KMP 跨端 UI 框架浅析

@rosu_h

KotlinConf'24

Shenzhen

About me

rosu

- 全职 Android 开发者，从业 6 年
- 领域：直播音视频、KMP 跨端能力、AI
- 开源项目：简单水印 (1.5k 🌟)、AndroidFilePicker (900+🌟)
- 独立开发：AICommit

KotlinConf

今天关注哪些内容？

- 移动跨端框架原理对比
- 跨端 UI 框架的关键因素
- 为什么选择 Kotlin Multiplatform？
- 从 0 开始，使用 KMP 构建跨平台 UI
- Q & A

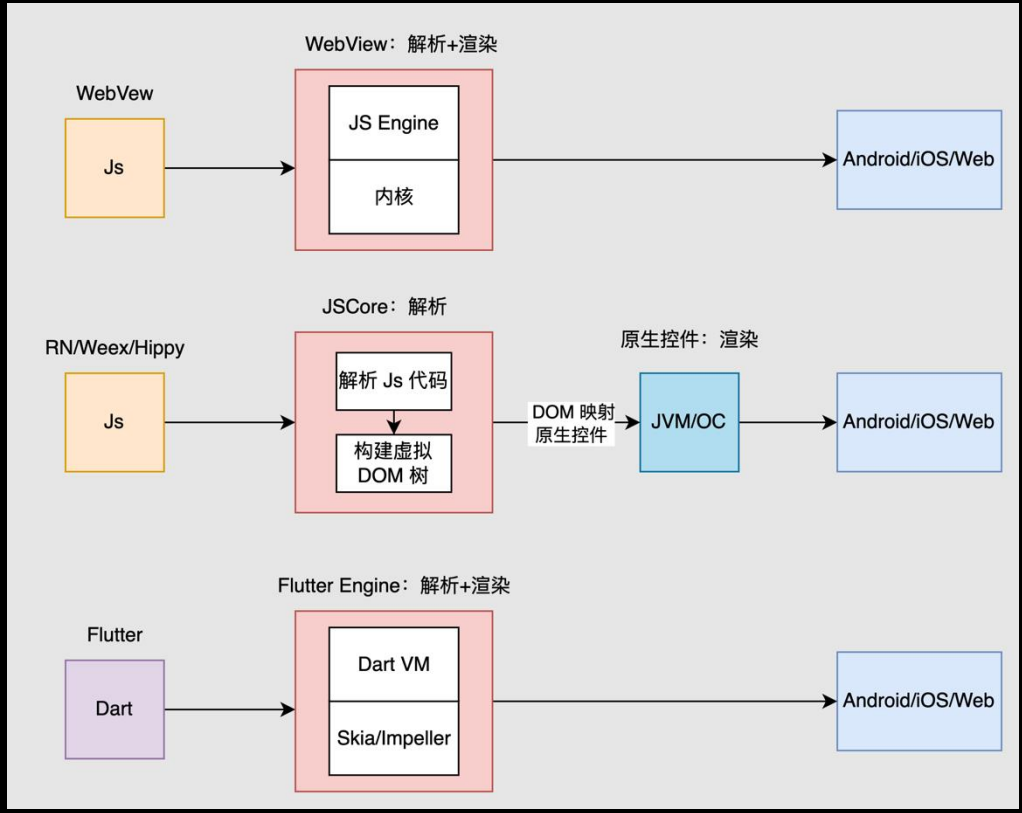
移动跨端框架原理对比

框架	Web	RN/Weex/Hippy	Flutter
原理	前端技术栈 + WebView	Js 引擎解释执行 + 原生控件映射	Dart + 自绘制
技术栈	Web 前端技术栈	Web 前端技术栈	Dart
性能	很差	略差	略差
兼容性	良好	良好	一般
粒度	页面	控件	APP

注：RN/Weex/Hippy 在具体工程实践中存在诸多不同，此处仅比较技术架构

跨端 UI 框架的关键因素

- 开发者编写代码所使用的语言
- 真正执行渲染所运行的环境（或语言）



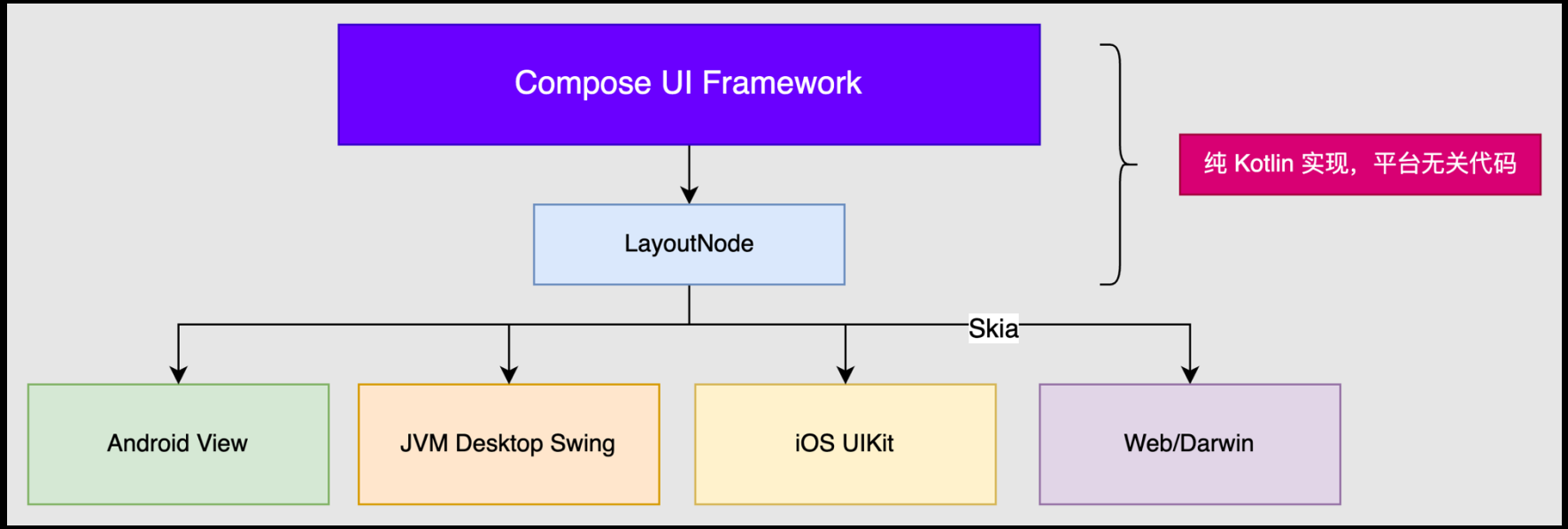
为什么选择 Kotin Multiplatform

1. **Kotlin Compiler** 已经帮我们适配目标平台代码

2. 运行时不再需要一个中间层解释执行

Chromium 之于 Web
JsCore 之于 RN/Weex

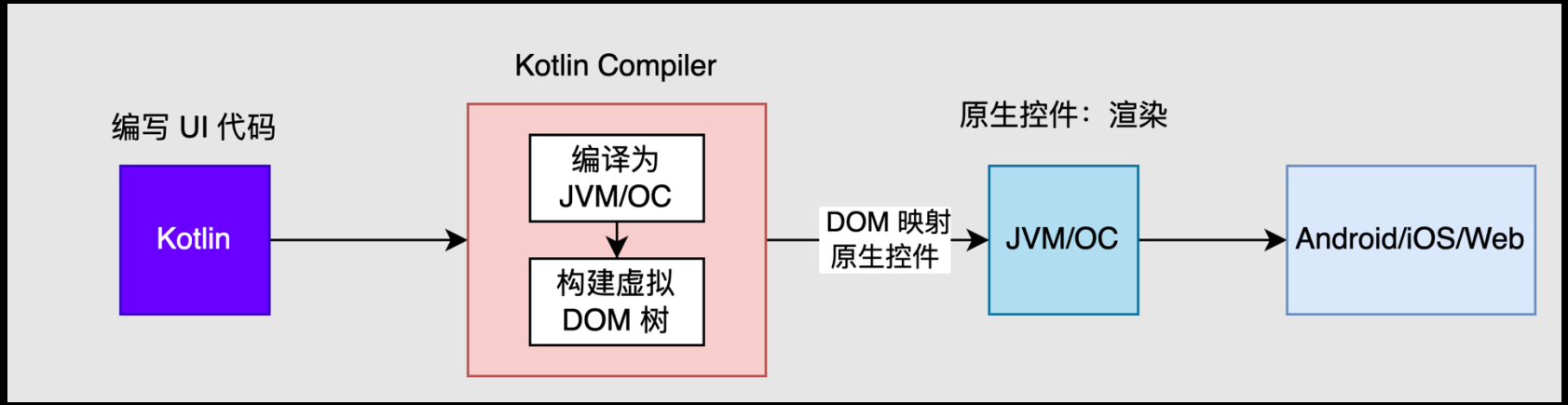
Compose Multiplatform 是如何工作的？



使用 KMP 构建跨平台 UI

KMP UI 工作流程

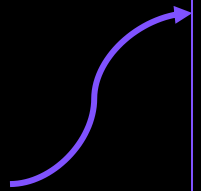
Kotlin 编写代码，编译为目标平台代码（JVM/OC），在目标平台原生执行



1. 描述 UI 的基础控件

```kotlin
/**
 * 布局组件：最小单元
 */
abstract class Component {
    /**
     * 生成平台无关的组件树，类似于DOM
     */
    abstract fun dumpTree(): Node
}

/**
 * 容器组件：包含多个子组件
 */
abstract class Container : Component() {
    abstract fun addView(component: Component)
}
```



```kotlin
/**
 * 组件节点信息，于平台无关，纯 Kotlin 代码
 */
data class Node(
    val nodeName: String,
    val attributes: Map<String, String>,
    val left: Int = 0, // margin left
    val top: Int = 0, // margin top
    val right: Int = 0, // margin right
    val bottom: Int = 0, // margin bottom
    val width: Int = -2, // wrap_content
    val height: Int = -2, // wrap_content
    val child: List<Node> = emptyList()
)
```

2. 实现 Text 和容器

```kotlin
class Text(
    private val parent: Container?,
    private val init: TextAttributes.() -> Unit
) : Component() {
    class TextAttributes {
        var text: String = ""
        var textSize: String = "16"
        var width = -2
        var height = -2
    }

    override fun dumpTree(): Node {
        val attributes = TextAttributes().apply(init)
        return Node(
            nodeName: "Text",
            mapOf("text" to attributes.text, "textSize" to attributes.textSize),
            width = attributes.width,
            height = attributes.height
        )
    }
}
```

```kotlin
/**
 * 垂直布局组件
 */
class Column(
    val parent: Container?,
    init: Column.() -> Unit
) : Container() {
    private val children = mutableListOf<Component>()

    init {
        init()
    }

    override fun addView(component: Component) {
        children.add(component)
    }

    override fun dumpTree(): Node {
        val child = children.map { it.dumpTree() }
        return Node( nodeName: "Column", emptyMap(), child = child)
    }
}
```

3. Android Native 实现容器并解析规范

Android 实现：容器接管解析逻辑，并且对接 View 的原生控件

```
class KMPUIView(context: Context) : FrameLayout(context) {
    init {
        val node = HelloScreen().dumpTree()
        when (node.nodeName) {
            "Text" -> {
                val textView = TextView(context)
                textView.text = node.attributes["text"]
                textView.textSize = node.attributes["textSize"]?.toFloat() ?: 16f
                addView(textView)
            }
        }
    }
}
```

4. iOS Native 实现容器并解析规范

iOS 实现：容器接管解析逻辑，并且对接 Swift UI 的原生控件

```
struct NodeView: View {
    let node: Node

    var body: some View {
        switch node.nodeName {
        case "Text":
            Text(node.attributes["text"] ?? "")
                .font(.system(size: CGFloat((Float(node.attributes["textSize"] ?? "16") ?? 16.0))))
        case "Column":
            VStack {
                ForEach(node.child, id: \.nodeName) { childNode in
                    NodeView(node: childNode)
                }
            }
        default:
            EmptyView()
        }
    }
}
```

KMP UI Demo 展示

总结

KMP 本身

- 基于 KMP 的跨端框架技术原理和现有跨端框架无本质差别，但 Kotlin Compiler 使得高性能原生运行变得唾手可得
- KMP 将会降低 JVM Desktop/Darwin 的开发门槛，极大丰富桌面 UI 的开发可能性
- 借助 Compose Multiplatform，任何会 Kotlin 的开发者，都可以迅速上手编写 Desktop App；同时依赖 kotlin 庞大生态，编写业务代码

移动端开发

- KMP 是比 Web¥RN¥Flutter （性能）更佳的选择，因为减少了中间层解释执行的开销。
- 使用 Kotlin 开发跨端 UI，对 Android 开发者成本更低；对熟悉 Swift 的开发者，熟悉起来也非常快
- 移动端开发者本身对 Native 特性更熟悉，由他们来编写跨端代码，本身是更合适的

Q & A

Share your feedback to help us better understand your KotlinConf’24 Global experience!

Thank you