ZenonHuang Lv2
2020年08月13日 阅读 2067

关注

iOS 的自动构建流程

前言

一个对效率有追求的公司，都应该有一套自动构建系统。

目前使用的这套 iOS 构建流程，经历 2 年的使用，基本稳定下来。

这篇文章主要用来记录📝 目前自己使用的 Jenkins 打包📦脚本。

用来打包做类似事情的工具很多，更主要的是为什么使用自动构建:

1. 效率上，解放出开发人员的时间。也更方便其他同事使用。
2. 保证打包的标准，避免配置或环境问题，带来的失败。把事情做对，比做快更重要。
3. 权限安全上，通过构建系统集中管理，对于使用者来说是一个黑盒。
4. 项目流程上，便于有需求时做 Daily Build 或者 自动测试。

对于 `如何安装 Jenkins` 或者 `Jenkins 参数配置` 之类的基本配置不做涉及。

网上已经有不少详细的文章进行介绍。比如 手把手教你利用Jenkins持续集成iOS项目。

大体的iOS 构建流程

先介绍整体的构建流程，具体的内容会在下面分步骤介绍.

> 下面使用的相应 ruby 脚本已经上传 github 仓库,注意的是，里面的变量进行了脱敏处理，根据自己需要去稍作修改

构建前

- 配置 app 图标水印（build号，分支）
- ruby 脚本根据参数，修改工程 bundleID ， 宏等
- 安装第三方依赖，pod update

执行构建

- xcodebuild clean
- xcodebuild archive
- xcodebuild exportArchive

构建完成

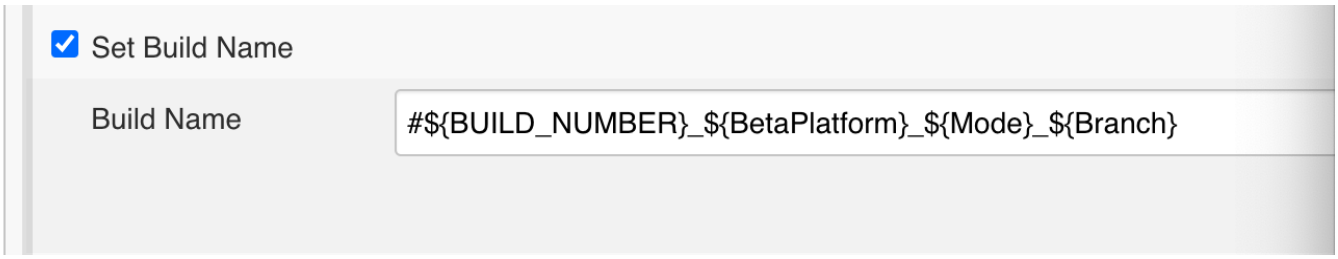
- 上传分发平台: 蒲公英/fir/appstore（历史版本记录: git tag ）
- 符号表处理：上传 bugly
- 归档产物: 上传 FTP 服务器
- 清理: 删除 IPA 等
- 设置构建描述
- 通知: 企业微信 webhook 机器人推送

构建前

设置构建名

首先设置我们的构建名称，我这里使用到几个参数:

- BUILD_NUMBER ,Jenkins 自带的参数，代表第几次构建
- BetaPlatform ,设置的选项参数，代表分发平台。我这里的值分别是: `fir` , `pgyer` , `appstore`
- Mode,设置的选项参数，代表 Xcode 构建的环境设置，为 `Snapshot` 和 `Release`
- Branch,Jenkins 自带的参数，代表 Git 分支名称



配置 APP 图标

为了打包后进行测试的 APP ,便于定位问题，可以在 App Logo 上打上水印，加入构建使用的 `git 分支名`， `jenkins 构建号`， `app 版本号` 等关键信息。

配置图标水印的流程为:

- 判断此次是否为 appstore 分发平台。如果是 appstore 的话，将旧有的图标目录清理掉，然后将图标复制到使用的目录中。
- 如果不是 appstore ，则为测试平台分发，进行水印处理。

打包前替换资源

> Note: 在处理图标做替换时，原来有两种方式，一种是在构建完成后，进入 app 的资源中进行替换（现在行不通了）。另一种是，直接修改工程中的资源。 目前是用的方法，就是 直接修改工程目录中的图标源文件 .所以要在构建之前完成加水印替换 Logo.

因为要使用替换资源的方式，所以准备两个目录。

一个目录作为 `源目录`，存放未处理的图片。一个目录作为`目标目录`，存储 App Logo 使用的图片。

为什么使用两个图片目录存储? 假设只用一个，原图为A，当第一次处理，图片为 A1水印图片，当第二次再拿到的图片，已经是被处理过的 A1水印图片了，而不是原图A 。

这里注意 `icons_path` 为存放原图的地址， `icons_dest_path` 为要修改使用的目标路径。 命名为 `AppIcon-Internal`。

可以参考 iOS APP图标版本化

关于 version 的获取， 因为目前版本有改动，使用 ruby 去获取,脚本会在后面提供链接:

```
version=$(ruby ./ToolChain/ruby/dy_build_version.rb ${Mode})
```

还有一个临时存放路径，要提前创建好这个文件夹:

```
tmp_path="/Users/${sys_username}/Desktop/iOS_IPA/IconVersioning
```

ImageMagick

```
brew install imagemagick
# 安装Ghostscript，它提供了支持ImageMagick的字体。
brew install ghostscript
```

脚本内容

具体的脚本如下:

```
#!/bin/bash -l

echo "🦠 ------------- 配置 app 图标 --------------------"

#本机 Mac 的用户名
sys_username="$USER"
#Jenkins 构建的任务名
jenkinsName=${JOB_NAME}
# 工程名
APP_NAME="your app name"
#项目 repo 目录
Workspace="${WORKSPACE}"


project_infoplist_path="./${APP_NAME}/Info.plist"
#临时图片存放路径
tmp_path="/Users/${sys_username}/Desktop/iOS_IPA/IconVersioning"

# 如果平台为  appstore
if [ "$BetaPlatform" = "appstore" ];then
   echo "🍃🍃🍃  上传平台 为 appstore 🍃🍃🍃"
   echo "icons_path: ${icons_path}"
   echo "icons_dest_path: ${icons_dest_path}"

#1.清除原来 png 文件
find "${icons_dest_path}" -type f -name "*.png" -print0 |
while IFS= read -r -d '' file; do
echo "rm file $file"
rm -rf $file
done

#2. icons_path 复制到icons_dest_path
```

```
image_name=$(basename $file)
echo "copy image: ${image_name}"
cp $file ${icons_dest_path}/${image_name}
done



else
# 如果平台为其它内测分发平台
   echo "🍃🍃🍃 上传平台 为 pagyer/fir,加水印 🍃🍃🍃"

   convertPath=`which convert`
   echo ${convertPath}
   if [[ ! -f ${convertPath} || -z ${convertPath} ]]; then
      echo "warning: Skipping Icon versioning, you need to install ImageMagick and ghostscr
      brew install imagemagick
      brew install ghostscript"
      exit -1;
   fi

    # 说明
    # version    app-版本号
    # build_num  app-构建版本号.
    version=$(ruby ./ToolChain/ruby/dy_build_version.rb ${Mode})
    build_num=${BUILD_NUMBER}

    # 检查当前所处Git分支
    cut="$Branch"
     echo ${cut#*/}
    #shell 截取字符串
    branch=${cut#*/}

    shopt -s extglob
    build_num="${build_num##*( )}"
    shopt -u extglob

    #图片显示的文字内容
    if [ "${isBeta}" = "YES" ];then
       echo " 🍜🍜🍜 为Beta 版本"
       caption="${version}($build_num)\n${branch}(Beta)"
    else
      caption="${version}($build_num)\n${branch}"
    fi

    echo $caption

function abspath() { pushd . > /dev/null; if [ -d "$1" ]; then cd "$1"; dirs -l +0; el
```

```
    base_file=$1
    temp_path=$2
    dest_path=$3

    if [[ ! -e $base_file ]]; then
    echo "error: file does not exist: ${base_file}"
    exit -1;
    fi

    if [[ -z $temp_path ]]; then
    echo "error: temp_path does not exist: ${temp_path}"
    exit -1;
    fi

    if [[ -z $dest_path ]]; then
    echo "error: dest_path does not exist: ${dest_path}"
    exit -1;
    fi

    file_name=$(basename "$base_file")
    final_file_path="${dest_path}/${file_name}"

    base_tmp_normalizedFileName="${file_name%.*}-normalized.${file_name##*.}"
    base_tmp_normalizedFilePath="${temp_path}/${base_tmp_normalizedFileName}"

# Normalize
    echo "Reverting optimized PNG to normal"
    echo "xcrun -sdk iphoneos pngcrush -revert-iphone-optimizations -q '${base_file}' '${ba
    xcrun -sdk iphoneos pngcrush -revert-iphone-optimizations -q "${base_file}" "${base_tmp

    width=`identify -format %w "${base_tmp_normalizedFilePath}"`
    height=`identify -format %h "${base_tmp_normalizedFilePath}"`

    band_height=$((($height * 50) / 100))
    band_position=$(($height - $band_height))
    text_position=$(($band_position - 8))
    point_size=$(((15 * $width) / 100))

    echo "Image dimensions ($width x $height) - band height $band_height @ $band_position -

#
# blur band and text
#
    convert "${base_tmp_normalizedFilePath}" -blur 10x8 /tmp/blurred.png
    convert /tmp/blurred.png -gamma 0 -fill white -draw "rectangle 0,$band_position,$wi
    convert -size ${width}x${band_height} xc:none -fill 'rgba(0,0,0,0.2)' -draw "recta
    convert -background none -size ${width}x${band_height} -pointsize $point_size -fill whi
```

```
    rm /tmp/blurred.png
    rm /tmp/mask.png

#
# compose final image
#
    filename=New"${base_file}"
    convert /tmp/temp.png /tmp/labels-base.png -geometry +0+$band_position -composite /tmp/

# clean up
    rm /tmp/temp.png
    rm /tmp/labels-base.png
    rm /tmp/labels.png
    rm "${base_tmp_normalizedFilePath}"

    echo "Overlayed ${final_file_path}"
}



#把 appIcon 的图片，复制到 AppIcon-Internal
icons_path="${Workspace}/${APP_NAME}/Resources/Assets.xcassets/AppIcon.appiconset"
icons_dest_path="${Workspace}/${APP_NAME}/Resources/Assets.xcassets/AppIcon-Internal.appico

icons_set=`basename "${icons_path}"`

echo "icons_path: ${icons_path}"
echo "icons_dest_path: ${icons_dest_path}"

    mkdir -p "${tmp_path}"

    if [[ $icons_dest_path == "\\" ]]; then
        echo "error: destination file path can't be the root directory"
        exit -1;
    fi

    rm -rf "${icons_dest_path}"
    cp -rf "${icons_path}" "${icons_dest_path}"

    # Reference: https://askubuntu.com/a/343753
    find "${icons_path}" -type f -name "*.png" -print0 |
    while IFS= read -r -d '' file; do
        echo "$file"
        processIcon "${file}" "${tmp_path}" "${icons_dest_path}"
    done
```

Ruby 修改工程参数

这里使用 ruby 实现参数修改(当然也可使用 python 等各种语言，自己方便就 OK)。

> 根据自己的场景做区分，有的参数时不要的可以不做。这里主要记录笔者自己用到的，修改参数和添加参数标记的方法

目前做的操作：

- 区分是否 beta 版本 -- 修改定义 `beta` 宏 的真假值
- 不同分发平台，使用不同 bundleID -- 对 bundleID 进行修改

```
#!/bin/bash -l

export LANG=en_US.UTF-8
export LANGUAGE=en_US.UTF-8
export LC_ALL=en_US.UTF-8

echo ${isBeta}
echo ${channel}

if [ "${isBeta}" = "YES" ];then
  echo " 🍜🍜🍜 为Beta 版本"
  ruby ./ToolChain/ruby/dy_build_global.rb -isbeta-BETA -channel-${channel}
else
  echo " 🍜🍜🍜 不是 Beta 版本"
  ruby ./ToolChain/ruby/dy_build_global.rb -channel-${channel}
fi


if [ "$BetaPlatform" = "pgyer" ];then
  echo "pgyer 🌹 修改bundleID  com.xx.yy.test , profile"
  ruby ./ToolChain/ruby/dy_edit_profile.rb
fi

if [ "$BetaPlatform" = "appstore" ];then
  echo "appstore 🚀  保持 bundleID,profile"
fi

if [ "$BetaPlatform" = "fir" ];then
   echo "fir 🚀  保持 bundleID,profile"
fi
```

脚本里依靠 CocoaPods 开源的 Xcodeproj ， 对工程的 name.xcodeproj/project.pbxproj 文件进行配置修改。

python 的话，可以使用这个项目 mod-pbxproj

Pod 操作

安装/更新第三方库，这里使用到的是 Cocoapods,其它的包管理器可使用其它方式。

```
echo "🌲 ------------- Pod 操作 ---------------------"

pod update --verbose --no-repo-update

echo "🌲 ------------- Pod 完成 ---------------------"
```

执行构建

准备工作

在开始之前，我们要做些准备工作，比如设置要使用的变量，常量。

需要提前写好，尽量避免散落。

```
echo "🏀 ------------- 获取材料 ---------------"

#本机 Mac 的用户名
sys_username="$USER"
#Jenkins 构建的任务名
jenkinsName=${JOB_NAME}
# 工程名
APP_NAME=""
#scheme名
SCHEME_NAME=""

#工程绝对路径
project_path="${WORKSPACE}"
#时间
DATE="$(date +%Y-%m-%d-%H-%M-)"
#info.plist路径
project_infoplist_path="./${APP_NAME}/Info.plist"
```

```
buglyPath=/Users/${sys_username}/Desktop/buglySymboliOS
```

Build 号相关

旧有的方式，是直接通过 info.plist 取:

```
#version
bundleVersion=$(/usr/libexec/PlistBuddy -c "print CFBundleShortVersionString" "${project_in

#bundleID
BundleID=$(/usr/libexec/PlistBuddy -c "print CFBundleIdentifier" "${project_infoplist_path}
```

然而在新的 Xcode 取 `版本号` 和 bundleID 的方式发生变化， 现在 `info.plist` 里的值是变量名,取版本号为 `$(MARKETING_VERSION)`, bundleID 为 `$(PRODUCT_BUNDLE_IDENTIFIER)`.

结局思路是通过脚本到工程配置里去获取，下面使用 ruby 实现了这两个目的。

我们将 App 与 Jenkins 的 build number 设置为同一个，方便需要时，查看构建的参数以及符号表等:

```
#通过脚本取得取版本号 x.x.x
bundleShortVersion=$(ruby ./ToolChain/ruby/dy_build_version.rb "${Mode}")

#通过脚本取得 bundleID
BundleID=$(ruby ./ToolChain/ruby/dy_build_bundIeID.rb "${Mode}")

#修改 ipa 的 build 号，和 jenkins 构建号相同
/usr/libexec/PlistBuddy -c "Set :CFBundleVersion $BUILD_NUMBER" "${project_infoplist_path}"

#取build值
bundleVersion=$(/usr/libexec/PlistBuddy -c "print CFBundleVersion" "${project_infoplist_pat

#  bundleVersion 正常情况要与 BUILD_NUMBER 一样
echo "BundleID:${BundleID} Verision:${bundleVersion} Jenkins Build: $BUILD_NUMBER "
```

加入 `security` 解锁操作的原因,是在子节点 ssh 登录上去之后，keychain 没有被解锁.导致打包失败.

解决方案是用 security unlock-keychain 命令将证书解锁。

```
# 这里默认是 login keychain, login keychain 的密码默认是用户的登录密码
security -v unlock-keychain -p "password"
```

另外可以通过命令查看描述文件的详细信息 包括UUID等信息

```
/usr/bin/security cms -D -i 文件路径
```

Xcodebuild

对工程进行构建打包，主要在于使用 Xcodebuild .

分为三个阶段:

- Clean
- Archive
- Export

如果在执行过程中又不喜欢日志输出的，可以在命令行最后加上

```
-quiet    #只有 warn 和 error 才会输出
```

清理工程

每次构建时，对工程进行 clean ,保证没有其它影响的因素。

使用 `xcodebuild clean [-optionName]...` 清除编译过程生成文件,使用如下:

```
#//下面是集成有Cocopods的用法
echo "  =================clean=================  "

xcodebuild clean -workspace "${APP_NAME}.xcworkspace" -scheme "${APP_NAME}"  -configuration
```

非 cocoapods 的工程，将 `-workspace "${APP_NAME}.xcworkspace"` 换成 `-project ${APP_NAME}.xcodeproj` 即可。

新版本的 Xcode 有了新的构建系统，使用 `-UseModernBuildSystem=<value>` 来做新旧区分。

命令	说明
-workspace NAME	指定工作空间文件XXX.xcworkspace
-scheme NAME	指定构建工程名称
-configuration [Debug/Release]	选择Debug或者Release构建
-sdk NAME	指定编译时使用的SDK

构建 archive 包

Xcodebuild archive

```
echo "🚗🚗🚗 *** 正在 编译工程 For ${development_mode} 🚗🚗🚗"

xcworkspace=${project_path}/${APP_NAME}.xcworkspace
echo "acrhivie xcworkspace : ${xcworkspace}"

xcodebuild \
archive -workspace  ${xcworkspace} \
-scheme ${SCHEME_NAME} \
-configuration ${development_mode} \
-archivePath ${build_path}/${APP_NAME}.xcarchive \
-quiet


echo '✅ *** 编译完成 ***'
```

导出 IPA 包

```
security -v unlock-keychain -p "yourpassword"
```

```
xcodebuild -exportArchive -archivePath ${build_path}/${APP_NAME}.xcarchive \
-exportPath ${exportFilePath} \
-exportOptionsPlist ${exportOptionsPlist_path} \
-allowProvisioningUpdates \
-quiet
```

更新到Xcode9.0后，之前写的自动打包脚本不可用了。

需要添加 `-allowProvisioningUpdates`，获取访问钥匙串权限的关键所在，设置了这个字段就会在打包过程弹框请求获取钥匙串内容权限。

exportOptionsPlist 设置

特别说明的是，exportOptionsPlist 一定要检查,不同的环境和分发平台要选择对。

最简单方式，就是调好需要的环境后，直接手动 archive ,export 出来，使用产物里的 exportOptionsPlist 文件。

Key	Type	Value
▼ Root	Dictionary	(9 items)
destination	String	export
method	String	app-store
▼ provisioningProfiles	Dictionary	(1 item)
com.[redacted]n.buding	String	[redacted]_pro
signingCertificate	String	Apple Distribution
signingStyle	String	manual
stripSwiftSymbols	Boolean	YES
teamID	String	8[redacted]
uploadBitcode	Boolean	YES
uploadSymbols	Boolean	YES

检查 ipa

检查对应路径是否有 **.ipa 文件:

```
if [ -e ${exportFilePath}/${APP_NAME}.ipa ]; then
echo "✅ *** .ipa文件已导出 ***"
```

```
else
echo "❌ *** 创建.ipa文件失败 ***"
exit 1
fi

echo '📦  *** 打包完成 ***'
```

构建完成

上传分发平台

这里分为 蒲公英,fir,appstore 三个平台，上传 IPA.

> 如果为 appstore, 则多出一个 git tag 的相关操作，标记上当前版本的提交，方便需要时直接回退代码进行查看。

下面使用的三个上传命令，最好先提前在机器上实验可以正常用再构建。

```
if [ "$BetaPlatform" = "pgyer" ];then

      echo "🚀 上传蒲公英 +++++++++++++++upload+++++++++++++"
      #User Key
      uKey="User Key"
      #API Key
      apiKey="API Key"
      #执行上传至蒲公英的命令
      curl -F "file=@${IPA_PATH}" -F "uKey=${uKey}" -F "_api_key=${apiKey}" -F "buildPasswo
      echo "✅ Finsh - 蒲公英上传完毕"
fi


if [ "$BetaPlatform" = "fir" ];then
      echo "🚀 上传Fir +++++++++++++++upload+++++++++++++"

      fir p ${IPA_PATH} -T your_token

      echo "✅ Finsh - Fir 上传完毕"
fi
```

```
if [ "$BetaPlatform" = "appstore" ];then

   echo "🏠 ------------appstore xcrun 上传到 appstore  ----------"

   xcrun altool --upload-app -f ${IPA_PATH} -u your_account -p your_app_password  --verbose

   echo "📝 ------------appstore 增加 Git Tag ----------"

    echo "--------- 当前 Tag -----------"
    git tag

    echo "--------- 打 Tag ------------"
    GitTag=V${bundleShortVersion}_${bundleVersion}

    git tag -a ${GitTag} -m "Tag:${GitTag} "
    echo "Tag ${GitTag}"

   #推送标签
    git push origin ${GitTag}

    echo "✅ ---------- Git Tag 推送完毕 ----------"
fi
```

符号表处理

上传 bugly

```
echo " 📦 ------ 开始符号表 相关工作 ------"

echo " © ----- 上传符号表 ------- ©"

if [ "$BetaPlatform" = "appstore" ];then
   echo "🚀  Bugly  正式版本符号表"
   buglyID= your_product_buglyID
   buglyKey= your_product_buglyKey

else
   echo "🚀  Bugly  测试版本符号表"
   buglyID= your_dev_buglyID
   buglyKey=your_dev_buglyKey
fi

dSYMPath=$exportFilePath/${APP_NAME}.xcarchive/dSYMs/
```

```
echo "----- 开始上传符号表 ---------- "
java -jar buglySymboliOS.jar \
-i ${dSYMPath}/${APP_NAME}.app.dSYM \
-u -id ${buglyID} \
-key  ${buglyKey} \
-package ${BundleID} \
-version ${bundleShortVersion}

echo "✅ ---------- 上传符号表完毕 ------ ✅ "
```

归档产物

进行完所有操作后，对于产物做一次保存，需要时可以用上。

压缩

首先将文件压缩

```


echo "📦 ---------- 压缩文件 ------ 📦 "

#打开目录
cd $exportFilePath
zip -r ./${JOB_NAME}_${BUILD_NUMBER}.zip ./*

#清理文件 **.xcarchive
rm -rf ${APP_NAME}.xcarchive
```

上传 FTP 服务器

通过 FTP 插件，把 zip 文件上传到归档的路径下

产物清理

删除 IPA 等文件，注意的是，当状态为 `success` 才清理，避免有时上传出问题，可以进行手动上传。

构建描述

设置构建描述

进行通知

完成后，企业微信 webhook 机器人推送，效果如下：

这里设置成可选项，避免频繁打扰其它同事。脚本如下:

```
if [ "${BotPush}" = "YES" ];then

version=$(ruby ./ToolChain/ruby/dy_build_version.rb ${Mode})
downUrl="pgyer url"

if [ "$BetaPlatform" = "fir" ];then
  downUrl="fir url"
fi

#群里机器人地址
ROBOT=https://qyapi.weixin.qq.com/cgi-bin/webhook/send?key=yourkey

curl ''${ROBOT}'' \
   -H 'Content-Type: application/json' \
   -d '
   {
   "msgtype": "markdown",
   "markdown": {
            "content": "### iOS 构建 \n版本 <font color=\"warning\">'${version}'</font>
```

```
fi
```

参考文章

- Xcode10 新特性
- Xcodebuild命令使用
- Xcodebuild命令官方说明
- 使用 Xcodebuild 命令进行自动化打包
- Xcode自动打包那些事

> **关注下面的标签，发现更多相似文章**

iOS

ZenonHuang Lv2 (作者) 野生程序猿 @ 广...

可以的，其实所有流程，跟语言不是强相关的。原来有的宏设置，你可以不用。纯Swift我也还没试过，有问题可以一起交流

5天前

哈哈哈士奇

Swift我试过实现简单的打包SDK，计算SDK增量，切换Xcode版本的这类小工具，中间没有接入Jenkins。但是实现的时候遇到一个问题是命令行的权限问题，有些cmd需要Mac用户授权，那就需要用到AppleScript或者一个原生的什么框架，可以在执行cmd时触发等待用户授权，我觉得用AppleScript会比较简单

5天前

天空中的球球

真详细，👍

6天前 回复

ZenonHuang Lv2 (作者) 野生程序猿 @ 广...

谢谢啦，也是不断完善做出来的。实际还是有坑要踩的，有问题多交流 🤝

6天前

相关推荐

橘子不酸丶 · 2天前 · iOS

iOS优化篇之App启动时间优化

32 1

老司机技术周报 · 1天前 · iOS / SwiftUI

【WWDC20】10037 - SwiftUI 中的 App 要领

8

FengyunSky · 4天前 · iOS

一文读懂iOS线程调用栈原理

12

JeremyHuang37 · 1天前 · iOS

【译】自定义 Collection View Layout -- 一个简单的模板

2

首页 探索掘金

94 109

Chouee · 3天前 · iOS

造轮子 - UITableView字母索引条

7 1

路过看风景 · 2天前 · iOS

CocoaPods原理 及 组件化

3

阿里巴巴淘系技术 · 5天前 · iOS

Apple Widget：下一个顶级流量入口?

9

路过看风景 · 1天前 · iOS

iOS事件处理 UIResponder

2