广东外语外贸大学

本科生毕业论文（设计）

论文题目：基于条件触发的Android自动任务执行App

作者姓名：郭伟铖，王俊胜
作者学号：20111003828，20111003810
指导教师：马文华
年级专业：2011级计算机科学与技术
所在学院：思科信息学院

提交日期：2015年6月

摘要

在移动互联网的时代，android平台早已经普及开来。人类正在前所未有的享受着科技所带来的前所未有的便利。而随着各种移动设备以及物联网的普及，人类可以让软件实现更加智能的管理各种设备。比如在晚上休息时不想被人打扰，可以自动的屏蔽手机上Email、短信、电话的接受，而等到明天早上才自动提醒用户；又比如当你打开Instagram分享照片时，你可能想把所分享的照片同时上传到你的Dropbox同步盘上。

本文就基于条件触发的Android自动化任务执行App进行设计和研究。本文首先介绍分析了当前基于情景等条件触发的app研究现状。然后介绍可以触发任务的情景，例如获取到用户的位置就应该调用相关的位置服务（LBS），设计好相关的情景是这个系统成功的关键。然后在介绍运用的Android开发技术，
论文给出了Android应用Tasker的实例。该实例实现了几种常用的自动化情景，以及提供接口给用户自定义情景任务。

关键词：*Android*, 自动任务，*LBS*, 情景

Abstract

In the mobile Internet era, android platform already gaining in popularity. Mankind is enjoying unprecedented technology brought unprecedented convenience. With a variety of mobile devices and the popularity of things that humans can make software more intelligent management of various devices. For example, do not want to be disturbed during the night to rest, the phone can automatically shield Email, SMS, telephone acceptance, and wait until tomorrow morning to automatically alert users; another example, when you open Instagram to share photos, you might want to have shared uploading photos to your Dropbox sync disk.
In this paper, based on the conditions to trigger automated tasks Android App for the design and execution of research. This paper describes the current study analyzes the status quo scenario-based app trigger conditions. Then introduce scenarios can trigger tasks, such as access to the user's location should call the relevant location-based services (LBS), good design is the key to this scenario related to the system's success. Then introduce the use of Android development technology,
Paper gives an example of Tasker's Android application. This example implements several common automation scenarios, as well as providing an interface to a user-defined profile task.
Keywords: *Android*, *Automated Tasks*, *LBS*, *Context*

目录

第一章 引言

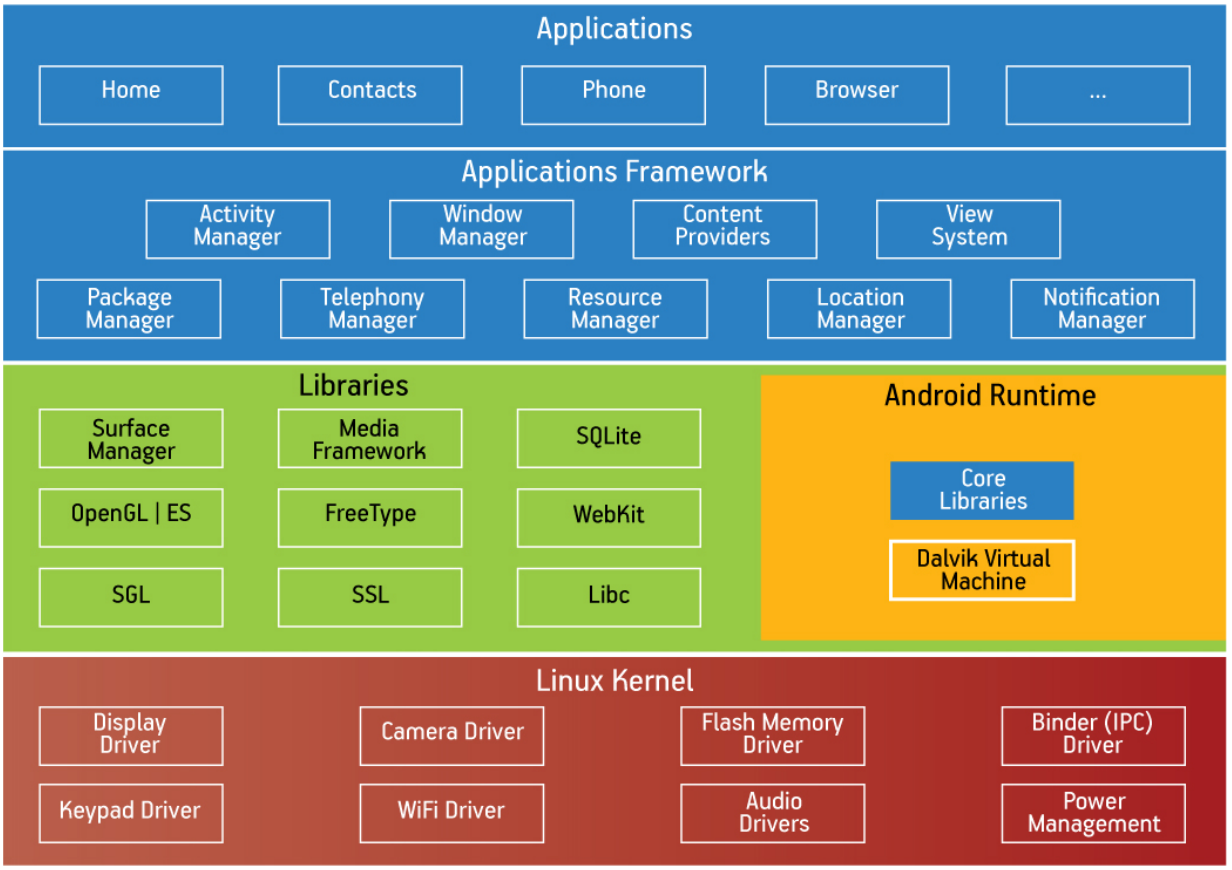
1.1 Android系统介绍

Android，是一个以Linux为基础的开放源代码移动设备操作系统，主要用于智能手机和平板电脑，由Google成立的Open Handset Alliance（OHA，开放手持设备联盟）持续领导与开发中。Android已发布的最新版本为Android 5.1(Lollipop)。

Android系统最初由安迪·鲁宾（Andy Rubin）等人开发制作，最初开发这个系统的目的是创建一个数码相机的先进操作系统；但是后来发现市场需求不够大，加上智能手机市场快速成长，于是Android被改造为一款面向智能手机的操作系统。于2005年8月被美国科技企业Google收购。2007年11月，Google与84家硬件制造商、软件开发商及电信营运商成立开放手持设备联盟来共同研发改良Android系统，随后，Google以Apache免费开放源代码许可证的授权方式，发布了Android的源代码，让生产商推出搭载Android的智能手机，Android操作系统后来更逐渐拓展到平板电脑及其他领域上。2010年末数据显示，仅正式推出两年的Android操作系统在市场占有率上已经超越称霸逾十年的诺基亚Symbian系统，成为全球第一大智能手机操作系统。

1.2 Android平台架构及特性

从宏观的角度来看，Android是一个开放的软件系统，它包含了众多的源代码。从下至上，Android系统分成4个层次：



1. **应用程序层**
   Android平台不仅仅是操作系统，也包含了许多应用程序，诸如SMS短信客户端程序、电话拨号程序、图片浏览器、Web浏览器等应用程序。这些应用程序都是用Java语言编写的，并且这些应用程序都是可以被开发人员开发的其他应用程序所替换，这点不同于其他手机操作系统固化在系统内部的系统软件，更加灵活和个性化。

2. **应用程序框架层**
   应用程序框架层是我们从事Android开发的基础，很多核心应用程序也是通过这一层来实现其核心功能的，该层简化了组件的重用，开发人员可以直接使用其提 供的组件来进行快速的应用程序开发，也可以通过继承而实现个性化的拓展。
   a. Activity Manager（活动管理器）
   管理各个应用程序生命周期以及通常的导航回退功能
   b. Window Manager（窗口管理器）
   管理所有的窗口程序
   c. Content Provider（内容提供器）
   使得不同应用程序之间存取或者分享数据
   d. View System（视图系统）
   构建应用程序的基本组件
   e. Notification Manager(通告管理器)
   使得应用程序可以在状态栏中显示自定义的提示信息
   f. Package Manager（包管理器）
   Android系统内的程序管理
   g. Telephony Manager(电话管理器)
   管理所有的移动设备功能
   h. Resource Manager（资源管理器）
   提供应用程序使用的各种非代码资源，如本地化字符串、图片、布局文件、颜色文件等
   i. Location Manager(位置管理器)
   提供位置服务
   j. XMPP Service（XMPP服务）
   提供Google Talk服务

3. **系统运行库层**
   从图中可以看出，系统运行库层可以分成两部分，分别是系统库和Android运行时，分别介绍如下：
   a. **系统库**
   系统库是应用程序框架的支撑，是连接应用程序框架层与Linux内核层的重要纽带。其主要分为如下几个：

   - Surface Manager：
     执行多个应用程序时候，负责管理显示与存取操作间的互动，另外也负责2D绘图与3D绘图进行显示合成。
   - Media Framework：
     多媒体库，基于PacketVideo OpenCore支持多种常用的音频、视频格式录制和回放，编码格式包括MPEG4、MP3、H.264、AAC、ARM。
   - SQLite:
     小型的关系型数据库引擎
   - OpenGLES：
     根据OpenGL ES 1.0 API标准实现的3D绘图函数库
   - FreeType：
     提供点阵字与向量字的描绘与显示
   - WebKit：
     一个浏览器的渲染引擎

- SGL：
  底层的2D图形渲染引擎
- SSL：
  在Andorid上通信过程中实现握手
- Libc：
  从BSD继承来的标准C系统函数库，专门为基于embedded linux的设备定制

b. **Android Runtime**
Android应用程序时采用Java语言编写，程序在Android运行时中执行，其运行时分为核心库和Dalvik虚拟机两部分。

- 核心库
  核心库提供了Java语言API中的大多数功能，同时也包含了Android的一些核心API,如android.os、android.net、android.media等等。
- Dalvik虚拟机
  Android程序不同于J2me程序，每个Android应用程序都有一个专有的进程，并且不是多个程序运行在一个虚拟机中，而是每个Android程序都有一个Dalivik虚拟机的实例，并在该实例中执行。Dalvik虚拟机是一种基于寄存器的Java虚拟机，而不是传统的基于栈的虚拟机，并进行了内存资源使用的优化以及支持多个虚拟机的特点。需要注意的是，不同于J2me,Android程序在虚拟机中执行的并非编译后的字节码，而是通过转换工具dx将Java字节码转成dex格式的中间码。

4. **Linux内核层**
   Android是基于Linux2.6内核，其核心系统服务如安全性、内存管理、进程管理、网路协议以及驱动模型都依赖于Linux内核。

辅助工具
除了软件本身的代码之外，Android还提供了一系列工具来辅助系统开发，这些主要的工具包括：

- aapt（Android Asset Packaging Tool）：用于建立zip兼容的包（zip、jar、apk），也可用于将资源编译到二进制的assets。
- adb（Android Debug Bridge，Android调试桥）：使用adb工具可以在模拟器或设备上安装应用程序的Apk 文件，并从命令行访问模拟器或设备。也可以用它把Android模拟器或设备上的应用程序代码和一个标准的调试器连接在一起。
- avd manager：android工具是一个脚本，用于创建和管理Android Virtual Devices（AVDs）。
- aidl（Android Interface Description Language，Android接口描述语言工具），AIDL工具可以生成进程 间接口的代码，诸如Service可能使用的接口。
- DDMS（Dalvik Debug Monitor Service，Dalvik调试监视器服务）：这个工具集成了Dalvik，能够在模拟器 或者设备上管理进程并协助调试。可以使用它杀死进程，选择某个特定的进程来调试，产生跟踪数据，观察堆（heap）和线程信息，截取模拟器或设备的屏幕画面，还有更多的功能。
- dx：dx用于将.class字节码（bytecode）转换为Android字节码（保存在.dex文件中）这个字节码文件 是给Android的Java虚拟机运行用的。
- Draw 9-patch：Draw 9-patch工具允许使用所见即所得（WYSIWYG）的编辑器轻松

地创建NinePatch图形。 „ Emulator（模拟器）：模拟器是一个运行于主机上的程序，可以使用模拟器来模拟一个实际的Android系统 的运行，使用模拟器非常适合调试和测试应用程序。

- Hierarchy Viewer（层级观察器）：层级观察器工具允许调试和优化用户界面。它用可视的方法把视图（view） 的布局层次展现出来，此外，还给当前界面提供了一个具有像素栅格（grid）的放大镜观察器。 „ mksdcard：帮助创建磁盘映像（disk image），可以在模拟器环境下使用磁盘映像来模拟外部存储卡（例如 SD卡）。
- Monkey：Monkey是在模拟器或设备上运行的一个小程序，它能够产生随机的用户事件流，例如：点击 （click）、触摸（touch）、挥手（gestures），还包括一系列系统级事件。可以使用Monkey 给正在开发的程序做随机的但可重复的压力测试。
- sqlite3：sqlite3工具能够方便地访问SQLite 数据文件，这是一个sqlite标准命令行工具。
- Traceview：这个工具可以将Android应用程序产生的跟踪日志（trace log）转换为图形化的分析视图。

第二章 项目背景

2.1 应用场景

在移动互联时代，智能手机作为继PC后性能最强的移动运算平台，已经成为了人们生活中必不可少的一部分。当今软硬件发展迅速，更新迭代周期短，智能手机的性能越来越强大，使得人们以前必须在PC上完成的操作如今在移动端即可轻松完成。从收发邮件，查询天气，发微信、微博，网购，查地图，看电影，买机票，我们能想到的各种场景，都已经在智能手机上实现了，而且用户体验比线下更好，更方便。正是由于这些原因，人们花在手机上的时间比以往都要多，可以说，现代人们的生活和工作都已经离不开智能手机了。但是智能手机给人们带来便利的同时也带来了很多操作上的麻烦。例如，为了修改某个系统设置，常常需要用户进入到层级很深的设置里面。这些操作是有规律的，每次手工去修改不仅浪费时间，也让智能手机的使用门槛变高，降低了用户体检。另外，我们在日常生活中经常会有这样的现实需求：如果我到了公司/学校，自动静音；如果我在开车，自动发短信回复来电等等。如果能提供这样的一个工具，用户可以自定义触发的条件，并能选择满足条件后会执行什么动作，那将会大大增强智能手机的可用性。

2.2 竞品分析

通过前期调研，我们发现市场上主要有Locale和Tasker两款与我们的构思类似的产品。
Locale比较有特色的是其结合了人工智能的地理位置服务以及可扩展的插件机制。其定位机制由于缺乏资料无从考究，但可以了解到的是Locale的定位服务综合利用了GPS，Wi-Fi，移动网络等数据源去确定用户当前的位置，号称能瞬间定位，并且能把电量消耗降到最少。
经过对Locale插件开发者指南的研究，我们发现Locale的插件机制主要是通过Android系统的广播实现的。Locale通过发送类型如Edit，Query，Fire的广播并将数据存放在EXTRA_BUNDLE中与插件进行数据交互，定时检测插件Condition是否满足，若满足则通过发送Fire类型的广播唤起对应的插件执行后续的Action。通过这种方式，Google应用商店已经出现了数百款Locale的第三方插件，极大的增强了Locale的功能性和实用性。
另一款同类型的产品Tasker，提供的Action无所不包，从来电、电量、位置到通知被点击等应有尽有，可以说只有想不到，没有做不到，但同时也会造成应用异常之难用，部分选项更是要求用户具备一定的计算机知识，提高了用户的使用门槛。
我们在构思这款App的时候，决定在功能性和实用性两者间，把程序的实用性放在第一位。因为这个App的初衷是让智能手机变得更智能，如果要以复杂的操作和高昂的学习成本为代价，则会偏离了初衷，人为的制造了另一种“不智能”。
通过调研和从用户收集需求，我们列举了以下几个App应该具备的功能：

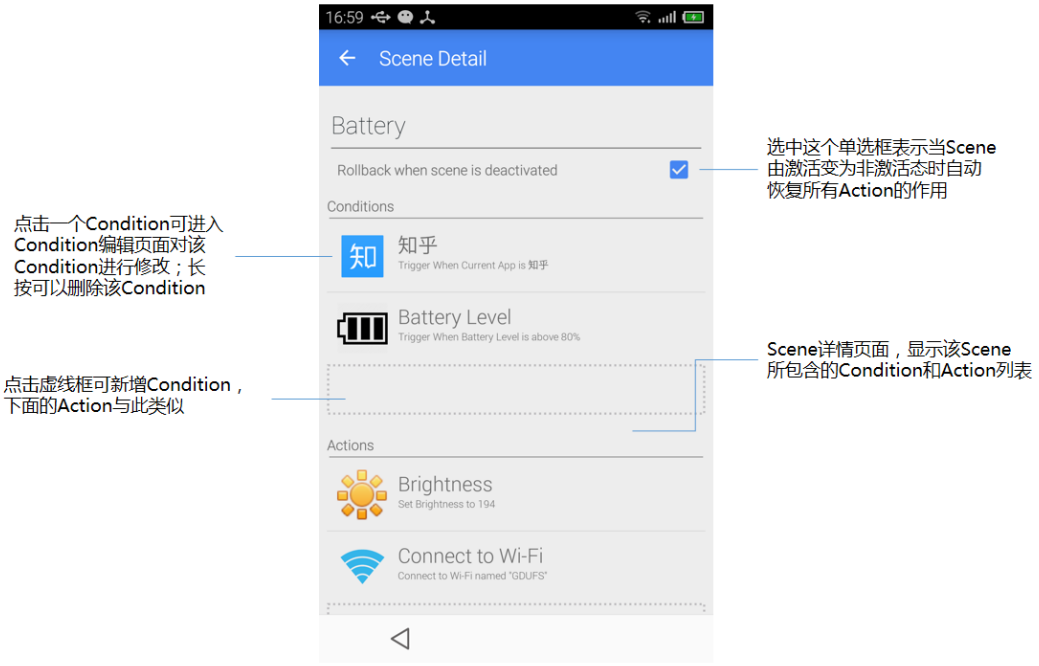
1. 到家或者公司自动开wifi
2. 连到特定的wifi，减低音量，减低亮度
3. 连上HDMI，把亮度调到最低
4. 某些应用延长关闭屏幕时间
5. 插上耳机/蓝牙调整音量
6. 电量低于10%，关闭wifi的数据/降低音量/亮度最低/去振动/关蓝牙gps等
7. 蓝牙15分钟没连上自动关闭
8. 忘带手机，发一条信息，触发a.所有短信转发到gmail b.来电回短信说手机没带并发送手机号到gmail
9. 开某些软件关闭转屏
   从以上几个需求看来，App应该具备修改系统的各项设置的能力，并且能定时的检测系统的状态，能在后台常驻系统。

第三章 功能设计

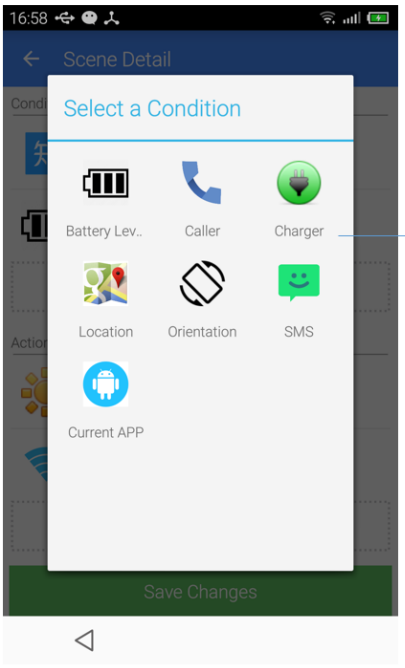
交互设计 & 界面设计

主界面

Scene详情页面



新增Condition页面



点击创建Condition按钮后，弹出创建Condition对话框，可选择创建不同类型的Condition

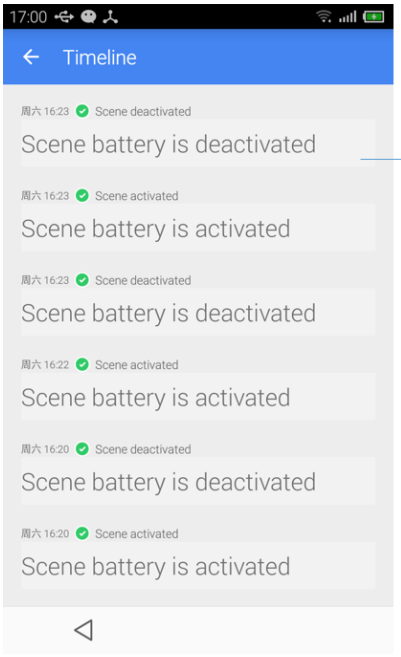
Condition编辑页面(以LocationCondition为例)

LocationCondition编辑界面举例

点击”取消”，放弃对LocationCondition的修改

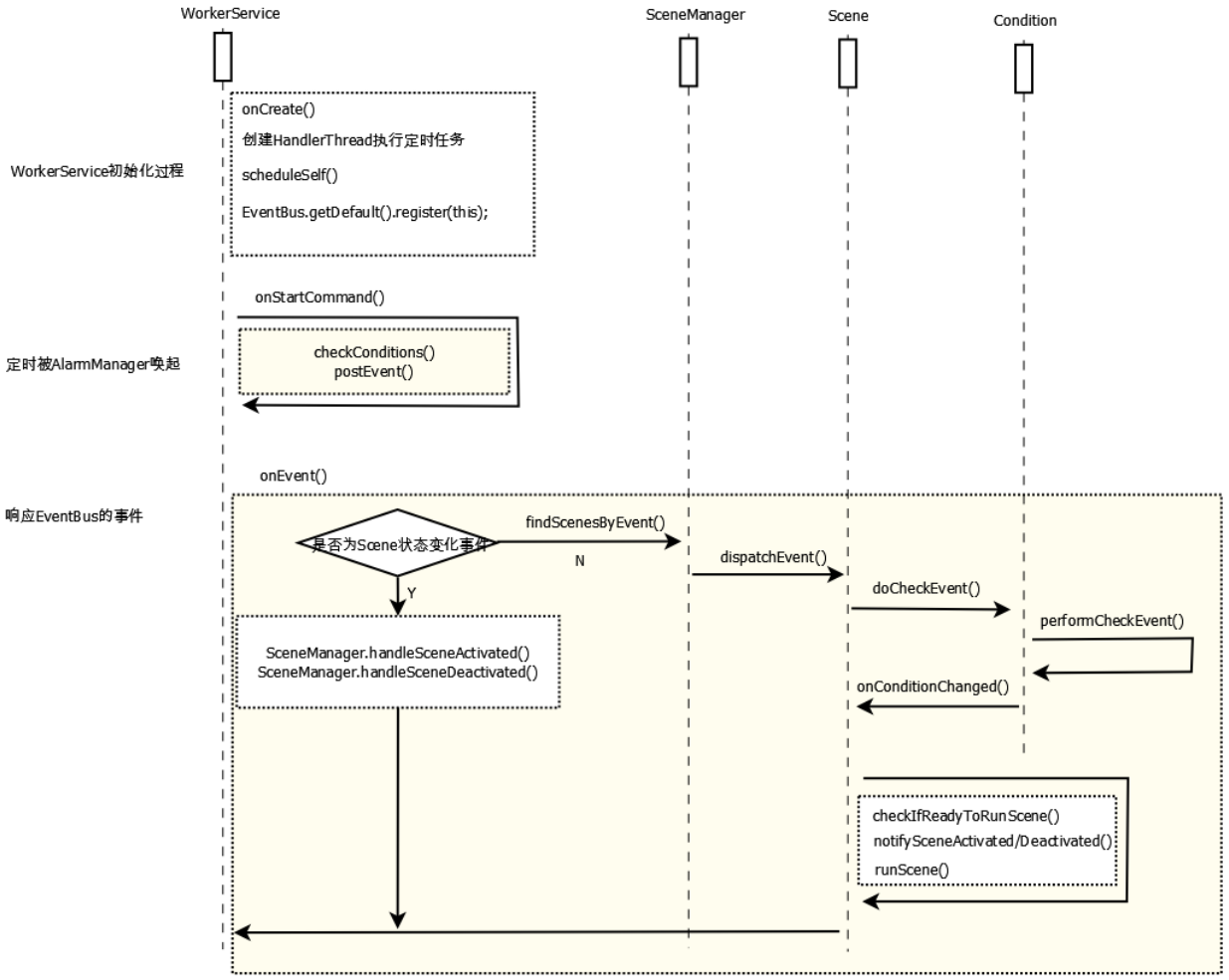
点击”完成”按钮保存对Condition的修改

Scene执行情况时间轴页面



第四章 系统设计与技术实现

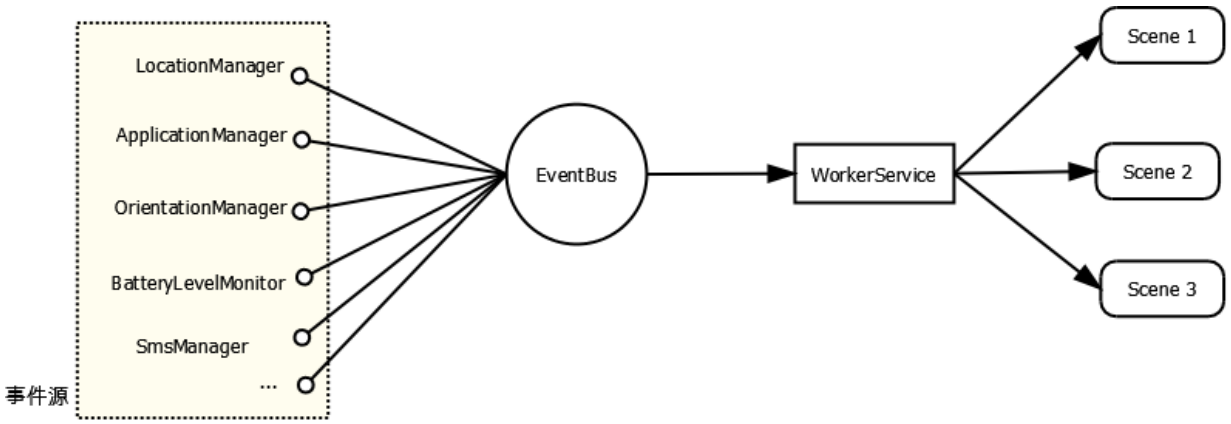
4.1 整体框架与模块划分



注意：淡黄色部分表示在HandlerThread中执行的代码

Tasker主要包括WorkerService、SceneManager和各种监控或修改系统状态的Manager三大模块组成。以上是整个App的时序图，展示了Tasker从初始化启动到触发Scene执行的整个控制流。在WorkerService的`onCreate()`方法中，创建了一个HandlerThread以执行定时任务和处理各种事件，避免因为耗时的后台操作阻塞主线程。`scheduleSelf()`方法通过Android系统的`AlarmManager`调度Alarm，定时唤起WorkerService。最后WorkerService把自己注册到EventBus上，接收EventBus分发的消息。

每次由Alarm唤起WorkerService后，执行`checkConditions()`，检测当前所有活动的Scene对应的Condition状态是否发生了变化。各种Condition的状态变更也是通过EventBus分发出去的。在WorkerService的`onEvent()`方法中，处理EventBus分发过来的各种事件。首先判断是否为Scene状态变更的事件(SCENE_ACTIVATED或SCENE_DEACTIVATED)，若是，则调用SceneManager的`handleSceneActivated/Deactivated()`方法处理；否该事件是来自各种事件源的事件。通过调用SceneManager的`findScenesByEvent()`找到与该事件类型关联的Scenes列表，再通过Scene的`dispatchEvent()`分发给对应的Scenes。Scene获得了这个事件后，具体的Condition满足性判断是有Scene所包含的Condition的`performCheckEvent()`处理的。如果Condition的满足性发生了变化，就会调用listener的`onConditionChanged()`通告其监听者，而Scene实现了这一方法。Scene收到Condition Changed这一通告以后，会考虑是否所有的Condition都已经满足。若是，则会向EventBus投递一个SCENE_ACTIVATED事件，带上Scene自己为参数，交由WorkerService去执行。WorkerService最终会调用Scene的`runScene()`执行某个Scene，具体的执行动作是由每个Action的`performAction()`进行的。

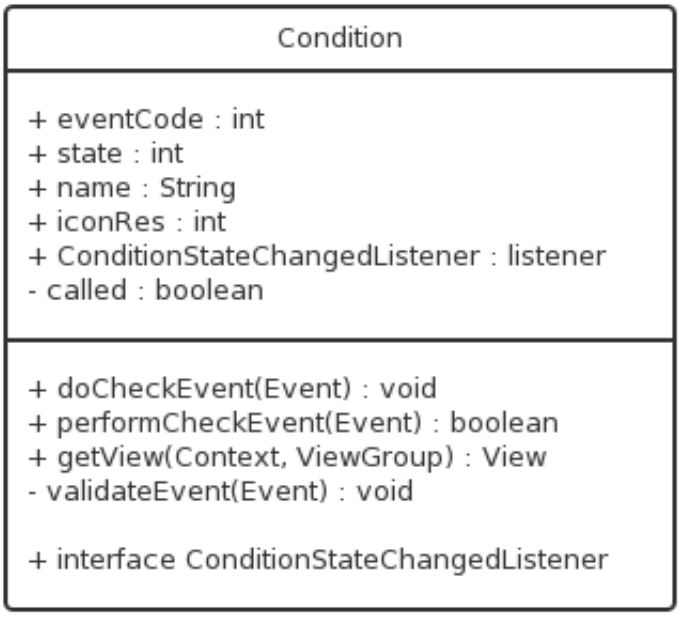


上图是各种事件源、EventBus、WorkerService和各个Scene的关系。事件源分为两种，一种是可以通过BroadcastReceiver或者Listener机制注册的通告型的；另一种是需要主动去检测和获取值的，如当前运行的APP，手机朝向等。所有事件源的事件都会首先被WorkerService截获，然后再通过WorkerService分发到每个Scene。

4.2 数据模型定义

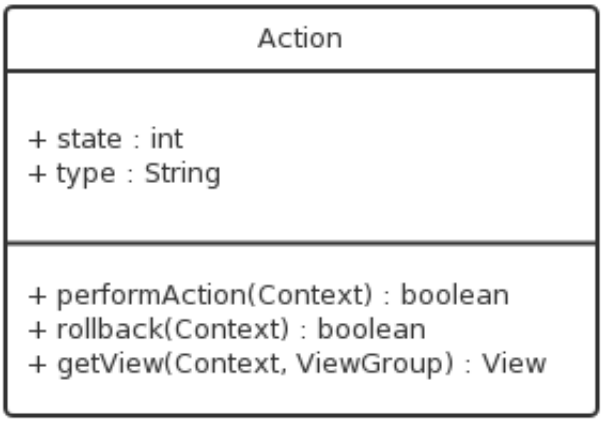
经过分析，我们建立了以下几个Model表示App的数据结构，分别是Condition，Action，Scene和Event。其中Event用于在EventBus中传递消息，携带了事件类型及其参数。Condition是对一个条件的抽象，包含条件的状态，参数等信息。Action是各种操作的抽象表示，泛指一个可以执行的操作，例如修改系统某项设置，发短信，播放音乐等等。Scene表示情景，一个情景是由若干个Condition和若干个Action组成的。当且仅当所有Condition都满足的时候，该情景的Action才会执行。下面详细讨论每个Model的定义。

Condition



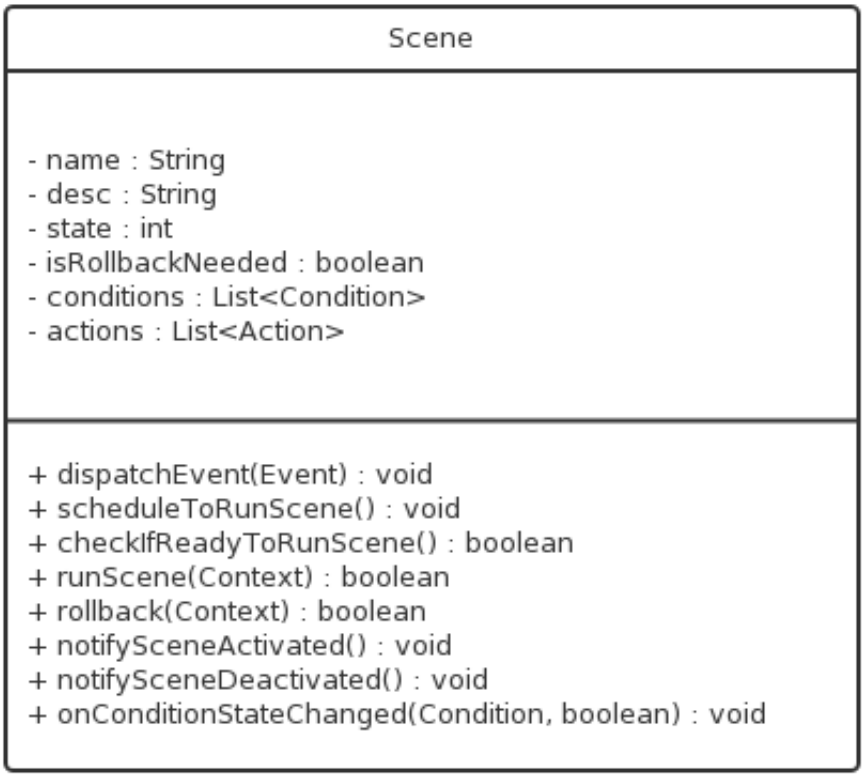
Condition表示一个Scene的前置条件，只有所有的前置条件都满足的情况下，这个Scene才会被执行。Condition和Event都带有一个eventType字段，基本上Event和Condition是一对一的关系。所有的条件都继承自Condition，派生类需要重写 `performCheckEvent()` 和 `getView()` 两个方法。派生类在 `performCheckEvent()` 实现条件是否满足的判断逻辑，在 `getView()` 中创建UI布局，返回的View将会作为一个Item在Condition的列表中显示。派生类在重写 `performCheckEvent()` 时，还必须先调用父类的实现，否则会抛出 `IllegalStateException` 。通过这种异常机制的约束，能确保Event和其派生类间能正确的转型。另外，Condition还定义了 `ConditionStateChangedListener` 接口，用于通告监听者Condition的状态发生了变更，这个监听者是由Scene来实现的。Scene在Condition的状态发生了变更的时候，可以判断是否需要执行或回滚Action，并变更自己的状态。

Action



Action是一个可执行的操作的抽象表示，例如开启Wi-Fi，修改铃声，发送短信等。Action的关键方法是 `performAction()` 和 `rollback()` 。 `performAction()` 提供了Context对象作为入参，派生类在这实现Action的执行逻辑。 `rollback()` 提供了一个Action的回滚，能让一个Scene在由STATE_ACTIVATED变为STATE_DEACTIVATED的时候还原对系统的修改。

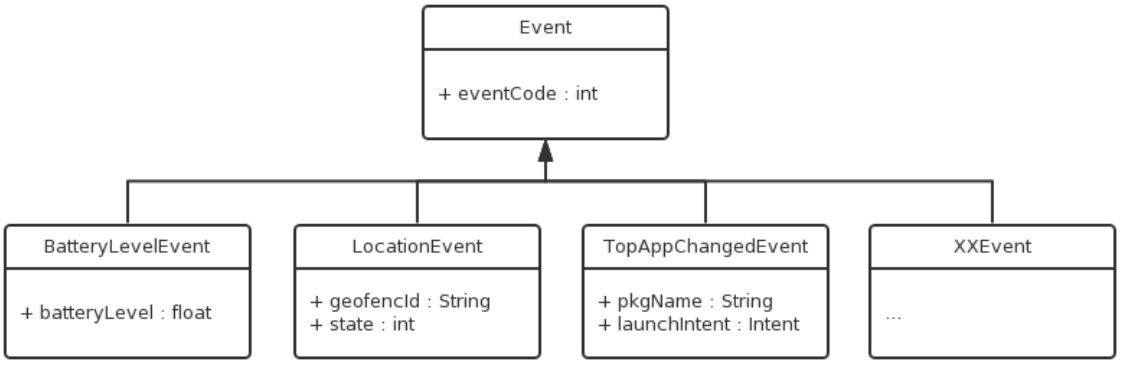
Scene



Scene包含了一个Condition List和一个Action List，类型都为CopyOnWriteArrayList。由于涉及到多线程的并发访问，所以对Condition List和Action List的访问都要放到synchronized同步块中，并且使用线程安全的容器。

Event

所有事件都继承自Event，这使得可以在一个中心化的位置统一处理所有事件，并能根据需求实现事件的分发逻辑。针对不同类型的事件，我们将eventCode这个字段作为事件类型的标识。由于涉及到事件类型的转换，所以在进行事件的比较和转换前，我们加上了对这个字段的校验，只有eventCode相同才能通过校验，否则会抛异常。



4.3 EventBus的使用

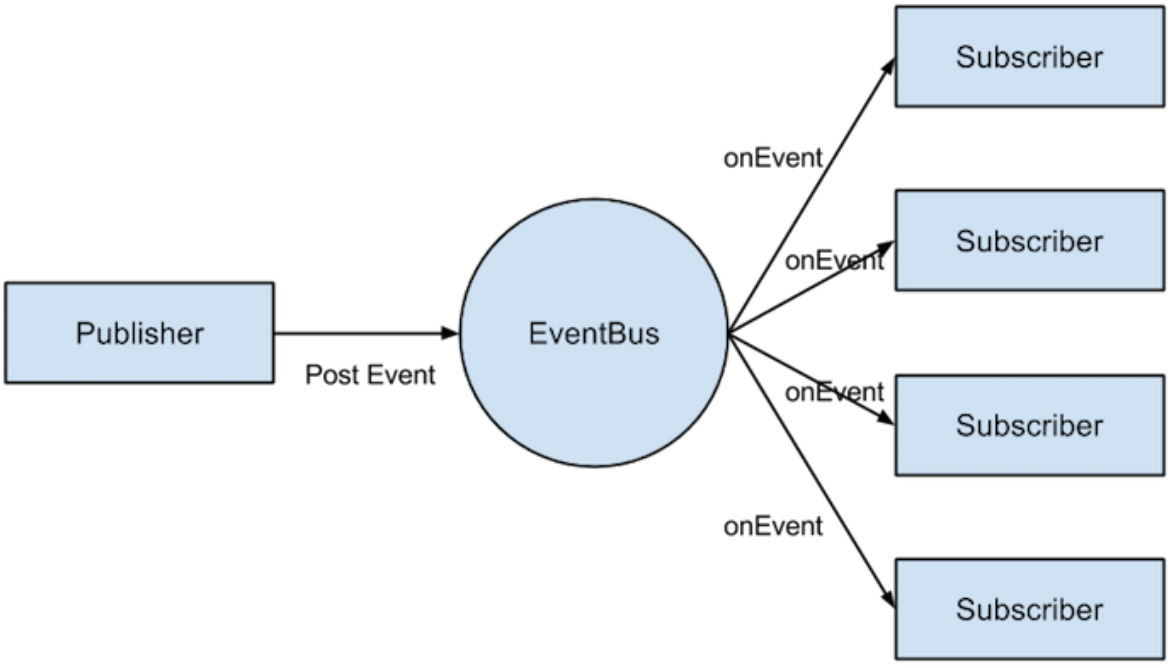
本App使用了第三方库EventBus作为消息分发机制，下面简要介绍一下EventBus的功能和架构。

EventBus 是一个 Android 事件发布/订阅框架，通过解耦发布者和订阅者简化 Android 事件传递，这里的事件可以理解为消息，本文中统一称为事件。事件传递既可用于 Android 四大组件间通讯，也可以用户异步线程和主线程间通讯等等。
传统的事件传递方式包括：Handler、BroadCastReceiver、Interface 回调，相比之下 EventBus 的优点是代码简洁，使用简单，并将事件发布和订阅充分解耦。

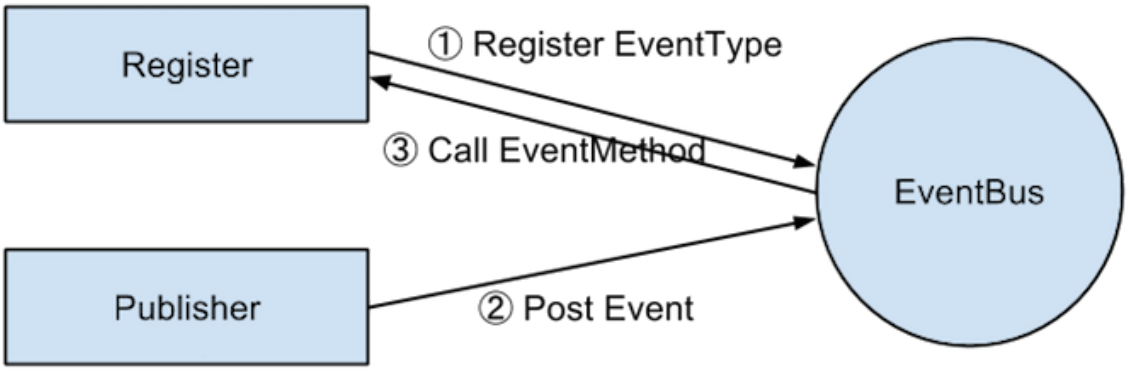
事件(Event)：又可称为消息，本文中统一用事件表示。其实就是一个对象，可以是网络请求返回的字符串，也可以是某个开关状态等等。事件类型(EventType)指事件所属的 Class。
事件分为一般事件和 Sticky 事件，相对于一般事件，Sticky 事件不同之处在于，当事件发布后，再有订阅者开始订阅该类型事件，依然能收到该类型事件最近一个 Sticky 事件。

订阅者(Subscriber)：订阅某种事件类型的对象。当有发布者发布这类事件后，EventBus 会执行订阅者的 onEvent 函数，这个函数叫事件响应函数。订阅者通过 register 接口订阅某个事件类型，unregister 接口退订。订阅者存在优先级，优先级高的订阅者可以取消事件继续向优先级低的订阅者分发，默认所有订阅者优先级都为 0。

发布者(Publisher)：发布某事件的对象，通过 post 接口发布事件。



EventBus 负责存储订阅者、事件相关信息，订阅者和发布者都只和 EventBus 关联。



订阅者首先调用 EventBus 的 register 接口订阅某种类型的事件，当发布者通过 post 接口发布该类型的事件时，EventBus 执行调用者的事件响应函数。

EventBus还提供了对多线程的支持，事件分发可以与事件投递线程不一样。例如，Android对UI相关代码的修改都必须在MainThread中进行，而网络、数据库等其他耗时的操作则不能再MainThread中执行。EventBus提供了三种不同的ThreadMode，可选择调用onEvent()方法的执行线程。

- PostThread 订阅者的onEvent()方法与事件发布线程一致；
- BackgroundThread 如果事件发布线程不是MainThread，则直接在事件发布线程执行；否则，使用唯一的后台线程执行onEvent()方法。
- Async 事件处理方法将会在单独的线程中执行，而且不会影响事件发布线程和MainThread。EventBus使用线程池来有效的管理线程的复用。

4.4 持久化存储

Gson是Google公司发布的一个开放源代码的Java库，主要用途为序列化Java对象为JSON字符串，或反序列化JSON字符串成Java对象。Gson提供了toJson()和fromJson()方便的接口用于在Java对象和JSON字符串之间互转，并实现了对所有Java基本数据类型的自动序列化/反序列化，同时还支持用户自定义的序列化器/反序列化器。对于我们的App，我们需求是实现Condition、Action和Scene对象的序列化/反序列化，由于涉及到继承关系，所以需要提供自定义的序列化器/反序列化器。其具体实现是在序列化的时候，往待返回的JSONObject中增加”type”和”properties”两个key，其中”type”保存了派生类的ClassName，”properties”保存了派生类的包括基类在内的所有成员；而在反序列化时，从”type”中获取派生类的ClassName，并提供给JsonDeserializationContext.deserialize()作为参数，让Gson的ClassLoader能知道派生类的路径，确保能加载到正确的派生类。实现了对象的序列化，下一步就是要解决存在哪里的问题了。鉴于我们的数据量比较少，且数据关系较简单，所以我们采用了Android提供的SharedPreferences存储。SharedPreferences提供了只在App内部才可访问的权限机制，并且能保证数据读写的一致性。将Scene的ID作为Key，可以将Scene的序列化存储到SharedPreferences中。

4.5 LBS

本APP很重要的一个功能是LBS，即能根据用户所处的位置作出响应。为了开发方便，我们使用了百度地图的SDK。百度地图 Android SDK是一套基于Android 2.1及以上版本设备的应用程序接口，可以使用该套 SDK开发适用于Android系统移动设备的地图应用，通过调用地图SDK接口，可以轻松访问百度地图服务和数据，构建功能丰富、交互性强的地图类应用程序。根据我们的需求，我们用到了百度地图SDK的以下几个功能：2D地图的展示，反地理编码，地图选点，定位服务和地理围栏服务。

4.6 常驻后台

在实现过程中另一个比较棘手的问题是如何常驻后台，在系统杀死进程后能自动重启，并且能保持一定的时间间隔唤醒CPU，实现对系统状态的监控。由于涉及到比较多的后台操作，所以我们实现了自己的WorkerService类，利用Android的AlarmManager每隔一段时间发出Alarm，调度执行WorkerService。为了进程被杀死后，Service能重启，需要在`onStartCommand()`返回`START_STICKY`，告诉Android系统重新创建我们的Service。使用AlarmManager的一个问题是无法精确控制Alarm之间的时间间隔。根据官方文档，从API 19开始所有的Alarm都是不精确的，Android系统为了减少手机被唤醒的次数以及较少耗电，会优化Alarm的间隔。如何实现一个可靠的调度机制，是一个值得商榷的问题。。。

WorkerService是整个程序的事件处理中心，我们把所有事件都注册到WorkerService上，这样我们就能方便的观察到各模块的事件流动和实现统一的事件分发。当WorkerService接收到一个事件时，会根据事件的类型作出不同的处理。如EVENT_SCENE_ATIVATED/DEACTIVATED事件会交由SceneManager处理，其他事件则会分发给对应的Scene。Android系统Service是默认运行在主线程的，为了避免后台操作阻塞UI显示，所以我们在WorkerService新创建的时候，开启了一个HandlerThread线程。当WorkerService接收到Alarm时，会将具体操作封装成一个Runnable，Post到后台线程的消息队列中执行，能保持访问的一致性。

4.7 耗电优化

为了降低耗电量，我们采用了动态注册Manager的策略，即按需启动Manager。当一个Manager没有对应的活动的Condition的时候，通过调用`unregister()`方法关闭该Manager，当有新的Condition的时候，再动态的开启。这部分逻辑是在`SceneManager`里面实现的，`addScene()`和`removeScene()`方法分别对应有`registerManager()`和`unregisterManager()`方法。在`registerManager()`中，会对新增的Condition进行检查，如果发现新增的Condition有Manager还没启动的，会自动启动该Manager；而在`unregisterManager()`中，若发现去掉的Condition中有不再使用的Manager，则会自动关闭，减少资源占用。

4.8 插件机制

Tasker能定义的条件和动作是有限的，只有向第三方开放接口，让更多的第三方应用接入进来，才能丰富Tasker的使用场景，让Tasker更好用。经过考虑，我们决定采用Android系统的Intent机制实现我们的插件模块。基本思路是第三方插件继承我们的ConditionProvider和ActionProvider的接口，通过定义一些协议性的参数借助Android的Intent机制，由Tasker进行调度，定时的调用ConditionProvider和ActionProvider的回调方法。在需要的时候唤起第三方插件处理Condition检测，Condition状态报告，Action触发通告等事件。待完善。。。

总结与展望

5.1 总结与体会

略。。

5.2 项目展望

略。。