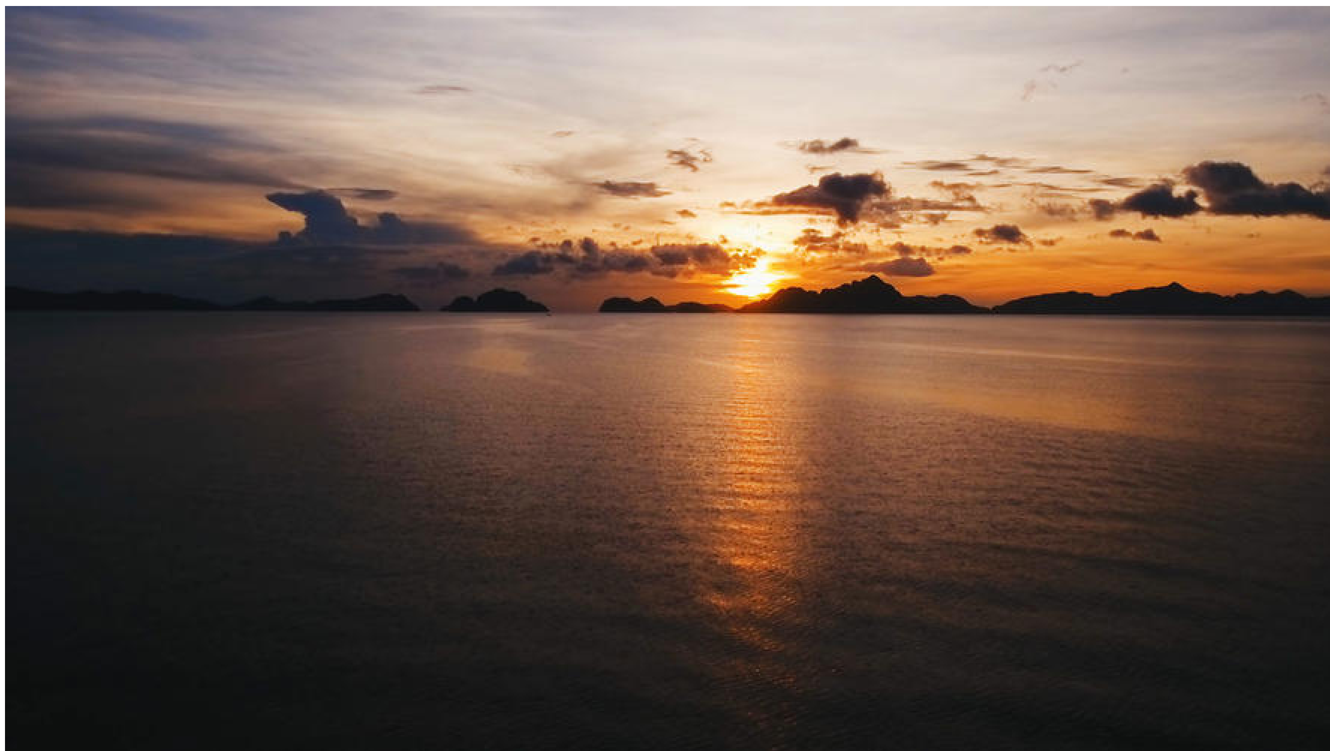
Flutter 入门到头秃 - 依赖注入 Inject

作者：Moyck

原文链接：https://ld246.com/article/1588753860965

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



序言

依赖注入（IOC）就是通过容器，将当前这个类所需的对象实例化，而不需要这个类自身去实例化这对象。目的是为了类的解耦。在小项目里面可能无法体现依赖注入的价值，但是在大型多人合作的项里面，依赖注入能让整个项目更加健壮和易于维护。

Inject

说起依赖注入，最大名鼎鼎的莫过于Java的Spring系列。在Flutter开发中也有很多的依赖注入框架，中官方推荐的框架就是本文的主角 Inject

导入

由于Inject不支持导包的形式，因此只能通过导入源码的方式引入。

在lib同级别目录新建vendor文件夹

在vendor文件夹里导入inject源码

```
git clone https://github.com/google/inject.dart.git
```

然后在pubspec.yaml文件中引用

注意 Inject依赖于build runner，因此也需要引入build runner

```
dependencies:
  inject:
    path: ./vendor/inject.dart/package/inject

dev_dependencies:
  build_runner: ^1.0.0
```

```
  inject_generator:
    path: ./vendor/inject.dart/package/inject_generator
```

之后在项目根目录下创建一个名为inject_generator.build.yaml的文件复制以下配置内容

```
builders:
  inject_generator:
    target: ":inject_generator"
    import: "package:inject_generator/inject_generator.dart"
    builder_factories:
      - "summarizeBuilder"
      - "generateBuilder"
    build_extensions:
      ".dart":
        - ".inject.summary"
        - ".inject.dart"
    auto_apply: dependents
    build_to: source
```

为了隐藏生成的文件，请导航至Android Studio-> Preferences-> Editor-> File Types并将以下代粘贴在ignore files and folders部分下

.inject.summary;.inject.dart;*.g.dart;

在Visual Studio Code中，导航到Preferences-> Settings并搜索Files:Exclude。添加以下代码：

```
**/*.inject.summary
**/*.inject.dart
**/*.g.dart
```

简单入门

直接说原理和概念容易让人一头雾水，因此先展示一个简单的示例

首先写一个非常简单的Presenter

```
@provide
class UserPresenter{

  String getUserName() {
    return "Nike";
  }

}
```

可以看到我们给这个类加了一个@provide的注解

provide这个注解用于告诉Inject，当前这个类是需要让Inject来帮助我们实例化的，当我们编译时Inject会帮我们把这个类放入注射器里面，当有类需要用到UserPresenter的实例化对象时，Inject会再帮们从注射器里将UserPresenter对象注入到该类。

接下来需要一个Widget来使用我们的Presenter

```
@provide
class UserWidget extends StatelessWidget {
```

```
  UserPresenter _userPresenter;

  UserWidget(this._userPresenter);

  @override
  Widget build(BuildContext context) {
    return Text(_userPresenter.getUserName());
  }

}
```

使用这个Presenter也很简单，只需要在构造函数里传入即可，并且给该类加上@provide的注解。

这么写很好理解，但是下一个问题就来了，UserPresenter UserWidget 这两个对象既然不能通过直new来实例化，那它们到底是在哪里被实例化的呢?

按照正常的Ioc来说，必然是有一个注射器来注入实例化对象的，Inject当然也不例外，这里就要用到外一个注解@Injector() 来配置我们需要的注射器。

```
import 'app_injector.inject.dart' as g;

@Injector()
abstract class AppInjector {

  @provide
  UserWidget get userWidget;

  static Future<AppInjector> create() {
    return g.AppInjector$Injector.create();
  }

}
```

由于我们还没有编译，会出现找不到app_injector.inject.dart的错误提示

@Injector()这个注解告诉Inject，当前这个类是一个注射器，我们需要的实例都可以通过注射器来获。比如当前我们需要UserWidget的实例。我们需要这么一行代码

```
 @provide
  UserWidget get userWidget;
```

现在基本的代码已经完成了，我们编译一下生成所需的代码

在命令行执行

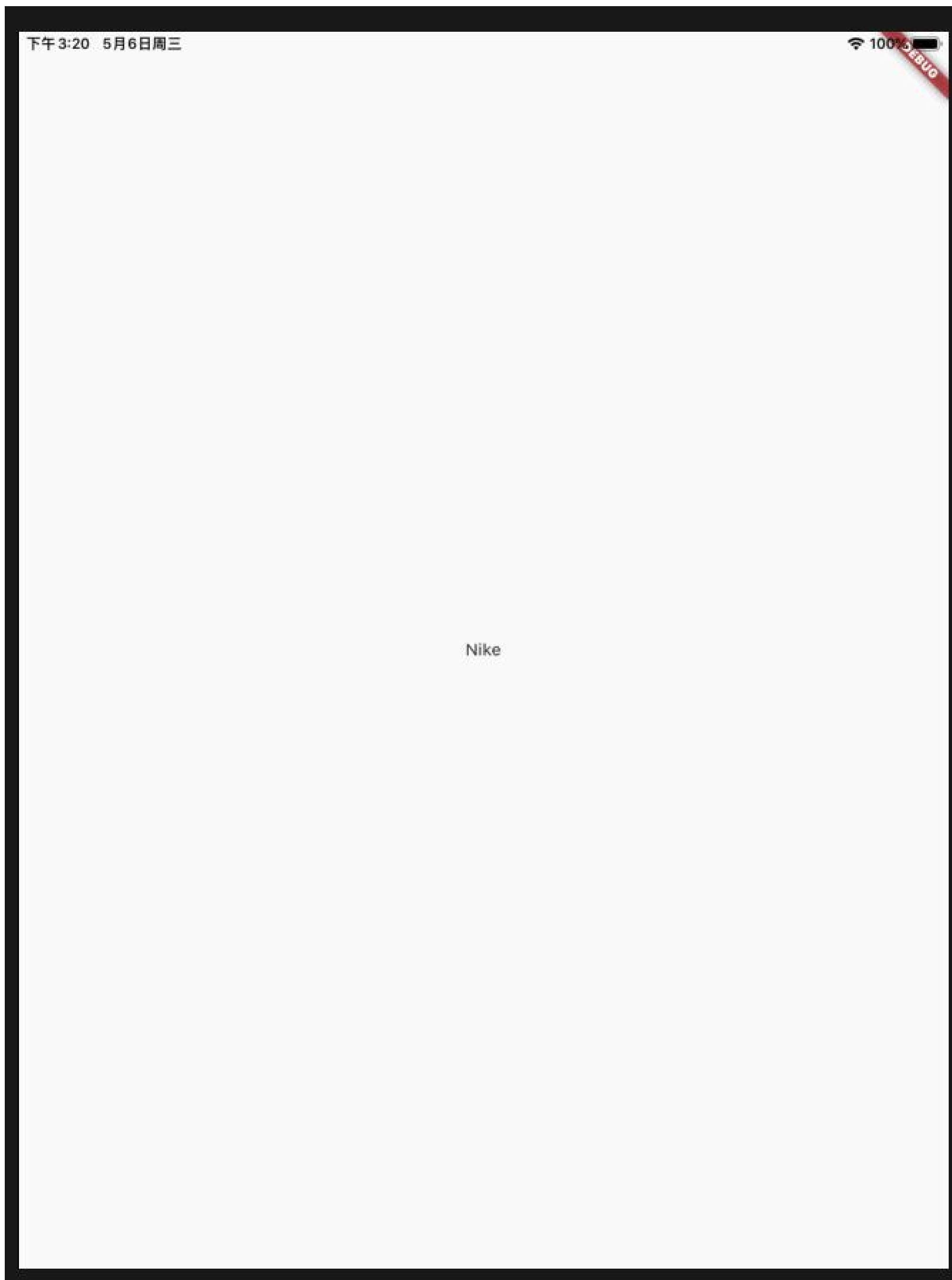
```
flutter packages pub run build_runner build
```

最后 在main函数里使用该注射器

```
void main() async {
  final container = await AppInjector.create();
  runApp( MaterialApp(
    home: Scaffold(
      body: Center(child: container.userWidget),
```

```
    ),
  ));
}
```

可以看到，整个依赖注入已经完成了。



进阶

上面是比较常规的使用，Inject还有别的注解

@singleton

这个注解的含义比较简单，让当前类成为一个单例，只会实例化一次。

```
@singleton
@provide
class UserWidget extends StatelessWidget {
```

我们给之前的UserWidget增加一个singleton注解，重新build之后

我们进入到自动生成的app_injector.inject.dart代码里面

```
  _i2.UserWidget _singletonUserWidget;

  _i2.UserWidget _createUserWidget() =>
      _singletonUserWidget ??= _i2.UserWidget(_createUserPresenter());
```

可以看到当UserWidget不为空时，会重用之前的对象。

@module

平时开发经常会有一个module里面包含多个Repository的写法，如果像上面一样把所有东西扔到Injector 会使Injector变得很臃肿。我们可以通过 @module 注解来解决这个问题

首先 写两个有依赖的Repository

```
@provide
class UserRepository {

  GoodRepository _goodRepository;

  UserRepository(this._goodRepository);

  Future<List<String>> getAllUsers() {
    print("getAllUsers");
    _goodRepository.getAllGoods();
      return null;
  }

}

@provide
class GoodRepository {

  Future<List<String>> getAllGoods() {
    print("getAllGoods");
    return null;
  }

}
```

然后 需要一个module来提供Repository

```
@module
class UserService{

  @provide
  UserRepository userRepository(GoodRepository goodRepository) => UserRepository(goodRepository);
```

```
}
```

最后，在上面的UserPresenter里使用这个Repository

```
@provide
class UserPresenter{

  UserRepository _userRepository;

  UserPresenter(this._userRepository);

  String getUserName() {
    _userRepository.getAllUsers();
    return "Nike";
  }

}
```

重新运行后，可以看到UserRepository也已经被成功注入了

```
flutter: getAllUsers
flutter: getAllGoods
```

原文链接：Flutter 入门到头秃 - 依赖注入 Inject