第 5 章　Android 控件进阶操作实战

5.1　控之经典——ListView

ListView 是最经典的控件之一，虽然现在其江山地位不稳，将要被 RecylerView 取代，但设计理念是很经典的，而且很多程序员还是习惯了 ListView，因此我们还需要对 ListView 进行深入学习。ListView 内容非常多，读者要有足够的耐心进行学习，每一个功能点都有可能应用到项目中。

ListView 经常被用在列表显示上，每一个列表项都具有相同的布局，一个 ListView 通常都有三个要素组成：

- ListView 控件。
- 适配器类，用到了设计模式中的适配器模式，它是视图和数据之间的桥梁，负责提供对数据的访问，生成每一个列表项对应的 View。常用的适配器类有 ArrayAdapter、SimpleAdapter 和 SimpleCursorAdapter。
- 数据源。

当然最重要、最复杂的部分就是适配器类的编写和设计，在一些复杂的界面，常常需要对适配器类进行相关逻辑处理。

ListView 的常用属性如表 5.1 所示。

表 5.1　ListView 的常用属性

属　性	说　明
android:divider	子项分割线
android:dividerHeight	分割线高度
android:listSelector	子项单击效果
android:scrollbars	滑动条

ListView 的常用方法如表 5.2 所示。

表 5.2　ListView 的常用方法

方　法	说　明
addFooterView(View v)	在列表尾部加入一个 View
addHeaderView(View v)	在列表头部加入一个 View
setAdapter(ListAdapter adapter)	设置适配器
setDivider(Drawable divider)	设置子项分隔栏
setDividerHeight(int height)	设置分隔栏高度

5.1.1　ArrayAdapter 适配器

ListView 的数据渲染都需要借助适配器来完成，首先看一下结合最简单的 ArrayAdapter 来实现 ListView。

主布局文件（activity_main.xml）代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListView
        android:id="@+id/lv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:divider="@android:color/holo_red_dark"
        android:dividerHeight="3dp"
        android:scrollbars="none"/>
</RelativeLayout>
```

上述代码设置了 divider 属性，在 ListView 的子项之间添加分隔栏；设置了 dividerHeight 属性，决定了分隔栏的高度；将 scrollbars 属性的值设置为 none 表示上下拖动时在右侧没有滑动条。

MainActivity.java 代码如下：

```
public class MainActivity extends Activity {
    private ListView mListView;
    private String mDatas[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday", "Sunday", "Monday", "Tuesday",
                "Wednesday","Thursday", "Friday", "Saturday"};// 准备数据源
    private ArrayAdapter<String> mAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);// 隐藏标题栏
        setContentView(R.layout.activity_main);
        mListView = (ListView) findViewById(R.id.lv);
        // 实例化 ArrayAdapter
        mAdapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, mDatas);
        // 设置适配器
        mListView.setAdapter(mAdapter);
    }
}
```

创建一个 ArrayAdapter 对象需要传入三个参数：上下文对象、子项布局 id（用到了 Android 内置的 list 布局）、数据源。上述代码中 ArrayAdapter 的数据源传入的是字符数组，最后调用 ListView 的 setAdapter 方法为 ListView 设置适配器。

运行实例，如图 5.1 所示。

可以看出，每个子项之间存在分隔栏，上下拖动 ListView 时最右边也不会有滑动条出现。

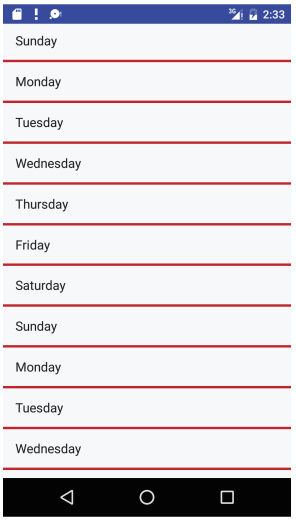

图 5.1 ListView 之 ArrayAdapter

5.1.2 SimpleAdapter 适配器

ArrayAdapter 适用于信息显示比较单一的场景，若显示项中包含多种形式的数据，就不太适用了。下面介绍可以适配多种数据类型的适配器类 SimpleAdapter 的使用方法。

当存在多种数据类型时首先要考虑布局问题，因此首先要设置子项目布局（item_layout.xml）文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/img"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:src="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="50dp"
        android:layout_marginLeft="50dp"
        android:gravity="center"
        android:text="hello"
        android:textSize="28sp" />
</LinearLayout>
```

上述代码采用线性布局，设置其 orientation 属性为 horizontal（水平布局），添加了一个 ImageView 控件用于显示图片，添加了一个 TextView 用于显示文本。

主布局文件（activity_main.xml）代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/listview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

上述代码在主布局中添加了一个 ListView 控件并设置了 id 属性，设置宽、高属性的属性值都是 match_parent。

MainActivity.java 代码如下：

```
public class MainActivity extends Activity {
    private ListView mListView;
    private SimpleAdapter mSimpleAdapter;
    private List<Map<String, Object>> mDatas = new ArrayList<Map<String,
    Object>>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mListView = (ListView) findViewById(R.id.listview);
        //初始化数据集
        initDatas();
        //实例化 SimpleAdapter
        mSimpleAdapter = new SimpleAdapter(
                this,
                mDatas,
                R.layout.item_layout, new String[]{"img", "name"},
                new int[]{R.id.img, R.id.tv});
        //设置配置器
        mListView.setAdapter(mSimpleAdapter);
    }

    private void initDatas() {
        Map map1 = new HashMap();
        map1.put("img", R.drawable.fish);
        map1.put("name", "小金鱼");
        Map map2 = new HashMap();
        map2.put("img", R.drawable.horse);
        map2.put("name", "千里马");
        Map map3 = new HashMap();
        map3.put("img", R.drawable.mouse);
        map3.put("name", "米老鼠");
        mDatas.add(map1);
        mDatas.add(map2);
        mDatas.add(map3);
```

```
        }
    }
```

SimpleAdapter 的构造函数如下：

```
SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int
resource, String[] from, int[] to)
```

实例化 SimpleAdapter 时要传入如下参数：

- context：即上下文对象；
- data：一个包裹 Map 集合的 List 数据集，这里传入 datas，即 initDatas 方法初始化的数据集；
- resource：子项布局文件，这里是自定义的 item_layout.xml 文件，传入 R.layout.animal_layout；
- from：一个字符串数组，字符串指的是 Map 中的键值，这里有两个键，即 img 和 name；
- to：一个 int 型数组，表示子项布局中对应控件的 id，这里传入 ImageView 的 id（R.id.img）和 TextView 的 id（R.id.tv）即可。

运行实例，如图 5.2 所示。

图 5.2 ListView 之 SimpleAdapter

5.1.3 BaseAdapter 适配器

上面讲解了 SimpleAdapter 作为 ListView 的适配器，通过源码可以看出 SimpleAdapter 继承自 BaseAdapter，也就是说这个 SimpleAdapter 可以看作是 Android 帮我们实现的适配器，下面研究如何通过继承 BaseAdapter 实现自定义的适配器。

主布局文件（activity_main.xml）代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

上述代码在相对布局中添加了一个 ListView 控件，设置了 id 属性为 lv，并添加了宽、高属性为 match_parent。

子项布局代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView
```

```
        android:id="@+id/img"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginLeft="20dp"
        android:src="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="50dp"
        android:layout_marginLeft="50dp"
        android:gravity="center"
        android:text="hello"
        android:textSize="28sp" />
</LinearLayout>
```

子布局和 SimpleAdapter 的布局一致。

BaseAdapter 的数据源可能比较复杂，因此这里创建一个 JavaBean 类对数据进行封装：

```
public class Animal {
    public Animal(String animal, int imgId) {
        this.animal = animal;
        this.imgId = imgId;
    }

    private String animal;
    private int imgId;

    public String getAnimal() {
        return animal;
    }

    public void setAnimal(String animal) {
        this.animal = animal;
    }

    public int getImgId() {
        return imgId;
    }

    public void setImgId(int imgId) {
        this.imgId = imgId;
    }
}
```

这个类中封装了两个属性：String 型的动物名和 int 型的图片 id，添加了一个包含这两个属性的构造方法并设置了相应的 Setter 和 Getter 方法。

下面看一下自定义的适配器类，它继承自 BaseAdapter，代码如下：

```
public class AnimalAdapter extends BaseAdapter {
    private Context context;
    private List<Animal> datas;
```

```
    //构造函数需要传入两个必要的参数：上下文对象和数据源
    public AnimalAdapter(Context context, List<Animal> datas) {
        this.context = context;
        this.datas = datas;
    }
    //返回子项的个数
    @Override
    public int getCount() {
        return datas.size();
    }
    //返回子项对应的对象
    @Override
    public Object getItem(int position) {
        return datas.get(position);
    }
    //返回子项的下标
    @Override
    public long getItemId(int position) {
        return position;
    }
    //返回子项视图
    @Override
    public View getView(int position, View convertView, ViewGroup
    parent) {
        Animal animal = (Animal) getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(context).inflate(R.layout.item_
            layout, null);
            viewHolder = new ViewHolder();
            viewHolder.animalImage = (ImageView) view.findViewById(R.
             id.img);
            viewHolder.animalName = (TextView) view.findViewById(R.
             id.tv);
            view.setTag(viewHolder);
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag();
        }
        viewHolder.animalName.setText(animal.getAnimal());
        viewHolder.animalImage.setImageResource(animal.getImgId());
        return view;
    }
    //创建ViewHolder类
    class ViewHolder {
        ImageView animalImage;
        TextView animalName;
    }
}
```

自定义的AnimalAdapter类继承自BaseAdapter类，必须要覆写[①]四个方法，每个方法的具体含义已经在代码中做了注释。此外，为了提高加载效率，这里创建了内部类ViewHolder，可以避免每次调用getView方法时都要通过findViewById方法去实例化控件，可以提高运行效率。

MainActivity.java代码如下：

```
public class MainActivity extends Activity {
    private ListView mListView;
    private List<Animal> datas = new ArrayList<Animal>();
    private AnimalAdapter mAnimalAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //隐藏标题栏
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        //初始化数据源
        initDatas();
        mListView = (ListView) findViewById(R.id.lv);
        mAnimalAdapter = new AnimalAdapter(this, datas);
        mListView.setAdapter(mAnimalAdapter);
        mListView.setOnItemClickListener(new AdapterView.
        OnItemClickListener() {
            @Override
           public void onItemClick(AdapterView<?> parent, View view,
           int position, long id) {
                Toast.makeText(MainActivity.this,
                        "您单击了" + datas.get(position).getAnimal(),
                        Toast.LENGTH_SHORT).show();
            }
        });
    }

    private void initDatas() {
        Animal animal0 = new Animal("兔八哥", R.drawable.rabbit);
        Animal animal1 = new Animal("眼镜蛇", R.drawable.snack);
        Animal animal2 = new Animal("小金鱼", R.drawable.fish);
        Animal animal3 = new Animal("千里马", R.drawable.horse);
        Animal animal4 = new Animal("米老鼠", R.drawable.mouse);
        Animal animal5 = new Animal("大国宝", R.drawable.panda);
        datas.add(animal0);
        datas.add(animal1);
        datas.add(animal2);
        datas.add(animal3);
        datas.add(animal4);
        datas.add(animal5);
    }
}
```

① “覆写”为Java里的术语。

上述代码为 ListView 设置了 setOnItemClickListener 方法监听单项单击事件，采用匿名内部类的方式实现了 AdapterView.OnItemClickListener 接口并覆写了其 onItemClick 方法，由参数 positon 通过 List 的 get 方法并传入 Position 来获取 Animal 对象，再通过对象封装的 getAnimal 方法获得对应的动物名，通过 Toast 显示出来。

运行实例，如图 5.3 所示。查看动态图，请扫描图 5.4 中的二维码。

图 5.3　ListView 之 BaseAdapter

图 5.4　ListView 之 BaseAdapter 二维码

5.2　控之经典——ListView 进阶

如何在 ListView 的上添加滑动监听、实现上拉加载功能，是面试中和实际工作中经常会遇到的问题。能否很从容地处理这部分问题，也反映了程序员的基础能力。因此，有必要要对这部分知识点进行研究和学习。

首先看一下 API 文档中滑动监听的定义，其继承结构如下：

public static interface

AbsListView.OnScrollListener

android.widget.AbsListView.OnScrollListener

由继承结构可以看出，OnScrollListener 是一个静态接口，接口中都是未实现需要覆写的方法。OnScrollListener 中有两个需要覆写的方法：

- onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount)：正在滑动时不断触发，主要有四个参数，分别是 ListView 对象、当前可以看见的第一个子项（也就是当前屏幕最上方的子项）、可以看到子项的个数、总的子项个数。
- onScrollStateChanged(AbsListView view, int scrollState)：顾名思义，这个是在滑动状态变化的情况下触发，里面有两个参数，分别是 ListView 对象和滑动状态。

那么滑动状态又分为哪些呢？ API 文档中也进行了说明，共有三个状态：

- SCROLL_STATE_FLING：手指正在拖着滑动（手指没离开屏幕）。
- SCROLL_STATE_IDLE：滑动停止。
- SCROLL_STATE_TOUCH_SCROLL：手指使劲在屏幕上滑了一下，由于惯性屏幕继续滚动（手指离开屏幕）。

下面通过一个小实例来介绍，如何通过监听滑动变化实现上拉加载的功能。

首先要定义一个底部布局，用于加载时的提示，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ll_footer"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <ProgressBar
        android:id="@+id/progress"
        style="?android:attr/progressBarStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

    <TextView
        android:id="@+id/tv_wait"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text=" 正在加载 ..."
        android:textSize="10sp" />
</LinearLayout>
```

上述代码中布局采用线性布局，添加了一个 ProgressBar 在 TextView 的上方，设置了 ProgressBar 的 style 属性，其值为“ ?android:attr/progressBarStyleSmall ”，也就是小圆形的进度条形式，设置了 layout_gravity 属性，其值为 center，居中显示。添加了一个 TextView 显示提示信息，同样也设置其 layout_gravity 属性为 center。

为了实现上拉加载的功能，这里自定义了一个控件继承自 ListView，并实现了 OnScrollListener 接口：

```
public class UpAddListView extends ListView implements AbsListView.
OnScrollListener {
    private View footer;
    // 标志位 是否正在加载
    private Boolean isAdding = false;
    private int totalItems, totalItemCount;
    private IUpAddListener iUpAddListener;

    public UpAddListView(Context context) {
        super(context);
        initView(context);
    }

    public UpAddListView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initView(context);
    }
```

```
public UpAddListView(Context context, AttributeSet attrs, int
defStyleAttr) {
    super(context, attrs, defStyleAttr);
    initView(context);
}

@Override
public void onScrollStateChanged(AbsListView view, int scrollState)
{
    // 当前可见第一项和当前屏幕可见性个数之和等于总的子项个数 && 滑动停止状态
    if ((totalItemCount == totalItems) && scrollState == SCROLL_
    STATE_IDLE) {
        if (!isAdding) {
            footer.findViewById(R.id.ll_footer).setVisibility(View.
            VISIBLE);
            // 回调方法
            iUpAddListener.onAdd();
            isAdding = true;
        }
    }
}

@Override
public void onScroll(AbsListView view, int firstVisibleItem, int
                    visibleItemCount, int totalItemCount) {
    // 当前看到第一个子项 + 可以看到的子项个数
    totalItems = firstVisibleItem + visibleItemCount;
    // 总的子项个数
    this.totalItemCount = totalItemCount;
}

private void initView(Context context) {
    footer = LayoutInflater.from(context).inflate(R.layout.footer_
    layout, null);
    // 初始状态底部布局不显示
    footer.findViewById(R.id.ll_footer).setVisibility(View.GONE);
    // 添加底部布局
    this.addFooterView(footer);
    // 设置滑动监听
    this.setOnScrollListener(this);
}

public interface IUpAddListener {
    void onAdd();
}

public void setInterface(IUpAddListener iUpAddListener) {
    this.iUpAddListener = iUpAddListener;
}
```

```
        //加载完毕
        public void addCompleted() {
            isAdding = false;
            footer.findViewById(R.id.ll_footer).setVisibility(View.GONE);
        }
    }
```

实现 OnScrollListener 接口要覆写其两个方法：

- onScroll：滚动时会触发。为了在 onScrollStateChanged 中使用这个方法中的参数，我们定义了两个 int 型的全局变量，将这个方法中的参数传递给全局变量以便在 onScrollStateChanged 中使用。
- onScrollStateChanged：滚动状态改变时会触发。这里思考一下，怎么才能判断滚动到最底部了呢？首先滚动到底部，不能再继续滑动了，滑动状态 scrollState 必须是停止状态了，即“ scrollState==SCROLL_STATE_IDLE ”。滑动停止了就滑动到底了吗？显然不是，还必须要满足一个条件，这个条件就要借助 onScroll 中传递的三个参数了，当满足当前屏幕中第一个可见项 + 当前屏幕可以看到的子项个数 = 总体子项个数时，就说明滚动到底了。

这里使用了回调方法用于数据传递，定义了一个内部接口，里面有一个 onAdd 的抽象方法。此外，设置了 setInterface 方法，进行接口注册。

addCompleted 方法，在加载完成后调用，将加载标志位设置成 false 并将底部栏隐藏。

下面将自定义控件引入到主布局（activity_main.xml）文件：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ad.listviewonscrolllistenerdemo.UpAddListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

引入自定义控件需要在标签中加入完整的包 . 类名，这里是 ad.listviewonscrolllistenerdemo. UpAddListView，设置了宽、高属性为 match_parent，占据整个界面。

MainActivity.java 代码如下：

```
public class MainActivity extends Activity implements UpAddListView.
IUpAddListener {
    private UpAddListView mUpAddListView;
    private List datas = new ArrayList<String>();
    private ArrayAdapter mArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initViews();
```

```
        initDatas();
    }

    private void initViews() {
        mUpAddListView = (UpAddListView) findViewById(R.id.lv);
        mUpAddListView.setInterface(this);
        mArrayAdapter = new ArrayAdapter(this,
                android.R.layout.simple_list_item_1, datas);
        mUpAddListView.setAdapter(mArrayAdapter);
    }

    private void initDatas() {
        for (int i = 0; i < 10; i++) {
            datas.add("测试数据" + i);
        }
    }

    private void addMoreDatas() {
        for (int i = 0; i < 2; i++) {
            datas.add("新数据" + i);
        }
    }

    @Override
    public void onAdd() {
        //为了便于观察，并模仿请求操作时间，这里采用延迟执行的方法
        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                //添加更多数据
                addMoreDatas();
                //通知刷新
                mArrayAdapter.notifyDataSetChanged();
                //完成加载
                mUpAddListView.addCompleted();
            }
        }, 2000);
    }
}
```

为了方便，这里采用了ArrayAdapter作为适配器类，初始时添加了10条测试数据，每次上拉加载时调用addMoreDatas方法添加两条新数据。Activity实现了在UpAddListView中定义的IUpAddListener接口，覆写了其onAdd方法，此处为了模拟数据加载的时间，采用了延迟Handler类的postDelayed方法，延迟2000ms再进行加载和加载完成的操作。加载完成后记得调用notifyDataSetChanged方法，刷新ListView显示。

运行实例，如图5.5所示。滑动到底部然后上拉，如图5.6所示。

查看动态图，请扫描图5.7中的二维码。

图 5.5　ListView 上拉加载一

图 5.6　ListView 上拉加载二

图 5.7　ListView 上拉加载二维码

5.3　控之经典——GridView

上面的章节中，介绍了 ListView 的方法，本节将对 GridView（网格视图）的属性和方法进行讲解，与 ListView 一般用于列表项的展示相比，GridView 是按照行列的方式来显示内容的，一般用于显示图片。图文等内容，例如实现九宫格图，用 GridView 是首选，也是最简单的。首先我们来看一下 GridView 有哪些常用属性和相关方法，如表 5.3 所示。

表 5.3　ListView 的常用属性和相关方法

属　性	相 关 方 法	说　明
android:columnWidth	setColumnWidth(int)	设置列的宽度
android:horizontalSpacing	setHorizontalSpacing(int)	定义列之间水平间距
android:numColumns	setNumColumns(int)	设置列数
android:stretchMode	setStretchMode(int)	缩放模式
android:verticalSpacing	setVerticalSpacing(int)	定义行之间默认垂直间距

下面通过一个简单实例看一下如何使用 GridView 控件。

主布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <GridView
        android:id="@+id/gv"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:horizontalSpacing="5dp"
        android:numColumns="3"
        android:stretchMode="columnWidth"
```

```
        android:verticalSpacing="5dp" />
</RelativeLayout>
```

在 RelativeLayout 布局中引入了一个 GridView 控件，设置了宽、高属性为 match_parent，控件占据整个界面；设置了 numColumns 属性，显示 3 列；设置了 stretchMode 属性为 columnWidth，表示图片的缩放与列宽的大小一致。

子项布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/iv"
        android:layout_width="100dp"
        android:layout_height="100dp" />

    <TextView
        android:id="@+id/tv"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20sp" />
</LinearLayout>
```

子项布局文件中包括一个 ImageView 控件和一个 TextView 控件，采用线性布局并设置 orientation 属性为 vertical。

为了数据操作方便，同样这里也设置了一个 JavaBean 类对数据进行封装，代码如下：

```
public class Animal {
  // 和上一节 JavaBean 类一样
}
```

其中主要包含 animal（即动物名）和 imgId（即图片 ID）这两个属性，并设置了相应的 setter 和 getter 方法。

这里自定义了 GridView 的适配器类，这个类继承自 BaseAdapter，代码如下：

```
public class GridAdapter extends BaseAdapter {
    private Context context;
    private List<Animal> datas;

    public GridAdapter(Context context, List<Animal> datas) {
        this.context = context;
        this.datas = datas;
    }
    // 返回子项的个数
    @Override
    public int getCount() {
```

```
        return datas.size();
    }
    //返回子项对应的对象
    @Override
    public Object getItem(int position) {
        return datas.get(position);
    }
    //返回子项的下标
    @Override
    public long getItemId(int position) {
        return position;
    }
    //返回子项视图
    @Override
    public View getView(int position, View convertView, ViewGroup
    parent) {
        Animal animal = (Animal) getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(context).inflate(R.layout.item_
             layout, null);
            viewHolder = new ViewHolder();
            viewHolder.animalImage = (ImageView) view.findViewById(R.
             id.iv);
            viewHolder.animalName = (TextView) view.findViewById(R.
             id.tv);
            view.setTag(viewHolder);
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag();
        }
        viewHolder.animalName.setText(animal.getAnimal());
        viewHolder.animalImage.setImageResource(animal.getImgId());
        return view;
    }
    //创建 ViewHolder 类
    class ViewHolder {
        ImageView animalImage;
        TextView animalName;
    }
}
```

细心的读者可以看到，这里的适配器类除了名字和 ListView 的适配器类不同之外，其余都相同，GridView 和 ListView 都是继承自 AbsListView，用法也有很多相似之处。学习要有举一反三的能力，找出控件使用时的共性，可以提高学习速度并能加深理解。这里就不再重复解释上面的代码，不清楚的同学可以翻看前面的 ListView 部分。

MainActivity.java 代码如下：

```
public class MainActivity extends AppCompatActivity {
    private GridView mGridView;
    private GridAdapter mGridAdapter;
    private List<Animal> mDatas = new ArrayList<Animal>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mGridView = (GridView) findViewById(R.id.gv);
        //初始化数据
        initDatas();
        //实例化适配器
        mGridAdapter = new GridAdapter(this, mDatas);
        //设置适配器
        mGridView.setAdapter(mGridAdapter);
        mGridView.setOnItemClickListener(new AdapterView.
        OnItemClickListener() {//设置子项单击监听
            @Override
           public void onItemClick(AdapterView<?> parent, View view,
           int position, long id) {
                String name = mDatas.get(position).getAnimal();
                ImageView imageView = new ImageView(MainActivity.this);
                imageView.setScaleType(ImageView.ScaleType.CENTER);
                imageView.setLayoutParams(new LinearLayout.
                LayoutParams(
                        ViewGroup.LayoutParams.WRAP_CONTENT,
                        ViewGroup.LayoutParams.WRAP_CONTENT));
                imageView.setImageResource(mDatas.get(position).
                getImgId());
                Dialog dialog = new AlertDialog.Builder(MainActivity.
                this)
                        .setIcon(android.R.drawable.ic_btn_speak_now)
                        .setTitle("您选择的动物是：" + name)
                        .setView(imageView)
                        .setNegativeButton("取消", new DialogInterface.
                         OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
                            int which) {
                            }
                        }).create();
                dialog.show();
            }
        });
    }

    private void initDatas() {
        Animal animal0 = new Animal("兔八哥", R.drawable.rabbit);
```

```
            Animal animal1 = new Animal("眼镜蛇", R.drawable.snack);
            Animal animal2 = new Animal("小金鱼", R.drawable.fish);
            Animal animal3 = new Animal("千里马", R.drawable.horse);
            Animal animal4 = new Animal("米老鼠", R.drawable.mouse);
            Animal animal5 = new Animal("大国宝", R.drawable.panda);
            Animal animal6 = new Animal("千里马", R.drawable.horse);
            Animal animal7 = new Animal("米老鼠", R.drawable.mouse);
            Animal animal8 = new Animal("大国宝", R.drawable.panda);
            mDatas.add(animal0);
            mDatas.add(animal1);
            mDatas.add(animal2);
            mDatas.add(animal3);
            mDatas.add(animal4);
            mDatas.add(animal5);
            mDatas.add(animal6);
            mDatas.add(animal7);
            mDatas.add(animal8);
        }
    }
```

总的来讲，实现 GridView 需要三个步骤：

- 准备数据源（initDatas()）；
- 新建适配器（GridAdapter extends BaseAdapter）；
- 加载适配器（setAdapter()）。

这里我们还实现了 GridView 的子项单击监听（setOnItemClickListener），单击某个子项时，弹出 Dialog 对话框，通过 List 的 get 方法获取单击子项对应的 Animal 对象，然后再通过 Animal 类的 getAnimal 方法获取对应的动物名。同理，根据上面的方法获得图片资源对象，调用 Dialog 的 setView 方法，将资源图片对象传入，可以把对应子项的图片显示在 Dialog 对话框中。

运行实例，结果如图 5.8 所示，单击任意子项（这里单击眼镜蛇），结果如图 5.9 所示。

查看动态图，请扫描图 5.10 中的二维码。

图 5.8　GridView 实例

图 5.9　GridView 单击效果

图 5.10　GridView 单击效果二维码

5.4 控之经典——GridView 进阶

5.4.1 GridView 动态图删除子项

用过 UC 浏览器的人相信都不会对图 5.11 所示的功能陌生。

图 5.11 UC 浏览器单击删除 Tab

长按图标时会在左上角显示出一个删除的图片，单击这个图片就可以删除与之对应的子项，下面介绍如何借助 GridView 来实现这个功能。

主布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <GridView
        android:id="@+id/gv"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:horizontalSpacing="5dp"
        android:numColumns="3"
        android:stretchMode="columnWidth"
        android:verticalSpacing="5dp" />
</RelativeLayout>
```

其功能和上一小节的布局文件一致，这里就不再介绍。

子项布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginRight="4dip"
        android:layout_marginTop="4dip"
        android:gravity="center"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/iv"
            android:layout_width="60dip"
            android:layout_height="55dip" />

        <TextView
            android:id="@+id/tv"
            android:layout_width="70dip"
```

```
            android:layout_height="wrap_content"
            android:layout_marginTop="10dip"
            android:gravity="center"
            android:textColor="@android:color/black"
            android:textSize="15sp"
            android:textStyle="bold" />
    </LinearLayout>

    <ImageView
        android:id="@+id/delete_markView"
        android:layout_width="20dip"
        android:layout_height="20dip"
        android:layout_gravity="right|top"
        android:adjustViewBounds="true"
        android:src="@drawable/delete"
        android:visibility="gone" />
</FrameLayout>
```

上述代码中子项布局采用FrameLayout帧布局模式，设置删除图片的layout_gravity属性为right|top，即布局在右上角并设置其visibility属性为gone，即初始时不显示该图片。

JavaBean类Animal和上一节一样，这里就不再进行介绍。

下面看一下自定义的适配器类：

```
public class GridAdapter extends BaseAdapter {
    private Context context;
    private List<Animal> datas;
    final int position = 0;
    private boolean mIsShowDelete;

    public GridAdapter(Context context, List<Animal> datas) {
        this.context = context;
        this.datas = datas;
    }
    //返回子项的个数
    @Override
    public int getCount() {
        return datas.size();
    }
    //返回子项对应的对象
    @Override
    public Object getItem(int position) {
        return datas.get(position);
    }
    //返回子项的下标
    @Override
    public long getItemId(int position) {
        return position;
    }
    //返回子项视图
    @Override
```

```
    public View getView(final int position, View convertView, final
    ViewGroup parent) {
        Animal animal = (Animal) getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(context).inflate(R.layout.item_
             layout, null);
            viewHolder = new ViewHolder();
            viewHolder.animalImage = (ImageView) view.findViewById(R.
             id.iv);
            viewHolder.animalName = (TextView) view.findViewById(R.
             id.tv);
            viewHolder.deleteImage = (ImageView) view.findViewById(R.
             id.delete_markView);
            view.setTag(viewHolder);
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag();
        }
        viewHolder.animalName.setText(animal.getAnimal());
        viewHolder.animalImage.setImageResource(animal.getImgId());
        viewHolder.deleteImage.setVisibility(mIsShowDelete ? View.
        VISIBLE : View.GONE);
        if (mIsShowDelete) {
            viewHolder.deleteImage.setOnClickListener(new View.
            OnClickListener() {
                @Override
                public void onClick(View v) {
                    datas.remove(position);
                    setmIsShowDelete(false);
                }
            });
        }
        return view;
    }
    // 创建 ViewHolder 类
    class ViewHolder {
        ImageView animalImage, deleteImage;
        TextView animalName;
    }

    public void setmIsShowDelete(boolean mIsShowDelete) {
        this.mIsShowDelete = mIsShowDelete;
        notifyDataSetChanged();
    }
}
```

这里设置了一个全局变量 mIsShowDelete 作为 DeleteImage 是否显示的标志位，并添加了一个 setIsShowDelete 方法用于改变 mIsShowDelete 的值，在这个方法中还调用了 notifyDataSetChanged 方法刷新 GridView 显示。

这里还为 DeleteImage 添加了一个单击事件的监听，覆写了 onClick 方法，在这个方法中调用 List 集合的 remove 方法去除这个子项的数据并调用 setmIsShowDelete 传入 false 隐藏“删除图片”。

MainActivity.java 代码如下：

```
public class MainActivity extends Activity {
    private GridView mGridView;
    private GridAdapter mGridAdapter;
    private boolean isShowDelete;
    private List<Animal> datas = new ArrayList<Animal>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mGridView = (GridView) findViewById(R.id.gv);
        //初始化数据
        initDatas();
        //实例化适配器类
        mGridAdapter = new GridAdapter(this, datas);
        //设置适配器类
        mGridView.setAdapter(mGridAdapter);
        //设置长按事件监听
        mGridView.setOnItemLongClickListener(newAdapterView.
        OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> parent,
                                           View view, int position, long id) {
                if (isShowDelete) {
                    //删除图片显示时长按隐藏
                    isShowDelete = false;
                    mGridAdapter.setmIsShowDelete(isShowDelete);
                } else {
                    //删除图片隐藏时长按显示
                    isShowDelete = true;
                    mGridAdapter.setmIsShowDelete(isShowDelete);
                }
                return false;
            }
        });
    }

    private void initDatas() {
        Animal animal0 = new Animal("兔八哥", R.drawable.rabbit);
        Animal animal1 = new Animal("眼镜蛇", R.drawable.snack);
        Animal animal2 = new Animal("小金鱼", R.drawable.fish);
        Animal animal3 = new Animal("千里马", R.drawable.horse);
        Animal animal4 = new Animal("米老鼠", R.drawable.mouse);
        Animal animal5 = new Animal("大国宝", R.drawable.panda);
        datas.add(animal0);
        datas.add(animal1);
```

```
            datas.add(animal2);
            datas.add(animal3);
            datas.add(animal4);
            datas.add(animal5);
        }
    }
```

这里设置了 GridView 的长按事件监听，当“删除图片”显示时，将 isShowDelete 标志位设置成 false，然后调用 GridAdapter 适配器类的 setmIsShowDelete 方法隐藏右上角的“删除图片”。反之，当“删除图片”隐藏时，则调用 setmIsShowDelete 方法传入 true 参数显示右上角的“删除图片”。

运行实例并长按任一子项，如图 5.12 所示。单击右上角的“删除图片⊗”则删除当前子项，右上角的“删除图片”也随之隐藏。查看动态图，请扫描图 5.13 中的二维码。

图 5.12　GridView 单击删除子项

图 5.13　GridView 单击删除子项二维码

5.4.2　GridView 动态图增加子项

下面研究如何实现动态增加子项，在上述基础上动态增加子项的功能。布局文件不做调整，因此，这里就不再介绍代码。

在 GridAdapter 适配器类中动态增加子项的功能，参考代码如下：

```
public class GridAdapter extends BaseAdapter {
    // 和上一节一样，省略部分相同代码
    // 返回子项视图
    @Override
    public View getView(final int position, View convertView, final
    ViewGroup parent) {
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(context).inflate(R.layout.item_layout,
            null);
            viewHolder = new ViewHolder();
            viewHolder.animalImage = (ImageView) view.findViewById(R.id.iv);
```

```
            viewHolder.animalName = (TextView) view.findViewById(R.id.tv);
            viewHolder.deleteImage = (ImageView) view.findViewById(R.id.delete_
             markView);
            // 设置 tag
            view.setTag(viewHolder);
        } else {
            view = convertView;
            // 由 tag 获取对象
            viewHolder = (ViewHolder) view.getTag();
        }
        if (position < datas.size()) {
            Animal animal = (Animal) getItem(position);
            viewHolder.animalName.setText(animal.getAnimal());
            viewHolder.animalImage.setImageResource(animal.getImgId());
            // 根据标志位 isShowDelete 决定是否显示删除图片按钮
            viewHolder.deleteImage.setVisibility(isShowDelete ? View.VISIBLE:
            View.GONE);
            if (isShowDelete) {
                viewHolder.deleteImage.setOnClickListener(new View.
                OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        datas.remove(position);
                        setIsShowDelete(false);
                    }
                });
            }
        } else {
            viewHolder.animalName.setText(" 单击添加 ");
            viewHolder.animalImage.setImageResource(R.drawable.add);
            viewHolder.deleteImage.setVisibility(View.GONE);
        }
        return view;
    }
// 和上一节一样，省略部分相同代码
}
```

注意，因为这里多了最后一个子项用来作为“添加项”，所以在 getCount 方法的返回中要返回 datas.size()+1。

在 getView 方法中添加了判断，在 position<datas.size 时，加载 datas 里面的数据，而在 position=datas.size() 的地方加载“添加项”。

MainActivity 中也做了一些调整，代码如下：

```
public class MainActivity extends Activity {
        // 和上一节一样，省略部分相同代码
        mGridView.setOnItemClickListener(new AdapterView.
        OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                                    int position, long id) {
                // 单击了最后一张 "+" 图片
```

```
                if (position == parent.getChildCount() - 1) {
                    addDatas();
                }
            }
        });
        mGridView.setOnItemLongClickListener(new AdapterView.
                OnItemLongClickListener() {
          // 和上一节一样，省略部分相同代码
        });
    }

    private void addDatas() {
        Animal animalAdd = new Animal(" 大国宝 ", R.drawable.panda);
        datas.add(animalAdd);
        mGridAdapter.notifyDataSetChanged();
    }

    private void initDatas() {
       // 和上一节一样，省略部分相同代码
    }
}
```

这里添加了子项单击事件监听（OnItemClickListener），判断当“ position==parent.getChildCount-1”时，即单击最后一个“添加项”时调用 addDatas 方法，添加一条记录到 datas 里面。注意，添加完成数据后，要调用 notifyDataSetChanged 方法刷新列表。

运行项目实例并单击最后一个“单击添加”图片，如图 5.14 所示，单击最后一个子项（“单击添加”），将会新插入一个子项。查看动态图，请扫描图 5.15 中的二维码。

图 5.14　GridView 单击增加子项

图 5.15　GridView 单击增加子项二维码

5.5　新控件——RecyclerView 控件

Android 5.0 引入了一个全新的列表控件 RecyclerView，说它新，是相对于其他控件而言。它更为灵活，同时也拥有比 ListView 和 GridView 控件较多的优点，例如子项 View 的

创建、View 的回收以及重用等机制。

为了使用 RecyclerView 控件，需要创建一个 Adapter 和一个 LayoutManager 类。Adapter 继承自 RecyclerView.Adapetr 类，主要用来将数据和布局子项进行绑定。LayoutManager，布局管理器，设置每一项 View 在 RecyclerView 中的位置布局以及控件子项 View 的显示或者隐藏。当 View 重用或者回收时，LayoutManger 都会向 Adapter 请求新的数据进行替换原来数据的内容。这种回收重用的机制可以提高性能，避免创建很多的 View 或者是频繁地调用 findViewById 方法。

RecyclerView 提供了三种内置的 LayoutManager：

- LinearLayoutManager：线性布局，横向或者纵向滑动列表。
- GridLayoutManager：网格布局。
- StaggeredGridLayoutManager：流式布局（瀑布流效果）。

当然除了上面的三种内部布局之外，还可以继承 RecyclerView.LayoutManager 来实现一个自定义的 LayoutManager。

下面通过一个简单实例对 RecyclerView 有一个简单的理解。RecyclerView 控件需要引入 RecyclerView 兼容包，选中项目并右击，在弹出的快捷菜单中选择 Open Module Settings，如图 5.16 所示。

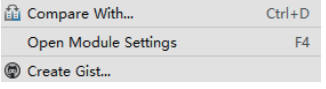

图 5.16　Open Module Settings

切换到 Dependencies 标签，单击右上角的“+”并选择第一项 Library dependency 选项，如图 5.17 所示。

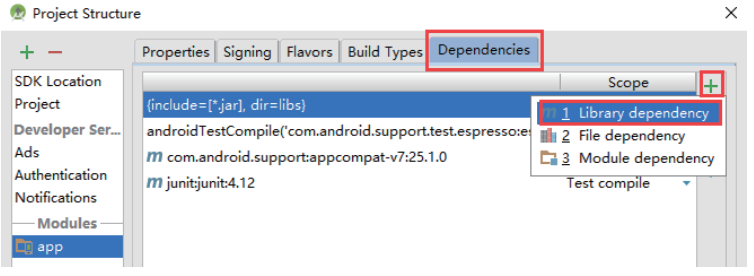

图 5.17　添加 Library dependency

在弹出的界面中输入 recyclerview，然后单击右边的搜索按钮，如图 5.18 所示。

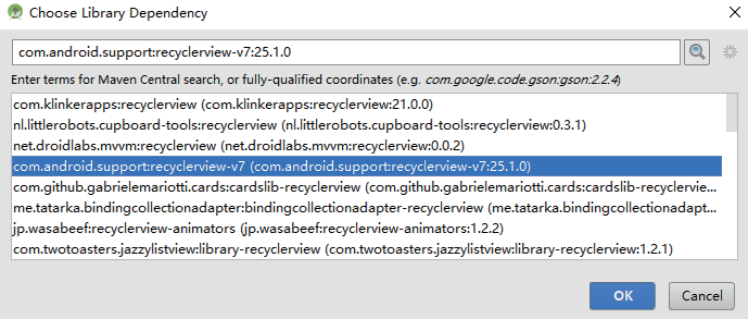

图 5.18　搜索 Library

选中兼容包，然后单击 OK 按钮，如图 5.19 所示。

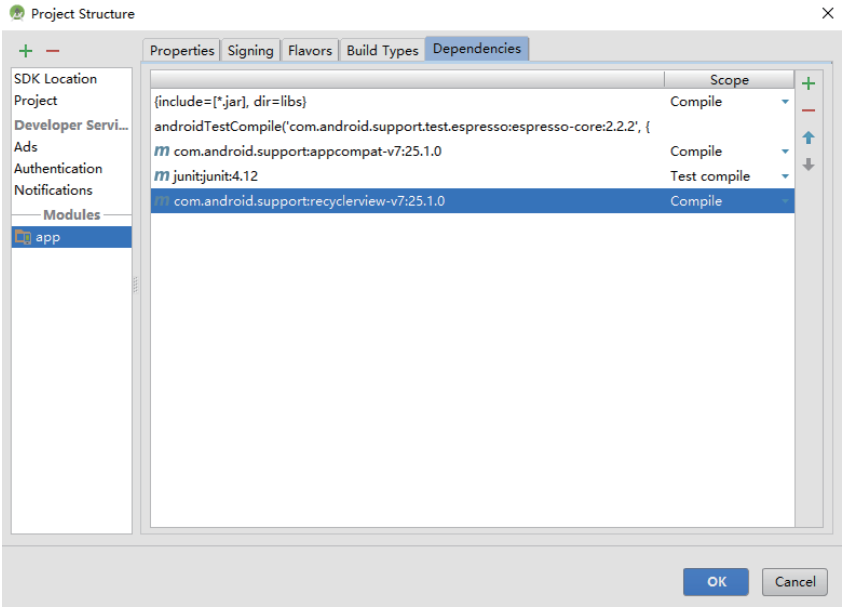

图 5.19　添加 Library dependency 成功

再次选中兼容包，然后单击 OK 按钮，等待 Gradle 编译完成就可以使用 RecyclerView 了。

5.5.1　RecyclerView 线性布局

主布局文件（activity_main.xml）中引入一个 RecyclerView，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="vertical" />
</RelativeLayout>
```

上述代码在相对布局中添加了一个 RecyclerView 控件，引入时要输入完整的“包 . 类”名，可以看出 RecyclerView 在 V7 包中。

为 RecyclerView 添加一个子项布局文件（item.xml），代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="50dp">

    <TextView
```

```
        android:id="@+id/id_num"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:gravity="center"
        android:text="1" />
</RelativeLayout>
```

和 ListView 相似，RecyclerView 同样需要创建一个适配器，代码如下：

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.
MyViewHolder> {
    private Context mContext;
    private List<String> mDatas;

    public MyAdapter(Context context, List<String> datas) {
        mContext = context;
        mDatas = datas;
    }

    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int
    viewType) {
        MyViewHolder holder = new MyViewHolder(LayoutInflater.from(
                mContext).inflate(R.layout.item, parent,
                false));
        return holder;
    }

    @Override
    public void onBindViewHolder(MyViewHolder holder, int position) {
        holder.tv.setText(mDatas.get(position));
    }

    @Override
    public int getItemCount() {
        return mDatas.size();
    }

    class MyViewHolder extends RecyclerView.ViewHolder {
        TextView tv;

        public MyViewHolder(View view) {
            super(view);
            tv = (TextView) view.findViewById(R.id.id_num);
        }
    }
}
```

适配器类继承自 RecyclerView.Adapter，这里有几个方法需要解释一下：

- 构造方法 MyAdapter：传入了 Context 对象和数据集合，这点和 ListView 一样。
- OnCreateViewHolder 方法：这是必须要覆写的方法，返回一个 ViewHolder 对象。创建这个内部类需要传入一个 View 对象，View 对象的获得同样是使用

LayoutInflator 类的相关方法。

- onBindViewHolder 方法：绑定控件数据。
- getItemCount 方法：返回数据项个数。

MainActivity.java 代码如下：

```
public class MainActivity extends Activity {
    private RecyclerView mRecyclerView;
    private List<String> mDatas;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initData();
        mRecyclerView = (RecyclerView) findViewById(R.id.recyclerview);
        mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
        mRecyclerView.setAdapter( new MyAdapter(this, mDatas));
    }

    protected void initData() {
        mDatas = new ArrayList<String>();
        for (int i = 1; i < 20; i++) {
            mDatas.add("" + i);
        }
    }
}
```

上述代码在 onCreate 方法中通过 findViewById 方法得到布局中添加的 RecyclerView 对象，调用 RecyclerView 类的 setLayoutManager 方法设置布局类型，这里传入的是 Android 提供的 LinearLayoutManager 对象（线性布局）。调用 setAdapter 方法为 RecyclerView 添加自定义的适配器。

运行实例，如图 5.20 所示。

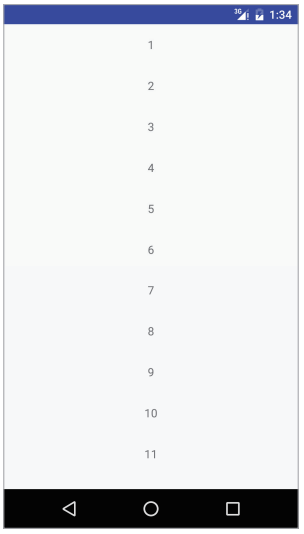

图 5.20　RecyclerView 线性布局

可以看出子项自上而下显示在屏幕中，可以向下滑动查看下面的内容，和 ListView 的效果基本一致。

5.5.2　RecyclerView 网格布局

修改 MainActivity.java 代码如下：

```
        // 省略部分相同代码
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            // 省略部分相同代码
            mRecyclerView.setLayoutManager(new GridLayoutManager(this,4));
            mRecyclerView.setAdapter( new MyAdapter(this, mDatas));
        }
        // 省略部分相同代码
    }
```

这时 setLayoutManager 方法传入了一个 GridLayoutManager 对象，这个对象需要传入两个参数：上下文对象和列数。再次运行实例，如图 5.21 所示。

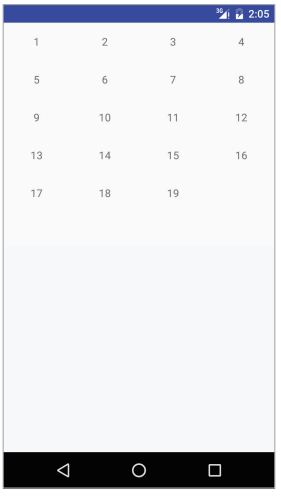

图 5.21　RecyclerView 网格布局

5.5.3　RecyclerView 瀑布流布局

上面讲解了两种基本的用法，下面讲解瀑布流布局的用法。

为了提高演示效果，这里修改了子项布局的代码，添加了 ImageView 控件用来显示图片，代码如下：

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/id_num"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="1" />

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/pic" />

    </LinearLayout>
```

瀑布流布局需要随机调整子项的高，修改自定义的适配器代码如下：

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.
MyViewHolder> {
    private Context mContext;
    private List<String> mDatas;
    private List<Integer> mHights;

    public MyAdapter(Context context, List<String> datas) {
        mContext = context;
        mDatas = datas;
        mHights = new ArrayList<>();
        initHeights();
    }
    // 省略部分相同代码

    @Override
    public void onBindViewHolder(MyViewHolder holder, int position) {
        ViewGroup.LayoutParams layoutparams = holder.itemView.
        getLayoutParams();
        layoutparams.height = mHights.get(position);
        holder.itemView.setLayoutParams(layoutparams);
        holder.tv.setText(mDatas.get(position));
    }
    // 省略部分相同代码
    private  void initHeights(){
        for (int i = 0; i < mDatas.size(); i++) {
            mHights.add((int) (50 + Math.random() * 300));
        }
    }
}
```

在构造方法中调用了initHeights方法初始化了一个高度的List集合，在onBindViewHolder方法中为子项设置了随机的高，调用View的getLayoutParams方法得到布局参数对象layoutparams，将高度集合List中的值赋给这个对象的高。

修改MainActivity.java代码如下：

```
    // 省略部分相同代码
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
// 省略部分相同代码
mRecyclerView.setLayoutManager(new StaggeredGridLayoutManager(4,
        StaggeredGridLayoutManager.VERTICAL));
mRecyclerView.setAdapter( new MyAdapter(this, mDatas));
// 省略部分相同代码
```

上述代码修改 setLayoutManager 方法中的参数为 StaggeredGridLayoutManager 对象，创建这个对象需要传入两个参数：第一个参数为每行列数，第二个参数为布局的方向，这里传入 StaggeredGridLayoutManager.VERTICAL 常量（垂直布局）。运行实例，结果如图 5.22 所示。

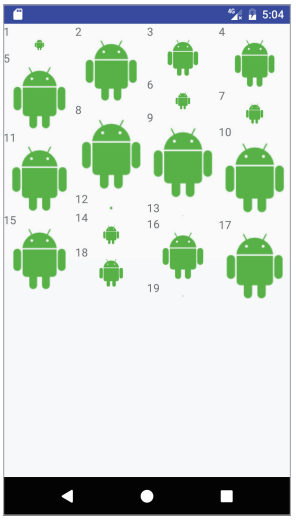

图 5.22　RecyclerView 瀑布流布局

5.6　多页面切换器——ViewPager 控件

一般 APP 都是由多个页面组成，页面的切换是开发中比较重要的部分，Android 提供了封装好的页面切换控件 ViewPager 供开发者使用，其继承结构如下：

public class
ViewPager
extends ViewGroup
java.lang.Object
　□ ①　android.view.View
　　□　android.view.ViewGroup
　　　□　android.support.v4.view.ViewPager

ViewPager 继承自 ViewGroup，可以看出来它是一个容器类，类前包名是 android.support.v4，这是一个兼容包。注意，在布局文件中引入该控件时，标签要写全，即：< android.support.v4.view.ViewPager >。API 文档中对 ViewPager 进行了描述，总结如下：

- ViewPager 类直接继承自 ViewGroup 类，作为一个容器类，可以向其中添加 View 类。

① 方框“□”代表继承，继承是 Java 三大特性之一。

- 数据源和显示之间需要一个适配器类 PagerAdapter 进行适配。
- ViewPager 经常和 Fragemnet 一起使用，并且提供专门的适配器类 FragmentPagerAdapter 和 FragmentStatePagerAdapter 类供开发者调用。

实现 PageAdapter 必须覆写四个方法：

- public Object instantiateItem(ViewGroup container, int position)：初始化一个子项。
- public void destroyItem(ViewGroup container, int position,Object object)：销毁一个子项。
- public int getCount()：返回子项的个数。
- public boolean isViewFromObject(View arg0, Object arg1)：返回一个布尔型变量，判断子项是否来自 Object。

5.6.1 ViewPager 的基本用法

下面通过一个实例看一下 ViewPager 的基本用法。

主布局文件（activity_main.xml）代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

这里引入了一个 ViewPager 控件，通过包.类名的方式引入，设置了宽高属性为 match_parent。ViewPager 是布局容器类，这里添加了三个子布局用来切换页面。

子布局一（layout1.xml）代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="页面 1"
        android:textSize="30sp" />
</LinearLayout>
```

上述子布局中仅添加了一个 TextView 用于区别不同的页面，其余两个子布局同样也是只包含一个 TextView，不同的仅是 text 的属性值不同，这里就不再介绍代码。

ViewPager 同样需要适配器类，代码如下：

```
public class MyViewPagerAdapter extends PagerAdapter {
    private List<View> datas;

    public MyViewPagerAdapter(List<View> datas) {
        this.datas = datas;
    }
    //返回页卡数量
    @Override
    public int getCount() {
        return datas.size();
    }
    //判断View是否来自Object
    @Override
    public boolean isViewFromObject(View view, Object object) {
        return view == object;
    }

    //初始化一个页卡
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        container.addView(datas.get(position));
        return datas.get(position);
    }

    //销毁一个页卡
    @Override
    public void destroyItem(ViewGroup container, int position, Object object) {
        container.removeView(datas.get(position));
    }
}
```

自定义适配器类MyViewPagerAdapter继承自PagerAdapter，创建了构造函数，用于在初始化时传入datas数据集。PagerAdapter是抽象类，继承这个类需要覆写它的四个抽象方法，这四个方法的说明请参考代码中的注释。

MainActivity.java代码如下：

```
public class MainActivity extends Activity {
    private ViewPager mViewPager;
    private List<View> mDatas;
    private MyViewPagerAdapter myViewPagerAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mViewPager = (ViewPager) findViewById(R.id.viewPager);
        //初始化数据集
        initDatas();
        myViewPagerAdapter = new MyViewPagerAdapter(mDatas);
        //设置适配器
        mViewPager.setAdapter(myViewPagerAdapter);
```

```
        }

        private void initDatas() {
            mDatas = new ArrayList<>();
            View view1 = LayoutInflater.from(this).inflate(R.layout.layout1, null);
            View view2 = LayoutInflater.from(this).inflate(R.layout.layout2, null);
            View view3 = LayoutInflater.from(this).inflate(R.layout.layout3, null);
            mDatas.add(view1);
            mDatas.add(view2);
            mDatas.add(view3);
        }
    }
```

总结以下，ViewPager 的实现可以分为三个步骤：

- 准备数据源（initDatas），这里是调用 LayoutInflater 的静态方法 from 并传入上下文对象获得一个 LayoutInflater 对象，和前面获取 LayoutInflater 对象的方式稍有不同，但查看源码可以看出，它们其实调用的方法是一致的，是 Android 封装好的方法。参考如下源码：

```
public static LayoutInflater from(Context context) {
    LayoutInflater LayoutInflater = (LayoutInflater) context.
            getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    if (LayoutInflater == null) {
        throw new AssertionError("LayoutInflater not found.");
    }
    return LayoutInflater;
}
```

可以看出，from 是一个静态方法，这个方法内部也是通过 getSystemService 方法并传入 Context.LAYOUT_INFLATER_SERVICE 常量来获取 LayoutInflater 对象，然后返回这个对象。

- 准备适配器类并初始化（MyViewPagerAdapter），传入布局数据源。
- 设置适配器，调用 ViewPager 的 setAdapter 方法传入初始化的自定义适配器。

运行实例后并向右滑动即可切换到第二个页面，如图 5.23 所示。查看动态图，请扫描图 5.24 中的二维码。

图 5.23　ViewPager 基本用法第二个页面

图 5.24　ViewPager 基本用法二维码

可以看出，左右滑动屏幕就可以切换不同的 View 了。

5.6.2 ViewPager 导航条

上面的是通过页面中的内容来区别不同的页面，其实 ViewPager 还提供了导航条来区别不同页面和切换页面。下面介绍如何添加顶部或底部导航。Android 提供了两种方式供开发者选择，即 PagerTitleStrip 和 PagerTabStrip，下面分别介绍。

1. PagerTitleStrip

API 中这么定义：PagerTitleStrip 是一个非交互的页面指示器，一般指示 ViewPager 中的前一页、当前页和下一页三个页面。可以通过 PagerTitleStrip 标签添加到 xml 布局中。我们可以设置 layout_gravity 属性为 TOP 或者 BOTTOM 来决定在页面顶部或者底部显示，添加 PagerTitleStrip 要在适配器中覆写 getPageTitle 方法。

上面是抽象的理论描述，下面通过实例来看一下如何在 ViewPager 中添加 PagerTitleStrip 控件。

主布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v4.view.PagerTitleStrip
            android:id="@+id/pagerTitleStrip"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </android.support.v4.view.PagerTitleStrip>
    </android.support.v4.view.ViewPager>
</RelativeLayout>
```

PagerTitleStrip 标签也要设置全路径并放在 ViewPager 标签内，默认没有添加 layout_gravity 属性，标签显示在页面顶部，若想设置在底部，添加这一属性设置其值为 BOTTOM 即可。

修改适配器类如下：

```
public class MyViewPagerAdapter extends PagerAdapter {
    private List<View> datas;
    private List<String> titles;

    public MyViewPagerAdapter(List<View> datas, List<String> titles) {
        this.datas = datas;
        this.titles = titles;
    }
```

```
        // 省略部分相同代码
        @Override
        public CharSequence getPageTitle(int position) {
            return titles.get(position);
        }
    }
```

为了方便观察，对上一个实例增加或修改的代码部分进行了加粗。首先修改了构造方法，多传入了一个标题的数据集；然后覆写了一个 getPagerTitle 的方法，这个方法可以根据 position 参数返回对应的 title。

MainActivity.java 代码如下：

```
public class MainActivity extends Activity {
    private ViewPager mViewPager;
    private PagerTitleStrip mPagerTitleStrip;
    private List<View> mDatas;
    private List<String> mTitles;
    private MyViewPagerAdapter myViewPagerAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mViewPager = (ViewPager) findViewById(R.id.viewPager);
        mPagerTitleStrip = (PagerTitleStrip) findViewById(R.
        id.pagerTitleStrip);
        initDatas();
        myViewPagerAdapter = new MyViewPagerAdapter(mDatas, mTitles);
        mViewPager.setAdapter(myViewPagerAdapter);
    }

    private void initDatas() {
        mDatas = new ArrayList<>();
        mTitles = new ArrayList<>();
        View view1 = LayoutInflater.from(this).inflate(R.layout.layout1, null);
        View view2 = LayoutInflater.from(this).inflate(R.layout.layout2, null);
        View view3 = LayoutInflater.from(this).inflate(R.layout.layout3, null);
        mDatas.add(view1);
        mDatas.add(view2);
        mDatas.add(view3);
        mTitles.add("第一页");
        mTitles.add("第二页");
        mTitles.add("第三页");
    }
}
```

与上一个实例相比，这里添加了一个标题的数据集titles，初始化MyViewPagerAdapter时传入了两个参数：页面布局数据集（mDatas）和标题数据集(mTitles)，标题数据集数据将传到自定义的适配器中。

运行实例并向右滑动屏幕，如图 5.25 所示。可以看出页面切换到第二个页面，同时

顶部的页面标签也切换到了“第二页”。查看动态图，请扫描图 5.26 中的二维码。

图 5.25　ViewPager 之 PagerTitleStrip 用法

图 5.26　ViewPager 之 PagerTitleStrip 用法二维码

单击顶部的标题栏，不会进行页面切换，正如 API 文档中所描述的，non-interactive indicator 只能作为一个页面指示器，不具有交互作用。下面介绍具有交互效果的 PagerTabStrip。

2. PagerTabStrip

API 中这么描述 PagerTabStrip：

```
    PagerTabStrip is an interactive indicator of the current, next, and
previous pages of a ViewPager. It is intended to be used as a child view of a
ViewPager widget in your XML layout. Add it as a child of a ViewPager in your
layout file and set its android:layout_gravity to TOP or BOTTOM to pin it to
the top or bottom of the ViewPager. The title from each page is supplied by
the method getPageTitle(int) in the adapter supplied to the ViewPager.
    For a non-interactive indicator, see PagerTitleStrip.
```

大致含义如下：PagerTabStrip 是一个关于当前页、下一页和上一页可交互的页面指示器，作为一个子项布局在 ViewPager 控件内部。同时，也可以通过设置 layout_gravity 属性为 TOP 或 BOTTOM 来决定显示在页面顶部或底部。每个页面标题是通过适配器类中覆写 getPageTitle 方法提供给 ViewPager 的。最后一句也点明了，若要使用一个非交互指示器，可以参考 PagerTitleStrip。

从 API 文档可以看出，两个方式使用方法一样，因此，这里只要在布局文件中更换一下标签，代码如下：

```
    <?xml version="1.0" encoding="utf-8"?>
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <android.support.v4.view.ViewPager
            android:id="@+id/viewPager"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <android.support.v4.view.PagerTabStrip
                android:id="@+id/pagerTabStrip"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </android.support.v4.view.PagerTabStrip>
    </android.support.v4.view.ViewPager>
</RelativeLayout>
```

上述代码将标签换成 android.support.v4.view.PagerTabStrip。

MainActivity.java 中，将 PagerTitleStrip 换成 PagerTabStrip 即可，其余代码不变：

```
    private PagerTabStrip pagerTabStrip= (PagerTabStrip)findViewById(R.
id.pagerTabStrip);
```

运行实例并单击顶部 Tab“第二页”，结果如图 5.27 所示。

图 5.27　ViewPager 之 PagerTabStrip 用法第二个页面

单击顶部指示页，可以进行页面切换，还有动画效果。除此之外，相对 PagerTitleStrip 而言，PagerTabStrip 当前页的下面还多了一个小横标。以上功能基本实现了，下面来研究如何让外观变得更漂亮。Android 提供了一些方法用于改变指示栏的样式。常用方法如表 5.4 所示。

表 5.4　PagerTabStrip 的常用方法

方　法	说　明
setBackgroundColor(int color)	设置背景颜色
setBackgroundResource(int resId)	设置背景图片
setDrawFullUnderline(boolean drawFull)	设置是否显示分隔栏
setTabIndicatorColor(int color)	设置指示器颜色
setTextColor(int color)	设置指示器文字颜色

在 MainActivity.java 的 onCreate 方法中加入如下代码：

```
    mPagerTabStrip = (PagerTabStrip) findViewById(R.id.pagerTabStrip);
    // 取消标题栏子 View 之间的分割线
    mPagerTabStrip.setDrawFullUnderline(false);
    // 改变指示器颜色为白色
    mPagerTabStrip.setTabIndicatorColor(Color.YELLOW);
    // 该变字体颜色为白色
    mPagerTabStrip.setTextColor(Color.GREEN);
```

```
    // 设置字体大小
    mPagerTabStrip.setTextSize(1,24);
    // 设置标题栏背景图片
    mPagerTabStrip.setBackgroundResource(
            android.R.drawable.screen_background_light_transparent);
```

再次运行实例并向右滑动切换界面，如图 5.28 所示。查看动态图，请扫描图 5.29 中的二维码。

图 5.28　PagerTabStrip 自定义标题栏

图 5.29　PagerTabStrip 自定义标题栏二维码

可以看出，除了通过左右滑动切换页面之外，还可以单击顶部标题来切换页面。