第 12 章 对 话 框 12

对话框既能引起用户的注意也可接收用户的输入。在提示重要信息或提供用户选项方面，它都非常有用。本章，我们将添加一个对话框，以供用户改变crime记录日期。点击CrimeFragment上的日期按钮，即可弹出对话框，如图12-1所示。

图12-1　可供选择crime日期的对话框

图12-1所示的对话框是AlertDialog类的一个实例。实际开发中，AlertDialog类是一个经常会用到的多用途Dialog子类。

（AlertDialog还有一个DatePickerDialog子类。从类名来看，它应该就是满足我们当前需

求的类。然而，直至本书写作之时，DatePickerDialog类还存在一些问题。而使用AlertDialog要比解决这些问题容易得多。）

图12-1所示的AlertDialog视图封装在DialogFragment（Fragment的子类）实例中。不使用DialogFragment，也可显示AlertDialog视图，但Android开发原则不推荐这种做法。使用FragmentManager管理对话框，可使用更多配置选项来显示对话框。

另外，如果设备发生旋转，独立配置使用的AlertDialog会在旋转后消失，而配置封装在fragment中的AlertDialog则不会有此问题。

就CriminalIntent应用来说，我们首先会创建一个名为DatePickeFragment的DialogFragment子类。然后，在DatePickeFragment中，创建并配置一个显示DatePicker组件的AlertDialog实例。DatePickeFragment也将交给CrimePagerActivity来托管。

图12-2展示了以上各对象间的关系。

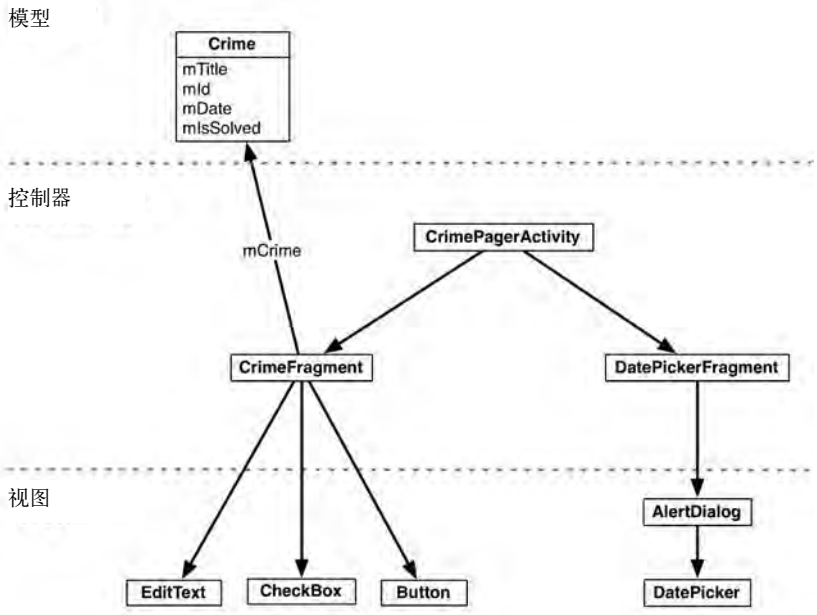

图12-2　两个由CrimePagerActivity托管的fragment对象图解

总结下来，要实现对话框显示，首先应完成以下任务：

- 创建DatePickeFragment类；
- 创建AlertDialog；
- 通过FragmentManager在屏幕上显示对话框。

在本章的后面，我们将配置使用DatePicker，并实现CrimeFragment和DatePickerFragment之间必要数据的传递。

继续学习之前，请参照代码清单12-1添加所需的字符串资源。

代码清单12-1　为对话框标题添加字符串资源（values/strings.xml）

```
<resources>

  ...
  <string name="crime_solved_label">Solved?</string>
  <string name="crimes_title">Crimes</string>
  <string name="date_picker_title">Date of crime:</string>

</resources>
```

12.1　创建 DialogFragment

创建一个名为DatePickerFragment的新类，并设置其DialogFragment超类为支持库中的android.support.v4.app.DialogFragment类。

DialogFragment类有如下方法：

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

在屏幕上显示DialogFragment时，托管activity的FragmentManager会调用以上方法。

在DatePickerFragment.java中，添加onCreateDialog(...)方法的实现代码，创建一个带标题栏和OK按钮的AlertDialog，如代码清单12-2所示。（DatePicker组件稍后会添加。）

代码清单12-2　创建DialogFragment（DatePickerFragment.java）

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

如代码清单12-2所示，使用AlertDialog.Builder类，以流接口（fluent interface）的方式创建了一个AlertDialog实例。下面我们来详细解读一下这段代码。

首先，通过传入Context参数给AlertDialog.Builder类的构造方法，返回一个AlertDialog.Builder实例。

然后，调用以下两个AlertDialog.Builder方法，配置对话框：

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

调用setPositiveButton(...)方法，需传入两个参数：一个字符串资源以及一个实现DialogInterface.OnClickListener接口的对象。代码清单12-2中传入的资源ID是Android的OK常量。对于监听器参数，暂时传入null值。我们会在本章后面实现一个监听器接口。

（Android有3种可用于对话框的按钮：positive按钮、negative按钮以及neutral按钮。用户点击positive按钮接受对话框展现信息。如同一对话框上放置多个按钮，按钮的类型与命名决定着它们

在对话框上显示的位置。在Froyo以及Gingerbread版本的设备上，positive按钮出现在对话框的最左端。而在较新版本设备上，positive按钮则出现在对话框的最右端。）

最后，调用AlertDialog.Builder.create()方法，返回已配置完成的AlertDialog实例，完成对话框的创建。

使用AlertDialog和AlertDialog.Builder类，还可实现更多个性化的需求。可查阅开发文档了解更多相关使用信息。接下来，我们开始学习如何在屏幕上显示对话框。

12.1.1 显示DialogFragment

和其他fragment一样，DialogFragment实例也是由托管activity的FragmentManager管理着的。

要将DialogFragment添加给FragmentManager管理并放置到屏幕上，可调用fragment实例的以下方法：

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

string参数可唯一识别存放在FragmentManager队列中的DialogFragment。可按需选择究竟是使用FragmentManager还是FragmentTransaction。如传入FragmentManager参数，则事务可自动创建并提交。这里我们选择传入FragmentManager参数。

在CrimeFragment中，为DatePickerFragment添加一个tag常量。然后，在onCreateView(...)方法中，删除禁用日期按钮的代码。为实现用户点击日期按钮展现DatePickerFragment界面，实现mDateButton按钮的OnClickListener监听器接口，如代码清单12-3所示。

代码清单12-3 显示DialogFragment（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String DIALOG_DATE = "date";

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                        .getSupportFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(fm, DIALOG_DATE);
            }
        });
```

```
            mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
            ...

            return v;
        }

        ...
    }
```

运行CriminalIntent应用。点击日期按钮弹出对话框，单击OK按钮消除对话框，如图12-3所示。

图12-3　带有标题和OK按钮的AlertDialog

12.1.2　设置对话框的显示内容

接下来，使用下列AlertDialog.Builder的setView(...)方法，添加DatePicker组件到AlertDialog对话框：

```
public AlertDialog.Builder setView(View view)
```

该方法配置对话框，实现在标题栏与按钮之间显示传入的View对象。

在包浏览器中，创建一个名为dialog_date.xml的布局文件，设置其根元素为DatePicker。该布局仅包含一个View对象，即我们生成并传给setView(...)方法的DatePicker视图。

参照图12-4，配置DatePicker的XML布局文件。

DatePicker
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:id="**@+id/dialog_date_datePicker**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:calendarViewShown="**false**"

图12-4 DatePicker布局（layout/dialog_date.xml）

在DatePickerFragment.onCreateDialog(...)方法中，生成DatePicker视图并添加到对话框中，如代码清单12-4所示。

代码清单12-4 添加DatePicker给AlertDialog（DatePickerFragment.java）

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

运行CriminalIntent应用。点击日期按钮，确认对话框上是否出现了DatePicker视图，如图12-5所示。

图12-5 显示DatePicker的AlertDialog

采用下列代码即可完成DatePicker对象的创建，又为何要费事的去定义XML布局文件，再去生成视图对象呢？

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker dp = new  DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(dp)
        ...
        .create();
}
```

这是因为，如想调整对话框的显示内容时，直接修改布局文件会更容易些。例如，如想在对话框的DatePicker旁再添加一个TimePicker，这时，只需更新布局文件，即可完成新视图的显示。

至此，将对话框显示在屏幕上的工作就完成了。下一节，我们会将DatePicker同Crime的日期关联起来，并支持用户对其进行修改。

12.2　fragment 间的数据传递

前面，我们已经实现了activity之间以及基于fragment的activity之间的数据传递。现在需实现由同一activity托管的两个fragment，即CrimeFragment和DatePickerFragment间的数据传递，如图12-6。

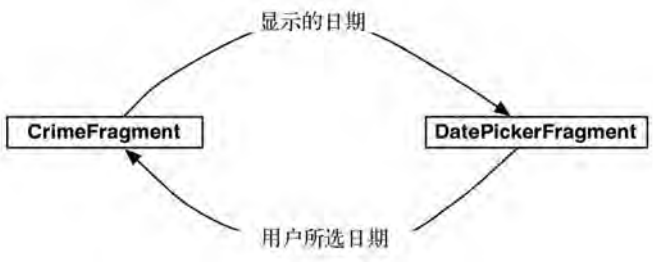

图12-6　CrimeFragment与DatePickerFragment间的对话

要传递crime的记录日期给DatePickerFragment，需实现一个newInstance(Date)方法，然后将Date作为argument附加给fragment。

为返回新日期给CrimeFragment，并实现模型层以及对应视图的更新，需将日期打包为extra并附加到Intent上，然后调用CrimeFragment.onActivityResult(...)方法，并传入准备好的Intent参数，如图12-7所示。

在本章后面的代码实施中，可以看到，我们没有选择调用托管activity的Activity.onActivityResult(...)方法，而是调用了Fragment.onActivityResult(...)方法，这似乎有点奇怪。然而，通过调用onActivityResult(...)方法将数据从一个fragment返还给另一个fragment，这种做法不仅行得通，而且可以更灵活地展现对话框fragment。

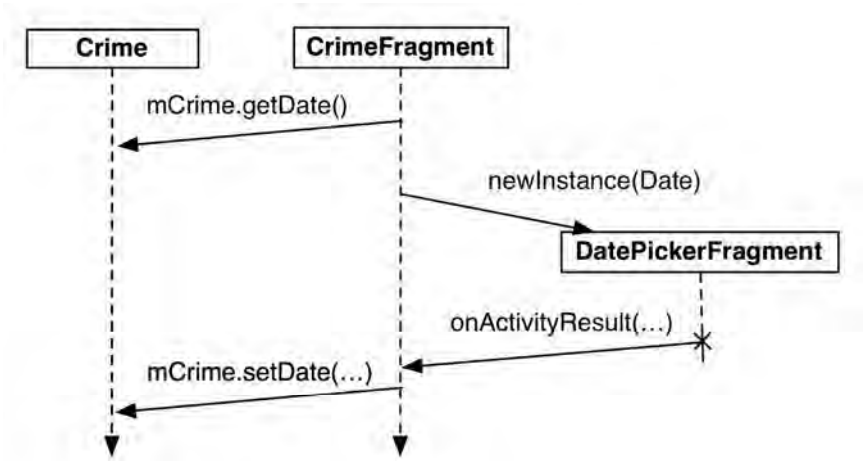

图12-7 CrimeFragment和DatePickerFragment间的事件流

12.2.1 传递数据给DatePickerFragment

要传递crime记录日期给DatePickerFragment，需将记录日期保存在DatePickerFragment的argument bundle中，这样，DatePickerFragment便可直接获取到它。

回顾第10章的内容，我们知道，替代fragment的构造方法，创建和设置fragment argument通常是在一个newInstance()方法中完成的。在DatePickerFragment.java中，添加newInstance(Date)方法，如代码清单12-5所示。

代码清单12-5 添加newInstance(Date)方法（DatePickerFragment.java）

```
public class DatePickerFragment extends DialogFragment {
    public static final String EXTRA_DATE =
        "com.bignerdranch.android.criminalintent.date";

    private Date mDate;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(EXTRA_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);

        return fragment;
    }

    ...
}
```

然后，在CrimeFragment中，用DatePickerFragment.newInstance(Date)方法替换掉Date- PickerFragment的构造方法，如代码清单12-6所示。

代码清单12-6　添加newInstance()方法（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater,
        ViewGroup parent, Bundle savedInstanceState) {
    ...

    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(fm, DIALOG_DATE);
        }
    });

    return v;
}
```

DatePickerFragment需使用Date中的信息来初始化DatePicker对象。然而，DatePicker对象的初始化需整数形式的月、日、年。Date就是个时间戳，它无法直接提供整数形式的月、日、年。

要想获得所需的整数数值，必须首先创建一个Calendar对象，然后用Date对象对其进行配置，即可从Calendar对象中取回所需信息。

在onCreateDialog(...)方法内，从argument中获取Date对象，然后使用它和Calendar对象完成DatePicker的初始化工作，如代码清单12-7所示。

代码清单12-7　获取Date对象并初始化DatePicker（DatePickerFragment.java）

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mDate = (Date)getArguments().getSerializable(EXTRA_DATE);

    // Create a Calendar to get the year, month, and day
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(mDate);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    DatePicker datePicker = (DatePicker)v.findViewById(R.id.dialog_date_datePicker);
    datePicker.init(year, month, day, new OnDateChangedListener() {
        public void onDateChanged(DatePicker view, int year, int month, int day) {
            // Translate year, month, day into a Date object using a calendar
            mDate = new GregorianCalendar(year, month, day).getTime();

            // Update argument to preserve selected value on rotation
            getArguments().putSerializable(EXTRA_DATE, mDate);
```

```
            }
        });

        ...
    }
```

如代码所示，初始化DatePicker对象时，同时也在该对象上设置了OnDateChanged-Listener监听器。这样，用户改变DatePicker内的日期后，Date对象即可得到同步更新。下一节，我们将把该Date对象回传给CrimeFragment。

为防止设备旋转时发生Date数据的丢失，在onDateChanged(...)方法的尾部，我们将Date对象回写保存到了fragment argument中。如发生设备旋转，而DatePickerFragment正显示在屏幕上，那么FragmentManager会销毁当前实例并产生一个新的实例。新实例创建后，FragmentManager会调用它的onCreateDialog(...)方法，这样新实例便可从argument中获得保存的日期数据。相比以前使用onSaveInstanceState(...)方法保存状态，在fragment argument中保存数据应对设备旋转显然更简单。

（如经常使用fragment，可能会困惑为何不直接保存DatePickerFragment？使用保留的fragment处理设备旋转问题（详见第14章）确实是个好办法。但不幸的是，目前DialogFragment类有个bug，会导致保存的实例行为异常，因此，尝试保存DatePickerFragment现在还不是个好的选择。）

现在，CrimeFragment可成功将要显示的日期传递给DatePickerFragment。运行CriminalIntent应用，查看最终效果。

12.2.2 返回数据给 CrimeFragment

为使CrimeFragment接收到DatePickerFragment返回的日期数据，需以某种方式追踪记录二者间的关系。

对于activity的数据回传，我们调用startActivityForResult(...)方法，ActivityManager负责跟踪记录父activity与子activity间的关系。当子activity回传数据后被销毁了，ActivityManager知道接收返回数据的应为哪一个activity。

1. 设置目标fragment

类似于activity间的关联，可将CrimeFragment设置成DatePickerFragment的目标fragment。要建立这种关联，可调用以下Fragment方法：

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

该方法接受目标fragment以及一个类似于传入startActivityForResult(...)方法的请求代码作为参数。随后，目标fragment可使用该请求代码通知是哪一个fragment在返回数据信息。

目标fragment以及请求代码由FragmentManager负责跟踪记录，我们可调用fragment（设置目标fragment的fragment）的getTargetFragment()和getTargetRequestCode()方法获取它们。

在 CrimeFragment.java 中，创建一个请求代码常量，然后将 CrimeFragment 设为DatePickerFragment实例的目标fragment，如代码清单12-8所示。

代码清单12-8　设置目标fragment（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String DIALOG_DATE = "date";
    private static final int REQUEST_DATE = 0;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                        .getSupportFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                    .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(fm, DIALOG_DATE);
            }
        });

        return v;
    }

    ...
}
```

2. 传递数据给目标fragment

建立CrimeFragment与DatePickerFragment间的联系后，需将数据返还给CrimeFragment。返回的日期数据将作为extra附加给Intent。

使用什么方法发送intent信息给目标fragment？不要奇怪，我们使用DatePickerFragment的类方法将intent传入CrimeFragment.onActivityResult(int, int, Intent)方法。

Activity.onActivityResult(...)方法是ActivityManager在子activity销毁后调用的父activity方法。处理activity间的数据返回时，无需亲自动手，ActivityManager会自动调用Activity.onActivityResult(...)方法。父activity接收到Activity.onActivityResult(...)方法的调用后，其FragmentManager会调用对应fragment的Fragment.onActivityResult(...)方法。

处理由同一activity托管的两个fragment间的数据返回时，可借用Fragment.onActivityResult(...)方法。因此，直接调用目标fragment的Fragment.onActivityResult(...)方法，即可实现数据的回传。该方法有我们需要的信息：

- 一个与传入setTargetFragment(...)方法相匹配的请求代码，用以告知目标fragment返回结果来自于哪里。
- 一个决定下一步该采取什么行动的结果代码。
- 一个含有extra数据信息的Intent。

在DatePickerFragment类中，新建一个sendResult(...)私有方法。通过该方法，创建一个intent，将日期数据作为extra附加到intent上。最后调用CrimeFragment.onActivityResult(...)方法。在onCreateDialog(...)方法中，取代setPositiveButton(...)的null参数，实现一个DialogInterface.OnClickListener监听器接口。然后在监听器接口的onClick(...)方法中，调用新建的sendResult(...)私有方法并传入结果代码，如代码清单12-9所示。

代码清单12-9 回调目标fragment（DatePickerFragment.java）

```
    private void sendResult(int resultCode) {
        if (getTargetFragment() == null)
            return;

        Intent i = new Intent();
        i.putExtra(EXTRA_DATE, mDate);

        getTargetFragment()
            .onActivityResult(getTargetRequestCode(), resultCode, i);
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        ...

        return new AlertDialog.Builder(getActivity())
            .setView(v)
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .setPositiveButton(
                android.R.string.ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
                        sendResult(Activity.RESULT_OK);
                    }
                })
            .create();
    }
```

在CrimeFragment中，覆盖onActivityResult(...)方法，从extra中获取日期数据，设置对应Crime的记录日期，然后刷新日期按钮的显示，如代码清单12-10所示。

代码清单12-10 响应DatePicker对话框（CrimeFragment.java）

```
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;
        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
        }
    }
```

在onCreateView(...)与onActivityResult(...)方法中，设置按钮上显示信息的代码完全一样。因此，为避免代码冗余，将其封装到一个公有的updateDate()方法中，然后分别在两

个方法中调用它，如代码清单12-11所示。

代码清单12-11　使用公共的updateDate()方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

   ...

    public void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        ...
        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        updateDate();

    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;
        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
            updateDate();
        }
    }
}
```

日期数据的双向传递完成了。运行CriminalIntent应用，确保可以控制日期的传递与显示。修改某项Crime的日期，确认CrimeFragment视图显示了新的日期。然后返回crime列表项界面，查看对应Crime的日期，确认模型层数据已得到更新。

3. 更为灵活的DialogFragment视图展现

编写需要大量用户输入以及更多空间显示输入要求的应用时，使用onActivityResult(...)方法返还数据给目标fragment还是比较方便的。对于这样的应用，我们需要它能够很好地支持手机和平板设备。

手机的屏幕空间非常有限。因此，通常需要使用一个activity托管全屏的fragment界面，以显示用户输入要求。该子activity会由父activity的fragment以调用startActivityForResult(...)方法的方式启动。子activity被销毁后，父activity会接收到onActivityResult(...)方法的调用请求，并将之转发给启动子activity的fragment，如图12-8所示。

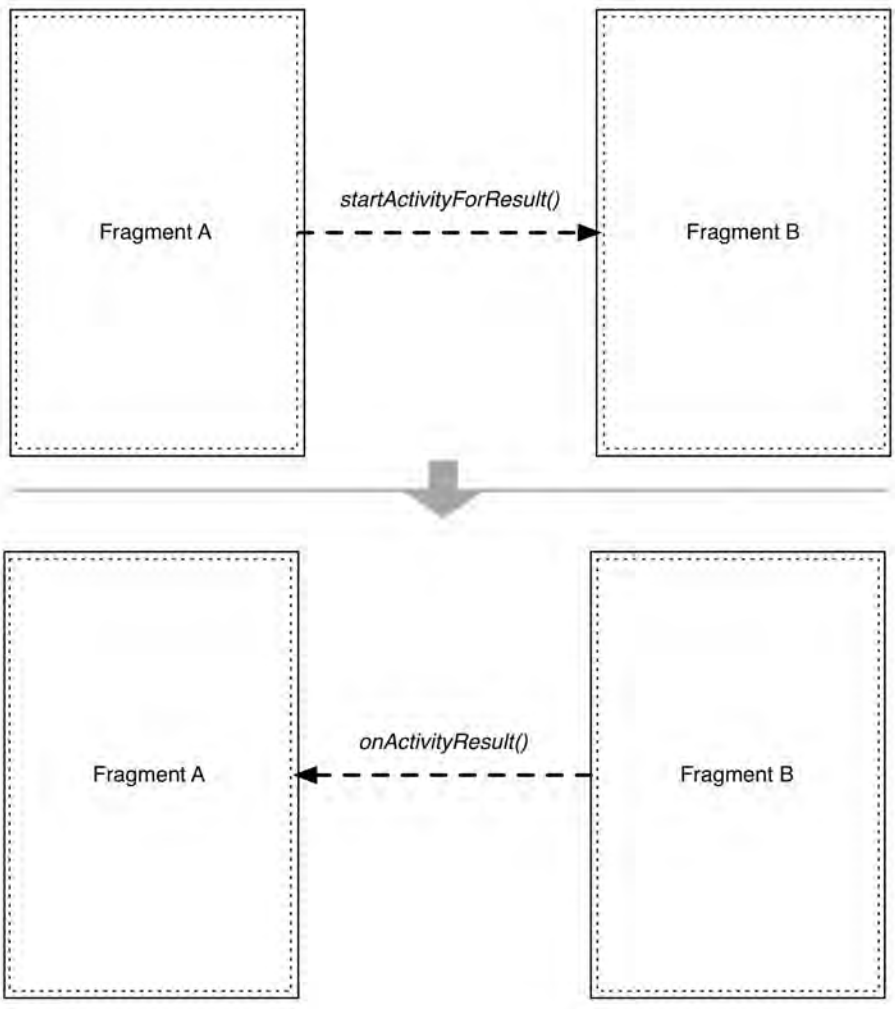

图12-8 手机设备上activity间的数据传递

平板设备的屏幕空间较大。因此，以弹出对话框的方式显示信息和接收用户输入通常会更好。这种情况下，应设置目标fragment并调用对话框fragment的`show(...)`方法。对话框被取消后，对话框fragment会调用目标fragment的`onActivityResult(...)`方法，如图12-9所示。

无论是启动子activity还是显示对话框，fragment的`onActivityResult(...)`方法总会被调用。因此，可使用相同代码实现不同方式的信息呈现。

编写同样的代码用于全屏fragment或对话框fragment时，可选择覆盖`DialogFragment.onCreateView(...)`方法，而非`onCreateDialog(...)`方法，来实现不同设备上的信息呈现。

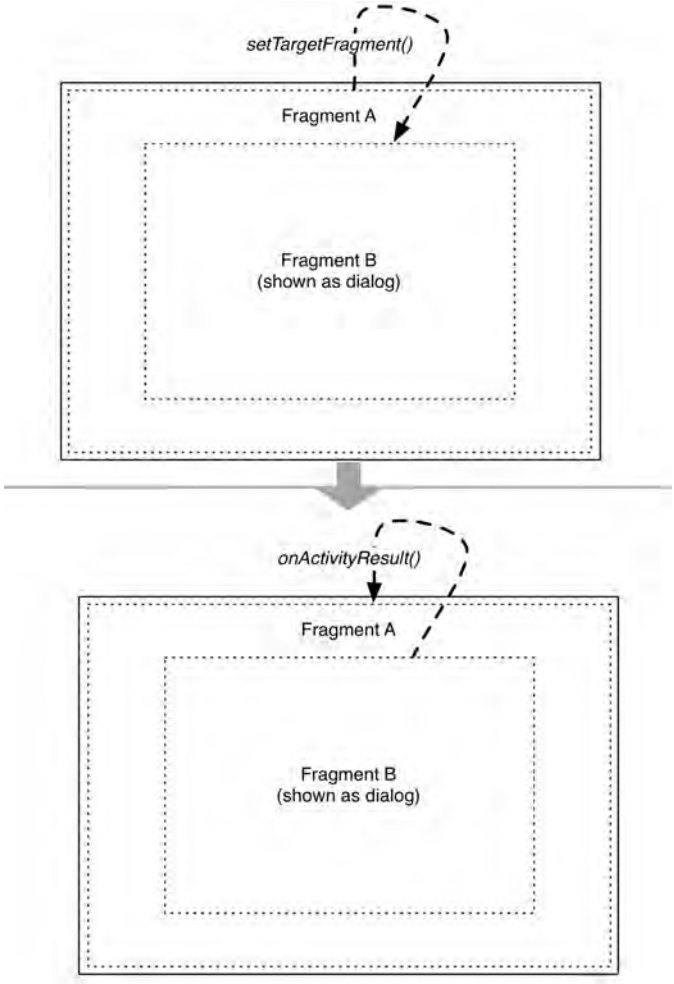

图12-9　平板设备上fragment间的数据传递

12.3　挑战练习：更多对话框

首先看个简单的练习。编写另一个名为TimePickerFragment的对话框fragment，允许用户使用TimePicker组件选择crime发生的具体时间。TimePickerFragment视图界面的弹出显示，是通过在CrimeFragment用户界面上再添加一个按钮实现的。

再来看个有点难度的练习。CrimeFragment用户界面保持只有一个按钮，单击该按钮弹出对话框，让用户选择是修改时间还是修改日期。用户做出选择后，弹出相应的第二个对话框。