Experiment

实 验 手 册

软通大学教研部

记录编号：_____

HarmonyOS 手机通讯录开发
实验指导手册

版本：V 1.0

北京软通动力教育科技有限公司

2021/2/25

目录

(一) 实验目的

1. 掌握 HarmonyOS 移动应用开发工具的使用；
2. 掌握移动应用开发的需求分解与实现步骤拆解；
3. 掌握 UI 组件与布局、ListContainer 子布局结合 RecycleItemProvider 使用、日志打印、消息提示、页面跳转与传参、数据库操作等等相关开发技术。
4. 锻炼开发文档阅读能力以及知识转化能力；
5. 掌握知识点深入探讨研究、架构设计思维、造轮子思想，通过自定义组件进行锻炼；
6. 养成良好的编程规范，培养清晰的逻辑思维与编程思想；

(二) 实验涉及知识点

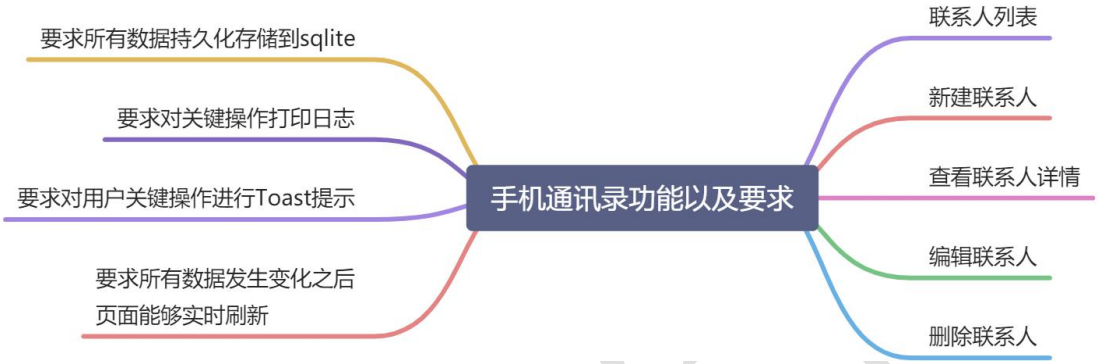
1. HarmonyOS 移动应用开发工具（DevEco Studio）使用；
2. UI 组件使用，包括 Text，Button，TextFiled，Image，RadioButton，RadioContainer，ToastDialog，ListContainer；
3. UI 布局的使用，包括 DirectionalLayout，DependentLayout；
4. 日志打印，HiLog 的使用；
5. **各种事件监听操作与业务逻辑实现；（重难点）**
6. **ListContainer 子布局结合 RecycleItemProvider 的使用；（重难点）**
7. **对话框以及自定义对话框的使用；（重难点）**
8. 页面生命周期以及页面之间跳转与传参；
9. **自定义组件（同时涉及到 Canvas）以及如何调用；（重难点）**
10. **数据存储操作，主要涉及 sqlite 数据库存储；（重难点）**
11. **代码编程规范、设计模式；（重难点）**

(三) 实验课时

8 课时。

(四) 实验内容与要求、最终效果图

1. 操作人员开发技能前置要求，良好的 Java 基础和 sql 基础能力；
2. 配置好 HarmonyOS 移动应用开发环境
3. 完成鸿蒙手机通讯录功能开发，具体功能要求如下图所示



4. 主要功能页面效果图展示

(五) 实验过程分解与实验目标对应矩阵

序号	阶段内容	本阶段核心知识点	对应实验目标点
1	联系人列表页面静态数据呈现	1,2,3,6,11	1,2,3,6
2	自定义圆形图片的实现	9,11	4,5,6

3	添加联系人以及实时刷新联系人列表	2,3,4,5,7,11	3,5,6
4	点击联系人列表查看联系人详情，以及点击自定义的返回按钮返回	8,11	3,4,6
5	长按删除联系人列表删除联系人	5,7,11	3,4,6
6	对接 Sqlite，实现数据持久化与数据动态更新	10,11	3,4,6

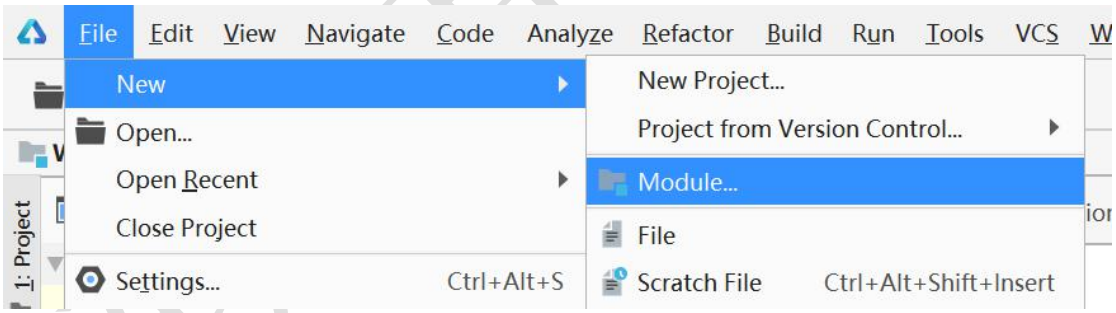
(六) 详细实验过程操作

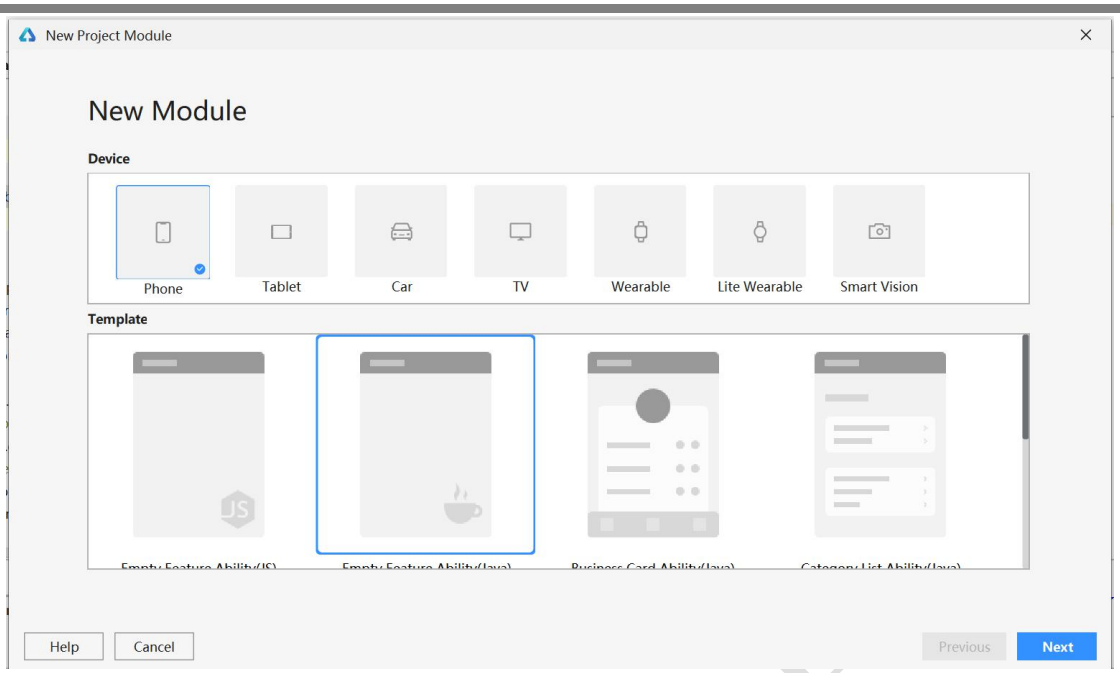
实验中所有涉及到的 java 代码注释详细，编码规范，通过注释和步骤解释进行消化理解。

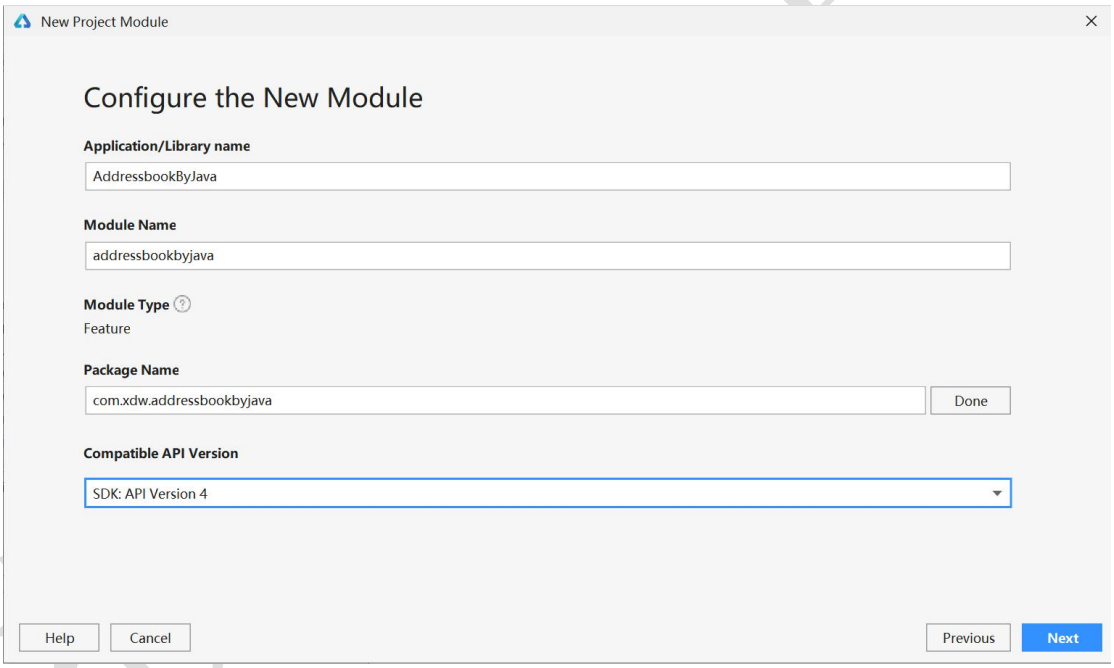
1、准备工作

（1）新建工程和模块

首先打开 DevEco Studio,新建一个工程，工程类型可以任意选择，然后在工程下新建一个 Module，该 Module 选择为 Java FA 类型，具体操作如下图所示：

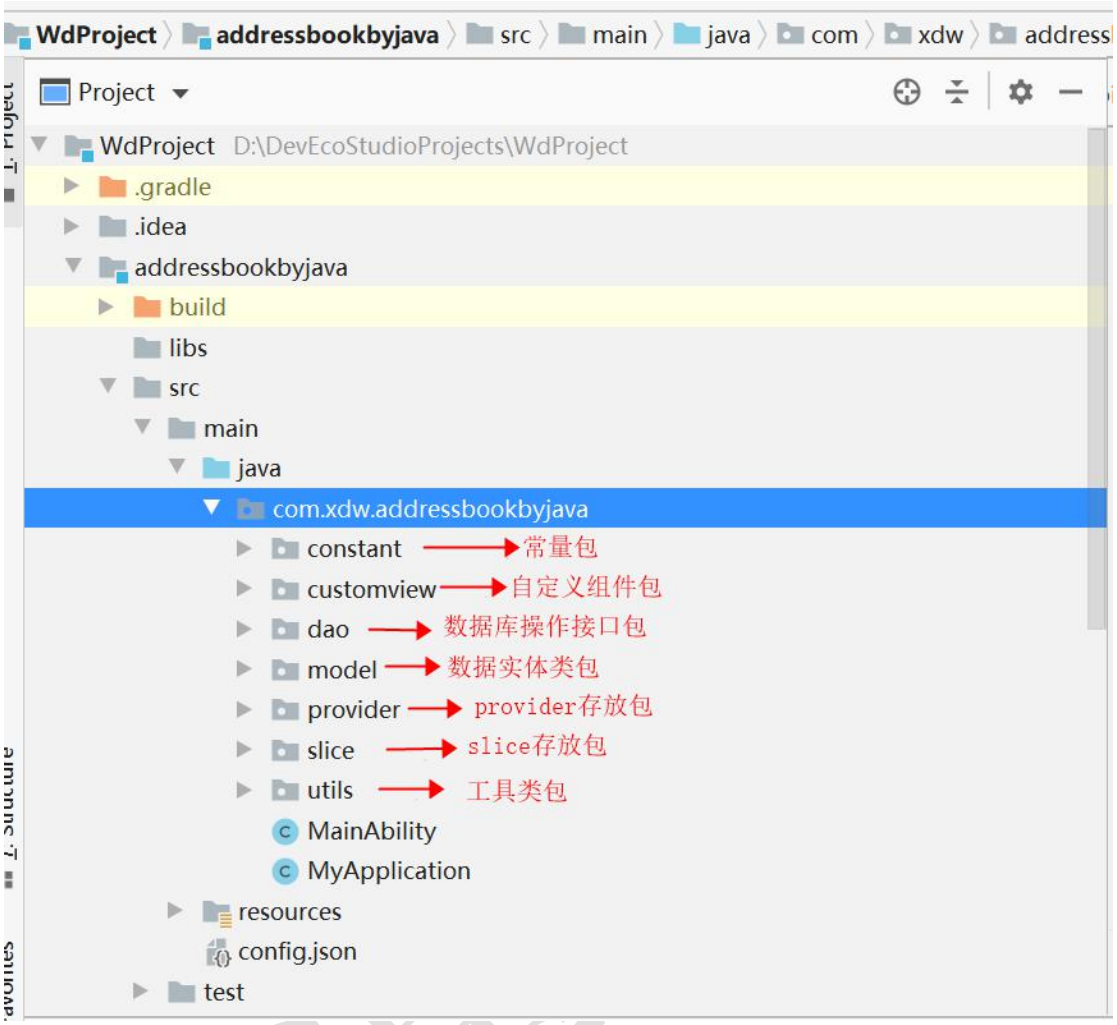






（2）规划包结构

待上面的工程和模块创建好之后，规划好后面开发的包组织结构，可以参考下图：



注意：良好的包结构划分能更好的体现编程思路清晰，更方便他人阅读代码，后面还会有好多编程规范的提示；

（3）导入外部资源文件

下面从随实验手册的附件或者源码中获取相关图片资源进行导入到工程中，直接将所有图片复制到 media 目录下即可。

（4）创建常量类

先在常量包 constant 包下创建一个接口 Constant 专门用来存放定义的常量标识符，代码如下：

```
package com.xdw.addressbookbyjava.constant;


/**
 * Created by 夏德旺 on 2021/1/4 0:06
```

```
*/public interface Constant {
    int GENDER_MAN = 0;
    int GENDER_LADY = 1;
    String GENDER_MAN_STRING = "男";
    String GENDER_LADY_STRING = "女";}
```

常量标识符的定义能更方便后期阅读和维护代码，采用魔鬼数字和字符串是不符合编程规范的。

（5）设置桌面快捷方式、图标、名称、主题（无自带标题栏主题）等

修改模块下的 config.json 文件，代码如下：

```
{
  "app": {
    "bundleName": "com.example.wdproject",
    "vendor": "xdw",
    "version": {
      "code": 1,
      "name": "1.0"
    },
    "apiVersion": {
      "compatible": 4,
      "target": 4,
      "releaseType": "Beta1"
    }
  },
  "deviceConfig": {},
  "module": {
    "package": "com.xdw.addressbookbyjava",
    "name": ".MyApplication",
    "deviceType": [
```

```
    "phone"
  ],
  "distro": {
    "deliveryWithInstall": true,
    "moduleName": "addressbookbyjava",
    "moduleType": "feature"
  },
  "abilities": [
    {
      "skills": [
        {
          "entities": [
            "entity.system.home"
          ],
          "actions": [
            "action.system.home"
          ]
        }
      ],
      "orientation": "unspecified",
      "visible": true,
      "name": "com.xdw.addressbookbyjava.MainAbility",
      "icon": "$media:icon",
      "description": "$string:mainability_description",
      "label": "通讯录",
      "type": "page",
      "launchType": "standard",
      "metaData": {
        "customizeData": [
```

```
            {
              "name": "hwc-theme",
              "value": "androidhwext:style/Theme.Emui.Light.NoTitleBar"
            }
          ]
        }
      }
    ]
}}
```

2、联系人列表页面静态数据呈现

（1）编写布局文件

这里联系人列表页就是本项目的首页，直接采用项目自动创建的页面 MainAbilitySlice，它对应的布局文件为 ability_main.xml。编辑该布局文件代码，如下：

```
<?xml version="1.0" encoding="utf-8"?><DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:height="match_parent"
    ohos:width="match_parent"
    ohos:orientation="vertical"
    >

    <DependentLayout
        ohos:id="$+id:dl"
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:background_element="$graphic:color_light_gray_element"
        ohos:padding="10vp"
        >
```

```
        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:align_parent_start="true"
            ohos:center_in_parent="true"
            ohos:text="本地联系人"
            ohos:text_size="24fp"/>

        <Image
            ohos:id="$+id:image_menu"
            ohos:height="30vp"
            ohos:width="30vp"
            ohos:align_parent_end="true"
            ohos:center_in_parent="true"
            ohos:image_src="$media:menu"/>

        <Image
            ohos:id="$+id:image_add"
            ohos:height="30vp"
            ohos:width="30vp"
            ohos:center_in_parent="true"
            ohos:image_src="$media:add"
            ohos:left_of="$id:image_menu"
            ohos:right_margin="10vp"/>
    </DependentLayout>

    <ListContainer
        ohos:id="$+id:list_contacts"
        ohos:height="match_parent"
```

```
        ohos:width="match_parent"
        ohos:background_element="$graphic:background_ability_main"
        ohos:text_size="20fp"
        ohos:left_margin="10vp"
        ohos:top_margin="10vp"
        />

</DirectionalLayout>
```

注意：这里用到了设置背景颜色功能，目前鸿蒙不支持直接设置背景色，需要通过在 graphic 下定义一个背景的方式，然后来指定背景来进行设置，这里定义了一个背景的 xml 文件 color_light_gray_element，它的代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?><shape
xmlns:ohos="http://schemas.huawei.com/res/ohos"
        ohos:shape="rectangle">
    <solid
        ohos:color="#D3D3D3"/></shape>
```

（2）新建 ListContainer 列表组件所对应的子布局文件 item_contacts.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?><DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:height="match_content"
    ohos:width="match_parent"
    ohos:alignment="vertical_center"
    ohos:orientation="horizontal"
    ohos:padding="10vp">

    <Image
        ohos:id="$+id:item_gender_icon"
```

```
        ohos:height="50vp"
        ohos:width="50vp"/>

    <Text
        ohos:id="$+id:item_name"
        ohos:height="match_content"
        ohos:width="match_content"
        ohos:left_margin="20vp"
        ohos:text="Item0"
        ohos:text_size="20fp"/></DirectionalLayout>
```

（3）面向对象设计，构建一个联系人信息对应的数据实体类 **Contacts**，代码如下：

```
package com.xdw.addressbookbyjava.model;

import java.io.Serializable;

/**
 * Created by 夏德旺 on 2020/12/31 20:57
 */public class Contacts implements Serializable {
    private int id;       //主键 id
    private String name;     //联系人姓名
    private int gender;     //0 代表男士，1 代表女士
    private String phone;    //联系人电话
    private int groupId;     //分组 id,0 代表朋友，1 代表家人，2 代表同事

    public int getId() {
        return id;
    }
```

```
public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getGender() {
    return gender;
}

public void setGender(int gender) {
    this.gender = gender;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}

public int getGroupId() {
```

```
        return groupId;
    }

    public void setGroupId(int groupId) {
        this.groupId = groupId;
    }

    public Contacts() {
    }

    public Contacts(String name, int gender) {
        this.name = name;
        this.gender = gender;
    }}
```

（4）编写 ListContainer 对应的 provider，取名为 ContactsProvider 继承自 RecycleItemProvider，具体代码如下：

```
package com.xdw.addressbookbyjava.provider;

import com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.model.Contacts;import ohos.aafwk.ability.AbilitySlice;import
ohos.agp.components.*;

import java.util.List;

/**
 * Created by 夏德旺 on 2020/12/31 20:58
 */public class ContactsProvider extends RecycleItemProvider {
    private List<Contacts> list;    //数据源
    private AbilitySlice slice;     //context 对象
```

```
/**
 * 构造函数
 *
 * @param list
 * @param slice
 */
public ContactsProvider(List<Contacts> list, AbilitySlice slice) {
    this.list = list;
    this.slice = slice;
}

/**
 * 数据源的大小
 *
 * @return
 */
@Override
public int getCount() {
    return list.size();
}

/**
 * 通过索引获取 item
 *
 * @param position
 * @return
 */
@Override
```

```
    public Object getItem(int position) {
        return list.get(position);
    }

    /**
     * 获取索引
     *
     * @param position
     * @return
     */
    @Override
    public long getItemId(int position) {
        return position;
    }

    /**
     * 加载填充每一个 Item 内部的控件的数据
     *
     * @param position
     * @param component
     * @param componentContainer
     * @return
     */
    @Override
    public Component getComponent(int position, Component component, ComponentContainer componentContainer) {
        Component cpt = component;
        if (cpt == null) {
            cpt =
```

```
LayoutScatter.getInstance(slice).parse(ResourceTable.Layout_item_contacts, null, false);
        }
        Contacts contacts = list.get(position);
        Text textName = (Text) cpt.findComponentById(ResourceTable.Id_item_name);
        Image imageGenderIcon = (Image)
cpt.findComponentById(ResourceTable.Id_item_gender_icon);
        textName.setText(contacts.getName());
        if (contacts.getGender() == 0) {
            imageGenderIcon.setPixelMap(ResourceTable.Media_man);
        } else {
            imageGenderIcon.setPixelMap(ResourceTable.Media_lady);
        }
        return cpt;
    }}
```

（5）编写 MainAbilitySlice 的代码，渲染 ItemContainer 组件

首先在里面初始化各个组件，将 xml 布局中的组件和 java 代码中的对象关联起来，然后构建一个生成 N 条联系人信息的方法，然后调用 ItemContainer 的关键 api 即 setItemProvider，将生成的静态数据填充到列表组件，并且渲染列表中的各个 item 子组件，从而显示一个完整的图文并茂的 list 列表详细代码如下：

```
package com.xdw.addressbookbyjava.slice;


import com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.model.Contacts;import
com.xdw.addressbookbyjava.provider.ContactsProvider;import
ohos.aafwk.ability.AbilitySlice;import ohos.aafwk.content.Intent;import
ohos.agp.components.Component;import ohos.agp.components.Image;import
ohos.agp.components.ListContainer;
```

```
import java.util.ArrayList;import java.util.List;


public class MainAbilitySlice extends AbilitySlice implements Component.ClickedListener {
    private Image addContactsBtn;      //添加联系人按钮
    private Image menuBtn;             //菜单按钮
    private List<Contacts> list;       //列表数据源
    private ListContainer listContainer;     //list 列表
    private ContactsProvider contactsProvider;      //list 列表绑定的 provider



    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);
        initView();
        initListContainer();


    }


    @Override
    public void onActive() {
        super.onActive();
    }


    @Override
    public void onForeground(Intent intent) {
        super.onForeground(intent);
    }
```

```
    /**
     * 初始化视图组件以及绑定监听事件
     */
    private void initView() {
        addContactsBtn = (Image) findComponentById(ResourceTable.Id_image_add);
        menuBtn = (Image) findComponentById(ResourceTable.Id_image_menu);
        addContactsBtn.setClickedListener(this);
        menuBtn.setClickedListener(this);
    }

    /**
     * 初始化列表
     */
    private void initListContainer() {
        listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
        list = getData();    //静态数据对接
        contactsProvider = new ContactsProvider(list, this);
        listContainer.setItemProvider(contactsProvider);

    }

    //生成静态的列表数据进行模拟，在对接 sqlite 或者服务端之后不再使用
    private List<Contacts> getData() {
        List<Contacts> list = new ArrayList<>();
        String[] names = {"克里斯迪亚洛罗纳尔多", "王霜", "梅西", "孙雯", "莱万多夫斯基", "玛
塔", "樱木花道", "赤木晴子"};
        for (int i = 0; i <= 7; i++) {
            list.add(new Contacts(names[i], i % 2));
```

```
        }
        return list;
    }


    @Override
    public void onClick(Component component) {


    }}
```

该步骤进行完成之后，我们可以打开模拟器，运行该 app 看下运行效果了，此时的运行效果图如下：

步骤小结与思考：

完成此步骤，对基本布局和组件运用以及与 java 代码关联有了一定认识，并且能知晓 ListContainer 组件加载数据渲染列表的操作流程。

思考：如果这里列表的每一个 item 显示效果更复杂，比如有多张图片和多张文本混合排列该怎么做呢？

3、自定义圆形图片的实现

上面我们已经完成了一个最基础的效果，但是发现和原型设计的效果不符，原型设计中图片是圆形显示的，而这里是方形，而通过阅读官方资料我们知道官方的 Image 组件不支持设置成圆形，于是就需要我们自定义组件一个圆形组件。

此步操作是一个相对高级难点的步骤，请大家反复练习和思考来加以掌握。

自定义组件用到了两个官方的核心 api，分别是分别是 Component 的 addDrawTask 方法和其内部静态接口 DrawTask。

具体操作如下：

（1）在 customview 包下面新建一个类 RoundImage 继承 Image 组件，然后编写相关代码如下：

```
package com.xdw.addressbookbyjava.customview;


import ohos.agp.components.AttrSet;import ohos.agp.components.Image;import
ohos.agp.render.PixelMapHolder;import ohos.agp.utils.RectFloat;import
ohos.app.Context;import ohos.hiviewdfx.HiLog;import ohos.hiviewdfx.HiLogLabel;import
ohos.media.image.ImageSource;import ohos.media.image.PixelMap;import
ohos.media.image.common.PixelFormat;import ohos.media.image.common.Rect;import
ohos.media.image.common.Size;


import java.io.InputStream;


/**
 * Created by 夏德旺 on 2021/1/1 11:00
 */public class RoundImage extends Image {
```

```
    private static final HiLogLabel LABEL = new HiLogLabel(HiLog.DEBUG, 0,
"RoundImage");
    private PixelMapHolder pixelMapHolder;//像素图片持有者
    private RectFloat rectDst;//目标区域
    private RectFloat rectSrc;//源区域
    public RoundImage(Context context) {
        this(context,null);

    }

    public RoundImage(Context context, AttrSet attrSet) {
        this(context,attrSet,null);
    }

    /**
     * 加载包含该控件的 xml 布局，会执行该构造函数
     * @param context
     * @param attrSet
     * @param styleName
     */
    public RoundImage(Context context, AttrSet attrSet, String styleName) {
        super(context, attrSet, styleName);
        HiLog.error(LABEL,"RoundImage");
    }



    public void onRoundRectDraw(int radius){
        //添加绘制任务
```

```
        this.addDrawTask((view, canvas) -> {
            if (pixelMapHolder == null){
                return;
            }
            synchronized (pixelMapHolder) {
                //给目标区域赋值，宽度和高度取自 xml 配置文件中的属性
                rectDst = new RectFloat(0,0,getWidth(),getHeight());
                //绘制圆角图片
                canvas.drawPixelMapHolderRoundRectShape(pixelMapHolder, rectSrc,
rectDst, radius, radius);
                pixelMapHolder = null;
            }
        });
    }

    //使用 canvas 绘制圆形
    private void onCircleDraw(){
        //添加绘制任务，自定义组件的核心 api 调用，该接口的参数为 Component 下的 DrawTask
接口
        this.addDrawTask((view, canvas) -> {
            if (pixelMapHolder == null){
                return;
            }
            synchronized (pixelMapHolder) {
                //给目标区域赋值，宽度和高度取自 xml 配置文件中的属性
                rectDst = new RectFloat(0,0,getWidth(),getHeight());
                //使用 canvas 绘制输出圆角矩形的位图，该方法第 4 个参数和第 5 个参数为 radios
参数，
                // 绘制图片，必须把图片的宽度和高度先设置成一样，然后把它们设置为图片宽度
```

```
或者高度一半时则绘制的为圆形
                    canvas.drawPixelMapHolderRoundRectShape(pixelMapHolder, rectSrc,
rectDst, getWidth()/2, getHeight()/2);
                    pixelMapHolder = null;
                }
            });
        }


        /**
         *获取原有 Image 中的位图资源后重新检验绘制该组件
         * @param pixelMap
         */
        private void putPixelMap(PixelMap pixelMap){
            if (pixelMap != null) {
                rectSrc = new RectFloat(0, 0, pixelMap.getImageInfo().size.width,
pixelMap.getImageInfo().size.height);
                pixelMapHolder = new PixelMapHolder(pixelMap);
                invalidate();//重新检验该组件
            }else{
                pixelMapHolder = null;
                setPixelMap(null);
            }
        }


        /**
         * 通过资源 ID 获取位图对象
         **/
```

```
    private PixelMap getPixelMap(int resId) {
        InputStream drawableInputStream = null;
        try {
            drawableInputStream = getResourceManager().getResource(resId);
            ImageSource.SourceOptions sourceOptions = new ImageSource.SourceOptions();
            sourceOptions.formatHint = "image/png";
            ImageSource imageSource = ImageSource.create(drawableInputStream, null);
            ImageSource.DecodingOptions decodingOptions = new
ImageSource.DecodingOptions();
            decodingOptions.desiredSize = new Size(0, 0);
            decodingOptions.desiredRegion = new Rect(0, 0, 0, 0);
            decodingOptions.desiredPixelFormat = PixelFormat.ARGB_8888;
            PixelMap pixelMap = imageSource.createPixelmap(decodingOptions);
            return pixelMap;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try{
                if (drawableInputStream != null){
                    drawableInputStream.close();
                }
            }catch (Exception e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    /**
```

```
    * 对外调用的api，设置圆形图片方法
    * @param resId
    */
    public void setPixelMapAndCircle(int resId){
        PixelMap pixelMap = getPixelMap(resId);
        putPixelMap(pixelMap);
        onCircleDraw();
    }

    /**
    * 对外调用的api，设置圆角图片方法
    * @param resId
    * @param radius
    */
    public void setPixelMapAndRoundRect(int resId,int radius){
        PixelMap pixelMap = getPixelMap(resId);
        putPixelMap(pixelMap);
        onRoundRectDraw(radius);
    }}
```

这样这个圆形图片组件就定义好了，下面可以使用它替换之前系统原生的 Image 组件。

（2）在原来的布局文件 item_contacts.xml 中进行应用，替换原生的 Image 组件，具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?><DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:height="match_content"
    ohos:width="match_parent"
    ohos:alignment="vertical_center"
    ohos:orientation="horizontal"
```

```
    ohos:padding="10vp">

    <com.xdw.addressbookbyjava.customview.RoundImage
        ohos:id="$+id:item_gender_icon"
        ohos:height="50vp"
        ohos:width="50vp"/>

    <Text
        ohos:id="$+id:item_name"
        ohos:height="match_content"
        ohos:width="match_content"
        ohos:left_margin="20vp"
        ohos:text="Item0"
        ohos:text_size="20fp"/></DirectionalLayout>
```

与之前的代码区别就是，这里自定义组件的调用使用的是完整类名，即带包名的类名，同时这里的图片的宽度和高度必须设置成一致才能呈现圆形效果。

（3）在 **ContactsProvider** 的代码中，将原生的 **Image** 进行替换，代码如下：

```
package com.xdw.addressbookbyjava.provider;

import com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.customview.RoundImage;import
com.xdw.addressbookbyjava.model.Contacts;import ohos.aafwk.ability.AbilitySlice;import
ohos.agp.components.*;

import java.util.List;

/**
 * Created by 夏德旺 on 2020/12/31 20:58
```

```
*/public class ContactsProvider extends RecycleItemProvider {
    private List<Contacts> list;    //数据源
    private AbilitySlice slice;     //context 对象

    /**
     * 构造函数
     *
     * @param list
     * @param slice
     */
    public ContactsProvider(List<Contacts> list, AbilitySlice slice) {
        this.list = list;
        this.slice = slice;
    }

    /**
     * 数据源的大小
     *
     * @return
     */
    @Override
    public int getCount() {
        return list.size();
    }

    /**
     * 通过索引获取 item
     *
     * @param position
```

```
     * @return
     */
    @Override
    public Object getItem(int position) {
        return list.get(position);
    }

    /**
     * 获取索引
     *
     * @param position
     * @return
     */
    @Override
    public long getItemId(int position) {
        return position;
    }

    /**
     * 加载填充每一个 Item 内部的控件的数据
     *
     * @param position
     * @param component
     * @param componentContainer
     * @return
     */
    @Override
    public Component getComponent(int position, Component component, ComponentContainer
componentContainer) {
```

```
        Component cpt = component;
        if (cpt == null) {
            cpt =
LayoutScatter.getInstance(slice).parse(ResourceTable.Layout_item_contacts, null, false);
        }
        Contacts contacts = list.get(position);
        Text textName = (Text) cpt.findComponentById(ResourceTable.Id_item_name);
        RoundImage imageGenderIcon = (RoundImage)
cpt.findComponentById(ResourceTable.Id_item_gender_icon);
        textName.setText(contacts.getName());
        if (contacts.getGender() == 0) {
            imageGenderIcon.setPixelMapAndCircle(ResourceTable.Media_man);
        } else {
            imageGenderIcon.setPixelMapAndCircle(ResourceTable.Media_lady);
        }
        return cpt;
    }}
```

注意：这里在调用的时候，需要调用一个自己编写的设置圆形图片的 api，即 setPixelMapAndCircle 方法，同时配合布局文件中的宽度和高度一致，即可完成圆形图片的设置。

该步骤操作完毕之后，我们可以运行检验下运行效果了，如下图所示：

步骤小结与思考：

完成此步骤，首先需要具备查找和阅读官方 API 文档的能力，同时需要具备扎实的 java 面向对象编程能力，请反复进行训练了解封装流程，以及进行黑盒封装，让外部可以直接进行调用。

该组件后面可以作为独立模块进行打包编译，编译好之后可以发布到 maven 或者 gradle 中心仓库，以后全世界任何人任何工程都可以通过依赖导入的方式进行导入。

4、添加联系人以及实时刷新联系人列表

（1）添加联系人 UI 页面的选择

添加联系人的操作，我们可以通过再新建一个 AbilitySlice 的方式，也可以通过弹出一个对话框来进行输入相关信息并提交，这里我们选取弹出对话框的方式，但是系统原生的对话框无法满足该需求，因此需要继承原生对话框进行自定义扩展。

（2）自定义一个添加联系人信息的对话框并且调用

由于该对话框只在联系人列表这个页面里面出现，于是没有对其做独立封装，就在 MainAbilitySlice 内部编写了一个方法，该方法生成这个对话框并显示，该方法命名为 showAddContactsDialog，同时编写一个对话框中要加载的布局 dialog_add_contacts.xml，该布局代码如下：

```
<?xml version="1.0" encoding="utf-8"?><DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:height="match_content"
    ohos:width="match_parent"
    ohos:background_element="$graphic:color_light_gray_element"
    ohos:orientation="vertical"
    ohos:padding="20vp">

    <Text
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:text="新建联系人"
        ohos:text_size="24fp"
        ohos:text_style="bold"/>

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:orientation="horizontal"
        ohos:top_margin="20vp">

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:right_margin="10vp"
```

```
            ohos:text="姓名："
            ohos:text_size="20fp"/>

        <TextField
            ohos:id="$+id:tf_name"
            ohos:height="32vp"
            ohos:width="match_parent"
            ohos:background_element="$graphic:background_text_field"
            ohos:text_alignment="vertical_center"
            ohos:text_size="20fp"
            ohos:hint="请输入联系人姓名"/>
    </DirectionalLayout>

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:orientation="horizontal"
        ohos:top_margin="20vp">

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:right_margin="10vp"
            ohos:text="性别："
            ohos:text_size="20fp"/>

        <RadioContainer
            ohos:id="$+id:rc_gender"
```

```
            ohos:height="match_content"
            ohos:width="match_parent"
            ohos:orientation="horizontal">

            <RadioButton
                ohos:id="$+id:rb_man"
                ohos:height="match_content"
                ohos:width="match_content"
                ohos:text="男"
                ohos:text_color_off="#808080"
                ohos:text_color_on="#00BFFF"
                ohos:text_size="20fp"/>

            <RadioButton
                ohos:id="$+id:rb_lady"
                ohos:height="match_content"
                ohos:width="match_content"
                ohos:text="女"
                ohos:text_color_off="#8B8B7A"
                ohos:text_color_on="#00BFFF"
                ohos:text_size="20fp"/>
        </RadioContainer>
    </DirectionalLayout>

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:orientation="horizontal"
```

```
        ohos:top_margin="20vp">

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:right_margin="10vp"
            ohos:text="联系电话："
            ohos:text_size="20fp"/>

        <TextField
            ohos:id="$+id:tf_phone"
            ohos:height="32vp"
            ohos:width="match_parent"
            ohos:background_element="$graphic:background_text_field"
            ohos:text_alignment="vertical_center"
            ohos:text_size="20fp"
            ohos:text_input_type="pattern_number"
            ohos:hint="请输入联系人电话号码"/>
    </DirectionalLayout>


    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:orientation="horizontal">

        <Button
            ohos:height="match_content"
            ohos:width="match_parent"
```

```
            ohos:id="$+id:btn_cancel"
            ohos:background_element="$graphic:capsule_button_gray"
            ohos:margin="10vp"
            ohos:padding="10vp"
            ohos:text="取消"
            ohos:text_size="24fp"
            ohos:text_color="white"
            ohos:weight="1"/>

        <Button
            ohos:id="$+id:btn_confirm"
            ohos:height="match_content"
            ohos:width="match_parent"
            ohos:background_element="$graphic:capsule_button_green"
            ohos:margin="10vp"
            ohos:padding="10vp"
            ohos:text="确定"
            ohos:text_size="24fp"
            ohos:text_color="white"
            ohos:weight="1"/>
    </DirectionalLayout></DirectionalLayout>
```

然后编写“添加联系人”图标按钮的点击响应时间，弹出该对话框，并且编写 generateContactsFromInput 方法获取对话框中的数据生成联系人。在点击对话框的确定按钮之后，弹框消失，更新联系人列表，加载最新添加的联系人。

ListContainer 列表更新的核心 api 代码如下：

```
contactsProvider.notifyDataChanged();      //更新 listcontainer
```

最后，此步骤完整的 MainAbilitySlice 的代码如下：

```
package com.xdw.addressbookbyjava.slice;
```

```
import com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.constant.Constant;import
com.xdw.addressbookbyjava.model.Contacts;import
com.xdw.addressbookbyjava.provider.ContactsProvider;import
ohos.aafwk.ability.AbilitySlice;import ohos.aafwk.content.Intent;import
ohos.agp.components.*;import ohos.agp.window.dialog.CommonDialog;import
ohos.agp.window.dialog.ToastDialog;import ohos.hiviewdfx.HiLog;import
ohos.hiviewdfx.HiLogLabel;

import java.util.ArrayList;import java.util.List;

public class MainAbilitySlice extends AbilitySlice implements Component.ClickedListener {
    public static final String TAG = "MainAbilitySlice";
    private static final HiLogLabel LABEL = new HiLogLabel(HiLog.DEBUG, 0, "TAG");
    private Image addContactsBtn;      //添加联系人按钮
    private Image menuBtn;             //菜单按钮
    private List<Contacts> list;       //列表数据源
    private ListContainer listContainer;     //list 列表
    private ContactsProvider contactsProvider;      //list 列表绑定的 provider
    private TextField nameTf;        //联系人姓名输入框
    private RadioContainer genderRc;      //联系人性别单选框组合容器
    private TextField phoneTf;   //联系人电话号码输入框
    private RadioButton manRb;   //单选按钮，男
    private RadioButton ladyRb; //单选按钮，女

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);
```

```
        initView();
        initListContainer();

    }

    @Override
    public void onActive() {
        super.onActive();
    }

    @Override
    public void onForeground(Intent intent) {
        super.onForeground(intent);
    }

    /**
     * 初始化视图组件以及绑定监听事件
     */
    private void initView() {
        addContactsBtn = (Image) findComponentById(ResourceTable.Id_image_add);
        menuBtn = (Image) findComponentById(ResourceTable.Id_image_menu);
        addContactsBtn.setClickedListener(this);
        menuBtn.setClickedListener(this);
    }

    /**
     * 初始化列表
     */
    private void initListContainer() {
```

```
        listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
        list = getData();    //静态数据对接
        contactsProvider = new ContactsProvider(list, this);
        listContainer.setItemProvider(contactsProvider);
    }

    //生成静态的列表数据进行模拟，在对接 sqlite 或者服务端之后不再使用
    private List<Contacts> getData() {
        List<Contacts> list = new ArrayList<>();
        String[] names = {"克里斯迪亚洛罗纳尔多", "王霜", "梅西", "孙雯", "莱万多夫斯基", "玛塔", "樱木花道", "赤木晴子"};
        for (int i = 0; i <= 7; i++) {
            list.add(new Contacts(names[i], i % 2));
        }
        return list;
    }

    //创建并显示自定义的添加联系人操作的对话框
    private void showAddContactsDialog() {
        CommonDialog commonDialog = new CommonDialog(this);

        Component dialogComponent = LayoutScatter.getInstance(getContext())
                .parse(ResourceTable.Layout_dialog_add_contacts, null, true);

//          commonDialog.setSize(800, 600);       //设置对话框尺寸
        Button btnConfirm = (Button)
dialogComponent.findComponentById(ResourceTable.Id_btn_confirm);
        Button btnCancel = (Button)
dialogComponent.findComponentById(ResourceTable.Id_btn_cancel);
```

```
        nameTf = (TextField)
dialogComponent.findComponentById(ResourceTable.Id_tf_name);
        phoneTf = (TextField)
dialogComponent.findComponentById(ResourceTable.Id_tf_phone);
        genderRc = (RadioContainer)
dialogComponent.findComponentById(ResourceTable.Id_rc_gender);
        manRb = (RadioButton)
dialogComponent.findComponentById(ResourceTable.Id_rb_man);
        ladyRb = (RadioButton)
dialogComponent.findComponentById(ResourceTable.Id_rb_lady);
        manRb.setChecked(true);
        genderRc.setMarkChangedListener(new
RadioContainer.CheckedStateChangedListener() {
            @Override
            public void onCheckedChanged(RadioContainer radioContainer, int i) {


            }
        });
        //采用传统的匿名内部类实现按钮监听器
        btnConfirm.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                new ToastDialog(MainAbilitySlice.this).setText("确定
").setDuration(2000).show();
                //list.add(new Contacts("内马尔", 0));
                Contacts contacts = generateContactsFromInput();
                list.add(contacts);          //更新 list 中存储的数据
                contactsProvider.notifyDataChanged();      //更新 listcontainer
                commonDialog.remove();
```

```
            }
        });

        //采用 lamda 表达式实现按钮监听器
        btnCancel.setClickedListener((Component component) -> {
            new ToastDialog(MainAbilitySlice.this).setText("取消").setDuration(2000).show();
            commonDialog.remove();
        });


        commonDialog.setContentCustomComponent(dialogComponent);

        commonDialog.show();
    }

    @Override
    public void onClick(Component component) {
        switch (component.getId()) {
            case ResourceTable.Id_image_add:    //添加按钮点击触发
                showAddContactsDialog();
                break;
            case ResourceTable.Id_image_menu:    //菜单按钮点击触发
                new ToastDialog(MainAbilitySlice.this).setText("菜单功能待完成
").setDuration(2000).show();
                break;
        }
    }

    //通过对话框的输入页面输入的数据产生联系人
```

```
    private Contacts generateContactsFromInput() {
        Contacts contacts = new Contacts();
        HiLog.info(LABEL, "checked id =" + genderRc.getMarkedButtonId());
        if (genderRc.getMarkedButtonId() == Constant.GENDER_MAN) {
            contacts.setGender(Constant.GENDER_MAN);
        } else {
            contacts.setGender(Constant.GENDER_LADY);
        }
        contacts.setName(nameTf.getText().trim());
        contacts.setPhone(phoneTf.getText().trim());
        return contacts;
    }

    private void showToast(String msg) {
        new ToastDialog(MainAbilitySlice.this).setText(msg).setDuration(2000).show();
    }

}
```

注意：此步骤的代码添加比较多，同时还加入了 HiLog 日志和 ToastDialog 消息提示的演示。编码的时候注意编码规范，变量和方法的命名，方法的封装与调用，一个方法体代码不要过长（控制在 50 行以内），注释的添加。

操作完成这步之后，我们再来运行检验下效果，效果图如下：

录入一条数据，观察联系人列表是否被更新。

步骤小结与思考：

此步骤涉及核心知识点是自定义布局的对话框的使用，同时关联多种组件（输入框、单选按钮、按钮等）的使用，以及各种组件的监听事件的使用，以及对话框与宿主页面之间的数据交互操作。

思考：如何头脑清晰的编写出大量代码完成此功能？反复操作积累之后，梳理出流程思路，然后照着自己的思路一步步进行完善，同时穿插面向对象的思维，不要所有代码在一个方法里编写，一条路走到黑，可以先根据思路定义出方法，然后对各方法进行完善。

5、点击联系人列表查看联系人详情，以及点击自定义的返回按钮返回

（1）添加 ListContainer 组件的 Item 的点击事件

此时，由于还没有添加第二个页面，无法做页面跳转操作，可以先使用 HiLog 或者 ToastDialog 配合 Item 点击事件进行调试。

修改上个步骤编写好的 initListContainer 方法的代码，修改后的代码如下：

```
/**
 * 初始化列表
 */private void initListContainer() {
    listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
    list = getData();    //静态数据对接


    contactsProvider = new ContactsProvider(list, this);
    listContainer.setItemProvider(contactsProvider);
    //设置列表 item 的点击事件
    listContainer.setItemClickedListener(new ListContainer.ItemClickedListener() {
        @Override
        public void onItemClicked(ListContainer listContainer, Component component, int i, long
l) {
            showToast("第" + i + "行被点击");
        }
    });


}
```

此时，可以运行进行测试，看看是不是点击列表的每一行，都会弹出消息提示，并且能获取到改行的位置索引，效果图如下：

（2）新建一个联系人详情页的页面。

首先，在 layout 下新建一个布局文件 ability_slice_contacts_detail.xml 文件，代码如下

```
<?xml version="1.0" encoding="utf-8"?><DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:height="match_parent"
    ohos:width="match_parent"
    ohos:left_margin="10vp"
    ohos:orientation="vertical"
    >

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:background_element="$graphic:color_light_gray_element"
```

```
        ohos:orientation="horizontal">

        <Image
            ohos:id="$+id:image_left"
            ohos:height="64vp"
            ohos:width="64vp"
            ohos:image_src="$media:left"/>

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:text="联系人详情"
            ohos:text_size="24fp"/>
    </DirectionalLayout>

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:orientation="horizontal">

        <Image
            ohos:height="64vp"
            ohos:width="64vp"
            ohos:image_src="$media:people"/>

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
```

```
            ohos:text="姓名："
            ohos:text_size="24vp"/>

        <Text
            ohos:id="$+id:text_name"
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:text_size="24vp"/>
    </DirectionalLayout>

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:orientation="horizontal">

        <Image
            ohos:height="64vp"
            ohos:width="64vp"
            ohos:image_src="$media:gender"/>

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:text="性别："
            ohos:text_size="24vp"/>

        <Text
            ohos:id="$+id:text_gender"
```

```
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:text_size="24vp"/>
    </DirectionalLayout>

    <DirectionalLayout
        ohos:height="match_content"
        ohos:width="match_parent"
        ohos:alignment="vertical_center"
        ohos:orientation="horizontal">

        <Image
            ohos:height="64vp"
            ohos:width="64vp"
            ohos:image_src="$media:phone"/>

        <Text
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:text="电话："
            ohos:text_size="24vp"/>

        <Text
            ohos:id="$+id:text_phone"
            ohos:height="match_content"
            ohos:width="match_content"
            ohos:text_size="24vp"/>
    </DirectionalLayout></DirectionalLayout>
```

然后，在 slice 包下面新建一个 ContactsDetailSlice 类继承 AbilitySlice，在此类中要关联上刚创建的布局文件，详细代码如下：

```
package com.xdw.addressbookbyjava.slice;


import com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.constant.Constant;import
com.xdw.addressbookbyjava.model.Contacts;import
com.xdw.addressbookbyjava.utils.ToastUtil;import ohos.aafwk.ability.AbilitySlice;import
ohos.aafwk.content.Intent;import ohos.agp.components.Component;import
ohos.agp.components.Image;import ohos.agp.components.Text;


/**
 * Created by 夏德旺 on 2021/1/3 22:53
 */public class ContactsDetailSlice extends AbilitySlice {
    private Text nameText, genderText, phoneText;
    private Image leftImage;


    @Override
    protected void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_slice_contacts_detail);
        initView();
        //接收上一个页面传递过来的数据
        Contacts contacts = (Contacts) intent.getSerializableParam("contacts");
        if (contacts != null) {//
ToastUtil.showToast(this,"name="+contacts.getName());
            nameText.setText(contacts.getName());
            if (contacts.getGender() == Constant.GENDER_MAN) {
                genderText.setText(Constant.GENDER_MAN_STRING);
```

```
            } else {
                genderText.setText(Constant.GENDER_LADY_STRING);
            }
            phoneText.setText(contacts.getPhone());
        }

    }

    private void initView() {
        nameText = (Text) findComponentById(ResourceTable.Id_text_name);
        genderText = (Text) findComponentById(ResourceTable.Id_text_gender);
        phoneText = (Text) findComponentById(ResourceTable.Id_text_phone);
        leftImage = (Image) findComponentById(ResourceTable.Id_image_left);
        //左侧箭头的返回按钮，设置点击事件，并且调用系统的返回方法
        leftImage.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                ToastUtil.showToast(ContactsDetailSlice.this, "onclick");
                ContactsDetailSlice.this.onBackPressed();
            }
        });
    }}
```

这里在页面顶部自己加了一个返回图标按钮，通过调用系统 Api 的 onBackPressed()方法可以实现返回上级页面操作。

（3）在联系人列表页面对应的 MainAbilitySlice 代码中，添加页面跳转的功能

刚已经实现了 ListContainer 的 Item 点击事件，之前是使用的 ToastDialog 进行演示的，现在完善这个点击事件的代码，实现页面跳转逻辑，并且传递数据参数到联系人详情页。

修改后的详细代码如下：

```
/**
     * 初始化列表
     */
    private void initListContainer() {
        listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
        list = getData();    //静态数据对接


        contactsProvider = new ContactsProvider(list, this);
        listContainer.setItemProvider(contactsProvider);
        //设置列表 item 的点击事件
        listContainer.setItemClickedListener(new ListContainer.ItemClickedListener() {
            @Override
            public void onItemClicked(ListContainer listContainer, Component component, int i,
long l) {//                  showToast("第" + i + "行被点击");
                //点击 item 之后跳转 slice 页面，并且传递数据
                Intent intent = new Intent();
                intent.setParam("contacts", list.get(i));
                intent.setParam("name", list.get(i).getName());
                present(new ContactsDetailSlice(), intent);
            }
        });


    }
```

现在，可以运行查看效果了。如下图：

步骤小结与思考：

完成此步骤，对事件监听更加熟悉了，同时熟悉了页面跳转和传参的流程，并且更加清楚了如何分步骤实现想要的功能。

思考：可以尝试编写更多的页面了。

6、长按删除联系人列表删除联系人

（1）给 ListContainer 组件添加长按监听事件，先使用 HiLog 或者 ToastDialog 进行调试

修改之前的 initListContainer 方法的代码，详细代码如下：

```
/**
 * 初始化列表
 */
private void initListContainer() {
    listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
    list = getData();    //静态数据对接

    contactsProvider = new ContactsProvider(list, this);
    listContainer.setItemProvider(contactsProvider);
```

```
        //设置列表 item 的点击事件
        listContainer.setItemClickedListener(new ListContainer.ItemClickedListener() {
            @Override
            public void onItemClicked(ListContainer listContainer, Component component, int i,
long l) {//                  showToast("第" + i + "行被点击");
                //点击 item 之后跳转 slice 页面，并且传递数据
                Intent intent = new Intent();
                intent.setParam("contacts", list.get(i));
                intent.setParam("name", list.get(i).getName());
                present(new ContactsDetailSlice(), intent);
            }
        });

        //设置列表 item 的长按点击事件，弹出系统自带对话框，点击确认删除联系人
        listContainer.setItemLongClickedListener(new ListContainer.ItemLongClickedListener() {
            @Override
            public boolean onItemLongClicked(ListContainer listContainer, Component
component, int i, long l) {
                showToast("item" + i + "长按操作");
                return false;
            }
        });
    }
```

此时可以运行测试下长按操作功能，运行效果如下图：

（2）长按 item 之后，弹出对话框提醒用户是否进行删除操作，点击确认按钮删除 item 并刷新列表页面

这个步骤的对话框，此处就直接采用系统自带对话框样式，之前自定义对话框的操作已经在前面的步骤操作过了。

首先，在 Item 长按监听事件下生成对话框，给对话框的确定和取消按钮绑定监听事件，同时给 MainAbilitySlice 定义一个成员变量 currentIndex 标记当前长按点击的 item 的位置。然后在对话框的确定按钮的点击事件中进行 ListContainer 的数据源的删除和显示的刷新操作。

经过修改后的 MainAbilitySlice 的完整代码如下：

```
package com.xdw.addressbookbyjava.slice;


import com.xdw.addressbookbyjava.MyApplication;import
com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.constant.Constant;import
```

```
com.xdw.addressbookbyjava.dao.ContactsDao;import
com.xdw.addressbookbyjava.dao.impl.ContactsDaoImpl;import
com.xdw.addressbookbyjava.model.Contacts;import
com.xdw.addressbookbyjava.provider.ContactsProvider;import
ohos.aafwk.ability.AbilitySlice;import ohos.aafwk.ability.OnClickListener;import
ohos.aafwk.content.Intent;import ohos.aafwk.content.IntentParams;import
ohos.agp.components.*;import ohos.agp.window.dialog.CommonDialog;import
ohos.agp.window.dialog.IDialog;import ohos.agp.window.dialog.ToastDialog;import
ohos.hiviewdfx.HiLog;import ohos.hiviewdfx.HiLogLabel;


import java.util.ArrayList;import java.util.List;


public class MainAbilitySlice extends AbilitySlice implements Component.ClickedListener {
    public static final String TAG = "MainAbilitySlice";
    private static final HiLogLabel LABEL = new HiLogLabel(HiLog.DEBUG, 0, "TAG");
    private Image addContactsBtn;      //添加联系人按钮
    private Image menuBtn;             //菜单按钮
    private List<Contacts> list;       //列表数据源
    private ListContainer listContainer;     //list 列表
    private ContactsProvider contactsProvider;      //list 列表绑定的 provider
    private int currentIndex;     //当前 item 的索引位
    private TextField nameTf;       //联系人姓名输入框
    private RadioContainer genderRc;      //联系人性别单选框组合容器
    private TextField phoneTf;   //联系人电话号码输入框
    private RadioButton manRb;   //单选按钮，男
    private RadioButton ladyRb; //单选按钮，女
    private ContactsDao contactsDao = new ContactsDaoImpl();


    //定义系统自带的对话框相关的操作 start
```

```
    private static final int DIALOG_BUTTON_CANCEL = 1;
    private static final int DIALOG_BUTTON_CONFIRM = 2;
    private IDialog.ClickedListener clickedListener = new IDialog.ClickedListener() {
        @Override
        public void onClick(IDialog iDialog, int i) {
            switch (i) {
                case DIALOG_BUTTON_CANCEL://                    showToast("对话框取消");
                    iDialog.remove();
                    break;
                case DIALOG_BUTTON_CONFIRM://                    showToast("对话框确认");
                    if (list != null && list.size() > currentIndex) {
                        list.remove(currentIndex);  //list 数据源中删除该项联系人
                        contactsProvider.notifyDataSetItemRemoved(currentIndex);  //刷新listcontainer 显示
                        iDialog.remove();           //删除这里有点小 bug，可能是系统问题，这里必须还要在点一下手机页面 listcontainer 才能刷新显示
                    }
                    break;
            }
        }
    };
    //定义系统自带的对话框相关的操作 end

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);
```

```
        initView();
        initListContainer();

    }

    @Override
    public void onActive() {
        super.onActive();
    }

    @Override
    public void onForeground(Intent intent) {
        super.onForeground(intent);
    }

    /**
     * 初始化视图组件以及绑定监听事件
     */
    private void initView() {
        addContactsBtn = (Image) findComponentById(ResourceTable.Id_image_add);
        menuBtn = (Image) findComponentById(ResourceTable.Id_image_menu);
        addContactsBtn.setClickedListener(this);
        menuBtn.setClickedListener(this);
    }

    /**
     * 初始化列表
     */
    private void initListContainer() {
```

```
        listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
        list = getData();    //静态数据对接


        contactsProvider = new ContactsProvider(list, this);
        listContainer.setItemProvider(contactsProvider);
        //设置列表 item 的点击事件
        listContainer.setItemClickedListener(new ListContainer.ItemClickedListener() {
            @Override
            public void onItemClicked(ListContainer listContainer, Component component, int i,
long l) {//                  showToast("第" + i + "行被点击");
                //点击 item 之后跳转 slice 页面，并且传递数据
                Intent intent = new Intent();
                intent.setParam("contacts", list.get(i));
                intent.setParam("name", list.get(i).getName());
                present(new ContactsDetailSlice(), intent);
            }
        });


        //设置列表 item 的长按点击事件，弹出系统自带对话框，点击确认删除联系人
        listContainer.setItemLongClickedListener(new ListContainer.ItemLongClickedListener() {
            @Override
            public boolean onItemLongClicked(ListContainer listContainer, Component
component, int i, long l) {
                showToast("item" + i + "长按操作");
                currentIndex = i;
                CommonDialog commonDialog = new CommonDialog(MainAbilitySlice.this);
                commonDialog.setContentText("确认删除该联系人吗？");
                commonDialog.setButton(DIALOG_BUTTON_CANCEL, "取消",
clickedListener);
```

```
                commonDialog.setButton(DIALOG_BUTTON_CONFIRM, "确定",
clickedListener);
                commonDialog.show();
                return false;
            }
        });
    }

    //生成静态的列表数据进行模拟，在对接 sqlite 或者服务端之后不再使用
    private List<Contacts> getData() {
        List<Contacts> list = new ArrayList<>();
        String[] names = {"克里斯迪亚洛罗纳尔多", "王霜", "梅西", "孙雯", "莱万多夫斯基", "玛
塔", "樱木花道", "赤木晴子"};
        for (int i = 0; i <= 7; i++) {
            list.add(new Contacts(names[i], i % 2));
        }
        return list;
    }

    //创建并显示自定义的添加联系人操作的对话框
    private void showAddContactsDialog() {
        CommonDialog commonDialog = new CommonDialog(this);

        Component dialogComponent = LayoutScatter.getInstance(getContext())
                .parse(ResourceTable.Layout_dialog_add_contacts, null, true);

//          commonDialog.setSize(800, 600);       //设置对话框尺寸
        Button btnConfirm = (Button)
dialogComponent.findComponentById(ResourceTable.Id_btn_confirm);
```

```
        Button btnCancel = (Button)
dialogComponent.findComponentById(ResourceTable.Id_btn_cancel);
        nameTf = (TextField)
dialogComponent.findComponentById(ResourceTable.Id_tf_name);
        phoneTf = (TextField)
dialogComponent.findComponentById(ResourceTable.Id_tf_phone);
        genderRc = (RadioContainer)
dialogComponent.findComponentById(ResourceTable.Id_rc_gender);
        manRb = (RadioButton)
dialogComponent.findComponentById(ResourceTable.Id_rb_man);
        ladyRb = (RadioButton)
dialogComponent.findComponentById(ResourceTable.Id_rb_lady);
        manRb.setChecked(true);
        genderRc.setMarkChangedListener(new
RadioContainer.CheckedStateChangedListener() {
            @Override
            public void onCheckedChanged(RadioContainer radioContainer, int i) {


            }
        });
        //采用传统的匿名内部类实现按钮监听器
        btnConfirm.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                new ToastDialog(MainAbilitySlice.this).setText("确定
").setDuration(2000).show();
                //list.add(new Contacts("内马尔", 0));
                Contacts contacts = generateContactsFromInput();
                list.add(contacts);         //更新 list 中存储的数据
```

```
                contactsProvider.notifyDataChanged();      //更新 listcontainer
                commonDialog.remove();
            }
        });

        //采用 lamda 表达式实现按钮监听器
        btnCancel.setClickedListener((Component component) -> {
            new ToastDialog(MainAbilitySlice.this).setText("取消").setDuration(2000).show();
            commonDialog.remove();
        });


        commonDialog.setContentCustomComponent(dialogComponent);

        commonDialog.show();
    }

    @Override
    public void onClick(Component component) {
        switch (component.getId()) {
            case ResourceTable.Id_image_add:    //添加按钮点击触发
                showAddContactsDialog();
                break;
            case ResourceTable.Id_image_menu:    //菜单按钮点击触发
                new ToastDialog(MainAbilitySlice.this).setText("菜单功能待完成
").setDuration(2000).show();
                break;
        }
    }
```

```
    //通过对话框的输入页面输入的数据产生联系人
    private Contacts generateContactsFromInput() {
        Contacts contacts = new Contacts();
        HiLog.info(LABEL, "checked id =" + genderRc.getMarkedButtonId());
        if (genderRc.getMarkedButtonId() == Constant.GENDER_MAN) {
            contacts.setGender(Constant.GENDER_MAN);
        } else {
            contacts.setGender(Constant.GENDER_LADY);
        }
        contacts.setName(nameTf.getText().trim());
        contacts.setPhone(phoneTf.getText().trim());
        return contacts;
    }

    private void showToast(String msg) {
        new ToastDialog(MainAbilitySlice.this).setText(msg).setDuration(2000).show();
    }

}
```

此时，可以运行测试效果了，如下图：

注意：由于远端模拟器原因，此时点击确定删除后，可能需要用鼠标点击下模拟器页面，才会出现 Item 被删除。

步骤小结与思考：

完成此步骤，进一步夯实了 ListContainer 组件的关键 api 操作和监听事件，同时熟悉了系统对话框的 API 使用。

思考：嫌弃系统自带的对话框太丑？参照之前自定义的对话框步骤自己编写一个漂亮的对话框取代它吧。

7、对接 Sqlite，实现数据持久化与数据动态更新

（1）在 dao 包下面创建一个数据库操作接口 ContactsDao

代码如下：

```
package com.xdw.addressbookbyjava.dao;


import com.xdw.addressbookbyjava.model.Contacts;
```

```
import java.util.List;

/**
 * Created by 夏德旺 on 2021/1/5 21:14
 */public interface ContactsDao {
    long insertContacts(Contacts contacts);    //插入联系人

    long deleteContacts(int id);               //根据 id 删除联系人

    List<Contacts> queryContactsById(int id);     //根据 id 查询联系人

    long updateContacts(Contacts contacts); //更新查询联系人

    List<Contacts> getAllContacts();           //获取所有联系人}
```

（2）在 Hap 的入口类 MyApplication 中集成 Sqlite 的关键 API 操作，并封装出一个对外调用的方法

详细代码如下：

```
package com.xdw.addressbookbyjava;

import ohos.aafwk.ability.AbilityPackage;import ohos.data.DatabaseHelper;import
ohos.data.rdb.RdbOpenCallback;import ohos.data.rdb.RdbStore;import
ohos.data.rdb.StoreConfig;

public class MyApplication extends AbilityPackage {
    //该项目案例开发了两个版本进行演示，一个无数据库的静态数据版本，一个带数据库操作的
sqlite 版本
    public static final int STATIC_DATA_VERSION =1;
```

```
    public static final int SQLITE_DATA_VERSION =2;
    //定义 DEV_DATA_VERSION，通过切换该版本号来快速切换静态数据版和数据库版本
    public static final int DEV_DATA_VERSION = SQLITE_DATA_VERSION;
    private static RdbStore store;   //在入口类中定义 store 对象，用来进行 sqlite api 操作的关键对象
    private static final RdbOpenCallback callback = new RdbOpenCallback() {
        @Override
        public void onCreate(RdbStore store) {
            store.executeSql("CREATE TABLE IF NOT EXISTS contacts (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, gender INTEGER,phone TEXT NOT NULL, groupId INTEGER)");
        }

        @Override
        public void onUpgrade(RdbStore store, int oldVersion, int newVersion) {
        }
    };

    @Override
    public void onInitialize() {
        super.onInitialize();
        //在程序入口处初始化数据库信息
        StoreConfig config = StoreConfig.newDefaultConfig("RdbContacts.db");
//RdbContacts.db 为数据库名称，不存在则自动创建
        DatabaseHelper helper = new DatabaseHelper(this);
        store = helper.getRdbStore(config, 1, callback, null);   //生成 store 对象
    }

    /**
```

```
     * 定义获取 store 对象的方法，后面可以随时调用进行数据库相关操作
     *
     * @return
     */
    public static RdbStore getRdbStore() {
        return store;
    }}
```

注意：这里为了灵活的和前面静态数据版本功能加以对比和切换，定义了一个配置项常量 **DEV_DATA_VERSION**，后面可以通过切换它的值，快速切换静态数据版本和数据库版本进行测试。

（3）新建一个 **ContactsDaoImpl** 类实现 **ContactsDao** 接口。

详细代码如下：

```
package com.xdw.addressbookbyjava.dao.impl;


import com.xdw.addressbookbyjava.MyApplication;import
com.xdw.addressbookbyjava.dao.ContactsDao;import
com.xdw.addressbookbyjava.model.Contacts;import ohos.data.rdb.RdbPredicates;import
ohos.data.rdb.ValuesBucket;import ohos.data.resultset.ResultSet;import
ohos.hiviewdfx.HiLog;import ohos.hiviewdfx.HiLogLabel;


import java.util.ArrayList;import java.util.List;


/**
 * Created by 夏德旺 on 2021/1/5 21:18
 */public class ContactsDaoImpl implements ContactsDao {
    public static final String TAG = "ContactsDaoImpl";
    private static final HiLogLabel LABEL = new HiLogLabel(HiLog.DEBUG, 0, TAG);
```

```
    @Override
    public long insertContacts(Contacts contacts) {
        ValuesBucket values = new ValuesBucket();
        values.putString("name", contacts.getName());
        values.putInteger("gender", contacts.getGender());
        values.putString("phone", contacts.getPhone());
        long id = MyApplication.getRdbStore().insert("contacts", values);
        return id;
    }

    @Override
    public long deleteContacts(int id) {
        RdbPredicates rdbPredicates = new RdbPredicates("contacts").equalTo("id", id);
        HiLog.error(LABEL,"deleteid="+id);
        int num = MyApplication.getRdbStore().delete(rdbPredicates);
        return num;
    }

    @Override
    public List<Contacts> queryContactsById(int id) {
        List<Contacts> contactsList = new ArrayList<>();
        String[] columns = new String[]{"id", "name", "gender", "phone"};
        RdbPredicates rdbPredicates = new RdbPredicates("contacts").equalTo("id",
id).orderByAsc("id");
        ResultSet resultSet = MyApplication.getRdbStore().query(rdbPredicates, columns);
        while (resultSet.goToNextRow()) {
            Contacts contacts = new Contacts();
            HiLog.error(LABEL, "id="+resultSet.getString(0));
            contacts.setId(resultSet.getInt(0));
```

```
            contacts.setName(resultSet.getString(1));
            contacts.setGender(resultSet.getInt(2));
            contacts.setPhone(resultSet.getString(3));
            contactsList.add(contacts);
        }
        return contactsList;
    }

    @Override
    public long updateContacts(Contacts contacts) {
        return 0;
    }

    @Override
    public List<Contacts> getAllContacts() {
        List<Contacts> contactsList = new ArrayList<>();
        String[] columns = new String[]{"id", "name", "gender", "phone"};
        RdbPredicates rdbPredicates = new RdbPredicates("contacts").orderByAsc("id");
        ResultSet resultSet = MyApplication.getRdbStore().query(rdbPredicates, columns);
        while (resultSet.goToNextRow()) {
            Contacts contacts = new Contacts();
            HiLog.error(LABEL, "id="+resultSet.getString(0));
            contacts.setId(resultSet.getInt(0));
            contacts.setName(resultSet.getString(1));
            contacts.setGender(resultSet.getInt(2));
            contacts.setPhone(resultSet.getString(3));
            contactsList.add(contacts);
        }
        return contactsList;
```

```
}}
```

注意：这样就通过 Dao 接口的方式将页面 Slice 和 Sqlite 的操作进行了解耦，让我们可以用通常 MVC 操作数据库的思维来进行操作。后期还可以轻松切换成与服务器对接，而不用怎么动 Slice 里面的代码。

（4）在 MainAbilitySlice 中调用 dao 接口的功能。

最终集成好所有功能的 MainAbilitySlice 的代码如下：

```
package com.xdw.addressbookbyjava.slice;


import com.xdw.addressbookbyjava.MyApplication;import
com.xdw.addressbookbyjava.ResourceTable;import
com.xdw.addressbookbyjava.constant.Constant;import
com.xdw.addressbookbyjava.dao.ContactsDao;import
com.xdw.addressbookbyjava.dao.impl.ContactsDaoImpl;import
com.xdw.addressbookbyjava.model.Contacts;import
com.xdw.addressbookbyjava.provider.ContactsProvider;import
ohos.aafwk.ability.AbilitySlice;import ohos.aafwk.ability.OnClickListener;import
ohos.aafwk.content.Intent;import ohos.aafwk.content.IntentParams;import
ohos.agp.components.*;import ohos.agp.window.dialog.CommonDialog;import
ohos.agp.window.dialog.IDialog;import ohos.agp.window.dialog.ToastDialog;import
ohos.hiviewdfx.HiLog;import ohos.hiviewdfx.HiLogLabel;


import java.util.ArrayList;import java.util.List;


public class MainAbilitySlice extends AbilitySlice implements Component.ClickedListener {
    public static final String TAG = "MainAbilitySlice";
    private static final HiLogLabel LABEL = new HiLogLabel(HiLog.DEBUG, 0, "TAG");
    private Image addContactsBtn;      //添加联系人按钮
    private Image menuBtn;             //菜单按钮
```

```
    private List<Contacts> list;      //列表数据源
    private ListContainer listContainer;     //list 列表
    private ContactsProvider contactsProvider;      //list 列表绑定的 provider
    private int currentIndex;     //当前 item 的索引位
    private TextField nameTf;       //联系人姓名输入框
    private RadioContainer genderRc;      //联系人性别单选框组合容器
    private TextField phoneTf;   //联系人电话号码输入框
    private RadioButton manRb;   //单选按钮，男
    private RadioButton ladyRb; //单选按钮，女
    private ContactsDao contactsDao = new ContactsDaoImpl();


    //定义系统自带的对话框相关的操作 start
    private static final int DIALOG_BUTTON_CANCEL = 1;
    private static final int DIALOG_BUTTON_CONFIRM = 2;
    private IDialog.ClickedListener clickedListener = new IDialog.ClickedListener() {
        @Override
        public void onClick(IDialog iDialog, int i) {
            switch (i) {
                case DIALOG_BUTTON_CANCEL://                    showToast("对话框
取消");
                    iDialog.remove();
                    break;
                case DIALOG_BUTTON_CONFIRM://                    showToast("对话框
确认");
                    if (list != null && list.size() > currentIndex) {
                        if (MyApplication.DEV_DATA_VERSION ==
MyApplication.SQLITE_DATA_VERSION) {
                            contactsDao.deleteContacts(list.get(currentIndex).getId()); //从数
据库删除联系人
```

```
                        }
                        list.remove(currentIndex);  //list 数据源中删除该项联系人
                        contactsProvider.notifyDataSetItemRemoved(currentIndex);  //刷新
listcontainer 显示
                        iDialog.remove();           //删除这里有点小 bug，可能是系统问题，
这里必须还要在点一下手机页面 listcontainer 才能刷新显示
                    }
                    break;
                }
            }
        };
        //定义系统自带的对话框相关的操作 end

        @Override
        public void onStart(Intent intent) {
            super.onStart(intent);
            super.setUIContent(ResourceTable.Layout_ability_main);
            initView();
            initListContainer();

        }

        @Override
        public void onActive() {
            super.onActive();
        }

        @Override
        public void onForeground(Intent intent) {
```

```
        super.onForeground(intent);
    }

    /**
     * 初始化视图组件以及绑定监听事件
     */
    private void initView() {
        addContactsBtn = (Image) findComponentById(ResourceTable.Id_image_add);
        menuBtn = (Image) findComponentById(ResourceTable.Id_image_menu);
        addContactsBtn.setClickedListener(this);
        menuBtn.setClickedListener(this);
    }

    /**
     * 初始化列表
     */
    private void initListContainer() {
        listContainer = (ListContainer) findComponentById(ResourceTable.Id_list_contacts);
        if (MyApplication.DEV_DATA_VERSION == MyApplication.SQLITE_DATA_VERSION) {
            list = getSqliteData(); //sqlite 数据库数据对接
        } else {
            list = getData();    //静态数据对接
        }

        contactsProvider = new ContactsProvider(list, this);
        listContainer.setItemProvider(contactsProvider);
        //设置列表 item 的点击事件
        listContainer.setItemClickedListener(new ListContainer.ItemClickedListener() {
```

```
            @Override
            public void onItemClicked(ListContainer listContainer, Component component, int i,
long l) {//                    showToast("第" + i + "行被点击");
                //点击 item 之后跳转 slice 页面，并且传递数据
                Intent intent = new Intent();
                intent.setParam("contacts", list.get(i));
                intent.setParam("name", list.get(i).getName());
                present(new ContactsDetailSlice(), intent);
            }
        });

        //设置列表 item 的长按点击事件，弹出系统自带对话框，点击确认删除联系人
        listContainer.setItemLongClickedListener(new ListContainer.ItemLongClickedListener() {
            @Override
            public boolean onItemLongClicked(ListContainer listContainer, Component
component, int i, long l) {
                showToast("item" + i + "长按操作");
                currentIndex = i;
                CommonDialog commonDialog = new CommonDialog(MainAbilitySlice.this);
                commonDialog.setContentText("确认删除该联系人吗？");
                commonDialog.setButton(DIALOG_BUTTON_CANCEL, "取消",
clickedListener);
                commonDialog.setButton(DIALOG_BUTTON_CONFIRM, "确定",
clickedListener);
                commonDialog.show();
                return false;
            }
        });
    }
```

```
    //生成静态的列表数据进行模拟，在对接 sqlite 或者服务端之后不再使用
    private List<Contacts> getData() {
        List<Contacts> list = new ArrayList<>();
        String[] names = {"克里斯迪亚洛罗纳尔多", "王霜", "梅西", "孙雯", "莱万多夫斯基", "玛塔", "樱木花道", "赤木晴子"};
        for (int i = 0; i <= 7; i++) {
            list.add(new Contacts(names[i], i % 2));
        }
        return list;
    }

    //创建并显示自定义的添加联系人操作的对话框
    private void showAddContactsDialog() {
        CommonDialog commonDialog = new CommonDialog(this);

        Component dialogComponent = LayoutScatter.getInstance(getContext())
                .parse(ResourceTable.Layout_dialog_add_contacts, null, true);

//          commonDialog.setSize(800, 600);       //设置对话框尺寸
        Button btnConfirm = (Button)
dialogComponent.findComponentById(ResourceTable.Id_btn_confirm);
        Button btnCancel = (Button)
dialogComponent.findComponentById(ResourceTable.Id_btn_cancel);
        nameTf = (TextField)
dialogComponent.findComponentById(ResourceTable.Id_tf_name);
        phoneTf = (TextField)
dialogComponent.findComponentById(ResourceTable.Id_tf_phone);
        genderRc = (RadioContainer)
```

```
dialogComponent.findComponentById(ResourceTable.Id_rc_gender);
        manRb = (RadioButton)
dialogComponent.findComponentById(ResourceTable.Id_rb_man);
        ladyRb = (RadioButton)
dialogComponent.findComponentById(ResourceTable.Id_rb_lady);
        manRb.setChecked(true);
        genderRc.setMarkChangedListener(new
RadioContainer.CheckedStateChangedListener() {
            @Override
            public void onCheckedChanged(RadioContainer radioContainer, int i) {


            }
        });
        //采用传统的匿名内部类实现按钮监听器
        btnConfirm.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                new ToastDialog(MainAbilitySlice.this).setText("确定
").setDuration(2000).show();
                //list.add(new Contacts("内马尔", 0));
                Contacts contacts = generateContactsFromInput();
                if (MyApplication.DEV_DATA_VERSION ==
MyApplication.SQLITE_DATA_VERSION) {
                    contactsDao.insertContacts(contacts); //添加到数据库中，当没有对接 sqlite
而是使用静态数据测试的时候，没有这一步
                }
                list.add(contacts);        //更新 list 中存储的数据
                contactsProvider.notifyDataChanged();      //更新 listcontainer
                commonDialog.remove();
```

```
            }
        });

        //采用 lamda 表达式实现按钮监听器
        btnCancel.setClickedListener((Component component) -> {
            new ToastDialog(MainAbilitySlice.this).setText("取消").setDuration(2000).show();
            commonDialog.remove();
        });


        commonDialog.setContentCustomComponent(dialogComponent);

        commonDialog.show();
    }

    @Override
    public void onClick(Component component) {
        switch (component.getId()) {
            case ResourceTable.Id_image_add:    //添加按钮点击触发
                showAddContactsDialog();
                break;
            case ResourceTable.Id_image_menu:    //菜单按钮点击触发
                new ToastDialog(MainAbilitySlice.this).setText("菜单功能待完成
").setDuration(2000).show();
                break;
        }
    }

    //通过对话框的输入页面输入的数据产生联系人
```

```
private Contacts generateContactsFromInput() {
    Contacts contacts = new Contacts();
    HiLog.info(LABEL, "checked id =" + genderRc.getMarkedButtonId());
    if (genderRc.getMarkedButtonId() == Constant.GENDER_MAN) {
        contacts.setGender(Constant.GENDER_MAN);
    } else {
        contacts.setGender(Constant.GENDER_LADY);
    }
    contacts.setName(nameTf.getText().trim());
    contacts.setPhone(phoneTf.getText().trim());
    return contacts;
}

private void showToast(String msg) {
    new ToastDialog(MainAbilitySlice.this).setText(msg).setDuration(2000).show();
}

//获取 sqlite 数据库中的数据，集成 sqlite 之后调用这个方法取代之前的静态数据方法
private List<Contacts> getSqliteData() {
    return contactsDao.getAllContacts();
}

/**
 * 测试数据库是否生效
 */
private void testInsertContacts() {
    Contacts contacts = new Contacts();
    contacts.setName("xiadewang");
    contacts.setPhone("13437124333");
```

```
        contacts.setGender(0);
        contactsDao.insertContacts(contacts);
    }}
```

完成所有步骤之后，请做个完整的功能测试。

步骤小结与思考：

完成此步骤，熟悉了 sqlite 集成的相关操作以及 API 的调用，同时对设计模式加深了了解和使用。

思考：这里的 sqlite 操作采用的原生的 ResultSet 的 api 调用，操作数据转化比较麻烦，尝试阅读官方文档进行修改成更加简便的 ORM 模型的操作方式。

(七) 扩展内容

1. 本实验中，编辑修改联系人信息的功能暂未实现，留给大家自行实现；
2. 本实验中，联系人信息录入的数据元素相对较少，请大家扩充联系人其它信息，比如分组，然后进行功能实现。

(八) 源代码获取

https://gitee.com/xdw1019/wd-project