本科毕业设计（论文）

基于 Android 的 Wi-Fi 音箱多媒体控制系统的设计与实现

Design and Implementation of Wi-Fi Speaker Multimedia Control System Based on Android

学　　院：软件学院

专　　业：软件工程

学生姓名：XXXX

学　　号：XXXX

指导教师：XXXX

北京交通大学

2016 年 5 月

学士论文版权使用授权书

本学士论文作者完全了解北京交通大学有关保留、使用学士论文的规定。特授权北京交通大学可以将学士论文的全部或部分内容编入有关数据库进行检索，提供阅览服务，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：　　指导教师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

中文摘要

摘要：随着智能设备的流行，智能硬件产品的需求量逐渐增大，其中无线音箱备受关注，成为了很多年轻人家里必不可少的家居设备；在市场上无线音箱种类多样，包括蓝牙音箱、2.4G 无线音箱和 Wi-Fi 音箱等。目前，蓝牙音响凭借着技术成熟、成本低等优势仍然是当今无线音响的主力军，但蓝牙的缺点也很明显，传输距离短并且带宽窄。由于 2.4 无线音箱和蓝牙音箱本身技术有很大的局限性，很难或者根本无法承载未来智能家居这一方向的需求。为了能够满足人们对未来智能音箱的需求,另一种同样能给人们带来极大便捷性的 Wi-Fi 音箱成为研究重点。

A8 音箱是我所在实习单位研发的一款基于 Android 平台的 Wi-Fi 音箱，Wi-Fi 的带宽基本能保证在 150Mbps 以上，极大地提升了音乐的传输效果，让人们可以享受无损音乐；除了音质上的良好用户体验，本论文重点研究的是如何从多媒体控制角度给用户提供良好的使用体验。在这款 Wi-Fi 音箱研发中，作者在项目团队负责该产品的多媒体控制系统，主要工作是需求分析、系统设计、编码实现、后期测试等；本文首先分析了项目背景以及项目意义，接着明确了项目的需求与系统架构设计，在此基础上，对多媒体控制系统的各个功能模块提出了实现方案，本文还介绍了该系统的测试方案包括测试标准、测试用例等，展示了运行效果，接着本文还针对控制系统的复杂的逻辑控制，展示了几个极具代表性的复杂逻辑控制处理。

目前，A8 音箱的软件端正处于系统测试阶段，采用公司内部的缺陷管理系统进行缺陷管理，及时修复测试团队提交的 bug。

关键词：智能硬件产品；Wi-Fi 音箱；多媒体控制系统；Android 应用开发

ABSTRACT

ABSTRACT: With the popularity of smart devices, the demand of intelligent hardware product increases. The wireless speaker becomes the essential household equipment of many young people. There are kinds of wireless speakers on the market, including Bluetooth speakers, 2.4 G wireless speakers, Wi-Fi wireless speakers and so on. Currently, with mature technology and low cost bluetooth audio is still the main force in today's wireless audio. But its disadvantage is also obvious: short transmission distance and narrow bandwidth. Since bluetooth speakers and 2.4 wireless speakers have many limitations, it is difficult or impossible to meet future demand for smart-home. In order to meet the demand for future intelligent speaker, Wi-Fi speaker which can also bring great convenience becomes a research focus.

A8 speaker is a kind of Wi-Fi speaker based on Android. Generally, Wi-Fi bandwidth can be guaranteed more than 150Mbps, and it can enhance the transmission effect of music greatly, so that people can enjoy high-quality music. In addition to improve sound quality, this thesis focuses on how to control the angle from multimedia to provide a good user experience. For project implementation, the author is responsible for the project's multimedia control system. The main work is needs analysis, system design, coding implementation and system testing. Firstly, The thesis analyzes the project background and significance of the project, then define requirements and system architecture design of the project. On this basis of previous analysis, the thesis analyzes the implementation of multimedia control system. This thesis also describes system testing and shows the operating results; For the complex control logic, the paper analyzes several complex logic control cases in detail.

Currently, the application of A8 speaker is in system testing phase, and we use the internal defect management system for bug management, timely repairing bug submitted by the test team.

KEYWORDS： Intelligent hardware products ; Wi-Fi speaker ; Multimedia control system; Android application development

目　　录

1 引言

本文阐述的是基于 Android 的 Wi-Fi 智能音箱多媒体控制系统的设计与实现，该系统基于本人在实习期间参与的一个重要项目——Wi-Fi 智能音箱，该控制系统作为项目的子系统，扮演 Wi-Fi 智能音箱的控制中心角色。根据软件工程理论、Android 应用开发、软件设计模式等，系统地对该控制系统进行了设计与实现，独立完成了整个多媒体控制系统的研发。下面将从项目背景与意义、国内外发展现状、论文主要内容以及论文组织结构 4 个方面对项目进行简要介绍。

1.1 论文背景与意义

本项目来源于工程实际，是作者在实习期间负责的重要项目——A8 Wi-Fi 智能音箱；随着互联网的发展,人们对无线电子产品的需求量越来越大，不仅仅是手机、pad 等移动设备，智能设备中无线音箱的需求量也逐渐增大,成为了很多年轻人家里必不可少的家居设备。这些电子产品的出现让我们的生活充满了更多的乐趣，也带给我们很多便利。在市场上无线音箱种类多样，包括蓝牙音箱、2.4G 无线音箱和 Wi-Fi 音箱等等。目前，蓝牙音响凭借着技术成熟、成本低等优势仍然是当今无线音响的主力军[1]，但蓝牙的缺点也很明显，有效传输距离短，一般在 10m 左右，并且带宽窄[1][2]。由于 2.4 无线音箱和蓝牙音箱本身技术有很大的局限性，很难或者根本无法承载未来智能家居这一方向的需求。为了能够满足人们对未来智能硬件产品—智能音箱的需求,另一种同样能给人们带来极大便捷性的 Wi-Fi 技术成为研究重点。除此之外，Android 操作系统在移动设备开发中应用得越来越广泛，其开源这一特性能够给开发人员提供创新空间，不断对应用版本升级。基于 Android 的无线音箱具有良好的应用前景，Wi-Fi 音箱被认为在未来 2-3 年将增速迅猛，将成为人们购买无线音箱产品的首选。

Wi-Fi 音箱是一种全新的品类，它不仅包括了听音乐这一个特点，同时它还加入了多房间控制组网，支持多设备控制，这意味着多人可以控制一台音箱，实现设备共享；设计良好的多媒体控制系统将会为多设备控制带来全新的用户体验。Wi-Fi 音箱还有一个重要特点—独立工作，Wi-Fi 音箱不会占用手机音频通道，手机可以多任务工作。

总结来看，从用户角度，Wi-Fi 音箱改变了人们的传统听歌方式，提升生活质量，带给大家全新体验；从行业发展角度，Wi-Fi 音箱推进了智能家居的发展，为智能家居行业注入了新的力量。从开发商角度，Wi-Fi 音箱的竞争力越来越大，如何把握好用户需求，开发出用户体验好的产品将是最需要思考的问题。

1.2 国内外发展现状

智能音箱作为智能家居的重要组成部分，已经成为国内外研究和开发的重点[1]。在国外，最早研发 Wi-Fi 音箱是美国的 SONOS 公司，之后索尼、飞利浦等电子产品制造商也相继推出类似 Wi-Fi 音箱的产品。随着音箱的智能化，无线化，国内很多厂家也已经加入了 Wi-Fi 音箱研发的行列，市场上也已经出现了一些产品。不过从目前的现状来看，Wi-Fi 音箱仍处于起步阶段，用户在使用过程中经常遇到很多问题。例如初次使用连不上网（或者不会进行网络配置）、连接步骤太繁琐、APP 没有添加音乐服务功能（仍需播放手机里的音乐）、音箱控制方式复杂等等，用户体验还很初级。

对于 Wi-Fi 音箱的研发，从目前的市场和技术来看，目前的研究方向集中在 Wi-Fi 连接过程的优化、传输协议的采用、音乐内容丰富性、交互方式等。

1.3 论文主要内容

作者在魅族科技有限公司智能硬件产品部实习期间，有幸参与到了的 Wi-Fi 音箱的研发。根据项目的需求，作者在项目开始之前针对 Android 应用开发进行了深入学习理解，主要是 Android 的 4 大组件以及消息处理机制等方面的知识[3]。作者在研发过程中付出了大量的精力与时间，最终完成了音箱的多媒体控制系统的设计与实现，本文所涉及的研究内容基于作者的实习工作。

对于项目中的多媒体控制系统，作者负责需求分析、系统设计、编码实现；对于项目测试，作者主要参与了单元测试和部分功能性测试，其他测试由公司外包的测试团队负责，这些都是本文的研究内容。在需求分析过程中，作者将多媒体控制系统划分为 7 个模块，分别是投射来源管理、远程命令控制、媒体音乐播放控制、播放状态管理、网络状态实时检测、用户界面管理、物理按键控制。在系统设计上，作者以自身经验，尽量从可拓展性上，采用一定设计模式思想进行了设计。除此之外，由于与 Wi-Fi 音箱控制系统相关的参考资料有限，作者在研发过程遇到一些复杂问题，这些问题也将会在本文中进行分析。目前项目还在系统测试阶段，本文在最后也会展示项目的目前运行效果。

1.4 论文组织结构

按照软件开发的规范流程从需求分析、系统架构设计、系统详细设计与实现、项目测试这四个方面进行论述，本文主要分为八个章节，论文组织结构如下：

第一章： 引言，从背景与意义、国内外发展现状、论文主要内容、论文结构安排 4 个方面对本文进行了论述。

第二章：关键技术与名词解释，主要介绍本文中涉及的关键技术以及解释文中相关的特有名词；

第三章：系统需求分析，从项目描述以及功能模块划分、多媒体控制系统描述以及模块划分和功能模块需求分析（包括功能性和非功能性需求）三个部分进行论述；

第四章：系统概要设计，对多媒体控制系统这一子系统进行模块分解以及架构设计；

第五章：系统详细设计，对本文研究的 7 个功能模块从设计和实现过程进行详细论述；

第六章：测试和运行效果，从系统的测试方案以及项目的运行效果图来论述；

第七章：系统难点问题分析，主要介绍系统开发中遇到的复杂业务问题以及对应的解决方案；

第八章：总结与展望，主要总结项目的主要成果与不足。

2 关键技术概述与名词解释

A8 Wi-Fi 音箱的软件端是基于 Android 平台进行研发的，本文所论述的多媒体控制系统作为软件端的重要组成部分，在项目设计实现过程中，用到的关键技术主要有：Android Service 与 Activity 通信、Android 消息处理机制、回调机制与异步调用、universal-image-loader 网络图片加载、Gson 数据类型转换器；本章节除了介绍项目中使用的关键技术，还将对项目中的名词进行解释，便于读者理解。

2.1 关键技术概述

2.1.1 Android Service 与 Activity 通信

Android 应用开发四大基本组件是 Activity、Service、BroadcastReceiver 以及 ContentProvider。本项目使用了其中的 Activity、Service 服务和 BroadcastReceiver 广播接收器三个基本组件来实现。

Activity 是 Android 应用中负责与用户交互的组件，提供可视化用户界面；Service 通常在后台运行，一般不需要与用户交互，Service 没有图形用户界面[4]。与 Activity 组件需要继承 Activity 基类相似，Service 组件同样需要继承 Service 基类；Activity 和 Service 都有生命周期。对于广播接收器，它的使用相对简单，开发者只需实现 BroadcastReceiver 的子类，并且重写 onReceive(Context context, Intent intent)方法[4][5]。广播的注册方式有两种：在配置文件中静态注册和在 Java 代码调用 registerReceiver()方法动态注册；如果该广播为内部类，则只能使用动态注册，并且一定要有构造方法。

对于 Activity 与 Service 间的通信方式，主要有两种：第一种是通过调用 bindService(Intent service, ServiceConnection conn, int flags)方法，得到 Service 对象的引用；第二种是在 Activity 中注册接收器，Service 通过广播向 Activity 发送消息。本项目中采用了第一种通信方式，通过绑定服务来通信[6]。

2.1.2 Android 消息处理机制

Android 的消息传递机制，包括 5 个基本要素[7]：

（1） 消息 Message：线程间的通讯数据单元

（2） 消息队列 Message Queue：用于存放通过 Handler 发布的消息，按照先进先

出的原则执行

（3） Handler：Message 的主要处理者，它主要负责将 Message 添加到消息队列和处理消息队列中的 Message

（4） 循环器 Looper：在 Message Queue 和 Handler 间充当桥梁的角色，从 Message Queue 中循环取出 Message，并传递给相应的 Handler 处理

（5） 线程 Thread：在 Android 应用中，UI Thread 就是主线程，在启动程序时，主线程会建立一个 Message Queue。每个线程含有一个 Looper 和一个 Message Queue。[7]

Android 的消息传递机制的基本流程：首先需要包装 Message 对象；接着，通过处理器 Handler 的 sendMessage()等方法发送消息（可根据需求选择不同发送消息的方法）；然后，通过 Looper 的 loop()方法不断从 Message Queue 中取出 Message 传递给 Handler，并且将处理完毕的消息从队列中移除；最后通过调用 Handler 的 dispatchMessage()方法完成对 Message 的处理。[7]

Android 的消息传递机制在本项目中应用广泛，除了在用户界面更新上[8]，在处理多个投射来源播放歌曲时采用 Android 的消息处理机制思想，通过继承 Handler 类来实现的。

2.1.3 回调机制与异步调用

在一个应用系统中，各软件模块之间存在调用接口，从调用方法上可以分为：同步调用、回调和异步调用。同步调用是一种阻塞式调用，调用方要等待对方执行完毕才返回，它是一种单向调用；回调是一种双向调用模式，也就是说，被调用方在接口被调用时也会调用对方的接口；异步调用是一种类似消息或事件的机制，不过它的调用方向刚好相反，接口的服务在收到某种讯息或发生某种事件时，会主动通知客户方（即调用客户方的接口）。回调和异步调用的关系非常紧密，通常我们使用回调来实现异步消息的注册，通过异步调用来实现消息的通知。同步调用是三者当中最简单的，而回调又常常是异步调用的基础[9]，如图 2-1 所示。在本文论述的项目中，回调机制在按键事件处理、投射来源管理、播放状态通知等几处都得到了广泛应用[10]。并且在获取网络音乐资源上，主要是指从内容提供商虾米音乐以及喜马拉雅电台等获取歌曲信息，也是采用了基于回调机制的异步调用。

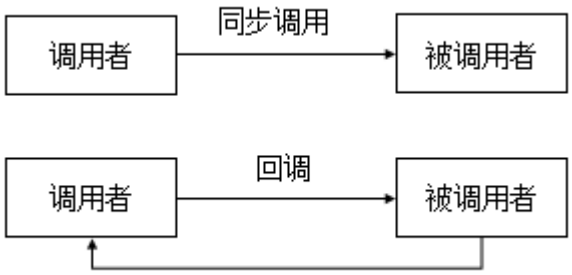

图 2-1 同步调用与回调

2.1.4 Android-Universal-Image-Loader 网络图片加载

Android-Universal-Image-Loader 是一个开源的异步图片加载库，在本文论述的项目中在播放界面，需要对网络图片资源加载显示，为了图片加载流畅、避免出现 OOM 现象，采用了该开源库来加载显示图片资源。

该类库有以下几大优点：

（1）多线程图片加载；

（2）使用者可以根据自身需要配置 ImageLoader；

（3）支持图片缓存和下载过程监听；

（4）支持在网络状况不好的情况下加载图片。

2.1.5 Gson 数据类型转换器

在本文论述的项目中，模块与模块间的数据交互基本都是轻量级的 JSON 格式。对于 JSON 的语法规则，本文将不再论述，本小节主要介绍 JSON 格式与 Java 类之间的格式转换器 Gson。Gson 是 Google 提供的用来在 Java 对象和 JSON 数据之间进行映射的 Java 类库。在数据类型转换过程中，可以通过 Gson 对象调用 toJson()和 fromJson()方法将一个 JSON 字符串转成一个 Java 对象，或者反过来。Gson 在转换数据类型时，不仅仅是简单对象转化，还可以是带泛型的 List 转化。此外还可以使用 transient 指定不需要转换为 JSON 格式的属性，或者使用注解,指定哪些是要暴露转换的属性。

2.2 名词解释

在本文之后的论述中将可能出现以下名词，在本小节对这些名词给出相应解释，便于读者理解全文，具体见表 2-2 名词解释（注：序号 1-2 是对两种歌曲投射来源的介绍）；

序号 3 是一个连接魅族科技有限公司有关的智能产品的 Application；序号 4-21 是 LifeKit 为控制音箱发送的远程命令；序号 22-27 则为音箱播放状态。

表 2-1 名词解释

序号	名词	对应解释
1	DLNA	全称是 DIGITAL LIVING NETWORK ALLIANCE(数字生活网络联盟)，目前安卓系统中支持 DLNA 的音乐播放器主要为网易云音乐、酷狗音乐、QQ 音乐
2	Aux	音频接入音箱的接口，
3	LifeKit	与 A8 这款 Wi-Fi 音箱配套使用的手机 Application，用来配置网络，投射歌曲，获取音箱播放信息等；
4	Info	获取音箱设备信息
5	Play	请求音箱播放歌曲或者歌单
6	Pause	请求音箱暂停播放
7	Resume	请求音箱继续播放
8	Next	请求音箱播放下一首歌曲
9	Previous	请求音箱播放上一首歌曲
10	Seek	跳转到当前播放曲目的某个时刻
11	SetVolume	设置音量
12	SetPlayMode	设置播放模式
13	SetAlarm	设置闹钟
14	SetEQmode	设置预设均衡模式
15	Rename	修改音箱显示名字（蓝牙连接需要用音箱名字进行连接）
16	Suspend	定时关机
17	AddFavorite	添加歌曲到“我的收藏”
18	RevomeFavorite	取消歌曲收藏
19t	AddUserPlayList	添加一个用户歌单
20	DelUserPlayList	删除一个用户歌单
21	UpdateUserPlayList	更新一个用户歌单
22	OnPlayStart	播放开始
23	OnBuffering	缓冲中
24	OnPlayError	播放时出错
25	OnPlayProgress	播放进度
26	OnPlayStateChange	播放状态改变（BUFFERING/STARTED/PAUSED/ERROR）
27	OnVolumeChange	音量改变

2.3 本章小结

本章主要介绍了项目中使用的 5 个主要技术，此外对项目涉及的专业术语以及项目中自定义的名词进行了介绍。

3 Wi-Fi 音箱多媒体控制系统需求分析

Wi-Fi 音箱多媒体控制系统在 Wi-Fi 整个项目的核心模块，对于音箱与用户间良好的交互影响重大。下面将从 A8 Wi-Fi 音箱项目简介，多媒体控制系统功能性需求分析以及非功能性需求分析进行论述。

3.1 A8 Wi-Fi 音箱项目简介

A8 音箱是基于 Android 平台的 Wi-Fi 音箱，从用户角度可以分为三步：Wi-Fi 连接、控制音箱、音频播放。Wi-Fi 连接是指通过 LifeKit 或者微信公众号让音箱连接家里局域网，简单快速的网络连接是提升用户体验的重要步骤；控制音箱的方式有多种，包括音箱物理按键，LifeKit，蓝牙连接，DLNA 连接[11]，Aux in 连接；其中 LifeKit 控制音箱是最为丰富的，涉及多种控制命令；物理按键的功能也具有丰富性，用户可以通过 4 个物理按键的多种操作方式（单击、双击、长按以及组合按键）来控制音箱。这部分内容也就是本文在后续章节重点论述的内容。对于音频播放，由于与多个内容提供商合作，部分商家的音乐资源需要用对应的播放器进行播放，该项目采用了多个播放器，除了 Android 系统默认的 MediaPlayer，还包括 DlnaPlayer 和 SpotifyPlayer。根据音频资源的 url，播放器进行加载播放。通过对系统业务逻辑的分析，该 Wi-Fi 音箱可以划分为 6 个大模块，如图 3-1 所示。本文将重点论述多媒体控制系统这一子系统。

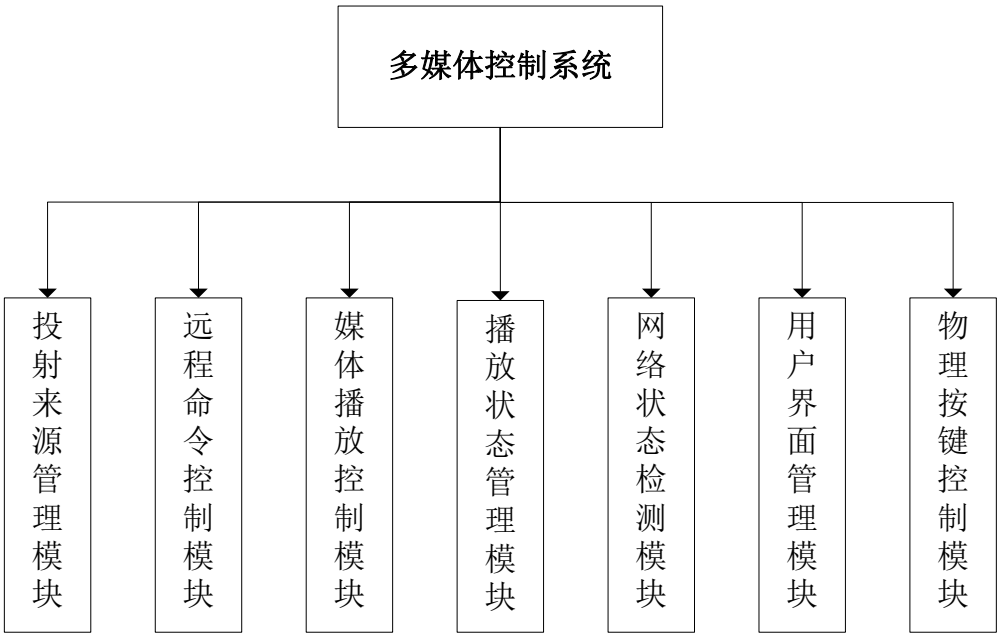

图 3-1 A8 音箱功能分解图

3.2 系统功能性需求分析

根据产品描述，本小节将通过用例图形象简洁地进行系统功能性需求分析。由于用例太多，本文将只介绍几个主要的用例描述。具体如下：

3.2.1 系统用例分析

本小节将从用户角度对多媒体控制系统这一子系统进行分析，使用用例图展示系统的基本功能。

首先，本项目支持蓝牙连接、DLNA 以及 Aux-in 连接，此外 LifeKit 作为音箱的控制 Application 也一定可以与音箱连接通信。具体用例如图 3-2 所示。

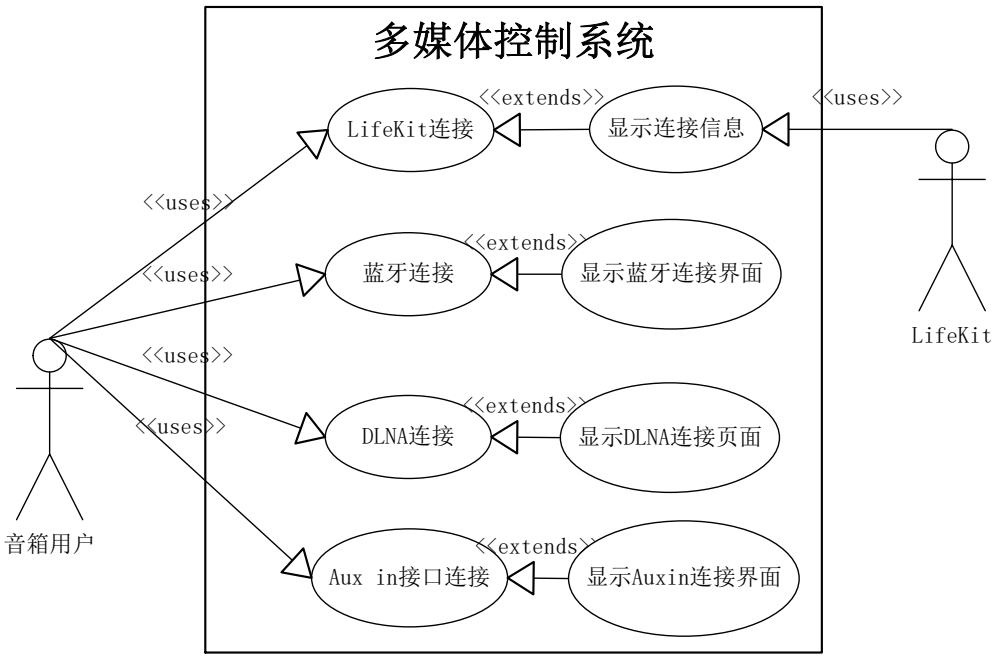

图 3-2 用户投射用例图

接着，作为音箱的专门控制 Application，LifeKit 除了投射曲目之外还具有很多其他的与播放曲目相关的远程控制功能，例如 Pause、Seek、AddFavorite、UpdateUserPlayList 等。具体用例如图 3-3 所示。

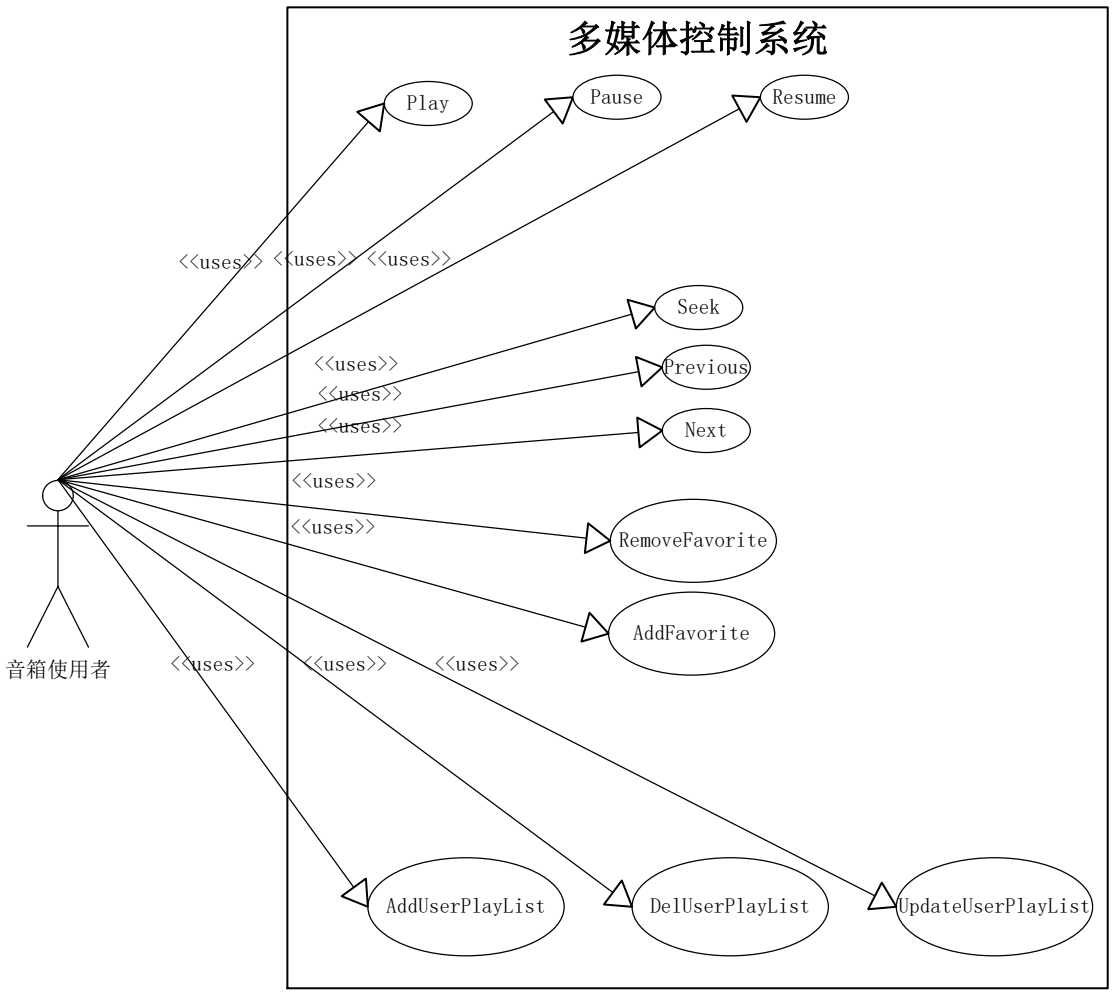

图 3-3 LifeKit 控制命令用例图

对于歌曲播放，本项目 A8 音箱是一个独立的“计算机系统”，它可以通过网络获取内容提供商的音乐资源，独立工作。音箱启动并且网络连接成功便可以播放歌曲，用户还可以自由选择频道。除了播放内容提供商虾米音乐以及喜马拉雅电台等资源，用户还可以通过蓝牙或者 DLNA 或者 Line-in 进行歌曲投射播放。详细用例介绍如图 3-4 所示。

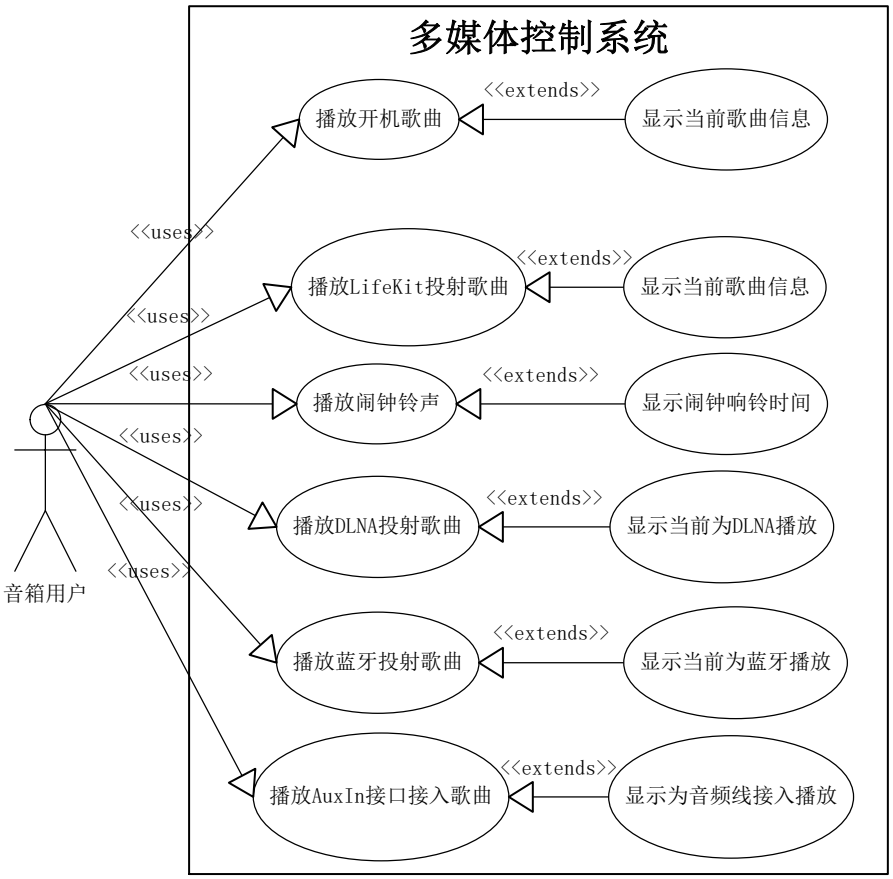

图 3-4 播放曲目用例图

从整体外观设计来看，音箱有 4 个物理按键，分别是音量加、音量减、Next 键以及 Play 键。这四个按键有多种操作方式，短按 Next 键可以切换频道，长按可以切换歌单，双击便可以切换歌曲，同时按下音量加减键可以进入网络配置。这 4 个按键的用处不限于以上提及的用法，具体如图 3-5 所示。

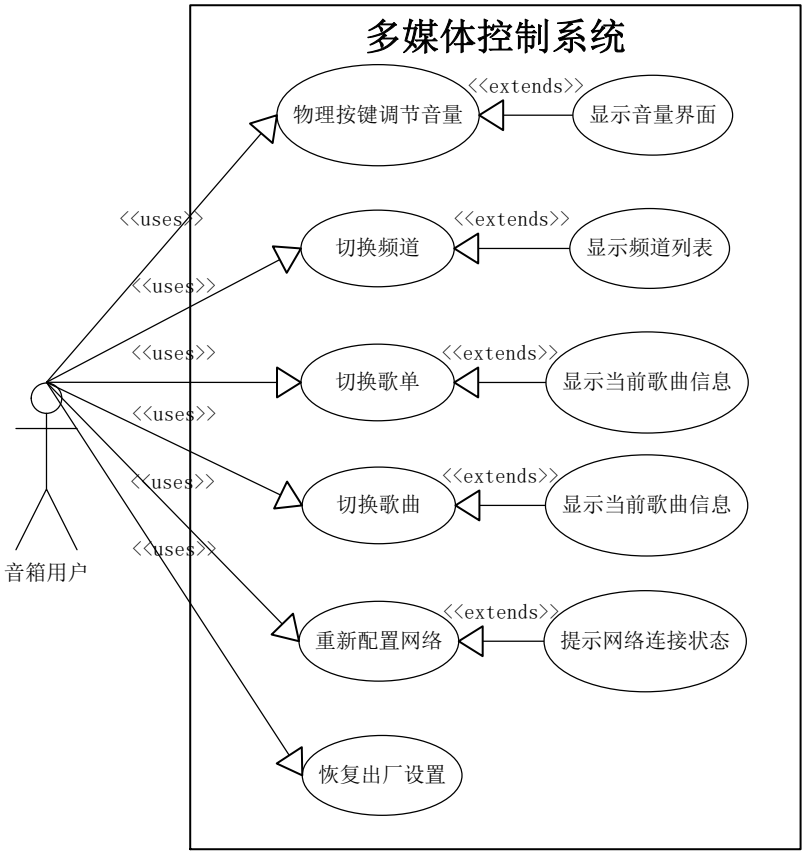

图 3-5 物理按键操作用例图

LifeKit 不仅能够对音箱进行播放相关的命令控制，它还具有设置功能。用户可以给音箱设置新名字，设置音箱大小，设置播放模式，设置均衡模式等。详细用例如图 3-6 所示。

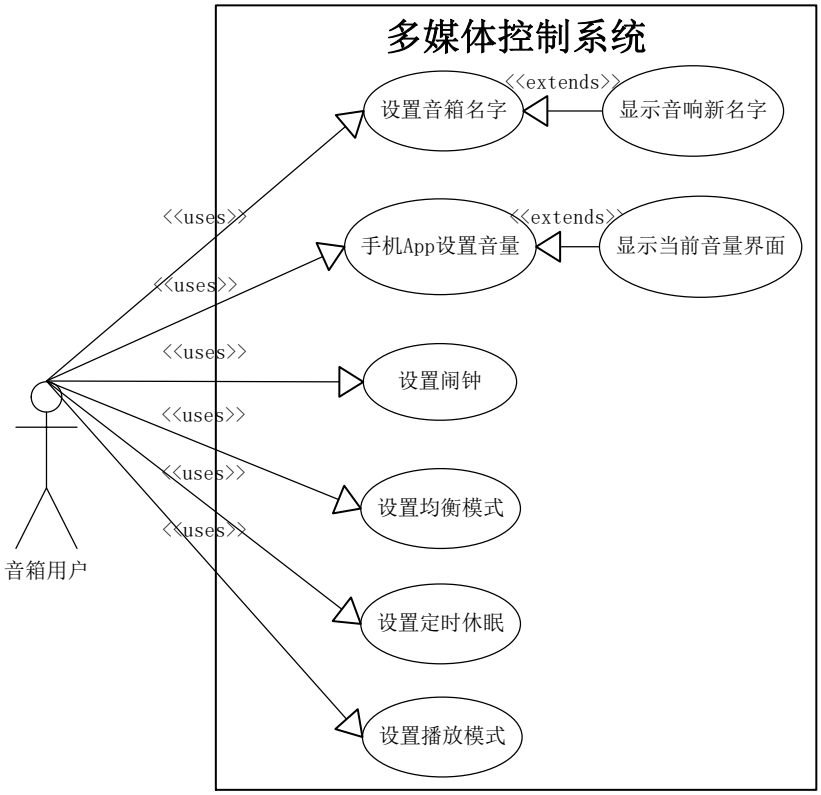

图 3-6 音箱设置用例图

对于网络状态，用户可以通过断开路由器、插入路由器以及通过 LifeKit 更改网络等改变网络状态。详细用例如图 3-7 所示。

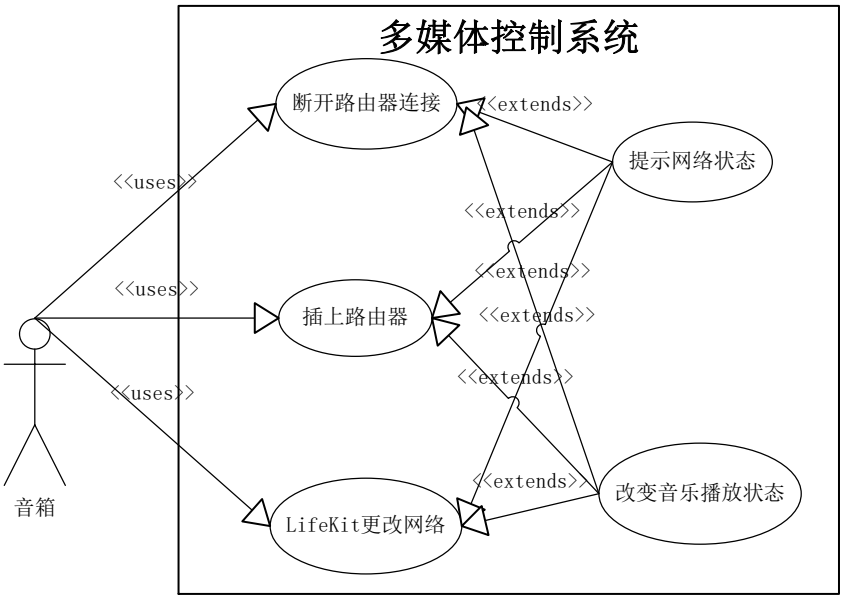

图 3-7 网络控制用例图

3.2.2 系统用例描述

本小节主要从播放开机歌曲、添加歌曲到“我的收藏”、切换频道、音量键调节音量、闹钟响铃、DLNA 投射歌曲这六个用例对系统进行用例描述。

播放开机歌曲是指当音箱启动后，在网络连接成功的条件下，音箱会根据上一次播放频道记录，自动获取相应频道的内容进行播放。对于播放开机歌曲这一用例的详细描述如表 3-1 所示。

表 3-1 播放开机歌曲用例描述

<table>
<tr><td>用例名：</td><td colspan="2">播放开机歌曲</td><td>SC1</td></tr>
<tr><td>参与者：</td><td colspan="3">音箱拥有者</td></tr>
<tr><td>其他参与者：</td><td colspan="3">无</td></tr>
<tr><td>项目相关人员兴趣：</td><td colspan="3">音箱拥有者：希望在音箱启动后播放歌曲</td></tr>
<tr><td>描述</td><td colspan="3">该用例用于在用户给音箱接上电源启动后播放歌曲。用户第一次启动音箱，音箱进入配网流程，用户需要用 LifeKit 或者定制的微信公众号进行网络配置，网络连接成功后，音箱播放“本地音乐”这一频道的歌曲；非第一次启动音箱，音箱开机后自动联网，联网成功后，将播放音箱上一次频道的歌曲。</td></tr>
<tr><td>前置条件</td><td colspan="3">音箱已经联网成功、播放歌曲信息已经获取</td></tr>
<tr><td>后置条件</td><td colspan="3">当前播放频道已经保存</td></tr>
<tr><td>触发条件</td><td colspan="3">1. 音箱开机
2. 音箱联网成功
3. 播放歌单已获取</td></tr>
<tr><td rowspan="2">基本流程</td><td colspan="2">动作参与者</td><td>系统响应</td></tr>
<tr><td colspan="2">1. 用户第一次启动音箱
2. 用户给音箱配置网络
3. 用户非第一次启动音箱</td><td>1. 进入配网流程
2. 收到联网信息，尝试连接网络，等待几秒提示网络连接成功，音箱开始播放音乐，显示播放界面
3. 提示网络连接成功，开始播放音乐</td></tr>
<tr><td>拓展流程</td><td colspan="3">非第一次启动音箱时，用户想要切换网络，用户可以通过同时按下音量加减键进入网络配置。</td></tr>
<tr><td>结束</td><td colspan="3">音箱已经播放音乐</td></tr>
<tr><td>业务规则</td><td colspan="3">网络可以使用，LifeKit 与音箱在同一局域网</td></tr>
<tr><td>假设</td><td colspan="3">音箱收到正确的网络信息，并且获取的播放歌单不为空</td></tr>
<tr><td>待解决问题</td><td colspan="3">音箱从内容提供商获取的部分音乐信息不符合要求，例如 url 为空，音乐封面 url 为空等</td></tr>
</table>

用户可以通过 LifeKit 对当前播放曲目添加收藏，收藏成功后，该曲目便是“我的收藏”这一频道里的曲目，当用户切换到“我的收藏”频道时，可以播放该曲目。详细用例描述如表 3-2 所示。

表 3-2 添加收藏歌曲用例描述

<table>
<tr><td>用例名：</td><td colspan="2">添加歌曲到“我的收藏”即收藏歌曲</td><td>SC2</td></tr>
<tr><td>参与者：</td><td colspan="3">音箱拥有者</td></tr>
<tr><td>其他参与者：</td><td colspan="3">无</td></tr>
<tr><td>项目相关人员兴趣：</td><td colspan="3">音箱拥有者：添加一首收藏曲目</td></tr>
<tr><td>描述</td><td colspan="3">当音箱在播放喜马拉雅电台频道或者虾米推荐的内容时，用户可以通过 LifeKit 点击“收藏”将当前曲目加入“我的收藏”频道，并且返回收藏结果。如果收藏成功，当用户切换到“我的收藏”，歌单里将包含这首歌，收藏失败，则不会加入到“我的收藏”列表。</td></tr>
<tr><td>前置条件</td><td colspan="3">当前曲目非 DLNA 投射歌曲、蓝牙投射歌曲、闹钟铃声以及 Aux in 投射歌曲</td></tr>
<tr><td>后置条件</td><td colspan="3">当前曲目成功添加到数据库</td></tr>
<tr><td>触发条件</td><td colspan="3">1．用户通过 LifeKit 点击“收藏”</td></tr>
<tr><td rowspan="2">基本流程</td><td colspan="2">动作参与者</td><td>系统响应</td></tr>
<tr><td colspan="2">1． 用户启动音箱
2． 控制音箱播放前置条件中提及之外的内容
3． 用户通过 LifeKit 点击“收藏”</td><td>1． 提示联网成功
2． 播放音乐，显示播放界面
3． 返回收藏结果</td></tr>
<tr><td>拓展流程</td><td colspan="3">用户可以重新点击“收藏”</td></tr>
<tr><td>结束</td><td colspan="3">系统给 LifeKit 返回收藏结果</td></tr>
<tr><td>业务规则</td><td colspan="3">该曲目必须为 LifeKit 投射歌曲或者喜马拉雅电台或者虾米推荐的内容</td></tr>
<tr><td>假设</td><td colspan="3">在添加收藏过程中，网络良好</td></tr>
<tr><td>待解决问题</td><td colspan="3">无</td></tr>
</table>

音箱有特定的内置频道，用户也可以自定义频道。由于音箱可以脱离手机独立工作，因此切换频道这一用例非常有用。对于这一用例的详细描述如表 3-3 所示。

表 3-3 切换频道用例描述

用例名：	**切换频道**	**SC3**
参与者：	音箱拥有者	
其他参与者：	无	
项目相关人员兴趣：	音箱拥有者：切换到其他频道，让音箱播放其他内容	
描述	该用例用于音箱启动后，用户想要切换到某个频道。用户通过短按 Next 切换频道，播放对应内容；在频道界面显示时，短按音量加减键同样可以上下切换选中频道，并且 10 秒内按下 Play 切换到选中频道，否则继续播放原来的内容。	
前置条件	音箱已经联网成功、播放歌曲信息已经获取	
后置条件	当前播放频道已经保存	
触发条件	1. 在频道页面显示时，10 秒内按下 Play 键	
基本流程	动作参与者	系统响应
	1. 用户启动音箱 2. 用户短按 Next 键 3. 用户短按音量加减键 4. 用户短按 Play 键	1. 音箱播放音乐，显示播放界面 2. 显示频道列表界面 3. 更改选中频道显示 4. 切换到用户选中频道，播放相应内容
拓展流程	无	
结束	音箱播放用户选择的相应频道内容，标记当前频道为用户选中频道	
业务规则	10 秒内按下 Play 键才视为真正地切换频道	
假设	相应频道内容不为空	
待解决问题	无	

在不同的环境下，用户对音量的需求是不一样的，音量调节的途径有两种，分别是LifeKit 和音量键。音量键调节音量的用例描述如表 3-4 所示。

表 3-4 音量键调节音量用例描述

用例名：	**音量键调节音量**	**SC4**
参与者：	音箱拥有者	
其他参与者：	无	
项目相关人员兴趣：	音箱拥有者：希望改变音箱播放曲目时的音量	
描述	该用例用于音箱启动后的任意时刻，改变音箱音量。当用户按下音量加减键，音量页面显示，并且随着用户操作，音量和音量界面随之改变。	
前置条件	音箱已经启动	
后置条件	调节后的音量已保存为当前音量	
触发条件	1. 用户按下音量加减键	
基本流程	动作参与者	系统响应
	1. 用户启动音箱 2. 用户按下音量加减键	1. 播放音乐并且显示播放界面 2. 音量改变，音量界面显示
拓展流程	用户可以连续按下按键，音量随之改变，用户不再按下音量键后，音量界面 5 秒后消失	
结束	音量已经改变，音量界面消失	
业务规则	最小音量为 0，最大音量不超过 100	
假设	无	
待解决问题	无	

除了播放电台以及音乐外，用户还可以设置闹钟。闹钟响铃也是一种播放投射源，关于闹钟这一功能详细描述如表 3-5 所示。

表 3-5 闹钟响铃用例描述

用例名：	**闹钟响铃**	**SC5**
参与者：	音箱拥有者	
其他参与者：	无	
项目相关人员兴趣：	音箱拥有者：在用户设置的提醒时刻，音箱定时响铃	
描述	该用例用于在用户通过 LifeKit 设置闹钟后。系统将根据闹钟触发时间，进行响铃操作。在断网的情况下，播放 SD card 里的默认铃声，网络已连接的情况下，播放对应的铃声。 闹钟 Mode：单次，每天，工作日,自定义；当 mode 是自定义时，列举出每周那几天需要响，范围 1-7,表示从周一到周日	
前置条件	用户已经设置闹钟，并且闹钟设置有效	
后置条件	无	
触发条件	1. 时间	
基本流程	动作参与者	系统响应
	1. 用户设置闹钟 2. 时间为闹钟设置的时间	1. 返回闹钟设置结果 2. 播放铃声，显示闹钟界面
拓展流程	无	
结束	闹钟响铃结束，并且闹钟界面消失	
业务规则	闹钟设置的时间为有效时间，	
假设	音箱收到正确的网络信息，并且获取的播放歌单不为空	
待解决问题	音箱从内容提供商获取的部分音乐信息不符合要求，例如 url 为空，音乐封面 url 为空等	

目前市场上大部分播放都支持 DLNA 功能，用户可以通过 DLNA 投射曲目通过音箱来播放，体验更好的听觉效果。关于 DLNA 投射歌曲的详细描述如表 3-6 所示。

表 3-6 DLNA 投射歌曲用例描述

用例名：	**DLNA 投射歌曲**	**SC6**
参与者：	音箱拥有者	
其他参与者：	无	
项目相关人员兴趣：	音箱拥有者：希望播放网易云音乐、酷狗音乐、QQ 音乐等播放器的歌	
描述	该用例用于音箱接上电源启动后，用户通过手机上的支持 DLNA 的播放器来投射歌曲让音乐播放，达到音质音量绝佳的效果。播放完投射歌曲后，音箱需要恢复投射歌曲之前的状态（如果之前是播放状态，音箱需要继续播放之前的曲目；而如果是暂停状态，恢复后依然是暂停状态。两种情况下用户界面俊需要更新为投射之前的界面）	
前置条件	音箱已启动，手机上的播放器支持 DLNA	
后置条件	无	
触发条件	在歌曲页面，用户点击支持 DLNA 的图标，选择音箱这一设备	
基本流程	动作参与者	系统响应
	1. 用户打开支持 DLNA 的播放器 2. 在歌曲页面点击相应图标，选择音箱这一设备	1. 音箱播放投射歌曲，并且显示歌曲信息，歌曲封面为 DLNA 图标
拓展流程	用户可以在播放器上控制播放暂停或者继续，还能 Seek 和调节音量；播放完投射歌曲后，音箱需要恢复投射歌曲之前的状态	
结束	音箱已经播放投射的曲目，手机端播放器已暂停	
业务规则	手机端播放器支持 DLNA	
假设	手机端播放器能够发现音箱设备	
待解决问题	连接过程容易中断，需要多次尝试	

3.3 系统非功能性需求分析

随着软件工程的成熟及软件行业的发展，对软件非功能性需求的研究越来越得到关注，人们普遍认为，软件非功能性需求是否能被满足，在很大程度上决定了软件项目的成败[12]。本小节将从 5 个非功能性需求对项目进行论述。

（1）可靠性

LifeKit 使用者必须是 Flyme 的注册用户。当音箱系统和 LifeKit 在同一局域网下，用户用 Flyme 账号登入 LifeKit 之后才可以用 LifeKit 远程操作。

（2）易用性

用户的每一次操作，系统均需要给出相应响应（语音或者文字提示）。当网络状态信号发生改变时，系统立即更新信号显示，当网络断开或者重新连接成功，系统通过语音和界面提示用户当前网络状态。在从内容提供商获取音乐资源的基础上，通过短按、长按、双击以及组合按键操作系统的 4 个物理按键：音量加减键、Play 键、Next 键，可以让音箱脱离手机，独立工作。

（3）效率

在易用性这一非功能性需求中已经提及对于用户的每一次操作，系统均需要给出相应响应。在网络状态良好的情况下，切换频道时系统在按下 Play 键后 5 秒内成功切换频道；切换上下曲目时，系统在 3 秒内成功切换；在播放曲目时，用户界面需要 3 秒内容更新，包括图片资源的显示；当用户使用 LifeKit 控制音箱时，用户的每次操作，系统在 5 秒内返回操作结果；每次物理按键按下后系统在 3 秒内响应；网络状态发生改变（已连接、已断开、信号强度改变），系统在 5 秒内给出提示。

（4）用户友好性

音箱的物理按键操作符合用户的习惯，具有易用性的特点。对于各种功能操作，用户及时得到可视化的反馈，容易达到用户期望。界面简洁美观，显示关键信息，用户界面设计符合产品总体规划，与硬件条件相符。此外音量值需要符合用户听觉习惯，根据声学理论知识对音量值进行调整。

（5）可拓展性

由于项目需求会根据用户反馈进行更改，多媒体控制系统必须能够适应需求更改，能够拓展新功能或者修改原有功能，因此良好的可拓展性对于项目后期升级十分重要。例如目前系统没有支持 Airplay，根据产品计划后期升级中将支持 Airplay；目前内容合作商只有虾米音乐和喜马拉雅电台，后期计划加入其它提供商。

3.4 本章小结

本章从总体介绍了 Wi-Fi 音箱，并且将其分解为七个子系统。本章重点论述了多媒体控制子系统的需求分析，通过 6 个用例图以及部分用例描述论述了功能性的需求。接着从可靠性、易用性、效率、用户友好性、可拓展性对多媒体控制子系统进行了非功能性需求分析。

4 Wi-Fi 音箱多媒体控制系统概要设计

本章将对多媒体控制系统进行架构设计，在上一章需求分析的基础上，将系统用例图转换为软件结构。本章的主要任务是将复杂的系统按照功能进行模块划分，确定系统整体结构以及模块间的层次结构[13]。

4.1 MVC 框架模式

A8 Wi-Fi 智能音箱这一项目的软件端是基于 Android 平台开发的，作为软件端的子系统多媒体控制系统毫无疑问也是基于 Android 平台。本项目的开发工具是 Android Studio，使用 Android SDK 提供的 API，最小 SDK 版本为 18。Android 自身采用了经典的 MVC 设计结构[14]，本项目基于 Android，所以也采用了这种设计结构。MVC 即 Model/View/Controller 的缩写，它分为模型、视图以及控制器三层[15]，采用该设计结构，能够使得项目整体结构清晰，每一层分工明确[16]。接下来结合本项目论述 MVC 框架在本项目中的应用。

模型 Model 用于处理数据逻辑的部分，本系统与其他系统间的数据交互采用的是轻量级的 JSON 格式，键值对的格式使得参数意义明确。模型对象负责将 JSON 格式数据存储到 SharedPreference 中，方便视图层和控制层访问数据[17]。

视图 View 是除了音乐播放外，给用户最直观响应的一层。区别于大多的 Android 应用开发，在本系统中，由于硬件上的设计，视图层主要扮演展示信息的角色，用户不需要在屏幕上操作。在 Android 开发中，视图层主要有 xml 布局文件构成，开发人员根据 UI 视觉图，定义每个页面的布局。

控制层在 MVC 结构设计中充当桥梁，承接视图层和模型层。根据本项目特性，控制层除了需要监听物理按键事件，主要是与其他子系统进行交互，监听其他子系统的命令，然后访问 SharedPreference 存储的数据，通知视图层更新视图。

4.2 系统架构设计

本系统由前台程序和后台服务共同实现，主要用到了 Android 四大基本组件中的 Activity、Service 服务以及 BroadcastReceiver 广播接收器。音乐播放程序一般是在后台进行的，因此本系统使用 Service 在后台执行操作。多媒体控制系统着重于控制，因此需要一个中心控制类，也是一个 Service 的继承类；前台 Activity 作为程序的主入口，绑定中心控制 Service。具体架构设计见下一页的图 4-1 所示。

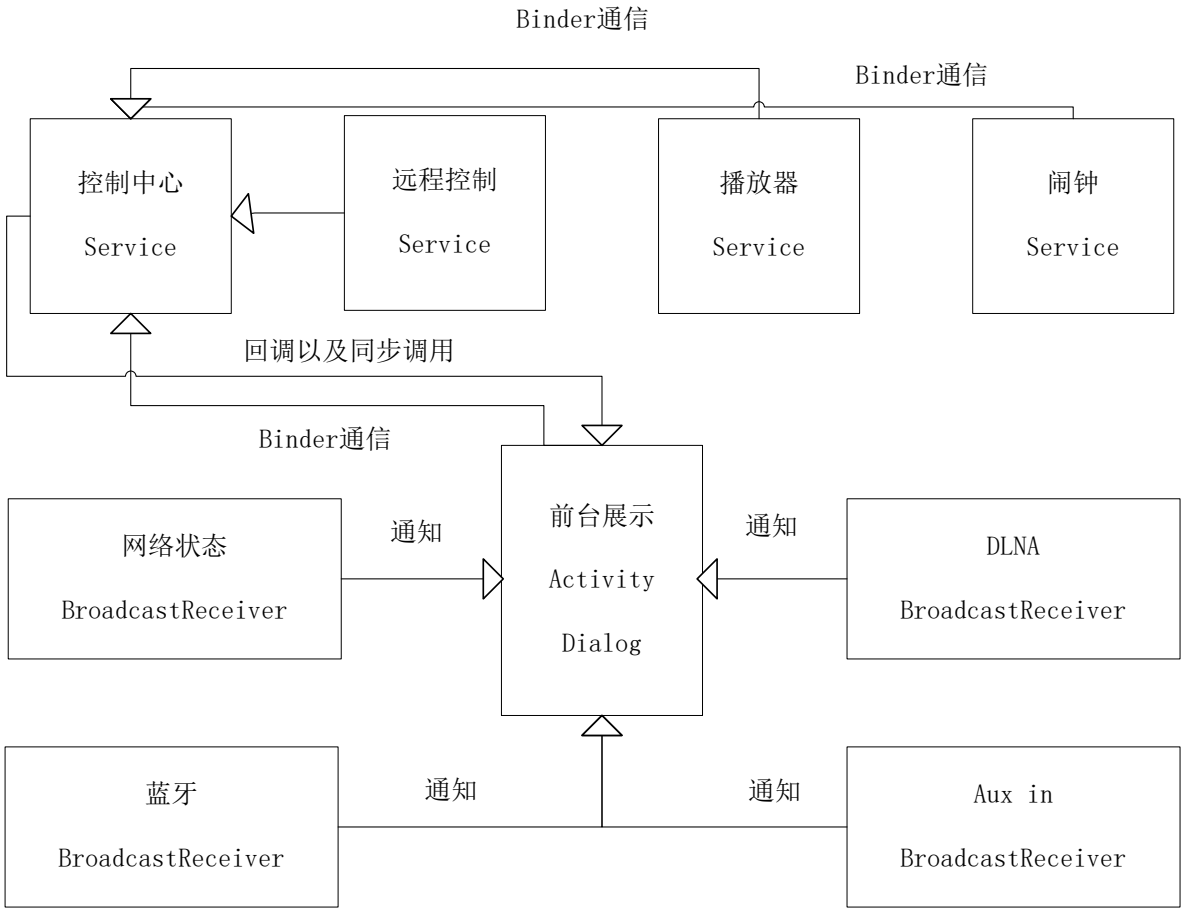

图 4-1 架构设计

4.3 系统模块分解

本小节将根据系统用例将系统进行模块划分，主要包括两部分内容：模块分解图和模块描述。

4.3.1 模块分解图

根据需求分析，多媒体控制系统可以划分为 7 个模块，分别是：投射来源管理、远程命令控制、媒体播放控制、播放状态管理、网络状态监测、用户界面管理、物理按键控制。模块分解图如图 4-2 所示。

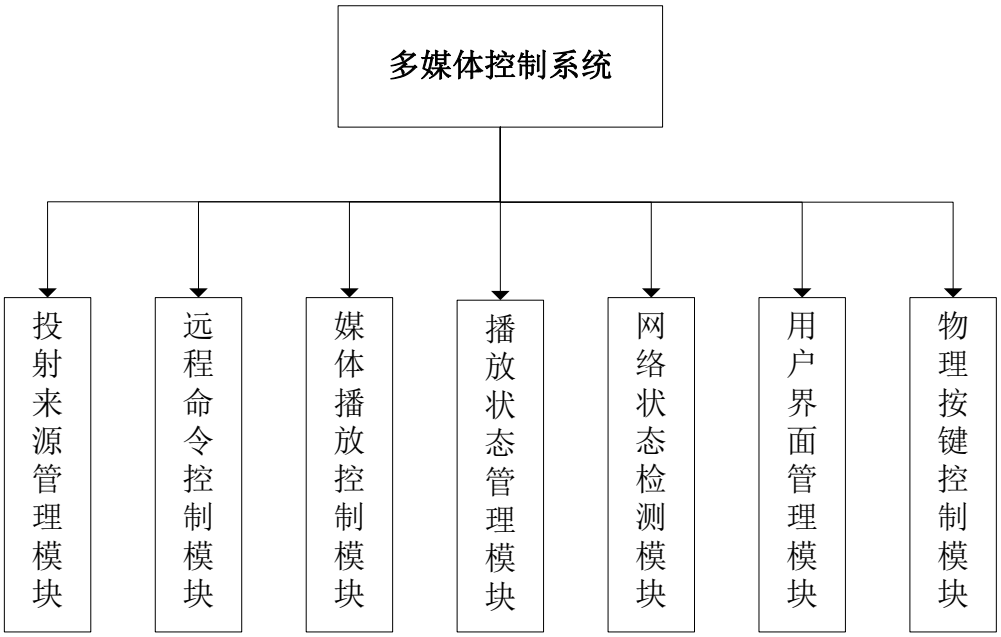

图 4-2 模块分解图

4.3.2 模块描述

（1）投射源管理包括本机投射、LifeKit 投射、DLNA 投射、闹钟投射、蓝牙投射、Aux in 投射；对于 LifeKit 和 DLNA 投射，通过主动注册命令以及控制类，之后相应的控制类通过回调进行播放命令控制。

（2）远程命令控制是指 LifeKit 与音箱端连接远程连接成功后，LifeKit 可以通过 HTTP 连接发送控制命令，这些命令包括：获取音箱的设备信息命令、获取音箱的播放状态、请求音箱播放歌单、设置播放模式、请求音箱暂停播放、请求音箱继续播放、请求音箱播放下一首歌曲、请求音箱播放上一首歌曲、设置音量、获取音箱音量、跳转到当前曲目的某个时刻、设置闹钟、获取闹钟列表、修改音箱显示名字、请求音箱升级、设置预设的均衡模式等；音箱端收到命令后，根据命令对音响进行相应的控制或者将音箱信息返回给远程手机端 LifeKit。

（3）媒体播放控制这一部分采用了操作系统中的抢占式调用原则。当不同的投射来源发送播放歌单的命令时，采用抢占式进行控制处理。闹钟投射、DLNA 播放投射这两种投射方式与 LifeKit 投射、本机投射这两种方式有不同之处；前两种播放完投射的歌单之后需要恢复投射播放之前的状态，所以在前两种播放投射之前需要保存音箱的播放状态，以便于在投射播放完成之后音箱立即恢复原状。

（4）播放状态管理这一部分主要是网络断开、网络连接与多媒体播放、暂停、重新播放这三种播放控制之间的逻辑处理；例如开机联网成功后需要多媒体立即播放；网

络断开需要立即暂停媒体播放，网络重新连接需要重新播放；还有网络断开后，用户重新配置连接网络，网络连接成功后需要重新播放；还有其他情况不在此一一列举。

（5）网络状态检测是采用的 Android 的 4 大组件之一广播来进行检测的，网络状态包括：网络已连接、网络已断开、网络信号改变。一旦 Wi-Fi 状态发生改变，就会向相应的 activity 发送广播通知，系统做出响应。

（6）用户界面管理模块包括播放界面管理、音量调节界面管理、频道切换界面管理、网络状态提示界面管理、投射来源控制时的提示界面显示等。对于作为主界面的播放界面使用 Activity 来展示，媒体播放器通过回调函数通知更新用户界面，这一部分还涉及播放进度条的及时更新（每隔一秒更新一次）。其他界面显示均是使用 Dialog 展示页面信息，并且定义显示时长，达到显示时长之后页面消失。

（7）物理按键管理控制模块涉及的按键有：音量加、减键，Play 键，Next 键；Play 键有长按、短按两个单键操作；Next 键有长按、短按、双击三个单键操作；音量加、减键在没有出现频道对话框时只有调节音量的功能，当出现频道对话框，音量加减键可以上下切换频道；音量加减组合按键同时按下进入网络配置；

4.4 系统接口设计

本小节将对于系统接口设计进行论述，主要包括 LifeKit 与音箱的控制协议，LifeKit 与音箱的状态协议以及多媒体控制系统的数据存储接口设计。控制协议和状态协议是多媒体控制系统的对外接口定义。

4.4.1 控制协议接口设计

LifeKit 和音箱间通过建立两条 HTTP 长连接通信，一条用于从手机向音箱发送命令，另一条用于音箱向手机推送状态变化，控制协议接口设计如表 4-1 所示。

表 4-1 控制协议接口设计

Action	请求方式	携带参数	音箱应答内容	备注

Info	Get	无	{
			status:Int,
			statusMsg:String,
			data:{
			deviceName:String,
			deviceVersion:String,
			
			deviceVersionCode:Int,
			deviceID:String
			maxVolume:Int,
			currentVolume:Int,
			ip:String,
			loginStatus:
			[
			{
			cp:String,
			isLogin:boolean
			}
			]
			}
			}
				currentVolume 表示最大音量的百分比，例如 50，表示 50%音量

Status	Get	无	{
			status:Int,
			statusMsg:String,
			data:
			{
			status:String,
			playList:PlayList,
			trackIndex:Int,
			elapsedTime:Int,
			playMode:Int
			EQMode:Int
			isFavorite:Int
			}
			}
				status 有 4 种：
				BUFFERING
				STARTED
				PAUSED
				ERROR
				
				isFavorite：1 表示收藏，0 表示没有收藏

表 4-1（续） 控制协议接口设计

Action	请求方式	携带参数	音箱应答内容	备注

Play	Post	{
		playList:PlayList,
		trackIndex:Int,
		playMode:Int
		}
			{
			status:Int,
			statusMsg:String,
			}
				trackInedx: 播放的起始歌曲序号
				playmode:顺序 0，乱序 1，单曲循环 2，列表循环 3

Pause	Get	无	{
			status:Int,
			statusMsg:String,
			}	

Resume	Get	无	{
			status:Int,
			statusMsg:String,
			}	

Next	Get	无	{
			status:Int,
			statusMsg:String,
			}	

Prev	Get	无	{
			status:Int,
			statusMsg:String,
			}	

Seek	Post	{
		position:Int
		}
			{
			status:Int,
			statusMsg:String,
			}	

AddFavourite	Post	{
		track:Track
		}
			{
			status:Int,
			statusMsg:String,
			}	

RemoveFavourite	Post	{
		track:Track
		}
			{
			status:Int,
			statusMsg:String,
			}	

表 4-1（续） 控制协议接口设计

Action	请求方式	携带参数	音箱应答内容	备注

AddUserPlayList	Post	{
		playList:playList
		}
			{
			status:Int,
			statusMsg:String,
			data:{
			albumId:String
			}
			}	

DelUserPlayList	Post	{
		albumId:[String]
		}
			{
			status:Int,
			statusMsg:String,
			}	

UpdateUserPlayList	Post	{
		playList:playList
		}
			{
			status:Int,
			statusMsg:String,
			}	

SetAlarm	Post	{
		alarmList:
		[
		{
		time:String
		mode:Int
		dayList:[int]
		playList:PlayList
		volume:Int
		enable:Boolean
		},
		]
		}
			{
			status:Int,
			statusMsg:String,
			}	

SetEQmode	Post	{
		EQMode:int
		}
			{
			status:Int,
			statusMsg:String,
			}	

SetPlayMode	Post	{
		playMode:Int
		}
			{
			status:Int,
			statusMsg:String,
			}	

表 4-1（续） 控制协议接口设计

Action	请求方式	携带参数	音箱应答内容	备注

SetVolume	Post	{
		currentVolume:Int
		}
			{
			status:Int,
			statusMsg:String,
			}	

GetVolume	Get	无	{
			status:Int,
			statusMsg:String,
			data:{
			currentVolume:Int
			}
			}	

GetAlarm	Get	无	{
			status:Int,
			statusMsg:String,
			data:{
			alarmList:
			[
			{
			time:String
			mode:Int
			dayList:[int]
			
			playList:PlayList
			volume:Int
			enable:Boolean
			},
			]
			}
			}	

Suspend	Post	{
		time:int
		}
			{
			status:Int,
			statusMsg:String,
			}	

4.4.2 状态协议接口设计

状态协议是指音箱向 LifeKit 回报音箱播放时的数据交互方式，详细状态协议接口定义如表 4-2 所示。

表 4-2 状态协议接口设计

Action	请求方式	携带参数	备注

OnPlayStart	Post	{
		playList:PlayList,
		trackIndex:Int,
		playMode:Int,
		isFavorite:Int
		}
			isFavorite：0 表示没有本歌曲没有被收藏，1 表示已经被收藏

OnBuffering	Post	{
		percent:int
		}	

OnPlayError	Post	{
		errorCode:int
		errorMsg:String
		}	

OnPlayProgress	Post	{
		currentPosition:int
		duration:int
		}	

OnPlayStateChange	Post	{
		state:String
		}
			state 有 4 个值：BUFFERING、STARTED、PAUSED、ERROR

OnUpgradeError	Post	{
		errorCode:int
		errorMsg:String
		}	

OnVolumeChange	Post	{
		maxVolume:Int,
		currentVolume:Int
		}	

4.4.3 数据存储接口设计

多媒体控制系统这一子系统通过 SharedPreference 进行数据存储，详细接口设计如表 4-3 所示。

表 4-3 数据存储接口设计

接口方法	参数描述	返回值	方法描述
savePlayAlbumInfo	Context context：上下文 PlayInfo playInfo：播放信息	无	保存播放歌单信息
loadPlayAlbumInfo	Context context：上下文	String：JSON 格式的 PlayInfo	获取播放歌单信息
loadDeviceName	Context context：上下文	String：音箱名字	获取音箱名字
saveDeviceName	Context context：上下文 String：音箱名字	无	保存音箱名字
loadVolume	Context context：上下文	int：音量值	获取音箱音量
saveVolume	Context context：上下文 int：音量值	无	保存音量
saveAlarmList	Context context：上下文 String alarmListStr：闹钟信息	无	保存闹钟信息
loadAlarmList	Context context：上下文	String：JSON 格式的闹钟信息	获取闹钟信息

4.5 本章小结

本章从系统采用的框架模式 MVC、系统模块分解以及系统架构设计三个方面对多媒体控制系统进行了概要设计，这部分内容起着承前启后的作用，承接上一章的需求分析，并为下一章系统的详细设计也就是功能模块设计与实现做铺垫。

5 系统功能模块详细设计与实现

本章主要分析每个模块的类实现、类之间的调用关系以及接口设计，详细介绍各模块的实现方法。UML 又称“统一建模语言，是一种用于软件系统制品规约的、可视化的构造及建档语言，也可用于业务建模 及其它非软件系统”[18]。UML 表示法由 9 种图构成[19]，它们分别是：类图、对象图、用例图、顺序图、协作图、状态图、活动图、部署图以及组件图[20][21]。本章主要通过 UML 图对各个功能模块进行详细设计。

5.1 数据层类图

本小节论述的是系统的数据层类设计，由于系统对外的数据交互采用的都是 JSON 格式，为了方便数据类型转换，数据层的部分类是基于基础类进行封装的，只是多加了两个属性，本文在论述过程中称其为“数据封装类”。具体类图描述如图 5-1 和 5-2 所示。

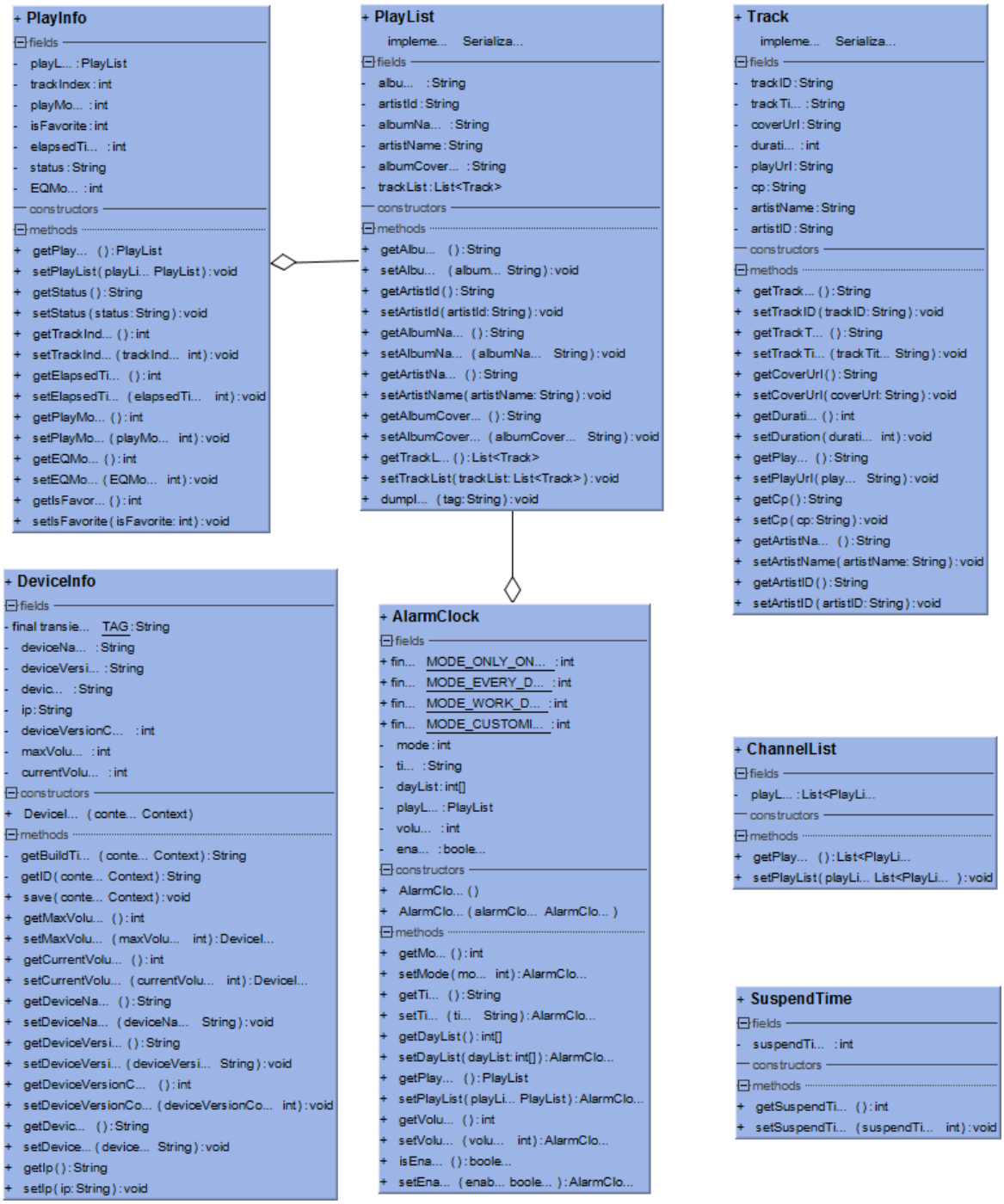
PlayInfo
playL... : PlayList
trackIndex : int
playMo... : int
isFavorite : int
elapsedTi... : int
status : String
EQMo... : int
getPlay... () : PlayList
setPlayList (playLi... PlayList) : void
getStatus () : String
setStatus (status: String) : void
getTrackInd... () : int
setTrackInd... (trackInd... int) : void
getElapsedTi... () : int
setElapsedTi... (elapsedTi... int) : void
getPlayMo... () : int
setPlayMo... (playMo... int) : void
getEQMo... () : int
setEQMo... (EQMo... int) : void
getIsFavor... () : int
setIsFavorite (isFavorite: int) : void
PlayList
impleme... Serializa...
albu... : String
artistId : String
albumNa... : String
artistName : String
albumCover... : String
trackList : List<Track>
getAlbu... () : String
setAlbu... (album... String) : void
getArtistId () : String
setArtistId (artistId: String) : void
getAlbumNa... () : String
setAlbumNa... (albumNa... String) : void
getArtistNa... () : String
setArtistName (artistName: String) : void
getAlbumCover... () : String
setAlbumCover... (albumCover... String) : void
getTrackL... () : List<Track>
setTrackList (trackList: List<Track>) : void
dumpI... (tag: String) : void
Track
impleme... Serializa...
trackID : String
trackTi... : String
coverUrl : String
durati... : int
playUrl : String
cp : String
artistName : String
artistID : String
getTrack... () : String
setTrackID (trackID: String) : void
getTrackT... () : String
setTrackTi... (trackTit... String) : void
getCoverUrl () : String
setCoverUrl (coverUrl: String) : void
getDurati... () : int
setDuration (durati... int) : void
getPlay... () : String
setPlayUrl (play... String) : void
getCp () : String
setCp (cp: String) : void
getArtistNa... () : String
setArtistName (artistName: String) : void
getArtistID () : String
setArtistID (artistID: String) : void
DeviceInfo
final transie... TAG : String
deviceNa... : String
deviceVersi... : String
devic... : String
ip : String
deviceVersionC... : int
maxVolu... : int
currentVolu... : int
DeviceI... (conte... Context)
getBuildTi... (conte... Context) : String
getID (conte... Context) : String
save (conte... Context) : void
getMaxVolu... () : int
setMaxVolu... (maxVolu... int) : DeviceI...
getCurrentVolu... () : int
setCurrentVolu... (currentVolu... int) : DeviceI...
getDeviceNa... () : String
setDeviceNa... (deviceNa... String) : void
getDeviceVersi... () : String
setDeviceVersi... (deviceVersi... String) : void
getDeviceVersionC... () : int
setDeviceVersionCo... (deviceVersionCo... int) : void
getDevic... () : String
setDevice... (device... String) : void
getIp () : String
setIp (ip: String) : void
AlarmClock
fin... MODE_ONLY_ON... : int
fin... MODE_EVERY_D... : int
fin... MODE_WORK_D... : int
fin... MODE_CUSTOMI... : int
mode : int
ti... : String
dayList : int[]
playL... : PlayList
volu... : int
ena... : boole...
AlarmClo... ()
AlarmClo... (alarmClo... AlarmClo...)
getMo... () : int
setMode (mo... int) : AlarmClo...
getTi... () : String
setTi... (ti... String) : AlarmClo...
getDayList () : int[]
setDayList (dayList: int[]) : AlarmClo...
getPlay... () : PlayList
setPlayList (playLi... PlayList) : AlarmClo...
getVolu... () : int
setVolu... (volu... int) : AlarmClo...
isEna... () : boole...
setEna... (enab... boole...) : AlarmClo...
ChannelList
playL... : List<PlayLi...
getPlay... () : List<PlayLi...
setPlayList (playLi... List<PlayLi...) : void
SuspendTime
suspendTi... : int
getSuspendTi... () : int
setSuspendTi... (suspendTi... int) : void

图 5-1 基础类类图

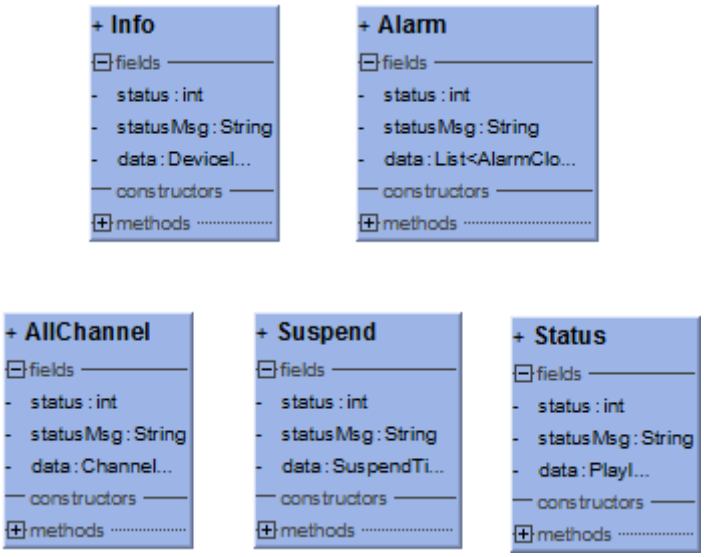

图 5-2 数据转换类图

5.2 投射源管理模块

在本文中，可以让音箱播放音乐的源称为“投射源”。本项目的投射源有 6 个，分别是本机投射，LifeKit 投射，DLNA 投射，闹钟投射，蓝牙投射，Aux in 投射。本机投射是在音箱启动后，音箱可以通过网络获取资源，使得音箱可以独立工作；LifeKit 是一个与音箱配套使用的远程控制软件，在投射源管理模块通过控制中心主动向设备发现模块注册一个事件处理类，从而对该投射源进行管理，DLNA 投射和闹钟投射也是通过主动注册进行管理的；而蓝牙投射和 Aux in 投射则是通过广播机制进行管理。

在该模块的设计中，涉及到两个接口：ISourceCallBack 和 IController，除了本机投射，每一个投射源对应一个类。具体类设计如图 5-3 所示，活动流程如图 5-4 所示。

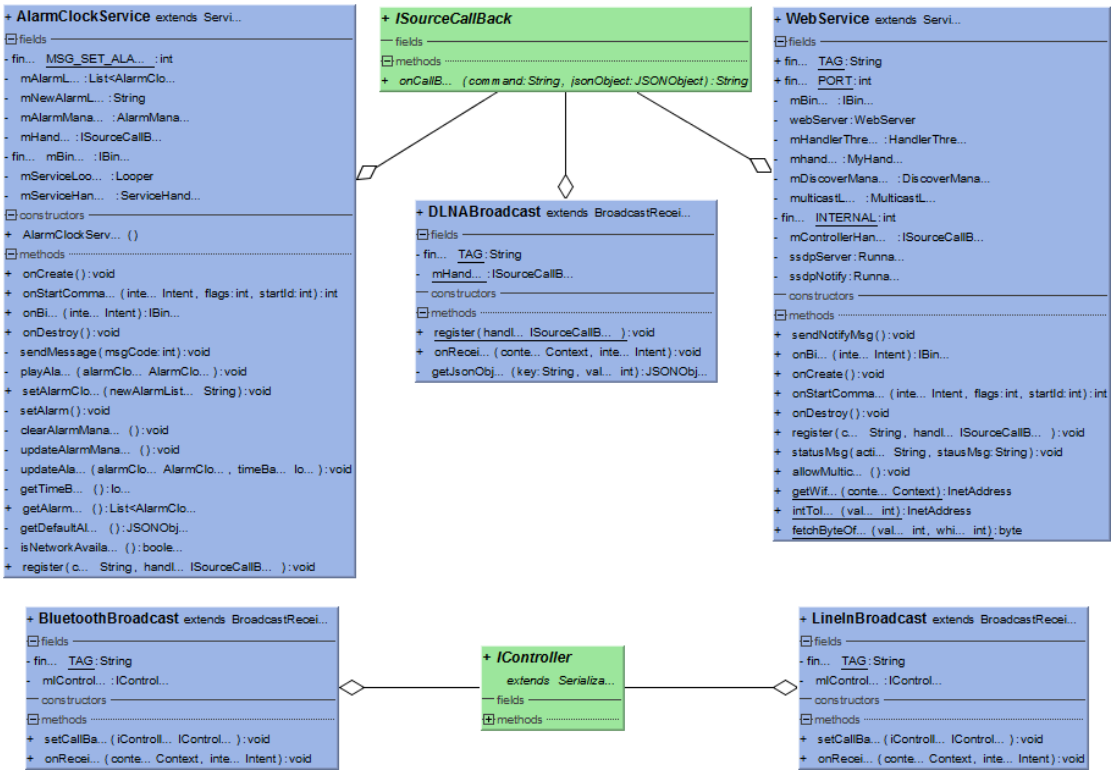

图 5-3 投射源管理类图

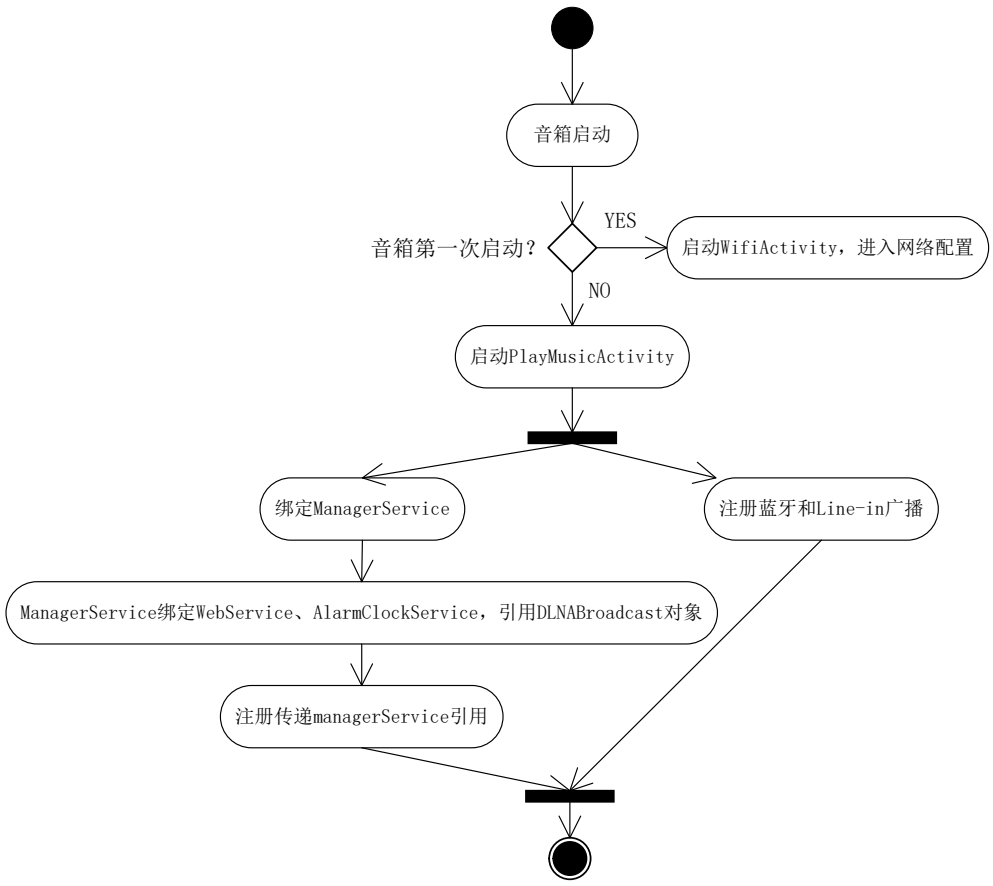

图 5-4 投射源管理活动图

5.3 远程命令控制模块

远程命令控制模块是与 LifeKit 的通信模块，主要是指用户可以通过 LifeKit 控制音箱，该模块通过 WebControllerHandler 和 ManagerService 两个控制类来实现，WebControllerHandler 实现 ISourceCallBack 接口，ManagerService 是项目的控制中心，继承于 Service，该模块的类设计如图 5-5 所示。针对于用户使用 LifeKit 与音箱进行通信的过程，通过序列图进行论述，如图 5-6 所示。

该模块涉及多个控制命令，每个命令的携带的参数以及音箱的应答内容均不相同详细控制命令接口定义见第 4.4 系统接口设计章节。

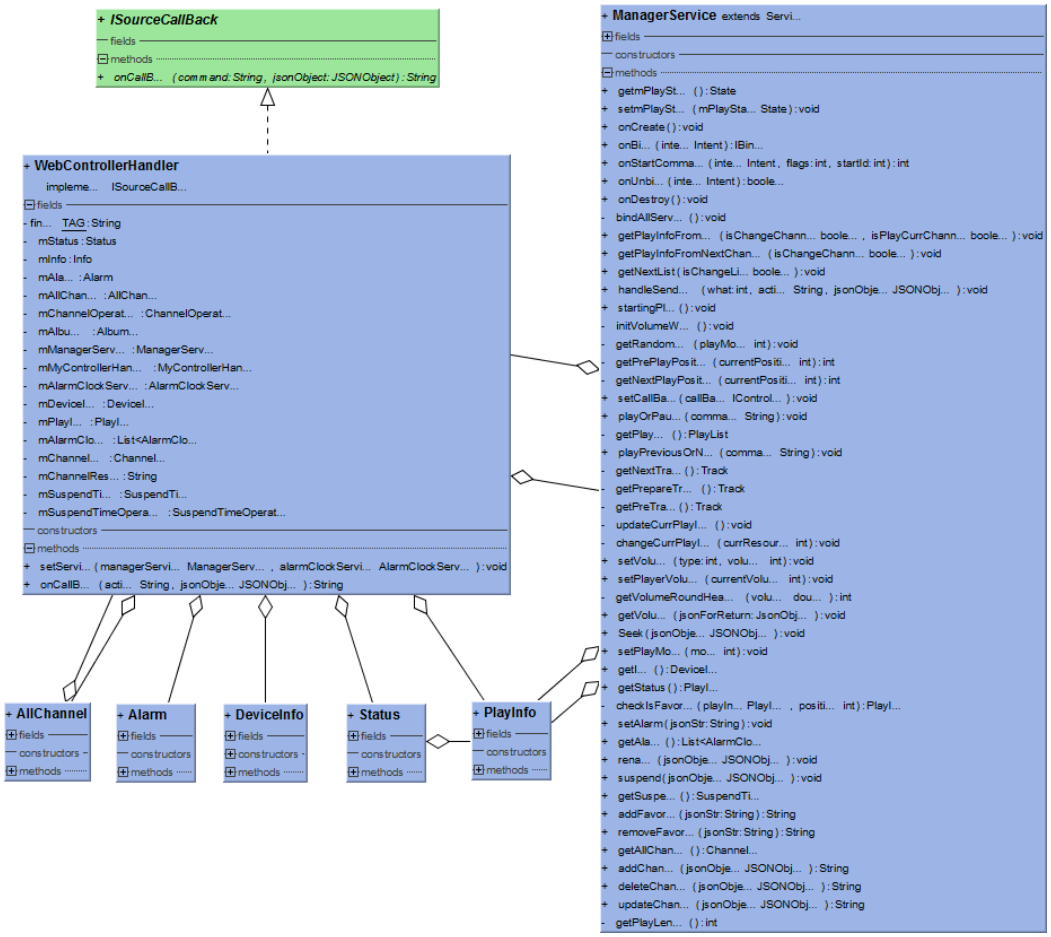

图 5-5 远程命令控制类图

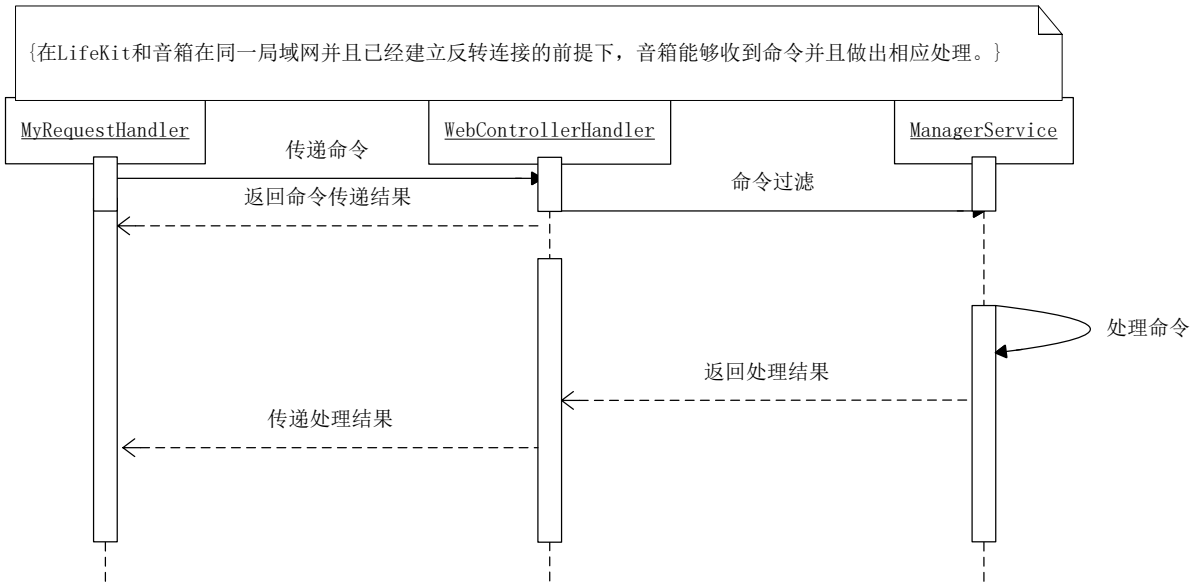

图 5-6 远程命令控制序列图

5.4 媒体播放控制模块

该模块主要负责播放逻辑控制，采用“立即处理”原则，当一个投射源投射歌曲，系统将立即播放。区别去 LifeKit 投射歌曲，DLNA 和闹钟投射在播放完投射曲目后，系统将立即恢复之前的播放状态，包括 UI 显示。

该模块主要涉及两个数据类 PlayInfo 和 PlayList，这两个类包含了曲目的所有信息。对于逻辑的控制，主要是由 MyControllerHandler 这一内部类进行管理，这个类继承自 Handler 系统类，重写了 handleMessage()方法，更加详尽的类信息如图 5-7 所示。对播放控制的流程，如图 5-8 所示。

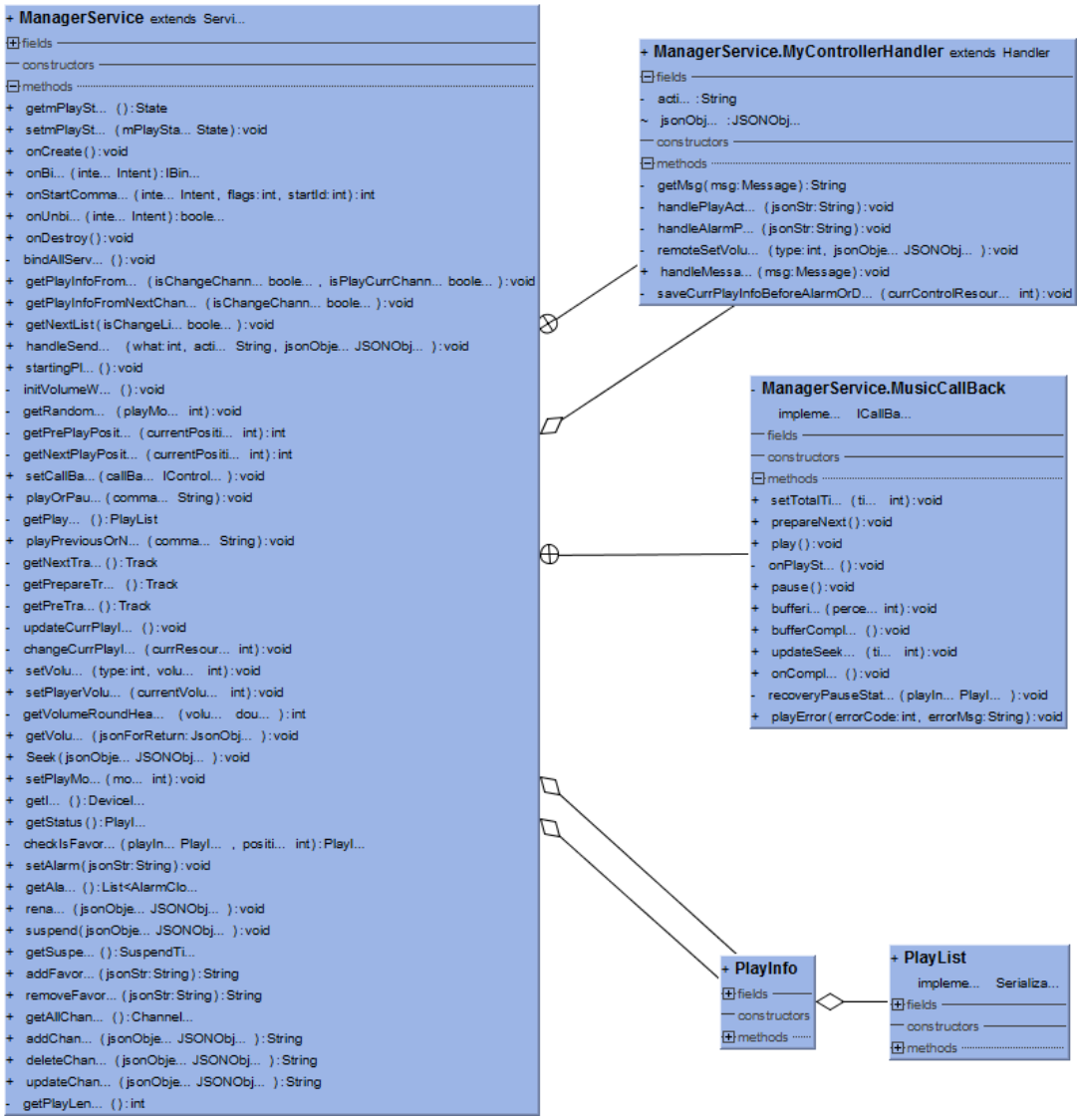
+ ManagerService extends Servi...
fields
constructors
methods
+ getmPlaySt... ( ) : State
+ setmPlaySt... (mPlaySta... State) : void
+ onCreate ( ) : void
+ onBi... (inte... Intent) : IBin...
+ onStartComma... (inte... Intent , flags: int , startId: int) : int
+ onUnbi... (inte... Intent) : boole...
+ onDestroy ( ) : void
- bindAllServ... ( ) : void
+ getPlayInfoFrom... (isChangeChann... boole... , isPlayCurrChann... boole...) : void
+ getPlayInfoFromNextChan... (isChangeChann... boole...) : void
+ getNextList (isChangeLi... boole...) : void
+ handleSend... (what: int , acti... String , jsonObje... JSONObj...) : void
+ startingPl... ( ) : void
- initVolumeW... ( ) : void
- getRandom... (playMo... int) : void
- getPrePlayPosit... (currentPositi... int) : int
- getNextPlayPosit... (currentPositi... int) : int
+ setCallBa... (callBa... IControl...) : void
+ playOrPau... (comma... String) : void
- getPlay... ( ) : PlayList
+ playPreviousOrN... (comma... String) : void
- getNextTra... ( ) : Track
- getPrepareTr... ( ) : Track
- getPreTra... ( ) : Track
- updateCurrPlayI... ( ) : void
- changeCurrPlayI... (currResour... int) : void
+ setVolu... (type: int , volu... int) : void
+ setPlayerVolu... (currentVolu... int) : void
- getVolumeRoundHea... (volu... dou...) : int
+ getVolu... (jsonForReturn: JsonObj...) : void
+ Seek (jsonObje... JSONObj...) : void
+ setPlayMo... (mo... int) : void
+ getI... ( ) : DeviceI...
+ getStatus ( ) : PlayI...
- checkIsFavor... (playIn... PlayI... , positi... int) : PlayI...
+ setAlarm (jsonStr: String) : void
+ getAla... ( ) : List<AlarmClo...
+ rena... (jsonObje... JSONObj...) : void
+ suspend (jsonObje... JSONObj...) : void
+ getSuspe... ( ) : SuspendTi...
+ addFavor... (jsonStr: String) : String
+ removeFavor... (jsonStr: String) : String
+ getAllChan... ( ) : ChannelI...
+ addChan... (jsonObje... JSONObj...) : String
+ deleteChan... (jsonObje... JSONObj...) : String
+ updateChan... (jsonObje... JSONObj...) : String
- getPlayLen... ( ) : int
+ ManagerService.MyControllerHandler extends Handler
fields
- acti... : String
~ jsonObj... : JSONObj...
constructors
methods
- getMsg (msg: Message) : String
- handlePlayAct... (jsonStr: String) : void
- handleAlarmP... (jsonStr: String) : void
- remoteSetVolu... (type: int , jsonObje... JSONObj...) : void
+ handleMessa... (msg: Message) : void
- saveCurrPlayInfoBeforeAlarmOrD... (currControlResour... int) : void
- ManagerService.MusicCallBack
impleme... ICallBa...
fields
constructors
methods
+ setTotalTi... (ti... int) : void
+ prepareNext ( ) : void
+ play ( ) : void
- onPlaySt... ( ) : void
+ pause ( ) : void
+ bufferi... (perce... int) : void
+ bufferCompl... ( ) : void
+ updateSeek... (ti... int) : void
+ onCompl... ( ) : void
- recoveryPauseStat... (playIn... PlayI...) : void
+ playError (errorCode: int , errorMsg: String) : void
+ PlayInfo
fields
constructors
methods
+ PlayList
impleme... Serializa...
fields
constructors
methods

图 5-7 播放控制类图

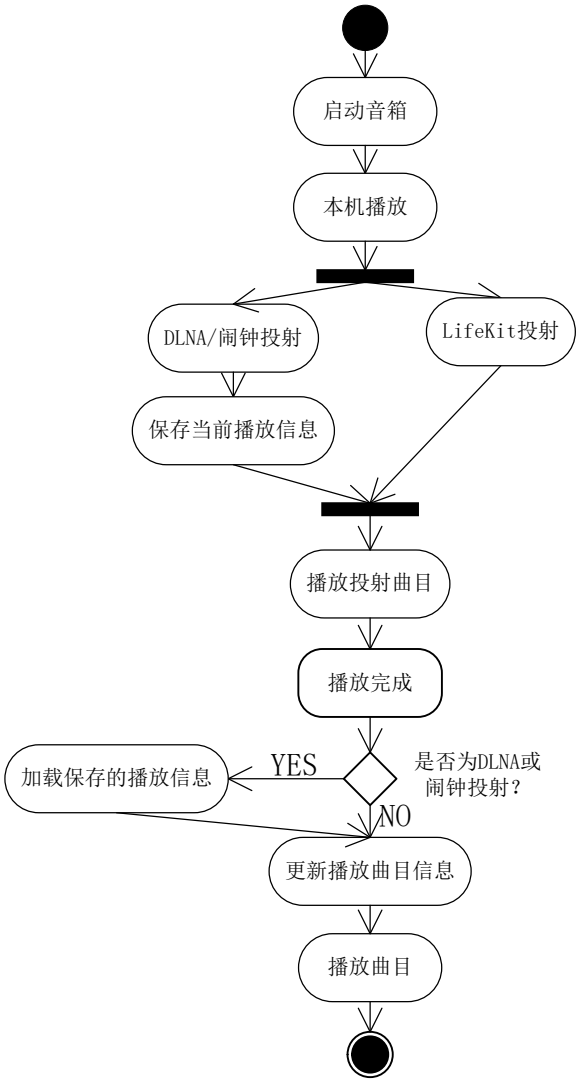

图 5-8 媒体播放控制活动图

5.5 播放状态管理

播放状态模块定义的播放状态 5 个，分别是 IDLE，Buffering，Started，Paused，Error。播放状态由 MusicCallBack 这一控制类进行管理，它实现 IcallBack 这个接口，由 MusicPlayer、DLNAPlayer 以及 SpotifyPlayer 对象进行回调通知播放状态。MusicCallBack 类收到回调后通知 WebService 当前播放状态，并且通过网络传递到 LifeKit 端。具体类设计如图 5-9 所示，播放状态间的迁移通过 UML 状态图来论述，如图 5-10 所示。

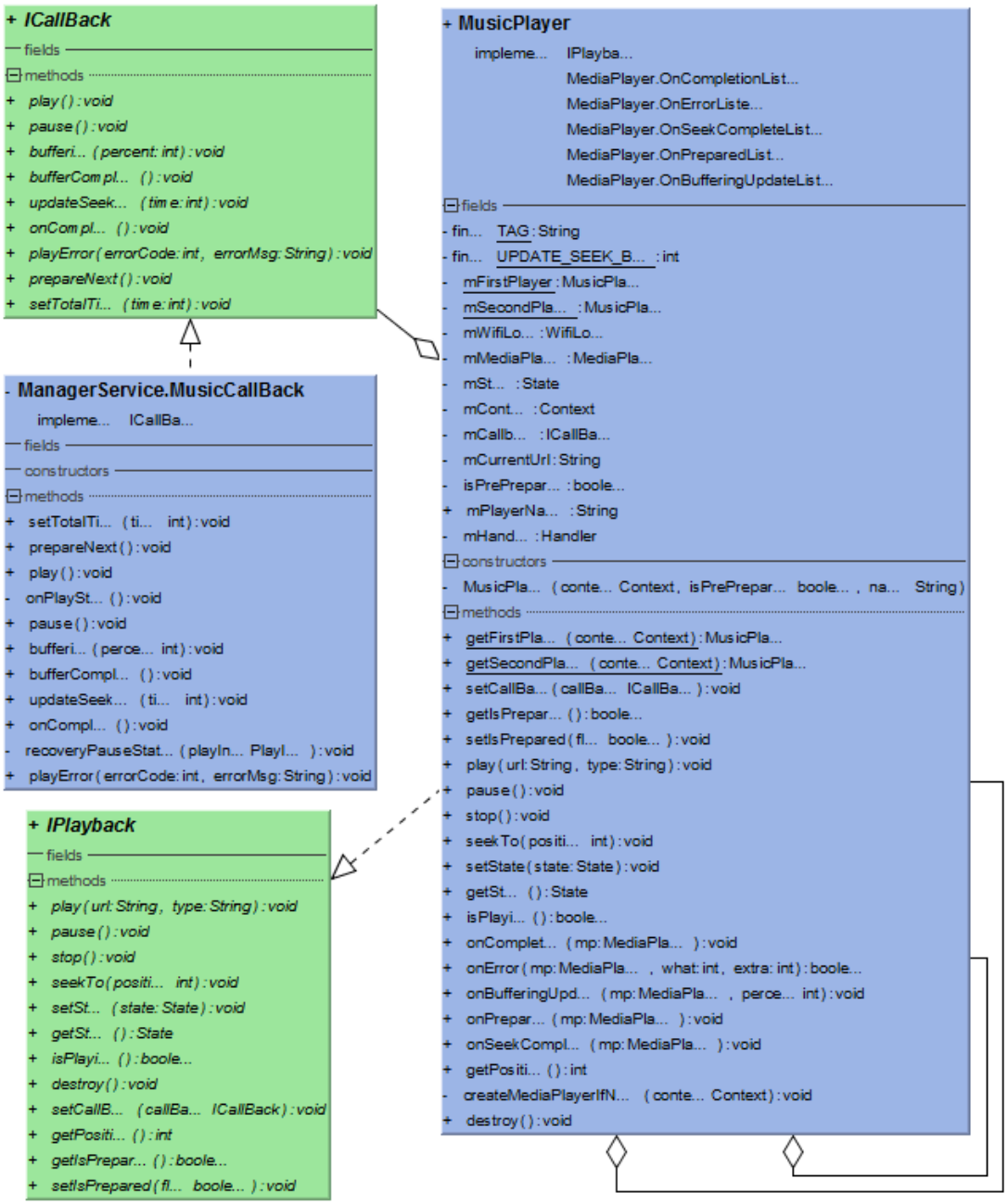

图 5-9 播放状态管理类图

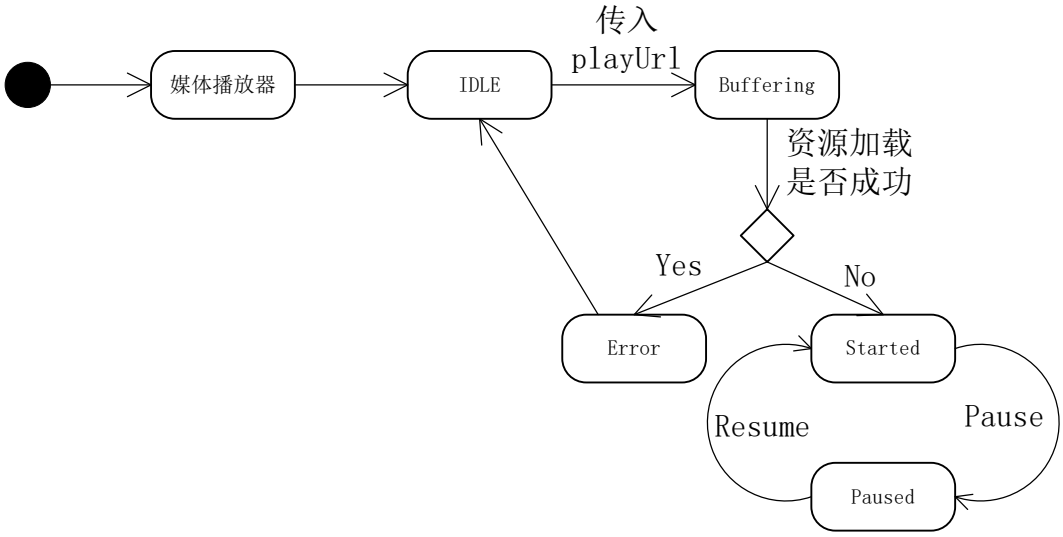

图 5-10 播放状态管理状态图

5.6 网络状态监测模块

网络状态会直接影响播放状态，因此网络状态需要实时监测。此模块采用广播来监测网络连接状态以及信号状态。WifiSignalReceiver 是该模块的控制类，它继承自 BroadcastReceiver，使用动态注册。Android 的广播使用了设计模式中的观察者模式，达到解耦的目的。该模块的具体类设计如图 5-11 所示。

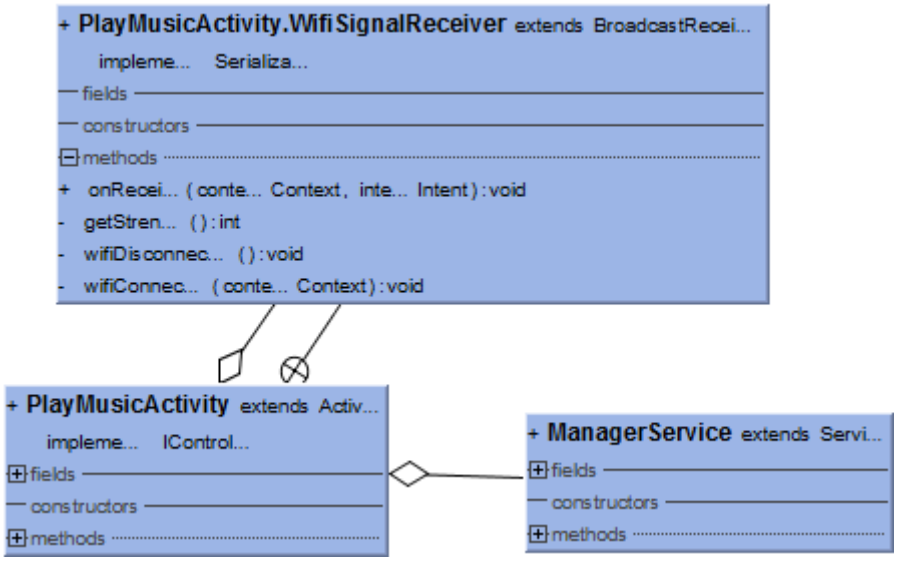

图 5-11 网络状态监测类图

5.7 用户界面管理模块

在 A8 Wi-Fi 音箱项目中，用户界面只是为了显示信息，用户无需使用界面来进行数据交互。由于这一区别于其他 Application 的特点，在用户界面管理这一模块除了播放信息的展示，其他信息展示均使用 Dialog，并且定义了展现时间，超过时间会自动消失。这一模块涉及的 Dialog 很多，包括频道切换显示，设备信息显示，Wi-Fi 状态改变显示，重设音箱名字显示，闹钟显示等等。详细的类设计以及类关系如图 5-12 所示，控制流程如图 5-13 所示。

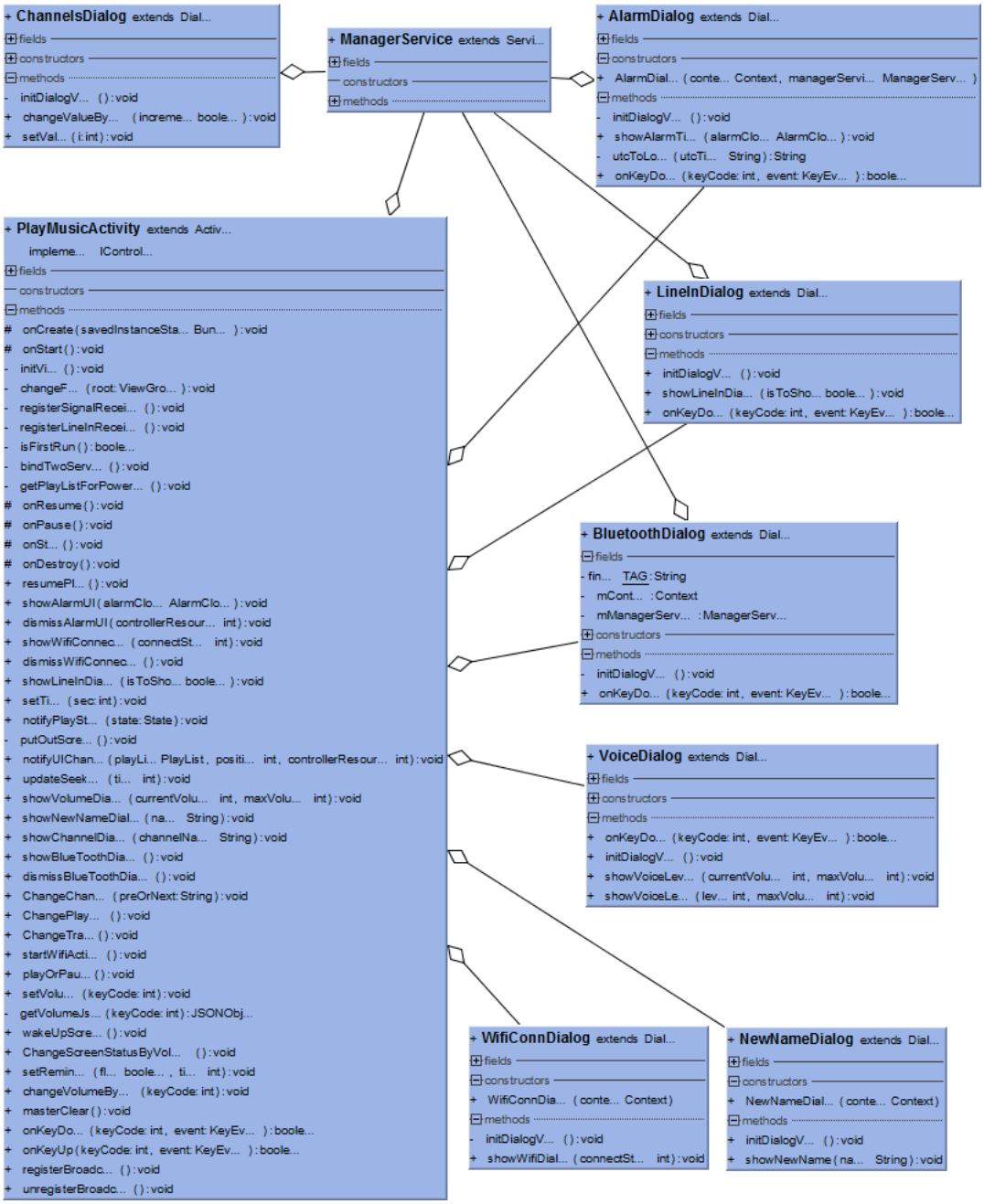

图 5-12 用户界面管理类图

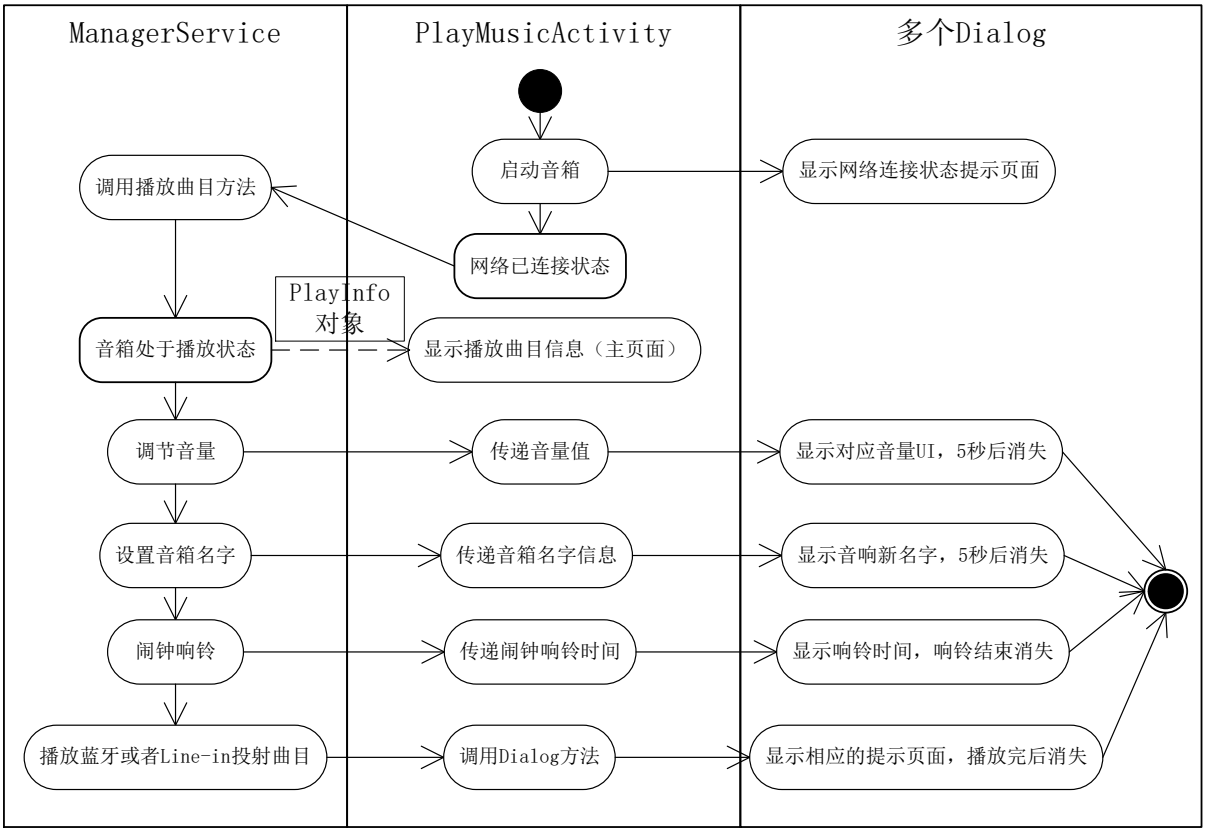

图 5-13 用户界面管理类图

5.8 物理按键控制模块

系统的物理按键有 4 个，分别是音量加、音量减、Play 键以及 Next 键。这四个按键的操作方式有短按、双击、长按，并且音量加和音量减是组合按键，用来强制重置网络信息。用户按下物理按键，PlayMusicActivity 或者其他 Dialog 通过 onKeyDown 事件进行监听，然后将事件的处理转向工具类 KeyUtil。所有操作方式的判断均在工具类里进行判断。具体设计如图 5-14 所示。

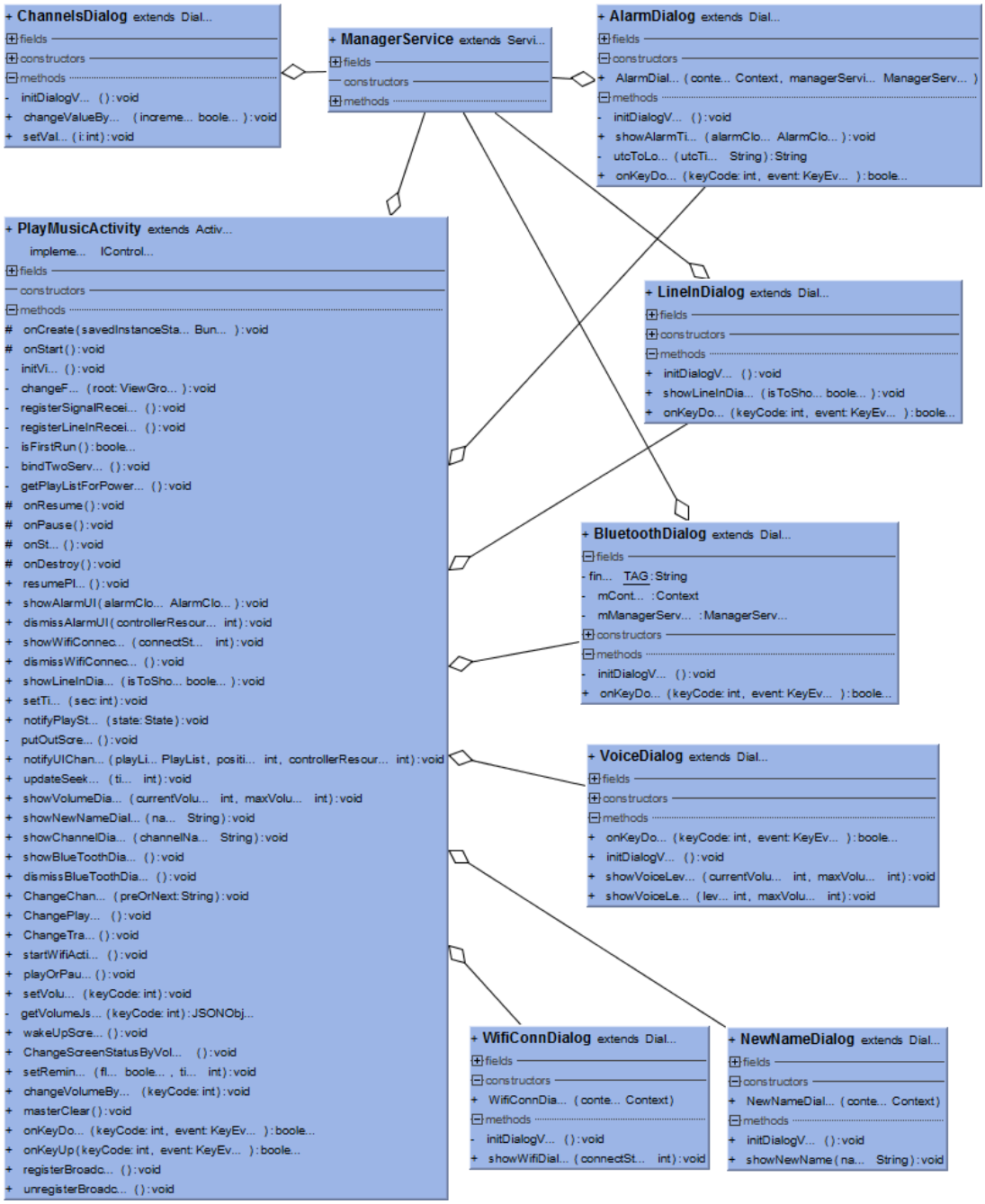

图 5-14 物理按键控制类图

5.9 本章小结

本章充分利用 UML 建模来对系统功能进行设计，在设计上本文尽量将类进行单一化，一个控制类尽量只负责一件事，能够降低耦合性。在系统内部之间的接口设计，统一入口，尽可能使用局部变量，避免多处改变属性值。根据概要设计和详细设计，已经完成了项目的编码。

6 测试方案以及运行效果

按照软件开发的规范流程本文已经完成了需求分析、系统架构设计、系统详细设计与实现，本章将从测试方案和运行效果进行论述。

6.1 测试方案

测试方案将从 Bug 等级划分标准、Bug 状态划分标准、问题类别和测试日程安排进行介绍，此项目采用公司内容的缺陷管理系统进行管理，由测试人员提交 Bug，相关 Bug 的负责人将领取任务。

6.1.1 Bug 等级划分标准

此项目中，Bug 等级分为 4 个级别，具体划分信息以表格形式呈现，如下表。

表 6-1 Bug 等级划分标准

No	Type	Detail	Comment	Redime
1	Blocker（崩溃）	（1）主要功能丧失，基本模块缺失等问题；（2）造成系统崩溃、死机、死循环；（3）阻碍开发或测试工作的问题；（4）导致数据库数据丢失，与数据库连接错误；。如：代码错误、死循环、数据库发生死锁等。	该问题在测试中较少出现，一旦出现应立即中止当前版本测试	紧急
2	Critical（严重）	（1）功能设计与需求严重不符；（2）数据库保存调用错误、用户数据丢失；（3）一级功能菜单不能使用但是不影响其他功能的测试；（4）模块无法启动或调用；（5）安全问题、稳定性等。如：用户所要求的功能缺失，软件中数据保存后数据库中显示错误，程序接口错误等。	该等级问题出现在不影响其他功能测试的情况下可以继续该版本测试	高

表 6-1（续） Bug 等级划分标准

No	Type	Detail	Comment	Redime
3	Major（一般）	（1）功能没有完全实现但是不影响使用；（2）功能菜单存在缺陷但不会影响系统稳定性。如：操作时间长等。	该问题实际测试中存在最多，合理安排解决BUG，解决率关系版本的优化程度	普通
4	Minor（次要）	（1）界面缺陷但不影响操作功能的执行；（2）性能缺陷；如：错别字、界面格式不规范，描述不清楚，提示语与文字不符，文字排列不整齐，用户体验感受不好，可以优化性能的方案等。	此类问题在测试初期较多，优先程度较低；在测试后期出现较少，应及时处理	低

6.1.2 Bug 状态划分标准

在缺陷管理系统管理 Bug 有 7 个状态，分别是待处理、已确认、已解决、已关闭、重新打开、不是问题、暂不处理。下面以表格形式呈现更详细的内容。

表 6-2 Bug 状态划分标准

Type	Detail	Object
待处理	发现新问题后提交的状态	任意角色
已确认	经测试人员和研发人员讨论后确认是 BUG	测试人员
已解决	经研发人员确定 BUG 修复的状态，还未验证	开发人员
已关闭	测试人员认为问题已修改，通过验证	测试人员
重新打开	测试人员认为 BUG 未修复成功，BUG 重新打开	测试人员
不是问题	研发人员确认不是 BUG，或者不采纳该建议	开发人员
暂不处理	当前版本不做处理，后续版本再考虑	开发人员

6.1.3 问题类别

根据项目特点，测试方案里将问题进行分类，具体分类描述见表 6-3 问题分类。

表 6-3 问题分类

Type	Detail
Wi-Fi Connection	Wi-Fi 与音箱连接的各状态的异常类
Aux-in Connection	Aux-in 与音箱连接的各状态的异常类
BlueTooth Connection	蓝牙与音箱连接的各状态的异常类
UI	画面迁移，显示异常
Caution	提示信息异常（文字，提示信息，声音）
Key Event	音箱物理按键操作异常
式样规格	实际动作与式样存在差异（播放控制和 Wi-Filianjie ）

6.1.4 测试用例

对于本项目的测试，将通过验收测试、系统测试、兼容性测试以及弱点强化测试进行全面测试，力争完成高质量的产品。由于项目还在测试中，在本小节将给出部分验收测试用例如表 6-4 所示。

表 6-4 Aux in 时按键操作用测试例

控件名称：	**音箱屏幕显示和按键（Aux in 时按键操作）**		
前置条件	1.接入音箱电源 2.aux in 接口接入 3.aux in 切换提示结束		
编号	**输入/动作**	**期望的输出/相应**	**实际情况**
1	1.按一下[音量-]	1.播放声音减小，且无调节声音提示音 2.时间和日期提示切换至声音调节提示 3.声音调节提示 15 秒后消失，再次显示时间和日期提示	
2	1.按一下[播放]	1.显示时间和日期提示 2.屏幕和声音均无变化	

表 6-5 物理按键操作用测试例

控件名称：	音箱屏幕显示和按键（播放时按键操作）		
前置条件	1.接入音箱电源 2.音箱与 Wi-Fi 连接 3.正在播放音箱内置网络音乐		
编号	输入/动作	期望的输出/相应	实际情况
1	1.按一下音箱的[播放/暂停]物理按键	1.网络音乐播放界面中，显示当前播放信息 2.播放进度停留在按下暂停的一刻(显示被暂停的时刻）	
2	1.按一下音箱的[音量+]/[音量-]物理按键	1.网络音乐播放界面切换至音量调节界面 2.无调节音量的音效 3.播放音乐同时，音量调节界面显示 15 秒后消失，切换至网络音乐播放界面	
3	1.多次按下音箱的[NEXT]物理按键，直至成为一个循环	1.网络音乐播放界面上提示频道切换信息，且音乐播放停止 2.频道切换信息界面显示 15 秒后消失 3.播放切换至所切换的频道的第一首音乐 4.保存的频道依次循环切换	
4	1.记录当前音乐播放信息 2.拔插音箱电源 3.等待开机画面结束	1.显示并播放被记录的音乐信息和内容（最后一次播放的音乐）	
5	1.插拔与音箱连接的 Wi-fi 路由电源 2.等待 Wi-Fi 连接成功提示消失	1.网络重连后，播放的音乐信息和内容与断网前播放的一致。	
6	1.同时按下【音量+】和【音量-】	1.音箱进入 Wi-Fi 进连接等待提示（文字，语音）	

表 6-6 物理按键操作测试用例

控件名称：	音箱屏幕显示和按键（闹钟设置）		
前置条件	1.接入音箱电源 2.音箱连接 Wi-Fi 3.手机连接与音箱同网段 Wi-Fi 4.已设置音箱闹钟 5.音箱处于低功耗状态		
编号	输入/动作	期望的输出/相应	实际情况
1	1.等待闹钟时间到达	1.屏幕背光点亮 2.音乐播放界面切换至闹铃唤醒页面	
2	1.断开 Wi-fi 路由电源 2.等待闹钟时间到达	1.屏幕背光点亮 2.音乐播放界面切换至闹铃唤醒页面	
4	1.打开某个歌单详细页 2.选择歌曲 2.点击右下角【+】	1.显示【添加至】歌单列表页面	
5	1.手机端对音箱进行修改命名操作	1.修改成功，屏幕显示音箱新命名 5 秒后退出该页面	

表 6-7 远程命令控制测试用例

控件名称：	远程命令控制		
前置条件	1.音箱 A 与 Wi-Fi 路由 A 连接 2.手机与 Wi-Fi 路由 A 连接 3.打开手机 App		
编号	输入/动作	期望的输出/相应	实际情况
1	1.打开喜马拉雅首页->歌单列表页 2.点选歌单，点击全部播放	1.所选歌单内歌曲存在播放列表中，且可依照歌单内歌曲排列顺序播放 2.歌单内最后一首歌曲播放完毕后，再次播放第一首歌曲	
2	1.打开虾米首页->->歌单列表页 2.点选收藏歌单	1.在手机【我的】页面内存在该歌单	
3	1.打开【我的】页面 2.点击添加歌单->为 歌单命名->完成创建。	1.在手机【我的】页面内存在该被创建的歌单	
4	1.打开某个歌单详细页 2.选择歌曲 2.点击右下角【+】	1.显示【添加至】歌单列表页面	
5	1.手机端对音箱进行修改命名操作	1.修改成功，屏幕显示音箱新命名 5 秒后退出该页面	

表 6-8 音箱默认播放逻辑测试用例

控件名称：	音箱默认播放逻辑		
前置条件	1.音箱 A 与 Wi-Fi 路由 A 连接 2.手机与 Wi-Fi 路由 A 连接 3.打开手机 App 4.手机 App 中存在 1 个以上用户自建的歌单		
编号	输入/动作	期望的输出/相应	实际情况
1	1.打开虾米首页->歌单列表页->歌单详情页 2.点选播放该歌单内全部歌曲 3.重启 Wi-Fi 路由 A 4.等待音箱连接 Wi-fi 成功	1.音箱当前播放歌单为断网前所选歌单曲目。	
2	1.打开虾米首页->歌单列表页->歌单详情页 2.点选播放该歌单内全部歌曲 3.重启音箱 4.等待音箱连接 Wi-fi 成功	1.音箱当前播放歌单为音箱重启前所选歌单曲目。	
3	1.打开虾米首页->歌单列表页->歌单详情页 2.点选播放该歌单内全部歌曲 3.手机 A 断开 Wi-Fi，手机 B 连接 Wi-Fi 且控制该音箱	1.音箱当前播放歌单为手机 A 所选歌单曲目。	
4	1.打开虾米首页->歌单列表页->歌单详情页 2.点选播放该歌单内全部歌曲 3.手机 A 断开 Wi-Fi，手机 B 连接 Wi-Fi 且控制该音箱 4.打开虾米首页->歌单列表页->歌单详情页 5.点选播放该歌单内全部歌曲	1.音箱当前播放歌单为手机 B 所选歌单曲目。	
5	1.打开虾米首页->歌单列表页->歌单详情页 2.点选播放该歌单内全部歌曲	1.音箱当前播放歌单所选歌单曲目。 2.该歌单所有歌曲播放完毕后，依次播放手机与音箱同步的自建歌单和音箱手机非显示的默认歌单 3.手机与音箱同步的自建歌单和音箱手机非显示的默认歌单的音乐播放完毕后，再次播放该所选歌单内音乐	

表 6-9 LifeKit 播放控制测试用例

控件名称：	**LifeKit 播放控制**		
前置条件	1.音箱连接 wifi 2.手机连接音箱 3.音箱正在播放音乐		
编号	**输入/动作**	**期望的输出/相应**	**实际情况**
1	1.打开手机 app 音乐播放页面 2.点击手机 app 音乐播放页面中的播放或者暂停、上一曲、下一曲、快进或乱序播放	1.手机 app 音乐播放页面显示实时状态，根据播放或者暂停、上一曲、下一曲、快进或者快进、乱序播放的动作实时响应 2.音箱播放内容根据手机实时动作响应	
2	1.打开手机 app 2.打开音箱播放列表 3.设置顺序或乱序播放 4.停留当前播放列表页面，记录音箱播放音乐顺序	1.当前播放的音乐结束后，下一首播放播放列表的音乐 2.在播放列表中显示当前播放音乐列表中的位置 3.音箱顺序或乱序播放时，播放列表实时显示当前播放音乐在列表中的位置	
3	1.打开手机 app 2.任意选择一个歌单 3.手机 app 打开音箱播放列表 4.删除播放列表内音乐	1.当前播放的音乐结束后，下一首播放播放列表的音乐 2.手机 app 的播放列表显示删除后的曲目 3.音箱当前播放为所选音乐，下一首则根据播放顺序或乱序播放，播放列表实时显示当前播放音乐在列表中的位置	

6.2 运行效果

由于产品物理结构上设计，该产品的屏幕宽 320px，高 240px。在进行 UI 设计时，总体风格定义为简约系。播放页面主要用来展示曲目的重点信息，包括曲目名称，专辑名称，歌手名字，内容来源，专辑封面以及播放进度条，运行效果如图 6-1 所示。

图 6-1 主播放界面实际效果图

当前播放为 Aux-in 接入播放时，则显示系统当前时间，及时更新页面时间显示，如图 6-2 所示。

图 6-2 Aux in 播放界面实际效果图

如果当前播放为蓝牙投射播放，则显示为一个有蓝牙图标的图片，如图 6-3 所示。

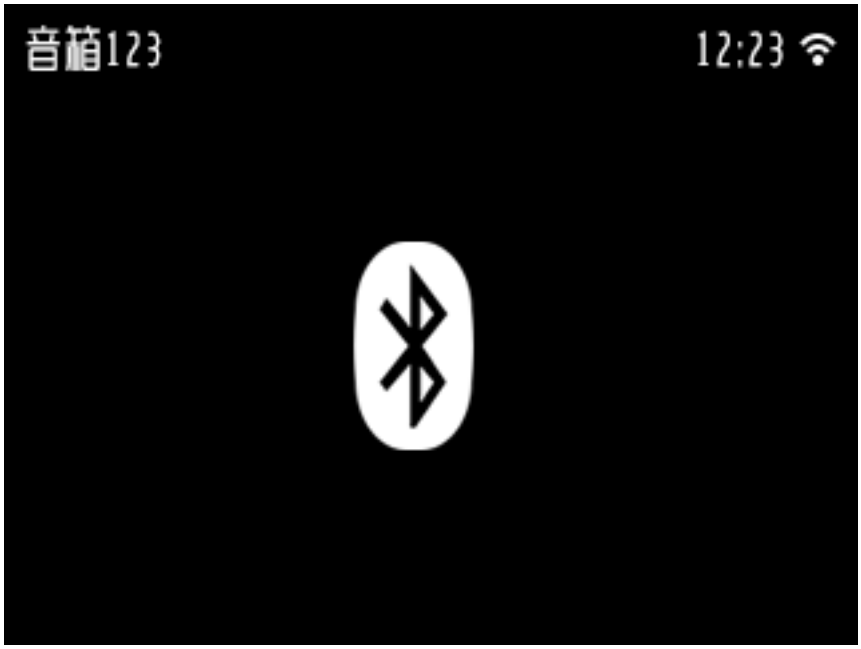

图 6-3 蓝牙播放界面实际效果图

当闹钟响铃时，音箱显示为闹铃时间，当闹铃歌曲播放完毕该页面自动消失，并且恢复响铃之前的显示，闹铃响铃实际界面效果如图 6-4 所示。

图 6-4 闹钟响铃界面实际效果图

用户可以通过多种方式调节音量，当用户使用物理按键调节音量时，音量条则根据用户按键按下次数进行变化，即用户按下一次音量键，则音量条向左或者向右变化一格。音量调节界面如图 6-5 所示。

图 6-5 音量调节界面实际效果图

用户可以通过 LifeKit 设置音箱名字，当用户设置名字成功时，音箱会通过显示新名字来提示用户此次修改已成功，该显示页面显示 5 秒自动消失，实际效果如图 6-6 所示。

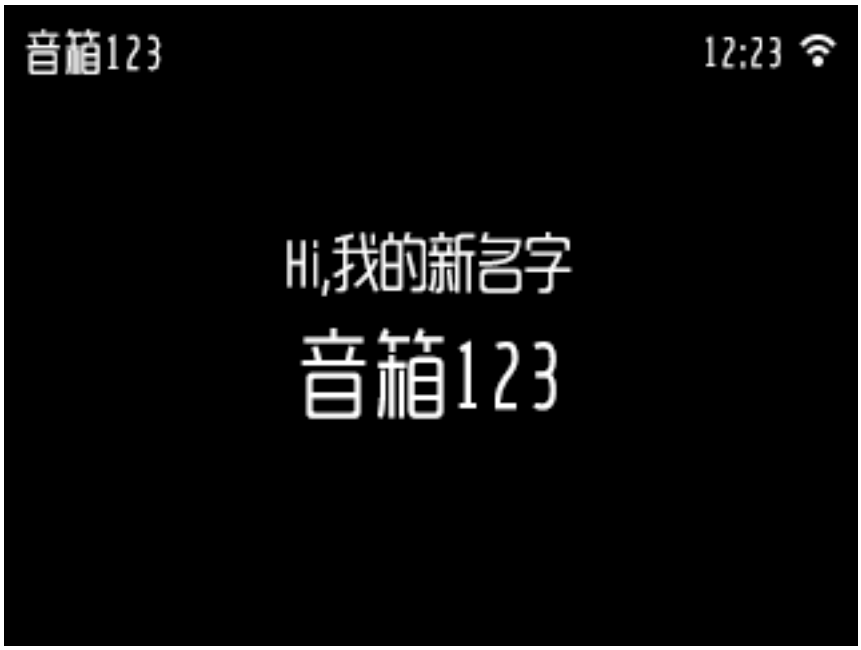

图 6-6 设置名字界面实际效果图

当用户按下 Next 物理键时，则显示切换频道界面，显示该音箱用户所有频道名称，用户可通过音量加减键向前或向后切换频道，实际效果如图 6-7 所示。

图 6-7 切换频道界面实际效果图

6.3 本章小结

本章主要从缺陷划分标准、测试用例以及运行效果进行论述。在需求分析、概要设计以及详细设计的基础上，项目完成了编码工作。本章主要是从功能性方面验证项目实现效果。

7 系统难点问题分析

本章主要是针对在系统开发实现中遇到的复杂业务逻辑问题进行分析，将从问题描述、解决方案和代码验证三个方面来介绍。

7.1 音箱启动立即播放

问题描述：根据需求，在音箱开机启动后，音箱会继续播放上一次关机前系统个记录的播放频道曲目。由于曲目资源均为网络资源，因此音箱播放的一个前提条件是“网络已连接”。在系统设计时，整个项目的主入口是 PlayMusicActivity，媒体播放作为需要长期运行的后台任务，使用 Service 来负责。入口 Activity 与控制中心 ManagerService 绑定，其中控制中心 ManagerService 还与负责媒体播放的 Service 也就是 MusicService 进行绑定，通过调用播放方法，传递参数实现媒体播放。通过分析，我们认为影响播放的条件是：网络已连接，ManagerService 已绑定，PlayInfo 不为空，MusicService 已绑定，只有当这四个条件均满足音箱才能开始播放。其中 PlayInfo 的获取以及两个 Service 的绑定是异步操作。由于影响因素复杂，在开发实现这个需求时，遇到了困难，最后通过与项目经理的讨论分析，最后寻找到了解决方案。

解决方案：通过问题描述，我们得知影响因素有 4 个，并且在 PlayMusicActivity 里存在两个影响因素（一级影响因素），而另外两个均在 ManagerService 中（二级影响因素）。解决方案是：将这一需求统一用一个方法来实现，这个方法分为两层调用，分别拦截两组影响因素。在 PlayMusicActivity 中，判断 ManagerService 是否已绑定和网络是否连接（用一个工具类判断），在 ManagerService 里封装一个方法，判断 PlayInfo 是否为空和 MusicService 是否已绑定。在 PlayActivity 中封装一个 resumePlay()方法，当一级影响因素符合条件时，调用 ManagerService 中的 startingPlay()方法，此方法判断二级影响因素是否已经满足条件。在改变这四个影响因素状态的地方，统一调用 resumePlay()方法。

代码验证：从分析问题到找到解决方法，最后通过具体代码来验证方案的正确性。代码如图 7-1 和图 7-2 所示

```
@Override
public void resumePlay() {
    Log.i(TAG, "resumePlay");
    if (isManagereServiceBind && NetUtils.isWifiConnected(this)) {
        if (!mPlayState.equals(MusicService.State.STARTED.toString()) &&
                !mPlayState.equals(MusicService.State.PAUSED.toString())) {
            Log.i(TAG, "resumePlay里调用startingPlay");
            mManagerService.startingPlay();
        } else if (toBeRemuse) {
            Log.i(TAG, "resumePlay里调用Resume");
            mManagerService.handleSendMsg(Config.LOCAL, Config.RESUME, null);
            toBeRemuse = false;
        }
    }
}
```

图 7-1 resumePlay()方法

```
public void startingPlay() {
    if (isMusicServiceBind == true && mCurrPlayInfo != null) {
        try {
            JSONObject jsonObject = new JSONObject(GsonUtils.toJson(mCurrPlayInfo));
            String action = Config.PLAY;
            handleSendMsg(Config.LOCAL, action, jsonObject);
            isStartingPlay = true;
            Log.i(TAG, "startingPlay播放音乐");
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```

图 7-2 startingPlay()方法

7.2 随机播放模式曲目选取

问题描述：对于随机播放模式的取歌方式，现有的音乐平台如网易云音乐、QQ 音乐、酷狗音乐等每一家平台都有不同的随机选曲方法。如何确保上下曲目不会与当前曲目重复是随机模式下的最重要的问题。

解决方案：针对上述问题，本项目最终采用的是根据当前歌单曲目数量 n，创建一个从 0 至 n-1 的 ArrayList，对 List 进行随机洗牌，采用 Java 中 Collections 类中的 shuffle

方法。对了保证曲目不重复，在随机模式下选取上下曲目时，判断当前播放曲目在随机列表中的下标 index，上下曲目的播放位置分别为随机列表中下标为 index-1 和 index+1 的值。由于当前下标 index 对应的值为当前播放位置，所以一定能够保证 index-1 和 index+1 对应的值与当前播放位置不同。

代码验证：根据解决方案的思路，在本项目最后代码实现如图 7-3 所示。

```
private void getRandomList(int playMode) {
    if (playMode == Config.SHUFFLE) {
        playPositionList = new ArrayList<>();
        for (int i = 0; i < mCurrPlayInfo.getPlayList().getTrackList().size(); i++) {
            playPositionList.add(i);
        }
        Collections.shuffle(playPositionList);
        Log.i(TAG, "getRandomList random: " + playPositionList.toString());
    }
}
```

图 7-3 getRandomList()方法

7.3 歌单获取

问题描述：本项目总共定义了 4 种播放模式：随机播放、顺序播放、列表循环以及单曲播放。歌曲获取不仅与播放模式有关，还有当前投射源有关。当歌单播放到最后一曲时，如果投射源为本机投射和 LifeKit 投射，则需要根据播放模式选择继续播放当前歌单或者重新获取歌单，而如果是闹钟投射或者 DLNA 投射，则需要切换为之前的歌单。因此用户可以通过手机 LifeKit 控制上下曲，因此歌单最后一曲的判断需要在两个地方进行检测。投射源的多样性，播放模式的多样性以及最后一曲判断均是使得该问题复杂的原因。

解决方案：统一最后一曲判断的入口，在每个需要判断当前曲目是否是当前歌单的最后一曲的地方，调用该方法。对于投射源的多样性，通过分类的思想，分别进行歌单的更新操作。对于播放模式，只有顺序播放需要通过网络获取歌单，而其他三种模式，只需要更新播放位置即可，不需要更新歌单。

代码验证：在两处地方调用 updateCurrPlayInfo()方法进行最后曲目的检测，在方法内部将投诉源分为两类，分别进行歌单更新操作。具体代码如图 7-4、图 7-5、图 7-6 所示。

```
private void updateCurrPlayInfo() {
    if (mCurrControllerResource == Config.LOCAL || mCurrControllerResource == Config.PHONE_CONTROL) {
        if (mCurrentPosition == mCurrPlayInfo.getPlayList().getTrackList().size() - 1) {
            changeCurrPlayInfoForLocalOrPhone();
        }
    } else if (mCurrControllerResource == Config.ALARM_CLOCK) {
        if (mCurrentPosition == mAlarmClock.getPlayList().getTrackList().size() - 1) {
            changeCurrPlayInfo(mCurrControllerResource);
        }
    } else if (mCurrControllerResource == Config.DLNA) {
        if (mCurrentPosition == mCurrPlayInfo.getPlayList().getTrackList().size() - 1) {
            changeCurrPlayInfo(mCurrControllerResource);
        }
    }
}
```

图 7-4 updateCurrPlayInfo()方法

```
private void changeCurrPlayInfoForLocalOrPhone(){
    if (mPlayMode == Config.ORDER) {
        Log.i(TAG, "准备播放从网络获取的下一个歌单");
        if (mPlayInfoFromNet.getPlayList().getTrackList().size() != 0) {
            mCurrPlayInfo = mPlayInfoFromNet;
            mPreference.savePlayAlbumInfo(getApplicationContext(), mCurrPlayInfo);
            mCurrentPosition = mCurrPlayInfo.getTrackIndex() - 1;
            mLastControllerResource = mCurrControllerResource;
            getNextList(false);
        }
    }
}
```

图 7-5 changeCurrPlayInfoForLocalOrPhone()方法

```
private void changeCurrPlayInfo(int currResource) {
    PlayInfo playInfo = (PlayInfo) GsonUtils.fromJson(mPreference.loadPlayAlbumInfo
            (getApplicationContext()), PlayInfo.class);
    if (playInfo != null) {
        if (playInfo.getPlayList().getTrackList().size() != 0) {
            mCurrPlayInfo = playInfo;
            mPlayMode = mCurrPlayInfo.getPlayMode();
            mCurrentPosition = mCurrPlayInfo.getTrackIndex();
            if (mLastControllerResource == Config.LOCAL ||
                    mLastControllerResource == Config.PHONE_CONTROL) {
                mCurrControllerResource = mLastControllerResource;
            }
            mLastControllerResource = currResource;
            Log.i(TAG, "接下来播放歌曲是从内存中保存的：" + mCurrControllerResource);
        }
    }
}
```

图 7-6 changeCurrPlayInfo()方法

7.4 播放状态恢复

问题描述：闹钟响铃以及 DLNA 投射是被视为特殊投射的两种投射源，当这两种特殊投射源投射曲目时，如果当前曲目处于播放或者暂停状态，当投射曲目播放完后，音箱需要恢复之前的播放或者暂停状态，包括曲目播放位置以及 UI 显示。这个问题的难点在于暂停状态的恢复。

解决方案：当需要恢复暂停状态时，在 MusicCallBack 类的 onComplete()方法中设置一个 Flag，先发送播放命令，然后在 bufferComplete()方法中根据这个 flag 的值系统自动调用 seekTo()方法并且发送 pause 命令，这样就能保证在用户按下 Play 键时，音乐会继续播放。

代码验证：代码实现是在 onComplete()方法和 bufferComplete()方法中，此问题的具体实现细节如图 7-7、图 7-8 和图 7-9 所示。

```
@Override
public void onComplete() {
    PlayInfo playInfo = (PlayInfo) GsonUtils.fromJson(mPreference.loadPlayAlbumInfo(getApplicationContext()), PlayInfo.class);
    Log.i(TAG, "onComplete: 播放源" + mCurrControllerResource);
    if (mCurrControllerResource == Config.PHONE_CONTROL || mCurrControllerResource == Config.LOCAL
            || !playInfo.getStatus().equals(MusicService.State.PAUSED.toString())) {
        playPreviousOrNext(Config.NEXT);
    } else if (mCurrControllerResource == Config.ALARM_CLOCK) {
        updateCurrPlayInfo();
        if (mLastControllerResource == Config.ALARM_CLOCK) {
            mIController.dismissAlarmUI(mCurrControllerResource);
            recoveryPauseStatus(playInfo);
        } else {
            playPreviousOrNext(Config.NEXT);
        }
    } else if (mCurrControllerResource == Config.DLNA) {
        updateCurrPlayInfo();
        recoveryPauseStatus(playInfo);
    }
}
```

图 7-7 onComplete()方法

```
@Override
public void bufferComplete() {
    if (recovery == 1) {
        mMusicService.seekTo(mCurrPlayInfo.getElapsedTime() * 1000);
        recovery = 2;
    } else if (recovery == 2) {
        playOrPause(Config.PAUSE);
        recovery = 3;
    }
}
```

图 7-8 bufferComplete()方法

```
private void recoveryPauseStatus(PlayInfo playInfo) {
    mCurrPlayInfo = playInfo;
    mCurrentPosition = mCurrPlayInfo.getTrackIndex();
    recovery = 1;
    playOrPause(Config.PLAY);
    Log.i(TAG, "recoveryPauseStatus ElapsedTime: " + mCurrPlayInfo.getElapsedTime());
}
```

图 7-9 recoveryPauseStatus()方法

7.5 屏幕状态控制

问题描述：音箱屏幕有熄灭和点亮两种状态，当播放暂停时，10 秒音箱没有任何操作则进行熄灭状态；当屏幕熄灭的时候，播放状态改变或者 Dialog 显示都会使屏幕点亮，而如果仅仅是 Dialog 显示使得屏幕点亮，5 秒之后屏幕又将进入熄灭状态。改变播放状态的因素除了远程播放控制命令还有其他投射源投射歌曲以及音箱物理按键控制。

解决方案：经过分析，发现在处理远程命令控制的方法和按键事件加入屏幕状态改变操作即可，前提是将控制协议中 post 请求方式的命令统一通过自定义的 Handler 中的 handleMessage()方法来处理。

代码验证：在两个方法进行了屏幕状态控制，具体如图 7-10 和图 7-11 所示，这两个方法都是重写了 android 系统方法。

```
@Override
public void handleMessage(Message msg) {
    String jsonStr = getMsg(msg);
    if (action.equals(Config.SET_VOLUME) || action.equals(Config.RENAME)) {
        mIController.ChangeScreenStatus();
    } else {
        mIController.wakeUpScreen();
    }
    if (msg.what == Config.LOCAL) {
        switch (action) {
            case Config.PLAY:
```

图 7-10 handleMessage()方法

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    Log.i(TAG, "key " + keyCode);
    wakeUpScreen();
    if (mPlayState.equals(MusicService.State.PAUSED.toString())) {
        mHandler.removeCallbacks(screenRunnable);
        mHandler.postDelayed(screenRunnable, 50000);
    }
    switch (keyCode) {
        //音量键-
        case KeyEvent.KEYCODE_DPAD_DOWN:
            Log.i(TAG, "onKeyDown KEYCODE_DPAD_DOWN");
            keyUtil.dispatchKeyEvent(event);
            return true;
```

图 7-11 onKeydown()方法

7.6 定时休眠

问题描述：用户通过 LifeKit 设置定时休眠即设置定时暂停当前播放，并且音箱屏幕熄灭，音箱进入低耗状态。当用户设置定时休眠后，根据用户设置时间，从当前时间进行倒计时，当时间到达便触发休眠事件。使用什么方法来触发完成定时任务成为解决该问题的关键。

解决方案：利用 Android 提供的 AlarmManager 来完成定时任务，利用自定义广播实现触发通知。从用户设置完定时休眠那一刻开始，AlarmManager 的 void set(int type, long triggerAtTime, PendingIntent operation)方法设置定时服务 triggerAtTime，在。利用 AlarmManager 的特点，当时间到达 triggerAtTime 执行由 operation 参数的操作。

代码验证：自定义广播（作为 PlayMusicActivity 的内部类）以及 AlarmManager 如何设置定时服务的代码如图 7-12 和图 7-13 所示。

```
public class TimeReminderReceiver extends BroadcastReceiver {

    //一定要有空的构造函数
    public TimeReminderReceiver() {

    }

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "TimeReminderReceiver onReceive");
        if (mPlayState.equals(MusicService.State.STARTED.toString())) {
            Log.i(TAG, "TimeReminderReceiver 定时休眠");
            mManagerService.handleSendMsg(Config.LOCAL, Config.PAUSE, null);
        }
        setReminder(false, 0);//取消当前alarm
    }
}
```

图 7-12 自定义广播

```
@Override
public void setReminder(boolean flag, int time) {
    Log.i(TAG, "setReminder 设置提醒 time:" + time);
    AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
    Intent intent = new Intent();
    intent.setAction(GlobalConstant.BROADCAST_TIME_REMINDER);
    PendingIntent pi = PendingIntent.getBroadcast(PlayMusicActivity.this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    if (flag && time != 0) {
        Date t = new Date();
        t.setTime(java.lang.System.currentTimeMillis() + time * 1000);
        long triggerAtTime = t.getTime();
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");
        Date curDate = new Date(triggerAtTime);//获取当前时间
        am.set(AlarmManager.RTC, triggerAtTime, pi);
    } else {
        am.cancel(pi);
    }
}
```

图 7-13 AlarmManager 定时任务设置

8 总结与展望

本项目来源工程实际项目，实习期间作者在整个团队和项目软件研发经理的悉心指导下，独立完成了多媒体控制系统这一子系统的开发工作。随着智能家居行业的发展，部门成员一致认为 Wi-Fi 音箱在未来十分有前景。通过对国内外 Wi-Fi 音箱的发展现状分析，部门成员确定了 Wi-Fi 音箱的存在意义与价值。

本文从需求分析，系统设计与实现，系统测试以及复杂业务逻辑分析对本子系统进行论述。需求分析这一章节内容采用用例图以及用例描述来论述，简洁明了地说明了系统的主要功能。接着系统设计与实现又分为概要设计和详细设计两大部分内容。概要设计主要从架构设计、模块分解以及接口设计三大部分进行论述；整个子系统采用的 MVC 框架，主要采用 Android 中 Activity、Service 和 BroadcastReceiver 这三大组件。详细设计主要是根据各个功能模块，设计出类图，并且根据不同模块的特点，采用不同的 UML 图进行论述，本文主要采用了活动图、序列图以及状态图。在概要设计以及详细设计的基础上，完成编码实现。本文第 6 章从系统测试以及运行效果进行论述，介绍了测试标准、最后运行结果等内容。本文第 7 章针对系统实现过程遇到的难点问题进行分析，从问题分析、解决方案进行阐述。通过系统测试，多媒体控制系统各个功能已经测试通过，符合项目要求。

本位对于音箱频道的设计还有不足，这也是后期优化的重点。当音箱拥有较多用户后，通过收集用户数据、分析数据并且建立模型，做出一套推荐系统，为推荐歌单这一频道推荐更符合用户习惯的歌曲。此外本文在设计部分还存在不足之处，例如全局变量使用太多、代码的复用性不够等等，在后续的系统版本升级中会进行相应的重构，使得系统更加完善、可拓展性更强。

参考文献

[1] 杨为民. 基于 ARM 和 Android 平台之无线音响体系设计与实现.华南理工大学.华南理工大学.2014.第 01 页

[2] 张浩, 赵千川. 蓝牙手机室内定位系统[J]. 计算机应用, 2011, 31(11): 3152-3156.

[3] Liu J, Yu J. Research on Development of Android Applications[C]//2011 Fourth International Conference on Intelligent Networks and Intelligent Systems. IEEE, 2011: 69-72.

[4] 李刚.疯狂 Android 讲义.北京.电子工业出版社.2011 年

[5] Enck W, Ongtang M, McDaniel P. Understanding android security[J]. IEEE security & privacy, 2009 (1): 第 50-57 页.

[6] Kasai H. Implementation of distributed mobile storage and location-based message sharing among smart phones[J]. ACM SIGMOBILE Mobile Computing and Communications Review, 2013, 16(4): 第 20-21 页.

[7] 黄蓉. Android 消息处理机制研究[J]. 黑龙江科技信息, 2012 (33): 87-87.第 01-02 页

[8] 陈伟. Android Handler 消息传递机制在人机界面软件设计中的应用[J]. 科技信息, 2014 (13): 91-91.

[9] 彭涛, 孙连英. 回调机制及其在 Android 应用开发中的应用[J]. 北京联合大学学报: 自然科学版, 2013, 27(2): 第 68-72 页.

[10] 董晓刚. android 中基于回调机制的事件处理[J]. 中国电子商务, 2013 (13): 88-88.

[11] 郭永鑫,乐嘉锦,夏小玲等.一种基于 DLNA 协议的 Android 设备多媒体共享方案[J].计算机与现代化,2013,(11):124-127,132.DOI:10.3969/j.issn.1006-2475.2013.11.030.

[12] 魏博.软件非功能性需求建模与设计决策评估方法研究[D].中国科学院大学,2013.

[13] 祝洪娇.基于 Android 平台的位置服务系统的设计与实现[D].北京交通大学,2012.

[14] Sokolova K, Lemercier M, Garcia L. Android passive MVC: a novel architecture model for the android application development[C]//Proceedings of the fifth international conference on pervasive patterns and applications (PATTERNS'13). IARIA. 2013: 7-12.

[15] Li Y, Cui D. Improvement and application of MVC design patterns [J][J]. Computer Engineering, 2005, 9: 35.

[16] 薛晋.基于 Android 的 MSN V7.0 手机客户端的设计与实现[D].北京交通大学,2012.

[17] 韦慕华.Android 平台优化和防护工具的设计与实现[D].北京交通大学,2015.

[18] 邵维忠.统一建模语言 UML 述评.计算机研究与发展,1999,36(4);385-394

[19] Rumbaugh J, Jacobson I, Booch G. Unified Modeling Language Reference Manual, The[M]. Pearson Higher Education, 2004.

[20] Papajorgji P J, Pardalos P M. Software engineering techniques applied to agricultural systems: an object-oriented and UML approach[M]. Springer, 2014.

[21] Kumar B, Singh K. Testing UML Designs Using Class, Sequence and Activity Diagrams[J].

International Journal for Innovative Research in Science and Technology, 2015, 2(3): 71-81.

致　　谢

随着实习的结束以及毕业设计的完成，四年的大学时光也将画上圆满句点。在此，我要感谢陪伴我走过这个充实旅途的老师们。

首先，我要特别感谢我的指导老师邢薇薇老师。在整个毕业设计的过程中，从选题到内容取舍再到论文修改，邢老师都给予了宝贵的建议和细心的指导。在与老师讨论毕业设计的过程中，邢老师认真负责的教学态度、严谨的科研态度以及丰富的教学经验让作者受益匪浅。

同时，我也需要借此机会感谢在实习过程中给予作者帮助的企业导师，感谢他对我专业知识的耐心指导以及工作能力的肯定。正是有了他的教导，我才能成功完成多媒体控制系统的开发工作。除了对于该项目的指导，针对我的个人特点，他还从职业规划以及行业发展等方面给我提出了建设性的建议。

最后，感谢学院的精心栽培，感谢学院老师和同学们的帮助和陪伴。在这美好的大学 4 年时光里，有大家的陪伴是我的幸运，感谢大家。最后，祝愿大家身体好，心情好！

附　　录

附录 A 外文翻译原文

Understanding Android security

The next generation of open operating systems won't be on desktops or mainframes but on the small mobile devices we carry every day. The openness of these new environments will lead to new applications and markets and will enable greater integration with existing online services.However, as the importance of the data and services our cell phones support increases, so too do the opportunities for vulnerability. It's essential that this next generation of platforms provides a comprehensive and usable security infrastructure.

Developed by the Open Handset Alliance (visibly led by Google),Android is a widely anticipated open source operating system for mobile devices that provides a base operating system, an application middleware layer, a Java software development kit (SDK), and a collection of system applications. Although the Android SDK has been available since late 2007, the first publicly available Android-ready"G1" phone debuted in late October 2008. Since then, Android's growth has been phenomenal:T-Mobile's G1 manufacturer HTC estimates shipment volumes of more than 1 million phones by the end of 2008, and industry insiders expect public adoption to increase steeply in 2009. Many other cell phone providers have either promised or plan to support it in the near future.

A large community of developers has organized around Android, and many new products and applications are now available for it. One of Android's chief selling points is that it lets developers seamlessly extend online services to phones. The most visible example of this feature is, unsurprisingly, the tight integration of Google's Gmail, Calendar, and Contacts Web applications with system utilities. Android users simply supply a username and password, and their phones automatically synchronize with Google services. Other vendors are rapidly adapting their existing instant messaging, social networks, and gaming services to Android, and many enterprises are looking for ways to integrate their own internal operations (such as inventory management, purchasing, receiving, and so forth) into it as well.

Traditional desktop and server operating systems have struggled to securely integrate such personal and business applications and services on a single platform. Although doing so on a mobile platform such as Android remains nontrivial, many researchers hope it provides a clean slate devoid of the complications that legacy software can cause. Android doesn't

officially support applications developed for other platforms: applications execute on top of a Java middleware layer running on an embedded Linux kernel, so developers wishing to port their application to Android must use its custom user interface environment. Additionally, Android restricts application interaction to its special APIs by running each application as its own user identity. Although this controlled interaction has several beneficial security features, our experiences developing Android applications have revealed that designing secure applications isn't always straightforward. Android uses a simple permission label assignment model to restrict access to resources and other applications, but for reasons of necessity and convenience, its designers have added several potentially confusing refinements as the system has evolved.

This article attempts to unmask the complexity of Android security and note some possible development pitfalls that occur when defining an application's security. We conclude by attempting to draw some lessons and identify opportunities for future enhancements that should aid in clarity and correctness.

Android Applications

The Android application framework forces a structure on developers. It doesn't have a main() function or single entry point for execution—instead, developers must design applications in terms of components.

Example Application

We developed a pair of applications to help describe how Android applications operate. Interested readers can download the source code from our Web site (http://siis.cse.psu.edu/android_sec_tutorial.html).

Let's consider a location-sensitive social networking application for mobile phones in which users can discover their friends' locations. We split the functionality into two applications: one for tracking friends and one for viewing them. As Figure 1 shows, the FriendTracker application consists of components specific to tracking friend locations (for example, via a Web service), storing geographic coordinates, and sharing those coordinates with other applications. The user then uses the Friend-Viewer application to retrieve the stored geographic coordinates and view friends on a map.

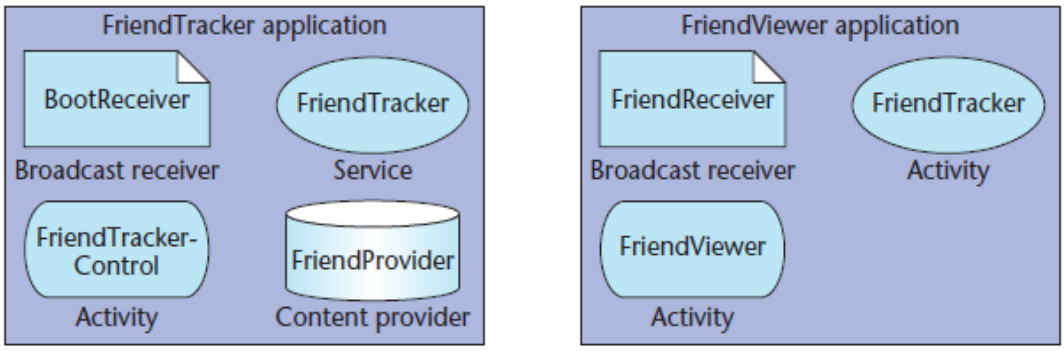

Figure 1. Example Android application. The FriendTracker and FriendViewer applications consist of multiple components of different types, each of which provides a different set of functionalities. Activities provide a user interface, services execute background processing, content providers are data storage facilities, and broadcast receivers act as mailboxes for messages from other applications.

Both applications contain multiple components for performing their respective tasks; the components themselves are classified by their *component types*. An Android developer chooses from predefined component types depending on the component's purpose (such as interfacing with a user or storing data).

Component Types

Android defines four component types:

- *Activity* components define an application's user interface. Typically, an application developer defines one activity per "screen." Activities start each other, possibly passing and returning values. Only one activity on the system has keyboard and processing focus at a time; all others are suspended.
- *Service* components perform background processing. When an activity needs to perform some operation that must continue after the user interface disappears (such as download a file or play music), it commonly starts a service specifically designed for that action. The developer can also use services as application-specific daemons, possibly starting on boot. Services often define an interface for Remote Procedure Call (RPC) that other system components can use to send commands and retrieve data, as well as register callbacks.
- *Content provider* components store and share data using a relational database interface. Each content provider has an associated "authority" describing the content it contains. Other components use the authority name as a handle to perform SQL queries (such as SELECT, INSERT, or DELETE) to read and write content. Although content providers typically store values in database records, data retrieval is implementation-specific—for example, files are also shared through content provider interfaces.

- *Broadcast receiver* components act as mailboxes for messages from other applications. Commonly, application code broadcasts messages to an implicit destination. Broadcast receivers thus subscribe to such destinations to receive the messages sent to it. Application code can also address a broadcast receiver explicitly by including the namespace assigned to its containing application.

Figure 1 shows the FriendTracker and FriendViewer applications containing the different component types. The developer specifies components using a manifest file (also used to define policy as described later). There are no restrictions on the number of components an application defines for each type, but as a convention, one component has the same name as the application. Frequently, this is an activity, as in the FriendViewer application. This activity usually indicates the primary activity that the system application launcher uses to start the user interface; however, the specific activity chosen on launch is marked by meta information in the manifest. In the FriendTracker application, for example, the FriendTracker-Control activity is marked as the main user interface entry point. In this case, we reserved the name “FriendTracker” for the service component performing the core application logic.

The FriendTracker application contains each of the four component types. The FriendTracker service polls an external service to discover friends’ locations. In our example code, we generate locations randomly, but extending the component to interface with a Web service is straightforward. The FriendProvider content provider maintains the most recent geographic coordinates for friends, the FriendTrackerControl activity defines a user interface for starting and stopping the
tracking functionality, and the BootReceiver broadcast receiver obtains a notification from the system once it boots (the application uses this to automatically start the FriendTracker service).

The FriendViewer application is primarily concerned with showing information about friends’ locations. The FriendViewer activity lists all friends and their geographic coordinates, and the FriendMap activity displays them on a map. The FriendReceiver broadcast receiver waits for messages that indicate the physical phone is near a particular friend and displays a message to the user upon such an event. Although we could have placed these components within the FriendTracker application, we created a separate application to demonstrate cross-application communication. Additionally, by separating the tracking and user interface logic, we can create alternative user interfaces with different displays and features—that is, many applications can reuse the logic performed in FriendTracker.

Component Interaction

The primary mechanism for component interaction is an *intent*, which is simply a message object containing a destination component address and data. The Android API defines methods that accept intents and uses that information to start activities (startActivity(Intent)), start services (startService(Intent)), and broadcast messages (sendBroadcast(Intent)). The invocation of these methods tells the Android framework to begin executing code in the target application. This process of intercomponent communication is known as an *action*. Simply put, an intent object defines the "intent" to perform an "action."

One of Android's most powerful features is the flexibility allowed by its intent-addressing mechanism. Although developers can uniquely address a target component using its application's namespace, they can also specify an implicit name. In the latter case, the system determines the best component for an action by considering the set of installed applications and user choices. The implicit name is called an *action string* because it specifies the type of requested action—for example, if the "VIEW" action string is specified in an intent with data fields pointing to an image file, the system will direct the intent to the preferred image viewer. Developers also use action strings to broadcast a message to a group of broadcast receivers. On the receiving end, developers use an *intent-filter* to subscribe to specific action strings. Android includes additional destination resolution rules, but action strings with optional data types are the most common.

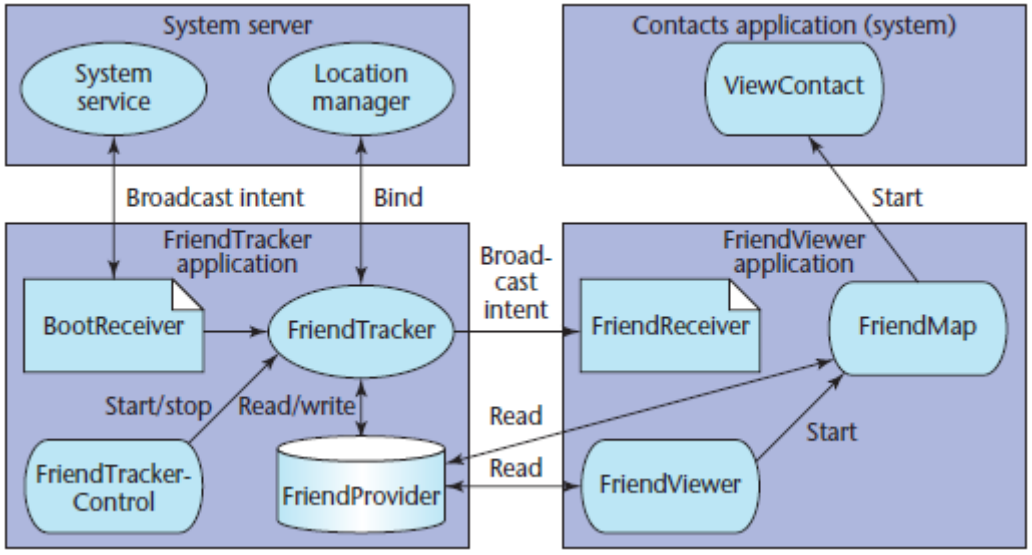

Figure 2. Component interaction. Android's application-level interactions let the FriendTracker and FriendViewer applications communicate with each other and system-provided applications. Interactions occur primarily at the component level.

Figure 2 shows the interaction between components in the FriendTracker and FriendViewer applications and with components in applications defined as part of the base Android distribution. In each case, one component initiates communication with another. For simplicity, we call this intercomponent communication (ICC). In many ways, ICC is analogous to interprocess communication(IPC) in Unix-based systems. To the developer, ICC functions identically regardless of whether the target is in the same or a different application, with the exception of the security rules defined later in this article.

The available ICC actions depend on the target component. Each component type supports interaction specific to its type——for example, when FriendViewer starts FriendMap, the FriendMap activity appears on the screen. Service components support start, stop, and bind actions, so the FriendTrackerControl activity, for instance, can start and stop the FriendTracker service that runs in the background. The bind action establishes a connection between components, allowing the initiator to execute RPCs defined by the service. In our example, FriendTracker binds to the location manager in the system server. Once bound, FriendTracker invokes methods to register a callback that provides updates on the phone’s location. Note that if a service is currently bound, an explicit “stop” action won’t terminate the service until all bound connections are released.

Broadcast receiver and content provider components have unique forms of interaction. ICC targeted at a broadcast receiver occurs as an intent sent (broadcast) either explicitly to the component or, more commonly, to an action string the component subscribes to. For example, FriendReceiver subscribes to the developer-defined “FRIEND_NEAR” action string.FriendTracker broadcasts an intent to this action string when it determines that the phone is near a friend; the system then starts FriendReceiver and displays a message to the user.

Content providers don’t use intents—rather, they’re addressed via an authority string embedded in a special content URI of the form content://<authority>/ <table>/[<id>]. Here, <table> indicates a table in the content provider, and <id> optionally specifies a record in that table. Components use this URI to perform a SQL query on a content provider, optionally including WHERE conditions via the query API.

附录 B 外文翻译译文

理解 Android 的安全性

新一代开放式的操作系统将不会是在台式机或大型机而是我们每天都随身携带小型移动设备。这些新环境的开放性将导致新的应用和市场，将使现有的在线服务更大程度的整合。然而，由于我们的手机支持的数据和服务重要性增加，也是如此做的对于漏洞的机会。它的重要的是这种新一代的平台提供了一个综合和可用的安全基础设施。

由开放手机联盟（由谷歌明显的领导）开发的， Android是一个普遍预期的开源操作系统为移动设备提供了一个基本的操作系统，应用中间件层，一个Java软件开发工具包（SDK），以及集合系统的应用程序。虽然自2007年年底，第一个公开发布的Android就绪“G1”手机在Android SDK已经提供 2008年10月下旬推出自此， Android的增长已经非常显着： T-Mobile的G1厂商宏达电在2008年年底估计超过100万部手机出货量，而业内人士预计市民领养 在2009年许多其它手机供应商陡峭增加要么承诺或计划，以支持它在不久的将来。

一个围绕Android大型开发人员社区已经建立，许多新的产品和应用现在都是可用的。Android的首要卖点是，它让开发者无缝扩展手机在线服务。此功能的最明显的例子就是谷歌的紧密集成 Gmail，日历和联系人Web应用程序的系统工具。 Android用户只需提供用户名和密码，自己的手机自动同步与谷歌的服务。其他厂商正在迅速改变现有的即时通讯，社交网络，以及Android的游戏服务，许多企业也都在寻找方法来整合自己内部的操作（如库存管理，采购，接收，等等）。

传统的桌面和服务器操作系统一直在努力在单一平台上整合安全等个人和企业应用程序和服务。虽然在移动平台如Android这样做仍然是平凡的，很多研究人员希望它提供一个美好的希望，没有传统软件可能导致的并发症。Android不正式支持其他平台开发的应用程序：应用程序执行上依赖嵌入式Linux内核的顶层Java中间件层，所以部署他们的应用程序到Android，开发者必须使用其自定义的用户界面环境。此外，Android的运行每个应用程序作为自己的用户身份限制了应用程序的交互其特殊的API。尽管这种控制交互有几个有利的安全功能，我们的经验开发Android应用程序的人士透露，设计安全应用程序并不简单。 Android使用一个简单的许可标签分配模式限制访问资源和其他应用程序，但由于其的必要性和方便性的原因，它的设计者增加了一些潜在的混乱改进作为系统的演变。

本文试图揭露了Android的安全的复杂性，并注意定义应用程序的安全性时出现的一些可能的发展陷阱。最后，我们试图得出一些经验教训，确定今后的增强清晰度和正确性帮助的机会。

Android应用程序

Android应用程序框对于开发者是一个强制架构。它不具有一个主方法或单一入口点执行；相反，开发商必须设计条款中的应用的组分。

示例应用程序

我们开发了一个应用程序帮助描述如何创建Android应用 。感兴趣的读者可以从我们的网站（ http://siis.cse.psu.edu/android_sec_tutorial.html ）下载源代码。

让我们考虑一个位置敏感的社交网络应用，该应用让移动电话用户可以发现他们朋友的位置。我们将功能分为两个应用程序：一个是跟踪朋友，一个用于观看朋友位置。如图1所示， FriendTracker应用程序由特定于跟踪朋友位置组件（例如，通过Web服务），存储地理坐标，并与其它应用程序共享那些位置坐标。然后，用户使用FriendViewer应用程序检索存储的地理坐标和在地图上查看好友。

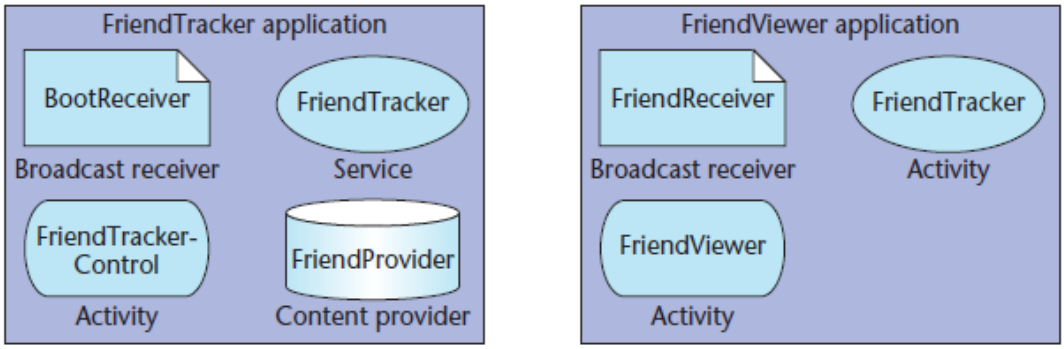

图1.示例Android应用程序。该FriendTracker和FriendViewer应用由不同类型的多个组件组成，每一个组件都提供一组不同的功能。Activity活动提供了一个用户界面，Service服务执行后台处理，ContentProvider内容提供商的数据存储设施，以及BroadcastReceivers广播接收器充当其他应用程序的消息的邮箱。

这两个应用程序包含执行各自的任务的多个组件；组件本身由它们的组件类型进行分类。一个Android开发者根据组件的目的从预定的组件类型（如与用户接口或存储数据）选择。

组件类型

Android的定义了四种成分类型：

- 活动组件定义应用程序的用户界面。通常，应用程序开发者定义一个界面一个活动。Activity之间相互启动，也可能传递值和返回值。在一时间只有一个系统活动可以进行处理，这个时候所有其他activity被暂停。
- 服务组件进行后台处理。当一个活动需要执行某些只有用户界面消失之后才能继续的操作（如下载文件或播放音乐），它通常为这个活动专门启动一个服务。开发人员还可以使用服务作为应用程序特定的守护进程，可能在开机启动的时候。服务通常定义为远程过程调用（RPC ），其他系统组件可以使用它发送命

令和检索数据，以及注册回调接口。

- 内容提供组件用来存储和使用关系数据库接口共享数据。每个内容提供商都有一个相关的“权威”描述它包含的内容。其他组件使用的机构名称作为手柄来执行SQL查询（如SELECT ，INSERT或DELETE ）来读取和写入内容。虽然内容提供商通常存储数据在数据库中的记录，数据检索是一个特定实现，例如，文件也可以通过内容提供商接口共享。
- 广播接收器组件作为其他应用程序消息的邮箱。通常，应用程序代码广播消息到一个个隐含的目的地。因此，广播接收器订阅这样的目的地来接收发送给它的消息。应用程序代码也可以明确发布一个广播接收器包括分配名称空间寻址给它包含应用程序的。

图1显示了包含不同的组件类型的FriendTracker和FriendViewer应用程序。开发者使用一个配置文件细化组件（也用来定义政策如后述）。一个应用程序对于使用每种类型的组件的数量没有限制，但作为一个惯例，一个组件具有相同的名称作为应用程序。通常，它是一个activity，如在FriendViewer应用。此活动通常表明，该系统应用程序启动器用来启动用户界面的主要活动；然而，所选择的发射的特定activity在配置文件中的元信息中被标识。在FriendTracker应用，例如，FriendTrackerControl活动被标记为主用户界面入口点。在这种情况下，我们保留用于执行核心应用逻辑的服务组件的名称为“FriendTracker”。

该FriendTracker应用程序包含四个组件类型。该FriendTracker服务轮询外部服务发现朋友的位置。在我们的例子代码中，我们生成随机的位置，但在扩展组件与Web服务接口非常简单。所述FriendProvider内容提供商可维护好友最近的地理坐标，FriendTrackerControl活动定义了用户界面，用于启动和停止跟踪功能，一旦它启动了BootReceiver广播接收器获得来自系统的通知（应用程序使用它来自动启动FriendTracker服务）。

该FriendViewer应用主要关心的是显示关于朋友的位置的信息。该FriendViewer活动列出了所有的朋友和他们的地理坐标，以及FriendMap活动显示他们在地图上的位置。该FriendReceiver广播接收器等待指示附近的一个特别的朋友的物理电话并通过该事件显示一条消息给用户。虽然我们可以放在FriendTracker应用程序中的这些成分，我们创建了一个单独的应用程序来演示跨应用程序通信。此外，通过分离跟踪和用户界面逻辑，我们可以创建不同的显示和功能，也就是替代用户界面，许多应用程序可以重复使用FriendTracker执行的逻辑。

组件交互

对于组件交互的主要机制是一个意图，这简直是包含目标组件地址和数据的消息对象。Android的API定义接受意图的方法和使用该信息来启动活动（startActivity（Intent

intent）方法），启动服务（startService（Intent intent）方法），和广播消息（sendBroadcast（Intent intent））。这些方法的调用告诉Android框架开始在目标应用中执行代码。互相通信的这个过程被称为一个动作。简单地说，一个意图对象定义的“意图”来执行“行动”。

一个Android的最强大的特点是它的意图寻址机制的灵活性。尽管开发者可以使用其应用程序的命名空间解决一个目标组件，他们还可以指定一个隐含的名称。在后一种情况下，系统通过考虑安装的应用程序和用户的选择来决定action的最佳组件。这个隐式名称称为操作串因为它特殊类型的请求的动作，例如如果指定的“VIEW”动作字符串在一个intent中和数据域指向一个图像文件，系统会直接将意图指向优选图像浏览器。开发人员还用行动字符串来广播消息给一组广播接收器。在接收端，开发者使用意图过滤器订阅特定动作字符串。Android包括其附加目标的解析规则，但是可选数据类型的操作字符串是最常见的。

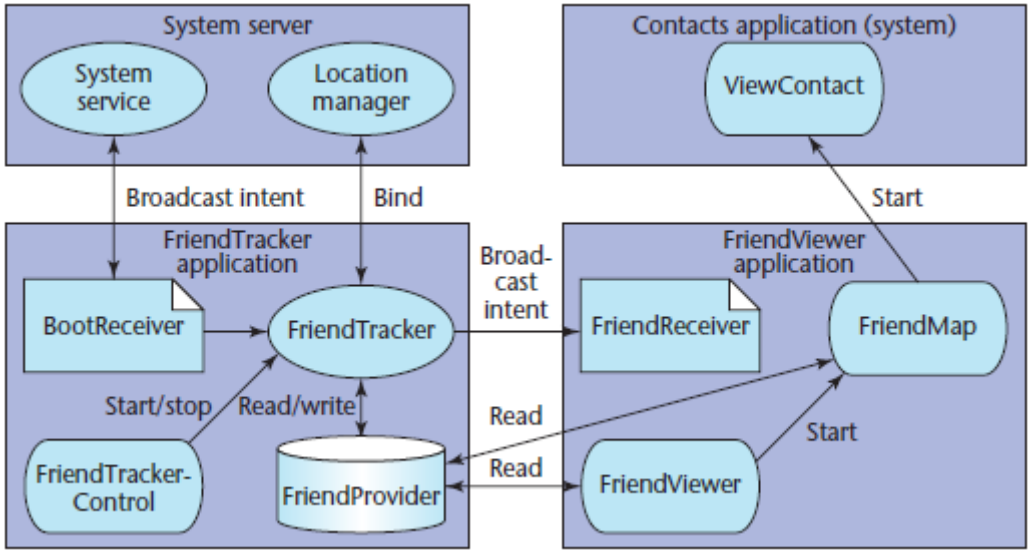

图2 组件交互。Android应用层交互使得FriendTracker和FriendViewer应用可以相互并且与其系统提供的应用进行通信。交互通常发生在组件层。

图2展示了FriendTracker和FriendViewer应用程序之间以及与被定义为Android分发的应用程序的组件交互。每种情况下，一个组件启动与另一组件通信。简单得说，我们称这种情况为组件间通信（简称为ICC）。很多时候，组件间通信类似基于Unix的系统进程间通信（IPC）。

可用的组件间通信取决于目标组件。每个组件支持其特定类型的交互——例如，当FriendViewer启动FriendMap时，FriendMap这一活动显示在屏幕上。服务组件支持启动，停止以及绑定行为，因此FriendTrackerControl活动可以启动和暂停在后台运行的FriendTracker服务。服务绑定建立组件间的连接，允许执行服务定义的RPC。在我们的例子中，在系统服务器FriendTracker绑定定位管理。一旦绑定后，FriendTracker调用方

法来注册一个提供更新手机位置的回调。需要注意的是，如果一个服务已经绑定，一个“停止”操作不会终止该服务直到所有绑定的连接都已经解绑。

广播接收器和内容提供者组件有不一样的交互形式。ICC定义广播接收器的发生是由于显式意图（广播）发送或者更常见的是以一个动作串组件预订。例如，FriendReceiver接受开发者定义的“FRIEND_NEAR”行为字符串。当它确定该手机用户是附近的一个朋友，FriendTracker广播一个意图给这个行为字符串；然后系统启动FriendReceiver并且发送一条信息给用户。

内容提供商不使用意图，相反，他们通过嵌入在形式内容的特殊内容URI权威字符串处理：// <授权>/<TABLE>/[<编号>]。在这里，<TABLE>表示在内容提供商的表，和<编号>可选指定在表中的记录。组件使用此URI对内容提供商执行一个SQL查询，任选包括WHERE通过查询API条件。