**知道Android客户端架构**

樊溪

2014/6/25

# 背景简介

Android客户端经历两次技术重构，但重构的思路主要机遇web前端的设计经验。对于客户端的特性没有考虑太多，导致app在业务膨胀与用户体验要求组件提升的过程逐渐显现出：代码耦合度高、数据通用性差、线程管理混乱等问题。针对上诉问题，结合其他app的经验与Android系统的一些特性，我们重新设计了知道客户端的基础架构，即Data-Moel-Presenter-Ui为主的新架构模型。在小房间重构的过程中，该架构模式基本上完善，稳定性与易用性都有了一定程度的优化。为此Android客户端将在后续的版本采取新的架构模型。本文将针对新架构中的技术要点做出详细的介绍。

# 主体结构

客户端软件开发面对比较多的问题主要包括：网络通讯、UI渲染、消息、数据存储与同步、对象生命周期管理、业务封装与剥离。针对这几个主要技术点，我们将整个App的代码结构划分为如图中所示的结构：

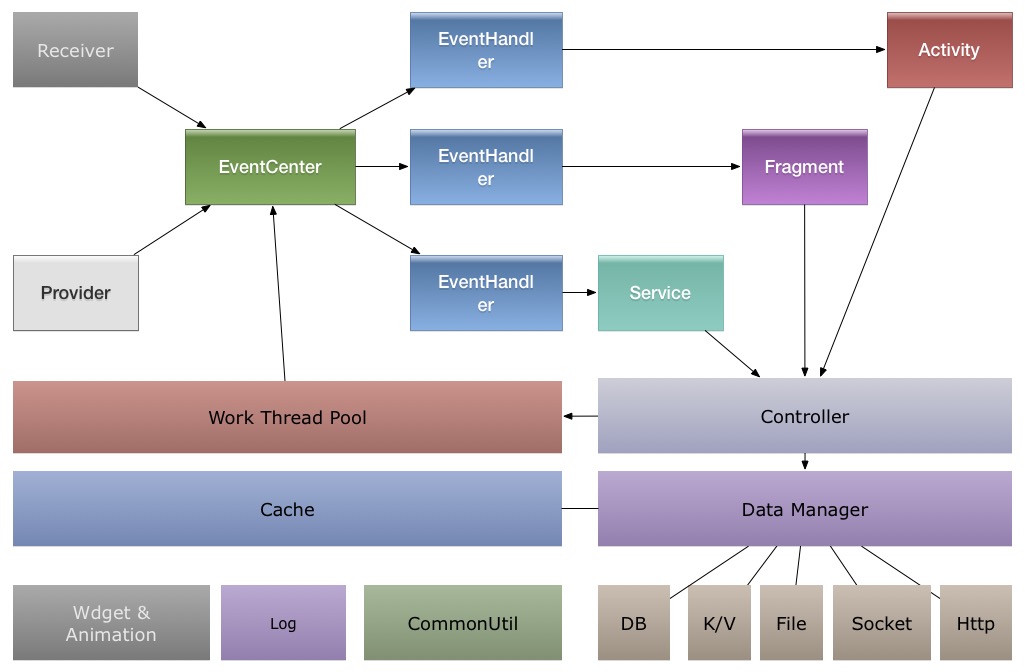


图 1

整个架构基于EventCenter提供的全局消息机制，将Recever，Prorivider，以及线程的通知转化为全局事件，在依靠EventCenter派发给不同的业务实体。

整个App代码层次讲划分为如下几层（图中右边从上往下）：

* UI(业务实体)：包括Activity、Fragment、View、Service等
* Controller（逻辑控制）：负责所有非UI逻辑控制（问题状态，用户财富值计算等）
* DataManager（数据管理）：负责所有数据的存储，管理，升级等
* 底层数据实体：包括DB,文件，网络等。真正负责请求数据的代码，与业务逻辑完全隔离

同时系统将核心公共资源统一封装，提供如图1中左边的所示的基础设施，为不同的业务实体提供服务。

* EventCenter：全局消息队列与消息派发部件
* Work ThreadPool：全局线程池，统一管理所有线程资源
* Cache：通用缓存与数据共享部件
* Log：统计、日志的接口封装
* CommonUtil：通用工具集，提供各种通用的小工具
* Widget & Animation：UI组件与交互动画

# 代码结构划分原则

## UI（业务实体）

业务实体是指直接对接Android系统中：Activity、Fragment、View以及Service几个基础组件的业务逻辑。这些业务实体中的代码主要功能是与系统交互，完成交互事件输入、数据呈现、系统行为控制等用户可直接感知的控制逻辑。其中的代码应遵循以下原则：

* 与系统接口直接交互的代码应该保留在该层实现
* View的控制逻辑应该保留在该层实现
* 不允许直接访问Controller以下的任何层次的借口和数据
* 不允许在该层直接处理与业务相关的操作（比如数据合并等）
* 一个UI实体，只能包涵一个EventHandler , 但可包含多个Controller

## Controller(逻辑控制)

逻辑控制层主要负责响应UI层的请求、数据格式转化、数据状态更新等。其中的代码遵循一下原则：

* 不包含任何与系统直接交互的代码逻辑
* 不包含任何View的控制逻辑
* 全局资源以单例模式提供服务，一般推荐与持有者生命周期绑定
* 不包含任何数据的直接访问，必须通过DataManager完成
* 一般一个业务模块对应一个Controller
* 不持有任何数据，只在生命周期内保存业务状态
* 网络请求暂时在该层操作

## DataManager（数据控制器）

数据管理器是与业务耦合最少的部件，只负责数据的管理（创建、更新、删除等）。管理的内容包括：文件、db、k/v等。划分原则如下：

* 一切数据相关的直接操作都在这一层做
* 一切数据缓存都在该层完成

# 公共组件使用简介

## EventCenter

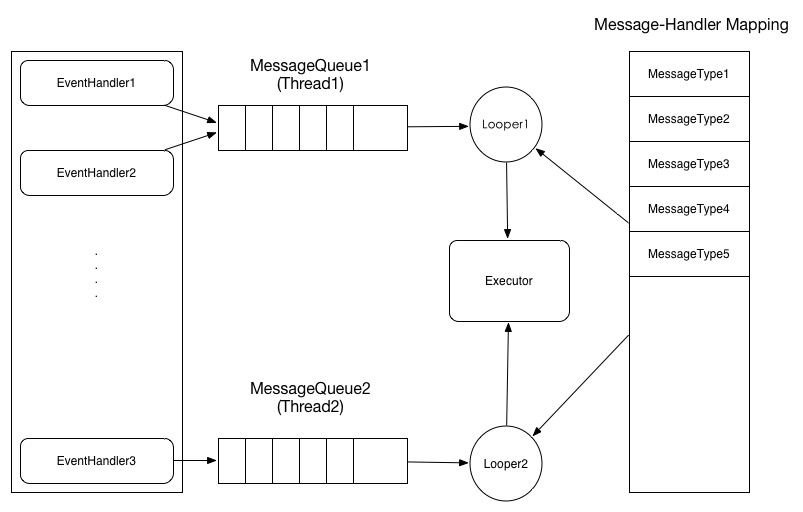
EventCenter 是基于Handler搭建的全局消息通知部件，依靠每个线程自有的消息队列，完成不同线程间的数据传递与状态通知（图2）。

图 2

EventCenter作为全局单例存在，维护EventHandler的注册与解注册、消息派发与时序保证。

EventHandler 依赖当前创建线程的looper，构建自己的消息队列，同事完成消息类型到回掉函数的映射管理。

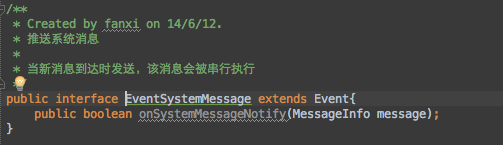
EventCenter的使用主要需要了解三方面的知识：消息定义、消息通知、消息处理

### 消息定义

为简化消息格式定义以及消息号的管理，我们结合java 接口多继承的语法优势，采用了机遇借口的消息格式定义方式。消息定义只需要继承Event接口，在定义的接口中添加相应的回掉函数（即消息处理接口定义）即可。但消息定义必须遵守一下准则：

* 每个消息只能有一个处理方法定义
* 处理方法的定义只能有void 或者 Boolean 类型的返回值

例如：

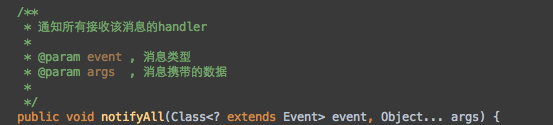


其中如果消息处理函数的定义返回值为boolean，则当该函数返回为true的时候消息将不再继续传递。

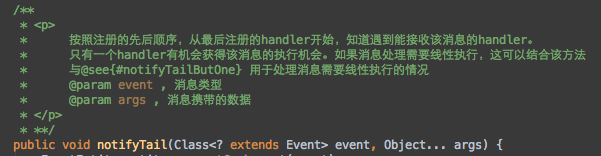
### 消息通知

消息通知有两种模式：广播模式和单播模式。广播模式即向所有接收该消息的Handler发送该消息；单播模式即每次只会向一个接收该消息的Handler发送该消息，Handler选择的顺序依赖其注册的顺序，越晚注册的Handler越先向其发送消息。具体使用如下：

* 广播

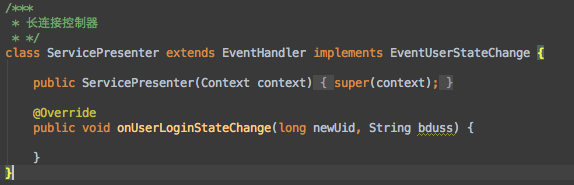


* 单播



### 消息处理

EventHandler是消息最终的处理类，也是负责解析消息以及完成消息映射的最终部件。当EventHandler需要接受一个消息时，只需要基层该消息定义的接口即可。如下图所示：



该Handler将会接收并处理用户状态变化的消息。

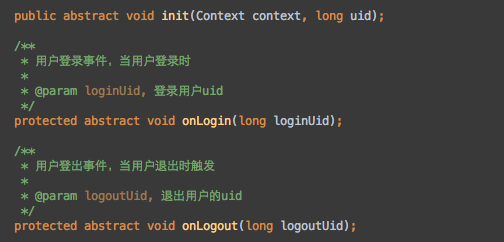
同时EventHandler提供两个生命周期：onStart（）与 onStop（），分别在eventHandler需要开始接收消息与不再接收消息调用。

## Controller

所有Controller 是BaseBizModule 的子类，BaseBizModule封装了DataManager 、EventCenter、ThreadPool的接口。Controller 没有固定的生命周期管理，一般以单例模式存在，严禁任何与生命周期直接关联的资源保存在Controller中。

## DataManager

数据管理器是BaseDataManager的子类，BaseDataManager有三个必须实现的抽象函数：



* init：数据集初始化方法，由DataManagerFactory调用，可在此出初始化数据库连接，创建Dao对象。
* onLogin 与 onLogin：dataManager两个生命周期回掉，当用户登录状态变化时调用。可在此处切换数据库，更新Dao对象。

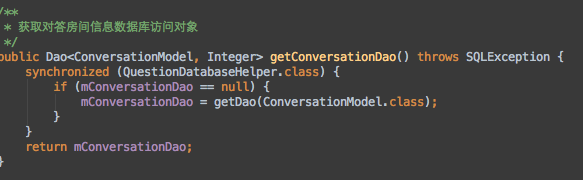
DataManager 负责数据的管理，直接与底层数据存储借口、网络等通讯。每个一个数据管理器都是单例，且都只能由DataManagerFactory创建。

## DataBaseHelper

数据库管理类，维护数据库的连接、升级、创建等操作。每个数据库对应一个Helper, 每个表对应一个Dao.所有的Helper和Dao都是全局唯一的单例模式。DatabaseHelper 有两个固定的回掉借口：

* onCreate()：当数据库创建时回掉该方法
* onUpgrader()：当数据库需要升级时回掉该方法

除此以外，DatabaseHelper 必须提供Dao对象的创建和维护工作。所有的数据库表都必须提供一个Dao对象，并通过get方法对外暴露。例如：



# 模块划分与数据库设计

## 模块划分

知道app根据现有业务的功能，划分为如下11个模块：

* 问答流程
* 消息
* 用户
* 公共模块
* 芝麻圈
* 商城
* 任务榜单
* 排行榜
* 日报
* 私密问答
* 检索模块

每一个模块将创建一个独立的数据库，同时对应独立的一个DataManager。即每个独立业务模块，都将拥有自己的Controller、DataManager。

## 数据库设计

待定