文章编号:1671-4598(2021)06-0147-06 DOI:10.16526/j.cnki.11—4762/tp.2021.06.030 中图分类号:TP301 文献标识码:A

基于华为统一扫码服务编程实现装备信息的交互查询

杜国祥，刘 洋，杨小龙，王 君
（中国人民解放军32382部队，武汉 430311）

摘要：为实现装备信息的精确化、智能化查询使用管理，研究利用华为统一扫码服务技术，通过扫描装备部组件及其对外接口连接线缆的二维码标签以完成身份识别，进而自动从装备信息数据库中查询与之相对应的技术信息，并以文字、图片、表格等形式将信息展示给用户，此种使用方式更加契合现今用户使用习惯，更易激发用户兴趣，在装备信息使用管理、技术培训、维修保障等方面具有极大的军事使用价值。

关键词：华为统一扫码服务；关键字搜索；LitePal；barteksc android—pdf—viewer；Glide；smartTable

Interactive Query of Equipment Information Based on HMS Core Scankit Programming

Du Guoxiang，Liu Yang，Yang Xiaolong，Wang Jun
（PLA Unit 32382，Wuhan 430311，China）

Abstract：In order to realize the accurate and intelligent query and use management of equipment information，Huawei unified code scanning service technology is studied and used，and the identification is completed by scanning the two—dimensional code labels of equipment components and their external interface connecting cables，and then the corresponding technical information is automatically queried from the equipment information database，and displayed to users in the form of words，pictures and tables. This usage mode is more suitable for today's users' usage habits，and it is easier to stimulate users' interest. It has great advantages in equipment information use management，technical training，maintenance support and so on.

Keywords：Huawei HMS core Scankit；keyword search；LitePal；barteksc android—pdf—viewer；Glide；smartTable

0 引言

为适应当今复杂多变的国际环境变化，各类高新技术在新型装备研制中得到越来越多的应用，在极大提升装备技战性能的同时，也导致装备变得越来越复杂，传统的纸质随装技术资料存在体积与重量大、编辑与更新不及时、使用不便、易污染、防火性差等诸多弊端。交互式电子技术手册（IETM，interactive electronic technical manual）采用文字、图形、表格、音视频等形式，以人机交互的方式为用户提供装备相关技术信息[1]，但作为一种新型技术出版物，局限于与用户的交互，要求用户对装备有一定的了解，IETM未与装备产生直接关联。采用射频识别技术（RFID，radio frequency identification）设计备件物资保障系统和导弹武器管理系统，通过非接触双向通信，以完成装备、备件、物资的目标识别和数据交换，可实现备件物资和装备技术状态的精确化、智能化管理[2]，但RFID在使用过程中会产生射频信号，可能会对装备正常运行产生电磁干扰，多个RFID标签在进行识别时也易相互产生干扰。

本文将采用二维码标签对装备部组件及其对外接口连接线缆进行标识，用户通过运行于Android智能设备的客户端软件，扫码完成身份识别、解码后，自动触发业务逻辑处理，查询与此二维码标签标识对象所相关的技术信息，以此为用户提供各类信息服务，具有技术成熟、识别速度快、易于制作、成本低、开发周期短等特点，用户即扫即查，更加契合现今用户使用习惯，更易激发用户兴趣。

1 总体构想

按照装备配套表，为其所有部组件及其连接线缆统一制作、更换二维码铭牌，二维码参考格式为“A：加电控制组合”，或者“B：6S2208—7D X1”，其中A和B为特征标识字，可自行定义，主要用于区别部组件和部组件对外接口连接的线缆。本装备信息交互查询APP运行于常见的Android系统智能设备，软件调用设备的摄像头进行扫码，以解析出二维码中的特征标识字、部组件名称或部组件连接线缆代号，之后从数据库中查询出其相关数据信息，最后自动跳转至部组件详情页或部组件对外接口明细页将数

收稿日期:2020-10-20； 修回日期:2020-11-17。

作者简介:杜国祥(1979-)，男，湖北随州人，大学本科，工程师，主要从事装备维修保障技术方向的研究。

引用格式:杜国祥，刘 洋，杨小龙，等.基于华为统一扫码服务编程实现装备信息的交互查询[J].计算机测量与控制，2021，29(6)：147-152，168.

据信息展示给用户。

用户可以通过页面工具栏上面的搜索菜单项，输入关键字查询相关数据信息，也可以用作装备电子技术手册，浏览数据信息。

2 软件设计

2.1 软件总体设计思路

采用模型－视图－控制器（MVC，model－view－controller）的架构模式进行分层设计，如图 1 所示。用户在视图层进行交互操作，控制器层根据用户请求进行业务逻辑处理，从数据模型层查询相关数据信息，再回送视图层将查询结果显示给用户。

在数据模型层，装备信息分成两类存储在 assets 文件夹下，编译时打包至 APK 安装包内，第一类是装备的部组件清单、部组件接口明细清单、部组件连接线缆针脚定义明细等，此类信息数据量较小，利用开源数据库框架 LitePal 进行管理[3]。第二类是装备部组件的详细介绍、实物图、结构图、电路图、接线图，此类信息数据量较大，则以 PDF、JPEG 格式文件存储在 assets 文件下，利用 barteksc：android－pdf－viewer 开源库加载显示 PDF 文件[4]，利用 Glide 开源库加载显示 JPEG 文件[5]。

控制器层 4 个活动对象 MainActivity（首页）、EquipmentActivity（部组件清单页）、DetailsPageActivity（部组件详情页）、InterfaceDatasActivity（部组件接口明细页）相互间跳转关系如图 2 所示，分别对应视图层中的 activity_main. xml、activity_equipment. xml、activity_details_page. xml、activity_interface_datas. xml 四个布局页。

在视图层，首页列出装备的结构组成和功能组成供用户选择，用户点击选择后将自动跳转至部组件清单页。在部组件清单页利用 RecyclerView 控件展示用户选中的某结构组成或分系统所包含的部组件，RecyclerView 设置为 StaggeredGridLayoutManager 纵向 2 列瀑布流布局，条目则采用自定义的图片＋文字纵向布局，用户点击选中某一部组件后将自动跳转至部组件详情页。部组件详情页将为用户展示其详细相关技术信息，包括详细介绍、对外接口、实物图、结构图、电路图、接线图，采用 TabLayout 控件＋ViewPager 控件实现，ViewPager 控件包含 6 个页面，可通过左右滑动或者点击标签进行页面，其中详细介绍、实物图、结构图、电路图、接线图 4 个页面展示装备信息均为 PDF 格式文件，对外接口页则采用 smartTable 开源库加载显示数据表格，用户点击某对外接口后将自动跳转至部组件对外接口明细页。部组件对外接口明细页也采用 smartTable 开源库加载显示数据表格，展示该对外接口所连接线缆的针脚定义列表。

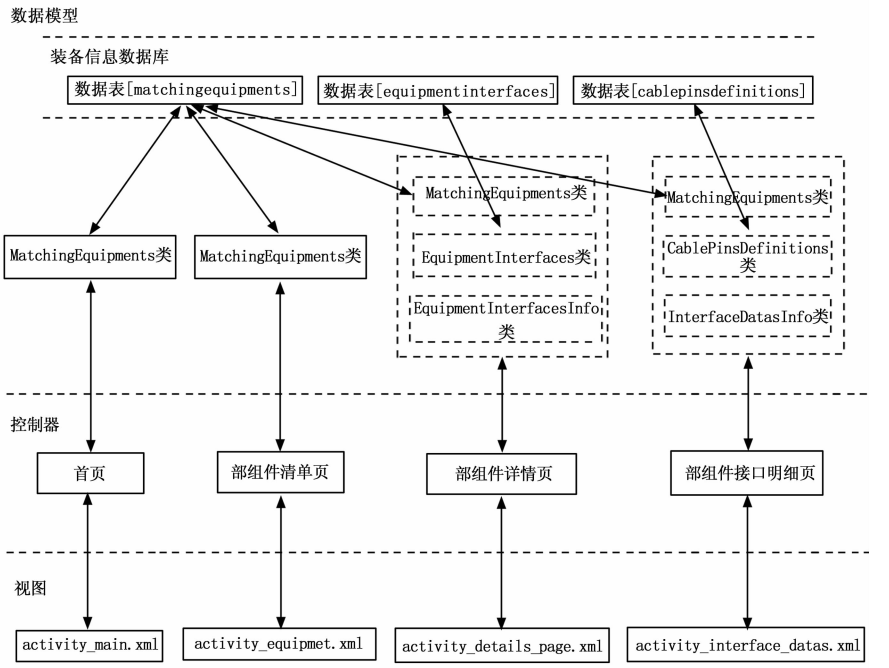

图 1 应用软件对象分解图

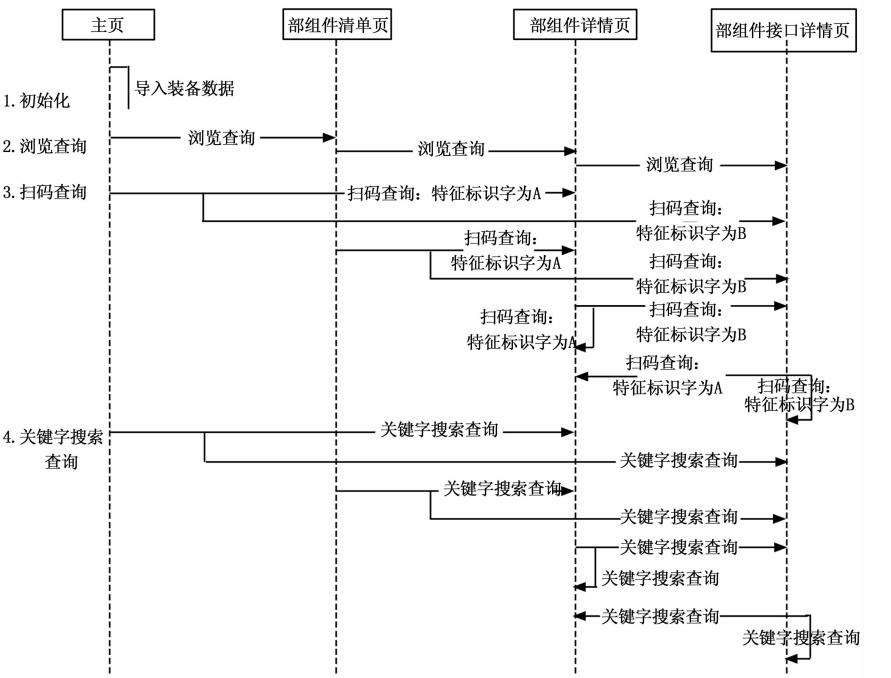

图 2 应用软件顺序图

此外，4 个布局页顶部均设有提供扫码查询和关键字搜

索查询菜单项的标题栏，其中首页和部组件清单页采用 Toolbar，部组件详情页和部组件接口明细页则改用可折叠式标题栏 CollapsingToolbarLayout，下拉标题栏式可显示部组件或部组件对外接口的照片。

2.2　关键软件编程设计

2.2.1　装备信息数据管理设计

1）利用 LitePal 配置管理数据：

首先在 app/build.gradle 文件中的 dependencies 闭包中添加“implementation 'org.litepal.android:java:3.0.0'”以引入 LitePal 开源数据库框架。之后在 assets 文件夹下新建 litepal.xml 配置数据库，关键代码如下：

```
<litepal>
<! -- 定义数据库名字,后缀.db -->
<dbname value="ElectricCableManagementDatas" />
<! -- 版本号 -->
<version value="10" />
<! -- 定义三个数据表 -->
<list>
<mapping class="com.example.electriccablemanagement2.model.MatchingEquipments" />
<mapping class="com.example.electriccablemanagement2.model.CablePinsDefinitions" />
<mapping class="com.example.electriccablemanagement2.model.EquipmentInterfaces" />
</list>
</litepal>
```

对应配置文件中定义的三张数据表定义三个对应的数据管理类：Matchingequipments：其实例为装备某一部组件，包含属性有序号、名称、代号、所属分系统、安装位置、接口清单、备注；Equipmentinterfaces：其实例为装备部组件某一对外接口的定义，包含的属性有序号、名称、代号、接口、连接线缆、线缆去向、备注；Cablepinsdefinitions：其实例为部组件对外接口连接线缆的针脚定义，包含的属性有序号、名称、接口 A 针脚 pin_nx、电路特性、接口 B（信号去向）、针脚 pin_ny（信号去向）。

LitePal 新生成的数据库文件默认保存在/data/data/packagename/databases 目录下，APP 启动运行后，将自动创建数据库文件，之后就可以利用 LitePal 查询装备数据信息，比如由组合名称查找组合对应代号：

```
String mMark = LitePal.select("mark").where("name = ?", mNameTxt).find(MatchingEquipments.class).get(0).getMark();
```

2）从 assets 文件下的 SQLite 数据库导入数据：

因 LitePal 只支持自己创建的数据库，为使用已有装备信息数据库，在利用 LitePal 配置管理数据时，对应数据表定义的三个数据管理类应与现有数据库中数据表的结构保持一致，然后新建 DBManager 类，通过其定义的 openDatabase（）和 closeDatabase（）两个函数，将 assets 文件下的已有装备信息数据库导入至 LitePal 创建的数据库，采取读取写入字节流的方式导入已有装备信息数据，关键代码如下：

```
//数据库文件名
private static final String DB_NAME = "ElectricCableManagementDatas.db";
//包名
private static final String PACKAGE_NAME = "com.example.electriccablemanagement2";
//LitePal 创建数据库的存放目录
private static final String DB_PATH = "/data"+Environment.getDataDirectory().getAbsolutePath() + "/" + PACKAGE_NAME + "/databases";
//打开数据库
public void openDatabase() {
//判断目录是否存在,若不存在则创建目录
File dFile = new File(DB_PATH);
if (! dFile.exists())
dFile.mkdir();
//打开数据库文件并拷贝复制数据块
this.database = this.openDatabase(DB_PATH + "/" + DB_NAME);
}
//从 assets 文件夹导入已有装备信息数据库
private SQLiteDatabase openDatabase(String dbfile) {
……
if ((new File(dbfile).exists())) {
//创建字节输入流
FileInputStream fis = this.context.getResources().getAssets().open("ElectricCableManagementDatas.db");
//创建字节输出流
FileOutputStream fos = new FileOutputStream(dbfile);
int BUFFER_SIZE = 1028 * 10;
byte[] buffer = new byte[BUFFER_SIZE];
//字节输入流实际读取的字符数
int count = 0;
//从已有装备信息数据库文件读取字符并写入 LitePal 创建的数据库
while ((count = fis.read(buffer)) > 0) {
fos.write(buffer, 0, count);
}
//关闭字节输入流和字节输出流
fos.close();
fis.close();
……
}
```

在 MainActivity 的 onCreate（）中添加如下代码以在 APP 启动时自动导入装备数据库。

```
//从 assets 文件夹导入外部数据库
DBManager dbHelper =new DBManager(this);
dbHelper.openDatabase();
```

```
dbHelper.closeDatabase();
```

3）加载显示 PDF 文件和 JPEG 文件：

首先在 app/build.gradle 文件中的 dependencies 闭包中添加“implementation 'com.github.barteksc：android－pdf－viewer：2.8.2”、“implementation 'com.github.bumptech.glide：glide：4.11.0”以引入库文件。加载显示 PDF 文件和 JPEG 文件关键代码如下：

```
//加载显示 PDF 文件
mPdfView=(PDFView)findViewById(R.id.equipment_details_pdfview);
mPdfView.fromAsset(mName.pdf).enableSwipe(true).swipeHorizontal(true).enableDoubletap(true).defaultPage(0).load();
//加载显示 JPEG 文件
Glide.with(this).load(mPicturePath).into(pictureImage);
```

2.2.2 标题栏扫码查询菜单项设计

华为统一扫码服务（Scan Kit）[6]提供便捷的条形码和二维码扫描与解析能力，默认支持 13 种码制式，可以实现远距离条码或二维码的检测与自动放大，并针对常见复杂扫码场景（如：强光照、污损、柱面等）做了针对性识别优化，以提升扫码成功率与用户体验。Scan Kit 提供 Default View Mode、Customized View Mode、Bitmap API Mode 三种调用方式，其中 Default View Mode 提供相机扫码和导入图片扫码两个功能，提供了完整的 Activity 和扫码界面 UI，Scan Kit 直接控制相机实现最优的相机 Zoom 控制、自适应的曝光调节、自适应对焦调节等操作，保障最佳的扫码体验，特别适用于快速集成。将 HMS SDK 集成到 Android Studio 开发环境中的操作步骤为：

首先打开 Android Studio 项目级 build.gradle 文件，在 allprojects －>repositories 和 buildscript －>repositories 里面分别添加代码“maven {url 'http://developer.huawei.com/repo/'}”、在 buildscript －>dependencies 里面添加代码“classpath 'com.huawei.agconnect：agcp：1.2.1.301'”以配置 HMS SDK 的 maven 仓地址。之后打开 app/build.gradle 文件，在文件头添加代码“apply plugin：'com.huawei.agconnect'”、在“dependencies”中添加代码“implementation 'com.huawei.hms：scanplus：1.1.1.301'”以添加编译依赖。调用 Scan Kit 时，需要在 Manifest 文件中注册申明“com.huawei.hms.hmsscankit.ScanKitActivity”，构建相机扫码和导入图片扫码功能，还需要申请“CAMERA”（相机权限）和“READ_EXTERNAL_STORAGE”（读文件权限）。配置完成后，使用 Default View 调用方式进行扫码查询装备数据信息的主要业务流程如图 3 所示。

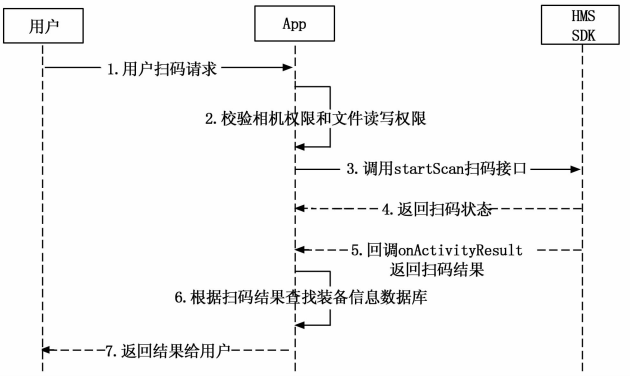

图 3 应用软件顺序图

扫码查询关键代码如下所示：

```
//工具栏菜单选择响应处理
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
switch (item.getItemId()) {
//扫码查询
case R.id.scanview:
//动态申请相机和文件读写权限
ActivityCompat.requestPermissions(MainActivity.this, new String[]{Manifest.permission.CAMERA, Manifest.permission.READ_EXTERNAL_STORAGE}, PERMISSION_CODE);
break;
default:
}
return true;
}
//权限返回的结果处理
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)
{
//判断是否通过权限申请
if (permissions == null || grantResults == null || grantResults.length < 2 || grantResults[0] != PackageManager.PERMISSION_GRANTED || grantResults[1] != PackageManager.PERMISSION_GRANTED) {
return;
}
//默认扫码功能,支持选择相册识别
if (requestCode == DEFINED_CODE) {
//设置支持扫码识别的类型码
ScanUtil.startScan(this, REQUEST_CODE_SCAN, new HmsScanAnalyzerOptions.Creator().setHmsScanTypes(HmsScan.ALL_SCAN_TYPE).create());
}
}
//扫码识别并查询数据库
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
……
```

```
//读取扫码返回结果
if (requestCode == REQUEST_CODE_SCAN) {
HmsScan obj = data.getParcelableExtra(ScanUtil.RESULT);
if (obj ! = null) {
//从扫码结果中取出原始码值
String scan_result = obj.originalValue;
//跳转至电子组合页面,需提供组合的名称和对应图片的绝对存储路径
//根据扫码结果自动跳转至对应界面,若 scan_result 以 A 开头,比如"A:加电控制组合",则跳转至电子组合页面
if(scan_result.startsWith("A")){Intent intent = new Intent(MainActivity.this, DetailsPageActivity.class);
Bundle check_bd = new Bundle();
//从二维码中识别出组合名称
String mName = scan_result.substring(2);
//由组合名称查找组合对应代号
String mMark = LitePal.select("mark").where("name = ?", mName).find(MatchingEquipments.class).get(0).getMark();
//组合对应实物照片存储绝对路径
String mPicturePath = "file:///android_asset/Pictures/" + mMark + ".jpg";
//跳转至组合详情页
check_bd.putString("名称",mName);
check_bd.putString("图片绝对路径",mPicturePath);
intent.putExtras(check_bd);
startActivity(intent);
}
//若以 B 开头,比如"B:6S2208-7D X1",则跳转至组合对外接口页面,需提供组合接口代号
else if(scan_result.startsWith("B")){
Intent intent = new Intent(MainActivity.this, InterfaceDatasActivity.class);
//从二维码中识别出组合接口对应的代号,并自动跳转至对应界面 intent.putExtra("CHECK_KEY",scan_result.substring(2));
startActivity(intent);
}}
else {
Toast.makeText(MainActivity.this, "未扫码到二维码或条形码!", Toast.LENGTH_SHORT).show();
}}}
```

2.2.3　标题栏关键字搜索查询菜单项设计

关键字搜索查询采用 SearchView 控件实现，用户输入关键字比如部组件的代号时，点击提交查询按钮，将自动从装备数据库从查询该部组件的相关技术信息，并自动跳转至部组件详情页展示相关数据。搜索栏具备自动完成功能，在用户输入 1 个字符后，将自动展开一个下拉列表列出与改字符匹配的预设项供用户选择，用户选中下拉列表中的某一待选项后，将自动完成提交、查询，并自动跳转至部组件详情页展示相关数据。关键代码如下：

```
//加载菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
//引用 menu 文件
getMenuInflater().inflate(R.menu.toolbar, menu);
//获取搜索框组件 SearchView
mSearchView = (SearchView) MenuItemCompat.getActionView(menu.findItem(R.id.searchview));
//设置显示搜索框展开时的提交按钮
mSearchView.setSubmitButtonEnabled(true);
//设置搜索框提示语
mSearchView.setQueryHint("请输入关键字");
//获取 mSearchView 的子组件以自定义样式
mSearchAutoComplete = (SearchAutoComplete) mSearchView.findViewById(R.id.search_src_text);
//设置输入框提示文字的颜色、背景色、大小
mSearchAutoComplete.setHintTextColor(getResources().getColor(android.R.color.darker_gray));
mSearchAutoComplete.setTextColor(getResources().getColor(android.R.color.background_light));
mSearchAutoComplete.setTextSize(14);
//设置输入框输入 1 个字符时触发查询
mSearchAutoComplete.setThreshold(1);
//创建数组适配器,list 为存放预设项数据的字符类型数组
ArrayAdapter adapter = new ArrayAdapter<String>(this, android.R.layout.simple_dropdown_item_1line, 0,list);
//为输入框设置数组适配器
mSearchAutoComplete.setAdapter(adapter);
//监听搜索框输入字符,实时显示匹配预设项,实现快速搜索查询功能
mSearchAutoComplete.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
//获取选择项的值,也即组合代号
String selectStr = (String) parent.getItemAtPosition(position);
mSearchAutoComplete.setText(selectStr);
//由组合代号查找组合名称
String mNmae = LitePal.select("name").where("mark = ?", selectStr).find(MatchingEquipments.class).get(0).getName();
//组合对应实物照片存储绝对路径
String mPicturePath = "file:///android_asset/Pictures/" + selectStr + ".jpg";
//自动跳转至部组件详情页
Intent intent = new Intent(MainActivity.this, DetailsPageActivity.class);
Bundle check_bd = new Bundle();
check_bd.putString("名称",mNmae);
check_bd.putString("图片绝对路径",mPicturePath);
intent.putExtras(check_bd);
```

```
startActivity(intent);
return;
}
});
//监听搜索框文字变化
mSearchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
@Override public boolean onQueryTextSubmit(String searchStr) {
//由组合代号查找组合名称
String mNmae = LitePal.select("name").where("mark = ?", searchStr).find(MatchingEquipments.class).get(0).getName();
//组合对应实物照片存储绝对路径
String mPicturePath = "file:///android_asset/Pictures/" + searchStr + ".jpg";
//自动跳转至部组件详情页
Intent intent = new Intent(MainActivity.this, DetailsPageActivity.class);
Bundle check_bd = new Bundle();
check_bd.putString("名称",mNmae);
check_bd.putString("图片绝对路径",mPicturePath); intent.putExtras(check_bd);
startActivity(intent);
return false;
}
……
return super.onCreateOptionsMenu(menu);
}
```

2.2.4 表格数据展示编程设计

部组件详情页中的对外接口标签页和部组件对外接口连接线缆针脚定义明细，采用 smartTable 开源控件[7]以表格的形式进行展示。使用前需在 app/build.gradle 文件中的 dependencies 闭包中添加“com.github.huangyanbin：SmartTable：2.2.0”以引入库文件。关键代码如下所示：

```
//在 activity_interface__datas.xml 中添加 smartTable 控件
<com.bin.david.form.core.SmartTable
android:id="@+id/interface_datas_table"
android:layout_width="match_parent"
android:layout_height="match_parent" />
//在 InterfaceDatasActivity 中使用 smartTable 控件
public class InterfaceDatasActivity extends AppCompatActivity {
//InterfaceDatasInfo 为自定义类对象，包含 pinNumber、circuitCharacteristic、connectionDestination 三个 String 变量
private SmartTable<InterfaceDatasInfo> interfaceDatasTable;
//表格表题
private String smartTableTitle;
//针脚号
Column<String> pinNumber;
//电路特性
Column<String> circuitCharacteristic;
//信号去向
Column<String> connectionDestination;
……
@Override
protected void onCreate(Bundle savedInstanceState) {
……
//读取数据
Intent intent = getIntent();
//接口代号，比如 6S2208－7D X1
String check_key = intent.getStringExtra("CHECK_KEY");
//接口对应实物图片存储路径
mPicturePath = PATH_HEAD + check_key + ".jpg";
//由组合接口代号分析出组合代号
String mMark = check_key.substring(0,(check_key.indexOf("7D"))+2);
//由组合代号查找组合名称
String mNameTxt = LitePal.select("name").where("mark = ?", mMark).find(MatchingEquipments.class).get(0).getName();
smartTableTitle = mNameTxt + check_key;
……
//查询数据库，获取组合对应接口针脚定义数据
List< CablePinsDefinitions > check_CablePinsDefinitions = new List<CablePinsDefinitions>() {};
check_CablePinsDefinitions = LitePal.where("connectiona = ?",check_key).find(CablePinsDefinitions.class);
//表格填充数据
List<InterfaceDatasInfo> list = new ArrayList<>();
for(int i=0;i<check_CablePinsDefinitions.size();i++){
String pinnumber = check_CablePinsDefinitions.get(i).getPinnumberA();
String define = check_CablePinsDefinitions.get(i).getCircuitCharacteristic();
String connect = check_CablePinsDefinitions.get(i).getConnectionB();
connect = connect + ";" + check_CablePinsDefinitions.get(i).getPinnumberB();
list.add(new InterfaceDatasInfo(pinnumber,define,connect));
}
//定义表格的列条目
pinNumber = new Column<>("针脚号", "pinNumber");
//滑动到表格左边时固定列
pinNumber.setFixed(true);
circuitCharacteristic = new Column<>("电路特性", "circuitCharacteristic");
//设置列左对齐
circuitCharacteristic.setTextAlign(Paint.Align.LEFT);
//定义表格的列条目
connectionDestination = new Column<>("信号去向", "connectionDestination");
//设置列左对齐
connectionDestination.setTextAlign(Paint.Align.LEFT);
//定义表格数据
```

（下转第 168 页）