Часть четвёртая ответника, платформа Android.

GoncharovA Follow
Jan 2 · 15 min read

Самая длинная часть списка ответов по платформе Android.

. . .

35. Launch Modes for activity (standart, singleTop, singleTask, singleInstance):

Атрибут launchMode указывает инструкцию по запуску операции в задаче. Существует четыре различных режима запуска, которые вы можете назначить атрибуту launchMode:

1.“standard” — Режим по умолчанию. Система создает новый экземпляр операции в задаче, из которой она была запущена, и направляет ему намерение. Может быть создано несколько экземпляров операции, каждый экземпляр может принадлежать различным задачам, и одна задача может содержать несколько экземпляров.

2. “singleTop” — Если экземпляр операции уже существует на вершине текущей задачи, система направляет намерение в этот экземпляр путем вызова его метода onNewIntent(), а не путем создания нового экземпляра операции. Может быть создано несколько экземпляров операции, каждый экземпляр может принадлежать различным задачам, и одна задача может содержать несколько экземпляров (но только если операция на вершине стека переходов назад не является существующим экземпляром операции).Предположим, что стек переходов назад задачи состоит из корневой операции A с операциями B, C и D

на вершине (стек имеет вид A-B-C-D и D находится на вершине). Поступает намерение для операции типа D. Если D имеет режим запуска “standard” по умолчанию, запускается новый экземпляр класса и стек принимает вид A-B-C-D-D. Однако, если D имеет режим запуска “singleTop”, существующий экземпляр D получает намерение через onNewIntent(), так как этот экземпляр находится на вершине стека — стек сохраняет вид A-B-C-D. Однако, если поступает намерение для операции типа B, тогда в стек добавляется новый экземпляр B, даже если он имеет режим запуска “singleTop”.

3. “singleTask” — Система создает новую задачу и создает экземпляр операции в корне новой задачи. Однако, если экземпляр операции уже существует в отдельной задаче, система направляет намерение в существующий экземпляр путем вызова его метода onNewIntent(), а не путем создания нового экземпляра. Одновременно может существовать только один экземпляр операции.

4. “singleInstance” То же, что и “singleTask”, но при этом система не запускает никаких других операций в задаче, содержащей этот экземпляр. Операция всегда является единственным членом своей задачи; любые операции, запущенные этой операцией, открываются в отдельной задаче.

36. Intent flags для запуска activity (FLAG_ACTIVITY_NEW_TASK, FLAG_ACTIVITY_SINGLE_TOP, FLAG_ACTIVITY_CLEAR_TOP):

При запуске операции вы можете изменить связывание операции с ее задачей по умолчанию путем включения флагов в намерение, которое доставляется в startActivity(). Для изменения поведения по умолчанию вы можете использовать следующие флаги:

FLAG_ACTIVITY_NEW_TASK — Запуск операции в новой задаче. Если задача уже работает для операции, которую вы запускаете сейчас, эта задача переводится на передний план, ее последнее состояние восстанавливается, и операция получает новое намерение в onNewIntent().

FLAG_ACTIVITY_SINGLE_TOP — Если запускаемая операция является текущей операцией (находится на вершине стека переходов назад), тогда вызов в onNewIntent() получает существующий экземпляр, без создания нового экземпляра операции.

FLAG_ACTIVITY_CLEAR_TOP — Если запускаемая операция уже работает в текущей задаче, тогда вместо запуска нового экземпляра этой операции уничтожаются все другие операции, расположенные в стеке выше нее , и это намерение доставляется в возобновленный экземпляр этой операции (которая теперь находится на вершине стека) посредством onNewIntent()).

37. Что такое ContentProvider:

ContentProvider являются одним из основных строительных блоков приложений Android, предоставляя контент приложениям. Они инкапсулируют данные и предоставляют их приложениям через единый интерфейс ContentResolver. ContentProvider требуется только в том случае, если вам нужно обмениваться данными между несколькими приложениями. Например, данные контактов используются несколькими приложениями и должны храниться в поставщике контента.

38. Типы сервисов (запущенный, привязанный, IntentService), их отличия и жизненные циклы:

Служба является «запущенной», когда компонент приложения (например, операция) запускает ее вызовом startService(). После запуска служба может работать в фоновом режиме в течение неограниченного времени, даже если уничтожен компонент, который ее запустил. Обычно запущенная служба выполняет одну операцию и не возвращает результатов вызывающему компоненту. Например, она может загружать или выгружать файл по сети. Когда операция выполнена, служба должна остановиться самостоятельно.Жизненный цикл: onCreate(), onStartCommand(), onDestroy().

Служба является «привязанной», когда компонент приложения привязывается к ней вызовом bindService(). Привязанная служба предлагает интерфейс клиент-сервер, который позволяет компонентам взаимодействовать со службой, отправлять запросы, получать результаты и даже делать это между разными процессами посредством межпроцессного взаимодействия (IPC). Привязанная служба работает только пока к ней привязан другой компонент приложения. К службе могут быть привязаны несколько компонентов одновременно, но когда все они отменяют привязку, служба уничтожается. Жизненный цикл: onCreate(), onBind(), onUnbind(), onDestroy().

IntentService — Это подкласс класса Service, который использует рабочий поток для обработки всех запросов запуска поочередно. Это оптимальный вариант, если вам не требуется, чтобы ваша служба обрабатывала несколько запросов одновременно. Достаточно реализовать метод onHandleIntent(), который получает намерение для каждого запроса запуска, позволяя выполнять фоновую работу. Жизненный цикл: onCreate(), onHandleIntent(), onDestroy().

39. Handler, Looper, MessageQueue — как это всё работает вместе:

Looper: который ещё иногда ещё называют «цикл обработки событий» используется для реализации бесконечного цикла который может получать задания используется. Класс Looper позволяет подготовить Thread для обработки повторяющихся действий. Такой Thread, как показано на рисунке ниже, часто называют Looper Thread. Главный поток Android на самом деле Looper Thread. MessageQueue — очередь Message: сообщение представляет собой контейнер для набора инструкций которые будут выполнены в другом потоке.

Handler: данный класс обеспечивает взаимодействие с Looper Thread. Именно с помощью Handler можно будет отправить Message с реализованным Runnable в Looper, которая будет выполнена (сразу или в заданное время) потоком с которым связан Handler. Код ниже иллюстрирует использование Handler. Этот код создаёт Activity которая завершиться через определённый период времени.

40. Планировщики задач:

AlarmManager, Handler, Service — решения для запуска бэкграунд-задач на основе сервисов.

JobScheduler — Это механизм, с чьей помощью можно в фоне выполнять различную работу, начало выполнения которой оптимизировалось и упрощалось за счёт централизованной системы запуска этих задач и возможности задавать условия для этого самого запуска.

GCM Network Manager — аналог JobScheduler — GCM Network Manager. Это библиотека, которая предоставляет схожий функционал, но работает уже с API 9. Правда, взамен требует наличие Google Play Services.

Firebase Job Dispatcher — Firebase JobDispatcher также является библиотекой для планирования фоновых заданий. Он также используется для поддержки обратной совместимости (ниже API 21) и работает во всех последних версиях Android (API 9+). Эта библиотека также будет работать, если на устройстве нет установленных сервисов Google Play. В этом состоянии эта библиотека внутренне использует AlarmManager. Если на устройстве доступно приложение Google Play, он использует механизм планирования в службах Google Play.

Sync Adapter — Sync adapters разработаны специально для синхронизации данных между устройством и облаком. Он должен использоваться только для этого типа задач. Синхронизация может быть вызвана изменениями данных в облаке или на устройстве или по истекшему времени. Система будет пытаться синхронизировать только тогда, когда устройство подключено к сети.

41. Serializable vs Parcelable:

Serializable — это стандартный интерфейс Java. Вы можете просто реализовать интерфейс Serializable и переопределить методы. Проблема этого подхода заключается в том, что используется рефлексия, и это медленный процесс. Этот метод создает много временных объектов и вызывает довольно много мусора. Тем не менее, Serializable интерфейс проще в реализации. Процесс Parcelable намного быстрее, чем Serializable. Parcelable является частью Android SDK. Для Parcelable мы явно реализуем процесс сериализации.

42. Виды Intent:

Явные объекты Intent указывают компонент, который требуется запустить, по имени (полное имя класса). Неявные объекты Intent не содержат имени конкретного компонента. Вместо этого они в целом объявляют действие, которое требуется выполнить, что дает возможность компоненту из другого приложения обработать этот запрос.

Позвонить — Intent.ACTION_CALL

Отправить смс — Intent.ACTION_VIEW setType(“vnd.android-dir/mms-sms”)

Получить контакт из телефонной книги — Intent.ACTION_PICK, android.provider.ContactsContract.Contacts.CONTENT_URI

Открыть ссылку в браузере — Intent.ACTION_VIEW, Uri.parse(“http://www.google.com")

Расшарить контент/написать письмо — Intent.ACTION_SEND setType(“text/plain”)

Открыть карту по определенным координатам либо запросу — Intent(android.content.Intent.ACTION_VIEW, Uri.parse(“geo:”+ latitude + “,” + longitude))

Сделать снимок с камеры — Intent(MediaStore.ACTION_IMAGE_CAPTURE)

Использовать Speech to Text для распознавания голоса — Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
putExtra(RecognizerIntent.EXTRA_PROMPT, “Speak please”)
startActivityForResult(speechIntent, RESULT_SPEECH_TO_TEXT)

PendingIntent — Передав PendingIntent другому приложению, вы предоставляете ему право выполнять указанную вами операцию, как если бы другое приложение было вами (с теми же разрешениями и идентификацией). Таким образом, вы должны быть осторожны с тем, как вы строите PendingIntent: почти всегда, например, базовый Intent, который вы предоставляете, должен иметь имя компонента, явно заданное для одного из ваших собственных компонентов, чтобы гарантировать, что оно в конечном итоге отправляется туда и никуда больше.

43. addOnBackStackChangeListener — что это и для чего:

public static interface FragmentManager.OnBackStackChangedListener

Добавлен в API level 11, Устарел в API level 28. Интерфейс для отслеживания изменений в backStack.

Предположим, что название вашего приложения изменяется в зависимости от имени фрагмента. Теперь, если вы перейдете от Fragment1 к Fragment2, а затем нажмете кнопку «Назад», с помощью addOnBackStackChangeListener вы узнаете, какой у вас фрагмент.

44. Dialog vs DialogFragment — отличия:

Класс Dialog — это базовый класс для создания диалоговых окон, но реализовывать напрямую класс Dialog не рекомендуется. Вместо этого следует использовать один из следующих подклассов:

AlertDialog — Диалоговое окно, в котором могут отображаться заголовок, кнопки вплоть до трех штук, список из выбираемых элементов либо пользовательский макет.

DatePickerDialog или TimePickerDialogДиалоговое окно с предопределенным пользовательским интерфейсом, с помощью которого пользователь указывает значения даты или времени.Эти классы определяют стиль и структуру вашего диалогового окна, однако следует использовать DialogFragment в качестве контейнера вашего диалогового окна.

Класс DialogFragment предоставляет все функции, необходимые для создания диалогового окна и управления его внешним видом, вместо вызова методов к объекту Dialog. Использование DialogFragment для управления диалоговым окном обеспечивает корректную обработку событий жизненного цикла , таких как нажатие пользователем кнопки Назад или поворот экрана. С помощью класса DialogFragment также происходит повторное использование пользовательского интерфейса диалогового окна в качестве встраиваемого компонента в пользовательский интерфейс более высокого уровня — подобно традиционному классу Fragment (например, когда необходимо различное отображение пользовательского интерфеса диалогового окна на больших и маленьких экранах).

45. Реализация Custom View и custom ViewGroup. Отличия в реализациях:

Расширьте существующий класс или подкласс View своим собственным классом. Переопределите некоторые методы из суперкласса. Методы суперкласса для переопределения начинаются с ‘on’, например, onDraw (), onMeasure () и onKeyDown (). Это похоже на события on … в Activity или ListActivity, которые вы переопределяете для жизненного цикла и других функций. Используйте свой

новый класс расширения. После завершения ваш новый класс может использоваться вместо View, на котором он основан.

ViewGroup — это специальное View, которое может содержать другие View (называемые дочерними). Этот класс определяет класс ViewGroup.LayoutParams, который служит базовым классом для параметров макетов.

46. MVC, MVP, MVVM, MVI — рассказать общие принципы:

Model View Controller — Model получает входные данные от контроллера и решает, какие данные необходимы для выполнения запроса, выданного контроллером. Модель также может выполнять транзакции с базами данных или другим постоянным хранилищем по мере необходимости. Как только он собирает необходимую информацию, он информирует представление (через контроллер) о новых данных, вызывая соответствующие события. View содержит элементы управления пользовательского интерфейса, необходимые конечному пользователю для взаимодействия с приложением. Его задача — отслеживать жесты конечного пользователя и передавать его контроллеру для дальнейшей обработки. View обрабатывает уведомление о событии, полученное от Модели, и может запрашивать любые новые данные, необходимые для его отображения на экране. Однако в некоторых вариантах контроллер может быть назначен для получения набора данных из модели, форматирования и отправки его в представление. Здесь задача View состоит в том, чтобы визуализировать данные без какой-либо другой обработки. Контроллер можно представить как расширение или личный помощник View, все события, происходящие из View, обрабатываются контроллером. Контроллер также проинформирует Model от имени View о том, что в представлении произошло какое-то событие и могут потребоваться некоторые новые данные.

Model-View-Presenter — Каждое View должно реализовывать соответствующий интерфейс. Интерфейс IView определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, IView.ShowErrorMessage(string msg)). Презентер должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе. Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и

практически никогда не обрабатываются логикой View (в т.ч. создания других View).

Model-View-View Model — Данный подход позволяет связывать элементы View со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя. View-модель — это абстракция View. Обычно означает, что свойства View совпадают со свойствами View-модели. View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет View и наоборот, поскольку используется механизм связывания данных (Bindings).

Model-View-Intent — Это такой шаблон проектирования, в котором Модель (Model) является активным компонентом, принимающим на вход Намерения (Intents) и производящая Состояния (State). Представление (View) в свою очередь принимает Модели Представления (View Model) и производит те самые Намерения. Состояние преобразуется в Модель Представления при помощи функции-трансформера (View Model Mapper).

47. Clean Architecture — рассказать общие принцип:

в Clean Architecture код разделен на несколько уровней, с правилом инверсии зависимостей: внутренний уровень не должен зависеть от каких-либо внешних уровней.

Слои:

Entities — бизнес сущности

UseCases — сценарии использования приложения

Presenters/Gateways/Controllers — слой представления

UI/Web/DB/Devices — реализации

Команды и данные всегда передаются следующим путем: view вызывает метод презентера(или ViewModel); Презентер обращается к экземпляру интерактора, который в него (в презентер) заинджекчен. Интерактор — это класс, соеджащий бизнес логику и реализующий интерфейс UseCase.

Интерактор, в свою очередь вызывает методы экземпляра репозитория. Причем на бизнес слое описан только интерфейс IRepository, реализация которого находится на слое данных. Таким образом, выполняется правило инверсии зависимостей.

На слое данных репозиторий обрящается к базе данных через DataAccesObject, к сети через retrofit или получает данные иным образом.

Данные затем передаются из репозитория через интерактор и презентер обратно во view.

48. MVP vs MVVM (отличия):

Презентер должен иметь ссылку на реализацию соответствующего интерфейса IView, которую обычно передают в конструкторе. ViewModel не может общаться со View напрямую. Вместо этого она представляет легко связываемые свойства и методы в виде команд. View может привязываться(Data Binding) к этим свойствам, чтобы получать информацию из ViewModel и вызывать на ней команды (методы). Это не требует того, чтобы View знала о ViewModel.

49. Dagger 2 (модули, компоненты, сабкомпоненты, скоупы, binds vs provides, named):

Dagger 2 представляет собой библиотеку, которая помогает разработчику реализовать паттерн “Внедрение зависимости” (Dependency Injection), который в свою очередь является “специфичной формой инверсии управления (Inversion of control)”.

Module — классы, чьи методы “предоставляют зависимости”.

Component — мост между @Module и @Inject(базовая аннотация, с помощью которой “запрашивается зависимость”)

Subcomponent — Необходимо прописывать в интерфейсе родителя метод получения Сабкомпонента (упрощенное название Subcomponent). Для Сабкомпонента доступны все объекты родителя. Родитель может быть только один.

Scope — Данный механизм берет на себя создание и хранение единственного экземпляра необходимого класса до тех пор, пока соответствующий scope существует.

Для Binds в отличии от Provides класс Module стал абстрактным, как и метод provide. У provide убрали аннотацию @Provide, зато добавили @Binds. А в аргументы передаем не необходимые зависимости, а конкретную реализацию.

Часто бывает, что нам необходимо провайдить несколько объектов одного типа. В этом случае нам приходит на помощь “qualifier annotation”. Это кастомная аннотация, которая имеет в себе аннотацию @Qualifier. Dagger2 предоставляет нам уже одну готовую “qualifier annotation” : @Named

50. Doze Mode — ЧТО ЭТО:

Когда устройство на Android api >= Marshmallow лежит без движения и без зарядки, спустя час оно переходит в Doze Mode. Режим сна, когда почти все приложения перестают потреблять энергию батареи. Это происходит не сразу, а по шагам:

ACTIVE — Устройство используется или на зарядке.

INACTIVE — Устройство недавно вышло из активного режима (пользователь выключил экран, выдернул зарядку и т.п.) …30 минут.

IDLE_PENDING — Устройство готовится перейти в режим ожидания …30 минут.

IDLE — Устройство в режиме бездействия.

IDLE_MAINTENANCE — Открыто короткое окно, чтобы приложения выполнили свою работу.

51. LiveData (value vs postValue, active() vs onInactive(), MediatorLiveData):

LiveData — хранилище данных, работающее по принципу паттерна Observer (наблюдатель). Это хранилище умеет делать две вещи:

В него можно поместить какой-либо объект;

На него можно подписаться и получать объекты, которые в него помещают.

Для обновления значения мы должны передать его с помощью метода setValue(T), будьте внимательны поскольку этот метод нужно вызывать с main треда, в противном случае мы получим IllegalStateException, если же нам нужно передать значение из другого потока можно использовать postValue(T), этот метод в свою очередь обновит значение в main треде. Интересной особенностью postValue(T) является еще то, что он в случае множественного вызова, не будет создавать очередь вызовов на main тред, а при исполнении кода в main треде возьмет последнее полученное им значение. Также, в классе присутствует два колбека:

onActive() — будет вызван когда количество подписчиков изменит свое значение с 0 на 1.

onInactive() — будет вызван когда количество подписчиков изменит свое значение с 1 на 0.

Класс MediatorLiveData — реализует поведенческий паттерн который определяет объект, инкапсулирующий способ взаимодействия множества объектов, избавляя их от необходимости явно ссылаться друг на друга.

52. ViewModel — рассказать общие принципы:

Класс ViewModel предназначен для хранения и управления данными, связанными с пользовательским интерфейсом, с учетом жизненного цикла. Класс ViewModel позволяет данным переживать изменения конфигурации, такие как поворот экрана. Объекты ViewModel автоматически сохраняются во время изменений конфигурации, поэтому данные, которые они хранят, сразу же становятся доступны для Activity или экземпляра фрагмента.

53. Room — рассказать общие принципы:

Room — это способ сохранить данные приложений в Android-приложении. Это часть Android Architecture, группы библиотек от Google, которые поддерживают уместную архитектуру приложений. Room предлагается в качестве альтернативы Realm, ORMLite, GreenDao и многим другим.

Это высокоуровневый интерфейс для низкоуровневых привязок SQLite, встроенных в Android, о которых вы можете узнать больше в документации. Он выполняет большую часть своей работы во время компиляции, создавая API-интерфейс поверх встроенного SQLite API, поэтому вам не нужно работать с Cursor или ContentResolver.

Room имеет три основных компонента: Entity, Dao и Database. Аннотацией Entity нам необходимо пометить объект, который мы хотим хранить в базе данных. В объекте Dao надо описать методы для работы с базой данных. нужны методы для получения списка объектов и для добавления/изменения/удаления объектов. Аннотацией Database помечаем основной класс по работе с базой данных. Этот класс должен быть абстрактным и наследовать RoomDatabase.

54. Moxy — рассказать общие принципы:

Библиотека Moxy отличается от всех прочих MVP-инструментов тем, что между View и Presenter появился ViewState. Он отвечает за то, чтобы каждая View всегда выглядела именно так, как того хочет Presenter. ViewState хранит в себе список команд, которые были переданы из Presenter во View. И когда „новая“ View присоединяется к Presenter, ViewState автоматически применяет к ней все команды, которые Presenter выдавал раньше. Таким образом получается, что не зависимо от того, что произойдёт со View по вине Android, View останется всё-равно в правильном состоянии. Для этого вам нужно будет только привыкнуть изменять View исключительно командами из Presenter.

55. Realm — рассказать общие принципы:

Это нативная no-sql база данных для Android. Так же есть backend, который позволяет синхронизировать данные из всех источников. Из ключевых особенностей стоит отметить zero copy, MVCC (MultiVersion Concurrency Control) и ACID(**A**tomicity, **C**onsistency, **I**solation, **D**urability). Встроенного механизма устаревания и очистки данных нет. Можно выделить три главные особенности:

Live Objects — Все объекты, полученные из Realm, являются, по сути, прокси к базе данных. За счет этого достигается zero copy (объекты не копируются из базы).

Transactions — Все изменения привязанных объектов данных нужно проводить внутри транзакции.

Open\Close — Необходимость открытия\закрытия instance базы данных.

56. Метод invalidate() in view — как работает:

Отрисовка выполняется путем обхода дерева и записи команд отрисовки любого view, который необходимо обновить. После этого на экран выводятся команды отрисовки всего дерева, обрезанные до вновь изменённой области. Дерево в основном записывается и рисуется по порядку, родители рисуются раньше (то есть позади) своих детей, а братья и сестры рисуются в том порядке, в котором они появляются на дереве. Если вы устанавливаете фон для рисования для View, то View будет рисовать его до вызова его метода onDraw(). Дочерний порядок рисования может быть переопределен с помощью ViewGroup.setChildrenDrawingOrderEnabled(boolean) в ViewGroup и с помощью setZ (float) пользовательских значений Z, установленных в Views. Чтобы заставить view отрисоваться, вызовите invalidate(). Основной цикл view выглядит следующим образом: Событие приходит и отправляется в соответствующий view. View обрабатывает событие и уведомляет всех слушателей. Если в ходе обработки события, возможно, потребуется изменить границы view, представление вызовет requestLayout(). Точно так же, если в ходе обработки события может потребоваться изменить внешний вид представления, представление вызовет invalidate(). Если были вызваны requestLayout () или invalidate(), framework позаботится об измерении, разметке и построении дерева соответствующим образом.

57. ValueAnimator, ObjectAnimator — рассказать общие принципы:

ObjectAnimator, подкласс ValueAnimator обеспечивает поддержку анимации свойств целевых объектов. Конструкторы этого класса принимают параметры, чтобы определить целевой объект, который будет анимирован, а также имя свойства, которое будет анимировано. Соответствующие функции set/get затем определяются внутри, и анимация будет вызывать эти функции по мере необходимости для анимации свойства.

58. Paging library — виды пагинации:

Библиотека Paging libraryпомогает вам загружать и отображать небольшие порции данных. Библиотека Paging library поддерживает следующие архитектуры данных:

Обслуживается только с внутреннего сервера.

Хранится только в базе данных на устройстве.

Сочетание других источников, использующих базу данных на устройстве в качестве кэша.

59. Zygote — что это и как работает:

Zygote — это процесс, порождающий все android-приложения. Когда вы нажимаете на иконку того же clash of clans, zygote создаёт копию своего процесса с помощью fork(), и в этой копии запускает нужное приложение.

60. Bundle что туда можно положить:

Примитивные типы, строки, массивы примитивных типов и массивы строк; Parcelable, Serializable и массивы Parcelable; Экземпляры классов Size и SizeF.

61. Spannable — для чего используется:

Это интерфейс для текста, к которому можно прикреплять и отсоединять объекты стилизации. Задание этих объектов заключается в присвоению части текста некоторого определенного стиля. Такими маркировочными объектами могут быть экземпляры классов, которые реализуют интерфейс ParcelableSpan.

62. Context activity vs application context:

application context это singleton-экземпляр (единственный на всё приложение), и к нему можно получить доступ через функцию getApplicationContext(). Этот контекст привязан к жизненному циклу приложения. Контекст приложения может использоваться там, где вам нужен контекст, жизненный цикл которого не связан с текущим контекстом или когда вам нужно передать контекст за пределы Activity.

Context activity контекст доступен в Activity и привязан к её жизненному циклу. Контекст Activity следует использовать для операций, связанных. с графическим интерфейсом.

63. Что прописываем в манифесте:

Список всех элементов, расположенных в алфавитном порядке, которые могут присутствовать в файле манифеста. Там могут находиться только эти элементы, а никакие другие элементы или атрибуты добавлять нельзя.

<action>, <activity>, <activity-alias>, <application>, <category>, <data>, <grant-uri-permission>, <instrumentation>, <intent-filter>, <manifest>, <meta-data>, <permission>, <permission-group>, <permission-tree>, <provider> <receiver>, <service>, <supports-screens>, <uses-configuration>, <uses-feature>, <uses-library>, <uses-permission>, <uses-sdk>

64. setRetainInstance in fragment — что даёт:

Свойство retainInstance фрагмента, по дефолту, является false. Это означает, что при поворотах девайса Fragment не сохраняется, а уничтожается и создается по новому вместе с Activity-host. При вызове setRetainInstance(true) фрагмент не уничтожается вместе с его хостом и передается новому Activity в не измененном виде.

65. AsyncTask — для чего и какие имеет недостатки:

Класс AsyncTask предлагает простой и удобный механизм для перемещения трудоёмких операций в фоновый поток. Недостатки:

Жизненный цикл — AsyncTask продолжает работать, пока его метод doInBackground() не завершится. AsyncTask запустит метод onCancelled(Result result), если мы отменяем задачу вызовом cancel(boolean), либо метод onPostExecute(Result result), если задача не была отменена. Предположим, наш AsyncTask не был отменен до того как Activity была уничтожена. Теперь, если AsyncTask попытается взаимодействовать с компонентами этой Activity программа упадет, т.к. Activity больше не существует.

Утечки памяти — Так как AsyncTask имеет методы, которые работают в фоновом потоке (doInBackground()), а также методы, которые работают в потоке

пользовательского интерфейса (например OnPostExecute()), он сохраняет ссылку на Activity, до тех пор пока работает. Но, если Activity уничтожена, он будет по-прежнему хранить эту ссылку в памяти.

Потеря результатов — Еще одной проблемой является то, что мы потеряем результаты выполнения AsyncTask если наша Activity будет пересоздана. Например, это происходит при повороте экрана. Activity будет уничтожена и создана заново, но наш AsyncTask теперь будет иметь недействительную ссылку на эту Activity, поэтому onPostExecute() не будет иметь никакого эффекта.

Последовательно или параллельно? До API версии 1.6 (Donut): В первой версии AsyncTask задачи выполнялись последовательно. Это значит, что следующая задача не будет запущена пока предыдущая не завершит свою работу.С API версии 1.6 до API версии 2.3 (Gingerbread): Разработчики Android решили изменить поведение так, что бы несколько AsyncTask могли работать параллельно в отдельных потоках. С API версии 3.0 (Honeycomb) по настоящее время AsyncTask'и снова выполняются последовательно. Разработчики Android предоставили возможность выполнять задачи параллельно. Это делается методом executeOnExecutor(Executor).

. . .

Все ответы собраны из следующих источников:

https://tproger.ru/

https://askdev.ru/

https://developer.android.com/

https://www.codeproject.com/

https://proglib.io/

https://emunix.org/post/the-dark-side-of-asynctask/

Android Development