Тема 10. Роль фонових процесів, явні та неявні виклики служб

Фоновими називаються процеси, які працюють у фоновому режимі.

Зазвичай додаток містить кілька операцій. Кожна операція повинна розроблятися в зв'язку з дією певного типу, яку користувач може виконувати, і може запускати інші операції. Наприклад, програма електронної пошти може містити одну операцію для відображення списку нових повідомлень. Коли користувач вибирає повідомлення, відкривається нова операція для перегляду цього повідомлення.

Операція може навіть запускати операції, що існують в інших додатках на пристрої. Наприклад, якщо ваше додаток хоче відправити повідомлення електронної пошти, можна визначити намір для виконання дії «відправити» і включити в нього деякі дані, наприклад адресу електронної пошти та текст повідомлення. Після цього відкривається операція з іншої програми, яка оголосила, що вона обробляє наміри такого типу. В цьому випадку намір полягає в тому, щоб відправити повідомлення електронної пошти, а тому в додатку електронної пошти запускається операція «скласти повідомлення» (якщо один намір може оброблятися декількома операціями, система пропонує користувачу вибрати, яку з операцій використовувати). Після відправлення повідомлення електронної пошти ваша операція відновлює роботу, і все виглядає так, ніби операція відправлення електронної пошти є частиною вашої програми. Хоча операції можуть бути частинами різних додатків, система Android підтримує зручність роботи користувача, зберігаючи обидві операції в одному завданні.

Задача – пов'язаний блок, який може переходити у фоновий режим, коли користувачі починають нову задачу або переходять на головний екран за допомогою кнопки Додому. У фоновому режимі всі операції задачі зупинені, але стек зворотного виклику для завдання залишається незмінним.

Потім задача може повернутися на передній план, а отже, користувачі можуть продовжити її з перерваного місця. Припустимо, наприклад, що поточна задача (задача А) містить три операції в своєму стеку – дві операції під поточною

операцією. Користувач натискає кнопку Додому, а потім запускає новий додаток із засобу запуску додатків. Коли з'являється головний екран, задача А переходить у фоновий режим. Коли запускається новий додаток, система запускає задачу для цього додатка (задача В) зі своїм власним стеком операцій. Після взаємодії з цим додатком користувач знову повертається на головний екран і вибирає спочатку запущену задачу А. Тепер задача А переходить на передній план – всі три операції її стека залишилися незмінними, і поновлюється операція, яка перебуває на вершині стека. У цей момент користувач може також переключитися назад на задачу В, перейшовши на головний екран і вибравши значок програми, яка запустила цю задачу (або вибравши задачу додатка на екрані огляду). Це приклад багатозадачності в системі Android.

У фоновому режимі може знаходитися декілька задач одночасно. Однак якщо користувач запускає багато фонових задач одночасно, система може почати знищення фонових операцій для звільнення пам'яті, що призведе до втрати стану завдань.

Як йшлося вище, система за замовчуванням зберігає стан операції, коли вона зупиняється. Таким чином, коли користувачі повертаються назад в попередню операцію, відновлюється її користувацький інтерфейс в момент зупинки.

Однак можна, і треба, з попередженням зберігати стан ваших операцій за допомогою методів зворотного виклику на випадок знищення операції і необхідності її повторного створення.

Коли система зупиняє одну з ваших операцій (наприклад, коли запускається нова операція або коли завдання переміщається у фоновий режим), вона може повністю знищити цю операцію, якщо необхідно відновити пам'ять системи. Коли це відбувається, інформація про стан операції втрачається, а система знає, що операція знаходиться в стеку переходів назад. Але коли операція переходить на вершину стека, система повинна створити її повторно (а не відновити її). Щоб уникнути втрати роботи користувача, необхідно з

попередженням зберігати її шляхом реалізації методів зворотного виклику onSaveInstanceState() у вашій операції.

Режими запуску дозволяють вам визначати зв’язок нового екземпляру операції з поточним завданням. Можна задавати різні режими запуску двома способами, використовуючи:

1. Файл маніфесту. Коли ви оголошуєте операцію у файлі маніфесту, ви можете вказати, як операція повинна зв’язуватися з задачами при її запуску

2. Прапорці намірів. Коли ви викликаєте startActivity(), ви можете включити прапорець в Intent, який оголошує, як повинна бути пов’язана нова операція з поточною задачею (і чи повинна).

По суті, якщо операція А запускає операцію В, операція В може визначити у своєму маніфесті, як вона має бути пов’язана з поточною задачею (якщо взагалі має), а операція А може також запросити, як Операція В повинна бути пов’язана з поточною задачею. Якщо обидві операції визначають, як операція В має бути пов’язана з задачею, тоді запит операції А (як визначено в намірі) обробляється через запит операції В (як визначено в її маніфесті).

Якщо ваша служба використовується тільки локальним додатком і не взаємодіє з різними процесами, то можна реалізувати власний клас Binder, за допомогою якого клієнт отримує прямий доступ до загальнодоступних методів в службі.

Цей варіант підходить тільки в тому випадку, якщо клієнт і служба виконується всередині однієї програми і процесу, що є найбільш поширеною ситуацією. Наприклад, розширення класу відмінно підійде для музичного додатка, в якому необхідно прив’язати операцію до власної служби додатка, яка відтворює музику у фоновому режимі.

Це можна зробити таким чином:

1. Створіть у вашій службі екземпляр класу Binder, що має одну з таких характеристик:

– екземпляр містить загальнодоступні методи, які може викликати клієнт;

– екземпляр повертає поточний екземпляр класу Service, який містить загальнодоступні методи, які може викликати клієнт;

– екземпляр повертає екземпляр іншого класу, розміщений в службі, що містить загальнодоступні методи, які може викликати клієнт.

2. Поверніть цей екземпляр класу Binder з методу зворотного виклику onBind().

3. У клієнті отримайте клас Binder від методу зворотного виклику onServiceConnected () і виконайте виклики до прив’язаної служби за допомогою наданих методів.

Служба і клієнт повинні виконуватися в одному і тому ж додатку, оскільки в цьому випадку клієнт може транслювати повернутий об’єкт і належним чином викликати його API-інтерфейси. Крім того, служба та клієнт повинні виконуватися в рамках одного і того ж процесу, оскільки цей спосіб не має на увазі будь-якого розподілу за процесами.

Об’єкт LocalBinder надає клієнтам метод getService(), щоб вони могли отримати поточний екземпляр класу LocalService. Завдяки цьому клієнти можуть викликати загальнодоступні методи в службі. Наприклад, клієнти можуть викликати метод getRandomNumber() зі служби.

Якщо необхідно, щоб служба взаємодіяла з віддаленими процесами, то для надання інтерфейсу служби можна скористатися об’єктом Messenger. Такий підхід дозволяє організувати взаємодію між процесами (IPC) без необхідності використовувати AIDL.

Ось короткий огляд того, як слід використовувати об’єкт Messenger:

– Служба реалізує об’єкт Handler, який отримує зворотний виклик для кожного виклику від клієнта.

– Об’єкт Handler використовується для створення об’єкта Messenger (який є посиланням на об’єкт Handler).

– Об’єкт Messenger створює об’єкт IBinder, який служба повертає клієнтам з методу onBind().

– Клієнти використовують отриманий об’єкт IBinder для створення екземпляра об’єкта Messenger (який посилається на о бъект Handler служби), що використовується клієнтом для відправки об’єктів Message в службу.

– Служба отримує кожен об’єкт Message в своєму об’єкті Handler – зокрема, в методі handleMessage().

Таким чином, для клієнта відсутні «методи» для відправки виклику служби. Замість цього клієнт відправляє «повідомлення» (об’єкти Message), які служба отримує в своєму об’єкті Handler.

Клієнту потрібно лише створити об’єкт Messenger на основі об’єкта IBinder, повернутого службою, і відправити повідомлення за допомогою методу send().

Коли необхідно організувати взаємодію між процесами, використання об’єкта Messenger для інтерфейсу набагато простіше від реалізації за допомогою AIDL, оскільки об’єкт Messenger поміщає в чергу всі запити до служби, тоді як інтерфейс, оснований виключно на AIDL, відправляє службі кілька запитів одночасно, а для цього потрібна підтримка багатопотоковості.

У більшості додатків служба не повинна підтримувати багатопотоковість, тому при використанні об’єкта Messenger служба може одночасно обробляти один запит. Якщо вам важливо, щоб служба була багатопотоковою, то для визначення інтерфейсу слід використовувати AIDL.

Для прив’язки до служби компоненти додатка (клієнти) можуть використовувати метод bindService(). Після цього система Android викликає метод onBind() служби, який повертає об’єкт IBinder для взаємодії зі службою.

Прив’язка виконується асинхронно; bindService() повертається відразу ж і не повертає клієнту об’єкт IBinder. Для отримання об’єкта IBinder клієнту необхідно створити екземпляр ServiceConnection і передати його в

метод bindService(). Интерфейс ServiceConnection включає метод зворотного виклику, який система використовує для того, щоб видати об’єкт IBinder.

Виконати прив’язку до служби можуть тільки операції, інші служби і постачальники контенту – не можна самостійно виконати прив’язку до служби з ресивера.

Тому для прив’язки до служби з клієнта необхідно виконати такі дії.

1. Реалізувати інтерфейс ServiceConnection. Ваша реалізація повинна перевизначати два методи зворотного виклику:

– onServiceConnected(). Система викликає цей метод, щоб видати об’єкт IBinder, повернутий методом onBind() служби.

– onServiceDisconnected(). Система Android викликає цей метод в разі непередбаченої втрати з’єднання зі службою, наприклад, при збої в роботі служби або в разі її завершення. Цей метод не викликається, коли клієнт скасовує прив’язку.

2. Викликати метод bindService(), передавши в нього реалізацію інтерфейсу.

3. Коли система викликає ваш метод зворотного виклику onServiceConnected(), можна приступити до виконання викликів до служби за допомогою методів, визначених інтерфейсом.

4. Щоб відключитися від служби, треба викликати метод unbindService().

У разі знищення клієнта виконується скасування його прив’язки до служби, проте вам завжди слід відміняти прив’язку по завершенні взаємодії зі службою або в разі припинення операції, щоб служба могла завершити свою роботу, коли вона не використовується.

Коли виконується скасування прив’язки служби до всіх клієнтів, система Android знищує таку службу (якщо вона не була запущена разом з onStartCommand()). У такому випадку не потрібно управляти життєвим циклом своєї служби, якщо вона виключно прив’язана служба – система Android управляє нею за вас на підставі прив’язки служби до будь-яких інших клієнтів.

Однак ви якщо вирішите реалізувати метод зворотного виклику onStartCommand(), то необхідно явно зупинити службу, оскільки в цьому випадку вона вважається запущеною. В такому випадку служба виконується

доти, поки сама не зупинить свою роботу за допомогою методу stopSelf() або доти, поки інший компонент не викличе метод stopService() незалежно від прив’язки служби до будь-яких клієнтів.

Крім того, якщо ваша служба запущена і приймає прив’язку, то при виклику системою вашого методу onUnbind() можна повернути true, якщо ви бажаєте отримати виклик до onRebind() при наступній прив’язці до служби (замість отримання виклику до методу onBind()). Метод onRebind() повертає значення void, однак клієнт, як і раніше, отримує об’єкт IBinder в своєму методі зворотного виклику onServiceConnected().