Kotlin 元编程

从注解处理器 KAPT --> 符号处理器 KSP

霍丙乾 Benny Huo（bilibili：**bennyhuo 不是算命的**）

Kotlin GDE（Google 认证开发专家），《深入理解 Kotlin 协程》作者

Benny Huo Kotlin GDE（Google 认证开发专家）

2016.3	Bugly公众号	为什么说Kotlin值得一试
2017.11	Android 技术大会	将 Kotlin 投入 Android 生产环境中
2018.11	JetBrains 北京开发者大会	优雅地使用 Data Class
2019.12	慕课网	Kotlin 从入门到精通(基于 Kotlin 1.3)
2020.5	机械工业出版社	《深入理解 Kotlin 协程》
2020.5	GDG Android 11 Meetup	Kotlin 协程那些事儿
2020.10	全球移动开发者峰会	Kotlin多平台在移动端应用与展望
2020.11	GDG Kotlin Day	
2021.7	GDG 社区说	Kotlin 编译器插件：我们究竟在期待什么？
2021.11	2021 Kotlin 中文开发者大会	**从注解处理器 KAPT --> 符号处理器 KSP**

Kotlin 元编程

2

从注解处理器 KAPT --> 符号处理器 KSP

霍丙乾 Benny Huo（bilibili：**bennyhuo 不是算命的**）

Kotlin GDE（Google 认证开发专家），《深入理解 Kotlin 协程》作者

内容概要

- 认识 Kotlin 元编程
- Kotlin 注解处理器 (KAPT) 存在的问题
- Kotlin 符号处理器 (KSP) 有哪些优势
- 如何从 KAPT 迁移至 KSP
- 基于 KAPT 和 KSP 的进一步抽象
- 单元测试的编写

什么是元编程 Meta Programming

- 元编程：编写以程序作为数据的程序
  - 编译器、链接器、解释器、调试工具、程序分析工具等等
  - 编译时处理源码、中间代码以生成或修改源码、中间代码的程序
  - 运行时读取类、函数的数据以执行某种动态逻辑的程序
- 内省：运行时读取程序自身信息
- 反射：运行时读取程序自身信息并修改其结构和行为

什么时候需要元编程?

- 当我们写了很多**模板代码**的时候
- 当我们写了很多**重复代码**的时候
- 当我们想要**隐藏一些实现细节**的时候
- 当我们想要**创造语法糖**的时候
- ……

```kotlin
data class District(var name: String)

data class Location(var lat: Double, var lng: Double)

data class Company(
    var name: String,
    var location: Location,
    var district: District
)

data class Speaker(var name: String, var age: Int, var company: Company)

data class Talk(var name: String, var speaker: Speaker)
```

```kotlin
fun Talk.deepCopy(
    name: String = this.name,
    speaker: Speaker = this.speaker)
: Talk = Talk(name, speaker.deepCopy())
```

```kotlin
fun Location.deepCopy(
    lat: Double = this.lat,
    lng: Double = this.lng
): Location = Location(lat, lng)
```

```kotlin
fun Speaker.deepCopy(
    name: String = this.name,
    age: Int = this.age,
    company: Company = this.company
): Speaker = Speaker(name, age, company.deepCopy())
```

```kotlin
fun District.deepCopy(
    name: String = this.name
): District = District(name)
```

```kotlin
fun Company.deepCopy(
    name: String = this.name,
    location: Location = this.location,
    district: District = this.district
): Company = Company(name, location.deepCopy(), district.deepCopy())
```

Kotlin 元编程的常见实现手段

- Kotlin 反射/Java 反射
- Kotlin 注解处理器 (Kotlin Annotation Processor Tool，KAPT)*
- **Kotlin 符号处理器 (Kotlin Symbol Processing，KSP)**
- Kotlin 编译器插件 (Kotlin Compiler Plugin，KCP)

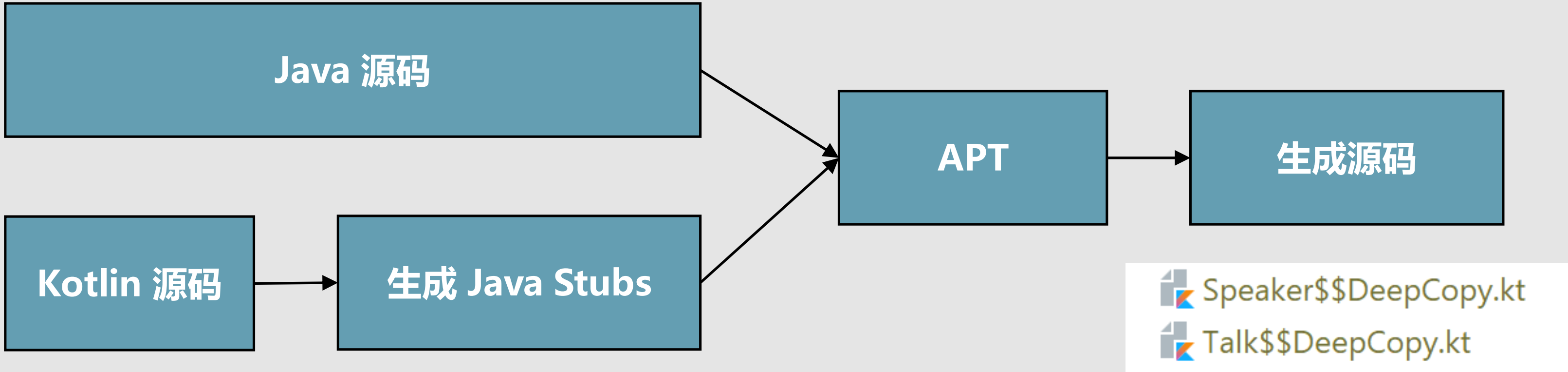
*__KAPT__ 是基于 **Java** 注解处理器实现的 **Kotlin** 编译器插件

Kotlin 注解处理器（KAPT）存在的问题

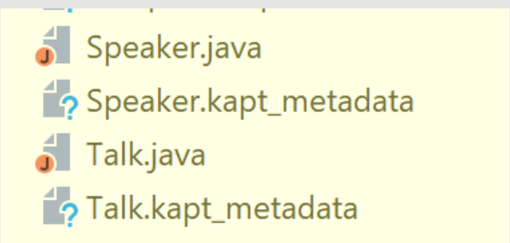
```kotlin
data class Company(
    var name: String,
    var location: Location,
    var district: District
)
```

```kotlin
fun Company.deepCopy(
    name: String = this.name,
    location: Location = this.location,
    district: District = this.district
): Company = Company(
    name, location.deepCopy(), district.deepCopy()
)
```

KAPT 的工作机制

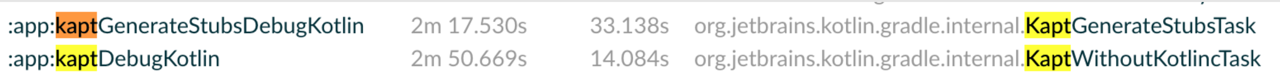


Speaker$$DeepCopy.kt
Talk$$DeepCopy.kt



Speaker.java
Speaker.kapt_metadata
Talk.java
Talk.kapt_metadata

build/tmp/kapt3/stubs



```
:app:kaptGenerateStubsDebugKotlin   2m 17.530s   33.138s   org.jetbrains.kotlin.gradle.internal.KaptGenerateStubsTask
:app:kaptDebugKotlin                2m 50.669s   14.084s   org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask
```

KAPT 是 Java 视角

- 如何判断类型是否为 data class?
- 如何获取 data class 对应的构造器以及其参数?

Kotlin 的类信息

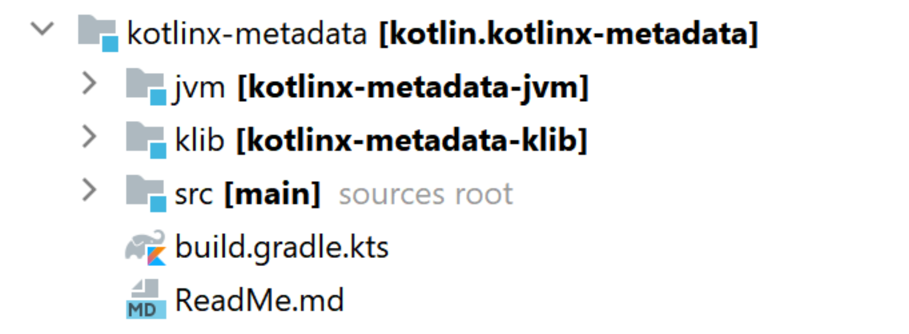
```
public annotation class Metadata(
    @get:JvmName("k")
    val kind: Int = 1,
    @get:JvmName("mv")
    val metadataVersion: IntArray = [],
    @get:JvmName("d1")
    val data1: Array<String> = [],
    @get:JvmName("d2")
    val data2: Array<String> = [],
    @get:JvmName("xs")
    val extraString: String = "",
    @get:JvmName("pn")
    val packageName: String = "",
    @get:JvmName("xi")
    val extraInt: Int = 0
)
```

Kotlin 的类信息

```
@Metadata(
 mv = {1, 4, 3},
 bv = {1, 0, 3},
 k = 1,
 d1 = {“\u0000(\n\u0002……"},
 d2 = {
  "Lcom/bennyhuo/kotlin/deepcopy/sample/Talk;",
  "",
  "name",
  "...",
 }
)
```

```
message Class {
  enum Kind {
    // 3 bits
    CLASS = 0;
    INTERFACE = 1;
    ENUM_CLASS = 2;
    ENUM_ENTRY = 3;
    ANNOTATION_CLASS = 4;
    OBJECT = 5;
    COMPANION_OBJECT = 6;
  }

  /*
    hasAnnotations
    Visibility
    Modality
    ClassKind
    isInner
    isData
    isExternal
    isExpect
    isInline
    isFun
  */
  optional int32 flags = 1 [default = 6 /* public final class, no annotations */];

  required int32 fq_name = 3 [(fq_name_id_in_table) = true];
  repeated int32 nested_class_name = 7 [packed = true, (name_id_in_table) = true];

  repeated Constructor constructor = 8;
  repeated Function function = 9;
  repeated Property property = 10;
  repeated TypeAlias type_alias = 11;

  repeated EnumEntry enum_entry = 13;

  repeated int32 sealed_subclass_fq_name = 16 [packed = true, (fq_name_id_in_table) = true];

  optional int32 inline_class_underlying_property_name = 17 [(name_id_in_table) = true];

  optional Type inline_class_underlying_type = 18;
  optional int32 inline_class_underlying_type_id = 19 [(type_id_in_table) = true];

  optional TypeTable type_table = 30;

  // Index into the VersionRequirementTable
  repeated int32 version_requirement = 31;

  optional VersionRequirementTable version_requirement_table = 32;

  extensions 100 to 18999;
}
```

Kotlin 官方用于解析 Metadata 的库



kotlinx-metadata **[kotlin.kotlinx-metadata]**

- jvm **[kotlinx-metadata-jvm]**
- klib **[kotlinx-metadata-klib]**
- src **[main]** sources root
- build.gradle.kts
- ReadMe.md

```
api("org.jetbrains.kotlinx:kotlinx-metadata-jvm:0.3.0")
```

```kotlin
open class KmTypeVisitorImpl(...) :KmTypeVisitor() {

    private var name: ClassName = ""

    private var isReified = true

    val rawType: TypeName by lazy {
        ...
    }

    val type: TypeName by lazy {
        ...
    }

    val wildcardTypeName by lazy {
        ...
    }

    override fun visitAbbreviatedType(flags: Flags): KmTypeVisitor? {
        return KmTypeVisitorImpl(flags, typeParametersInContainer, parent = this).also {
            abbreviatedTypeVisitor = it
        }
    }

    override fun visitArgument(flags: Flags, variance: KmVariance): KmTypeVisitor? {
        return ...
    }
    override fun visitArgument(flags: Flags, variance: KmVariance): KmTypeVisitor? {
        return ...
    }

    override fun visitClass(name: ClassName) {
        super.visitClass(name)
        this.name = name
    }

    override fun visitStarProjection() {
        super.visitStarProjection()
        typeParameters += KmTypeVisitorImpl(0, typeParametersInContainer, parent = this).also {
            it.visitClass("*")
            it.isReified = false
        }
    }

    override fun visitTypeAlias(name: ClassName) {
        super.visitTypeAlias(name)
        this.name = name
    }

    override fun visitTypeParameter(id: Int) {
        super.visitTypeParameter(id)
        this.name = typeParametersInContainer[id].name
        this.isReified = false
    }
}
```

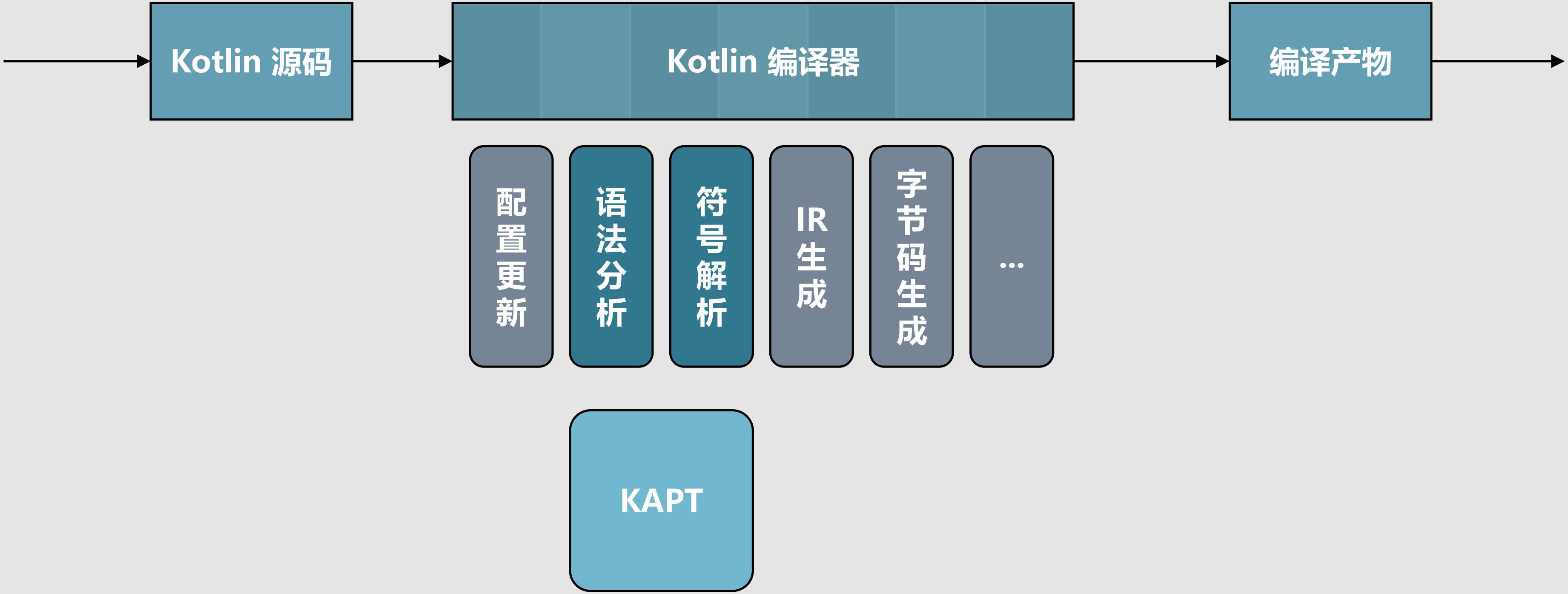
获取 data class 的信息

```kotlin
class KClassMirror(kotlinClassMetadata: KotlinClassMetadata.Class) {

    data class Component(val name: String, val type: TypeName) {
        val typeElement: KTypeElement? by lazy {
            KTypeElement.from(type)
        }
    }

    var isData: Boolean = false
        private set

    val components = mutableListOf<Component>()

    val typeParameters = mutableListOf<KmTypeParameterVisitorImpl>()

    ...
}
```

KAPT 处理 Kotlin 源码存在的问题

- 实现复杂，需要手动解析 Kotlin 类信息
- 编译耗时，KAPT 需将 Kotlin 类转成 Java Stubs
- 只支持 Kotlin-JVM

KAPT 的本质

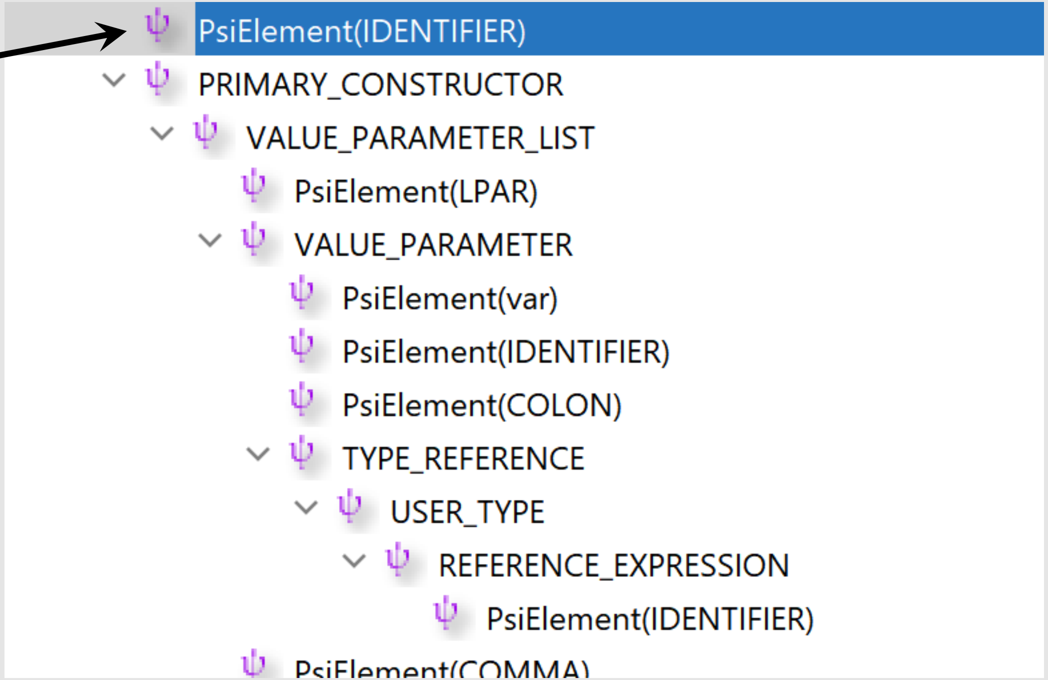


为什么要迁移至 Kotlin 符号处理器(KSP)

KSP 是什么

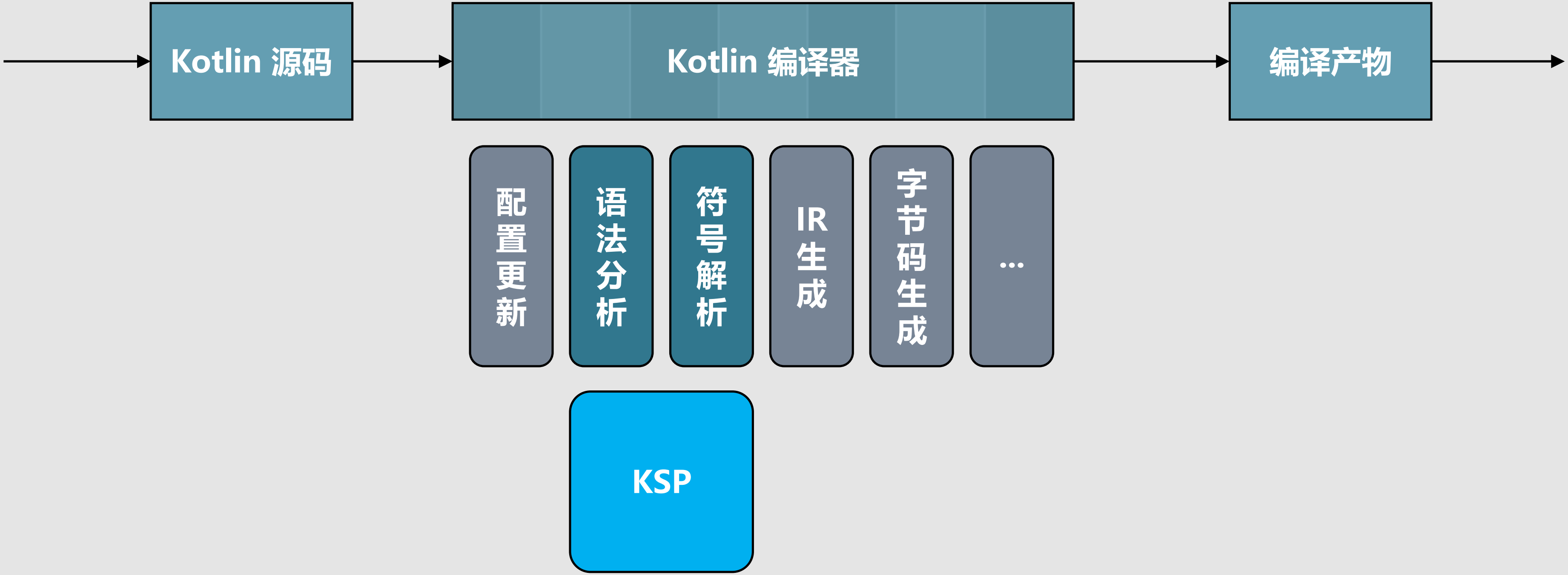
- Kotlin Symbol Processing API **by Google**

```kotlin
@DeepCopy
data class Company(
    var name: String,
    var location: Location,
    var district: District
)
```

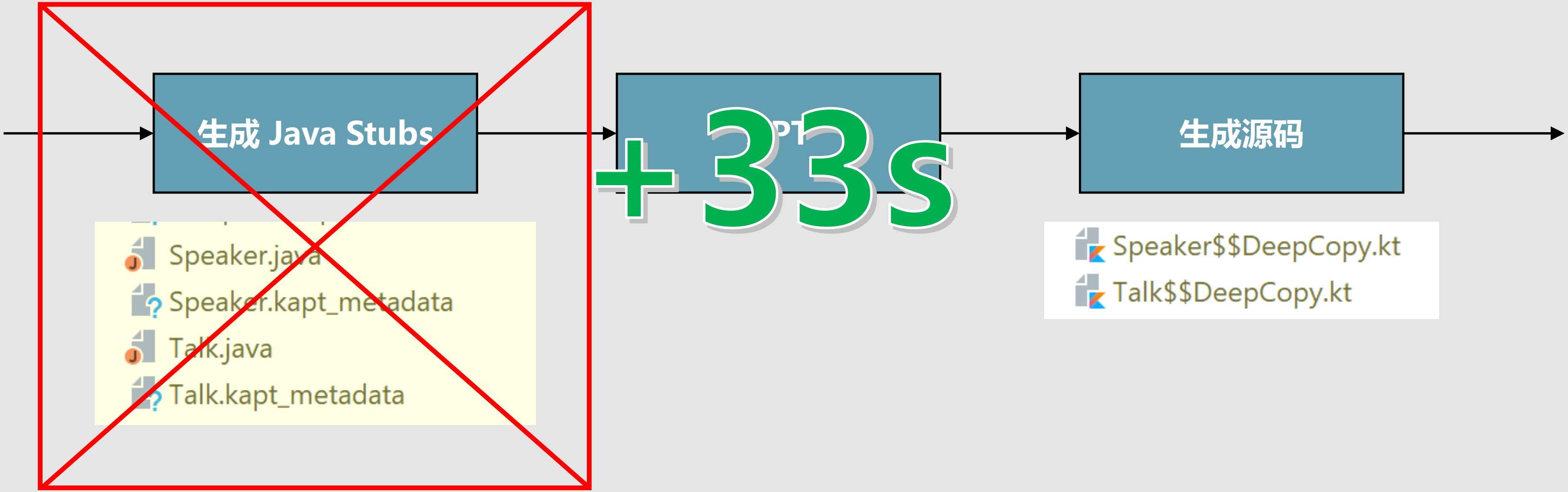


```
PsiElement(IDENTIFIER)
PRIMARY_CONSTRUCTOR
  VALUE_PARAMETER_LIST
    PsiElement(LPAR)
    VALUE_PARAMETER
      PsiElement(var)
      PsiElement(IDENTIFIER)
      PsiElement(COLON)
      TYPE_REFERENCE
        USER_TYPE
          REFERENCE_EXPRESSION
            PsiElement(IDENTIFIER)
    PsiElement(COMMA)
```

KSP 也是 Kotlin 编译器插件



KSP 优势(1)：省去生成 Java Stubs 的耗时



~~Speaker.java~~

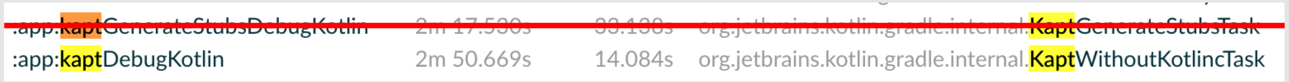
~~Speaker.kapt_metadata~~

~~Talk.java~~

~~Talk.kapt_metadata~~

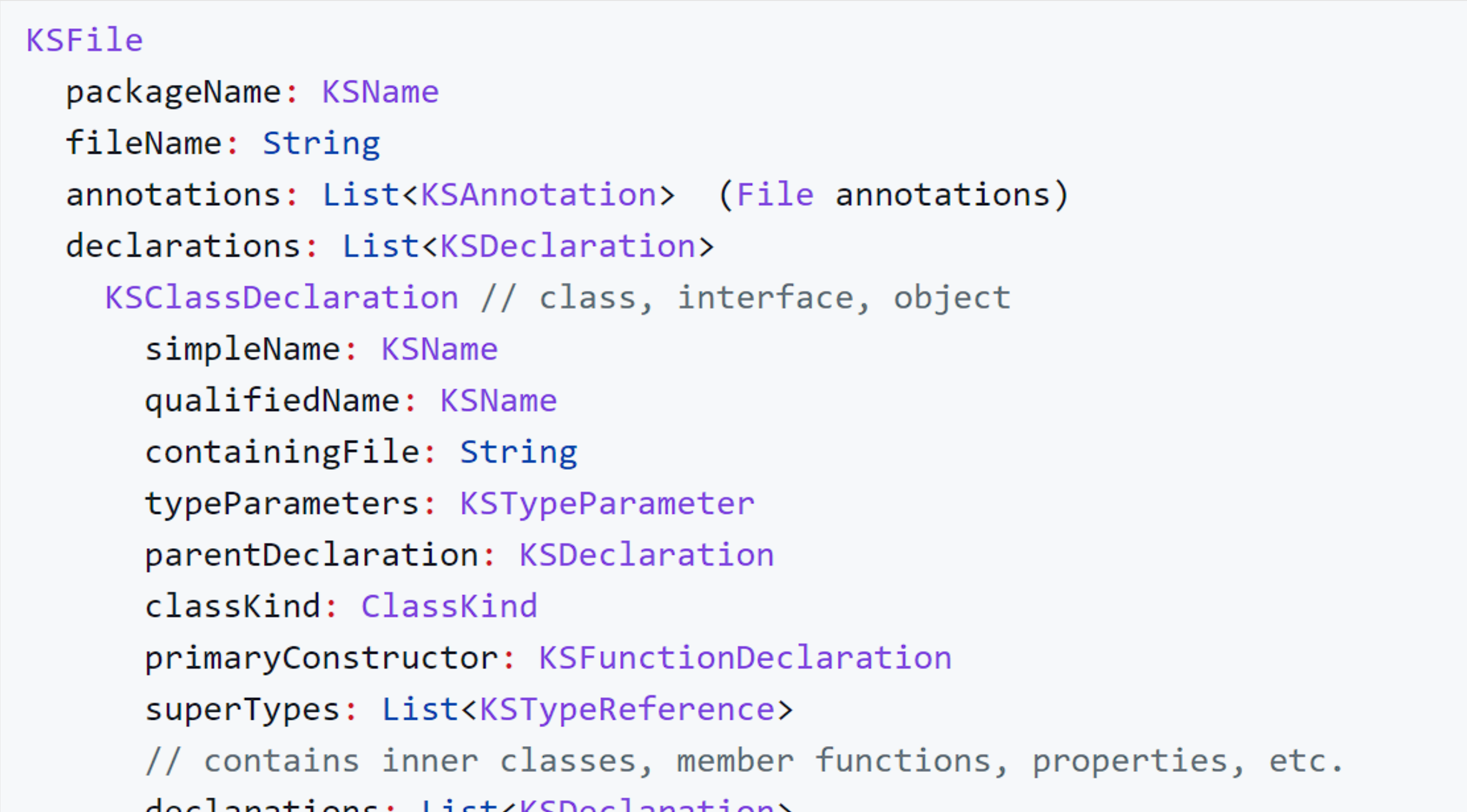
Speaker$$DeepCopy.kt

Talk$$DeepCopy.kt



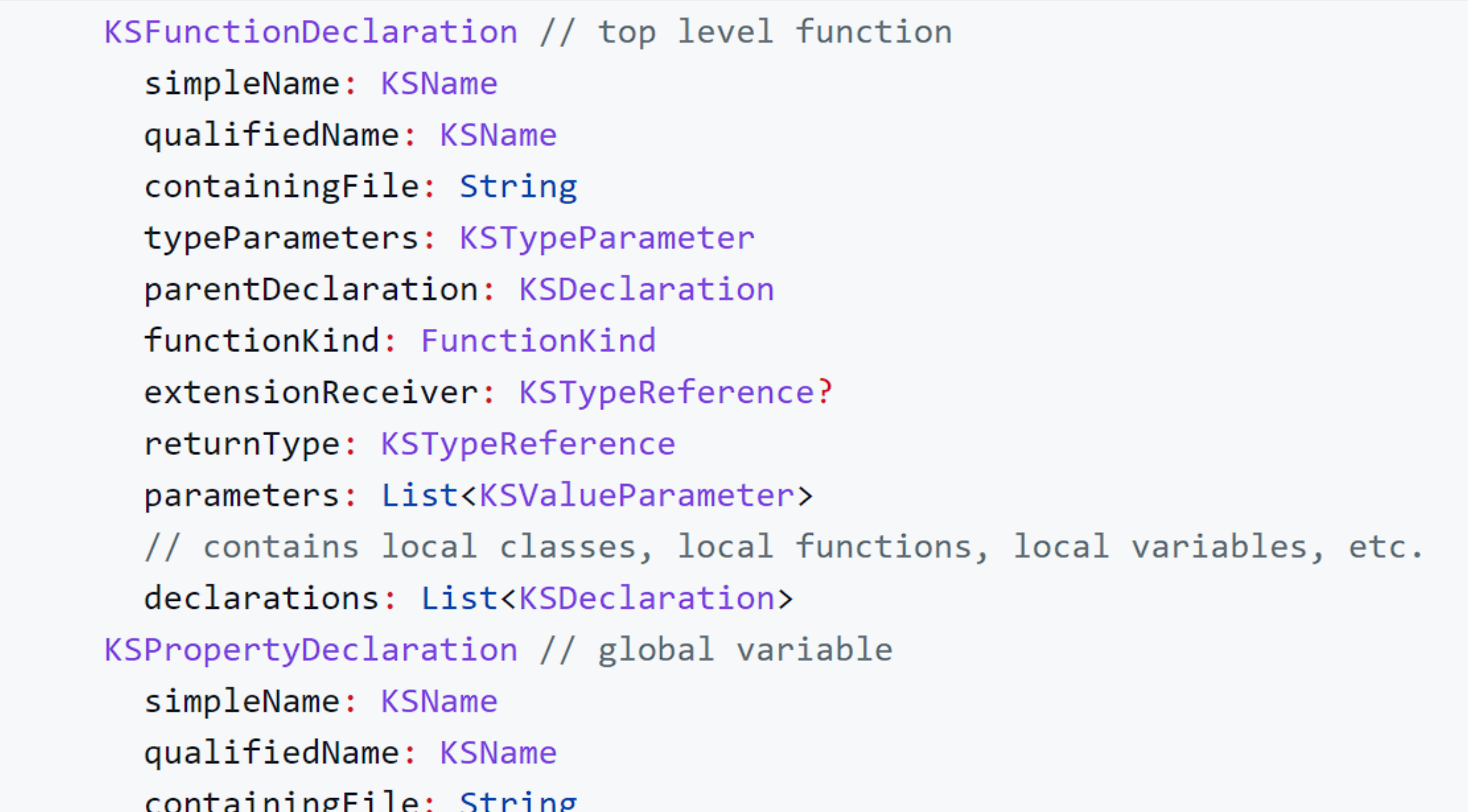
~~:app:kaptGenerateStubsDebugKotlin~~	~~2m 17.530s~~	~~33.138s~~	~~org.jetbrains.kotlin.gradle.internal.KaptGenerateStubsTask~~
:app:kaptDebugKotlin	2m 50.669s	14.084s	org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask

KSP 的主要类型



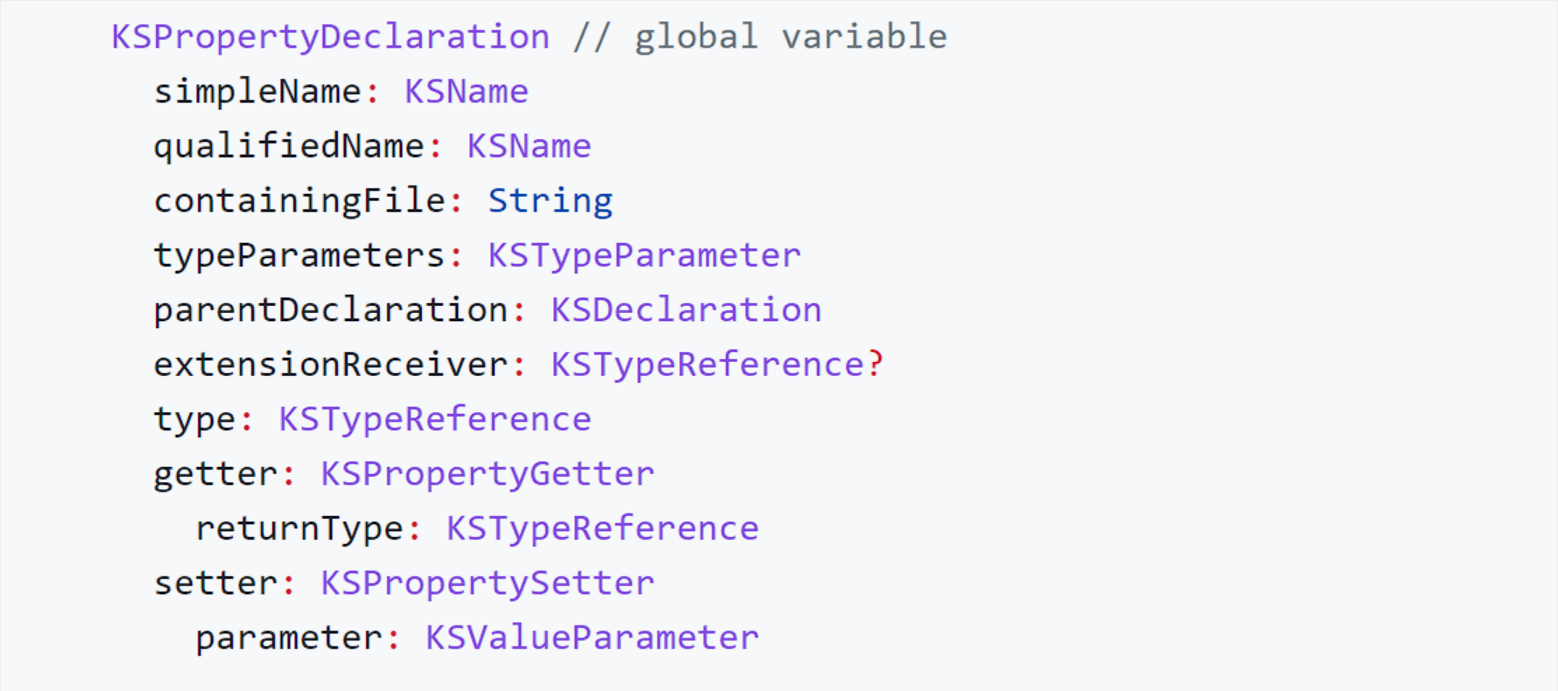
```
KSFile
  packageName: KSName
  fileName: String
  annotations: List<KSAnnotation>  (File annotations)
  declarations: List<KSDeclaration>
    KSClassDeclaration // class, interface, object
      simpleName: KSName
      qualifiedName: KSName
      containingFile: String
      typeParameters: KSTypeParameter
      parentDeclaration: KSDeclaration
      classKind: ClassKind
      primaryConstructor: KSFunctionDeclaration
      superTypes: List<KSTypeReference>
      // contains inner classes, member functions, properties, etc.
      declarations: List<KSDeclaration>
```

KSP 的主要类型



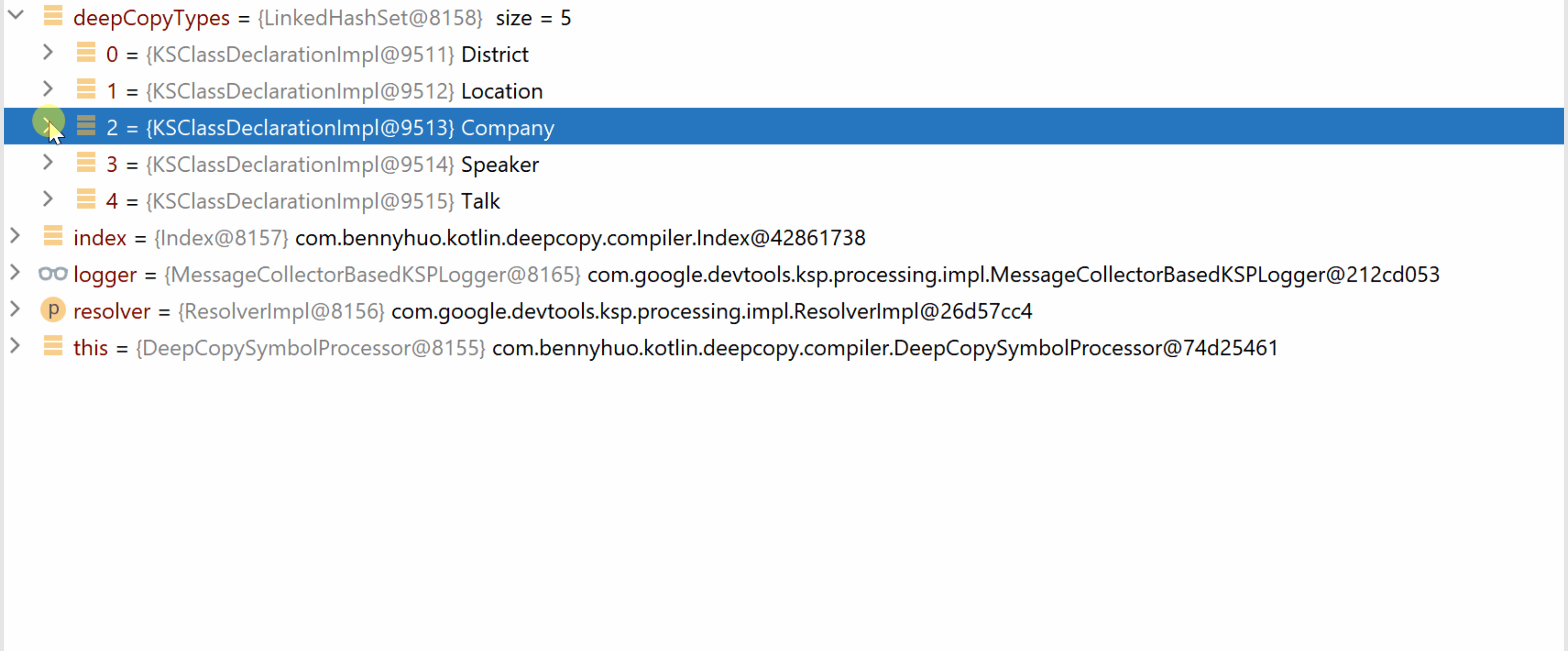
```
KSFunctionDeclaration // top level function
  simpleName: KSName
  qualifiedName: KSName
  containingFile: String
  typeParameters: KSTypeParameter
  parentDeclaration: KSDeclaration
  functionKind: FunctionKind
  extensionReceiver: KSTypeReference?
  returnType: KSTypeReference
  parameters: List<KSValueParameter>
  // contains local classes, local functions, local variables, etc.
  declarations: List<KSDeclaration>
KSPropertyDeclaration // global variable
  simpleName: KSName
  qualifiedName: KSName
  containingFile: String
```

KSP 的主要类型



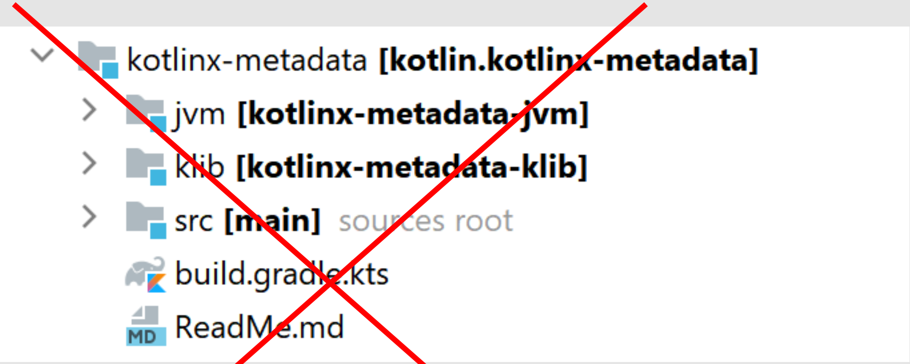
```
KSPropertyDeclaration // global variable
  simpleName: KSName
  qualifiedName: KSName
  containingFile: String
  typeParameters: KSTypeParameter
  parentDeclaration: KSDeclaration
  extensionReceiver: KSTypeReference?
  type: KSTypeReference
  getter: KSPropertyGetter
    returnType: KSTypeReference
  setter: KSPropertySetter
    parameter: KSValueParameter
```

示例：



```
deepCopyTypes = {LinkedHashSet@8158}  size = 5
  0 = {KSClassDeclarationImpl@9511} District
  1 = {KSClassDeclarationImpl@9512} Location
  2 = {KSClassDeclarationImpl@9513} Company
  3 = {KSClassDeclarationImpl@9514} Speaker
  4 = {KSClassDeclarationImpl@9515} Talk
index = {Index@8157} com.bennyhuo.kotlin.deepcopy.compiler.Index@42861738
logger = {MessageCollectorBasedKSPLogger@8165} com.google.devtools.ksp.processing.impl.MessageCollectorBasedKSPLogger@212cd053
resolver = {ResolverImpl@8156} com.google.devtools.ksp.processing.impl.ResolverImpl@26d57cc4
this = {DeepCopySymbolProcessor@8155} com.bennyhuo.kotlin.deepcopy.compiler.DeepCopySymbolProcessor@74d25461
```

KSP 优势(2)：直接提供 Kotlin 的符号信息



```
kotlinx-metadata [kotlin.kotlinx-metadata]
  jvm [kotlinx-metadata-jvm]
  klib [kotlinx-metadata-klib]
  src [main]  sources root
  build.gradle.kts
  ReadMe.md
```

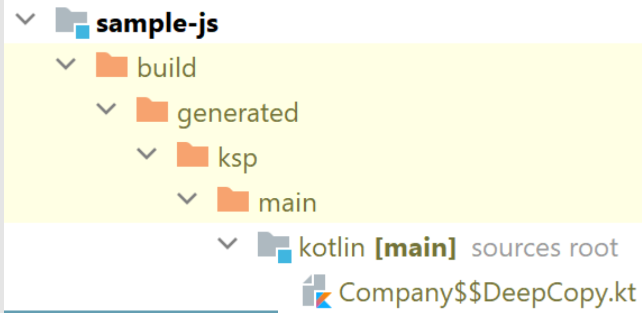
```
api("org.jetbrains.kotlinx:kotlinx-metadata-jvm:0.3.0")
```

```kotlin
data class Company(
    var name: String,
    var location: Location,
    var district: District
)
```

```kotlin
fun Company.deepCopy(
    name: String = this.name,
    location: Location = this.location,
    district: District = this.district
): Company = Company(
    name, location.deepCopy(), district.deepCopy()
)
```

平台无关

KSP 优势(3)：支持 Kotlin 多平台



```
sample-js
  build
    generated
      ksp
        main
          kotlin [main]  sources root
            Company$$DeepCopy.kt
            District$$DeepCopy.kt
            Location$$DeepCopy.kt
```

Company$$DeepCopy.kt

```kotlin
public fun Company.deepCopy(
  name: String = this.name,
  location: Location = this.location,
  district: District = this.district
): Company = Company(name,
location.deepCopy(), district.deepCopy())
```

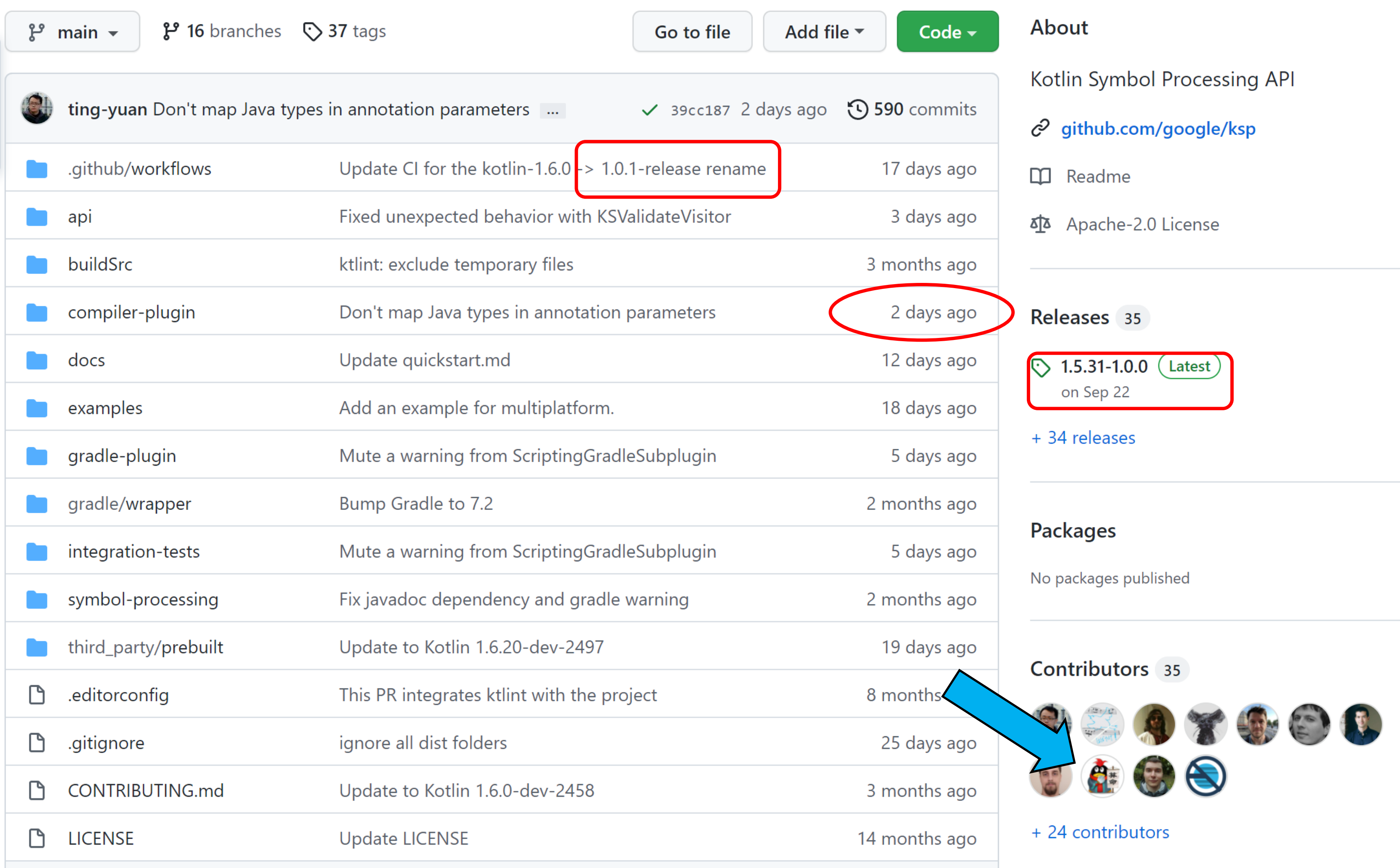
sample.js

```javascript
function deepCopy($receiver, name, location, district) {
  if (name === void 0)
    name = $receiver.name;
  if (location === void 0)
    location = $receiver.location;
  if (district === void 0)
    district = $receiver.district;
  return new Company(name, deepCopy_1(location), deepCopy_0(district));
}
```

Java Annotation 简史

- JSR 175: A Metadata Facility for the Java. (Java 5, Annotations)
- **JSR 269: Support for pluggable annotations. (Java 6, APT)**
- JSR 308, JEP 104: Annotation on Java types. (Java 8)
- JSR 337, JEP 120: Repeating annotations. (Java 8)

—— APT 集成在 Java 编译器当中发布，鲜有更新



ting-yuan Don't map Java types in annotation parameters … ✓ 39cc187 2 days ago 590 commits

.github/workflows	Update CI for the kotlin-1.6.0 -> 1.0.1-release rename	17 days ago
api	Fixed unexpected behavior with KSValidateVisitor	3 days ago
buildSrc	ktlint: exclude temporary files	3 months ago
compiler-plugin	Don't map Java types in annotation parameters	2 days ago
docs	Update quickstart.md	12 days ago
examples	Add an example for multiplatform.	18 days ago
gradle-plugin	Mute a warning from ScriptingGradleSubplugin	5 days ago
gradle/wrapper	Bump Gradle to 7.2	2 months ago
integration-tests	Mute a warning from ScriptingGradleSubplugin	5 days ago
symbol-processing	Fix javadoc dependency and gradle warning	2 months ago
third_party/prebuilt	Update to Kotlin 1.6.20-dev-2497	19 days ago
.editorconfig	This PR integrates ktlint with the project	8 months
.gitignore	ignore all dist folders	25 days ago
CONTRIBUTING.md	Update to Kotlin 1.6.0-dev-2458	3 months ago
LICENSE	Update LICENSE	14 months ago

About

Kotlin Symbol Processing API

github.com/google/ksp

Readme

Apache-2.0 License

Releases 35

1.5.31-1.0.0 Latest
on Sep 22

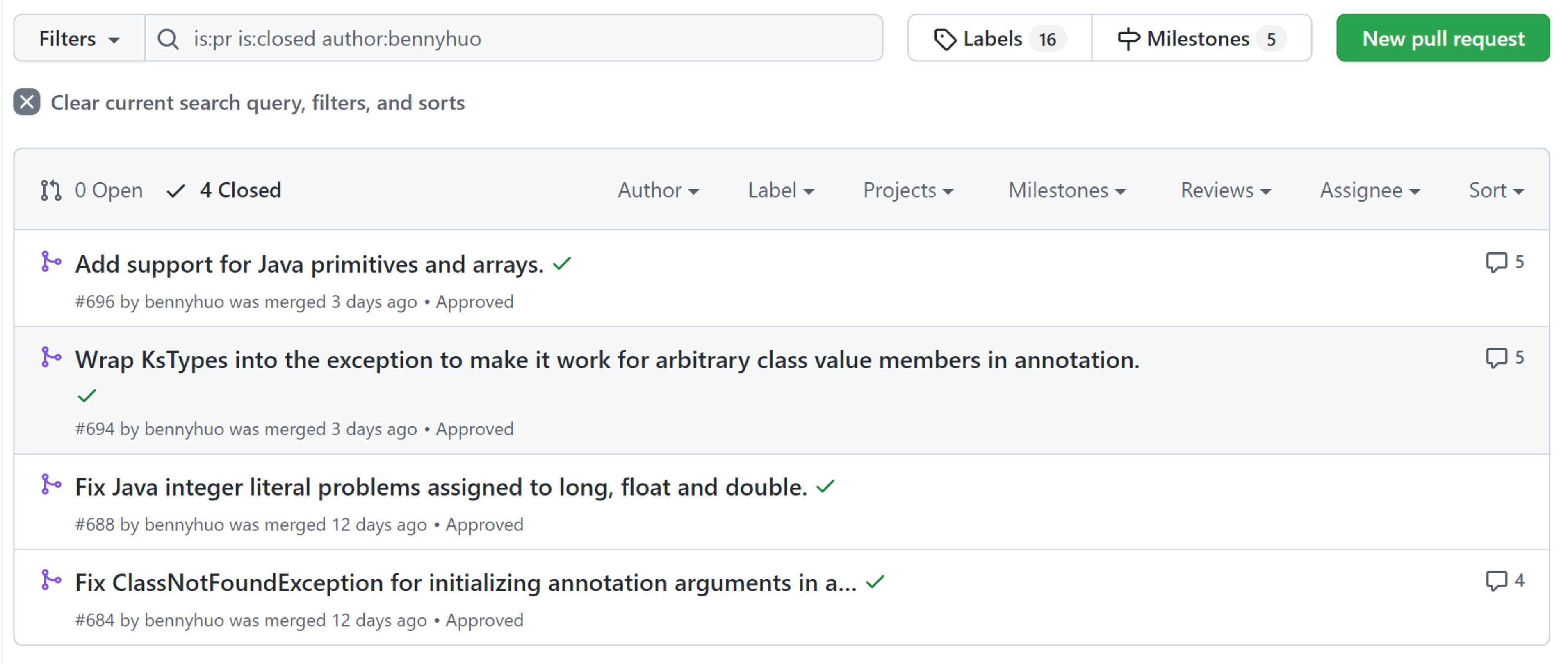
+ 34 releases

Packages

No packages published

Contributors 35

+ 24 contributors



is:pr is:closed author:bennyhuo

Clear current search query, filters, and sorts

0 Open ✓ 4 Closed

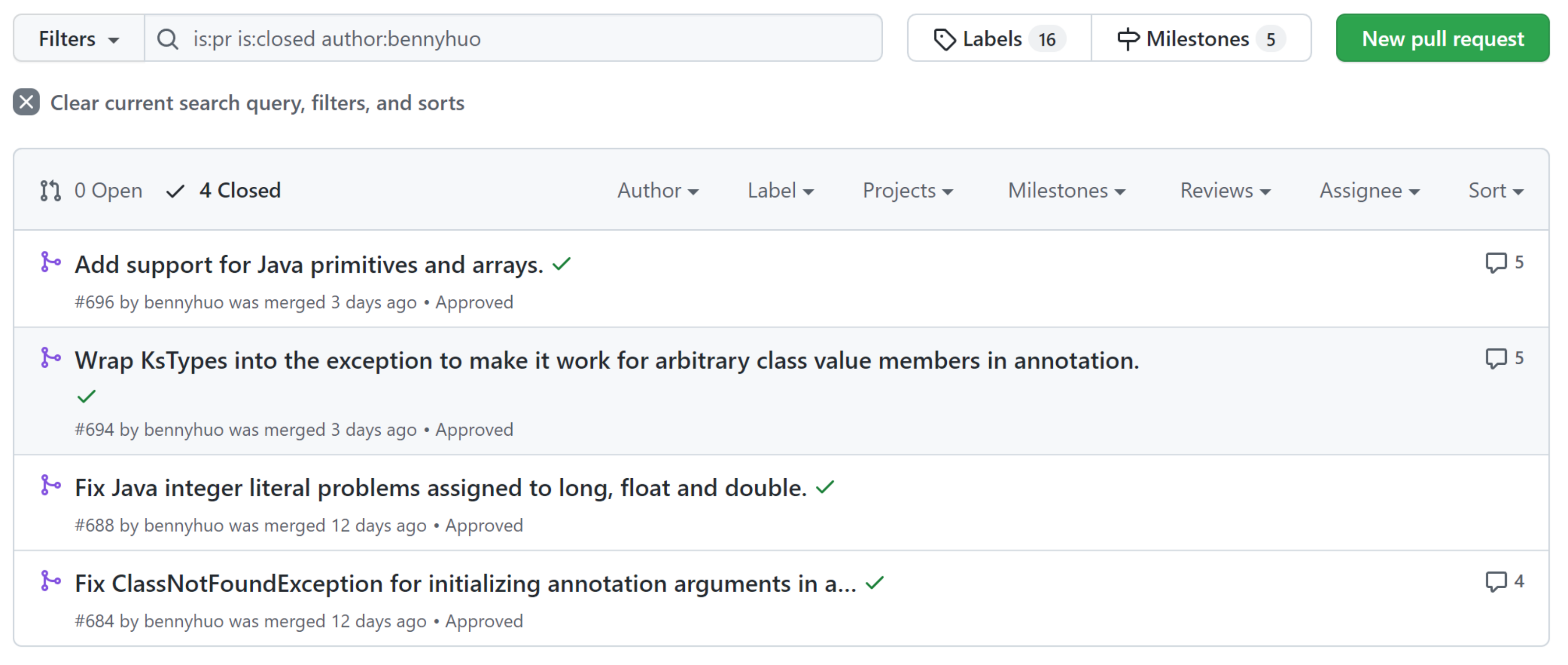
Add support for Java primitives and arrays. ✓ 5
#696 by bennyhuo was merged 3 days ago • Approved

Wrap KsTypes into the exception to make it work for arbitrary class value members in annotation. ✓ 5
#694 by bennyhuo was merged 3 days ago • Approved

Fix Java integer literal problems assigned to long, float and double. ✓
#688 by bennyhuo was merged 12 days ago • Approved

Fix ClassNotFoundException for initializing annotation arguments in a... ✓ 4
#684 by bennyhuo was merged 12 days ago • Approved

KSP 优势(4)：社区活跃，未来可期



Add support for Java primitives and arrays. ✓ 5
#696 by bennyhuo was merged 3 days ago • Approved

Wrap KsTypes into the exception to make it work for arbitrary class value members in annotation. ✓ 5
#694 by bennyhuo was merged 3 days ago • Approved

Fix Java integer literal problems assigned to long, float and double. ✓
#688 by bennyhuo was merged 12 days ago • Approved

Fix ClassNotFoundException for initializing annotation arguments in a… ✓ 4
#684 by bennyhuo was merged 12 days ago • Approved

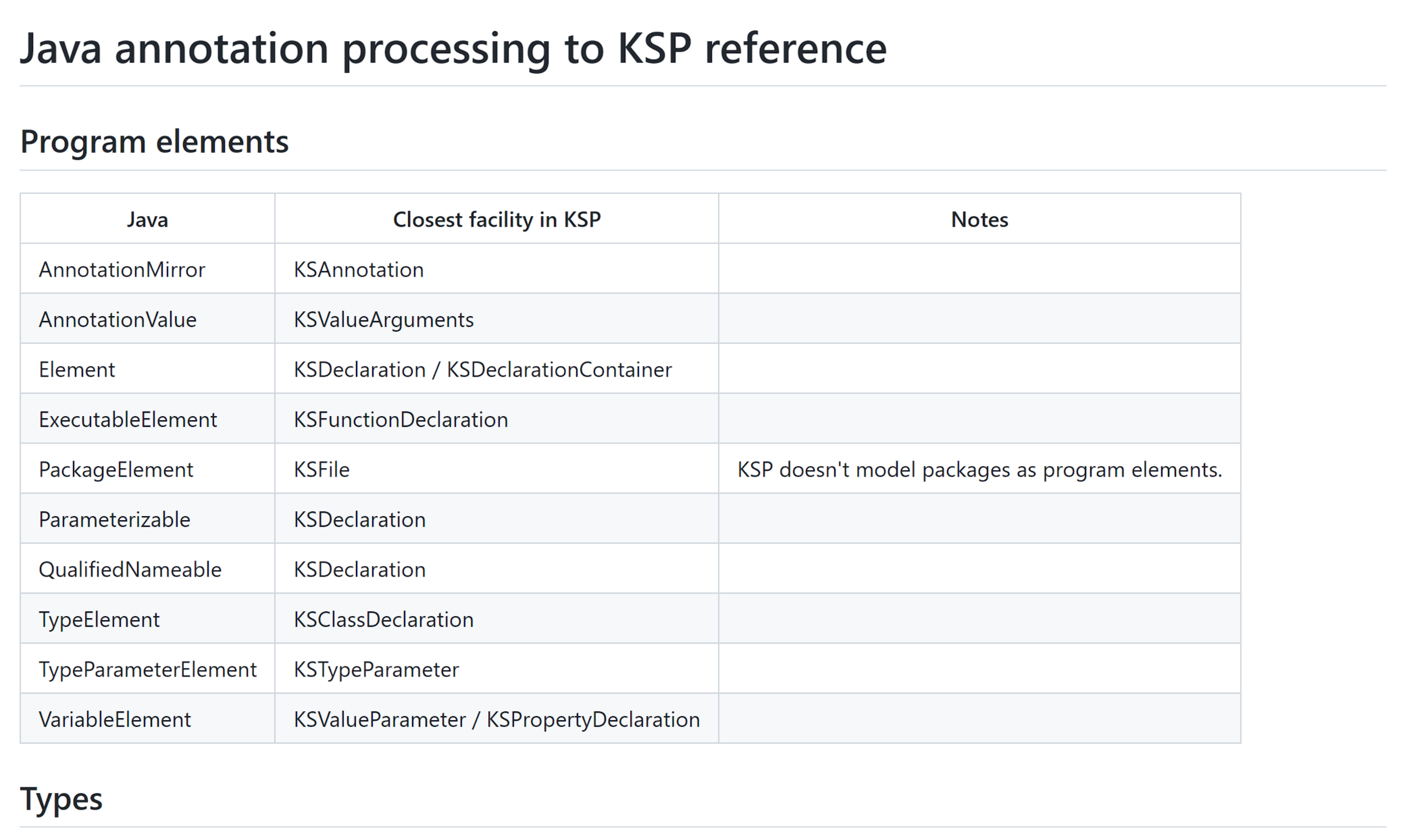
Kotlin 元编程的几种方案对比

	Reflection	KAPT	KSP	KCP
运行时	慢	无	**无**	无
编译时	无	需解析 metadata	**基于 Kotlin AST**	基于 Kotlin AST
复杂度	较低	中	**中**	较高
主要场景	提供动态能力	生成源码	**生成源码**	生成、修改 IR
现状	稳定	稳定	**1.0**	实验
多平台	JVM + JS	只 JVM	**全部**	全部

Kotlin 元编程的几种方案对比

	Reflection	KAPT	KSP	KCP
运行时	慢	无	**无**	无
编译时	无	需解析 metadata	**基于 Kotlin AST**	基于 Kotlin AST
复杂度	较低	中	**中**	较高
主要场景	提供动态能力	生成源码	**生成源码**	生成、修改 IR
现状	稳定	稳定	**1.0**	实验
多平台	JVM + JS	只 JVM	**全部**	全部

如何迁移至 Kotlin 符号处理器(KSP)



Java annotation processing to KSP reference

Program elements

Java	Closest facility in KSP	Notes
AnnotationMirror	KSAnnotation	
AnnotationValue	KSValueArguments	
Element	KSDeclaration / KSDeclarationContainer	
ExecutableElement	KSFunctionDeclaration	
PackageElement	KSFile	KSP doesn't model packages as program elements.
Parameterizable	KSDeclaration	
QualifiedNameable	KSDeclaration	
TypeElement	KSClassDeclaration	
TypeParameterElement	KSTypeParameter	
VariableElement	KSValueParameter / KSPropertyDeclaration	

Types

Types

Because KSP requires explicit type resolution, some functionalities in Java can only be carried out by KSType and the corresponding elements before resolution.

Java	Closest facility in KSP	Notes
ArrayType	KSBuiltIns.arrayType	
DeclaredType	KSType / KSClassifierReference	
ErrorType	KSType.isError	
ExecutableType	KSType / KSCallableReference	
IntersectionType	KSType / KSTypeParameter	
NoType	KSType.isError	N/A in KSP
NullType		N/A in KSP
PrimitiveType	KSBuiltIns	Not exactly same as primitive type in Java
ReferenceType	KSTypeReference	
TypeMirror	KSType	
TypeVariable	KSTypeParameter	

Misc

Java	Closest facility in KSP	notes
Name	KSName	
ElementKind	ClassKind / FunctionKind	
Modifier	Modifier	
NestingKind	ClassKind / FunctionKind	
AnnotationValueVisitor		
ElementVisitor	KSVisitor	
AnnotatedConstruct	KSAnnotated	
TypeVisitor		
TypeKind	KSBuiltIns	Some can be found in builtins, otherwise check KSClassDeclaration for DeclaredType
ElementFilter	Collection.filterIsInstance	
ElementKindVisitor	KSVisitor	
ElementScanner	KSTopDownVisitor	
SimpleAnnotationValueVisitor		No needed in KSP
SimpleElementVisitor	KSVisitor	

处理器上下文

KAPT

```
ProcessingEnvironment
RoundEnvironment
```

KSP

```
SymbolProcessorEnvironment
Resolver
```

读取被标注的类型

KAPT

```kotlin
env.getElementsAnnotatedWith(<AnnotationType>)
  .filterIsInstance<TypeElement>()
  .forEach { element ->
    val type = element.asType()
    ...
  }
```

KSP

```kotlin
resolver.getSymbolsWithAnnotation(<AnnotationClassName>)
  .filterIsInstance<KSClassDeclaration>()
  .forEach { declaration ->
    val type = declaration.asStarProjectedType()
  }
```

通过类名获取类定义

KAPT

```
val types: Types = ...
val elements: Elements = ...

val element = elements.getTypeElement("...")
```

KSP

```
val resolver: Resolver = ...

val declaration = resolver.getClassDeclarationByName("...")
```

判断类型继承关系

KAPT

```
val types: Types = ...
val elements: Elements = ...

fun TypeMirror.erasure() = types.erasure(this)

fun TypeMirror.isSubTypeOf(className: String): Boolean {
  val type = elements.getTypeElement(className)
    ?.asType() ?: return false
  return types.isSubtype(this.erasure(), type.erasure())
}
```

KSP

```
val resolver: Resolver = ...

fun KSType.isSubTypeOf(typeName: String): Boolean {
  return resolver.getClassDeclarationByName(typeName)
    ?.asStarProjectedType()
    ?.isAssignableFrom(this) == true
}
```

获取注解实例

```
annotation class DeepCopyConfig(val values: Array<KClass<*>> = [])
```

KAPT

```
val config = element.getAnnotation(DeepCopyConfig::class.java)
val classes = config.values
```

KSP

```
val config = declaration
  .getAnnotationsByType(DeepCopyConfig::class)
  .first()
val classes = config.values
```

KotlinPoet 的扩展支持

KAPT

```
public fun TypeMirror.asTypeName(): TypeName
  = TypeName.get(this, mutableMapOf())
```

KSP

```
@KotlinPoetKspPreview
public fun KSType.toTypeName(
  typeParamResolver: TypeParameterResolver = ...
): TypeName {
  ...
}

implementation("com.squareup:kotlinpoet-ksp:1.10.0")
```

生成文件

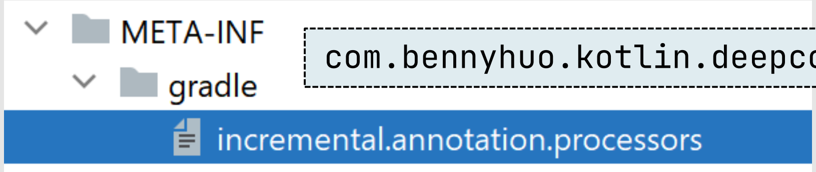
KAPT

```
filer.createResource(
  StandardLocation.SOURCE_OUTPUT,
  packageName, name + ".kt"
).openWriter().use {
  ...
}
```

KSP

```
codeGenerator.createNewFile(dependencies, packageName, name)
  .writer().use {
    ...
  }
```

KAPT 增量编译



```
META-INF
  gradle
    incremental.annotation.processors
```

```
com.bennyhuo.kotlin.deepcopy.compiler.DeepCopyProcessor,aggregating
```

Filer

```
FileObject createResource(JavaFileManager.Location location,
                          CharSequence moduleAndPkg,
                          CharSequence relativeName,
                          Element... originatingElements);
```

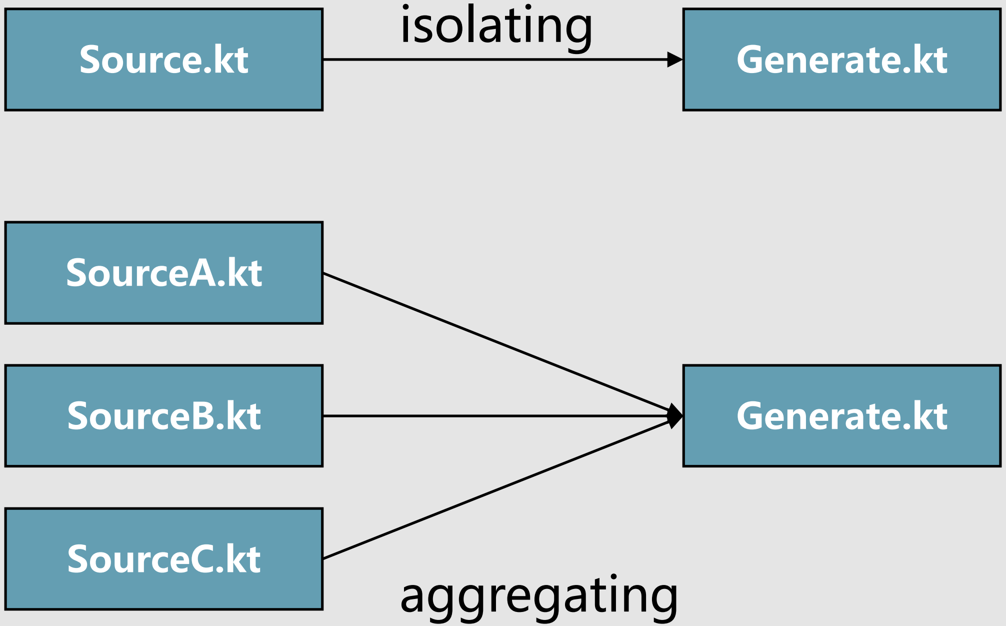
```
val functionBuilder = FunSpec.builder("deepCopy")
  .addOriginatingElement(typeElement)

fileSpecBuilder.addFunction(functionBuilder.build()).build()
  .writeTo(filer)
```

KSP 增量编译

```
class Dependencies private constructor(
    val isAllSources: Boolean,
    val aggregating: Boolean,
    val originatingFiles: List<KSFile>
) { ... }
```

```
functionBuilder.addOriginatingKSFile(it)

fileSpecBuilder.addFunction(functionBuilder.build()).build()
    .writeTo(environment.codeGenerator, aggregating = false)
```

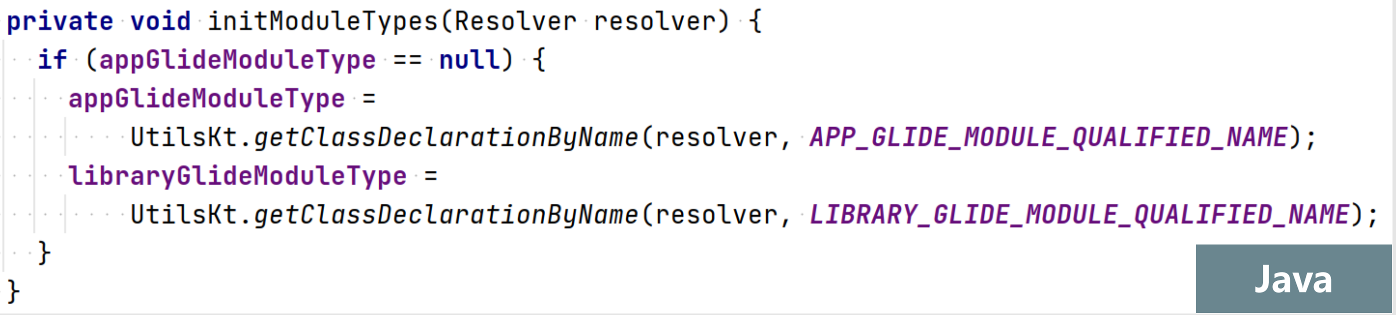
isolating vs aggregating



迁移 KSP 的几点注意事项

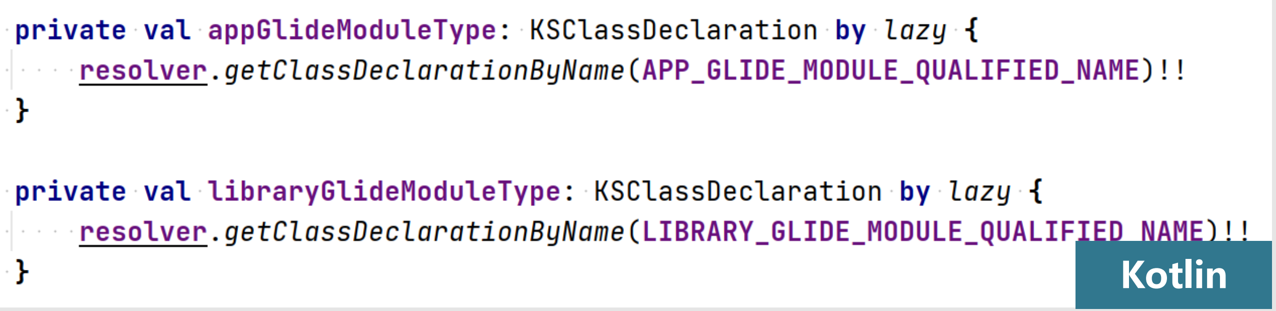
KSP 程序源码尽量迁移至 Kotlin

- KSP 的 API 对于 Java 不友好，最好使用 Kotlin 编写



```java
private void initModuleTypes(Resolver resolver) {
  if (appGlideModuleType == null) {
    appGlideModuleType =
        UtilsKt.getClassDeclarationByName(resolver, APP_GLIDE_MODULE_QUALIFIED_NAME);
    libraryGlideModuleType =
        UtilsKt.getClassDeclarationByName(resolver, LIBRARY_GLIDE_MODULE_QUALIFIED_NAME);
  }
}
```

Java



```kotlin
private val appGlideModuleType: KSClassDeclaration by lazy {
    resolver.getClassDeclarationByName(APP_GLIDE_MODULE_QUALIFIED_NAME)!!
}

private val libraryGlideModuleType: KSClassDeclaration by lazy {
    resolver.getClassDeclarationByName(LIBRARY_GLIDE_MODULE_QUALIFIED_NAME)!!
}
```

Kotlin

尽量生成 Kotlin 源码

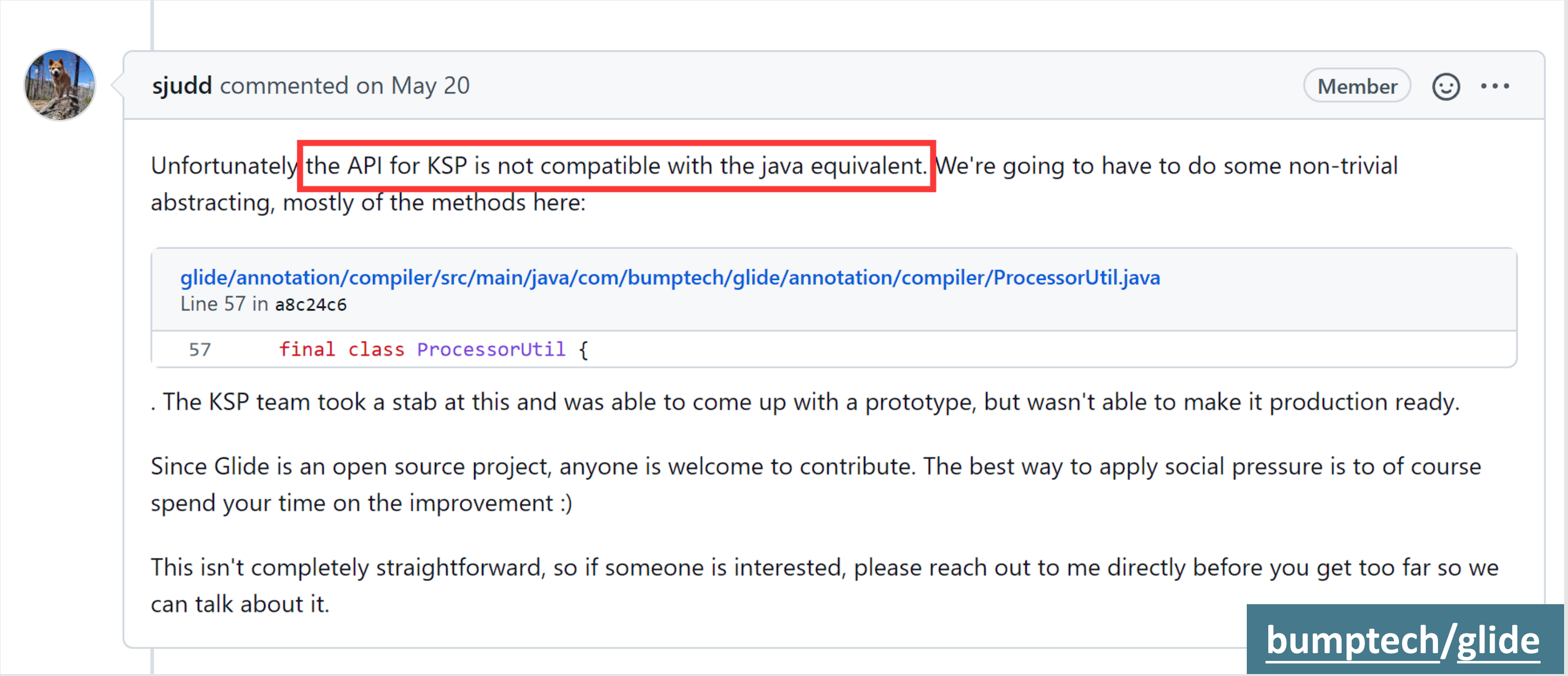
- JavaPoet 没有提供对 KSP 的支持
- KSP 不太容易区分 Java 基本类型（例如：int.class/Integer.class）

请问是什么支撑你
仍然坚持写 Java 的?

你以为
shi山
想改就能改吗

基于 KAPT 和 KSP 实现进一步抽象

Feature request: Support KSP #4492



sjudd commented on May 20

Unfortunately the API for KSP is not compatible with the java equivalent. We're going to have to do some non-trivial abstracting, mostly of the methods here:

glide/annotation/compiler/src/main/java/com/bumptech/glide/annotation/compiler/ProcessorUtil.java
Line 57 in a8c24c6

```
57        final class ProcessorUtil {
```

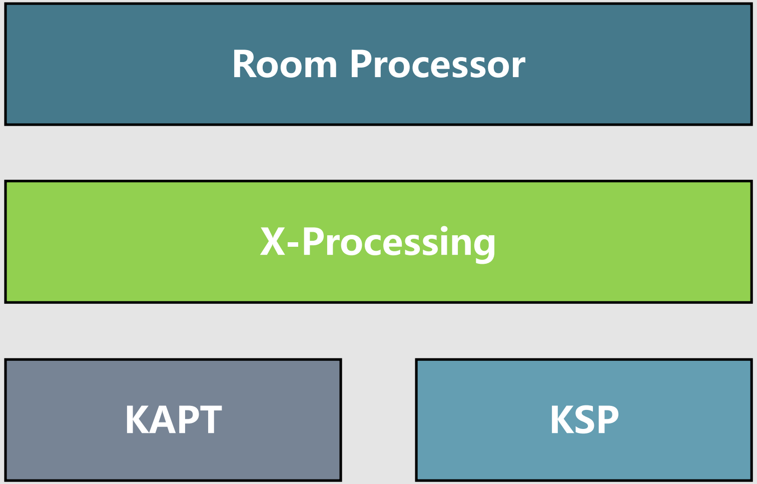
. The KSP team took a stab at this and was able to come up with a prototype, but wasn't able to make it production ready.

Since Glide is an open source project, anyone is welcome to contribute. The best way to apply social pressure is to of course spend your time on the improvement :)

This isn't completely straightforward, so if someone is interested, please reach out to me directly before you get too far so we can talk about it.

bumptech/glide

X Processing

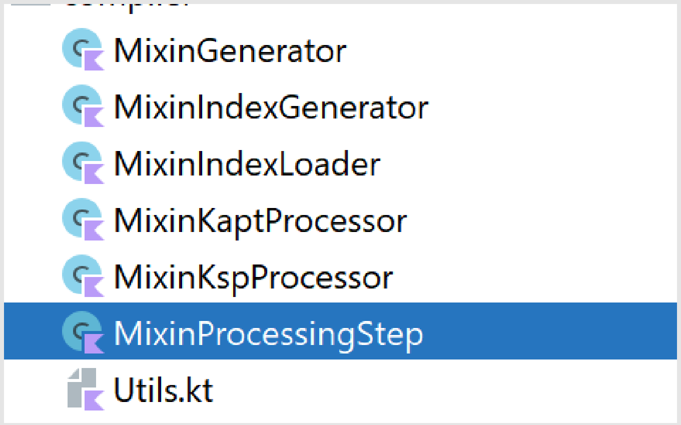


Room Processor

X-Processing

KAPT

KSP



MixinGenerator

MixinIndexGenerator

MixinIndexLoader

MixinKaptProcessor

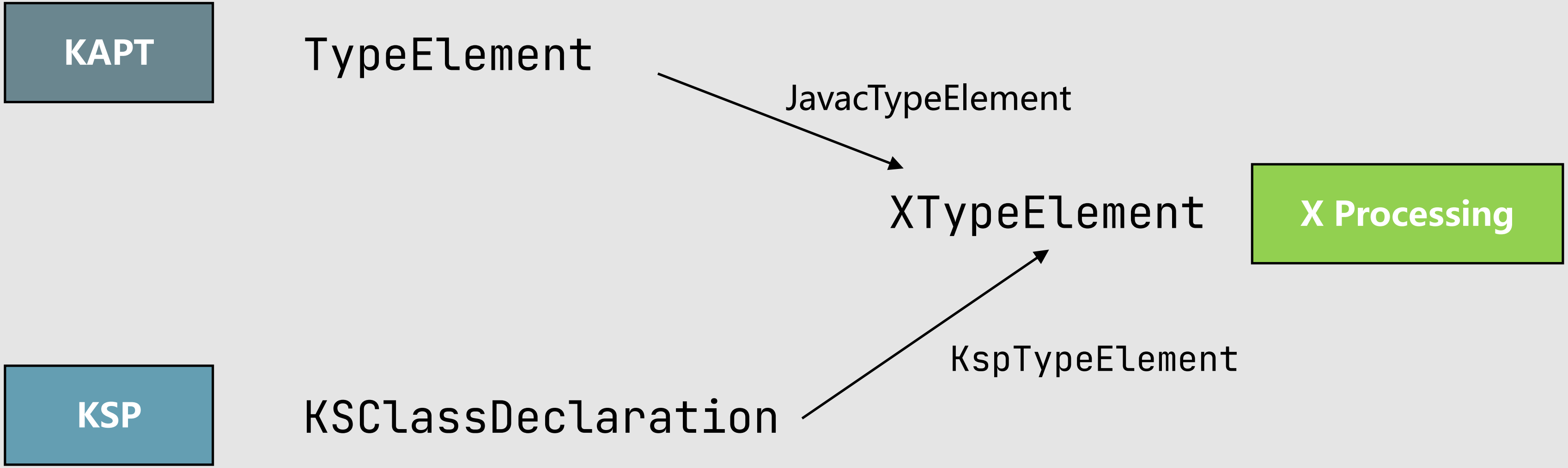
MixinKspProcessor

MixinProcessingStep

Utils.kt

```
implementation("androidx.room:room-compiler-processing:2.4.0-rc01")
```

X Processing



KAPT

TypeElement

JavacTypeElement

XTypeElement

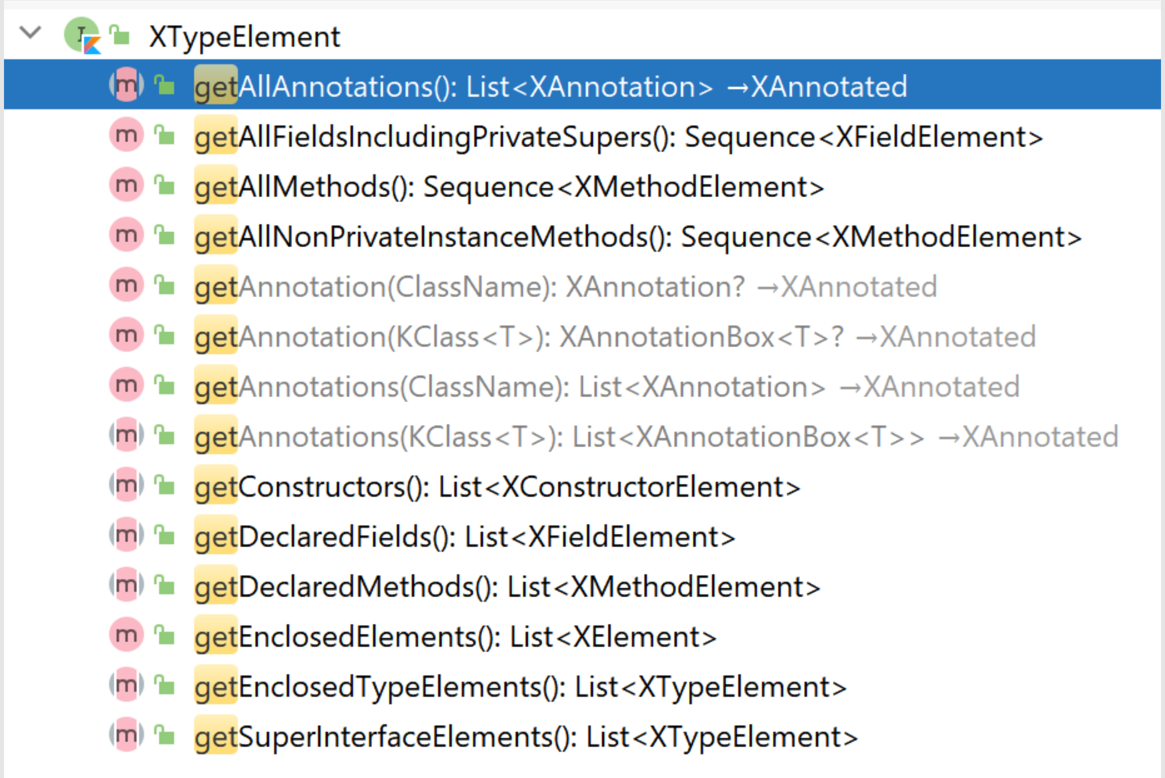
X Processing

KspTypeElement

KSP

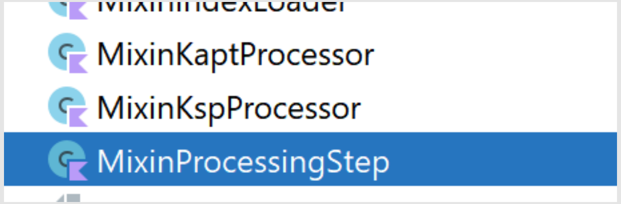
KSClassDeclaration

X Processing

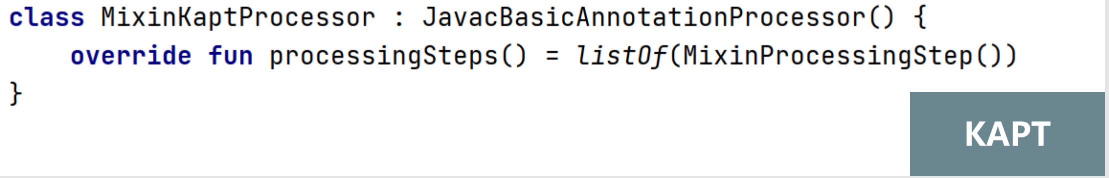


```
XTypeElement
    getAllAnnotations(): List<XAnnotation> →XAnnotated
    getAllFieldsIncludingPrivateSupers(): Sequence<XFieldElement>
    getAllMethods(): Sequence<XMethodElement>
    getAllNonPrivateInstanceMethods(): Sequence<XMethodElement>
    getAnnotation(ClassName): XAnnotation? →XAnnotated
    getAnnotation(KClass<T>): XAnnotationBox<T>? →XAnnotated
    getAnnotations(ClassName): List<XAnnotation> →XAnnotated
    getAnnotations(KClass<T>): List<XAnnotationBox<T>> →XAnnotated
    getConstructors(): List<XConstructorElement>
    getDeclaredFields(): List<XFieldElement>
    getDeclaredMethods(): List<XMethodElement>
    getEnclosedElements(): List<XElement>
    getEnclosedTypeElements(): List<XTypeElement>
    getSuperInterfaceElements(): List<XTypeElement>
```

X Processing

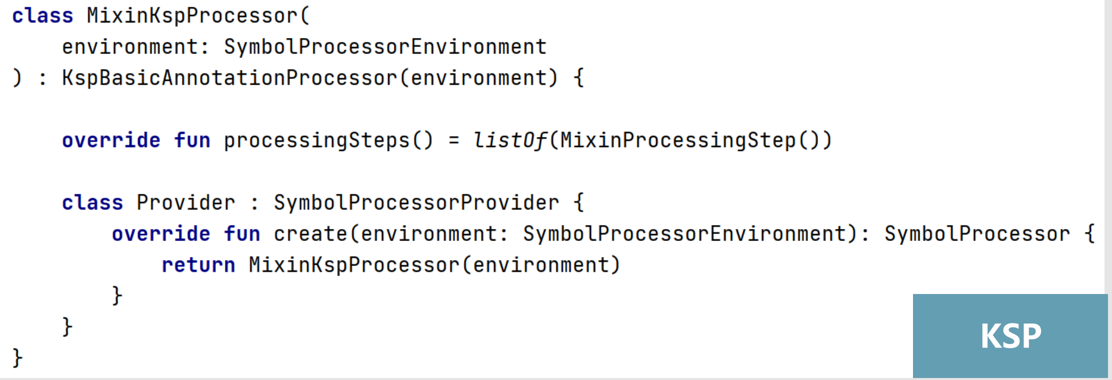


MixinIndexLoader
MixinKaptProcessor
MixinKspProcessor
MixinProcessingStep



```kotlin
class MixinKaptProcessor : JavacBasicAnnotationProcessor() {
    override fun processingSteps() = listOf(MixinProcessingStep())
}
```

KAPT



```kotlin
class MixinKspProcessor(
    environment: SymbolProcessorEnvironment
) : KspBasicAnnotationProcessor(environment) {

    override fun processingSteps() = listOf(MixinProcessingStep())

    class Provider : SymbolProcessorProvider {
        override fun create(environment: SymbolProcessorEnvironment): SymbolProcessor {
            return MixinKspProcessor(environment)
        }
    }
}
```

KSP

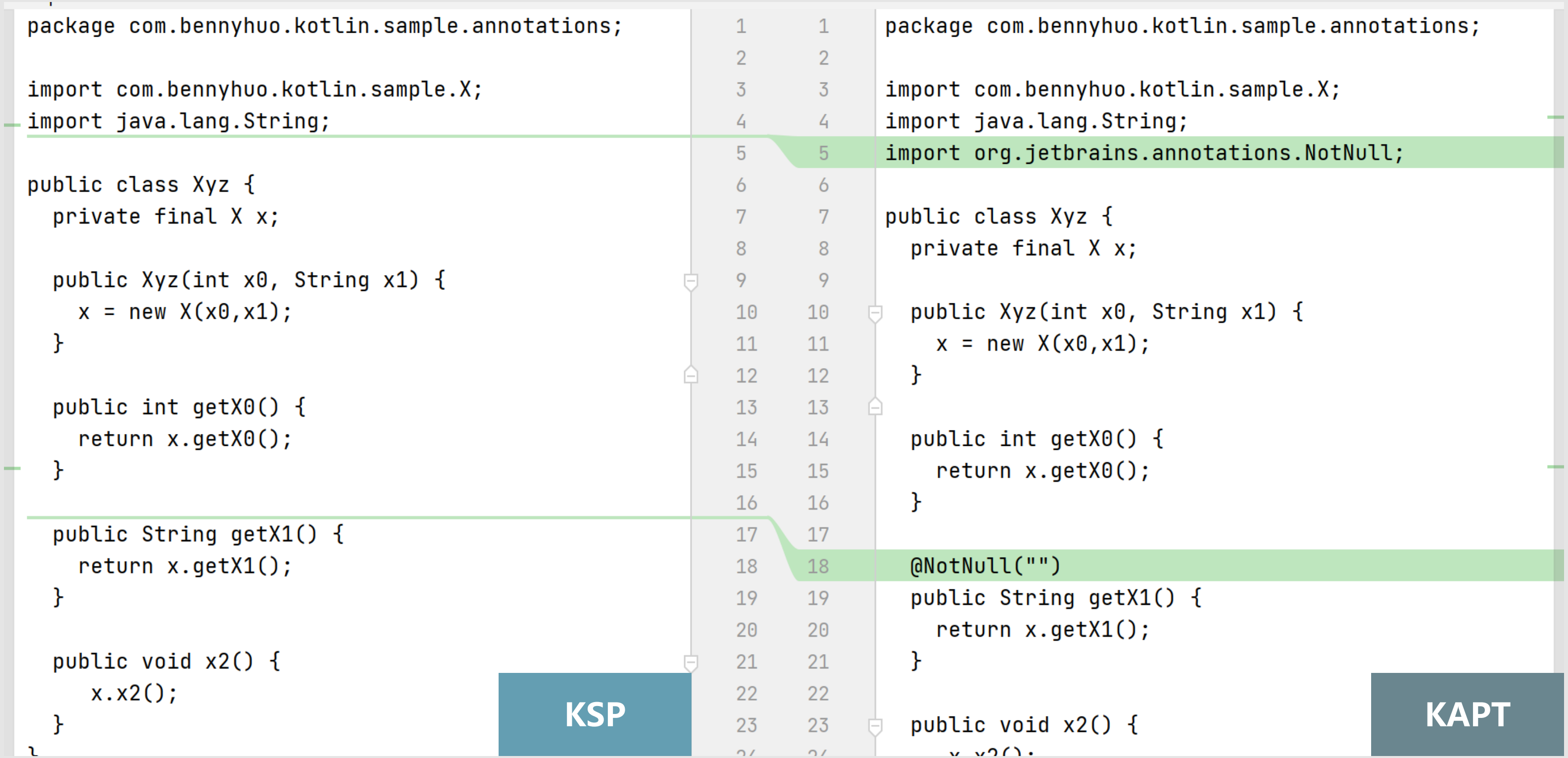
对 JavaPoet 和 KotlinPoet 的支持

```
interface XFiler {

    fun write(javaFile: JavaFile, mode: Mode = Mode.Isolating)

    fun write(fileSpec: FileSpec, mode: Mode = Mode.Isolating)


}
```

对 JavaPoet 和 KotlinPoet 的支持

```
interface XType {
    /**
     * The Javapoet [TypeName] representation of the type
     */
    val typeName: TypeName
    ...
}

interface XTypeElement : XHasModifiers, XElement, XMemberContainer {

    /**
     * Javapoet [ClassName] of the type.
     */
    override val className: ClassName
    ...
}
```

X Processing 注解处理的差异



```java
package com.bennyhuo.kotlin.sample.annotations;

import com.bennyhuo.kotlin.sample.X;
import java.lang.String;

public class Xyz {
  private final X x;

  public Xyz(int x0, String x1) {
    x = new X(x0,x1);
  }

  public int getX0() {
    return x.getX0();
  }

  public String getX1() {
    return x.getX1();
  }

  public void x2() {
     x.x2();
  }
}
```

KSP

```java
package com.bennyhuo.kotlin.sample.annotations;

import com.bennyhuo.kotlin.sample.X;
import java.lang.String;
import org.jetbrains.annotations.NotNull;

public class Xyz {
  private final X x;

  public Xyz(int x0, String x1) {
    x = new X(x0,x1);
  }

  public int getX0() {
    return x.getX0();
  }

  @NotNull("")
  public String getX1() {
    return x.getX1();
  }

  public void x2() {
    x.x2();
```

KAPT

如何编写处理器的单元测试?

单元测试的目标

- 明确各种场景下的代码逻辑的稳定性
- 确保不同的处理器的产物的一致性
- 方便单步调试编译器的编译过程
- 支持基于多文件、多模块的 Case 编写

测试编译器的框架

- **google/compile-testing**
  - Testing tools for javac and annotation processors
- **tschuchortdev/kotlin-compile-testing**
  - A library for testing Kotlin and Java annotation processors, compiler plugins and code generation

kotlin-compile-testing - 读取测试用例

```
val kotlinSource = SourceFile.kotlin(
    "KClass.kt", """
    class KClass {
        fun foo() {
            val testEnvClass = TestEnvClass()
        }
    }
""")

val javaSource = SourceFile.java(
    "JClass.java", """
    public class JClass {
        public void bar() {
            KClass kClass = new KClass();
        }
    }
""")
```

kotlin-compile-testing - 执行编译

```kotlin
val result = KotlinCompilation().apply {
    sources = listOf(kotlinSource, javaSource)

    // for KAPT
    annotationProcessors = listOf(MyAnnotationProcessor())

    // for KSP
    symbolProcessorProviders = listOf(MyKspProcessorProvider())

    // for KCP
    compilerPlugins = listOf(MyComponentRegistrar())
    commandLineProcessors = listOf(MyCommandlineProcessor())

    inheritClassPath = true
}.compile()
```

kotlin-compile-testing - 编译结果

```
inner class Result(
      /** The exit code of the compilation */
      val exitCode: ExitCode,
      /** Messages that were printed by the compilation */
      val messages: String
) {
      val outputDirectory: File

      val sourcesGeneratedByAnnotationProcessor: List<File>

      val compiledClassAndResourceFiles: List<File>

      val generatedStubFiles: List<File>

      val generatedFiles: Collection<File>
}
```

kotlin-compile-testing - 编译结果

```
class KotlinCompilation : AbstractKotlinCompilation<...>() {
   val classesDir: File
   val kaptSourceDir: File
   val kaptStubsDir: File
   ...
}

val KotlinCompilation.kspSourcesDir: File
```

添加多模块的支持

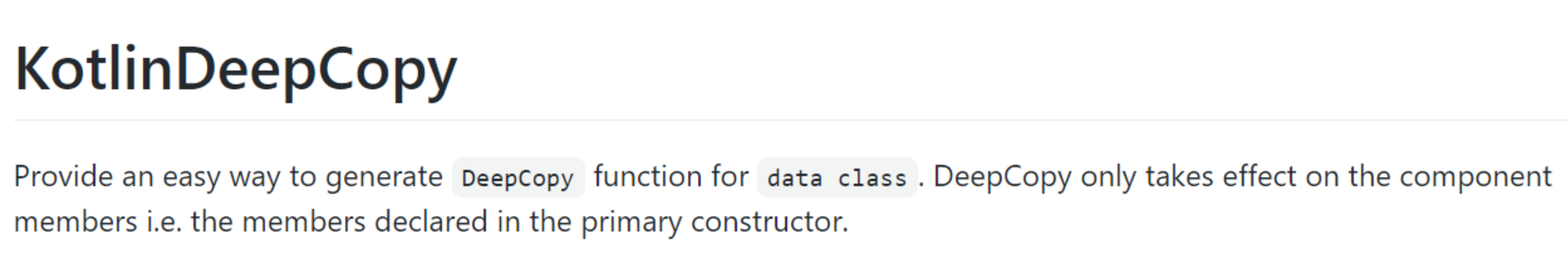
```
abstract class Module(val name: String) {
    abstract val classesDir: File
    val classpaths = ArrayList<File>()

    val dependencies = ArrayList<Module>()

    fun dependsOn(libraryUnit: Module) {
        classpaths += libraryUnit.classesDir
        classpaths += libraryUnit.classpaths
        dependencies += libraryUnit
    }

    protected val compilation: KotlinCompilation = newCompilation()

    protected fun newCompilation(): KotlinCompilation {
        return KotlinCompilation().also { compilation ->
            compilation.inheritClassPath = true
            compilation.classpaths = classpaths
        }
    }
    ...
}
```

基于多模块的测试用例示意

```
// SOURCE
// MODULE: library-a
class X(val x0: Int, val x1: String) { ... }

// MODULE: library-b / library-a   , library-c
class Y(val y0: IntArray, val y1: Array<String>) { ... }

class Z { ... }

// GENERATED
// MODULE: library-a
// FILE: Xyz.java
public class Xyz { ... }

// MODULE: library-b
// FILE: Xyz.java
public class Xyz { ... }
```

DeepCopy 项目地址

- https://github.com/bennyhuo/KotlinDeepCopy

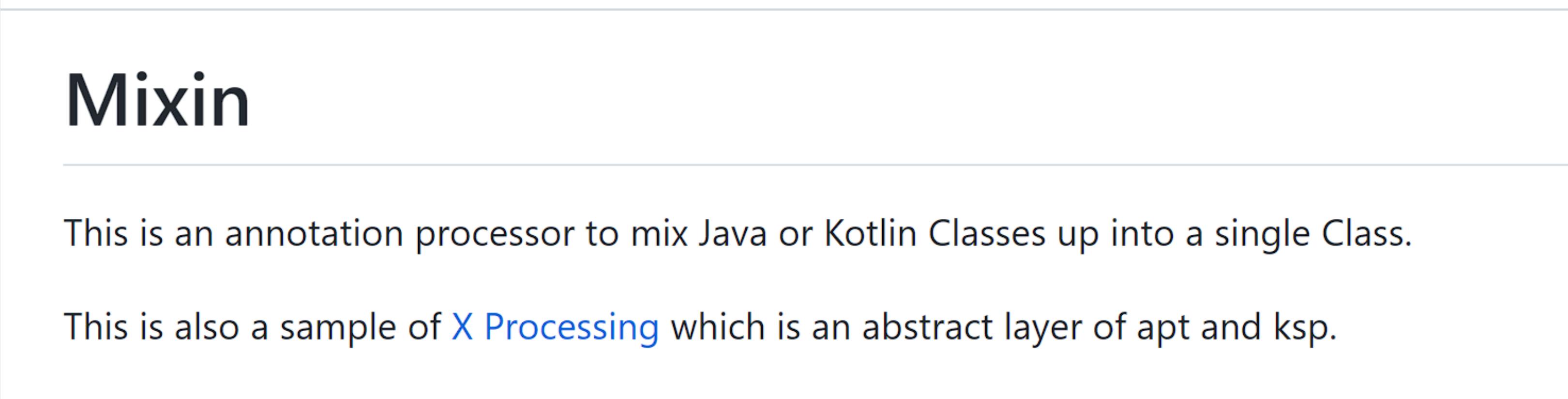


KotlinDeepCopy

Provide an easy way to generate `DeepCopy` function for `data class`. DeepCopy only takes effect on the component members i.e. the members declared in the primary constructor.

Mixin 项目地址

- https://github.com/bennyhuo/Mixin



Mixin

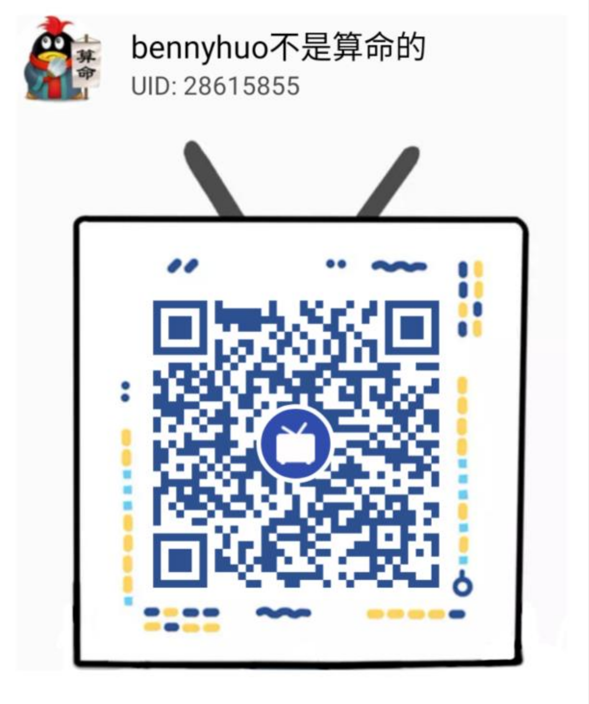
This is an annotation processor to mix Java or Kotlin Classes up into a single Class.

This is also a sample of X Processing which is an abstract layer of apt and ksp.

关注我

微信搜一搜

Kotlin



bennyhuo不是算命的

UID: 28615855

谢谢大家