Dagger 2 完全解析（三），Component 的组织关系与 SubComponent

最帅的 JohnnyShieh 2017-08-04 19:23

阅读本文大概需要 8 分钟。

理解前面两篇文章后，可以使用 Dagger 2 框架完成一个对象的依赖注入。但是在实际项目中有多个需要注入依赖的对象，也就是说会有多个 Component，它们之间会有相同的依赖，那么该如何处理它们之间的关系呢?

1. 前言

先看下面一个场景：

```
public class Man {
    @Inject
    Car car;

    public void goWork() {
        ...
        car.go();
        ...
    }
}
```

```
public class Friend {
    @Inject
    Car car;    // 车是向 Man 借的

    public void goSightseeing() {
        ...
        car.go();
        ...
    }
}
```

Man 有一辆车，Friend 没有车，但是他可以借 Man 的车出去游玩下，提供 Car 实例的 `CarModule` 不变，那么我们应该怎么设计 Component 呢?

很多人第一时间会想到下面这种设计：

```
@ManScope
@Component(modules = CarModule.class)
public interface ManComponent {
    ...
}

@FriendScope
@Component(modules = CarModule.class)
public interface FriendComponent {
    ...
}
// @ManScope 和 @FriendScope 都是自定义的作用域
```

这种做法是最简单的，把 Component 需要的依赖都在 `modules` 属性中声明，但是这样有两个问题：

（1）有时依赖实例需要共享，例如上面场景中，Friend 的 car 是向 Man 借的，所以 `FriendComponent` 应该使用 `ManComponent` 中的 car 实例。

（2）Scope 作用域容易失效，例如 CarModule 的 `provideCar()` 使用 @Singleton 作用域， `FriendComponent` 和 `ManComponent` 也要用 Singleton 标注，但它们都会持有一个 car 实例。

所以 FriendComponent 需要依赖 ManComponent 提供的 car 实例，这就是 Component 组织关系中的一种。

2. Component 的组织关系

Component 管理着依赖实例，根据依赖实例之间的关系就能确定 Component 的关系。这些关系可以用 `object graph` 描述，我称之为依赖关系图。在 Dagger 2 中 Component 的组织关系分为两种：

- 依赖关系，一个 Component 依赖其他 Compoent 公开的依赖实例，用 Component 中的 `dependencies` 声明。
- 继承关系，一个 Component 继承（也可以叫扩展）某 Component 提供更多的依赖，SubComponent 就是继承关系的体现。

所以前文中场景 `FriendComponent` 和 `ManComponent` 是依赖关系。

2.1 依赖关系

Friend 与 Man 场景中的依赖关系图：

具体的实现代码：

```
@ManScope
@Component(modules = CarModule.class)
public interface ManComponent {
    void inject(Man man);

    Car car();  //必须向外提供 car 依赖实例的接口，表明 Man 可以借 car 给别人
}

@FriendScope
@Component(dependencies = ManComponent.class)
public interface FriendComponent {
    void inject(Friend friend);
}
```

注：因为 FriendComponent 和 ManComponent 是依赖关系，如果其中一个声明了作用域的话，另外一个也必须声明，而且它们的 Scope 不能相同，ManComponent 的生命周期 >= FriendComponent 的。FriendComponent 的 Scope 不能是 @Singleton，因为 Dagger 2 中 @Singleton 的 Component 不能依赖其他的 Component。

编译时生成的代码中 DaggerFriendComponent 的 `Provider<Car>` 实现中会用到 `manComponent.car()` 来提供 car 实例，如果 ManComponent 没有向外提供 car 实例的接口的话，DaggerFriendComponent 就会注入失败。

依赖注入：

```
ManComponent manComponent = DaggerManComponent.builder()
    .build();

FriendComponent friendComponent = DaggerFriendComponent.builder()
    .manComponent(manComponent)
    .build();
friendComponent.inject(friend);
```

依赖关系就跟生活中的朋友关系相当，注意事项如下：

1. 被依赖的 Component 需要把暴露的依赖实例用显式的接口声明，如上面的 `Car car()`，我们只能使用朋友愿意分享的东西。
2. 依赖关系中的 Component 的 Scope 不能相同，因为它们的生命周期不同。

2.1 继承关系

继承关系跟面向对象中的继承的概念有点像，SubComponent 称为子 Component，类似于平常说的子类。下面先看看下面这个场景：

```
public class Man {
    @Inject
    Car car;
    ...
}

public class Son {
    @Inject
    Car car;

    @Inject
    Bike bike;
}
```

Son 可以开他爸爸 Man 的车 car，也可以骑自己的自行车 bike。依赖关系图：

上图中 SonComponent 在 ManComponent 之中，SonComponent 子承父业，可以访问 parent Component 的依赖，而 ManComponent 只知道 SonComponent 是它的 child Component，可以访问 SubComponent.Builder，却无法访问 SubComponent 中的依赖。

```
@ManScope
@Component(modules = CarModule.class)
public interface ManComponent {
    void inject(Man man);    // 继承关系中不用显式地提供暴露依赖实例的接口
}
```

```
@SonScope
@SubComponent(modules = BikeModule.class)
public interface SonComponent {
    void inject(Son son);

    @Subcomponent.Builder
    interface Builder { // SubComponent 必须显式地声明 Subcomponent.Builder
        SonComponent build();
    }
}
```

`@SubComponent` 的写法与 `@Component` 一样，只能标注接口或抽象类，与依赖关系一样，SubComponent 与 parent Component 的 Scope 不能相同，只是 SubComponent 表明它是继承扩展某 Component 的。怎么表明一个 SubComponent 是属于哪个 parent Component 的呢？只需要在 parent Component 依赖的 Module 中的 `subcomponents` 加上 SubComponent 的 class，然后就可以在 parent Component 中请求 SubComponent.Builder。

```
@Module(subcomponents = SonComponent.class)
public class CarModule {
    @Provides
    @ManScope
    static Car provideCar() {
        return new Car();
    }
}

@ManScope
@Component(modules = CarModule.class)
public interface ManComponent {
    void injectMan(Man man);

    SonComponent.Builder sonComponent();    // 用来创建 Subcomponent
}
```

SubComponent 编译时不会生成 DaggerXXComponent，需要通过 parent Component 的获取 SubComponent.Builder 方法获取 SubComponent 实例。

```
ManComponent manComponent = DaggerManComponent.builder()
    .build();

SonComponent sonComponent = manComponent.sonComponent()
    .build();
sonComponent.inject(son);
```

继承关系和依赖关系最大的区别就是：继承关系中不用显式地提供依赖实例的接口，SubComponent 继承 parent Component 的所有依赖。

2.3 依赖关系 vs 继承关系

相同点：

- 两者都能复用其他 Component 的依赖
- 有依赖关系和继承关系的 Component 不能有相同的 Scope

区别：

- 依赖关系中被依赖的 Component 必须显式地提供公开依赖实例的接口，而 SubComponent 默认继承 parent Component 的依赖。
- 依赖关系会生成两个独立的 DaggerXXComponent 类，而 SubComponent 不会生成独立的 DaggerXXComponent 类。

在 Android 开发中，Activity 是 App 运行中组件，Fragment 又是 Activity 一部分，这种组件化思想适合继承关系，所以在 Android 中一般使用 SubComponent。

3. SubComponent 的其他问题

3.1 抽象工厂方法定义继承关系

除了使用 Module 的 `subcomponents` 属性定义继承关系，还可以在 parent Component 中声明返回 SubComponent 的抽象工厂方法来定义：

```
@ManScope
@Component(modules = CarModule.class)
public interface ManComponent {
    void injectMan(Man man);

    SonComponent sonComponent();     // 这个抽象工厂方法表明 SonComponent 继承
}
```

这种定义方式不能很明显地表明继承关系，一般推荐使用 Module 的 `subcomponents` 属性定义。

3.2 重复的 Module

当相同的 Module 注入到 parent Component 和它的 SubComponent 中时，则每个 Component 都将自动使用这个 Module 的同一实例。也就是如果在

SubComponent.Builder 中调用相同的 Module 或者在返回 SubComponent 的抽象工厂方法中以重复 Module 作为参数时，会出现错误。（前者在编译时不能检测出，是运行时错误）

```
@Component(modules = {RepeatedModule.class, ...})
interface ComponentOne {
    ComponentTwo componentTwo(RepeatedModule repeatedModule); // 编译时报
    ComponentThree.Builder componentThreeBuilder();
}

@Subcomponent(modules = {RepeatedModule.class, ...})
interface ComponentTwo { ... }

@Subcomponent(modules = {RepeatedModule.class, ...})
interface ComponentThree {
    @Subcomponent.Builder
    interface Builder {
        Builder repeatedModule(RepeatedModule repeatedModule);
        ComponentThree build();
    }
}

DaggerComponentOne.create().componentThreeBuilder()
    .repeatedModule(new RepeatedModule()) // 运行时报错 UnsupportedOperati
    .build();
```

4. 总结

Component 之间共用相同依赖，可以有两种组织关系：依赖关系与继承关系。在 Android 开发中，一般使用继承关系，以 AppComponent 作为 root Component，AppComponent 一般还会使用 @Singleton 作用域，而 ActivityComponent 为 SubComponent。

下篇文章介绍在 Android 开发中如何使用 Dagger 2，以及如何使用 Activities 的 multibinding（多重绑定）。

END

一个白日做梦的工程师！

不只有技术，还有咖啡和彩蛋！

个人博客：http://johnnyshieh.me