KOTLIN INTERVIEW SECRETS

ADROID DEVELOPEMENT IN KOTLIN
INTERVIEW QUESTION & ANSWER

RZ RASEL

TECH
INTERVIEWS

lateinit vs lazy?

Coroutines Interview Questions For Android Kotlin

What are Coroutines?

A coroutine can be thought of as a worker that performs some long-running/memory-intensive operations asynchronously. The asynchronous nature of a coroutine ensures that any long-running/memory-intensive operations do not block the main thread of execution. In essence, it takes a block of code and runs it on a particular thread.

How are they different from Threads?

Coroutines may be thought of as lightweight threads. They are called so because multiple coroutines can be scheduled to be executed on the same thread. So, the creation of 100,000 threads results in the creation of 100,000 threads. Whereas, the creation of 100,000 coroutines doesn't necessarily mean that 100,000 threads get created. Since the resources of a single thread are shared by multiple coroutines, they are much lighter when compared to creating raw threads.

What problem do Coroutines solve?

Coroutines solve many problems related to concurrency. They simplify the process of writing asynchronous code.

Elimination of chaining callbacks a.k.a "Callback Hell" : "Callback hell" refers to the infamous practice of nesting callback functions. There are a lot of scenarios wherein callback functions need to be nested. This results in the nesting of callback functions making the code extremely difficult to read and debug. The sequential nature of coroutines precludes the need for using nested callbacks, making the code much more readable and maintainable.

Exception handling and cancellation
Exception handling and cancellation can be difficult to manage in a concurrent environment. They make it very easy to handle exceptions. They not only provide methods of propagating and handling exceptions but also allow us to define the cancellation behavior. This feature allows us to write safe concurrent code.

Sequential Execution - Makes concurrent code easy to read and debug
Each operation within a coroutine executes sequentially. For example, if there is a network call to fetch the details of a product followed by another network call to fetch the image of the product, then, the second request doesn't execute unless the first request was successfully executed. This has several benefits. First and foremost, it allows us to skip unnecessary network calls. In this example, if a fetch operation to fetch the details of an invalid product is made, then, the fetch operation to get the image of the product can be completely skipped. Secondly, it also makes debugging very easy, since asynchronous code is represented synchronously. Now, this doesn't mean that it is not possible to execute more than one network call at the same time. The async{}

block provided by the coroutines library can be used to achieve concurrent execution.

Scoped Execution - Prevent inefficient use of resources
All coroutines must be started in a coroutine scope. If the scope gets canceled, then all coroutines that were started within that scope get canceled. This helps to prevent unnecessary use of resources. Especially in Android, where lifecycles are involved, canceling un-necessary background tasks is of paramount importance. Moreover, in the context of Android development, many jetpack libraries such as room and datastore, provide built-in coroutine scope. This is a big advantage because the users of these libraries do not need to think of when to start and stop the execution of coroutines. We just define what we have to execute and the library will take care of stopping/canceling/restarting the coroutines. This helps to utilize the resources in a very efficient manner.

What are dispatchers? Explain their types

A dispatcher allows us to specify which pool of threads the coroutines are executed. The dispatcher can be specified as a part of the coroutine context. There are 5 types of dispatchers that are available for use:

Default - The default dispatcher is used to schedule coroutines that perform CPU-intensive operations such as filtering a large list.

IO - This is the most common dispatcher. It is used to run coroutines that perform I/O operations such as making network requests and fetching data from the local database.

Main - The main dispatcher is used to execute coroutines on the main thread. It generally doesn't block the main thread, but, if several coroutines containing long-running operations get executed in the context of this dispatcher, then the main thread has a possibility of getting blocked. This is mainly used in conjunction with the withContext(){} method to switch the context of execution to the main thread. This comes in handy if it is required to perform some operation on a background thread and switching the context of execution to the main thread to update the UI. Since touching the UI from a background thread is not permitted in Android, this allows us to switch the context of execution to the main thread before updating the UI.

Unconfined - This dispatcher is very rarely used. Coroutines scheduled to run on this dispatcher run on the thread that they are started/resumed. When they are first called, they get executed in the thread that they are called in. When they resume, they get resumed in the thread that they are resumed in. In summary, they are not confined to any thread pool.

Immediate - The immediate dispatcher is a recently introduced dispatcher. It is used to reduce the cost of re-dispatching the coroutine to the main thread. There is a slight cost associated when switching the dispatcher using the withContext{} block. Using the Immediate dispatcher ensures the following.

If a coroutine is already executing in the main dispatcher, then the coroutine wouldn't be re-dispatched, therefore removing the cost associated with switching dispatchers.

If there is a queue of coroutines waiting to get executed in the main dispatcher, the immediate dispatcher ensures that the coroutine will be executed as immediately as possible.

What is the importance of a coroutine scope?

A coroutine scope manages the lifecycle of coroutines that are launched inside the scope. It determines when the coroutines inside the scope get started, stopped, and restarted. Coroutine scopes are especially helpful for the following reasons.

Allows the grouping of coroutines. If the scope gets canceled, then all coroutines that were started within that scope get canceled. This helps to prevent unnecessary use of resources when the coroutines are no longer needed.

Coroutine scope helps to define the context in which the coroutines are executed.

https://medium.com/@theAndroidDeveloper/5-common-kotlin-coroutines-interview-questions-f084d098f51d

What are coroutines?

Coroutines are a type of light-weight thread that can be used to improve the performance of concurrent code. Coroutines can be used to suspend and resume execution of code blocks, which can help to avoid the overhead of creating and destroying threads.

Can you explain the difference between a thread and a coroutine?

A thread is a unit of execution that can run independently from other threads. A coroutine is a light-weight thread that can be suspended and resumed.

How do threads compare with coroutines in terms of performance?

Threads are typically heavier than coroutines, so they can be more expensive in terms of performance. However, this is not always the case, and it really depends on the specific implementation. In general, coroutines tend to be more efficient when it comes to CPU usage, but threads may be better when it comes to I/O bound tasks.

Does Kotlin have any built-in support for concurrency? If yes, then what is it?

Yes, Kotlin has built-in support for concurrency via coroutines. Coroutines are light-weight threads that can be used to improve the performance of concurrent code.

Why are coroutines important?

Coroutines are important because they allow you to write asynchronous code that is more readable and easier to reason about than traditional asynchronous code. Coroutines also have the ability to suspend and resume execution, which can make your code more efficient.

What makes coroutines more efficient than threads?

Coroutines are more efficient than threads because they are lightweight and can be suspended and resumed without incurring the overhead of a context switch. This means that they can be used to perform tasks that would otherwise block a thread, without incurring the same performance penalty.

Can you explain what suspend functions are?

Suspend functions are functions that can be paused and resumed at a later time. This is useful for long-running tasks that might need to be interrupted, such as network requests. By using suspend functions, you can ensure that your code is more responsive and can avoid potential errors.

Are suspend functions executed by default on the main thread or some other one?

By default, suspend functions are executed on a background thread.

Is it possible to use coroutines outside Android development? If yes, then how?

Yes, it is possible to use coroutines outside of Android development. One way to do this is by using the kotlinx-coroutines-core library. This library provides support for coroutines on multiple platforms, including the JVM, JavaScript, and Native.

What's the difference between asynchronous code and concurrent code?

Asynchronous code is code that can run in the background without blocking the main thread. Concurrent code is code that can run in parallel with other code.

What happens if we call a suspend function from another suspend function?

When we call a suspend function from another suspend function, the first function will suspend execution until the second function completes. This can be used to our advantage to create asynchronous code that is easy to read and debug.

Can you give me an example of when you would want to run two tasks concurrently instead of sequentially?

There are a few reasons you might want to run two tasks concurrently instead of sequentially. One reason is if the tasks are independent of each other and can be run in parallel. Another reason is if one task is dependent on the other task and you want to avoid blocking the main thread. Finally, if you have a limited number of resources

available, you might want to run tasks concurrently in order to make better use of those resources.

What is the difference between launch() and async()? Which should be used in certain situations?

The main difference between launch() and async() is that launch() will create a new coroutine and start it immediately, while async() will create a new coroutine but will not start it until something calls await() on the resulting Deferred object. In general, launch() should be used when you want a coroutine to run in the background without blocking the main thread, while async() should be used when you need to wait for the result of a coroutine before continuing.

What is the best way to cancel a running job in Kotlin?

The best way to cancel a running job in Kotlin is to use the cancel() function. This function will cancel the job and any associated children jobs.

What do you understand about Job objects? How are they different from CoroutineScope?

Job objects are the basic building blocks of coroutines. They define a coroutine's lifecycle and provide a way to cancel it. CoroutineScope is used to define a scope for a coroutine, which determines its lifetime and other properties.

What happens if there's an exception thrown inside a coroutine?

If there's an exception thrown inside a coroutine, then the coroutine will be cancelled. All the coroutine's children will also be cancelled, and any pending work in those coroutines will be lost.

What does the {Dispatchers.Main} expression mean?

The {Dispatchers.Main} expression is used to specify that a particular coroutine should run on the main thread. This is important because some operations can only be performed on the main thread, and so specifying that a coroutine should run on the main thread ensures that it will be able to perform those operations.

What are some common mistakes made when working with coroutines in Kotlin?

One common mistake is not using the right context when launching a coroutine. This can lead to your coroutine being cancelled when it shouldn't be, or not being able to access the data it needs. Another mistake is not using a structured concurrency approach, which can lead to race conditions and other issues. Finally, not using the right tools for debugging can make it difficult to find and fix problems with your coroutines.

What are some good practices to follow when using Kotlin coroutines?

Some good practices to follow when using Kotlin coroutines include:

Use coroutines for short-lived background tasks
Use coroutines for tasks that can be executed in parallel
Use coroutines for tasks that need to be executed on a different thread than the UI thread
Do not use coroutines for tasks that need to be executed on the UI thread
Do not use coroutines for tasks that need to be executed synchronously

What is your opinion on the future of coroutines in Kotlin?

I believe that coroutines will continue to be a popular feature of Kotlin, as they offer a convenient and efficient way to manage concurrency. Additionally, the Kotlin team has been very supportive of coroutines and is constantly working to improve the experience of using them.

https://climbtheladder.com/kotlin-coroutines-interview-questions/

What's the best way to manage the coroutines lifecycle in Android ViewModel?

However, you should treat them as guidelines and adapt them to your requirements as needed.
Inject Dispatchers.
Suspend functions should be safe to call from the main thread.
The ViewModel should create coroutines.
Don't expose mutable types.
The data and business layer should expose suspend functions and Flows.

How do I use coroutines in ViewModel?

Add a CoroutineScope to your ViewModel by creating a new scope with a SupervisorJob that you cancel in the onCleared() method. The coroutines created with that scope will live as long as the ViewModel is being used.

What is the function used for launching for coroutines?

Launching a Coroutine as a Job

The launch {} function returns a Job object, on which we can block until all the instructions within the coroutine are complete, or else they throw an exception. The actual execution of a coroutine can also be postponed until we need it with a start argument. If we use CoroutineStart.

What is the difference between MainScope and GlobalScope?

MainScope is a CoroutineScope that uses the Dispatchers. Main dispatcher by default, which is bound to the main UI thread. The GlobalScope is a CoroutineScope that has

no dispatcher in its coroutine context.

What is the difference between GlobalScope and CoroutineScope?

Since it's alive along with application lifetime, GlobalScope is a singleton object. It's actually same as CoroutineScope but the syntax doesn't have CoroutineContext on its parameter. By default it will have default CoroutineContext which is Dispatchers.

What is the difference between GlobalScope async and launch?

The difference between async and launch is that async returns a value and launch doesn't. This is referring to the suspend lambda passed in, not the launch or async method itself. launch returns Job . async returns Deferred (which extends Job) which also has methods for getting the suspend lambda's result.

What is the difference between coroutineScope launch and runBlocking?

The main difference is that the runBlocking method blocks the current thread for waiting, while coroutineScope just suspends, releasing the underlying thread for other usages. Because of that difference, runBlocking is a regular function and coroutineScope is a suspending function.

What is the difference between launch and async coroutines?

launch: fire and forget. async: perform a task and return a result.

Launch vs Async in Kotlin Coroutines.

launch{} returns a Job and does not carry any resulting value.

async{} returns an instance of Deferred<T> , which has an await() function that returns the result of the coroutine.

https://developer.android.com/courses/quizzes/android-coroutines/use-coroutines-common-android-cases

Interceptor Interview Question For Android Kotlin

What is network interceptor?

Network interceptors are designed to not invoke cached responses and observe data between transmissions and also access to the Connection in the request.

What is the usage of interceptors?

Interceptors are another important component of application development. They are used to alter HTTP requests by including various functions. Interceptors are an excellent way to modify the HTTP request or response to make it more simple and

understandable.

Why HTTP interceptor is used?

Interceptors help us ensure to process all HTTP requests and responses before sending or getting the request, giving us the power to manage communication. We have several places or scenarios to use them: Logging and reporting progress. Adding headers to the request.

What is an interceptor in technology?

The Interceptor™ solution is a floating system that has been designed to concentrate all plastic waste before it is extracted from rivers. It consists of a 600-metre-long "floater" on the water surface and a 3.5-metre-deep screen to prevent microplastics from escaping underneath.

What is interceptor in Microservices?

An interceptor is added before or after the call to the process responsible for handling the declared logic within the application. This interceptor involves your code and will act as a security guardian.

What are interceptors in API?

Interceptor is an API gateway server built for accepting API requests from the client applications and routing them to the appropriate backend services. May it be a single service or multiple services to be called in a single API call, interceptor provides you with the necessary tools to control your API request flow.

Are interceptors and filters different?

Interceptors share a common API for the server and the client side. Whereas filters are primarily intended to manipulate request and response parameters like HTTP headers, URIs and/or HTTP methods, interceptors are intended to manipulate entities, via manipulating entity input/output streams.

What is interceptor for HTTP calls?

A request interceptor is a piece of code that gets activated for every HTTP request sent and received by an application. It's a layer in between clients and servers that modifies the request and responses, therefore, by intercepting the HTTP request, we can change the value of the request.

What does intercept HTTP mean?

Intercepting HTTP Requests means manipulating the HTTP requests made by the browser through the user or through asynchronous event handlers. With the help of web extension API, you can intercept any kind of web requests which are done in your

browser.

How HTTP can be intercepted?

To intercept HTTP requests, use the webRequest API. This API enables you to add listeners for various stages of making an HTTP request. In the listeners, you can: Get access to request headers and bodies and response headers.

What is interceptor in authentication?

The main purpose of the interceptor to capture and modify HTTP requests and responses. The interceptor can help with a variety of tasks: using in authorization processes by adding a token for the request, changing headers, modifying response from server, retrying failed requests, caching and many other common tasks.

What is interceptor and how it works?

Interceptors are used in conjunction with Java EE managed classes to allow developers to invoke interceptor methods on an associated target class, in conjunction with method invocations or lifecycle events. Common uses of interceptors are logging, auditing, and profiling.

What is interceptor in HttpClient?

HTTP protocol interceptor is a routine that implements a specific aspect of the HTTP protocol. Usually protocol interceptors are expected to act upon one specific header or a group of related headers of the incoming message or populate the outgoing message with one specific header or a group of related headers.

What is retrofit interceptor?

Retrofit is a commonly used type-safe REST client for android and java. Retrofit provides additional functionalities for the custom headers, multiparts request body, file upload and download, mocking responses and many more. Interceptors and authenticators are one of them.

https://square.github.io/okhttp/features/interceptors/
https://amitshekhar.me/blog/okhttp-interceptor

Dependency Injection Interview Question For Android Kotlin - Rz Rasel

Miscellaneous Useful Interview Question For Android Kotlin - Rz Rasel

What is mean by data class in kotlin android?

Data class is a simple class which is used to hold data/state and contains standard functionality. A data keyword is used to declare a class as a data class.

Why use data class in Kotlin?

A Kotlin Data Class is used to hold the data only and it does not provide any other functionality apart from holding data.

What is the difference between data class and class Kotlin?

A data class is a class that only contains state and does not perform any operation. The advantage of using data classes instead of regular classes is that Kotlin gives us an immense amount of self-generated code.

What are the two classes of data?

Data can be classified as qualitative and quantitative.

What are getters and setters in Kotlin data class?

In Kotlin, setter is used to set the value of any variable and getter is used to get the value. Getters and Setters are auto-generated in the code. Let's define a property 'name', in a class, 'Company'. The data type of 'name' is String and we shall initialize it with some default value.

What is the difference between VAR and VAL in Kotlin data class?

val is a keyword in Kotlin that allows us to define properties as read-only. Because they are read-only, they cannot be modified. var , on the other hand, is a keyword that can be used to declare properties in Kotlin that are mutable. These properties are not read-only (as with val) and they can be modified at will.

What is the difference between data class and POJO in Kotlin?

A data class is specified by the keyword data when declaring the class definition in Kotlin, it is like defining a pojo class in Java. The difference is that Kotlin will take care of all these getter and setter as well as equals and hashCode method for you.

What is a data object vs class?

Class is a user-defined datatype that has its own data members and member functions whereas an object is an instance of class by which we can access the data members and member functions of the class.

Can data class be open Kotlin?

To ensure consistency and meaningful behavior of the generated code, data classes have to fulfill the following requirements: The primary constructor needs to have at least one parameter. All primary constructor parameters need to be marked as val or var . Data classes cannot be abstract, open, sealed, or inner.

What is the singleton class in Android?

The Singleton Pattern is a software design pattern that restricts the instantiation of a class to just “one” instance. It is used in Android Applications when an item needs to be created just once and used across the board.

What is singleton class in Kotlin with example?

A singleton class is a class that is defined in such a way that only one instance of the class can be created and used everywhere. It is used where we need only one instance of the class like NetworkService, DatabaseService, etc.

How do you define a singleton in Kotlin?

Using Singletons in Kotlin. By using the keyword object in your app, you're defining a singleton. A singleton is a design pattern in which a given class has only one single instance inside the entire app.

How to create a singleton class in Kotlin Android?

Rules for making Singleton Class:
1) A private constructor.
2) A static reference of its class.
3) One static method.
4) Globally accessible object reference.
5) Consistency across multiple threads.

Why use singleton class in Android?

A singleton is a design pattern that restricts the instantiation of a class to only one instance. Notable uses include controlling concurrency and creating a central point of access for an application to access its data store.

What is the purpose of singleton class?

The Singleton's purpose is to control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields.

What is the difference between object and singleton in Kotlin?

In Kotlin, the singleton pattern is used as a replacement for static members and fields that don't exist in that programming language. A singleton is created by simply declaring an object . Contrary to a class , an object can't have any constructor, but init blocks are allowed if some initialization code is needed.

What is the difference between singleton and object Kotlin?

The singleton pattern restricts the instantiation of a class to a single object. It is useful in cases wherein you only need a single object to contain a global state. An object in Kotlin (among its many purposes) is both a class definition and an instantiation of single instance combined.

What are companion objects?

A companion object is an object that's declared in the same file as a class , and has the same name as the class. A companion object and its class can access each other's private members. A companion object's apply method lets you create new instances of a class without using the new keyword.

Why use companion objects?

Companion objects are beneficial for encapsulating things and they act as a bridge for writing functional and object oriented programming code. Using companion objects, the Scala programming code can be kept more concise as the static keyword need not be added to each and every attribute.

What is the difference between companion and object in Kotlin?

Companion objects are essentially the same as a standard object definition, only with a couple of additional features to make development easier. Companion objects allow their members to be accessed from inside the companion class without specifying the name.

What is the difference between object and companion object?

If we need to provide the Singleton behavior, then we are better off with Objects , else if we just want to add some static essence to our classes, we can use Companion objects .

What is the benefit of companion object Kotlin?

So why use companion objects? Because they provide a convenient shorthand for accessing “static” properties/functions. That's it. If, for some reason, you absolutely must have lazy initialization, then just use a regular object instead.

What is difference between singleton object and companion object?

In scala, when you have a class with same name as singleton object, it is called companion class and the singleton object is called companion object. The companion class and its companion object both must be defined in the same source file.

When should I use companion Kotlin?

Companion object is not a Singleton object or Pattern in Kotlin – It's primarily used to define class level variables and methods called static variables. This is common across all instances of the class.

What is Coroutines in Kotlin?

A coroutine is a concurrency design pattern that you can use on Android to simplify code that executes asynchronously. Coroutines were added to Kotlin in version 1.3 and are based on established concepts from other languages.

What's the difference between == and === operators in Kotlin?

Equality In Kotlin there are two types of equality: Structural equality (== - a check for equals()) Referential equality (=== - two references point to the same object)

What's the difference between lazy and Lateinit?

lateinit can only be used with a var property whereas lazy will always be used with val property. A lateinit property can be reinitialised again and again as per the use whereas the lazy property can only be initialised once.

How do I use companion object in Kotlin?

To create a companion object, you need to add the companion keyword in front of the object declaration. The output of the above code is " You are calling me :) " This is all about the companion object in Kotlin. Hope you liked the blog and will use the concept of companion in your Android application.

Why Kotlin removed static?

Be advised: Kotlin has removed Java-style statics to encourage more maintainable (dare I say 'better') coding practices. Static globals are generally against the OOP-paradigm but they can be quite convenient. Hence, Kotlin has provided us with companions, a more OOP-friendly way of having statics.

Can we create 2 object of singleton class?

The Singleton's purpose is to control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields.

Is Encapsulation a singleton?

A Singleton doesn't bring any encapsulation to the table that a class doesn't already have. If a class has poor encapsulation, it will still have poor encapsulation after it is turned into a Singleton. A Singleton restricts a class to a single instance, and that's all that it does.

What is lazy in Kotlin?

lazy in Kotlin is useful in a scenario when we want to create an object inside a class, but that object creation is expensive and that might lead to a delay in the creation of the object that is dependent on that expensive object.

Companion Object in Interface?

The companion object provides a common constant. It defines the minimum amount of wheels in a double-track vehicle. Additionally, we provide a helper method isDoubleTrack(). Common constants and helper methods are common use cases for using companion objects in interfaces.

How many types of coroutines are in Kotlin?

Dispatchers: Dispatchers help coroutines in deciding the thread on which the work has to be done. There are majorly three types of Dispatchers which are IO, Default, and Main.

What is the difference between coroutines and threads?

Coroutines are lighter than threads . Since they stack less . I.e coroutines don't have a dedicated stack . It means coroutine suspend execution by returning to the caller and the data that is required to resume execution is stored separately from the stack.

What is mutable and immutable in Kotlin?

Mutable – The contents of the list can be freely changed. Read-Only – The contents of the collection are not meant to be changed. However, the underlying data can be changed. Immutable – Nothing can change the contents of the collection.

Which is thread safe late init or lazy?

The Lazy initialization is thread-safe. Lateinit can only be used with var. Lazy initialization is used with the val property.

What is difference between Val and const in Kotlin?

The variables declared as const are initialized at the runtime. val deals with the immutable property of a class, that is, only read-only variables can be declared using val. val is initialized at the runtime. For val, the contents can be muted, whereas for const val, the contents cannot be muted.

What is scope function in Kotlin?

In Kotlin, there are five types of scope functions: let, with, run, apply, and also. As an Android mobile developer, you must understand this critical concept. The beauty of

Kotlin emerges from its unique features, which make it suitable for both frontend and backend development.

What is reflection in Kotlin?

In Kotlin, Reflection is a combination of language and library capabilities that allow you to introspect a program while it's running. Kotlin reflection is used at runtime to utilize a class and its members, such as properties, methods, and constructors.

Is Kotlin static or dynamic?

Statically typed language.
Being a statically typed language, Kotlin still has to interoperate with untyped or loosely typed environments, such as the JavaScript ecosystem.

Why all classes are final in Kotlin?

All Kotlin classes are final, so they cannot be inherited. To make a class inheritable, the keyword open needs to be present at the beginning of the class signature, which also makes them non-final.

Why singleton is stateless?

In these terms, singleton - scoped beans are usually stateless, because they are used by multiple clients simultaneosly and their states are not client-specific.

What are the three types of encapsulation?

There are three basic techniques to encapsulate data in Object Programming.
Types of Encapsulation in OOPs
Member Variable Encapsulation.
Function Encapsulation.
Class Encapsulation.

What is zip in Kotlin?

Returns a sequence of values built from the elements of this sequence and the other sequence with the same index. The resulting sequence ends as soon as the shortest input sequence ends. The operation is intermediate and stateless.

Are coroutines faster than threads?

You can compare the time taken to create many threads to the time taken to create many coroutines. This provides a practical way to compare coroutines to threads in the context of the often-quoted statement that coroutines are faster to create than threads.

Do coroutines run every frame?

Coroutines are updated every frame. Your loop is a counter loop. If each frame takes more time, then your loop will take longer to complete.

What is the purpose of coroutines?

A coroutine is a concurrency design pattern that you can use on Android to simplify code that executes asynchronously. Coroutines were added to Kotlin in version 1.3 and are based on established concepts from other languages.

Which thread is faster user or kernel?

User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed. User-level threads can be run on any operating system. There are no kernel mode privileges required for thread switching in user-level threads.

How many coroutines a thread can have?

A thread can only execute one coroutine at a time, so the framework is in charge of moving coroutines between threads as necessary.

What is a sealed class in Kotlin?

Sealed classes are used for representing restricted class hierarchies, when a value can have one of the types from a limited set, but cannot have any other type.

How do I use synchronized in Kotlin?

There are two types of synchronization available in Java (and Kotlin). Synchronized methods and synchronized statements. To use synchronized methods, the method needs to be denoted by the synchronized function modifier keyword in Java or @ Synchronized annotation in Kotlin.

Is String immutable in Kotlin?

Strings are immutable which means the length and elements cannot be changed after their creation. Unlike Java, Kotlin does not require a new keyword to instantiate an object of a String class.

What is null safety Kotlin?

Kotlin null safety is a procedure to eliminate the risk of null reference from the code. Kotlin compiler throws NullPointerException immediately if it found any null argument is passed without executing any other statements. Kotlin's type system is aimed to eliminate NullPointerException form the code.

What is the difference between list and Mutablelist in Kotlin?

In Kotlin, a list is an ordered collection of elements. Immutable lists and mutable lists are the two types of lists in Kotlin. Immutable Lists can not be modified, but mutable lists are editable. Immutable Lists are read-only lists and are created with the listOf() method.

Socket.IO Framework

What is Socket.IO?

Socket.IO is a JavaScript library that provides real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the client like mobile app, browser app, and a server-side library for Node.js. Socket.IO is event-driven and uses a publish/subscribe model.

What is WebSockets?

WebSocket is a communication protocol, providing full-duplex communication channels over a single TCP connection.

How does the Socket.IO framework work?

Socket.IO is a framework that allows for real-time, bi-directional communication between clients and servers. It works by using a JavaScript library on the client side to open a socket connection with the server. The server can then push data to the client, and the client can respond back. This makes it ideal for applications that need to update in real-time, such as chat applications or games.

Can you explain what the difference between WebSockets and Ajax polling is?

WebSockets provide a full-duplex connection between a client and a server, meaning that both sides can send and receive data at the same time. Ajax polling, on the other hand, is a technique used to simulate a full-duplex connection over a half-duplex connection. With Ajax polling, the client sends a request to the server, and the server then responds with any new data that it has. This process is then repeated at regular intervals, giving the illusion of a full-duplex connection.

What are some common use cases for Socket.IO?

Socket.IO is commonly used for real-time applications such as chat applications, gaming applications, and collaborative editing applications.

What do you know about of error handling in Socket.IO?

Error handling is an important part of any programming, and Socket.IO is no different. There are a few different ways to handle errors in Socket.IO, but the most common is to use the 'error' event. This event is fired whenever there is an error with a socket connection, and it will provide information about what went wrong. This information can then be used to debug the issue and prevent it from happening again in the

future.

What's your opinion on Socket.IO security issues?

There have been some security issues with Socket.IO in the past, but the developers have worked hard to address them. Overall, I believe that Socket.IO is a secure platform for real-time communication, but as with any technology there are always potential risks.

There is a lot of data being sent to a client using Socket.IO. How do we handle this scenario efficiently?

One way to handle a large amount of data being sent to a client using Socket.IO is to use pagination. This way, the client can request only the data that it needs, in small chunks, and the server can send only the data that the client has requested. This can help to improve efficiency and reduce bandwidth usage.

How can we avoid sending too much data over the network using Socket.IO?

One way to avoid sending too much data over the network is to compress the data before sending it. Another way is to use Socket.IO's built-in "rooms" feature to send data to only the clients that need it.

How do we make sure that messages aren't lost when using Socket.IO?

We can use the 'acknowledge' feature built into Socket.IO. This feature allows the server to confirm that a message has been received by the client. If the client does not receive a confirmation, it will assume that the message was lost and will resend it.

How can we limit the number of concurrent connections allowed by a server using Socket.IO?

We can limit the number of concurrent connections allowed by a server using Socket.IO by setting the 'maxConnections' option to a desired value. This will ensure that no more than the specified number of clients are able to connect to the server at any given time.

What do you understand about socket.io-redis?

Socket.IO-redis is a Redis adapter for Socket.IO. It allows you to use Redis as a back-end for your Socket.IO applications. This can be useful if you want to scale your Socket.IO applications across multiple servers.

What is clustering with Socket.IO?

Clustering with Socket.IO means that you can have multiple Socket.IO servers running behind a load balancer. This allows you to scale your Socket.IO application more easily.

What are rooms? How are they useful?

Rooms are a way of grouping socket.IO connections. This can be useful if you want to broadcast a message to all clients in a specific room. For example, you could have a chat application where each room represents a different chat room. Joining and leaving rooms is done with the join and leave methods on the socket object.

What are volatile events?

Volatile events are events that are not persisted by Socket.IO. This means that if a client disconnects and reconnects, they will not receive any events that were emitted after they disconnected. This can be useful for events that are not important enough to warrant being persisted, or for events that are too expensive to persist.

What is an ACK?

An ACK is a message sent by a Socket.IO client to confirm that it has received a message from the server. This is used to ensure that messages are delivered reliably.

What is the role of the ACK callback function?

The ACK callback function is used to confirm that a message has been received by the server. This is important in ensuring that messages are not lost in transmission.

Is it possible to broadcast messages to all connected clients from outside the context of a connection handler? If yes, then how?

Yes, it is possible to broadcast messages to all connected clients from outside the context of a connection handler. This can be done by using the socket.io.sockets.emit() function.

Is Socket.IO better than WebSocket?

WebSocket is a technology that enables two-way realtime communication between client and server. In contrast, Socket.IO is a library that provides an abstraction layer on top of WebSockets, making it easier to create realtime applications.

How does Android Socket work?

Two sockets communicate, one on the client-side and one on the server-side. A socket's address consists of an IP and a port. The server application starts to listen to clients over the defined port. The client establishes a connection over the IP of the server and the port it opens.

How to use Socket = io in Kotlin?

import io.socket.client.IO. import io.socket.client.Socket. ...
// The following lines connects the Android app to the server. SocketHandler.setSocket()

```
// The follwing line disconnects the Android app to the server.
val mSocket = SocketHandler.getSocket()
mSocket.emit("eventName", variable)
```

What is the difference between Android Socket.IO and WebSocket?

Key Differences between WebSocket and socket.io

It provides the Connection over TCP, while Socket.io is a library to abstract the WebSocket connections. WebSocket doesn't have fallback options, while Socket. io supports fallback. WebSocket is the technology, while Socket.io is a library for WebSockets.

Does Socket.IO reconnect automatically?

In the first case, the Socket will automatically try to reconnect, after a given delay.

Can Socket.IO run without web server?

Socket.io, and WebSockets in general, require an http server for the initial upgrade handshake. So even if you don't supply Socket.io with an http server it will create one for you.

What are the key disadvantages of SocketIO?

It's a library, Where WebSocket is a protocol SocketIO is a library and so have a dependency on those libraries doing what you need and working as expected. Client and Server, Both client and server need to have implemented the SocketIO libraries for it to work.

How do I keep my socket connection alive?

The SO_KEEPALIVE option for a socket is disabled (set to FALSE) by default. When this socket option is enabled, the TCP stack sends keep-alive packets when no data or acknowledgement packets have been received for the connection within an interval.

How long does a socket connection last?

However, the connection between a client and your WebSocket app closes when no traffic is sent between them for 60 seconds.

How do I keep my socket connection alive?

The SO_KEEPALIVE option for a socket is disabled (set to FALSE) by default. When this socket option is enabled, the TCP stack sends keep-alive packets when no data or acknowledgement packets have been received for the connection within an interval.

Is Socket.IO ID unique?

Each Socket in Socket.IO is identified by a random, unguessable, unique identifier Socket#id. For your convenience, each socket automatically joins a room identified by its own id.

Reactive Programming RxJava

What is Reactive Programming?

Reactive programming is programming with asynchronous data streams. Event buses or your typical click events are really an asynchronous event stream, on which you can observe and do some side effects. Reactive is that idea on steroids. You are able to create data streams of anything, not just from click and hover events. Streams are cheap and ubiquitous, anything can be a stream: variables, user inputs, properties, caches, data structures, etc. For example, imagine your Twitter feed would be a data stream in the same fashion that click events are. You can listen to that stream and react accordingly.

What is Stream?

A stream is a sequence of ongoing events ordered in time. It can emit three different things: a value (of some type), an error, or a "completed" signal.

Is RxJava Following The “Push” Or “Pull” Pattern?

In RxJava new data is being “pushed” to observers.

What’s The Difference Between OnNext(), OnComplete() And OnError()?

These are the callbacks an Observable / Flowable will receive. The first one is called for each emission of the Observable / Flowable (e.g. zero to infinity times). onComplete() and onError() are mutually exclusive – only ONE of them will be called at most once. In other words a stream cannot complete and error out at the same time.

How Many Times Can Each Of The OnNext(), OnComplete() And OnError() Called?

onNext() – from zero between infinite number of times
onComplete() – maximum once per stream
onError() – maximum once per stream

When Does An Observable Start Emitting Items?

There’s two types of Observables – Cold and Hot. Cold ones perform work (and subsequently emit items) only once someone is listening for it (e.g. someone has subscribed to them). Hot observables perform work and emit items regardless if there are any observers or not.

What’s The Difference Between A COLD And HOT Observables?

They start emitting items differently.

Cold observables are created multiple times and each instance can be triggered on it's own. Hot observables are like a "stream" of ongoing events – observers can come and go, but the stream is created ones and just goes on.

Can You Transform A COLD Observable To A HOT One And Vice-Versa?

One way to make a Cold observable Hot is by using publish().connect(). publish() converts the Cold observable to a ConnectableObservable, which pretty much behaves like a Hot one. Once triggered with the .connect() operator, it'll publish events regardless if there are any subscribers.

Another way to transform a Cold observable to a Hot one is by wrapping it with a Subject. The Subject subscribes to the Cold observable immediately and exposes itself as an Observable to future subscribers. Again, the work is performed regardless whether there are any subscribers … and on the other hand multiple subscribers to the Subject won't trigger the initial work multiple times.

What's A Scheduler? Why Does RxJava Use Schedulers?

By default RxJava is single-threaded – all operations are executed on a single thread. Schedulers are the means to switch the execution to a different thread. They're also an abstraction over the concept of "time", which is needed for time-sensitive operations (delay(), timeout(), buffer(), window(), etc).

https://veskoiliev.com/40-rxjava-interview-questions-and-answers/

Dependency Injection

What is Dependency Injection?

Dependency injection makes it easy to create loosely coupled components, which typically means that components consume functionality defined by interfaces without having any first-hand knowledge of which implementation classes are being used.

Dependency injection makes it easier to change the behaviour of an application by changing the components that implement the interfaces that define application features. It also results in components that are easier to isolate for unit testing.

What is Dagger?

Dagger is a dependency injection library for Android. It is used for injecting objects into other objects, such as activities and fragments. This allows for a more modular codebase, and can help to make code more testable and easier to maintain.

```
object AppConstant {
    const val baseUrl="https://jsonplaceholder.typicode.com/"
}

interface RetrofitApi {

    @GET("posts")
    suspend fun getPost() : List<Post>
}

object RetrofitBuilder {

    private val retrofit:Retrofit by lazy {
        Retrofit.Builder()
            .baseUrl(AppConstant.baseUrl)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
    }
    val retrofitApi:RetrofitApi by lazy {
        retrofit.create(RetrofitApi::class.java)
    }
}

class PostRepository {

    fun getPost(): Flow<List<Post>> = flow {
        val postList = RetrofitBuilder.retrofitApi.getPost()
        emit(postList)
    }.flowOn(Dispatchers.IO)
}

class PostViewModel(private val postRepository: PostRepository) : ViewModel() {

    val postData: MutableLiveData<List<Post>> = MutableLiveData()

    // Way: 1
    fun getPost() {
        viewModelScope.launch {
            postRepository.getPost()
                .catch { e ->
                    Log.d("main", "getPost: ${e.message}")
                }
                .collect { postData1 ->
                    postData.value = postData1
                }
        }
    }
```

```
    // Way: 2
    val postData1: LiveData<List<Post>> = liveData {
        postRepository.getPost()
            .catch { }
            .collect { postData ->
                emit(postData)
            }
    }

    // Way: 2
    val postData2: LiveData<List<Post>> = postRepository.getPost()
        .catch { }
        .asLiveData()
}

class PostViewModelFactory(private val postRepository: PostRepository)
    : ViewModelProvider.Factory {
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        return PostViewModel(postRepository) as T
    }
}

class MainActivity : AppCompatActivity() {
    private val TAG = "main"
    private lateinit var recyclerView: RecyclerView
    private lateinit var postAdapter: PostAdapter
    private lateinit var postViewModel: PostViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        initUi()

        val postViewModelFactory = PostViewModelFactory(PostRepository())
        postViewModel = ViewModelProvider(this, postViewModelFactory)
                        [PostViewModel::class.java]
        postViewModel.getPost()

         postViewModel.postData.observe(this, Observer {
             Log.d(TAG, "onCreate: ${it[0].body}")
             postAdapter.setPostData(it as ArrayList<Post>)
             progressBar.visibility = View.GONE
             recyclerView.visibility = View.VISIBLE
         })
    }
```

```
    private fun initUi() {
        recyclerView = findViewById(R.id.recyclerView)
        postAdapter = PostAdapter(this, ArrayList())
        recyclerView.apply {
            setHasFixedSize(true)
            layoutManager = LinearLayoutManager(this@MainActivity)
            adapter = postAdapter
        }
    }
}
```

-