dagger2 使用教程第一节

九风特 关注
0.101 2020.09.15 18:34:46 字数 1,947 阅读 750

我翻译了官方文档，说实话，那个文档只是介绍了很多概念，第一次了解dagger2的兄弟，看完了可能更迷了。这没关系，也正常。
下面我们来一步一步的学习它，既然要使用它当做我们架构的材料，那就要真正的了解它，不过凡事都要循序渐进，那么我们就从最简单的开始吧

为什么要使用dagger2
关于这方面可以参考官方文档的自述, 使用dagger2的目的就是为了实现依赖注入，dagger2很好的实现了依赖注入，那么只要了解什么是依赖注入，以及它的好处，就能明白为什么要使用dagger2了。先来了解什么是依赖，在我们程序设计中，类于类之间肯定是存在各种依赖关系的，这是无法避免的。在现实中更是如此，比如每个公民都有一张身份证，那么公民和身份证就存在依赖关系。（依赖可以理解为2者相互有联系，并不是字面意义的依赖）。现在再来看看按照传统的编程方式，这种依赖会带来什么问题，我们用个简单的实例来进行阐述。(我写的实例全都是使用的Android studio+kotlin, 也很容易翻译成java,我们现在完全不考虑界面布局美观度等等)
现在我们要做这样一个程序：程序中有2个按钮 点击按钮1显示小明的信息，点击按钮2显示小红的信息，我们拿到这个需求后感觉，设计两个类：Person类和IdCard类。为了简单我们只做很少的字段和方法

```
class IdCard(val number:Long, val name:String)
{
}
class Person(val company:String, val idCard:IdCard)
{
    override fun toString(): String {
        return "姓名:${idCard.name}, 公司:$company, 身份证号:${idCard.number}"
    }
}
```

这真是两个再简单不过类了，然后写一下activity的逻辑

```
class MainActivity : AppCompatActivity() {
    val personXiaoMing = Person("首钢", IdCard(1000, "小明"))
    val personXiaoHong = Person("北汽", IdCard(2000, "小红"))
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        buttonXiaohong.setOnClickListener {
            textViewInfo.text = personXiaoHong.toString()
        }
        buttonXiaoMing.setOnClickListener {
            textViewInfo.text = personXiaoMing.toString()
        }
    }
}
```

最终得到这样的效果:

first.png

点击不同的按钮，会显示不同的信息
这看起来没有任何问题。 但是现在我们发现我们提供的身份证信息太少了，需要增加地址和年龄字段，那么我们只好把现在的IdCard类这样修改

```
class IdCard(val number:Long, val name:String, val address:String, val age:Int)
{
}
```

在修正一下Person为了输出函数

```
class Person(val company:String, val idCard:IdCard)
{
    override fun toString(): String {
        return "姓名:${idCard.name}, " +
                "公司:$company, " +
                "身份证号:${idCard.number}, " +
                "地址:${idCard.address}, " +
                "年龄:${idCard.age}"
    }
}
```

此时编译会编译不过，因为我们忘记修改创建两个Person类实例的Activity,好吧 我们来修改他们（是不是感觉有点烦了呢?还好是两个实例...）

修改这两行:

```
    val personXiaoMing = Person("首钢", IdCard(1000, "小明", "北京望京南", 28))
    val personXiaoHong = Person("北汽", IdCard(2000, "小红", "南京新街口", 18))
```

现在运行程序，一切达到了预期，输出了更多信息

对实例的思考

现在我们应该已经感觉到了点什么，我改IdCard的类会关联到Person类 还得进一步关联修改activity类，这种设计模式在更复杂的用例中，会更加糟糕。其实类似的事情，我想稍微老点的程序员都干过。由于更改了某个模块，不得不大幅度修改整个工程代码，改完了感觉整个工程都有危险，都得要重新测试的感觉。 那么这种事情就是我们说的依赖，我们说这种依赖关系破坏了模块的独立性，彻底的动一发牵全身。这是我们不愿意看到的。 那么用什么方法解决这个问题呢，答案是:依赖注入。 所谓的依赖注入就是当我需要A的时候，我不是自己去制造，而是委托第三方去制造，而且我不关心第三方是怎么制造A的。 这也符合现实，毕竟我们用的大多数东西都是委托别人制造的，我们只需要使用就可以了，比如手机。 那么这个第三方是谁呢，没错，他就是我们接下来的主角 dagger2.(在没有这东西的时候，我们可能会制造一些干这活的类，类似XXXFactory, xxxWrapper等等,dagger2 说白了就是帮你干这事儿的)。

首次使用Dagger

类似Dagger这种第三方模块，首先要考虑的是如何把它引入到你工程中，让它可用
我的开发工具是Android studio 语言是kotlin,就目前来说dagger2的最新版本是2.28.3
那么按照一般引入模块的规范，我们在build.gradle(project)下定义版本号

```
buildscript {
    ext{
        kotlin_version = "1.4.0"
        dagger_version = "2.28.3"//dragger2 版本号
    }
    ...
 }
```

接下来在build.gradle(app)下添加

```
apply plugin: 'kotlin-kapt'
```

添加依赖:

```
    //dagger2 support begin
    implementation "com.google.dagger:dagger-android:$dagger_version"
    implementation "com.google.dagger:dagger-android-support:$dagger_version" // if yo
    //annotationProcessor "com.google.dagger:dagger-android-processor:$dagger_version".
    //annotationProcessor "com.google.dagger:dagger-compiler:$dagger_version"//java us
    kapt "com.google.dagger:dagger-android-processor:$dagger_version"//kotlin use
    kapt "com.google.dagger:dagger-compiler:$dagger_version"//kotlin use
    //dagger2 support end
```

这样 准备工作就做完了，我们说了循序渐进，那么就用一个最简单的实例来完成本节的说明
我们现在要写一个User类，在主activity中放一个按钮，点击按钮后把User信息显示出来。这仅仅是让你了解什么是注入，这应该是最最简单的情况了，旨在通过它体验到什么是注入！！！

常规方式设计

User类定义

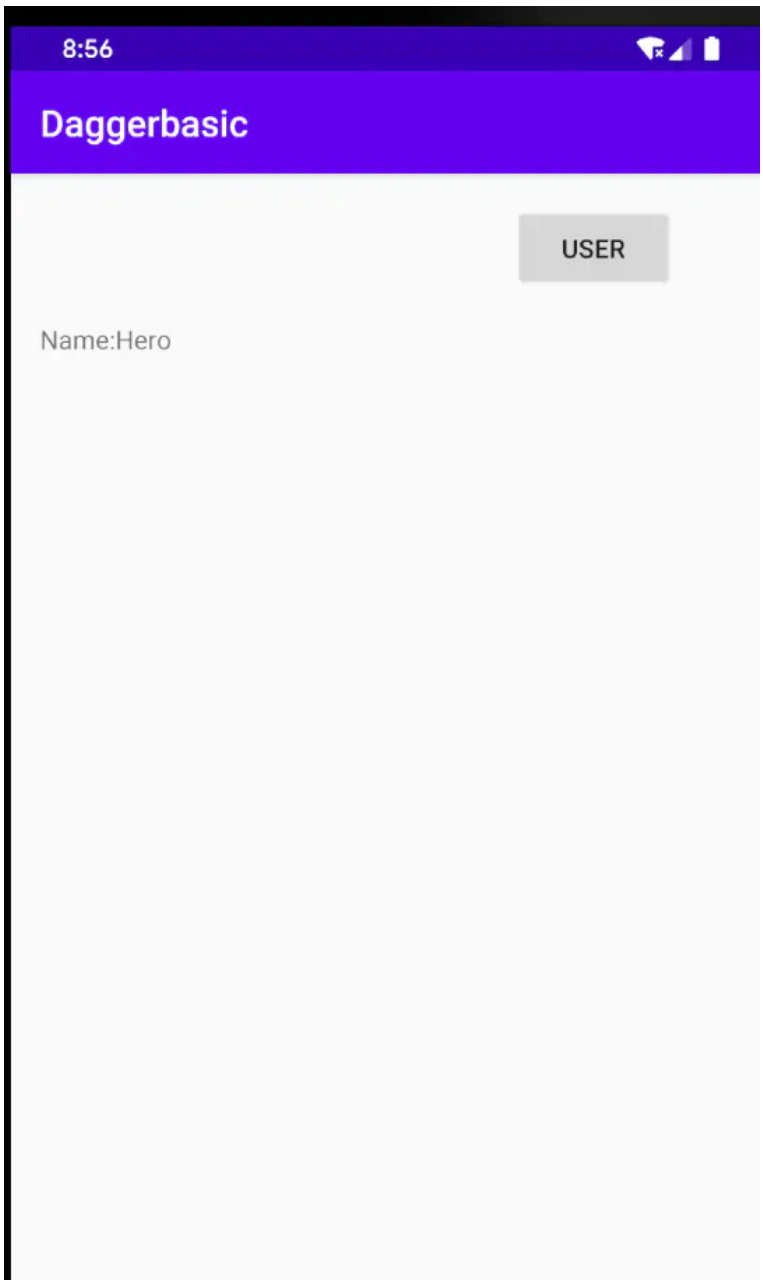
```
class User constructor()//constructor本来可以省略，但为了后面的说明，我们添加上它
{
    var name:String?=null
    override fun toString(): String {
        return "Name:$name"
```

```
    }
}
```

MainActivity:

```
class MainActivity : AppCompatActivity() {
    lateinit var user:User
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        user = User()//创建User类
        user.name="Hero"
        buttonUser.setOnClickListener {
            textViewInfo.text = user.toString()
        }
    }
}
```

运行效果:



sec.png

那么现在我们试图让MainActivity和user尽量脱离依赖

首先我们改造User类

@Inject登场

```
class User @Inject constructor()
{
    var name:String?=null
    override fun toString(): String {
        return "Name:$name"
    }
}
```

现在的User类被@Inject标记了，而且标记的是构造函数. 这会让dagger知道怎么创建User实例。

User :Hi, dagger,我告诉你怎么能够新建一个我的实例。请看我的构造函数！！！

dagger：没问题，我知道了，我会记住你的构造方式.

(由于是第一节，所以我们先不深究，dagger是怎么知道这件事的，它又是怎么记住的，我们会循序渐进，下同).

接下来的问题是如何让dagger2帮我们创建User实例，进而让我们的MainActivity不在管创建这件事儿。

首先通过@Inject标记MainActivity的user变量，这就相当于告诉dagger, 我这个user变量是需要你帮我创建的. 那么dagger2会为这个user变量实现一个注入器的方法，通过调用该方法则能初始化user变量，但相当于是后台实现的，你不能自己去调用这个注入器方法，否则和直接new没区别。 那接下来就该@Component登场了,通过它来调用这个注入器方法就对了,我们先来添加它，然后在解释

```
@Component
public interface MainComponent{
    fun inject(activity: MainActivity)
}
```

dagger2会通过这个被注解了的接口生成一个类DaggerMainComponent，并在inject方法中调用我们前边说的注入器方法，从而完成对user的创造。现在我不太想贴这些自动生成的代码，这和本节入门的初衷不符。只要知道怎么做就可以了。

那么现在我们再来改写MainActivity的代码

```
class MainActivity : AppCompatActivity() {
    @Inject
    lateinit var user:User
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        DaggerMainComponent.builder().build().inject(this)
        user.name="Hero"
        buttonUser.setOnClickListener {
            textViewInfo.text = user.toString()
        }
    }
}
```

现在你发现什么了吗？ 我们的MainActivity 不在需要调用User的构造函数了，它也不需要清楚User是怎么构造的了，这些都交给了Dagger2 具体代码是这行

```
DaggerMainComponent.builder().build().inject(this)
```

仅仅就这个实例来说，这样做确实有点得不偿失，但从解耦和结构设计的角度来讲，即便如此简单的实例，这样做也更优秀，不是吗？ 以后你User构造再怎么变MainActivity 都不用改了，各管各的事情，这很好！！！

如果你想继续学习dagger的话，请把此节的内容 都上机真正的实验下吧。