## 声明依赖

dagger构造应用的类实例和提供他们的依赖，使用的是javax.inject.Inject注解来标识它需要哪一个构造方法和字段。

Dagger应该使用@Inject注解的构造方法来创建一个类的实例。当需要一个新实例的时候，Dagger就会获取需要的参数值和调用这个构造器。

class Thermosiphon implements Pump{

private final Heater heater;

@Inject

Thermosiphon(Heater heater){

this.heater=heater;

}

…

}

Dagger可以直接注入字段。在这个示例中，它为heater字段获取一个Heater实例和为pump字段获取一个Pump实例。

class CoffeMaker{

@Inject Heater heater;

@Inject Pump pump;

…

}

如果你的类有一个@Inject注解的字段但是没有@Inject注解的构造器，如果有必要，Dagger就会注入这些字段，但是不会创建新的实例。添加一个无参的构造函数并使用@Inject注解来表明Dagger也许会创建实例。

Dagger也支持方法注入，尽管构造器和字段注入更常见一些。

Dagger不能构建缺少@Inject注解的类。

**提供依赖**

默认情况下，Dagger通过构建上面描述的需要的类型的实例来满足每一个依赖。当你需要一个CoffeeMaker，它会通过new CoffeeMaker和设置它的可注解的字段来获取一个实例。

但是@Inject不是在所有的地方都有效的：

不能构建interface

不能注解第三方库的类

可配置的对象必须设置好

对于这些@Inject不合适的或者不能处理的情况，使用一个@Provides注解方法来提供一个依赖。方法的返回类型定义了它提供哪个依赖。

举个例子，当需要一个Heater实例的时候，provideHeater()方法就会被调用：

@Provides static Heater provideHeater(){

return new ElectricHeater();

}

被@provides注解的方法也可以有他们自己的依赖。当需要一个Pumb实例的时候，这个方法返回一个Thermosiphon实例：

@Provides static Pump providePump(Thermosiphon pump){

return pump;

}

所有@Provides注解的方法必须在一个module里面。这些module就是有@Module注解的类。

@Module

class DripCoffeeModule{

@Provides static heater provideHeater{

return new ElectricHeater();

}

@Provides static Pump providePump(Thermosiphon pump){

return pump;

}

}

通常，@Provides注解的方法是以provide为前缀命名，module java类命名以Module为后缀。

**构建依赖图**

被@Inject和@Provides注解的类构成了一个根据依赖连接的对象图。调用代码像应用程序的主要方法或Android应用程序通过一个定义良好的根访问该图形。在Dagger2中，该集合由具有无参并返回所需类型的方法的接口定义。通过对这样一个interface应用@Component注解和传递module类型到modules参数，Dagger2就可以完全生成这个协议的实现。

@Component(modules=DripCoffeeModule.class)

interface CoffeeShop{

CoffeeMaker maker();

}

实现类的名字是在interface名字的基础上添加了Dagger前缀。实现类通过调用builder（）方法得到一个build实例，然后使用返回的builder来设置依赖并调用build（）方法生成一个新的实例。

CoffeeShop coffeeShop=DaggerCoffeeShop.builder().dripCoffeeModule(new DripCoffeeModule).build();

注意：如果你的@Component注解不是一个顶级类型，生成的component名字会包含他的外部类名字以下划线连接。

举例来说，这个代码：

class Foo{

static class Bar{

@Component

interface BazComponent{}

}

}

会生成一个component叫做DaggerFoo\_Bar\_BazComponent.

任何有可访问的默认构造函数的module都可以省略，因为builder会自动构造一个实例如果没有设置的话。然而对于任意module的@Provides注解的方法都是static的情况，实现类并不需要一个它的实例。如果所有的依赖，不需要用户创建依赖实例，都可以被构建，那么生成的实现类也会有个create（）方法可以被用来不通过builder获得一个新的实例。

CoffeeShop coffeeShop=DaggerCoffeeShop.create();

现在，我们的CoffeeApp可以简单地使用Dagger生成的CoffeeShop实现来获得一个完全注解的CoffeeMaker。

public class CoffeeApp{

public static void main(String[] args){

CoffeeShop coffeeShop=DaggerCoffeeShop.create();

coffeeShop.maker().brew(0;

}

}

现在依赖地图构建出来了，入口点被注解了，我们允许我们的制作咖啡app.欢乐吧。