Deep dive into Swift Literal

脱Swiftリテラル初心者

freddi @ LINE Fukuoka and HAKATA.swift

try! Swift Tokyo 2019, 21st March

こんにちは。freddiと申します。LINE Fukuokaでインターンシップエンジニアをしています。

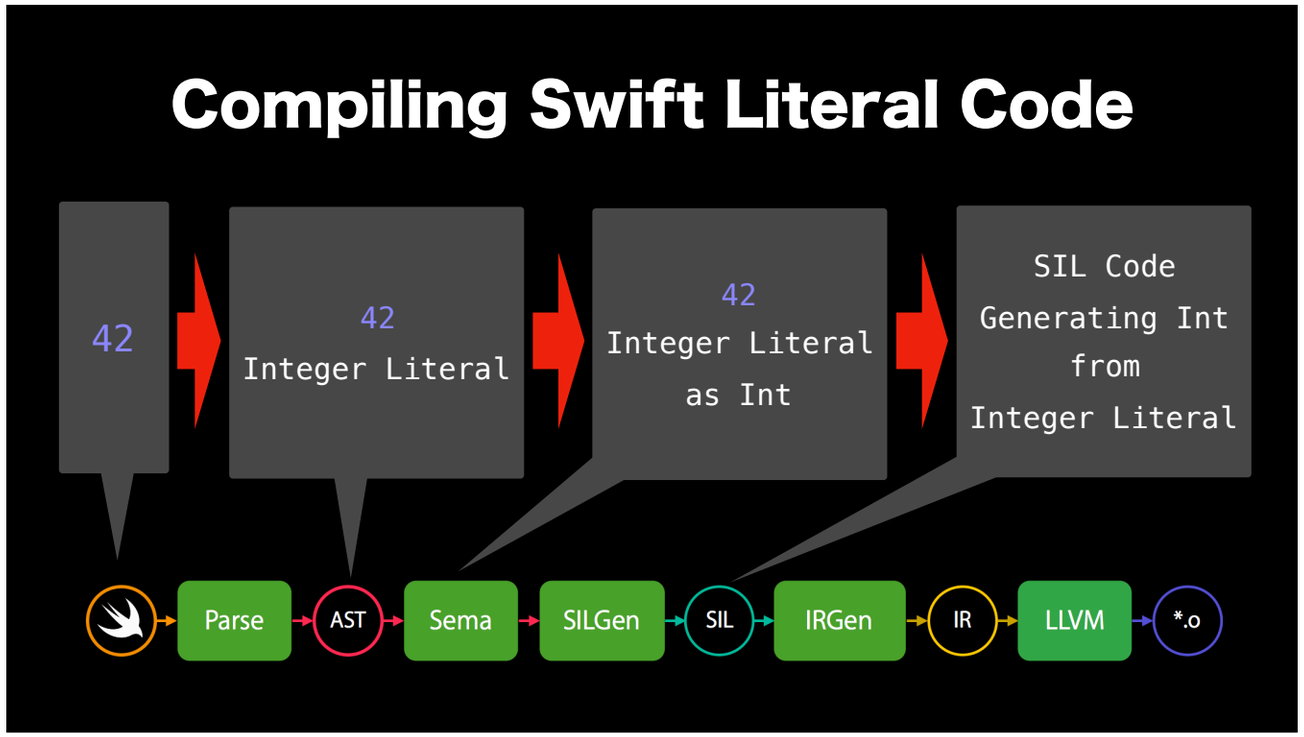
このトークでは、私達が普段触っているSwiftのリテラルをいつもとは違う面から理解するお話をさせていただきます。

From Swift Integer Literal To `Int`

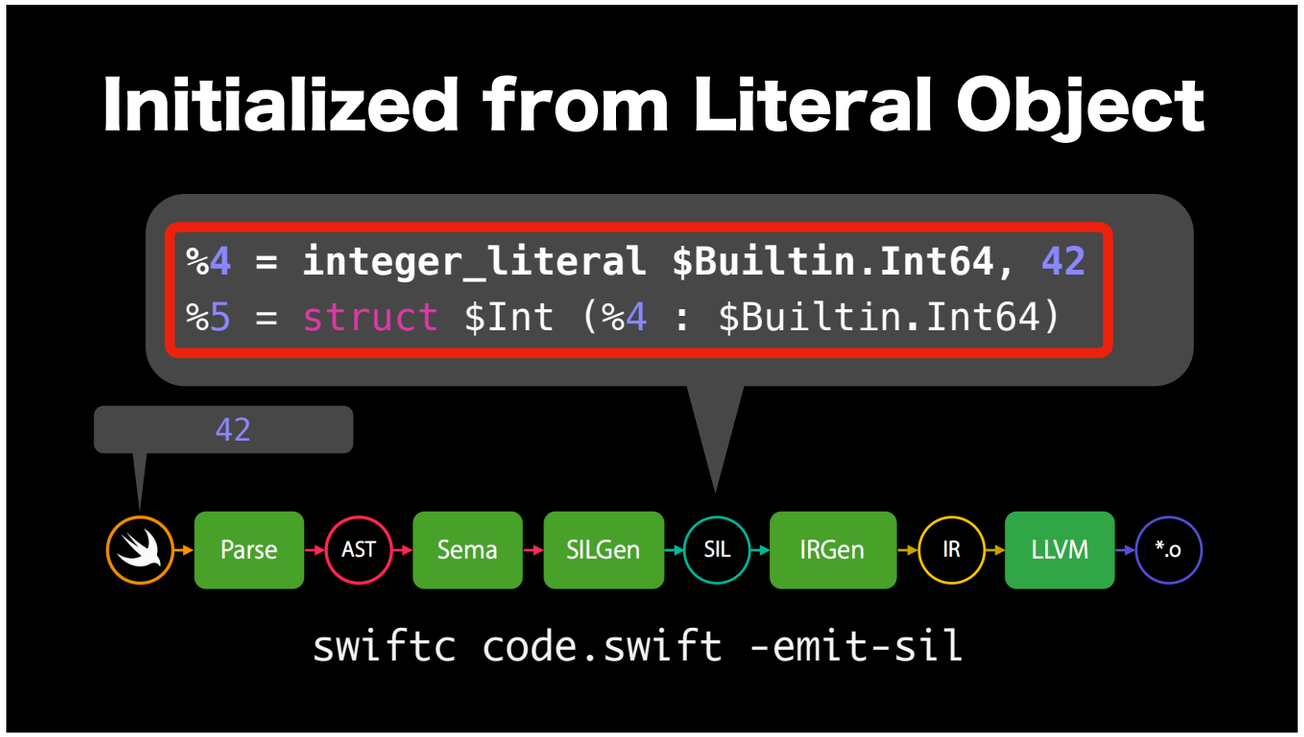
```
var myVariable = 42
```

これはSwift Tourにある、変数に整数リテラル42を代入しているだけのコードです。

この「４２という整数リテラルがSwiftのIntになる」までのフローを知ることで、いつもとは違う深い世界を見ることが出来ます。

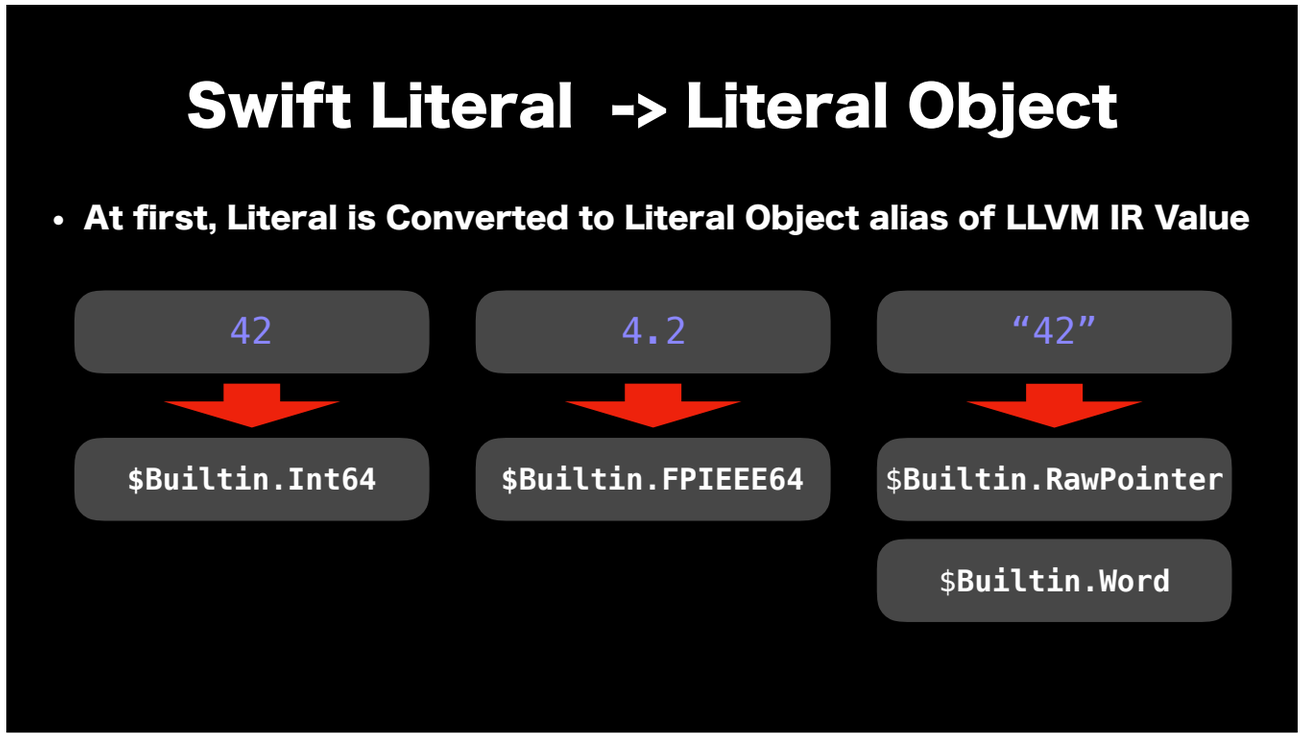


リテラルはコンパイルされる時、Parseというフェーズでどの種類のリテラルであることを解釈され、Semaというフェーズで解釈される型の情報が付与されます。
SwiftとLLVM IRの中間言語であるSILでは、型の情報とリテラルを利用してオブジェクトを作るコードを生成します。



Swiftコンパイラの-emit-sil(pronounse: emit sil)オプションを利用して、最終的に生成されるSILを生成しました。

Builtin.Int64という謎のオブジェクトが生成されて、解釈されたIntの型のイニシャライザに引数として渡されます。



この謎のオブジェクトは、SwiftリテラルがLLVM IRのリテラルのエイリアスになるオブジェクトに解釈されたもので、私はこれをリテラルオブジェクトと呼んでいます。
例えば、整数リテラルからはBuiltin.Int64という名前のオブジェクトが生成されます。

Literal Object will be stored

- **Literal Object will become stored property of Converted Value Structure**

```
public struct Int : FixedWidthInteger, SignedInteger {
  // ...
  public var _value: Builtin.Int64

  public init(_ _value: Builtin.Int64) {
    self._value = _value
  }
```

そして、リテラルオブジェクトは、先程の解釈される型のstored propertyになります。

Intであれば、整数のリテラルオブジェクトの型の_value（pronounce: underscore value）という型の変数にstoreされます。

Editing Value with Literal Object

- **Referencing Literal Object information When editing value**

```
%3 = struct_extract %0 : $Int, #Int._value      // user: %6
%4 = struct_extract %1 : $Int, #Int._value      // user: %6
%5 = integer_literal $Builtin.Int1, -1          // user: %6
%6 = builtin "sadd_with_overflow_Int64" %3 : $Builtin.Int64 ...
```

Reading _value from Int

_value + _value by LLVM IR Function

そして、演算子などでIntを弄るときはこのリテラルオブジェクトの情報を元に、LLVM IRのレイヤーで値を操作します。

例えば、Intのプラス演算子を呼んだとき、SILでは先程の_value　(pronounce: underscore value)を利用して、LLVM IRレイヤーの整数同士の足し算である関数に引数として渡しています。

Expressible by Literal

```
var myVariable = 42
```

次は、この「リテラルから直接、オブジェクトが作成できること」について着目してみましょう。
実は、このような、「型がリテラルによって表現できる」ことには、プロトコルが大きく関わっています。

Expressible by Literal?

Initalizable from Literal by Type Annotation

```
var myVariable: CGFloat = 42
```

Implicitly Converted from Float Literal to CGFloat

```
func bar(_ value: CGFloat) {}
bar(42.0) // 42 -> CGFloat
```

型をリテラルで表現可能ということの例は、私達が使っているCGFloatで見ることができます。

CGFloatの型アノテーションがついている変数宣言ではリテラルの代入が可能だったり、引数がCGFloatの場合に浮動小数点リテラルを渡すと暗黙的にCGFloatになります。

Intrinsic Protocol

- Protocols make Type having feature by Conforming
  - e.g.) CaseIteretable, …
- We can see Intrinsic protocols at CompilerProtocol.swift

準拠すると、コンパイラにより様々な機能がサポートされる、コンパイラ組み込みのプロトコルがあります。
CaseIteratableも、この組み込みのプロトコルの１つです。
SwiftコンパイラのCompilerProtocol.swiftを見れば、どういう組み込みのプロトコルがあるかを知ることが出来ます。

Literal Intrinsic Protocol

- Make types possible to Expressible by each Literal
  - ExpressibleByIntegerLiteral
  - ExpressibleByFloatLiteral
  - ExpressibleByStringLiteral
  - ExpressibleByBooleanLiteral
  - ExpressibleBy …

```
var myVariable = 42
```

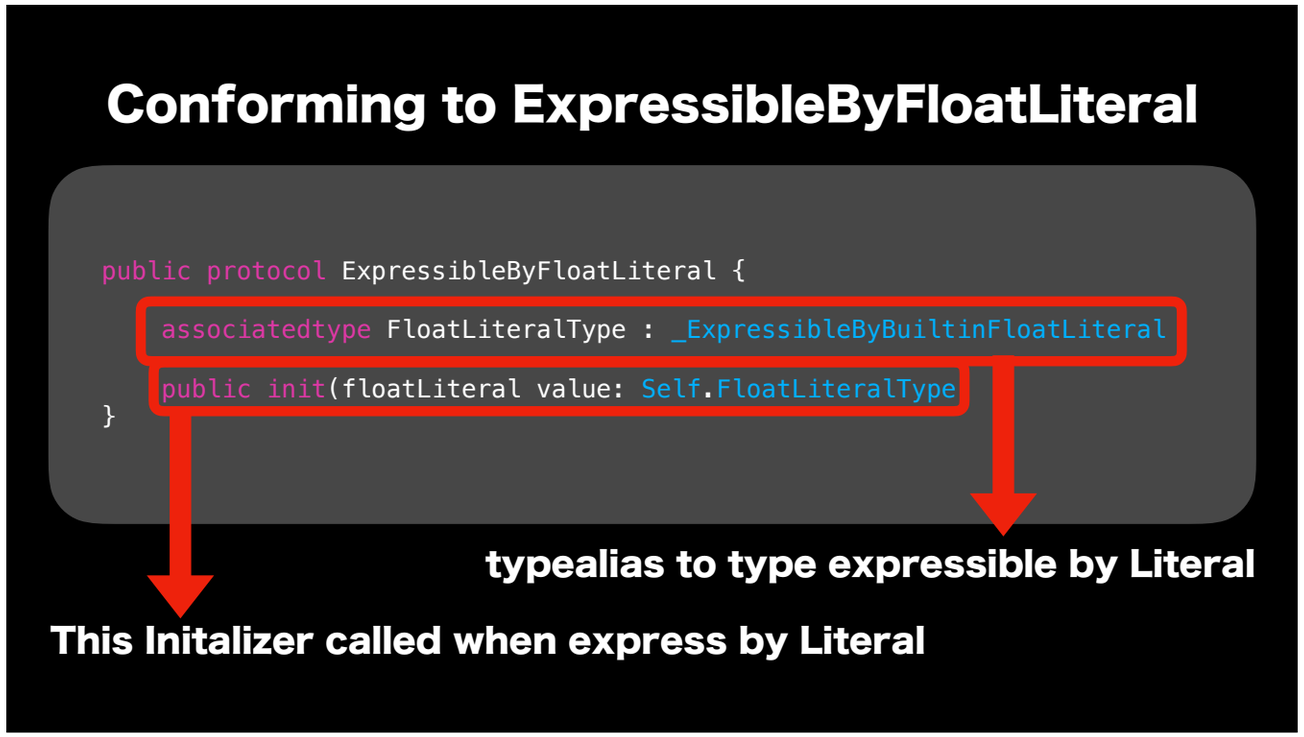
```
var myVariable = 42.0
```

```
var myVariable = "try!"
```

```
var myVariable = true
```

組み込みのプロトコルのうち準拠した型が、Swiftに存在するリテラルを利用して表現可能になるものが存在します。

リテラルから生成できるIntやFloatのような型は、このプロトコルに準拠しています。



このプロトコルへの準拠の流れを、浮動小数点リテラルが対象のExpressibleByFloatLiteralを例にして見てみましょう。

associatedtypeとして、浮動小数点リテラルが表現可能な型をtypealiasし、その型のオブジェクトを引数に取るイニシャライザを実装します。

対象のリテラルから型が表現される時、このイニシャライザが呼ばれます。

CGFloat is Float type for CoreGraphics

```
public struct CGFloat: ExpressibleByFloatLiteral {

#if arch(i386) || arch(arm)
    public typealias NativeType = Float
#elseif arch(x86_64) || arch(arm64)
    public typealias NativeType = Double
#endif

    public typealias FloatLiteralType = NativeType
```

- **Changing associatedType by CPU Architecture**
  - **To handle CoreGraphics effectively by each env**

リテラルによる表現を可能にする組み込みのプロトコルを上手く活用している良い例が、CGFloatです。

CGFloatはassociatedtypeをCPUアーキテクチャによってコンパイル時に変更しており、それぞれの環境で効率的に扱えるCoreGraphicsのFloat型を提供しています。

Swift 5.0 changes with ExpressibleBy…Literal

- **SE-0213**

```
UInt(0xffff_ffff_ffff_ffff)
```

4.2 → 0xffff_ffff_ffff_ffff -> Int
Error: overflow!

5.0 → 0xffff_ffff_ffff_ffff -> UInt
No Error!

- **Let's try Swift 5.0 comparing SIL with 4.2!**

そして、リテラルが関わるSwift5.0からのある変更点をご紹介します。
ExpressibleBy…Literalに準拠した型が特定のイニシャライザを呼ぶとき、リテラルが最初に解釈される型の優先度が変わります。
この変更点ですが、4.2と5.0のSILをそれぞれ読んでみてご自身の手でこのトークの復習としてtryしてみてください。詳しい体験の方法は、このあと共有するスライドに加筆します。

What we learned?

- From Swift Literal to Swift Type, with Literal Object
  - How it works in LLVM IR literal
- Expressing by Literal is feature of Intrinsic protocol
  - CGFloat is Float Type for CoreGraphics

このトークでは、普段見るコードから、リテラルについて、深いレイヤーを見ないと、わからないようなところをちょっとだけ解説してみました。

もし、何気なく書いているSwiftのコードから、リテラルを大きく知るキッカケや、Swiftへの深いレイヤーへの興味の種になれば、私も嬉しいです。
今日はありがとうございました。