项目：yourDiary

软件设计文档

项目组成员：

邓杰友 陈颂熙

彭志锋 郑嘉俊

张浩宇 张树盛

2017 年 6 月 23 日

Version history

日期	修订者	审阅人	版本	备注
2017-06-23	陈颂熙	邓杰友	V1.0	SD

备注：这个课程设计采用的项目是大三上 Android 应用开发的期末项目，其中CircleImageView.java和CircleButton.java不是本小组原创而是使用了网络上的开源的代码文件。

目录

一、技术选型理由

我们开发的是安卓手机上的一个本地日记 app，在 Android Studio 这个 IDE 的基础上，我们可以很方便的进行开发。展示页面用 XML 来进行设计，具体的交互操作由 Activity 来承担，而数据持久化则使用 SQLite 来实现。所以大体的技术是 XML+Java+SQLite。

1、XML

在 Android Studio 中，XML 作为界面元素展示是非常恰当，其基础类型非常齐全，样式定义也丰富，学习成本不高，适合我们年轻团队快速学习快速开发。

2、Java

Java 本身就具有对 Android 提供大量支持的 Android 库，在这些库的支持下，编写 Android 程序是十分轻松的。而且 Java 语言本身的特点，让我们可以集中更多的精力在代码开发编写上而不是担忧 app 运行的过程。

3、SQLite

选用 SQLite，很大程度上也是因为 Android 库本身也提供了完整成熟的数据存储的库，其中 SQLite 的轻量便捷，得到了我们的青睐，它非常适合像我们这样对数据库操作压力不大，存储量不多的情景。

二、架构设计

1、Object-Oriented Programming

基于 Android 程序的开发的特点，特别是 Java 的语言特点，在具体的架构设计上，我们运用了较多的 Object-Oriented Programming。

例如，一份日记这个对象有着它的创建时间、文本内容、图片内容、标题等信息，若我们仅仅只是利用一个 Map 或者一个结构体来实现，那么在开发的过程中，需要面对的就不仅仅是这几种属性的协同，还带来了代码的臃肿以及他人理解上的障碍。所以我们将其抽象成为一个类，利用这个类的 getter 和 setter 来完成这些信息的操作，很大程度上弱化了开发人员对这些属性信息的关注程度，以便于他们能集中在具体的业务开发上。

```
21▼ public class Diary {
22▶ ...
30     private int month;
31     private int day;
32     private String title;
33     private String content;
34     private byte[] picture;
35▶ ...
41▼    Diary(int month, int day, String title, String content, byte[] picture) {
42         this.month   = month;
43         this.day     = day;
44         this.title   = title;
45         this.content = content;
46         this.picture = picture;
47     }
48▶ ...
52     public byte[] getPicture() {
53         return picture;
54     }
55▶ ...
58     public int getDay() {
59         return day;
60     }
61▶ ...
64     public int getMonth() {
65         return month;
```

又例如，在数据库操作上，我们负责业务逻辑的开发人员其实并不在意数据库的操作，他们需要的仅仅只是一个能告知操作是否成功的 restful 接口。所以，这里我们则将繁琐的数据库操作（增删查改以及伴随的异常处理）封装成为 API，并将这些 API 集中到一个名为 DatabaseAdapter 的类里面，利用这个类，业务开发人员就可以将精力集中在业务流程上，使得开发效率大大提升。由此而来的还有代码可读性的大大提高。

```
60     /** DatabaseAdapter API index:
61      *      TABLE USER:
62      *          boolean insertUser(String username, String password, byte[] head)
63      *          boolean updateUser(String username, String password, byte[] head)
64      *          boolean deleteUser(String username, String password)
65      *
66      *          boolean isUserExist()
67      *          // TODO: getPasswordByUsername(String username)
68      *
69      *      TABLE DIARIES:
70      *          boolean insertDiary(int month, int day, String title, String content, byte[] picture)
71      *          boolean updateDiary(int month, int day, String title, String content, byte[] picture)
72      *          boolean deleteDiary(int month, int day)
73      *
74      *          ArrayList<ArrayList<Diary>> queryDiaryAll()
75      *          ArrayList<Diary> queryDiaryByMonth(int month)
76      *          Diary queryDiaryByMonthAndDay(int month, int day)
77      *
78      *
79      */
```

2、Model-View-Controller (MVC)

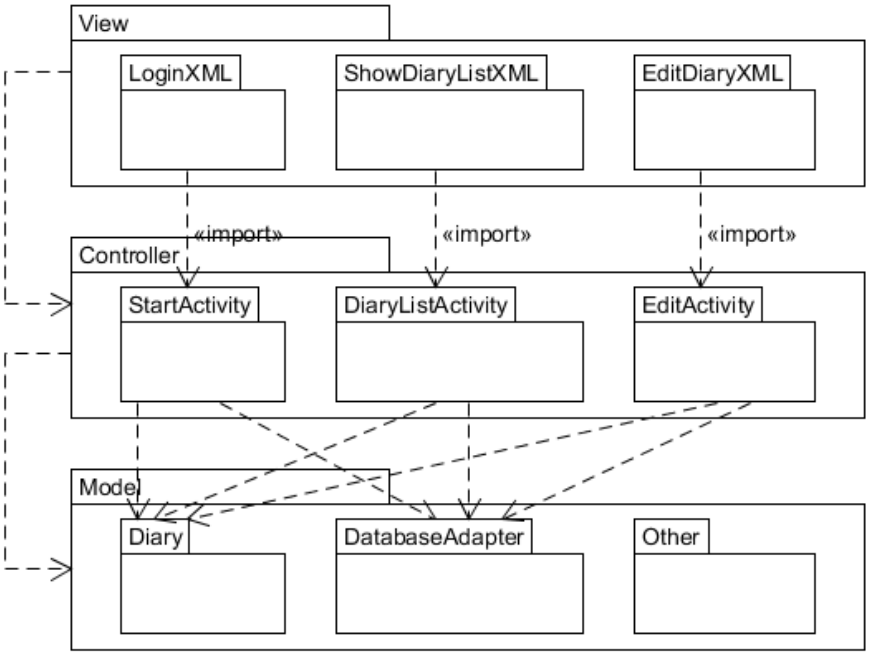
关于整个项目的架构，我们基于 XML+Java+SQLite 的这样的技术选型，也提出了 MVC

的模式架构。在具体的每个页面上，我们利用 XML 来提供视图（view），其对应的具体交互落实在 Java（Activity 活动）上，这部分则相当于控制器（controller），最后的模型（model）则利用我们之前谈到的 Diary 类和 DatabaseAdapter 类来提供支持。这样我们的开发过程则可以实现一定程度上的分离，视图 xml、控制器 controller 和模型 model 可以相对独立的同时进行开发。最后只要完成之间的接口调整，就能顺利实现组合。一定程度上加快了我们开发的过程，也弱化了我们项目代码之间的耦合程度。

具体体现在我们代码组织结构和具体实现上。

三、模块划分

在模块划分上，由于我们的项目需求比较清晰，可以根据其将具体的模块大致上分为日记入口、日记展示、日记编辑。而结合我们 MVC 架构的设计，以及技术选型的特点，模块则可分为如下图所示：



在这里视图和控制器之间的耦合程度较高，到现在仍没有很好的解决方案，只能让对应的开发人员自己沟通协同，这点值得商榷；但在控制器和模型之间，我们做到了高内聚低耦合，MVC 架构的特点在这里得到了不错的体现。