文章编号:1671-4598(2019)08-0120-05 DOI:10.16526/j.cnki.11—4762/tp.2019.08.025 中图分类号:TP391.8 文献标识码:A

基于 Android 的车胎监测系统软件的设计与实现

刘马飞
(无锡职业技术学院 物联网技术学院,江苏 无锡 214121)

摘要:目前车胎监测系统中,车胎内传感器实时监测胎温胎压,通过无线发送到无线接收端,无线接收端通过 CAN 总线传送到仪表盘区的专用液晶屏进行展示;由于基于 Android 的中控大屏由于具有丰富的娱乐功能、方便与手机和网络互联的优点,已经成为高端新能源客车的流行配置;通过设计转换模块将车胎监测无线接收端 CAN 接口设备转成 USB 串口,接到安卓中控大屏进行展示,并设计 Android 应用程序进行车胎监测信息展示,可自行设定报警阈值,车胎监测信息展示更直观便捷;复用中控大屏进行车胎信息展示,节省了专用液晶屏,从而降低系统成本和方便司机操作,经实际应用满足了实时车胎监测的功能。

关键词:车胎监测;Android;USB

Design and Implementation of Tire Monitoring System Software Based on Android

Liu Mafei
(School of Internet of Things Technology, Wuxi Institute of Technology, Wuxi 214121, China)

Abstract: In the current tire monitoring system, the tire temperature and pressure are monitored in real time by the sensor inside the tire. The sensor is sent to the wireless receiver, and the wireless receiver is transmitted to the special LCD screen in the dashboard area for display. Because of its rich entertainment function, convenient connection with mobile phones and networks, Android system has become a popular configuration for high-end new energy buses. Through the design of conversion module, the CAN interface device of tire monitoring wireless receiver is converted to USB serial port, which is displayed on Android system, and Android application program is designed to display tire monitoring information. The alarm threshold can be set by oneself, and tire monitoring information display is more intuitive and convenient, then save the special LCD screen and reduce the system cost. The function of real-time tire monitoring has been proved through practical application.

Keywords: tire monitoring; Android; USB

0 引言

目前车胎监测系统实现方案中,车胎内传感器实时监测胎温胎压信息,通过无线发送到无线接收端,无线接收端通常通过 CAN 总线传送到仪表盘的专用的液晶显示屏进行显示,当超过相应阈值,在液晶显示屏进行告警提醒并配合声音告警[1-4]。在目前的车胎监测实现方案中存在以下两点不足:1)车胎告警阈值由硬件设定导致使用过程中不能动态调整告警阈值;2)液晶显示屏通常只显示车胎状态异常,并不显示实际胎温和胎压数值,因此告警显示不够具体、直观,不便于驾驶员对车胎异常情况的精确判断。

由于基于 Android 的中控大屏由于具有丰富的娱乐功能、方便与手机和网络互联的优点,已经成为高端新能源客车的流行配置。在配置 Android 大屏的客车上,司机需要同时关注 Android 中控大屏和车胎监测信息显示液晶屏,造成操作不方便。此外通过 Android 大屏进行胎温胎压的显示,可以更直观显示精确的胎温胎压数值,同时告警阈值为软件设定值,可以动态地进行告警阈值调整,使得驾驶员可以根据季节不同来选择合适的告警阈值。因此有必要通过设计转换模块,并设计和实现相应的 Android 端应用软件,将车胎监测信息接到 Android 中控大屏进行显示,从而省去专用液晶屏,降低系统成本和方便司机操作,且可自行设定报警阈值,车胎监测信息展示更直观便捷。

本文介绍车胎监测 Android 端应用软件的设计和实现,其综合利用 startService 和 bindService,实现程序界面前台运行时,实时接收车胎监测报文并显示胎温胎压实时值,若超过告警阈值,则相应位置的车胎显示异常告警;当程序界面后台运行时,后台 Service 持续接收车胎监测报文,当超过相应阈值,调出程序界面进行显示和告警,经实际应用满足了实时车胎监测的功能。

1 系统硬件结构及原理

系统硬件结构框图如图 1 所示。上半部分,轮胎模块和 TPMS 节点模块采用目前市场主流的商用车胎监测系统。

收稿日期:2019-02-21; **修回日期**:2019-03-15。
基金项目:江苏省高校品牌专业项目(PPZY2015C240)。
作者简介:刘马飞(1984-),男,广西北流人,博士,讲师,主要从事物联网和嵌入式智能终端产品设计的教学与科研风险抵押金方向的研究。

由于车胎监测系统通常接入 CAN 总线，而 Android 中控大屏通常使用 USB 通信接口，因此需要设计相应转换模块。考虑到 TTL 转 USB 串口（CH340 芯片）的模块较为通用，因此设计 CAN 接口转 TTL 的模块和通用的 TTL 转 USB 模块相连接，完成 CAN 总线接口和 USB 接口的转换。Android 中控大屏支持 USB OTG 接口，其作为 USB 主设备与 USB 从设备进行 USB 串口通信。车胎监测系统通过转换模块转成 USB 接口接到 Android 系统。

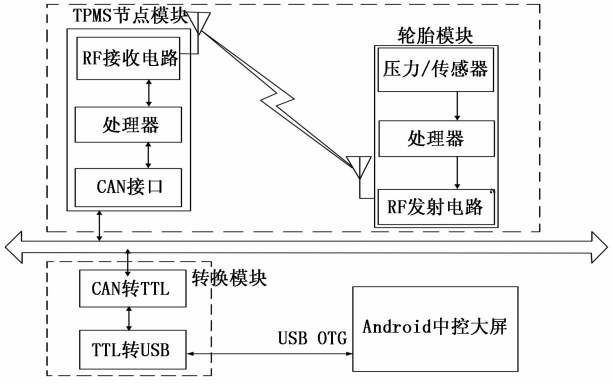

图 1　系统硬件结构图

系统的运行原理如下：在车辆行驶时，车轮处于转动状态，安装在车轮内的监测传感器将被触发，并每隔 100 ms通过无线方式发送车胎监测数据。TPMS 节点无线接收模块收到无线上报的监测数据，并通过 CAN 总线接口发送到 CAN 总线。利用自行设计的转换模块，从 CAN 总线接收监测数据，并转变 USB 串口数据向上发送给 Android 中控大屏。车胎监测系统需要设计 Android 应用程序，并安装部署在中控大屏上。该 Android 应用程序将通过 USB 串口接收传感器周期上报的每个车胎的胎温和胎压数据报文。每个数据报文固定 10 个字节，格式如图 2 所示。

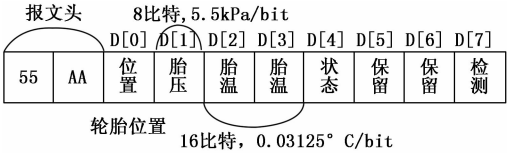

图 2　胎温胎压监测报文格式

程序根据报文计算胎温 t 和胎压 p，计算公式如下（$D[i]$ 表示车胎监测报文除去报文头部分中下标为 i 的字节的 8 位无符号整数值）：

$$t(Kpa) = D[1] * 5.5 \quad (1)$$

$$p(℃) = (D[3] * 256 + D[2]) * 0.03125 - 273 \quad (2)$$

系统软件定义胎温上限阈值 t_H、胎压下限阈值 p_L 和胎压上限阈值 p_H。当 $t >= t_H$ 或 $p <= p_L$ 或 $p >= p_H$ 任一条件满足时，表示车胎状态异常，程序产生告警提醒驾驶员。

2　软件总体模块设计

Android 中控大屏的车胎监测应用程序需要持续接收上报的 USB 串口数据报文进行胎温和胎压的数据展示，当胎温或胎压超过指定阈值时，进行相应的告警，同时告警阈值可进行动态调整。本文设计的车胎监测应用程序总体架构如图 3 所示。

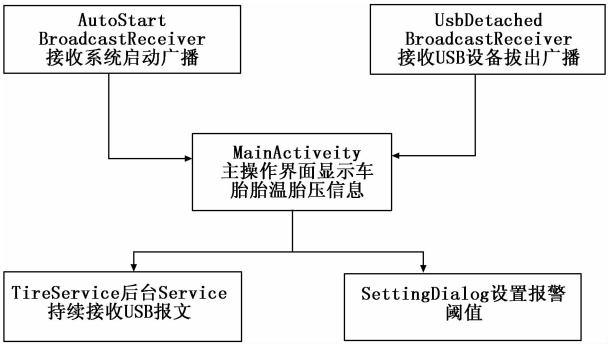

图 3　胎压监测程序架构

MainActivity 是主操作界面，用来实时展示传感器上报的车辆所有车胎的胎温胎压信息。为了满足主操作界面后台运行时亦能持续接收胎压监测报文，通过 TireService 后台服务进行 USB－OTG 串口数据接收和胎压监测报文解析操作，当超出相应胎温和胎压阈值时调出程序主界面进行告警显示。胎温和胎压阈值可手动进行调节，通过点击设置按钮弹出对话框 SettingDialog 进行胎温和胎压阈值设置。

AutoStartBroadcastReceiver 用于实现当 Android 开机启动完成后，自动打开本监测程序进行接收胎压胎温报文信息。当车胎监测硬件断开连接后，监测程序应当结束运行，通过 UsbDetachedBroadcastReceiver 实现。

3　软件详细实现

3.1　程序运行流程

程序运行流程如图 4 所示。系统开机和转换模块插入 Android 系统后，跳转到本程序运行。程序运行后，启动后台服务 TireService，TireService 创建 USB 串口数据接收线程 ReadThread，在接收线程中打开 USB 串口设备，并循环接收 USB 串口车胎监测报文，通过 Callback 回调机制更新到 Activity 界面。

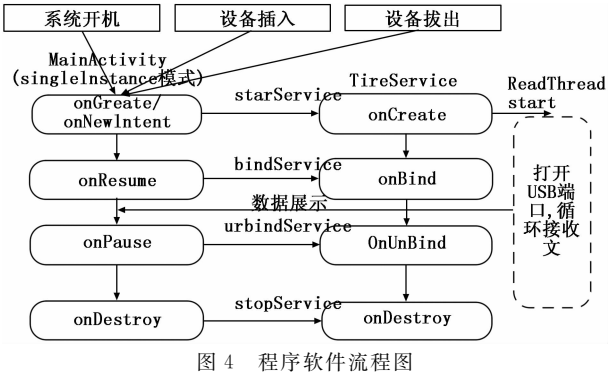

图 4　程序软件流程图

3.2　USB 串口通信编程实现

Android 系统使用 USB Host API 与 USB 接口的转换设备进行通信。主要用到的 API 类包括 UsbManager 类（用来管理 USB 设备的类）、UsbDevice 类（代表一个 USB 设

备的类）、UsbInterface 类（表示 USB 设备的一个接口）、UsbEndpoint 类（表示一个接口的某个节点）以及 UsbConnection 类（用连接进行发送和接收数据）[5-6]。

与一般 USB 设备不同，USB 串口设备还需要配置串口传输的波特率、数据位、停止位和奇偶校验位等信息[7-8]。本文设计的转换模块采用的 TTL 和 USB 转换芯片为 CH340 芯片，网络上可以搜索到该芯片的免驱动 Android USB Host 模式的参考代码，在实现 USB 串口通信代码编程时，可参照进行剪裁。

3.3 程序启动退出代码实现

1）系统开机启动

AutoStartBroadcastReceiver 采用静态注册接收 BOOT_COMPLETED，重写 OnReceive（）方法，并在该方法中通过 startActivity 启动程序[9]。其在 AndroidManifest. xml 设置为：

```
<receiver android:name=". AutoStartBroadcastReceiver">
<intent-filter>
<action android:name="android. intent. action. BOOT_COMPLETED" />
<category android:name="android. intent. category. HOME" />
</intent-filter>
</receiver>
```

AutoStartBroadcastReceiver 实现如下：

```
public class AutoStartBroadcastReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
String action = "android. intent. action. MAIN";
String category = "android. intent. category. LAUNCHER";
Intent myIntent = new Intent(context, MainActivity. class);
myIntent. setAction(action);
myIntent. addCategory(category); myIntent. addFlags(Intent. FLAG_ACTIVITY_NEW_TASK);
context. startActivity(myIntent);
}
}
```

2）设备插入启动

程序主界面 MainActivity 采用单例模式 singleInstance，并监听转换模块插入动作 USB_DEVICE_ATTACHED，并对 USB 插入动作设置过滤条件为对应转换模块的 PID 和 VID[5]。主界面在 AndroidManifest. xml 中设置如下：

```
<activity
android:name=". MainActivity" android:launchMode="singleInstance">
<intent-filter>
<action android:name="android. intent. action. MAIN" />
<category android:name="android. intent. category. LAUNCHER" />
</intent-filter>
<intent-filter>
<action android:name="android. hardware. usb. action. USB_DEVICE_ATTACHED" />
</intent-filter>
<meta-data android:name="android. hardware. usb. action. USB_DEVICE_ATTACHED"
android:resource="@xml/device_filter" />
</activity>
```

其中 device_filter. xml 文件定义了转换模块 USB 设备的 product-id 和 vendor-id。

```
<? xml version="1. 0" encoding="utf-8"? >
<resource>
<usb-device product-id="29987" vendor-id="6790" />
</resource>
```

3）设备拔出退出

当车胎监测 USB 设备从安卓拔出时，监测程序程序需要停止程序运行。因此在程序正常启动后，通过代码动态注册广播接收器 usbDetachedBroadcastReceiver 监听 USB 设备拔出动作 USB_DEVICE_DETACHED。

```
usbDetachedBroadcastReceiver=new UsbDetachedBroadcastReceiver();
intentFilter = new IntentFilter();
intentFilter. addAction("android. hardware. usb. action. USB_DEVICE_DETACHED");
registerReceiver(usbDetachedBroadcastReceiver,intentFilter);
```

usbDetachedBroadcastReceiver 在代码实现时，重写 OnReceive（）方法，判断为转换模块拔出，则设置退出标记。

```
public void onReceive(Context context, Intent intent) {
UsbDevice device = (UsbDevice) intent. getParcelableExtra(UsbManager. EXTRA_DEVICE);//获取当前拔出的 USB 设备
if (device ! = null)
{if(device. getProductId()==29987&&device. getVendorId()==6790) {//判断拔出设备为车胎监测转换模块,设置退出标记
Intent myIntent = new Intent(context, MainActivity. class);
myIntent. putExtra("IsUsbDetached",true);
context. startActivity(myIntent);
}
}
}
```

在 MainActivity 中重写 onNewIntent（）方法，判断退出标志为真，调用 finish 方法结束程序[10]。

```
protected void onNewIntent(Intent intent) {
boolean IsUsbDetached=
intent. getBooleanExtra("IsUsbDetached",false);
if(IsUsbDetached)
{
//判断已设置退出标记,则弹出提示框提示设备断开,并结束程序运行
}
super. onNewIntent(intent);
```

```
}
```

3.4 后台服务运行

为了保证程序前台运行与否，后台服务均需要持续接收车胎监测报文，采取了下列措施：

1）后台服务 Service 有两种启动方式 startService 和 bindService。bindService 启动服务后，通过 unbindService 结束服务，其启动的服务与 Activity 是捆绑关系，当 Activity 退出后，Service 也会退出。Activity 可与绑定服务通过回调接口的方式，方便快速进行数据交互。

startService 启动服务后，服务与 Activity 是分离的，当 Activity 退出后，服务继续存在，只有调用 stopService，后台 Service 才会结束，但 startService 启动的服务通常只能通过广播的方式将数据发回界面进行显示。

这 2 种模式不是完全分离的，可以同时使用。本程序 Activity 在 OnCreate 阶段，调用 startService 启动车胎监测服务，在 onResume 阶段通过 bindService 进行服务绑定；在 onPause 阶段调用 unbindService 进行服务解绑，在 onDestroy 阶段方调用 stopService 进行结束服务操作。从而达到程序启动后，后台服务便启动持续接收 USB 车胎监测报文，当程序在前台运行时，又可通过绑定服务，简便实现与 Activity 的双向数据交互。

在 bindService 进行服务绑定时使用服务 TireService 内定义的到 Binder 对象，MainActivity 通过 Binder 对象可以将当前设定的告警阈值以及当前是否处于报警暂停状态传递到服务 TireService。

```
public class Binder extends android.os.Binder{
//界面传递报警阈值到 Service
public void Sethreshold(float twh,float tpl,float tph)
{
TireService.this.tiretempth=twh;
TireService.this.tirepresstl=tpl;
TireService.this.tirepressth=tph;
}
public void SetShowAlarm(boolean isshowalarm)
{
bshowAlarm=isshowalarm;
}
public TireService getService()
{
return TireService.this;
}
}
```

在 TireService 内需要定义回调接口，供服务中接收到 USB 车胎监测报文后，通过回调接口将数据更新发回界面进行显示。

```
private Callback callback=null;
public void setCallback(Callback callback) {
this.callback = callback;
}
public Callback getCallback() {
return callback;
}
public static interface Callback{
void OnSerialData(tireparam data);
}
```

主界面 MainActivity 需要实现 ServiceConnection 接口，实现 onServiceConnected 和 onServiceDisconnected 方法。

```
public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
isBinded=true;
binder = (TireService.Binder) iBinder; binder.Sethreshold(tiretempth,tirepresstl,tirepressth);
binder.getService().setCallback(new TireService.Callback() {
@Override
public void OnSerialData(tireparam data) {
//tvout.setText(data);
Message msg=new Message();
msg.obj=data;
msg.what = IS_PktArr;
handler.sendMessage(msg);
}
});
}
@Override
public void onServiceDisconnected(ComponentName componentName) {
isBinded=false;
}
```

2）设置 Activity 为 singleInstance 模式，并重写 onKeyDown 函数对返回按键 KEYCODE_BACK 的处理，通过调用 moveTaskToBack 方法将 Activity 移动到后台运行，避免 Activity 被销毁。

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
if (keyCode == KeyEvent.KEYCODE_BACK) {//点击返回按键
if(bIsExit)
{//如果 USB 设备拔出,则结束 Activity 和 Service
finish();
Intent intent= new Intent(this, TireService.class);
stopService(intent);
return true;
}
moveTaskToBack(true);
return true;
}
return super.onKeyDown(keyCode, event);
}
```

3）为了避免 TireService 因为资源不足被系统销毁或在任务管理器中被用户强行关闭导致无法持续接收 USB 车胎监测报文，采用了关闭重启的策略。在服务的 onStartCommand 中设置返回 START_FLAG_REDELIVERY 标记，

代表 service 被 kill 后重新启动，并携带上次启动参数输入参数 intent。

```
public int onStartCommand(Intent intent, int flags, int startId)
{
flags =flags| Service.START_FLAG_REDELIVERY;
return super.onStartCommand(intent, flags, startId);
}
```

4） TireService 服务启动后，启动读取线程 ReadThread。为了 TireService 被强行关闭重启后仍能持续接收 USB 串口车胎监测数据报文，因此将 USB 设备打开操作在读取线程中。

读取线程 ReadThread 工作流程如图 5 所示，首先枚举查找 USB 转换模块设备，检查程序是否被授予操作 USB 转换模块设备的权限，如果未授权该 USB 设备的操作权限，则通过 PendingIntent 申请 USB 操作权限。然后打开 USB 设备并循环接收 USB 车胎监测报文，收到报文后获得车胎位置以及相应胎温胎压信息。当程序界面 MainActivity 处在前台运行时，通过 Callback 回调机制发回程序界面进行显示，当超过阈值进行相应报警。当程序界面处在后台运行时，如果胎温胎压超过相应阈值，则通过 startActivity 将程序界面转到前台运行。

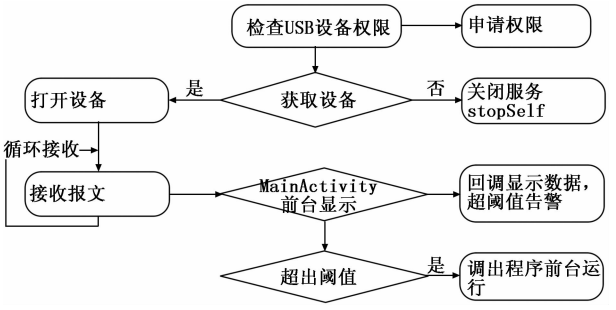

图 5 ReadThread 工作流程

5）告警阈值处理和告警暂停

告警阈值需要通过 SettingDialog 进行调整，该对话框通过 Callback 机制将阈值设置数值返回主界面。

当用户在设置对话框修改了告警阈值时，告警阈值将更新到主界面 MainActivity 中，并通过绑定服务时返回的 Binder 对象传递到后台服务 TireService 中。同时为了程序被强行关闭时，也可获得正确的告警阈值，在告警阈值更改的同时，将利用 SharedPreference 保存告警阈值进行持久化存储。在 TireService 服务启动 onStartCommand 阶段，判断是否为 MainActivity 正常启动 TireService，还是因此程序被强行关闭而对服务的重启。当服务启动为程序被强行关闭而对服务的重启时，需要在持久化存储中读取保存的告警阈值，否则重启服务时告警阈值为局部变量的默认初始值 0。

为了避免持续收到告警车胎监测报文时，持续调出程序主界面导致无用转到其他程序执行的情况，因此在告警显示时，将出现告警暂停操作按钮，点击该按钮后，在 5 分钟内即使收到胎温或胎压超过阈值的车胎监测报文，也不会显示告警情况。当告警暂停定时结束后，程序即又回到接收到异常车胎监测报文，便进行正常告警的状态。

4 系统结果测试

将车胎监测系统安装在新能源客车上，大客车配备 Android 显示屏，通过转换模块将车胎监测数据接到 Android 系统，并通过本软件进行展示。由于实际部署在车辆上时，不便连接调试和查看调试信息，因此采用如图 6 测试模型，Ginkgo USB－CAN 总线适配器连接到 PC 上作为 CAN 数据发送源，PC 机通过使用 Ginkgo USB－CAN 配套软件按照车胎协议报文格式按照周期 100 ms 不间断发送车胎监测报文。作为 CAN 数据源的 Ginkgo USB－CAN 总线适配器，与车胎监测转换模块连接，从而完成 CAN 接口报文转换成 USB 串口数据包接入到 Android 系统。

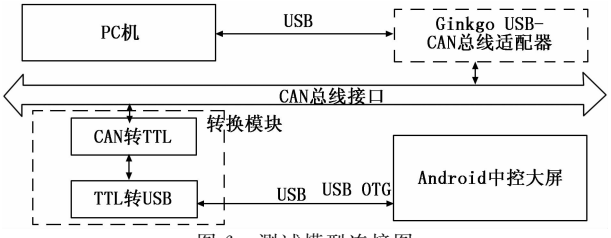

图 6 测试模型连接图

当 Android 系统进行上电启动，系统启动后自动运行本应用程序主界面 MainActivity 显示车胎监测胎温胎压信息。在测试告警时采用发送相同的车胎监测数据报文，通过阈值设置对话框调节胎压告警下限阈值超出该车胎特定监测报文的胎压数值，则进行显示告警。当点击返回键后，MainActivity 退到后台运行，并通过后台 Service 持续接收并解析 USB 串口车胎监测报文，当胎温或胎压超过指定阈值，则将 MainActivity 调到前台进行显示。当从通过任务管理将程序强行关闭时，后台 Service 将自动重启并接收解析 USB 串口车胎监测信息，当胎温或胎压超过指定阈值，则调出程序界面 MainActivity 进行显示。

5 结束语

本文设计现实了一个利用 Android 端进行信息展示的车胎监测系统，系统实时接收 USB 接口上报的车胎监测报文，当超过相应阈值则进行告警显示。通过利用综合利用 startService 和 bindService 两种类型的后台 Service、利用 moveTaskToBack 将程序移到后台运行、通过服务关闭重启来确保后台 Service 持续接收 USB 接口上报的车胎监测报文，完成实时监测的功能，经过实际部署应用，满足的实时车胎监测的需要。

参考文献：

[1] 彭何欢，郑红平，麻则运. 基于 CAN 总线与无线传感器的轿车胎压监测系统 [J]. 制造业自动化，2011，33 (3)：24－25，43.

（下转第 129 页）