Objective-Cを使ってみる

4J 広松 悠介

平成 16 年 11 月 1 日

1 Objective-Cとは

Objective-C は C 言語に Smalltalk 風のオブジェクト指向の機能を拡張した言語です。C++が C 言語とほぼ変わらない速度が出るように設計されたのに対して、Objective-C はオブジェクト指向しやすいように設計された言語といえるでしょう。ただ、C++の方が明らかにメジャーなのですが…

C++の方がメジャーな今、Objective-C をやる意味は、Smalltalk 風のオブジェクト指向がどんなものかを理解したり、MacOS X の Cocoa 環境や gnu の Window Maker で開発を行うために利用したりするくらいです。

1.1 Objective-C のコンパイル環境

Objective-C のヘッダの拡張子は.h、ソースの拡張子は.m です。Objective-C のソースコードは gcc でコンパイルできるので、コンパイル用の環境を整えるだけなら無料でできます。

2 Objective-C のクラス

2.1 クラスの宣言

クラスの宣言は、@interface で始まり、@end で終わります。”@interface クラス名 : 親クラス”と書いた後の {} の中にメンバ変数を宣言し、} から@end までの間に、メンバ関数を宣言します。
具体的には、

```
//Round クラスを基底クラスとするクラス Disk の宣言
@interface Disk : Round
{
  //メンバ変数の宣言
  int diskWeight;
  int diskThickness;
}
//メンバ関数の宣言

//int 型の引数を 1 つ受け取るメンバ関数 setName:の宣言
-(void)setWeight:(int)weight;

//int, int, id 型の引数を受けとり、
//id 型の返値を返すメンバ関数 init: withThickness name:の宣言。
-(void)init:(int)weight withThickness:(int)thickness;

//返値が int 型であるクラス関数 diskCount の宣言
```

```
+(int)dishCount;
@end
```

2.1.1 注意事項

Objective-C では多重継承ができません。また、基底クラスはインスタンスの生成などの重要な関数を定義しなければならず、その実装は複雑です。MacOS X の Cocoa 環境の場合、API に基底クラスとして NSObject が定義されているので、よほどのことでない限り、クラスは NSObject を継承して定義した方がいいです。

2.2 関数の定義

クラスのメンバ関数の定義は@implementation から@end の間で行います。実は、クラス宣言を行った時に宣言していなかったメンバ関数を定義することもできてしまいます。そのような関数は見た目存在しないだけで、実際には外側からでも呼び出すことができます。

```
//クラス Disk の関数定義
@implementation Disk

//ヘッダで宣言していた関数の実装
-(void)setWeight:(int)weight
{
  diskWeight = weight;
}

-(void)init:(int)weight withThickness:(int)thickness
{
diskWeight = weight;
diskThickness = thickness;

DISK_COUNT++;
}

+(int)dishCount
{
  return DISK_COUNT;
}


//ヘッダで宣言されていない関数の実装
-(void)play
{
...
}
@end
```

2.3 クラスの型

全てのクラスインスタンスは、クラスのポインタとして扱えるほかにも、id 型として扱うことができます。つまり、id 型にはあらゆるクラスのインスタンスを格納することができるのです。基本的に、クラスのインスタンスは id 型として扱った方が楽です。

```
Disk* dish;
Ball* ball;
id object;
...
//Dish クラスのインスタンスを受け取る
dish = GetDish();
//Ball クラスのインスタンスを受け取る
ball = GetBall();

object = dish;  //OK
ball = dish;      //ダメ

object = ball;  //OK
dish = ball;  //ダメ
```

2.4 メンバ関数呼び出し

Objective-C のメンバ関数の呼び出しは、[インスタンス 関数] と書きます。関数の呼び出しは、インスタンスには見かけ上 (実際にも) 存在しない関数でも呼ぶことができてしまいます。もしメンバ関数が存在するのなら、その関数を呼びだしますし、存在しなければ例外を出します。これと同時に、全てのクラスインスタンスは id 型として扱えるため、Objective-C では、継承元が同じでないクラスであっても、実装されているメンバ関数名が同じであれば、正常に関数呼び出しが行える のです。

```
id dish;
...
//メンバ関数の呼び出し
[dish init:12 withThickness:3];
//クラス関数の呼び出し
[Dish dishCount];
```

self に対してメンバ関数を呼び出すことで、メンバ関数を呼び出している最中に、さらに自分のメンバ関数を呼び出せます。

```
-(void)init:(int)weight withThickness:(int)thickness
{
  diskThickness = thickness;
  //自分のメンバ関数を呼び出す
  [self setWeight:weight];
}
```

2.5 関数のオーバーライド

親クラスで定義していた関数と同じ名前の関数を子クラスでも定義すると、親の関数は隠ぺいされて呼び出されなくなってしまいます。ただし、子クラスのメンバ関数内で super に対してメンバ関数を呼び出せば、親の関数を呼び出すことができます。

```
@implementation Cup
...
-(void)override
{
  ...  //このクラスでの処理
  [super override];  //親の override 関数も呼び出す
}
...
@end
```

2.6 クラスインスタンスの使い方

特定のクラスのインスタンス生成は、生成用のクラス関数を呼び出すことで行います。MacOS X の Cocoa 環境なら、NSObject クラスにある alloc 関数がインスタンスを生成するための関数です。
例えば、Disk クラスのインスタンスを生成したい場合、以下のように書きます。

```
id disk = [Disk alloc];
```

Objective-C の場合、インスタンス生成時に呼ばれる関数 (コンストラクタ) は存在しません。なので、生成してから別の関数で初期化を行う必要があります。

2.7 その他の機能

Objective-C に搭載されているオブジェクト指向の機能はまだまだあるのですが、基本的な機能はこれまであげた程度です。その他の機能には以下のようなものがあります。

2.7.1 プロトコル

クラスが準拠しなければならないメンバ関数の宣言で、Java でいう interface、C++でいう仮想クラスです。しかし、Objective-C では、どんなクラスも id 型として扱えて、あらゆるメンバ関数を (動くかどうかはともかく) 呼び出せるので、ほとんど形式的な機能です。

2.7.2 セレクタ

呼び出すメンバ関数名を格納する型で、これのさす名前のメンバ関数を呼び出すこともできます。関数へのポインタと似ているのですが、あくまでこの型は『メンバ関数名』を格納しているのです。そのため、名前が同じで実装が異なる関数を持つインスタンスどうしで、同じセレクタを使ってメンバ関数を呼び出すと、別の処理になるわけです。

2.7.3 カテゴリ

普通、クラスの機能を拡張しようと思ったら、クラスを継承して新たに実装を記述するわけですが、カテゴリの機能を使うと、メンバ関数だけなら既存のクラスに追加することができます。もちろん、そのクラスの子クラスにも影響は及びます。ただし、拡張しようとしたクラスに既に同名の関数が定義されていた場合、カテゴリの方が優先されて、しかも元の関数は呼び出せなくなってしまいます。

予約後の名前	説明
@try...	例外を投げる処理をくくるのに使う
@catch(...)...	@try の後に記述。受け取った例外に対する処理を行う。
@finally...	@try の後に記述。@catch が受け取らない例外をここで処理できる。

```
Cup *cup = [[Cup alloc] init];

@try{
  [cup fill];
}
@catch(NSException *exception){
  //NSExeption クラスが例外で返ってきた時は例外ログを表示
  NSLog(@"main: Caught %@: %@",
    [exception name], [exception reason]);
}
//その他の例外であった場合、cup を破棄する以外は何もしない
@finally{
  [cup release];
}
```

図 1: 例外処理の例

3　クラス以外の拡張機能

Objective-C には、オブジェクト指向をサポートするための機能の他にも、いくつか機能が追加されています。

3.1　ファイルのインクルード

C 言語では、ファイルは#include でインクルードしていたのですが、Objective-C には、ファイルのインクルードに#import も使えます。使い方は#include と同じです。#import を使ってインクルードしたファイルは、以降に#import でインクルードしようとしても、読み込みません。なので、わざわざヘッダファイルを書く時、わざわざマクロで囲う必要がないわけです。

3.2　BOOL 型

Objective-C では、BOOL 型が定義されています。この型は、真 (YES=1) か偽 (NO=0) のみを値として持つものです。C++にも、bool 型という、同じような機能を持つ型が定義されています。

3.3　例外処理

何かしら、思わぬエラーが発生した時、それを上のレベルに知らせ、エラー時の処理を行ってもらうための機能で、C++にも、同じような機能があります。普通にプログラムを書いている限り、お世話になることは少ないでしょう。

```
-(void)criticalMethod
{
  //このオブジェクトのクリティカルな部分へのアクセスは
  //このスレッドだけであることを保証する
  @synchronized(self) {
    //クリティカルな処理
    ...
  }
}
```

図 2: synchronized の例

3.4 スレッドの同期

Objective-C には複数のスレッドと同期をとるための機能@synchronized があります。@synchronized の {} に囲われた処理は、() 内に書いた id 型などの変数を他のスレッドの@synchronized が扱っていない時のみ実行できます。

3.5 C++と Objective-C の両用

gcc は、拡張子が.mm のファイルは C++と Objective-C の両方の機能を使っているものとしてコンパイルします。なので、C++と Objective-C の両方を同時に使うことができるのです。ただし、C++のクラスと Objective-C のクラスは互換性がないので、互いに継承ができないなどの制約があります。C++に比べ、Objective-C は動作が遅いので、速度が欲しい部分を C++、その他の部分を Objective-C というように、うまく使いこなしていくべきでしょう。