## 10+版本迭代

1. 简单介绍一下这个项目？

Royole是做柔性传感器（FS）和柔性显示器（FD），柔记手写本就是基于柔性传感器（FS，电容式）的手写设备，将写在纸上的笔迹转换为轨迹坐标数据，柔记APP通过蓝牙BLE与手写本连接，将传输到手机上轨迹坐标数据还原为用户笔记。还有笔记分享，转换图片、微视频，回放，笔记拆分，合并，笔记搜索、笔记本管理、安全连接、更换画笔，笔记转文本、笔记云同步等功能。

柔记APP公司唯一个互联网化的APP，使用的技术栈跟主流的互联网APP相同，上架了国内各大主流应用市场以及谷歌市场。大概进行了10多个版本迭代，开发流程相对规范，团队大小适中，也有一定的用户量。

代码量大概10万行，java文件

1. 项目成员有多少人？你在这个项目担当什么角色？

项目是从17年10月开始的，我是18年4月加入的。前期项目快速迭代，多的时候6个人左右，目前迭代速度变慢，两个人在小功能的开发与维护。

本人在项目中的角色是安卓端开发负责人，主要工作包括需求评审、工作量评估、技术评审、编码实现、代码审核、版本发布等工作。

1. 在这个项目中又遇到什么让你印象深刻的事情？

1）柔派适配，风险评估与处理；

在一个版本迭代过程中，老板希望在一个展会上使用柔派手机来展示柔记APP。 柔派手机是公司的一款折叠手机，因为屏幕可以折叠，所以屏幕有三种分辨率：大屏、小屏（主屏、辅屏）。在App运行过程中折叠手机屏幕，屏幕分辨率变化，Activity会销毁重建，一些用户操作就会被打断。所以必须对柔派进行适配。当时就把柔派手机的适配工作交给一个同事去做。当时觉得就是屏幕适配，所以也不觉得会有什么风险。中间有多次问那位同事，是否有风险，回复说没有问题，直到发版前一周，发现不断有适配导致的问题冒出来。才发现问题不是那么简单，在折叠过程中，由于屏幕分辨率会发生变化，导致Activity会销毁重建，在开发过程中并未考虑Activity销毁重建的情况，并未对运行时数据进行保存，导致一些运行时的状态会丢失；同时有一些用户操作是不允许会被打断，比如绘制过程中，折叠一下，所有的新绘制的笔记都丢失了。整个App总共20个页面左右，这样引发了很多的问题。当时两个开发人员，在没有现成解决方案的情况下，一周时间无论如何都无法完美适配。所以发邮件给项目负责人，告知风险，寻找解决方案。

解决方案：

1. 短期（一周内）适配常用的界面；
2. 适配方法：简单的页面，可打断的页面，走销毁重建的逻辑，在onSaveInstanceState中保存状态数据，在onRestoreInstanceState中恢复状态数据；复杂的页面，有很多中间状态的页面不销毁重建，在onConfigurationChanged回调中对视图元素的布局参数进行调整。

MVP+ViewModel框架，将业务逻辑与运行时数据放到ViewModel中，由于ViewModel在Activity因分辨率变化而导致的销毁重建并不会销毁，所以用ViewModel作为Presenter，保存运行时数据。再配合LiveData可以在Activity销毁重建之后恢复销毁前的状态。

2）笔记本同步，流程设计与实现；

V2.5.0 版本为了适配最新的柔记二代手写本硬件的外观设计。将APP进行了重构，更加强调笔记本的概念。增加了笔记本的管理，如：设置默认笔记本、设置常用笔记本、对笔记本进行排序、修改笔记本名称、封面等。对默认笔记本的同步流程进行梳理，绘制流程图，提供给IOS、服务器端编码实现。这部分的难点在于：默认笔记本多端修改冲突、默认笔记本切换等。

3）手写本连接库，过程改进；

手写本与APP之间是通过蓝牙BLE连接的，这部分的逻辑相对独立，就专门封装了一个SDK，专门处理手写本与APP之间的连接通信。手写本与APP之间的通信包括，扫描手写本、获取存储信息、获取电量信息、分页获取存储数据、删除数据、获取固件版本号、设置/获取手写本的模式（实时or离线）、安全连接逻辑等。这个连接库是一个独立的工程，因为这个连接库在供内部使用之外，还提供给外部如方正公司使用。一开始APP使用这个依赖库的方式是，直接编译出jar，手动拷贝到柔记APP工程中。但这样手动拷贝容易出错，中间就有一次开发人员修改了连接库，但是忘记了拷贝jar到APP工程中，导致出错。后面将依赖库发布到Maven私服上，APP工程通过maven的方式来引用。

4）柔记项目Https安全传输总结；

见 **柔记项目Https安全传输总结.ppt**

1. 项目是从零开始做的吗，你是什么时候加入到这个项目的？

项目是从17年10月开始的，我是18年4月加入的。

1. APP的日活，用户数等有多少

## 组件化

1. 为什么要组件化？
2. 业务模块之间耦合度太高，项目难以扩展与维护；
3. 任何一个改动都要编译整个项目，影响开发效率；
4. 组件化有哪些方案？

基于ARouter的组件化方案：路由+接口下沉

基于ActivityRouter的组件化方案

得到DDComponentForAndroid

CC、ModularizationArchitecture

聚美组件化方案，基于聚美Router

1. 组件化主要关注点？

a.业务解耦，识别业务边界，整理业务之间的通信，通过路由来跳转和暴露服务；

b.用gradle的属性来切换组件模式还是集成模式。

c.服务接口下沉，暴露组件服务。

1. 你是如何实现组件化的？

识别业务边界，拆分组件，组件间的界面跳转通过路由实现

组件的对外服务接口下沉到依赖库

实体下沉到依赖库

一些通用的工具类事件总线等下沉到依赖库；

路由的实现原理？？？

1. 你这个项目分了几个模块？

分成了5个：登录、笔记书写、手写识别、商城Web、笔记分享；

## 引入MVP+ViewModel架构；

1. 介绍一下MVP架构，与MVC、MVVM的区别与联系
2. 为什么要引入这个架构？
3. 跟普通的MVP有什么不同的地方？
4. 说说ViewModel是怎么实现应用数据保存的？

## UI适配 pt适配

1. 介绍一下怎么适配的？

一开始是采用百分比适配的方法，百分比适配的原理：

1. 将UI设计稿（1080X1920）缩放到目标屏幕，宽高上每个刻度对应的像素值生成到lay\_x.xml和lay\_y.xml。
2. 系统根据屏幕分辨率选择对应的lay\_x.xml和lay\_y.xml的尺寸值。

问题：

1. 当目标屏幕的宽高比与UI稿的宽高比不一致时，x、y方向缩放系数不一致，界面会出现变形。
2. 新出一种尺寸的屏幕的时候，需要制作一份对应的尺寸资源，否则使用默认的尺寸资源。

今日头条适配方案：根据设计稿的尺寸与目标屏幕的尺寸，计算缩放系数，修改DisplayMatrix的density、densityDpi、xdpi等变量，以实现一套UI尺寸适配所有机型屏幕尺寸。

今日头条适配方案特点：

* x、y方向的缩放系数都是一致的，不存在界面变形的问题
* 只需一套尺寸资源
* 适用于新项目

1. 官方适配方法？

为不同的屏幕分辨率、屏幕密度生成不同尺寸文件，放在不同的value文件夹下。在不同的分辨率的手机上，使用不同资源目录下的资源。

1. 百分比适配，今日头条适配方案对比，开源库AndroidAutoSize

百分比适配：1）需要为每一种屏幕分辨率生成一种尺寸资源；2）当屏幕分辨率与设计稿的尺寸宽高比例不一致时，图片会被拉伸变形；3）每新增一个屏幕分辨率的设备时，需要重新生成尺寸资源；

## 引入Checkstyle，编码规范

1. 为什么要引入Checkstyle？

为了执行统一的编码规范，配合gitlab，可以让团队成员强制执行编码规范检查，不符合编码规范的代码，不可以合并到仓库中。Checkstyle可以自定义编码规范扫描规则。

1. 如何执行code review？

分为两步：

1. 代码提交到gitlab服务器，会自动触发编码规范扫描任务，如果发现有不符合编码规范的代码，不允许合并入仓库。
2. 代码通过自动化编码规范扫描任务之后，

代码提交人员需指定审核人员进行审核，审核通过之后代码才允许合并入仓库。

Gitlab可以指定代码团队成员，如代码维护者、开发者等，可以给不同成员分配不同的权限，如维护者才有权限合并代码。

1. Gradle插件；

自定义编码规范插件。可以在AndroidStudio的gradle task中找到并执行，当然也是可以在命令终端中执行。

1. gitlab、git工具的使用；

gitlab是一个代码仓库管理系统，类似github，以git为代码管理工具为基础搭建起来的web服务，它集成了issue跟踪、代码合并与审核、持续构建与集成等功能。

代码上传到gitlab服务器会触发构建活动，在项目根目录的yaml文件中，编写构建活动脚本。

## 接入LeakCanary Refwatcher 检测ViewModel内存泄漏。

## 引入Tinker热修复，Bugly

1. 已经弃用、谷歌官方已经禁止；
2. 主流热修复方案原理及对比；
3. Tinker热修复方案原理；

BSDiff生成增量更新文件，APP启动，在onAttachBaseContext方法中，检测到增量更新文件，会合成一个完整的class.dex文件。

## 笔记绘制

## 蓝牙通信

## 笔记云同步

## 生成笔记视频

## 团队角色

1. 负责柔记项目，还有参与一些小项目：永兴元定制版柔记、练字系统、签名系统、会议系统、OneUI、一拖多项目、残影修复项目、电梯项目等。
2. 技术培训
3. 问题解决
4. Code Review