

# flutter 学习四

---

## 1. 课前复习

---

- 路由的使用
- 数据持久化
- 动画
- Provider

## 2. 课堂目标

---

- 列表的刷新控件实现
- 登录界面开发
- flutter插件开发
- Flutter打包与发布

## 3. 知识点

---

### 1.列表的刷新控件实现

#### 方式一：系统自带API

- 自带的下拉刷新RefreshIndicator

```
@override
Widget build(BuildContext context) {

  return new Scaffold(
    backgroundColor: Color.fromARGB(255, 242, 242, 245),
    appBar: new AppBar(
      elevation: 0.0,
      title: new Text('大前端',
        style: TextStyle(
          fontSize: 20.0, color: Colors.white
        ),
      ),
    ),
    body: RefreshIndicator(
      onRefresh: _onRefresh,
      child: new ListView.builder(
        itemCount: _jobs.length,
        itemBuilder: buildJobItem,
      ),
    ),
  );
}
```

```

    )
  );
}

Future<void> _onRefresh() async {
  await Future.delayed(Duration(seconds: 3), () {
    print('refresh');
  });
}

```



- 上拉加载更多：对于加载更多的组件在Flutter中是没有提供的，但是在ListView中有一个ScrollController属性，它就是专门来控制ListView滑动事件，在这里我们可以根据ListView的位置来判断是否滑动到了底部来做加载更多的处理

```

ScrollController _scrollCtrl = ScrollController();

@override
void initState() {
  super.initState();

  //监听
  _scrollCtrl.addListener(() {
    if(_scrollCtrl.position.pixels ==
    _scrollCtrl.position.maxScrollExtent) {
      print('滑动到最下面了');
    }
  });
}

@override

```

```
Widget build(BuildContext context) {

  return new Scaffold(
    backgroundColor: Color.fromARGB(255, 242, 242, 245),
    appBar: new AppBar(
      elevation: 0.0,
      title: new Text('大前端',
        style: TextStyle(
          fontSize: 20.0, color: Colors.white
        ),
      ),
    ),
    body: RefreshIndicator(
      onRefresh: _onRefresh,
      child: new ListView.builder(
        itemCount: _jobs.length,
        itemBuilder: buildJobItem,
        controller: _scrollCtrl,
      ),
    ),
  );
}
```

另外在加载更多的时候设置最后一行显示加载更多相关的样式即可。

上面的这套上拉下拉的方案有点事使用简单方便，但是缺点是UI可定制性差，无法满足公司的需要

## 方式二：三方插件

- 使用方式
  - 第一步：在pubspec.yaml 文件配置

```
dependencies:
  pull_to_refresh: ^1.5.6
```

- 使用的时候

```
import 'package:pull_to_refresh/pull_to_refresh.dart';
```

[使用点击我](#)

## 2.“花伴侣”登录模块开发

### Flutter表单组件

- TextField：就是一个简单的输入框
- 常用属性如下：
- **autocorrect**：是否自动校正

- **autofocus**: 是否自动获取焦点
- **enabled**: 是否启用
- **inputFormatters**: 数组, 对输入的文字进行限制和校验
- **keyboardType**: 获取焦点时, 启用的键盘类型
- **maxLines**: 输入框最大的显示行数
- **maxLength**: 允许输入的字符长度
- **maxLengthEnforced**: 是否允许输入的字符长度超过限定的字符长度
- **obscureText**: 是否隐藏输入的内容
- **onChanged**: 输入框内容变化的时候回调的事件
- **onSubmitted**: 当用户确定已经完成编辑时触发
- **controller**: 编辑框的控制器, 通过它可以设置/获取编辑框的内容、选择编辑内容、监听编辑文本改变事件。

## Form

Flutter提供了一个Form 组件, 它可以对输入框进行分组, 然后进行一些统一操作, 如输入内容校验、输入框重置以及输入内容保存。

```
Form( {
  Key key,
  @required this.child,
  this.autovalidate = false,
  this.onWillPop,
  this.onChanged,
})
```

- **key**: 通过key属性, 来获取表单对象。
- **child**: 是表单里的一些布局, 结合TextFormField组件来实现表单的验证。
- **autovalidate**: 是否自动校验输入内容; 当为true时, 每一个子FormField内容发生变化时都会自动校验合法性, 并直接显示错误信息。否则, 需要通过调用FormState.validate()来手动校验。
- **onWillPop**: 决定Form所在的路由是否可以直接返回(如点击返回按钮), 该回调返回一个Future对象, 如果Future的最终结果是false, 则当前路由不会返回; 如果为true, 则会返回到上一个路由。此属性通常用于拦截返回按钮。
- **onChanged**: Form的任意一个子FormField内容发生变化时会触发此回调。

```
class FormTest extends StatefulWidget {
  @override
  _FormTestState createState() => new _FormTestState();
}

class _FormTestState extends State<FormTest> {
  TextEditingController _unameController = new TextEditingController();
  TextEditingController _pwdController = new TextEditingController();
  GlobalKey _formKey= new GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```

appBar: AppBar(
  title: Text("Form Test"),
),
body: Padding(
  padding: const EdgeInsets.symmetric(vertical: 16.0, horizontal: 24.0),
  child: Form(
    key: _formKey, //设置globalKey, 用于后面获取FormState
    autovalidate: true, //开启自动校验
    child: Column(
      children: <Widget>[
        TextFormField(
          autofocus: true,
          controller: _unameController,
          decoration: InputDecoration(
            labelText: "用户名",
            hintText: "用户名或邮箱",
            icon: Icon(Icons.person)
          ),
          // 校验用户名
          validator: (v) {
            return v.trim().length > 0 ? null : "用户名不能为空";
          }
        ),
        TextFormField(
          controller: _pwdController,
          decoration: InputDecoration(
            labelText: "密码",
            hintText: "您的登录密码",
            icon: Icon(Icons.lock)
          ),
          obscureText: true,
          //校验密码
          validator: (v) {
            return v.trim().length > 5 ? null : "密码不能少于6位";
          }
        ),
        // 登录按钮
        Padding(
          padding: const EdgeInsets.only(top: 28.0),
          child: Row(
            children: <Widget>[
              Expanded(
                child: RaisedButton(
                  padding: EdgeInsets.all(15.0),
                  child: Text("登录"),
                  color: Theme.of(context).primaryColor,
                  textColor: Colors.white,
                  onPressed: () {

```

```

// 通过_formKey.currentState 获取FormState后,
// 调用validate()方法校验用户名密码是否合法, 校验
// 通过后再提交数据。
if((_formKey.currentState as FormState).validate()){
  //验证通过提交数据
}
},
),
),
1,
),
)
1,
),
),
),
);
}
}

```

使用表单完成如下界面：



### 3.Flutter插件开发

Flutter插件开发就是一种特殊的Flutter库，可以和原生程序交互，因为Flutter的上层是Dart语言，无法完成底层操作，所以一些Dart无法实现的功能需要通过插件开发调用原生。

插件开发涉及两个移动平台的知识：

- iOS: Objective-C、Swift
- Android: Java、Kotlin

## 新建插件的方式：

- 方式一：命令行的方式创建

```
flutter create --org com.example --template=plugin hello
```

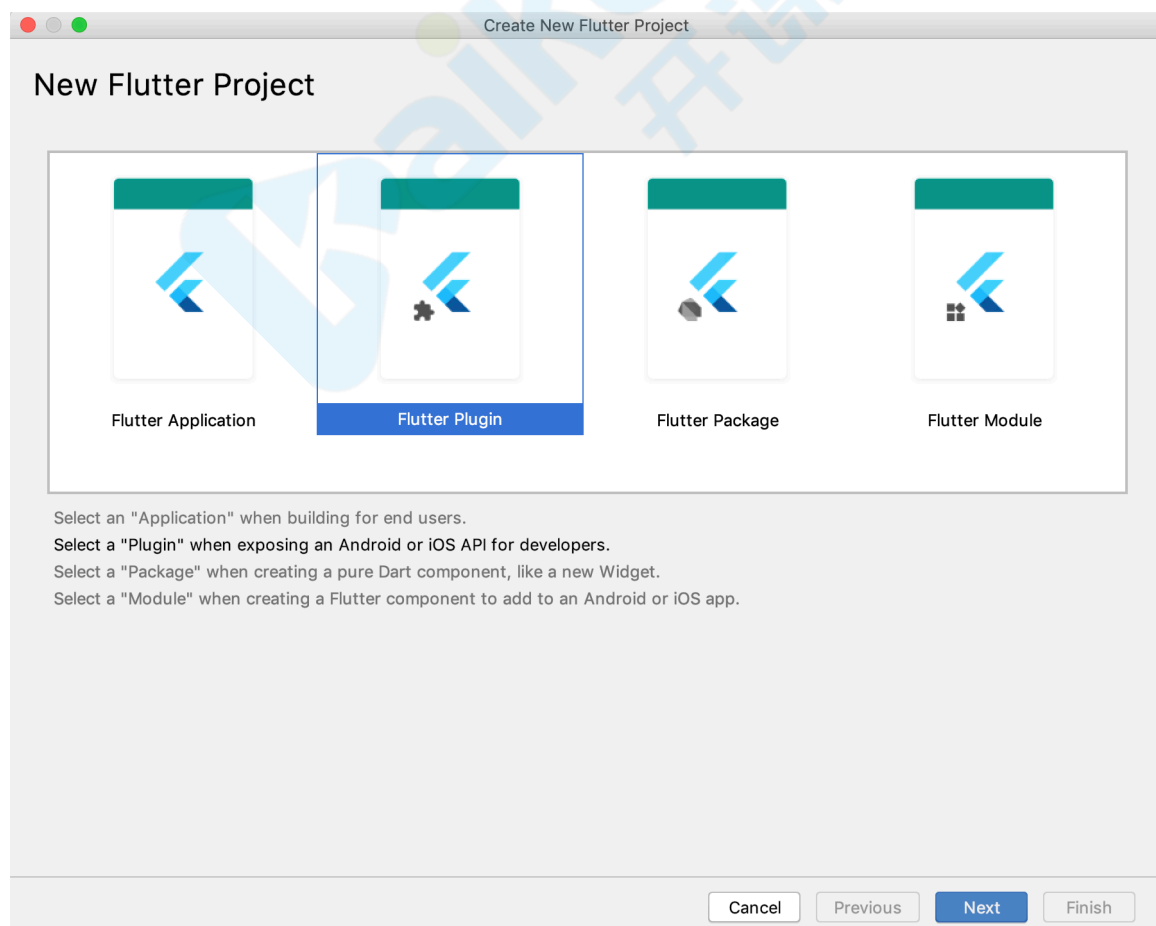
如果想支持swift或者kotlin，可以用如下命令进行创建：

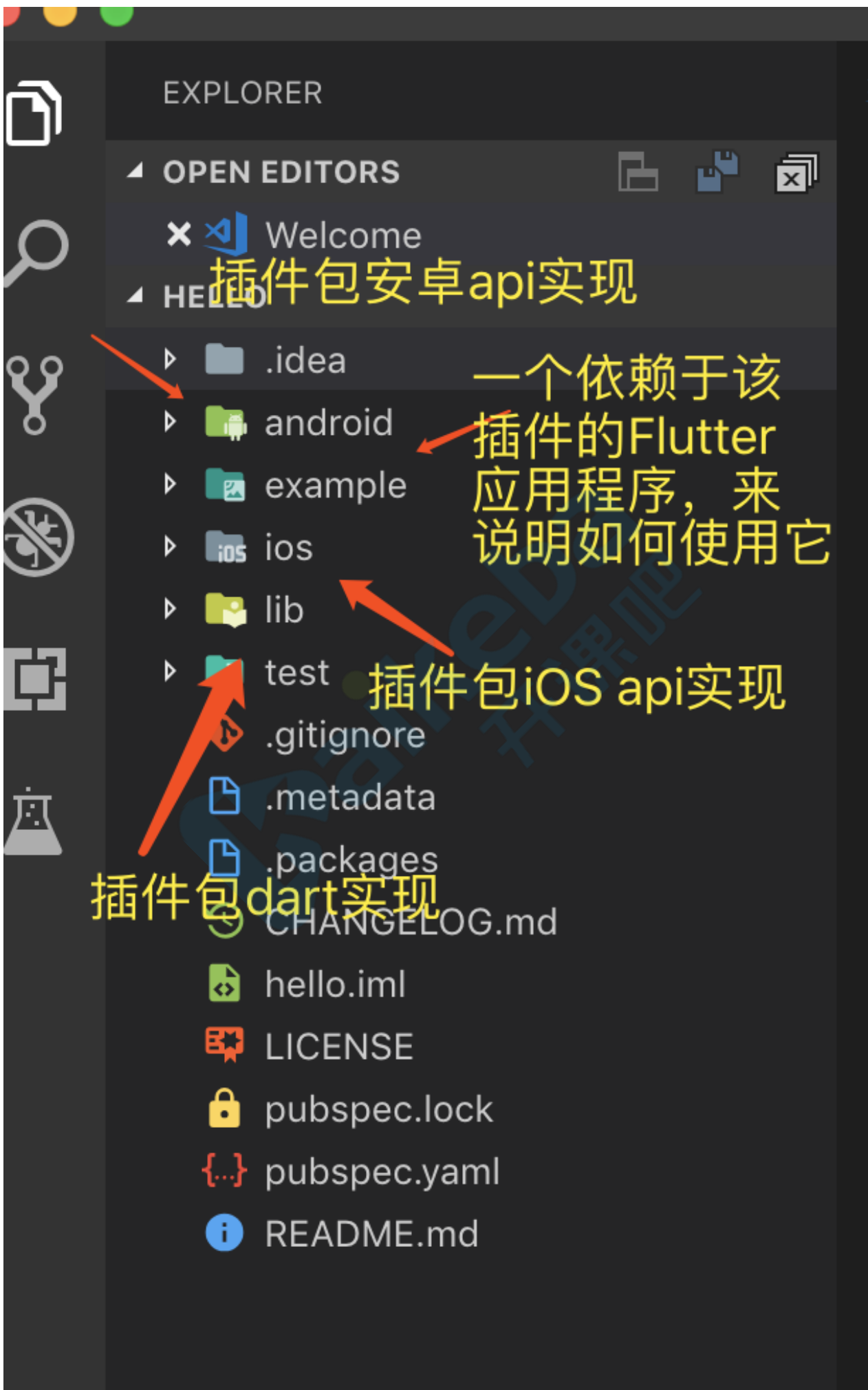
```
flutter create --template=plugin -i swift -a kotlin hello
```

如果想只生成OC和java，可以用如下命令进行创建：

```
flutter create --template=plugin -i objc -a java hello
```

- 方式二：Android Studio创建

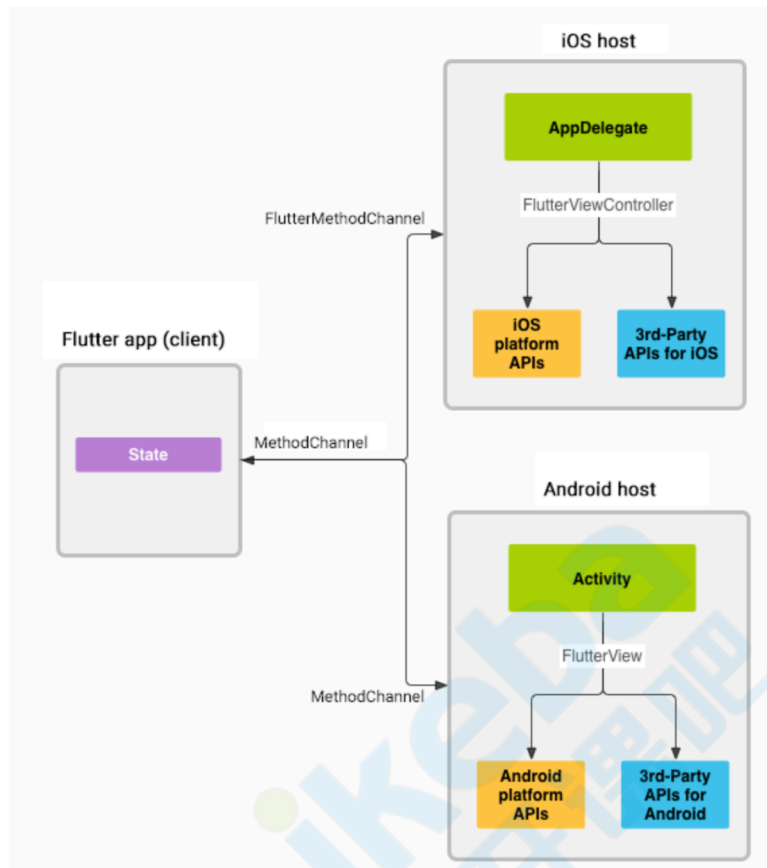






## 插件的通信原理

Flutter插件提供的Android和iOS的底层封装，在Flutter侧提供组件功能，使Flutter可以较方便地调取Native的模块。其原理如下图所示：



整个插件的消息和响应以异步的方式进行传递，以确保用户界面不会卡顿；

从上述的架构图中，可以很明确的知道在 Dart 端使用 MethodChannel API 来发送消息或调用对应的方法，而 Native 平台上 Android 的 MethodChannel 和 iOS 的 FlutterMethodChannel 处理了接收调用和返回结果，这一过程也可以反向调用，即 Native 主动的给 Dart 端发送消息。对于数据转换的过程，如果了解过 JavaScriptCore 和 Objective-C 的互转就能明白，比如 JavaScript 端的 string 转换成 Objective-C 的 NSString，number 转换成 NSNumber。对于 Dart 而言也有这样数据转换的对照表可以点击[这里](#)。

## channel使用

Flutter定义了三种不同类型的Channel，它们分别是

- BasicMessageChannel：用于传递字符串和半结构化的信息
- MethodChannel：用于传递方法调用（method invocation）
- EventChannel：用于数据流（event streams）的通信

三种Channel之间互相独立，各有用途，但它们在设计上却非常相近。每种Channel均有三个重要成员变量：

- name: String类型，代表Channel的名字，也是其唯一标识符
- messenger: BinaryMessenger类型，代表消息信使，是消息的发送与接收的工具
- codec: MessageCodec类型或MethodCodec类型，代表消息的编解码器

## 发布插件到pub

- 终端执行 `flutter packages pub publish --dry-run`

- yaml文件添加配置:

author: email homepage:urlPage

- 执行发布命令:

```
flutter packages pub publish --server=https://pub.dartlang.org
```

此步骤需要访问Google, 需要科学上网, 进行授权

## 4.Flutter打包与发布

- Android:

- 生成key

在命令行中输入 `keytool -genkey -v -keystore D:/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key`, 后面需要设置一系列信息, 记住设置的密码, 后面需要使用

- 创建key.properties

- 在android目录下创建 `key.properties`, 在文件中写入以下信息

```
storePassword=创建密钥库时的密码
keyPassword=创建密钥的密码
keyAlias=key
storeFile=key.jks的路径
```

- 配置app的build.gradle

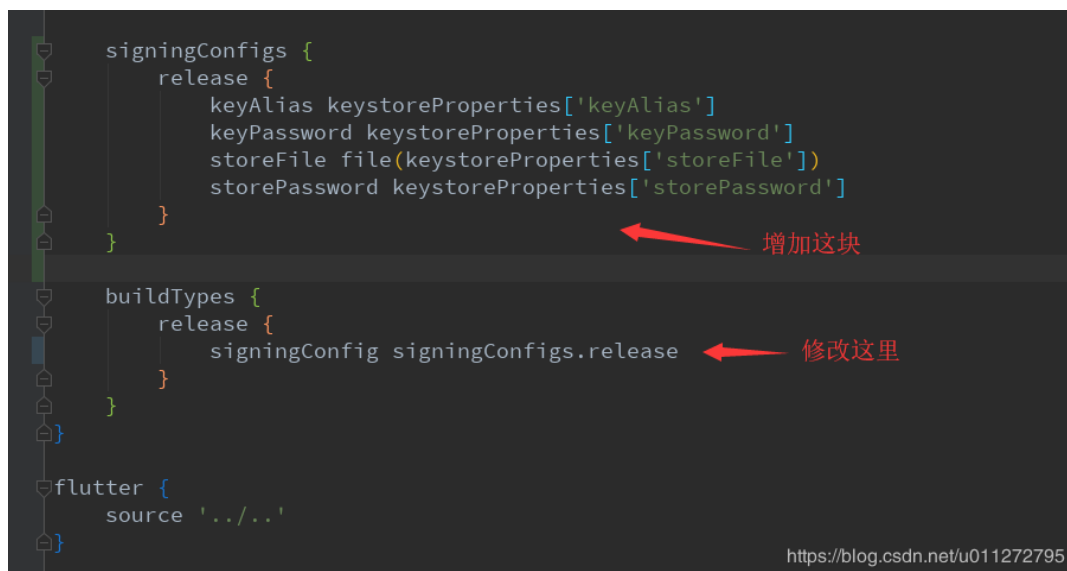
- 在"android{"上面添加以下内容

```
def keystorePropertiesFile = rootProject.file("key.properties")
def keystoreProperties = new Properties()
keystoreProperties.load(new
FileInputStream(keystorePropertiesFile))
```

- 在"buildTypes {"上面添加release配置信息

```
signingConfigs {
    release {
        keyAlias keystoreProperties['keyAlias']
        keyPassword keystoreProperties['keyPassword']
        storeFile file(keystoreProperties['storeFile'])
        storePassword keystoreProperties['storePassword']
    }
}
```

注: buildTypes中从debugger改成release



- 在命令行中输入 `flutter build apk`
- 生成的app-release.apk 就在项目的output/apk/release目录下
- 安装 生成的apk包到手机 `adb install app-release.apk`

成功后有如下log

```
You are building a fat APK that includes binaries for android-arm, android-arm64.
If you are deploying the app to the Play Store, it's recommended to use app bundles or split the APK to reduce the APK size.
To generate an app bundle, run:
  flutter build appbundle --target-platform android-arm,android-arm64
Learn more on: https://developer.android.com/guide/app-bundle
To split the APKs per ABI, run:
  flutter build apk --target-platform android-arm,android-arm64 --split-per-abi
Learn more on: https://developer.android.com/studio/build/configure-apk-splits#configure-abi-split
Initializing gradle... 0.7s
Resolving dependencies... 1.3s
Running Gradle task 'assembleRelease'... 7.8s
Running Gradle task 'assembleRelease'... Done
Built build/app/outputs/apk/release/app-release.apk (13.1MB).
```

- 安装apk则将真机通过usb连接在电脑上（且打开开发者模式）执行flutter install

注意：如果打包过程中报错误 `Error:Execution failed for task ':app:lintVitalRelease'`.

则需要在gradle中添加内容

```
android {
    ...
    lintOptions {
        checkReleaseBuilds false
        abortOnError false
    }
    ...
}
```

- iOS:
  - cd到项目目录下
  - flutter build ios --release
  - 进行iOS原生侧打包

扩展：BLoc

## 4. 作业 && 答疑

---

实现今天的内容

## 5. 下节课内容

---

