Mobile Data Monitoring System Based on MVP Mode

Jia-jie FAN*, Yu ZENG and Ming-yu GAO
College of Electron Information, HangZhou Dianzi University, Hangzhou, Zhejiang, China
*Corresponding author

Keywords: MVP, Android, LitePal.

Abstract. The traditional MVC pattern has several shortcomings in Android application development, such as complex structure and too tight connection between Controller and View. The key class Activity in Android is the combination of Controller and View classes, which is not only responsible for the operational logic, but also responsible for the display. The Activity has a high degree of coupling. The Presenter introduced by MVP model completely separates the Model layer from the View layer, which solves the problem of overly bloated activities and also helps in later testing and maintenance. This article analyzes the shortcomings of traditional MVC application development in Android, and explores the feasibility and advantages of MVP mode in Android development, and finally implements the application of MVP mode in Android-based data monitoring system.

Introduction

MVC operation mode refers to that the View layer would send instructions to the Controller layer, and then the Controller would inform the Model layer to get the data, and finally the View would read the data obtained by the Model layer when an event is triggered. Activity in Android is almost undertaken two kinds of roles of the View layer and the Controller layer, and they are severely coupled with the Model layer, which is troublesome to maintain in a logically complex interface[1].

MVC mode and MVP mode comparison diagram is shown in Figure 1. In the MVP mode, there is no direct connection between the View and Model, but through the Presenter to connect, the MVC pattern exists in the Activity or The logic code in Fragment is placed in the Present layer, which relieves the high coupling of the MVC pattern and the excessively bloated Activity code. The communication between the three layers is mainly accomplished through the interface.

Activity in the MVP mode only acts as the View layer. The Controller layer is completely responsible by the Presenter. The communication between the View layer and the Presenter layer is implemented through the interface. Therefore, there is no coupling problem. The View layer and the Model layer are completely decoupled due to the Presenter layer. The Presenter is highly reusable and can be moved to any interface[2].

![Comparison between MVC mode and MVP mode.](image)

**Figure 1.** Comparison between MVC mode and MVP mode. (a) MVC mode (b) MVP mode.

Application of MVP Mode in Data Monitoring System

System Function Design

The main functions of the data mobile monitoring system are divided into two blocks: real-time data and historical data. Historical data includes historical failure data and historical equipment data, as
shown in Figure 2.

![Figure 2. Chart of system functions.](image1)

![Figure 3. Data Monitoring System Architecture.](image2)

After setting the server’s IP address and port number correctly, we can click any device name in the device list to jump to the real-time data display interface. The app actively requests the server for real-time data of the device and can automatically refresh real-time data. We click any data name in the real-time data interface to jump to the historical data interface, which would stop the real-time data refresh task. After selecting the format yyyy:MM:dd, the app requests the server for all records of the data on the day. If the data is about temperature information, the graph is used to display the data. If the data is output results, the bar graph is used to display the data. The graph can be saved to the local album by clicking the “save” button which is named after the format of “Data Name + Date (yyyy:MM:dd)”. The historical failure data is displayed in the form of a list in the interface, with the main title and subtitle showing the time of failure description and fault recording.

**System Architecture**

The architecture diagram of the data monitoring system is shown in Figure 3. The first layer is the UI layer for data display and operational interaction, the second layer is the business layer of real-time data, historical failure data, historical equipment data, the third layer is the database layer, and the fourth layer is the common tool layer, including the text class, data parsing class, network communication class and network state detection class.

**Software Architecture Design**

**View Module Design.** Three interfaces are defined in the View layer, and that are CurrentDataView, ProblemDataView, and DataLogView. Take real-time data as an example, as shown in Figure 4, the CurrentDataView interface defines six methods to complete the different stages of the interface display work, when click on the real-time data of the device, the requestToServer() method would call the real-time data network request method in the Model layer through Presenter, and then Presenter calls the related method to display the load frame. If the request is successful, CurrentDataView would call the showData() to display the real-time data and closes the load frame by calling the hideProgress(). Otherwise, the corresponding failure information is displayed according to the failure type.

**Figure 4. Real-time data view layer interface.**
**Presenter Module Design.** The Presenter layer is the middle layer bridging the View layer and the Model layer, which would construct View layer after acquiring the data from the Model layer. After receiving feedback commands on the View layer, the Presenter layer can also distribute processing logic to the Model layer. It can also determine the various operations of the View layer. Take real-time data as an example. As shown in Figure 5, the CurrentDataPresenter defines four methods to control the related methods in the View layer and the Model layer, via `queryFromServer(String address, int port, byte[] buffer, ExecutorService singleThreadExecutor)`. Presenter would call the real-time data network request method in the Model, and `showCurrentProgress()` and `hideCurrentProgress()` are the methods that control the display and hidden state of the loaded box in the View. Presenter would release the reference to CurrentDataView via `onDestroy()` to prevent memory leaks. The Model's callback interface is also included in the CurrentDataPresenter layer to notify the result of the View layer operation. As shown in Figure 6, when the network fails, `onNetworkError()` is called back. When the server is abnormal, `onServerError()` is called, otherwise callback is `onSuccess()` to inform CurrentDataView to display real-time data[5].

![Figure 5. Real-time data presenter layer interface.](image1)

![Figure 6. Callback interface.](image2)

**Model Module Design.** Two methods of real-time data model interface definition are shown in Figure 7. They are respectively responsible for network request and data parsing. They notify the result of View operation through the callback interface defined in the Presenter layer. We also take real-time data as an example. Through the method of `socketRequest(String address, int port, byte[] buffer, ExecutorService singleThreadExecutor, OnRequestFinishedListener listener)`, we can request real-time data from the server. The data returned by the server can be processed and stored to database through the method of `currentDataParse(String data)` .We can inform CurrentDataView of the operation result through the callback interface defined in CurrentDataPresenter[6].

![Figure 7. Real-time data model layer interface.](image3)

**Database Design**

LitePal is an open source Android database framework, using the object relational mapping (ORM) model, which encapsulates some of the most commonly used database functions in development, allowing developers to complete a variety of tables without writing a single line of SQL statements. And LitePal is very light, the jar package size is less than 100k, and almost zero configuration, which is very different from the Hibernate framework[7].

We create four new tables to meet the needs of local data storage.

1. Table. 1 Data is responsible for storing real-time data.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>did(int)</td>
<td>Data number</td>
</tr>
<tr>
<td>vid(int)</td>
<td>Device number</td>
</tr>
<tr>
<td>val(char[20])</td>
<td>Data content</td>
</tr>
<tr>
<td>txt(char[50])</td>
<td>Data meaning</td>
</tr>
<tr>
<td>dtime(datetime)</td>
<td>Recording time</td>
</tr>
</tbody>
</table>

2. Table. 2 Alarm is responsible for storing historical fault data.

(1) Table. 1 Data is responsible for storing real-time data.

(2) Table. 2 Alarm is responsible for storing historical fault data.
Table 2. Alarm.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>vid(int)</td>
<td>Faulty equipment</td>
</tr>
<tr>
<td>wtype(int)</td>
<td>Fault type</td>
</tr>
<tr>
<td>wtxt(char[50])</td>
<td>Fault description</td>
</tr>
<tr>
<td>dtime(datetime)</td>
<td>Recording time</td>
</tr>
</tbody>
</table>

(3) Table 3 Datalog is responsible for storing historical device data.

Table 3. Datalog.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>vid(int)</td>
<td>Device number</td>
</tr>
<tr>
<td>datas(byte[524])</td>
<td>Data content</td>
</tr>
<tr>
<td>dtime(datetime)</td>
<td>Recording time</td>
</tr>
</tbody>
</table>

(4) Table 4 Ip is responsible for storing the IP address and port number of the server that the user needs to connect to.

Table 4. Ip.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip(String)</td>
<td>IP</td>
</tr>
<tr>
<td>port(int)</td>
<td>Port</td>
</tr>
</tbody>
</table>

Communication Protocol Design

The app creates a new socket connection using the server’s domain name (IP address) and port. It sends a connection request to the server through this port number. If the connection is unsuccessful, an exception is thrown in order to reduce server stress between the client and the server. The connection is a short connection. The specific request parameter protocol is shown in Table 5[8].

Table 5. Communication protocols.

<table>
<thead>
<tr>
<th>APP send to the server</th>
<th>Real-time data</th>
<th>Historical failure data</th>
<th>Historical equipment data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(APVV),(data number),(check code)</td>
<td>(APVW),(data number),(number limitation),(check code)</td>
<td>(APVH),(data number),(date),(check code)</td>
</tr>
<tr>
<td>The server send to APP</td>
<td>Real-time data</td>
<td>Historical failure data</td>
<td>Historical equipment data</td>
</tr>
<tr>
<td></td>
<td>(SPVV),(data number),(data),(recorded time),(check code)</td>
<td>(SPVW),(data number),(fault type),(data),(recorded time),(fault description),(check code)</td>
<td>(SPVH),(data number),(recording time),(data),(check code)</td>
</tr>
</tbody>
</table>

The data sent and returned are represented by the hexadecimal array. In fact, the sending and receiving frame headers are represented by a hexadecimal number of 4 bytes. The data number is in the range of 100 to 137, and the temperature and the output data is represented by a double-byte integer. The time data format is yy:MM:dd: HH:mm:ss which length is 6 bytes, the length of the fault description is 63 bytes, and the check mode is cumulative sum check.

Thread Pool

The biggest difference between MVP mode and MVC mode is that View does not use Model directly in MVP mode. The communication between them is through Presenter (Controller in MVC). All the interactions take place inside Presenter and in MVC. Views would read data directly from the Model instead of the Controller. From the process of clicking and viewing data to displaying data, time-consuming operations such as network communication are not allowed to be performed in the
main thread, so the thread pool is particularly important at this time. It is not only an important tool for implementing network interaction but also the hub and flag in the overall process control[9].

First, an important feature of JAVA thread pools is introduced. Under normal circumstances, the method isTerminated() could be true after the thread pool object executing shutdown(), and thread pool would stop accepting the new task. When the tasks in the current queue is processed, the thread pool would be recycled by the garbage collector. Therefore, isTerminated() can be used to judge whether the background network tasks and database tasks have been completed. Finally the data would be sent to the View via the Presenter[10].

The following is the specific process to achieve this logic shown in Figure 8.

![Figure 8. Thread pool workflow.](image)

**Software Testing**

After testing, the data monitoring system based on the MVP mode is running normally. There is no interface when the interface is switched and network requests are made. The operation results of some interfaces are shown in Figure 9.

![Figure 9. Software operation screenshot.](image)

**Summary**

This paper analyzes the advantages of MVP compared with MVC, and analyzes the feasibility of
MVP software design mode in Android development. It also uses MVP mode in the development of Android data monitoring system software to reduce the coupling of code. It improves the readability of the code and the efficiency of the unit test. The tested software has good running fluency and stability, and has certain promotion and reuse value.

References


