Android Dependency Injection with Hilt

Thuy Phan

What is Dependency Injection?

```
class Car {

    private val engine = Engine()

    fun start() {
        engine.start()
    }
}

fun main(args: Array) {
    val car = Car()
    car.start()
}
```

```
class Car(private val engine: Engine) {
    fun start() {
        engine.start()
    }
}

fun main(args: Array) {
    val engine = Engine()
    val car = Car(engine)
    car.start()
}
```

Advantages of DI

- Reusability of classes and decoupling dependencies
- Ease of refactoring
- Ease of testing

Is DI hard in Android?

- Framework classes are instantiated by the SDK
- Factories added on API 28, not realistic

Dagger?

- Hard to configure
- Multiple paths to do the same thing

Android Dagger

Dagger

@ContributesAndroidInjector

AndroidInjector

@Component

@Subcomponent

@Inject

@Module

HasAndroidInjector

DispatchingAndroidInjector

Wait! I just want to code my feature???!!!

There are some days when i hate Dagger and other days, when i really really hate Dagger. #AndroidDev 🗡️ woes

17 19 186

Out of the most popular libraries, if I was forced to stop using one it would be Dagger without a second thought.

Give Award Share Report Save

PS: I know Dagger is not maintained by Android team but it is a big pain point on Android community.

10 13 84

DI Solution Golds

- Opinionated
- Easy to setup
- Let’s us focus on what is important

Hilt

Hilt

- Standardize DI in Android
- Build on the top of Dagger
- Tooling support
- AndroidX Extensions

Setup

1. Hilt application class

```
@HiltAndroidApp
class ExampleApplication : Application() { ... }
```

1. Hilt application class
2. Inject dependencies into Android classes

```
@AndroidEntryPoint
class ExampleActivity : AppCompatActivity() {

 @Inject lateinit var analytics: AnalyticsAdapter
 ...
}
```

1. Hilt application class
2. Inject dependencies into Android classes
3. Define Hilt bindings

```
class AnalyticsAdapter @Inject constructor(
   private val service: AnalyticsService
  ) { ... }
```

1. Hilt application class
2. Inject dependencies into Android classes
3. Define Hilt bindings
4. Hilt modules

```
interface AnalyticsService {
    fun analyticsMethods()
}
// Constructor-injected, because Hilt needs to know how to
// provide instances of AnalyticsServiceImpl, too.
class AnalyticsServiceImpl @Inject constructor(
    ...
) : AnalyticsService { ... }
@Module
@InstallIn(ActivityComponent::class)
abstract class AnalyticsModule {
    @Binds
    abstract fun bindAnalyticsService(
        analyticsServiceImpl: AnalyticsServiceImpl
    ): AnalyticsService
}
```

```
@Module
@InstallIn(ActivityComponent::class)
object AnalyticsModule {

 @Provides
 fun provideAnalyticsService(
   // Potential dependencies of this type
 ): AnalyticsService {
     return Retrofit.Builder()
               .baseUrl("https://example.com")
               .build()
               .create(AnalyticsService::class.java)
 }
}
```

Provide multiple bindings for the same type

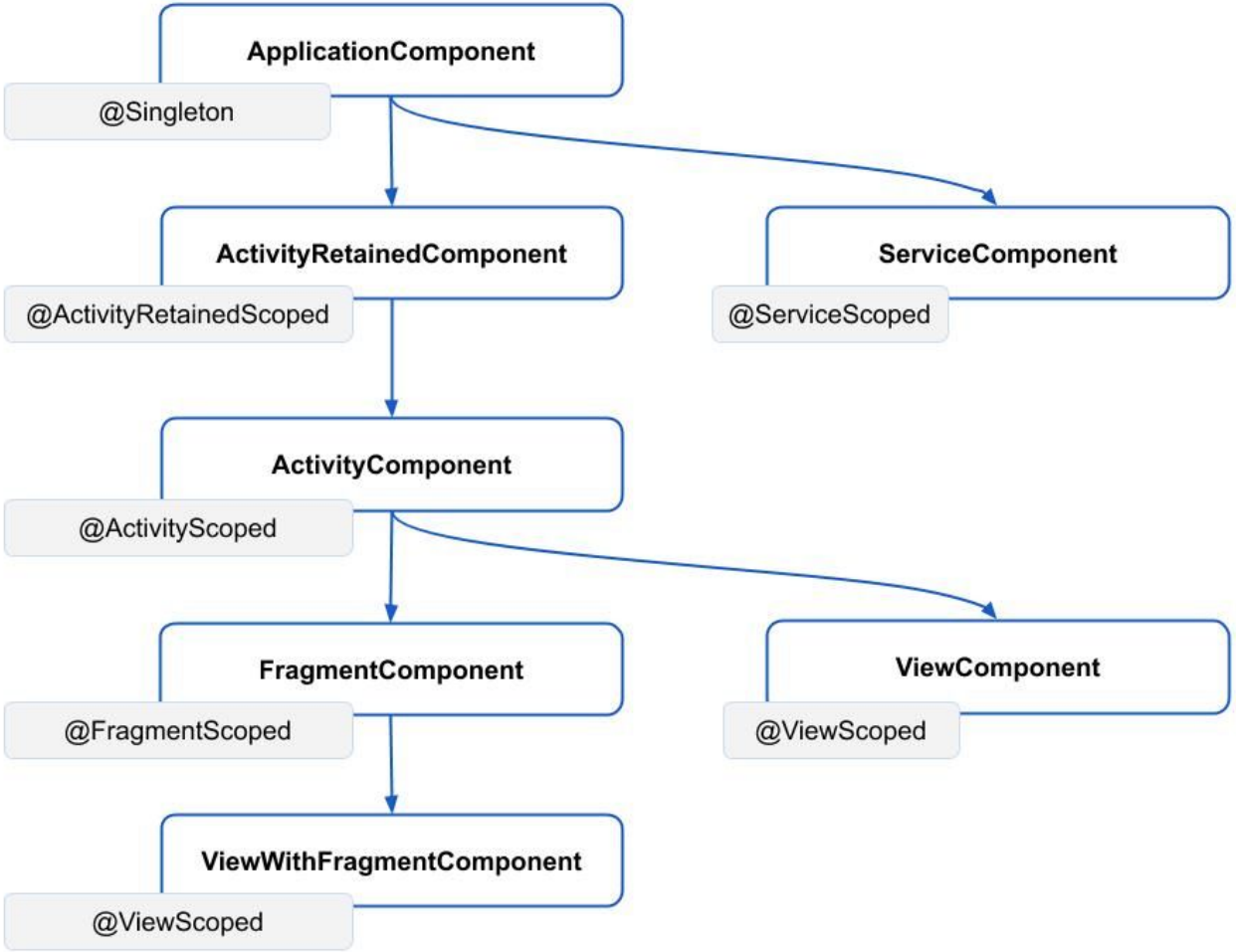
```
@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class AuthInterceptorOkHttpClient

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class OtherInterceptorOkHttpClient
```

Predefined qualifiers in Hilt

@ApplicationContext

@ActivityContext


ApplicationComponent
@Singleton
ActivityRetainedComponent
@ActivityRetainedScoped
ServiceComponent
@ServiceScoped
ActivityComponent
@ActivityScoped
FragmentComponent
@FragmentScoped
ViewComponent
@ViewScoped
ViewWithFragmentComponent
@ViewScoped

@EntryPoint

```
@EntryPoint
 @InstallIn(ApplicationComponent::class)
 interface ExampleContentProviderEntryPoint {
   fun analyticsService(): AnalyticsService
 }
```

```
val hiltEntryPoint = EntryPointAccessors.fromApplication(
    appContext,
    ExampleContentProviderEntryPoint::class.java
)
```

Hilt Annotations

@HiltAndroidApp

@AndroidEntryEnpoint

@InstallIn

@EntryPoint

Hilt and Jetpack integrations

```
class ExampleViewModel @ViewModelInject constructor(
 private val repository: ExampleRepository,
 @Assisted private val savedStateHandle: SavedStateHandle
) : ViewModel() {
 ...
}

@AndroidEntryPoint
class ExampleActivity : AppCompatActivity() {
 private val exampleViewModel: ExampleViewModel by viewModels()
 ...
}
```

```
class ExampleWorker @WorkerInject constructor(
 @Assisted appContext: Context,
 @Assisted workerParams: WorkerParameters,
 workerDependency: WorkerDependency
) : Worker(appContext, workerParams) { ... }

@HiltAndroidApp
class ExampleApplication : Application(), Configuration.Provider {

 @Inject lateinit var workerFactory: HiltWorkerFactory

 override fun getWorkManagerConfiguration() =
     Configuration.Builder()
           .setWorkerFactory(workerFactory)
           .build()
}
```

Testing

```kotlin
@HiltAndroidTest
class SettingsActivityTest {

 @get:Rule
 var hiltRule = HiltAndroidRule(this)

 @Inject
 lateinit var analyticsAdapter: AnalyticsAdapter

 @Before
 fun init() {
   hiltRule.inject()
 }

 @Test
 fun `happy path`() { ... }
}
```

```
@UninstallModules(AnalyticsModule::class)
@HiltAndroidTest
class SettingsActivityTest {

 @Module
 @InstallIn(ApplicationComponent::class)
 abstract class TestModule {

   @Singleton
   @Binds
   abstract fun bindAnalyticsService(
     fakeAnalyticsService: FakeAnalyticsService
   ): AnalyticsService
 }

 @BindValue @JvmField
 val analyticsService: AnalyticsService = FakeAnalyticsService()

}
```

```
@UninstallModules(AnalyticsModule::class)
@HiltAndroidTest
class SettingsActivityTest {

 @BindValue @JvmField
 val analyticsService: AnalyticsService = FakeAnalyticsService()

}
```

```
@CustomTestApplication(BaseApplication::class)
interface HiltTestApplication
```

```
class CustomTestRunner : AndroidJUnitRunner() {

    override fun newApplication(cl: ClassLoader?, name: String?, context: Context?):
Application {
        return super.newApplication(cl, HiltTestApplication::class.java.name, context)
    }
}
```

```
android {
    defaultConfig {
        testInstrumentationRunner "com.example.android.dagger.CustomTestRunner"
    }
}
```

Hilt Testing Annotations

@HiltAndroidTest

@UninstallModules

@BindValue

@CustomTestApplication

Pros

Android Official

Build on Dagger but easy to use

IDE support

Android lifecycle aware

Build Type award

Coexist with Dagger

Cheatsheet - Good docs - Codelabs

Cos

Alpha

Annotation processors and code generation could make build slower

Learn more

Dependency injection with Hilt

Dagger & Hilt documentation

Dagger & Hilt repository

Migrating your Dagger app to Hilt

Thank you!