

Делаем удобное взаимодействие с Kotlin из Swift



**Алексей
Михайлов**

IceRock Development

ICEROCK mobius

- О себе
- Kotlin Multiplatform Mobile
- Ограничения Swift <> Kotlin interop
- Как должен выглядеть Swift <> Kotlin interop
- Решения
 - Sourscery
 - KMP-NativeCoroutines
 - MOKO-KSwift



IceRock Development

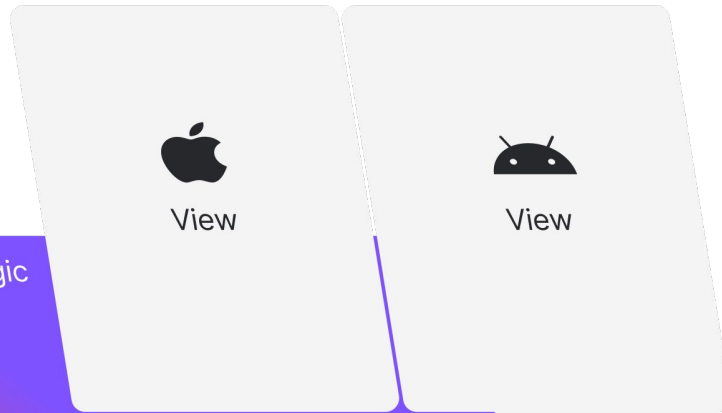
- Разработка проектов на заказ
 - android, ios, web
- 4+ лет с Kotlin Multiplatform
- 10+ проектов с Kotlin Multiplatform
- Open Source contributors



Алексей Михайлов

- 9 лет в разработке под мобилки
- Разработка под iOS и Android
- 5 лет CTO
- Продвигаю Kotlin Multiplatform в компании и за её пределами

Native Code



Shared Code

Business logic
and core

View



View

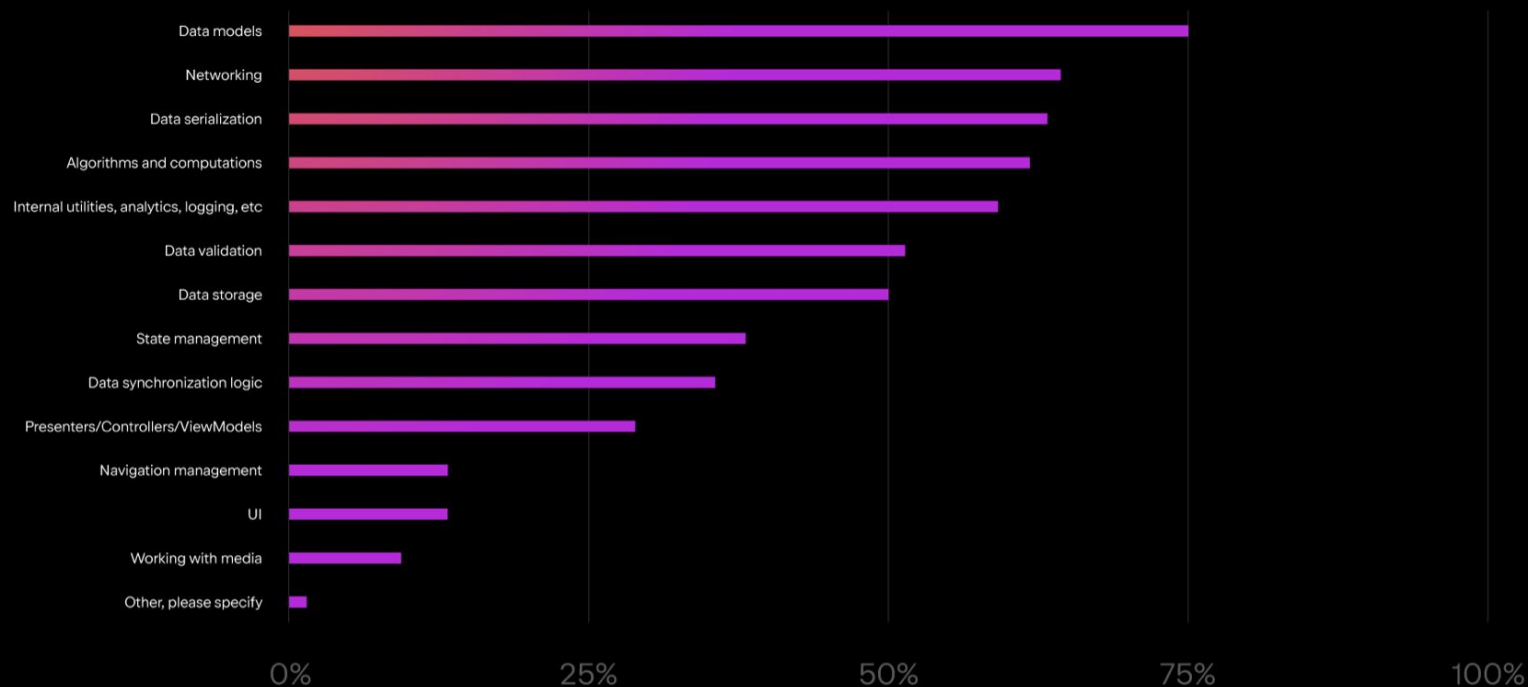
iOS
specific APIs

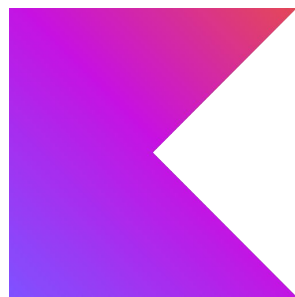
Android
specific APIs

- SDK для создания общего кода под Android и iOS
- Интеграция Swift <-> Kotlin
- UI остается полностью нативным

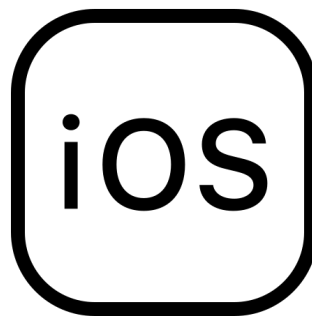


What parts of your code were you able to share between platforms?





Kotlin работаем с Kotlin
бесшовная интеграция



Swift работаем с Kotlin через
Objective-C bridge
есть ограничения в public api

**Все ограничения накладываются
только на Kotlin public API**

internal и private код не имеет ограничений

B Swift hem sealed class/interface

sealed class/interface



```
sealed interface UIState<out T> {  
    object Loading : UIState<Nothing>  
    object Empty : UIState<Nothing>  
    data class Data<T>(val value: T) : UIState<T>  
    data class Error(val throwable: Throwable) : UIState<Nothing>  
}
```



```
public protocol UIState { }  
public class UIStateData<T> : KotlinBase, UIState where T : AnyObject {  
    open var value: T? { get }  
}  
public class UIStateEmpty : KotlinBase, UIState { }  
public class UIStateError : KotlinBase, UIState {  
    open var throwable: KotlinThrowable { get }  
}  
public class UIStateLoading : KotlinBase, UIState { }
```

Kotlin/Native не умеет объявлять
Objective C extension

extensions



```
fun UILabel.fillByKotlin() { ... }
```



```
class UILabelExt {  
    class func fillByKotlin(_ receiver: UILabel)  
}
```

extensions

```
interface SomeInterface {  
    fun doSomeJob(arg: Int)  
}  
fun SomeInterface.doSomeJob() = doSomeJob(arg = 1)
```



```
protocol SomeInterface {  
    func doSomeJob(arg: Int32)  
}  
class SomeInterfaceKt {  
    class func doSomeJob(_ receiver: SomeInterface)  
}
```

В Objective C нем generic protocol / func

Поэтомум из Swift у Kotlin може нем generic interface / func

generic interface / func

```
interface Flow<out T> {  
    suspend fun collect(collector: FlowCollector<T>)  
}
```



```
protocol Flow {  
    func collect(collector: FlowCollector, completionHandler: @escaping  
    func collect(collector: FlowCollector) async throws -> KotlinUnit  
}
```

В Swift нем covariant u contravariant generic'ов

in/out generics



```
class InOutTester {  
    fun inFun(inClass: GenInClass<Number>)  
    fun outFun(outClass: GenOutClass<Number>)  
}
```



```
class InOutTester {  
    func inFun(inClass: GenInClass<AnyObject>)  
    func outFun(outClass: GenOutClass<AnyObject>)  
}
```

B Swift hem abstract class

Нем abstract class в Swift



```
class NormalClass {  
    fun someNotAbstractFun() {}  
}
```

```
abstract class AbstractClass {  
    abstract fun shouldImplemented()  
}
```



```
public class NormalClass : KotlinBase {  
    open fun someNotAbstractFun()  
}
```

```
open class AbstractClass : KotlinBase {  
    open fun shouldImplemented()  
}
```

Нем abstract class в Swift



```
class NormalClass {  
    fun someNotAbstractFun() {}  
}
```

```
abstract class AbstractClass {  
    abstract fun shouldImplemented()  
}
```



```
public class NormalClass : KotlinBase {  
    open func someNotAbstractFun()  
}
```

```
open class AbstractClass : KotlinBase {  
    open func shouldImplemented()  
}
```

Нем abstract class в Swift



```
class NormalClass {  
    fun someNotAbstractFun() {}  
}  
  
abstract class AbstractClass {  
    abstract fun shouldImplemented()  
}
```



```
public class NormalClass : KotlinBase {  
    open fun someNotAbstractFun()  
}  
  
open class AbstractClass : KotlinBase {  
    open fun shouldImplemented()  
}
```

В Objective C нет default аргументов

extensions



```
class MyDefaultClass(arg: Int = 1)
```



```
class MyDefaultClass {  
    init(arg: Int32)  
}
```

Проблемные API

- sealed class/interface
- extension к нативным типам и interface
- default args
- generic типы в interface и функциях
- covariant, contravariant generics
- abstract class
- а также есть [ряд мелочей](#) описанных командой HeadHunter

Kotlin-Swift interopedia

Kotlin-Swift interopedia

maven-central v1.7.0 License MIT

Мы создали этот репозиторий, чтобы помочь разработчикам, интересующимся технологией KMM, понять, как будет выглядеть описанное ими публичное API общего модуля.






В сети есть [подробная документация от JetBrains](#) про интероп между Kotlin и Swift, однако в ней не рассматриваются подробно все возможности языка Kotlin.

Поэтому мы свели в единую табличку перечень возможностей Kotlin-а и отметили, какими возможностями можно пользоваться без каких-либо проблем, а с какими потребуются те или иные доработки.

Таблица интеропа

Как пользоваться таблицей:

- Знак  означает, что указанной фишой Kotlin-а можно спокойно пользоваться, она генерирует Swift-friendly код без необходимости доработок;
- Знак  означает, что либо для работы фиши требуются какие-то доработки, либо есть несоответствие между ожидаемым и реальным поведением фиши, которое не сильно мешает разработке;
- Знак  означает, что фишой нельзя пользоваться, она генерирует не Swift-friendly код, она работает совсем не так как ожидается, и это может стать помехой в разработке.

Фича Kotlin-а	Ожидание	Статус	Комментарий
Common			
Internal modifier	Internal-функции и классы не видны в Swift		Так и есть, с iOS-разработчиками придётся обсуждать только публичное API общего кода
JavaDoc comments	Комментарии видны в XCode		Комментарии видны, если добавить специальный аргумент для компилятора
Data types			
Primitive types	Типы, объявленные в Kotlin, можно без изменений использовать в Swift		Может требоваться маппинг для целочисленных типов данных / маппинги для Char-а
Optional (nullable) primitive types	Тип, объявленный как Nullable, является таковым и на стороне Swift / Пользоваться nullable-типами можно без изменений		Для примитивных типов требуется маппинг в специальные optional-типы / особенности с Char?
Mutable, immutable collections	Сигнатуры List / MutableList / etc имеют значение в Swift-мире и тоже регулируют мутабельность ; Использование коллекций не отличается от Kotlin		Для регулировки мутабельности используются ключевые слова let, var / Для мутабельных коллекций требуются дополнительные маппинги



Хотелось чтоб было лучше...

sealed class/interface



```
sealed interface UIState<out T> {  
    object Loading : UIState<Nothing>  
    object Empty : UIState<Nothing>  
    data class Data<T>(val value: T) : UIState<T>  
    data class Error(val throwable: Throwable) : UIState<Nothing>  
}
```



```
enum UIState<T> {  
    case loading  
    case empty  
    case Data(let value: T)  
    case Error(let error: Error)  
}
```

Хотелось бы
—
extensions



```
fun UILabel.fillByKotlin() { ... }
```



```
extension UILabel {  
    func fillByKotlin() { ... }  
}
```

Хотелось бы

—
default args



```
fun withDefault(someArg: Int = 1) { ... }
```



```
func withDefault(someArg: Int) { ... }
```

```
func withDefault() {  
    withDefault(someArg: 1)  
}
```

generic interface



```
interface Flow<T> {  
    suspend fun collect(collector: FlowCollector<T>)  
}
```



```
protocol Flow {  
    associatedtype T  
    func collect(collector: FlowCollector<T>)  
}
```

С чем можно смириться

- generic типы в функциях
- covariant, contravariant generics
- abstract class
- и другие мелочи

Решение одно на все - не использовать это в публичном API

Как достичь желаемого...

Кодогенерація



krzysztofzablocki/ **Sourcery**

Meta-programming for Swift, stop writing boilerplate code.



 124

Contributors

 51

Issues

 7k

Stars

 512

Forks



- <https://luisramos.dev/marrying-kmm-and-swift-with-sourcery>
- Генерируем Swift код на основе анализа всего доступного в Swift API



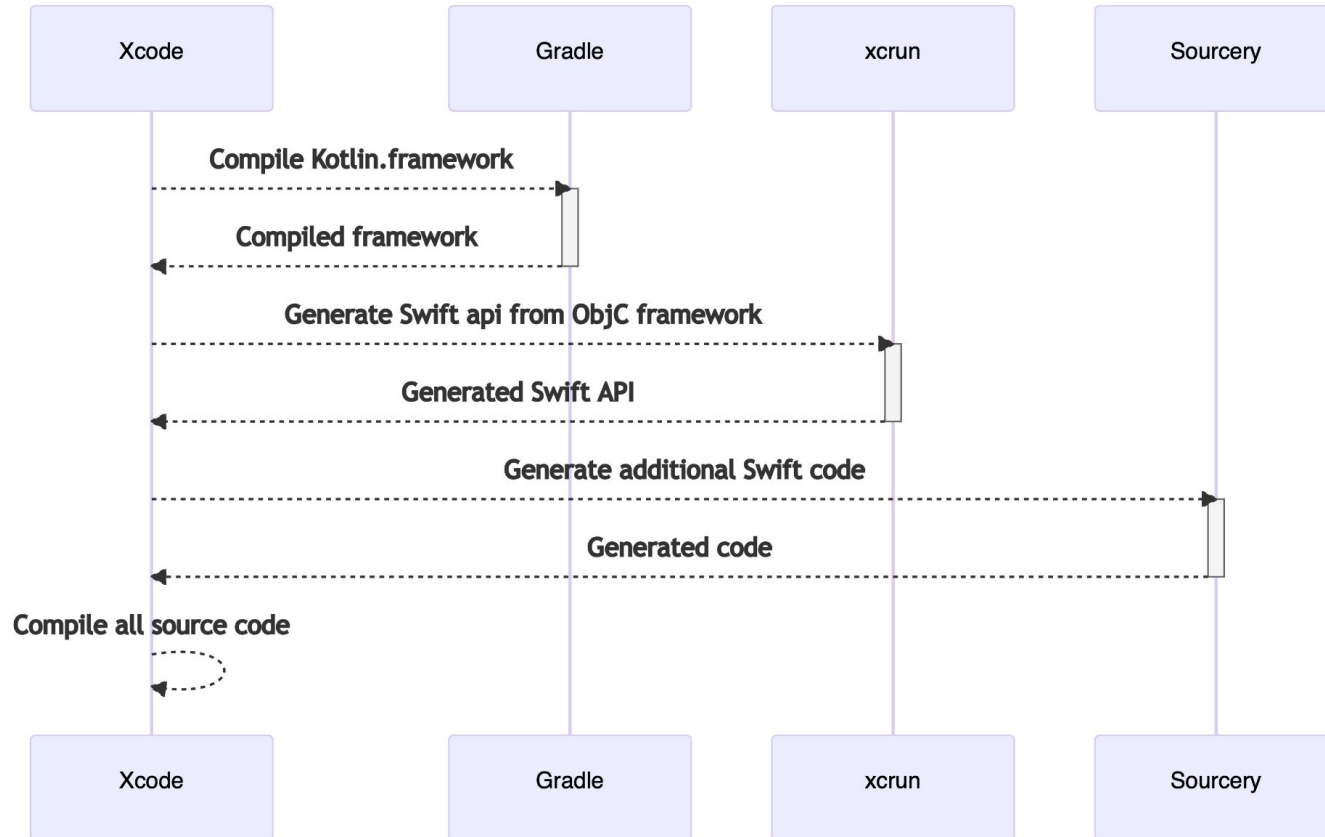
```
// file name is Rocket.kt  
fun Rocket.launch(): Boolean
```

```
// Swift generate code  
import shared
```



```
extension Rocket {  
    func launch() -> Bool {  
        RocketKt.launch(self)  
    }  
}
```

Sourcery



```
echo "import shared\n:type lookup shared" \  
| xcrun --sdk macosx swift -F. \  
| tail -n+2 > \  
| ./Shared.swift
```

- Берет shared фреймворк
- Ищет все типы относящиеся к этому фреймворку
- Выводит все эти типы в виде swift кода
- Записывает в файл Shared.swift

- Уже имея сгенерированное Swift API от shared.framework используем Sourcery также, как для типичного Swift проекта
- <https://github.com/krzysztofzablocki/Sourcery>
- Гайдов, статей и видео по этому инструменту хватает в сети (7K звезд на GitHub)
- Единственный минус подхода - нет информации о оригинальном Kotlin API
 - что не дает генерировать например enum'ы для sealed классов

Kotlin compiler plugin

Kotlin compiler plugin

Подойдет **частично** - для генерации **нового Kotlin кода**,
но не для генерации Swift

rickclephas/**KMP- NativeCoroutines**

Library to use Kotlin Coroutines from Swift code in
KMP apps



 3

Contributors

 9

Issues

 317

Stars

 15

Forks



KMP-NativeCoroutines

- <https://github.com/rickclephas/KMP-NativeCoroutines>
- Для интерфейсов Flow, StateFlow генерирует классы с generic типом
- Дополнительно генерирует свойства с новым сгенерированным типом там, где использовались свойства с интерфейсами

```
class Clock {  
    // Somewhere in your Kotlin code you define a Flow property  
    val time: StateFlow<Long> // This can be any kind of Flow  
  
    // The plugin will generate this native property for you  
    val timeNative  
        get() = time.asNativeFlow()  
}
```

KMP-NativeCoroutines

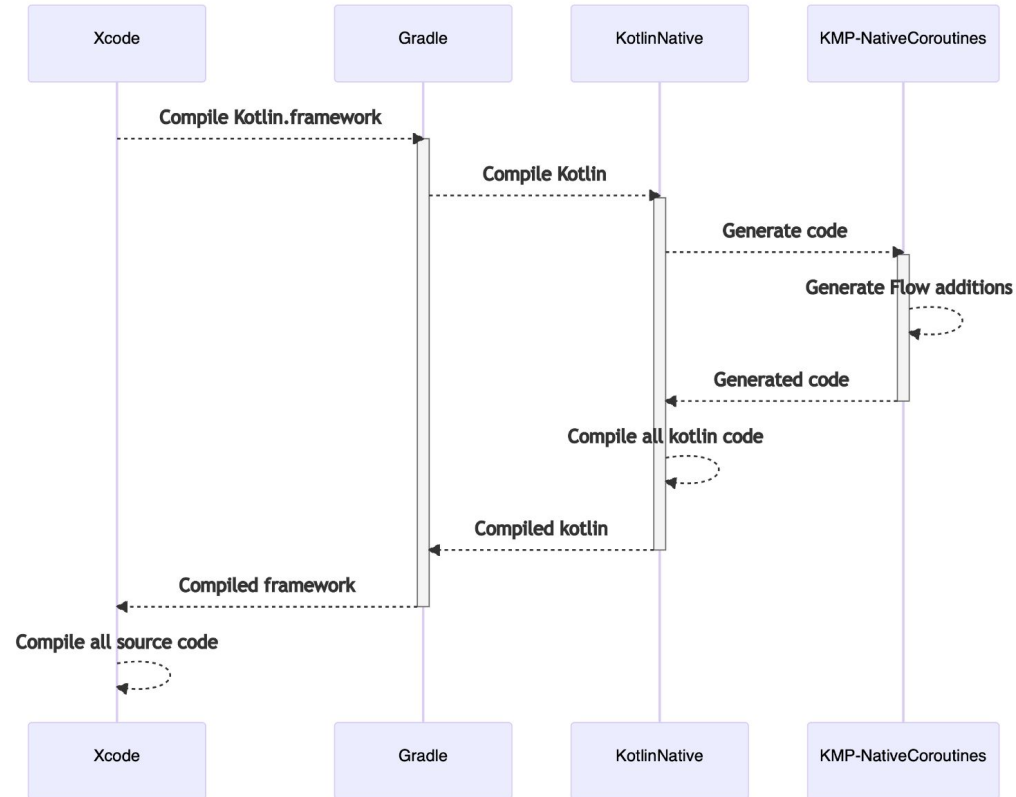
```
// Create an AnyPublisher for your flow
let publisher = createPublisher(for: clock.timeNative)

// Now use this publisher as you would any other
let cancellable = publisher.sink { completion in
    print("Received completion: \(completion)")
} receiveValue: { value in
    print("Received value: \(value)")
}

// To cancel the flow (collection) just cancel the publisher
cancellable.cancel()
```



KMP-NativeCoroutines



KMP-NativeCoroutines

- Решает проблему generic интерфейсов в coroutine
- Можно взять за основу для других интерфейсов (если потребуется)
- Минус - не может генерировать Swift код, а значит мы также ограничены возможностями Objective C

Gradle plugin

Gradle plugin

Подойдет для генерации любого кода на основе Kotlin

icerockdev/moko- kswift



Swift-friendly api generator for Kotlin/Native frameworks

 2

Contributors

 14

Issues

 164

Stars

 9

Forks



MOKO KSwift

- <https://github.com/icerockdev/moko-kswift>
- Читаем Kotlin Intermediate Representation (IR) из klib'ов и генерируем на основе него Swift код
- Имеет расширяемую структуру - можно добавлять свои генераторы
- Встроено два генератора:
 - Swift enum из sealed interface/class
 - Swift extension для расширений к платформенным типам или интерфейсам

sealed interface / class

```
public protocol UIState { }
public class UIStateData<T> : KotlinBase, UIState where T : AnyObject {
    open var value: T? { get }
}
public class UIStateEmpty : KotlinBase, UIState { }
public class UIStateError : KotlinBase, UIState {
    open var throwable: KotlinThrowable { get }
}
public class UIStateLoading : KotlinBase, UIState { }
```

```
public enum UIStateKs<T : AnyObject> {
    case loading
    case empty
    case data(UIStateData<T>)
    case error(UIStateError)

    var sealed: UIState

    public init(_ obj: UIState)
}
```

Kotlin/Native

KSwift

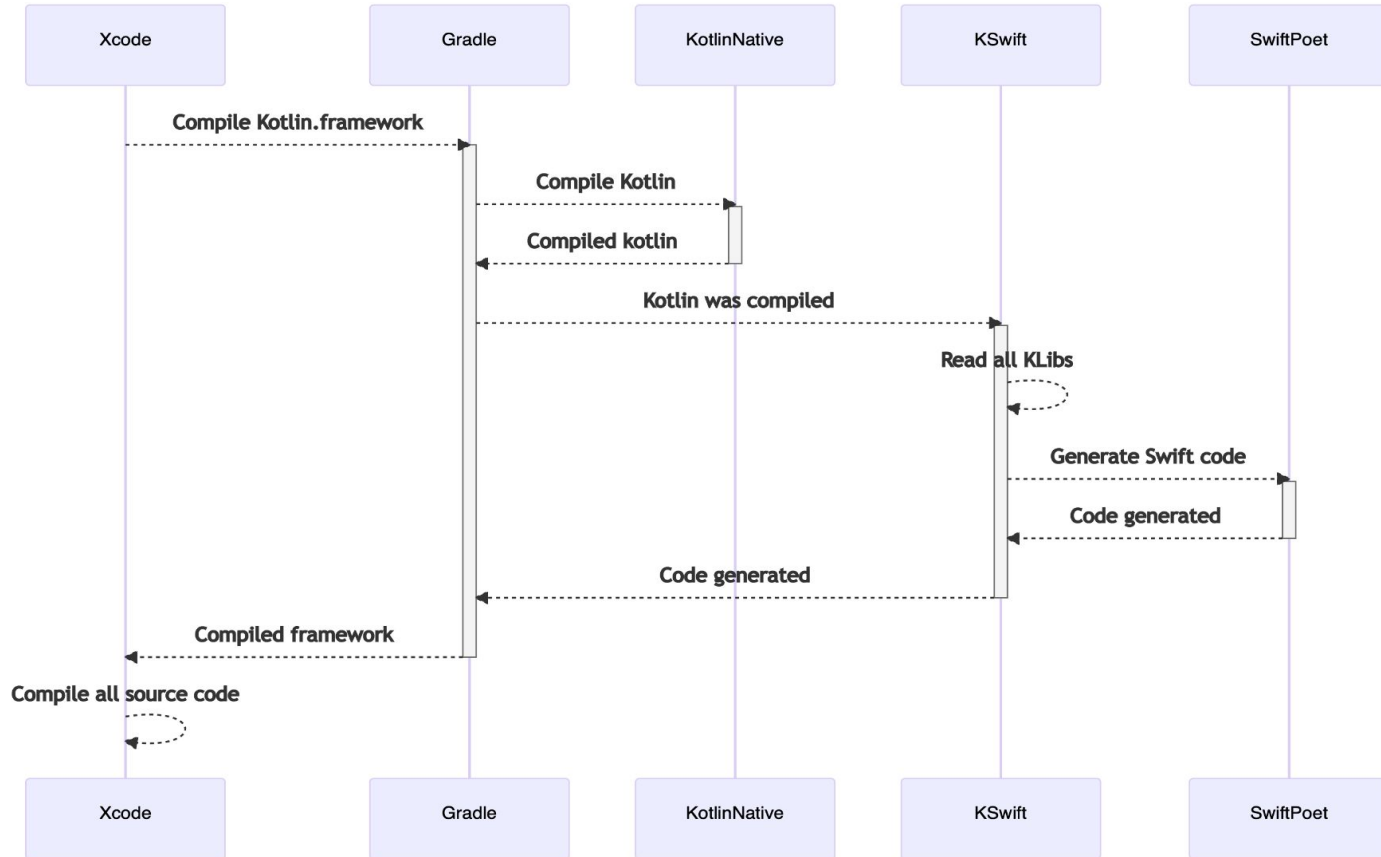
extensions on platform / interfaces

Kotlin/Native

```
class UILabelExt {  
    class func fillByKotlin(_ receiver: UILabel)  
}
```

KSwift

```
public extension UIKit.UILabel {  
    @discardableResult  
    public func fillByKotlin() {  
        return UILabelExtKt.fillByKotlin(self)  
    }  
}
```



Чтение klib

- Kotlin/Native компилятор работает только с klib форматом
- Kotlin код компилируется в klib для конкретного таргета (iosArm64, ...)
- Из множества klib'ов компилируется framework / application для iOS
- Каждый klib содержит IR
 - классы, свойства, методы
 - generic'и включая пометки in / out
 - пометки suspend и m.g.
 - extension функции
- Используется официальная библиотека JetBrains чтения klib
 - `org.jetbrains.kotlinx:kotlinx-metadata-klib`

Чмо макое IR

```
@SSA
fun f(x: Boolean): Int {
    val a = if (x) {
        A(5)
    } else {
        A(6)
    }
    a.print()
    return 0
}
```



Что такое IR

```

FUN name:f visibility:public modality:FINAL <>
(x:kotlin.Boolean) returnType:kotlin.Int
  annotations:
    SSA
    VALUE_PARAMETER name:x index:0 type:kotlin.Boolean
    BLOCK_BODY
    VAR name:a type:<root>.A [val]
    WHEN type=<root>.A origin=IF
    BRANCH
      if: GET_VAR 'x: kotlin.Boolean'
type=kotlin.Boolean
      then: BLOCK type=<root>.A
        CONSTRUCTOR_CALL 'public constructor <init>'
(arg: kotlin.Int) [primary]' type=<root>.A
        arg: CONST Int type=kotlin.Int value=5
    BRANCH
      if: CONST Boolean type=kotlin.Boolean value=true
      then: BLOCK type=<root>.A origin=null
        CONSTRUCTOR_CALL 'public constructor <init>'
(arg: kotlin.Int) [primary]' type=<root>.A
        arg: CONST Int type=kotlin.Int value=6
    CALL 'public final fun print (): kotlin.Unit'
type=kotlin.Unit
    $this: GET_VAR 'val a: <root>.A [val]' type=<root>.A
    RETURN type=kotlin.Nothing
    CONST Int type=kotlin.Int value=0
  
```



полный IR

Чмо макое IR

```
FUN name:f visibility:public modality:FINAL <>  
(x:kotlin.Boolean) returnType:kotlin.Int  
  annotations:  
    SSA  
  VALUE_PARAMETER name:x index:0 type:kotlin.Boolean
```



Что такое IR

Kotlin IR: прошлое, настоящее и будущее

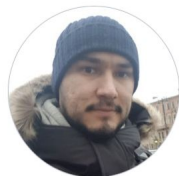
📅 День 4 🕒 20:00 🌐 RU 🍹 ☑️

В 1.5 новый JVM бэкенд - JVM_IR - стал бэкендом по умолчанию. В 1.5.30 новый JS бэкенд - JS_IR - перешел в бету. Ильмир расскажет, зачем в компании стали разрабатывать новые бэкенды, почему решили переписать половину компилятора, с какими проблемами столкнулись и какие дальнейшие планы.

#jetbrains #android #компилятор #buildspeed

📄 [Скачать презентацию](#)

Спикеры



Ильмир Усманов

JetBrains



Зачем нам IR

- Зная полную информацию о Kotlin коде в публичном API мы можем сгенерировать Swift код, который будет упрощать использование оригинального Kotlin кода

Какие есть проблемы?

- Kotlin/Native при линковке всех klib'ов в framework / application можем изменить итоговые имена - добавить "_" для решения конфликтов (в swift нем пакетов, которые в Kotlin решают эту задачу)
 - Возможно сможем дотянуться до логики подбора имен и правильные имена генерировать в Swift - [IrBasedKotlinManglerImpl](#)
 - Либо использовать подход как в Sourscery для анализа сгенерированного Swift кода

- 1 Для работы с Flow из Swift - KMP-NativeCoroutines
- 2 Для использования sealed class/interface как enum в Swift - MOKO KSwift
- 3 Для использования extension'ов к платформенным классам - MOKO KSwift
- 4 Свой шаблон - Sourscery (если не нужна информация о Kotlin коде) или MOKO KSwift
- 5 И некоторые возможности Kotlin не использовать в public API

Всем спасибо



Ссылки на презентацию и
материалы