Посвящается Анне, Фабиану и Иде

Содержание

Предисловие 13

Глава 1. Компоненты ОС Android и необходимость параллельных вычислений 19

Стек программной среды ОС Android 19
Архитектура приложения 21
Приложение 21
Компоненты 21
Activity 22
Service 24
ContentProvider 24
BroadcastReceiver 25
Выполнение приложения 25
Процесс Linux 25
Жизненный цикл 26
Запуск приложения 26
Завершение приложения 27
Структурирование приложений для улучшения производительности 29
Создание отзывчивых приложений с помощью потоков 30
Резюме 32

Часть I. Основы 33

Глава 2. Многопоточность в Java 34

Основы использования потоков 34
Выполнение 35
Приложение с одним потоком 37
Многопоточное приложение 37
Увеличение потребления ресурсов 38
Повышенная сложность 38
Нарушение целостности данных 38
Безопасное состояние потока 40
Внутренняя блокировка и монитор Java 41
Синхронизация доступа к совместно используемым ресурсам 43
Использование внутренней блокировки 43
Явное использование механизмов блокировки 45
Пример: потребитель и производитель 46
Стратегии выполнения задачи 48
Проектирование параллельного выполнения 49
Резюме 50

Глава 3. Потоки в ОС Android 51
Потоки приложения ОС Android 51
UI-поток 51
Связующие потоки 52
Фоновые потоки 53
Процесс Linux и потоки 53
Планирование 57
Приоритет 58
Управляющие группы 59
Резюме 61

Глава 4. Взаимодействие потоков 62
Программные каналы 62
Основы использования программного канала 64
Пример: обработка текста в рабочем потоке 66
Совместно используемая память 68
Механизм сигналов 69
Блокирующая очередь BlockingQueue 71
Передача сообщений в ОС Android 73
Пример: простая передача сообщений 74
Классы, используемые при реализации механизма передачи сообщений 77
Класс MessageQueue 77
Интерфейс MessageQueue.IdleHandler 79
Пример: использование интерфейса IdleHandler для завершения ненужного потока 80
Класс Message 82
Состояние «инициализировано» 84
Состояние «ожидание» 85
Состояние «передано» 85
Состояние «готово к повторному использованию» 85
Класс Looper 86
Завершение работы объекта Looper 87
Объект Looper в UI-потоке 88
Класс Handler 88
Создание и настройка 89
Создание сообщения 90
Вставка сообщения в очередь 90
Пример: передача сообщений в двух направлениях 92
Обработка сообщений 95
Удаление сообщений из очереди 97
Наблюдение за очередью сообщений 99
Получение текущего состояния очереди сообщений 99

Отслеживание обработки очереди сообщений 102
Взаимодействие с UI-потоком 103
Резюме 104

Глава 5. Взаимодействие между процессами 105

Механизм вызова удалённых процедур в ОС Android 105
Объект Binder 106
Язык AIDL 108
Синхронные вызовы удалённых процедур 110
Асинхронные вызовы удалённых процедур 113
Передача сообщений с использованием объекта Binder 115
Однонаправленное взаимодействие 117
Взаимодействие в двух направлениях 119
Резюме 120

Глава 6. Управление памятью 121

Сборка мусора 121
Утечки памяти, связанные с использованием потоков 123
Выполнение потока 125
Внутренние классы 126
Статические внутренние классы 127
Рассогласование жизненных циклов 128
Взаимодействие потоков 131
Отправка сообщения с данными 132
Передача сообщения с задачей 133
Устранение утечек памяти 134
Использование статических внутренних классов 135
Использование слабых ссылок 135
Остановка рабочего потока 136
Переключение рабочих потоков 136
Очистка очереди сообщений 136
Резюме 137

Часть II. Механизмы асинхронного выполнения 138

Глава 7. Управление жизненным циклом простого потока 139

Основы использования потоков 139
Жизненный цикл 139
Прерывания 141
Неперехватываемые исключения 143
Управление потоком 145
Определение и запуск потока 145
Анонимный внутренний класс 145

Общедоступный поток 146
Определение потока как статического внутреннего класса 146
Обзор возможных вариантов выбора определения потока 147
Сохранение потока в рабочем состоянии 147
Сохранение потока в рабочем состоянии средствами класса Activity 148
Сохранение потока в рабочем состоянии средствами класса Fragment 151
Резюме 153

Глава 8. HandlerThread: механизм очереди сообщений высокого уровня 155

Основы использования HandlerThread 155
Жизненный цикл HandlerThread 157
Случаи использования 159
Повторяющееся выполнение задачи 159
Связанные задачи 160
Пример: обеспечение надёжности данных с помощью SharedPreferences 160
Объединение задач в цепочку 163
Пример: сетевые вызовы в цепочке задач 163
Вставка задач по условию 166
Резюме 167

Глава 9. Управление выполнением потока средствами фреймворка Executor 168

Executor 169
Пулы потоков 171
Предопределённые пулы потоков 172
Пулы потоков, определяемые разработчиком 173
Конфигурация ThreadPoolExecutor 173
Проектирование пула потоков 175
Определение размера 175
Динамические потоки в пуле 177
Ограниченная или неограниченная очередь задач 177
Конфигурация потока 178
Расширение возможностей ThreadPoolExecutor 179
Жизненный цикл 180
Корректное завершение работы пула потоков 181
Варианты использования пула потоков и возникающие при этом сложности 183
Предпочтение отдаётся созданию потока, а не организации очереди 183

Обработка предварительно подготовленных очередей задач.....183
Опасная ситуация при нулевом количестве базовых потоков в пуле.....184
Управление задачами.....184
Представление задачи.....185
Добавление задач.....186
Заявление отдельной задачи.....187
Метод invokeAll.....188
Метод InvokeAny.....190
Отвергнутые задачи.....191
ExecutorCompletionService.....191
Резюме.....194

Глава 10. Связывание фоновой задачи с UI-потоком с помощью AsyncTask.....196

Основы использования класса AsyncTask.....196
Создание и начало работы.....199
Отмена.....200
Состояния.....202
Пример: ограничение режима выполнения AsyncTask только одной задачей в любой момент времени.....203
Реализация AsyncTask.....203
Пример: загрузка изображений.....204
Выполнение задачи в фоновом режиме.....207
Глобальная среда выполнения в приложении.....209
Выполнение в разных версиях платформы.....211
Настраиваемое выполнение.....213
Пример: неглобальное последовательное выполнение.....213
Альтернативы AsyncTask.....214
Случаи излишне упрощённой реализации AsyncTask.....215
Фоновые задачи, для которых требуется объект Looper.....216
Локальная служба.....216
Использование метода execute(Runnable).....216
Резюме.....217

Глава 11. Службы.....218

Причины использования служб для асинхронного выполнения.....218
Локальные, удалённые и глобальные службы.....220
Создание и выполнение.....222
Жизненный цикл.....223
Запускаемая служба.....226
Реализация метода onStartCommand.....226
Повторный запуск.....227
Служба, управляемая пользователем.....230

Пример: соединение по протоколу Bluetooth....230
Служба, управляемая задачей....234
Пример: параллельная загрузка....234
Подключаемая служба....237
Локальное подключение....239
Выбор механизма асинхронного выполнения....242
Резюме....243

Глава 12. Класс IntentService....244

Основы использования IntentService....244
Эффективные способы использования IntentService....246
Задачи, выполнение которых должно быть последовательным....246
Пример: взаимодействие с веб-службой....246
Асинхронное выполнение в BroadcastReceiver....249
Пример: периодически выполняемые длительные операции....250
Сравнение IntentService и Service....252
Резюме....253

Глава 13. Доступ к провайдерам контента с помощью AsyncQueryHandler....254

Краткий обзор основ использования провайдеров контента....254
Настройка ContentProvider для обработки в фоновом режиме....256
Использование AsyncQueryHandler....258
Пример: список контактов с раскрывающимися элементами....260
Как работает AsyncQueryHandler....263
Ограничения....264
Резюме....265

Глава 14. Автоматическое выполнение в фоновом режиме с помощью загрузчиков Loader....266

Фреймворк Loader....268
Класс LoaderManager....268
Сравнение методов initLoader() и restartLoader()....270
Интерфейс LoaderCallbacks....272
Класс AsyncTaskLoader....274
Надёжная загрузка данных с помощью CursorLoader....275
Использование CursorLoader....275
Пример: список контактов....276
Добавление поддержки CRUD....277
Пример: использование CursorLoader вместе с обработчиком AsyncQueryHandler....278
Реализация специализированных загрузчиков....281
Жизненный цикл загрузчика....282
Фоновый режим загрузки....283

Пример: простой специализированный загрузчик....284
Управление контентом....286
Доставка кэшированных результатов....287
Пример: специализированный загрузчик файлов....288
Работа с несколькими загрузчиками....291
Резюме....292

Глава 15. Подведение итогов: выбор механизма асинхронного выполнения....294

Сохраняйте простоту....295
Управление потоками и ресурсами....296
Организация обмена сообщениями для улучшения отзывчивости....297
Как избежать неожиданного и нежелательного завершения задачи....298
Простой доступ к провайдерам контента....299

Список литературы....301

Предметный указатель....304

Предисловие

Книга «Эффективное использование потоков в операционной системе Android» демонстрирует методы проектирования высококачественных и надёжных в эксплуатации многопоточных приложений для операционной системы Android. В книге рассматриваются асинхронные механизмы, доступные в программной среде Android SDK, и соответствующие приемы создания быстрых, эффективных и правильно структурированных приложений.

Не вызывает сомнений, что без многопоточности немыслимо создание действительно полезных и удобных приложений, но ее применение влечет увеличение сложности приложений и вероятности появления ошибок во время выполнения. Одна из причин этой проблемы – трудности, непосредственно связанные с самим механизмом одновременной работы нескольких потоков, и другая причина – нерациональное использование возможностей платформы Android.

Цель этой книги – научить разработчиков приложений правильно выбирать требуемый асинхронный механизм, основываясь на полном понимании всех его достоинств и недостатков. При использовании правильно выбранного асинхронного механизма там, где он наиболее уместен, изрядная доля сложности перекладывается на программную платформу, при этом исходный код приложения становится более простым для сопровождения и поддержки, а вероятность ошибок в нём снижается. Как правило, асинхронное выполнение не должно повышать сложность исходного кода сверх реальной необходимости, и этого можно добиться посредством разумного выбора асинхронного механизма из комплекта, предлагаемого в программной среде Android.

Несмотря на то что любой асинхронный механизм высокого уровня уже с первого взгляда может показаться весьма удобным для практического применения, постоянно требуется не бездумное использование, но глубокое его понимание, иначе в приложении будут возникать труднообъяснимые ошибки времени выполнения, снижение производительности или утечки памяти. Именно поэтому в книге содержатся не только практические рекомендации и примеры, но и подробные объяснения лежащих в их основе механизмов асинхронного выполнения.

Для кого предназначена эта книга

Эта книга предназначена для программистов на языке Java, уже знакомых с общими принципами программирования для платформы

Android. Здесь представлены методики, образующие прочную основу для написания надёжных и эффективных приложений с использованием стандартных библиотек ОС Android.

Краткое содержание книги

Книга состоит из двух частей. В части I описываются основы многопоточного программирования в ОС Android, то есть на языке Java, в Linux, с применением различных обработчиков, и их влияние на разработку приложений. В части II основное внимание уделено практическому применению многопоточности с более глубоким исследованием асинхронных механизмов, предоставляемых в распоряжение любому приложению.

В части I рассказывается, как программы на Java работают с потоками. Программистам для ОС Android иногда приходится напрямую использовать библиотеки Java, поэтому понимание их возможностей важно для правильного применения конструкций более высокого уровня, описываемых в части II.

Глава 1 описывает устройство среды выполнения ОС Android и влияние различных компонентов типичного Android-приложения на использование многопоточности и параллельных вычислений.

Глава 2 рассматривает основы параллельного выполнения задач в языке Java.

Глава 3 обсуждает функционирование потоков в ОС Android и особенности выполнения прикладных потоков в Linux, в том числе такие важные темы, как планирование и управление группами потоков, а также их влияние на время ответной реакции приложения.

Глава 4 рассматривает основные механизмы обмена данными между потоками, такие как совместно используемая (разделяемая) память, сигналы и широко применяемые сообщения ОС Android.

Глава 5 показывает, как ОС Android дополняет механизмы взаимодействия процессов, предоставляемые ядром Linux, такими механизмами, как вызов удаленных процедур (RPC) и обмен сообщениями.

Глава 6 объясняет, как предотвращать утечки памяти в приложениях, которые могут приводить к существенному ухудшению работы системы в целом, из-за чего пользователи, вероятнее всего, будут просто избавляться от подобных приложений.

В части II рассматриваются библиотеки и программные конструкции более высокого уровня, которые делают программирование с использованием потоков более безопасным и простым.

Глава 7 описывает самую простую программную конструкцию поддержки асинхронного выполнения – `java.lang.Thread`, а также обработку различных проблем и нестандартных ситуаций, которые могут возникать при её использовании.

Глава 8 демонстрирует удобный способ последовательного запуска задач в фоновом режиме.

Глава 9 предлагает вниманию читателя методики планирования, обработки ошибок и некоторые другие аспекты использования потоков, такие как организация пулов потоков.

Глава 10 рассматривает `AsyncTask`, один из наиболее популярных механизмов асинхронного выполнения, а также правильное его применение и пути обхода потенциальных сложностей.

Глава 11 описывает важный компонент `Service`, особенно полезный для обеспечения сразу нескольких приложений одними и теми же функциональными возможностями или для поддержки приложения в рабочем состоянии во время выполнения в фоновом режиме.

Глава 12 продолжает последнюю тему предыдущей главы описанием эффективной методики запуска новых потоков выполнения из основного потока обслуживания графического пользовательского интерфейса.

Глава 13 посвящена механизму высокого уровня, упрощающему быстрый асинхронный доступ к компонентам типа `ContentProvider`.

Глава 14 объясняет, как обновлять компоненты графического пользовательского интерфейса с помощью загрузчиков (loaders) при асинхронной передаче новых данных вне зависимости от времени изменения их содержимого.

Глава 15 – итоговое резюме по всем методикам, описанным в книге, и обсуждение выбора приемов, наиболее подходящих для конкретного приложения. Здесь предлагаются правила и рекомендации, помогающие разработчику сделать правильный выбор.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения:

Курсив используется для выделения важных положений, новых терминов, имён команд и утилит, а также имён файлов и каталогов.

`Моноширинный шрифт` используется для обозначения имён переменных, функций, типов, объектов и других программных конструкций и элементов исходного кода.

Моноширинный курсив используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены фактическими значениями.


Так обозначается совет, указание или примечание общего характера.


Эта пиктограмма предупреждает о неочевидных и обычно скрытых от неопытного разработчика «подводных камнях» и «ловушках», которых следует избегать.

Примеры исходного кода

Дополнительные материалы (примеры исходного кода, учебные задания и т. п.) можно получить по ссылке **https://github.com/andersgoransson/eatbookexamples**.

Эта книга написана для того, чтобы помочь программистам в работе. Вообще говоря, вы можете использовать код из данной книги в своих программах и в документации. Если вы копируете для собственных нужд фрагмент исходного кода незначительного размера, то нет необходимости обращаться к автору и издателям для получения разрешения на это. Например, при включении в свою программу нескольких небольших фрагментов кода из книги вам не потребуется какое-либо специальное разрешение. Но для продажи или распространения CD-диска с примерами из книг издательства O'Reilly необходимо будет получить официальное разрешение на подобные действия. При ответах на вопросы можно цитировать текст данной книги и приводить примеры кода из неё без дополнительных условий. При включении крупных фрагментов исходного кода из книги в документацию собственного программного продукта также потребуется официальное разрешение.

Мы будем благодарны за добавление библиографической ссылки на источник при цитировании. Обычно ссылка состоит из названия книги, имени автора, наименования издательства и номера по ISBN-классификации.

Если у вас возникли сомнения в легальности использования примеров исходного кода без получения специального разрешения при условиях, описанных выше, то без колебаний обращайтесь по адресу электронной почты: **permissions@oreilly.com**.

Примеры исходного кода из книги доступны по адресу: **https://github.com/andersgoransson/eatbookexamples**.

Онлайн-сервис Safari® Books

Safari Books Online – это электронная библиотека, содержащая авторитетную информацию в виде книг и видеоматериалов, созданных ведущими специалистами в области технологий и бизнеса.

Профессионалы в различных областях техники, разработчики программного обеспечения, веб-дизайнеры, бизнесмены и творческие работники используют Safari Books Online как основной ресурс для научных исследований, решения профессиональных задач, обучения и подготовки к сертификационным испытаниям.

Библиотека Safari Books Online предлагает широкий выбор продуктов и тарифов для организаций, государственных учреждений и частных лиц. Подписчики получают доступ к поисковой базе данных, содержащей информацию о тысячах книг, видеоматериалов и ещё не опубликованных рукописей от таких известных издательств, как O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, и многих других. Более подробную информацию о Safari Books Online можно получить на сайте **https://www.safaribooksonline.com/**.

От издательства

Замечания, предложения и вопросы по этой книге отправляйте по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (звонок из США или Канады)
707-829-0515 (международный или местный звонок)
707-829-0104 (факс)

или по электронной почте:

bookquestions@oreilly.com

Список опечаток и ошибок, файлы с примерами и другую дополнительную информацию можно найти по адресу: **http://bit.ly/efficient-android-threading**.

Дополнительную информацию о книгах, учебных курсах, конференциях и новостях издательства O'Reilly вы найдете по адресу: **http://www.oreilly.com/**.

Ищите нас в Facebook: **http://facebook.com/oreilly**.

Следуйте за нами в Twitter: **http://twitter.com/oreillymedia**.

Смотрите нас в YouTube: **http://www.youtube.com/oreillymedia**.

Благодарности

Зачастую написание книги может показаться работой одного человека, но такие мысли приходят только поздней ночью, когда хочется как можно быстрее дописать последний абзац и хотя бы немного поспать. В действительности в процесс создания книги вовлечено множество людей, обеспечивающих саму возможность этого процесса.

Прежде всего я хотел бы поблагодарить Рэчел Румелиотис (Rachel Roumeliotis) из издательства O'Reilly за обращение ко мне с идеей написания книги и за помощь на начальном этапе этого процесса. Все сотрудники O'Reilly, с которыми я имел честь работать, демонстрировали высочайший профессионализм и готовность помочь на протяжении всего рабочего цикла создания этой книги, и именно благодаря им мне было проще уделять основное внимание творческому процессу. В особенности я благодарен редактору Энди Орэму (Andy Oram), чья роль в превращении книги из замысла в реальность является самой главной. Он терпеливо работал со мной над этим проектом, всегда скрупулёзно вычитывал мои рукописи и вносил неизменно точные и уместные правки и замечания.

Как и при создании сложного программного обеспечения, при написании книги никуда не деться от множества ошибок, появляющихся постоянно, поэтому каждая глава проходила этап поиска и исправления ошибок и «шлифовки» текста перед завершением работы над ней. Огромную помощь по устранению разнообразных огрехов, которые не так-то легко заметить в рукописях, оказали технические редакторы Джефф Сикс (Jeff Six) и Ян Дарвин (Ian Darwin), чьи многочисленные замечания касались и пропущенных запятых, и ошибок в программном коде, и предложений по улучшению структуры текста. Большое им спасибо!

Эту книгу невозможно было бы написать без поддержки семьи. Я благодарен моим родным за понимание необходимости моей работы в поздние ночные часы. Честно говоря, я не думаю, что они когда-нибудь прочтут эту книгу, но надеюсь, что она займёт достойное место на их книжной полке...

Глава 1

Компоненты ОС Android и необходимость параллельных вычислений

Прежде чем углубляться в многопоточный мир, для начала следует поближе познакомиться с платформой Android, с архитектурой её приложений и с особенностями выполнения приложений в этой среде. Данная глава закладывает основы, необходимые для дальнейшего обсуждения многопоточности в остальной части книги. Всё же следует отметить, что более полную информацию о платформе Android можно получить из официальной документации (**https://developer.android.com**) или из многочисленных книг по программированию в ОС Andriod, имеющихся в продаже.

Стек программной среды ОС Android

Приложения Android запускаются на верхнем уровне стека программной среды[1], в основе которого находится ядро Linux, уровнем выше – системные библиотеки на языках C/C++, и среда времени выполнения (runtime), отвечающая за выполнение кода приложения (рис. 1.1).

---

[1] Термин «стек программной среды» введён здесь по аналогии с термином «стек сетевых протоколов» во избежание путаницы, поскольку программным стеком уже достаточно давно называют область памяти, выделяемую программе для организации собственного внутреннего стека. – *Прим. перев.*

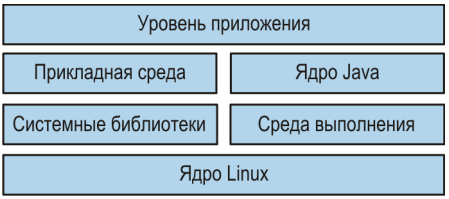

Рис. 1.1 ❖ Стек программной среды ОС Android

Главные составные части стека программной среды операционной системы (ОС) Android перечислены ниже:

- *Приложения* – приложения для платформы Android, написанные на языке Java, которые используют библиотеки языка Java и прикладной среды Android.
- *Ядро Java* – библиотеки ядра языка Java, используемые приложениями и прикладной средой. Это ядро не является полностью совместимым с реализацией Java SE или Java ME, а представляет собой некоторое подмножество более ранней, уже вышедшей из употребления реализации Apache Harmony, основанной на версии Java 5. Библиотеки ядра Java предоставляют основные механизмы работы с потоками: класс `java.lang.Thread` и пакет `java.util.concurrent`.
- *Прикладная среда* – классы платформы Android для работы с оконной системой (оконным интерфейсом), с компонентами графического пользовательского интерфейса, с ресурсами и т. п., то есть практически со всем, что необходимо для написания Android-приложения на языке Java. Прикладная среда определяет жизненные циклы компонентов Android и управляет этими циклами, а также взаимодействием компонентов и обменом данными между ними. Кроме того, прикладная среда определяет набор специализированных для платформы Android механизмов асинхронного выполнения, которыми приложения могут пользоваться для упрощения управления потоками: `HandlerThread`, `AsyncTask`, `IntentService`, `AsyncQueryHandler` и `Loaders`. Все эти механизмы будут подробно описаны в данной книге.
- *Системные библиотеки* – библиотеки на C/C++, которые работают непосредственно с графикой, системными ресурсами, базами данных, шрифтами, программным интерфейсом OpenGL и т. д. Обычно приложения на языке Java не взаимодействуют

напрямую с системными библиотеками, поскольку прикладная среда предоставляет функции-обёртки на Java, скрывающие обращения к этим библиотекам.

- *Среда выполнения* – надёжно изолированная и защищённая среда, выполняющая в виртуальной машине код Android-приложения, скомпилированный во внутренний байт-код. Каждое приложение работает в собственной среде выполнения – это либо Dalvik, либо ART (Android Runtime). Последняя была добавлена в версию KitKat (API level 19) как дополнительная. Ее можно активировать или отключить, но на момент написания данной книги Dalvik остаётся средой выполнения по умолчанию.
- *Ядро Linux* – заложенное в основу платформы ядро операционной системы Linux, позволяющее приложениям использовать аппаратные функции устройств: звуковых, сетевых, видео и т. п., а также управляющее процессами и потоками. Процесс порождается для каждого приложения, и каждый процесс предоставляет запускаемому приложению собственную среду выполнения. Внутри процесса код приложения может выполняться несколькими потоками. Ядро ОС распределяет доступное процессорное время, необходимое для выполнения кода, между процессами и потоками в них с помощью *механизма планирования* (*scheduling*).

Архитектура приложения

Основными элементами любого приложения являются объект `Application` и компоненты программной платформы Android: `Activity`, `Service`, `BroadcastReciever` и `ContentProvider`.

Приложение

В языке Java выполняющееся приложение представляет объект `android.app.Aplication`, экземпляр которого создаётся при запуске и уничтожается при завершении приложения (то есть время существования экземпляра класса `Application` совпадает со временем существования соответствующего процесса Linux). Когда процесс завершается и перезапускается, создаётся новый экземпляр класса `Application`.

Компоненты

Основными составными частями любого Android-приложения являются компоненты, управляемые средой выполнения: `Activity`, `Service`,

BroadcastReceiver и ContentProvider. Конфигурация и взаимодействие этих компонентов определяют поведение приложения. Перечисленные выше компоненты имеют разные области ответственности и жизненные циклы, но все они являются точками входа, посредством которых можно запускать приложения. После начала работы любой компонент может активировать другой компонент и т. д. на протяжении всего времени выполнения приложения. Запуск компонента в текущем либо в другом приложении производится с помощью объекта Intent[1]. Объект Intent определяет операции, запрашиваемые у принимающей стороны или получателя (receiver), например отправка электронной почты или фотографирование, и может передавать данные от отправителя (sender) получателю. Объект Intent может быть явным или неявным:

- *явный* объект Intent – определяет полностью классифицированное имя компонента, известное внутри приложения во время компиляции;
- *неявный* объект Intent – среда выполнения создаёт связь с компонентом, для которого определён набор характеристик в IntentFilter. Если Intent обнаруживает совпадение характеристик компонента с IntentFilter, такой компонент может быть активирован.

Компоненты и их жизненные циклы относятся к терминологии, специализированной для ОС Android, поэтому нельзя установить однозначное соответствие с объектами языка Java, лежащими в их основе. Объект Java может существовать дольше соответствующего ему компонента, а среда выполнения может содержать несколько Java-объектов, относящихся к одному и тому же активному компоненту. Из-за этого могут возникать некоторые недоразумения, и, как мы увидим в главе 6, такая ситуация увеличивает вероятность утечек памяти.

Реализация компонента в приложении осуществляется созданием производного класса, и все компоненты в приложении должны быть зарегистрированы в файле *AndroidManifest.xml*.

Activity

Компонент Activity – это образ экрана, который почти всегда полностью занимает площадь реального экрана устройства. Здесь выво-

---

[1] В некоторых книгах об ОС Android на русском языке объекты типа Intent обозначены термином «намерения». – *Прим. перев.*

дится информация, обрабатывается ввод пользователя и т. д. Здесь же располагаются элементы пользовательского интерфейса – кнопки, текстовые фрагменты, изображения и др., – отображаемые на экране и содержащие ссылки на объекты в иерархии, включающей все экземпляры класса `View`. Следовательно, объём памяти, потребляемой компонентом `Activity`, может увеличиваться весьма существенно.

Когда пользователь перемещается между экранами, экземпляры компонента `Activity` образуют стек. При переходе к новому экрану предыдущий экземпляр `Activity` помещается в стек, а возврат в обратном направлении вызывает извлечение очередного экземпляра из стека.

На рис. 1.2 пользователь начал работу с экземпляром A компонента `Activity`, завершив его, перешёл к экземпляру B, затем к C и D. Экземпляры A, B и C являются полноэкранными компонентами, а D занимает лишь часть дисплея устройства. В результате экземпляр A будет удалён, B полностью невидим, C отображается частично, а D показан полностью, поскольку находится на вершине стека. Следовательно, D получает фокус и принимает данные, вводимые пользователем. Положение в стеке определяет состояние каждого экземпляра `Activity`:

- активный, расположенный на переднем плане: D;
- приостановлен и отображается частично: C;
- остановлен и невидим: B;
- неактивен и удалён: A.

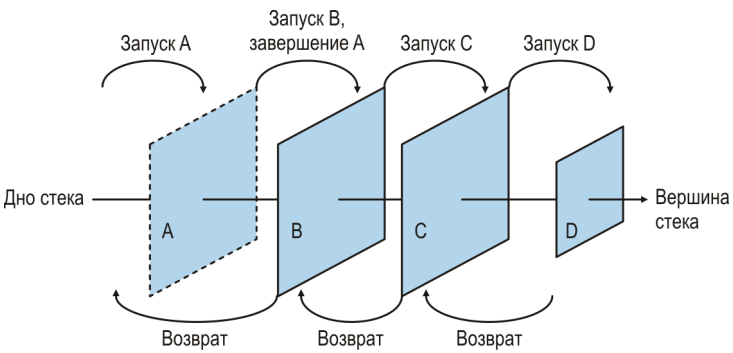

Рис. 1.2 ❖ Стек экземпляров Activity

Состояние экземпляра `Activity`, находящегося на вершине стека, определяет системный приоритет соответствующего приложения (системный приоритет также обозначается термином *«ранг процесса»*

(*process rank*)), который, в свою очередь, влияет на возможность завершения приложения (см. раздел «Завершение приложения» ниже) и на планирование выполнения потоков приложения (глава 3).

Жизненный цикл экземпляра компонента `Activity` завершается, когда пользователь даёт команду возврата к предыдущему экрану, например нажимает кнопку «Назад» (Back), или когда `Activity` явно вызывает метод `finish()`.

Service

Компонент `Service` может незаметно выполняться в фоновом режиме без прямого взаимодействия с пользователем. Обычно он используется для частичного снятия чрезмерной нагрузки с других компонентов, когда длительности операций превосходят длительности их существования. Компонент `Service` может работать в двух режимах: в режиме *запускаемой службы* (*started*) и в режиме *подключаемой службы* (*bound*):

- *в режиме запускаемой службы* – компонент `Service` запускается вызовом метода `Context.startService(Intent)`, с явным или неявным объектом `Intent`, и завершается вызовом метода `Context.stopService(Intent)`;
- *в режиме подключаемой службы* – несколько компонентов могут подключиться к компоненту `Service`, вызвав метод `Context.bindService(Intent, ServiceConnection, int)` и передав ему явный или неявный объект `Intent`. После подключения компонент может взаимодействовать с `Service` через интерфейс `ServiceConnection` и в любой момент разорвать установленную связь вызовом метода `Context.unbindService(ServiceConnection)`. Компонент `Service` автоматически удаляется, когда последний компонент разорвёт связь с ним.

ContentProvider

Если возникает необходимость использовать большие объёмы данных внутри одного приложения или совместно с другими приложениями, можно воспользоваться компонентом `ContentProvider`. Он способен обеспечить доступ к любому источнику данных, но наиболее часто применяется в связке с базами данных SQLite, которые всегда принадлежат только одному приложению. С помощью компонента `ContentProvider` любое приложение может предоставлять свои данные другим приложениям, выполняющимся в удалённых процессах.