

Компоненты android приложений

Продолжаем обсуждать компоненты, которые можно использовать при разработке android приложений.

Сервисы (Services)

Сервис (Service) является компонентом приложения, предназначенным для выполнения длительных операций в фоновом режиме. Существует два способа существования сервисов:

- первый заключается в том, что сервис запущен (started) и работает самостоятельно в фоновом режиме, так он может работать неопределенно долго, пока не выполнит свою задачу;
- второй заключается в том, что сервис привязан (bound) к некоторому компоненту или нескольким компонентам, в этом случае сервис предлагает интерфейс для взаимодействия с компонентом и работает пока привязан хотя бы к одному компоненту, как только связь со всеми компонентами разрывается сервис завершает свою работу.

Для создания сервиса необходимо создать класс–наследник класса Service напрямую или через любого его потомка. При этом в реализации класса необходимо переопределить (т. е. написать свою реализацию) некоторые методы, управляющие ключевыми аспектами жизненного цикла сервиса и обеспечивающие механизм связывания компонентов с сервисом, в соответствующем случае.

Рассмотрим наиболее важные методы требующие реализации при создании сервиса.

`onStartCommand()` – метод, вызываемый системой, когда некоторый компонент, например активность, вызывает метод `startService()`. В этом случае сервис запускается и может работать в фоновом режиме неопределенно долго, поэтому необходимо позаботиться об остановке сервиса, когда он выполнит свою работу. Для остановки сервиса используется метод `stopSelf()` в случае, когда сервис сам прекращает свою работу, или `stopService()` в случае, когда работу сервиса прекращает некоторый компонент. Нет необходимости писать реализацию метода `onStartCommand()`, если не предполагается самостоятельной работы сервиса (т.е. он будет работать только в связке с некоторыми компонентами)

`onBind()` – метод, вызываемый системой, когда некоторый компонент желает привязать к себе сервис и вызывает метод `bindService()`. Этот метод должен возвращать реализацию интерфейса `IBinder`, которая может быть использована компонентом–клиентом для взаимодействия с сервисом. Метод `onBind()` необходимо

реализовать в любом случае, но, если не предполагается связывания сервиса с какими-либо компонентами, возвращаемое значение должно быть равным null.

Необходимо отметить, что сервис может быть запущен как самостоятельная единица, а в последствии, может быть привязан к некоторым компонентам. В этом случае в сервисе должны быть обязательно реализованы оба метода `onStartCommand()` и `onBind()`.

`onCreate()` – метод, вызываемый системой, при первом обращении к сервису для выполнения первоначальных настроек. Этот метод вызывается до вызова методов `onStartCommand()` и/или `onBind()`.

`onDestroy()` – метод, вызываемый системой, когда сервис либо выполнил все действия, для которых создавался, либо больше не связан ни с одним компонентом, т.е. его услуги больше не требуются. В реализации этого метода необходимо предусмотреть освобождение всех ресурсов, таких как потоки, зарегистрированные слушатели, приемники и т. д. Вызов этого метода является последним вызовом, который может получить сервис.

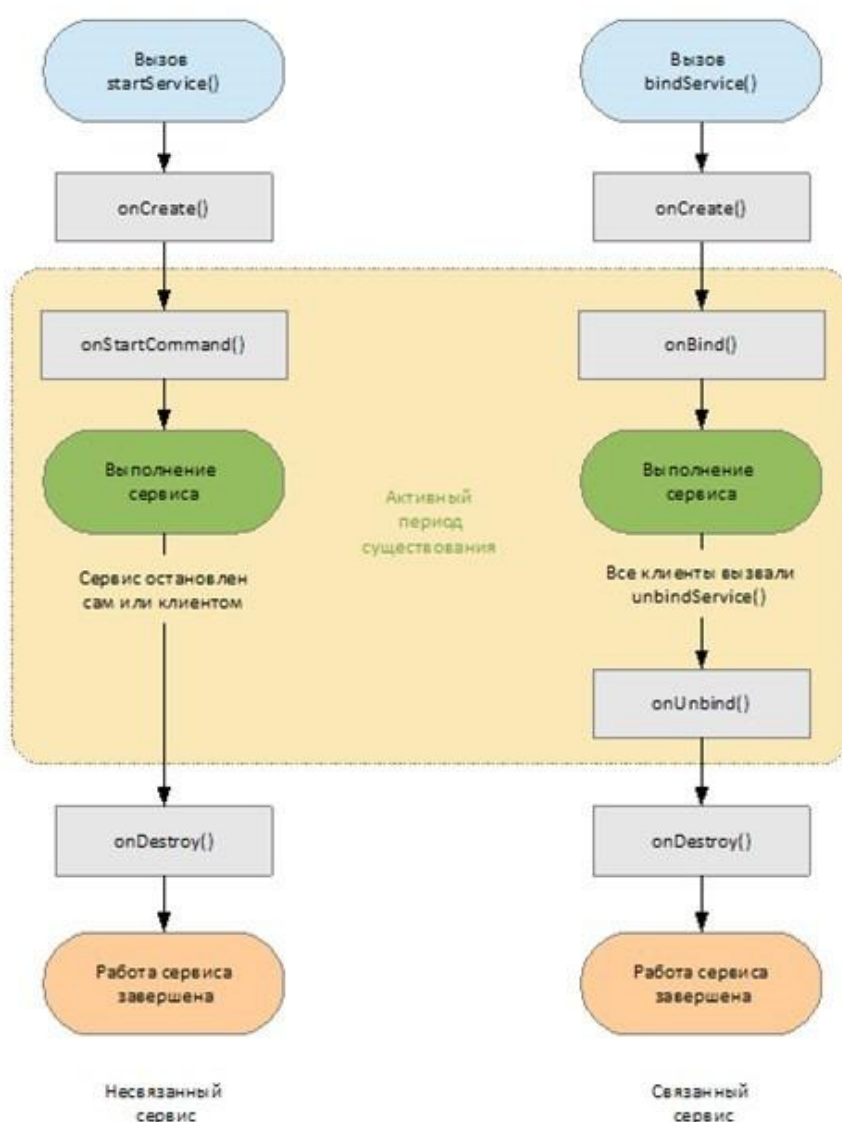


Рисунок 1. Жизненный цикл сервиса

На рисунке 1 показан жизненный цикл сервиса, левая диаграмма показывает жизненный цикл самостоятельного сервиса, правая – жизненный цикл сервиса, привязанного к некоторым компонентам. На рисунке хорошо видно, что жизненный цикл сервиса намного проще жизненного цикла активности.

Android принудительно останавливает работу сервисов, когда ресурсов системы не хватает для активности, которая в данный момент на экране. Приоритет работающих сервисов всегда выше, чем у приостановленных или полностью невидимых активностей, а если сервис привязан к выполняющейся активности, то его приоритет еще выше.

С другой стороны, со временем приоритет самостоятельно работающего сервиса понижается и его шансы быть принудительно остановленным системой в случае нехватки ресурсов повышаются. В связи с этим имеет смысл проектировать сервис таким образом, чтобы через некоторое время он требовал у системы перезапуска. В случае если система все таки экстренно завершила работу сервиса, она перезапустит его как только освободятся ресурсы.

Контент–провайдеры (Content Providers)

Контент–провайдер управляет доступом к хранилищу данных. Для реализации провайдера в Android приложении должен быть создан набор классов в соответствии с манифестом приложения. Один из этих классов должен быть наследником класса `ContentProvider`, который обеспечивает интерфейс между контент–провайдером и другими приложениями. Основное назначение этого компонента приложения заключается в предоставлении другим приложениям доступа к данным.

В мобильных приложениях контент–провайдеры необходимы в следующих случаях:

- приложение предоставляет сложные данные или файлы другим приложениям;
- приложение позволяет пользователям копировать сложные данные в другие приложения;
- приложение предоставляет специальные варианты поиска, используя поисковую платформу (framework).

Если приложение требует использования контент–провайдера, необходимо выполнить несколько этапов для создания этого компонента:

Проектирование способа хранения данных.

Данные, с которыми работают контент–провайдеры, могут быть организованы двумя способами:

Данные представлены файлом, например, фотографии, аудио или видео. В этом случае необходимо хранить данные в собственной области памяти приложения. В ответ на запрос от другого приложения, провайдер может возвращать ссылку на файл.

Данные представлены некоторой структурой, например, таблица, массив. В этом случае необходимо хранить данные в табличной форме. Строка таблицы представляет собой некоторую сущность, например, сотрудник или товар. А столбец – некоторое свойство этой сущности, например, имя сотрудника или цена товара.

В системе Android общий способ хранения подобных данных – база данных SQLite, но можно использовать любой способ постоянного хранения.

Создание класса-наследника от класса `ContentProvider`

напрямую или через любого его потомка. При этом в реализации класса необходимо переопределить (т. е. написать свою реализацию) обязательные методы.

`query()` – метод, извлекающий данные из провайдера, в качестве аргументов получает таблицу, строки и столбцы, а также порядок сортировки результата, возвращает объект типа `Cursor`.

`insert()` – метод, добавляющий новую строку, в качестве аргументов получает таблицу, и значения элементов строки, возвращает URI добавленной строки.

`update()` – метод, обновляющий существующие строки, в качестве аргументов получает таблицу, строки для обновления и новые значения элементов строк, возвращает количество обновленных строк.

`delete()` – метод, удаляющий строки, в качестве аргументов принимает таблицу и строки для удаления, возвращает количество удаленных строк.

`getType()` – метод, возвращающий `String` в формате MIME, который описывает тип данных, соответствующий URI.

`onCreate()` – метод, вызываемый системой, сразу после создания провайдера, включает инициализацию провайдера. Стоит отметить, что провайдер не создается до тех пор, пока объект `ContentResolver` не попытается получить к нему доступ.

Созданный контент-провайдер управляет доступом к структурированным данным, выполняя обработку запросов от других приложений. Все запросы, в конечном итоге, вызывают объект `ContentResolver`, который в свою очередь вызывает подходящий метод объекта `ContentProvider` для получения доступа. Все вышеперечисленные методы, кроме `onCreate()`, вызываются приложением-клиентом. И все эти методы имеют такую же сигнатуру, как одноименные методы класса `ContentResolver`.

Определение строки авторизации провайдера, URI для его строк и имен столбцов.

Если от провайдера требуется управление намерениями, необходимо определить действия намерений, внешние данные и флаги. Также необходимо определить разрешения, которые необходимы приложениям для доступа к данным провайдера. Все эти значения необходимо определить как константы в отдельном классе, этот класс в последствии можно предоставить другим разработчикам.

Приемники широковещательных сообщений (Broadcast Receivers)

Каждый широковещательный приемник является наследником класса `BroadcastReceiver`. Этот класс рассчитан на получение объектов-намерений отправленных методом `sendBroadcast()`.

Можно выделить две разновидности широковещательных сообщений:

Нормальные широковещательные сообщения передаются с помощью `Context.sendBroadcast` в асинхронном режиме. Все приемники срабатывают в неопределенном порядке, часто в одно и то же время.

Направленные широковещательные сообщения передаются с помощью `Context.sendOrderedBroadcast` только одному приемнику в один момент времени. Как только приемник сработает, он может передать сообщение следующему приемнику, а может прервать вещание так, что больше ни один приемник это сообщение не получит.

Даже в случае нормального широковещания могут сложиться ситуации, в которых система будет передавать сообщения только одному приемнику в один момент времени. Особенно это актуально для приемников, которые требуют создания процессов, чтобы не перегружать систему новыми процессами. Однако в этом случае ни один приемник не может прервать широковещание.

Объект типа `BroadcastReceiver` действителен только во время вызова метода `onReceive()`, как только метод выполнен, система завершает работу объекта и больше не активирует его.