External Resources

The official Swift blog has a lot of useful information, such as how runtime reflection works and how the compiler's new diagnostic architecture is structured.

This page lists some external resources apart from the Swift blog which may be helpful for people interested in contributing to Swift. The resources are listed in reverse-chronological order and organized by topic.

Contributing Guides and Tips

- Steps for first PR and asking questions by Varun Gandhi (Swift forum comment, Dec 2019).
- Contributing to Swift for the First Time by Robert Pieta (talk, Jun 2019): This talk describes Robert's experience contributing to `swift-corelibs-foundation` and later `swift`, providing a step-by-step guide for new contributors to get their feet wet.
- Contributing to Swift compiler by Yusuke Kita (lightning talk, Apr 2019).
- Contributing to Swift (Part 1, Part 2) by Suyash Srijan (blog post series, Dec 2018).
- Setting up a compile-edit-test cycle by Robert Widmann (Swift forum comment, May 2018)
- Becoming an Effective Contributor to Swift by Harlan Haskins and Robert Widmann (talk, Apr 2018): Covers the following topics:
  - The organization of different projects part of the Swift monorepo.
  - How to set up an environment for building Swift.
  - Common utilities and compiler flags for debugging different stages.
  - Common build configurations.
  - Tips and tricks for testing: early exiting, filtering tests, testing Swift and Objective-C together, using `PrettyStackTrace`.
  - A live demo fixing a crash:
    - Copying program arguments into Xcode for debugging.
    - Using `dump()` for print debugging.
- Getting Started with Swift Compiler Development by Brian Gesiak (blog post series, Aug 2017 - Jun 2018)
- Contributing to Open Source Swift by Jesse Squires (talk, Mar 2016): Covers the following topics:
  - The overall compiler pipeline.
  - The organization of different projects part of the Swift monorepo, including some "difficulty levels" for different projects.
  - Tips for contributing effectively.

AST

- The secret life of types in Swift by Slava Pestov (blog post, Jul 2016): This blog post describes the representation of Swift types inside the compiler. It covers many important concepts: `TypeLoc` vs `TypeRepr` vs `Type`, the representation of generic types, substitutions and more.

libSyntax and SwiftSyntax

- An overview of SwiftSyntax by Luciano Almeida (blog post, Apr 2019): This post provides a quick tour of libSyntax and SwiftSyntax.
- Improving Swift Tools with libSyntax by Harlan Haskins (talk, Sep 2017): This talk describes the design of libSyntax/SwiftSyntax and discusses some useful APIs. It also describes how to write a simple Swift formatter using the library.

Type checking and inference

- Implementing Swift Generics by Slava Pestov and John McCall (talk, Oct 2017): This talk dives into how Swift's compilation scheme for generics balances (a) modularity and separate compilation with (b) the ability to pass unboxed values and (c) the flexibility to optionally generate fully specialized code. It covers the following type-checking related topics: type-checking generic contexts, requirement paths and canonicalized generic signatures.
- A Type System from Scratch by Robert Widmann (talk, Apr 2017): This talk covers several topics related to type-checking and inference in Swift:
  - Understanding sequent notation which can be used to represent typing judgments.
  - An overview of how bidirectional type-checking works.
  - Examples of checking and inferring types for some Swift expressions.
  - Interaction complexity of different type system features with type inference.
  - Type variables and constraint graphs.

SIL

- Ownership SSA by Michael Gottesman (talk, Oct 2019): This talk describes efficiency and correctness challenges with automatic reference counting and how including ownership semantics in the compiler's intermediate representation helps tackles those challenges.
- How to talk to your kids about SIL type use by Slava Pestov (blog post, Jul 2016): This blog post describes several important SIL concepts: object vs address types, AST -> SIL type lowering, trivial vs loadable vs address-only SIL types, abstraction patterns and more.
- Swift's High-Level IR by Joe Groff and Chris Lattner (talk, Oct 2015): This talk describes the goals and design of SIL. It covers the following:
  - Some commonly used SIL instructions and how they are motivated by language features.
  - Some early passes in SIL processing, such as mandatory inlining, box-to-stack promotion and definite initialization.
  - Why SIL is useful as an intermediate representation between the AST and LLVM IR.

Code generation, runtime and ABI

- Bringing Swift to Windows by Saleem Abdulrasool (talk, Nov 2019): This talk covers the story of bringing Swift to Windows from the ground up through an unusual route: cross-compilation on Linux. It covers interesting challenges in porting the Swift compiler, standard library, and core libraries that were overcome in the process of bringing Swift to a platform that challenges the Unix design assumptions.
- The Swift Runtime by Jordan Rose (blog post series, Aug-Oct 2020): This blog post series describes the runtime layout for different structures, runtime handling for metadata, as well as basic runtime functionality such as retain/release that needs to be handled when porting Swift to a new platform, such as Mac OS 9.
  - Part 1: Heap Objects
  - Part 2: Type Layout
  - Part 3: Type Metadata
  - Part 4: Uniquing Caches
  - Part 5: Class Metadata
  - Part 6: Class Metadata Initialization
  - Part 7: Enums
- Implementing the Swift Runtime in Swift by JP Simard and Jesse Squires with Jordan Rose (podcast episode, Oct 2020): This episode describes Jordan's effort to port Swift to Mac OS 9. It touches on a wide variety of topics, such as ObjC interop, memory layout guarantees, performance, library evolution, writing low-level code in Swift, the workflow of porting Swift to a new platform and the design of Swift.
- How Swift Achieved Dynamic Linking Where Rust Couldn't by Alexis Beingessner (blog post, Nov 2019): This blog post describes Swift's approach for compiling polymorphic functions, contrasting it with the strategy

used by Rust and C++ (monomorphization). It covers the following topics: ABI stability, library evolution, resilient type layout, reabstraction, materialization, ownership and calling conventions.

- arm64e: An ABI for Pointer Authentication by Ahmed Bougacha and John McCall (talk, Oct 2019): This talk does not mention Swift specifically, but provides a good background on the arm64e ABI and pointer authentication. The ABI affects parts of IR generation, the Swift runtime, and small parts of the standard library using unsafe code.
- Exploiting The Swift ABI by Robert Widmann (talk, July 2019): This talk is a whirlwind tour of different aspects of the Swift ABI and runtime touching the following topics: reflection, type layout, ABI entrypoints, Swift's compilation model for generics and archetypes, witness tables, relative references, context descriptors and more.
- Efficiently Implementing Runtime Metadata by Joe Groff and Doug Gregor (talk, Oct 2018): This talk covers the use of relative references in Swift's runtime metadata structures. After describing some important metrics impacted by the use of dynamic libraries, it goes through the different kinds of relative references used in the Swift runtime and the resulting tooling and performance benefits.
- Coroutine Representations and ABIs in LLVM by John McCall (talk, Oct 2018): This talk describes several points in the design space for coroutines, diving into important implementation tradeoffs. It explains how different language features can be built on top of coroutines and how they impose different design requirements. It also contrasts C++20's coroutines feature with Swift's accessors, describing the differences between the two as implemented in LLVM.
- Implementing Swift Generics: This talk is mentioned in the type-checking section. It also covers the following code generation and runtime topics: value witness tables, type metadata, abstraction patterns, reabstraction, reabstraction thunks and protocol witness tables.