第 10 章 使用fragment argument

本章，我们将实现CriminalIntent应用的列表与明细部分的关联。用户点击某个crime列表项时，会生成一个负责托管CrimeFragment的CrimeActivity，并显示出某特定Crime实例的明细信息。

图10-1　从CrimeListActivity中启动CrimeActivity

我们已经在GeoQuiz应用里实现了从一个activity（QuizActivity）中启动另一个activity（CheatActivity）。接下来，我们准备在CriminalIntent应用里实现从fragment中启动CrimeActivity。准确地说，是从CrimeListFragment中启动CrimeActivity实例，如图10-1所示。

10.1　从 fragment 中启动 activity

从fragment中启动activity的实现方式，基本等同于从activity中启动另一activity的实现方式。我们调用Fragment.startActivity(Intent)方法，该方法在后台会调用对应的Activity方法。

在CrimeListFragment的onListItemClick(...)实现方法里，用启动CrimeActivity实例的代码，替换日志记录crime标题的代码，如代码清单10-1所示。（暂时忽略Crime变量未使用的提示信息，下一节会使用它。）

代码清单10-1　启动CrimeActivity（CrimeListFragment.java）

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Get the Crime from the adapter
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
    Log.d(TAG, c.getTitle() + " was clicked");

    // Start CrimeActivity
    Intent i = new Intent(getActivity(), CrimeActivity.class);
    startActivity(i);
}
```

以上代码中，指定要启动的activity为CrimeActivity，CrimeListFragment创建了一个显式intent。至于Intent构造方法需要的Context对象，CrimeListFragment是使用getActivity()方法传入它的托管activity来满足的。

运行CriminalIntent应用。点击任意列表项，屏幕上会出现一个托管CrimeFragment的CrimeActivity，如图10-2所示。

图10-2　空白的CrimeFragment

由于不知道该显示哪个Crime对象的信息，CrimeFragment也就没有显示出特定Crime对象的数据信息。

10.1.1　附加extra信息

通过将mCrimeId值附加到Intent的extra上，我们可以告知CrimeFragment应显示的Crime。在onListItemClick(...)方法中，将用户所选Crime的mCrimeId值附加到用来启动CrimeActivity的intent上。输入代码清单10-2所示代码，Eclipse会报告一个错误信息，这是因为CrimeFragment.EXTRA_CRIME_ID的key值还没有创建。暂时忽略该条错误信息，稍后我们会创建它。

代码清单10-2　启动附加extra的CrimeActivity（CrimeListFragment.java）

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Get the Crime from the adapter
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);

    // Start CrimeActivity
```

```
        Intent i = new Intent(getActivity(), CrimeActivity.class);
        i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
        startActivity(i);
    }
```

创建了显式intent后，调用putExtra(...)方法，传入匹配mCrimeId的字符串key与key值，完成extra信息的准备。这里，由于UUID是Serializable对象，我们调用了可接受Serializable对象的putExtra(...)方法，即putExtra(String, Serializable)方法。

10.1.2 获取extra信息

mCrimeId值现已安全存储到CrimeActivity的intent中。然而，要获取和使用extra信息的是CrimeFragment类。

fragment有两种方式获取保存在activity的intent内的数据信息，即简单直接的方式和复杂灵活的方式。在实现复杂但灵活的方式（该方式涉及到fragment argument的概念）前，我们首先试试简单的方式。

简单起见，CrimeFragment直接使用getActivity()方法获取CrimeActivity的intent。返回至CrimeFragment类，为extra添加key。然后，在onCreate(Bundle)方法中，得到CrimeActivity的intent内的extra信息后，再使用它获取Crime对象，如代码清单10-3所示。

代码清单10-3 获取extra信息并取得Crime对象 (CrimeFragment.java)

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

    private Crime mCrime;

    ...

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        UUID crimeId = (UUID)getActivity().getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);

        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);

}
```

在代码清单10-3中，除了getActivity()方法的调用，获取extra数据的实现代码与activity里获取extra数据的代码一样。getIntent()方法返回用来启动CrimeActivity的Intent。然后调用Intent的getSerializableExtra(String)方法获取UUID并存入变量中。

取得Crime的ID后，利用该ID从CrimeLab单例中调取Crime对象。使用CrimeLab.get(...)方法需要Context对象，因此CrimeFragment传入了CrimeActivity。

10.1.3 使用Crime数据更新CrimeFragment视图

既然CrimeFragment获取了Crime对象，它的视图便可显示该Crime对象的数据。参照代码

清单10-4，更新onCreateView(...)方法，显示Crime对象的标题及解决状态。（显示日期的代码早已就绪）

代码清单10-4　更新视图对象（CrimeFragment.java）

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

    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });

    ...

    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        ...
    });

    ...

    return v;
}
```

运行CriminalIntent应用。选中Crime #4，查看显示了正确crime数据信息的CrimeFragment实例，如图10-3所示。

图10-3　Crime #4列表项的明细内容

10.1.4 直接获取extra信息方式的缺点

只需几行简单的代码，就可实现让fragment直接获取托管activity的intent。然而，这种方式是以牺牲fragment的封装性为代价的。CrimeFragment不再是可复用的构建单元，因为它总是需要由某个具体activity托管着，该activity的Intent又定义了名为EXTRA_CRIME_ID的extra。

就CrimeFragment类来说，这看起来合情合理。但这也意味着，按照当前的编码实现，CrimeFragment便再也无法用于任何其他的activity了。

一个比较好的做法是，将mCrimeId存储在CrimeFragment的某个地方，而不是将它保存在CrimeActivity的私有空间里。这样，无需依赖于CrimeActivity的intent内指定extra的存在，CrimeFragment就能获取自己所需的extra数据信息。fragment的“某个地方”实际就是它的arguments bundle。

10.2 fragment argument

每个fragment实例都可附带一个Bundle对象。该bundle包含有key-value对，我们可以如同附加extra到Activity的intent中那样使用它们。一个key-value对即一个argument。

要创建fragment argument，首先需创建Bundle对象。然后，使用Bundle限定类型的“put”方法（类似于Intent的方法），将argument添加到bundle中（如以下代码所示）。

```
Bundle args = new Bundle();
args.putSerializable(EXTRA_MY_OBJECT, myObject);
args.putInt(EXTRA_MY_INT, myInt);
args.putCharSequence(EXTRA_MY_STRING, myString);
```


10.2.1 附加argument给fragment

附加argument bundle给fragment，需调用Fragment.setArguments(Bundle)方法。注意，该任务必须在fragment创建后、添加给activity前完成。

为满足以上苛刻的要求，Android开发者遵循的习惯做法是：添加名为newInstance()的静态方法给Fragment类。使用该方法，完成fragment实例及bundle对象的创建，然后将argument放入bundle中，最后再附加给fragment。

托管activity需要fragment实例时，需调用newInstance()方法，而非直接调用其构造方法。而且，为满足fragment创建argument的要求，activity可传入任何需要的参数给newInstance()方法。

如代码清单10-5所示，在CrimeFragment类中，编写可以接受UUID参数的newInstance(UUID)方法，通过该方法，完成arguments bundle以及fragment实例的创建，最后附加argument给fragment。

代码清单10-5 编写newInstance(UUID)方法（CrimeFragment.java）

```
public static CrimeFragment newInstance(UUID crimeId) {
    Bundle args = new Bundle();
    args.putSerializable(EXTRA_CRIME_ID, crimeId);

    CrimeFragment fragment = new CrimeFragment();
```

```
        fragment.setArguments(args);

        return fragment;
    }
```

现在，当CrimeActivity创建CrimeFragment时，应调用CrimeFragment.newInstance(UUID)方法，并传入从它的extra中获取的UUID参数值。回到CrimeActivity类中，在createFragment()方法里，从CrimeActivity的intent中获取extra数据信息，并将之传入CrimeFragment.newInstance(UUID)方法，如代码清单10-6所示。

代码清单10-6　使用newInstance(UUID)方法（CrimeActivity.java）

```
    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();

        UUID crimeId = (UUID)getIntent()
            .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);

        return CrimeFragment.newInstance(crimeId);
    }
```

注意，交互的activity和fragment不需要也无法同时保持通用独立性。CrimeActivity必须了解CrimeFragment的内部细节，比如知晓它内部有一个 newInstance(UUID)方法 。这很正常。托管activity就应该知道有关托管fragment方法的细节，但fragment则不必知道其托管activity的细节问题。至少在需要保持fragment通用独立性的时候是如此。

10.2.2　获取argument

fragment在需要获取它的argument时，会先调用Fragment类的getArguments()方法，接着再调用Bundle的限定类型的“get”方法，如getSerializable(...)方法。

现在回到CrimeFragment.onCreate(...)方法中，调整代码，改为从fragment的argument中获取UUID，如代码清单10-7所示。

代码清单10-7　从argument中获取crime ID（CrimeFragment.java）

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        UUID crimeId = (UUID)getActivity().getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);

        UUID crimeId = (UUID)getArguments().getSerializable(EXTRA_CRIME_ID);

        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);

    }
```

运行CriminalIntent应用。虽然运行结果仍与之前一致，但我们应该感到由衷地高兴。因为我们不仅保持了CrimeFragment类的独立性，又为下一章实现CriminalIntent应用更为复杂的列表项导航打下了良好基础。

10.3 重新加载显示列表项

运行CriminalIntent应用，点击某个列表项，然后修改对应的Crime明细信息。这些修改的数据被保存至模型层，但返回列表后，列表视图并没有发生改变。下面我们来处理这个问题。

如模型层保存的数据发生改变（或可能发生改变），应通知列表视图的adapter，以便其及时获取最新数据并重新加载显示列表项。在适当的时点，与系统的ActivityManager回退栈协同运作，可以完成列表项的刷新。

CrimeListFragment启动CrimeActivity实例后，CrimeActivity被置于回退栈顶。这导致原先处于栈顶的CrimeListActivity实例被暂停并停止。

用户点击后退键返回到列表项界面，CrimeActivity随即被弹出栈外并被销毁。CrimeListActivity继而被重新启动并恢复运行。应用的回退栈如图10-4所示。

图10-4 CriminalIntent应用的回退栈

CrimeListActivity恢复运行状态后，操作系统会向它发出调用onResume()生命周期方法的指令。CrimeListActivity接到指令后，它的FragmentManager会调用当前被activity托管的fragment的onResume()方法。这里，CrimeListFragment即唯一的目标fragment。

在CrimeListFragment中，覆盖onResume()方法刷新显示列表项，如代码清单10-8所示。

代码清单10-8 在onResume()方法中刷新列表项（CrimeListFragment.java）

```
@Override
public void onResume() {
```

```
        super.onResume();
        ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
    }
```

为什么选择覆盖onResume()方法来刷新列表项显示，而非onStart()方法呢？当一个activity位于我们的activity之前时，我们无法保证自己的activity是否会被停止。如前面的activity是透明的，则我们的activity可能只会被暂停。对于此场景下暂停的activity，onStart()方法中的更新代码是不会起作用的。一般来说，要保证fragment视图得到刷新，在onResume()方法内更新代码是最安全的选择。

运行CriminalIntent应用。选择某个crime项并修改其明细内容。然后返回到列表项界面，如预期那样，列表项立即刷新反映了更改的内容。

经过前两章的开发，CriminalIntent应用已获得大幅更新。现在，我们来看看更新后的应用对象图解，如图10-5所示。

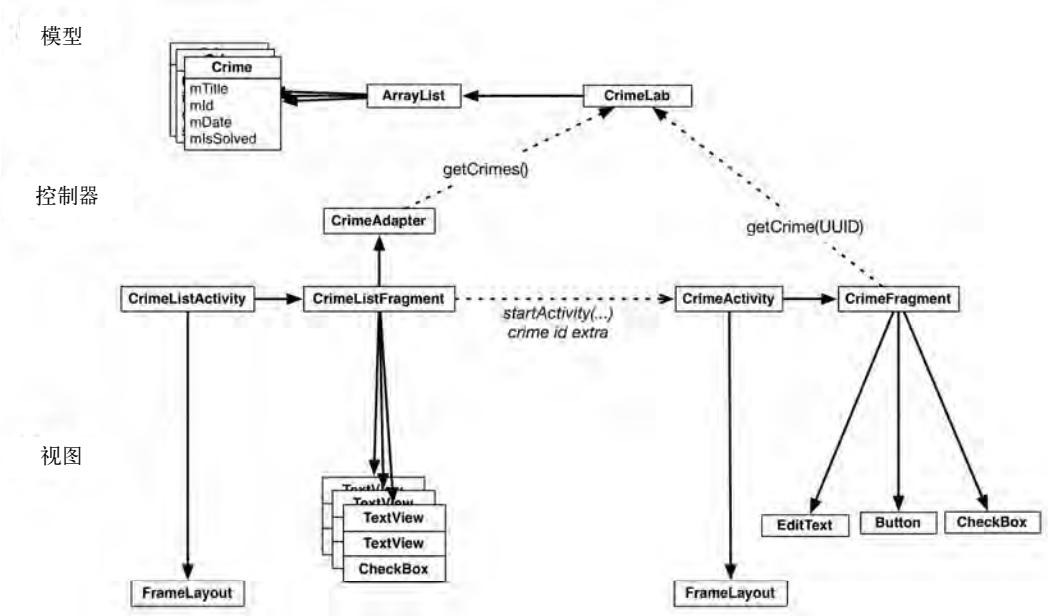

图10-5　应用对象图解更新版

10.4　通过 fragment 获取返回结果

如需从被启动的activity获取返回结果，可使用与GeoQuiz应用中类似的实现代码。但本章我们无需这么做。本章我们将选择调用Fragment.startActivityForResult(...)方法，而非Activity的startActivityForResult(...)方法；选择覆盖Fragment.onActivityResult(...)方法，而非Activity.onActivityResult(...)方法。

```
public class CrimeListFragment extends ListFragment {
    private static final int REQUEST_CRIME = 1;

    ...

    public void onListItemClick(ListView l, View v, int position, long id) {
        // Get the Crime from the adapter
        Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
        Log.d(TAG, c.getTitle() + " was clicked");
        // Start CrimeActivity
        Intent i = new Intent(getActivity(), CrimeActivity.class);
        startActivityForResult(i, REQUEST_CRIME);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CRIME) {
            // Handle result
        }
    }
}
```

除将返回结果从托管activity传递给fragment的额外实现代码之外，`Fragment.startActivityForResult(Intent,int)`方法的实现代码与`Activity`的同名方法基本相同。

从fragment中返回结果的处理稍有不同。fragment能够从activity中接收返回结果，但其自身无法产生返回结果。只有activity拥有返回结果。因此，尽管`Fragment`有自己的`startActivityForResult(...)`和`onActivityResult(...)`方法，但却不具有任何`setResult(...)`方法。

相反，我们应通知托管activity返回结果值。具体代码如下：

```
public class CrimeFragment extends Fragment {
    ...

    public void returnResult() {
        getActivity().setResult(Activity.RESULT_OK, null);
    }
}
```

我们会在第20章有关CriminalIntent应用的讲解中，学习应如何从fragment返回一个activity结果。