キー値監視プログラミングガイド

Cocoa > Data Management

2011-03-08

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

リスト

キー値監視の登録　11

KVOへの準拠　15

依存キーの登録　19

キー値監視プログラミングガイドの紹介

キー値監視（Key-Value Observing）は、ほかのオブジェクトに属する指定のプロパティの変化についての通知をオブジェクトが受け取れるようにする仕組みです。

重要： キー値監視を理解するには、まず「キー値コーディング」を読んでください。

キー値監視とは、ほかのオブジェクトに属する特定のプロパティの変化について通知をオブジェクトが受け取れるようにする仕組みです。特にアプリケーションのモデルレイヤとコントロールレイヤ間の通信に有用です（Mac OS Xでは、コントローラレイヤのバインディング技術は、キー値監視に大きく依存しています）。一般に、コントローラオブジェクトはモデルオブジェクトのプロパティを監視し、ビューオブジェクトはコントローラを介してモデルオブジェクトのプロパティを監視します。それだけでなく、モデルオブジェクトがほかのモデルオブジェクト（通常は依存元の値の変化）を監視する、あるいは自分自身（やはり依存元の値の変化）を監視することも可能です。

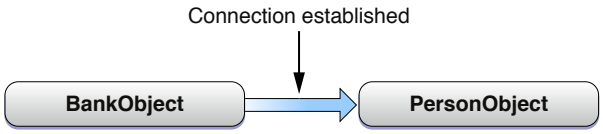
単純な属性や対1リレーションシップ、対多リレーションシップなど、あらゆるオブジェクトプロパティを監視できます。対多リレーションシップを監視するオブザーバには、生じた変化の種類に加え、変化にかかわったオブジェクトも通知されます。

プロパティのオブザーバは、3段階に分けて設定します。この各段階を把握すれば、KVOの動作原理も明快になるでしょう。

1. まず、キー値監視の長所を活かせる状況について、明確なシナリオを描くことから始めてください。たとえば、ほかのオブジェクトのあるプロパティが変化したとき、その旨の通知を受け取る必要があるオブジェクトを明らかにするのです。

   たとえば`PersonObject`（個人）は、`BankObject`（銀行口座）の`accountBalance`（残高）が変わったとき、その旨を通知して欲しいと思うでしょう。

2. この場合、PersonObjectを、BankObjectのaccountBalanceプロパティのオブザーバとして登録しなければなりません。これはaddObserver:forKeyPath:options:context:メッセージを送って行います。



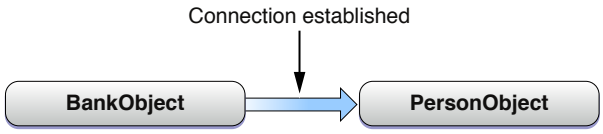

```
-(void) observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
        change:(NSDictionary *)change
            context:(void *)context
{

//custom implementation
//be sure to call the superclass' implementation
//if the superclass implements it

}
```

注意： addObserver:forKeyPath:options:context:メソッドは、指定されたオブジェクトのインスタンス間を結びつけます。2つのクラス間ではなく、そのインスタンス間を結びつける、という点が重要です。

3. 変更通知を受け取るため、オブザーバ側にはobserveValueForKeyPath:ofObject:change:context:メソッドを実装しなければなりません。オブザーバが変更通知にどのように応答するかは、このメソッドに定義することになります。監視対象プロパティの変化に対する応答方法をカスタマイズする場合も、このメソッドに実装します。

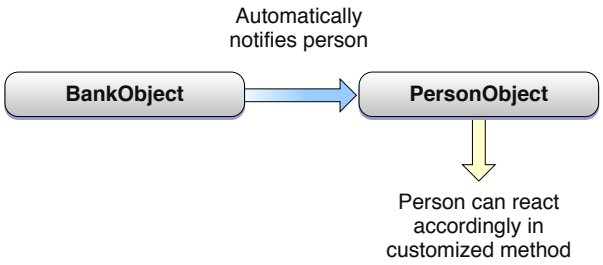



```
[bankInstance addObserver:personInstance
    forKeyPath:@"accountBalance"
        options:NSKeyValueObservingOptionNew
            context:NULL];
```

「キー値監視の登録」（11 ページ）で、監視対象の登録方法と通知の受け取り方を説明します。

4. `observeValueForKeyPath:ofObject:change:context:`メソッドは、監視対象プロパティの値がKVOに準拠した形で変化した、あるいはこれが依存するキーが変化したとき、自動的に起動されます。

```
[bankInstance setAccountBalance:50];
```



「依存キーの登録」（19 ページ）では、あるキーの値がほかのキーの値に依存している旨の指定方法について説明します。

KVOの一番の利点は、プロパティが変化する都度通知を送る、独自のスキームを実装する必要がないことです。基盤機能がフレームワークとして提供されているため、個々のプロジェクトでは多くの場合、別途コードを記述しなくても容易にKVOに準拠させることができます。さらに、必要な基盤機能がすべて揃っているので、同じプロパティや依存値を複数のオブザーバが監視する、というようなことも容易に実装できます。

「KVOへの準拠」（15 ページ）では、自動キー値監視と手動キー値監視の違い、およびそれぞれの実装方法を説明します。

`NSNotificationCenter`とは異なり、すべてのオブザーバに変更通知を送る中核的なオブジェクトはありません。その代わり、変化が生じると監視元オブジェクトに直接通知が送られます。`NSObject`は、キー値監視について土台となる実装を提供しており、これらのメソッドをオーバーライドする必要が生じることはまれです。

「キー値監視の実装の詳細」（23 ページ）では、キー値監視の実装方法を説明します。

キー値監視の登録

プロパティのキー値監視通知を受け取るには、次の3つの要件を満たす必要があります。

- 監視対象のクラスは、監視するプロパティについてキー値監視対応でなければなりません。
- `addObserver:forKeyPath:options:context:`メソッドを使用して、監視元オブジェクトを監視対象オブジェクトに登録する必要があります。
- 監視元クラスは`observeValueForKeyPath:ofObject:change:context:`を実装する必要があります。

重要： あらゆるクラスのあらゆるプロパティが、自動的にKVOに準拠するわけではありません。独自に開発したクラスをKVOに準拠させるためには、「KVOへの準拠」（15 ページ）に従って実装する必要があります。一般に、Appleが提供するフレームワークのプロパティがKVOに準拠しているのは、仕様にその旨が明記されている場合に限ります。

オブザーバとしての登録

プロパティの変化の通知を受け取るには、監視元オブジェクト（オブザーバ）を監視対象オブジェクトに登録するために、監視対象オブジェクトに`addObserver:forKeyPath:options:context:`メッセージを送信して、監視元オブジェクトと、監視対象プロパティのキーパスを渡す必要があります。optionsパラメータは、変更通知が送信されたときにオブザーバに提供する情報を指定します。`NSKeyValueObservingOptionOld`オプションを使用すると、変更前のオブジェクト値を変更辞書のエントリとしてオブザーバに送るよう指定できます。`NSKeyValueObservingOptionNew`オプションを使用すると、変更後の新しい値が変更辞書のエントリとして送られます。両方の値を受け取るには、オプション定数のビットごとの`OR`（論理和）を指定します。

リスト1は、`openingBalance`というプロパティに対し、インスペクタオブジェクトを登録する例です。

リスト 1　openingBalanceプロパティのオブザーバとしてのインスペクタの登録

```
- (void)registerAsObserver
{
    /*
     accountオブジェクトのopeningBalanceプロパティの変更通知を受け取る
     inspectorを登録し、openingBalanceの変更前と変更後の両方の値が
     与えられるよう、オブザーバのメソッドで指定します。
     */
    [account addObserver:inspector
              forKeyPath:@"openingBalance"
                 options:(NSKeyValueObservingOptionNew |
                          NSKeyValueObservingOptionOld)
                 context:NULL];
```

```
}
```

オブジェクトをオブザーバとして登録するときに、コンテキストポインタを提供することもできます。コンテキストポインタは、`observeValueForKeyPath:ofObject:change:context:`が呼び出されたときに、オブザーバに渡されます。コンテキストポインタとしては、C言語のポインタおよびオブジェクト参照を指定できます。コンテキストポインタは、監視の対象となる変化を識別する一意の識別子として、あるいは、オブザーバに何らかのデータを渡すために使用できます。コンテキストポインタは保持されません。また、監視元オブジェクトがオブザーバとして削除される前にコンテキストポインタが解放されないよう、アプリケーション側で対処する必要があります。

注意： キー値監視の`addObserver:forKeyPath:options:context:`メソッドは、監視元オブジェクトも監視対象オブジェクトも保持しません。アプリケーションの要件を確認し、監視元オブジェクトと監視対象オブジェクトの保持と解放を管理する必要があります。

変更通知の受信

オブジェクトの監視対象プロパティの値が変化すると、オブザーバは`observeValueForKeyPath:ofObject:change:context:`メッセージを受け取ります。オブザーバはすべてこのメソッドを実装しなければなりません。

オブザーバに通知されるのは、オブザーバへの通知をトリガしたオブジェクトとキーパス、変化の詳細を格納した辞書、そしてオブザーバの登録時に渡したコンテキストポインタです。

変更辞書のエントリである`NSKeyValueChangeKindKey`は、発生した変化の種類に関する情報を提供します。監視対象オブジェクトの値が変化した場合、`NSKeyValueChangeKindKey`エントリは`NSKeyValueChangeSetting`を返します。オブザーバの登録時に指定したオプションに応じて、変更辞書の`NSKeyValueChangeOldKey`エントリおよび`NSKeyValueChangeNewKey`エントリに、それぞれ変更前と変更後の値が格納されます。プロパティがオブジェクトであれば、値をそのまま使います。一方、スカラやCの構造体であれば、値を`NSValue`オブジェクトにラップします（キー値コーディングと同様）。

監視対象のプロパティが対多リレーションシップの場合、`NSKeyValueChangeKindKey`エントリは、リレーションシップに含まれているオブジェクトが挿入、削除、または置換されたのかどうかを示すことができます。挿入の場合は`NSKeyValueChangeInsertion`、削除は`NSKeyValueChangeRemoval`、置換は`NSKeyValueChangeReplacement`がそれぞれ返されます。

`NSKeyValueChangeIndexesKey`の変更辞書エントリは、変化のあったリレーションシップ内のインデックス位置を示す`NSIndexSet`オブジェクトです。オブザーバの登録時に`NSKeyValueObservingOptionNew`または`NSKeyValueObservingOptionOld`がオプションとして指定された場合、変更辞書の`NSKeyValueChangeOldKey`エントリおよび`NSKeyValueChangeNewKey`エントリはそれぞれ、変更前および変更後における関連オブジェクトの値を格納する配列となります。

リスト 2の例は、リスト 1 （11 ページ）で登録された`openingBalance`プロパティの変更前と変更後の値を反映するインスペクタに対する`observeValueForKeyPath:ofObject:change:context:`の実装です。

リスト 2 observeValueForKeyPath:ofObject:change:context:の実装

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
```

```
                        context:(void *)context
{
    if ([keyPath isEqual:@"openingBalance"]) {
        [openingBalanceInspectorField setObjectValue:
            [change objectForKey:NSKeyValueChangeNewKey]];
    }
    /*
     実装する場合は、必ずスーパークラスの実装を呼び出してください。
     NSObjectでメソッドの実装は行いません。
     */
    [super observeValueForKeyPath:keyPath
                         ofObject:object
                           change:change
                          context:context];
}
```

オブザーバとしてのオブジェクトの削除

キー値オブザーバの削除は、監視対象オブジェクトに対し、監視元オブジェクトとキーパスを指定して`removeObserver:forKeyPath:`メッセージを送信することによって行います。リスト 3は、`openingBalance`のオブザーバとしてのインスペクタを削除する例です。

リスト 3　　openingBalanceのオブザーバとしてのインスペクタの削除

```
- (void)unregisterForChangeNotification
{
    [observedObject removeObserver:inspector
                        forKeyPath:@"openingBalance"];
}
```

オブザーバの登録時に指定したコンテキストがオブジェクトだった場合、そのコンテキストを安全に解放できるのはオブザーバを削除した後だけです。監視元オブジェクトは、`removeObserver:forKeyPath:`を受け取ると、指定したキーパスとオブジェクトに関する`observeValueForKeyPath:ofObject:change:context:`メッセージは受け取らなくなります。

KVOへの準拠

特定のプロパティがKVO対応であるとするには、クラスが次の要件を満たす必要があります。

- 「KVOの準拠」で示したように、クラスがプロパティについてキー値コーディング対応でなければなりません。

  KVOはKVCと同じデータ型をサポートします。

- クラスはプロパティのKVO変更通知を送出できる必要があります。

変更通知が送出されるようにする方法は2通りあります。自動変更通知は、`NSObject`によって提供され、クラスにおいてキー値コーディング互換であるすべてのプロパティについて実行できます。一般に、標準的なCocoaのコーディング規約、名前付け規約に従っていれば、そのままで自動変更通知の機能が利用でき、別途コードを記述する必要はありません。

手動変更通知は、通知を送出するタイミングを細かく制御できますが、追加のコード作成が必要になります。サブクラスで、プロパティの自動通知を独自に制御したい場合は、クラスメソッド`automaticallyNotifiesObserversForKey:`を実装してください。

自動変更通知

`NSObject`は、自動キー値変更通知の基本的な実装を提供します。自動キー値変更通知は、キー値対応のアクセサやキー値コーディングのメソッドによって加えられた変更をオブザーバに通知します。`mutableArrayValueForKey:`などが返すコレクションプロキシオブジェクトにも、自動通知の機能が組み込まれています。

リスト 1は、`name`プロパティのオブザーバに変更の通知を行う例です。

リスト 1　　KVO変更通知を送出するメソッド呼び出しの例

```
// アクセサメソッドの呼び出し
[account setName:@"Savings"];

// setValue:forKey:を使用
[account setValue:@"Savings" forKey:@"name"];

// キーパスを使用。accountは "document" のKVC対応のプロパティ
[document setValue:@"Savings" forKeyPath:@"account.name"];

// mutableArrayValueForKey:を使ってリレーションシッププロキシオブジェクトを検索
Transaction *newTransaction = <#Create a new transaction for the account#>;
NSMutableArray *transactions = [account mutableArrayValueForKey:@"transactions"];
[transactions addObject:newTransaction];
```

手動変更通知

手動変更通知では、通知をオブザーバに送る方法やタイミングについて細かく設定できます。手動通知は、不要な通知がトリガされるのを最小限に抑えたり、多数発生する変更をグループとして扱い、単一の通知にまとめたりする場合に役立ちます。

手動通知を実装するクラスでは、`automaticallyNotifiesObserversForKey:`の`NSObject`による実装をオーバーライド必要があります。同一クラス内で、自動と手動の両方のオブザーバ通知を使用できます。手動通知を実行するプロパティの場合、`automaticallyNotifiesObserversForKey:`のサブクラス実装では`NO`を返す必要があります。サブクラス実装は、認識されないキーがあれば`super`を呼び出さなければなりません。リスト 2は、`openingBalance`プロパティの手動通知を有効にして、ほかのすべてのキーについて通知の判断をスーパークラスに委ねる例です。

リスト 2　automaticallyNotifiesObserversForKey:の実装例

```
+ (BOOL)automaticallyNotifiesObserversForKey:(NSString *)theKey {

    BOOL automatic = NO;
    if ([theKey isEqualToString:@"openingBalance"]) {
        automatic = NO;
    } else {
        automatic=[super automaticallyNotifiesObserversForKey:theKey];
    }
    return automatic;
}
```

手動通知を実装するには、値を変更する前に`willChangeValueForKey:`を、値を変更した後に`didChangeValueForKey:`をそれぞれ呼び出す必要があります。リスト 3は、`openingBalance`プロパティについて手動通知を実装する例です。

リスト 3　手動通知を実装するアクセサメソッドの例

```
- (void)setOpeningBalance:(double)theBalance {
    [self willChangeValueForKey:@"openingBalance"];
    openingBalance=theBalance;
    [self didChangeValueForKey:@"openingBalance"];
}
```

値が変化したかどうかを最初にチェックすることにより、不要な通知の送信を最小限に抑えることができます。リスト 4は、`openingBalance`の値をテストし、変化があった場合にのみ通知を送るようにする例です。

リスト 4　通知送信前に値をテストし、変化の有無を確認する例

```
- (void)setOpeningBalance:(double)theBalance {
    if (theBalance != openingBalance) {
        [self willChangeValueForKey:@"openingBalance"];
        openingBalance=theBalance;
        [self didChangeValueForKey:@"openingBalance"];
    }
}
```

1つの操作で複数のキーが変更される場合は、リスト 5のように、変化の通知をネストする必要があります。

リスト 5　　複数のキーの変更通知のネスト

```
- (void)setOpeningBalance:(double)theBalance {
    [self willChangeValueForKey:@"openingBalance"];
    [self willChangeValueForKey:@"itemChanged"];
    openingBalance=theBalance;
    itemChanged=itemChanged+1;
    [self didChangeValueForKey:@"itemChanged"];
    [self didChangeValueForKey:@"openingBalance"];
}
```

順序付き対多リレーションシップの場合は、変化のあったキーだけでなく、変化の種類と変化にかかわったオブジェクトのインデックスも指定する必要があります。変化の種類は、`NSKeyValueChangeInsertion`、`NSKeyValueChangeRemoval`、または`NSKeyValueChangeReplacement`を指定する`NSKeyValueChange`です。影響のあったオブジェクトのインデックスは、`NSIndexSet`として渡されます。

リスト6は、対多リレーションシップの`transactions`においてオブジェクトの削除をラップする方法を示したコードの抜粋です。

リスト 6　　対多リレーションシップにおける手動通知の実装

```
- (void)removeTransactionsAtIndexes:(NSIndexSet *)indexes {
    [self willChange:NSKeyValueChangeRemoval
     valuesAtIndexes:indexes forKey:@"transactions"];

    // 指定されたインデックス位置のtransactionオブジェクトをここで削除

    [self didChange:NSKeyValueChangeRemoval
    valuesAtIndexes:indexes forKey:@"transactions"];
}
```

注意： 参照カウント環境では、`willChange`メッセージを送信する前に、これから変化する値を解放してしまわないようにしなければなりません。

依存キーの登録

あるプロパティ値がほかのオブジェクトの属性に依存するといっても、さまざまな状況があります。ある属性値が変化したときには、それに依存するプロパティ値にも変化した旨の印をつけなければなりません。キー値監視通知を依存先プロパティに送る方法は、Mac OS Xの版と、関係の基数（対1か対多か）によって異なります。

Mac OS X v10.5以降、対1関係の場合

対象がMac OS X v10.5以降で、関係先オブジェクトとの対1関係がある場合、自動的に通知を送るためには、`keyPathsForValuesAffectingValueForKey:`をオーバーライドするか、依存キーを登録するために定義されたパターンに従い、適切な名前のメソッドを実装する必要があります。

たとえば、人のフルネームは、名（first name）と姓（last name）の両方に依存しています。フルネームを返すメソッドは、次のように記述できます。

```
- (NSString *)fullName {
    return [NSString stringWithFormat:@"%@ %@",firstName, lastName];
}
```

`firstName`や`lastName`が変化すれば`fullName`プロパティ値にも影響するので、このプロパティを監視するアプリケーションに通知を送出しなければなりません。

ひとつ考えられるのは、`keyPathsForValuesAffectingValueForKey:`をオーバーライドして、`fullName`プロパティが`lastName`および`firstName`に依存する旨を組み込む、という方法です。リスト 1 （19 ページ）に、実際にこれを実装した例を示します。

リスト 1　`keyPathsForValuesAffectingValueForKey:`の実装例

```
+ (NSSet *)keyPathsForValuesAffectingValueForKey:(NSString *)key
{
    NSSet *keyPaths = [super keyPathsForValuesAffectingValueForKey:key];

    if ([key isEqualToString:@"fullName"])
    {
        NSSet *affectingKeys = [NSSet setWithObjects:@"lastName",
@"firstName",nil];
        keyPaths = [keyPaths setByAddingObjectsFromSet:affectingKeys];
    }
    return keyPaths;
}
```

オーバーライドしたメソッドでは通常、superを実行して得られた（依存キーの）集合に、当該クラスに関係する依存キーを追加して返します。スーパークラスでも依存キーを追加している場合があるので、その結果を打ち消さないようにするためです。

同じ結果を、keyPathsForValuesAffecting<Key>というパターンの名前のクラスメソッドでも実装できます。ここで<Key>は、依存する側の属性名（先頭を大文字にしたもの）です。この方針によれば、リスト 1 （19 ページ）の機能は、keyPathsForValuesAffectingFullNameというクラスメソッドとして書き直すことができます（リスト 2 （20 ページ）を参照）。

リスト 2　　keyPathsForValuesAffecting<Key>メソッドの実装例

```
+ (NSSet *)keyPathsForValuesAffectingFullName
{
    return [NSSet setWithObjects:@"lastName", @"firstName", nil];
}
```

カテゴリの機能により、プロパティを動的に追加して既存のクラスを拡張している場合、keyPathsForValuesAffectingValueForKey:メソッドをオーバーライドすることはできません。カテゴリにより追加したメソッドは、オーバーライドを想定していないからです。この場合、クラスメソッドkeyPathsForValuesAffecting<Key>を用いれば、問題なく依存関係を組み込むことができます。

注意： 対多リレーションシップの依存関係を、keyPathsForValuesAffectingValueForKey:を実装することによりセットアップすることはできません。代わりに、対多コレクションの各オブジェクトの該当する属性を監視し、依存キー自身を更新することにより、値の変化に対応する必要があります。以下に、このような状況に対処するための方法について説明します。

Mac OS X v10.4、およびMac OS X v10.5で対多関係の場合

対象がMac OS X v10.4であれば、setKeys:triggerChangeNotificationsForDependentKey:にキーパスを渡すことはできないので、上記の方法は使えません。

Mac OS X v10.5であっても、対多関係であればやはり、keyPathsForValuesAffectingValueForKey:にキーパスを渡すことはできません。たとえば、Departmentオブジェクトに、Employeeとの対多関係（employees）があり、Employeeにはsalary属性があるとします。Departmentオブジェクトに totalSalaryという属性を定義し、関係で結ばれたEmployeeすべてのsalaryに依存して値が決まるようにしたいとしましょう。たとえばkeyPathsForValuesAffectingTotalSalaryを実装し、キーとして employees.salaryを返したとしても、これは実現できません。

いずれの場合も、2通りの解決法があります。

1. キー値監視を使って、親（この例ではDepartment）を、子（この例ではEmployee）すべての該当する属性の監視者として登録できます。子オブジェクトを追加/削除したときは、親を監視者として追加/削除しなければなりません（「キー値監視の登録」 （11 ページ）を参照）。observeValueForKeyPath:ofObject:change:context:メソッドで、次のコード例のように、依存元の値の変化に応じて依存先の値を更新します。

   ```
   - (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
   change:(NSDictionary *)change context:(void *)context
   {
       if (context == totalSalaryContext) {
           [self updateTotalSalary];
       }
       else
       // ほかのキー値を監視、またはsuperを起動...
   ```

```
}
- (void)updateTotalSalary
{
    [self setTotalSalary:[self valueForKeyPath:@"employees.@sum.salary"]];
}
- (void)setTotalSalary:(NSNumber *)newTotalSalary
{
    if (totalSalary != newTotalSalary) {
        [self willChangeValueForKey:@"totalSalary"];
        [totalSalary release];
        totalSalary = [newTotalSalary retain];
        [self didChangeValueForKey:@"totalSalary"];
    }
}
- (NSNumber *)totalSalary
{
    return totalSalary;
}
```

2. Core Dataを使っているならば、親を、アプリケーションの通知センターに、管理オブジェクトコンテキストの監視者として登録できます。親はキー値監視と同様の方法で、子が送信する変更通知に応答しなければなりません。

キー値監視の実装の詳細

自動キー値監視は、**isa入れ替え**（isa-swizzling）と呼ばれる手法を用いて実装します。

`isa`ポインタは、その名前が表しているように、ディスパッチテーブルを維持するオブジェクトのクラスを参照します。ディスパッチテーブルは基本的に、さまざまなデータのなかでも、クラスが実装するメソッドへのポインタを格納します。

オブジェクトの属性についてオブザーバが登録されている場合、監視対象オブジェクトのisaポインタは変更され、実際のクラスではなく、中間クラスを参照します。そのため、isaポインタの値がインスタンスの実際のクラスを必ずしも反映するわけではありません。

`isa`ポインタを使って、インスタンスが実際にどのクラスに属しているかを判断することはできません。代わりに`class`メソッドを使用してオブジェクトインスタンスのクラスを判断してください。

書類の改訂履歴

この表は「*キー値監視プログラミングガイド*」の改訂履歴です。

日付	メモ
2011-03-08	「依存キーの登録」の用語を明確にしました。
2009-08-14	Cocoaに関する重要な定義へのリンクをいくつか追加しました。
2009-05-09	細かな誤字を訂正しました。
2009-05-06	「依存キーの登録」のCore Data要件を明確にしました。
2009-03-04	「依存キーの登録」の章を更新しました。
2006-06-28	コード例を更新しました。
2005-07-07	willChangeValueForKey:メソッドを呼び出す前にオブジェクトを解放してはならない点を明確にしました。Javaはサポートされないことを明記しました。
2004-08-31	細かな誤字を訂正しました。
	手動キー値変更通知をネストする必要があることを明記しました。
2004-03-20	「依存キーの登録」（19 ページ）のソース例を修正しました。
2004-02-22	「キー値監視の登録」（11 ページ）のソース例を訂正しました。「キー値監視の実装の詳細」（23 ページ）を追加しました。
2003-10-15	『*Key-Value Observing*』の初版。