flutter 学习

1. 课前复习

2. 课堂目标

- 列表下拉刷新和上拉加载更多
- app配置
- 三方分享
- app打包

3. 知识点

1.自带的下拉刷新RefreshIndicator

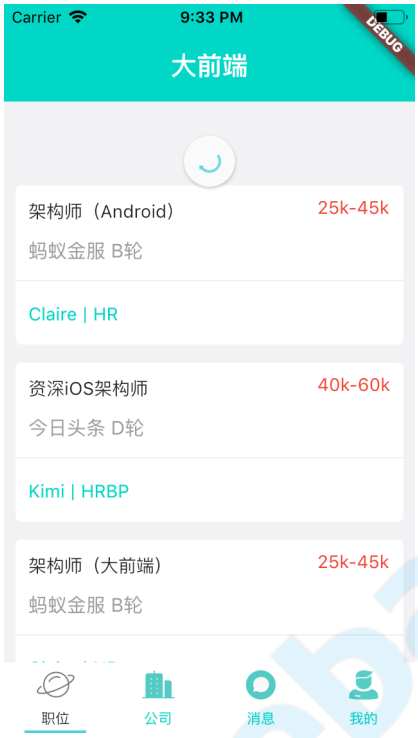
```
@override
  Widget build(BuildContext context) {

    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');
    });
```

```
  }
```

效果如下所示：



2.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`

3.pull_to_refresh上下拉刷新控件

- 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

4.app配置

- 包名（不建议修改，一般是项目创建初决定）
  - Android：
    - `android` ▸ `app` ▸ `src` ▸ `main` ▸ `AndroidManifest.xml` 中修改`package="xxx.xxx.xxx"`
    - 以及在 `android` ▸ `app` ▸ `src` ▸ `build.gradle`中修改`applicationId "xxx.xxx.xxx";`
    - 需要修改`android` ▸ `app` ▸ `src` ▸ `main` ▸ ... ▸ `MainActivity.java`对应的包路径

    - iOS： `ios` ▸ `Runner` ▸ `Info.plist` 中修改`CFBundleIdentifier`对应的`Value`
- 应用名称
    - Android 是在 `android` ▸ `app` ▸ `src` ▸ `main` ▸ `AndroidManifest.xml` 中修改`android:label="XXX"`
    - iOS 在 `ios` ▸ `Runner` ▸ `Info.plist` 中修改`CFBundleName`对应的`Value`
- 图标
    - Android 在`android` ▸ `app` ▸ `src` ▸ `res` ▸ `mipmap-...` 文件夹中替换相应图片
    - iOS 在 `ios` ▸ `Runner` ▸ `Assets.xcassets` ▸ `AppIcon.appiconset`文件夹中替换相应尺寸的图片， 如果使用不同的文件名，那还必须更新同一目录中的`Contents.json`文件。
- 启动图片
    - Android 在`android` ▸ `app` ▸ `src` ▸ `res` ▸ `drawable` ▸ `launch_background.xml` 通过自定义drawable来实现自定义启动界面。
    - iOS 在 `ios` ▸ `Runner` ▸ `Assets.xcassets` ▸ `LaunchImage.imageset`文件夹中替换相应尺寸的图片， 如果使用不同的文件名，那还必须更新同一目录中的`Contents.json`文件。

5.三方分享

Git传送门

- 引入需要使用的share

```
dependencies:
  fluwx: ^1.1.0
```

- 在使用Fluwx之前需要初始化

```
import 'package:fluwx/fluwx.dart' as fluwx;
fluwx.register(appId:"wxd930ea5d5a258f4f",universalLink:"https://your.univeral
.link.com/placeholder/");
```

- 分享文本

```
fluwx.share(fluwx.WeChatShareTextModel(
      text: "text from fluwx",
      transaction: "transaction}",//仅在android上有效，下同。
      scene: scene
    ));
```

6.打包

- 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'...
Running Gradle task 'assembleRelease'... Done                       7.8s
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