车载CAN通信Android应用的设计模式研究与应用

黄慧[1]　李荆轩[2]

（1.中国第一汽车股份有限公司研发总院，长春 130013；2.一汽大众汽车有限公司，长春 130011）

【欢迎引用】黄慧，李荆轩. 车载CAN通信Android应用的设计模式研究与应用[J]. 汽车文摘，2024(1): 1-7.

【Cite this paper】HUANG H, LI J X.Research and Application on Design Modes for Android Application on Vehicle[J].Automotive Digest (Chinese), 2024(1): 1-7.

【摘要】在汽车中，电子控制器件可以通过CAN协议连接成局域网络用于信息传输。车载娱乐系统在这些电子器件中扮演着与用户交互的重要角色，为了满足用户生态化需求和友好的界面操作体验，Android系统被引入车载娱乐系统中。文中主要探讨了通过Android应用程序采用模型-视图-控制器（Model-View-Controller，MVC）、模型-视图-表示器（Model-View-Presenter，MVP）和模型-视图-视图模型（Model-View-View Model，MVVM）设计模式来访问CAN网络的实现方式，并对这3种设计模式进行了比较，分析它们在访问CAN网络方面的优缺点，结果表明MVVM设计模式更适合于车载娱乐系统应用程序访问CAN网络。

关键词：Android应用；设计模式；MVC；MVP；MVVM

中图分类号：TN929.53　　**文献标识码：**A　　**DOI:** 10.19822/j.cnki.1671-6329.20220166

Research and Application on Design Modes for Android Application on Vehicle

Huang Hui[1], Li Jingxuan[2]

（1. Global R&D Center, China FAW Corporation Limited, Changchun 130013; 2. Technical Development Department, FAW-Volkswagen Automotive Co., Ltd, Changchun 130011）

【Abstract】 In automobiles, electronic control components can be connected into a local area network through the CAN protocol for information transmission. The in-car entertainment system plays an important role in interacting with users among these electronic components. In order to meet the users' ecological needs and provide a friendly interface operation experience, the Android system has been introduced into the in-car entertainment system. This study mainly explores the implementation of accessing the CAN network through Android applications using the Model-View-Controller (MVC), Model-View-Presenter (MVP), and Model-View-View Model (MVVM) design patterns, and compares these 3 design patterns to analyze their advantages and disadvantages in accessing the CAN network. Ultimately, it concludes that the MVVM design pattern is more suitable for the application of accessing the CAN network in the In-Vehicle Infotainment system.

Key words: Android application, Design mode, MVC, MVP, MVVM

缩略语

CAN	Controller Area Network
ECU	Electronic Control Unit
UI	User Interface
MVC	Model-View-Controller
MVP	Model-View-Presenter
MVVM	Model-View-View Model

0　引言

车载娱乐系统中的操作系统主要包括WinCE、QNX、Linux、Android、鸿蒙OS和阿里OS。由于汽车制造企业在定制方面的需求日益增多，Android系统因其生态化、灵活性和受用户欢迎的优点，在市场上占据越来越大的份额。随着搭载Android系统的车载娱乐系统增多，对Android应用程序访问整车电子控制器件的

需求也越来越多。车载娱乐系统的Android应用程序需要能够正确地发送控制指令给整车电子器件，并将其状态反馈到界面上，因此需要研究一种高效的设计模式来实现这一目标。因此，本文对车载娱乐系统中Android应用程序的设计模式进行了研究。

车载娱乐系统中使用Android应用程序访问CAN网络与Android终端应用程序访问HTTP网络有所不同。车载应用程序的主要功能特点在于管理涉及控制器局域网络的业务逻辑和用户界面。与Android终端应用程序相比，车载应用程序发送和接收CAN网络报文具有以下不同之处：

(1)请求CAN网络得到反馈所需的时间比请求HTTP网络更长，这是因为请求CAN网络需要考虑对方电子控制单元(Electronic Control Unit，ECU)计算的时间。

(2)CAN网络会定时反馈对方ECU的状态。

(3)如果对方ECU存在非预期错误，将会通过CAN网络反馈无效值或保留值。

本文研究了适用于车载应用的CAN网络应用程序设计模式和方法，并探讨了适用于车载应用的HTTP网络应用程序设计模式和方法。本文以Java作为Android系统的上层开发语言，并使用Java代码作为示例。

1 MVC设计模式

模型-视图-控制器(Model-View-Controller，MVC)是一种常用的设计模式，用于组织和管理软件应用程序中的交互和数据流。MVC设计模式，如图1所示[4]。

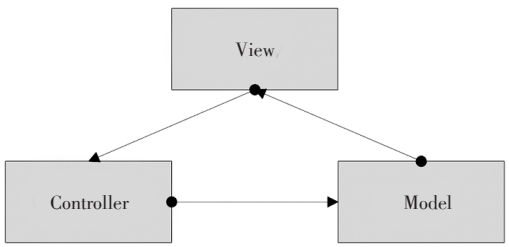

图1 MVC设计模式逻辑[4]

Model代表数据模型，负责保存和管理数据。当数据发生变化时，Model负责通知View更新界面显示。

View代表用户界面，接收用户的请求并将其传递给Controller。View主要负责与用户进行交互和展示数据。

Controller是控制器，负责处理业务逻辑。当Controller对业务进行处理后，会通知Model进行数据更新。

1.1 MVC优点

(1)模块职责划分明确。主要划分为Model、View和Controller模块，有利于代码维护。

(2)这种设计模式使得数据、界面和业务逻辑之间的耦合度降低，提高了代码的可维护性和可扩展性。

1.2 MVC缺点

(1)在Android开发中，通常将Activity组件作为控制器(Controller)角色来执行。然而，在实际业务应用中，Activity也会控制一些涉及业务逻辑刷新的用户界面(User Interface，UI)元素，例如进度条等。这就导致部分View和Controller的功能合并在同一个类别中，使得Activity的代码逐渐膨胀。这种情况下，View和Controller之间存在耦合，不符合模型设计的初衷，给维护带来了困难。

(2)此外，View和Model之间也存在一定的交互逻辑，并没有完全分离开来，导致View和Model之间也存在耦合。

(3)上述设计还仅仅实现了在车载应用中给对方ECU发送指令，并默认对方ECU能成功执行并刷新界面。但实际上，对方ECU并不一定能成功执行该指令，并且从CAN网络接收到反馈的时间也相对较长。因此，在Controller中需要额外设计，以便在对方ECU执行失败或用户频繁发送指令导致对方ECU不断返回反馈的情况下，能正确通知Model并相应地更新View。这使原本已经庞大的Controller复杂度和大小进一步增加。

1.3 MVC设计模式在车机上应用的结论

(1)MVC不适合用于处理与CAN网络相关的业务和UI逻辑。

(2)MVC可以应用于简单的数据界面控制逻辑。

1.4 MVC实现步骤

(1)Model层

通过抽象出符合业务逻辑的Model层，可以实现数据的保存和处理，并在处理完成后通知UI层更新显示内容。

Model实现程序如下：

```
public class ACModel {
    private intmTemperature = 0;
    public void upTemperature(ControllerActivity activity) {
        //更新数据
mTemperature = ++mTemperature;
        //发送CAN控制指令
CANManager.getInstance().setACTemperature(mTemperature);
        //更新UI
```

```
activity.upTemperature(mTemperature + "");
    }
}
```

（2）View层

在Android应用中，View层作为用户界面，通常指的是Activity的xml布局文件。这些布局文件定义了用户界面的组件和布局结构，包括各种UI元素，如按钮、文本框和图像等。通过使用这些xml布局文件，可以创建出具有交互性和可视化效果的用户界面。View实现程序如下：

```
activity_controller.xml ：
<?xmlversion="1.0" encoding="utf-8"?>
<RelativeLayoutxmlns:android="http://schemas.an -
droid.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:id="@+id/show_temperature"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textColorHint="@color/colorPrimaryDark"
android:text="temperature"
android:layout_centerInParent="true"/>
<Button
android:id="@+id/temperature_up"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="up"
android:layout_below="@+id/show_temperature"
android:layout_centerHorizontal="true"
android:layout_marginTop="50dp"/>
</RelativeLayout>
```

（3）Controller层

Controller层即创建Activity，Controller层起到了连接View层和Model层的作用，负责协调界面与数据之间的交互。通过处理View层传递过来的事件，Controller层可以响应用户的操作，并将必要的数据处理任务交给Model层来完成。这样可以实现界面和数据的分离，同时提高代码的可维护性和复用性。

Controller实现程序如下：

```
public class ControllerActivity extends Activity {
    private TextViewmShowTemp;
    private Button mUpTemp;
    private ACModelmACModel;
    @Override
    protected void onCreate(Bundle savedInstanceS-
tate) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_controller);
    mShowTemp = findViewById(R.id.show_tempera-
ture);
    mUpTemp = findViewById(R.id.up_temperature);
    mACModel = new ACModel();
        //接收View的事件
    mUpTemp.setOnClickListener(new View.OnClick-
Listener() {
            @Override
            public void onClick(View view) {
              //通知Model层处理数据
    mACModel.upTemperature(ControllerActivity.this);
            }
        });
    }
    public void upTemperature(String mTemp) {
    mShowTemp.setText(mTemp + "");
    }
}
```

2 MVP设计模式

模型－视图－表示器（Model－View－Presenter，MVP）设计模式逻辑如图2所示。

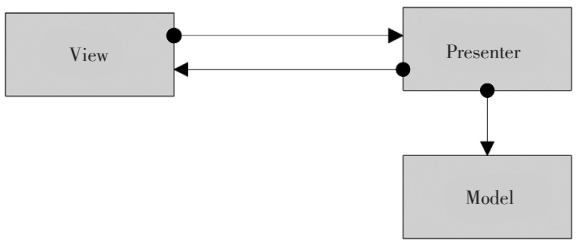

图2 MVP设计模式逻辑[9]

Model作为模型层，负责数据的加载、存储和处理，其角色和作用与MVC中的Model层是相同的。

View作为视图层，负责展示界面的数据并与用户进行交互。它接收用户的请求，并将请求传递给Presenter进行处理。View的角色和作用与MVC中的View层是相对应的。

Presenter负责逻辑业务的处理，接收来自View的

用户请求，并在需要的情况下修改Model，然后通知View进行界面的更新显示。与MVC中的Controller相比，Presenter的角色和职责有所区别。在MVP架构中，Presenter负责处理业务逻辑，并起到连接View和Model的桥梁作用。

2.1 MVP优点

（1）View和Model的分离

在MVP架构中，View负责处理用户界面的展示和用户输入的响应，而Model负责处理数据的获取和处理。二者之间的分离使得修改视图不会对模型造成影响，也使得View可以进行组件化，提高了代码的可维护性和可重用性。

（2）Presenter的复用和任务细分

Presenter负责处理业务逻辑和交互逻辑，所有的交互都发生在Presenter中。通过将复杂的任务分解成细小的任务，提高代码的可读性和可维护性。而且一个Presenter可以被多个View共享，实现了Presenter的复用，减少了代码的冗余。

（3）接口化的交互

Presenter通过接口和View进行交互，通过定义接口来规范交互方式，有利于测试和维护。通过接口的使用，可以实现模块之间的解耦，方便进行单元测试和模块替换。

（4）通用Presenter和HTTP请求

由于Model和Presenter的设计模式，可以定义多种通用的Presenter，用于处理不同的业务逻辑和请求。比如可以定义通用的Presenter用于处理HTTP网络请求，并通过即时的HTTP反馈进行界面设置，提高用户体验。这也是MVP架构的灵活性之一。

2.2 MVP缺点

（1）接口维护成本增加

在页面逻辑复杂的情况下，随着功能的增加，对应的Presenter和View的接口也会增多。这可能增加了代码的复杂性和维护成本。可以考虑根据实际情况对接口进行合理的设计和划分，避免接口过于庞大和冗余。

（2）状态保存与恢复

在Android中，一般使用OnSaveInstanceState和OnRestoreInstanceState来保存和恢复Activity的状态。然而，在MVP设计模式中，View不应该直接操作Model，这可能导致状态的保存和恢复变得不合理，并增加了Model和View之间的耦合性。可以考虑使用其他方式来实现状态的保存和恢复，如使用View Model层来管理状态。

（3）UI更改导致Presenter接口变更

当UI发生更改时，可能需要对Presenter中的一些接口进行修改，存在一定的耦合。这是因为Presenter负责处理业务逻辑和更新UI。可以考虑使用抽象和接口来降低耦合性，并且尽量将UI变更的影响范围控制在最小范围内。

2.3 MVP设计模式在车机上应用结论

（1）MVP不适合应用于处理CAN网络相关的业务及UI逻辑。

（2）MVP适合应用于处理HTTP网络相关的业务及UI逻辑。

2.4 MVP实现步骤

（1）Model层

Model层负责处理数据的获取、存储和处理，可以包含数据库操作、网络请求、文件读写等逻辑，并且Model只与Presenter发生交互。

Model实现程序如下：

```
public class ACModel {
   private intmTemperature = 0;
   public void up(ACModelCallback callback) {
callback.onSuccess(++ mTemperature);//通 知 Presenter结果
}
    public interface ACModelCallback {//数据回调接口
       void onSuccess(intmTemperature);
       void onFailed(String text);
    }
}
```

（2）View层

View层xml布局文件和MVC中View层的xml布局文件是相同的。

View实现程序如下：

```
public interface IView {
   void upTemperature(String text);
}
```

Activity达到接收到View的事件，通过实现View的接口[13]，通过持有Presenter的引用，将View和Presenter进行联系。

View实现程序如下：

```
public class PresenterActivity extends Activity implements Interface.IView {
```

```
    private TextViewmShowTemp;
    private Button mUpTemp;
    private Interface.IPresentmPresent;
    @Override
     protected void onCreate(Bundle savedInstanceS-
tate) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_controller);

    mShowTemp = findViewById(R.id.show_tempera-
ture);
    mUpTemp = findViewById(R.id.up_temperature);
    mPresent= new UpTemperaturePresenter(this);
    mUpTemp.setOnClickListener(new View.OnClick-
Listener() {
          @Override
          public void onClick(View view) {
    mPresent.up();
          }
        });
    }
    public void upTemperature(String mTemp) {
    mShowTemp.setText(mTemp + "");
    }
    }
```

（3）Presenter层

Presenter负责处理业务逻辑，通过IPresenter接口实现View调用，它可以定义一系列的方法，用于接收View层的事件和数据请求，并根据业务逻辑进行处理，以实现两者之间的交互。

Presenter实现程序如下：

```
    public interface IPresent {
      void up();
    }
    public class UpTemperaturePresenter implements
ACModel.ACModelCallback, Interface.IPresent {
      private ACModelmACModel;
      private Interface.IViewmView;
       public UpTemperaturePresenter(Interface.IView
view) {
    mView = view;
    mACModel = new ACModel();
      }
      @Override
      public void onSuccess(intmTemperature) {
    mView.upTemperature(mTemperature + "");
      }
      @Override
      public void onFailed(String text) {
    mView.upTemperature("失败");
      }
      @Override
      public void up() {
    mACModel.up(this);
      }
```

3 MVVM设计模式

模型-视图-视图模型（Model-View-View Model，MVVM）如图3所示[3,14]。

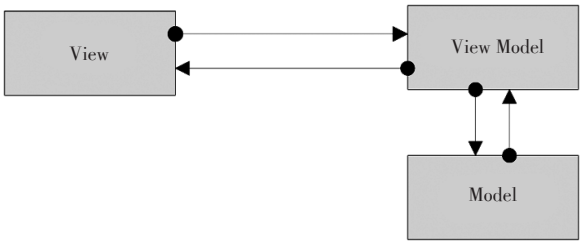

图3 MVVM设计模式逻辑[3]

MVVM模式可以理解为对MVC模式的改进和优化。在MVVM模式中，Model负责处理数据的加载和存储，与MVC模式中的Model层设计策略相同。当Model数据更新后，它会将新数据传递给View Model。

View是视图层，负责展示界面数据并与用户进行交互，与View Model之间存在双向交互关系。View层通过绑定View Model来实现，当ViewModel的数据改变时，View会自动更新相应的UI，反之亦然。

View Model是视图模型，负责完成View与Model的交互，处理业务逻辑，并通知Model进行更新操作。

3.1 MVVM的优点

（1）View Model和View的耦合度相比MVP模式更低。View Model负责处理和提供数据，UI的变化无需特殊处理，只需通过数据绑定实现。这样做，只需关注数据处理，UI处理也就自然完成了。

（2）View Model中只包含数据和业务逻辑，方便进行单元测试。

（3）由于车机CAN网络的消息经常发生并且不需要向对应的ECU发送请求，同时也会有来自对应ECU的反馈消息。因此，MVVM模式适合用于IVI车机端

对CAN网络的请求，实现通过数据驱动UI的修改。

3.2 MVVM设计模式在车机上应用结论

尽管使用Data Binding机制时，View层的Data Binding需要按规范实现，否则可能导致View布局问题以及与Activity中代码相关的问题，并且可能不利于调试，需要一定的开发经验。但是，由于MVVM模式能很好地适用于处理与CAN网络相关的复杂业务和UI逻辑，因此，在基于Android系统的车机应用中访问CAN网络时，MVVM是一种较为理想的设计模式。

3.3 MVVM实现步骤

（1）Model层

Model层在MVC、MVP和MVVM设计模式中都是负责数据的加载和处理。在这些设计模式中，Model层的职责是从数据源获取数据，并进行数据的处理和逻辑操作。

Model实现程序如下：

```
public class ACModel {
   private intmTemperature = 0;
   public void up(ACModelCallback callback) {
callback.onSuccess(++mTemperature);
    public interface ACModelCallback {//数据回调接
       void onSuccess(intmTemperature);
       void onFailed(String text);
   }
}
```

（2）View层

此布局与其它设计模式中的view层的不同之处在于，增加了Data Binding。在使用Data Binding的情况下，布局文件中的视图和数据源可以直接进行绑定，使得数据更新和显示更加自动化和简洁。通过在布局文件中使用特定的语法和属性，可以将数据源与对应的视图进行关联，当数据源发生变化时，视图会自动更新。

View实现程序如下：

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
<data>
<variable
       name="atcViewModel"
       type="com.faw.mvvm.AtcViewModel"/>
</data>
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
    android:id="@+id/show_temperature"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="temperature"
    android:text="@{atcViewModel.temperature}"
    android:layout_centerInParent="true"/>
    <Button
    android:id="@+id/up_temperature"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/show_temperature"
    android:onClick="@{(v) ->atcViewModel.onClick -Up(v)}"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:text="up"/>
    </RelativeLayout>
    </layout>
```

MVVM Activity作用为将View和View Model进行绑定。

MVVM实现程序如下：

```
public class MvvmActivity extends Activity {
   @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    AtcViewModelatcViewModel = new AtcViewModel();
    ActivityMvvmBinding binding = DataBindingUtil.setContentView(this, R.layout.activity_mvvm);
        //View与ViewModel绑定
    binding.setAtcViewModel(atcViewModel);
   }
}
```

（3）View Model层

View Model属于连接视图层和模型层的中间件，能够观察到绑定的数据的变化，并对视图内容做对应的更新，能够监听视图的变化，且通知数据发生改变。

View Model实现程序如下：

```
public class AtcViewModel extends BaseObservable {
    private String temperature;
    private final ACModelmACModel;
    public AtcViewModel() {
mACModel = new ACModel();
    }
    @Bindable
    public String getTemperature() {
        return temperature;
    }
    @Bindable
    public void setTemperature(String temp) {
this.temperature = temp;
notifyChange();
    }
    public void onClickUp(View view) {//点击事件处理
mACModel.up(new ACModel.ACModelCallback() {
        @Override
        public void onSuccess(int temp) {
setTemperature(temp + "");
        }
      @Override
        public void onFailed(String text) {
setTemperature(text);
        }
      });
    }
}
```

4 结束语

在此项研究中，对MVC、MVP、MVVM这3种车载Android应用设计模式进行了逻辑分析，并对其优缺点进行了比较。最后，总结了在车载终端应用中的实现步骤和结果。然而，结合实际业务的实现中，还需要进行更多的设计工作来进行优化和细化。

Android车载应用提供了强大的开发工具，能够优化用户交互，并推动了车辆智能化发展。随着技术逐渐进步，更多交互、更好体验的Android应用将会在车载终端应用。

参 考 文 献

[1] 马鑫. 基于Android车载娱乐系统的iPod功能实现[D]. 南京：东南大学, 2013: 3.

[2] 唐朝霞. 基于Web的高校院(系)资料室个性化信息服务体系研究[J]. 湖南科技学院学报, 2008, 29(8): 3-4.

[3] 锁玺. 基于REST的基层社区健康平台的设计与实现[D]. 西安：西安电子科技大学, 2015.

[4] 张志强, 刘巧玲. 基于J2EE的软件虚拟实训系统的MVC架构实现[J]. 中州大学学报, 2009, 26(1): 4-5.

[5] 王一锋. 基于Ruby on Rails和Flex3的RIA研究与应用[D]. 贵阳：贵州大学, 2009.

[6] 周康毅. 基于安卓平台的教学助手系统的设计与实现[D]. 武汉：华中师范大学, 2018.

[7] 张沈梅, 孙昊, 王玲, 等. 基于微信小程序的课程在线测试系统[J]. 电脑知识与技术：学术版, 2018(11Z): 4-5.

[8] 潘昊. 基于MVP模式的用户界面层的研究与实现[D]. 南京：东南大学, 2012.

[9] 崔绍龙, 彭玲, 李森, 等. 室内外一体化的消防设施巡检系统构建[J]. 消防科学与技术, 2017, 36(9): 2-3.

[10] 汪润泽. 车间刀具全生命周期管理系统设计与实现[D]. 武汉：华中科技大学，2015.

[11] 王汉肖. 手机购物方案生成系统的设计与实现[D]. 济南：山东大学, 2015.

（责任编辑　**姜明慧**）

【作者简介】

黄慧，工学学士学位，现就职于中国第一汽车股份有限公司研发总院智能网联开发院，主要研究方向为软件架构智能座舱。

E-mail：huanghui@faw.com.cn

李荆轩：工学硕士学位，现就职于一汽-大众技术开发部，主要研究方向为智能座舱。

E-mail：jingxuan.li@faw-vw.com