ブロックプログラミングトピック

Cocoa > Objective-C Language

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

はじめに

ブロックオブジェクトは、C言語レベルの構文によるランタイム機能です。標準Cの関数に似ていますが、実行可能なコードのほかに、自動（スタック）メモリまたはマネージド（ヒープ）メモリにバインドされている変数を含むことができます。したがって、ブロックでは、実行時の動作に影響を与える状態（データ）セットを維持できます。

ブロックを使用して関数を記述し、それをAPIに渡したり、必要に応じて格納したり、マルチスレッドで使用することもできます。ブロックは、コールバックとして特に便利です。それは、ブロックが、コールバックで実行するコードと、その実行中に必要なデータの両方を保持するからです。

ブロックは、Mac OS X v10.6のXcodeデベロッパツールに付属のGCCおよびClangで利用できます。ブロックは、Mac OS X v10.6以降、およびiOS 4.0以降で使用できます。ブロックのランタイムはオープンソースで、LLVMのcompiler-rtサブプロジェクトリポジトリから入手できます。ブロックは、「N1370: Apple's Extensions to C」（この拡張にはガベージコレクションも含まれる）として、C標準ワーキンググループに提案されています。Objective-CとC++は、どちらもC言語から派生しているため、ブロックは、これら3つのすべての言語（Objective-C++も含む）で動作するように設計されています（ブロックの構文は、この目標を反映しています）。

ブロックオブジェクトについて学習し、C、C++、またはObjective-Cからそれらを使用して、より効率的で保守性の高いプログラムを作成する方法を習得するには、この文書をお読みください。

この書類の構成

この文書は、次の章で構成されています。

- 「ブロック入門」（7 ページ）では、すぐにわかる実践的なブロックの紹介を行います。
- 「ブロックの概念」（11 ページ）では、ブロックの概念について紹介します。
- 「ブロックの宣言と作成」（13 ページ）では、ブロック変数の宣言方法とブロックの実装方法を示します。
- 「ブロックと変数」（15 ページ）では、ブロックと変数間のやり取りについて説明し、`__block` ストレージ型修飾子の定義について説明します。
- 「ブロックの使用」（19 ページ）では、さまざまな使用パターンを示します。

ブロック入門

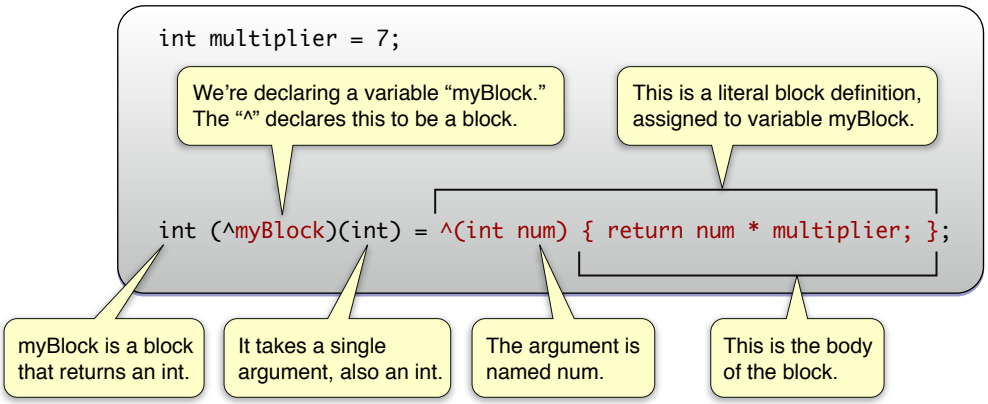
以降の各セクションでは、実践的な例を使用して、ブロックの入門として役立つ情報を提供します。

ブロックの宣言と使用

ブロック変数を宣言し、ブロックリテラルの始まりを表すには、^演算子を使用します。ブロックの本体は、この例に示すように、{}で囲まれます（C言語と同様に、;は式の終わりを表します）。

```
int multiplier = 7;
int (^myBlock)(int) = ^(int num) {
    return num * multiplier;
};
```

この例の説明を次の図に示します。



ブロックでは、ブロックが定義されているスコープ（有効範囲）に含まれている変数を利用できます。

ブロックを変数として宣言した場合には、ブロックを関数と同じように使用できます。

```
int multiplier = 7;
int (^myBlock)(int) = ^(int num) {
    return num * multiplier;
};

printf("%d", myBlock(3));
// "21"と出力する
```

ブロックの直接使用

多くの場合、ブロック変数を宣言する必要はありません。その代わりに、引数として必要になる場所にブロックリテラルをインラインで記述します。次の例ではqsort_b関数を使用しています。qsort_b は標準のqsort_r関数に似ていますが、最後の引数にブロックを取ります。

```
char *myCharacters[3] = { "TomJohn", "George", "Charles Condomine" };

qsort_b(myCharacters, 3, sizeof(char *), ^(const void *l, const void *r) {
    char *left = *(char **)l;
    char *right = *(char **)r;
    return strncmp(left, right, 1);
});

// myCharactersは{ "Charles Condomine", "George", TomJohn" }になっている
```

Cocoaでのブロックの使用

Cocoaフレームワークのいくつかのメソッドは、引数としてブロックを取ります。通常は、オブジェクトのコレクションに対して操作を実行するため、または操作の終了後にコールバックとして使用するためです。次の例に、NSArrayのsortedArrayUsingComparator:メソッドでブロックを使用する方法を示します。このメソッドは1つの引数（ブロック）を取ります。次に示すように、この例では、ブロックをNSComparatorローカル変数として定義しています。

```
NSArray *stringsArray = [NSArray arrayWithObjects:
                                  @"string 1",
                                  @"String 21",
                                  @"string 12",
                                  @"String 11",
                                  @"String 02", nil];

static NSStringCompareOptions comparisonOptions = NSCaseInsensitiveSearch |
NSNumericSearch |
        NSWidthInsensitiveSearch | NSForcedOrderingSearch;
NSLocale *currentLocale = [NSLocale currentLocale];

NSComparator finderSortBlock = ^(id string1, id string2) {

    NSRange string1Range = NSMakeRange(0, [string1 length]);
   return [string1 compare:string2 options:comparisonOptions range:string1Range
 locale:currentLocale];
};

NSArray *finderSortArray = [stringsArray
sortedArrayUsingComparator:finderSortBlock];
NSLog(@"finderSortArray: %@", finderSortArray);

/*
Output:
finderSortArray: (
    "string 1",
    "String 02",
```

```
    "String 11",
    "string 12",
    "String 21"
)
*/
```

__block変数

ブロックの強力な機能の1つは、同じレキシカルスコープ内の変数を変更できることです。__blockストレージ型修飾子を使用して、ブロックが変数を変更できることを指定します。「Cocoaでのブロックの使用」（8 ページ）の例に変更を加えて、次の例に示すように、ブロック変数を使用して、比較の結果、同じと判断された文字列の数をカウントできます。ここでは、ブロックを直接使用し、currentLocaleをこのブロック内の読み取り専用の変数として使用しています。

```
NSArray *stringsArray = [NSArray arrayWithObjects:
                         @"string 1",
                         @"String 21", // &lt;-
                         @"string 12",
                         @"String 11",
                         @"Strîng 21", // &lt;-
                         @"Striñg 21", // &lt;-
                         @"String 02", nil];

NSLocale *currentLocale = [NSLocale currentLocale];
__block NSUInteger orderedSameCount = 0;

NSArray *diacriticInsensitiveSortArray = [stringsArray
sortedArrayUsingComparator:^(id string1, id string2) {

    NSRange string1Range = NSMakeRange(0, [string1 length]);
    NSComparisonResult comparisonResult = [string1 compare:string2
options:NSDiacriticInsensitiveSearch range:string1Range locale:currentLocale];

    if (comparisonResult == NSOrderedSame) {
        orderedSameCount++;
    }
    return comparisonResult;
}];

NSLog(@"diacriticInsensitiveSortArray: %@", diacriticInsensitiveSortArray);
NSLog(@"orderedSameCount: %d", orderedSameCount);

/*
Output:

diacriticInsensitiveSortArray: (
    "String 02",
    "string 1",
    "String 11",
    "string 12",
    "String 21",
    "Str\U00eeng 21",
    "Stri\U00f1g 21"
)
orderedSameCount: 2
```

```
*/
```

これについては、「ブロックと変数」（15 ページ）で詳しく説明しています。

ブロックの概念

ブロックオブジェクトは、その場限りの関数の本体を、CやCから派生した言語（Objective-C、C++など）の式として作成する手段を提供します。ほかの言語や環境では、ブロックオブジェクトを「クロージャ」と呼ぶこともあります。ここでは、標準Cの用語のコードブロックと混同するおそれのない限り、通常は「ブロック」と呼びます。

ブロックの機能

ブロックは、以下のような特徴を持つ無名のインラインコードの集合体です。

- 関数と同様に、型付きの引数リストを持つ
- 推定または宣言された戻り値型を持つ
- ブロックの定義を含むレキシカルスコープの状態を把握できる
- 必要であれば、レキシカルスコープの状態を変更できる
- 変更できるかどうかは、同じレキシカルスコープ内で定義されているほかのブロックと共通である
- レキシカルスコープ（スタックフレーム）が破棄された後も、引き続きそのレキシカルスコープ内で定義されている状態を共有したり変更したりできる

ブロックをコピーしたり、遅延実行のために、ブロックをほかのスレッドに渡すこともできます（または、ブロック自身のスレッド内で、実行ループに渡すことができます）。コンパイラとランタイムは、ブロックのすべてのコピーが存続している間は、そのブロックから参照されるすべての変数を保持するように調整します。ブロックは純粋なCおよびC++で利用できますが、ブロックは常にObjective-Cのオブジェクトでもあります。

使用法

通常、ブロックは自己完結的な小さなコード部品です。このため、ブロックは、並列に実行されたり、1つのコレクション内の複数のアイテムを対象に実行される一連の作業をカプセル化する手段として、また、別の操作が終了したときのコールバックとして、特に有用です。

ブロックは、主に次の2つの理由から、従来のコールバック関数に代わる有用な手段になります。

1. ブロックを利用すると、メソッド実装のコンテキストで後から実行されるコードを、呼び出しの時点で記述できます。

   したがって、通常、ブロックはフレームワークメソッドのパラメータになります。

2. ブロックでは、ローカル変数にアクセスできます。

   操作の実行に必要なすべてのコンテキスト情報を具体化したデータ構造を要求するコールバックを使用する代わりに、単純にローカル変数に直接アクセスできます。

ブロックの宣言と作成

ブロックの参照の宣言

ブロック変数はブロックへの参照を保持します。関数へのポインタを宣言するために使用する構文と似た構文を使用して、ブロックを宣言します。ただし、*の代わりに^を使用します。ブロック型は、C言語のほかの型システムと完全に相互運用できます。以下はすべて有効なブロック変数の宣言です。

```
void (^blockReturningVoidWithVoidArgument)(void);
int (^blockReturningIntWithIntAndCharArguments)(int, char);
void (^arrayOfTenBlocksReturningVoidWithIntArgument[10])(int);
```

また、ブロックは可変長引数(...)もサポートします。引数を取らないブロックの場合は、引数リストにvoidを指定しなければなりません。

ブロックは完全にタイプセーフになるように設計されています。そのために、ブロックの使用、ブロックに渡されるパラメータ、および戻り値の代入の検証に使用する完全なメタデータセットをコンパイラに提供します。ブロックの参照は、任意の型のポインタにキャストできます。また、その逆も可能です。ただし、ポインタ演算子(*)によってブロックの参照をたどることはできません。ブロックのサイズをコンパイル時に計算することができないからです。

ブロックの型を作成することもできます。これは、一般に、特定のシグネチャを持つブロックを複数の場所で使用する場合のベストプラクティスと考えられています。

```
typedef float (^MyBlockType)(float, float);

MyBlockType myFirstBlock = // ... ;
MyBlockType mySecondBlock = // ... ;
```

ブロックの作成

ブロックリテラル式の始まりを表すには、^演算子を使用します。その後には、()で囲まれた引数リストが付くことがあります。ブロックの本体は、{}で囲まれています。次の例では、単純なブロックを宣言し、それをあらかじめ宣言しておいた変数（oneFrom）に代入しています。ここでは、ブロックの後に、Cのステートメントを終了する普通の;が付きます。

```
int (^oneFrom)(int);

oneFrom = ^(int anInt) {
    return anInt - 1;
};
```

ブロック式の戻り値を明示的に宣言しないと、そのブロックのコンテキストから自動的に推定されます。戻り値型が推定されて、パラメータリストが`void`の場合は、`(void)`パラメータリストも省略できます。**return**文が複数存在する場合は、（必要であれば、キャストを使用して）正確に一致させなければなりません。

グローバルなブロック

ファイルレベルでは、ブロックをグローバルなリテラルとして使用できます。

```
#import <stdio.h>

int GlobalInt = 0;
int (^getGlobalInt)(void) = ^{ return GlobalInt; };
```

ブロックと変数

ここでは、ブロックと変数間のやり取りについて、メモリ管理も含めて説明します。

変数のタイプ

ブロックオブジェクトのコード本体内では、変数は5つの異なる方法で扱うことができます。

関数の場合と同様に、次の3つの標準的な変数タイプを参照できます。

- グローバル変数（静的なローカル変数も含む）
- グローバル関数（技術的には変数ではない）
- スコープ内のローカル変数とパラメータ

ブロックは次の2つのタイプの変数もサポートします。

1. 関数レベルの`__block`変数。これらの変数は、ブロック（およびスコープ）内では可変です。また、参照しているブロックがヒープにコピーされている場合は保持されます
2. `const`のインポート

最後に、メソッド実装内では、ブロックはObjective-Cのインスタンス変数を参照できます（「オブジェクトとブロック変数」（17 ページ）を参照）。

以下の規則は、ブロック内で使用される変数に適用されます。

1. グローバル変数（レキシカルスコープ内に存在する静的変数も含む）はアクセス可能です。
2. ブロックに渡されたパラメータは（関数に渡されたパラメータと同様に）アクセス可能です。
3. レキシカルスコープに対してローカルなスタック上の（非スタティック）変数は、`const`変数として取得されます。

   これらの値は、プログラム内のブロック式の場所で取得されます。ネストしたブロックでは、この値は、それを含む最も内側のスコープから取得されます。
4. レキシカルスコープに対してローカルな変数が、`__block`ストレージ型修飾子付きで宣言されている場合は、参照渡しで提供されるため、可変です。

   すべての変更は、その変数が含まれる最も内側のレキシカルスコープ（同じレキシカルスコープ内で定義されているほかのすべてのブロックも含む）に反映されます。これらについては、「__blockストレージ型」（16 ページ）で詳しく説明します。

5. ブロックのレキシカルスコープ内で宣言されているローカル変数は、関数内のローカル変数とまったく同様に動作します。

   ブロックを呼び出すたびに、その変数の新しいコピーが作成されます。これらの変数は、そのブロック内に含まれるブロックで、今度はconstまたは参照渡しの変数として使用されます。

次の例に、静的でないローカル変数の使用を示します。

```
int x = 123;

void (^printXAndY)(int) = ^(int y) {

    printf("%d %d\n", x, y);
};

printXAndY(456); // "123 456"と出力する
```

次に示すように、このブロック内でxに新しい値を代入しようとすると、エラーが発生します。

```
int x = 123;

void (^printXAndY)(int) = ^(int y) {

    x = x + y; // エラー
    printf("%d %d\n", x, y);
};
```

ブロック内で変数を変更できるようにするには、__blockストレージ型修飾子を使用します（「__blockストレージ型」（16 ページ））。

__blockストレージ型

インポートされた変数を可変（読み書き可能）になるように指定できます。それには、__blockストレージ型修飾子を適用します。__blockストレージは、ローカル変数に対するregister、auto、およびstaticの各ストレージ型に似ていますが、これらとは互いに排他的です。

__block変数は、その変数のレキシカルスコープと、その変数のレキシカルスコープ内で宣言または作成されたすべてのブロックおよびブロックのコピーとの間で共有されるストレージ内に存在します。したがって、このストレージは、スタックフレーム内で宣言されているブロックのコピーが、フレームの終了後も存続する場合は（たとえば、後で実行するために、どこかのキューに入れられている場合）、そのスタックフレームが破棄された後も存続します。特定のレキシカルスコープ内の複数のブロックが、同時に1つの共有変数を使用できます。

最適化のために、ブロックストレージは、ブロック自身と同様に、スタック上に置かれます。Block_copyを使用してブロックがコピーされた場合（または、Objective-Cで、そのブロックにcopyが送信された場合）は、変数はヒープにコピーされます。したがって、*__block変数のアドレスは時間の経過とともに変化する可能性があります。*

__block変数には、さらに2つの制約があります。それは、可変長の配列でないことと、C99の可変長の配列を含む構造体でないことです。

次の例に、__block変数の使いかたを示します。

```
__block int x = 123; //  xはブロックストレージ内に存在する

void (^printXAndY)(int) = ^(int y) {

    x = x + y;
    printf("%d %d\n", x, y);
};
printXAndY(456); // "579,456"と出力する
// xは現在579
```

次の例には、さまざまなタイプの変数を含むブロック間のやり取りを示します。

```
extern NSInteger CounterGlobal;
static NSInteger CounterStatic;

{
    NSInteger localCounter = 42;
    __block char localCharacter;

    void (^aBlock)(void) = ^(void) {
        ++CounterGlobal;
        ++CounterStatic;
        CounterGlobal = localCounter; // localCounterはブロックの作成時に固定される
        localCharacter = 'a'; // ブロックを含むスコープ内のlocalCharacterを設定する
    };

    ++localCounter; // ブロックからは見えない
    localCharacter = 'b';

    aBlock(); // ブロックを実行する
    // localCharacterは現在'a'
}
```

オブジェクトとブロック変数

ブロックは、Objective-CやC++のオブジェクトと、その他のブロックを変数としてサポートします。

Objective-Cのオブジェクト

参照カウント環境では、ブロック内でObjective-Cのオブジェクトを参照すると、デフォルトでそれが保持されます。これは、そのオブジェクトのインスタンス変数を単純に参照した場合にも当てはまります。ただし、__blockストレージ型修飾子が付加されたオブジェクト変数は保持されません。

注： ガベージコレクション環境では、変数に__weak修飾子と__block修飾子の両方を適用した場合、ブロックはその変数が存続し続けることを保証しません。

メソッドの実装内でブロックを使用する場合は、オブジェクトのインスタンス変数のメモリ管理規則はさらに厳しくなります。

- 参照渡しのインスタンス変数にアクセスすると、selfは保持されます。

- 値渡しのインスタンス変数にアクセスすると、その変数は保持されます。

次の例に、この2つの異なる状況を示します。

```
dispatch_async(queue, ^{
    // instanceVariableは参照渡しで使われるため、selfは保持される
    doSomethingWithObject(instanceVariable);
});


id localVariable = instanceVariable;
dispatch_async(queue, ^{
    // localVariableは値渡しで使うため、localVariableは保持される（selfではない）
    doSomethingWithObject(localVariable);
});
```

C++のオブジェクト

一般に、ブロック内ではC++のオブジェクトを使用できます。メンバ関数内では、メンバ変数や関数への参照は、暗黙的にインポートされるthisポインタを利用して行われるため、可変に見えます。ブロックがコピーされる場合は、次の2点が適用されます。

- スタックベースのC++オブジェクトになるはずのものに__blockストレージクラスを指定した場合は、通常のcopyコンストラクタが使用されます。
- ブロック内からスタックベースのその他のC++オブジェクトを使用する場合は、そのオブジェクトはconst copyコンストラクタを持っていなければなりません。C++オブジェクトは、そのコンストラクタを使用してコピーされます。

ブロック

ブロックをコピーするとき、そのブロック内からほかのブロックへの参照があれば、それらは必要に応じてコピーされます（トップからツリー全体がコピーされる場合もあります）。ブロック変数が存在し、ブロック内からそのブロックを参照すると、そのブロックがコピーされます。

スタックベースのブロックをコピーした場合は、新しいブロックが作成されます。ただし、ヒープベースのブロックをコピーした場合は、単にそのブロックの保持カウントがインクリメントされて、その値がコピー関数またはメソッドの戻り値として返ります。

ブロックの使用

ブロックの呼び出し

ブロックを変数として宣言した場合は、次の2つの例に示すように、関数と同じように使用できます。

```
int (^oneFrom)(int) = ^(int anInt) {
    return anInt - 1;
};

printf("1 from 10 is %d", oneFrom(10));
// "1 from 10 is 9"と出力する

float (^distanceTraveled) (float, float, float) =
                          ^(float startingSpeed, float acceleration, float time)
 {

    float distance = (startingSpeed * time) + (0.5 * acceleration * time * time);
    return distance;
};

float howFar = distanceTraveled(0.0, 9.8, 1.0);
// howFar = 4.9
```

一方、ブロックを関数やメソッドの引数として渡すこともしばしばあります。このような場合は、通常、ブロックを「インライン」で作成します。

関数の引数としてのブロックの使用

ほかの引数と同様に、ブロックを関数の引数として渡すことができます。多くの場合、ブロック変数を宣言する必要はありません。その代わりに、引数として必要になる場所にブロックリテラルをインラインで記述します。次の例ではqsort_b関数を使用しています。qsort_bは標準のqsort_r関数に似ていますが、最後の引数にブロックを取ります。

```
char *myCharacters[3] = { "TomJohn", "George", "Charles Condomine" };

qsort_b(myCharacters, 3, sizeof(char *), ^(const void *l, const void *r) {
    char *left = *(char **)l;
    char *right = *(char **)r;
    return strncmp(left, right, 1);
});
// ブロックの実装は"}"で終わる

// myCharactersは{ "Charles Condomine", "George", TomJohn" }になっている
```

ブロックは、関数の引数リスト内に含まれています。

次の例では、dispatch_apply関数でブロックを使用する方法を示します。dispatch_applyは次のように宣言されています。

```
void dispatch_apply(size_t iterations, dispatch_queue_t queue, void
(^block)(size_t));
```

この関数は、複数回の呼び出しのためにディスパッチキューにブロックを送信します。この関数は3つの引数を取ります。1番目の引数は、繰り返しの実行回数を指定します。2番目の引数は、ブロックの送信先となるキューを指定します。3番目の引数はブロック自身です。このブロックは1つの引数（繰り返しの現在のインデクス）を取ります。

dispatch_applyを使用して、繰り返しの回数を出力できます。それを以下に示します。

```
#include <dispatch/dispatch.h>
size_t count = 10;
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_apply(count, queue, ^(size_t i) {
    printf("%u\n", i);
});
```

メソッドの引数としてのブロックの使用

Cocoaには、ブロックを使用したメソッドがたくさんあります。ほかの引数と同様に、ブロックをメソッドの引数として渡すことができます。

次の例は、配列内の最初の5つの要素のうち、与えられたフィルタセットに現れるもののインデックス（添字）を決定します。

```
NSArray *array = [NSArray arrayWithObjects: @"A", @"B", @"C", @"A", @"B",
@"Z",@"G", @"are", @"Q", nil];
NSSet *filterSet = [NSSet setWithObjects: @"A", @"Z", @"Q", nil];

BOOL (^test)(id obj, NSUInteger idx, BOOL *stop);

test = ^ (id obj, NSUInteger idx, BOOL *stop) {

    if (idx < 5) {
        if ([filterSet containsObject: obj]) {
            return YES;
        }
    }
    return NO;
};

NSIndexSet *indexes = [array indexesOfObjectsPassingTest:test];

NSLog(@"indexes: %@", indexes);

/*
Output:
```

```
indexes: <NSIndexSet: 0x10236f0>[number of indexes: 2 (in 2 ranges), indexes:
(0 3)]
*/
```

次の例は、NSSetオブジェクトに、ローカル変数で指定した単語が含まれるかどうかを判断し、含まれる場合は、別のローカル変数(found)の値をYESに設定して、検索を中止します。foundも__block変数として宣言しています。このブロックはインラインで定義しています。

```
__block BOOL found = NO;
NSSet *aSet = [NSSet setWithObjects: @"Alpha", @"Beta", @"Gamma", @"X", nil];
NSString *string = @"gamma";

[aSet enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
    if ([obj localizedCaseInsensitiveCompare:string] == NSOrderedSame) {
        *stop = YES;
        found = YES;
    }
}];

// この時点で found == YES
```

ブロックのコピー

通常は、ブロックをコピー（または保持）する必要はありません。ブロックの宣言を含むスコープが破棄された後も、そのブロックを使用する可能性がある場合にのみ、ブロックのコピーを作成する必要があります。ブロックをコピーすると、ブロックはヒープに移動します。

以下のC関数を使用して、ブロックのコピーと解放ができます。

```
Block_copy();
Block_release();
```

Objective-Cを使用している場合は、ブロックにcopy、retain、release（またはautorelease）の各メッセージを送信できます。

メモリリークを避けるために、Block_copy()とBlock_release()が必ず同数になるようにしなければなりません。ガベージコレクション環境でなければ、release（またはautorelease）を使用して、copyまたはretainのバランスを取る必要があります。

避けるべきパターン

ブロックリテラル（^{ ... }）は、ブロックを表す*スタック上のローカルな*データ構造のアドレスです。したがって、このスタック上のローカルなデータ構造のスコープは、それを含む複合文です。したがって、以下の例に示すようなパターンは*避けるべきです*。

```
void dontDoThis() {
    void (^blockArray[3])(void);  // 3つのブロックの参照の配列

    for (int i = 0; i < 3; ++i) {
        blockArray[i] = ^{ printf("hello, %d\n", i); };
        // 間違い：このブロックリテラルのスコープは"for"ループ
```

```
    }
}

void dontDoThisEither() {
    void (^block)(void);

    int i = random():
    if (i > 1000) {
        block = ^{ printf("got i at: %d\n", i); };
        // 間違い：このブロックリテラルのスコープは"then"節
    }
    // ...
}
```

デバッグ

ブレークポイントを設定して、ブロック内を1ステップずつ実行できます。次の例に示すように、`invoke-block`を使用して、GDBセッション内からブロックを呼び出すことができます。

```
$ invoke-block myBlock 10 20
```

Cの文字列を渡したい場合は、引用符で囲む必要があります。たとえば、`this string`を`doSomethingWithString`ブロックに渡すには、次のように記述します。

```
$ invoke-block doSomethingWithString "\"this string\""
```

書類の改訂履歴

この表は「*ブロックプログラミングトピック*」の改訂履歴です。

日付	メモ
2011-03-08	誤字を訂正しました。
2010-07-08	誤字を訂正しました。
2010-03-14	iOS 4.0用に更新しました。
2009-10-19	メモリ管理と型推論についての説明を明確にしました。
2009-05-28	Cプログラミング言語のブロック機能について説明した新規文書。

書類の改訂履歴