Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

iOS における ARC と非 ARC の同一ソースコード上での共存手法

平野　聡[†1]　官　森林[†2]　名嘉村　盛和[†2]

スマートデバイス iPhone/iPad において，iOS 5 からオブジェクトの自動メモリ管理(Automatic Reference Counting, ARC)が提供されたが，従来の retain/release による手動メモリ管理（非 ARC）のソースコードと互換性がない．ARC と非 ARC の両方に対応するライブラリやフレームワークの開発のために，同一のソースコードを ARC と非 ARC の両方に対応させる 6 つの手法を提示し，5 つの評価基準に基づいて比較した．応用として，その中で最も評価が高かった self による無害化の手法を Apache Thrift に適用した．Thrift は基幹的通信基盤として Facebook，Evernote，Cassandra 等で広く使用されているクロスプラットホームの通信ミドルウェアである．本手法による改良は Thrift プロジェクトにおいて有効性確認後に採用され，多くのプロジェクトで使用されつつある．

Coexisting Automatic and Non-Automatic Object Managements within a source file on iOS

HIRANO Satoshi [†1]
GUAN Senlin[†2]　　NAKAMURA Morikazu[†2]

iOS's new memory management scheme, Automatic Reference Counting (ARC) lacks source code level compatibility with its conventional memory management scheme by retain/release methods. We developed and evaluated six methods for coexisting both ARC and non-ARC within the same source code. We applied the best one, stub-by-self method to Apache Thrift, a popular cross-platform RPC technology used in many projects such as Facebook and Evernote. Since it was proven to be effective in the Thrift project, it was merged to the source tree and has become used widely in iOS based projects.

†1 産業技術総合研究所情報技術研究部門 National Agency of Advanced Industrial Science and Technology
†2 琉球大学 工学部情報工学科 The Department of Information Engineering, University of the Ryukyus

1. はじめに

iPhone/iPad や Android[a]等のスマートデバイスが広く普及し，関連する技術の重要性が増している．iOS の場合，プログラム（app）は 50 万種類を超え，モバイル OS における iOS のシェアは 54%である[1]. iOS とその開発環境 Xcode を用いて開発を行う開発者は増え続けており，初めてのプログラミングが iOS である人も多い．

iOS のプログラムでは retain/release メソッドによりプログラマがオブジェクトの参照カウントを増減するメモリ管理（非 ARC）が用いられてきた[2]．2011 年，iOS 5 において，参照カウントの管理をコンパイラが自動化する安全な自動メモリ管理(Automatic Reference Counting, ARC)が提供された[3][4]．ARC では retain/release は使わなくなり，従来の非 ARC のソースコードと互換性がない（2 章）．この互換性の喪失は iOS の多くの開発者にとって，小さいが困る，例えて言えば「足の裏の小さなトゲ」のような存在である．特にライブラリの開発者は ARC と非 ARC の両方に対応しつつ，徐々に安全な ARC に比重を移していきたいと考えている（3 章）．すぐに，retain/release を削除し ARC に変換するマクロや簡単なスクリプト，#ifdef で retain あるなしの行を切り分ける方法を思いつくが，コードの可読性や実装性に問題がある．本稿の目的は，この問題を解決するため，「同一のソースコードを ARC と非 ARC に対応させる，いくつかのスマートで簡便な手法」を提示することである．考案した 6 つの手法[b]を可読性等の 5 つの評価基準（4 章）によって比較する（5 章）．最も評価の高かった無害化による手法をクロスプラットホームの RPC 技術である Apache Thrift[5]に適応した経験[6]について紹介する（6 章）．考察として，読者に手法の選択の指針を示し，他技術との比較を行う（7 章）．

2. Objective-C 言語におけるメモリ管理の概要

iOS 上のプログラムはオブジェクト指向言語 Objective-C[7]により記述する．オブジェクト（インスタンス）person に文字列の引数@"Hello, world!"と共にメッセージ greeting:を送り（greeting:メソッドを呼び出す），返値 msg を受け取る操作は，次の上の行のように記述する．下の行のようにメッセージ送信を入れ子にしてもよい．

```
msg = [person greeting:@"Hello, world!"];
[view setTitle:[person greeting:@"You look like Smalltalk."]];
```

a) iPhone, iPad, iOS, Xcode は Apple 社の商標，Android は Google 社の商標である．
b Mac OS においても適用可能である．

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

2.1 伝統的な非 ARC による Objective-C プログラムの記述例

複数のオブジェクトから参照されるオブジェクトは，それを使用中の全てのオブジェクトからの参照がなくなるまで寿命を保たねばならない．オブジェクトの寿命の管理は，オブジェクトを生成した主体が廃棄にも責任を持つ「Ownership Rule」の考え方に基づき，プログラマが手動で参照カウントを操作する retain/release 等のメソッドを用いて行う手動メモリ管理方式により行われていた(非 ARC) [2] ．retain は参照カウントを+1 し, release は参照カウントを-1 する. Objective-C 2.0 において Mac OS ではガベージコレクション（GC）が導入されたが，リソースに制限のある iOS では導入されなかった．

図 1 に非 ARC のコードの例を示す. これは MyClass クラスのあるメソッド methodA と dealloc メソッド（Java の finalize に相当）である．Name，Person と Group はそれぞれクラス，aName，person と group はそれらのオブジェクト（インスタンス）である．

```
1 - (void) methodA:(Name *)aName {
2     Person *person = [[Person alloc] init];
3     [userManager addPerson:person];
4     [person release];
5
6     Group *group = [[[Group alloc] init] autorelease];
7     [group method2];
8
9     ivar = [aName retain];
10 }
11
12 - (void) dealloc {
13     [ivar release];
14     [super dealloc];
15 }
```

図 1　非 ARC の基本文例
Figure 1 A basic example of Non-ARC.

2 行目で Person クラスに alloc メッセージを送ることでオブジェクトが生成され，参照カウントが初期値 1 に設定される．3 行目で person オブジェクトが使用され，4 行目の release により参照カウントが 1 減じられる．（3 行目で呼ばれた先で person が retain されておらず）参照カウントが 0 になると，person オブジェクトは廃棄され，ゴミとしてメモリが回収される．その際，そのオブジェクトの dealloc メソッドが呼び出され終了処理を行う．この release を書き忘れるとメモリリークの原因となる．逆に，込み入ったコード内で release を必要以上の回数通ると，後で存在しないオブジェクトが参照され原因追及が難しいエラーとなる．

それを防ぐため，所有時間が短いオブジェクトには予め 6 行目にある autorelease を適用し release を省略することもある．これは autorelease pool を使用する．

9 行目では引数として渡された文字列 aName をインスタンス変数 ivar に保存している. その際, 自分が保持している限りは aName がゴミとして回収されないように, aName に retain メッセージを送り参照カウントを+1 している．

12 行目の dealloc メソッドでは，終了処理としてインスタンス変数 ivar に release メッセージを送り参照カウントを 1 減じている．もし，他のオブジェクトで retain されていなければ，ivar(aName)はこのタイミングで廃棄される．14 行目はスーパークラスの dealloc の呼び出しである．

この手動メモリ管理は複雑になりやすく，習熟者でもミスを完全になくすことは困難である．更に，Objective-C はインスタンス変数を obj.ivar のようなドット記法でアクセスしたり，Key-Value コーディングを可能としたりするプロパティという仕組みを有する．このプロパティと通常のインスタンス変数の参照カウントの管理方法が異なるため，なかなか理解が難しい．

2.1 ARC による Objective-C の記述例

ARC（Automatic Reference Counting）はこのような苦労から開発者を解放する．iOS 5 と同時に公開された Xcode 開発環境に含まれる LLVM コンパイラ[8][9]は，ARC 対応のソースコードをコンパイルする際に, 生成するバイナリコードに前節で説明した retain/release/autorelease 等に相当するコードを自動的に挿入する[3][4].

コンパイラはソースコード毎に ARC モードと非 ARC モードのどちらかでコンパイルを行う. それぞれ-fobjc-arc と-fno-objc-arc のコマンドラインオプションで指定する. デフォルトは ARC である. ひとつの app の中で ARC コンパイルされたオブジェクトファイル(.o)と非 ARC のオブジェクトファイルを一緒にリンクすることが可能である. Xcode 内からは，設定ダイアログで各ソースコード毎に-fno-objc-arc オプションを指定することができる．

ARC モードでコンパイルするソースコード内では retain/release といった参照カウントの操作を行うメソッドは使えない．retain/release を書いたり，これらのメソッドを再定義しようとするとコンパイルエラーになる. そのため, ARC を利用するには, これまで書きためてきたソースコードから retain/release 等の使用を削除する編集作業（ARC 化）を行わなければならない．ARC 機能は iOS 5.0 以降で使用可能であり（一部の機能は iOS 4.3 以降），ARC コンパイルした app はそれ以前の iOS を搭載する iPhone や iPod touch では動作しなくなる．メモリ管理のミスが出にくい ARC に移行するか非 ARC に留まるかは app の販売戦略の大きな岐路である. 開発者は ARC 化を決意すると，Xcode の助けを借りてこの書き換えを行う．LLVM コンパイラの提供者あるいは Apple がコンパイラの仕様を変更して retain 等を無視してくれれば何も

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

しなくても良いように思われるが，後述の理由でそうなってはいない．

以下に ARC コンパイルで無効にされた（エラーになる）メソッド等を示す．(id)はオブジェクトを返すことを，(void)は値を返さないことを表す．

- - (id)retain;　（参照カウンタを +1 し，self を返すメソッド）
- - (void)release;（参照カウンタを -1 し，0 になったら廃棄するメソッド）
- - (void)autorelease;　（イベントループ終了後に release するメソッド）
- dealloc 中の[super dealloc]　（上位クラスの終了処理を呼ぶ）
- NSAutoreleasePool　（一時的なオブジェクトプールを作成する）
  - ✧　ARC では@autoreleasepool ディレクティブを使用する
- C のポインタと Objective-C オブジェクトのキャスト
  - ✧　ARC では__bridge アノテーションもしくは__bridge_transfer アノテーションを挿入し，所有権の移転・不移転を明確化する．

図 1 を ARC 対応に書き換えたリストを図 2 に示す．retain 等はなく，dealloc はもはや不要である．片付けのために書いても良いが，インスタンス変数の release とスーパークラスの dealloc の呼び出しは書くとエラーになる．

```
1 - (void) methodA:(Name *)aName {
2     Person *person = [[Person alloc] init];
3     [userManager addPerson:person];
4
5     Group *group = [[Group alloc] init];
6     [group method2];
7
8     ivar = aName;
9 }
10
11 // dealloc は不要
```

図 2　ARC の基本文例
Figure 2 A basic example of ARC.

3. 同一ソースコード上での ARC と非 ARC の共存の必要性

前述のように，通常の app はソースファイル毎に ARC もしくは非 ARC で記述し，それぞれのコンパイルモードでコンパイルし，一緒にリンクすればよい．では，ひとつのソースファイルが ARC と非 ARC の両方に対応しなければならないのは，どのような場合であろうか？

3.1 ライブラリの場合

開発者が作成するプロダクトとして app の他にライブラリがある（フレームワークも含むとする）．それらは，多くの app で共通に使用する機能を抜き出して実装したコード群であり，GUI のコンポーネントや画像処理等多くがインターネット上で配布されている．app には ARC に移行する物と非 ARC に留まる物の両方があるため，ライブラリは両方をサポートしつつ，徐々により安全な ARC に比重を移してゆくことが望ましい．ライブラリとして配布されているソースコードは，それを利用する開発者よって Xcode に取り込まれ，コンパイルされる．ライブラリの提供方法としては，1）非 ARC のソースコードと ARC のソースコードの両方を提供する方法と，2）現在多くのライブラリがそうであるように，非 ARC のみのコードを提供し，前述の-fno-objc-arc フラグを設定して非 ARC コンパイルしてもらう方法がある．

1)は保守量が 2 倍に増えるため現実的ではない．2)は，開発者に，Xcode の app のプロジェクトにライブラリのソースコードを取り込んだ後，ライブラリ由来の全てのファイルに-fno-objc-arc オプションを設定してもらう必要がある．ファイル数が多いと手間がかかり初心者にも敷居が高い．　（Make は使われない）

3)として，ライブラリのソースコードを Xcode 上にドラッグして，そのままビルドボタンひとつで（ARC でも非 ARC でも）コンパイルできれば最も望ましい．ソースコードが ARC と非 ARC の両方に対応していればそれが可能である．初心者にも容易であり，ライブラリ提供側としてもサポートに時間を取られない．

3.2 コード生成を行うツールの場合

Thrift はインターフェース定義言語（IDL）による定義ファイルから Objective-C のソースコードを生成する．著者らは thrift コンパイラを使う際に以下のトラブルに遭遇した．thrift コンパイラの実行は，Xcode のビルドプロセス中に記述可能な外部スクリプト呼び出し機能を使用する．生成された（非 ARC の）Objective-C のソースコードは Xcode によって自動的にコンパイルされる．そして開発中の app は ARC 対応であるため，生成されたソースコードに含まれる retain/release が原因で大量のコンパイルエラーが発生した．生成されたソースコードに対して-fno-objc-arc を指定する設定場所はなく途方に暮れた．これは他のソースコード生成ツールにも共通する．

特に Thrift のような通信ミドルウェアにとって安全性の高い ARC に移行することは重要である．しかし，非 ARC との互換性も捨てられないというジレンマがある．

そこで，何らかの手段で，生成されたソースコードを非 ARC でも ARC でも正しくコンパイル可能にする方法を考えざるを得なくなった．

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

4. 共存手法の評価基準

目標：ひとつのソースコードを ARC モードでも非 ARC モードでも正しくコンパイル可能とすること．分かりやすくミスを発生しにくいこと．

これは３枚の絵に例えられる．両目で見ると女性が花を持っている美しい絵に見える．右目で見ると女性の絵に，左目で見ると花の絵に見える．即ち，何らかの表現方法で ARC・非 ARC 共存にしたソースコード（両目の絵）を，非 ARC コンパイルする際は図 1 に，ARC コンパイルする際は図 2 に見えるようにしたい．

その「何らかの表現方法」は，特殊な言語拡張や複雑怪奇なマクロ呼び出しではなく，つぎはぎのない自然な Objective-C プログラムに見えることが望ましい．

ソースコードの可読性，利便性等の評価基準を以下のように設定する．

正しさ

ARC，非 ARC とも期待と違う動作を行わないことが最も重要である．例えば，非 ARC のソースコードを ARC に変換するスクリプトの場合，Objective-C の文法と意味を解釈し正しいコードを生成することが必要である．

可読性

上記のようにソースコードの読みやすさも重要である．例えば，多くの#ifdef で埋まっているソースコードは読みにくく，発見しにくいエラーの原因となる．ARC として，あるいは非 ARC として，ひとつの流れとして読み書きできるコードが望ましい．

実装の容易性・利便性

実装が容易で，少しの改造で導入が可能であることは重要である．この基準は開発者にとっての利便性も含む．Thrift もそうだが，オープンソースのプロジェクトでは，スマートで改造量が小さくないと合意が難しく採用に至らない．

ARC・非 ARC の差異の判別の容易性（判別容易性）

LLVM コンパイラの提供者は GC 用のコンパイルの場合には単に retain を無視する仕様にした．ARC の場合もそうせずに retain の記述をエラーにしたのは，app の開発者が ARC と非 ARC との差異を明確に認識して開発を行う必要があるためであろう．ARC ではオブジェクト参照の閉ループはゴミ集めされないため，閉ループの一部に弱参照を示す__weak（あるいは__unsafe_unretained）アノテーションを使用する，また，C 構造体と Objective-C オブジェクトをキャストする際に__bridge アノテーションを使用してオブジェクトの所有権の移転を明記するといった作業を行わなければならない[3]．従って，可読性と同時に，開発者が ARC と非 ARC の差異を容易に判別可能であることが重要である．例えば，非 ARC とそっくりに見えて，実は ARC であるという見かけであると，後日別の開発者が手を入れる際に思わぬ失敗をすることになる[c].

実行速度

スマートデバイスの app がキビキビ動くにはタイトループ内のオーバーヘッドをできるだけ減らしたい．非 ARC と ARC とを共存化することで，実行時のオーバーヘッドが顕在化するほどでは困る．

5. 同一ソースコード上での ARC，非 ARC 共存化手法

本章では著者らが検討した順に 6 つの手法を説明し，評価基準によって評価する．

(1) ARC 変換スクリプト

[[[Group alloc] init] autorelease]という文を [[Group alloc] init]に変換するには，autorelease を消すだけでなく入り組んだ括弧を外す必要があるため，文脈に依存した処理が必要である．プリプロセッサで autorelease を空文字列に置換する単純なマクロでは対応できない. retain はメソッドだけではなくプロパティの属性でも使用し，これは ARC でも必要であるため，これもマクロによる単純な置き換えはできない．

例えば，ARC 変換スクリプト Super Objective-C Pre-Compiler(socpc)を作成し，app のビルド中に非 ARC から ARC に変換することが考えられる. これは Objective-C の文法，閉ループ，オブジェクトの所有権の移動を理解するコンパイラを作るに等しい．このスクリプトは大きく，挙動が完全に正しい事を検証することは難しい．

```
thrift --gen cocoa myservice.idl
socpc MyClass.m AnotherClass.m YetAnotherClass.m …
```

利点：元のソースコードはそのままであるため，可読性と差異判別性は高い．
欠点: 実装には多大なコストがかかり実装の容易性と正しさという点では大きく劣る．

(2) コンパイラのオプションフラグを用いる手法

thrift コンパイラのようなソースコード生成系の場合，コマンドラインのオプションフラグにより生成するソースコードを ARC 対応または非 ARC 対応にすることが可能である．

```
thrift --gen cocoa -ARC myservice.idl
```

この手法はコンパイラ処理系中で, フラグにより生成するコードを切り替えるため，コンパイラ内部の見通しが悪くなる欠点がある．第三者が正しく理解せずにコンパイラを改造すると，不適切なコードを生成する可能性もある．また，生成されたコード

c 例：オブジェクトが delegate への参照を保持すると閉ループができる．ARC では_weak の付加が必要．

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

は ARC，非 ARC のどちらかでしかコンパイルできないため利便性も低い．

利点：可読性，判別容易性は高い．生成されたソースコードの速度も速い．
欠点：実装が複雑になり，正しくないコードを生成する可能性もある．ソースコード生成系にしか適用できない．

(3) #ifdef で条件コンパイルする手法

コンパイル中に ARC モードであるか非 ARC モードであるかは#ifdef 条件文で判別可能である．従って以下のように編集すればよい．

```
#ifdef __has_feature(objc_arc)
    ivar = aName;
#else
    ivar = [aName retain];
#endif
```

利点：ARC コンパイルされた app の速度は損なわれない．
欠点：可読性，保守性が低い．例として，後述の Thrift ライブラリ中で retain/release/dealloc がある行はソースコード中の 5%程度を占め，#ifdef を入れる箇所は多い．修正時に ARC と非 ARC を混同したり，片方の修正を忘れるミスが起きやすい．

(4) 代理メソッドによる無害化

別種の方法として，retain/release を残したまま，それらを無害にする「無害化」を考える．

MyClass 中に ARC にも非 ARC にも対応する代理の retain/release/autorelease を実装し，それらを呼び出すようにすればよいのではないか？ 例えば，retain は非 ARC の際は retain として振る舞い，ARC の場合は単に self を返す無害なメソッドとする．そうすれば，コード中に多数ある retain は retain のままであり，ソースコードの見た目は非 ARC と変わらない，という考え方もできるが，retain メソッド等を再定義することは ARC の仕様で認められておらず，エラーになる[3].

そこで，以下のように retain を（例えば）retain_stub という retain の代理のメソッドで, release を release_stub という代理のメソッドで置き換える手法を考える[d]. #ifdef の手法と比較してすっきりしており，可読性が高い．

```
ivar = [aName retain_stub];
[ivar release_stub];
```

d stub とは切り株のことであるが, プログラミングでは代用品の意味で使われる. stub 以外の文字でも良い.

図 1 は図 3 のように書き換える．ARC でも非 ARC でもコンパイル可能である．一見すると， 図 1 とほとんど同じように見える．「非 ARC の時は retain_stub は retain と同じ」と考えてプログラミングしやすい．同時に，_stub が付いていることで非 ARC の場合は retain が有効化され, ARC の場合には無害化されることが明瞭である．

```
 1 - (void) methodA:(Name *)aName {
 2     Person *person = [[Person alloc] init];
 3     [userManager addPerson:person];
 4     [person release_stub];
 5
 6     Group *group = [[[Group alloc] init] autorelease_stub];
 7     [group method2];
 8
 9     ivar = [aName retain_stub];
10 }
11
12 - (void) dealloc {
13     [ivar release_stub];
14     [super dealloc_stub];
15 }
```

図 3　ARC と非 ARC に対応した基本文例
Figure 3　A basic example for both ARC and non-ARC.

図 4 はこれらの代理メソッドの実装である．aName オブジェクトは Name クラスのインスタンスであるため，Name クラス中に以下のように記述する．例えば，release_stub は非 ARC の際は release として振る舞い，ARC の場合は何もしない無害なメソッドとなる．本手法にはメモリリークや参照の閉ループを発見するために代理メソッド中にメッセージを表示する処理等の付加的な記述が可能である利点がある．

```
- (id) retain_stub {
#ifdef __has_feature(objc_arc)
    return self;
#else
    return [super retain];
#endif
}
```

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

```
- (id) release_stub {                    // dealloc_stub も同様
#ifdef __has_feature(objc_arc)
    // do nothing
#else
    [super release];
#endif
}
```

図 4　代理メソッドの実装
Figure 4　Implementation of stub methods.

速度に関して，コンパイラのアセンブリ出力を観察すると，最大の最適化を行っても代理メソッドへのメッセージ送信処理は行われる．インライン展開されて消えることはない．従って，ARC コンパイラが自動挿入する retain 相当の処理に加えて，代理メソッドへのメッセージ送信の時間と処理時間がかかる．しかし，代理メソッドはほとんど空であり，これが問題になる app はごく僅かであろう．

利点：可読性と判別容易性は高い．代理メソッド中に付加的な記述が可能である．
欠点：retain_stub は self を返すため，全てのクラスに同様の代理メソッドを備えなければならない．別のクラスの retain_stub を流用できない．ソースファイル数が多い場合は手間がかかる．

(5) カテゴリを用いた代理メソッドよる無害化

前項の，全てのクラスに代理メソッドを追加しなければならない欠点を解消したい．aName オブジェクトの retain_stub が self を返すということは，aName オブジェクトの中にこれらの代理メソッドが存在しなければならない．しかし，それは Name クラスでなく，そのスーパークラスでも構わない．Objective-C はカテゴリ機能により，既存のクラスのメソッドテーブルを変更し，実行時にメソッドを追加することが可能である．代理メソッドを全てのクラスのスーパークラスである NSObject に追加すれば，代理メソッドをひとつのソースファイルに集約することが可能である．

NSObject のカテゴリの定義 ARCStubs.h は下記のように行う．

```
// ARCStubs.h
@interface NSObject(ARCStubs)
- (id) retain_stub;
- (void) release_stub;
// 他の代理メソッドのシグニチャ
@end
```

カテゴリとなった retain_stub を使用するソースコードは最初に ARCStubs.h を import する．中身は図 3 と同じであり，ARC と非 ARC のコンパイルが可能である．

カテゴリの実装は，下記のようにひとつのソースファイルに記述する．この部分はランタイム・ライブラリとして提供し，コンパイルは ARC もしくは非 ARC で行う．実装部分は図 4 と同じである．これは実験により動作することを確認した．

```
#import ARCStubs.h
@implementation NSObject(ARCStubs)
// 実装部分として図 4 がここに入る
@end
```

利点：可読性と判別容易性は高い．代理メソッド中に付加的な記述が可能である．代理メソッドがひとつに集約され，前項よりは利便性と実装性が高い．

(6) 代理メソッド self による無害化

前項で代理メソッドがひとつに集約されたが，これを全く無くすことはできないだろうか？

retain_stub は self を返す．NSObject に self を返す self メソッドがあるため，ARC の場合のみそれを使用することにする．release_stub は，何もしないメソッドを用意する代わりに，これも ARC の場合のみ NSObject の self メソッドを使用する．返値は無視するところがポイントである．autorelease_stub も同様である．図 1 に示した dealloc はどうするか？　これも dealloc_stub として，[super self]に置き換える．

このような方法をとった場合の ARC コンパイルの基本文例を図 5 に示す（左目で見た場合に相当する）．一見奇妙に見えるが正しく動作する．

```
 1 - (void) methodA:(NSString *)aName {
 2     Person *person = [[Person alloc] init];
 3     [userManager addPerson:person];
 4     [person self];
 5
 6     Group *group = [[[Group alloc] init] self];
 7     [group method2];
 8
 9     ivar = [aName self];
10 }
11
12 - (void) dealloc {
13     [ivar self];
14     [super self];
15 }
```

図 5 代理メソッドとして self を用いた基本文例
Figure 5　A basic example after conversion with self.

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

```
#if __has_feature(objc_arc)
#define retain_stub self
#define autorelease_stub self
#define release_stub self
#define dealloc_stub self
#define bridge_stub __bridge
#define weak_stub __weak
#else
#define retain_stub retain
#define autorelease_stub autorelease
#define release_stub release
#define dealloc_stub dealloc
#define bridge_stub
#define weak_stub
#endif
```

図 6　self による無害化マクロ TObjective-C.h
Figure 6　Macro for de-harming by self. TObjective-C.h

一方，非 ARC コンパイルの場合は図 1 と同じにしなければならない（右目で見た場合）．ソースコードの表現方法は図 3 と同じとする（両目で見た場合）．すると，これらの変換はビルド中にプリプロセッサを使用し，図 6 のような無害化マクロで行うことが可能である．前半は ARC 用マクロ，後半は非 ARC 用マクロである．

使用時には，図 3 のソースコードの上部に図 6 の無害化マクロ TObjective-C.h を読み込む一行を挿入する．ARC コンパイル時にマクロが展開されると，図 5 のように_stub の部分が無害な self に置き換わる．非 ARC コンパイル時には図 1 になる．いずれもビルド中のことであり，開発者の目に触れることはない．

なお，図 6 中の bridge_stub と weak_stub は ARC 化で使用するアノテーションである．他のアノテーションも同様の方法で定義して，必要に応じて使用する．

利点：　可読性，相違認識性は高い．マクロだけで実現しているため利便性も実装容易性も高い．

5.1 評価結果のまとめ

以上の手法の評価結果を下表にまとめる．◎，○，△，×の順で評価が高い．

結論として，self による無害化の手法が最も高い評価となり，カテゴリ代理メソッドによる無害化の手法がそれに次いだ．

表 1　手法の評価結果のまとめ

手法名	正しさ	可読性	実装の容易性	判別の容易性	実行速度
ARC 変換スクリプト	×	○	×	○	◎
コンパイラのオプションフラグ	△	○	×	○	◎
#ifdef による条件コンパイル	△	×	×	×	◎
代理メソッドによる無害可	○	○	△	○	○
カテゴリ代理メソッドによる無害化	○	○	○	○	○
self による無害化	○	○	◎	○	○

6. self による無害化の手法 Apache Thrift への適用

Facebook で開発され Apache 財団に寄贈された Thrift[5]は，インターフェース定義言語（IDL）により記述されたデータ構造とサービスの定義ファイルから指定のプログラミング言語のソースコードを生成する．iOS の Objective-C, Android の Java, Windows の C#，クラウド Google App Engine の Python 等を含む 14 の言語を生成する能力を有するため人気が高い．Facebook，Evernote，Cassandra 等のスマートデバイスとクラウドの通信等で多く使われている，今日の基幹技術である．

Thrift は C++で記述された IDL コンパイラと iOS 用には Objective-C で記述された 32 ファイルからなるランタイム・ライブラリから構成される．IDL コンパイラは他の言語と共通の IDL パーサー部と言語毎の生成部から構成される．

前章で述べた self による無害化の手法を Thrift 処理系に対して適用した．下表に変更箇所の数を示す．コンパイラへの適用は，無害化マクロ定義を import するコードを生成することと，retain 等を書き出していた箇所を retain_stub 等を書き出すようにする変更である．ライブラリの実装ファイルのコメントと空行以外の行数は計 1044 行あり，下記の変更は約 5.3%に相当する．

表 2　Thrift の変更箇所の数

手法名	コンパイラ	ライブラリ
retain の書き換え	9	16

Vol.2012-MBL-61 No.41
Vol.2012-UBI-33 No.41
2012/3/13

release の書き換え	14	24
autorelease の書き換え	7	3
dealloc の書き換え	3	7
bridge_stub の追加	0	4
autoreleasepool の書き換え	0	2

回帰テスト用の 12 個の IDL 定義ファイルから生成された Objective-C ソースコード, 及びランタイム・ライブラリに対し, 静的アナライザによるエラーチェックと ARC コンパイルを行い問題がないことを確認した．動作にも問題がなかった．改良前との間で速度の差は，通信時間の揺らぎと比較すると小さすぎて計測できなかった．

以上により，app の開発者は Thrift ランタイム・ライブラリと生成されたソースコードを Xcode 上にドラッグして，設定の変更なくビルドボタンひとつで ARC でも非 ARC でもコンパイルすることが可能となった. 今まで Thrift が ARC 非互換のために採用できなかった多くの iOS プロジェクトにおいても Thrift を採用することが可能となり，開発効率と安全性の大幅な向上が見込まれる．

7. 考察

評価の結果，最もよい結果となったのは self による無害化の手法である．開発者にはこれを勧める．カテゴリを用いた代理メソッドによる無害化の手法にも self による方法にはない利点がある．メモリ管理を監視したい場合にはこれも選択肢となる．代理メソッドの名前を変えれば，これらを混在させて利用することも可能である．

高速性が要求されるタイトなループ内で self メソッドを呼び出すことが時間的に許容できない場合は，まず@autoleasepool を併用し，それでも遅い場合は#ifdef による方法をとるとよい．

代理メソッドを retain_stub ではなく，開発者の好きな文字列にしてもよいであろう. 短く RETAIN とする記法も考えられる. 例 RETAIN/RELEASE/AUTORELEASE. ただ，RETAIN は retain とアルファベットと発音が同一であるため，認知的に両者を混同する恐れがある．判別容易性の項にも書いたように，代理メソッドであることを意識しやすい retain_stub という記法の方が優れていると著者らは考えている．

Xcode は非 ARC のコードを ARC に変換するツールを内蔵し，ARC 変換後に問題になる箇所の指摘と，retain 等の削除を行う．変換は一方向で，ARC と非 ARC の共存は不可能である．また，コマンドラインからは使用できない．

著者らが調べた限りでは ARC と非 ARC の互換性実現の手法について比較検討した文献，retain を NSObject のカテゴリ中の代理メソッド，あるいは self に置き換えて無害化し ARC と非 ARC を共存させる手法に関する先行研究は存在しなかった. 他のライブラリ開発プロジェクトでどうしているかを調べたが, 非 ARC のまま ARC 対応を延期しているプロジェクトがほとんどであった．非 ARC コードのメモリ管理のミスはなかなか顕在化せず後で困ることが多い．本稿の手法を用いれば，非 ARC との互換性を保ちながら徐々に安全な ARC に移行することが可能である．

8. おわりに

Objective-C のひとつのソースコードに ARC と非 ARC を共存させ，どちらのコンパイルスイッチが指定されても自動的に対応する手法等を 6 種類提示し，5 つの評価基準によって評価を行った. その結果, self を用いた無害化の手法が最も優れていた．

この手法を Apache Thrift に適用し，設定の変更なしで ARC でも非 ARC でもコンパイルすることが可能となった. この改良は同時に行った別の改良と共に Thrift プロジェクトにおいて検討・テストされた結果，有効性が確認されたため採用された[6]. 現在は多くのプロジェクトによって使用が開始されつつある．

参考文献

1) マイナビ:11 月モバイル OS シェア(オンライン), 入手先 <http://news.mynavi.jp/news/2011/12/05/017/index.html> (参照 2012-01-05).
2) Apple: Memory Management Programming Guide, iOS Developer Library (2007).
3) Apple: Transitioning to ARC Release Notes, iOS Developer Library (2011).
4) Clang Language Extensions: Objecitve-C Automatic Reference Counting, LLVM Documentation (online) available from <http://clang.llvm.org/docs/AutomaticReferenceCounting.html> (accessed 2012-01-05).
5) Slee, M. Agarwal, A. and Kwiatkowski, M.: Thrift: Scalable Cross-Language Services Implementation, Facebook white paper (2007) (online) available from <http://thrift.apache.org/static/thrift-20070401.pdf> (accessed 2012-01-05).
6) Add support of ARC to Objective-C (online) available from <https://issues.apache.org/jira/browse/THRIFT-1340> (accessed 2012-01-05).
7) Cox, B. and Novabilsky, A.: Object-oriented Programming, An Evolutionary Approach Second Edition, Addison Wesley (1986).
8) Lattner, C.: LLVM: An Infrastructure for Multi-Stage Optimization, Masters Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign (2002).
9) Lattner, C.: LLVM and Clang: Advancing Compiler Technology", FOSDEM 2011: Free and Open Source Developers' European Meeting, Brussels, Belgium (2011).