华南理工大学

South China University of Technology

工程硕士学位论文

基于 Apache Cordova 的跨平台智能家居终端应用研发

作　者　姓　名	肖　敏
工　程　领　域	计算机技术
校内指导教师	刘发贵　教授
校外指导老师	戴晨昱　高级工程师
所　在　学　院	计算机科学与工程学院
论文提交日期	2015 年　5 月

Development of Apache Cordova based Cross-platform Smart Home Terminal Application

A Dissertation Submitted for the Degree of Master

Candidate：Xiao Min

Supervisor：Prof. Liu Fagui

South China University of Technology

Guangzhou, China

分类号:TP3 **学校代号:10561**

学　号：201221012968

华南理工大学硕士学位论文

基于 Apache Cordova 的跨平台智能家居终端应用研发

作者姓名：肖敏　　申请学位级别：工程硕士

工程领域名称：计算机技术

校内指导教师姓名、职称：刘发贵 教授　　校外指导教师姓名、职称：戴晨昱 高级工程师

论文形式：☐ 产品研发　☐ 工程设计　☑ 应用研究　☐ 工程/项目管理　☐ 调研报告

研究方向：嵌入式技术

论文提交日期： 2015 年 05 月 31 日　　论文答辩日期： 2015 年 04 月 25 日

学位授予单位：华南理工大学　　学位授予日期：　　年　月　日

答辩委员会成员：

主席：张星明

委员：刘发贵、高英、董守斌、陈仲驹

华南理工大学
学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：肖敏　　日期：2015年04月23日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属华南理工大学。学校有权保存并向国家有关部门或机构送交论文的复印件和电子版，允许学位论文被查阅（除在保密期内的保密论文外）；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。本人电子文档的内容和纸质论文的内容相一致。

本学位论文属于：

□保密，在_____年解密后适用本授权书。

☑不保密，同意在校园网上发布，供校内师生和与学校有共享协议的单位浏览；同意将本人学位论文提交中国学术期刊(光盘版)电子杂志社全文出版和编入CNKI《中国知识资源总库》，传播学位论文的全部或部分内容。

(请在以上相应方框内打“√”)

作者签名：肖敏　　日期：2015.04.23

指导教师签名：　　日期 2015.4.23

作者联系电话：　　电子邮箱：

联系地址(含邮编)：

摘　要

随着物联网的兴起，越来越多的家电厂商、互联网公司加入到智能家居解决方案的研究当中。目前主流的智能家居解决方案中，少不了对多移动平台的支持，这也让更多的用户能够通过移动设备进行家电控制等便利操作。但是在各平台的智能家居移动终端上都进行原生应用程序的开发，会耗费开发团队较大的人力物力，同时产生后期各平台维护的较高成本。

针对以上提及的问题，本文调研分析了目前多平台移动下的跨平台开发技术的发展现状，以及跨移动平台开发框架 Apache Cordova 在实现智能家居解决方案中的可行性，选择该框架并针对智能家居移动终端应用进行了功能扩展，结合 HTML、CSS 和 Javascript 三大 Web 开发语言，设计并实现了一套可扩展的跨平台智能家居移动终端应用。本文所做工作包括：

1. 研究目前跨移动平台的应用开发的国内外现状，分析现今主流的开发方法及工具，并比较其技术优、缺点，针对智能家居应用的特性选择了 Apache Cordova 框架。

2. 进行相关技术分析。分析 Apache Cordova 的主体框架、开发模式以及运行原理，结合智能家居应用所需的视频监控、安防消息推送等功能，研究了该框架的模块扩展技术；同时分析移动消息推送的相关技术。

3. 设计各功能模块。根据智能家居移动终端应用的功能需求，设计 Android 及 iOS 平台的 Apache Cordova 框架扩展模块，包括本地 Sqlite 数据缓存模块、家电控制模块、视频监控扩展模块以及消息推送模块。

4. 实现各功能模块。结合移动平台的本地 API，实现上述针对智能家居应用的 Apache Cordova 跨平台开发框架的扩展，通过 Javascript 接口向上层提供扩展后的跨平台智能家居应用开发接口，以及第三方视频监控 SDK 适配接口，并通过此构建应用。

5. 测试与分析工作。将本文的基于 Apache Cordova 扩展模块，可支持家电控制、视频监控以及安防警报的智能家居移动应用，并进行测试、分析总结。

本文针对智能家居移动终端应用的功能需求，提出了在 Apache Cordova 移动开发框架上的功能扩展方案，通过该开源框架的跨平台优势，来节省智能家居领域的移动终端应用开发成本，同时保障智能家居应用的功能性、可扩展性，可以满足目前针对智能家居的移动终端应用的快速、低成本开发要求。

关键词：智能家居；移动平台；Apache Cordova；跨平台

Abstract

With the rise of the Internet of Things, a growing number of home appliance manufacturers and Internet companies have joined to study among smart home solutions.In the current mainstream solutions of smart home, it's necessary to support multiple mobile platforms, which allow more mobile phone users to control home appliances through mobile devices more conveniently. However, developing native applications of smart home for each mobile each platform, will take the development team a greater manpower and resources. In addition, it will generate higher maintenance costs for each platform.

For the above-mentioned problems, the paper analyzes the current research development of cross-platform development technology in multi-mobile-platform and the feasibility of achieving smart home solutions with cross-platform mobile application development framework, Apache Cordova. The paper selects this framework and doing the functional expansion work for smart home mobile terminal applications. Using HTML, CSS and Javascript Web development language, this paper designs and implements a scalable cross-platform smart home application. Works done in this article include:

1. Researching the current domestic and foreign situation of cross-mobile-platform application development, analyzing the current mainstream development methods and tools, comparing their technical advantages and disadvantages and choosing Apache Cordova framework for the characteristics of smart home applications;

2. Related Technical Analysis. By analyzing the main framework, development model and operating principles of Apache Cordova, combining with smart home applications requirement such as video surveillance, security features message pushing and researching the extension work of module framework; simultaneously analyzing the mobile message pushing technologies;

3. Design of the functional modules. According to the functional requirements of smart home mobile terminal application, designing the extension modules of Apache Cordova framework on Android and iOS platforms, including local Sqlite data cache modules, appliance control module, video surveillance and message push expansion module;

4. Implement of the functional modules. Combined with the mobile platform native API, achieving the above extension of Apache Cordova for smart home applications cross-platform

development, through Javascript interface providing expanded cross-platform application development interface for smart home, as well as third-party video surveillance SDK adapter interface to the upper layer and building the applications;

5. The testing work, analyzing and summary of the samrt home mobile application based on the expansion modules of Apache Cordova in this paper, which supports appliance control, video surveillance and security warning.

According the functional requirements of the smart home applications, this paper proposed the extensions on Apache Cordova mobile development framework and built the application.Through the advantages of the open-source cross-platform development framework, it will save the cost of mobile terminal applications development in the field of smart home, additionally safeguard the functionality and scalability of the smart home application, meeting the rapid, low-cost development requirements of the current smart home mobile terminals applications.

Key words: smart home; mobile platforms; Apache Cordova; cross-platform;

目录

第一章 绪论

1.1 研究背景

1.1.1 课题研究背景

随着移动互联网的逐渐普及，面向移动互联网的应用、服务快速发展，各大移动平台如 Google 的 Android[1]、苹果的 iOS[2]等不断地拓展、丰富移动平台的功能。移动设备终端已逐渐成为日常生活中不可或缺的产品，而与此同时，移动互联网应用也正在向生活化、人性化的趋势发展，逐渐渗透人们的生活、社交当中，智能家居也开始走进一些家庭中。

智能家居系统[3,4]是一个集硬件、软件、通信多方面技术的应用系统，其中移动终端作为用户操作行为的接口，拥有远程控制、监控等丰富功能[5]。当前移动终端设备有各种品牌，拥有不同的系统平台，目前大部分厂商需要为智能家居移动终端开发适合多种平台的软件[6]，需要根据不同平台的 API 进行原生应用的开发[7]，而现今流行的主流移动平台操作系统有 Android、iOS、Windows Phone[8]等，分别在各自的移动操作系统上进行原生应用的开发，会带来昂贵的开发费用、耗费大量人力、增加修改维护难度，若仅仅对部分功能进行细微修改，也不能避免在各个操作系统上单独进行版本改动。

对于上述的问题，首先考虑到的解决方案，就是选择一个跨移动平台的应用开发解决方案，当前国内外主流的跨移动平台应用开发解决方案，主要有两种：1. 利用 B/S 架构开发基于 Web 的移动应用[9]，并针对移动设备进行浏览优化[10]，但这类应用能够实现的功能较少，缺乏本地支持，如设备传感器数据等；2. 利用跨移动平台开发框架[11,12]，如 Apache Cordova[13]、Titanium[14]、Corona[15]等，这类应用能够抽取出大部分公共模块，并结合对本地操作系统 API 的支持，应用的功能较丰富，又比较节省开发成本。在物联网的流行趋势下[16,17]，对于智能家居移动终端应用来说，大部分厂商都把远程控制家电、视频监控、安防警报等作为研发的重点功能[18,19]，而这些功能的实现，需要涉及家电设备信息数据处理、不同视频监控设备厂商的 SDK 整合以及移动平台上的消息推送技术，单纯的 B/S 架构的移动应用[20]，在实现功能上显得力不从心，即使使用基于当前较先进的 HTML5 技术进行开发[21,22,23]，也不能很好地满足智能家居移动终端应用程序的功能实现。

本文将针对智能家居移动设备应用的功能需求及特性，选择上述的第二种方案，比较分析并选择一个适合的跨移动平台应用开发框架，进行功能模块的扩展，更好地支持

要有一个提供本地数据缓存、更新的模块，而 Apache Cordova 只是简单地提供了本地文件读写的支持接口；

2. 如今大部分智能家居移动应用需要视频监控功能，而市场上的大部分视频监控设备，都通过提供移动平台的第三方 SDK 来供开发者使用，而未能提供标准的 HTML5 的视频流解决方案，另外 HTML5 标准也仍在改进，同时也面临浏览器内核的支持问题，故在接入第三方 SDK 方面，Apache Cordova 框架仍需要进行扩展；

3. 在移动互联网技术日益先进的情况下，智能家居移动应用自然少不了安防警报消息的推送功能，而 Apache Cordova 框架未提供有效的消息推送渠道，如果要满足消息推送的需求，则需要进行模块的扩展。

综上所述，要通过 Apache Cordova 来开发智能家居移动终端应用，仍需要对该框架做一些功能模块上的扩展，而 Apache Cordova 自身也具备良好的扩展性，故本文将在这方面做进一步的研究。

1.3 主要研究内容和研究意义

经过如上所述的针对智能家居移动应用的跨平台开发方案的调研，以及对较符合该领域开发需求的 Apache Cordova 开发框架的背景研究，如何针对智能家居移动应用所需的特定功能对 Apache Cordova 进行功能模块的扩展，将成为本论文的主要研究目标。本文所做的主要研究内容主要分为：

1. 对 Apache Cordova 进行深入分析。由于 Apache Cordova 属于在 Apache License 下的开源项目，我们可以很方便地从官方得到源代码，并可以从总体架构、开发模式以及运行原理等方面进行深入的分析，并研究 Apache Cordova 的模块扩展技术，能够更好地对其进行功能扩展，以满足智能家居移动应用的开发需要；

2. 研究分析当前的移动平台消息推送技术。由于需要把安防警报功能所需的消息推送模块，作为 Apache Cordova 的功能扩展模块集成进来，应该选择一种较适合智能家居移动应用的消息推送技术。当前的面向移动互联网的消息推送技术，已有比较成熟的解决方案，但都因平台而异。如果需要与 Apache Cordova 结合并作出多个平台的适配，还是一个比较棘手的问题。

3. 针对智能家居移动应用，对 Apache Cordova 加入相关功能扩展。如前面所述，Apache Cordova 官方封装的本地设备 API 支持功能有限，如果需要实现智能家居移动应用中的一些主流功能如家电控制、安防警报、视频监控等，本文将重点加入这些功能的

拓展模块，并在两大主流平台上做本地适配，同时预留其余平台的适配抽象接口，方便进一步扩展。

1.4 论文组织结构

第一章为绪论，主要陈述了本文的研究背景以及现状，分析了当前智能家居移动应用的开发模式，以及主流的跨平台移动应用开发框架及工具，选取适合智能家居移动应用的 Apache Cordova，并说明了本文需要在此框架上作进一步扩展研究的需求与意义。

第二章为 Cordova 框架的深入研究，从开发模式、运行原理、模块扩展等方面对该开源框架进行了深入的分析，同时分析了移动平台上的消息推送技术方案，为本文的应用研发做技术铺垫。

第三章为基于 Cordova 的智能家居移动应用的研发设计分析，从总体设计到本地 Sqlite 缓存模块设计、视频监控模块设计、消息推送模块设计和设备控制模块设计，描述了构建该应用所需对 Cordova 框架扩展的设计方案。

第四章是跨平台智能家居移动应用的实现，主要针对第三章的设计方案，进行各个扩展模块的技术实现，描述了实现的核心细节，包括主要类、主要函数以及机制等。

第五章是测试与分析，针对本文的基于 Cordova 框架的智能家居移动应用研发，进行了应用的测试与分析工作。

最后是对本文的应用研发做一个总结，并对下一步研究做简要说明。

1.5 本章小结

本章节主要讲述了本文的研究背景，调研了智能家居移动应用的发展现状，本文针对该领域进行跨平台移动应用开发的研究，通过比较当今主流的移动应用跨平台开发框架及工具，选取较适合智能家居移动应用功能需求的 Apache Cordova，并对其总体研究状况进行探讨，通过 Apache Cordova 与原生应用的功能、性能比较，确立该框架作为智能家居移动应用的开发框架的可行性，并简要分析了该框架的不足以及尚需扩展的方面，确立了研究内容及研究意义。

第二章 Apache Cordova 跨平台框架的技术研究

2.1 Apache Cordova 的总体框架

前面第一章调研了跨平台移动应用开发的主流方法，并针对智能家居移动应用的功能特性及需求，对多个跨平台开发方法及特性进行比较和探讨，最终选择了开源平台 Apache Cordova 来进行应用研究。

从总体来看，Apache Cordova 使开发者可以通过使用 Web 标准技术 HTML、CSS、Javascript 等，进行移动平台的应用程序开发，做到一次性编写逻辑、界面等程序代码，在多个移动平台上编译、运行，从而提高了软件开发效率、缩短软件开发周期、大大减少软件开发和维护的成本。本文研究的智能家居移动应用，将在 Apache Cordova 框架上进行扩展，而在这之前，本文将对该框架进行逐步分析研究。

2.1.1 总体框架分析

Apache Cordova 作为一个跨移动平台的开发框架，其提供了 Web 技术语言调用移动设备操作系统的本地 API 的功能。如图所示，基于 Apache Cordova 开发移动应用，主要通过标准的 Javascript 调用该框架提供的 Javascript 接口，并通过 Apache Cordova 的 Javascript 核心代码，调用经过其封装集成的各平台移动操作系统的本地 API，从而实现开发人员通过 Web 语言调用本地操作系统 API，结合了 Web 技术的跨平台性和本地操作系统的功能丰富性。从开发者的角度来看，就是用简单通用的 Javascript 代码，使用了不同主流移动平台的操作系统原生 API 接口。图中为了方便，仅列出了 Android、iOS、Windows Phone 以及 Blackberry，而现今支持的平台更多，已在上一章节叙述过。

如图所示，在 Apache Cordova 框架里，主要包含了 Javascript 接口模块、定制的 Webview 模块以及集成的各 Native API Plugin 模块。一个基于该框架的移动应用，在 Web 层用 HTML、CSS 和 Javascript 描述了应用 UI 界面，解析与渲染工作交给了 Webview 来负责，而这里的 Webview 根据平台的不同，其内核也有所不同。在开发者调用 Cordova 的 Javascript 接口模块的时，将通过 Webview 的解析后通过定制模块传递分发给 Apache Cordova 框架在各平台集成的本地 Plugin 模块，如调用本地摄像头的 API，这些模块通过本地代码执行完响应的功能后，返回结果给 Webview 并通知上层的 Javascript，开发者可以通过实现 Javascript 层的回调函数以执行返回结果的处理。在整一个调用过程里，涉及了一些模块之间的协同处理，Apache Cordova 框架里通过同步、异步事件来处理不

同层之间的消息传递，后面章节将对这些事件及相关原理进行深入剖析。

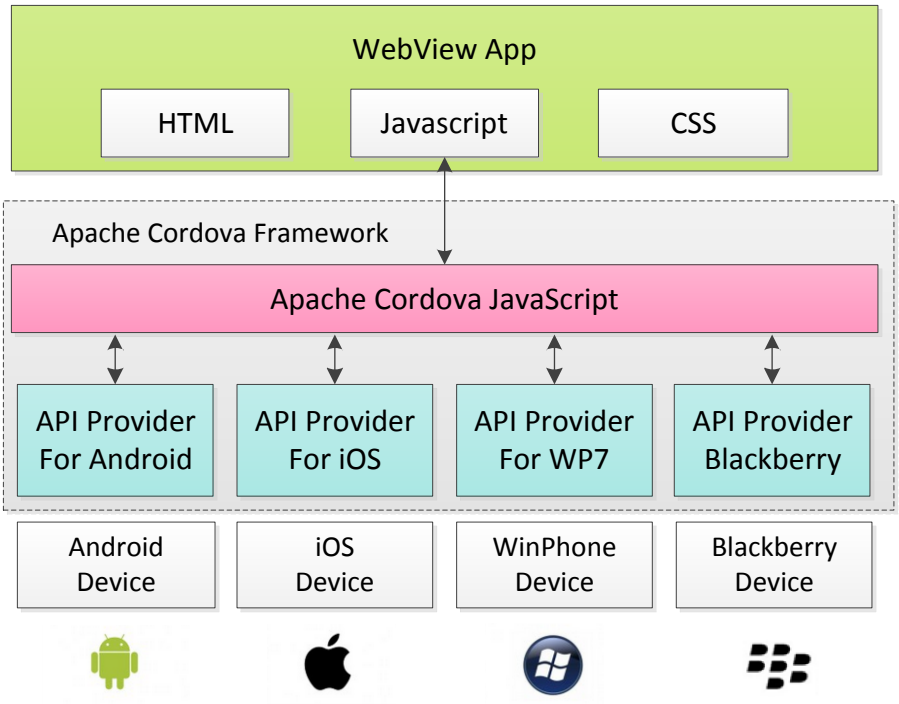

图 2-1 Apache Cordova 的总体框架分析

2.1.2 主要模块组件分析

在 Apache Cordova 框架里，对各个平台的本地 API 进行封装的，并暴露 Javascript 接口给开发者，下面从其框架内部进行主要模块的分析。在 Android 中，其主体类是 CordovaActivity，它继承自 Android 的用户界面类 Activity，主要负责与用户交互，包括用户触摸事件的接受等，处于一个 Android 进程的主线程循环里。其中 CordovaActivity 里一个核心的成员是 CordovaWebview，这个类继承了 Android 系统自带的交互类 Webview，其中封装了浏览器内核的主体功能，而 CordovaWebview 继承其并进行定制，加入一些 Webview 设置初始化、内容的加载、WebviewClient 的错误处理以及主要的用户交互的传递等。另外在 CordovaActivity 中还有 CordovaWebviewClient，其继承自 Android 的 WebviewClient，并重写了页面加载、用户 URL 请求过滤等方法，而 WebviewClient 作为 Wekit 浏览器内核组件，而 CordovaChromeClient 继承了 Wekit 内核的 WebviewChromClient 组件，负责处理 Javascript 的事件处理，如 onJsConfirm 事件、onJsPrompt 事件等。这两大核心组件负责从 Javascript 到 Webview 的事件处理，同时将渲染任务交给 Wekit 核心，最终通过 Webview 显示到用户界面的 Activity 中。

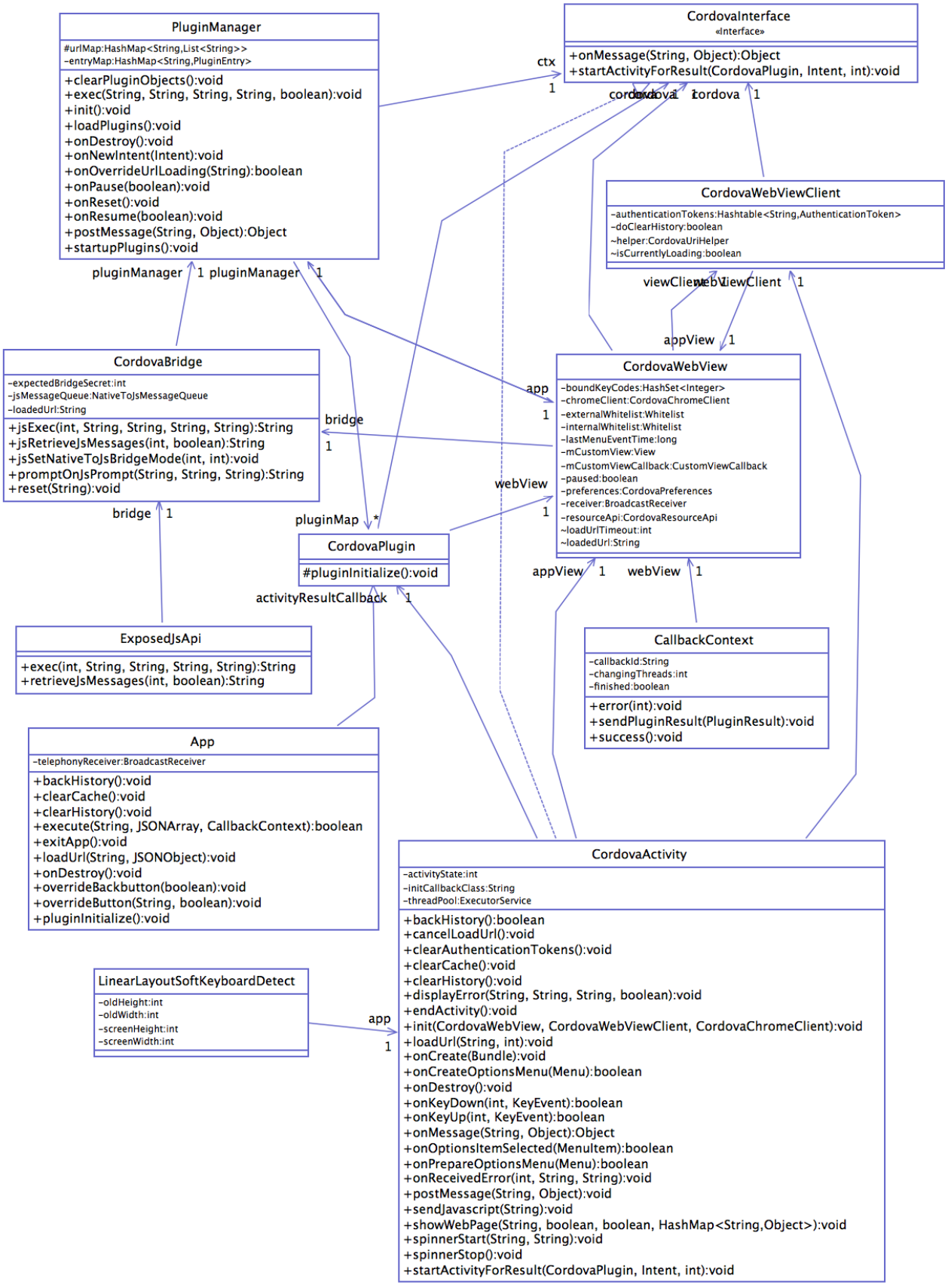

图 2-2 Apache Cordova 主要模块

在开发者通过 Javascript 层进行本地 API 调用的时候，Cordova 框架通过

WebviewChrome 得到 Javascript 调用时，通过 CordovaBridge 将该调用信息传递给 PluginManaer 来处理到底需要调用本地操作系统的哪一个 API，而本地 API 通过 Plugin 的模式，注册在 PluginManager 中，PluginManager 将上层的 Javascript 调用命令进行分发，最终调用经过封装的本地 API。图中显示的是 Android 中的 Apache Cordova 的主要类，而在 iOS 或者其他主流系统中，大致原理类似。总结起来就是基于如下几个部分：

1. 应用程序的主体交互界面，其中初始化了所需的各项配置属性，同时负责应用程序生命周期的管理；
2. 经过封装后的浏览器渲染组件、包含 Javascript 事件处理的浏览器核心组件；
3. CordovaBridge 负责桥接 Javascript 与本地层，将通过浏览器核心捕获的 Javascript 调用传递给本地组件层。
4. PluginManager 组件负责管理封装了本地 API 的 Plugin，以及将 CordovaBridge 传来的调用进行分发，实际地运行本地的 API。
5. Native 到 Javascript 的回调模块，负责处理本地 API 调用后的结果返回，并将结果交由 Web 层的 Javascript 代码处理。

2.2 Apache Cordova 的开发模式

2.2.1 跨平台应用的开发流程

基于 Apache Cordova 框架进行开发应用的过程中，主要分为两大步骤：编写 HTML、CSS 以及 Javascript 部分；在 Javascript 中利用 Cordova 的框架接口调用本地 API，并处理结果返回。而具体来看，Cordova 的应用首先需要对公共配置文件 config.xml 配置，其中指示了该应用的 Web 内容的路径、指定了核心的 Cordova API 属性、Web 层中所用到的本地 API 封装 Plugin 以及一些平台相关的配置信息等。Apache Cordova 提供了一种自动的命令行部署工具——Command-Line-Interface（简称 CLI），CLI 可以根据用户键入的命令来自动生成及部署平台相关的文件，但是由于框架本身也在不断地发展变化，应当注意版本变化带来的兼容性问题。

在 3.0 版本以后，可以使用两种工作流程模式来创建 Apache Cordova 的移动应用，分别是 Cross-platform (CLI) workflow 和 Platform-centered workflow 模式，这两种工作流程主要区别在于对本地 API 的支持问题，前者为单纯使用 Cordova 框架内部集成的接口，这类应用能够使用的本地 API 较少，但是应对逻辑功能较简单的、对复杂本地 API 没有

需求的应用已经绰绰有余；而第二种则会创建一种支持扩展 Plugin 的开发模式，这类应用需要对较复杂的本地 API 功能或者第三方本地组件进行 Plugin 封装、扩展，通过这种方式可以实现功能较丰富的移动应用程序，但是其开发的复杂性也将提高，不过鉴于应用本身具有 Web 层通用模块，既能兼顾跨平台特性，又能满足较复杂的功能需求。本文后续的应用研究设计将使用第二种方式对 Cordova 进行扩展，以满足智能家具移动应用的复杂功能。

2.2.2 开发工具的脚本分析

Cordova 自带的开发、部署脚本工具 Command-Line Interface（CLI），提供了方便的命令行功能，如编译不同平台的 Cordova 应用程序，部署到不同的设备上运行，添加 Cordova 的插件等。该工具基于 Node.js 框架开发，集成了许多基本的 Cordova 开发过程脚本，使开发者可以方便地管理 Cordova 应用程序开发的模块。假设当前开发平台添加了 Android 与 iOS，我们用 CLI 添加一个 Android 端的插件，CLI 脚本利用插件管理模块，帮助你在 Android 平台的目录下，加入相关插件资源文件以及相关编译命令，同理也根据 iOS 的项目结构，置入相关插件模块。在利用 CLI 进行项目编译时，将根据新添加的插件模块，针对不同平台进行编译。

2.3 Apache Cordova 的运行原理

2.3.1 WebView 与 Native 的通信原理

如果从 Javascript 层调用移动操作系统本地 API 的角度来分析。开发者使用的 Javascript API 将通过 Cordova.js 调用 cordova.exec()函数，并传入 callback 作为回调函数、Plugin 名称、调用的服务名以及字符串数组作为调用 API 的参数，该 exec()的实现是通过调用 Javascript 的 prompt 函数来实现的，该 prompt 函数将触发浏览器核心抛出给本地应用程序组件的 onJsPrompt()接口。在得到 Webview 的浏览器核心组件解析时，Cordova 定制的组件 CordovaChromeClient 将捕获到 Javascript 的 prompt 函数调用，并执行 onJsPrompt()函数，而该函数将通过 CordovaBridge 对本次调用进行判断，包括参数验证等。如果不通过则直接返回；若验证通过，将尝试此次调用通过 PluginManager 的 exec 函数进行分发，PluginManager 将查找其维护的一个键值映射，以找通过 getPlugin()函数找到将要执行的 Plugin，并执行该 Plugin 的 execute()函数，当然这个 Plugin 执行的过程，在另外新建立的 Thread 中运行，后续将描述 Cordova 的异步调用处理与消息通信。如果执行完毕，Plugin 将调用 CallbackContext 的函数返回结果，其通过 CordovaWebview 类

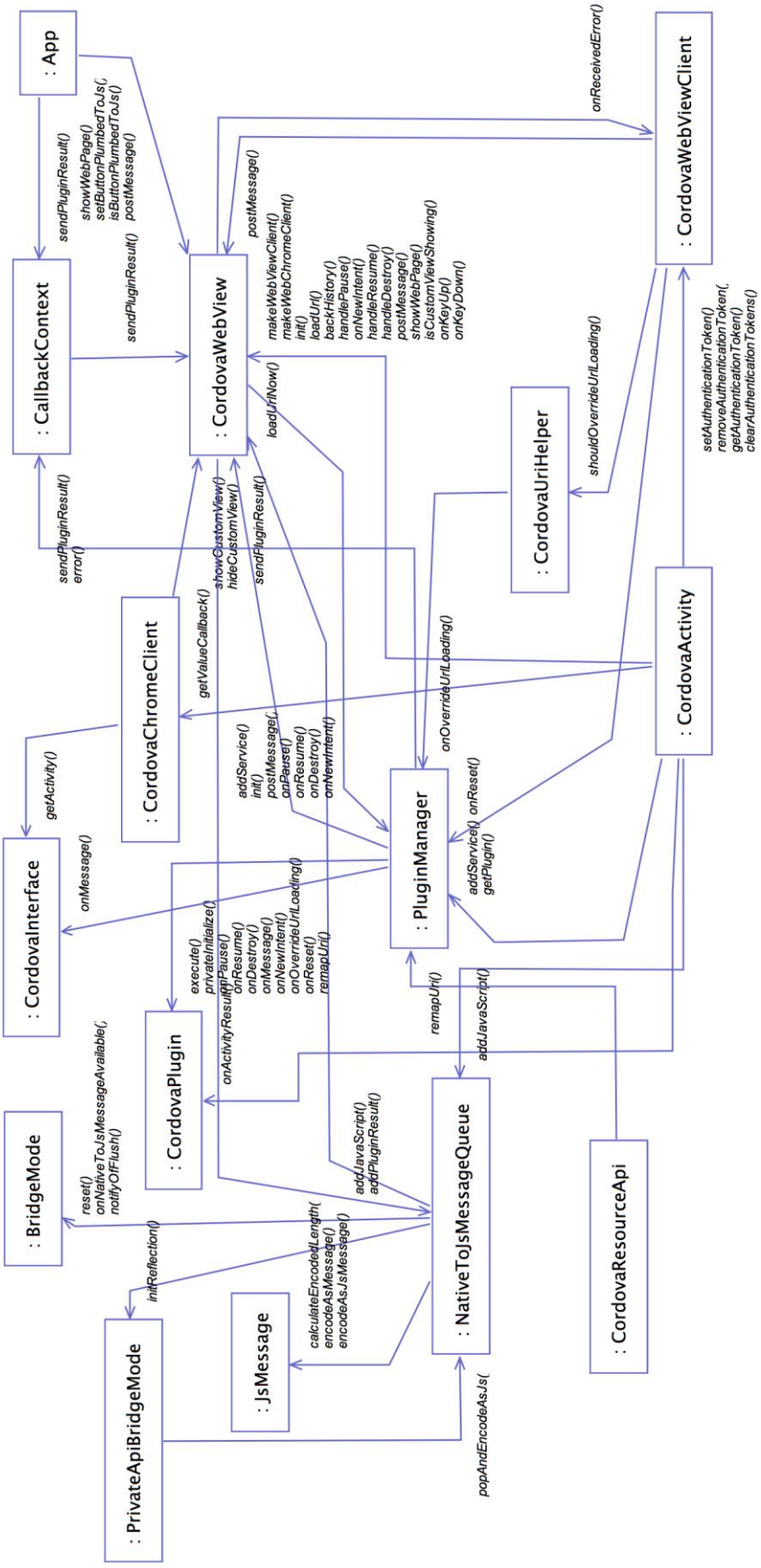

图 2-3 Cordova 的运行机制

的 sendPluginResult()函数，将结果返回，最终将触发 Web 层面的 Javascript 回调函数，而浏览器核心也随着将解析并执行，将结果经由界面响应，返回给用户。对于这个过程其中的具体细节，将在后文分析。

而对于 iOS 中的 Cordova 框架，其与 Android 中的通信原理类似，都是寻找一个切入点来桥接 Javascript 与本地代码。iOS 系统中 Javascript 到本地 Object-C 代码的通讯，首先在 Javascript 端使用了 XMLHttpRequest 发起地址为“/!gap_exec”的请求，然后 iOS 本地代码的 UIWebview 以及 UIViewController 拦截到该特定请求，iOS 通过 NSURLProtocol 拦截到该请求后，将 Javascript 的调用分发到不同的 Cordova Plugin 中去。而还有一种方法就是利用 iframe 的 src 变更来触发 UIWebview 的 delegate，不过主要是以前者为主。对于 Object-C 返回至 Javascript 层，则通过 UIWebview 的其中一个方法 stringByEvaluatingJavaScriptFromString 来实现，该方法可以通过 UIWebview 来执行 Javascript 层的代码。iOS 中的 Cordova 框架始终都通过一个 CallbackId 参数来贯穿整个执行过程，这个 CallbackId 维护了 Javascript 层的回调函数的标识，在本地 Object-C 代码执行完毕后，通过该 CallbackId 来进行 Javascript 层的回调。

当然在其他平台如 Blackberry、WebOS 等的 Cordova 框架实现方式，大致功能模块与 Android、iOS 的相仿，其精髓都是通过一个 Javascript 层到本地代码直接的入口来实现 Javascript 调用本地代码，并通过一个本地代码到 Javascript 层的入口，来实现本地功能模块的结果返回、回调。利用这两点，将上层的 Web 技术与各平台的本地代码桥接起来，同时整合了框架的公共调用接口，使基于 Cordova 的应用程序可以维持一份 Web 技术代码而部署在不同的移动操作系统平台上面。

2.3.2 同步与异步消息处理

在 Cordova 框架中，Javascript 调用本地代码，有两种方式的消息处理模式——同步与异步。在 Android 的 Cordova 框架实现里，通过 PluginManager 进行 Javascript 层的调用分发后，如果找到了对应请求 service 的插件类，则将在适当的时候，通过 Java 反射机制来实例化 Plugin 类，并交由相关 Plugin 去处理，如果找不到则会直接返回 Javascript 层错误代码。在 Cordova 框架里，有一个本地代码到 Javascript 层的队列——NativeToJsMessageQueue，本地代码的返回结果，不管是异步或者同步，都经由这个队列返回到 Javascript 层。而对于消息的同步与异步方式的选择，则在 Plugin 中可以通过重写 isSync()方法来进行设定。如果是同步的方式，则在执行中会计算该模块所执行的

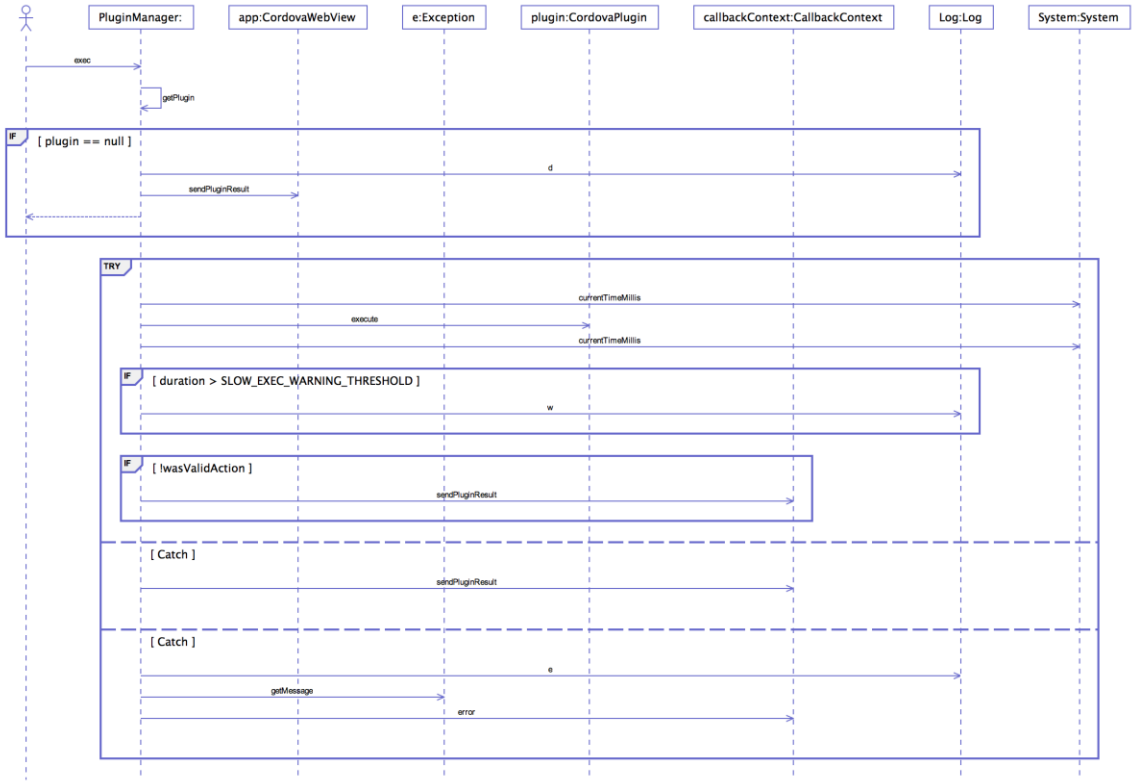

图 2-4 同步调用

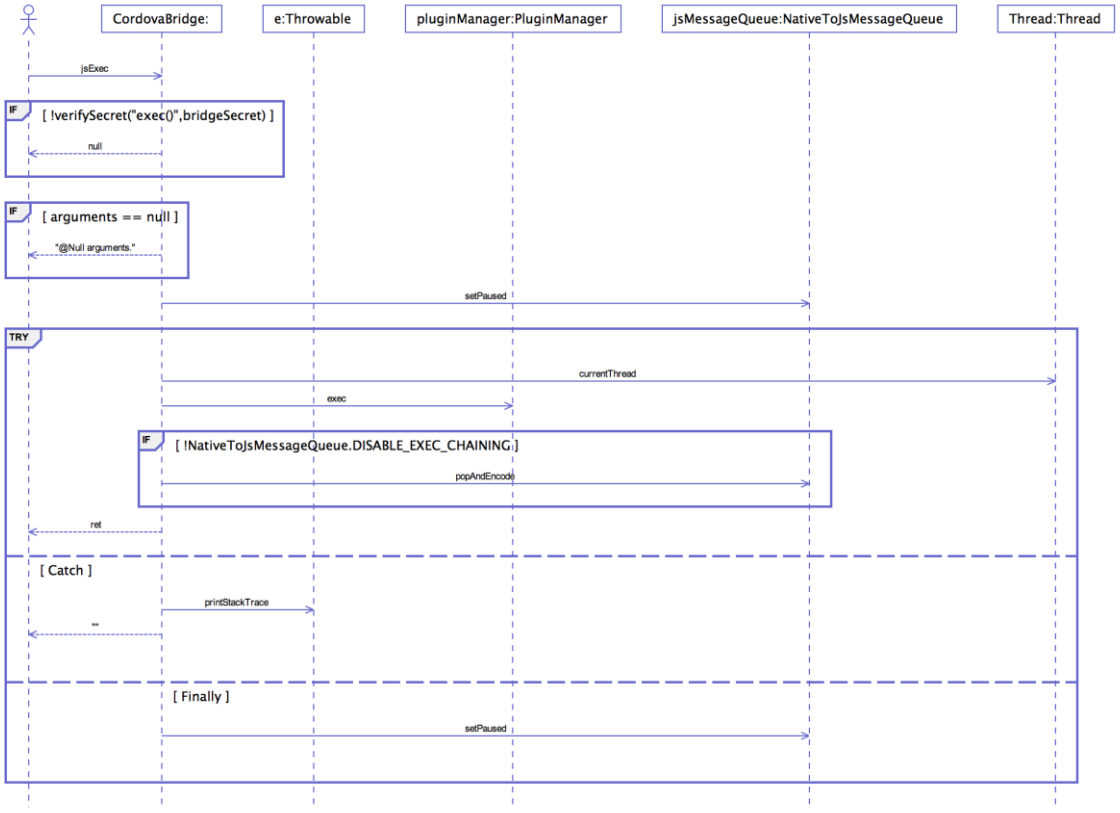

图 2-5 异步调用

事件，如果执行时间大于框架所设定的 SLOW_EXEC_WARNING_THRESHOLD 值，则给予警告。如果该 Plugin 模块使用的是异步的方式，将在本地代码里新建立一个线程来执行 Plugin 的 exec()函数，而最后在执行完毕之后，通过 NativeToJsMessageQueue 队列返回执行结果，达到与 Javascript 的异步通信目的。

而对于 Javascript 层，在异步执行的时候，Cordova 框架在执行 exec()函数的同时，通过 XMLRequest 请求或者 prompt()的方式来进行消息处理轮询，并从返回结果的队列 NativeToJsMessageQueue 中获得执行结果，并根据其中的 CallbackId 来调用 Javascript 层的回调函数。

2.4 Apache Cordova 的模块拓展

在 Apache Cordova 的框架中提供的本地接口中，对大部分轻量级应用已经足够，但是在进行基于 Cordova 的智能家居移动应用开发，需要涉及一些高级应用功能，在前文我们也已经对 Cordova 进行分析并计划将 Cordova 进行模块扩展，在进行模块扩展之前，我们将对 Apache Cordova 的模块扩展方面进行深一步分析。

2.4.1 Plugin 模块分析

在 Cordova 框架中，通过 config.xml 来标识了该应用程序的开发、部署属性，其中也包括应用所用到的 Plugin 的参数。在 Config.xml 中通过 xml 的<feature>节点标识一个插件，其中 feature 节点的 name 属性为插件的名字，而子节点<param>则包括了该插件的具体参数，包括该节点所指向的插件类名以及该插件的具体平台。而在前文提到的 CLI 脚本工具，其将自动管理 Config.xml 的属性，开发者也可以手动管理 config.xml 的插件属性。所有扩展模块必须继承 Plugin 并重写其 exec()函数，来实现扩展模块的功能。

在 Cordova 初始化的时候，框架将通过加载 config.xml 中标明的 Plugin 扩展模块，将他们暂时存放在一个 key-value Map 中，方便查找。而在运行时如果从 Javascript 调用新加入的 Cordova 插件扩展模块，在 exec()的 name 和 service 参数，标明了需要调用的扩展模块的类名和方法，Cordova 框架将会通过 PluginManager 通过已加载 config.xml 配置的 key-valueMap 查找是否有相应的类，并在此时，通过 Java 的反射机制实例化扩展模块。再通过 Plugin 实例中所重写的 exec()代码，来执行扩展功能，以及结果返回。

2.4.2 Hybrid 应用的 Native 适配

对于本文将设计实现的基于 Apache Cordova 的智能家居移动应用，我们将对应用的

平台相关层做适配，下面分析基于 Cordova 开发 Hybrid 应用的主要流程。

任何一个 Hybrid 移动应用，都将秉承跨平台应用开发的优势，就是一份代码在多平台上编译部署，而对于 Hybrid 应用来说，为了丰富跨平台应用程序的功能模块，需要调用本地的代码模块，而设计这类应用程序的时候，将基于 Web 技术的公共代码，与平台相关的本地代码分离。Apache Cordova 框架支持了众多主流移动操作系统，在利用 Cordova 开发时，进行功能模块的扩展需要对多平台进行本地 Plugin 的适配，其主要流程如下：

1. 在 config.xml 中加入扩展插件配置参数声明，如果使用 CLI 脚本，则需要在 CLI 脚本中创建新的代码库，并遵循 Cordova 的 Plugin Specification 编写规范建立 Plugin.xml 配置文件，包括的参数节点有插件名字 name、插件描述 description、Javascript 模块参数 js-module、平台参数 platform 等；
2. 设计并编写好 Javascript 层的调用接口，该接口将提供 Javascript 层调用本地代码的一些列方法，并接受传入的参数，以及返回结果的调用函数。该接口的实现中调用了 Cordova.exec()方法陷入本地代码层执行该 Plugin 的本地代码；
3. 编写对应平台的本地代码，继承 Cordova 的 Plugin 类，并改写类中的 exec()等方法，加入本地代码的执行逻辑以及同步或者异步方式的结果返回；
4. 针对不同平台，加入不同平台的依赖坏境，进行插件的编译；此时如果使用了 CLI 工具，则将根据 Plugin.xml 来生成 config.xml 并进行平台相关的编译过程；
5. 在 Web 层通过标准 Javascript 代码调用该 Plugin 扩展模块的 Javascript 接口，从而执行该 Plugin 的本地代码。

总体来说，如果需要对一个比较特殊的功能进行多平台适配，则需要针对不同的移动操作系统去实现相应的 Native 适配，向上层抛出 Javascript 接口，来完成 Hybrid 移动应用的 Native 适配工作。

2.5 移动应用推送技术分析

2.5.1 Android 与 iOS 平台推送技术分析

在主流的移动平台 Android 与 iOS 平台上，消息推送技术有所不同，这是因其系统架构不同而异。iOS 的推送会比 Android 更易于开发，因为其中对于设备与服务器间的连接维护，由苹果的服务器来负责；而 Android 中，由于 Google 的网络访问在中国大陆地区并不是很理想，所以需要开发者自己去维护服务器与移动设备直接的联系。

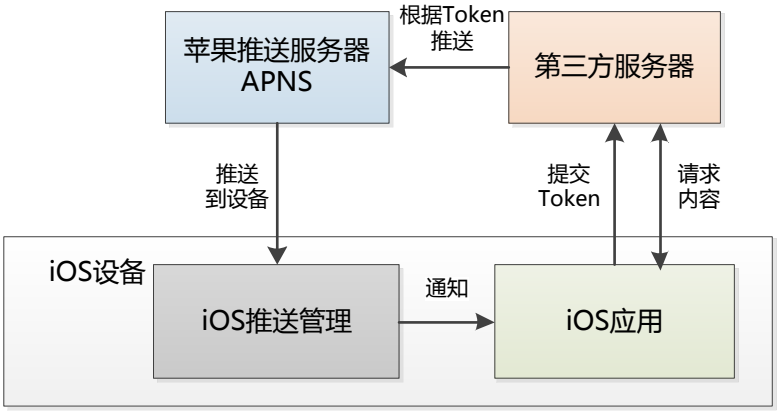

图 2-6 iOS 推送技术

iOS 中的推送技术如图所示，总体来说比 Android 要更简洁，其中第三方服务器将保存 iOS 设备的相关 Token，并与 iOS 的用户 id 进行对应，在需要推送消息时，经由苹果的 APNS 服务器，将内容推送到设备，再由设备通知管理统一处理。在这里，通常推送的消息较为简短，而当用户进行查看时，打开应用后，将再从第三方服务器拉取更多的相关内容。

在 Android 中就没有那么方便，需要应用在后台维护一个与第三方服务器的连接，现今主流技术是通过 TCP 长连接的保活维护，来保持应用与服务器间的连接。当有消息需要进行推送时，服务端直接通过这条链路直接将消息推送至 Android 设备上的应用，当然也可以在推送时推送较简单消息，当有用户交互操作激活应用后再请求更具体的内容。在 Android 中推送技术的应用会比 iOS 中更复杂但更灵活，比如可以直接快速回复消息等功能。

2.5.2 Apache Mina 框架的介绍与分析

在 Android 消息推送技术实现中，我们将进行一个长连接的维护，而服务端使用 Apache Mina 是一个不错的选择。Apache Mina 是一个开源的网络通信开发框架，其使用了 Java 的 NIO 异步网络 IO 处理机制，具有较高的并发性能，对于需要与客户端保持 TCP 长连接的服务端程序来说，是一个非常不错的选择。Apache MINA 框架的总体架构主要分为 IO Service、IO Filter Chain 与 IO Handler。IO Service 主要负责网络 IO 的处理；IO Filter Chain 可以加入编码的处理，如协议的拆包与封包工作；IO Handler 主要负责业务逻辑层的处理工作。基于 Apache Mina 的链式处理架构，我们可以很方便地进行连接管理与消息逻辑处理。

2.6 本章小结

本章主要分析了 Apache Cordova 平台的架构以及运行原理，深入理解 Cordova 的各个模块，研究了 Cordova 的开发模式，分析了其中最主要的运行原理——Javascript 与 Native 层的通信机制，包括 Javascript 调用 Native、Native 回调 Javascript，以及他们直接的同步、异步消息机制，同时深入分析了 Cordova 在 Android、iOS 中的主要实现方式。最后分析了在 Cordova 中进行模块扩展的相关问题，并总结了模块扩展的总体流程，为后文做铺垫。

第三章 基于 Apache Cordova 的智能家居终端应用设计及扩展

3.1 总体设计

本文设计的基于 Cordova 的智能家居移动应用，使用的 Apache Cordova 框架做了扩展。应用主要分为 Web 层、Cordova 框架层以及相关平台的 Native 适配层。基于该总体框架实现智能家居移动应用，能够具有更好的可扩展性，如果需要加入新的移动设备平台，则只要通过加入 Cordova 在该平台下的扩展即可，而上层的基于 Web 技术的代码层则可以复用，并在加入平台相关的 Native 适配后，即可在针对该平台编译出一个性能较好的基于 Cordova 的应用。

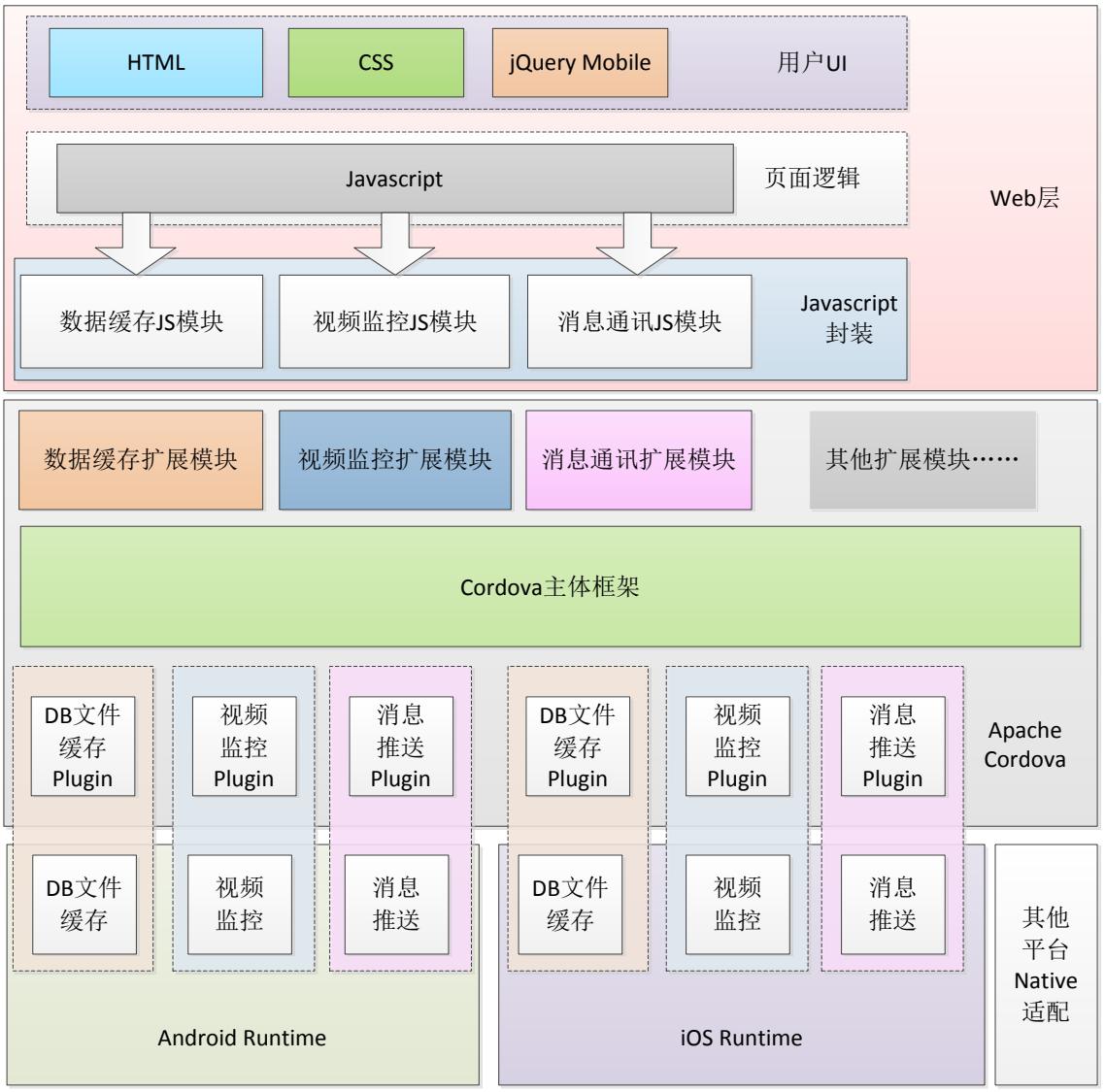

图 3-1 基于 Apache Cordova 的智能家居移动应用总体框架

如上文所述的总体框架设计，包括以下几个部分：Web 层的 UI 及页面逻辑部分、Cordova 层的各个扩展模块封装、所扩展模块基于 Cordova 基础上在各平台的适配模块，以及各平台的运行时环境。由以上几个部分组成了本文所研究的基于 Apache Cordova 构建的智能家居移动应用，该应用利用了 Cordova 的跨平台编译以及标准 Web 技术的优势，可以在满足应用功能需求的同时，减少逻辑的开发和维护成本、减少平台的扩展成本以及增加了功能扩展的灵活性。下面就这几个部分做一个详细分析。

上层的基于 Web 标准技术 HTML、CSS、JAVASCRIPT 的部分，主要包含了用户界面 UI 以及页面逻辑等，Web 标准技术的运用，使应用展现层更易于开发，也可以基于此定制更多的基于 CSS 以及 Javascript 的样式，其中 jQuery Mobile 为一个面向移动设备应用的 UI 框架，其基于纯 Javascript 编写而成，应用特定的 CSS 样式，可以构建出外观绚丽的移动应用。本文将在上层（展现层）主要使用 jQuery Mobile 来设计用户接口界面，同时运用 Javascript 来动态操作 WebView 页面，包括采用 Ajax 技术等局部动态更新用户图形界面，实现一些页面逻辑的跳转等。而在这一层的 Javascript 模块里，可以调用经过 Javascript 封装的 Cordova 扩展模块接口，来实现更复杂的本地功能支持。而具体平台相关的工作，如本地 sqlite 数据的缓存、视频监控模块、消息通讯模块等，则交由给下层的扩展后的 Cordova 框架来处理。

在 Web 层之下的是 Apache Cordova 框架及其扩展模块，其主要包括了 Cordova 主体框架本身，以及本文所做的针对智能家居移动应用提供的几个扩展模块，如本地 sqlite 数据缓存模块、视频监控扩展模块、设备控制扩展模块以及消息推送扩展模块等。本文在前面对智能家居移动应用分析，并针对所需的扩展功能，在 Cordova 框架上设计扩展模块，以满足智能家居移动应用的功能需求，当然如果将来还需要对某些新的功能模块进行扩展，也可以在 Cordova 框架层次上加入。在当前的智能家居移动应用中，主流功能的实现如设备控制、视频监控、安防警报等，可以通过这些所做的扩展上进行开发。

最下面是平台适配层，在这一层次上需要对相应的移动操作系统做扩展模块的 Native 适配。如本地数据缓存模块，在 Android 上需要结合 Android 的运行时环境，来进行扩展插件的开发，在 iOS 平台上，需要运用苹果公司提供的 api 基础上进行开发，而对于数据缓存的模式，也因平台而异，如 Android 的沙盒环境与 iOS 的沙盒环境略有区别，如何控制数据文件的缓存等。

3.2 本地 Sqlite 缓存模块设计

3.2.1 本地 Sqlite 缓存接口设计

在智能家居移动应用中，需要对家电控制的电器数据进行一个维护，通过这些家电设备的控制数据，经过处理后才能得到对应设备的具体控制指令，通常这些指令由不同的字段构成，如设备的 Mac 地址、设备类型编码、设备的功能编码、设备的状态编码等。维护这些数据，如果每次都通过网络请求，将大幅度增加服务器端的压力，而在进行变更或者有新的数据加入时，才进行更新。平时使用控制指令时利用数据缓存功能，则可以大大节省网络资源，同时减少服务端的负载压力。本文将针对此设计一个本地设备数据缓存扩展模块。

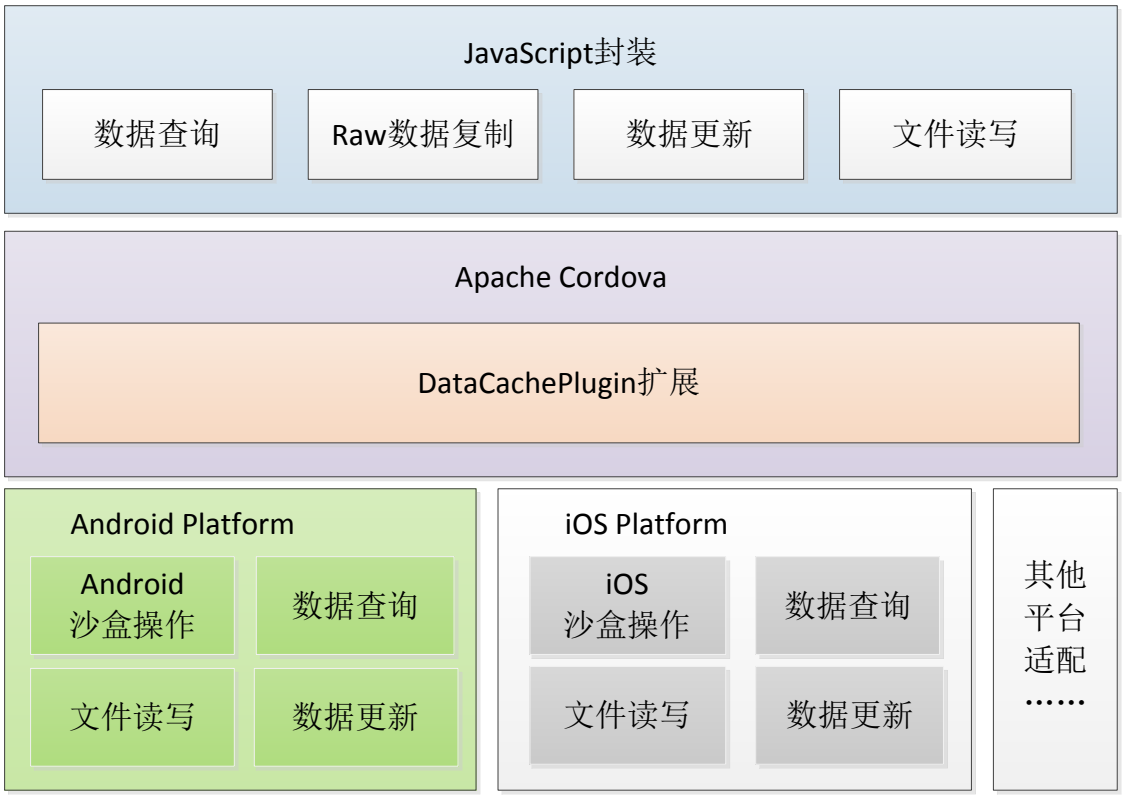

图 3-2 本地 Sqlite 数据缓存模块设计

在移动应用平台中，数据的组织方式有很多，主流的如 XML、Json 或者轻量级数据库 Sqlite 等，本文考虑到设备数据的灵活性以及数据间的关联关系，故采用面向文件形式的嵌入式数据库 Sqlite3 来进行数据缓存。Sqlite3 的数据库组织形式为存储在单一的数据库文件中，如 smart.db 以 db 为后缀的文件。本文将针对数据缓存模块，设计如下几个接口封装：

1. 数据操作；

使用 Sqlite 数据库来组织数据缓存，具备了关系数据库的优势，而 Sqlite 的轻量级也提供了较好的性能，能够满足数据缓存的查询、读取以及写入。本文对 Sqlite 的数据查询做了 Javascript 的接口封装。另外，本文将在 Javascript 层添加 Sqlite 数据写入接口，可以对设备数据缓存操作，在一些少量数据更新的场景，可以使用该接口来做一个增量更新的工作。同时还可以在 Javascript 层保存应用程序的自定义设置参数变量等，更具灵活性。

2. Raw 数据复制；

   Raw 数据作为在移动应用程序开发过程中置入程序内部的数据，程序可以直接对该数据进行操作，但是由于不同平台间的差异，本扩展模块将对此提供平台无关的统一 Javascript 接口。

3. 数据更新；

   此接口将提供对整个缓存数据文件的替换、更新，在 Javascript 层可以通过网络拉取最新的设备数据，并对数据缓存进行替换更新。

4. 文件读写；

   该接口能偶在 Javascript 层提供移动应用程序的本地文件读写功能。如果在程序中有文件 IO 操作，如特殊 Log 日志等，可以使用文件读写接口来操作。

以上四个接口部分，是本文将在本地 Sqlite 缓存扩展模块设计并实现的 Javascript 接口，其提供了在 Javascript 层面操作 Native 的沙盒环境下的缓存数据的功能，能够满足面向智能家居移动应用的功能需求，提高本地数据缓存的利用。

3.2.2 模块适配设计

本文将对此模块做 Android 与 iOS 上的适配设计，在该 Cordova 插件模块中，会针对 Android 平台与 iOS 平台的区别，在 Javascript 层做一个统一。以下分别对两个平台进行分析：

Android 平台是一个多进程系统，其基于 Linux 内核，而 Android 系统运行的权限管理，也是基于 Linux 安全权限管理，在文件读写方面，也沿用了 Linux 的文件权限管理，并在此基础上进行封装改进。Android 为应用程序执行，提供了一个沙盒环境，应用程序运行时，可以对这个沙盒环境进行文件读写等操作，但是不同应用程序之间不能直接访问，只能通过另外的内容分享机制。而在 Android 文件读写中，通常使用 2 种方式来存储缓存数据，第一种是读写外部存储的文件，这种方式可以与其他应用共享数据，但

也将文件暴露于外部，造成数据被串改的风险；第二种就是上述的沙盒环境文件读写，在这个路径内的文件将为应用独享，安全性更高。在实现 raw 数据复制的时候，将把存放于 Android 的 resource 目录下的程序 raw 数据拷贝到 Android 的沙盒环境下，而在下次程序更新的时候，不影响沙盒内的数据缓存。另外，关于 Sqlite 的读写，将通过 Android 自身的 Sqlite 数据库工具来操作数据缓存。

iOS 平台中，同样也有应用沙盒的概念，但是对于实现 raw 数据的复制，需要从应用程序程序包目录将文件复制到沙盒内的 Document 目录下，作为数据缓存来操作。不过应该注意的是，在沙盒内的 tmp 目录下，不能够存放应用程序离线后再次启动需要使用的数据，否则将丢失数据。在 iOS 内进行 Sqlite 数据库的操作，使用其开发环境提供的 Sqlite 库，并在与 Javascript 层通信时，将转换 iOS 的数据结构 NSdictionary 以及 NSArray 来实现。

3.3 视频监控模块的设计

3.3.1 视频监控模块的设计

视频监控模块是智能家居移动应用中一个较为重要的模块，其提供了用户通过移动互联网访问视频监控摄像头来达到远程可视化的目的。基于 Apache Cordova 的智能家居移动应用，需要使用到网络摄像头的视频流处理功能，而这个只能通过 2 种方式来实现：通过 HTML5 的 Video 特性来处理视频流，另外是通过模块扩展来实现。考虑到在市场上的主流网络摄像头产品没有统一的标准，他们大多数提供自己开发的开发包类库，加之 HTML5 技术仍处于发展中的状态，各大浏览器内核的支持也参差不齐，所以通过后者来实现是一个较可行的方案。本文将通过该模块的扩展，尽可能地抽象出公共接口，来支持第三方的视频监控开发方案。

本扩展模块将在 Javascript 层封装常用的视频监控操作接口，供上层的 Web 页面逻辑处理调用，主要有如下几个：

1. 打开、关闭监控。最基本的视频监控调用功能，该接口将跳转至参数设定好的监控视频流界面。
2. 设置视频控制接口。通过该接口可以在 Javascript 层配置具体的视频监控页面信息，包括可以动态地根据参数在监控页面加入控制操作 UI，而具体的 UI 动作响应，将根据第三方 SDK 接入而定。
3. 设置监控参数。该接口将根据 Javascript 层的具体监控参数，可以传入的参数包

括摄像头的网络 IP、网络端口号、用户登陆信息以及其他可选参数。这些设定的参数将作为视频监控初始化的参数。

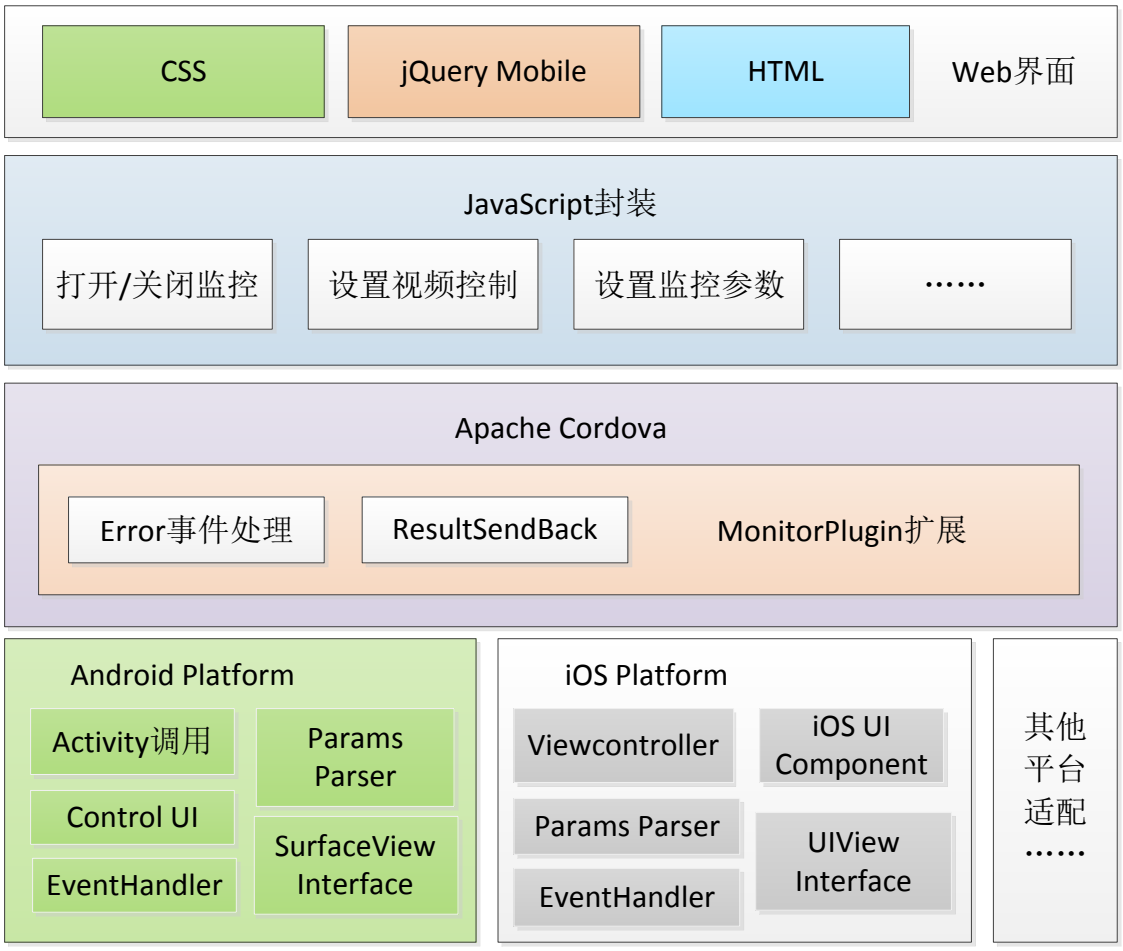

图 3-3 视频监控扩展模块设计

基于以上几个接口设计，上层的 Web 代码可以利用 Javascript 调用来实现视频监控功能。而 Javascript 的接口封装也将通过 Cordova 的 exec()接口与底层通信，通过 Plugin 的实际 Native 代码段来实现具体平台上的功能。

3.3.2 模块适配设计

在扩展模块的 Native 适配设计上，本文对 MonitorPlugin 扩展模块在 iOS 和 Android 上做了适配工作。这两个平台上在底层的 Native 实现上略有区别，下面将分析这两个平台上的主要适配设计：

在 Android 平台上，一个新的本地 APP 页面可以通过建立新的 Activity 来实现，在最新的 Android 版本中还有 Fragment 的碎片化视图，但大体与 Activity 类似。上层的 Javascript 开启视频接口，将通过调用新的 Activity 页面来实现监控界面的跳转。Android 的 Activity 是 Android 应用程序四大组件之一，主要提供了一个可视屏幕，用户可以通

过其进行行为交互，并将最终的处理结果通过此可视地反馈给用户。Activity 在 Android 中有其生命周期，如 onCreate()、onDestroy()、onPause()、onResume()等，应用程序将在 Activity 中进行窗口绘制，处理 UI 事件等。而本模块在调用新的 Activity 后，加入控制 UI、事件处理 Handler 以及视频流组件 SurfaceView 等，通过参数解析器解析后，对 SurfaceView 进行视频播放组件进行初始化，而第三方的视频监控 SDK 将通过对应接口，接入 SurfaceView，来处理网络接受到的视频流。Control UI 也可以通过抽象接口的实现，如果接入了可控制的监控摄像头，则可以通过 Control UI 来发送视频监控摄像头的控制指令。

在 iOS 平台上，一个可视窗口，通常对应了一个 ViewController，而页面的跳转，则通过 ViewController 的切换来实现，与 Android 中的 Activity 非常类似。本模块设计的在 iOS 的 MonitorPlugin，将通过参数解析后，跳转至新的 ViewController，并将接收到的 UI 参数，在监控窗口加入控制 UI 组件，而对于视频监控的视频流组件，则通过 iOS 组件中的 UIView 来实现，第三方的视频监控 SDK 可以通过 UIView 的绑定来接入。而视频监控的事件，则通过 EventHandler 来处理，必要时进行回调处理，将事件处理返回至 Javascript 层。

3.3.3 抽象接口扩展设计

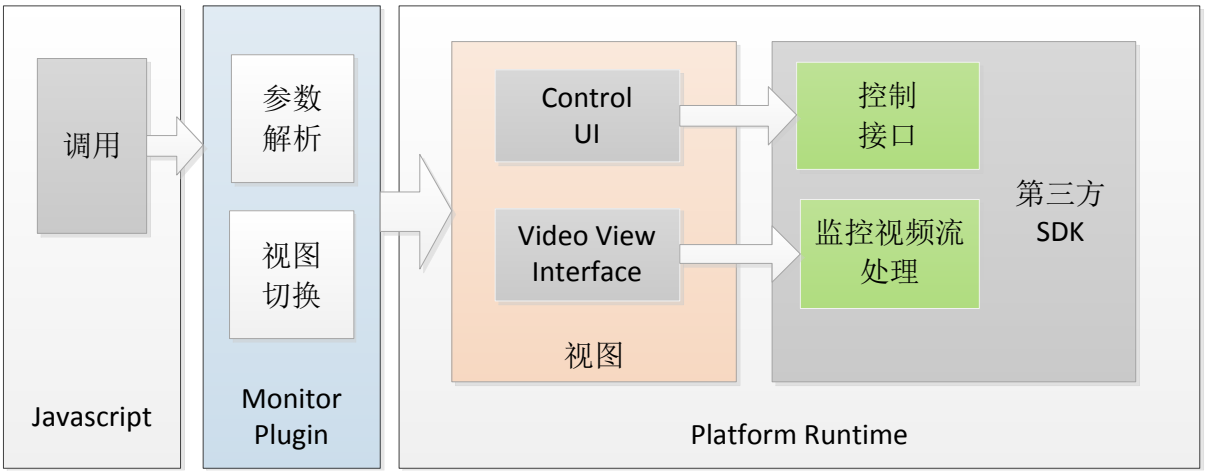

图 3-4 视频监控抽象接口设计

本文设计的视频监控扩展模块中，加入了第三方视频监控 SDK 接入的抽象接口设计。不同厂商不同视频监控摄像产品，在同一平台上具备了他们自己的开发 SDK 包，而本文将把常用的接口抽象出来，以便在对不同的第三方 SDK 进行适配接入时，更为方便，提高了本模块的可扩展性。

如图所示，从 Javascript 层调用入口开始，将传入所需的视屏监控视图初始化参数，包括 Control UI 的控制组件参数，在对视频监控页面初始化后，将调用新的显示窗口，该窗口根据传入的参数，进行控制组件的初始化，而第三方的 SDK 接入，需要遵循本模块的抽象接口设计。其中主要包含如下两个：

1. 控制 UI。在支持对摄像头进行控制的 SDK，需要实现 Control UI 的组件操作，包括注册 UI 的触摸事件，实现动作处理函数调用，将对应动作通过 SDK 的控制模块，发送指令以控制监控设备。
2. 视频流处理 View。该组件将作为主要界面展示给用户，第三方 SDK 的接入需要将视频流处理的显示放到该视图组件中去。

3.4 消息推送模块的设计

3.4.1 消息推送模块的接口设计

在前文的分析中，本文对智能家居移动应用的安防警报、远程消息通讯功能需求进行了分析。而通过 Cordova 框架来实现，需要在框架基础上加入一个消息推送的通讯模块扩展。本节将介绍该扩展模块的设计。

首先考虑到 Javascript 层的公共接口设计，为了让消息通讯更加方便快捷，直接在 Javascript 层封装了如下几个接口供上层 Web 代码调用：

1. 登陆接口 Login()；该接口将根据用户信息参数，做对应的用户登陆操作，当然具体的登陆动作实现，将会因平台而异。
2. 登出接口 Logout()；

   注销本客户端用户的服务端接入状态。
3. 消息接收回调注册接口 regReceived()；

   在 Javascript 层进行消息处理回调函数的注册，在 Native 层接收到的消息将通过回调函数处理。
4. 消息接收回调取消注册 unRegRecieved()；

   注销消息处理回调函数。
5. 发送消息接口 send()；

   在 Javascript 层通过 send 可以发送对应的消息到服务端，send 调用后将根据传入参数进行消息封装打包，包括消息类型、长度等，最后通过与服务端的连接发送出去。而消息的回应，将通过 received 模块进行处理。

6. 检查离线消息接口 check()；

   检查服务器中是否有离线消息。

7. 拉取离线消息接口 poll()；

   拉取未读的离线消息。

8. 设置通知模型接口 SetNotiModel()；

   设置接收到的对应某种类型的消息，需要进行 Notification 通知动作。

以上接口通过 Javascript 封装后供上层调用，在 Web 层的逻辑处理中，可以使用这些接口来与服务器进行消息通讯，其中包含了很主要的一个功能——消息推送，这样就可以通过 Push 的方式来发送某个家电设备的安防警报类信息了。

对于不同的平台，他们的消息通讯、推送的实现会有很大区别，而本文将对 Android 与 iOS 的消息推送做不同的适配。

3.4.2 Android 端的推送设计

在 Android 系统中，本文在前面已经分析过，可以通过 Google 官方的推送技术来实现移动应用程序的消息推送功能。但是由于当前国内对 Google 的网络访问条件不佳，故本文将设计一个轻量级的面向移动互联网络的消息通讯、推送系统。

本文设计的 Android 端的消息推送扩展模块，将采用 TCP 长连接的形式来维持客户端与服务端的连接。在客户端中，Javascript 层将通过已封装的 Javascript 调用，通过 MsgPlugin 消息扩展模块，调用 Android 运行时之上的一个 Service 服务组件，将具体的消息处理交由该服务组件。从总体来说，整个层次架构主要分为如下：

1. Javascript 接口封装。前一小节提到的接口，都将在这里向上层提供，而这些接口的实现，将通过调用 Cordova.js 里的 exec()来交由下层的 Cordova 框架处理。
2. 基于 Cordova 框架的 MsgPlugin 扩展模块。该模块继承了 Cordova 框架的 Plugin 类，重写其中的 exec()方法来加入本地代码的逻辑实现。模块中将处理传入的消息以及参数、处理回调操作以及一些错误事件的处理等。其中关于消息的发送与接收，将交给消息处理 service 组件来处理。
3. 消息处理服务组件。Service 组件是 Android 应用程序的四大组件之一，是运行在 Android 后台的组件，主要负责后台的逻辑处理等工作。在该组件中包括网络通信的心跳维护模块、消息队列的处理、消息通知响应的处理以及多线程间的事件处理器 Handler。其中心跳维护模块通过 Android 的 AlarmManager 与

BroadcastReceiver 结合来实现。AlarmManager 是 Android 中的一种时间服务，BroadcastReceiver 是 Android 的四大组件之一，其能够接收 Android 运行时环境的特定广播消息，本模块的间隔心跳发送将通过 AlarmManager 触发，并通过 BroadcastReceiver 接收心跳触发事件，再将心跳消息置入消息发送队列中去。

4. Android 运行时环境。最基础的就是 Android 运行时环境了，其提供了一些处理工具、服务管理等。

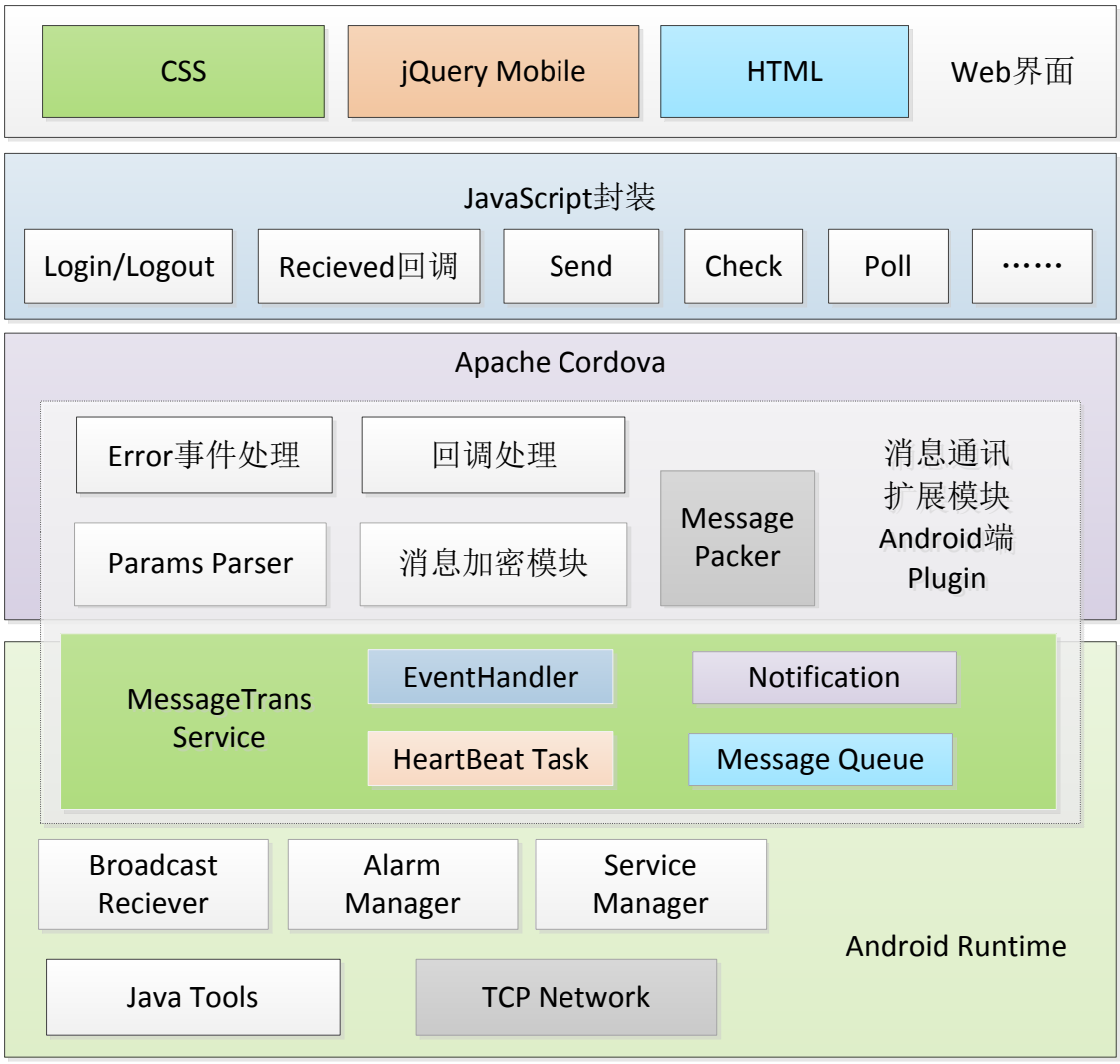

图 3-5 Android 端的消息推送扩展模块

总的来说，本模块就是基于 Android 的 Service 组件来实现的一个适配 Plugin，为 Android 端的智能家居移动应用提供消息推送、通信服务。

3.4.3 Apache MINA 的服务端接入设计

在消息推送模块的服务端程序设计，本文采用了开源框架 Apache MINA 来处理 Android 中的推送消息。当然这种基于 TCP 长连接的推送设计方案已有非常成熟的商业

化解决方案，本文将 Apache MINA 作为服务端接入的框架，利用了其开源的优势，更利于以研究为目的的跨平台智能家居移动应用设计。Apache MINA 基于 Java 的 NIO 来处理 TCP 长连接，拥有优秀的并发处理能力，其将 TCP 长连接抽象为独立的 IOsession，方便开发人员基于框架进行长连接的应用层处理，而 MINA 框架自身封装了一些底层的 TCP 网络 IO 管理，MINA 以链式的过滤器的设计，来处理 TCP 传输内容的封包、解包过程，具有很高的可靠性、易扩展性。

在基于 TCP 长连接的推送服务器设计中，消息的传输协议设计非常重要。XMPP 作为一个通用、开放、自由的消息传输协议，拥有很好的扩展性，但考虑到本文的智能家居移动应用基于移动互联网传输消息，而目前移动互联网大多数计费方式以流量计费为主，低流量的消耗也作为一个应用设计指标，XMPP 的消息传输略显臃肿，会消耗较多的数据流量，故本文设计一个较轻量级的消息传输协议。

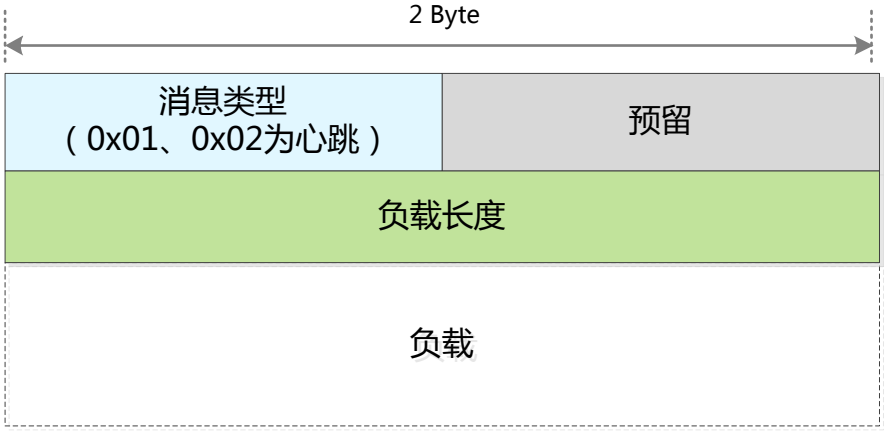

图 3-6 通信消息协议

本文设计的消息通信协议规定，正常的传输消息拥有 4 个字节的头部，第一个字节规定了消息类型，第二个字节为预留字节，第 3 到第 4 两个字节为本消息的负载长度，在消息头后面跟随的是本消息的负载，长度为消息头部 3、4 字节规定的长度。而心跳消息仅 2 字节，包括 1 个字节的消息类型和 1 个字节的预留。由于该消息协议是基于 TCP 传输协议之上，故将消息的完整性、有序性保障交由 TCP 层去处理。消息协议中的心跳包拥有极少的数据量，故在保持 TCP 长连接的过程中，将消耗非常小的数据流量，非常适合移动互联网应用的使用。在消息负载中，本文采取了 Json 格式来携带具体的消息数据，包括智能家居应用中的登陆注销处理信息、安防警报推送信息以及家电控制指令信息等。

3.4.4 iOS 的推送设计

iOS 的推送设计与 Android 的推送设计大不相同，这是因为他们的系统差异而导致的，前文已经介绍过 iOS 的推送技术。那么本文设计的针对 Apache Cordova 智能家居移动应用 iOS 消息推送模块，将利用苹果的 APNS 服务器来实现，iOS 注册用户信息将和设备标识 Device token 进行绑定注册，在服务端，我们将苹果的设备标识 Device token 与智能家居用户信息建立对应关系，在对智能家居移动应用用户进行消息推送时，将简版推送消息发送至苹果 APNS 服务器，当用户的 iOS 设备接收到推送消息时，将提示简版的推送消息，当用户进行交互操作查看推送消息时，将到服务端拉取最新的具体推送消息全文。比如安防警报消息推送过程中，交由 APNS 推送设备标识码后在 iOS 客户端中进行设备警报，当用户交互打开该推送消息并启动 Apache Cordova 的智能家居移动应用时，将到服务端拉取该用户的最新警报具体信息，比如传感器的最新数据值等。

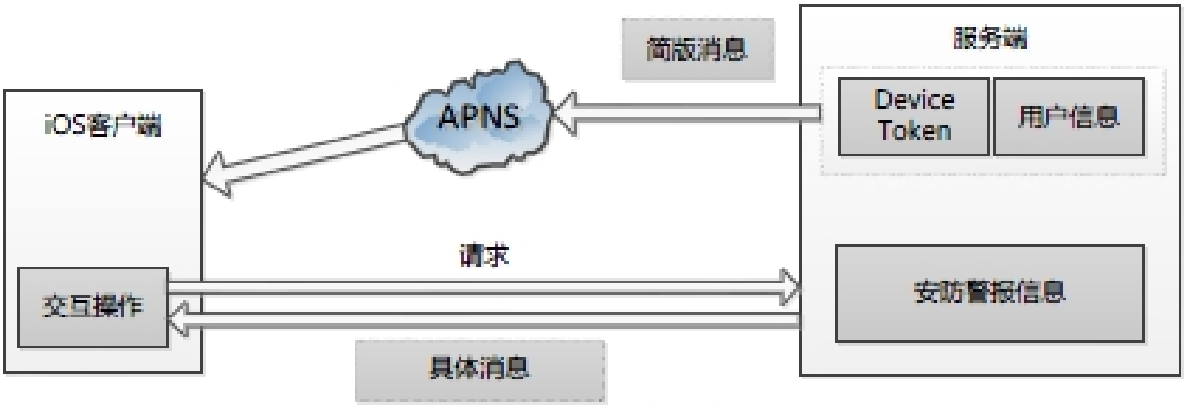

图 3-7 iOS 的推送设计

3.5 设备控制模块的设计

3.5.1 设备控制模块的接口设计

本文研究的基于 Apache Cordova 框架的智能家居移动应用，在对家电设备进行控制时，将使用 2 种方式进行设备指令发送，一种是 HTTP 请求方式，这种请求方法将通过 POST 负载来传输设备控制指令，而在 Cordova 应用的 Javascript 层就可以方便地利用 Ajax 技术来实现异步请求及回调处理，如 jQuery.post(url, [data], [callback], [type])，但是该方法有个劣势就是 HTTP 的无状态特性，使得每次请求都需要携带状态信息（比如用户验证信息、会话信息）；另一种方式是使用 TCP 流传输控制指令，根据前文所述的消息推送协议，由客户端通过该协议发送设备控制指令到服务端，再由服务端转发指令到家庭中控中交由家庭中控处理设备控制指令，并经由家庭内部网络将指令交由具体设

备执行。后者需要调用消息推送模块的接口来实现设备控制指令的传输。

3.5.2 设备控制模块 UI 设计

在设备控制模块中，UI 的生成是基于前文所述的本地 Sqlite 缓存模块来实现的，在 jQuery Mobile 层中通过本地缓存的 Sqlite 设备数据来生成 Webview 中的设备控制 UI 界面，并将用户交互加入其中，当用户进行界面操作时，将生成的具体指令通过设备控制模块发送至服务端进行家电设备的远程控制操作。而在 UI 生成过程中遵循 Sqlite 缓存数据的存储格式协议，该协议规定了房间的设备信息、设备的各种属性及功能等。

3.6 本章小结

本章是基于 Apache Cordova 的智能家居移动应用的设计，以及针对本地功能对 Cordova 进行扩展设计，主要包含了本地 Sqlite 缓存模块设计、视频监控模块设计、消息推送模块设计以及设备控制模块的设计。本地 Sqlite 缓存模块为设备数据缓存提供 Native 支持，视频监控模块为第三方视频监控接入提供 Native 支持，消息推送模块提供了 Android 及 iOS 上的消息推送、传输服务，设备控制模块主要包括了 UI 的生成以及控制指令的封装传输。这一章主要介绍了各模块的总体及功能设计。

第四章 跨平台智能家居终端应用的实现

本章将围绕上一章的各模块设计做各功能模块的实现。其中主要围绕 Apache Cordova 框架做功能扩展实现，实现 Native 的功能并通过 Javascript 层封装以向上提供接口。主要包括了本地 Sqlite 缓存模块的实现、视频监控模块的实现、消息推送模块的实现以及设备控制模块的实现。

4.1 本地 Sqlite 缓存模块的实现

本模块主要实现基于 Cordova 的扩展插件 DataCache Plugin，其中包含文件打开、读写，数据库查询、执行等操作。而 Javascript 层需要通过调用 exec 来陷入 Native 的执行过程，该插件继承了 Cordova Plugin 类并重写 execute 方法的实现，通过此方法进行具体方法调用的分发，并在插件中实现对应功能的私有方法来实现本地的功能逻辑。

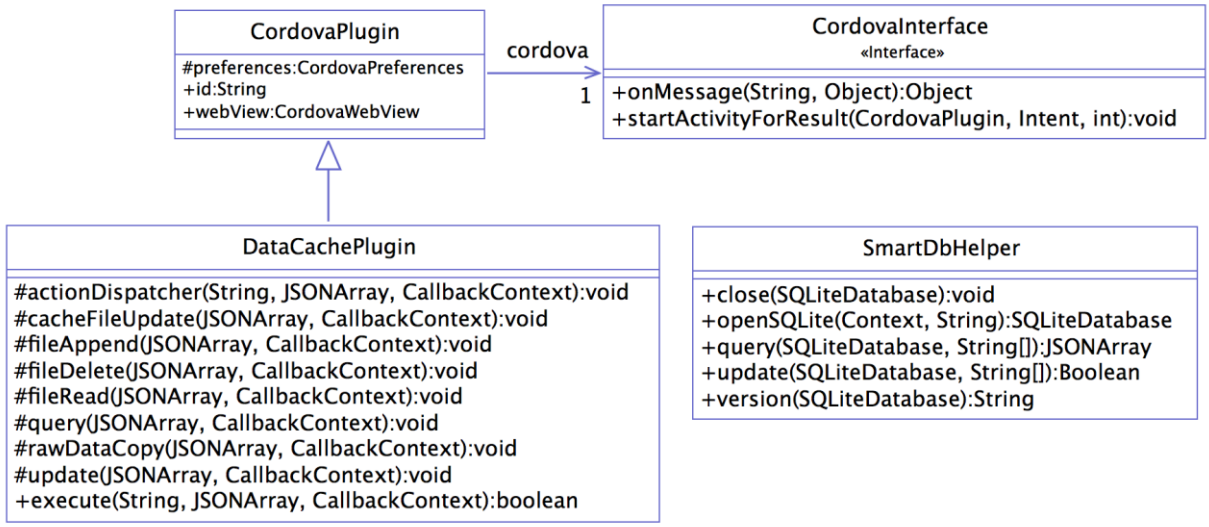

图 4-1 本地 Sqlite 缓存模块的实现

4.1.1 本地文件资源的读写

iOS 的应用程序和 Android 的应用程序，都有自己的运行沙盒，应用程序可以在私有的沙盒里来存取缓存文件，但在不同平台上略有区别。iOS 的应用程序文件读写主要在其沙盒的 document 目录下，而嵌入在应用程序的初始数据文件，不能够进行写操作，需要拷贝到 document 目录下进行读写操作，而在 Android 应用程序中，一般通过应用程序的 assets 或者 raw 目录下放置初始数据文件，并拷贝到应用程序沙盒环境内进行文件

存储操作。本文对本地文件资源的读写，主要通过 Javascript 与 Native 的 Jsonarray 数组来进行通信，将资源文件中读取到的数据以数组方式存储并供回调函数使用，同理在资源文件写入上也一致使用了 Array 的方式。在 Datacache Plugin 插件中实现的文件读取、删除、写入等操作，均传入了由 Javascript 层传递过来的参数，参数为一个 Json 数组结构，可以包括具体写入到缓存文件的内容等。对于文件读写操作，插件主要实现了如下：

A. 文件读操作，fileRead(JsonArray args, CallbackContext ctx)

传入 args 参数为文件名、读取起始字节数、文件读取字节数，通过回调 context 返回所读取的内容，经 JsonArray 传回；

B. 文件写操作，fileWrite(JsonArray args, CallbackContext ctx)

传入 args 参数为文件名、写入起始字节数、写入内容，如果文件不存在将自动新建文件，通过回调 context 返回所写大小，如果错误，将通过 Callbackcontext 返回 error；

C. 文件增加写操作，fileAppend(JsonArray args, CallbackContext ctx)

传入 args 参数为文件名、写入内容，同 fileWrite 相似；

D. 文件删除操作，fileDelete(JsonArray args, CallbackContext ctx)

传入 args 参数为文件名，如果删除成功，则通过 CallbackContext 的 success 返回，否则返回 Error；

如上几个方法实现，为 Javascript 层提供了文件的读写，利用此可以将数据以一定组织形式（如 XML 形式或者 JSON 形式）进行缓存。

4.1.2 本地 Sqlite 文件的读写

Sqlite 数据库在移动设备的运用非常广泛，大多数系统都提供了本地库的支持，在基于 Cordova 的智能家居移动应用中，需要在 Javascript 层来读取、写入应用沙盒内的 Sqlite 数据库，比如读取控制界面信息。本文的实现也是基于 Cordova 的 Javascript 与 Native 通信的，由 Javascript 封装函数的调用，通过 JsonArray 参数传入查询的 sql 语句来交由本地代码进行查询，待本地代码执行完毕后，同样通过 JsonArray 作为存储数据交由 Javascript 回调函数处理。本文封装了 Sqlite 的常用接口如下：

A．查询操作，query(JsonArray args, CallbackContext ctx)

传入 args 参数为查询语句模板、查询参数，查询语句模板的类型为字符串，并将替代的参数使用字符‘?’标识，将依序根据后续的查询参数填入，在 Native

层组成最终 sql 语句，进行查询。查询结果通过 JsonArray 类型返回至 Javascript 层，支持的类型如下：

表 4-1 query 返回至 Javascript 层支持类型

Cursor.FIELD_TYPE_STRING	字符串
Cursor.FIELD_TYPE_INTEGER	整数
Cursor.FIELD_TYPE_NULL	空
Cursor.FIELD_TYPE_FLOAT	浮点

B．更新操作，update(JsonArray args, CallbackContext ctx)

传入参数与 query 类似，通过 sql 模板与参数结合进行 sql 更新语句执行，通过 context 返回 success 或者 Error；

C．插入操作，insert(JsonArray args, CallbackContext ctx)

传入参数使用 sql 模板与参数结合，与上面类似，交由 Native 的 Helper 的 insert() 方法执行，通过 context 返回 success 或者 error；

D．执行操作，sqlexec(JsonArray args, CallbackContext ctx)

传入参数为直接 sql 语句，交由 Native 的 execSQL 来执行语句，由于应用通常只需要前两个操作，第四个操作不提倡使用。而本文将第四个操作进行封装为了操作的完整性。

上述几个方法均通过本地的 SmartDbHelper 来进行 Sqlite 的语句操作，并将 SmartDbHelper 的执行返回结果或者出错结果通过 Cordova 的 CallbackContext 返回至 Javascript 层，交由 Javascript 回调函数处理结果。

4.1.3 本地 Sqlite 文件缓存的插件实现

基于上面两节的文件资源读写与 Sqlite 文件读写的实现，本地 Sqlite 缓存模块的插件实现通过 DataCachePlugin 的 execute 函数陷入本地执行环境，接收参数包括 String 类型的 action，JSONArray 类型的 args 和 CallbackContext 类型的 Javascript 回调上下文。在 execute 函数中，通过 actionDispatcher 根据 action 来进行具体的本地执行逻辑分发，逻辑 action 服务名包括如下：

1. ACT_FILE_READ，文件读取逻辑，调用 fileRead()；
2. ACT_FILE_APPEND，文件附加逻辑，调用 fileAppend()；
3. ACT_FILE_WRITE，文件写入逻辑，调用 fileWrite()；

4. ACT_FILE_DELETE，文件删除逻辑，调用 fileDelete()；
5. ACT_RAW_COPY，应用程序资源文件复制，调用 rawDataCopy()；
6. ACT_SQL_QUERY，Sqlite 数据库查询，调用 SmartDbHelper 的 query()；
7. ACT_SQL_INSERT，Sqlite 数据库插入，调用 SmartDbHelper 的 insert()；
8. ACT_SQL_UPDATE，Sqlite 数据库插入，调用 SmartDbHelper 的 update()；
9. ACT_SQL_EXEC，Sqlite 数据库原生语句执行，调用 SmartDbHelper 的 execSQL()；

在 Javascript 层封装的 DataCache 类中通过 cordova.exec 来调用该插件本地代码，形式为 cordova.exec(successCallback,failCallback,"DbCachePlugin","serviceName",[args]); 其中 successCallback 为执行成功的 Javascript 回调函数；failCallback 为执行失败的 Javascript 回调函数；“DataCachePlugin”为本插件模块的类名；“serviceName”为需要调用的服务名；最后的[args]为传入参数数组，其将在 Native 层转换为 JasonArray 形式的 args。比如需要调用 Sqlite 的 data 缓存查询接口，Javascript 层的封装如下所示：

```
var DataCache = {
    //……
    /* 数据库查询 */
    query: function (sqlStrArray, callback){
        cordova.exec(callback, this.failCallback,
            "DbCachePlugin", "ACT_SQL_QUERY", sqlStrArray)
    },
    //……
    /* 失败回调函数 */
    failCallback: function(msg){
        alert(msg);
    }
    //……
}
```

而调用该接口来获取 Sqlite 数据库中的家电控制信息，只需要在回调函数内使用 Javascript 脚本或者 jQuery Mobile 类库对页面进行更新修改，使用如下所示：

```
//使用：
var sql_str_model = "select ? from ? where ? and ?;"
DataCache.query(
        [sql_str_model,
                "op_desc",
                "Operation",
                "op_devtype == '0x07'",
                "op_parent ='0'"],
        function(result){
                for (var i = 0;i < result.length;i++) {
                        document.getElementById('div_result').innerHTML +=
                                ('<a id="btn_'+i+'" data-role="button">' + result[i][0] + '</a>');
                };
        }
)
```

而对于其他接口的使用，具体的参数以及返回结果回调函数的参数，已经在前文叙述过，总结来说可以非常方便地使用这一系列接口来进行 Javascript 层与 Native 的缓存数据通信，通过本模块可以在应用程序的沙盒进行数据缓存的存取。

4.2 视频监控模块的实现

在视频监控模块中，在不同平台上的本地逻辑实现会有较大的不同，这主要是系统架构导致的，根据前面一章的设计，本文将在视频监控模块做 iOS 与 Android 平台的适配实现，连通 Javascript 与 Native 层的视频监控功能操作，并为第三方视频监控 SDK 接入做 Javascript 的封装以及 Plugin 的适配。

4.2.1 视频监控模块 Android 端插件实现

在 Android 系统中，其界面显示依赖于 Android 底层库的 OpenGL ES 绘制，包括视频流的最终显示，将通过 OpenGL ES 绘制到屏幕上。大部分第三方的视频监控 SDK 将基于接收到的网络视频流，通过视频流解析库，最后绘制到屏幕以播放视频。本文基于前面章节所述的设计方案，对视频监控模块进行 Android 端插件的实现。

在 Android 操作系统中，应用程序的 SurfaceView 是一个视图类，其是一个非常重要且非常灵活的 View 类，在 SurfaceView 里内嵌了 Surface，用它来进行屏幕输出的绘制，Surface 支持后台线程对视图进行更新操作，而在播放视频的过程中，通过网络获取的视频流经过视频解码库解析后，经由 Surface 绘制到屏幕上。

在大多数网络视频播放设备的开发工具包中，都会通过 Surface 的嵌入来实现视频解码播放。对于 SurfaceView，我们可以控制其长宽大小属性，来调整其在屏幕中的显示位置、大小。Android 中的 Activity 是 Android 四大组件之一，Activity 可以理解成一个活动页面，属于前台显示的组件，其生命周期包含了 onCreate 创建、onResume 恢复、onPause 暂停、onStop 停止、onDestroy 销毁等阶段，而在这些阶段都应该对存在于 Activity 内部的用于视频播放的 SurfaceView 做相应的动作，如初始化、暂停播放等操作。

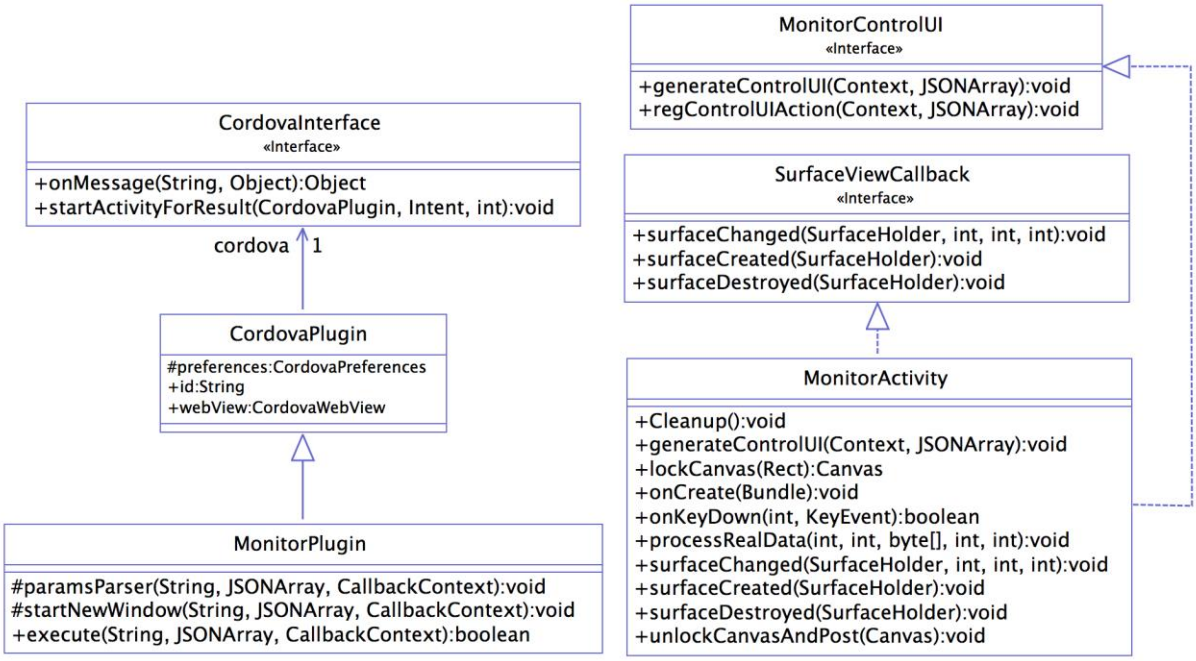

图 4-2 Android 端的 MonitorPlugin 实现

对于 Android 中的视频监控模块 MonitorPlugin 的实现，需要由 Javascript 层调用启动视频监控的接口，然后将视频监控的属性参数等传递到 Native 层进行初始化。由于基于 Cordova 的智能家居移动应用中的主页面是一个 Webview 视图，操作的 UI 也为 Webview 中的元素，故在调用后将启动新的 Activity 界面，再根据传递的参数来初始化视频监控页面元素，包括视频监控区域的大小等。

在 MonitorPlugin 执行 execute 函数时，由 Javascript 层陷入到本地代码执行逻辑，传入的第一个参数为 String 类型的 action，同时附加参数通过 JSONArray 类型的数组传入到 Native 世界，这与前面章节的 execute 一致。基于传入的参数，MonitorPlugin 调用 paramParser 进行参数分析，并通过使用 Bundle 进行参数的打包，并通过 Cordova 的上下文获取当前的 Activity 来启动新的界面 Activity，将打包的参数集合传入到新的界面。当视频监控的新界面进行初始化时，将取出前面传入的 Bundle 参数数据，根据这些数

据进行新界面元素的初始化。

在新界面的 Activity 中的，需要实现 Surface 的接口，包含了 surfaceCreated、surfaceChanged、surfaceDestroyed 三个方法，surfaceCreated 在创建 Surface 的时候调用，在此之前需要初始化第三方的视频监控 SDK，并初始化其解码播放库模块，在 SurfaceCreated 被调用时，将本 Activity 的 SurfaceHolder 传递给第三方的 SDK，交由 SDK 进行界面的绘制，而在解码库解码视频流后，将在此 Surface 上画出视频流，从而播放视频；SurfaceChanged 在 Surface 变化时将调用此方法，比如在移动设备横竖屏幕切换的时候，需要调整 SurfaceView 的大小时，调用此方法以进行相关调整工作。SurfaceDestroyed 方法将在该 SurfaceView 销毁时调用，主要做一些清理工作，比如告知后台的视频解码，已经没有画面可供绘制了。

在 Activity 进行初始化的时候，需要建立 SurfaceView 并进行回调函数的绑定，使用 SurfaceView 的 getHolder()以获得 ViewHolder 后，再进行 addCallback（this）来将本 Activity 实现的上述三个 Surface 操作调用函数的初始化。在初始化过程中，将 Webview 的 Activity 打包传来的参数，建立相对应的 UI 对象并进行初始化。对应的 UI 对象的交互操作则通过注册的形式，在第三方 SDK 接入时，通过实现对应接口并注册到 Plugin 中，既可以通过增加的 UI 对象进行远程摄像头控制。

抽象出来的 UI 控制主要包含：摄像头方向控制、摄像头聚焦操作等。这些 UI 生成的名称以及接口定义在 MonitorPlugin 中，具体定义如下：

表 4-2 控制 UI 参数名称与调用的接口

Javascript 传入参数名称	对应接口	说明
CTRL_TRANS_UP	onCtrlTransUp	摄像头方向上调
CTRL_TRANS_DOWN	onCtrlTransDown	摄像头方向下调
CTRL_TRANS_LEFT	onCtrlTransLeft	摄像头方向左调
CTRL_TRANS_RIGHT	onCtrlTransRight	摄像头方向右调
CTRL_FOCUS_OUT	onCtrlFocusOut	摄像头焦点拉远
CTRL_FOCUS_IN	onCtrlFocusIn	摄像头焦点拉近
CTRL_ZOOM_OUT	onCtrlZoomOut	摄像头视距拉远
CTRL_ZOOM_IN	onCtrlZoomIn	摄像头视距拉近

本文在第三方接入 SDK 方面，因网络摄像头使用了国内较为主流的海康威视的系列

产品，故接入了其官方的 SDK。其中包含了 2 个工具类，一个是 NetSDK 用于登陆网络视频监控、接收网络视频流，另一个是 PlaySDK 用于视频流的解码，当然也可以使用其他解码工具库来解析、播放视频流。在基于 Cordova 的智能家居移动应用中，进行视频监控的管理，在需要登陆并预览网络摄像头时，通过 MonitorPlugin 调用新的播放界面来播放视频流。在开发中需要接入的工作包括在视频监控页面 Activity 的初始化及关键生命周期函数中进行摄像头的登陆、播放、暂停等操作。并实现对应的摄像头控制操作接口，在初始化阶段注册绑定相关参数。另外需要编写好视频流处理函数，如 SDK 中的 NET_DVR_PlayBackControl_V40 函数是接收网络视频流的，并将播放模块的 play 函数注册到视频接收回调中，将 Surface 传递给 play 函数去绘制，最终将图像显示到屏幕中。从播放流程上来看，主要为如下：

1. 在 Cordova 的智能家居移动应用主页面进入视频监控设备管理页面；
2. 在用户交互制定播放某个视频监控设备时，根据 Sqlite 模块读取到的设备信息，包括登陆信息、设备端口、以及控制 UI 初始化参数等；
3. 通过 Javascript 层的 MonitorPlugin 封装接口陷入 MonitorPlugin 的本地模块执行具体逻辑，取出参数并接卸，接着调用 startActivity 来启动视频监控界面；
4. 在视频监控页面初始化阶段，即 Activity 的 onCreate 阶段，进行 SurfaceView 初始化，控制 UI 的生成，初始化 SDK，并注册控制操作的 SDK 相关函数；
5. 接下来根据用户交互进行摄像头登陆操作、回放操作等，在视频回放操作过程中，接收到的视频流通过解码工具解析并通过 SurfaceView 显示到屏幕中；
6. 在用户交互进行控制摄像头的操作时，调用相关的接口，而对应的已注册的 SDK 控制函数将被调用，从而进行摄像头控制操作。

至此 Android 端的 MonitorPlugin 实现已经完成并能够方便地在更换不同厂商摄像头后，进行第三方 SDK 的简易接入操作。均可以通过主界面的 Webview 来管理，并通过 MonitorPlugin 的本地功能来提供摄像头相关操作功能。

4.2.2 视频监控模块 iOS 端插件实现

在 iOS 系统中，MonitorPlugin 插件的总体适配工作与前一小节的 Android 适配大体类似，都遵循了由 Javascript 层传递参数至 Native 代码执行逻辑。在 iOS 中，实现类似前面所述的 Android 的活动界面 Activity 跳转，是由 ViewController 来负责这个工作的。在从 Javascript 陷入本地代码的执行逻辑中，当调用 cordova.exec()时，将根据传入参数

调用本地代码，比如：

cordova.exec(successCallback,failedCallback,'CDVMonitor','start',['user','pswd']);

将调用 CDVMonitor 类的 start 函数，而 CDVMonitor 是继承了 CDVPlugin 的类，在 start 执行函数中将带有 CDVInvokedUrlCommand 类型的参数 command，通过 command 获取 arguments 参数，其是 NSArray 类型的数组，我们可以从中获取 Javascript 层传递过来的参数，并用于本地代码的执行逻辑初始化工作。

在 MonitorPlugin 的实现中，将把从 command 里获取的 NSarray 数组里的参数进行解析，调用 paramsParse 函数来提取参数，将他们用 NSMutableDictionary 类型封装，在调用新的界面用于视频监控显示时，把 NSMutableDictionary 封装的参数，通过 setParams 函数设定传递参数，交由下一个 View 视图界面作初始化工作。

在新的 ViewController 启动时，在 viewDidLoad 函数中提取 NSMutableDictionary 封装的参数，进行此视图的初始化工作。第三方的视频 SDK 接入工作，也将在这里进行。与 Android 的 SurfaceView 类似，在 iOS 中，通过 UIView 类型来显示视频流播放视图，将把 UIView 传递给第三方视频 SDK 去进行视频流的解码绘制。

下面是 MonitorPlugin 的 Interface 声明：

```
@interface MonitorPlugin : CDVPlugin
{
CDVInvokedUrlCommand *thisCommand;
NSDictionary *params;
CDVPluginResult *result;
}

@property NSInteger status;

-(void) paramsParse: (CDVInvokedUrlCommand *) command;
-(void) initial;
-(void) presentNewViewController: (CDVInvokedUrlCommand *) command;
@end
```

其中 CDVPluginResult 是一个用于返回的类型工具，本地代码的逻辑执行结果，将通过此对象的 resultWithStatus 函数，设定本地代码返回状态，并将所带的参数通过 messageAsArray 打包，最后通过 CDVCommandDelegate 的 sendPluginResult 返回至 Javascript 层，调用回调函数处理结果。

在 iOS 中的整体调用流程与 Android 中类似，总结起来如图所示，当通过 Cordova

框架的 Javascript 的 exec()入口陷入本地代码后，将对所携带的参数数组进行 paramParse 解析，如果解析出错的话，将直接通过 CDVCommandDelegate 的 sendPluginResult 返回错误结果，用插件记录的 Javascript 层的 CallbackId 来调用回调函数处理结果。如果解析参数没有问题，将参数通过字典打包封装传递至下一个 Viewcontroller，在新的 ViewController 中初始化 UI，并在用户交互下运行各种操作，如视频流解析，通过 UIView 交由第三方的接入去处理。

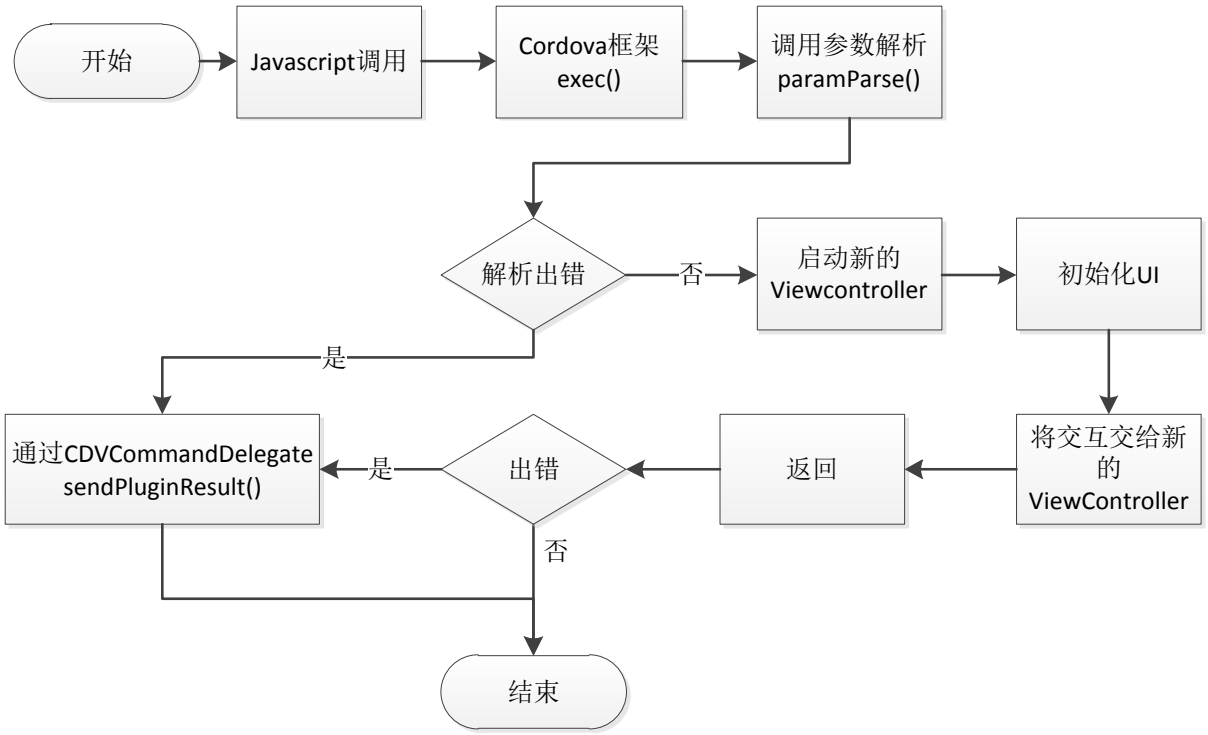

图 4-3 iOS 端 MonitorPlugin 调用流程

总体来说，iOS 中的适配与 Android 中的类似，同时也可以比较方便地接入不同厂商的视频监控设备的 SDK，并在智能家居移动应用的主界面 Webview 中管理、调用视频监控操作。

4.3 消息推送模块的实现

消息推送模块为 Javascript 上层逻辑提供消息通讯的 Native 支持，是面向连接的消息通讯服务。基于 TCP 通道建立与服务器之间的联系，并通过心跳包的形式进行连接保活，随时接受来自服务端的推送。在 Android 中的实现，是基于 Service 组件来实现 TCP 长连接的维护；而在 iOS 中的实现，是基于 APNS 与 iOS 设备间的连接来进行消息的推送处理。

4.3.1 消息推送插件的实现

在实现消息推送插件上，由于 iOS 的大多数保活与接收消息推送的工作交由 iOS 系统本身去处理，消息经由 APNS 推送至 iOS 设备显得非常方便、简单。而 Android 系统在中国境内由于网络访问的问题，不能够正常使用 Google 自带的推送服务，并且大多数国内的手机均经过不同程度的定制，故需要应用本身去维护一个推送的工作机制。本文针对前面章节的设计，下面详细描述在 Android 中的消息推送插件的实现。

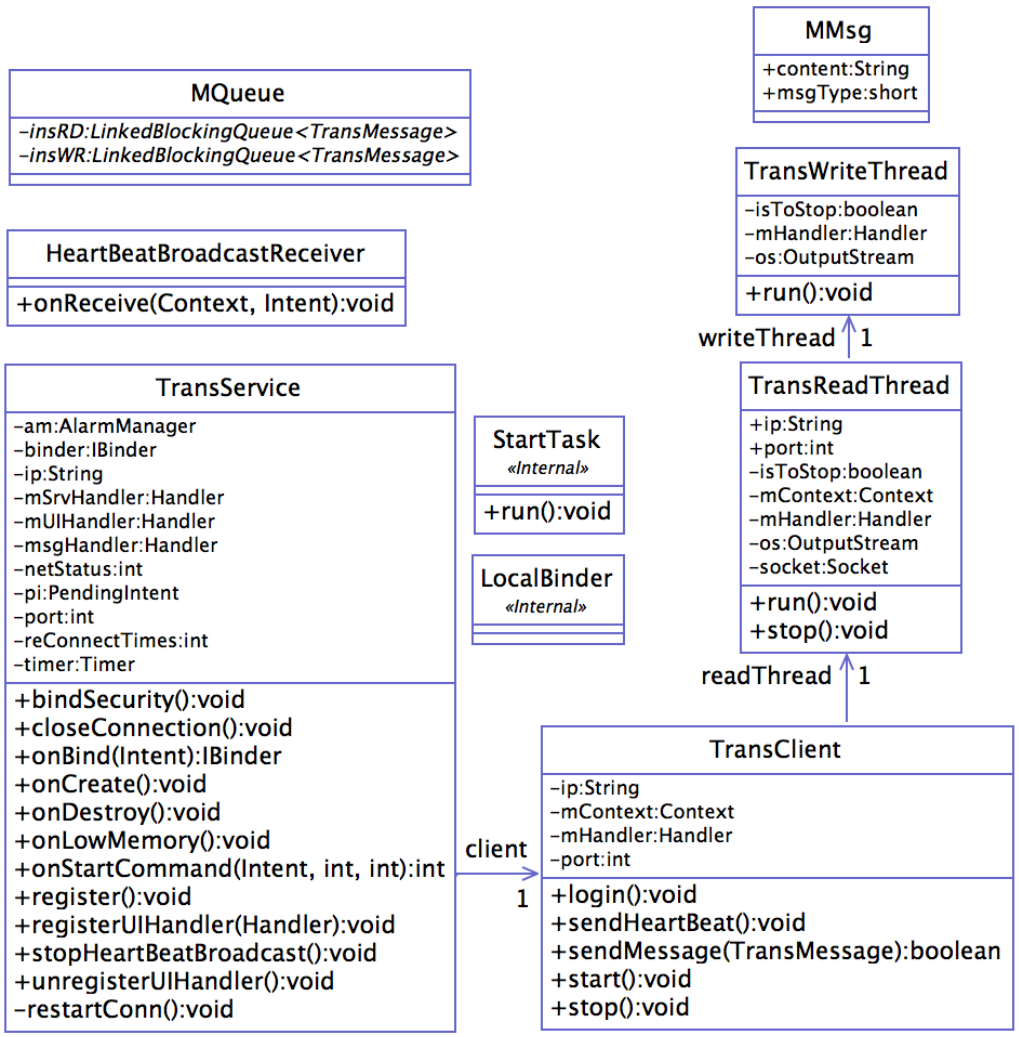

图 4-4 Android 端消息通信扩展模块 Service 实现

在整个插件的实现上，还是围绕 Cordova 框架的 Javascript 与本地代码直接的交互、切换。在 Javascript 层调用消息推送插件的接口时，将把需要发送的消息通过本地代码的后台组件发送出去，同时后台组件接收到的消息即可以通过回调函数来交由 Javascript 层去处理，比如对 Webview 中的元素进行 UI 更新，也可以交由本地代码来处理，比如

在系统中显示 Notification 消息提示用户。而具体的网络连接以及消息发送与接收的工作，则交由后台的 Service 组件 TransService 去处理。

TransService 是一个 Android 的 Service 服务，其生命周期较用户界面 Activity 有所不同，在用户切换到应用程序外时也可以工作运行。在 TransService 中主要包含了一下一些关键组件：

A. TransClient 对象，用来维护网络连接与消息通讯，所有网络通信的逻辑都交由它来处理；
B. 一些 Handler 对象，用来处理多线程间的消息通知处理，由于 Service 组件是运行于主线程上的，但是网络通信的组件将开辟多个线程，将使用 Handler 来进行线程间的通知；
C. AlarmManager 对象，注册 Android 的时钟管理，当该对象的计时触发时将启动一个系统广播，利用广播接收来做定时发送心跳包的任务；

通过这三个组件来进行 TransService 的逻辑执行，而 TransService 中定义的各项行为，将通过这些组件来工作，如用户登陆、登出、重连、发送消息、接收消息等。在连接由于网络不稳定因素中断时，将进行自动重连操作，每次自动重连的时间间隔为前一次的两倍，直至大于 30 分钟。消息推送插件还定义了接收系统网络连接状态变化的广播，在用户进行网络切换的时候将被告知，再进行连接、断开等操作。

TransClient 维护了 2 个线程，读线程与写线程，它们负责网络通信的消息接收与发送，对于消息的管理，采用的是队列的形式，利用生产者消费者设计模式来阻塞与唤醒线程。在每个线程中都传入一个 Handler 负责通知主线程，以便做出 UI 更新等操作。在这里主要有三个事件去通知已注册的 UI，分别为：

1. EVENT_NET_OFFLINE，离线状态；
2. EVENT_NET_ONLINE，在线状态；
3. EVENT_NET_CONNECTING，连接状态；

而在状态切换时，将通知注册到 TransService 的 UI 进行更新状态，在 PushPlugin 中，Javascript 层通过 registerEventHandler 来进行注册 Service 的事件处理，例如当由离线状态更换为在线状态时（即登陆成功），将发布给 EVENT_NET_ONLINE 事件的订阅者，然后通过 PushPlugin 中已注册的对象返回至 Javascript 层，交由 Javascript 层去处理这个事件的 UI 响应，如通过 HTML 元素变更来直观地显示用户登陆状态。

而对于 Handler 中的各消息处理，主要分为两类，一类是处理代码执行逻辑的消息

类型，主要用于协调网络通信与多线程间的各项逻辑关系，比如网络断开、网络不可用、线程的终止与启动等；另一类是与消息协议相关的消息通知类型，主要用于处理与服务器通信的不同消息类型的逻辑，比如心跳消息处理、警报消息处理、控制消息处理等。

在 TransService 中注册了一个系统的时钟管理对象 AlarmManager，用于触发心跳包的发送，本文的实现中采用了 2 字节的心跳包与服务器通信进行 TCP 保活，防止移动互联网中运营商的路由对时间较长的不活跃 TCP 连接进行路由端口回收。AlarmManager 触发时，将进行一次广播，并通过 HeartBeatBroadcast 接收，发送指令给 TransService，调用 TransClient 的心跳发送，这里采用这样的机制实现，是因为如果使用一个线程去发送心跳包，将受到 Android 系统的休眠影响，在进入手机休眠时线程将被暂停，故不能将心跳包成功发送出去，会导致连接断开、失效，最终导致接收不到服务器端的推送消息。而利用 AlarmManager，则可以利用 Android 休眠时的时钟处理器来唤醒 CPU 执行心跳任务。

总结消息推送插件 PushPlugin 的实现，是通过一个后台的 Service 提供网络通信服务，封装了网络通信的操作，并在 Plugin 内提供接口给 Javascript 进行调用、注册对象，当接收到消息或者接收到事件通知时，由 Service 的线程通知到主线程来触发 Plugin 回传给 Javascript 层进行回调。

4.3.2 Apache MINA 服务端实现

在消息推送模块的服务端程序的实现上，本文采用了 Apache Mina 开源框架。如图所示，Apache Mina 框架基于 Java 虚拟机，服务端程序采用 Java 语言开发，主要包含了三个子模块，分别是 LongConnection Handler 长连接控制模块、Logic Handler 逻辑控制模块以及 DataBase 数据库模块。下面主要说明一下各模块的功能实现：

A. Long Connection Handler 模块主要包含了 3 个主要部分：Protocol Parser 协议解析模块主要负责将新到达的消息进行本文设计的消息传输协议进行解析，将解包后的应用消息交由上层逻辑模块去分发处理；Heart Beat Handler 负责心跳控制管理，在接收到消息类型为心跳包时，将由此模块处理并发送心跳包给对端，以达到维护长连接的目的；Protocol Packer 用于将逻辑层的消息按照消息协议进行封包，并向上层逻辑指定的对端（在 Apache Mina 框架中通过 ioSessionId 来标识），将消息发送出去。

B. Logic Handler 模块主要为消息处理逻辑。当接收到的消息解封后，经由 Message

Dispatcher 进行分发，交给对应消息类型的处理模块去处理，包括：Login Handler 负责登陆逻辑、AlarmManager Handler 负责警报信息逻辑、Update Message Handler 负责程序更新消息处理等。Timer 负责一些时间上的计时，比如当一个新接入的连接，在规定时间内没有进行登陆动作的话，将主动断开与对端的连接。当对端发送了登陆消息并成功登入，则通过 LongConnHolder 来记录这个长连接的会话，其中包含了一个关联数组，以会话的 ioSessionId 作为 Key 来存储 IoSession 的引用。

C. Database 模块主要向 Logic 模块提供了数据存储的支持，包括一些离线消息的持久化。本文在实现时，使用了 Hibernate 框架与 Mysql 搭配来进行会话关系以及离线消息等信息的落地工作，防止了数据的丢失。

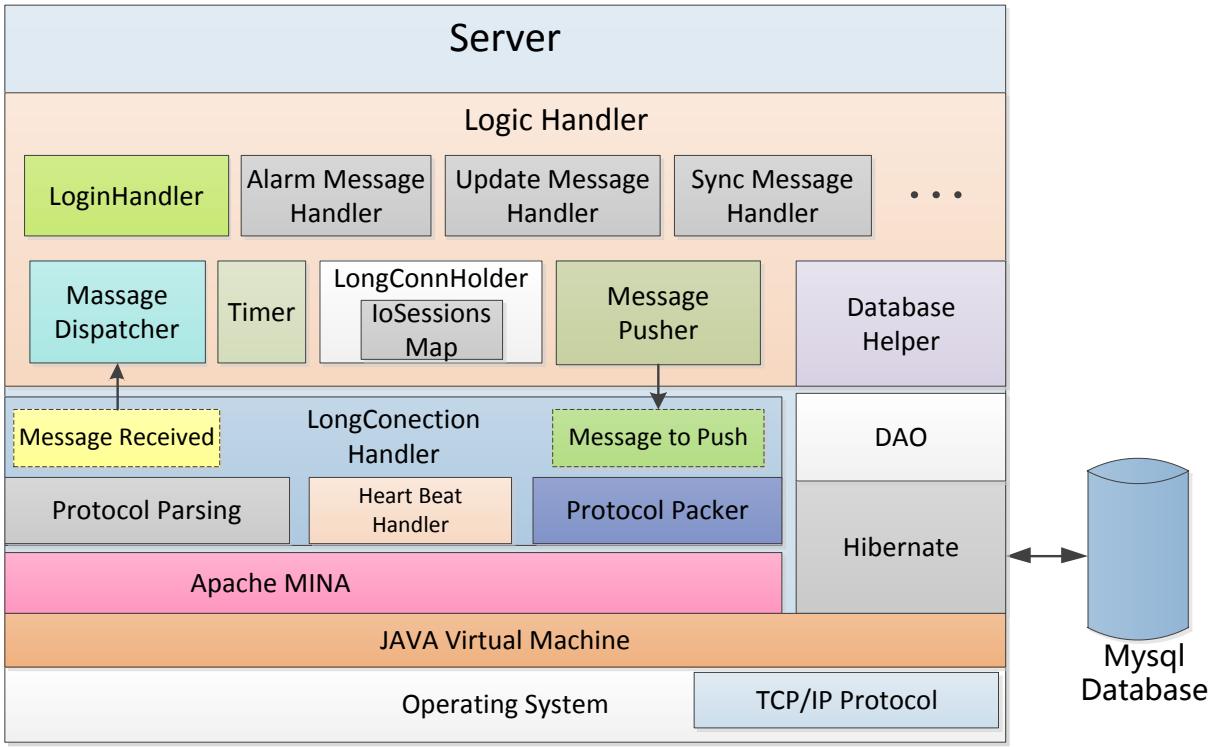

图 4-5 Apache Mina 服务端程序架构

基于以上几大模块，服务端程序可以处理远端的 TCP 连接接入，并进行消息推送，经由 Message 分发器分发后，由具体逻辑模块处理，再经由 Protocol Packer 封装消息推送出去。在智能家居移动应用与家庭主控端的程序接入服务端后，就可以达到远程控制消息转发（移动终端到主控）、警报消息推送（主控到移动终端）等功能。

4.4 设备控制模块的实现

设备控制模块主要负责智能家居移动应用中的主体页面显示、家电设备管理等，其中 UI 显示主要交由 HTML 与 jQuery 来进行渲染，其中的页面内容，将根据本地的设备缓存来生成。UI 的用户交互动作，将调用相关模块的 Plugin 交由本地代码执行，包括启动视频监控、进行设备控制消息的通信等。如图所示，在进入主页面后，将查询本地缓存数据，根据解析回传的结果数组进行 UI 页面更新，这里使用了 jQuery Mobile 来操控 Webview 页面的 HTML 元素，并加入响应的交互处理事件，在用户进行页面 UI 元素交互操作时，将触发对应的 Javascript 处理逻辑，如通过消息通讯接口进行远程设备控制指令的发送。

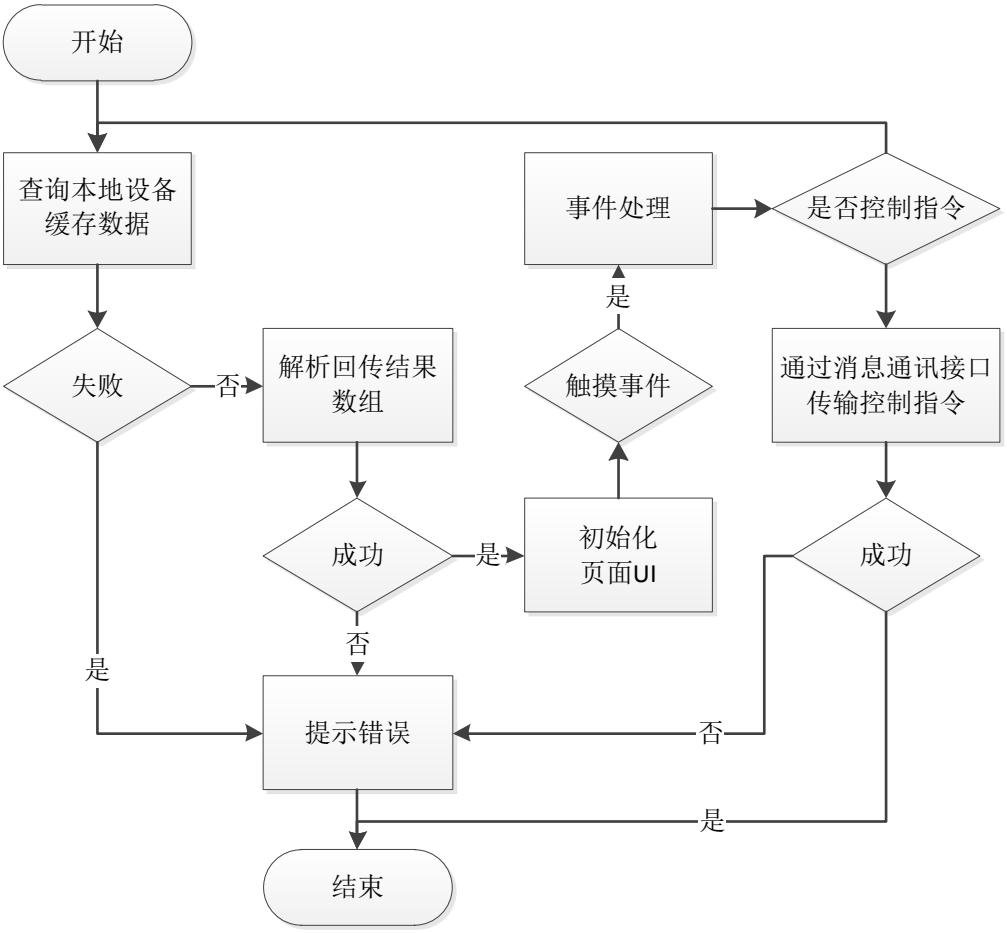

图 4-6 设备控制模块流程

4.4.1 设备控制 UI 生成实现

设备控制 UI 生成的实现主要基于 Javascript。通过 jQuery Mobile 来控制 Webview 页面中的 HTML 元素的渲染、事件处理。在此功能的实现上，利用了 jQuery Mobile 的分

页功能，并利用 Javascript 层调用 DataCache 插件来更新页面，比如 这样的 HTML 语句，规定了一个显示页面，而 id 命名规则，则用“page_[页面层级数]”这样的规则定义，在查询缓存数据进行页面更新时，只需要将对应页面的 HTML 元素进行更新（jQuery 里需要使用“$('body').trigger('create');”重构页面），就可以达到更换内容的目的。

在通过 jQuery Mobile 进行页面内容更新时，将根据 Javascript 逻辑模块的模板、DataCache 读取的数据，生成每一个具体的 HTML<button>元素，并在对应的元素中写入 onClick 等事件处理的 Javascript 函数。最终将渲染工作交由 Webview 的浏览器内核去处理。

4.4.2 设备控制指令通信

在用户交互点击控制 UI 元素时，将通过 Javascript 接口进入 Javascript 处理逻辑，而设备控制指令的通信，可以通过 HTTP 形式的请求，直接采用 Ajax 来发送 HTTP 的 Post 消息，但是这需要有 HTTP 服务器来接收处理，本文的实现将通过交由消息推送模块插件去处理，这种方式是直接通过 TCP 长连接经过服务器中转的形式送达家庭主控控制端。

在通过 PushPlugin 传输控制指令消息时，移动互联网络的不稳定将导致整个执行周期长短的不确定性。本文在控制指令传输的实现上，使用了 Cordova 的异步回调，在 Javascript 层将等待 TransService 模块的通信线程接收到控制指令回包后，再进行异步回调函数的执行，最终反映到页面显示上。

4.5 本章小结

本章主要从三大 Cordova 插件扩展模块以及在 Javascript 层封装的设备控制模块来说明本文研发的智能家居移动应用的实现，基于上一章节的设计，详细描述了本地 Sqlite 缓存模块 DataCache Plugin 的实现、视频监控模块 Monitor Plugin 的实现、消息推送模块 Push Plugin 的实现，以及基于 jQuery Mobile、Javascript 封装的设备控制模块的实现。

第五章 测试与分析

5.1 测试场景与环境

5.1.1 测试应用场景

如今移动互联网的日益发展，家电设备智能化的水平提高，许多家电设备厂商也正在与互联网公司合作，想在智能家居领域探索出发展之道，而本文基于此背景，分析了智能家居移动终端应用的主流开发方法，并尝试使用 Apache Cordova 框架来研发跨平台的智能家居移动应用，前面几个章节说明了跨平台智能家居终端应用研发的分析、设计到实现等方面，而本章将对该应用的进行测试分析，测试的场景主要为使用不同平台的移动设备，通过本文研发的基于 Cordova 的智能家居移动终端应用，来进行家电设备的远程控制、远程视频监控以及警报消息推送等应用场景测试。其中家电设备为定制的 433MHz 的智能化产品，包含电灯泡、压力锅、门磁警报传感器等，而家庭主控为一个运行 Linux 操作系统的工控机，其可接入互联网，同时可以通过其进行控制同一家庭网络内的所有家电设备；网络摄像头为基于 TCP/IP 网络接入的远程监控摄像头。

5.1.2 测试硬件环境

本文在测试基于 Apache Cordova 的智能家居移动应用时，根据上一小节提及的应用场景，对测试硬件环境进行了部署，主要包括如下硬件环境：

表 5-1 硬件环境

硬件设备	说明
智能家电设备	包括电灯泡、压力锅、门磁传感器等，它们通过 433MHz 与 Wifi 节点板进行通讯
Wifi 节点板	用于连接家庭主控工控机与设备电器间的电子电路板，支持 TCP/IP 网络通信与 433MHz 通信。
网络摄像头	海康威视网络摄像头，支持 TCP/IP 网络连接，设定了访问端口为 5000
Android 智能手机	魅族 MX4 型号智能手机，CPU 为 MTK 的 MT6596，A17 2.2GHz x 4 + A7 1.7GHz x 4 (8 核)处理器，运行内存 2GB RAM

表 5-1 硬件环境（续）

硬件设备	说明
苹果 Macbook Pro 笔记本电脑	CPU 为 intel i7 2.2GHz，运行内存 16GB RAM，其中 Mac OS 系统提供 iOS 模拟器测试环境
TPlink 无线路由器	使用其组建局域网
阿里云服务器	CPU 单核，运行内存 1GB RAM，带宽 1Mbps，用于运行服务器程序

5.1.3 测试软件环境

本文的测试软件环境描述如下：

表 5-2 测试软件环境

系统	说明
Android	版本 4.4.2
iOS 模拟器	iOS Simulator Version 8.0 (550.1)
阿里云服务器	操作系统： OpenSUSE 13.1 64 位，Java 版本：version "1.7.0_71"

5.2 测试与分析

5.2.1 本地 Sqlite 缓存模块的测试与分析

本模块主要负责在 Javascript 层调用并将结果返回至 Javascript 层

表 5-3 Sqlite 缓存模块测试用例

编号	测试用例	测试方法
1	Raw 数据复制	使用 rawDataCopy 来复制来自应用程序中的资源文件到运行环境的沙盒中。
2	js 调用读取文件	使用 DataCache 插件的 read 接口，读取文件指定位置、指定字节的内容。
3	js 调用写入文件	使用 append 或者 write 接口写入指定内容到运行环境的沙盒中的文件中。

表 5-3 Sqlite 缓存模块测试用例（续）

编号	测试用例	测试方法
4	js 调用 Sqlite 查询	调用 query 查询家电控制的 Sqlite 数据库文件，读取指定表格的筛选内容。
5	js 调用 Sqlite 插入	调用 insert 插入指定表的指定值。
6	js 调用 Sqlite 更新	调用 update 更高指定表的指定字段。
7	js 调用 Sqlite 操作	调用 execSql 删除表操作。

本文在测试如上用例时，在 Javascript 层编写了特定的测试用例，并根据用例方法，调用相应接口，测试结果如下表。

表 5-4 Sqlite 缓存模块测试结果

编号	测试用例	测试结果	是否通过
1	Raw 数据复制	使用 rawDataCopy 复制后，可以使用 sqlite 接口读取出沙盒中指定路径文件的内容，说明已经复制成功。	是
2	js 调用读取文件	读取文件指定位置、指定字节的内容后，可经过 Javascript 回调函数将对应内容显示出来无误。	是
3	js 调用写入文件	写入指定内容到运行环境的沙盒中的文件中后，再用读取接口读取出已经写入的内容，经由 Javascript 显示无误。	是
4	js 调用 Sqlite 查询	经由 query 接口读取出家电控制设备并通过 jQuery Mobile 显示到页面，显示无误	是
5	js 调用 Sqlite 插入	通过 Javascript 接口写入某设备加入到 Sqlite 文件的 Device 表中，再经过查询读取出来，结果一致。	是
6	js 调用 Sqlite 更新	经过 update 数据库文件中 Device 表的指定字段，经 query 读取出来，结果一致	是
7	js 调用 Sqlite 操作	调用 execSql 删除表操作，删除后经 query 查询，结果为不存在表格，查询失败。	是

经过以上测试用例，说明了 DataCache 插件模块的功能完成情况良好。其中模块与 Javascript 层的调用、回调均能够准确地完成工作。

5.2.2 视频监控模块的测试与分析

视频监控模块主要负责跳转至指定的视频监控页面，通过第三方的 SDK 进行网络摄像头的视频流获取以及视频流解析工作。本模块的主要测试如下：

表 5-5 视频监控模块测试用例

编号	测试用例	测试方法
1	视频设备管理	使用 DataCache 插件读取出 Sqlite 文件中的 Monitor 管理数据，并通过 UI 显示。
2	调用 Monitor Plugin 进行监控	点击指定摄像头进行视频监控跳转。
3	视频监控页面 UI 元素初始化	进入视频监控页面，对 UI 元素初始化包括登陆等信息的填入。
4	视频监控页面登陆	点击视频监控 UI 按钮，利用第三方 SDK 进行视频监控设备登陆。
5	视频监控实时预览	使用第三方视频 SDK 进行实时视频预览。
6	视频监控页面返回至 Webview	点击返回，跳转至 Webview 主页面

在测试视频监控模块时，在 Android 真机与 iOS 模拟器上进行用例测试，其所测试的用例均一致通过 Javascript 调用，跳转至视频监控页面，结果如下：

表 5-6 视频监控模块测试结果

编号	测试用例	测试结果	是否通过
1	视频设备管理	可以正确地显示通过 DataCache 读取的摄像头信息。	是
2	调用 Monitor Plugin 进行监控	正确跳转至摄像头的监控页面，交由本地代码执行逻辑。	是
3	视频监控页面 UI 元素初始化	可以看到在视频监控页面正确初始化了 UI 元素，包括登陆、地址、端口等 UI。	是
4	视频监控页面登陆	可以经由第三方 SDK 接入，正确调用登录接口，进行视频设备登陆后，可以预览。	是

表 5-6 视频监控模块测试结果（续）

编号	测试用例	测试结果	是否通过
5	视频监控实时预览	可以通过页面的 View 进行实时视频预览，网络视频流获取正常，流解码正常。	是
6	视频监控页面返回至 Webview	返回至主界面 Webview 正常，继续点击预览正常。	是

5.2.3 消息推送模块的测试与分析

本模块主要提供了与服务端的通信、消息接收回调处理等应用本地逻辑功能，供上层 Javascript 调用接口使用，主要测试用例如下：

表 5-7 消息推送模块测试用例

编号	测试用例	测试方法
1	TCP 长连接建立、登陆	在 Webview 主页面利用 Javascript 进行接口调用，参数传递至 TransService 组件进行长连接发起、并进行登陆动作。
2	心跳包发送、长连接维护	长时间监控服务端的心跳包接收情况。
3	发送固定类型的消息	发送应用消息至服务端处理。
4	消息接收的 Javascript 层回调	服务端推送消息至客户端，Javascript 进行回调处理，显示消息。
5	控制消息发送	通过 js 接口，将 Sqlite 读取的设备类型信息等，发送至服务端。
6	连接重连	断开网络，再开启网络。查看连接情况
7	连接断开	断开连接操作。

测试结果如下：

表 5-8 消息推送模块测试结果

编号	测试用例	测试结果	是否通过
1	TCP 长连接建立、登陆	通过查看服务器记录，已经建立连接并通过验证进行登陆，正常接收心跳。	是
2	心跳包发送、长连接维护	通过长时间未操作后，观察服务端扔有心跳接受，并可以推送至客户端。	是

表 5-8 消息推送模块测试结果（续）

编号	测试用例	测试结果	是否通过
3	发送固定类型的消息	服务端正确接收来自客户端的消息。	是
4	消息接收的 Javascript 层回调	经由服务端推送的消息，由 Javascript 层正确处理并显示。	是
5	控制消息发送	可以通过发送的控制消息，进行家电设备的远程控制。	是
6	连接重连	连接在断开后，重新接入服务器，正常登陆。	是
7	连接断开	正常断开连接，服务器断开后进行了清理工作。	是

5.2.4 总体的测试与分析

前面几小节主要针对了各模块的功能进行用例的测试与分析，下文将描述功能上的总体测试，基于前文所述的应用场景，主要包括了主页面功能、设备控制、视频监控等操作，这些操作都包含了上述关键模块的功能集成，比如设备控制的操作中就包含了 UI 生成、读取 Sqlite 缓存、通过 jQuery Mobile 显示、经由消息推送模块发送消息至服务端等操作。下面做一个总体的测试说明：

图 5-1 Android 与 iOS 的设备控制页面

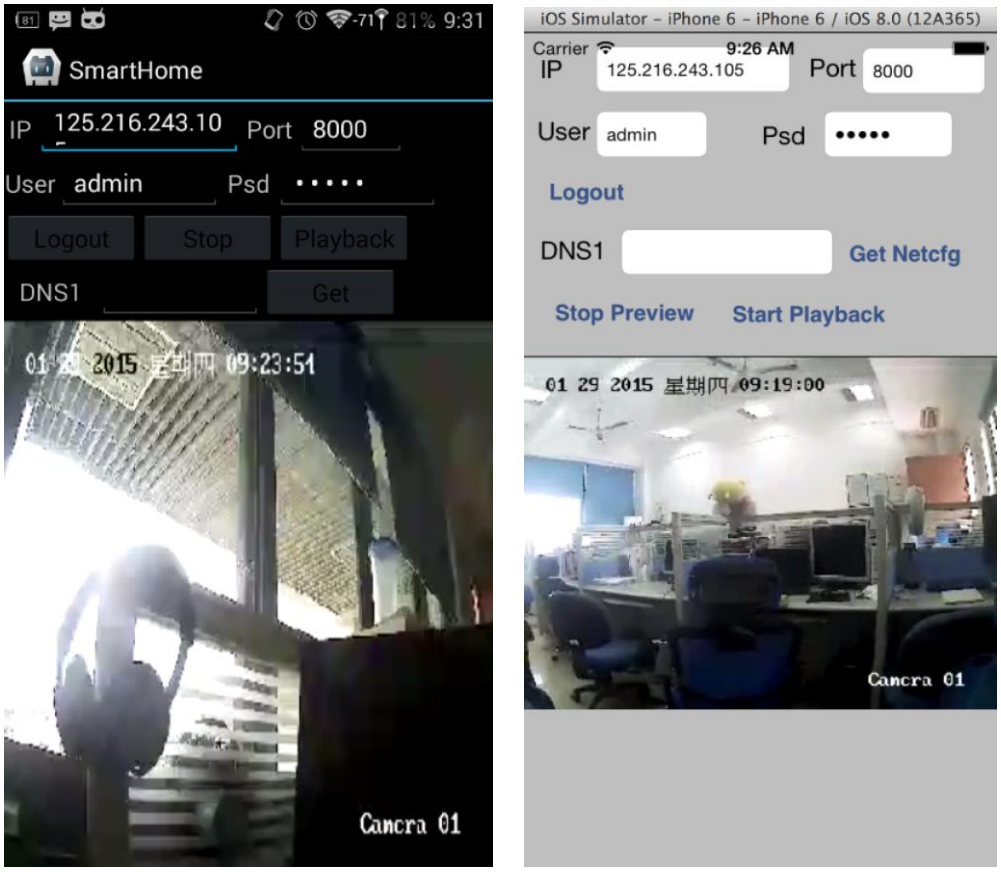

图 5-2 视频监控模块的 Android 与 iOS 端

图 5-3 阿里云服务器中基于 MINA 的程序的运行状况

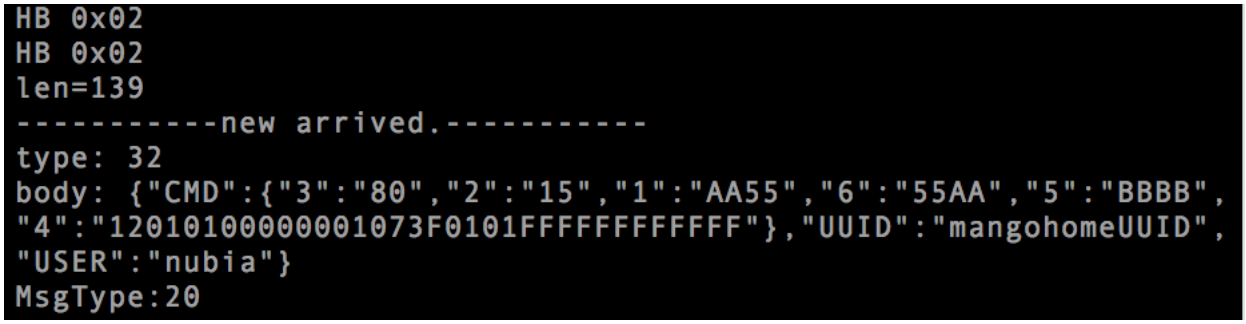

图 5-4 心跳的接收与控制消息的接收情况显示

本文基于 Cordova 框架，进行模块扩展，并基于此构建智能家居移动应用，相较于传统的方法，主要优势对比如下：

表 5-9 构建方法优势比较

项目	本文构建方法	传统构建方法
实现功能	基于扩展的模块来实现 Native 功能，具备完整性	完整功能
运行性能	初次启动时间较长，在目前主流设备上操作基本流畅，	启动速度快，较流畅的体验
平台扩展	如果新平台的加入，只需要进行 Cordova 模块适配，js 层代码公用，代价较小	新平台加入需要全新移植，代价非常大
可维护性	在逻辑的变更上，只需要更改 Javascript 层的逻辑代码，代价小	需要对每个平台进行同样逻辑的维护，工作量较大
开发成本	Cordova 模块适配与 Web 技术结合，上层逻辑跨平台，缩减成本	对各平台进行独立应用开发，包括所有逻辑实现，成本较高

5.3 本章小结

本章主要介绍了针对本文实现的 Apache Cordova 各扩展模块，包括 Sqlite 缓存模块、视频监控模块、消息推送模块进行了测试与分析工作。最后对总体进行了操作与测试工作，涵盖了各模块间的功能协作，通过测试结果表明，各模块均能很好地完成相应的功能。

结论与进一步研究

移动互联网与智能家居的兴起，让更多厂商对智能家居的发展投入了更大的兴趣，目前移动互联网平台的多样性，如果在每一个独立的平台上做独立的开发，不仅会提高了开发成本以及开发周期，同时会使后期的维护与变更更加复杂困难；如果选择使用单纯的 Web 的开发技术，又会受到许多功能的限制，不能够正常使用设备的许多本地功能，另一方面将给 Web 的服务器带来较大的访问压力，以及给移动设备带来较大的网络流量消耗。故本文寻求一种能够缩减开发、维护成本的方法来完成智能家居移动终端应用的研发。

首先，本文分析了目前大多数智能家居移动应用的开发模式，针对跨平台应用开发做了调研工作，在跨平台的开发上，有多款移动应用开发框架，而比较了目前主流的跨平台移动应用开发的方法后，针对智能家居移动应用所需的功能特性需要，最终选取了 Apache Cordova 来进行跨平台移动应用开发。Apache Cordova 相对于其他框架来说，提供了更多平台的支持，该框架支持标准的 HTML、CSS、Javascript 三大 Web 技术标准，使得更易于开发，同时本文分析了 Apache Cordova 的性能，其已经能够让用户体验达到较高水平。但是 Apache Cordova 框架官方只提供了一些基本的移动设备本地功能接口，需要满足智能家居移动应用的功能需求，需要在 Apache Cordova 框架上做相应的扩展，如本地缓存文件的使用、视频监控功能、消息推送通信功能等。

其次，本文针对 Cordova 框架进行了较详细的分析，从 Apache Cordova 的整体框架的理解、开发模式，到其中主要模块以及模块之间的联系。Cordova 框架主要基于 Javascript 与本地代码通信这个切入点，桥接了 Web 与本地代码两个世界，应用程序在 Webview 之上，使用 Javascript 便可以与本地系统功能相互通信。本文详细地分析了 Cordova 的运行原理，包括 Webview 如何与 Native 通信，并针对智能家居移动应用所需的模块扩展作了分析，同时分析了移动应用推送技术，为后面的基于 Cordova 来构建智能家居移动应用做技术铺垫。

本文根据对智能家居移动终端应用的分析，以及对 Apache Cordova 框架的深入分析，提出了一套设计方案，包括对 Apache Cordova 框架的部分复杂功能的扩展，在 iOS 与 Android 操作系统上做相应的适配，并利用 Web 技术构建应用。本文从总体设计到局部功能模块做了详细说明，他们包括总体的框架设计、本地 Sqlite 缓存接口设计、视频监控模块的设计、消息推送模块的设计以及设备控制模块的设计。针对这些模块功能，本

文叙述了在这些功能模块中，在不同平台上的适配工作以及一些接口说明。

本文根据对智能家居移动应用功能需求而设计的 Cordova 扩展，进行了各模块的具体实现。在通过各模块的本地适配工作后，通过 Javascript 来调用各模块的功能，并在 Webview 上，利用 jQuery Mobile 结合 HTML、CSS 等技术，构建移动设备上的智能家居应用，同时利用 Apache Mina 框架搭建了基于 TCP 长连接的消息通讯程序，在视频监控模块上，进行了第三方视频监控 SDK 的接入，而应用中 Javascript 层的公用主体逻辑，可以根据封装的接口调用这些模块来实现对应功能。

本文最后根据测试用例，做了每个功能模块的测试与分析，并搭建了整体的测试环境部署，包括智能家居移动应用进行远程监控所需的硬件、软件环境，最后在 iOS 平台与 Android 平台上进行了总体功能测试，结果表明 Cordova 框架在应对智能家居移动应用的开发功能需求是可以满足的，通过该框架的开发同时也抽取了大部分逻辑代码放到了 Web 技术层，而复杂的功能通过适配插件的方式，加入到应用中来。总的来说，比起单独在各平台上做开发，要更快捷。而将来如果加入更多的移动平台，只需要在此基础上，进行其他平台的插件扩展，既可以重复利用 Web 层面的公共代码接口，节省了开发时间。

本文的设计、实现方案，只是在当前最流行的两个移动平台上（iOS 平台与 Android 平台）做了相应的适配工作，接下来还可以在更多平台的加入、UI 界面的优化、性能的调优等相关工作上做进一步的研究，可以总结为如下：

1. 更多平台的加入；本文在 iOS 与 Android 两个平台做了 Cordova 的扩展设计与实现，若有需要，可以将来把 Windows Mobile 等其他移动操作系统进行适配，移植 Web 层的主体逻辑；
2. UI 界面的优化；本文在 Webview 层使用 jQuery Mobile 来构建主体操作页面的可视化元素，将来可以利用其余 Javascript 框架来美化 Webview 主体页面；
3. 更多视频监控 SDK 的接入；如果在需要其他第三方视频监控 SDK 接入，则可以根据视频监控模块 Monitor Plugin 预留的相关接口，进行更多厂商的 SDK 接入；
4. 性能的调优；本文基于 Webview 与 Cordova 框架进行扩展设计，将来可以在 Webview 上做性能优化工作，使用户交互操作以及 Native 与 Javascript 层通信拥有更好的效率。

参考文献

[1] 百度百科. Android 系统 [EB/OL].http://baike.baidu.com/view/4971681.htm

[2] 维基百科.Android[EB/OL].http://zh.wikipedia.org/wiki/IOS

[3] Gong, Qingchao; Li, Guangming; Pang, Yong; Design and implementation of smart home system based on ZigBee technology[J].International Journal of Smart Home, 2014,8:143-156

[4] Liao, GaoHua ; Zhu, JieBin. Smart home system network architecture and implementation[C].Communications in Computer and Information Science, 2011,235(5):498-503

[5] Ningqing, Liu ; Haiyang, Yan; Chunmeng, Guan. Design and implementation of a smart home control system[C]. IMCCC 2013, 1535-1538

[6] Luo, Wei ; Li, Wei; Li, Xin. Design and implement on smart home system. ISDEA 2013,2013, p229-231

[7] 王朝华.基于 Android 的智能家居系统的研究与实现[D].广东工业大学,2012.

[8] 维基百科.Windows Phone[EB/OL].http://zh.wikipedia.org/wiki/Windows_Phone

[9] 宁义双. 基于 HTML5 的移动终端应用中间件平台的研究与设计[D]. 北京工业大学, 2013

[10] 刘贵勇;Android 系统浏览器对 WML 标签语言支持的实现[D].南京理工大学,2012

[11] Hui, Ng Moon ; Chieng, Liu Ban; Ting, Wen Yin; Mohamed, Hasimah Hj; Hj Mohd Arshad, Muhammad Rafie. Cross-platform mobile applications for android and iOS[C].WMNC 2013, 2013.

[12]Ciman, Matteo ; Gaggi, Ombretta; Gonzo, Nicola. Cross-platform mobile development: A study on apps with animations[C].SAC 2014, 2014, p 757-759

[13]Shehab, Mohamed ; Aljarrah, Abeer. Reducing attack surface on cordova-based hybrid mobile apps[C]. Part of SPLASH 2014, p1-8

[14]Heitkötter, Henning ; Hanschke, Sebastian; Majchrzak, Tim A. Comparing cross-platform development approaches for mobile applications[C].WEBIST 2012,2012,p 299-311

[15] Corona [EB/OL] https://coronalabs.com/

[16] Shi, Qiao. Development of smart home environment based on internet of things technologies[J].Computer Modelling and New Technologies,2014,18(12):416-421

[17] Wang, Yanmin. The internet of things smart home system design based on ZigBee/GPRS technology[J]. Applied Mechanics and Materials, 2013, v263-266, nPART 1, p2849-2852

[18] Zhao, Xue-Feng. The application of bluetooth in the control system of the smart home with internet of things[J]. Advanced Materials Research, 2013, v712-715, p2753-2756

[19] Yu, Congmin ; Mao, Mingyi; Jiang, Yuanheng. Design and realization of smart home system based on internet of things[J]. Information Technology Journal, 2013, 12(13):2519-2525

[20] Andergassen, Monika; Guerra, Victor; Ledermüller, Karl; Neumann, Gustaf. Development of a browser-based mobile audience response system for large classrooms[J].International Journal of Mobile and Blended Learning, 2013,5(1):58-76

[21] Chen, Donghua; Zhu, Xiaomin. An integration framework for HTML5-based mobile applications[C].LISS 2012 - Proceedings of 2nd International Conference on Logistics, Informatics and Service Science, p 501-506, 2013

[22] Yi, Xiaolin ; Ning, Yishuang. The development of a mobile terminal middleware platform based on HTML5[J]. Journal of Software, 2013, 8(5):1186-1193

[23] Bouras, Christos ; Papazois, Andreas; Stasinos, Nikolaos. A framework for cross-platform mobile web applications using HTML5[C]. Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014, p420-424

[24] Li, Zhi Min ; Luo, De Han; Fan, Dan Jun; Wang, Chao. Design and realization of wireless LAN in smart home system based on 433MHZ communication[J]. Applied Mechanics and Materials, v 336-338, p 1645-1649, 2013

[25] Ding, Nan ; Xu, Fang Qin. Smart home application based on ZigBee protocol[J]. Applied Mechanics and Materials, v 513-517, p 2449-2452, 2014

[26] Yiqi, Wang ; Lili, He; Chengquan, Hu; Yan, Gao; Zhangwei, Zhu. A Zigbee-based smart home monitoring system[C]. Proceedings - 2014 5th International Conference on Intelligent Systems Design and Engineering Applications, ISDEA 2014, p 114-117, 2014

[27] Tian, Liguo ; Li, Meng; Chen, Zhiliang; Guan, Beibei. Design of smart home control terminal based on ZigBee and electronic technology[J]. Lecture Notes in Electrical Engineering, v 177 LNEE, n VOL. 2, p 339-344, 2012

[28] Shang, Hui-Liang ; Xu, Ren-Mei; Yuan, Jun-Kang. A Smart home system based on

ZigBee and iOS software[C].ICPADS, p 940-944, 2012

[29] Lien, Chia-Hung ; Chen, Po-Tsun; Bai, Ying-Wen; Lin, Ming-Bo. Monitoring system with moving object detection based on MSN messenger[C]. Conference Record - IEEE Instrumentation and Measurement Technology Conference, p 229-234, 2008

[30]Kumar, Shiu ; Lee, Seong Ro. Android based smart home system with control via Bluetooth and internet connectivity[C]. ISCE,2014

[31]Puder, Arno. An XML-based cross-language framework[J]. Lecture Notes in Computer Science, v 3762 LNCS, p 20-21, 2005

[32]rhomobile[EB/OL].http://rhomobile.com/

[33]维基百科. DragonRAD [EB/OL].http://en.wikipedia.org/wiki/DragonRAD

[34]维基百科. MoSync [EB/OL].http://en.wikipedia.org/wiki/MoSync

攻读硕士学位期间取得的研究成果

一、已发表（包括已接受待发表）的论文，以及已投稿、或已成文打算投稿、或拟成文投稿的论文情况：

序号	作者（全体作者，按顺序排列）	题　　目	发表或投稿刊物名称、级别	发表的卷期、年月、页码	相当于学位论文的哪一部分（章、节）	被索引收录情况
1	刘发贵 肖敏 蓝智宏	AN IMPROVEMENT FOR PROCESS MANAGEMENT IN ANDROID VEHICLE TERMINAL	ADVANCED MATERIALS RESEARCH	2013，V791, p1730-1734	-	EI 索引源
2	刘发贵 肖敏 冯炜君	Design of Cordova-based Message Push Module for Cross-platform Smart Home Application	Journal of Machine Learning and Signal Processing (ISSN 2409-5494)	拟投稿	第三章第四节	EI 索引源

二、与学位内容相关的其它成果（包括专利、著作、获奖项目等）

1. 参与的项目

(1) 无线物联家居系统与 RFID 路由器。2012 年广东省信息产业厅信息服务业专项。项目编号：GDEID2012IS054。（2010.05-2014.06）

(2) 康宝家庭信息系统。广东康宝电器有限公司委托开发项目。项目编号：x2jsD813104。（2013.01-2014.12）

(3) 家庭信息平台的产业化推广。广东省省级科技计划项目。项目编号：2013B090200055。（2013.09-2015.09）

2. 发明专利

(1) 《一种用于车载移动互联的安全管理方法和系统》 专利受理，申请号2012103097211，申请人：刘发贵；蓝智宏；肖敏。

(2) 《一种 Android 车载系统进程管理的改进方法》专利受理，申请号2013103159316，申请人：刘发贵；肖敏；蓝智宏。

致谢

研究生三年生活即将结束，这段意义非凡的校园生活将永远铭记于心。感谢华南理工大学给予我研究生进修机会、优越的教学条件和雄厚的师资力量，感谢小谷围岛提供的优美校园生活环境，感谢研究生课程中辛勤付出的老师们，感谢这几年一起走过的身边的同学和朋友。

特别感谢的是我的导师刘发贵教授，是她引导我踏入了研究生生涯，让我在学术研讨与各类工程项目中获益匪浅，同时教导我在工作、为人方面的道理，在我处于困难中时给予我关心、支持与信任，非常感谢刘老师！

感谢我的室友郭伟、林锦标、罗松超，三年来，我们相互勉励，相互关心，一起生活，一起学习，一起嗨皮。

感谢 EMOS 实验室的所有人，尤其是给予我莫大支持的师兄：李宝韩、蓝智宏、邢晓勇、缪汉威。

感谢我生命中非常重要的人：Loo，你改变了我。

最后感谢我的父母赐予我生命，并把我抚养长大，感谢我的亲友们一直对我的支持与关心，谢谢！

肖敏

2015 年 03 月 25 日

Ⅳ－2答辩委员会对论文的评定意见

肖敏同学的硕士论文“基于 Apache Cordova 的跨平台智能家居终端应用研发”有较好的理论意义和应用价值。

论文参照了较多的相关文献资料，依据智能家居移动终端的应用的功能需求，设计并实现了 Android 及 IOS 平台的 Apache Cordova 框架扩展模块，包括本地 Sqlite 数据缓存、家电控制、视频监控扩展以及消息推送等模块。最后，通过测试用例，对每个功能模块进行了分析。

作者对文献的调研、数据的收集较充分，论文资料丰富，论点正确，结论合理，实验数据真实可信。反映作者具有较强的理论基础、较好的科研和工程能力。

论文结构清晰，图文表达规范，达到硕士学位论文水平。答辩过程讲述清楚，回答问题正确。经答辨委员会无记名投票，同意该同学通过硕士论文答辩，同意授予硕士学位。

论文答辩日期：2015年04月25日

答辩委员会委员共 5 人，到会委员 5 人

表决票数：优秀（2）票；良好（3）票；及格（ ）票；不及格（ ）票

表决结果（打“√”）：优秀（ ）；良好（√）；及格（ ）；不及格（ ）

决议：同意授予硕士学位（√）　不同意授予硕士学位（ ）

答辩委员会成员签名			
	张圣明（主席）	刘好	
	高英	董子斌	陈小路

智能家居移动终端应用的控制、监控及警报等功能，设计并实现基于跨移动平台开发框架的易维护、可扩展的智能家居移动终端应用程序。

1.1.2 智能家居移动终端应用的发展现状

智能家居的概念提出之后，国内外很多厂商纷纷提出了各自的解决方案，他们大多数都基于 433MHz、Zigbee、Wifi 等无线通线技术实现[24,25,26]，其控制终端实现方式为：1. 基于 Arm 的嵌入式应用系统[27]，这种控制设备上运行的应用程序，通常比较稳定，定制性较强而可移植性、可扩展性较差，同时需要开发人员话费大量精力去维护；2. 基于移动设备操作系统的应用程序，这种方式通过移动设备操作系统（如 Android、iOS、Windows Phone 等）上的独立应用、或者第三方应用的接口[28]，进行智能家电的控制，有基于 Android 的 MSN（一种即使通讯软件）实现智能家居控制功能的方案[29]，有独立的原生应用通过互联网接入来实现家电远程控制的方案[30]，也有一些通过移动设备 APP 调用手机红外传感器来实现基于红外线的家电设备控制功能。

总结上述的主流智能家居移动终端应用的实现方式，我们可以看出他们都是基于独立的平台完成独立的应用开发，来满足家电设备控制的需求，这也符合大多数的移动设备应用程序的特点，根据移动操作系统的 API 进行应用软件的开发，随着操作系统本身的发展变化而需要做出调整，当今最流行、最普及的移动操作系统莫过于 Android 与 iOS，Android 自发布开始就一直在快速发展壮大，同时带来的 API 变化也使程序开发需要不断地去适应，如果再加上闭源的 iOS 操作系统变化所带来的变更，则维护成本会比较高。总之，我们可以通过跨平台的开发方式，抽取出公共的功能模块，来降低应用的开发成本。

1.2 国内外的跨平台移动应用框架研究状况

1.2.1 跨平台移动应用开发工具的研究现状

1.2.1.1 主流跨平台工具

最近几年随着移动互联网的普及、移动操作系统的快速发展，一些跨平台的开发工具、框架也随之兴起，能够支持平台变得更多，而较流行的有：Apache Cordova、Titanium、XMLVM[31]、Rhodes Rhomobile[32]、DragonRad[33]、MOSYNC[34]等，他们在实现原理上不大一样，所以带来的支持功能、运行效果、执行效能方面会有一些不同。

Titanium 是一款免费、开源的应用程序快速开发平台，其在 Apache 许可下发布，开发者可以通过它，使用 HTML、Javascript 和 CSS 进行桌面应用程序和移动应用程序的

开发，Titanium 还支持使用 Python、Ruby 和 PHP 语言，其所支持的 GUI 开发界面 Titanium Developer Dashboard 也使开发更为便捷。其实现原理主要是将开发者编写脚本语言解析、编译成本地的应用程序，而带来的缺点也显而易见，需要开发者去学习、适应 Titanium 提供的脚本语言格式及其定制的开发接口。

XMLVM 项目的是提供一种灵活的可扩展的交叉编译器工具链。XMLVM 基于 Sun 的 Java 虚拟机(Sun Microsystem's virtual machine)和微软的公共语言运行时(Microsoft's Common Language Runtime)上的字节码而非源码进行交叉编译。但是该项目仍处在研究阶段，并不广泛使用，故不作研究。

Rhodes Rhomobile 是一个可基于 Model View Controller (MVC)架构下开发移动应用的面向企业的开发工具，开发者可以使用 HTML 编写 UI、Ruby 编写逻辑，然后被编译为 Ruby 字节码，最后通过 Ruby 虚拟机翻译成平台执行代码。Rhodes 主要面向企业应用开发，但是其本地应用的功能扩展支持不够，而通过中间翻译层，难免会带来一些兼容性问题。

DragonRad 也是一个基于中间代码翻译的跨平台实现方案，主要面向企业应用为主。其原名为 Seregon，该公司开始是为 windows 移动平台和黑莓设备提供软件升级服务的。DragonRad 拥有一个基于 WYSIWYG 语言的可拖放 GUI 开发环境 DragonRad Designer，可以利用 Lua 脚本语言进行逻辑、业务模块的构建，同时对不同的数据库平台支持较好，比较适合企业移动应用，不过 DragonRad 对本地功能的支持不够丰富、稳定，不能够很好地满足本地功能要求较高的应用需求。

MoSync 也是一个多移动平台的跨平台开发工具，其开始时定位于 JavaME 设备，后发展成可基于 C++、HTML、JavaScript 为 iOS、Android、Windows Phone、Symbian 等移动平台的开发工具，MoSync 主要是是用了定制的 C++编译器来完成代码的翻译，构建基于 MoSync Intermediate Language 的代码，再转换成适用于平台运行时封装的字节码。其对 C++要求较高，也并不适合移动应用的快速开发。

Apache Cordova 是一个开源（在 Apache License, Version 2.0 下）的移动应用开发框架。通过它我们可以使用标准的 Web 技术如 HTML5、CSS3 和 JavaScript 来进行跨平台的移动应用软件开发，其封装了各平台的本地设备相关 API，只需方便地使用标准的 JavaScript 接口来调用即可以调用各平台上的本地设备 API 功能，而对原生 JavaScript、HTML 和 CSS 的支持也让开发者更容易上手，能够构建基于 Web 技术的 UI 界面，使应用有良好的交互体验。正式因为 Apache Cordova 对标准的 Web 技术与本地 API 的良

好结合，加之具有较好的扩展性，故作为本文将重点进行研究的一个跨平台移动应用开发框架，结合其优势进行模块扩展，构建智能家居移动终端应用程序。

1.2.2 Apache Cordova 的研究状况

Apache Cordova 的前身是 PhoneGap，最初的 PhoneGap 在 2009 年的 iPhoneDevCamp 大会（旧金山）上提出，由 Brock Whitten、Rob Ellis 和 Andre Charland 在 iOS 操作系统上建立起 Web 开发语言和 Objective-C 语言（苹果 iOS 应用开发语言）之间的桥梁，从而开发者可以通过 HTML5、Javasc Apache Cordova 的前身是 PhoneGap，最初的 PhoneGap 在 2009 年的 iPhoneDevCamp 大会（旧金山）上提出，由 Brock Whitten、Rob Ellis 和 Andre Charland 在 iOS 操作系统上建立起 Web 开发语言和 Objective-C 语言（苹果 iOS 应用开发语言）之间的桥梁，从而开发者可以通过 HTML5、Javascript 和 CSS 等 Web 标准技术进行快速应用开发，实现一次编译到处执行。PhoneGap 于 2011 年 10 月 4 日被 Adobe 公司正式收购，后来 PhoneGap 的源代码贡献于 Apache 软件基金会，并更名为 Apache Cordova。因为 Apache Cordova 在 2009 年后才出现，而其官方开发团队也在不断地更新完善其对各平台的功能支持及稳定性，所以近几年国内外对 Apache Cordova 的研究处于不断地探索的阶段，也有一些商业公司会使用 Cordova 进行轻量级的应用程序开发。

目前 Apache Cordova 已支持绝大部分的主流移动操作系统 iOS、Android、Windows Phone、BlackBerry OS 10、Symbian 等，而刚发布不久的 Ubuntu Touch 操作系统也加入了支持行列，各平台的功能支持如表所示。但在部分平台的本地设备 API 的支持上仍有欠缺，一部分是因为移动操作系统本身暂时无法提供该功能 API、或者硬件上的功能空缺，另外一部分是因为一些商业化的因素，不过相较于其他跨移动平台开发框架来说，已经做得非常出色。

表 1-1 Apache Cordova 支持平台情况

功能	加速器	照相机	指南针	通讯簿	档案	定位	多媒体	网络	通知	储存
iPhone /iPhone 3G	是	是	无	是	是	是	是	是	是	是
iPhone 3GS and newer	是	是	是	是	是	是	是	是	是	是
Android 1.0 - 4.2	是	是	是	是	是	是	是	是	是	是

Windows Phone	是	是	是	是	是	是	是	是	是	是
BlackBerry 10 and PlayBook OS	是	是	是	是	是	是	是	是	是	是
4.6-4.7	无	无	无	无	无	是	无	是	是	无
5.0-6.0+	是	是	无	是	是	是	无	是	是	是
Bada	是	是	是	是	无	是	无	是	是	无
Symbian	是	是	无	是	无	是	无	是	是	是
webOS	是	是	是	无	无	是	无	是	是	是
Tizen	是	是	是	是	是	是	是	是	是	是
Ubuntu Touch	是	是	是	无	是	是	是	是	是	是
Firefox OS	是	是	是	是	无	是	无	是	是	是

1.2.3 Apache Cordova 应用与原生应用比较

如果移动应用开发者选择了 Apache Cordova 进行跨平台开发，那么首先会考虑到其性能是否能够达到或者接近原生应用程序的水平，毕竟 Apache Cordova 是基于 Web 技术来实现 UI 表现层的，其中增加了对 Web 语言进行解析、通过 Webview 进行渲染的性能消耗。如果对同一功能，通过比较 Apache Cordova 应用程序与原生应用程序的执行效率，则可以最直观地体现出他们带来的用户体验差距。

表 1-2 Apache Cordova 应用与原生应用针对相同功能的比较（Android 与 iOS 上）

应用类型	基于 Apache Cordova		Android 4.2 原生应用	iOS 7 原生应用
测试功能	列表浏览、对话框弹出		列表浏览、对话框弹出	列表浏览、对话框弹出
实现方式	HTML、jQuery Mobile、Cordova Dialog Plugin		Listview、Dialog 组件	TableView、UIAlertView
运行平台	魅族 MX4（Android 4.2）	iOS 7	魅族 MX4（Android 4.2）	iOS 7
应用大小	1.9MB	1.7MB	572KB	396KB
内存占用	40.2MB	38MB	26.4MB	31MB
载入速度	1.9 秒	1.4 秒	0.9 秒	0.7 秒
操作流畅度	流畅	流畅	非常流畅	非常流畅
对话框弹出响应	0.6 秒	0.6 秒	0. 3 秒	0.2 秒
Pause 和 Resume	0.3 秒	0.2 秒	0.2 秒	0.2 秒

表 1-2 Apache Cordova 应用与原生应用针对相同功能的比较（续）

绘制滞后感	略微滞后。 由于采用了 Webview 的形式，而界面均由 HTML 和 jQuery Mobile 编写，故在测试过程中，会出现整个 UI 框架结构的绘制变化，这是因为浏览器内核渲染的时候会随着页面元素组成的 Dom 树的构建而进行，但是速度非常迅速，肉眼几乎察觉不出来。		无。 由于采用了 Android 操作系统的原生 UI，这个渲染过程是非常快速的，肉眼完全察觉不出来。	无。 同 Android 一样，也是 iOS 系统的原生 UI。
高操作负载	Android 手机下进行大量用户触摸时间注入，会导致列表的更新变缓慢	iOS 下也会变缓慢，但没有 Android 的影响那么大	保持流畅。 这里因为 Android 的原生 UI 的 ListView 组件的重用设计优化	保持流畅
变更成本	在 UI 上变更，只需要更改 HTML 与 jQuery Mobile 的代码，即可以做到 Apache Cordova 应用在 Android 与 iOS 平台上同时产生效果，跨平台的优势得以体现		独立变更 Android 代码	独立变更 iOS 代码

通过最直观的单一功能操作比较，我们能够看出基于 Apache Cordova 的移动应用会在响应时间上略落后原生应用，但这正如上述提到的，Apache Cordova 的应用程序是通过 Webview 提供用户交互，并处理基于 Web 技术的代码段，经过浏览器内核解析并完成渲染动作，这个过程比起原生应用的直接 UI 渲染会更耗费时间，不过总体来看，除了载入速度会比较慢之外，整体的用户体验已经很接近原生应用程序。而对于智能家居移动终端应用来说，也是可以满足其 UI 交互体验的需求的。

1.2.4 Apache Cordova 的不足与分析

从当前 Apache Cordova 对各个移动平台支持的本地设备接口来看，虽然已经支持了很多本地设备的 API，但是如果要构建一个面向智能家居移动终端设备的应用程序，少不了对家电设备控制数据的本地缓存、第三方视频监控 SDK 的整合接入和安防警报类型消息推送等的支持，更具体地说，如果单纯用 Apache Cordova 对多移动设备平台的本地 API 封装，不能够满足如下几点：

1. 智能家居移动应用需要同步更新家电设备数据，并把设备数据进行本地缓存。若仅仅通过 Web 的方式获取家电数据，频繁的拉取操作会导致服务端的压力巨大，故需