

SILOptimizerのCode Reading入門

freddi@LINE Fukuoka and HAKATA.swift

2019/05/10 Swift愛好会 談義枠

誰お前

freddi (@___freddi___)

新卒 Engineer@LINE Fukuoka

Co-organizer@HAKATA.swift

try! Swift NYC 2019 Speaker (の予定)

今日のおはなし

SILOptimizerのCode Readingと一緒に挑戦

SIL?

SILOptimizer?

SIL

Swift Code(のAST) と LLVM IRの間にある中間言語

最適化される前のraw SILとあとのcanonical SILがある



SILOptimizer

SILGenというモジュールからraw SILが出力される

SILOptimizerで最適化されたcanonical SILにする

SIL



SILOptimizerのCode Reading入門


SILOptimizerのコードを読んでみよう
























<https://github.com/apple/swift/tree/master/lib/SILOptimizer>




でも難しい？

長いし、C++だし、、、😂

Tree: 123fee960e ▾ [swift](#) / [lib](#) / [SILOptimizer](#) / [Transforms](#) / **AllocBoxToStack.cpp** Find file Copy path

 **slavapestov** SIL: Use better type lowering APIs in a couple of spots 5847e16 on Mar 6

23 contributors                       

924 lines (778 sloc) 34 KB Raw Blame History   

```
1 //==== AllocBoxToStack.cpp - Promote alloc_box to alloc_stack =====//
2 //
3 // This source file is part of the Swift.org open source project
4 //
5 // Copyright (c) 2014 - 2017 Apple Inc. and the Swift project authors
6 // Licensed under Apache License v2.0 with Runtime Library Exception
7 //
8 // See https://swift.org/LICENSE.txt for license information
9 // See https://swift.org/CONTRIBUTORS.txt for the list of Swift project authors
10 //
```

SILOptimizer/Transforms/AssumeSingleThreaded.cpp

AssumeSingleThreaded

swiftcの-assume-single-threaded オプションで発動

シングルスレッド向けのバイナリにコンパイルする

AssumeSingleThreaded.cpp

めっちゃ短い (60 lines)

見るのは正直モジュールの定義部分でいい

読んでみよう

```
namespace {  
class AssumeSingleThreaded : public swift::SILFunctionTransform {  
    /// The entry point to the transformation.  
    void run() override {  
        if (!getOptions().AssumeSingleThreaded)  
            return;  
        for (auto &BB : *getFunction()) {  
            for (auto &I : BB) {  
                if (auto RCInst = dyn_cast<RefCountingInst>(&I))  
                    RCInst->setNonAtomic();  
            }  
        }  
        invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);  
    }  
};  
} // end anonymous namespace
```

```
namespace {      SIL Functionを書き換えるモジュールが継承すべきもの
class AssumeSingleThreaded : public swift::SILFunctionTransform {
    /// The entry point to the transformation.
    void run() override {
        if (!getOptions().AssumeSingleThreaded)
            return;
        for (auto &BB : *getFunction()) {
            for (auto &I : BB) {
                if (auto RCInst = dyn_cast<RefCountingInst>(&I))
                    RCInst->setNonAtomic();
            }
        }
        invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);
    }
};
} // end anonymous namespace
```


SILTransform

(PassManager/Transforms.h#L25)

```
/// The base class for all SIL-level transformations.  
class SILTransform : public DeleteNotificationHandler {  
public:  
    /// The kind of transformation passes we use.  
    enum class TransformKind {  
        Function,  
        Module,  
    };  
};
```


**“SILTransformというものがある”
で深追いしない**

```
namespace {      SIL Functionを書き換えるモジュールが継承すべきもの
class AssumeSingleThreaded : public swift::SILFunctionTransform {
    /// The entry point to the transformation.
    void run() override { Optionが指定されてなければ何もしない
        if (!getOptions().AssumeSingleThreaded)
            return;
        for (auto &BB : *getFunction()) {
            for (auto &I : BB) {
                if (auto RCInst = dyn_cast<RefCountingInst>(&I))
                    RCInst->setNonAtomic();
            }
        }
        invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);
    }
};
} // end anonymous namespace
```

AssumeSingleThreaded

swiftcの-assume-single-threaded オプションで発動

シングルスレッド向けのバイナリにコンパイルする

```
namespace { SIL Functionを書き換えるモジュールが継承すべきもの
class AssumeSingleThreaded : public swift::SILFunctionTransform {
    /// The entry point to the transformation.
    void run() override { Optionが指定されてなければ何もしない
        if (!getOptions().AssumeSingleThreaded)
            return;
        for (auto &BB : *getFunction()) { SILのFunctionをget
            for (auto &I : BB) {
                if (auto RCInst = dyn_cast<RefCountingInst>(&I))
                    RCInst->setNonAtomic();
            }
        }
        invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);
    }
};
} // end anonymous namespace
```

SILTransform::getFunction

(PassManager/Transforms.h#L127)

```
/// Reoptimize the current function by restarting the pass
/// pipeline on it.
void restartPassPipeline() { PM->restartWithCurrentFunction(this); }

SILFunction *getFunction() { return F; }

void invalidateAnalysis(SILAnalysis::InvalidationKind K) {
    PM->invalidateAnalysis(F, K);
}
};
```


SILFunction(SIL/SILFunction.h#L113)

```
/// SILFunction – A function body that has been lowered to SIL. This consists of  
/// zero or more SIL SILBasicBlock objects that contain the SILInstruction  
/// objects making up the function.
```

```
class SILFunction
```

```
    : public llvm::ilist_node<SILFunction>, public SILAllocated<SILFunction> {  
public:  
    using BlockListType = llvm::iplist<SILBasicBlock>;
```

SILFunction(SIL.rst#functions)

Functions

```
decl ::= sil-function
sil-function ::= 'sil' sil-linkage? sil-function-name ':' sil-type
               '{' sil-basic-block+ '}'
sil-function-name ::= '@' [A-Za-z_0-9]+
```

```
namespace { SIL Functionを書き換えるモジュールが継承すべきもの
class AssumeSingleThreaded : public swift::SILFunctionTransform {
    /// The entry point to the transformation.
    void run() override { Optionが指定されてなければ何もしない
        if (!getOptions().AssumeSingleThreaded)
            return;
        for (auto &BB : *getFunction()) { SILのFunctionをget
            for (auto &I : BB) {
                if (auto RCInst = dyn_cast<RefCountingInst>(&I))
                    RCInst->setNonAtomic();
            }
        }
        invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);
    }
};
} // end anonymous namespace
```



```

namespace {
    SIL Functionを書き換えるモジュールが継承すべきもの
    class AssumeSingleThreaded : public swift::SILFunctionTransform {
        /// The entry point to the transformation.
        void run() override {
            Optionが指定されてなければ何もしない
            if (!getOptions().AssumeSingleThreaded)
                return;
            for (auto &BB : *getFunction()) {
                SILのFunctionをget
                for (auto &I : BB) {
                    SILのFunction中のBasic Blockを取得
                    if (auto RCInst = dyn_cast<RefCountingInst>(&I))
                        RCInst->setNonAtomic();
                }
            }
            invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);
        }
    };
} // end anonymous namespace

```

```

namespace {
    SIL Functionを書き換えるモジュールが継承すべきもの
    class AssumeSingleThreaded : public swift::SILFunctionTransform {
        /// The entry point to the transformation.
        void run() override { Optionが指定されてなければ何もしない
            if (!getOptions().AssumeSingleThreaded)
                return;
            for (auto &BB : *getFunction()) { SILのFunctionをget
                for (auto &I : BB) { SILのFunction中のBasic Blockを取得
                    if (auto RCInst = dyn_cast<RefCountingInst>(&I))
                        RCInst->setNonAtomic();
                }
            }
            Basic Blockがリファレンスカウンタ関係ならnonAtomicに
            invalidateAnalysis(SILAnalysis::InvalidationKind::Instructions);
        }
    };
} // end anonymous namespace

```

SIL読んで確認しよう！！！！

```
swiftc -emit-sil test.swift -o test-noAssumedSingleThread.sil
```

```
swiftc -emit-sil test.swift -assume-single-threaded -o test-assumedSingleThread.sil
```




まとめ

短いことから無理せずに読めばSwiftコンパイラのコードは入門可能

-> 特に「小さいモジュールから見てみる」とかおすすめ

SIL関連ならSIL.rstを読むといいかも

変数名からそれが何かを察する力があれば楽に読める

数多のソースコードを深掘りする必要がなくなる

参考文献

apple/swiftの各ソース/Doc（それぞれのページにリンクあり）

kitasukeさんの SIL Optimizations - AllocBoxToStack-

<https://www.slideshare.net/kitasuke/sil-allocboxtostack>