Kotlin Compiler Reading Notes

Yuxiang Jiang *

March 15, 2020

Contents

*yxjiang@linkedin.com

1 Methodology

It's often daunting to read through huge and complex codebase like the Kotlin compiler! Fortunately we have great tools at hand to deal with such complexity. In particular, IntelliJ Idea provides many priceless code navigation tools:

- Type Hierarchy (under Navigate)
- Structure (under View - Tool Windows)
- Find Usage
- Breakpoints

With these we'll be able to understand the compiler architecture and internals bit by bit...

2 Tl;dr: Pipeline

- Kotlin source
- KtElement (Psi)

- core.descriptor (ClassicTypeSystemContext)
- FirElement (nodes) / FirBasedSymbol (infotable?) (ConeTypeContext)
- IrElement (nodes) / IrSymbol (infotable?) (IrTypeSystemContext)

2.1 Source → KtElement

- See class KtVisitor for an overview of many Kotlin PsiElements
- compiler.psi defines KtElement and Stubs, also provides parser
  - Stubs represents the interface parts of a kt compilation unit (.h, .mli, .hi etc).
  - There's also a LighterASTNNode (KotlinLightParser) – what is it? (Flyweight pattern is like hash consing)
  - Some KtElements are related to types: e.g. KtTypeReference (some of them are not KtElements but stubs!)
- compiler.psi.KtPsiFactory is the entrypoint for creating KtFile (a PsiElement) from source text.

2.2 KtElement → DeclarationDescriptor

DeclarationDescriptor seems to be a high-level IR with type analyzed. It contains both the expr tree and the types.

2.2.1 LazyTopDownAnalyzer

compiler.frontend.LazyTopDownAnalyzer seems to be the psi analyzer. (Is there another analyzer that's not lazy?)

analyzeDeclarations returns an AnalysisResult which contains a ModuleDescriptor and a BindingContext.

2.3 DeclarationDescriptor → FIR

FIR (compiler.fir) seems to be an intermediate yet still high-level IR.

- See generated class FirVisitor for an overview of many of the Fir exprs
- compiler.frontend (not sure which step it is in, but it at least does symbol resolution, type checking, (Psi → CFG?)
  - See key classes: AnalysisResult, BindingContext, BindingTrace (records the collected binding / type substs?)
- compiler.resolution: tower/ReceiverValue etc
- compiler.fir: cones (types and symbols used in Fir?), fir2ir (lowering to Ir), psi2fir (lowering Psi to Fir), resolve, jvm, tree (Fir definitions and impl for psi2fir)

  - cones: StandardClassIds contains a bunch of core Kt (read: not JVM) type Ids. They have a JVM-like fqname.

    SyntheticCallableId contains when/try/nullcheck synthetic call exprs
  - tree.gen contains all Fir expressions (see tree.tree-generator's readme for how they are generated), as well as extra info (FirTypeRef). And even on Fir level, the generic types are not yet erased (FirTypeProjectionWithVariance)

2.4 FIR → IR

IR (compiler.ir) seems to be an lower-level IR.

2.5 Example stacktrace for running a Kotlin script

E.g. `kotlin -e <expr>`

- CLIDriver
  - compiler.cli / K2JVMCompiler
  - plugins.scripting-compiler
  - compiler.cli / TopDownAnalyzerFacadeForJVM (analyzeFilesWithJavaIntegration)
- Analyzer
  - compiler.frontend / LazyTopDownAnalyzer.analyzeDeclarations
  - (HUGE) go through all stmts
  - BodyResolver.resolveBodies
  - DeclarationChecker.process
  - (HUGE) go through files, annotations, class's modifiers, idents, header (super+generic bounds); function, property, destructionDecl, typealias's modifiers and idents.

3 KtElement

KtFile and KtScript for toplevel decls, KtClass for `class Foo`, KtNamedFunction for `fun foo()`, KtProperty for `val foo`.

4 DeclarationDescriptor

See core.descriptors.DeclarationDescriptorVisitor for an overview.

DeclarationDescriptorVisitor sounds like an type-instantiated/abstracted wrapper of an element. Also has a bunch of annotations (AnnotationDescriptor) and a name. And is also a tree node (has parent: getContainingDecl)

4.1 Survey of Class Hierarchy

Looks that descriptors are something that's used throughout the whole compilation pipeline. They are more often used in frontend, but even in backend I can see some usage of them.

4.1.1 CallableD: VisD & NonRootD & Subst

Receiver types (dispatch / extension), arg types, return types, type params; Parameter names, names may be unstable/synthesized (e.g. from JVM object code) Parameter values (what is ValueParameterDescriptor?) Cross ref to overridden methods User-DataKey¡A¿: stores typed user data

4.1.2 MemberD: VisD & NonRootD

Has member modifiers: ‘expect‘ / ‘actual‘ / ‘external‘.

4.1.3 CallableMemberD: CallableD & MemberD

Kind: decl / delegation / fakeOverride / synthesized (what's the last two?) Has a copy(owner, modality[final,sealed,open,abstract], visibility, kind, copyOverrides) method.

4.1.4 FunD: CallableMemberD

initialSignatureD: the initial D before renaming (didn't find SimpleFunctionD.rename)
hiddenToOvercomeSignatureClash: hack to handle corner case signature clash (said see nio.CharBuffer)
hiddenEverywhereBesideSupercalls: undocumented, another hack
Function modifiers: infix/inline/operator/suspend/tailrec

4.1.5 FunDImpl: NonRootDImpl & FunD

Base impl for function modifiers. Setters set the local modifier (mostly happen during conversion from KtElement), while some getters (infix, operator) respect super class methods.

Base impl for substitution (doSubstitute), and substituted value param. Worth reading.

Base impl for initialize.

Only here documents hiddenToOvercomeSignatureClash and hiddenEverywhere-BesideSupercalls: former makes the function completely hidden (even in super-call), latter permits super-call and propagates to overriden methods

4.1.6 ConD: FunD

containingD: ClassifierDWithTypeParams (what is this?)
constructedClass: ClassD

4.1.7 ClassConD: ConD

Just a bunch of return type specializations

4.1.8 ClassConDImpl: FunDImpl & ClassConD

Default (`<init>`) or synthesized.

Has a way to calculate dispatchReceiverParam. If inner, init's receiver is outer class instance (Whereas Java's outer class instance is passed as a param. Though in compiled code there's no difference, just at descriptor level they are different.); else null.

CommonizedClassConD: ClassConDImpl ClassConDImpl with source and originalD stripped.

DefaultClassConD: ClassConDImpl Read as (default constructor) of given class, not default implementation of (any class constructor). So this is just the no-param constructor of a class. Its visibility depends on the classD's visibility.

DeserClassConD: ClassConDImpl & DeserCallableMemberD ProtoBuf based deserialized ClassConD. Holds a bunch of TypeTable, NameResolver, ContainerSource etc to help with further deserialization (so this is also in some sense lazy).

JClassConD: ClassConDImpl & JCallableMemberD A Java imported class's constructor. Since this is from JVM object code, it provides property impl of hasStableParamNames / hasSynthesizedParamNames.

enhance() implementation makes a copy with enhanced receiver param and value params.

SamAdapterClass: ClassConDImpl & SamAdapterD<JavaClassConD>

4.2 DFactory

In core.descriptors / resolve.

5 Fir

compiler.fir.resolve: ResolutionStage sounds like something pipeline-related

6 Ir

compiler.ir.tree: IR (called IrSymbol) definitions. See IrSymbolVisitor for an overview. Looks that they have descriptors attached.

7 Type System

7.1 core.type-system

type system core (equality, bounds checking etc). However this is more of an interface module – the actual impls are in core.descriptors, fir and ir modules.

7.2 TypeSystemTypeFactoryContext

Contains a bunch of common type factories:

- flexibleType has lower/upper bounds
- simpleType has tycon, tyargs, nullablep
- tyarg has ty and variance
- star has tyarg (why?)
- there's also an error type used in diagnosis

7.3 TypeCheckerProviderContext

- modular axioms (errorType unifiable with all types etc)
- what is a stub type? (Probably PsiStub related?)

7.4 TypeSystemCommonSuperTypesContext

Used to check if two type has common super types, and lowest-common ancestor utils.
typeDepth is a safe overestimation of the depth (from 'Any').
Seems to also be used in Fir.

7.5 TypeSystemInferenceExtensionContext

Inference related.

7.5.1 Questions

- What is a isCapturedTypeConstructor?
- What is a singleBestRepresentative?
- What is a noInferAnnotation?
- What is mayBeTypeVariable?
- What is a defaultType?
- Read impl of isUnit vs isUnitTypeConstructor

- Read impl of createCapturedType
- Read impl of createStubType
- Read impl createEmptySubstitutor, typeSubstitutorByTypeConstructor, safeSubstitute

7.6 TypeSystemContext

Defines many marker types (guess it's used for disjoint classes). Also lots of getters on various marker types – Makes the code less intuitive…

- fastCorrespondingSupertypes has no actual impl? (No, it's just that intellij's search functionality fail to find overridden extension methods)
- isCommonFinalClassConstructor is implemented in three (Psi, Fir, Ir) stage's TypeSystemContext:
  - ClassicTypeSystemContext: get ClassDescriptor from TypeConstructor's declarationDescriptor, then check it's final but not (enum or annotation). So the method really checks that the tycon is "final" but is not a uncommon (enum/annotation) class.
  - ConeTypeContext: Does almost the same thing, but also return true if is anonymous object (final by design). Works on FirBasedSymbol (some sort of class infotable?). Check that this is a FirRegularClassSymbol, whose FirRegularClass is final but not uncommon.
  - IrTypeSystemContext: Check this is a IrClassSymbol whose owner is final and not uncommon.

    So basically ClassDescriptor, FirRegularClass and IrClassSymbol.owner are the same thing across three stages.

    Sounds that reading the common implemented methods of these three TySysCtx impl classes would be super helpful to understand the stages.

7.7 Type System for Fir

Read ConeTypeContext

7.8 Type System for Ir

Read IrTypeSystemContext

7.9 compiler.resolution/.inference

Type inference? constraint system, subst, fresh tycon, tyvar etc

7.10 compiler.frontend/.types

TypeIntersector (unify), DeferredType (I guess this is for when inference can't proceed at some first, and will retry when it has more information. Not really fully bidirectional (H-M style) type inference, but an approximation)

7.11 types.expressions

Contains a bunch of KtElement visitors that does type recon/checking:

- ExpressionTypingVisitorDispatcher
- ControlStructureTypingVisitor
- FunctionsTypingVisitor
- BasicExpressionTypingVisitor (constants etc)
  - This actually does a bit of parsing/validation work… e.g. understore on int literals.
  - Also uses ConstantExpressionEvaluator to check for possible compile time constants (this indeed sounds like something a parser would do).

    Folds boolean && and ||

    Look up simple unary and binary func in OperationsMapGenerated

8 Analysis

8.1 compiler.frontend.LazyTopDownAnalyzer

8.1.1 TopDownAnalysisContext

Stores the toplevel declarations (in typed maps) found during analysis.

8.2 BindingTrace

Has a BindingContext. Is writable. Can record/inquiry KotlinType for a KtElement. Impls: BindingTraceContext and ObservingBindingTrace

8.3 BindingContext

Sounds like a read-only counterpart to the BindingTrace.

8.4 Smartcasting

compiler.frontend smartcasts.DataFlowInfo: bunch of maps to stores the data flow analysis result useful for smart casts.

DataFlowValue: one instance of a value in a dataflow

DataFlowValue.Kind: classify exprs into smart cast enabled, possible, or disabled ones. Quite intuitive.

IdentifierInfo: represents both qualifier and ident name. what is this for?

9 Unsolved Questions

- What is a LazyTypeParameterDescriptor?
- What is a core.descriptors.MemberScope?
- What's the diff between TypeCheckerContext and TypeSystemContext?

Nomenclature

Commonization Seems to be a klib process to strip source and original information from descriptors. For example see CommonizedClassConD. Maybe this is to reduce storage size for these Ds in klibs?

Con Constructor. Can be either a type constructor like ∀ `a, List<a>`, or a value / class constructor like `<init>`

D DeclarationDescriptor or just Descriptor. See the DeclarationDescriptor section

Deser Deserialization. See core.deserialization – The compiler is able to serialize / deserialize descriptors. And it seems to be done lazily – Many deserialized descriptors still hold TypeTable / NameResolver / VersionRequirementTable to help with further deserialization.

FIR Intermediate level IR, below KtElement and above IR. See package compiler.fir.

Fun Function

Impl Implementation

IR Low level IR. See package compiler.ir.

J Java

PSI JetBrain's universal parse tree API. See its doc here.

Stub In the context of PSI, this is the interface part of a PSI tree. It's initially calculated from PSI trees and then cached for efficient retrival. See JetBrain's doc.

Subst Substitution, usually in the context of type checking / reconstruction / unification algorithms. See Wikipedia: Unification and nLab: Substitution.

Vis Visibility, as in public / private / internal etc.