ГЛАВА 12

Исследование пакетов

В главах 1–11 были приведены основы организации платформы Android. Однако материал этих глав представлял простой и прямолинейный путь изучения Android. В главах 12–15 мы детально рассмотрим другую грань ядра Android.

Мы начнем это исследование с анализа внутреннего устройства пакетов Android, процесса их подписания, разделения данных между пакетами и библиотечных проектов Android. Вы разберетесь в контексте процесса Linux, в котором выполняется файл `.apk`. Вы увидите, как множество файлов `.apk` могут разделять данные и ресурсы, предоставляемые этим контекстом.

В главе 10 предлагалось введение в процесс подписания файлов пакетов Android, а в этой главе вы ознакомитесь со значением, влиянием и использованием подписанных JAR-файлов. В контексте разделения данных будут описаны библиотечные проекты Android, их работа и применения для совместного использования ресурсов и кода.

Начнем обсуждение, вернувшись назад к структуре файла `.apk`, поскольку он формирует основу для процесса Android.

Пакеты и процессы

Как было показано в предыдущих главах, разработка приложения Android завершается построением файла `.apk`. Затем этот файл `.apk` подписывается и разворачивается на устройстве. Давайте рассмотрим более подробно, что собой представляют пакеты Android.

Детали спецификации пакета

Каждый файл `.apk` уникально идентифицируется своим именем корневого пакета, которое указано в файле манифеста. Ниже приведен пример определения пакета, который будет использоваться в этой главе (имя пакета выделено полужирным):

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.androidbook.library.testlibraryapp"
     ...>
     ...остальные узлы XML
</manifest>
```

Если вы разработали этот пакет, подписали его и установили на устройстве, то никто другой не сможет обновить его. Имя пакета привязано к его подписи. Это значит, что разработчик с другой подписью не сможет подписать и установить Java-пакет с тем же самым полностью квалифицированным именем.

Трансляция имени пакета в имя процесса

В Android имя пакета используется в качестве имени процесса, внутри которого выполняются компоненты этого пакета. Android также выделяет для этого процесса пакета уникальный идентификатор пользователя, от имени которого он выполняется. Этот идентификатор пользователя в сущности представляет собой идентификатор для лежащей в основе ОС Linux. Для просмотра этой информации необходимо обратиться к деталям установленного пакета.

Вывод списка установленных пакетов

Чтобы просмотреть список установленных пакетов в эмуляторе, необходимо перейти к браузеру пакетов, используя путь Home⇨Dev Tools⇨Package Browser (Домой⇨Инструменты разработки⇨Браузер пакетов). (Обратите внимание, что на реальном устройстве похожий браузер пакетов может быть как обнаружен, так и нет. Все зависит от выпуска Android.)

Открыв список пакетов, можно выделить пакет для определенного приложения, например, браузера, и щелкнуть на нем. В результате отобразится экран детальной информации о пакете, который похож на показанный на рис. 12.1.

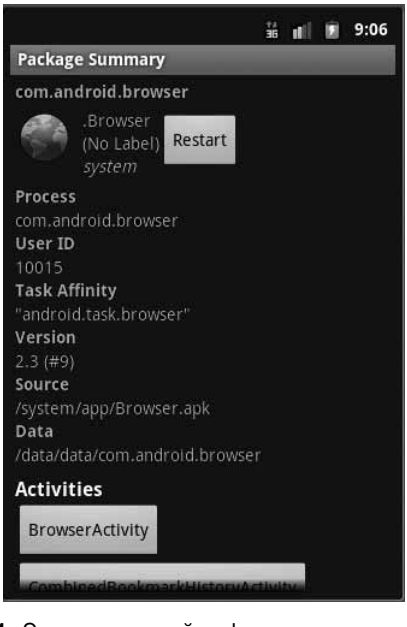

Рис. 12.1. Экран детальной информации о пакете Android

На рис. 12.1 можно видеть имя процесса, указываемое именем Java-пакета в файле манифеста, и уникальный идентификатор, назначенный этому пакету.

В случае браузера файл манифеста будет иметь указываемое на этот пакет имя вида `com.android.browser` (отображаемое атрибутом Process (Процесс) на рис. 12.1).

Любые ресурсы, созданные этим процессом или пакетом, защищаются идентификатором пользователя Linux. На экране детальной информации о пакете также перечислены внутренние компоненты пакета. Примерами таких компонентов могут быть действия, службы и широковещательные приемники.

Удаление пакета с помощью браузера пакетов

В браузере пакетов можно также удалить пакет из эмулятора, для чего понадобится выполнить следующие шаги.

1. Выделите пакет.
2. Щелкните на кнопке Menu (Меню).
3. Щелкните на ссылке delete package (удалить пакет) для удаления пакета.

Поскольку процесс привязан к имени пакета, а имя пакета связано с его подписью, подписи играют важную роль в защите данных, принадлежащих пакету. Чтобы полностью разобраться в этом, давайте ознакомимся с сущностью процесса подписания пакетов.

Еще раз о процессе подписания пакетов

В главе 10 было дано введение в механизм подписания приложения перед его установкой на устройстве. Однако потребности и последствия процесса подписания пакетов не рассматривались.

Например, когда мы загружаем приложение и устанавливаем его в среде Windows или другой ОС, подписывать его не нужно. Почему подписание является обязательным на устройствах Android? Что в действительности означает процесс подписания? Что он обеспечивает? Существуют ли параллели между реальным миром и процессом подписания? В этом разделе мы попытаемся получить ответы на все перечисленные вопросы.

Каждый установленный на устройстве пакет должен иметь уникальное или отличительное имя Java-пакета. Если вы попытаетесь установить новый пакет, имя которого совпадает с именем одного из существующих пакетов, устройство не позволит это сделать до тех пор, пока не будет удален предыдущий пакет. Чтобы разрешить обновление пакета такого типа, понадобится ассоциировать этот пакет с тем же самым издателем приложения. Это делается с помощью цифровых подписей. В следующих разделах этой главы будет показано, что процесс подписания резервирует разработчику необходимое имя для пакета.

Давайте взглянем на пару сценариев, которые позволят разобраться с цифровыми подписями.

Цифровые подписи: сценарий 1

Вообразите, что вы — коллекционер вин, проживающий в месте, где вина являются редкостью, например, в Сахаре. Кроме того, скажем, виноделы всего мира посылают вам бочонки с вином для сохранения или продажи.

Как коллекционер вин, вы замечаете, что вино в каждом бочонке имеет специфический оттенок и цвет, отличающий его от других. При дальнейшем исследовании вы выясняете, что если вино в двух бочонках имеет одинаковый оттенок, оно всегда поставляется одним производителем. После более глубоких исследований выясняется, что каждый винодел имеет секретный рецепт оттенка, который хранится под замком и никому не раскрывается. Это объясняет причину, по которой каждое вино отличается от других, а два вида вина с одним и тем же оттенком должны поступать от одного производителя. Разумеется, это определение совершенно не раскрывает идентичность винодела, а просто говорит о том, что каждый винодел является уникальным.

Оттенок становится подписью винодела, своего рода фамильной печатью, и винодел скрывает свою подпись от посторонних.

Важное различие в рассмотренном примере состоит в том, что для вас, как коллекционера, нет никакого способа узнать, какой винодел прислал конкретную партию вина — ни имя, ни адрес с подписью не ассоциированы. Но даже если бы они были, вполне возможно, что какой-то винодел отправил вино, произведенное другим. В этом случае вы могли бы предположить, что два бочонка с вином, отправленные с одного адреса, но имеющие разные оттенки, произведены разными виноделами, которые просто физически находятся по одному и тому же адресу.

Цифровые подписи: сценарий 2

А теперь рассмотрим другой сценарий, который может возникнуть в реальности. Посещая другие страны, вы включаете радио и слышите множество песен. Вы можете сказать, что существует много разных исполнителей, и способны отличить их друг от друга, но не знаете, кто они такие и как их зовут. Это называется самостоятельным подписанием (в данном случае сделанным с помощью голосовых связок). Когда кто-то вам говорит об исполнителе и ассоциирует его с голосом, который вы слышали, это аналогично стороннему подписанию.

Один исполнитель может имитировать голос другого исполнителя, вводя в заблуждение или разыгрывая слушателей. Однако цифровую подпись эмулировать намного сложнее, поскольку для шифрования подписей применяются математические алгоритмы.

Шаблон для понимания цифровых подписей

Когда мы говорим, что кто-то подписал JAR-файл, этот JAR-файл становится уникально идентифицируемым и может различаться от другого набора JAR-файлов. Однако нет никакого способа идентифицировать его разработчика или компанию. Такой JAR-файл называется самостоятельно подписанным.

Для определения *источника* необходимо, чтобы сторонняя компания, которой коллекционер вина доверяет, сообщила вам, что красный цвет вина поставляет производитель Company1. После этого, если встречается вино красного цвета, известно, что оно произведено Company1. Аналогично получаются и JAR-файлы, подписанные сторонней организацией. Браузер будет сообщать, что загружаемый файл поступает от Company1 или устанавливаемое приложение произведено Company1.

Как создать цифровую подпись?

Цифровые подписи, которые следуют семантике, похожей на изложенную в предыдущих сценариях, формально реализованы через шифрование с помощью открытого и секретного ключей. Применяемый при этом алгоритм генерирует два числа, из которых одно предназначено для шифрования (секретный ключ), а другое — для расшифровки (открытый ключ). Эти ключи являются ассиметричными. Даже если кому-то известен открытый ключ, нет никакого способа зашифровать сообщение, которое могло бы расшифровываться с помощью открытого ключа. Это можно сделать только посредством подходящего секретного ключа.

Давайте применим идею открытого и секретного к примеру с вином. Винодел, который желает отличать вино по цифровым подписям в противоположность оттенкам, создает код (оттенок) для бочонка, используя секретный ключ. Поскольку для генерации кода (оттетка) применялся секретный ключ, расшифровать код можно только с помощью соответствующего открытого ключа.

Винодел затем спокойно записывает имя открытого ключа имя и зашифрованный код на бочонке или передает открытый ключ с курьером. Когда вы, как коллекционер вин, получаете открытый ключ и успешно расшифровываете код, вы знаете, что откры-

тый ключ корректен, а сообщение могло быть зашифровано только виноделом, который записал на бочонке открытый ключ. При таком сценарии, даже если кто-то другой, выдающий себя за винодела, скопирует открытый ключ настоящего винодела и запишет его на бочонке, он не сможет написать секретное сообщение, которое будет расшифровано открытым ключом.

В сущности, открытый ключ становится подписью винодела. Даже если бы кто-то другой заявил права на открытый ключ, он не был бы в состоянии произвести сообщения, которые могли быть расшифрованы с помощью открытого ключа.

Благодаря этому сравнению цифровых и реальных подписей, была проведена параллель, которая помогла усвоить понятие цифровых подписей. В главе 10 рассматривались JDK-команды `keytool` и `jarsigner`, используемые в процессе подписания.

Последствия процесса подписания

Теперь становится понятно, что иметь две разных подписи для одного имени пакета не допускается. Подписи иногда называют сертификатами PKI (public key infrastructure — инфраструктура открытых ключей). Выражаясь более строго, сертификат PKI можно использовать для подписания пакета, JAR-файла, DLL-библиотеки или приложения.

Сертификат PKI связан с именем пакета, гарантируя, что два разработчика не смогут поставить пакеты с полностью совпадающими именами. Тем не менее, один и тот же сертификат можно использовать для подписания любого количества пакетов. Другими словами, один сертификат PKI поддерживает множество пакетов. Это отношение “один ко многим”. Однако один пакет имеет одну и только одну подпись, произведенную посредством сертификата PKI. Разработчик защищает секретный ключ сертификата с помощью пароля.

Эти факты важны не только для новых выпусков того же самого пакета, но также и для разделения данных между пакетами, когда эти пакеты имеют одну и ту же подпись.

Разделение данных между пакетами

В предыдущих главах было указано, что каждый пакет выполняется в собственном процессе. Все компоненты, установленные или созданные через этот пакет, принадлежат пользователю, идентификатор которого назначен пакету. Кроме того, известно, что Android выделяет уникальный основанный на Linux идентификатор пользователя для запуска пакета. На рис. 12.1 было показано, как может выглядеть этот идентификатор.

Согласно документации по Android SDK, идентификатор пользователя назначается, когда приложение устанавливается на устройстве, и остается неизменным на время его существования на этом устройстве. Любым данным, сохраняемым приложением, будет назначен идентификатор пользователя этого приложения, и данные, как правило, не доступны другим пакетам. При создании нового файла с помощью `getSharedPreferences(String, int)`, `openFileOutput(String, int)` или `openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)` можно использовать флаги `MODE_WORLD_READABLE` и/или `MODE_WORLD_WRITEABLE`, чтобы позволить другим пакетам читать/записывать в этот файл. Когда эти флаги указаны, файлом по-прежнему владеет приложение, но за счет установки глобальных прав на чтение и/или запись любое другое приложение может видеть этот файл.

Если вы намерены построить набор совместно работающих приложений, которые зависят от общего набора данных, можно явно указать идентификатор пользователя, уникальный для вас и общий для существующих потребностей. Этот разделяемый иден-

тификатор пользователя также определен в файле манифеста, подобно определению имени пакета. В листинге 12.1 приведен пример.

Листинг 12.1. Объявление разделяемого идентификатора пользователя

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.androidbook.somepackage"
     sharedUserId="com.androidbook.mysharedusrid"
     ...
     >
     ...остальные узлы XML
</manifest>
```

Природа разделяемых идентификаторов пользователей

Несколько приложений могут указать один и тот же разделяемый идентификатор пользователя, если они разделяют ту же самую подпись (подписаны одинаковым сертификатом PKI). Наличие такого идентификатора позволяет множеству приложений совместно использовать данные и даже выполняться в одном и том же процессе. Во избежание дублирования разделяемых идентификаторов пользователей применяйте соглашение, подобное соглашению об именовании классов Java. Ниже перечислены некоторые примеры разделяемых идентификаторов пользователей из системы Android:

```
"android.uid.system"
"android.uid.phone"
```

На заметку! В группах новостей, связанных с Android, встречаются сообщения о том, что разделяемый идентификатор должен быть указан в виде низкоуровневой строки, а не строкового ресурса.

В качестве предостережения: если вы планируете применять разделяемые идентификаторы пользователей, рекомендуется использовать их с самого начала. В противном случае они не будут нормально работать в ситуациях, когда производится обновление приложения с неразделяемым идентификатором пользователя на приложение с разделяемым идентификатором. Одна из упоминаемых причин в том, что Android не будет запускать `chown` на старых ресурсах из-за того, что идентификатор пользователя изменился. Следовательно, настоятельно рекомендуется:

- применять разделяемый идентификатор пользователя с самого начала, когда он необходим;
- не изменять идентификатор пользователя во время его применения.

Шаблон кода для разделения данных

В этом разделе будут исследованы возможности, доступные в ситуации, когда два приложения должны разделять ресурсы и данные. Как уже известно, во время выполнения ресурсами и данными каждого пакета владеет контекст этого пакета, который их же и защищает. Поэтому не должно вызывать удивления, что для разделения ресурсов и данных пакета необходим доступ к его контексту.

Чтобы помочь в этом, Android предоставляет API-интерфейс по имени `createPackageContext()`. Его можно использовать на любом существующем объекте контекста (таком как действие) и получить ссылку на целевой контекст для дальнейше-

го взаимодействия. В листинге 12.2 приведен пример (в этом примере демонстрируется только использование; он не предназначен для компилирования).

Листинг 12.2. Использование API-интерфейса `createPackageContext()`

```
// Идентификация пакета для использования
String targetPackageName="com.androidbook.samplepackage1";

// Соответствующий флаг контекста
int flag=Context.CONTEXT_RESTRICTED;

// Получение целевого контекста через одно из действий
Activity myContext = ...;
Context targetContext =
          myContext.createPackageContext(targetPackageName, flag);

// Использование контекста для разрешения путей к файлам
Resources res = targetContext.getResources();
File path = targetContext.getFilesDir();
```

Обратите внимание на то, как получается ссылка на контекст пакета с заданным именем, таким как `com.androidbook.samplepackage1`. Объект `targetContext` в листинге 12.2 идентичен контексту, переданному целевому приложению при его запуске. Наличие префикса `create` в имени метода говорит о том, что каждый его вызов возвращает новый объект контекста. Однако в документации утверждается, что возвращаемый объект контекста является облегченным.

Этот API-интерфейс применяется независимо от того, имеется разделяемый идентификатор пользователя или нет. Если идентификатор пользователя разделяется, ничего делать не понадобится. Если идентификатор пользователя не разделяется, целевое приложение должно объявить свои ресурсы доступными внешним пользователям. Метод `createPackageContext()` использует один из трех флагов.

- Флаг `CONTEXT_INCLUDE_CODE` приводит к тому, что Android разрешает загружать код целевого приложения в текущий процесс. Этот код затем будет запущен от вашего имени. Такое удастся, только если оба пакета имеют одинаковые подписи и разделяемый идентификатор пользователя. Если разделяемые идентификаторы пользователей не совпадают, применение этого флага приводит к генерации исключения безопасности.
- Флаг `CONTEXT_RESTRICTED` позволяет иметь доступ к путям ресурсов без экстремального случая запроса на загрузку кода.
- Флаг `CONTEXT_IGNORE_SECURITY` приводит к тому, что сертификаты игнорируются, а код загружается, но будет выполняться от имени вашего идентификатора пользователя. В документации дано серьезное предупреждение на случай применения этого флага.

Теперь вам известно, как с помощью пакетов, подписей и разделяемых идентификаторов пользователей управлять доступом к тому, чем приложения владеют и создают.

Библиотечные проекты

Поскольку речь идет о разделении кода и ресурсов, необходимо выяснить, чем может в этом помочь идея т.н. “библиотечных” проектов? Первым делом, понадобится понять, что собой представляют библиотечные проекты, как их создавать и каким образом использовать.

Что собой представляет библиотечный проект?

Начиная с подключаемого модуля ADT 0.9.7 для Eclipse, в Android поддерживается идея библиотечных проектов. Библиотечный проект — это коллекция Java-кода и ресурсов, которая выглядит как обычный проект, но никогда не завершается созданием файла `.apk` только из самого себя. Вместо этого код и ресурсы библиотечного проекта становятся частью другого проекта и компилируются в его файл `.apk`.

Особенности библиотечных проектов

Ниже перечислены некоторые важные факты, касающиеся библиотечных проектов.

- Библиотечный проект может иметь собственное имя пакета.
- Библиотечный проект не компилируется в собственный файл `.apk`, а внедряется в файл `.apk` проекта, который использует его как зависимость.
- Библиотечный проект может использовать другие JAR-файлы.
- Библиотечный проект не может быть помещен в JAR-файл сам по себе.
- Подключаемый модуль ADT для Eclipse объединяет библиотечный проект с главным проектом и компилирует их вместе во время компиляции главного проекта.
- Библиотечный и главный проекты могут получать доступ к ресурсам библиотечного проекта через соответствующие файлы `R.java`.
- В библиотечном и главном проектах могут существовать одинаковые идентификаторы ресурсов. В таком случае преимущество отдается идентификаторам ресурсов из главного проекта.
- Для различения идентификаторов ресурсов между двумя проектами можно использовать префиксы, такие как `lib_` для ресурсов библиотечного проекта.
- Главный проект может ссылаться на произвольное количество библиотечных проектов.
- Библиотечным проектам можно назначить приоритеты для обозначения более важных ресурсов.
- Библиотечные компоненты, такие как действия, необходимо определить в файле манифеста главного проекта. После этого имя компонента из библиотечного пакета должно полностью квалифицироваться с помощью имени библиотечного пакета.
- Определять компоненты в файле манифеста библиотечного проекта не обязательно, однако полезно для быстрого выяснения, какие компоненты библиотечный проект поддерживает.
- Создание библиотечного проекта начинается с создания обычного проекта Android и отметки флажка Is Library (Является библиотечным) в окне свойств проекта.
- Установить зависимые библиотечные проекты для главного проекта можно также в окне свойство проекта.
- Определенный библиотечный проект может быть включен во множество главных проектов.
- Средство библиотечных проектов доступно в ADT 0.9.7, инструментах SDK версии r6 и выше, Android 2.1 и последующих версиях.

- В этом выпуске библиотечные проекты не могут ссылаться друг на друга, хотя, по всей видимости, это можно будет делать в будущих выпусках.
- Библиотечные проекты не поддерживают файлы AIDL.
- Библиотечные проекты не поддерживают каталоги разделяемых компонентов.

Для исследования библиотечных проектов создадим один такой проект, а также главный проект. Ниже перечислены шаги, которые понадобится выполнить.

1. Создать простое действие в библиотечном проекте.
2. Создать меню для действия из шага 1, определив некоторые ресурсы меню.
3. Создать действие главного проекта, которое использует библиотечный проект как зависимость.
4. Создать действие в главном проекте из шага 3.
5. Создать меню для главного действия из шага 4.
6. Предусмотреть в главном действии пункт меню, который вызывает действие из библиотечного проекта.

После создания этих проектов становится доступным действие из главного проекта (действие из шага 4, показанное на рис. 12.2).

После щелчка на пункте меню invoke lib (вызов библиотеки) из действия главного проекта отобразится действие, обслуживаемое библиотечным проектом, как показано на рис. 12.3.

Пункты меню в действии библиотечного проекта поступает из ресурсов этого проекта. Щелчки на этих пунктах меню приводят к выводу на экран сообщения о том, что на конкретном пункте меню совершен щелчок. Начнем выполнение примера с создания библиотечного проекта.

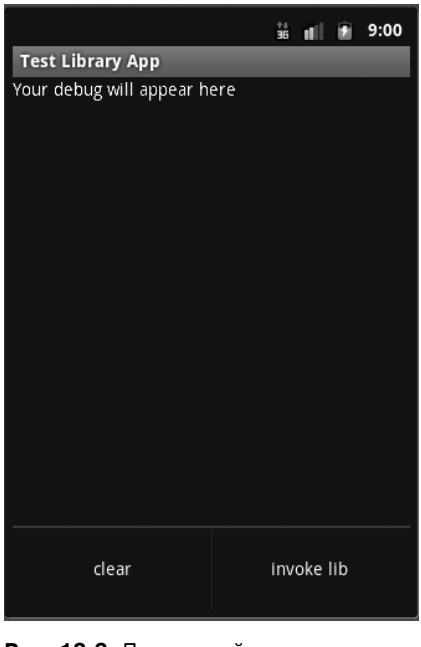

Рис. 12.2. Пример действия с меню в главном проекте

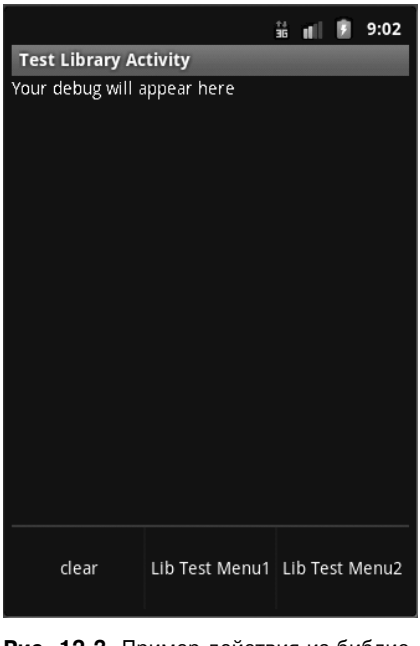

Рис. 12.3. Пример действия из библиотечного проекта

Создание библиотечного проекта

Пример библиотечного проекта будет состоять из следующих файлов:

- `TestLibActivity.java` (листинг 12.3)
- `layout/lib_main.xml` (листинг 12.4)
- `menu/lib_main_menu.xml` (листинг 12.5)
- `AndroidManifest.xml` (листинг 12.6)

Этих файлов должно быть достаточно для создания собственного проекта Android; их содержимое приведено в последующих листингах.

На заметку! В конце главы приводится URL-адрес, по которому можно загрузить проекты этой главы и импортировать их непосредственно в Eclipse.

Листинг 12.3. Действие для примера библиотечного проекта: `TestLibActivity.java`

```
package com.androidbook.library.testlibrary;
// ...базовые операторы import
// нажмите <CTRL+SHIFT+O>, чтобы Eclipse
// сгенерировал необходимые операторы import
public class TestLibActivity extends Activity
{
    public static final String tag="TestLibActivity";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lib_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        MenuInflater inflater = getMenuInflater(); // из действия
        inflater.inflate(R.menu.lib_main_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        appendMenuItemText(item);
        if (item.getItemId() == R.id.menu_clear){
            this.emptyText();
            return true;
        }
        return true;
    }

    private TextView getTextView(){
        return (TextView)this.findViewById(R.id.text1);
    }

    public void appendText(String abc){
        TextView tv = getTextView();
        tv.setText(tv.getText() + "\n" + abc);
    }
```

```
    private void appendMenuItemText(MenuItem menuItem){
        String title = menuItem.getTitle().toString();
        TextView tv = getTextView();
        tv.setText(tv.getText() + "\n" + title);
    }

    private void emptyText(){
        TextView tv = getTextView();
        tv.setText("");
    }
}
```

В листинге 12.4 приведено содержимое файла компоновки для этого действия: здесь имеется только простое текстовое представления для вывода названия пункта меню, на котором совершен щелчок.

Листинг 12.4. Файл компоновки для примера библиотечного проекта: `layout/lib_main.xml`

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

<TextView
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Your debug will appear here "
    />
</LinearLayout>
```

В листинге 12.5 представлено содержимое файла меню, используемого в действии библиотечного проекта (см. рис. 12.3).

Листинг 12.5. Файл меню для примера библиотечного проекта: `menu/lib_main_menu.xml`

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Эта группа использует категорию по умолчанию. -->
    <group android:id="@+id/menuGroup_Main">
        <item android:id="@+id/menu_clear"
              android:title="clear" />
        <item android:id="@+id/menu_testlib_1"
              android:title="Lib Test Menu1" />
        <item android:id="@+id/menu_testlib_2"
              android:title="Lib Test Menu2" />
    </group>
</menu>
```

В листинге 12.6 приведено содержимое файла манифеста для библиотечного проекта.

Листинг 12.6. Файл манифеста для примера библиотечного проекта: `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.androidbook.library.testlibrary"
          android:versionCode="1"
          android:versionName="1.0.0">
   <uses-sdk android:minSdkVersion="3" />
   <application android:icon="@drawable/icon"
                android:label="Test Library Project">
      <activity android:name=".TestLibActivity"
                android:label="Test Library Activity">
      </activity>
   </application>
</manifest>
```

Как было указано в разделе “Особенности библиотечных проектов”, определение действия в файле манифеста для библиотечного проекта добавляется только в целях документирования и во время выполнения необязательно.

После того, как все файлы готовы, можно создать обычный проект Android. Затем щелкните правой кнопкой мыши на имени проекта и выберите в контекстном меню пункт Properties (Свойства), чтобы открыть диалоговое окно свойств библиотечного проекта. Это окно показано на рис. 12.4. (Доступные целевые платформы сборки могут отличаться в зависимости от используемой версии Android SDK.) В этом диалоговом окне отметьте флажок Is Library (Является библиотечным), чтобы сделать этот проект библиотечным.

На этом создание библиотечного проекта завершено. Теперь давайте посмотрим, как создать проект приложения, в котором используется готовый библиотечный проект.

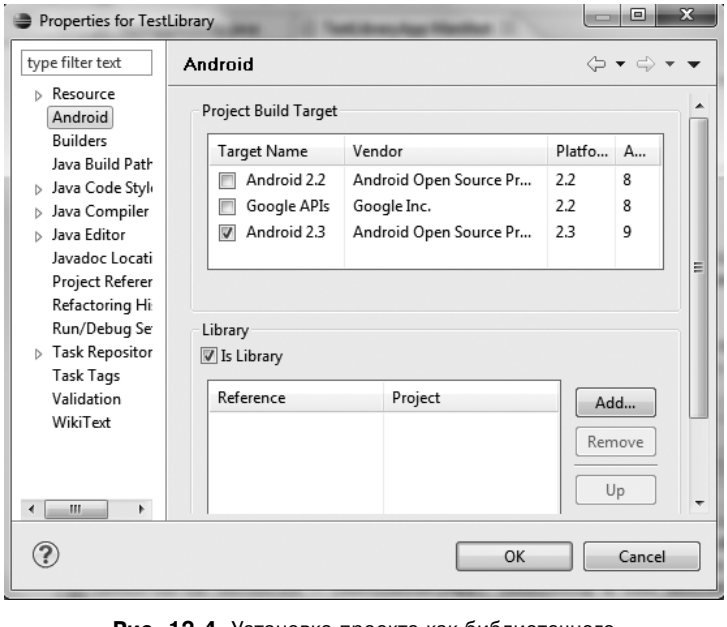

Рис. 12.4. Установка проекта как библиотечного

Создание проекта Android, который использует библиотечный проект

Мы воспользуемся похожим набором файлов для создания проекта приложения и затем установим предыдущий библиотечный проект в качестве зависимости. Ниже перечислены файлы, необходимые для главного проекта:

- `TestAppActivity.java` (листинг 12.7)
- `layout/main.xml` (листинг 12.8)
- `menu/main_menu.xml` (листинг 12.9)
- `AndroidManifest.xml` (листинг 12.10)

В листинге 12.7 приведено содержимое `TestAppActivity.java`.

Листинг 12.7. Код действия для главного проекта: `TestAppActivity.java`

```
package com.androidbook.library.testlibraryapp;
import com.androidbook.library.testlibrary.*;
//...другие операторы import

public class TestAppActivity extends Activity
{
    public static final String tag="TestAppActivity";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        MenuInflater inflater = getMenuInflater(); // из действия
        inflater.inflate(R.menu.main_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        appendMenuItemText(item);
        if (item.getItemId() == R.id.menu_clear)
        {
            this.emptyText();
            return true;
        }
        if (item.getItemId() == R.id.menu_library_activity){
            this.invokeLibActivity(item.getItemId());
            return true;
        }
        return true;
    }

    private void invokeLibActivity(int mid)
    {
        Intent intent = new Intent(this,TestLibActivity.class);
        intent.putExtra("com.ai.menuid", mid);
        startActivity(intent);
    }
```

```
    private TextView getTextView(){
        return (TextView)this.findViewById(R.id.text1);
    }
    public void appendText(String abc){
        TextView tv = getTextView();
        tv.setText(tv.getText() + "\n" + abc);
    }
    private void appendMenuItemText(MenuItem menuItem){
        String title = menuItem.getTitle().toString();
        TextView tv = getTextView();
        tv.setText(tv.getText() + "\n" + title);
    }
    private void emptyText(){
        TextView tv = getTextView();
        tv.setText("");
    }
}
```

Обратите внимание, что после создания этого файла вы можете получить ошибку компиляции при ссылке на класс действия, находящийся в библиотечном проекте. Эта ошибка не исчезнет до тех пор, пока библиотечный проект не будет корректно указан как зависимость проекта приложения.

В листинге 12.8 приведен файл компоновки для поддержки действия.

Листинг 12.8. Файл компоновки для главного проекта: `layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Debug Text Will Appear here"
    />
</LinearLayout>
```

В Java-коде действия главного проекта (см. листинг 12.7) используется пункт меню по имени `R.id.menu_library_activity` для вызова `TestLibActivity`. Вот этот код:

```
    private void invokeLibActivity(int mid)
    {
        Intent intent = new Intent(this,TestLibActivity.class);
        // Передача идентификатора меню как Extra намерения,
        // если он нужен действию библиотечного проекта.
        intent.putExtra("com.androidbook.library.menuid", mid);
        startActivity(intent);
    }
```

Обратите внимание, что `TestLibActivity.class` используется, как если бы он был локальным классом, за исключением импорта Java-классов из библиотечного пакета:

```
import com.androidbook.library.testlibrary.*;
```

В листинге 12.9 приведено содержимое файла меню.

Листинг 12.9. Файл меню для главного проекта: `menu/main_menu.xml`

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <!-- Эта группа использует категорию по умолчанию. -->
   <group android:id="@+id/menuGroup_Main">
      <item android:id="@+id/menu_clear"
            android:title="clear" />
      <item android:id="@+id/menu_library_activity"
            android:title="invoke lib" />
   </group>
</menu>
```

И, наконец, содержимое файла манифеста показано в листинге 12.10.

Листинг 12.10. Файл манифеста для главного проекта: `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.androidbook.library.testlibraryapp"
     android:versionCode="1"
     android:versionName="1.0.0">
   <application android:icon="@drawable/icon" android:label="Test Library App">
      <activity android:name=".TestAppActivity"
                android:label="Test Library App">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=
         "com.androidbook.library.testlibrary.TestLibActivity"
                android:label="Test Library Activity"/>
   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

В файле манифеста для главного приложения взгляните, как определяется действие `TestLibActivity` из библиотечного проекта. В определении действия используется полностью квалифицированное имя пакета. Кроме того, имена пакетов для библиотечного проекта могут отличаться от главного проекта приложения.

После того, как проект Android готов, в диалоговом окне свойств проекта, показанном на рис. 12.5, можно указать, что главный проект зависит от библиотечного проекта, который был создан ранее.

Щелкните на кнопке Add (Добавить) и добавьте библиотеку в качестве ссылки. Ничего другого делать не понадобится.

После этого библиотечный проект отобразится как дополнительный узел в главном проекте приложения. Это можно видеть на рис. 12.6.

Обратите внимание на узел, помеченный [Android Library] ([Библиотека Android]), и файлы исходного кода Java, на которые производится ссылка. Взгляните на структуру этого узла. Его имя состоит из имени библиотечного проекта, символа подчеркивания (_) и имени соответствующего каталога исходного кода внутри библиотечного проекта. Такая схема позволяет иметь в библиотечном проекте любое количество каталогов исходного кода. Это основное отличие между ADT 0.9.8 и более новыми выпусками ADT.

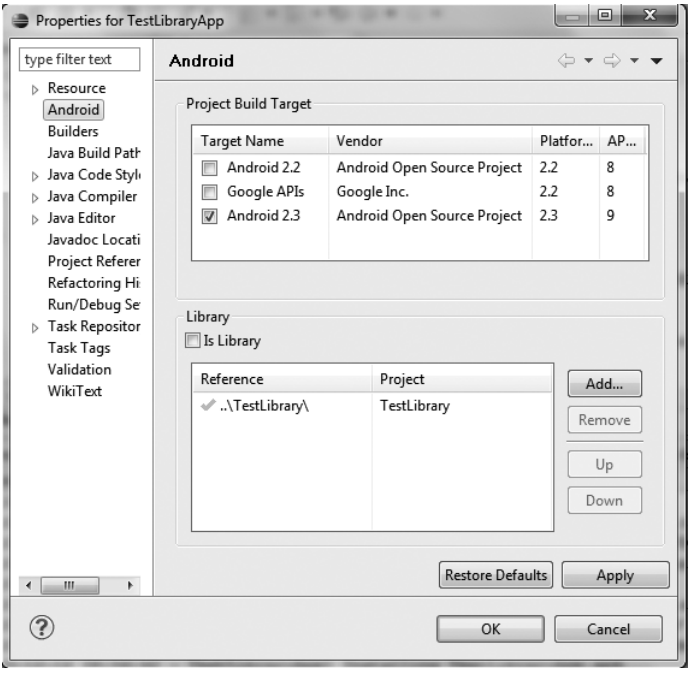

Рис. 12.5. Добавление зависимости от библиотечного проекта

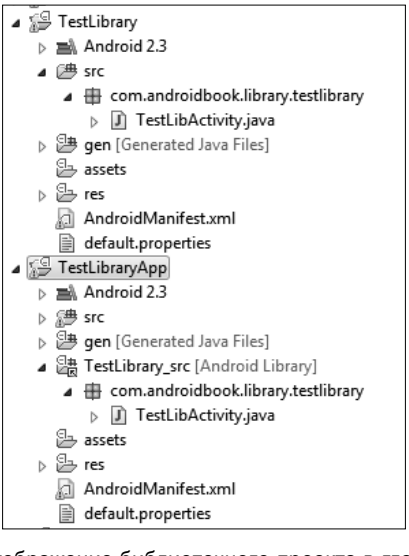

Рис. 12.6. Отображение библиотечного проекта в главном проекте

Изменяя файлы исходного кода, принадлежащие библиотечному проекту, в рамках проекта приложения, вы в действительности модифицируете их в самом библиотечном проекте. Иногда этот узел не виден. В таком случае может понадобиться перезапустить Eclipse.

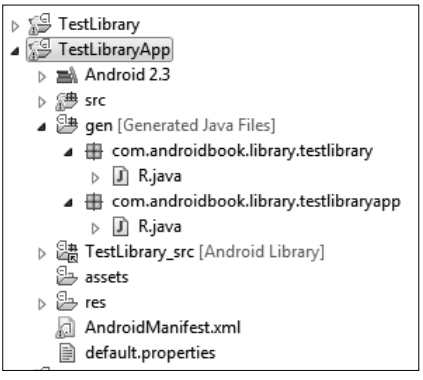

Рис. 12.7. Реплицированные ресурсы в `R.java`

Android делает кое-что интересное в отношении файлов `R.java`. Взгляните на рис. 12.7.

Сначала Android генерирует один файл `R.java` в библиотечном проекте для ресурсов, принадлежащих этому проекту. Затем Android генерирует один файл `R.java` для ресурсов главного проекта. Это вполне ожидаемо: два проекта — два файла `R.java`.

Однако интересно то, что Android создает идентификаторы ресурсов для библиотечных ресурсов также и в файле `R.java` главного приложения. Это значит, что программист может использовать синтаксис `R.id.` для идентификаторов файла `R.java`, принадлежащего главному приложению (следует отметить, что содержимое файла `R.java` генерируется автоматически, поэтому числа вроде `0x7f02000` в листинге 12.11 в вашем проекте могут быть другими).

Листинг 12.11. Повторно определенные разделяемые идентификаторы ресурсов в файле `R.java` главного проекта

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
        public static final int robot=0x7f020001;
    }
    public static final class id {
        public static final int menuGroup_Main=0x7f060001;
        public static final int menu_clear=0x7f060002;
        public static final int menu_library_activity=0x7f060005;
        public static final int menu_testlib_1=0x7f060003;
        public static final int menu_testlib_2=0x7f060004;
        public static final int text1=0x7f060000;
    }
    public static final class layout {
        public static final int lib_main=0x7f030000;
        public static final int main=0x7f030001;
    }
    public static final class menu {
        public static final int lib_main_menu=0x7f050000;
        public static final int main_menu=0x7f050001;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Как видите, ресурсы, идентифицируемые с префиксом `lib_`, теперь доступны также и в файле `R.java` главного проекта приложения. Это значит, что библиотечный проект будет иметь одну ресурсную константу для `lib_`, а главный проект — другую ресурсную константу для того же ресурса `lib_`. В Java-коде обоих проектов можно ссылаться

на этот ресурс, используя R.идентификатор. Значения констант могут совпадать, но идентификатор ресурса будет доступен в обоих пространствах имен Java: пространстве имен библиотечного пакета и пространстве имен главного проекта.

Вдобавок обратите внимание на имена меню: lib_main_menu и main_menu. Мы оказались бы в действительно сложной ситуации, если бы два ресурсных файла меню имели одинаковые имена, но содержали разные элементы меню. Суть заключается в том, что ресурсы собраны и доступны в одном месте для главного приложения. Особое внимание следует обратить на ресурсы, находящиеся на уровне файлов, такие как меню и компоновки, и на идентификаторы, сгенерированные для внутренних элементов этих ресурсных файлов.

Теперь, когда вы понимаете концепцию библиотечных проектов, приблизились ли вы к ответу на вопросы о разделении данных, которые были поставлены в начале?

Как видите, библиотечные проекты являются конструкциями времени компиляции. Очевидно, что ресурсы, которые принадлежат библиотеке, объединяются с главным проектом. Никаких проблем с их разделением во время выполнения не возникает, поскольку имеется всего лишь один файл пакета с именем главного пакета. Часто упоминают о возможности разработки бесплатной и платной версий приложения, совместно используя одну библиотеку обеими версиями.

Ссылки

Ниже перечислены некоторые полезные ссылки, которые помогут закрепить знания, полученные в настоящей главе.

- http://developer.android.com/guide/publishing/app-signing.html. По этому адресу доступна полезная информация по подписанию файлов .apk.
- http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html. Этот сайт предлагает документацию по keytool, jarsigner и самому процессу подписания.
- http://www.androidbook.com/item/3493. Авторские заметки, включая концептуальную модель, по поводу того, что означает подписание JAR-файла.
- http://www.androidbook.com/item/3279. Здесь собраны результаты наших исследований пакетов Android. Вы узнаете, как подписывать файлы .apk, увидите дополнительные ссылки на сведения о разделении данных между пакетами, о разделяемых идентификаторах пользователей и об установке и удалении пакетов.
- http://developer.android.com/guide/developing/eclipseadt.html#libraryProject. Эта статья поможет глубже понять концепцию библиотечных проектов.
- http://www.androidbook.com/projects. Обращайтесь сюда за списком загружаемых проектов, относящихся к этой книге. Для этой главы предназначен файл по имени ProAndroid3_Ch12_TestAndroidLibraries.zip. Он содержит оба рассмотренных проекта в отдельных корневых каталогах, так что вы сможете импортировать их в ADT.

Резюме

В этой главе рассматривалась работа с пакетами и процессами, разделение кода и данных между пакетами и создание библиотечных проектов Android. Было показано, что подписи играют важную роль в выяснении владельцев пакетов. Глава заложила фундамент для материалов следующей главы, в которой речь пойдет о компонентах, которые находятся в процессе пакета и (в основном) выполняются в главном потоке процесса. Там будет показано, как оптимально снабдить главный поток обработчиками и подчиненными потоками, чтобы приложение Android функционировало максимально эффективно.