Kotlin 中文开发者大会

# 给 Swifties/iOS 开发者的 KMP 指南

Pamela Hill

# Kotlin/Swift Interopedia

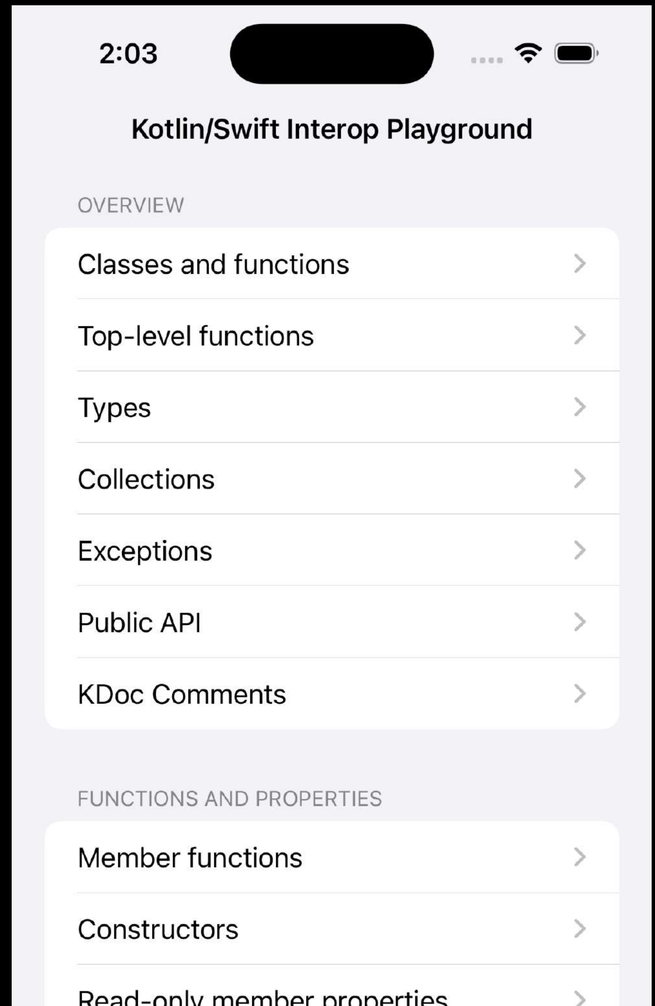
https://kotl.in/interopedia

## Overview

| | |
|---|---|
| Top-level functions | You can access a top-level function via the wrapper class: TopLevelFunctionKt.topLevelFunction(). |
| Exceptions | If you invoke a Kotlin function that throws an exception and doesn't declare it with `@Throws`, that crashes the app. Declared exceptions are converted to NSError and must be handled. |
| Public API | Public classes, functions, and properties are visible from Swift. Marking classes, functions, and properties internal will exclude them from the public API of the shared code, and they will not be visible in Swift. |
| KDoc comments | You can see certain KDoc comments at development time. In Xcode, use Option+Double left click to see the docs. Note that many KDocs features don't work in Xcode, like properties on constructors (@property) aren't visible. In Fleet, use the 'Show Documentation' action. |

## Functions and properties

| | |
|---|---|
| Member functions | You can call public member functions from Swift. Internal or private declarations aren't visible. |
| Constructor | You call constructors to create Kotlin classes from Swift. |



# Basics

## Classes, properties, functions

Kotlin

```kotlin
class UsualClassFunction {

    fun someFunction() {
    // do something
    }

}
```

Swift

```swift
let myClass = UsualClassFunction()
myClass.someFunction()
```

# Basics

## Top-level properties, functions

Kotlin

In TopLevelProperty.kt

```kotlin
val topLevelProperty = "A top-level property"
```

In TopLevelFunction.kt

```kotlin
fun topLevelFunction() {
    println("Hello from top-level function")
}
```

Swift

```swift
func topLevelPropertyExample() {
    print(TopLevelPropertyKt.topLevelProperty)
}
```

```swift
func topLevelFunctionExample() {
    TopLevelFunctionKt.topLevelFunction()
}
```

# Functions

## Default arguments

Kotlin

```kotlin
class FunctionWithDefaultArgumentsClass() {
    2 usages
    fun functionWithDefaultArgument(arg2: Int = 2) {
        println(arg2)
    }
}
```

Swift

```swift
let defaultArguments = FunctionWithDefaultArgumentsClass()
defaultArguments.functionWithDefaultArgument(arg2: 123)
```

# Default arguments with SKIE

Kotlin

```kotlin
class FunctionWithDefaultArgumentsClass() {
    @DefaultArgumentInterop.Enabled
    2 usages
    fun functionWithDefaultArgument(arg2: Int = 2) {
        println(arg2)
    }
}
```

Swift

```swift
let defaultArguments = FunctionWithDefaultArgumentsClass()
defaultArguments.functionWithDefaultArgument()
```

# Object-oriented programming

## Sealed interfaces/classes

Kotlin

```kotlin
sealed class SealedClass {
    object Object : SealedClass()

    class Simple(val param1: String) : SealedClass()

    data class Data(val param1: String, val param2: Boolean) : SealedClass()
}
```

Swift

```swift
func usingKotlinSealedClass(s: SealedClass) {
    switch s {
    case is SealedClass.Object:
        print("object")
    case is SealedClass.Simple:
        print("simple")
    case is SealedClass.Data:
        print("data")
    default:
        print("other")
    }
}
```

# Sealed interfaces and classes with SKIE

Kotlin

```kotlin
sealed class SealedClass {

    1 super
    object Object : SealedClass()


    1 super
    class Simple(val param1: String) : SealedClass()


    1 super
    data class Data(val param1: String, val param2: Boolean) : SealedClass()
}
```

Swift

```swift
func example(s: SealedClass) {
    switch onEnum(of: s) {
        case .object: print("object")
        case .simple(let simple): print("simple \(simple.param1)")
        case .data(let data): print("data \(data.param1) \(data.param2)")
    }
}
```

# Asynchronous programming

## Suspend functions

Kotlin

```kotlin
suspend fun getThingSimple(succeed: Boolean): Thing {
    delay(100.milliseconds)
    if (succeed) {
        return Thing("Thing")
    } else {
        error("oh no!")
    }
}
```

Swift

```swift
@MainActor
func suspendFunctionExample() {
    Task {
        do {
            let thing = try await ThingRepository().getThingSimple(succeed: true)
            print("Thing is \(thing).")
        }
        catch {
            print("Found error: \(error)")
        }
    }
}
```

# Asynchronous programming

## What happens here?

```
@MainActor
func suspendFunctionWithCancellationExample() {
    Task {
        do {
            let thing = try await ThingRepository().getThingSimple(succeed: true)
            print("Thing is \(thing).")
        }
        catch {
            print("Found error: \(error)")
        }
    }.cancel()
}
```

# KMP-NativeCoroutines

## Annotate suspend function (in Kotlin)

```kotlin
@NativeCoroutines
suspend fun getThingAnnotated(succeed: Boolean): Thing {
    delay(100.milliseconds)
    if (succeed) {
        return Thing("Thing")
    } else {
        error("oh no!")
    }
}
```

# KMP-NativeCoroutines

## Wrap function call (in Swift)

```swift
@MainActor
func suspendFunctionKMPNativeCoroutinesExample() {
    Task {
        do {
            let result = try await asyncFunction(for: ThingRepository().getThingAnnotated(succeed: true))
            print("Got result: \(result)")
        } catch {
            print("Failed with error: \(error)")
        }
    }
}
```

# KMP-NativeCoroutines

## Annotate flow function (in Kotlin)

```kotlin
@NativeCoroutines
fun getNumbersAnnotated(): Flow<Int> = flow {
    for (i in 1..10) {
        emit(i)
        delay(1.seconds)
    }
}
```

# KMP-NativeCoroutines

## Wrap function call (in Swift)

```swift
@MainActor
func flowKMPNativeCoroutinesExample() {
    Task {
        do {
            let sequence = asyncSequence(for: NumberFlowRepository().getNumbersAnnotated())
            for try await number in sequence {
                print("Got number: \(number)")
            }
        } catch {
            print("Failed with error: \(error)")
        }
    }
}
```

JET BRAINS Kotlin 中文开发者大会

# Have a nice Kotlin

@kotlin | Developed by JetBrains