QCon
全球软件开发大会

一码多端挑战下的新跨端方案思考与实践

演讲人：范绍贵

字节跳动 / 技术专家

关于我


Web页面、
原生App开发
2013年
首次创业
扫码点餐
2016年
接触 ReactNative
All In JavaScript
Android、IOS、服务器
恋爱君等
10 余款APP
累计600多万用户
赚了个房子首付
2021年
接触 WebAssembly
设计了 WasmTS
TypeScript 编译 wasm
2022年6月
结束创业
加入字节，孵化 WasmTS
2024年6月
基于 WasmTS 的转译技术
将 RTS 转译 Native 语言

QCon
全球软件开发大会

目录

contents

InfoQ 极客传媒

北京

QCon 全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

北京

AiCon 全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

上海

QCon 全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月 — 5月 — 6月 — 8月 — 10月 — 12月

上海

AiCon 全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

深圳

AiCon 全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

北京

AiCon 全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

极客邦科技 2025 年会议规划

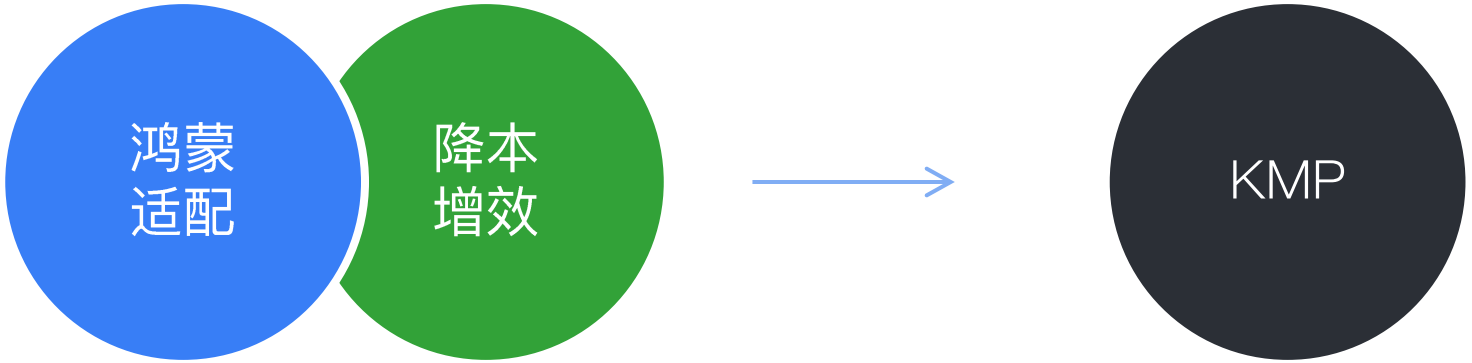
促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

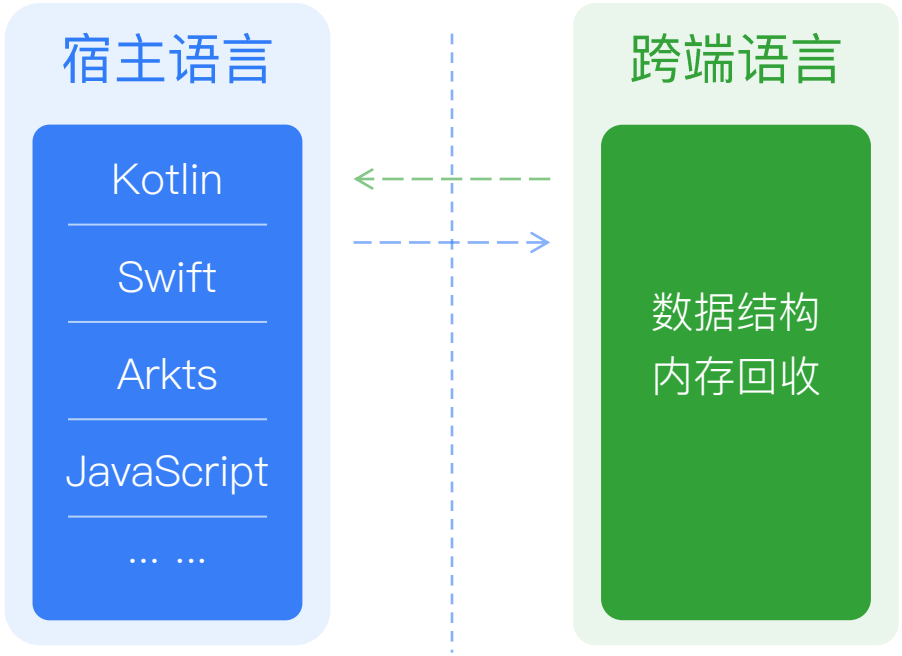
01

背景



有没有更好的
跨端方案?

跨端方案核心瓶颈：FFI 和 GC



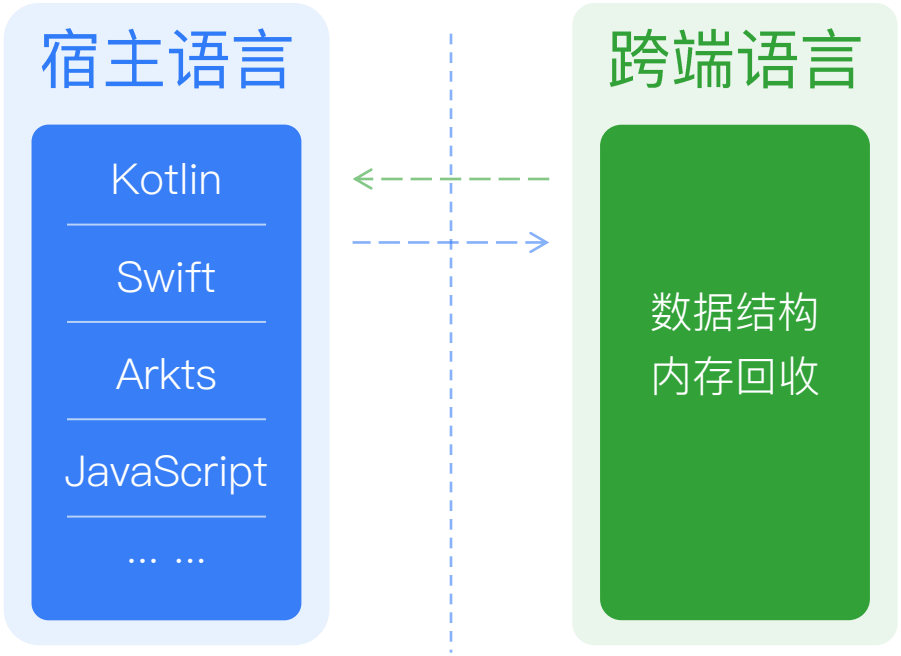
- 跨端语言存在独立的数据结构，比如容器 Array、Map。这些数据无法被宿主识别、消费。
- 数据传输：存在系列化、反序列化，或者对象复制问题
- 内存回收：对象的传输，被外部持有等，导致内存回收异常复杂，回收时机难以精细化，容易内存波动剧烈。

操作类型	Kotlin –> OC	Swift –> OC
空函数调用	8862 ns/100op	380 ns/100op
简单计算操作	6859 ns/100op	378 ns/100op
返回值处理	4914 ns/100op	378 ns/100op

KMP方案

Android	• 没有跨语言问题 • 没有 gc、ffi 问题
Harmony	• KotlinNative，直接对接操作系统 C api • 内部闭环时，可媲美原生 • 与 arkts 交互存在 ffi、gc 问题
iOS	• 与 Swift/OC 存在跨语言调用 • 存在 ffi、gc 问题
More	• 未来可能出现的新端 • 是否会对外暴露底层 C api ？

如何才能彻底解决 ffi 、gc 问题



FFI 和 GC 问题产生

根本原因是 **跨语言** 调用的问题

有没有一种方法

可以做到 **没有跨语言**

转译
是一种可行思路

基于转译的思路

我们围绕语言、工程、UI 框架

打造了一套全新的跨端方案

02

三位一体的跨端方案

三位一体的跨端方案

RTS 语言

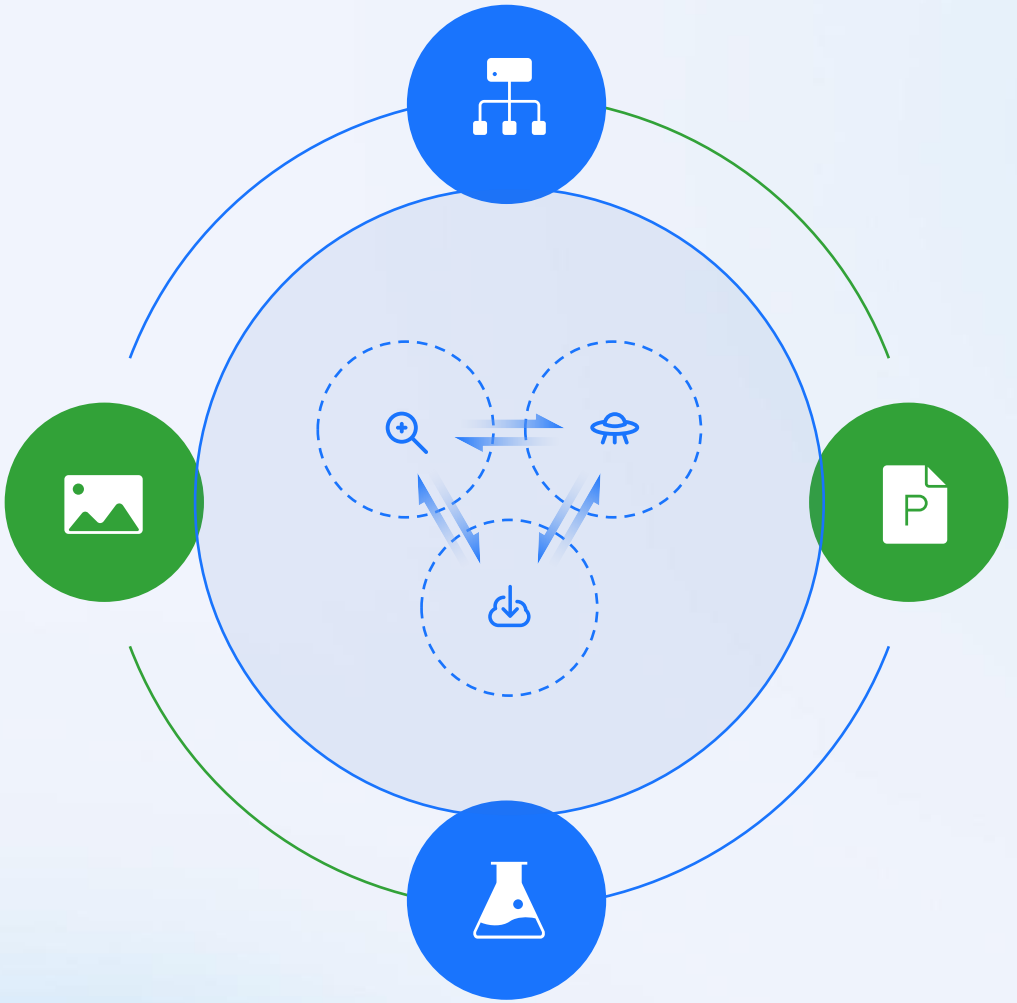
以转译为核心，实现无 FFI 高性能调用

Salamander 工程化

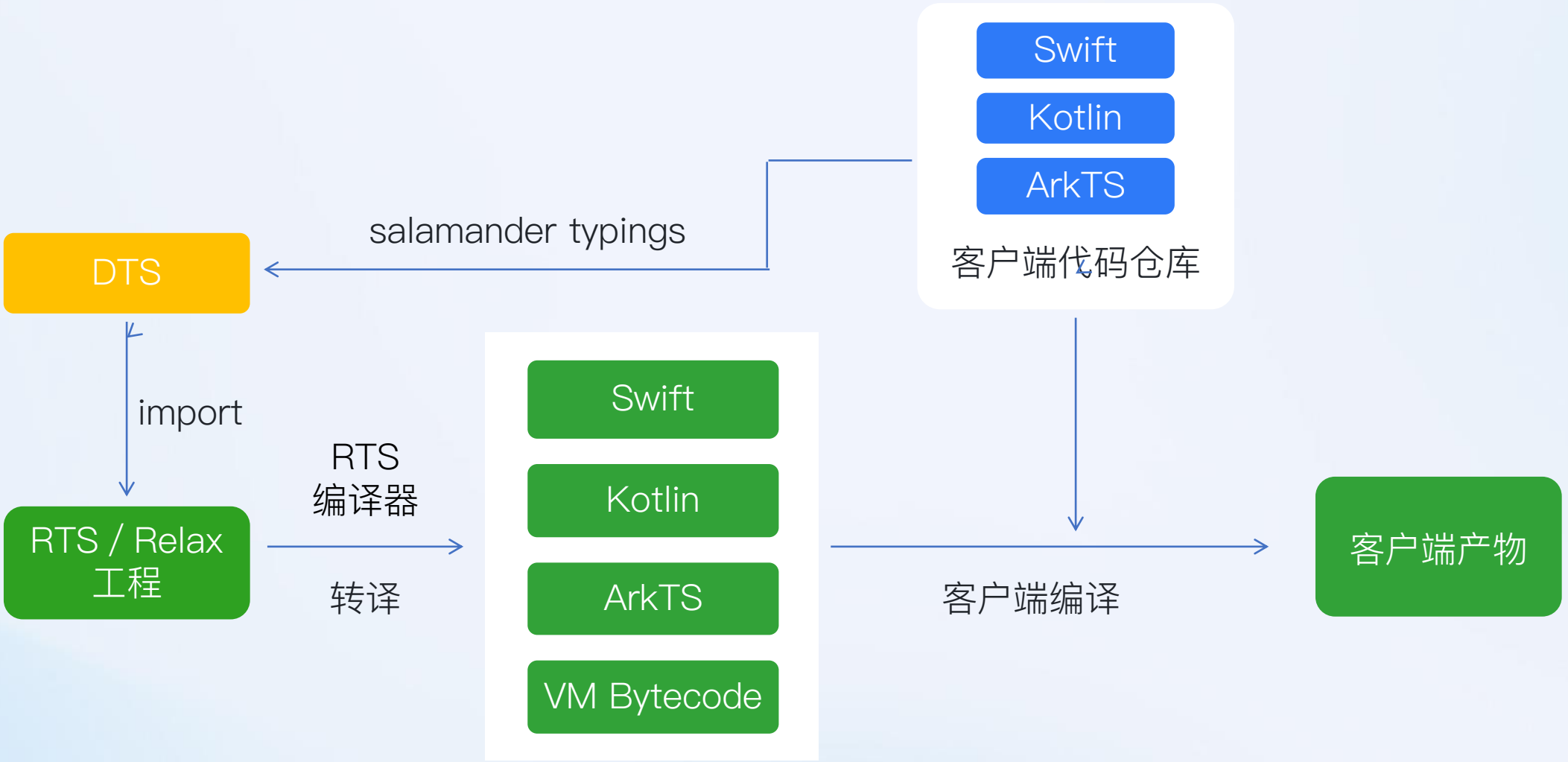
负责整个工程的构建、开发、Typing、IDE工具链

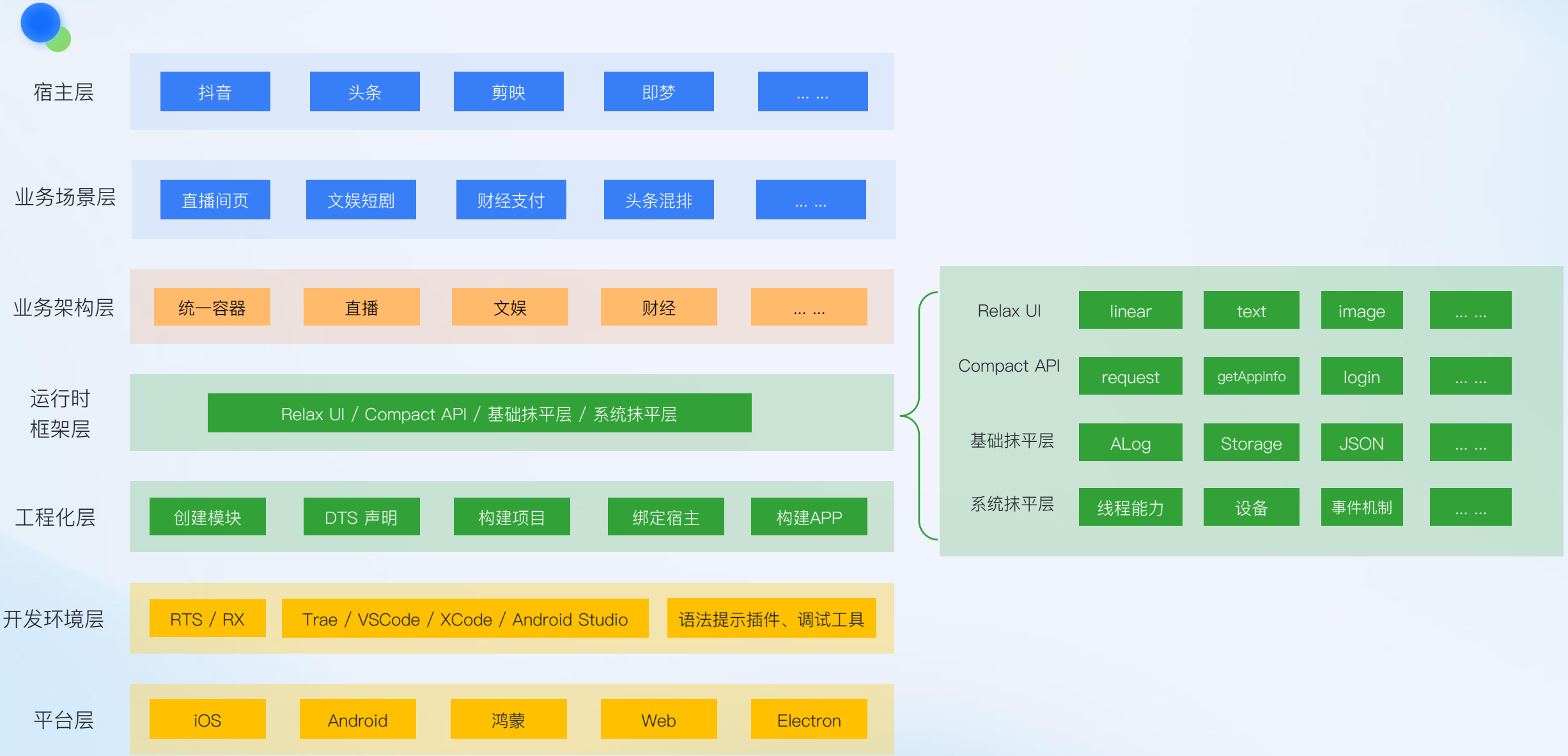
Relax UI 渲染

参考最前沿的前端框架长处，采用 jsx 语法，无运行时的 signal 更新机制。背靠 Lynx 端渲染能力，也能实现动态下发。



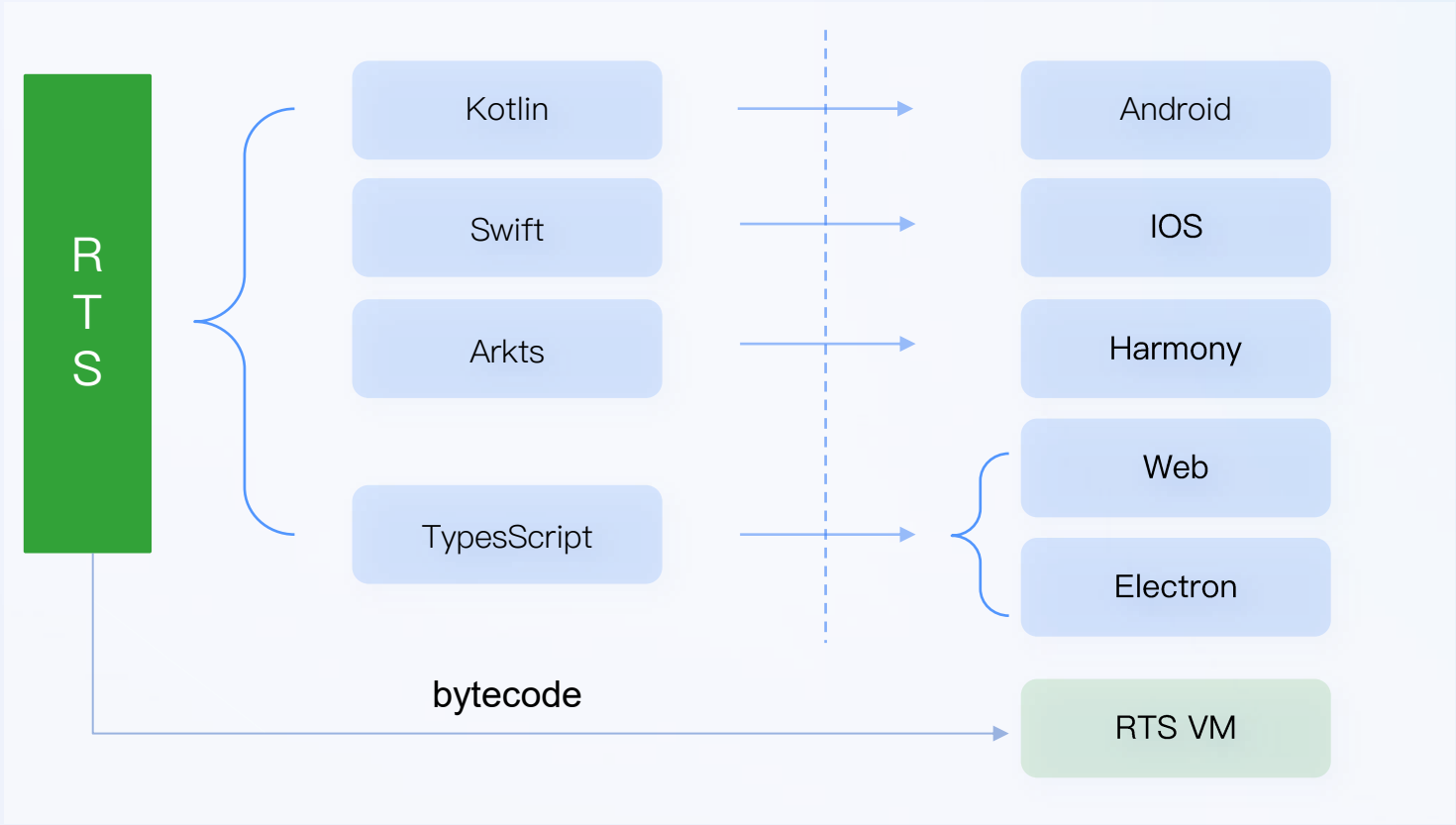
编译流程


Swift
Kotlin
ArkTS
客户端代码仓库
salamander typings
DTS
import
RTS / Relax
工程
RTS
编译器
转译
Swift
Kotlin
ArkTS
VM Bytecode
客户端编译
客户端产物



RTS 语言

- 类型安全的静态语言
- 可转译多种客户端 native 语言
- 拥有自己的虚拟机，可动态下发

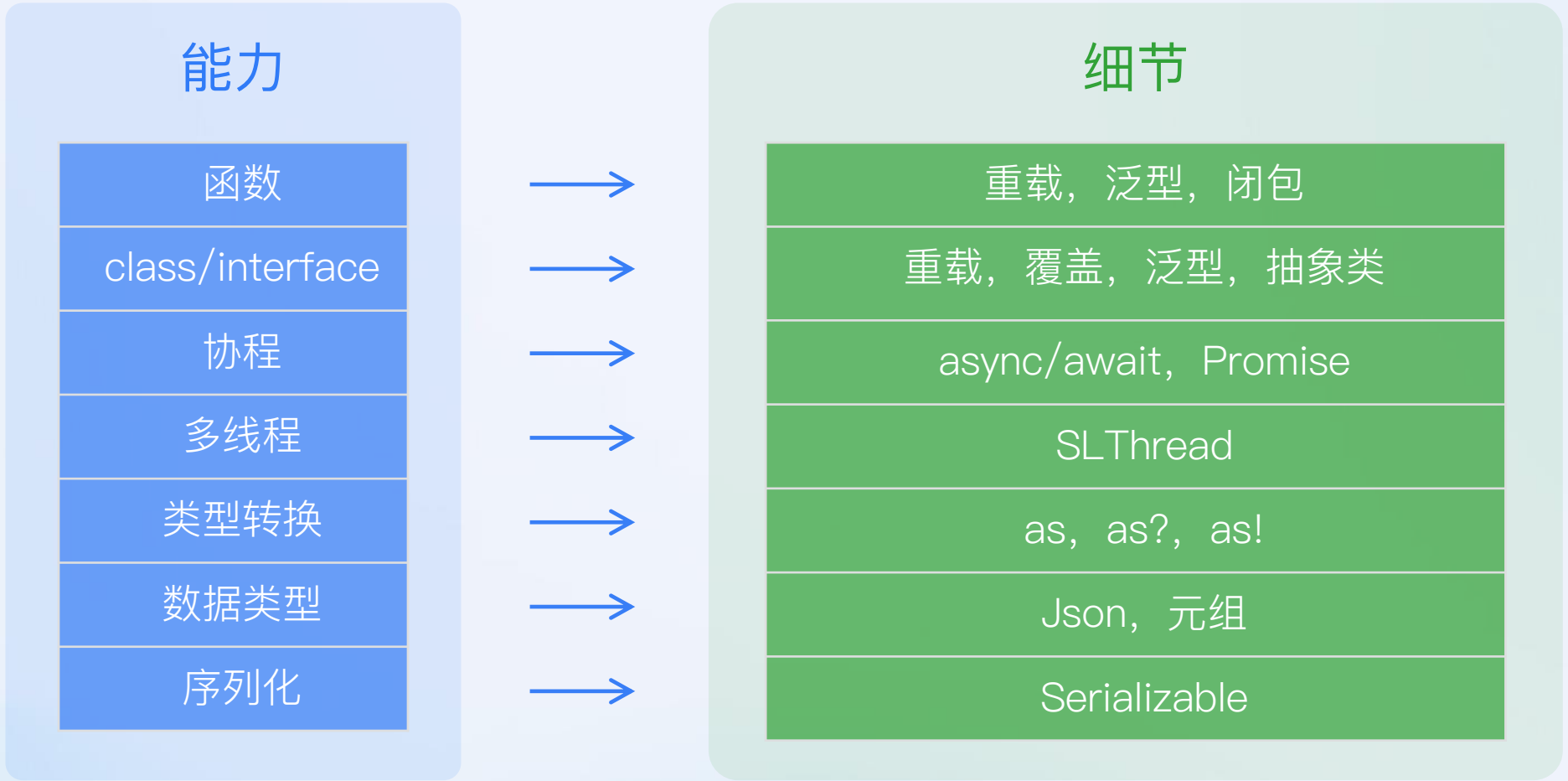


基础数据类型与转译后类型

RTS 类型	Kotlin 类型	Swift 类型	Arkts 类型	TypeScript 类型
int32	Int	Int32	number	number
int64	Long	Int64	bigint	bigint
float32	Float	Float	number	number
float64	Double	Float64	number	number
Array	Kotlin.ArrayList	NSMutableArray	Array	Array
Map	Kotlin.MutableMap	NSMutableDictionary	Map	Map
Set	MutableSet	NSMutableSet	Set	Set

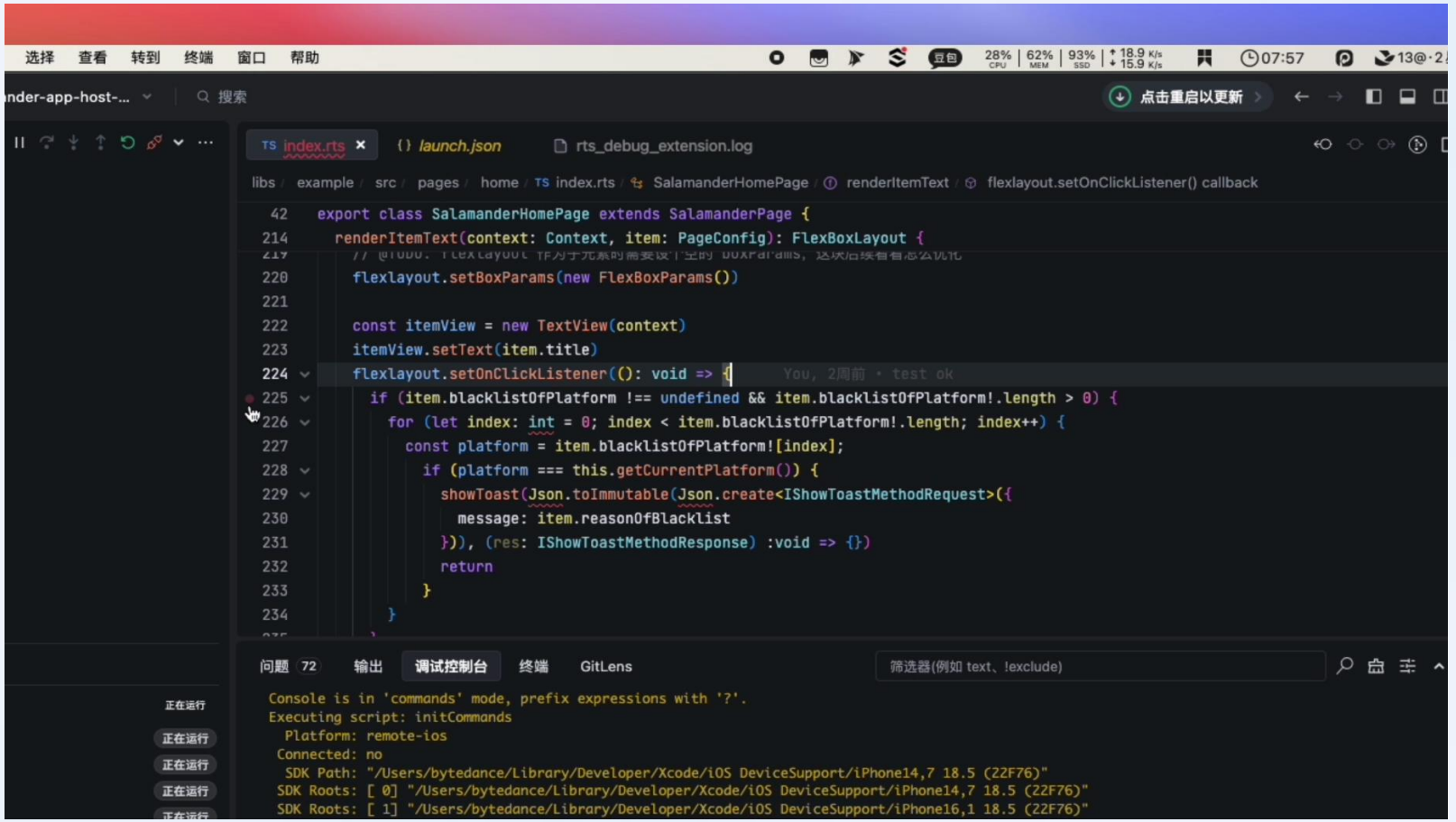
还有 boolean 和 string，int8，int16，uint8，uint16，uint32 等

核心语法支持


能力
函数
class/interface
协程
多线程
类型转换
数据类型
序列化
细节
重载，泛型，闭包
重载，覆盖，泛型，抽象类
async/await，Promise
SLThread
as，as?，as!
Json，元组
Serializable

工具链支持

- 基本的语法提示插件、断点调试能力等



RTS 转译产物示例

```
function main(){
  console.log("hello","world");
}
```

Kotlin

```
import com.bytedance.rts.foundation.*

public fun main() {
  console.log("hello", "world");
}
```

Swift

```
import rts2native

public func main() {
  console.log("hello", "world")
}
```

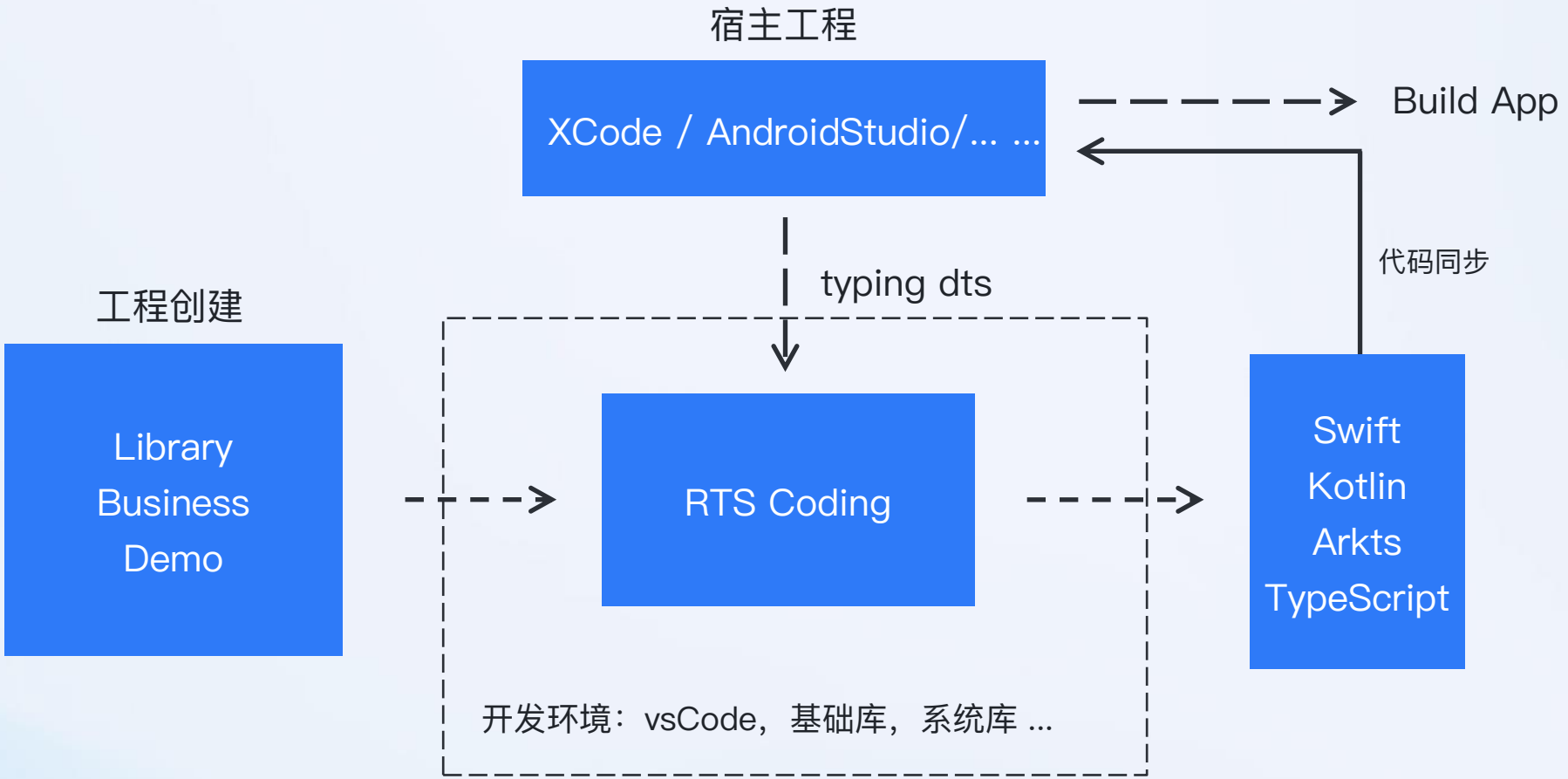
Kotlin/Swift 内置对象实现

```kotlin
package com.bytedance.rts.foundation

public class console {
    companion object {
        fun log(vararg messages: Any?){
            println(messages.joinToString(" "));
        }
        fun error(vararg messages: Any?){
            System.err.println("RTSError:" + messages.joinToString(" "));
        }
    }
}
```

```swift
public class console {

  public static func log(_ message: Any...) {
    let output = message.map { "\($0)" }.joined(separator: " ")
    print(output)
  }

  public static func error(_ error: Any...) {
    printError("RTSError:", terminator: "")
    let output = error.map { "\($0)" }.joined(separator: " ")
    printError(output)
  }

}
```

Salamander 工程化

Salamander 工程化

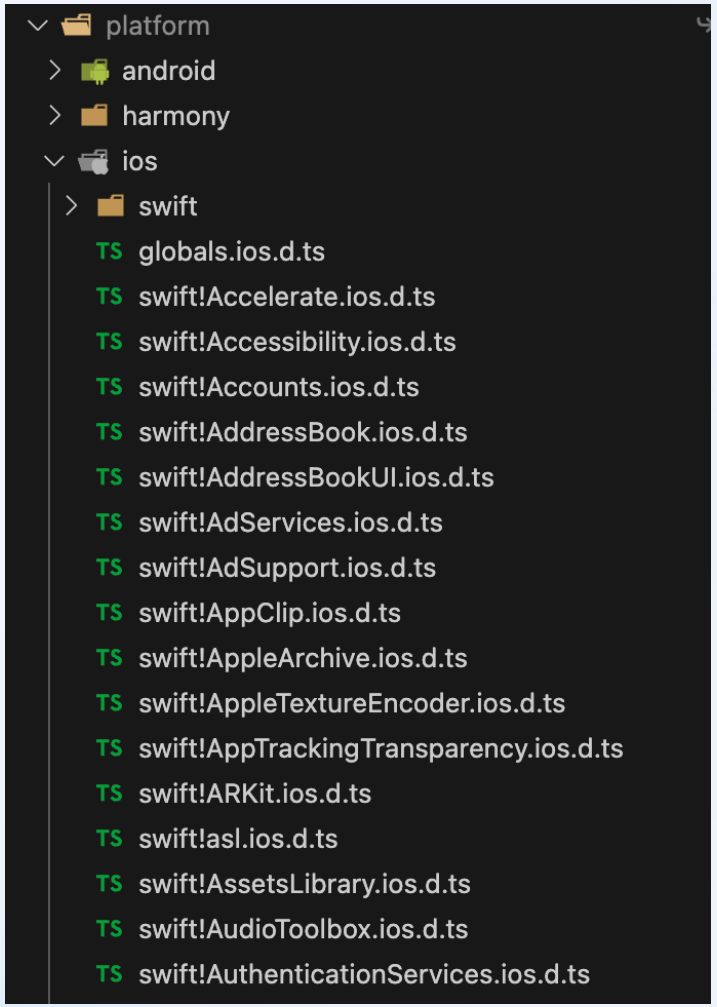


系统 API 能力

```ts
import { UIDevice } from '@byted-salamander/platform/ios/swift/UIKit';

function printOSVersion(): void {
  console.log('OS Version: ', UIDevice.current.systemVersion);
}
```

```ts
import { Build } from '@byted-salamander/platform/android/android/os';

function printOSVersion(): void {
  console.log('OS Version: ', Build.VERSION.RELEASE);
}
```

```ts
import { deviceInfo } from '@byted-salamander/platform/harmony/@kit.BasicServicesKit';

function printOSVersion(): void {
  console.log('OS Version: ', deviceInfo.osFullName);
}
```



Relax UI 框架

Relax 是声明式、响应式的高性能 UI 框架

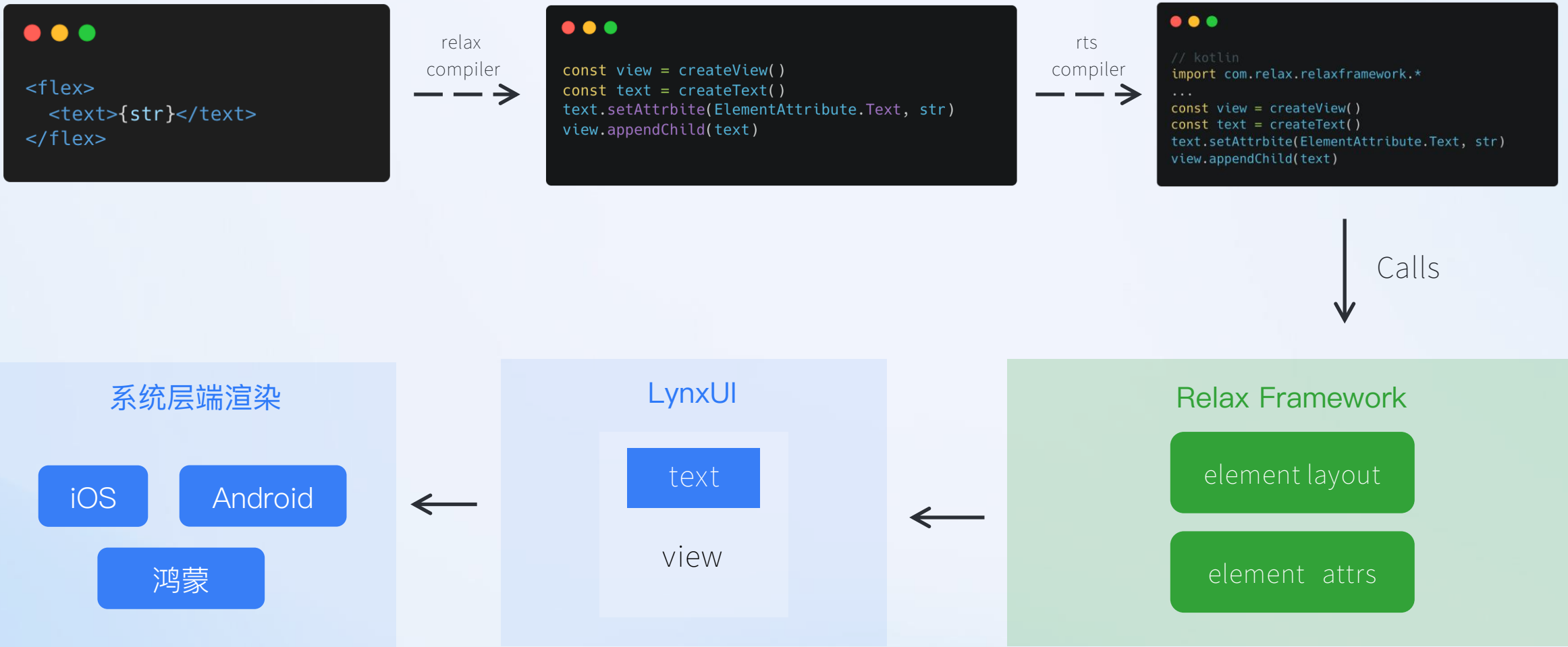
是使用 RTS 语言开发的 UI 框架

借鉴了非常多 React 和 Solid 的思想，提供了 Signal 的数据驱动理念，将 JSX 直接转译为创建视图及设置属性的函数调用，不需要 Diff 操作

另外在架构上沿用了端渲染方案，可媲美原生性能，并无缝调用已有的平台侧组件。

```
export type MyCompProps = {
    name: string;
};

export const Comp: RxComponent<MyCompProps> = (
    props: MyCompProps,
    children: Optional<RxComponentChildren>,
): RxElement => {

    const [counter, setCounter] = state<int32>(0);
    const onIncrease = (event: CommonEvent.Tap): void => {
        setCounter((counter: int32): int32 => counter + 1);
    }

    const modifier = Modifier.view
                             .height(40)
                             .borderWidth(1)
                             .borderColor(0xffff0000)
                             .width(200)
                             .marginTop(10);

    return (
        <linear>
            <linear>
                <text>{props.name}</text>
                <text>Counter: {counter}</text>
            </linear>
            <linear>
                <linear
                    bindTap={onIncrease}
                    modifiers={[modifier]}>
                    <text>Increase</text>
                </linear>
            </linear>
        </linear>
    )
};
```

编译与渲染




```
<flex>
  <text>{str}</text>
</flex>
```

```
const view = createView()
const text = createText()
text.setAttrbite(ElementAttribute.Text, str)
view.appendChild(text)
```

```
// kotlin
import com.relax.relaxframework.*
...
const view = createView()
const text = createText()
text.setAttrbite(ElementAttribute.Text, str)
view.appendChild(text)
```

03

核心挑战

核心挑战一

数据传输无FFI

内置对象的实现

```
function main(){
  const array = new Array<int32>();
  console.log("lenght", array.length);
}
```

Kotlin

```
import com.bytedance.rts.foundation.*

public fun main() {
  val array = RTSArray<Int>();
  console.log("lenght", array.length);
}
```

Swift

```
import rts2native

public func main() {
  let array = RTSArray()
  console.log("lenght", array.length)
}
```

内置对象 Array 的实现

```
typealias RTSArray<T> = ArrayList<T>

val <T>RTSArray<T>.length: Int
    get() = this.size
```

- 直接使用宿主对应的 Array 容器
- 如果没有相应的属性(比如 length)，则扩展

```
public typealias RTSArray = NSMutableArray

extension NSArray {
    public var length: Int {
        return self.count
    }
}
```

为什么 swift 侧 Array，不用 Swift 自身的 Array

而是用 OC 对象 NSMutableArray？？？

```
public typealias RTSArray = NSMutableArray

extension NSArray {
    public var length: Int {
        return self.count
    }
}
```

如果仅仅是值类型、引用类型的区别

为什么不包一层 Swift 的 Array

核心挑战二

与宿主侧的交互通讯

宿主侧接口的调用：系统库、二方库、三方库

swift 头文件

```swift
open class UITextView : UIScrollView, UITextInput,
UIContentSizeCategoryAdjusting {

    public init(frame: CGRect, textContainer: NSTextContainer?)

    public var text: String!

    public var font: UIFont?

    public var layoutManager: NSLayoutManager { get }

    public func scrollRangeToVisible(_ range: NSRange)
}
```

typings

rts 可消费的 .d.ts 文件

```typescript
export declare class UITextView
  extends UIScrollView
  implements UITextInput, UIContentSizeCategoryAdjusting,
UILetterformAwareAdjusting
{
  public delegate: UITextViewDelegate | undefined;

  /**
   * @realname text
   */
  public $_text: string;

  public textColor: UIColor | undefined;

  public textAlignment: NSTextAlignment;

  public selectedRange: NSRange;

  /**
   * @param range NoNameNeeded
   */
  scrollRangeToVisible(range: NSRange): void;

}
```

继承宿主侧 Class

rts 文件

```
import { UITextView } from "UIKit"

class MyClass extends UITextView {
  constructor(){
    super();
  }
}
```

.d.ts 文件

```
export declare class UITextView
  extends UIScrollView
  implements UITextInput, UIContentSizeCategoryAdjusting,
UILetterformAwareAdjusting
{
  public delegate: UITextViewDelegate | undefined;

  /**
   * @realname text
   */
  public $_text: string;

  public textColor: UIColor | undefined;

  public textAlignment: NSTextAlignment;

  public selectedRange: NSRange;

  /**
   * @param range NoNameNeeded
   */
  scrollRangeToVisible(range: NSRange): void;

}
```

如何用 .d.ts 描述各宿主语言的接口签名

swift 代码

```swift
public struct Resolution {
    var width = 0
    var height = 0
}
```

typings

.d.ts

```typescript
/**
 * @final
 * @struct
 */
export class Resolution {
}
```

复杂一点的例子

swift 代码

```swift
public class Celsius {
    var temperatureInCelsius: Double
    public init(fromFahrenheit fahrenheit: Double) {
        temperatureInCelsius = (fahrenheit - 32.0) / 1.8
    }
    public init(fromKelvin kelvin: Double) {
        temperatureInCelsius = kelvin - 273.15
    }
    public init(_ celsius: Double) {
        temperatureInCelsius = celsius
    }
}
```

typings

.d.ts 文件

```typescript
export class Celsius {
  /**
   * @constructor
   * The original Swift function name: init(fromFahrenheit:)
   * @realname init
   * @prefix fahrenheit fromFahrenheit
   */
  static initFromFahrenheit(fahrenheit: Double): Celsius;

  /**
   * @constructor
   * The original Swift function name: init(fromKelvin:)
   * @realname init
   * @prefix kelvin fromKelvin
   */
  static initFromKelvin(kelvin: Double): Celsius;

  /**
   * @param celsius NoNameNeeded
   */
  constructor(celsius: Double);
}
```

小结

- rts 编译器在编译时，识别 .d.ts 文件上的注释做相应翻译
- .d.ts 并不能描述所有 swift/kotlin 签名，只解决 80% 问题
- 剩余不能支持部分，需要手写 swift/kotlin 代码进行适配

核心挑战三

多端一致性保证

多端一致：内存回收机制

rts 文件

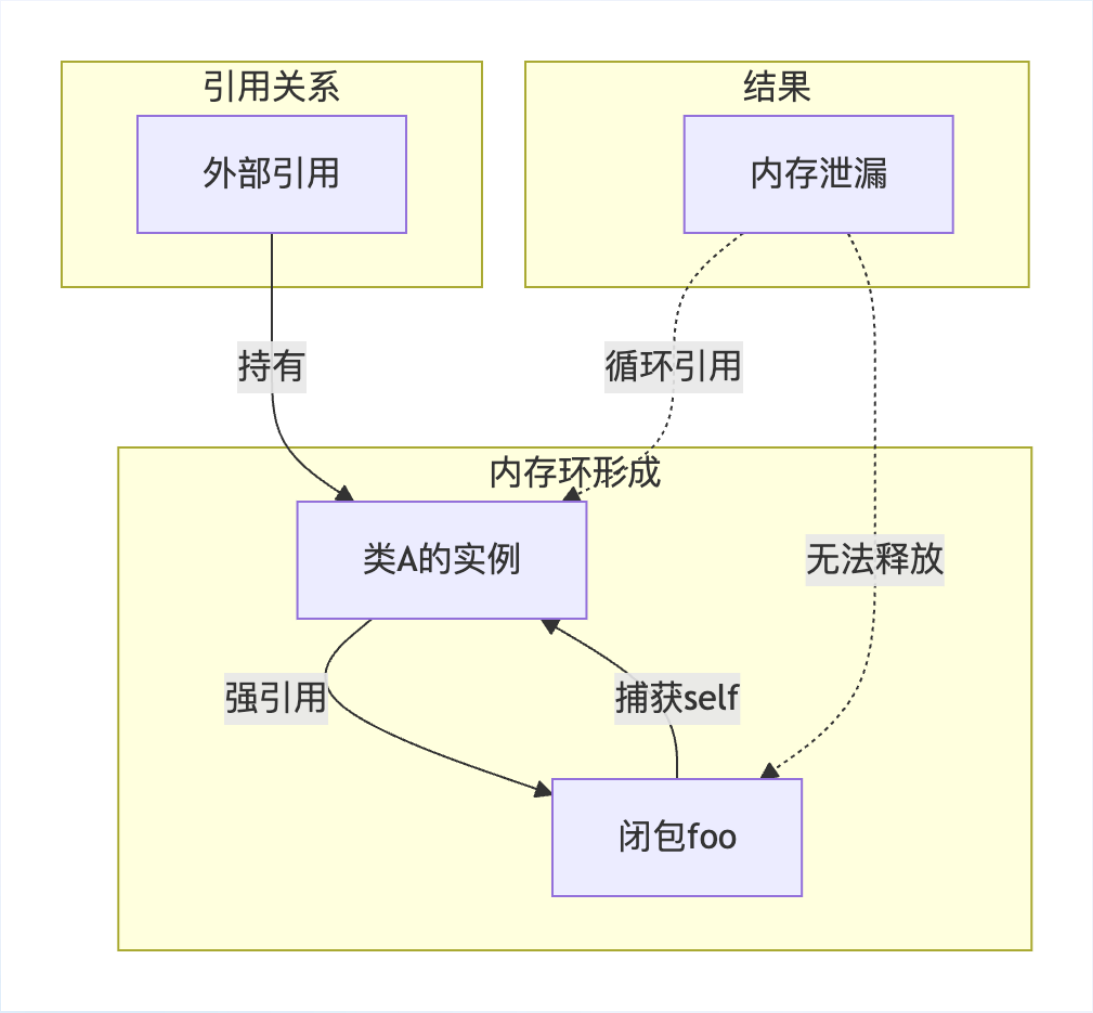
```
class A {
  name: string = "my name";
  foo?: () => void;
  constructor() {
    this.foo = () => {
      console.log(this.name);
    }
  }
}
```

swift

```
open class A {
  public var name: String = "my name"
  lazy public var foo: (() -> Void)? = {() in
    console.log(self.name)
  }
  public init() {
    self.foo = {() in
      console.log(self.name)
    }
  }
}
```

多端一致：swift 内存环问题

```swift
open class A {
  public var name: String = "my name"
  lazy public var foo: (() -> Void)? = {() in
    console.log(self.name)
  }
  public init() {
    self.foo = {() in
      console.log(self.name)
    }
  }
}
```



正确做法：使用 weak 关键字，手动解环

rts 文件

```
class A {
  name: string = "my name";
  weak foo?: () => void;
  constructor() {
    this.foo = () => {
      console.log(this.name);
    }
  }
}
```

swift

```
open class A {
  public var name: String = "my name"
  weak lazy public var foo: (() -> Void)? = {() in
    console.log(self.name)
  }
  public init() {
    self.foo = {() in
      console.log(self.name)
    }
  }
}
```

多端一致：int64 问题

rts 文件

```
function main() {

  const a: int64 = 1;
  const b: int64 = a + 2;
  console.log(b);

}
```

TypeScript

```
import { int64 } from '@rts2native/foundation';
function main() {
  const a: int64 = 1n;
  const b: int64 = BigInt(a + 2n);
  console.log(b);
}
```

多端一致：重载的支持问题

```
function foo(a: int32) { }
function foo(a: int32, b: float32) { }

function main() {

  foo(1);
  foo(1, 2);

}
```

重载不区分 int32/float32
数字类型
不支持 enum 类型重载

TypeScript

```
function foo(a: int32, b: float32);
function foo(a: int32);
function foo(...rtsOverloadArgs: any[]): any {
  if(rtsOverloadArgs.length === 1) {
    if(typeof rtsOverloadArgs[0] === 'number')
        return fooOverloadvariable0(rtsOverloadArgs[0]);
  }
  else if(rtsOverloadArgs.length === 2) {
    if(typeof rtsOverloadArgs[0] === 'number' && typeof rtsOverloadArgs[1] === 'number')
        return fooOverloadvariable1(rtsOverloadArgs[0], rtsOverloadArgs[1]);
  }

  throw new Error('overload function not found');
}
function fooOverloadvariable0(a: int32{ }
function fooOverloadvariable1(a: int32, b: float32) {}

function main() {
  foo(1);
  foo(1, 2);
}
```

多端一致：空安全问题

```
class A { }

function foo(obj: Object) {

  const a: A = obj as A;
  // 如果转换失败，则 b 为 nil
  const b: Optional<A> = obj as? A;
  // 如果 obj 不为 A，则运行时 crash
  const c: A = obj as! A;

}
```

TypeScript

```
class A {

}
function foo(obj: Object) {
  const a: A = obj as A;
  // 如果转换失败，则 b 为 nil
  const b: Optional<A> = CondCast.toClass<A>(obj, A);
  // 如果 obj 不为 A，则运行时 crash
  const c: A = ForcedCast.toClass<A>(obj, A);
}
```

多端一致：副作用问题

a.rts

```
import { B } from "./b";

function main() {
  const b: B = new B();
}
```

import

b.rts

```
function topfunc() {
  console.log("topfunc")
}

topfunc(); // 副作用

export class B { }

export class A {
  static b: B = new B(); // 副作用
}
```

- 对于 kotlin 来说，topfunc 会被执行
- web/arkts 则不会执行 topfunc

多端不一致：非 int64 的数字类型判断

```
interface Type {
  isInt32(arg: Optional<Object>): boolean;
  isInt64(arg: Optional<Object>): boolean;
  isFloat64(arg: Optional<Object>): boolean;
  isBoolean(arg: Optional<Object>): boolean;
  isString(arg: Optional<Object>): boolean;
  isNil(arg: Optional<Object>): boolean;
}

declare var Type: Type;
```

在鸿蒙和 web，
并不区分 int32 和
float64

04

方案亮点

极致的性能

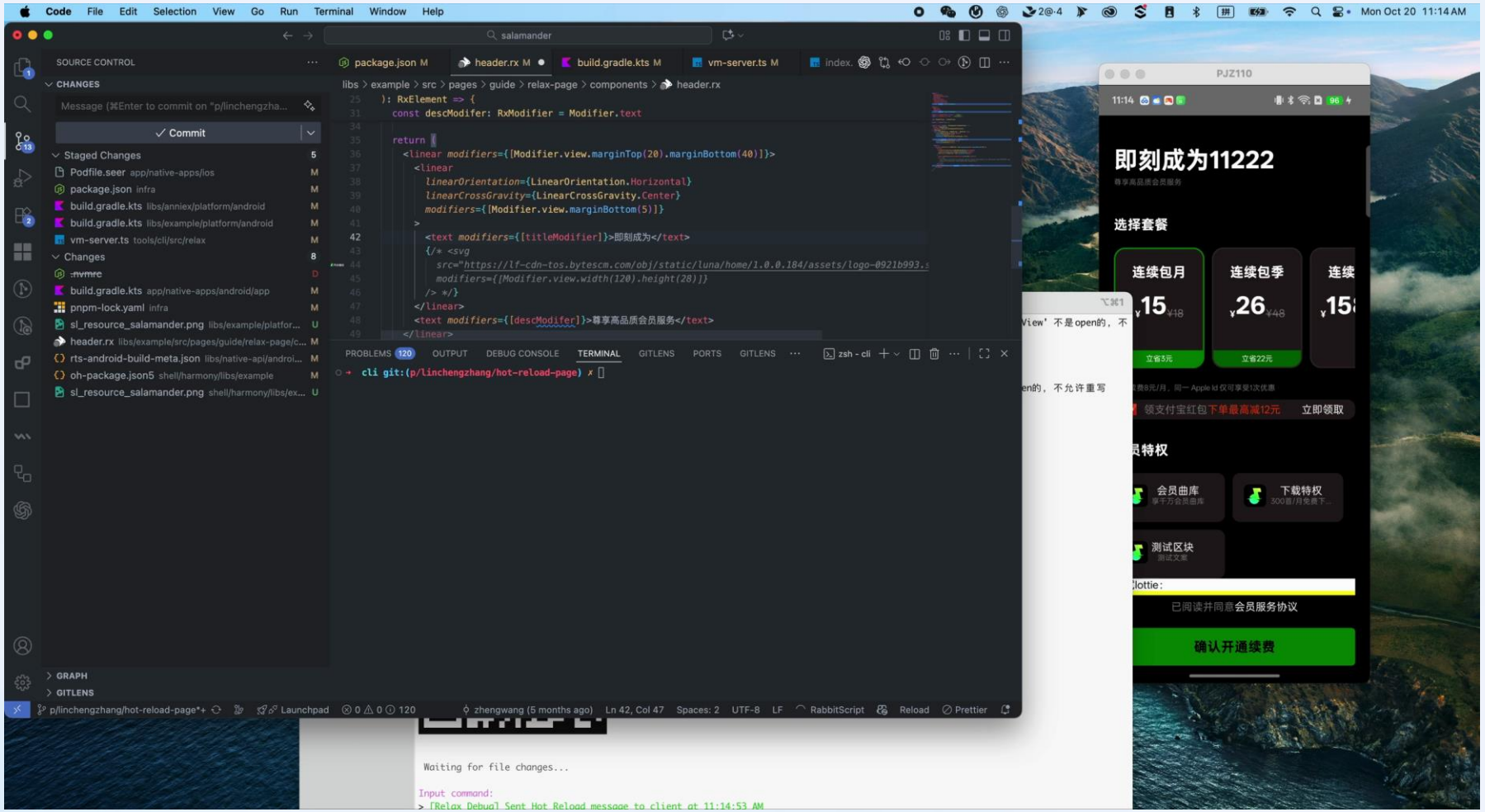
无 FFI ， GC 问题

轻量，集成成本低

可无缝使用客户端生态

UI 渲染无 diff

Hot Reload 开发



新的跨端方案形态：动静态兼顾

业务处于迭代期

通过下发动态代码，跑在 RTSVM

业务进入稳定期

转为原生代码，原生性能

05

性能与业务收益

单文本渲染 / Hello World

iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	47.92ms	53.19ms	50.00ms	48.61ms	63.89ms	59.72ms
CPU Cost Time	36.98ms	43.79ms	45.71ms	42.10ms	50.27ms	51.92ms
Memory	-79.00 KB	339.00 KB	49.67 KB	92.33 KB	506.67 KB	243.75 KB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Relax2Native
Render Time	16.70ms	15.67ms	16.54ms	16.49ms	53.23ms	33.76ms
CPU Cost Time	41.00ms	188.88ms	73.38ms	72.63ms	113.75ms	111.00ms
Memory	16.13 MB	21.69 MB	14.21 MB	11.99 MB	17.82 MB	13.74 MB

- 这个用例主要关注各方案的启动成本。
- IOS 上整体指标 Native 优于 Relax 优于 Swiftui, 但是差距数值较小
- Android 上 Compose CPU 负载显著较高。 Native 优于 Relax 优于 Compose

单列图文列表 / ImageTextLiest

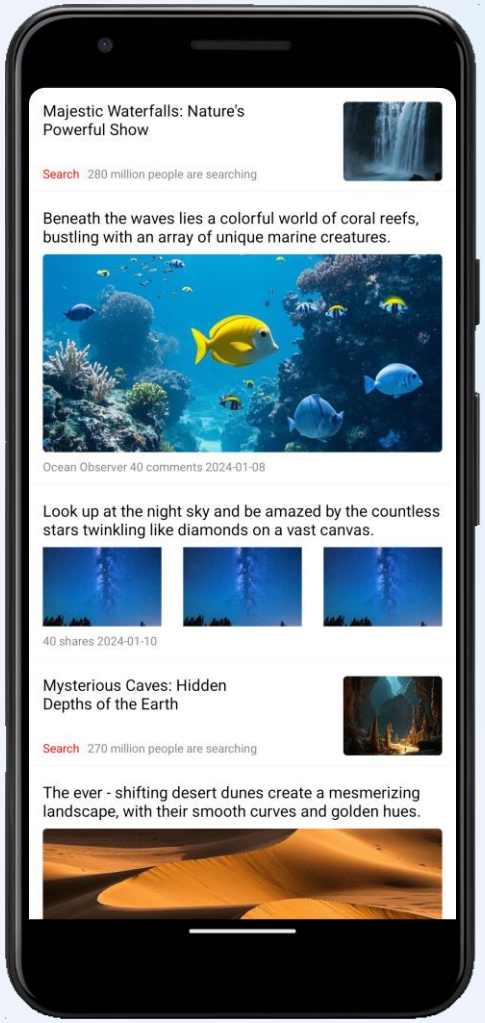
iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	108.33ms	145.83ms	108.33ms	112.50ms	129.17ms	59.72ms
CPU Cost Time	89.38ms	124.63ms	105.25ms	97.00ms	104.00ms	51.92ms
Memory	1.38 MB	5.46 MB	1.59 MB	2.08 MB	7.90 MB	243.75 KB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Relax2Native
Render Time	81.88ms	116.61ms	79.64ms	105.71ms	213.45ms	147.86ms
CPU Cost Time	310.63ms	592.25ms	453.38ms	225.75ms	408.75ms	277.25ms
Memory	67.89 MB	72.13 MB	66.86 MB	43.61 MB	49.63 MB	46.14 MB

- IOS 上整体指标 Relax ~= Native 优于 SwiftUI
- Android 上整体指标 Native 优于 Relax 优于 Compose



内嵌卡片列表 / EmbeddedImageTextLiest

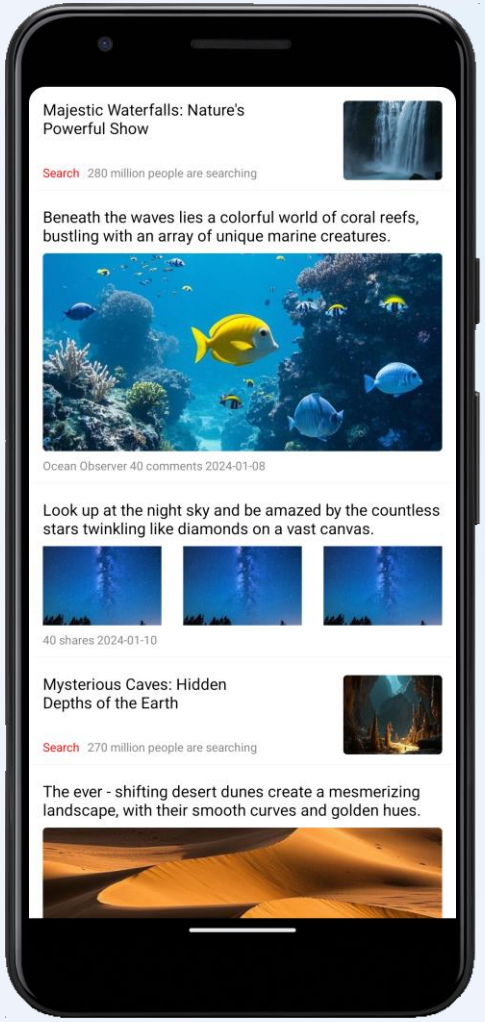
iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	91.88ms	110.42ms	83.13ms	102.08ms	133.33ms	106.25ms
CPU Cost Time	94.38ms	94.50ms	102.63ms	91.00ms	125.88ms	113.63ms
Memory	3.23 MB	7.83 MB	11.29 MB	1.49 MB	5.48 MB	9.19 MB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Relax2Native
Render Time	90.83ms	111.53ms	98.63ms	107.28ms	237.24ms	116.35ms
CPU Cost Time	318.25ms	613.88ms	422.00ms	228.13ms	418.00ms	293.00ms
Memory	67.47 MB	77.60 MB	65.30 MB	43.62 MB	48.72 MB	45.00 MB

- IOS 上整体指标 Relax ~= Native 优于 SwiftUI，内存占用 Relax 略高
- Android 上首屏指标 Relax ~= Native 优于 Compose，其他指标 Native 优于
- Relax 优于 Compose



大量色块（6000 个） / ColorfulView

iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	133.33ms	1647.00ms	291.67ms	229.17ms	6318.33ms	495.62ms
CPU Cost Time	128.13ms	1645.13ms	292.75ms	396.00ms	3634.50ms	487.38ms
Memory	12.99 MB	226.91 MB	43.83 MB	77.03 MB	271.56 MB	39.06 MB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Re
Render Time	200.21ms	382.01ms	223.99ms	470.33ms	1442.63ms	
CPU Cost Time	387.88ms	673.13ms	560.75ms	433.75ms	2007.00ms	
Memory	145.72 MB	77.13 MB	96.33 MB	137.04MB	48.63 MB	94

在SwiftUI视图树的递归 diff 和布局上以及 SwiftUI桥接到 UIKit后，UIKit 会继续递归遍历所有 UIView 层级。在极限场景下损耗较大。

- 主要关注创建大量节点时的数据(Flex 布局)
- IOS 上 整体数据 Native 优于 Relax 优于 SwiftUI
- Android 上整体数据 Native 优于 Relax 优于 Compose

文本渲染 / ComplicateText

iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	133.33ms	166.46ms	150.00ms	183.33ms	216.67ms	202.08ms
CPU Cost Time	130.63ms	154.50ms	159.88ms	176.13ms	214.50ms	215.25ms
Memory	21.21 MB	27.27 MB	43.99 MB	17.89 MB	27.62 MB	30.83 MB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Relax2Native
Render Time	75.74ms	147.14ms	99.16ms	165.83ms	300.28ms	264.60ms
CPU Cost Time	301.13ms	433.38ms	346.25ms	191.38ms	354.63ms	310.50ms
Memory	53.49 MB	55.57 MB	55.73 MB	44.32 MB	45.66 MB	47.21 MB

- 渲染大量的文字内容。这个用例主要关注各个框架对文字的处理性能。
- iOS 上，整体指标 Native 优于 Relax 优于 SwiftUI
- Android 平台上各项指标：Native 优于 Relax 优于 Compose

长列表 / LongList

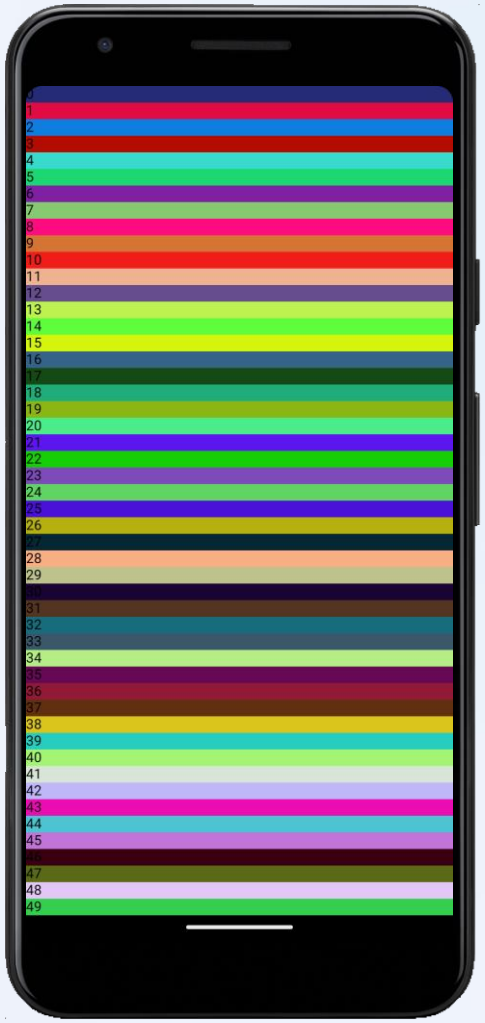
iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	95.83ms	91.67ms	83.33ms	112.50ms	104.17ms	100.00ms
CPU Cost Time	71.00ms	75.63ms	78.88ms	86.38ms	97.00ms	86.13ms
Memory	4.65 MB	2.76 MB	1.97 MB	3.46 MB	3.29 MB	2.24 MB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Relax2Native
Render Time	28.95ms	104.54ms	32.93ms	32.61ms	244.80ms	99.94ms
CPU Cost Time	138.88ms	382.88ms	311.75ms	52.25ms	277.25ms	136.13ms
Memory	18.27 MB	25.81 MB	37.41 MB	13.70 MB	20.49 MB	17.16 MB

- 1000 个纯色块和文字。关注长列表压力场景
- IOS 上整体指标 Relax 优于 Native 优于 SwiftUI
- Android 上整体指标 Native 优于 Relax 优于 Compose



用户列表 / UserInfoList

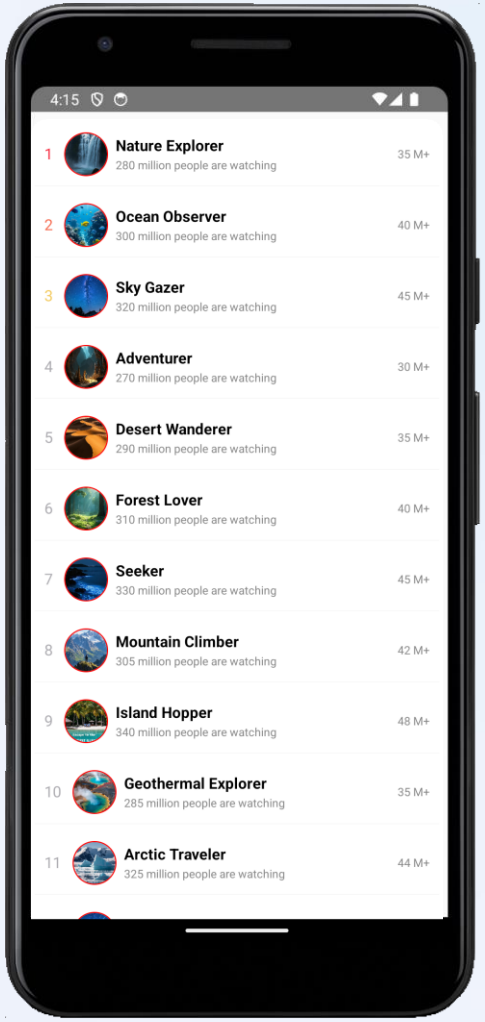
iOS

机型	IPHONE 14			IPHONE 11		
	Native	SwiftUI	Relax2Native	Native	SwiftUI	Relax2Native
Render Time	83.33ms	158.33ms	112.50ms	87.50ms	175.00ms	112.50ms
CPU Cost Time	78.13ms	128.88ms	99.13ms	84.75ms	164.50ms	106.75ms
Memory	3.49 MB	6.86 MB	3.06 MB	2.30 MB	5.98 MB	3.34 MB

Android

机型	Pixel 9			Pixel3		
	Native	COMPOSE	Relax2Native	Native	COMPOSE	Relax2Native
Render Time	97.53ms	185.83ms	114.21ms	161.07ms	307.01ms	209.95ms
CPU Cost Time	482.00ms	886.50ms	660.25ms	408.50ms	694.50ms	463.00ms
Memory	60.40 MB	174.86 MB	56.31 MB	33.51 MB	116.79 MB	33.90 MB

- IOS 上整体指标 Native 优于 Relax 优于 SwiftUI
- Android 上整体指标 Native 优于 Relax 优于 Compose



业务落地情况



相关客户端：抖音、头条、番茄小说、即梦、巨量等

业务	业务开发者数量	RTS 业务累计上线新增代码量	累计 RTS SDK 代码量	同构比例已上线多端复用	**迭代人效取直播数据**
10	25	5.6w	6.1w	65%	40%

相关业务：直播，文娱，财经，商业化，搜索等

06

AI 相关探索

一个问题，两个探索

一个问题

AI 时代，还需不需要跨端？如何是对 AI 更友好的跨端？

两个探索

辅助抹平
Compact API

生态迁移
Kotlin2RTS

InfoQ 极客传媒

北京

QCon 全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

北京

AiCon 全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

上海

QCon 全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月 — 5月 — 6月 — 8月 — 10月 — 12月

上海

AiCon 全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

深圳

AiCon 全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

北京

AiCon 全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

QCon
全球软件开发大会

InfoQ 极客传媒

开源计划

RTS 计划开源，期待你们的参与

一起参与大终端的浪潮，实现“前端、客户端”的大一统

THANKS