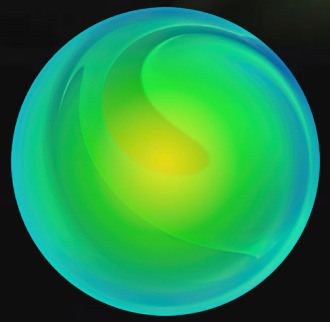
Как проводить собеседования интересно и продуктивно



Максим Сидоров,

Системные сервисы, Salute TV

SBER DEVICES

Я люблю проводить собеседования

— Принимаю активное участие в организации процессов найма
— Считаюсь одним из лучших экспертов по техническим интервью в SberDevices
— Провел более 500 технических интервью
— Прошел более 50 технических интервью

В 2021 г. провел большое исследование. Прошел собеседования в более чем 20 крупных компаниях на рынке.

Реальные отзывы от кандидатов

Самое интересное собеседование за последние несколько лет

Хотелось бы отметить второе собеседование, так глубоко по языкам никогда не спрашивали

Собеседование заставило взглянуть на многие вещи под другим углом

Честно не хотелось завершать собеседование, а хотелось продолжить общение

Хотел бы передать благодарность, потому что много нового для себя узнал

Но так было не всегда

Почему многим кандидатам нравятся мои собесы?

Давайте разбираться...

Типовая схема – скучно, просто, не информативно

— Ограниченный набор типовых вопросов
— Легко хакать
— Становится скучно уже после 3-5 собеседования
— Дает мало информации о реальных знаниях и опыте кандидата
— Не дает понимания о том, как кандидат думает и рассуждает

Что мы хотим узнать о кандидате из ответов на вопросы?

— Кругозор и глубина знаний кандидата
— Умение применять знания на реальных ситуациях
— Как кандидат думает и рассуждает
— Есть ли у него интерес к пониманию того как работает система под капотом

Вопросы должны быть интересными и не типовыми

— Вопрос не должен иметь простого ответа
— Вопрос должен вскрывать суть темы и не быть поверхностным
— Вопрос должен заставлять задуматься
— Форма вопроса должна быть КАК?, ЗАЧЕМ? и ПОЧЕМУ?
— Ответ на вопрос должен показывать глубину и уровень знаний, а не простое знание ответа

Давайте рассмотрим это на конкретных примерах

Попробуем взять наиболее избитые типовые вопросы и сделать их интересными и информативными для нас

А заодно разберем кейсы, которые не знает или не до конца понимает большинство разработчиков

основные компоненты Android приложения

Назови основные компоненты Android приложения?

—Не дает нам абсолютно никакой информации о знаниях кандидата

—Абсолютно бессмысленный вопрос даже для джунов

А почему Activity, Service, Broadcast и Content Provider называют основными компонентами?

Основные компоненты

По сути это точки входа в приложение

— Приложения могут общаться друг с другом только через основные компоненты

— Каждый компонент отвечает за свою функциональную задачу

— Android может управлять каждым компонентом независимо, запускать и уничтожать его

Что мы узнаем из ответа

— Раскрывает понимание кандидатом принципов архитектуры Android
— Понимание как приложения могут взаимодействовать друг с другом
— Понимание зачем нужен манифест и как система взаимодействует с приложением

Но я предпочитаю задавать вопрос про компоненты иначе

У меня есть два приложения, которые должны общаться друг с другом.

Как мне это сделать в Android?

По ответу на один этот вопрос я в 90% случаев правильно определяю финальный уровень кандидата!

По сути это тот же вопрос про основные компоненты

— Но тут важно понимать, знает ли кандидат про IPC, AIDL и про то, что взаимодействие через биндер лежит в основе всего Android

— Дойдет ли кандидат до взаимодействия через сокеты

— Скажет ли он про ограничение на размер bundle

— Предложит ли взаимодействие файлами через общую папку или скажет про FileProvider

— Мне остается лишь повторять «А еще?», пока кандидат перечисляет мне все основные способы общения процессов в Android.

Permissions

А как мне защитить обмен данными между моими приложениями?

—Чтобы третья сторона не могла вмешаться в это взаимодействие

Неправильный ответ

Абсолютное большинство кандидатов начинают городить схемы с шифрованием данных

И тут очень удобно задать вопрос про хранение приватных ключей в Android, про Android KeyStore Provider и аппаратное хранилище ключей

Хранилища приватных ключей

Android KeyStore Provider – общее api для доступа к хранилищу приватных ключей

StrongBox Keymaster - аппаратное хранилище ключей. Доступно начиная с API 28

Trusted Execution Environment (TEE) – внутри ОС Android существует изолированная OS Trusty на базе которой реализовано изолированное хранилище ключей. Доступно начиная с API 29

Signature Permissions

Вы можете объявить свои собственные permission с защитой подписи и закрыть ими доступ к любому компоненту вашего приложения

```
<permission android:name="com.myapp.WHAT_I_WANT_CONTROL"
    android:protectionLevel="signature" />
  <activity android:name="com.example.benchmark.ui.MainActivity"
      android:exported="true"
      android:permission="com.myapp.WHAT_I_WANT_CONTROL">
  </activity>
```

Что мы узнаем из ответа на вопрос

— Знает ли кандидат про встроенные механизмы безопасности Android
— Знает ли он про то, что permission можно не только запрашивать, но и объявлять свои
— Понимает ли кандидат, как можно защищать приложения в Android
— С сеньором можно поговорить о том, как можно обойти защиту через permissions

Да, это редкий кейс и не все с ним сталкиваются.

Я задаю этот вопрос, потому что он важен для нас и мы активно используем его в работе

Жизненный цикл фрагмента

Назови жизненный цикл фрагмента?

— Многие кандидаты знают ж.ц. фрагмента, но очень не многие понимают его

— Простое перечисление событий ж.ц. не дают нам информации о том, понимает ли кандидат нюансы жизненного цикла

Есть фрагмент **A**, мы заменяем его на фрагмент **B**

(replace + addToBackStack).

Какие события жизненного цикла вызовутся у фрагмента **A**?

Все знают жизненный цикл фрагмента, но не все его понимают

Жизненный цикл фрагмента содержит внутри себя жизненный цикл View фрагмента.

За время жизни фрагмента его View может многократно создаваться и уничтожаться.

Так как мы добавляем фрагмент в backStack, то будут вызваны следующие события: **onPause, onStop, onDestroyView**

Уничтожается только View фрагмента, сам фрагмент продолжает жить во FragmentManager

Контракт equals + hashCode

Расскажи контракт equals и hashCode

— Контракт equals – это как детский стишок без смысла

— Важна связь контрактов equals + hashCode, но понимает ли кандидат зачем она нужна и где она используется? Что будет, если ее нарушить?

А зачем вообще нужен контракт между equals и hashCode, как и где это используется?

— Плавно подводит нас к устройству HashMap, к разговору про коллизии и про потерю ключей в HashMap

— Непонимание и нарушение этих контрактов может приводить к очень серьезным ошибкам

Популярное заблуждение

hashCode ускоряет поиск и сравнение объектов

Внутри equals сначала проверяем объекты по hashCode и только если они равны, то сравниваем их по значению

Реализация equals в HashMap

```
public final boolean equals(Object o) {
    if (o == this)
        return true;
    if (o instanceof Map.Entry) {
        Map.Entry<?,?> e = (Map.Entry<?,?>)o;
        if (Objects.equals(key, e.getKey()) &&
            Objects.equals(value, e.getValue()))
            return true;
    }
    return false;
}
```

Реализация equals в data class

```
data class SampleData(val id: String, val name: String) {

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other !is SampleData) return false

        if (id != other.id) return false
        return name == other.name
    }
```

Как видите, никакой проверки hashCode тут нет

Сравнение LinkedList и ArrayList

Расскажи про плюсы и минусы ArrayList и LinkedList

—В целом это нормальный вопрос на понимание различий

—Но уж слишком он избитый и затасканный

У меня есть функция в которую передается массив из 1 млн элементов.

На выходе надо вернуть коллекцию из этих элементов. По коллекции я буду бегать только через итератор.

Какую коллекцию мне создать внутри моей функции: ArrayList или LinkedList?

Это задача ловушка

Условия задачи специально составлены таким образом, чтобы по скорости обе структуры работали одинаково

Рассуждая над этой задачей кандидат сам расскажет нам

— О разнице в скорости при доступе по индексу

— О разнице в скорости при вставке в середину

— О с

Но главное, что мы должны оценивать алгоритмы не только по скорости, но и по памяти. Об этом многие забывают, а в данном случае это важно!

LinkedList займет в 4 раза больше памяти, чем ArrayList

Как так то? 4X

В ArrayList каждый элемент в массиве занимает **ровно 1 указатель**

— В LinkedList каждый элемент хранится внутри объекта Node (+1)
— В каждом Node есть ссылка на предыдущий и следующий элемент (+2)
— И ссылка на само значение нашего элемента (+1)

```
private class Node<E> {
    var item: E? = null
    var next: Node<E>? = null
    var prev: Node<E>? = null
}
```

Итого 4 указателя

Расскажи про корутины

—Контекст корутины

—Ланучеры

—Scope, Job и SupervisorJob

А как происходит прерывание корутин?

Как и в каких местах может прерываться / приостанавливаться suspend функция?

Прерывание корутин

В какой точке может прерваться/приостановится данная функция?

```
private suspend fun testSuspending1() {
    doSuspend()
    doOrdinary()
    doSuspend()
    doOrdinary()
}
```

А какая разница, разве это важно? Говорят многие кандидаты

Прерывание корутин

Теперь стало важно!!!

```
private suspend fun testSuspending2() {
    veryImportantFlag = false
    doSuspend()
    doOrdinary()
    doSuspend()
    doOrdinary()
    veryImportantFlag = true
}
```

Всегда ли вызовется этот код?

Эксперимент

```
val job = launch(Dispatchers.Default) {
    println("job started")
    try {
        (0 ..< 3).forEach { iteration ->
            doSuspend(iteration)
            println("sleep 10")
            Thread.sleep(millis: 10)
            doOrdinary(iteration)
        }
    } finally {
        println("run finally block")
    }
}

delay(timeMillis: 5L)
println("canceling job.....................")
job.cancelAndJoin()
println("job canceled")
```

Termianl

```
job started
doSuspend(iteration 0)
sleep 10
canceling job.....................
doOrdinary(iteration 0)
doSuspend(iteration 1)
sleep 10
doOrdinary(iteration 1)
doSuspend(iteration 2)
sleep 10
doOrdinary(iteration 2)
doSuspend(iteration 3)
sleep 10
doOrdinary(iteration 3)
run finally block
job canceled
```

Эксперимент 2

```
val job = launch(Dispatchers.Default) { this: Coroutir
    println("job started")
    try {
        (0 ≤ .. ≤ 3).forEach { iteration ->
            doSuspend(iteration)
            println("delay 10")
            delay( timeMillis: 10) // Thread.sleep(10)
            doOrdinary(iteration)
        }
    } finally {
        println("run finally block")
    }
}

delay( timeMillis: 5L)
println("canceling job.....................")
job.cancelAndJoin()
println("job canceled")
```

Termianl

```
job started
doSuspend(iteration 0)
delay 10
canceling job.....................
run finally block
job canceled
```

Почему так происходит с delay???

Функция delay внутри проверяет активность Job и если он отменен, то бросает явное CancelationException

if the Job of the current coroutine is cancelled while this suspending function is waiting, this function immediately resumes with CancellationException. There is a **prompt cancellation guarantee**: even if this function is ready to return the result, but was cancelled while suspended, CancellationException will be thrown.

Эксперимент 3

```
val job = launch(Dispatchers.Default) { this: Coroutine
    println("job started")
    try {
        (0 ≤ .. ≤ 3).forEach { iteration ->
            doSuspend(iteration)
            println("delay 10")
            delay( timeMillis: 10) // Thread.sleep(10)
            doOrdinary(iteration)
        }
    } catch (e: Exception) {
        println("catch exception: ${e.message}")
        throw e
    } finally {
        println("run finally block")
    }
}
```

Termianl

```
job started
doSuspend(iteration 0)
delay 10
canceling job.....................
catch exception: StandaloneCoroutine was cancelled
run finally block
job canceled
```

Прерывание корутин

Механизм прерывания и приостановки suspend функций лежит в основе корутин

— Непонимание этого механизма часто приводит к ошибкам утечек файловых хендлеров или блокировки потоков
— В случае отмены Job, suspend функция доработает до конца
— В случае любого исключения, выполнение suspend функции будет прервано на следующем внутреннем вызове suspend функции
— Использование delay может прерывать выполнение suspend функций, так как внутри порождается CancelableException

synchronized

Расскажи про synchronized методы

—Расскажи про монитор объекта

—Методы wait/notify

—Реинтерабельность критической секции (рекурсивный synchronized)

А что вообще происходит внутри synchronized?

Почему говорят, что синхронизация это медленно?

synchronized

Блокировки и монопольное исполнение кода synchronized это скорее следствие, а не причина

Основная задача synchronized – это синхронизация кеша потока с общей памятью в куче

Поток хранит все свои данные в стековой памяти, которую называют кешем потока

— При входе в блок синхронизации происходит обновление кеша потока из общей памяти. Это физическое чтение всех доступных потоку переменных из общей памяти в локальный стек.

— При выходе из блока синхронизации происходит сброс локального кеша в общую память. Это физическая запись всех локальных переменных в стеке в общую память.

Кандидат должен думать

Все мои вопросы

— Не имеют простого ответа

— Кандидат приходит к ответам рассуждая, думая и иногда выдвигая гипотезы

— Связаны с темами, которые хорошо знакомы всем разработчикам

— Связаны с действительно важными вещами и потенциальными проблемами

— Позволяют закопаться в тему достаточно глубоко

— Иногда позволяют кандидату узнать что то новое

Мы оцениваем кандидата, а кандидат оценивает компанию

Один из важных критериев выбора компании – это уровень технической экспертизы. Хороший кандидат всегда думает о профессиональном росте.

— Есть ли в компании люди, у которых он может учится?

— На сколько интересными проектами и задачами ему предстоит заниматься?

Интересное собеседование – это очень хороший способ продать кандидату компанию

Совет

Ходите на собеседования к коллегам, смотрите ролики на Youtube

Старайтесь придумывать свои интересные вопросы

Очень часто вопросы на интервью дают нам самим новые знания

linkedIn:
sidorov-max

Мои статьи
на Хабр

Мои статьи на
ProAndroidDev