Build Insiderオピニオン：小野将之（4）

Swift 3.0でなぜ「Cスタイルのforループ」「++／--演算子」などの仕様が廃止されたのか

大規模な破壊的変更が行われる最終的なバージョンといわれているSwift 3.0がついに正式リリース。多数の変更から「廃止」となった言語仕様にフォーカスを当て説明する。

小野 将之

2016/09/xx

############################################################

　先日正式リリースされたSwift 3.0では数多くの変更が含まれたが、今回はその中から廃止となった言語仕様にフォーカスを当てる。

# ■仕様廃止のProposal

　◆第3回◇http://www.buildinsider.net/column/ono-masayuki/003◆で紹介したSwift Evolutionリポジトリの◆Proposalステータスページ◇https://apple.github.io/swift-evolution/◆を見ると、それぞれのProposalが「承認されたが実装待ち／Swift 3.0に実装済み／Swift 2.2に実装済み／後回し／リジェクト済み」のどの状態にあるかが分かる。これらのProposalの中で仕様の廃止に関係しているものは、その名前に「remove」「eliminate」などを含むものである。

## ●Swift 3.0で廃止された12件の仕様

　Swift 3.0で廃止されたのは、以下の12件である。

・ ◆SE-0002： カリー化関数の構文の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0002-remove-currying.md◆

・ ◆SE-0003： 関数引数への◎var◎指定の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0003-remove-var-parameters.md◆

・ ◆SE-0004： ◎++◎／◎--◎演算子の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md◆

・ ◆SE-0007： Cスタイルの◎for◎ループの廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0007-remove-c-style-for-loops.md◆

・ ◆SE-0029： タプルを関数引数に与えたときに暗黙的に展開・適用される仕様の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0029-remove-implicit-tuple-splat.md◆

・ ◆SE-0053： 関数引数への◎let◎明示指定を廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0053-remove-let-from-function-parameters.md◆：

　　－ 上述の◎SE-0003◎の廃止により明示指定の必要性がなくなったため

・ ◆SE-0102： 関数が何も返さないことを保障する◎@noreturn◎属性が廃止されて、◎Never◎列挙型を戻り値として指定するように変更◇https://github.com/apple/swift-evolution/blob/master/proposals/0102-noreturn-bottom-type.md◆

・ ◆SE-0072： Swift標準ライブラリの型とFoundationの型の暗黙的な変換を完全に削除◇https://github.com/apple/swift-evolution/blob/master/proposals/0072-eliminate-implicit-bridging-conversions.md◆

・ ◆SE-0109： ◎Boolean◎プロトコルを廃止して◎Bool◎構造体を直接利用するように変更◇https://github.com/apple/swift-evolution/blob/master/proposals/0109-remove-boolean.md◆

・ ◆SE-0111： 関数の型の一部として関数の引数ラベルを考慮していた仕様を廃止して、型システムをシンプルに◇https://github.com/apple/swift-evolution/blob/master/proposals/0111-remove-arg-label-type-significance.md◆

・ ◆SE-0121： ◎<◎／◎>◎／◎<=◎／◎=>◎の4つの比較演算子で◎Optional◎型の値を受けられないように◇https://github.com/apple/swift-evolution/blob/master/proposals/0121-remove-optional-comparison-operators.md◆：

　　－ジェネリクスが成熟しないと正確な実装が不可能なため、現状の中途半端な仕様をいったん廃止して、将来の対応に備える（Swift 3.0が破壊的変更をしやすいタイミングのため）

・ ◆SE-0125： ◎NonObjectiveCBase◎クラス／◎isUniquelyReferenced◎関数を廃止して、◎isUniquelyReferencedNonObjC◎に統合◇https://github.com/apple/swift-evolution/blob/master/proposals/0125-remove-nonobjectivecbase.md◆

　◎SE-0101◎／◎SE-0109◎／◎SE-0125◎の各Proposalが実装されたからといって言語の機能が減ったわけではないので、これらは「仕様の廃止」というよりは「仕様の変更」と見なすべきかもしれないが、ここでは廃止された仕様に含めた。こうして見ると、数としては12件と多いがマイナーな言語仕様の変更が多く、通常のアプリケーションコードに対して影響が出そうなのは半分程度であろう。

　以下では、この中から代表的な2つのProposalをかいつまんで、Swiftコミュニティがどういう考えによってその決定をしたのか詳しく解説する。

# ■「SE-0004： ++／--演算子の廃止」と「SE-0007： Cスタイルのforループの廃止」を深掘り

　廃止された仕様の中でも特に目を引き、分かりやすいのが「◆SE-0004： ◎++◎／◎--◎演算子の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md◆」と「◆SE-0007： Cスタイルの◎for◎ループの廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0007-remove-c-style-for-loops.md◆」の2つである。これらはC言語の影響を受けた言語、つまり現在使われている多くのプログラミング言語で当たり前のように採用されている仕様である（ちなみにPythonやRubyなど、いずれの仕様もない言語もある）。Swift 3.0での廃止の前に、Swift 2.2で先行して非推奨扱い（使用すると警告が発生する）となったが、その時も注目を集めた。

## ●廃止による影響の具体例

　変数◎i◎を◎for◎ループでインクリメントしながら10回出力する簡単なサンプルコードを示す。

　Swift 2系まではリスト1に示すような書き方もできた。C言語系の◎for◎ループと◎++◎演算子（△インクリメント演算子△）を組み合わせたなじみのある書き方である。これが、Swift 3.0ではコンパイルエラーになるように変わる。

□コード：Swift------------------------

for var i = 0; i < 10; i++ {

print("i: \(i)")

}

□□------------------------

リスト1　Cスタイルのforループ（Swift 3ではコンパイルエラーとなる）

□□□------------------------

　まず、◎i++◎という書き方はできなくなり、「◎i += 1◎」という形式に統一される（リスト2）。

□コード：Swift------------------------

for var i = 0; i < 10; ▲i += 1▲ {

print("i: \(i)")

}

□□------------------------

リスト2　インクリメント演算子は廃止された

□□□------------------------

　さらにいえば、◎for◎ループそのものの書き方がリスト3に示すような書き方に統一されるので、「◎i += 1◎」という記述自体が◎for◎ループからはなくなる。

□コード：Swift------------------------

// コード例A

for i in 0 ..< 10 {

print("i: \(i)")

}

□□------------------------

リスト3　Swiftらしいforループの書き方

□□□------------------------

　Swift 2系でもこの書き方はできた（むしろ、推奨されていた）ので、「変更」ではなく「Swiftらしくない書き方」をできないように一部の仕様が「廃止」された、ということである。

　記事の本筋からは少し逸れるが、以下の書き方もできる。これは文脈や好み・開発チームのポリシーなどによってどちらを使っても問題ないだろう。

□コード：Swift------------------------

(0 ..< 10).forEach { i in

print("i: \(i)")

}

□□------------------------

リスト4　リスト3と同値なforEachループ

□□□------------------------

　次に、これら2つのProposalが受け入れられた共通の理由について考えてみよう。

## ●2つのProposalに共通する判断理由

　SE-0004とSE-0007に関して、Swiftコミュニティのコアチームは以下のように判断した。

・ C言語から何となく持ってきた仕様であり、Swift言語の仕様としてふさわしいか熟考したわけではなかった

・ あらためて、その仕様があることのメリット／デメリットを羅列して熟考した結果、デメリットに対してメリットが薄い：

　　－ 今からSwiftを作り直すとしたら、この仕様は入れるべきか？ という観点でも検討された

・ その仕様がなくとも他の書き方ができ、さらにその他の書き方の方がSwiftらしい書き方である

・ 既存のSwiftコードベースへの影響が限定的である

・ 破壊的変更であるので影響は免れないが、利用頻度が少ないことと、自動変換である程度はカバーできることから、廃止を許容できる

　特に「Swift 3.0を最後の大きな仕様変更としたい」という事情もあり、今回、こういった不要かもしれない仕様の再検討がなされたのである。

　次にそれぞれのProposalについて、さらに詳しく見ていく。

## ●「SE-0004： ++／--演算子の廃止」について詳しく

　◎++◎演算子について、以下のように宣言した変数◎i◎を使って説明していく（◎--◎演算子は単純に逆の挙動なので省略する）。

□コード：Swift------------------------

var i = 0

□□------------------------

リスト5　変数iの宣言

□□□------------------------

　変数◎i◎の値をインクリメントしたい場合、◎++◎演算子を使うとリスト6に示すいずれかの形式で書ける。

□コード：Swift------------------------

++i // ≪A≫

i++ // ≪B≫

□□------------------------

リスト6　++演算子によるインクリメント

□□□------------------------

　◎++◎演算子が使えない場合は、リスト7のような書き方になる。

□コード：Swift------------------------

i += 1

□□------------------------

リスト7　+=演算子によるインクリメント

□□□------------------------

　いずれの場合も、変数◎i◎の値は▲1▲になる。

　では、リスト6の≪A≫（△前置インクリメント演算子△を使用）と≪B≫（△後置インクリメント演算子△を使用）の違いは何かというと、それぞれの戻り値が異なる。リスト6の≪A≫ではインクリメントした結果が返り、≪B≫では先に値を返してからインクリメント処理をする（リスト8）。

□コード：Swift------------------------

let x = ++i // → x: 1

let y = i++ // → y: 0

□□------------------------

リスト8　前置インクリメント演算子と後置インクリメント演算子ではその戻り値が異なる

□□□------------------------

　これらから、◎++◎／◎--◎演算子のメリットとして以下が挙げられる。

・ うまく使えば、インクリメント処理とその結果の返却処理を組み合わせたコンパクトなコードを書ける

・ 「◎++◎」は「◎+= 1◎」よりも短く書ける

　一方、それらのメリットについて、以下の指摘もできる。

・ ◎++◎演算子を変数の前に置くか後に置くかという些細（ささい）な差で、プログラムの挙動が変わるので、可読性の低下やバグの要因となることが多い：

　　－ ▲そもそも1つの演算子で、自身の値の書き換えと値の返却という2つの機能を持つこと自体がよくないのでは？ と見ることもできる▲

　　－ 最近は可読性優先などの理由で、前置と後置による挙動の違いを生かしたコードが少なくなってきており、そもそもその差を知らない開発者も存在し、学習コストなどもかかる

・ 「◎++◎」は「◎+= 1◎」よりも短く書けるといっても些細な差である

　上の太字部分についてもう少し詳しく書くと、Swiftでは△代入演算子△は戻り値が◎Void◎型である。例えば「◎x += 1◎」は▲Void▲を返すし、「◎x = 1◎」という単純な代入も▲Void▲を返す。C系の言語を中心として、プログラミング言語の中にはこのような仕様になっていないものが多い。例えば◎if◎文で比較をしようとしたが、「◎==◎」演算子と「◎=◎」演算子を間違えて代入をしてしまい、バグの原因になってしまうといったことがよくある。

　リスト9に示すコードはSwiftではコンパイルエラーとしてくれるが、多くのC系の言語では◎if◎文の条件式の評価に変数◎x◎の値が使われ、コンパイルが通ってしまう。この例の場合、変数◎x◎への1の代入と◎if◎文内の処理の両方が必ず実行されてしまう。

□コード：Swift------------------------

var x = 0

if x = 1 { // Swiftではここでコンパイルエラーが発生して、本来「x == 1」と書くべきだったと気付ける

print("x: \(x)")

}

□□------------------------

リスト9　Swiftではif文の条件式にこのような記述を誤って書いてしまうことはない

□□□------------------------

　代入演算子の処理結果が▲Void▲を返すようにすると、こういったバグを防げるとともに、有する機能が1つとシンプルになる。上述の通り、Swiftの代入演算子は◎++◎と◎--◎以外はそうなっているので、この2つの演算子を廃止する（戻り値を返さないようにするという別案もあったが）と、言語仕様の一貫性も向上する。

　このように、今まで当たり前のように使っていた◎++◎演算子と◎--◎演算子をあらためて見直してみると、これらにはメリットがあるどころか、デメリットばかりなのではと思えてくる。「『◎++◎』は『◎+= 1◎』よりも短く書ける」というのは確かにそうなのだが、これらの演算子が持つデメリットと比べるとその恩恵は小さく、この言語仕様は不要という判断がなされたのである。

　また、◎++◎演算子と◎--◎演算子は、次に述べるCスタイルの◎for◎ループ内で頻繁に利用されていたが、それが廃止になることもこの判断の追い風になった。

## ●「SE-0007： Cスタイルのforループの廃止」について詳しく

　Cスタイルの◎for◎ループの廃止は「リスト3のような書き方ができるにもかかわらず、冗長な文法を残しておくメリットが薄い」というひと言に尽きる。Cスタイルの◎for◎ループを使いたいという意見があるとすれば、その基本的な理由は「その書き方に慣れている」ということだろう。逆に、初学者にとっては、リスト3のような書き方しかできないようにしておいた方が学習コストも低くなり、余計な混乱も防げる。

　リスト10に示すコードをリスト3のような書き方に修正すると少し冗長になってしまいそうだが、これについても実は問題ない。

□コード：Swift------------------------

for var i = 0; i < 10; i += 2 {

print("i: \(i)")

}

□□------------------------

リスト10　カウンターが2ずつ増加するforループ

□□□------------------------

　Swiftには◎stride◎関数が用意されていて、リスト11のように書けるからだ。

□コード：Swift------------------------

for i in stride(from: 0, to: 10, by: 2) {

print("i: \(i)")

}

□□------------------------

リスト11　Swiftではstride関数を使って初期値、終端値、差分を指定できる

□□□------------------------

　リスト11で使われている◎stride(from: 0, to: 10, by: 2)◎は、◎from◎引数と◎to引数で指定された範囲◎0..<10◎に含まれ、◎by◎引数で指定された2を差分とするSequence、つまり◎[0, 2, 4, 6, 8]◎という配列に相当するSequenceが返される（◎to◎引数ではなく◎through◎引数を用いると、範囲が◎0...10◎となるので、◎[0, 2, 4, 6, 8, 10]◎という配列に相当するSequenceが得られる）。

　◎filter◎メソッドや◎where◎句などを組み合わせることで、より複雑な条件も指定でき、大抵のケースでCスタイルの◎for◎ループより簡潔かつ分かりやすくループを書ける。

　ここまではProposalをかみ砕いたような内容だったが、ここから筆者の考察を交えながら述べていく。

## ●同じ処理を記述するのに複数の書き方ができた方がよいか

　「メリットが薄いといっても、せっかく今、備わっている機能なのだから残しておいてもよいのでは？」という意見もあるだろう。しかし、複数の書き方がありつつも、Swiftとしてどちらの書き方が推奨されているかが明確であれば、結局、推奨されない書き方の使いどころがなくなってしまう。ベターな書き方にそろえるために、コード規約／レビュー／Lintツールなどで推奨される方にそろえる労力なども発生するので、それならいっそのことコンパイルエラーで正してくれた方が楽である。

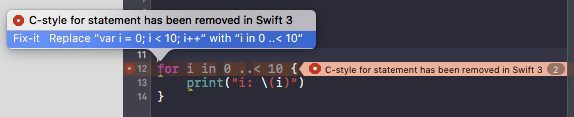
　また、仕様はシンプルなほど、Swift言語自体の開発もしやすいという面もある。一応残しておいた仕様が足かせになって、本来入れたい言語機能の実装に苦労するのはとてももったいない。

## ●既存コードへの影響

　廃止される構文を使っていた場合、当然、コンパイルエラーが発生してしまう。ただ、Xcodeの自動変換機能がコードを修正する手助けをしてくれるので、修正は簡単だ。以下はその例である（図1は個別のコンパイルエラーに対する修正機能で、図2はプロジェクト単位の自動変換機能）。

□------------------------------

★画像【01.png】★



□□----------------------------

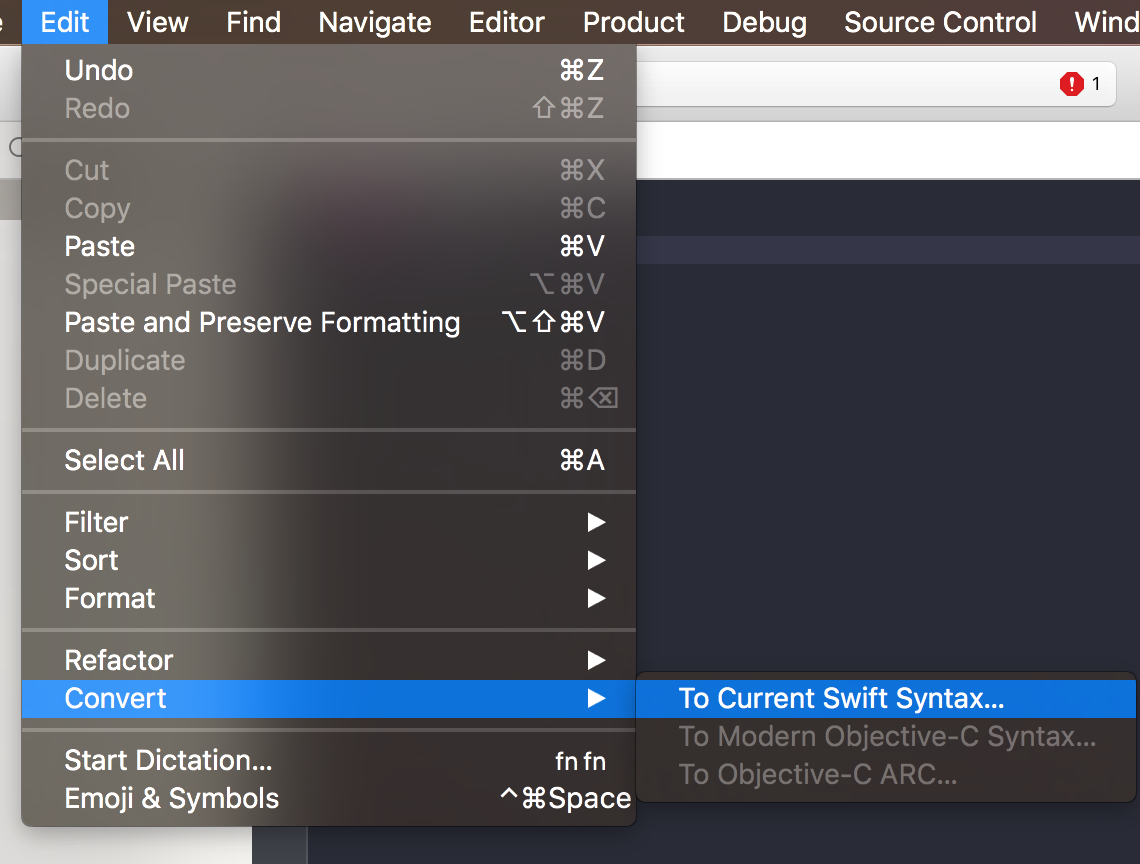
図1　個別のコンパイルエラー指摘とともに自動修正を促す機能

［Fix-it ～］というメニュー項目をクリックすると、修正が実行される。

□□□--------------------------

□------------------------------

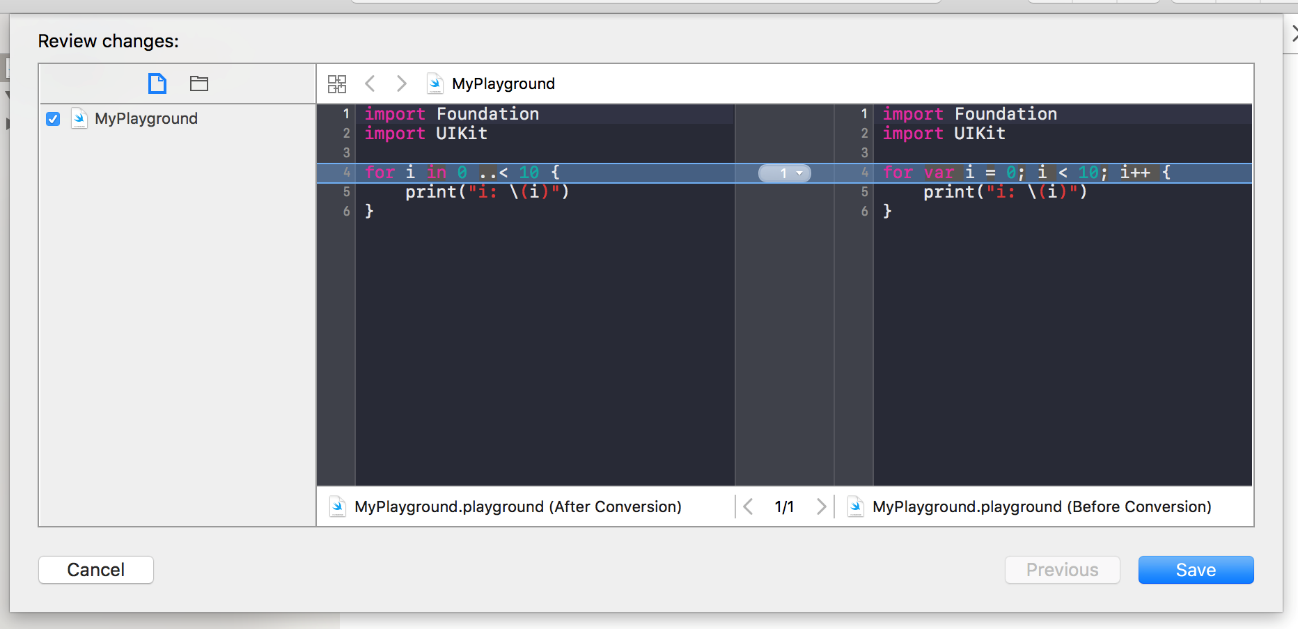
★画像【02.png】★



メニューバーから［Edit］－［Convert］－［To Current Swift Syntax］を実行

▼

★画像【03.png】★



変換前（右側のコード）と変換後（左側のコード）が並んで表示されるので、確認して問題なければ［Save］ボタンをクリックして変換を実行する

□□----------------------------

図2　プロジェクト単位で一括して自動変換する機能

□□□--------------------------

　Xcodeの自動変換機能は便利ではあるが、全ての場合でうまくいくわけではないので注意してほしい。うまくいく場合とうまくいかない場合について、具体的な例を説明しておこう。

### ○自動変換でうまくいく場合

　以下の2つのような単純な利用であれば、Xcodeが自動でコードを変換してくれる。

□コード：Swift------------------------

var i = 0

i++ // 「i += 1」に自動変換

□□------------------------

リスト12　単純なコードであればXcodeが自動で対処してくれる（++演算子）

□□□------------------------

□コード：Swift------------------------

for var i = 0; i < 10; i++ { // リスト3の形式に自動変換

print("i: \(i)")

}

□□------------------------

リスト13　単純なコードであればXcodeが自動で対処してくれる（Cスタイルのforループ）

□□□------------------------

### ○自動変換ではうまくいかない場合

　リスト14に示す処理は自動変換で対応できない。

□コード：Swift------------------------

var i = 0

let x = ++i

□□------------------------

リスト14　このコードは自動変換では対応できない

□□□------------------------

　このコードはリスト15のように変換される。変換の前（リスト14）では、変数◎x◎の値は▲1▲になるが、変換後（リスト15）は▲Void▲になってしまう（型が異なるので、バグにはならずに後続の処理がコンパイルエラーになるだろう）。

□コード：Swift------------------------

var i = 0

let x = i += 1

□□------------------------

リスト15　変換後のコード

□□□------------------------

　先ほどの◎stride◎関数の例で見たCスタイルの◎for◎ループも自動変換で対応できない（リスト16に再掲）。

□コード：swift------------------------

for var i = 0; i < 10; i += 2 {

print("i: \(i)")

}

□□------------------------

リスト16　for文のインクリメント式が単純なインクリメントでない場合はうまく自動変換できない

□□□------------------------

　Xcodeでは「C-style for statement has been removed in Swift 3」とコンパイルエラーのメッセージが表示されるのみである。ここは技術的には変換可能であるが細かいところまでカバーするとキリがないので単純な◎for◎ループのみの対応にとどめた、ということであろう。

　ちなみに、この自動変換処理もSwiftリポジトリに含まれているため、その部分のソースコード（リスト17）を見ると「変数の初期値と終点値が定まっていて、▲1▲ずつインクリメントかデクリメントしていくだけの単純な◎for◎ループ」のときのみ自動変換が利くことが分かる。

□コード：C++------------------------

VarDecl \*loopVar = dyn\_cast<VarDecl>(initializers[1]);

Expr \*startValue = loopVarDecl->getInit(0);

OperatorKind OpKind;

Expr \*endValue = endConditionValueForConvertingCStyleForLoop(FS, loopVar, OpKind);

bool strideByOne = unaryIncrementForConvertingCStyleForLoop(FS, loopVar) ||

plusEqualOneIncrementForConvertingCStyleForLoop(TC, FS, loopVar);

bool strideBackByOne = unaryDecrementForConvertingCStyleForLoop(FS, loopVar) ||

minusEqualOneDecrementForConvertingCStyleForLoop(TC, FS, loopVar);

if (!loopVar || !startValue || !endValue || (!strideByOne && !strideBackByOne))

return;

□□------------------------

リスト17　自動変換が可能かどうかをチェックするコード

◆Swiftのリポジトリ◇https://github.com/apple/swift/blob/master/lib/Sema/MiscDiagnostics.cpp◆より。

□□□------------------------

　さらに余談になるが、これは「『◆［SR-226］ Implement warning about the use of C-style for loops in Swift 2.2 - Swift◇https://bugs.swift.org/browse/SR-226◆』で起票され、『◆［SR-226］ Deprecation of C-style for loops by gregomni ・ Pull Request #552 ・ apple/swift◇https://github.com/apple/swift/pull/552◆』のプルリクエストで実際にコード対応された」という流れまで追うことができる。Swiftがオープンソース化されたことで、このように隅々まで深掘りできるようになったのである。

### ○そもそも廃止される構文の利用が少ない

　Xcodeによる、廃止された構文の自動変換対応について書いたが、そもそも廃止される構文の利用が少ないということもある。◎++◎演算子と◎--◎演算子については、副作用のないコードを心がけていれば、それらを使っていた箇所でも戻り値を使用せずに、単純なインクリメント／デクリメント処理を行っているだけで、自動変換がうまくいく場合に該当するだろう。

　Cスタイルの◎for◎ループに関しては、Proposalでも指摘されている通り、Swiftのコードベースでの利用例が極めてまれである。筆者もSwiftではこれまで一度も書いたことがない。もし既存コードでうっかり書いてしまっていたところがあっても、Swiftらしいコードに正せる良い機会と思いながら、新しい仕様に寄り添うのが、Swiftとうまく付き合うコツであると感じる。

# ■リジェクトまたは先送りになった仕様廃止系のProposal

　ここまでSwift 3.0で廃止になった12個の仕様を挙げ、そのうち2つについて詳しく見てきたが、もちろん仕様を廃止してシンプルにすることが常に正義というわけではない。慎重な議論の末に「廃止」がリジェクトされたり、先送りになったりした、つまり仕様が生き残ったProposalもあるので、最後にそれらを簡単に紹介する（そもそもProposalはメーリングリストなどでの十分な議論をくぐり抜けてきたものであり、実際の提案数自体はさらに多い）。

## ●リジェクトされた仕様廃止系のProposal

　リジェクトされた仕様廃止系のProposalは次の4件である。

・ ◆SE-0013： 非◎final◎の親メソッドの部分適用の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0013-remove-partial-application-super.md◆

・ ◆SE-0105： ◎for◎ループの◎where◎句の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0105-remove-where-from-forin-loops.md◆

・ ◆SE-0108： プロトコルの◎associatedtype◎推論を廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0108-remove-assoctype-inference.md◆

・ ◆SE-0119： ◎extension◎からアクセス修飾子を廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0119-extensions-access-modifiers.md◆

　リジェクトの判断理由は、それぞれのProposalページにある「Decision Notes: Rationale」のリンク先に記述されている。それらを見ると、採択されたProposalとは逆に、「廃止するための労力、仕様があることのメリット ＞ 廃止するメリット」と見なしたことが理由だと分かる。これらについては明確なリジェクト判断が下されたので、その理由が覆されるような新たな発見などがない限り、蒸し返されることもなく、今後もSwiftの仕様に残り続けるはずである。

## ●先送りになった仕様廃止系のProposal

　先送りになった仕様廃止系のProposalは次の2件である。

・ ◆SE-0083： 動的なキャスト機能からCocoa API／Swift間のブリッジを削除してイニシャライザー形式に変更◇https://github.com/apple/swift-evolution/blob/master/proposals/0083-remove-bridging-from-dynamic-casts.md◆：

　　－ 採用された◆SE-0072: Swift標準ライブラリの型とFoundationの型の暗黙的な変換を完全に削除◇https://github.com/apple/swift-evolution/blob/master/proposals/0072-eliminate-implicit-bridging-conversions.md◆と関連して、Objective-Cの◎id◎型周りで問題が発生するため、この解決を練る必要があるために先送りとなった

・ ◆SE-0090： 型を参照する際の◎.self◎の廃止および、利用箇所で◎.self◎指定なしで型推論で解決◇https://github.com/apple/swift-evolution/blob/master/proposals/0090-remove-dot-self.md◆

　これらは「提案自体は妥当だが、時間や技術的な課題がありSwift 3.0に含めることはできない」と判断されたものである。課題が解決され次第、将来のバージョンに入る可能性が高い。

# ■まとめ

　今回は、Swift 3.0の廃止系の変更について紹介し、特に目立つ「◆SE-0004： ◎++◎／◎--◎演算子の廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md◆」と「◆SE-0007： Cスタイルの◎for◎ループの廃止◇https://github.com/apple/swift-evolution/blob/master/proposals/0007-remove-c-style-for-loops.md◆」について深掘りして、Swiftの言語開発がどのような考えでなされているのかを解説した。

　一見、大胆な仕様変更に見えたかもしれないが、こうやって見ると納得感が出てきたのではなかろうか。なぜこの仕様になったのだろう？ などと気になることがあったら、Proposalを自ら読み解いていくといろいろな発見があってお勧めである。