dagger2 使用教程第三节

九风特 关注
2020.09.16 17:54:20 字数 614 阅读 191

第三节引言
本节我们来解决上一节遗留下来的问题，如何用dagger构建有参数的User

构建有参数的User

这个问题的核心是，Dagger在哪里去配置构造参数，类似这种需求引出了@Provides和@Module,首先来看看官方文档对它们的阐述
对于@Inject不足或不合适的情况，可以使用 `@Provides` 注释的方法来满足依赖关系。方法的返回类型定义了它满足的依赖项。
这说明@Provides使用来修饰一个方法的，而该方法应该返回我们需要构建的实例，对于现在则是应该返回User
那么我们第一步给User添加一个age字段，并放在构造函数中

```
class User @Inject constructor(var age:Int)
{
    lateinit var name:String
    override fun toString(): String {
        return "Name:$name, Age:$age"
    }
}
```

接下来试着写一个Provides

```
@Provides
fun provideUser():User
{
    return User(28)
}
```

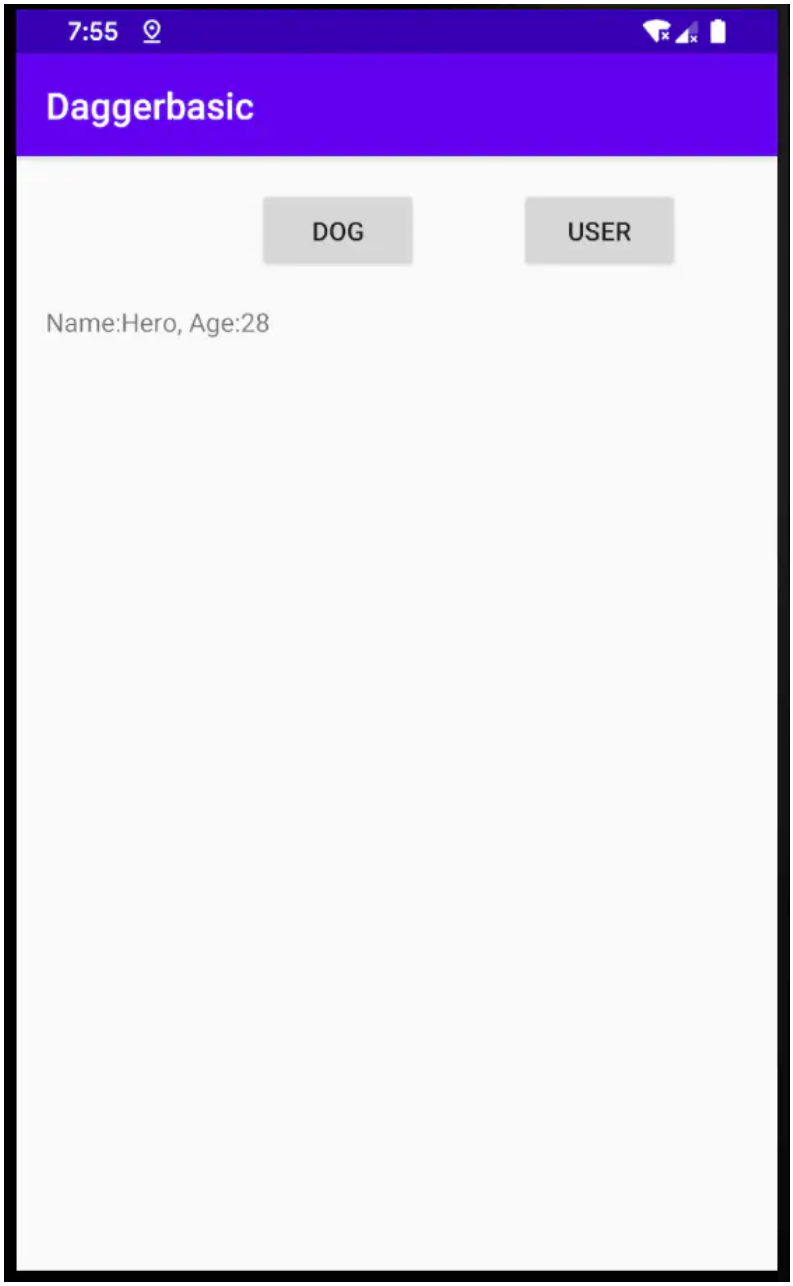
是不是感觉光秃秃的，provideUser这方法缺少归属感。没错，@Provides修饰的方法，按照约定以provide前缀开头，并且该方法应该属于一个用@Module修饰的类或接口,那么我们来引入@Module

```
@Module
class UserModule
{
    @Provides
    fun provideUser():User
    {
        return User(28)
    }
}
```

在修改@Component来和Module联系起来

```
@Component(modules = [UserModule::class])
interface MainComponent{
    fun inject(activity: MainActivity)
}
```

现在运行程序，发现我们可以正确显示出用户的年龄信息了。 跟踪的话，会发现dagger最终是调用provideUser()方法来构造user字段的。



dog2.png

现在我们基本了解了有构造参数的类怎么进行注入,对了我们原来的dog不是也有个owner的User实例吗，它现在什么情况，稍微修改下Dog的输出函数，我们发现dog的User实例也通过provideUser()被提供了。当然年龄也是28。 进一步调查，发现虽然两个User实例虽然年龄都是28 但不是同一个实例。 现在问题来了，如果我要求在MainActivity和dog实例中的User实例是 同一个（单例User）怎么办呢?
解决办法是使用作用域限定注解:@Singleton

```
@Module
class UserModule{
    @Singleton
    @Provides
    fun provideUser():User
    {
        return User(28)
    }
}
@Singleton
@Component(modules = [UserModule::class])
interface MainComponent{
    fun inject(activity: MainActivity)
}
```

现在我们通过调查，发现两个User完全是同一个了，这就是我们说的单例。 看来Dagger的一个很好的用处就是，可以帮我们创建单例。

现在可能有人会说，我不希望单例，但希望MainActivity的user是28岁， Dog的owner是38岁。 下一节我们继续