

Flutter（二十四）打包发布

AlanGe

一. Android 打包和发布

1.1. 填写应用配置

1.1.1. 基本信息

在之前讲解应用程序的配置信息时，我们已经介绍过，这里不再过多介绍

- 应用的 AppID
- 应用的名称
- 应用的 Icon
- 应用的 Launcher

1.1.2. 版本信息

Flutter 的版本信息在哪里填写呢？在 pubspec.yaml 中

```
version: 1.0.0+1
```

第一次见到这个会很疑惑，后面的 +1 是什么意思呢？

在 Android 中，应用的版本分为 `versionCode` & `versionName`

- `versionCode`: 内部管理的版本号
- `versionName`: 用户显示的版本号

在 iOS 中，应用的版本分为 `version` & `build`

- `version`: 用户显示的版本
- `build`: 内部管理的版本

Flutter 中我们如何管理这两个版本号呢？

1.0.0

- ~~1.0.0.0~~: 用户显示的版本
- 1: 内部管理的版本

1.1.3. 用户权限配置

在 Android 中某些用户权限需要在 `AndroidManifest.xml` 进行配置：

- 比如默认情况下应用程序是 **不能发送网络请求的**，如果之后 App 中有用到网络请求，那么需要在 `AndroidManifest.xml` 中进行如下配置（**默认 debug 模式下有配置网络请求**）
- 比如我们需要访问 **用户的位置**，那么需要在 `AndroidManifest.xml` 中进行如下配置

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.catefavor">
    <uses-permission
android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
</manifest>
```

1.2. 应用程序签名

Android 系统在安装 APK 的时候，首先会检验 APK 的签名，如果发现签名文件不存在或者校验签名失败，则会拒绝安装，所以应用程序在发布之前一定要进行签名。

1.2.1. 创建一个秘钥库

在 macOS 或者 Linux 系统上，执行下面的命令：

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

在 Windows 系统上，执行下面的命令：

```
keytool -genkey -v -keystore c:/Users/USER_NAME/key.jks -storetype JKS -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

```
why:catefavor xiaomage$ keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
输入密钥库口令：
再次输入新口令：
您的名字与姓氏是什么？
[Unknown]: coderwhy
您的组织单位名称是什么？
[Unknown]: coderwhy
您的组织名称是什么？
[Unknown]: coderwhy
```



图片

1.2.2. 在 app 中引用秘钥库

创建一个名为 `/android/key.properties` 的文件，它包含了密
钥库位置的定义：

```
storePassword=<上一步骤中的密码>
keyPassword=<上一步骤中的密码>
keyAlias=key
storeFile=<密钥库的位置, e.g. /Users/<用户名>/key.jks>
```

注意：这个文件一般不要提交到代码仓库

- 修改.gitignore 文件

```
# Android ignore
/android/key.properties
```

1.2.3. 在 gradle 中配置签名 gradle 格莱尔

通过编辑 `/android/app/build.gradle` 文件来为我们的 app 配
置签名：

1.在 `android` 代码块之前添加：

```
android {  
    ...  
}
```

替换为秘钥库的信息

- 将 `key.properties` 文件加载到 `keystoreProperties` 对象中。

```
def keystoreProperties = new Properties()  
def keystorePropertiesFile =  
rootProject.file('key.properties')  
if (keystorePropertiesFile.exists()) {  
    keystoreProperties.load(new  
FileInputStream(keystorePropertiesFile))  
}  
  
android {  
    ...  
}
```

2.在 `buildTypes` 代码块之前添加：

```
buildTypes {  
    release {  
        // TODO: Add your own signing config for the  
release build.  
        // Signing with the debug keys for now,  
        // so `flutter run --release` works.  
        signingConfig signingConfigs.debug  
    }  
}
```

```
}  
}
```

替换为下面的代码：

- `build.gradle` 文件中配置 `signingConfigs` 部分

```
signingConfigs {  
    release {  
        keyAlias keystoreProperties['keyAlias']  
        keyPassword keystoreProperties['keyPassword']  
        storeFile keystoreProperties['storeFile'] ?  
file(keystoreProperties['storeFile']) : null  
        storePassword keystoreProperties['storePassword']  
    }  
}  
buildTypes {  
    release {  
        signingConfig signingConfigs.release  
    }  
}
```

现在我们发布的 `app` 就会被自动签名了

1.3. 打包应用程序

目前 Android 支持打包两种应用程序：APK、AAB

APK 文件：

- Android application package
- 目前几乎所有的应用市场都支持上传 APK 文件
- 用户直接安装的就是 APK 文件

```
# 运行 flutter build apk (flutter build 默认带有 --release 参数)  
flutter build apk
```

AAB 文件：(推荐)

- Android App Bundle
- Google 推出的一种新的上传格式，某些应用市场不支持的
- 会根据用户打包的 aab 文件，动态生成用户设备需要的 APK 文件

```
# 运行 flutter build appbundle。 (运行 flutter build 默认构建一个发布版本。)  
flutter build appbundle
```

1.4. 发布应用程序

Android 应用程序可以发布到很多的平台，包括国内的平台和国外的 Google Play

国内的应用市场非常多，包括 360、百度、小米等等

- 可以根据不同的应用市场相关的规则，上传对应的 APK 或者 AAB 文件，填写相关的信息审核即可

国外的应用市场通常只有一个 Google Play

- 1.需要申请一个 Google Play 开发者账号
- 需要支付 25 美元注册费用的信用卡，信用卡需要支持 Visa, Master Amex, Discover, JCB。
- <https://play.google.com/apps/publish/signup/>
- 2.进入到管理中心，创建应用发布即可
- 进入了 Google Play Console 管理中心



图片

二. iOS 打包和发布

2.1. 填写应用信息

2.1.1. 基本信息

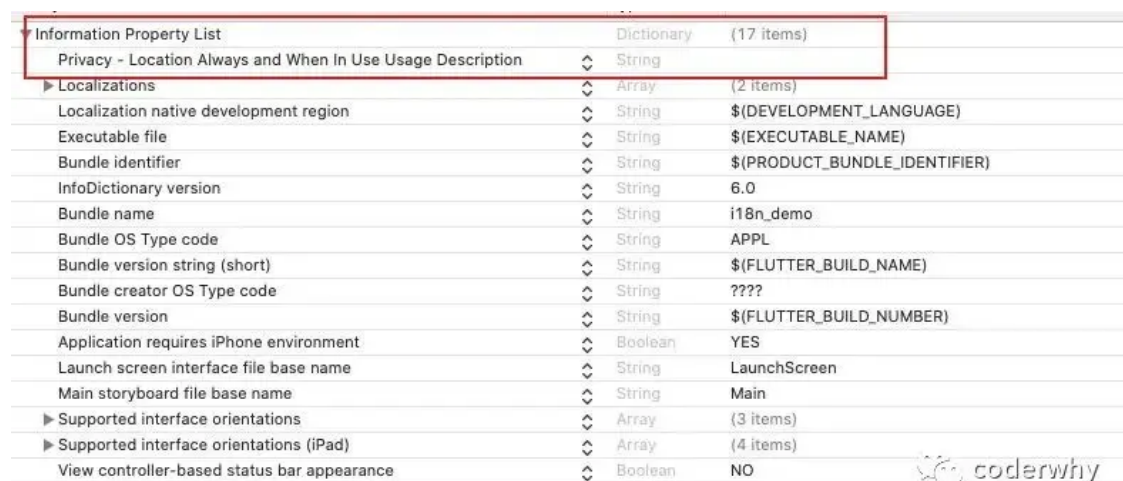
和 Android 一致

2.1.2. 版本信息

和 Android 一致

2.1.3. 用户权限配置

在 iOS 中某些权限，需要用户允许，为了添加这些权限需要配置 info.plist 文件：



Information Property List	Dictionary	(17 items)
Privacy - Location Always and When In Use Usage Description	String	
▶ Localizations	Array	(2 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	i18n_demo
Bundle OS Type code	String	APPL
Bundle version string (short)	String	\$(FLUTTER_BUILD_NAME)
Bundle creator OS Type code	String	????
Bundle version	String	\$(FLUTTER_BUILD_NUMBER)
Application requires iPhone environment	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (iPad)	Array	(4 items)
View controller-based status bar appearance	Boolean	NO

图片

2.2. Apple 开发者账号

2.2.1. 申请开发者账号

苹果发布应用程序，必须申请一个 Apple 开发者账号：

- <https://developer.apple.com/>
- 1.填写 AppleID（没有的话先申请 AppleID）
- 2.加入开发者计划
- 个人和公司：\$99/年
- 企业账号：\$299/年

2.2.2. 配置相关证书

发布 iOS 应用程序需要配置相关的 AppID 和证书：

- <https://developer.apple.com/account/>

- 登录开发者的账号：
- 下载和安装证书，电脑才具备发布程序的能力
- 1.创建 AppID（和自己的应用程序的 AppID 是一直的）
- 2.配置发布者证书（iOS Distribution）

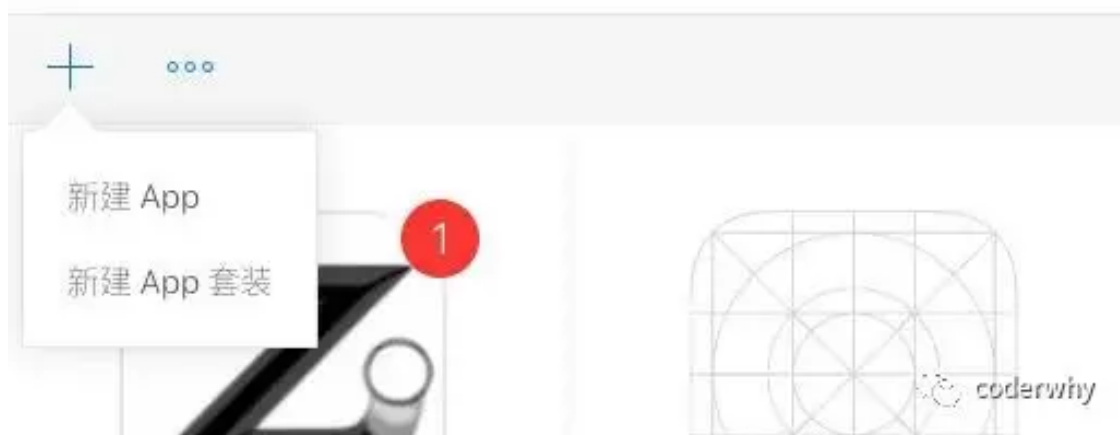


图片

2.2.3. 创建发布 App

我们需要在 App Store 创建一个新的应用程序：

- <https://appstoreconnect.apple.com/>
- 新建 App，并且填写相关信息即可

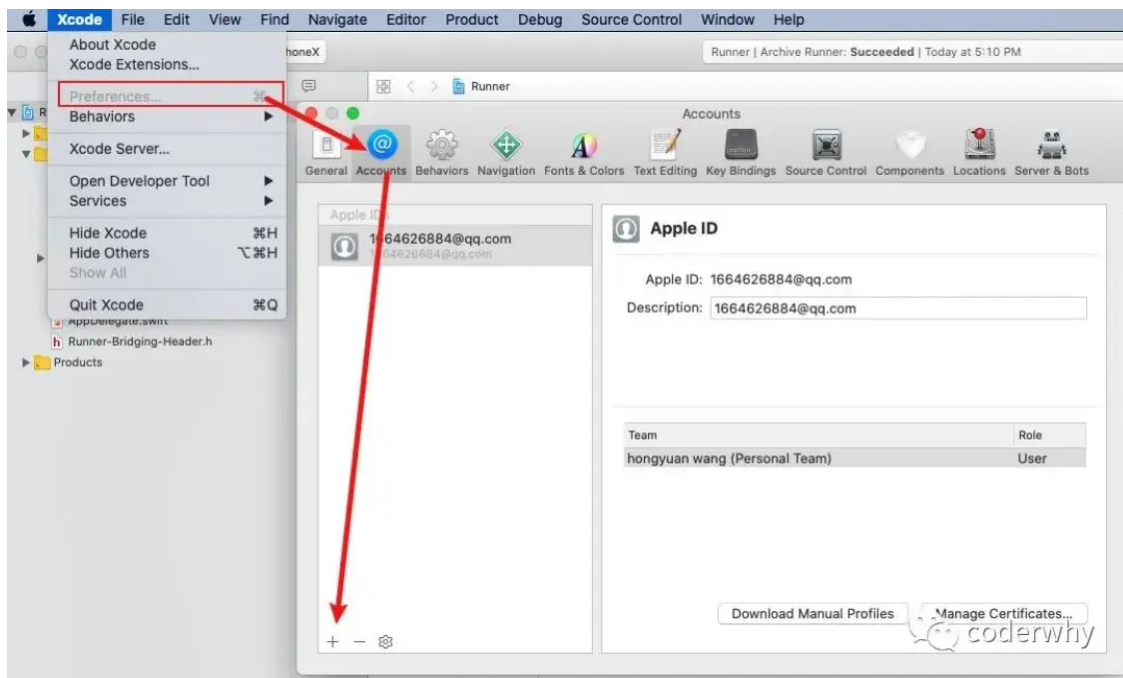


图片

2.3. 打包发布应用

2.3.1. Xcode 登录 AppleID

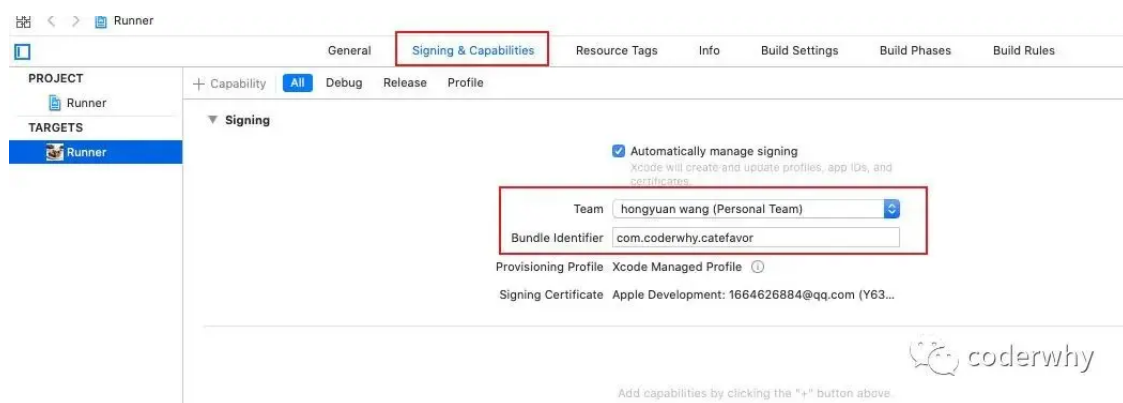
在 Xcode 中登录自己的 AppleID



图片

2.3.2. 打包和发布应用

应用签名信息：



图片

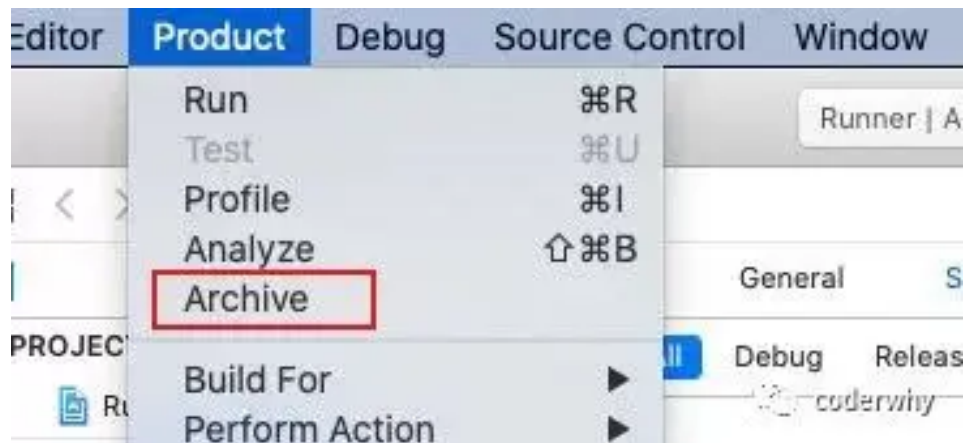
打包应用程序：

1.设备选择真机（这里必须选择真机）

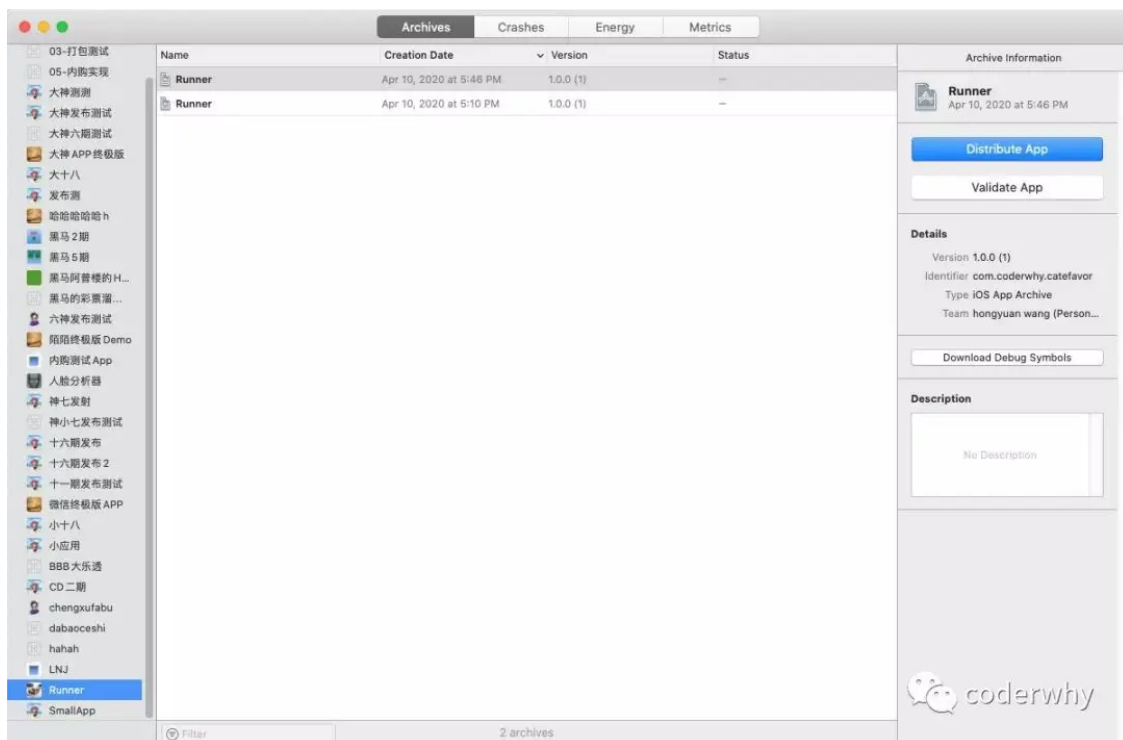


图片

2.Product -> Archive -> Distribute App



图片



图片

注意：如果之前的应用程序是运行在模拟器上的，那么 Archive 时会报错

- 需要删除 ios/Flutter 目录下之前生成的 App.framework
- 因为这个 framework 默认是给模拟器生成的，我们发布的程序要跑在真机设备上

2.4. Application loader

目前很多应用程序的发布喜欢借助于 Application loader，所有的流程都可以在这个工具中完成

- 具体的使用过程可以查找相关的资料，用法比较简单

参考：[小码哥 Flutter](#)