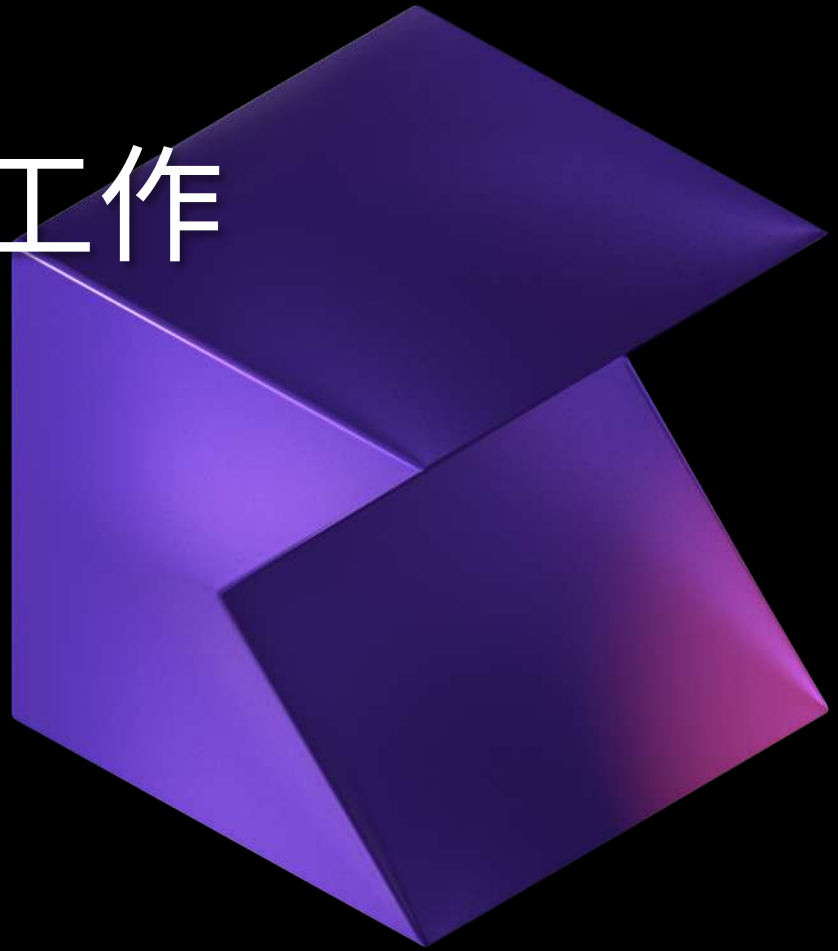
Kotlin

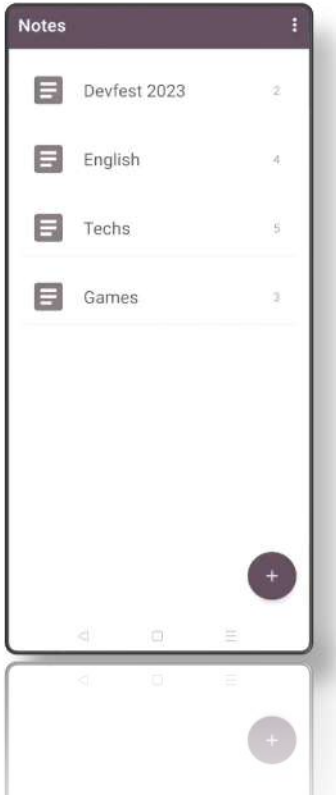
KSP让你我的工作更加轻松



叶楠
Orange中国创新实验室

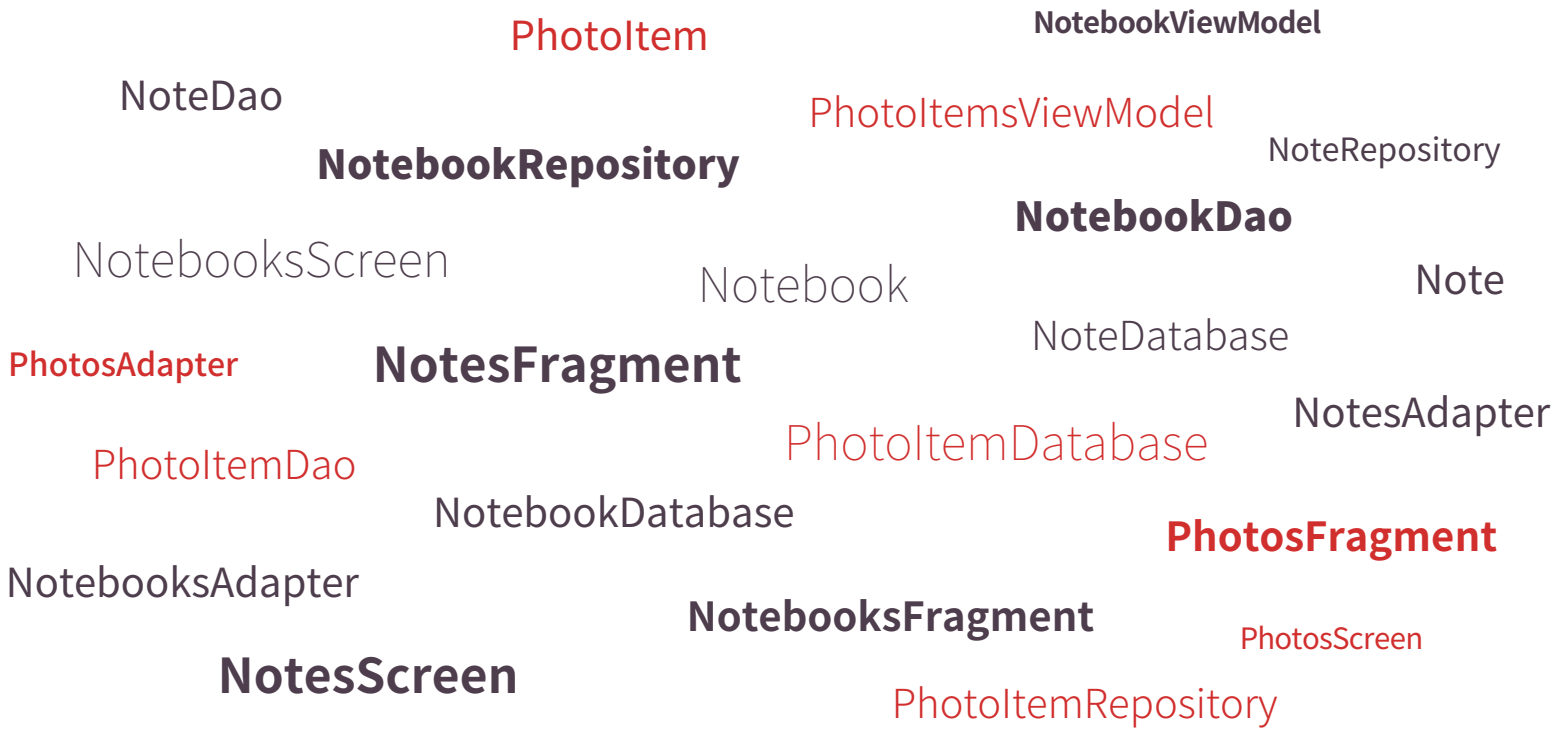
这个我们怎么做?

记事本应用



线上图片浏览器

这些我们都要做吗?


PhotoItem
NotebookViewModel
NoteDao
PhotoItemsViewModel
NoteRepository
NotebookRepository
NotebookDao
NotebooksScreen
Notebook
Note
NoteDatabase
PhotosAdapter
NotesFragment
NotesAdapter
PhotoItemDatabase
PhotoItemDao
NotebookDatabase
PhotosFragment
NotebooksAdapter
NotebooksFragment
PhotosScreen
NotesScreen
PhotoItemRepository

自动生成代码

Note

Notebook

PhotoItem

手动编写

@RoomCompanion

NotebookDao　NoteDao　PhotoItemDao

NotebookDatabase　NoteDatabase　PhotoItemDatabase

NotebookRepository　NoteRepository　PhotoItemRepository

@ViewModel

NotebookViewModel　PhotoItemsViewModel

@Adapter

NotebooksAdapter　NotesAdapter　PhotosAdapter

@ListFragement

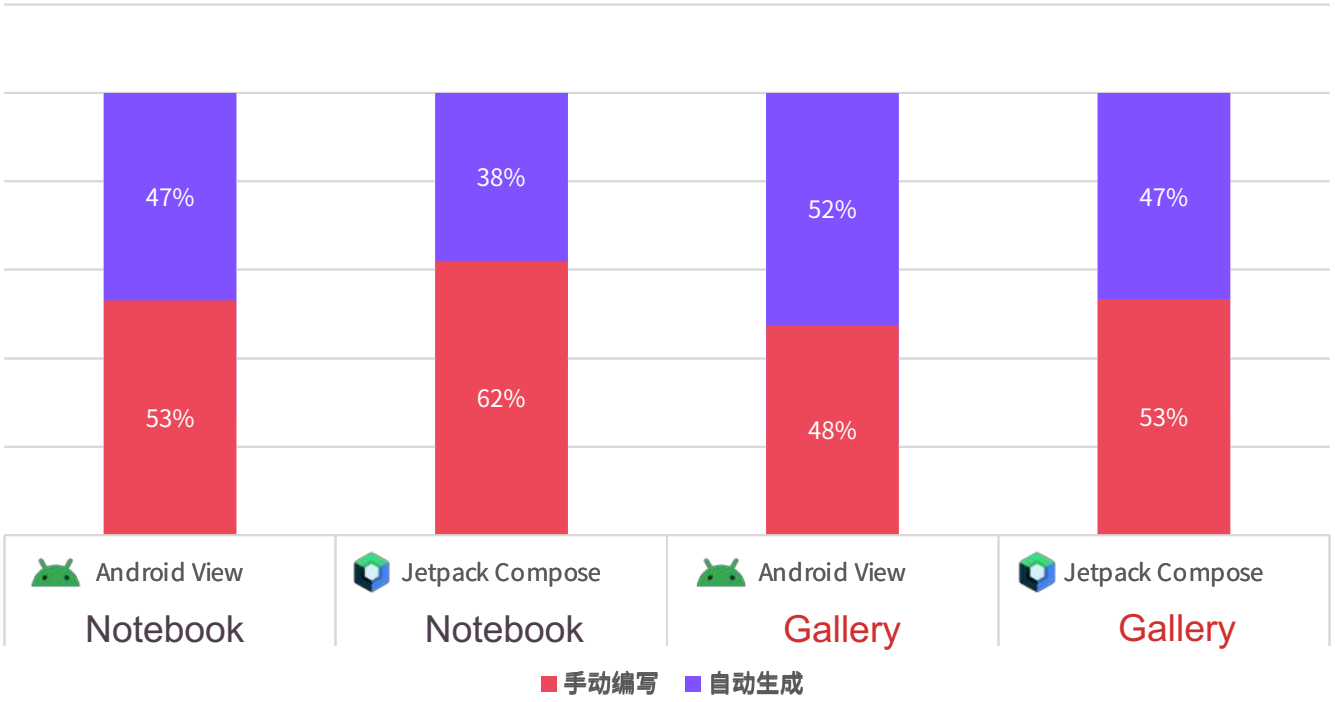
NotebooksFragment　NotesFragment　PhotosFragment

@ListScreen

NotebooksScreen　NotesScreen　PhotosScreen

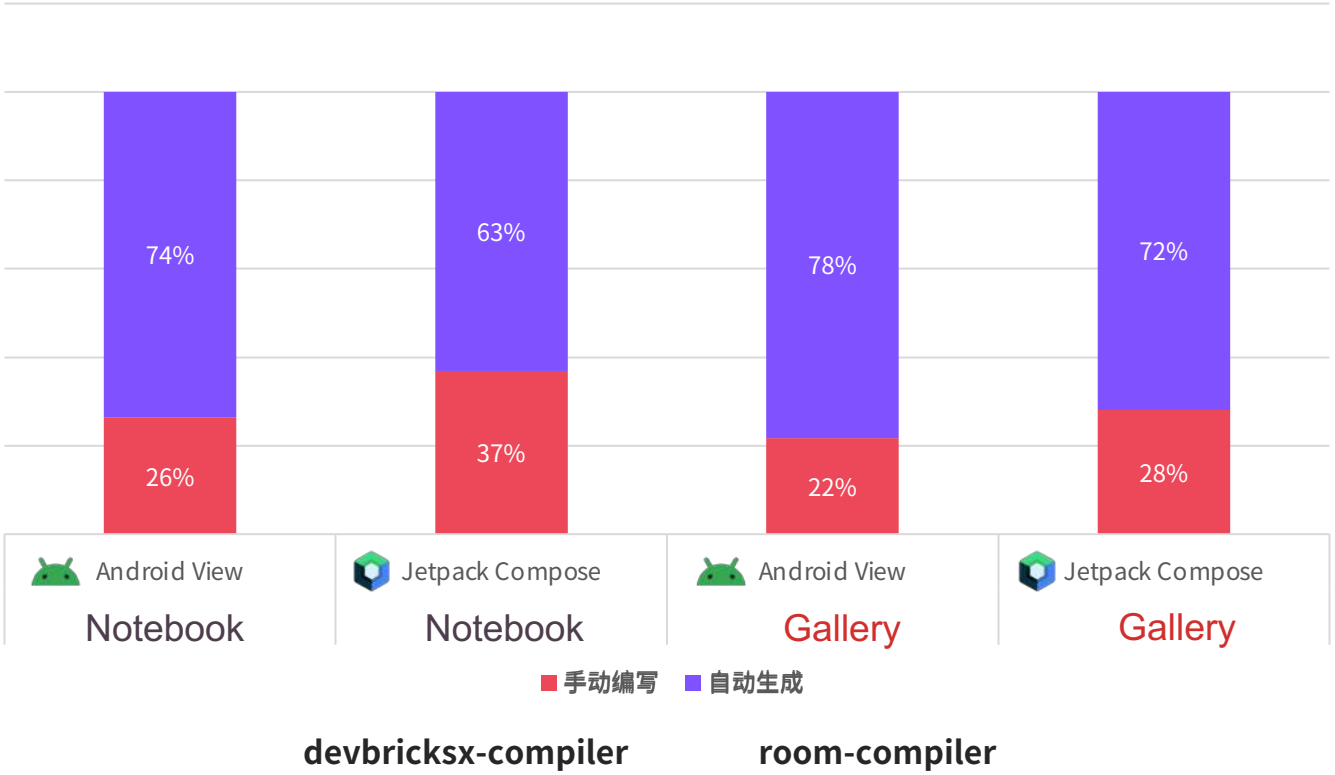
自动生成

代码少写了吗?



devbricksx-compiler

代码少写了吗?


74%
26%
63%
37%
78%
22%
72%
28%
Android View
Notebook
Jetpack Compose
Notebook
Android View
Gallery
Jetpack Compose
Gallery
手动编写
自动生成
devbricksx-compiler
room-compiler

我们应该选择什么？

kapt

Kotlin Annotation Processing Tool

支持组件全面　维护阶段　效率低

不支持多轮编译　内存溢出严重

KSP

Kotlin Symbol Processing

编译效率高　支持多轮编译　内存溢出少

组件支持有待完善

我们应该选择什么？

kapt

Kotlin Annotation Processing Tool

支持组件全面　维护阶段　效率低

不支持多轮编译　内存溢出严重

KSP

Kotlin Symbol Processing

编译效率高　支持多轮编译　内存溢出少

组件支持有待完善

我们应该选择什么?

KSP

Kotlin Symbol Processing

编译效率高

不再需要生成类Stub的中间产物去兼容Java注解处理器，节省了大半的编译时间

支持多轮编译

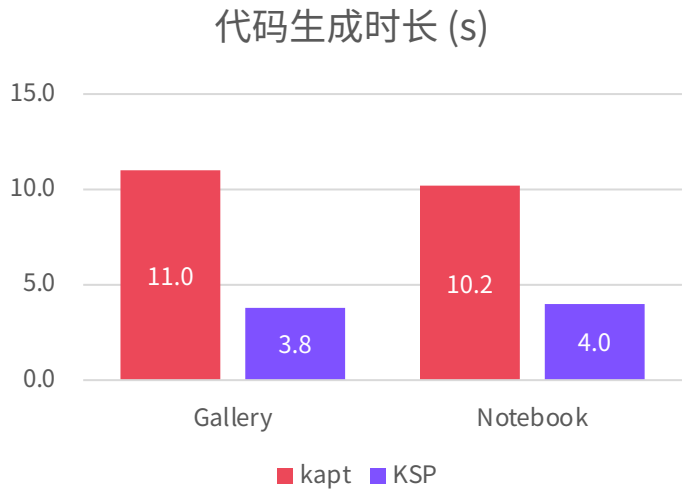
生成的文件中，如果还包含代码的注解，可以在新的一轮符号处理中继续处理

内存溢出少

增量编译得到了很好的支持，因此重复编译而导致内容溢出的现象大大减少

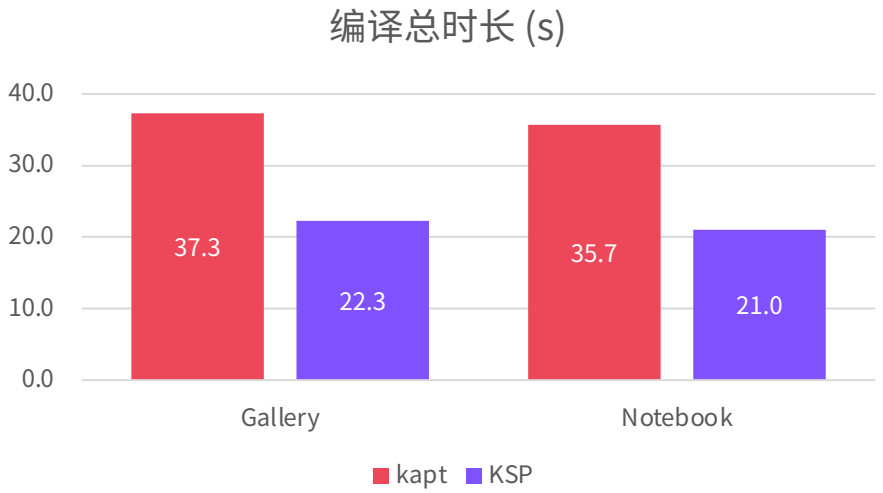
编译效率高

用时（秒）/ 模块		Gallery	Notebook
kapt	kapt任务	3.7	3.7
	kapt生成Stub	7.3	6.5
	代码生成时长	11.0	10.2
	编译总时长	37.3	35.7
KSP	ksp任务	3.8	4.0
	代码生成时长	3.8	4.0
	编译总时长	22.3	21.0



编译效率高

用时（秒）/ 模块		Gallery	Notebook
kapt	kapt任务	3.7	3.7
	kapt生成Stub	7.3	6.5
	代码生成时长	11.0	10.2
	编译总时长	37.3	35.7
KSP	ksp任务	3.8	4.0
	代码生成时长	3.8	4.0
	编译总时长	22.3	21.0



我们要怎么做?


NotebookViewModel
PhotoItem
NoteDao
PhotoItemsViewModel
NoteRepository
NotebookRepository
NotebookDao
NotebooksScreen
Notebook
Note
NoteDatabase
NotesFragment
NotesAdapter
PhotoItemDao
PhotoItemDatabase
NotebookDatabase
PhotosFragment
NotebooksAdapter
NotebooksFragment
PhotosScreen
NotesScreen
PhotoItemRepository

我们要怎么做?

Notebook

界面层（UI Layer）

NotebooksFragment　NotebooksAdapter　NotebooksScreen

NotebookViewModel

仓库（Repositories）

NotebookRepository

数据源（Data Sources）　Room

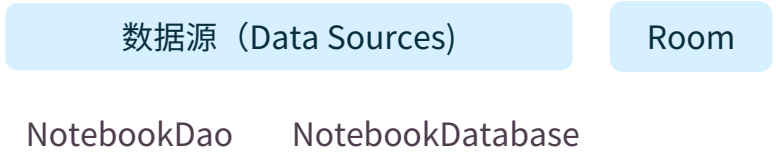
NotebookDao　NotebookDatabase

我们要怎么做?

@Entity

@RoomCompanion

Notebook



我们要做什么？

@RoomCompanion

Notebook

数据源（Data Sources）

Room

_Notebook

NotebookDao

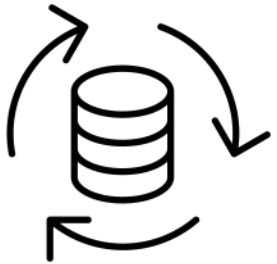
NotebookDatabase

@Entity

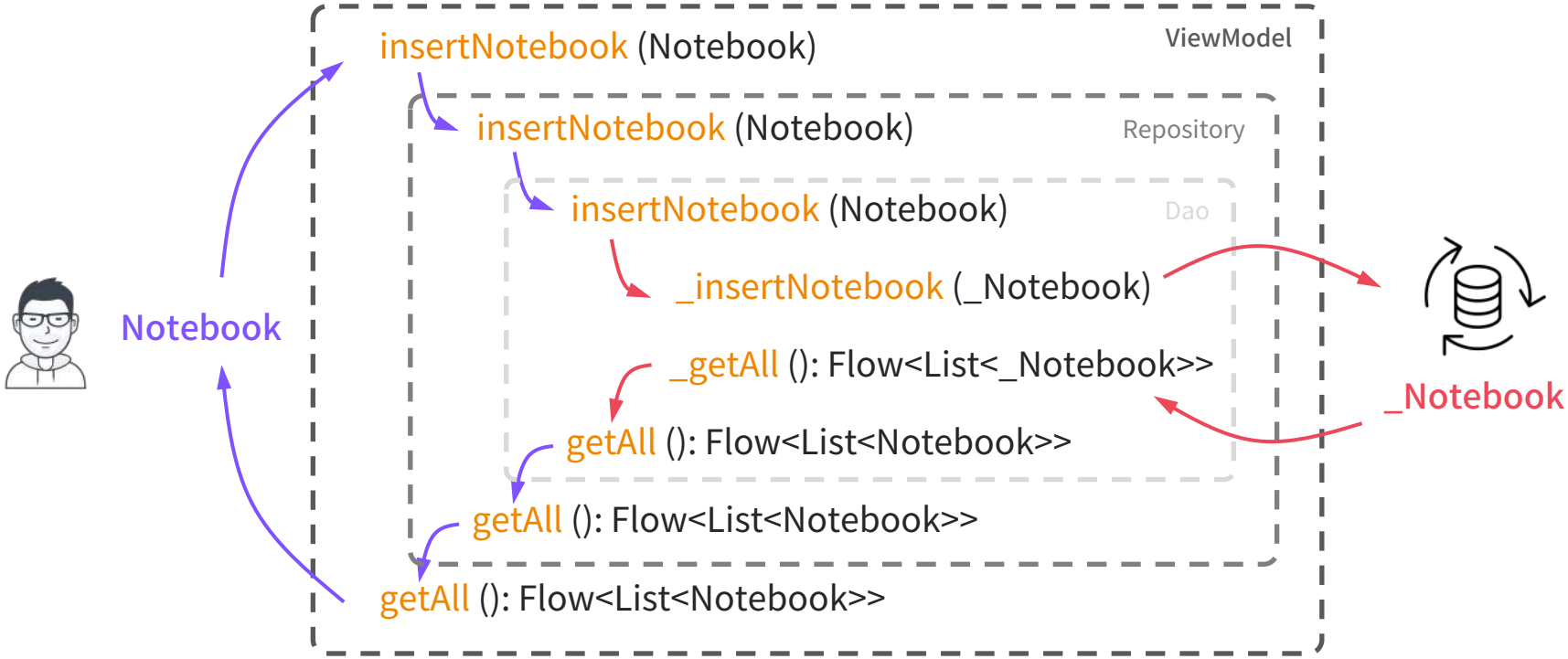
生成一个伙伴类

Notebook

_Notebook



生成一个伙伴类


ViewModel
insertNotebook (Notebook)
Repository
insertNotebook (Notebook)
Dao
insertNotebook (Notebook)
_insertNotebook (_Notebook)
_getAll (): Flow<List<_Notebook>>
getAll (): Flow<List<Notebook>>
getAll (): Flow<List<Notebook>>
getAll (): Flow<List<Notebook>>
Notebook
_Notebook

生成一个伙伴类

Notebook

id name created lastModified

id name created lastModified

_Notebook

生成一个伙伴类

Notebook

id name created lastModified

id
@ColumnInfo
@PrimaryKey

name
@ColumnInfo

created
@ColumnInfo

lastModified
@ColumnInfo

_Notebook
@Entity

生成一个伙伴类

Notebook

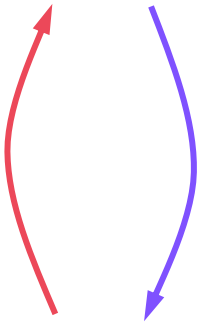
id , name , created , lastModified

copyFieldsToObject (Notebook)

toObject (): Notebook

fromObject (Notebook): _Notebook

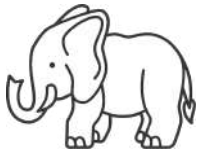
copyFieldsFromObject (Notebook)



id , name , created , lastModified

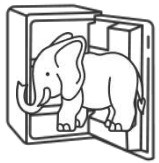
_Notebook

开发的步骤



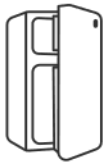
定义注解

根据需要，定义代码注解。在需要处理的符号上，用注解标注



访问符号

通过KSP提供的API访问被标记的符号的内容，例如名称，属性，方法，等等



生成代码

根据访问到的符号内容，使用第三方的库，生成相应的代码

使用代码注解

```
@RoomCompanion(
    primaryKeys = ["id"],
    autoGenerate = true,
    converters = [DateConverter::class],
    extension = NotebookDaoExtension::class,
    database = "notes",
)

open class Notebook(id: Int = 0) : Record(id) {

    var name: String? = null

}
```

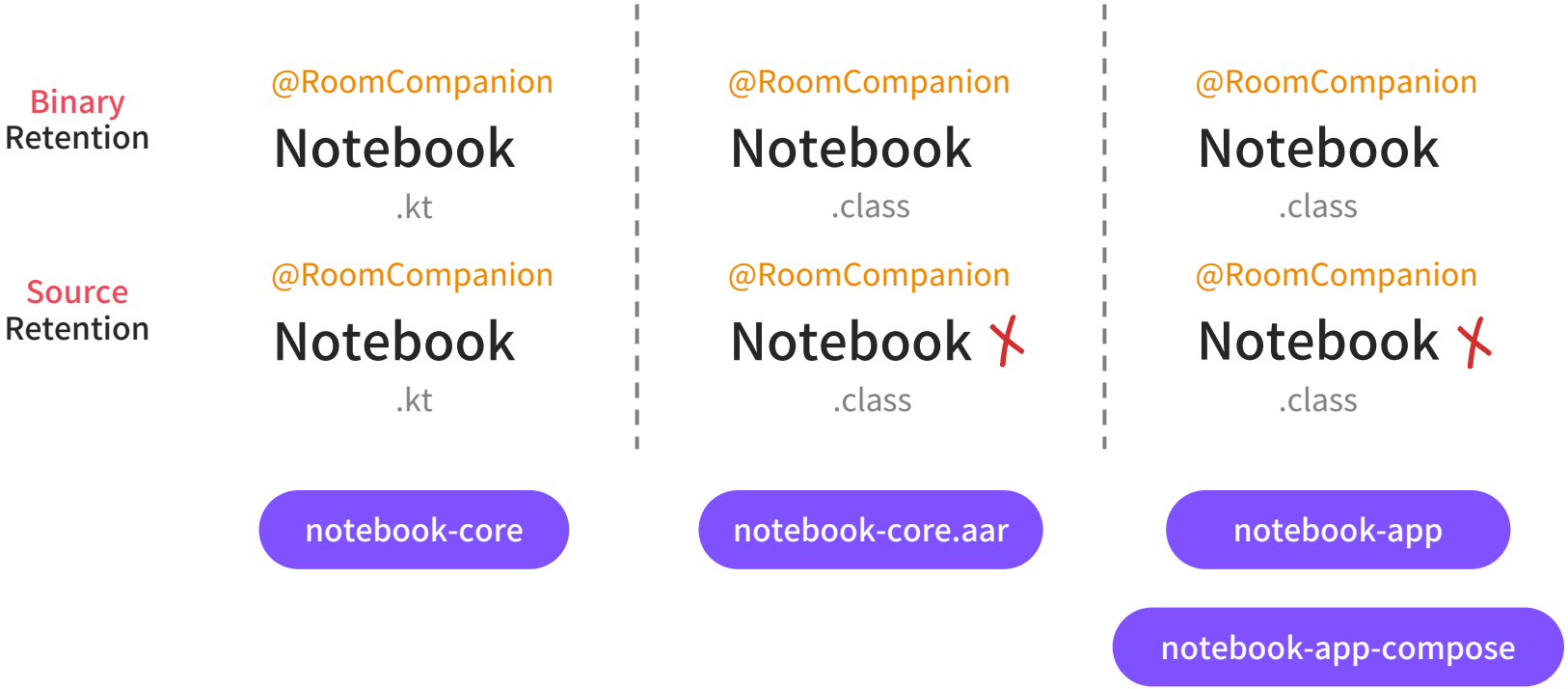
定义代码注解

```
@Retention(AnnotationRetention.BINARY)
@Target(AnnotationTarget.CLASS)
annotation class RoomCompanion(

    val primaryKeys: Array<String> = [],
    val autoGenerate: Boolean = false,
    val converters: Array<KClass<*>> = [],
    val extension: KClass<*> = Unit::class,
    val database: String = "",

)
```

注解的保留


Binary
Retention
Source
Retention
@RoomCompanion
Notebook
.kt
@RoomCompanion
Notebook
.kt
@RoomCompanion
Notebook
.class
@RoomCompanion
Notebook
.class
@RoomCompanion
Notebook
.class
@RoomCompanion
Notebook
.class
notebook-core
notebook-core.aar
notebook-app
notebook-app-compose

寻找代码符号

```
val symbols = resolver.getSymbolsWithAnnotation(
    RoomCompanion::class.qualifiedName!!
).filterIsInstance<KSClassDeclaration>()
```

Resolver 提供编译器和符号相关的各种操作

getSymbolsWithAnnotation(RoomCompanion)

找到被注解标记的符号

KSClassDeclaration 过滤除了类声明以外的符号

访问符号内容

```
package com.dailystudio.devbricksx.notebook.db

@RoomCompanion
class Notebook(id: Int = 0) : Record(id) {
    var name: String? = null
    var lastModified: Date? = null
    override fun countNotes(): Int { ... }
    fun displayName(prefix: String): String { ... }
}
```

访问符号内容

```
package com.dailystudio.devbricksx.notebook.db

@RoomCompanion
class Notebook(id: Int = 0) : Record(id) {
    var name: String? = null
    var lastModified: Date? = null
    override fun countNotes(): Int { ... }
    fun displayName(prefix: String): String { ... }
}
```

KSClassDeclaration.packageName() 获取符号所在包名

访问符号内容

KSClassDeclaration.packageName() 获取符号所在包名

```
package com.dailystudio.devbricksx.notebook.db

@RoomCompanion
class Notebook(id: Int = 0) : Record(id) {
    var name: String? = null
    var lastModified: Date? = null
    override fun countNotes(): Int { ... }
    fun displayName(prefix: String): String { ... }
}
```

KSClassDeclaration.typeName() 获取符号类名

访问符号内容

KSClassDeclaration.packageName() 获取符号所在包名

KSClassDeclaration.typeName() 获取符号类名

```
package com.dailystudio.devbricksx.notebook.db

@RoomCompanion

class Notebook(id: Int = 0) : Record(id) {
    var name: String? = null
    var lastModified: Date? = null
    override fun countNotes(): Int { ... }
    fun displayName(prefix: String): String { ... }
}
```

KSClassDeclaration.primaryConstructor 获取符号的主构造函数

访问符号内容

KSClassDeclaration.packageName() 获取符号所在包名

KSClassDeclaration.typeName() 获取符号类名

KSClassDeclaration.primaryConstructor 获取符号的主构造函数

```
package com.dailystudio.devbricksx.notebook.db

@RoomCompanion
class Notebook(id: Int = 0) : Record(id) {
    var name: String? = null
    var lastModified: Date? = null
    override fun countNotes(): Int { ... }
    fun displayName(prefix: String): String { ... }
}
```

KSClassDeclaration.getAllProperties() 获取符号的所有属性

访问符号内容

KSClassDeclaration.packageName() 获取符号所在包名

KSClassDeclaration.typeName() 获取符号类名

KSClassDeclaration.primaryConstructor 获取符号的主构造函数

KSClassDeclaration.getAllProperties() 获取符号的所有属性

```
package com.dailystudio.devbricksx.notebook.db

@RoomCompanion
class Notebook(id: Int = 0) : Record(id) {
    var name: String? = null
    var lastModified: Date? = null

    override fun countNotes(): Int { ... }

    fun displayName(prefix: String): String { ... }
}
```

KSClassDeclaration.getAllFunctions() 获取符号的所有函数

访问符号内容

KSClassDeclaration.packageName()	获取符号所在包名
KSClassDeclaration.typeName()	获取符号类名
KSClassDeclaration.primaryConstructor	获取符号的主构造函数
KSClassDeclaration.getAllProperties()	获取符号的所有属性
KSClassDeclaration.getAllFunctions()	获取符号的所有函数

代码的生成

Square Open Source

KotlinPoet

Version 1.15.1

代码的生成

Square Open Source

KotlinPoet

Version 1.14+

```
import androidx.lifecycle.map as LifecycleMap
import kotlinx.coroutines.flow.map as FlowMap
```

代码的生成

```
@Entity(tableName = “notebook”)
public open class _Notebook(
  @ColumnInfo(name = “id”)
  @PrimaryKey(autoGenerate = true)
  public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = “name”)
  public open var name: String? = null

  public fun copyFieldsToObject(notebook: Notebook) {
    super.copyFieldsToObject(notebook)
```

代码的生成

```
@Entity(tableName = "notebook")
public open class _Notebook (
  @ColumnInfo(name = "id")
  @PrimaryKey(autoGenerate = true)
  public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = "name")
  public open var name: String? = null

  public fun copyFieldsToObject(notebook: Notebook) {
```

```
TypeSpec.classBuilder(...)
  .addModifiers(...)
  .build()
```

TypeSpec

生成类的定义

代码的生成

TypeSpec

```
@Entity(tableName = “notebook”)
public open class _Notebook(
  @ColumnInfo(name = “id”)
  @PrimaryKey(autoGenerate = true)
  public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = “name”)
  public open var name: String? = null
```

```
AnnotationSpec.builder(...)
    .addMember(...)
    .build()
```

AnnotationSpec

生成注解的定义

代码的生成

TypeSpec

AnnotationSpec

```
    public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = "name")
  public open var name: String? = null

  public fun copyFieldsToObject(notebook: Notebook) {
    super.copyFieldsToObject(notebook)
    notebook.name = this.name
  }

}
```

代码的生成

TypeSpec

AnnotationSpec

```
  public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = "name")
  public open var name: String? = null

  public fun copyFieldsToObject(notebook: Notebook) {
    super.copyFieldsToObject(notebook)
    notebook.name = this.name
  }

}
```

```
PropertySpec.builder(...)
    .addModifiers(...)
    .initializer(...)
    .build()
```

PropertySpec

生成属性的定义

代码的生成

TypeSpec *PropertySpec*

AnnotationSpec

```
  public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = "name")
  public open var name: String? = null

  public fun copyFieldsToObject( notebook: Notebook) {
    super.copyFieldsToObject(notebook)
    notebook.name = this.name
  }

}
```

```
FunSpec.builder(...)
    .addModifiers(...)
    .build()
```

FuncSpec

生成函数的定义

代码的生成

TypeSpec *PropertySpec*

AnnotationSpec *FuncSpec*

```
    public open override var id: Int,
) : _Record(id) {

    @ColumnInfo(name = "name")
    public open var name: String? = null

    public fun copyFieldsToObject( notebook: Notebook ) {
        super.copyFieldsToObject(notebook)
        notebook.name = this.name
    }

}
```

```
ParameterSpec.builder(...)
    .defaultValue(...)
    .build()
```

ParameterSpec

生成函数参数的定义

代码的生成

TypeSpec *PropertySpec* *ParameterSpec*

AnnotationSpec *FuncSpec*

```
public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = "name")
  public open var name: String? = null

  public fun copyFieldsToObject(notebook: Notebook) {
    super.copyFieldsToObject(notebook)
    notebook.name = this.name
  }

}
```

```
CodeBlock.builder()
    .addStatement(...)
    .build()
```

CodeBlock

生成函数体的定义

代码的生成

TypeSpec *PropertySpec* *ParameterSpec*

AnnotationSpec *FuncSpec* *CodeBlock*

```
    public open override var id: Int,
) : _Record(id) {

  @ColumnInfo(name = "name")
  public open var name: String? = null

  public fun copyFieldsToObject(notebook: Notebook) {
    super.copyFieldsToObject(notebook)
    notebook.name = this.name
  }

}
```

代码的生成

TypeSpec *PropertySpec*

AnnotationSpec

```
FuncSpec.builder()
        .addParameter( ParameterSpec )
        .add( CodeBlock )
        .build()
```

代码的生成

```
TypeSpec.builder()
        .addAnnotation( AnnotationSpec )
        .addProperty( PropertySpec )
        .addFunction( FuncSpec )
        .build()
```

代码的生成

```
FileSpec.builder()
      .addType( TypeSpec )
      .build()
      .writeTo()
```

增量编译

```
Dependencies(aggregating = false,
    KSClassDecalaration.containingFile,
    ...
)
```

增量编译

Dependencies

```
FileSpec.builder()
      .addType( TypeSpec )
      .build()
      .writeTo()
```

增量编译

```
FileSpec.builder()
      .addType( TypeSpec )
      .build()
      .writeTo( Dependencies )
```

神奇的影子类

Notebook

@RoomCompanion

NotebookDao

NotebookDatabase

NotebookRepository

@ViewModel

NotebookViewModel

notebook-core

Notebook

@Adapter ✗

NotebooksAdapter

@ListFragement ✗

NotebooksFragment

notebook-app

Notebook

@ListScreen ✗

NotebooksScreen

notebook-app-compose

神奇的影子类

Notebook

@RoomCompanion

NotebookDao

NotebookDatabase

NotebookRepository

@ViewModel

NotebookViewModel

notebook-core

__Notebook

@Adapter

NotebooksAdapter

@ListFragement

NotebooksFragment

notebook-app

__Notebook

@ListScreen

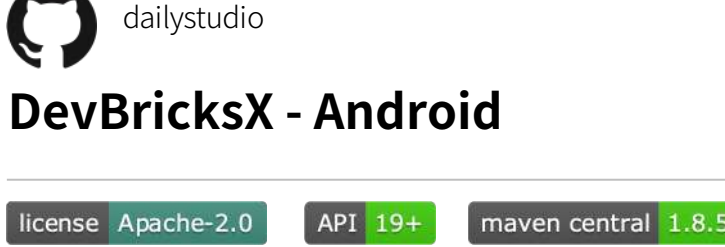
NotebooksScreen

notebook-app-compose

开源代码

dailystudio

DevBricksX - Android



DevBricksX提供了许多在日常Android开发中会用到的有用类，我们称之为“砖块”。有了这些“砖块”，您的开发将变得有效，可靠和高度一致的体验。

https://github.com/dailystudio/devbricksx-android

Thanks!
Have a nice Kotlin

@kotlin | Developed by JetBrains

Thanks!
Have a nice Kotlin

技术资源

代码生成

- **kapt** https://kotlinlang.org/docs/kapt.html
- KSP https://kotlinlang.org/docs/ksp-overview.html
- KotlinPoet https://square.github.io/kotlinpoet/

参考实例

- DevBricks X
  https://github.com/dailystudio/devbricksx-android

素材展示

- IconFinder https://www.iconfinder.com/
- Unsplash http://unsplash.com/

特别感谢

内容支持

- 范圣佑老师
- 霍丙乾老师
- Orange中国创新实验室

活动组织

JET BRAINS — **JetBrains中国**

@kotlin | Developed by JetBrains