dagger2 使用教程第二节

九风特 关注
2020.09.16 12:58:25 字数 1,552 阅读 276

第二节引言

在上第一节中我们了解了如何引入Dagger，什么是依赖注入，对Dagger有了一个初步的认识。

总结@Inject, @Compoent

上一节我们使用了这两个字段，并进行了说明，现在我们先来对@Inject进行一下总结

- 使用@Inject注释构造函数。当请求一个新实例时，Dagger将获得所需的参数值并调用这个构造函数

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

- Dagger可以直接注入字段。在本例中，Dagger为user字段获得一个User实例。

```
class MainActivity : AppCompatActivity() {
    @Inject
    lateinit var user:User
    ...
   }
```

以上两条要理解透彻，还有几条知道即可吧，因为我没太想到什么时候能有用。。。

- 如果您的类有@Inject注解的字段，但没有@Inject注解的构造函数，那么Dagger会在被请求时注入这些字段，但不会创建新的实例。添加一个带@Inject注释的无参数构造函数，Dagger才会创建实例。
- Dagger也支持方法注入，不过构造函数或字段注入通常是首选。
- 缺少@Inject注解的类不能由Dagger构造。

这其实是官方文档的总结，但如果不结合实例很难理解它们

你可能已经注意到了，我们的User类没有参数，这个可以先不管。后边会进行说明。
接下来我们来具体看看@Component吧，相信通过上一节的介绍，对它算是有个印象了，那它到底是个什么呢?
@Component相当于是Dagger的调度中心，我们先来看看上一节用@Component注解的接口

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

这段代码会让Dagger自动生成一个类，我还是贴一下吧

```
public final class DaggerMainComponent implements MainComponent {
  private DaggerMainComponent() {

  }
```

```
  public static Builder builder() {
    return new Builder();
  }

  public static MainComponent create() {
    return new Builder().build();
  }

  @Override
  public void inject(MainActivity activity) {
    injectMainActivity(activity);
  }

  private MainActivity injectMainActivity(MainActivity instance) {
    MainActivity_MembersInjector.injectUser(instance, new User());
    return instance;
  }

  public static final class Builder {
    private Builder() {
    }

    public MainComponent build() {
      return new DaggerMainComponent();
    }
  }
}
```

本来不想分析源码，但发现不分析，光说结果不太清晰。。。
那么在来结合添加@Inject后生成的类

```
public final class MainActivity_MembersInjector implements MembersInjector<MainActivit
  private final Provider<User> userProvider;

  public MainActivity_MembersInjector(Provider<User> userProvider) {
    this.userProvider = userProvider;
  }

  public static MembersInjector<MainActivity> create(Provider<User> userProvider) {
    return new MainActivity_MembersInjector(userProvider);
  }

  @Override
  public void injectMembers(MainActivity instance) {
    injectUser(instance, userProvider.get());
  }

  @InjectedFieldSignature("com.study.daggerbasic.MainActivity.user")
  public static void injectUser(MainActivity instance, User user) {
    instance.user = user;
  }
}
```

我们在MainActivity_MembersInjector看到两个方法

```
public void injectMembers(MainActivity instance)
@InjectedFieldSignature("com.study.daggerbasic.MainActivity.user")
  public static void injectUser(MainActivity instance, User user) {
    instance.user = user;
  }


```

记住@Component是用来注释接口的，从而生成一个Dagger开头的生成类，本例生成是：DaggerMainComponent。
被注释的接口要满足一个条件

- **@Component注释的接口或抽象类，必须至少包含一个抽象组件方法。组件方法可以有任何名称，但是必须具有符合members-injection注入契约的签名**

什么是members-injection注入契约的签名？ 没错，就是它！！！

```
public void injectMembers(MainActivity instance)
```

现在就解释了 我们为什么要在我们的接口中写这样的接口了

```
@Component
interface MainComponent{
    fun inject(activity: MainActivity)//符合members-injection注入契约的签名
}
```

你现在应该能举一反三，理解为何调用这个inject接口就能给user变量赋值了吧。(调试下，看看各个接口调用顺序就明白了)
(请注意这是一个连贯的教程，您必须从头看，并在机器上有真正的工程才能很好的理解。如果用的是java，翻译下代码就好了，代码很简单)
现在你可以把inject接口改个名字试试，来验证总结的正确性（当然需要clean,rebuild)

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

一切okay，看来总结的没问题（毕竟我这个总结就是@Component注释内容的翻译 :D)。
好吧，别高兴的太早，我们之前提出的问题还没解决呢，毕竟大多数时候类都是有参数的，哪有那么多无参构造类让您老@Inject呢？ 既然这么麻烦，我不干程序员了，爱咋地咋地。本来想这么说，但想想自己除了敲代码好像啥都不会了，那还是继续吧。。。

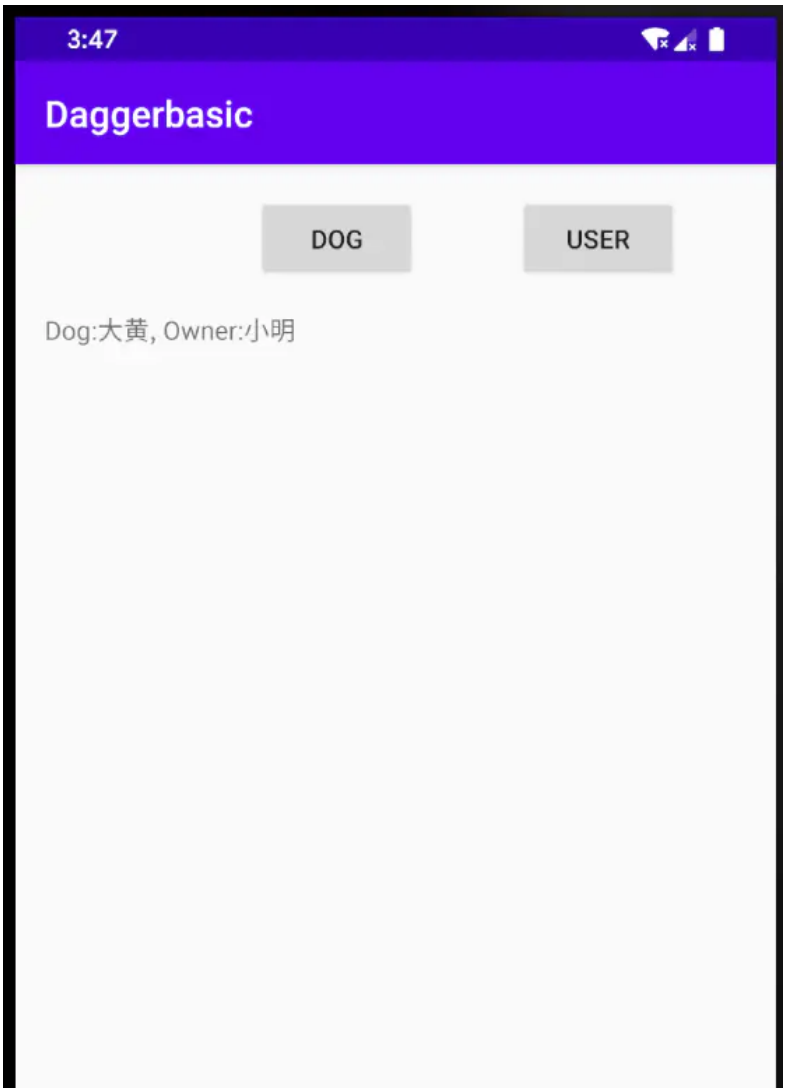
先来看看官方自己对@Inject的吐槽吧:
Inject并不是在任何地方都有效:

- 接口不能被构造。
- 第三方类不能被注解（annotated）。
- 可配置对象必须被配置!(Configurable objects must be configured!)

前两条好理解，我们来重点说说第三条:
现在我们修改User类,添加一个age字段到构造函数

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

此时我们编译工程，发现编译不过。这也是显然的事情，没任何人配置了age字段。编译过才奇怪。
这就是所谓的“可配置对象必须被配置”，我们一会在来说如何改正这个错误。我们先来看看如果构造函数的参数也是被注入过的，又会如何呢，我们把User类改回最初的模样,然后在添加一个新的测试类叫做Dog,界面上在添加一个按钮显示Dog的名字以及其主人的名字



dog.png

先来写Dog类，并修改@Componet修饰的接口MainComponent。通过之前的学习，你应该能知道Dog的owner字段是如何被创建的！！！

```
class Dog @Inject constructor(var owner:User)
{
    lateinit var name: String
    override fun toString(): String {
        return "Dog:$name, Owner:${owner.name}"
    }
}
@Component
interface MainComponent{
    fun inject(activity: MainActivity)
}
```

在来贴一下MainActivity的代码,还是很简单

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

```
        }
        buttonDog.setOnClickListener {
            textViewInfo.text = dog.toString()
        }
    }
}
```

此时运行，一切okay。此时你观察生成类的注入签名方法，会发现Dagger2已经发现Activity的dog里面有个owner他会自动帮我们创建实例

```
  @Override
  public void injectMembers(MainActivity instance) {
    injectUser(instance, userProvider.get());
    injectDog(instance, dogProvider.get());
  }
  @InjectedFieldSignature("com.study.daggerbasic.Dog.owner")
  public static void injectOwner(Dog instance, User owner) {
    instance.owner = owner;
  }
```

这进一步阐述了"可配置对象必须被配置", 你在dog内配置了owner因为owner本身是可注入的, 一切都是通过注入完成的！如果你看完这个教程，在去看官方文档，会轻松很多，不然看到这句话会很蒙圈的。
解释一句话还真是，可能我有强迫症，生怕我说的不清楚吧。当然要特别特别注意，这都是我的理解，如果有问题，请您指正，我会非常感谢您的。毕竟我刚看到这句话时，很蒙圈。
现在回到最初的问题， 如果User有参数，而且参数还是不可注入的，那咋办？dagger2说，这我考虑到了，但你这篇文章篇幅过长了吧，不如下节再说吧。
我说:好的，我们下节见!