【文字の装飾タグに関するヒント】

△ストロング△＝強調したいキーワード（太字になります）

▲太字▲＝単なるデザイン目的での装飾で、キーワードではない

本文中に◎Abc("aaa")◎と記載すると、そこは<code>タグで囲まれ、その文字列が□の枠で囲まれて表示されます（GitHubなどでよくあり開発者が慣れていそうなので、最近、クラス名などのコード関連はほとんどこれで囲んでいます）。Markdownだと`Abc("aaa")`です。

ほかにHTML上などでカスタム表現してほしいことがあれば、★これでコメント★を書いていただければ、その部分は制作時に内容を見て手動でHTML化します。※ちなみに斜体は非推奨としています。

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

Build Insiderオピニオン：小野将之（2）

Swiftは3.0で安定するのか？ リリース予定日と新機能リスト

―― Swift 3へのロードマップ（前編） ――

2016年後半のリリースが予定されているSwift 3。そのリリースに先駆けて、どんな変更点があるのか、懸案となっている互換性はどうなるのかなどを見ていく。

小野 将之

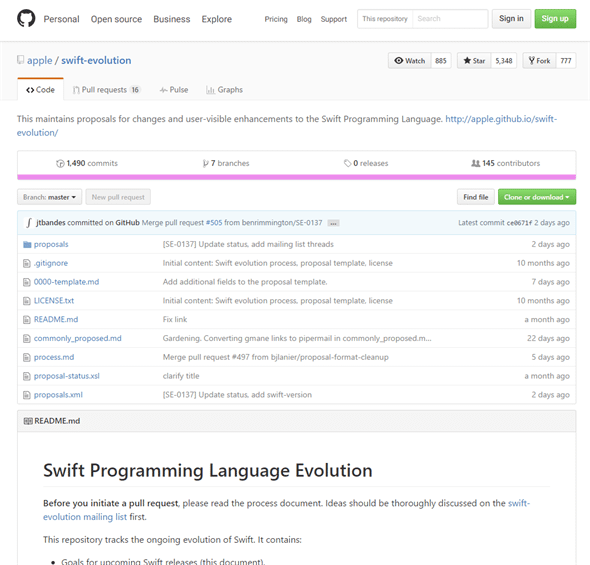
2016/08/xx

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

　第1回では、2015年12月3日にSwiftがオープンソース化されたことを紹介したが、それと同じタイミングで◆Swift Evolutionリポジトリ◇https://github.com/apple/swift-evolution◆でSwift 3のロードマップも公開された。そのロードマップの内容をベースに、今回と次回は前後編に分けてSwift 3.0リリースに向けての動向を紹介する。

□------------------------------

★画像【01.png】★



□□----------------------------

Swift Evolutionリポジトリ

□□□--------------------------

# ■Swift 3.0のリリース予定

　Swift 3.0リリース予定が◆Swift EvolutionのREADME◇https://github.com/apple/swift-evolution/blob/master/README.md◆（英語）に明記されている。リリース予定日は「△今年後半△」と書かれているが、毎年9月に新しいiOS／Xcodeの正式版をリリースするのが恒例となっているので、よほどのことがない限りSwift 3.0の正式リリースも△9月△となるだろう。◆7月の「Endgame for Swift 3」と題したメーリングリストの投稿（英語）◇https://lists.swift.org/pipermail/swift-evolution/Week-of-Mon-20160711/024424.html◆でも「Swift 3の破壊的変更は7月27日に終える」と宣言されており、今はリリースに向けた最終調整が進んでいるもようである。

　Swift 3.0開発においてフォーカスされた点は以下である。

**・ △◆Swift APIデザインガイドライン◇https://swift.org/documentation/api-design-guidelines/◆△▲をSwiftとしてあるべきものに更新▲**

　　－ Swiftでコードを書くためのガイドラインであり、またSwiftライブラリ自体もこれに沿うようにあらためて統一を図っている（詳細後述）

**・ ▲インポートされたObjective-C APIに、ガイドラインに沿った名前規則への自動変換の適用▲**

　　－ Objective-Cでそのガイドラインに沿って書かれたAPIを、Swiftで参照した際に、Swiftらしいインターフェースに◆規約ベースで自動変換◇https://github.com/apple/swift-evolution/blob/master/proposals/0005-objective-c-name-translation.md◆

　　－ 自動変換の結果が所望のものでない場合などには、◆◎NS\_SWIFT\_NAME◎◇https://developer.apple.com/reference/foundation/ns\_swift\_name◆で明示的にSwift用の名前を指定することも可能

**・ △Swift標準ライブラリ／コアライブラリ△▲を、Swift APIデザインガイドラインに沿うように変更▲**

　　－ 参照型のセマンティクスを持つ◎NS◎プレフィックス付きの◎Foundation◎クラスはSwiftにそぐわないので、◆それに対応した構造体型を導入◇https://github.com/apple/swift-evolution/blob/master/proposals/0069-swift-mutability-for-foundation.md◆（◎NSData◎クラスに対して◎Data◎構造体を導入など）

　　－ libdispatch（マルチスレッド処理用のコアライブラリ）のインターフェースを◆Swiftらしいインターフェースに変更◇https://github.com/apple/swift-evolution/blob/master/proposals/0088-libdispatch-for-swift3.md◆

**・ ▲インポートしたObjective-C APIの▲「△Swiftification△」▲（＝Swiftらしく加工すること）▲**

　　－ ◆型パラメーター付きのObjective-CジェネリクスのSwiftへのインポート対応◇https://github.com/apple/swift-evolution/blob/master/proposals/0057-importing-objc-generics.md◆

　　－ ◆インポートされたC言語ライブラリ／フレームワークを、オブジェクト指向型のインターフェースに変換◇https://github.com/apple/swift-evolution/blob/master/proposals/0044-import-as-member.md◆

　　－ ◆Objective-Cで定義された定数をSwiftの列挙型に変換◇https://github.com/apple/swift-evolution/blob/master/proposals/0033-import-objc-constants.md◆

　　－ ◆Selectorを安全に扱える構文◇https://github.com/apple/swift-evolution/blob/master/proposals/0022-objc-selectors.md◆（2.2で導入済み。古い構文は2.2で非推奨、3.0で廃止という扱い）

　　－ ◆KVO（Key-Value Observing：オブジェクトやプロパティの変更通知機構）などで必要になるkey pathを文字列リテラルではなく◎#keyPath◎式で安全に扱えるように◇https://github.com/apple/swift-evolution/blob/master/proposals/0062-objc-keypaths.md◆

**・ ▲言語仕様の再考・見直し▲**

　　－ 3.0を最後の大きな破壊的変更を含むリリースとするために全面的に見直し

　　－ ◆引数のラベルの仕様を統一化◇https://github.com/apple/swift-evolution/blob/master/proposals/0046-first-label.md◆など

**・ ▲ツールの品質向上▲**

　　－ コンパイラー／IDEの品質や速度の向上など

　今回はそれぞれ箇条書きの簡単な紹介にとどめたが、細かいところは今後の連載で補足していく予定である。

# ■Swiftは3.0で安定するのか？

　現時点で最新版のSwift 2.2まではソースコードの破壊的変更につながる言語仕様変更が多く、それが今後どうなるのかは気になるところだと思う。これも◆Swift Evolutionリポジトリ◇https://github.com/apple/swift-evolution◆などで公開されている情報から詳しく読み解くことができる。

## ●Swift 3.0以降もまだ破壊的な仕様変更は続く

　まずSwift 3.0では「それ以降（Swift 3.xや4以降）で合理的に可能な限り、ソースコードレベルで互換性を保つようにする」と記載されている。言い換えると、バージョンが上がるたびにコンパイルエラーが発生するようなことは少なくなるがゼロとはいえないということである。

　Swiftは発表当初から各種言語の良いところ取りの優れた言語仕様を持った言語であり、さらにその後の約2年で予想を上回る開発速度でさらに完成度を上げてきていると感じている。この開発速度は、これまで後方互換性を大胆に切り捨ててきたからこそなせる業であると思っている。例えばC#は後方互換性がかなりしっかりしている言語である（参考： 「◆見えてきたC# 7： C#の短期リリースサイクル化 - Build Insider◇http://www.buildinsider.net/column/iwanaga-nobuyuki/008◆」）が、それはつまり互換性の維持にそれなりの労力をかけている、ということである。Swiftの場合は、労力がフルに次のバージョンへの開発に向けられてきた。

　ただ、そのトレードオフとして、今までは開発者にとっての負担となることが多かった。言語仕様をキャッチアップしていく負担が大きめであったり、書籍やインターネット上のコード例がすぐに古くなってしまい、最新版のコンパイラーでは警告やコンパイルエラーが発生したりすることも多い（個人的にはこういう作業を含めて楽しめているが）。これまではやはりアーリーアダプター向けの言語という感じが否めないところもあった。

　「まだ仕様変更が続くのか」と残念に思う人もいるかもしれないが、筆者は「Swiftは3.0の段階でもまだ大きな進化の途中だ」と感じており、今の段階で無理に後方互換性を保つことを優先して言語仕様に妥協が生まれるよりずっと良いことだと思っている。これはトレードオフの問題であり、そういった仕様の破壊的変更が一切許容できない場合は、現状では枯れた言語（iOSアプリ開発の場合はObjective-C）を選択するしかないであろう。とはいえSwiftはバージョン3.0を境に、これまでと比べるとずっと安定していくように見受けられるので、多少敬遠していた慎重な開発者にも薦められる時期ではあると思う。

## ●ABIの安定はSwift 4に持ち越しされた

　△ABI（Application Binary Interface）△とは、プログラムのモジュール間のバイナリレベル（オブジェクトコードレベル）でのインターフェースのことである。Swift言語自体のバージョンが異なっても、ABIレベルで互換性が保障されていれば、コンパイルされたバイナリはリンクできる。

　ABIを安定させるには、実行時のデータ構造／呼び出し規約／ABIに影響を与えるであろう言語機能などを確定させる必要がある。2015年12月にSwift 3.0のマイルストーンが初めて提示された時点では、その目標として「◆ABIの安定対応」が含まれていた◇https://github.com/apple/swift-evolution/commit/06b69a6e51a71a462c268da60b51a18966dba31b◆が◆今年の5月に3.0では「見送る」と判断された◇https://www.infoq.com/jp/news/2016/06/swift-3-no-stable-abi◆。

　さらにこの後、ABIの安定化をSwift 4で最優先の達成事項として取り組むことが、「◆Swift 3の開発の振り返りとSwift4の計画が記された、メーリングリストの投稿（英語）◇https://lists.swift.org/pipermail/swift-evolution/Week-of-Mon-20160725/025676.html◆」で発表された。Swift 4に向けた議論はSwift 3開発と並行して2016年8月1日からスタートしており、またABIの安定化とともに、ソースの後方互換性の維持にも取り組む、とのことである（筆者による日本語での紹介記事: 「◆Swift 3の開発の振り返りとSwift 4の計画が記されたメールの紹介 - Qiita◇http://qiita.com/mono0926/items/c51b74fae679e1e39d2b◆」）。

## ●ABI互換がない現状の問題点

　つまり、現時点の2.2でも3系でもABIが安定しないので、Swiftのバージョンが上がると、以前のバージョンでビルドしたライブラリを開発中のアプリケーションで利用している場合にコンパイルエラー（リンクエラー）が発生してしまう。このコンパイルエラーを解決するには、古いライブラリを新しいバージョンのSwiftコンパイラーでビルドし直す必要がある。

　OSS（Open Source Software：オープンソースソフトウェア）ライブラリを使っているときなどには、自分たちが使用しているSwiftのバージョンに対応したソースコードがあれば、それを利用してリビルドできるが、そうでなければ、自らソースコードを対応させる必要が生じる。Swiftの場合、バージョンアップサイクルが早いこともあり、最新版やベータ版のSwiftを使おうとしたときに、ライブラリがまだそのバージョンに対応していなくて、そのような手間が発生することが多い。

　とはいえ、Swiftの定番人気OSSライブラリ（◆Alamofire◇https://github.com/Alamofire/Alamofire◆や◆RxSwift◇https://github.com/ReactiveX/RxSwift◆など、GitHubスターの数が数千以上程度のもの）であれば、大抵の場合はかなり早いタイミングでSwiftの最新版用のブランチで対応されるので、それを使ってビルドし直すことで解決することも多い。Swiftでサードパーティのライブラリを使う場合、このようにアップデートが迅速であるか、あるいはそうでない場合には、いざとなれば自力でコードの修正を施せるかどうか、などを視野に入れて選定する必要がある。

## ●ABIが互換すればSwiftで書いたアプリのバイナリサイズ削減効果もある

　また、ABIが安定すれば、アプリバイナリにSwiftのランタイムバイナリを含んで配布する必要がなくなるというメリットもある。上ではバージョンの違うSwiftコンパイラーでビルドしたアプリケーションとライブラリのリンクの際にABI互換の必要があると述べたが、ABIはアプリケーションバイナリとその実行環境についての互換性にも同様に関係する。

　ABIが安定すればSwiftのランタイムバイナリをアプリに含めるのではなく、OSに同梱できるようになる。具体的には現在、同じようにコーディングしたシンプルなアプリを配布する場合、Swiftで作成したアプリはObjective-Cで記述したものと比べて10Mbytes近くそのサイズが大きくなってしまう。ABIが安定化することでこの問題が解消されるはずである。

　以上をまとめると次の通りである。

・ ▲Swift 3.0まで：▲ 積極的な仕様変更・後方互換性切り捨て

・ ▲Swift 3.x： ▲ 互換性をなるべく保つようにはする

　　－ バージョンアップのたびにコード対応が必要な場面は減りそう

　　－ とはいえ、ABIは変更されていくので、古いバージョンでビルドしたライブラリの扱いに手間がかかる状態は継続

・ ▲Swift 4.0： ▲ ABI安定化とソースコード後方互換性が最優先事項

　　－ 当初は3.0で予定されていたが延期された

　初めの2年で高速に言語の完成度を高めて、その後の1年で互換性担保にフォーカス、というかなり効率の良い開発となっているように感じる。

# ■前編のまとめ

　前編では、Swift 3.0が間近に迫った中、いつどのような内容を含んでリリースされるかについての最新情報を伝えた。こういった情報は全てオープンになっていて誰でも深いところまで調べられる状態であり、後編ではそれら情報がどうまとめられているのかを紹介する。