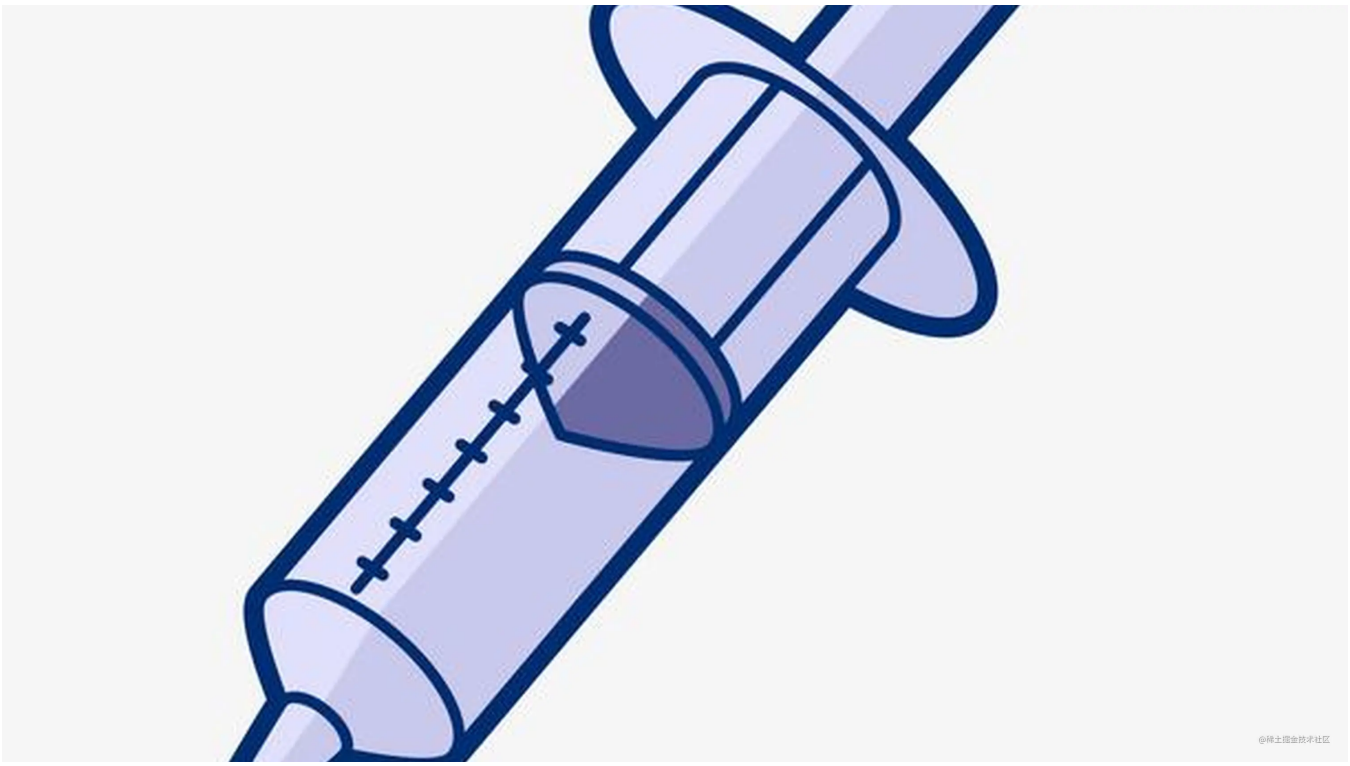
android hilt使用详解

卡布达学安卓 LV.4
2021年05月28日 09:36 · 阅读 3796

关注



参考

➤ developer.android.google.cn/training/de...

尤其要注意 studio4.2.1上是不能跑官网文档中的hilt版本的

我使用的依赖版本配置

1、根gradle依赖

groovy 复制代码

```
ext.kotlin_version = "1.5.0"//kotlin版本
ext.hilt_version = '2.35'//hilt版本
repositories {
```

```
    google()
    mavenCentral()
}
dependencies {
    classpath "com.android.tools.build:gradle:4.2.1"
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    classpath 'com.google.dagger:hilt-android-gradle-plugin:2.35'
}
```

2、app gradle依赖

```groovy 复制代码
implementation "com.google.dagger:hilt-android:$hilt_version"
kapt "com.google.dagger:hilt-android-compiler:$hilt_version"
```

3、所使用的插件

```groovy 复制代码
id 'kotlin-kapt'
id 'kotlin-android-extensions'
```

代码地址

➤ github.com/ymeddmn/Hil...

official分支：抄的官网的demo

onlysampletest:最简单的hilt使用

hilt相关的注解

@HiltAndroidApp：所有使用hilt的应用都需要使用这个注解，被使用在Application类上

@AndroidEntryPoint: Hilt可以为带有 `@AndroidEntryPoint` 注释的其他 Android 类提供依赖项，@AndroidEntryPoint可以被用在四大组件以及View上面

@Inject：获取依赖

@HiltAndroidApp使用举例

java 复制代码

```
@HiltAndroidApp
class App: Application() {

    override fun onCreate() {
        super.onCreate()
    }
}
```

@AndroidEntryPoint和@Inject使用举例

java 复制代码

```
@AndroidEntryPoint
public class ExampleActivity extends AppCompatActivity {

  @Inject
  AnalyticsAdapter analytics;
  ...
}
```

@Binds

module向外提供对象生成方法，修饰的方法必须是抽象的

@Provides

作用和@Binds一样，方法需要有实现，提供对象生成方法

hilt基本使用

必须配置选项

Application中添加注解

```kotlin 复制代码
@HiltAndroidApp
class App: Application() {
}
```

onlysampletest分支代码详解

这个分支实现的功能是在MainActivity中注入Car对象

1、首先加入必须配置选项

2、Car对象代码：

```kotlin 复制代码
class Car @Inject constructor() {
    fun drive(name: String) {
        println("老司机$name 在线开车")
    }
}
//Car的构造使用@Inject表示Car对象是一个可被注入的对象
```

3、MainActivity代码：

```kotlin 复制代码
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var car: Car//Car对象使用@Inject注解，会自动被实例化，这里必须使用lateinit延迟初始化才会
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        car.drive("mage")
    }
}
```

代码运行后日志输出：

2021-05-27 21:02:49.874 14104-14104/com.mage.hiltdaggerdemo I/System.out: 老司机mage 在线开车

complextest分支代码详解（更复杂一些的使用）

场景

onlysampletest分支中我们简单的将Car注入到了MainActivity中，但是实际的业务场景我们面临的业务肯定是远比这个负责的，比如我们现在想给Car分配一个司机。正常情况下我们会创建一个Driver司机对象，然后再Car对象的构造函数中传入Driver司机对象。

但是这明显不是hilt所想要的

hilt的场景实现

下面说一下hilt的实现方式：

首先我们需要创建多个类

App Car Driver DriverImpl DriverModule MainActivity

Car的构造中会传入Driver的实现，

Driver是司机的抽象实现，是一个接口。

DriverImple是Driver的实现类可以是任何一个具体的司机。

DriverModule负责提供Driver类的注入规则

下面是代码展示：

1、添加必须配置选项

2、Car实现

kotlin 复制代码

```kotlin
class Car @Inject constructor(val driver: Driver) {//Car的构造函数中增加了Driver司机对象的实现
    fun drive() {
        println("老司机${driver.name} 在线开车")
    }
}
```

2、Driver司机接口

```kotlin
interface Driver {
    val name :String
}
```

3、DriverImple司机接口的实现

```kotlin
class DriverImpl @Inject constructor() : Driver {//因为DriverImpl需要被注入到Car的构造中，所以Dr
    override val name: String
        get() = "mage"
}
```

4、DriverModule 配置Driver的注入规则

```kotlin
@Module//必须配置的注解，表示这个对象是Module的配置规则
@InstallIn(ActivityComponent::class)//表示这个module中的配置是用来注入到Activity中的
abstract class DriverModule {
    @Binds
    abstract fun bindDriver(driver: DriverImpl): Driver//形参中的DriverImple表示真实要注入Car构
}
```

5、MainActivity代码

```kotlin
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var car: Car//这里必须使用lateinit延迟初始化才会有效
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        car.drive()
    }
}
```

6、日志输出

2021-05-27 21:30:43.462 18874-18874/com.mage.hiltdaggerdemo I/System.out: 老司机mage 在线开车

provider分支代码

provider分支是对@Provides注解的解释

complextest中我们使用@Binds注解将我们自定义的对象共享出去，那么如果要共享的对象并非是我们自己创建的（例如OkhttpClient和Request）我们该如何处理呢，这个时候就可以使用@Provides了。代码示例如下：

1、添加必须配置选项

2、创建OkhttpModule

kotlin 复制代码

```kotlin
@Module
@InstallIn(ActivityComponent::class)
object OkhttpModule {
    @Provides
    fun provideOkhttpClient(
        // Potential dependencies of this type
    ): OkHttpClient {
        return OkHttpClient()
    }

    @Provides
    fun providdeRequest():Request{
        return Request.Builder()
            .get()
            .url("https://developer.android.google.cn/training/dependency-injection/hilt-an
            .build()
    }
}
```

3、MainActivity代码

kotlin 复制代码

```kotlin
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var client: OkHttpClient
    @Inject
    lateinit var request: Request
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_main)
        btn.setOnClickListener {
            client.newCall(request).enqueue(object :Callback{
                override fun onFailure(call: Call, e: IOException) {
                    println("请求失败")
                }
                override fun onResponse(call: Call, response: Response) {
                    println("请求成功")
                }
            })
        }
    }
}
```

4、日志输出

2021-05-28 07:36:25.477 14692-14775/com.mage.hiltdaggerdemo I/System.out: 请求成功

Hilt进阶使用

multipleobjtest分支使用（同一类型提供多实现）

有些场景，同一类型的对象我们可能会提供多个实例，这个时候就需要特殊处理。举例：

比如我们现在提供一个Car的接口抽象，然后同时给Car提供两个接口实现TruckCar和TaxtCar。在MainActivity中代码如下：

kotlin 复制代码

```
@Inject
lateinit var truck: Car

@Inject
lateinit var taxi: Car
```

因为注入的对象类型都是Car，所以编译过程中就无法区分我们到底想注入TruckCar还是TaxtCar，所以编译过程中代码就好报错。要解决这个问题就需要使用自定义注解，下面晒代码：

1、必须配置选项

2、Car、TruckCar、TaxiCar代码

kotlin 复制代码

```kotlin
interface Car {
    fun drive(name:String)
}
class TruckCar @Inject  constructor():Car{
    override fun drive(name: String) {
        println("$name 卡车老司机开车，呜呜")
    }

}
class TaxiCar @Inject  constructor():Car{
    override fun drive(name: String) {
        println("$name 出租车老司机开车，呜呜")
    }

}
```

3、CarModule代码

kotlin 复制代码

```kotlin
@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class Truck//卡车类注入标记

@Qualifier
@Retention(AnnotationRetention.BINARY)
annotation class Taxi//出租车类注入标记

@Module
@InstallIn(ActivityComponent::class)
class CarModule {

    @Truck//卡车类生成规则加上这个注解，注入的时候也是用该注解编译器就可以知道我们真实想注入的类型是TruckC
    @Provides
    fun bindTruckCar(truckCar: TruckCar): Car {
        return TruckCar()
    }

    @Taxi//出租车类生成规则加上这个注解，注入的时候也是用该注解编译器就可以知道我们真实想注入的类型是TaxiCa
    @Provides
    fun bindTaxtCar(taxiCar: TaxiCar): Car {
        return TaxiCar()
    }
}
```

4、日志输出

2021-05-28 08:12:28.780 20418-20418/com.mage.hiltdaggerdemo I/System.out: mage 卡车老司机开车，呜呜 2021-05-28 08:12:28.780 20418-20418/com.mage.hiltdaggerdemo I/System.out: mage 出租车老司机开车，呜呜

预定义限定符(@ApplicationContext @ActivityContext)

抄一下官方的解释，Hilt 提供了一些预定义的限定符。例如，由于您可能需要来自应用或 Activity 的 `Context` 类，因此 Hilt 提供了 `@ApplicationContext` 和 `@ActivityContext` 限定符。

代码示例：

1、必须配置选项

2、BaseInfo类

kotlin 复制代码

```kotlin
class BaseInfo @Inject constructor(@ActivityContext private val context: Context) {
    fun printPackageName(){
        println("应用包名 ${context.packageName}")
    }
}
```

3、MainActivity类

kotlin 复制代码

```kotlin
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var baseInfo: BaseInfo
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        btn.setOnClickListener {
            baseInfo.printPackageName()
        }
    }
}
```

4、日志输出

2021-05-28 08:20:56.292 21928-21928/com.mage.hiltdaggerdemo I/System.out: 应用包名 com.mage.hiltdaggerdemo

官方提供的类生成组件（抄官方）

对于您可以从中执行字段注入的每个 Android 类，都有一个关联的 Hilt 组件，您可以在 `@InstallIn` 注释中引用该组件。每个 Hilt 组件负责将其绑定注入相应的 Android 类。

所有组件如下，组件的生命周期和是依赖于注入组件类的生命周期的

ApplicationComponent	**Application**	
ActivityRetainedComponent	ViewModel	作用于ViewModel
ActivityComponent	Activity	
FragmentComponent	Fragment	
ViewComponent	View	
ViewWithFragmentComponent	带有 @WithFragmentBindings 注释的 View	
ServiceComponent	Service	

组件作用域（抄官方）

默认情况下，Hilt 中的所有绑定都未限定作用域。这意味着，每当应用请求绑定时，Hilt 都会创建所需类型的一个新实例。

不过，Hilt 也允许将绑定的作用域限定为特定组件。Hilt 只为绑定作用域限定到的组件的每个实例创建一次限定作用域的绑定，对该绑定的所有请求共享同一实例。

比如当组件作用域为@Singleton的时候，这个组件就是一个全局单例的

比如当组件作用域为@ActivityScoped的时候，这个组件在同一个Activity内都是同一个对象实例

所以默认的组件和作用域

Android 类	生成的组件	作用域
`Application`	`ApplicationComponent`	`@Singleton`
`View Model`	`ActivityRetainedComponent`	`@ActivityRetainedScope`
`Activity`	`ActivityComponent`	`@ActivityScoped`
`Fragment`	`FragmentComponent`	`@FragmentScoped`
`View`	`ViewComponent`	`@ViewScoped`
带有 `@WithFragmentBindings` 注释的 `View`	`ViewWithFragmentComponent`	`@ViewScoped`
`Service`	`ServiceComponent`	`@ServiceScoped`

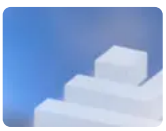
组件默认绑定（抄官方）

每个 Hilt 组件都附带一组默认绑定，Hilt 可以将其作为依赖项注入您自己的自定义绑定。请注意，这些绑定对应于常规 Activity 和 Fragment 类型，而不对应于任何特定子类。这是因为，Hilt 会使用单个 Activity 组件定义来注入所有 Activity。每个 Activity 都有此组件的不同实例。

Android 组件	默认绑定
ApplicationComponent	Application
ActivityRetainedComponent	Application
ActivityComponent	Application 和 Activity
FragmentComponent	Application、Activity 和 Fragment
ViewComponent	Application、Activity 和 View
ViewWithFragmentComponent	Application、Activity、Fragment 和 View
ServiceComponent	Application 和 Service

分类： Android 标签： Android

文章被收录于专栏：



android

android相关知识

关注专栏