはすべて`frame`プロパティによって指定された位置に固定され、地図コンテンツと一緒にはスクロールしません。コンテンツを特定の地図座標に対して相対的に固定（そして地図と一緒にスクロール）する必要がある場合は、「地図の注釈」（35 ページ）で説明するように、注釈またはオーバーレイを使用する必要があります。

新しい地図は、ユーザによる対話操作を受け付け、地図データのみを表示するように設定されます。衛星画像、または衛星データと地図データの組み合わせを表示するように地図を設定するには、Interface Builderで地図のType属性を変更するか、または`mapType`プロパティの値を変更します。ユーザによる対話操作を制限したい場合は、`zoomEnabled`プロパティと`scrollEnabled`プロパティの値を変更します。ユーザによる対話操作に応答したい場合は、「地図に対するユーザの対話操作への応答」（34 ページ）で説明するようにデリゲートを使用する必要があります。

地図のプロパティの設定

`MKMapView`クラスには、プログラミングによって設定可能なプロパティがいくつかあります。これらのプロパティは、現在地図のどの部分が表示されているかや、ユーザのどのような対話操作が可能かといった重要な情報を制御します。

地図の可視部分の設定

`MKMapView`クラスの`region`プロパティは、地図の現在の可視部分を制御します。地図の可視領域は、初めて作成された時点では、通常は全世界に設定されます。つまり、表示領域は、地図のできる限り広い範囲を表示します。この領域は、`region`プロパティに新しい値を割り当てることでいつでも変更できます。このプロパティには、次のような定義を持つ`MKCoordinateRegion`構造体が含まれています。

```
typedef struct {
   CLLocationCoordinate2D center;
   MKCoordinateSpan span;
} MKCoordinateRegion;
```

`MKCoordinateRegion`構造体の注目すべき部分は、スパン(span)です。スパンは矩形の幅と高さの値に似ていますが、地図座標で指定されるため、単位は度、分、秒です。緯度1度は約111キロメートルに相当しますが、経度の長さは緯度によって変わります。赤道では、経度1度は約111キロメート

ルに相当しますが、極ではゼロになります。スパンをメートル単位で指定したい場合は、MKCoordinateRegionMakeWithDistanceを使用して、度数ではなくメートル値で領域データ構造体を作成します。

regionプロパティに割り当てる値（またはsetRegion:animated:メソッドで設定する値）は通常、このプロパティによって最終的に保存される値と同じではありません。領域のスパンを設定すると、表示したい矩形が名目的に定義されるだけでなく、Map View自体の拡大縮小レベルも暗黙的に設定されます。Map Viewは、任意の拡大縮小レベルを表示することができないため、指定された領域を、Map Viewがサポートする拡大縮小レベルに合うように調節しなければなりません。Map Viewは、できる限り画面いっぱいに表示しながら、指定した可視全体を表示できる拡大縮小レベルを選択します。その後、それに応じてregionプロパティを調節します。regionプロパティの値を実際には変更せずに結果の領域を確認するには、Map ViewのregionThatFits:メソッドを使用できます。

地図コンテンツの拡大縮小とパン

拡大縮小とパンによって、地図の可視部分をいつでも変更することができます。

- 地図を（拡大縮小レベルを維持したまま）パンするには、Map ViewのcenterCoordinateプロパティの値を変更するか、またはsetCenterCoordinate:animated:メソッドを呼び出します。
- 拡大縮小レベルを変更する（さらに必要に応じて地図をパンする）には、Map Viewのregionプロパティの値を変更するか、またはsetRegion:animated:メソッドを呼び出します。

地図のパンだけを行う場合は、centerCoordinateプロパティを変更する方法でのみ行ってください。regionプロパティを変更することで地図をパンしようとすると、通常は拡大縮小レベルも変わります。これは、領域の一部が変更されると、Map Viewはその領域を適切に表示するために必要な拡大縮小レベルを評価するためです。現在の緯度を変更すると、ほとんどの場合拡大縮小レベルが変更され、さらにその他の変更によって異なる拡大縮小レベルが選択されることもあります。centerCoordinateプロパティ（またはsetCenterCoordinate:animated:メソッド）を使用すると、Map Viewは拡大縮小レベルを変更せずに、必要に応じてスパンを更新します。たとえば、現在の地図の幅の半分だけ地図を左にパンするには、次のようなコードを使用して地図の左端の座標を検出し、それを新しい中心点として使用します。

```
CLLocationCoordinate2D mapCenter = myMapView.centerCoordinate;
mapCenter = [myMapView convertPoint:
               CGPointMake(1, (myMapView.frame.size.height/2.0))
               toCoordinateFromView:myMapView];
[myMapView setCenterCoordinate:mapCenter animated:YES];
```

地図を拡大縮小するには、表示されている地図の領域のスパンを変更します。拡大するには、スパンに小さな値を割り当てます。縮小するには、大きな値を割り当てます。したがって現在のスパンが1度の場合、スパンを2度にすると2分の1に縮小されます。

```
MKCoordinateRegion theRegion = myMapView.region;

// 縮小する
theRegion.span.longitudeDelta *= 2.0;
theRegion.span.latitudeDelta *= 2.0;
[myMapView setRegion:theRegion animated:YES];
```

地図上でのユーザの現在位置の表示

Map Kitには、ユーザの現在位置を地図上に表示する機能が組み込まれています。この位置を表示するには、Map Viewオブジェクトの`showsUserLocation`プロパティを`YES`に設定します。これによって、Map ViewはCore Locationを使用してユーザの現在位置を検出し、`MKUserLocation`型の注釈を地図に追加します。

`MKUserLocation`注釈オブジェクトが地図に追加されると、カスタム注釈が追加されたときと同様にデリゲートによってそのことが報告されます。カスタム注釈ビューをユーザの位置に関連付けたい場合は、デリゲートオブジェクトの`mapView:viewForAnnotation:`メソッドからそのビューを返す必要があります。デフォルトの注釈ビューを使用する場合は、このメソッドからは`nil`を返す必要があります。

地図に対するユーザの対話操作への応答

`MKMapView`クラスは、地図関連の重要なイベントを対応するデリゲートオブジェクトに報告します。デリゲートオブジェクトは、`MKMapViewDelegate`プロトコルに準拠したオブジェクトです。次の種類のイベントに応答する必要がある状況で、このオブジェクトを使用できます。

- 地図の可視領域の変更
- ネットワークからの地図タイルの読み込み
- ユーザの位置の変更
- 注釈とオーバーレイに関連する変更

注釈とオーバーレイに関連する変更への対処についての詳細は、「地図の注釈」（35 ページ）を参照してください。

地図の注釈

MKMapViewクラスは、スクロール可能な地図を表示するための不透過のビュー階層を実装します。地図自体はスクロール可能ですが、Map Viewに追加するサブビューはすべて位置が固定され、スクロールしません。地図に固定のコンテンツを追加して地図と一緒にスクロールさせたい場合は、注釈とオーバーレイを作成しなければなりません。

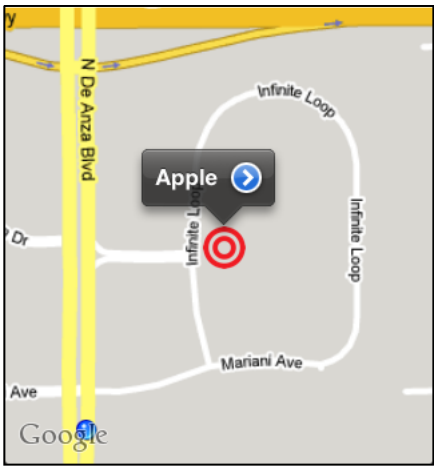
注釈は、単一の座標点で定義できるコンテンツの表示に使用されます。これに対して**オーバーレイ**は、任意の数の点で定義されるコンテンツの表示に使用され、連続または非連続の1つ以上の図形を構成できます。たとえば、注釈を使用してユーザの現在位置、特定の住所、関心のある場所などの情報を表示します。オーバーレイは、交通情報のような複雑な情報や、公園、湖、市区町村、州、県、国などの境界線のある領域の表示に使用します。

Map Kitは、注釈やオーバーレイに関連付けられているデータと、地図上でのその可視表現を切り離しています。切り離すことで、地図は、表示される注釈とオーバーレイを非常に効率良く管理できます。つまり何百もの注釈とオーバーレイを地図に追加しても、適切なパフォーマンスを期待できるということです。

地図への注釈の追加

注釈は、地図上で特定の座標位置を目立たせ、その位置に関する追加情報を提供する手段です。注釈を使用して、特定の住所、関心のある場所、その他さまざまな目的地を吹き出しで示すことができます。地図上に表示されるときには、注釈には通常、その位置を示す何らかの画像が付き、情報やより詳しい内容を表示するリンクを含んだ注釈用吹き出しが付くこともあります。図 5-1に、特定の場所を目立たせるためのカスタム画像を使用した注釈を示します。

図 5-1 地図での注釈の表示



地図に注釈を表示するには、アプリケーションは2つのオブジェクトを指定する必要があります。

- MKAnnotationプロトコルに準拠し、注釈のデータを管理するオブジェクト（**注釈オブジェクト**）。
- 地図の表面に注釈の可視表現を描画する（MKAnnotationViewクラスから派生した）ビュー（**注釈ビュー**）。

注釈オブジェクトは通常、地図座標と注釈に関連するその他の情報（タイトル文字列など）が格納された比較的小さいデータオブジェクトです。注釈はプロトコルを使用して定義されるため、アプリケーション内のどのクラスも注釈オブジェクトになり得ます。実際には、大量の注釈を地図に追加する予定の場合には特に、注釈オブジェクトは軽量に保つことが推奨されます。Map Viewは、追加された注釈オブジェクトの参照を保持しており、注釈オブジェクト内のデータを使用して、対応する注釈ビューをいつ表示すべきかを判断します。

Map Kitは標準的な注釈ビューをいくつか提供しています。また必要であればカスタムの注釈ビューを定義することもできます。ただし、注釈ビューを地図の表面に直接追加することはできません。代わりに、要請されたら注釈ビューを提供し、Map Viewがそのビューを不透過のビュー階層に組み入れるようにします。オーバーレイビューを提供するにはMap Viewデリゲートオブジェクトを使用します。

作成する注釈は通常、不変の単一の地図座標位置に固定されます。しかし、必要であれば注釈の座標位置をプログラミングによって変更して、ユーザが地図上で注釈をドラッグできるように実装することができます。iOS 4.0以降ではさらに、注釈のドラッグのサポートがMap Kitクラスに組み込まれています。以前のバージョンでこのサポートを実装するには、デベロッパ側でカスタムコードを追加する必要があります。

地図に注釈を追加するためのチェックリスト

以下に、地図を利用するアプリケーションで注釈を実装して使用するための手順を示します。これらの手順は、アプリケーションのインターフェイスのどこかにMKMapViewオブジェクトが組み込まれていることを前提としています。

1. 次のいずれかの方法で、適切な注釈オブジェクトを定義します。
   - MKPointAnnotationクラスを使用して、シンプルな注釈を実装する。このタイプの注釈は、画面上の注釈用吹き出しに表示されるタイトルとサブタイトルの文字列を指定するプロパティを格納します。
   - 「カスタム注釈オブジェクトの定義」（37 ページ）で説明する、MKAnnotationプロトコルに準拠したカスタムオブジェクトを定義する。このタイプの注釈は、任意の種類のデータを格納できます。
2. 画面上にデータを表示するための注釈ビューを定義します。注釈ビューをどのように定義するかは、アプリケーションのニーズに応じて次のいずれかが考えられます。
   - 注釈を静止画像で表現できる場合は、MKAnnotationViewクラスのインスタンスを作成し、そのimageプロパティに対象画像を割り当てる（「標準注釈ビューの使用」（38 ページ）参照）
   - 標準的なピン留めの注釈を使用する場合は、MKPinAnnotationViewクラスのインスタンスを作成する（「標準注釈ビューの使用」（38 ページ）参照）

- 静止画像だけでは注釈を十分に表現できない場合は、MKAnnotationViewをサブクラス化し、これを表現するために必要なカスタム描画コードを実装する。カスタム注釈ビューを実装する方法については、「カスタム注釈ビューの定義」（39 ページ）を参照。

3. Map Viewデリゲートに`mapView:viewForAnnotation:`メソッドを実装します。

   このメソッドの実装では、既存の注釈ビューがあればそれをキューから取り出し、なければ新規の注釈ビューを作成します。アプリケーションで複数種類の注釈をサポートする場合は、提供する注釈オブジェクトに適したタイプのビューを作成するロジックをメソッドに追加します。このメソッドの実装方法の詳細については、「デリゲートオブジェクトからの注釈ビューの作成」（40 ページ）を参照してください。

4. `addAnnotation:`メソッドまたは`addAnnotations:`メソッドを使用して、Map Viewに注釈オブジェクトを追加します。

Map Viewに注釈を追加すると、Map Viewは、表示された地図矩形内にその注釈の座標があれば、対応する注釈ビューが表示されます。特定の注釈を非表示にするには、デベロッパ自身がMap Viewから注釈を手動で削除する必要があります。注釈の追加と削除はいつでも行うことができます。

注釈はすべて、地図の現在の拡大縮小レベルに関係なく、毎回同じ縮尺で描画されます。そのため、地図に多数の注釈が含まれていると、ユーザが地図を縮小したときに注釈ビューが重なり合う可能性があります。この動作に対処するために、地図の現在の拡大縮小レベルに応じて注釈の追加と削除を行うことができます。たとえば、天気情報アプリケーションであれば、地図が州や県全体を表示するように縮小されたときには、主要都市の情報のみ表示するなどが考えられます。ユーザが表示を拡大したときには、アプリケーションは、より小さな市や地域の天気情報を示す新しい注釈を追加することができます。注釈の追加と削除に必要なロジックを実装するのは、デベロッパの責任です。

Map Viewの注釈の効果的な管理方法の詳細については、「地図の注釈オブジェクトの管理」（41 ページ）を参照してください。

カスタム注釈オブジェクトの定義

注釈のうち最も重要な部分は、MKAnnotationプロトコルに準拠したオブジェクトである、注釈オブジェクトです。タイトルと地図座標を関連付けるだけであれば、MKPointAnnotationクラスを注釈オブジェクトに使用できます。しかし、その他の情報を注釈に表示する必要がある場合は、カスタムの注釈オブジェクトを定義する必要があります。

カスタムの注釈オブジェクトは、地図座標および注釈に関連付ける必要のあるその他のデータから成ります。リスト 5-1に、カスタム注釈クラスを宣言するために最小限必要となるコードを示します。coordinateプロパティの宣言は、MKAnnotationプロトコルからのもので、すべての注釈クラスに含める必要があります。これは単純な注釈であるため初期化メソッドも含まれており、これを使用して、読み取り専用のcoordinateプロパティに値を設定しています。アプリケーションの宣言には、さらに追加の注釈データを定義するメソッドとプロパティを含めることになります。

リスト 5-1　シンプルな注釈オブジェクトの作成

```
@@interface MyCustomAnnotation : NSObject <MKAnnotation> {
    CLLocationCoordinate2D coordinate;
}
@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;
- (id)initWithLocation:(CLLocationCoordinate2D)coord;
```

```
// その他のメソッドとプロパティ
@end
```

カスタムクラスの実装では、座標プロパティと、そのプロパティ値の設定方法の実装を提供する必要があります。coordinateは宣言済みプロパティであるため、@synthesizeキーワードを使用して、実装に必要なコードを簡単に合成することができます。あとは、リスト 5-2に示すように、カスタムのinitWithLocation:メソッドのためのコードを実装するだけです。

リスト 5-2　MyCustomAnnotationクラスの実装

```
@implementation MyCustomAnnotation
@synthesize coordinate;

- (id)initWithLocation:(CLLocationCoordinate2D)coord {
    self = [super init];
    if (self) {
        coordinate = coord;
    }
    return self;
}
@end
```

重要： クラスにcoordinateプロパティを実装する際には、その作成を合成することが推奨されます。このプロパティのメソッドをデベロッパ自身で実装する場合、あるいは地図に注釈を追加した後でクラスのほかの部分でそのプロパティの元になる変数を手動で変更する場合は、必ずキー値監視(KVO)通知を送信してください。Map KitはKVO通知を使用して、注釈のプロパティであるcoordinate、title、subtitleへの変更を検出し、必要な変更を地図表示に施します。KVO通知を送信しないと、注釈の位置が地図上で適切に更新されない場合があります。

KVOに準拠したアクセサメソッドを実装する方法については、『*Key-Value Observing Programming Guide*』を参照してください。

Core Dataオブジェクトをベースにした注釈オブジェクトの例については、サンプルプロジェクト『*WeatherMap*』を参照してください、

標準注釈ビューの使用

アプリケーションの地図に注釈を表示する最も簡単な方法は、標準で提供されている注釈ビューの1つを使用することです。MKAnnotationViewクラスは、すべての注釈ビューの基本動作を定義した具象ビューです。MKPinAnnotationViewクラスはMKAnnotationViewのサブクラスで、標準のシステムピン画像の1つを、対応する注釈の座標点に表示します。

MKAnnotationViewクラスは、注釈用に表示する静止画像がある場合に適しています。このクラスのインスタンスを作成したあと、オブジェクトのimageプロパティにカスタム画像を割り当てます。この画像は、注釈が表示されるときに、対象の地図座標を中心において表示されます。地図座標を中心において画像を配置したくない場合は、centerOffsetプロパティを使用して、中心点を縦横の任意の方向に移動できます。リスト 5-3に、カスタム画像とオフセットを指定した注釈ビューの作成方法の例を示します。

リスト 5-3　標準注釈ビューの作成

```
MKAnnotationView* aView = [[[MKAnnotationView alloc] initWithAnnotation:annotation
```

```
                                    reuseIdentifier:@"MyCustomAnnotation"]
autorelease];
aView.image = [UIImage imageNamed:@"myimage.png"];
aView.centerOffset = CGPointMake(10, -20);
```

標準的な注釈ビューは、デリゲートのmapView:viewForAnnotation:メソッドで作成します。このメソッドの実装方法の詳細については、「デリゲートオブジェクトからの注釈ビューの作成」（40 ページ）を参照してください。

カスタム注釈ビューの定義

静止画像だけでは注釈を十分に表現できない場合は、次のどちらかの方法で、MKAnnotationViewをサブクラス化して動的にコンテンツを描画します。この場合も引き続きMKAnnotationViewのimageプロパティを使用しながら画像を一定間隔で差し替えたり、ビューのdrawRect:メソッドをオーバーライドして毎回動的にコンテンツを描画したりすることもできます。ビューで行うほかのカスタム描画の場合と同様に、アプローチを決める前に必ずパフォーマンスを考慮するべきです。カスタム描画は柔軟性は優れていますが、コンテンツのほとんどが一定のままの場合は、画像を使用したほうが速くなる可能性があります。

カスタム注釈ビューを定義することに決めた場合は、ほかのビューと同様にサブクラス化を行います。唯一の違いは、UIViewではなくMKAnnotationViewをサブクラス化する点です。リスト 5-4にこれを示します。

リスト 5-4　カスタム注釈ビューの宣言

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

@interface MyCustomAnnotationView : MKAnnotationView
{
   // カスタムのデータメンバ
}
// カスタムのプロパティとメソッド
@end
```

drawRect:メソッドを使用してコンテンツを描画する場合は、必ず初期化の直後に注釈ビューに対してゼロ以外のフレームサイズを指定しなければなりません。注釈ビューのデフォルトの初期化メソッドは、パラメータとしてフレーム矩形を取りません。代わりに、imageプロパティに指定した画像を使用して、後からフレームサイズを設定します。ただし、画像を設定しない場合は、リスト 5-5に示すように、レンダリングされたコンテンツを表示するためにビューのframeプロパティを明示的に設定しなければなりません。このビューはフレームの一部分に対してのみ描画されるため、不透過処理のプロパティをNOに設定して、残りの地図コンテンツが透けて見えるようにします。このようにしないと、描画システムが現在の背景色でビューを塗りつぶしてからdrawRect:メソッドを呼び出します。

リスト 5-5　カスタム注釈ビューの初期化

```
- (id)initWithAnnotation:(id <MKAnnotation>)annotation reuseIdentifier:(NSString
 *)reuseIdentifier
{
    self = [super initWithAnnotation:annotation reuseIdentifier:reuseIdentifier];
     if (self)
     {
         // フレームサイズを適切な値に設定する
```

```
        CGRect  myFrame = self.frame;
        myFrame.size.width = 40;
        myFrame.size.height = 40;
        self.frame = myFrame;

        // 不透過プロパティのデフォルト値はYESです。これを次のように設定します。
        // NOに設定することで、地図コンテンツが、レンダリング対象外のビューの領域を
        // 透かして見えるようになる。
        self.opaque = NO;
    }
    return self;
}
```

その他すべての点では、注釈ビューでのカスタムコンテンツの描画は、ほかのビューの場合と同じです。システムは必要に応じてビューの`drawRect:`メソッドを呼び出し、ビューのうち再描画が必要な部分を再描画しますが、ビューの`setNeedsDisplay`メソッドまたは`setNeedsDisplayInRect:`メソッドを呼び出していつでも強制的に再描画を行うこともできます。ビューのコンテンツをアニメーション化するには、一定間隔で作動するタイマーを設定してビューを更新する必要があります。

タイマーの設定方法については、『*Timer Programming Topics*』を参照してください。iOSでビューにコンテンツを描画する方法の詳細については、『*View Programming Guide for iOS*』を参照してください。

デリゲートオブジェクトからの注釈ビューの作成

注釈ビューが必要になると、Map Viewはそのデリゲートオブジェクトの`mapView:viewForAnnotation:`メソッドを呼び出します。このメソッドを実装していない場合、または実装していて常に`nil`が返される場合、Map Viewはデフォルトの注釈ビュー（通常はピン注釈ビュー）を使用します。デフォルト以外の注釈ビューが返されるようにするには、このメソッドをオーバーライドして、使用するビューをそこで作成する必要があります。

`mapView:viewForAnnotation:`メソッドで新しいビューを作成する前に、必ず、既存の類似の注釈ビューがないか確認してください。Table Viewと同様、Map Viewには、使用されていない注釈ビューがあればそれをキャッシュに入れるという選択肢があります。そのようにした場合、Map Viewは、使用されていないビューを`dequeueReusableAnnotationViewWithIdentifier:`メソッドを使用して取得できるようにします。このメソッドから`nil`以外の値が返された場合は、ビューの属性を更新してそれを返す必要があります。このメソッドから`nil`が返された場合は、単純に適切な注釈ビュークラスの新しいインスタンスを作成します。どちらの場合も、このメソッドに渡された注釈を受け取り、それを注釈ビューに割り当てるのはデベロッパの責任です。また、デベロッパはこのメソッドでビューを返す前にビューを更新しておく必要があります。

リスト 5-6は、`mapView:viewForAnnotation:`メソッドの実装例を示します。このメソッドは、カスタム注釈オブジェクトにピン注釈を提供します。既存のピン注釈ビューがすでに存在する場合、このメソッドは注釈オブジェクトをそのビューに関連付けます。再利用キューにビューが存在しない場合、このメソッドは新しいビューを作成してそのビューの基本プロパティを設定し、注釈の吹き出し用のアクセサリビューを設定します。地図が現在、ユーザの位置を示していれば、このメソッドはすべての`MKUserLocation`オブジェクトについて`nil`を返し、デフォルトの注釈ビューが使用されるようにします。

リスト 5-6　注釈ビューの作成

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
```

```
                    viewForAnnotation:(id <MKAnnotation>)annotation
{
    // これがユーザの位置の場合は、単にnilを返す
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;

    // カスタム注釈を処理する
    if ([annotation isKindOfClass:[MyCustomAnnotation class]])
    {
        // まず、既存のピン注釈ビューをキューから取り出すことを試みる
        MKPinAnnotationView*    pinView = (MKPinAnnotationView*)[mapView
       dequeueReusableAnnotationViewWithIdentifier:@"CustomPinAnnotationView"];

        if (!pinView)
        {
            // 既存のピン注釈ビューが利用できない場合は、新しいビューを作成する
          pinView = [[[MKPinAnnotationView alloc] initWithAnnotation:annotation
                        reuseIdentifier:@"CustomPinAnnotation"]
                               autorelease];
            pinView.pinColor = MKPinAnnotationColorRed;
            pinView.animatesDrop = YES;
            pinView.canShowCallout = YES;

            // 詳細ディスクロージャボタンを吹き出しに追加する
            UIButton* rightButton = [UIButton buttonWithType:
                                     UIButtonTypeDetailDisclosure];
            [rightButton addTarget:self action:@selector(myShowDetailsMethod:)
                                     forControlEvents:UIControlEventTouchUpInside];
            pinView.rightCalloutAccessoryView = rightButton;
        }
        else
            pinView.annotation = annotation;

        return pinView;
    }

    return nil;
}
```

地図の注釈オブジェクトの管理

アプリケーションで多数の注釈を扱う場合は、これらの注釈オブジェクトをどのように管理するか検討する必要があります。Map Viewはアクティブな注釈とそうでない注釈を区別せず、認識した注釈オブジェクトはすべてアクティブであると見なします。そのため、特定の座標点が画面上にあるときには、常に対応する注釈ビューを表示しようとします。これにより、2つの注釈の座標が近接していると、対応する注釈ビューが重なり合うこともあります。また、地図に数百もの注釈が含まれていると、ある程度以上に縮小表示した状態では、注釈ビューが大量に表示されて見栄えが悪くなります。さらに、ビューが密集しすぎると、ユーザがアクセスできないビューも出てきます。

注釈の過密状態を解消する唯一の方法は、Map Viewから注釈オブジェクトをいくつか取り除くことです。このためには通常、`mapView:regionWillChangeAnimated:`メソッドと`mapView:regionDidChangeAnimated:`メソッドを実装して、地図の拡大縮小レベルの変更を検出する必要があります。拡大縮小率の変更中に、注釈同士の距離に応じて、必要であれば注釈の追加または削除を行うことができます。また、注釈を取り除くその他の基準（ユーザの現在の位置など）を検討することもできます。

iPhone OS 4.0以降では、Map Kitには地図上の点と点の近さを簡単に求める関数が多数含まれています。注釈の地図座標を地図点の座標空間に変換する場合は、`MKMetersBetweenMapPoints`メソッドを使用して2点間の絶対距離を取得できます。各座標を地図矩形の中心として使用し、`MKMapRectIntersectsRect`関数を使用して交差部分を求めることもできます。すべての関数の一覧については、『*Map Kit Functions Reference*』を参照してください。

注釈ビューをドラッグ可能として登録する

iOS 4.0以降では、注釈ビューはドラッグ操作に対応しています。これにより、地図上での注釈のドラッグと、それに応じた注釈データの更新が非常に簡単にできるようになりました。ドラッグのための最小限のサポートを実装するには、以下のことを行う必要があります。

- 注釈オブジェクト内で、`setCoordinate:`メソッドを実装してMap Viewが注釈の座標点を更新できるようにする。
- 注釈ビューの作成時に、`draggable`プロパティを`YES`に設定する。

ドラッグ可能な注釈ビューをユーザがタッチアンドホールドすると、Map Viewは注釈ビューのドラッグ操作を開始します。ドラッグ操作が進行すると、Map Viewはデリゲートの`mapView:annotationView:didChangeDragState:fromOldState:`メソッドを呼び出して、ビューのドラッグ状態への変更を通知します。このメソッドを使用して、ドラッグ操作に関与したり応答したりできます。

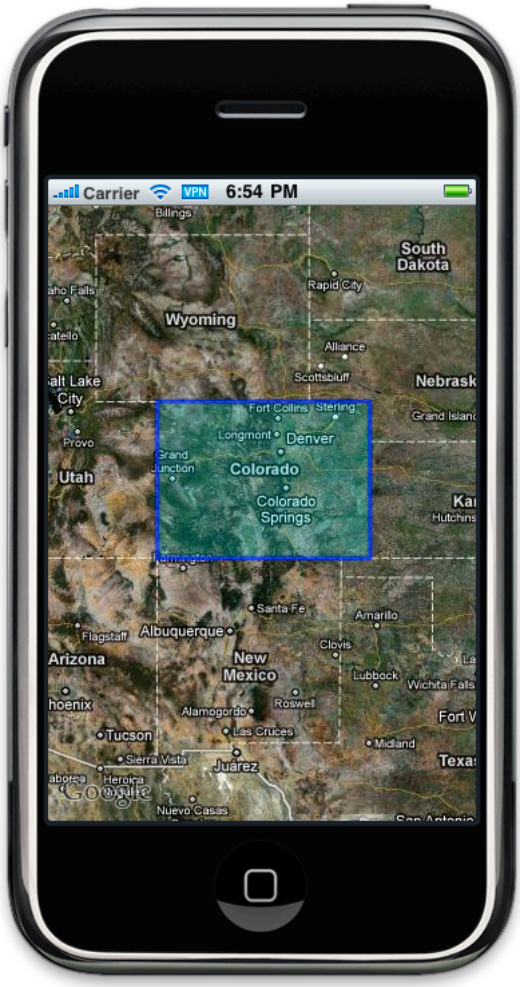
ドラッグ操作中にビューをアニメーション化したい場合は、注釈ビューに`dragState`メソッドを実装することでできます。Map Viewは、ドラッグ関連のタッチイベントを処理するたびに、影響を受ける注釈ビューの`dragState`プロパティを更新します。カスタムの`dragState`メソッドを実装すれば、アプリケーションは、これらの変更を横取りして追加の処理を実行する（ビューの外観をアニメーション化するなど）機会が得られます。たとえば、`MKPinAnnotationView`クラスは、ドラッグ操作が開始すると地図上のピンを抜き、ドラッグ操作が終了するとピンを刺します。

以前のバージョンのiOSでドラッグ可能な注釈をサポートする必要がある場合は、デベロッパ自身がドラッグ操作のサポートを実装しなければなりません。ドラッグ操作を独自に実装する方法の詳細およびサンプルコードについては、「従来のマップ技術」（51 ページ）を参照してください。

地図へのオーバーレイの表示

オーバーレイは、地図上の任意の領域の上にコンテンツを積み重ねる手段を提供します。注釈は常に1つの地図座標によって定義されるのに対し、オーバーレイは通常は複数の座標によって定義されます。これらの座標を使用して、連続または非連続の線、矩形、円、その他の一連の図形を作成し、さらに色で塗りつぶしやストロークの処理を施すことができます。たとえば、オーバーレイを使用して道路の上に交通情報を重ねたり、公園の区画を目立たせたり、市区町村、州、県、国の境界を示したりなどが可能です。図 5-2に、コロラド州を覆うオーバーレイの塗りつぶしとストロークを表示します。

図 5-2 地図へのオーバーレイの表示



地図にオーバーレイを表示するには、アプリケーションは2つのオブジェクトを指定する必要があります。

- `MKOverlay`プロトコルに準拠し、オーバーレイのデータ点を管理するオブジェクト（**オーバーレイオブジェクト**）。
- 地図の表面にオーバーレイの可視表現を描画する（`MKOverlayView`クラスから派生する）ビュー（**オーバーレイビュー(Overlay View)**）。

オーバーレイオブジェクトは通常、オーバーレイを定義する点とタイトル文字列などの関連情報が格納された小さなデータオブジェクトです。オーバーレイはプロトコルを使用して定義されるため、アプリケーション内のどのクラスもオーバーレイオブジェクトになり得ます。さらに、Map Kitは、各種の標準的な図形を指定する具象オーバーレイオブジェクトをいくつか定義しています。Map Viewは、追加されたオーバーレイオブジェクトの参照を保持しており、それらのオブジェクト内のデータを使用して、対応するビューをいつ表示すべきかを判断します。

Map Kitは、具象オーバーレイオブジェクトによって表現される任意の図形を描画できる、標準的なオーバーレイビューを提供します。注釈と同様、オーバーレイビューを地図の表面に直接追加することはできません。代わりに、要請されたらオーバーレイビューを提供し、Map Viewがそのビューを不透過のビュー階層に組み入れるようにします。オーバーレイビューを提供するにはMap Viewデリゲートオブジェクトを使用します。

一度定義した地図上のオーバーレイの位置は通常は変わりません。ドラッグ可能なオーバーレイを作成することは可能ですが、そうすることは稀であり、ドラッグ操作の追跡とオーバーレイ座標点の更新を行うコードをデベロッパ自身で実装する必要があります。

地図にオーバーレイを追加するためのチェックリスト

以下に、地図を利用するアプリケーションにオーバーレイを実装して使用するための手順を示します。これらの手順は、アプリケーションのインターフェイスのどこかに`MKMapView`オブジェクトが組み込まれていることを前提としています。

1. 次のいずれかの方法で、適当なオーバーレイデータオブジェクトを定義します。
    - `MKCircle`、`MKPolygon`クラスまたは`MKPolyline`クラスをそのまま使います。
    - `MKShape`または`MKMultiPoint`をサブクラス化して、アプリケーション固有の動作を提供するオーバーレイ、またはカスタムの図形を使用するオーバーレイを作成します。
    - アプリケーションの既存のクラスを利用し、それを`MKOverlay`プロトコルに適合させます。
2. 次のいずれかの方法で、画面に表示するオーバーレイビューを定義します。
    - 標準的な図形の場合は、`MKCircleView`、`MKPolygonView`または`MKPolylineView`を使用して注釈を描画します。これらのクラスを使って最終的な図形のさまざまな描画属性をカスタマイズできます。
    - `MKShape`から派生したカスタムの図形の場合は、`MKOverlayPathView`の適切なサブクラスを定義して、図形を描画します。
    - それ以外のカスタムの図形やオーバーレイの場合は、`MKOverlayView`をサブクラス化してカスタムの描画コードを実装します。
3. Map Viewデリゲートに`mapView:viewForOverlay:`メソッドを実装します。
4. `addOverlay:`メソッドやその他のメソッドを使って、Map Viewにオーバーレイデータオブジェクトを追加します。

注釈とは異なり、描画したオーバーレイは、地図の現在の拡大縮小レベルに合うように、自動的に拡大縮小されます。オーバーレイは一般的に、区画や道路など、ズーム操作時に拡大縮小するコンテンツを強調表示するので、オーバーレイの拡大縮小が必要になります。さらに、地図におけるオーバーレイのZ軸方向の重なり再配置して、特定のオーバーレイが常に最前面に表示されるようにすることができます。

標準のオーバーレイオブジェクトとビューの使用

指定した地図領域を強調表示するだけの場合は、標準のオーバーレイクラスを使用するのが最も簡単です。標準のオーバーレイクラスとしては、MKCircle、MKPolygon、MKPolylineがあります。これらのクラスはオーバーレイの基本的な図形を定義し、地図の表面上でこれらの図形の描画を処理するクラスMKCircleView、MKPolygonView、MKPolylineViewと一緒に使用されます。

リスト5-7に、図5-2 （43ページ）に示した矩形のポリゴン（多角形）の作成方法の例を示します。このポリゴンは、コロラド州の4つの角に対応する4つの地図座標で構成されています。ポリゴンを作成した後は、addOverlay:メソッドを使用して地図に追加するだけです。

リスト 5-7　ポリゴンのオーバーレイオブジェクトの作成

```
// コロラド州を覆うオーバーレイを定義する
CLLocationCoordinate2D  points[4];

points[0] = CLLocationCoordinate2DMake(41.000512, -109.050116);
points[1] = CLLocationCoordinate2DMake(41.002371, -102.052066);
points[2] = CLLocationCoordinate2DMake(36.993076, -102.041981);
points[3] = CLLocationCoordinate2DMake(36.99892, -109.045267);

MKPolygon* poly = [MKPolygon polygonWithCoordinates:points count:4];
poly.title = @"Colorado";

[map addOverlay:poly];
```

オーバーレイを地図に表示できるようにするために、Map Viewデリゲートの mapView:viewForOverlay:メソッドで適切なオーバーレイビューを用意する必要があります。標準のオーバーレイ図形の場合は、表示する図形のタイプに合致するビューを作成することで、オーバーレイビューを準備できます。リスト 5-8に、コロラド州を覆うポリゴンビューを作成するようにこのメソッドを実装する例を示します。この例では、このメソッドは図形とボーダー幅を描画する色を設定しています。

リスト 5-8　図形描画のためのポリゴンビューの作成

```
- (MKOverlayView *)mapView:(MKMapView *)mapView viewForOverlay:(id
<MKOverlay>)overlay
{
    if ([overlay isKindOfClass:[MKPolygon class]])
    {
        MKPolygonView*    aView = [[[MKPolygonView alloc]
initWithPolygon:(MKPolygon*)overlay] autorelease];

        aView.fillColor = [[UIColor cyanColor] colorWithAlphaComponent:0.2];
        aView.strokeColor = [[UIColor blueColor] colorWithAlphaComponent:0.7];
        aView.lineWidth = 3;

        return aView;
    }

    return nil;
}
```

標準のオーバーレイビューは、オーバーレイによって表現される図形の塗りつぶしとストロークを簡素化するためにあります。ほかに情報を表示する場合は、必要な描画を行うカスタムのオーバーレイビューを作成する必要があります。追加のコンテンツをレンダリングするために既存のオー

バーレイにサブビューを追加することは避けてください。オーバーレイに追加するサブビューはすべて、オーバーレイと一緒に拡大縮小され、地図の拡大縮小レベルに合わせられます。サブビューに含まれているコンテンツが拡大縮小に適切に対応できるのでない限り、結果の見栄えはあまり良くないことが考えられます。

カスタムオーバーレイオブジェクトの定義

オーバーレイオブジェクトの役割は、オーバーレイの座標データおよび関連する追加情報を管理することです。Map Kitは、カスタムオーバーレイを定義する2つの方法を提供しています。`MKShape`または`MKMultiPoint`をサブクラス化して新しいタイプの図形ベースのオーバーレイを定義する方法と、アプリケーションの既存のクラスの1つに`MKOverlay`プロトコルを組み入れる方法です。どちらの方法を選択するかは、必要なデータを持っているクラスがすでにあるかどうかに大きく依存します。クラスがあれば、その既存のクラスにプロトコルを組み入れます。クラスがなければ、`MKShape`または`MKMultiPoint`をサブクラス化してカスタム図形サブクラスを作成します。

サブクラス化するか、`MKOverlay`プロトコルを採用するかを問わず、カスタムオーバーレイオブジェクトで行う作業は同じです。オーバーレイオブジェクトの主要な役割は、2つの重要な情報を提供することです。

- オーバーレイの中心点を定義する座標
- オーバーレイのコンテンツを完全に囲む境界矩形

この2つの情報のうち、境界矩形がオーバーレイ自身にとって最も重要な情報です。Map Viewは、オーバーレイオブジェクトによって指定される境界矩形を手がかりに、対応するオーバーレイビューをいつ地図に追加するかを決定します（オーバーレイを注釈としても地図に追加する場合、同様に座標値によって、対応する注釈ビューをいつ地図に追加すべきかが決まります）。境界矩形自身は、地図座標ではなく、地図点を使用して指定されます。2つの座標系の間は、Map Kit関数を使用して変換できます。

オーバーレイの表示に伴う実際の作業のほとんどは、対応するオーバーレイビューオブジェクトが責任を負います。オーバーレイオブジェクトは、地図上でオーバーレイを配置する場所を単に定義するのに対し、オーバーレイビューは、オーバーレイに表示される情報（もしあれば）など、オーバーレイの最終的な表示内容を定義します。カスタムオーバーレイビューの作成については、「カスタムオーバーレイビューの定義」（46 ページ）で詳しく述べます。

カスタムオーバーレイビューの定義

オーバーレイ図形に対して、境界線の描画やコンテンツの塗りつぶし以上の処理を施したい場合は、カスタムオーバーレイビューを作成する必要があります。カスタムオーバーレイでは、任意のコンテンツを描画する機会が与えられます。たとえば、交通情報のオーバーレイを描画する場合は、カスタムオーバーレイビューを使用して、交通状況に基づいて道路を色分けすることができます。また、カスタムの描画コードを使用して、オーバーレイの表示をアニメーション化することもできます。

カスタムのオーバーレイビューを作成するには、`MKOverlayView`をサブクラス化します（既存の図形ベースのオーバーレイの描画動作を単に変更するだけであれば、代わりに`MKOverlayPathView`をサブクラス化することもできます）。カスタムの実装においては、次のメソッドを実装する必要があります。

- drawMapRect:zoomScale:inContext:。このメソッドはカスタムコンテンツを描画します。
- canDrawMapRect:zoomScale:。描画コードが、常に利用可能とは限らないコンテンツに依存する場合はこのメソッドを使用します。

canDrawMapRect:zoomScale:メソッドは、コンテンツが必ずしも描画する準備ができているとは限らない状況で使用します。たとえば、交通情報オーバーレイであれば、描画する前に、必要な交通データをネットワークからダウンロードする必要があります。このメソッドからNOを返されると、Map Viewは、アプリケーション側から準備ができたことを知らされるまでビューの描画を行いません。準備ができたことを知らせるには、setNeedsDisplayInMapRect:メソッドまたはsetNeedsDisplayInMapRect:zoomScale:メソッドのどちらかを使用して、ビューをダーティとしてマークします。

ビューの描画の準備ができたら、Map ViewはdrawMapRect:zoomScale:inContext:メソッドを呼び出して実際の描画を行います。標準のビューへの描画とは異なり、オーバーレイビューへの描画では、以下を含め、いくつか特別な配慮が必要です。

- 描画コードは、ビューの境界やフレームを、描画の基準点として使用することはできません。代わりに、オーバーレイオブジェクトに関連付けられている地図点を使用して図形を定義する必要があります。描画コードは、MKOverlayViewクラスの変換ルーチンを使用して、描画の直前にこれらの地図点を点（CGPointなど）に変換する必要があります。

  また、このメソッドに渡された拡大縮小の尺度値は、通常は直接コンテンツには適用しません。この値は、Map Kit関数またはメソッドが特別に必要とする場合に限り指定します。地図点を使用してコンテンツを指定し、それを点に変換する限り、コンテンツは自動的に正しいサイズに拡大縮小されるはずです。

- UIKitのクラスと関数を使用して描画する場合は、描画環境の設定と後処理を明示的に行う必要があります。呼び出しに先立ち、UIGraphicsPushContext関数を呼び出して、メソッドに渡されたコンテキストをカレントコンテキストにします。描画を終えたら、UIGraphicsPopContextを呼び出してコンテキストを削除します。
- Map Viewは大きなオーバーレイをタイル表示して、各タイルを別々のスレッドでレンダリングできる点に留意してください。そのため、描画コードは、変数やその他のデータをスレッドセーフな方法で変更することができない限り、変更しようとしてはなりません。

リスト 5-9に、オーバーレイの境界矩形をグラデーションを使って塗りつぶす描画コードを示します。グラデーションを描画する際は、対象の描画領域にクリッピング矩形を適用することによって、この描画処理を封じ込めることが特に重要です。ビューのフレームはオーバーレイの境界矩形よりも実際には大きいため、クリッピング矩形が適用されなければ、グラデーションは想定している領域の範囲を超えてレンダリングされるためです。オーバーレイの境界矩形は、この場合は実際の図形を定義するため、このメソッドは単に境界矩形に合わせてクリッピングします。より複雑なオーバーレイの場合は、オーバーレイを表すパスに合わせてクリッピングを行う必要があります。この描画コードの結果を、図 5-3 （49 ページ）に示します。

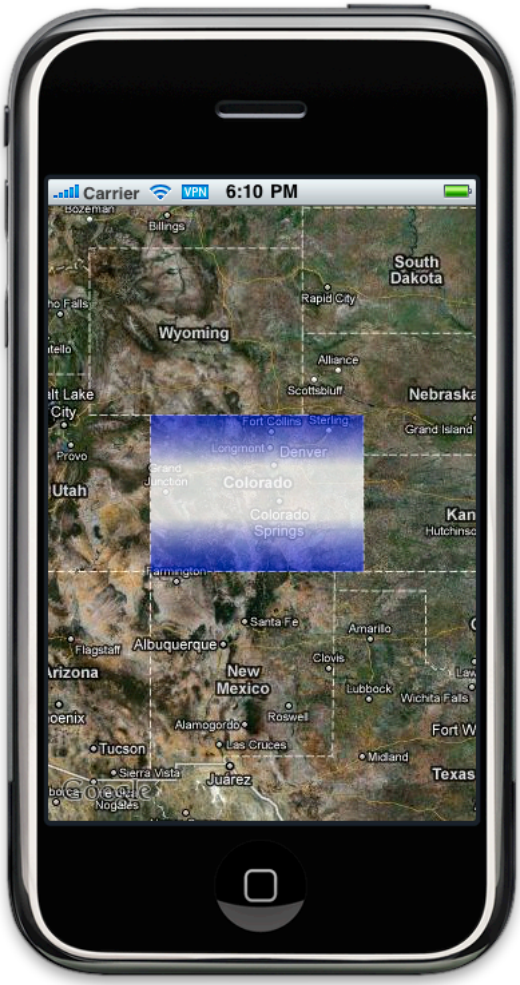
リスト 5-9　カスタムオーバーレイビューでのグラデーションの描画

```
- (void)drawMapRect:(MKMapRect)mapRect zoomScale:(MKZoomScale)zoomScale
inContext:(CGContextRef)context
{
   // オーバーレイの境界矩形を取得する
   MKMapRect  theMapRect = [self.overlay boundingMapRect];
   CGRect theRect = [self rectForMapRect:theMapRect];
```

```
    // コンテキストを境界矩形に合わせてクリップする
    CGContextAddRect(context, theRect);
    CGContextClip(context);

    // グラデーションの色と位置情報を設定する
    CGColorSpaceRef myColorSpace = CGColorSpaceCreateDeviceRGB();
    CGFloat locations[4] = {0.0, 0.33, 0.66, 1.0};
    CGFloat components[16] = {0.0, 0.0, 1.0, 0.5,
                              1.0, 1.0, 1.0, 0.8,
                              1.0, 1.0, 1.0, 0.8,
                              0.0, 0.0, 1.0, 0.5};

    // グラデーションを作成する
   CGGradientRef myGradient = CGGradientCreateWithColorComponents(myColorSpace,
 components, locations, 4);
    CGPoint start, end;
    start = CGPointMake(CGRectGetMidX(theRect), CGRectGetMinY(theRect));
    end = CGPointMake(CGRectGetMidX(theRect), CGRectGetMaxY(theRect));

    // 描画
    CGContextDrawLinearGradient(context, myGradient, start, end, 0);

    // クリーンアップ。
    CGColorSpaceRelease(myColorSpace);
    CGGradientRelease(myGradient);
}
```

図 5-3に、コロラド州を表すオーバーレイ上にカスタムコンテンツを描画した結果を示します。この場合、オーバーレイビューはそのコンテンツをカスタムグラデーションで塗りつぶしています。

図 5-3 カスタムオーバーレイビューを使用した描画



デリゲートオブジェクトからのオーバーレイビューの作成

オーバーレイビューが必要になると、Map Viewはそのデリゲートオブジェクトの`mapView:viewForOverlay:`メソッドを呼び出します。このメソッドを実装していない場合、または実装していても必ず`nil`を返す場合、Map Viewは指定されたオーバーレイに対して何も表示しません。そのため、地図に表示したいすべてのオーバーレイについて、このメソッドを実装して有効なオーバーレイビューを返す必要があります。

ほとんどの場合、オーバーレイはそれぞれに異なります。オーバーレイビューは必ず`mapView:viewForOverlay:`メソッドで作成する必要がありますが、オーバーレイビューをどのように設定するかに関しては多少の独創性を加える必要があるかもしれません。すべてのビューの描画属性が共通の場合は、このメソッドを、リスト 5-8 （45 ページ）に示した方法と同様の方法で実装できます。しかし、オーバーレイがそれぞれ異なる色や異なる属性を使用している場合は、このメソッドに巨大なディシジョンツリーを持たせるのではなく、注釈オブジェクトを使って情報を初期化する方法を探らなくてはなりません。

オーバーレイは通常は1つ1つ異なるため、地図ビューは、地図から削除されたオーバーレイビューを再利用することはありません。既存のオーバーレイビューをキューから取り出す代わりに、毎回新しいオーバーレイビューを作成する必要があります。

地図のオーバーレイオブジェクトの管理

アプリケーションで複数のオーバーレイを扱う場合は、これらのオーバーレイオブジェクトをどのように管理するかについて検討する必要があります。注釈と同様に、地図に関連付けられているオーバーレイは、オーバーレイのいずれかの部分が地図上の可視部分と交わるときには常に表示されます。また、注釈とは異なり、オーバーレイは地図に比例して拡大縮小するため、オーバーレイ同士が自動的に重なり合うことはありません。これは、オーバーレイの過密状態を解消するために、オーバーレイを削除したり後から追加したりする可能性はほとんどないことを意味します。2つのオーバーレイの境界矩形が重なり合う状況では、オーバーレイの1つを取り除くか、オーバーレイのZ軸における重なり順を並べ替えて、どのオーバーレイを最前面に表示するかを調整することができます。

`MKMapView`クラスの`overlays`プロパティには、登録済みのオーバーレイが、順序付きの配列として格納されています。この配列内のオブジェクトの順番は、レンダリング時のオブジェクトのZ軸における重なり順と一致し、配列の先頭のオブジェクトがZ軸における重なり順の一番下を表しています。あるオーバーレイを残りのすべてのオーバーレイの最前面に配置するには、この配列の最後に追加します。またMap Viewのメソッドを使用して、配列内のさまざまな位置にオブジェクトを挿入したり、配列内の2つのオブジェクトの位置を入れ替えたりすることができます。

オーバーレイに関して何らかのオーバーラップ検出アルゴリズムを実装する場合は、Map Viewデリゲートの`mapView:didAddOverlayViews:`メソッド内でこれを行います。このメソッドが呼び出されたときに、`MKMapRectIntersectsRect`関数を使用して、追加されたオーバーレイがほかのオーバーレイの境界と交差しているかを確認できます。オーバーラップがある場合は、何らかのカスタムロジックを使用して、どのオーバーレイをレンダリングツリーの最前面に配置すべきかを選び、必要に応じて位置を入れ替えます（Map Viewはインターフェイス項目であるため、`overlays`配列への変更はすべて、アプリケーションのメインスレッド上で同期化し実行しなければなりません。ただし、実際の比較は別のスレッドで生じるかもしれません）。

オーバーレイを注釈として使用

`MKOverlay`プロトコルは、`MKAnnotation`プロトコルに準拠しています。そのため、オーバーレイオブジェクトはすべて注釈オブジェクトでもあり、コード内ではどちらか一方として扱うことも、両方として扱うこともできます。オーバーレイオブジェクトを注釈としても扱う場合は、2つの場所でそのオブジェクトを管理するのはデベロッパの責任です。オーバーレイビューとその注釈ビューを両方とも表示したい場合は、アプリケーションデリゲートに`mapView:viewForOverlay:`メソッドと`mapView:viewForAnnotation:`メソッドの両方を実装する必要があります。また、オブジェクトの追加と削除は、地図の`overlays`配列および`annotations`配列の両方で行う必要がります。

従来のマップ技術

以前のバージョンのiOSでのドラッグ可能な注釈の作成

iOS 3.xでドラッグ可能な注釈をサポートする必要がある場合は、ドラッグ関連のタッチを追跡するコードを独自に実装する必要があります。注釈ビューは地図コンテンツ上の特殊レイヤにありますが、タッチイベントを受け取ることができる完全なビューです。これらのイベントを使用して、注釈ビュー内でのヒットを検出してドラッグ操作を開始することができます。

注： 地図はスクロールするインターフェイスに表示されるため、一般に、ユーザがカスタムビューをタッチした時点とそれに対応するイベントが配信される時点の間に若干の遅れが生じます。この遅れの間に、基盤のスクロールビューはこのタッチイベントがスクロールジェスチャの一部かを判別できます。

次の一連のコードリストは、iOS 3.xでユーザによる移動が可能な注釈ビューを実装する方法を示します。この例では、注釈ビューには注釈の座標点の真上に標的画像が表示されます。また、ターゲットについての詳細情報を表示するカスタムアクセサリビューが含まれています。

リスト A-1は、`BullseyeAnnotationView`クラスの定義を示しています。このクラスには、ビューを正しく移動させるために追跡中に使用するメンバ変数がいくつか追加されています。このクラスにはMap View自身へのポインタも格納されています。この値は、注釈ビューを作成または初期化し直すときに、`mapView:viewForAnnotation:`メソッド内のコードで設定されます。Map Viewオブジェクトは、イベントの追跡が終わったときに注釈オブジェクトの地図座標を調整するために必要です。

リスト A-1 `BullseyeAnnotationView`クラス

```
@interface BullseyeAnnotationView : MKAnnotationView
{
    BOOL isMoving;
    CGPoint startLocation;
    CGPoint originalCenter;

    MKMapView* map;
}

@property (assign,nonatomic) MKMapView* map;

- (id)initWithAnnotation:(id <MKAnnotation>)annotation;

@end

@implementation BullseyeAnnotationView
@synthesize map;
- (id)initWithAnnotation:(id <MKAnnotation>)annotation
{
    self = [super initWithAnnotation:annotation
```

```
                reuseIdentifier:@"BullseyeAnnotation"];
    if (self)
    {
        UIImage*    theImage = [UIImage imageNamed:@"bullseye32.png"];
        if (!theImage)
            return nil;

        self.image = theImage;
        self.canShowCallout = YES;
        self.multipleTouchEnabled = NO;
        map = nil;

        UIButton*    rightButton = [UIButton buttonWithType:
                      UIButtonTypeDetailDisclosure];
        [rightButton addTarget:self action:@selector(myShowAnnotationAddress:)
                      forControlEvents:UIControlEventTouchUpInside];
        self.rightCalloutAccessoryView = rightButton;
    }
    return self;
}
@end
```

標的ビューに最初にタッチイベントが届くと、このクラスの`touchesBegan:withEvent:`メソッドは、リスト A-2に示すように、最初のタッチイベントについての情報を記録します。後から`touchesMoved:withEvent:`メソッドでこの情報を使用して、ビューの位置を調整します。すべての位置情報はスーパービューの座標空間に格納されます。

リスト A-2　ビューの位置の追跡

```
@implementation BullseyeAnnotationView (TouchBeginMethods)
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    // ビューをシングルタッチ専用に設定する
    UITouch* aTouch = [touches anyObject];
    startLocation = [aTouch locationInView:[self superview]];
    originalCenter = self.center;

    [super touchesBegan:touches withEvent:event];
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch* aTouch = [touches anyObject];
    CGPoint newLocation = [aTouch locationInView:[self superview]];
    CGPoint newCenter;

    // ユーザの指が5ピクセル以上移動した場合、ドラッグが始まる
    if ( (abs(newLocation.x - startLocation.x) > 5.0) ||
         (abs(newLocation.y - startLocation.y) > 5.0) )
         isMoving = YES;

    // ドラッグが始まった場合は、ビューの位置を調整する
    if (isMoving)
    {
        newCenter.x = originalCenter.x + (newLocation.x - startLocation.x);
        newCenter.y = originalCenter.y + (newLocation.y - startLocation.y);
        self.center = newCenter;
    }
```

```
    else    // 親クラスにそれを処理させる
        [super touchesMoved:touches withEvent:event];
}
@end
```

ユーザが注釈ビューのドラッグを停止したときは、元の注釈の座標を調整してそのビューが新しい位置に確実に留まるようにする必要があります。リスト A-3は、BullseyeAnnotationViewクラスのtouchesEnded:withEvent:メソッドを示しています。このメソッドは、mapメンバ変数を使用してピクセルベースの点を地図座標値に変換します。注釈のcoordinateプロパティは通常は読み取り専用のため、ここでは注釈オブジェクトがcoordinateプロパティを使用して、ローカルに格納したり報告する値を更新するために、changeCoordinateカスタムメソッドを実装しています。何らかの理由でタッチイベントがキャンセルされた場合は、touchesCancelled:withEvent:メソッドが注釈ビューを元の位置に戻します。

リスト A-3　最後のタッチイベントの処理

```
@implementation BullseyeAnnotationView (TouchEndMethods)
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (isMoving)
    {
        // 新しい位置を表すように地図座標を更新する
        CGPoint newCenter = self.center;
        BullseyeAnnotation* theAnnotation = self.annotation;
        CLLocationCoordinate2D newCoordinate = [map convertPoint:newCenter
                            toCoordinateFromView:self.superview];

        [theAnnotation changeCoordinate:newCoordinate];

        // 状態情報をクリーンアップする
        startLocation = CGPointZero;
        originalCenter = CGPointZero;
        isMoving = NO;
    }
    else
        [super touchesEnded:touches withEvent:event];
}

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (isMoving)
    {
        // ビューを最初の位置に戻す
        self.center = originalCenter;

        // 状態情報をクリーンアップする
        startLocation = CGPointZero;
        originalCenter = CGPointZero;
        isMoving = NO;
    }
    else
        [super touchesCancelled:touches withEvent:event];
}
@end
```

書類の改訂履歴

この表は「*位置情報対応プログラミングガイド*」の改訂履歴です。

日付	メモ
2011-10-12	新しいCore LocationのGeocoderについての情報を追加しました。
2010-05-20	領域観測についての情報を追加しました。
	オーバーレイの作成についての情報を追加しました。
	地図と注釈についての既存の情報に補足を加えました。
	ユーザの位置を取得する新しいテクノロジーを盛り込むように、位置情報関連のセクションを更新しました。
2010-03-24	アプリケーションで位置情報サービスおよび地図サービスを利用する方法について説明した新規文書。

書類の改訂履歴