Software Engineering and Applications 软件工程与应用, 2019, 8(6), 379-389
Published Online December 2019 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2019.86047

Hans汉斯

Componentization Application and Practice Based on Android Development

Xiaoan Bao, Qi Wei, Na Zhang, Lu Xu

School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou Zhejiang
Email: weiqi_j@163.com

Received: Nov. 29th, 2019; accepted: Dec. 18th, 2019; published: Dec. 25th, 2019

Abstract

Aiming at the current problems of Android application development, such as conflicts in collaborative development, high coupling of business code, low development efficiency, and large application volume, this article proposes a componentized development solution. The project is divided into multiple business components through the IDE; each component is independent of each other, and the development mode is adjusted to a component mode or an integration mode using Gradle tools. In component mode, compiling and debugging are based on a single component; in integrated mode, the final APP is generated with all components integrated. However, the intercomponent communication routing framework commonly used in Android development is complex in function and large in volume, and the independence between components easily results in resource redundancy, which will increase the volume of the application and increase the cost of installation traffic. Therefore, a new lightweight inter-component communication routing framework ERouter is proposed, which combines resource compression and selection of lightweight third-party libraries to achieve application optimization. This article takes the intelligent dormitory manager platform as an example. The above scheme effectively solves the problems including many collaborative development conflicts, high code coupling, slow project engineering compilation, and large application volume in application development.

Keywords

Android Application Development, Componentized Development, ERouter, Application Optimization, Intelligent Dormitory Manager Platform

基于Android开发的组件化应用与实践

包晓安，韦　奇，张　娜，徐　璐

浙江理工大学信息学院，浙江　杭州
Email: weiqi_j@163.com

文章引用: 包晓安,韦奇,张娜,徐璐. 基于 Android 开发的组件化应用与实践[J]. 软件工程与应用, 2019, 8(6): 379-389.
DOI: 10.12677/sea.2019.86047

收稿日期：2019年11月29日；录用日期：2019年12月18日；发布日期：2019年12月25日

摘　要

针对当前Android应用开发存在协作开发冲突多、业务代码耦合高、开发效率低以及应用体量大等问题，本文提出一种组件化开发方案。通过IDE将项目划分成多个业务组件，各个组件间相互独立，并使用Gradle工具将开发模式调整为组件模式或集成模式。在组件模式下，基于单个组件进行编译调试；在集成模式下，整合所有组件生成最终APP。但是Android开发中常用的组件间通信路由框架功能复杂、体量较大，而且组件间相互独立易造成资源冗余，这会增加应用的体量，提高安装的流量成本。因此，提出一种新的轻量级组件间通信路由框架ERouter并结合资源压缩、选取轻量级第三方库等操作实现应用优化。本文以智慧宿管平台为例，以上方案有效地解决了应用开发中协作开发冲突多、代码耦合高、项目工程编译慢以及应用体量大等问题。

关键词

Android应用开发，组件化开发，ERouter，应用优化，智慧宿管平台

1. 引言

基于 Android 开发的移动端 APP 已被广泛应用在各个领域，不断改变着人们的工作生活方式。在互联网+的时代[1] [2]，Android 应用开发技术[3] [4] [5]不断革新，Android 应用需要更高的开发效率和更小的安装成本才能保持长久的生命力。传统开发过程中，项目模块都位于一个主工程目录下，每个模块之间的隔离不严格，代码调用相对随意。与此同时，开发独立的功能模块时，仍需要在主工程下进行编译调试。传统开发适用于 APP 功能需求很少的应用，但是当项目体量足够大或是后续业务会逐渐增长，就会出现模块间耦合过高、不易维护、多人协作开发提交代码易产生冲突等问题。若稍微改动一处代码，整个工程需要重新进行编译运行，由于项目工程体量大，编译需要等待的时间将会十分漫长，最终开发的 APP 所需下载的流量成本也会很高，不利于应用的推广[6]。因此如何提高开发效率、减少 APP 安装成本是 Android 应用开发的首要问题。

周宇等人[7]通过 Eclipse 工具进行 Android 应用的开发，但是由于 Eclipse 开发工具的局限性只能使用传统的项目开发方式，当项目业务需求增大，项目内部结构的耦合度便会提高，编译调试时间也将显著增加，对于多人协作开发的中大型项目，开发效率会大大降低。邹绍武等人[8]为减少 Android 应用开发的体量问题，对图片资源的压缩进行了相关研究。殷涛等人[9]利用组件化思想将业务模块按照功能进行划分，减少了业务模块间的耦合，但是未结合实际的应用开发场景，对于多人合作开发的效率提升没有显著地提升，也没有考虑到组件间通信路由框架体量大的问题。

综上存在的问题，本文提出一种组件化方案。通过对实际项目进行业务组件划分以提高多人协作开发效率，并利用自定义轻量级组件间通信路由框架 ERouter 结合资源压缩、选取轻量级第三方库等操作[10] [11]对最终业务组件整合后的 APP 进行深度优化，从而有效地解决 Android 开发中存在的这些问题。本

文以智慧宿管平台开发为例，详细介绍相关技术应用。

2. 移动端设计原理

在移动端应用开发中，移动端设计[12]尤为重要，随着项目规模逐渐增加、项目需求与版本不断迭代，整个项目工程便会变得愈加庞大。在开发初期就应具有合理的分析设计，才能应对后期的项目维护与版本迭代。因此，本文提出基于组件化思想[13] [14]进行项目开发来解决以上问题。如下将对组件化原理进行分析。

组件化开发原理

在 Android 开发中主要是通过 Android Studio 编译工具来实现组件化，每个要划分的业务模块新建成 Module 并通过编译工具中的 Gradle 编译插件设置是否能够独立编译运行。在不同的项目场景下，组件划分的方式也不同，目前常见的有从功能进行划分和从业务进行划分的两种组件划分方式。

从功能上划分组件常用于单业务类型项目的开发，如一些工具类的 Android 应用。通常会将项目划分为用户模块、业务逻辑模块和通信模块。用户模块中具有用户注册、登录、注销和个人信息管理等功能。业务逻辑模块负责具体的业务操作，如查询展示等操作。通信模块负责模块间的交互以及模块的网络通信等。

从业务上划分组件常用于多业务类型项目的开发，其中多业务是指 APP 中的业务相互独立，交集相对很少。在中大型项目开发中往往需要增加额外的业务，由于划分后业务组件间相互独立，添加新的业务相对便捷。以业务进行组件划分，方便业务复用与业务扩展，这中组件划分方式可有效的提高协作开发的效率。

经过组件划分后，组件化开发框图如图 1 所示：

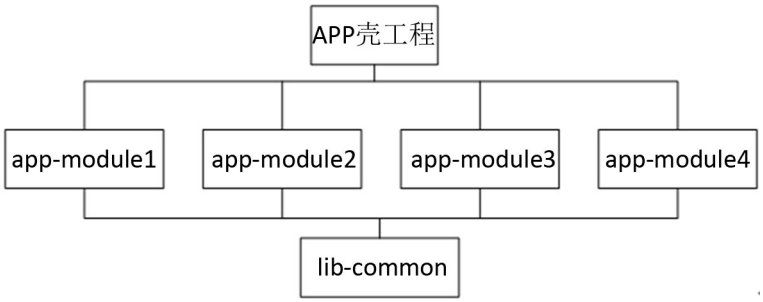

Figure 1. Component development block diagram
图 1. 组件化开发框图

在实际项目开发中，新建业务功能模块 Module 时以“app-”为前缀命名，代表该业务功能模块可独立编译运行，而新建核心模块以“lib-”为前缀命名，代表业务功能模块可依赖该核心模块。这种命名形式简洁明了，有利于项目开发维护。其中业务功能模块是具体的业务功能，核心模块为业务功能模块提供基本的数据服务和功能服务，即业务功能模块需要依赖核心模块。依赖关系需要在业务模块对应的 Gradle 文件中通过“compile project”设置。业务模块的独立运行需要在 gradle.properties 文件中添加标识 moduling，并在相应业务模块的 Gradle 文件中判断标识，当 moduling 为 true，业务功能模块为 application 可单独运行，反之为 library 不可单独运行。

综上，Gradle 插件工具实现了项目编译的动态改变。在开发阶段，每个业务功能模块独立运行，每个开发人员负责相应的业务功能模块，做到更深层次的解耦，解决了协作开发的冲突，提高了开发效率。

在调试阶段，所有业务功能模块整合成最终的移动应用。通过组件化开发方式，项目进行合理的业务划分，真正实现了高内聚、低耦合，缩短了开发周期。

3. 移动端设计与分析

3.1. 功能分析

本文以智慧宿舍平台为例，APP 需实现五个端口，分别为学生端、学工端、后勤端、维修端以及家长端，基本包含了学校管理的所有相关人员，每个端口具有鲜明的业务功能，具体功能框图如图 2 所示：

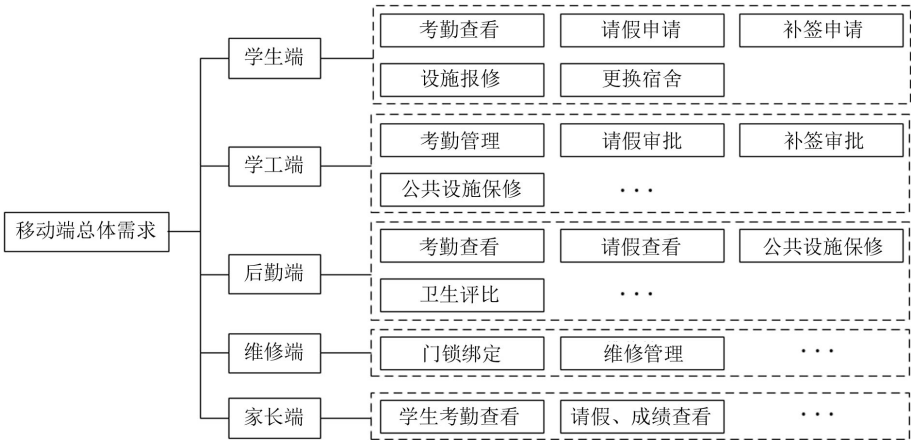

Figure 2. Functional block diagram of smart dormitory platform
图 2. 智慧宿舍平台功能框图

由于用户需求不断增长，项目内部结构逐渐复杂，应用开发应进行合理的项目设计，从而提高开发效率，降低后期维护和项目迭代的成本。

3.2. 组件化设计与应用

根据本文智慧宿舍平台功能分析，每个端口业务彼此分离明确，业务延伸性较大，选取组件化开发方式代替传统开发方式是项目开发的必然选择。而且各端口间相互独立，因此采用从业务上划分组件方式得到具体组件化流程框图，如图 3 所示：

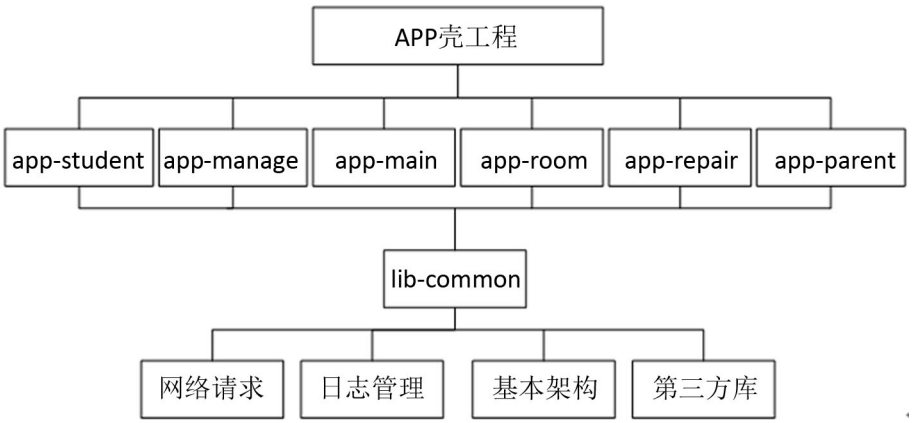

Figure 3. Componentized process block diagram
图 3. 组件化流程框图

其中 app-student、app-manage、app-room、app-repair、app-parent 分别代表业务模块学生端、学工端、后勤端、维修端、家长端，app-main 模块为启动页功能模块。

通过设置 gradle.properties 文件中的全局变量 moduling 为 true 或 false，业务模块中对这个标识做出相应改变，从而改变项目模式为组件模式或集成模式，如学工端模块 Gradle 文件代码。

```
if (moduling.toBoolean()) {
    apply plugin: 'com.android.application'
} else {
    apply plugin: 'com.android.library'
}
sourceSets {
    main {
        if (moduling.toBoolean()) {
            manifest.srcFile 'src/main/debug/AndroidManifest.xml'
            println '[Module-Manage]: Appling Application...'
        } else {
            manifest.srcFile 'src/main/AndroidManifest.xml'
            // exclude debug java class in release mode
            java {
            exclude 'debug/**'
            }
            println '[Module-Manage]: Appling Library...'
        }
    }
}
```

从集成模式到组件模式具体在项目中的变化如图 4 示：

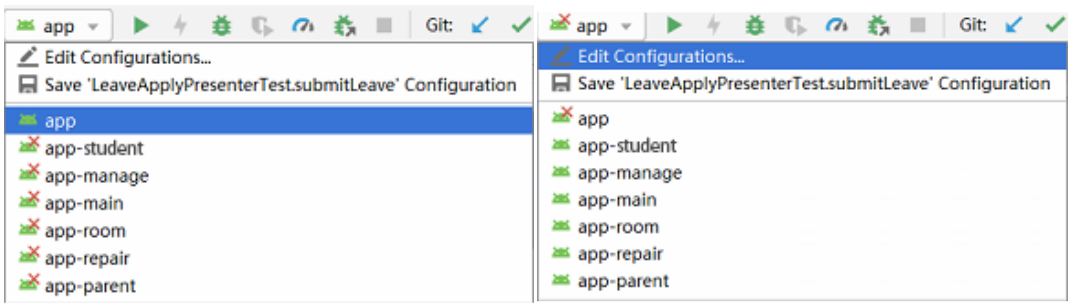

Figure 4. Integration mode and component mode diagram
图 4. 集成模式与组件模式图

处于组件模式下，每个业务端口都可独立运行，如图 4 中右侧图相应业务端口，点击运行即可编译调试。

采用组件化思想，在多人协作开发智慧宿舍平台 APP 中，每人负责一个业务组件。由于组件间相互独立，因此消除了组件模块间的耦合，大大减少了开发冲突，整合所有组件，就形成了最终的 APP，从而缩短了整个项目的任务周期。

3.3. 组件化应用优化

目前，常用的组件化通信路由框架是阿里的 ARouter 框架，但是该框架功能复杂、体量较大，在只需要简单通信的项目开发中并不适用，而且各业务组件相互独立，在开发过程中容易造成代码冗余等问题，这无疑增加了应用的体量。为解决此问题，本文以智慧宿管平台 APP 为例进行相应优化，并从 APK 的构成和具体的优化设计进行研究分析。

3.3.1. APK 构成

APK 作为项目完成后安装运行的枢纽，其生成过程是应用程序性能优化最终的关键步骤，对于生成 APK 资源打包过程如图 5 所示：

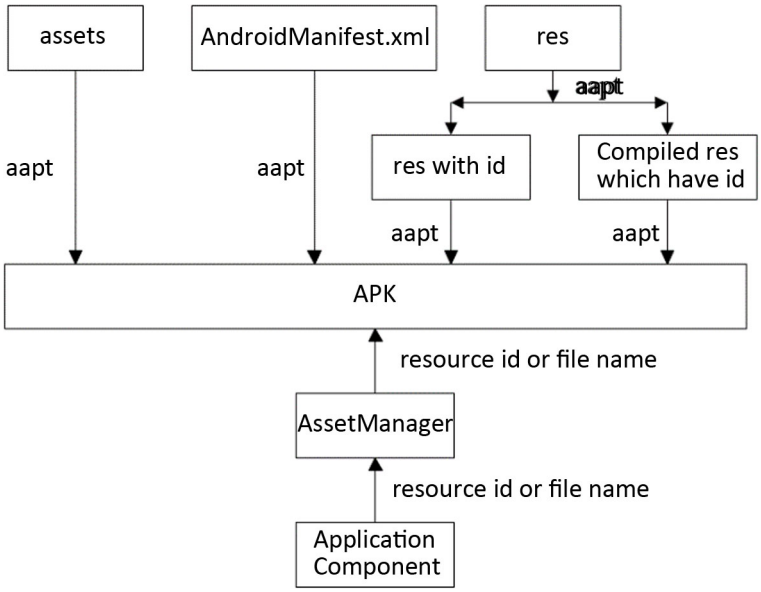

Figure 5. APK formation diagram
图 5. APK 形成示意图

针对图 5 中 APK 的构成分析具体的资源变化情况，在 assets 和 res 文件目录下的资源会直接打包进 APK 中，其余文件下的资源都会被编译或者被处理成二进制文件，通过 Android Studio 编译出发布版本 APK 文件进行解压，由此分析解压后的文件结构，从而找寻可进行 APK 瘦身的方法，详情见表 1：

Table 1. APK parse file table
表 1. APK 解析文件表

文件名	文件内容
Lib	包含一些引入的第三方库，so 文件等资源主要是 native 层的
Res	包含了 APK 的签名文件后缀为.S 与.RSA 以及后缀为.MF 的清单文件
META-INF	包含了 APK 的签名文件后缀为.S 与.RSA 以及后缀为.MF 的清单文件
Resources.arsc	包含的是编译后的资源文件，主要有 res/values/文件夹下的全部资源，如语言字符文件、样式文件以及.arsc 文件下的所有资源路径信息，如 layout 文件、图片资源文件等
Classes.dex	包含通 ART 虚拟机编译后的所有 java 类文件，经压缩优化转化为文件格式为.dex 的形式
AndroidManifest.xml	作为 Android 应用的清单文件，包含着应用程序的名称内容、可访问手机系统的权限清单、应用的版本号以及 build.gradle 引用的第三方库文件等。其存储形式为二进制 XML 格式
Assets	包含一些视频文件，db 文件等资源

由表 1 可知，在项目开发过程中，可以对 res 目录下的图片资源、classes.dex 以及 lib 下的库所对应的项目目录进行优化。

3.3.2. APK 优化设计

通过分析 APK 结构，本文提出一种 APK 优化的方案。对于图片资源，在开发中将应用程序内显示的 PNG 或 JPG 图片转化为 Webp 格式，Webp 格式的图片保留了 JPG 和 PNG 的优点，还能提供更好的有损压缩效果；对于一些简单的图片，完全可以通过 drawable 中的 XML 进行绘制；对于 classes.dex 文件，由于其源头是 Java 类文件，项目开发中应避免类的重复定义以及无用类的生成，如定义工具类应添加注释，放在 Common 目录下，这在多人开发项目尤为重要；对于库文件传统做法就是更换体量更小的库，但是很多优秀的库不易替代，或是用来替代的库存在内存泄漏[15] [16] [17]。

目前，组件化开发中组件间的通信方式主要通过阿里的 ARouter 路由框架，但是在实际应用场景下，组件间只需要进行简单的通信。此时，ARouter 框架体量大的缺点就暴露无疑。因此本文提出一种组件间通信路由框架 ERouter，以实现更轻量级的组件间通信，从而进一步降低应用 APK 的内存。

对于 ERouter 路由设计描述如下：

步骤一：新建消息中转类 ERouter.java 用于处理业务模块间的消息跳转，内部维护 HashMap 来存储消息路由，并通过该类注册路由路径；

步骤二：对于路径类 EAction.java，用于实现消息路由的建立和发送消息到具体指定的路由信道，每个业务模块的 Activity 都应在相应的 Action 中注册自己的路径；

步骤三：新建消息载体类 ERouterRequest.java，其包括消息路径和消息内容，即发送消息实体；

步骤四：当路由消息发送后需要收到反馈，从而使整个链路完整，这里注册 ERouterResponse.java 里面有相应的状态码以及实体信息；

步骤五：初始化路由，通过 ERouter 注册所有需要跳转的业务模块路径，并通过抽象路径类 EAction 去实现具体的业务模块跳转路径；

步骤六：通过 ERouter 的 sendMessage 方法进行消息发送，消息实体通过 ERouterRequest 进行构建，最终通过 ERouterResponse 完成通信。

核心注册代码如下：

```
private void initRouter() {
    BLog.d("[App]: ERouter initing...");
    ERouter.register(MainAction.NAME, new MainAction());
    ERouter.register(MineAction.NAME, new MineAction());
    ERouter.register(MessageAction.NAME, new MessageAction());
    ERouter.register(RoomAction.NAME, new RoomAction());
    ERouter.register(ManageAction.NAME, new ManageAction());
    ERouter.register(RepairAction.NAME, new RepairAction());
    ERouter.register(ParentAction.NAME, new ParentAction());
    ERouter.register(TerminalAction.NAME, new TerminalAction());
}
```

业务模块路由跳转代码如下：

```
ERouterResres=ERouter.push(getContext(),ERouterReq.build().action("repair").path("repair/main/clazz"));
Intent intent = new Intent(getActivity(), (Class) res.data());
```

```
intent.putExtra(ManageActivity.ROOM_SELECTION, "selection");
getActivity().startActivityForResult(intent,ManageActivity.REQUEST_ROOM_SELECTION);
```

4. 实验研究与分析

为校验 ERouter 路由框架的效果以及最终组件化应用瘦身的程度，本文通过集成目前最流行准确的 Leakcanary 框架来检测 ERouter 路由在业务模块跳转中是否存在内存泄漏，并通过 Android Studio 自带的 Analyze APK 工具对 APK 瘦身优化前后进行分析。

实验环境如下：Windows 7 操作系统，Intel(R) Core(TM) i5-3230 CPU @ 2.60 GHa，8 GB 内存，编译环境 Android Studio3.2，测试机华为荣耀 9，6 GB 内存，测试如下：

4.1. 组件通信框架内存泄漏检测

本文通过智慧宿管移动端 APP 学工端入口进入考勤页面点击并跳转到班级详情页面，当点击具体学生时则会跳转到该学生的考勤页面，而学生考勤页面是学生端业务模块的内容，这就完成了一次业务模块间的路由跳转。

使用 LeakCanary 检测从学工端班级页面跳转到学生端该学生考勤页面，跳转详情如图 6 所示：

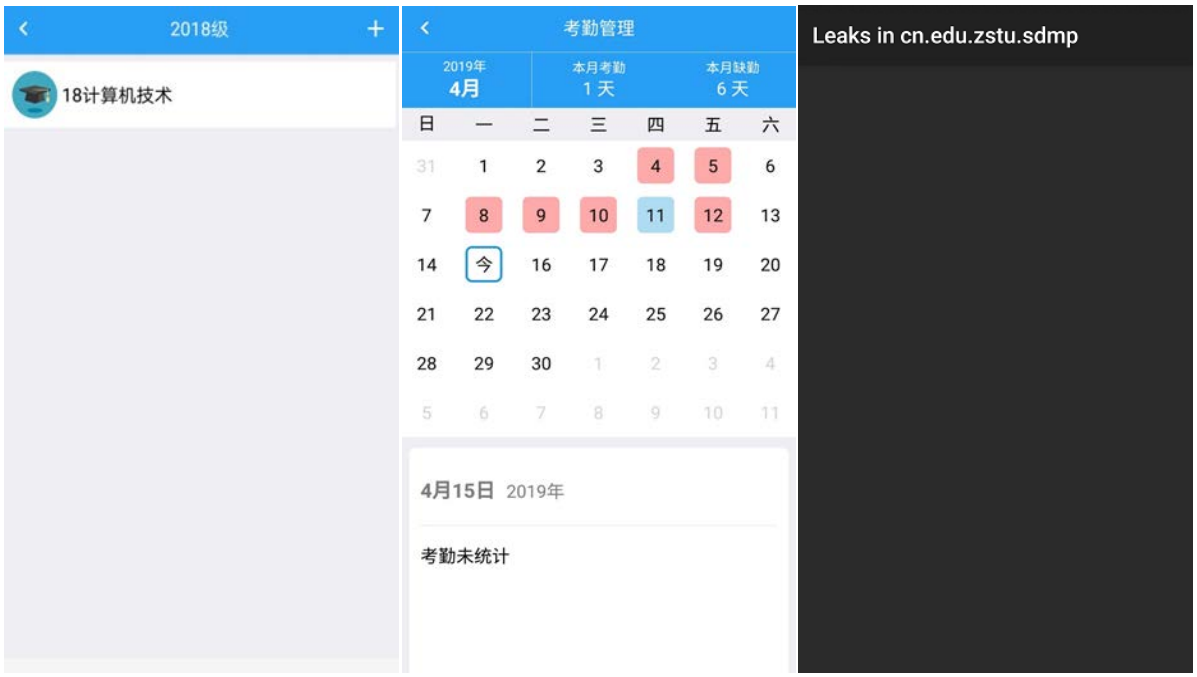

Figure 6. Route jump and memory leak detection diagram
图 6. 路由跳转及内存泄漏检测图

由图 6 可知，从学工端到学生端未检测到内存泄漏证明自定义 ERouter 路由框架符合开发要求。实验结果表明，本文提出的组件间通信路由框架 ERouter 完成了项目中对阿里 ARouter 路由框架的替换，从繁至简实现了开销成本的进一步减小。

4.2. 应用优化检测

通过轻量级组件间通信路由框架 ERouter 的替换以及相应的资源压缩等操作，得到两个移动端应用 APK 版本，分别为应用优化前与应用优化后。借助 Android studio 编译器对 release 版本的 APK 进行拆分，

使用自带的 Analyze APK 工具对优化前后的两个 APK 进行对比如下：

优化前如图 7 所示：

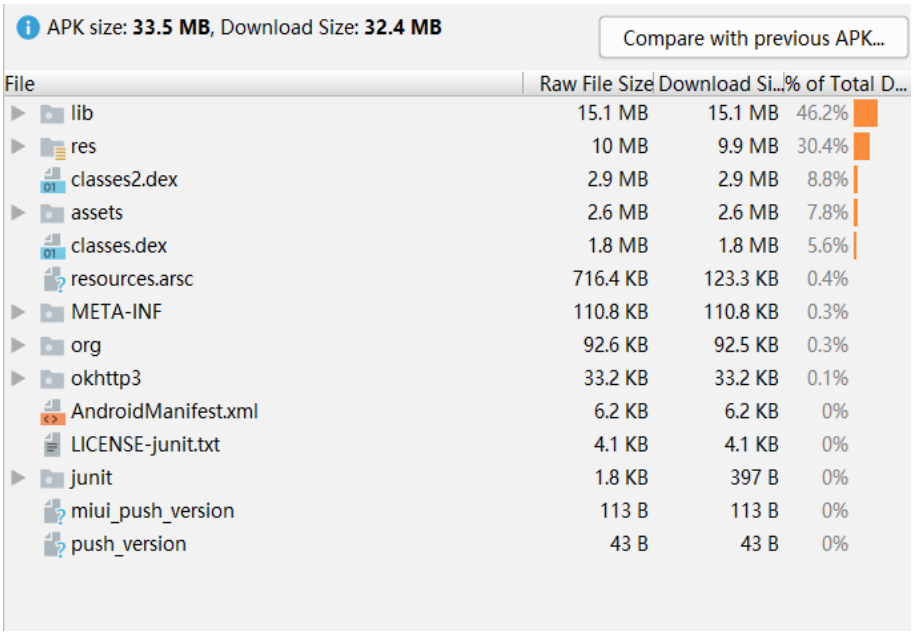

Figure 7. Analyze the file size of APK before applying optimization
图 7. 应用优化前 APK 解析文件大小图

优化后如图 8 所示：

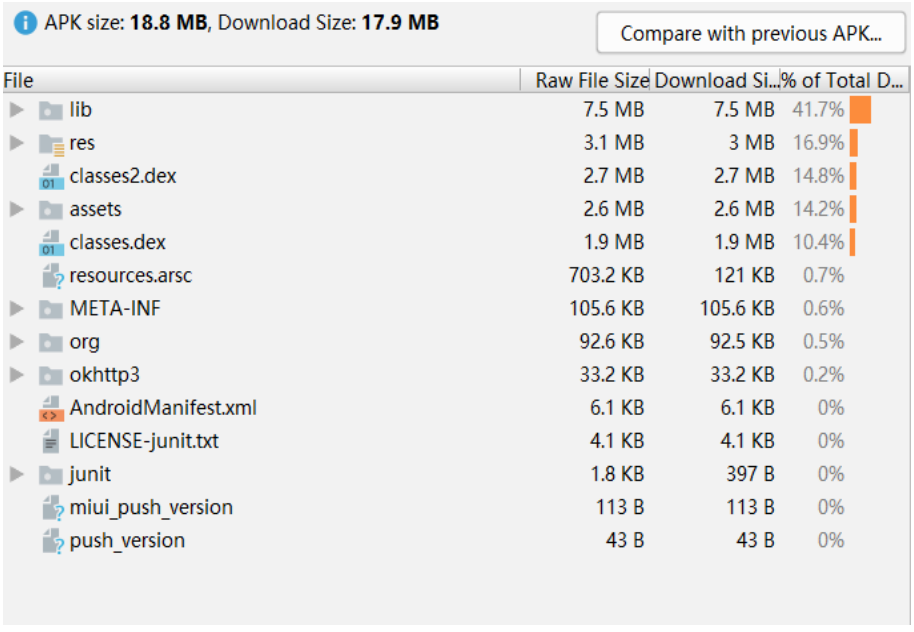

Figure 8. Analyze the file size of APK after applying optimization
图 8. 应用优化后 APK 解析文件大小图

由图 7、图 8 可观察到 res、lib 与 class.dex 文件的大小变化，实验结果见表 2：

Table 2. Comparison table before and after APK optimization
表 2. APK 优化前后对比表

文件名	瘦身优化前	瘦身优化后
Lib	15.1 MB	7.5 MB
Res	10 MB	3.1 MB
Class.dex	10 MB	4.6 MB

由以下压缩公式计算优化后的 APK 优化率：

$$\text{CompressedRatio} = \frac{\text{InitializedSize} - \text{CompressedSize}}{\text{InitializedSize}} \quad (1)$$

其中：InitializedSize 为应用优化前 APK 的大小；CompressedSize 为应用优化后 APK 的大小；CompressedRatio 为 APK 的优化率。

由压缩公式(1)计算可得应用优化的程度为 0.448。实验表明：本文提出的轻量级组件间通信路由框架 ERouter 结合资源压缩等操作有效地减小了应用的体量。

5. 结束语

本文介绍了一种基于 Android 开发的组件化方案，并将其应用在智慧宿管平台上。通过项目的需求分析，详细介绍了业务组件划分的过程，减小了组件间的耦合，方便了二次开发的功能重用，提高了多人协作开发效率。在项目中集成 LeakCanary 内存泄漏检测工具，验证了 APP 界面跳转时自定义组件间通信路由框架 ERouter 的可用性。并运用 Android Studio 自带的 Analyze APK 工具分析该轻量级框架结合资源压缩后智慧宿管应用体量的变化，通过该操作减小了组件化应用的体量，降低了用户的安装流量成本。在今后的工作中，将会继续对组件化方案中组件划分的方式、组件间通信框架以及组件化应用体量的优化进行更加深入的研究，从而更好地应对中大型项目的 Android 开发。

基金项目

本论文获得了浙江省重点研发计划项目(2020C03094)、浙江省自然科学基金青年基金(2020C03094)、浙江理工大学本科生科研创新计划重点项目(2019ZD-28)以及浙江理工大学本科生科研创新计划一般项目(2019YB-24)的支持。

参考文献

[1] 李平, 陈杰, 王杰. 基于互联网+的专家服务平台 APP 设计与实现[J]. 自动化技术与应用, 2019, 38(8): 53-58.

[2] 包晓安, 徐海, 张娜, 吴彪, 钱俊彦. 基于深度学习的语音识别模型及其在智能家居中的应用[J]. 浙江理工大学学报(自然科学版), 2019, 41(2): 217-223.

[3] 尹孟征. 基于 Android 的 APP 开发平台综述[J]. 通信电源技术, 2016, 33(4): 154-155.

[4] 曾健平, 邵艳洁. Android 系统架构及应用程序开发研究[J]. 微计算机信息, 2011, 27(9): 1-3.

[5] 赵静. Android 系统架构及应用程序开发研究[J]. 自动化与仪器仪表, 2017(1): 86-87+90.

[6] 夏羿, 徐振宇. 移动平台竞争: 开放策略与市场结构——引入可竞争市场的双边市场竞争模型[J/OL]. 经济与管理, 2019(6): 34-43.

[7] 周宇, 尹生强, 王冬青, 王英杰. 基于 Eclipse 和 Android 系统的 App 开发平台搭建研究[J]. 青岛大学学报(工程技术版), 2016, 31(3): 49-53.

[8] 邹绍武, 苏贵斌. Android 应用开发中图片压缩技术的研究应用[J]. 计算机技术与发展, 2015, 25(6): 106-109+113.

[9] 殷涛, 崔佳冬. 基于 Android 软件开发组件化研究[J]. 计算机应用与软件, 2019, 36(9): 16-20.

[10] 季通明, 鲍胜利. JPEG2000 图像压缩算法在 Android 平台的应用[J]. 计算机应用, 2017, 37(S2): 203-206.

[11] Giaime, G., Maurizio, P. and Daniele, D.G. (2012) Objective Assessment of the WebP Image Coding Algorithm. *Signal Processing*: *Image Communication*, **27**, 867-874. https://doi.org/10.1016/j.image.2012.01.011

[12] Tenev, V., Zhang, B. and Becker, M. (2016) Android Build Dependency Analysis. 2016 *IEEE* 24*th International Conference on Program Comprehension*, Austin, TX, 16-17 May 2016, 1-4.

[13] Buchgeher, G. and Weinreich, R. (2009) Tool Support for Component-Based Software Architectures. 2009 16*th Asia-Pacific Software Engineering Conference*, Penang, Malaysia, 1-3 December 2009, 127-134. https://doi.org/10.1109/APSEC.2009.67

[14] Mahdi, D., Ebert, J., Marvin, G. and Engels, G. (2018) Model-Integrating Development of Software Systems: A Flexible Component-Based Approach. *Software & Systems Modeling*, **18**, 2557-2586.

[15] Riganelli, O., Micucci, D. and Mariani, L. (2019) From Source Code to Test Cases: A Comprehensive Benchmark for Resource Leak Detection in Android Apps. *Software*: *Practice & Experience*, **49**, 540-548. https://doi.org/10.1002/spe.2672

[16] 刘洁瑞, 巫雪青, 严俊, 等. 针对 Android 资源泄漏的基准测试集的构造与评测[J]. 计算机应用, 2017, 37(4): 1129-1134.

[17] 何群芳, 时招军. 基于 Android 的内存泄漏与溢出研究[J]. 软件导刊, 2018, 17(2): 50-52.