Kotlin

bilibili 的鸿蒙之路：从 Kotlin/JS 到 Kotlin/Native 的进化之路

臧至聪（Vicky的饲养员）

Developed by JetBrains | zangzhicong@bilibili.com

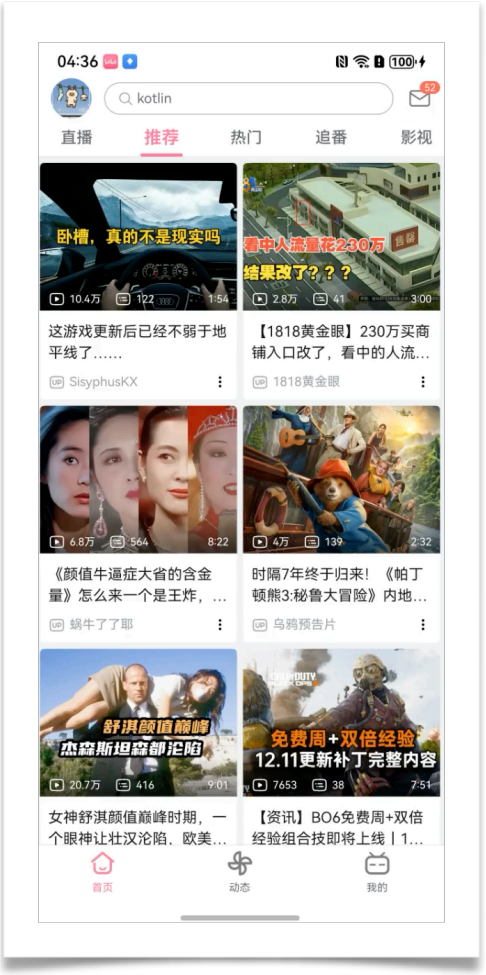
个人介绍

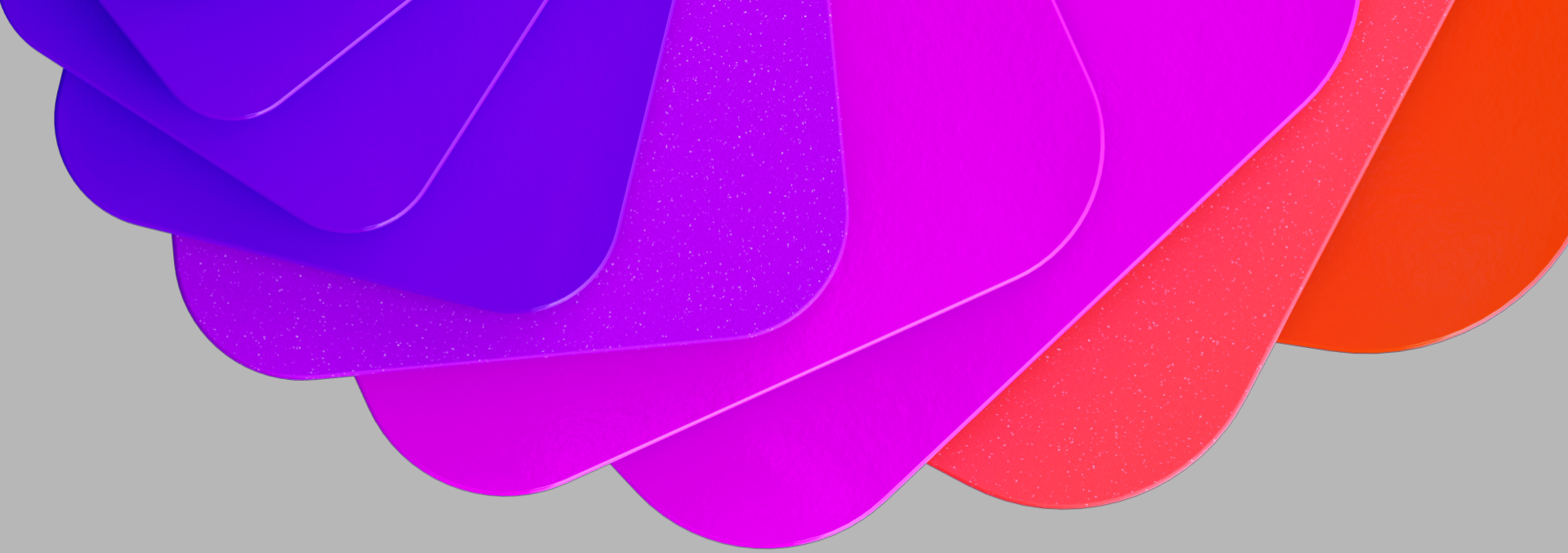
bilibili，移动端开发工程师

深耕 bilibili APP 弹幕，评论等社区相关业务。
目前从事 bilibili HarmonyOS APP 的研发工作。

bilibili Harmony APP

- 2023年底正式开始开发，2024年Q2正式上架，持续迭代
- 选择 KMP 作为 Harmony APP 的跨端方案
- UI 部分使用 ArkUI，业务逻辑大量使用 KMP

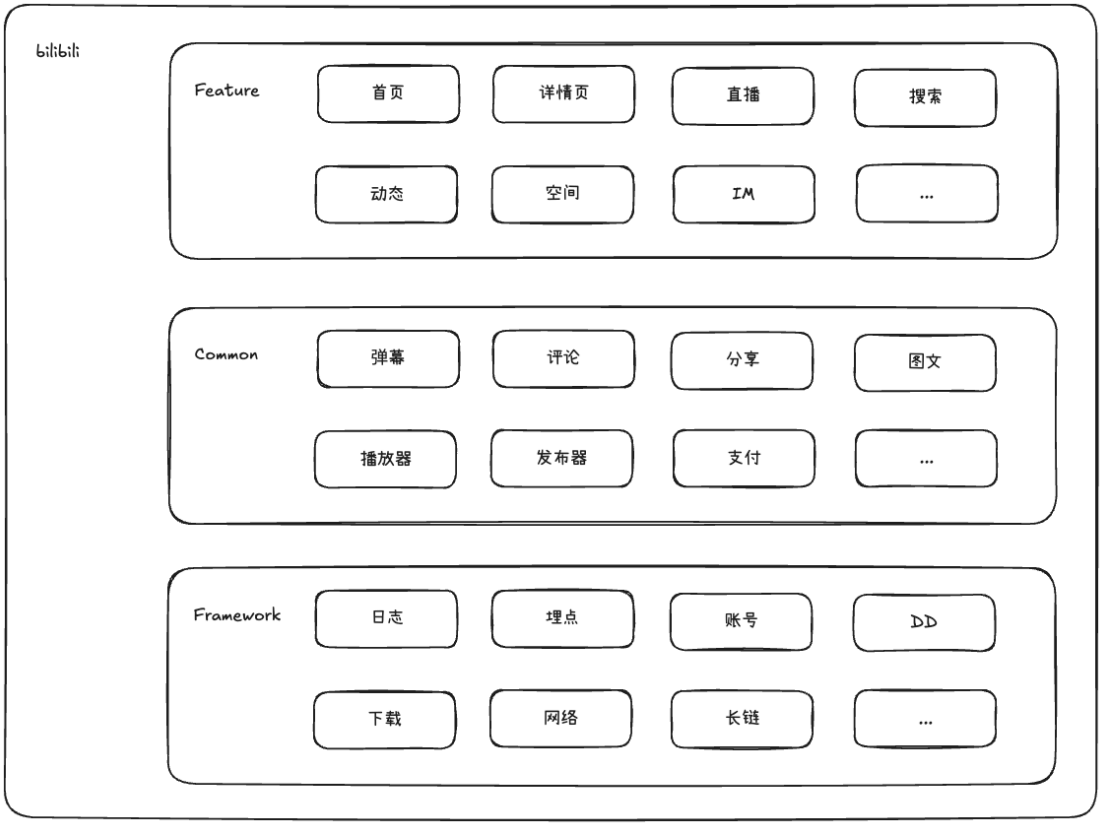




Kotlin JS For bilibili Harmony

Kotlin JS For bilibili Harmony

- Framework 中大部分代码使用 KMP 复用现有基建
- Kotlin JS 产物可以直接运行在 Ark Runtime 上
- 在 jsMain 中可以很容易使用平台提供的 ArkTS API



Kotlin JS 在鸿蒙平台存在的一些问题

- 调试成本较大
- 一些三方库的 jsMain 实现是默认在 Node 或 Browser 环境中运行，与鸿蒙环境在一些场景下会存在些许差异
- 编译产物体积较大
- Kotlin 代码难以使用 ArkTS 提供的多线程能力
- 存在性能瓶颈

Kotlin JS 的调试

虽然 DevEco Studio 支持对 .js 文件的断点调试功能， 但由于没有办法直接将 .js 代码映射到 .kt 代码，这就增加了些许调试的成本。

同时，在使用了 Coroutines 后，K/JS 生成的 js 代码，会变得比较晦涩难懂...

```
doResume_5yljmg_k$() {
  var suspendResult = this.result_1;
  $sm: do
    try {
      var tmp = this.state_1;
      switch (tmp) {
        case 0:
          this.exceptionState_1 = 4;
          this._iterator_0_yqohr0__1 = numberRangeToNumber(1, 10).iterator_jk1svi_k$();
          this.state_1 = 1;
          continue $sm;
        case 1:
          if (!this._iterator_0_yqohr0__1.hasNext_bitz1p_k$()) {
            this.state_1 = 3;
            continue $sm;
          }

          this.i1__1 = this._iterator_0_yqohr0__1.next_20eer_k$();
          console.log('' + this.i1__1 + ':', 'Hello, World!');
          this.state_1 = 2;
          suspendResult = delay(new Long(500, 0), this);
          if (suspendResult === get_COROUTINE_SUSPENDED()) {
            return suspendResult;
          }

          continue $sm;
        case 2:
          this.state_1 = 1;
          continue $sm;
        case 3:
          return Unit_instance;
        case 4:
          throw this.exception_1;
      }
    } catch ($p) {
      var e = $p;
      if (this.exceptionState_1 === 4) {
        throw e;
      } else {
        this.state_1 = this.exceptionState_1;
        this.exception_1 = e;
      }
    }
   while (true);
}
```

Kotlin JS 运行环境差异

```kotlin
// Charset.js.kt
internal actual fun CharsetEncoder.encodeImpl(
    input: CharSequence,
    fromIndex: Int,
    toIndex: Int,
    dst: Sink
): Int {
    require(fromIndex <= toIndex)
    if (charset == Charsets.ISO_8859_1) {
        return encodeISO88591(input, fromIndex, toIndex, dst)
    }

    require(charset === Charsets.UTF_8) {
        "Only UTF-8 encoding is supported in JS"
    }

    val encoder = TextEncoder()
    val result = encoder.encode(input.substring(fromIndex,
toIndex))
    dst.write(result.unsafeCast<ByteArray>())
    return result.length
}

// TextEncoder.js.kt
internal external class TextEncoder {
    val encoding: String

    fun encode(input: String): Uint8Array
}
```

ktor-io

```javascript
const encoder = new TextEncoder();
const view = encoder.encode("€");
// view: Uint8Array(3) [226, 130, 172]
console.log(view);
```

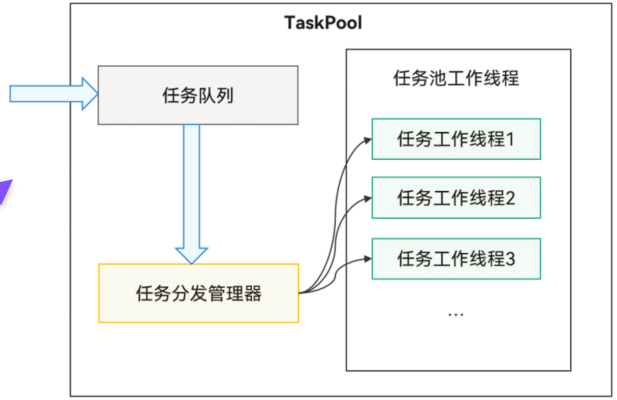
Node or Browser

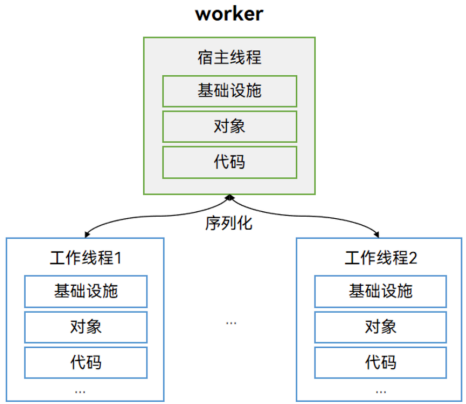
```javascript
const textEncoder = new util.TextEncoder();
const result = textEncoder.encodeInto("€");
// result = 226, 130, 172
console.info("result = " + result);
```

Harmony OS

ArkTS 多线程并发

ArkTS 提供了基于 Actor 并发模型的多线程能力



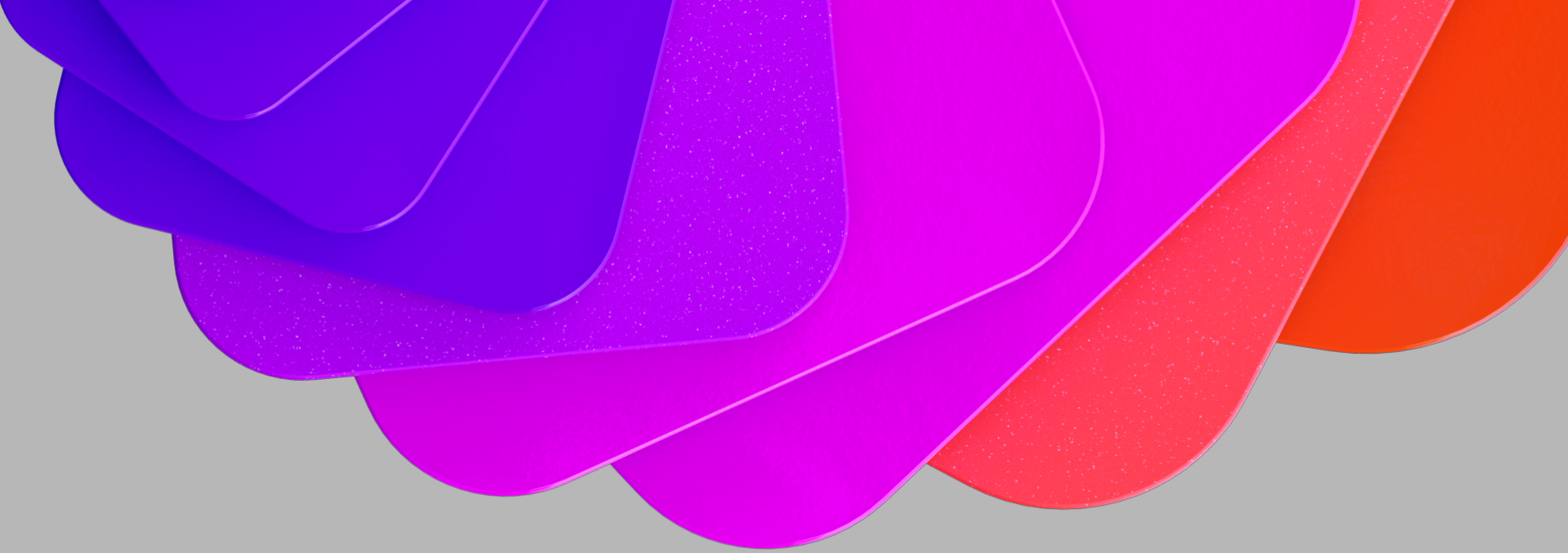


Kotlin JS Coroutines

- 并没有额外的线程支持异步任务
- 缺乏 runBlocking 能力

```
public actual object Dispatchers {
    public actual val Default: CoroutineDispatcher =
createDefaultDispatcher()
    public actual val Main: MainCoroutineDispatcher
        get() = injectedMainDispatcher ?: mainDispatcher
    public actual val Unconfined: CoroutineDispatcher =
kotlinx.coroutines.Unconfined

    private val mainDispatcher = JsMainDispatcher(Default, false)
    private var injectedMainDispatcher: MainCoroutineDispatcher? = null

    @PublishedApi
    internal fun injectMain(dispatcher: MainCoroutineDispatcher) {
        injectedMainDispatcher = dispatcher
    }
}

private class JsMainDispatcher(
    val delegate: CoroutineDispatcher,
    private val invokeImmediately: Boolean
) : MainCoroutineDispatcher() {
    override val immediate: MainCoroutineDispatcher =
        if (invokeImmediately) this else JsMainDispatcher(delegate,
true)
    override fun isDispatchNeeded(context: CoroutineContext): Boolean
= !invokeImmediately
    override fun dispatch(context: CoroutineContext, block: Runnable) =
delegate.dispatch(context, block)
    override fun dispatchYield(context: CoroutineContext, block:
Runnable) = delegate.dispatchYield(context, block)
    override fun toString(): String = toStringInternalImpl() ?:
delegate.toString()
}
```

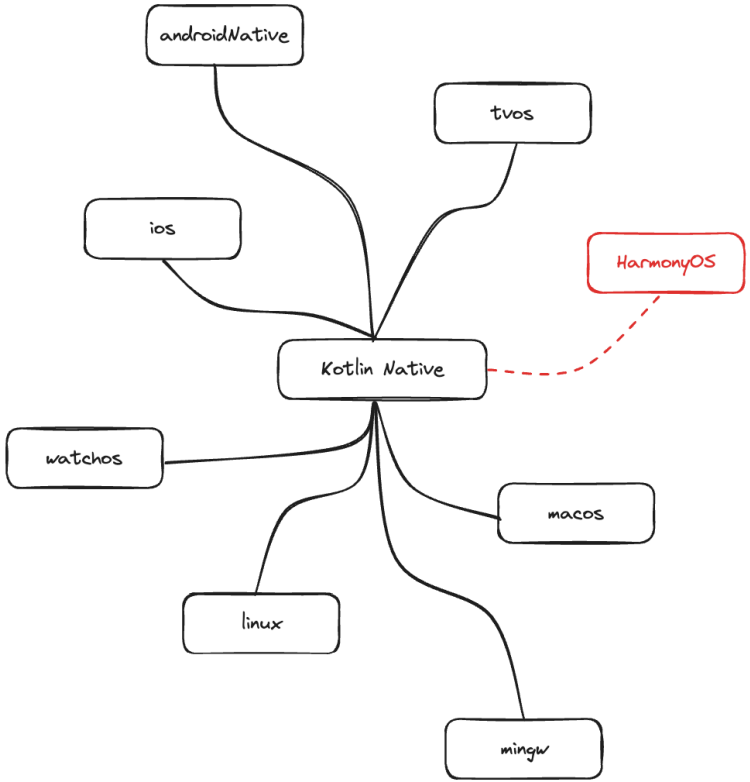
kotlinx.coroutines-1.9.0



Kotlin Native For bilibili Harmony

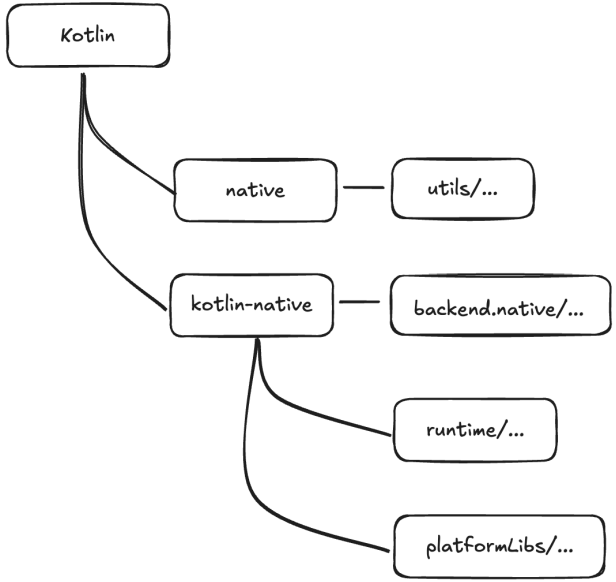
Kotlin Native For bilibili Harmony

- Kotlin Native 需要支持 Harmony Target
- 需要支持在 Kotlin 中使用 Harmony 平台 API
- 三方依赖库需要支持 Harmony Target
- 需要支持 Kotlin 与 ArkTS 互操作



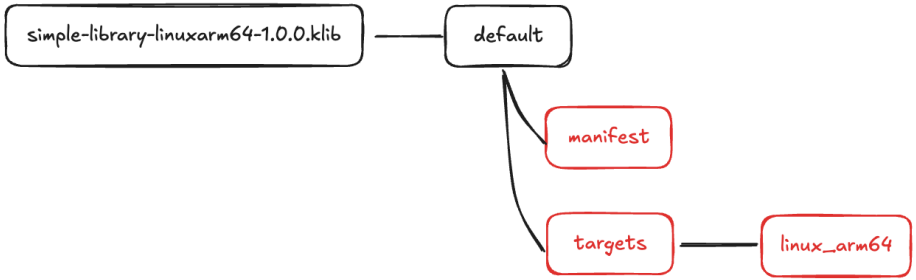
Kotlin Native Compiler + Runtime

- 新增 HarmonyArm64 Konan Target
- 交叉编译 toolchain 的配置
- Kotlin Native Runtime 的适配
- platformLibs 新增鸿蒙平台 NDK 相关声明

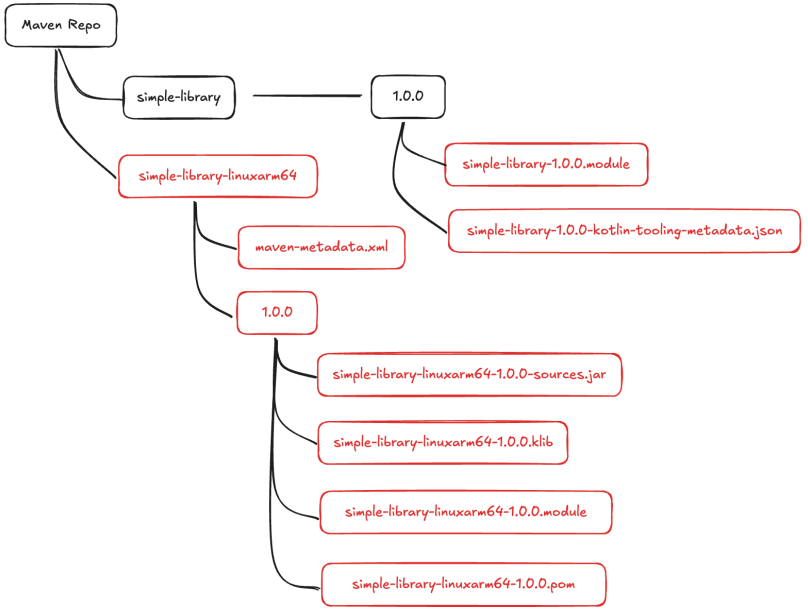


Kotlin-2.0.21

Libraries Supported

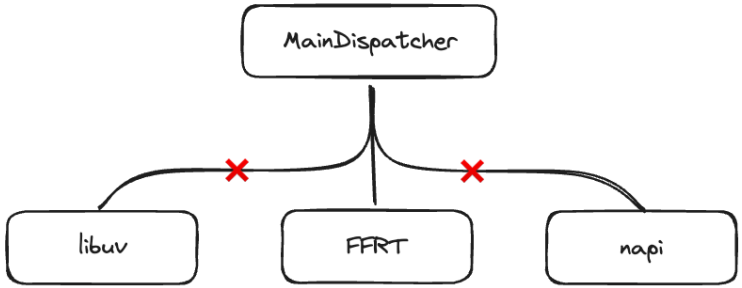


修改红色部分



- 对于没有平台实现的库，只需要对已有的 Maven 产物进行清洗即可
- 对于有平台实现的库，需要 Fork 源码，新增 Harmony 相关实现
- 通过 ComponentMetadataDetails.addVariant 为 Harmony Target 提供依赖库

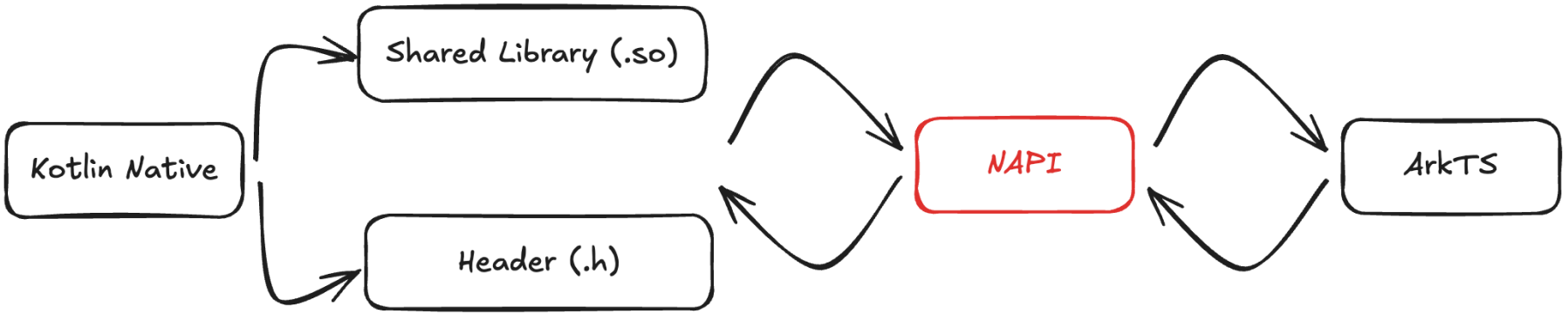
Kotlinx Coroutines For Harmony



- 三种方式均可实现 MainDispatcher 能力
- libuv 已不被官方推荐使用
- napi 使用强依赖 napi env
- FFRT 作为 libuv 的底层实现，完美契合当前场景

```
@OptIn(ExperimentalForeignApi::class, InternalCoroutinesApi::class)
private class HarmonyMainDispatcher(
    private val invokeImmediately: Boolean,
) : MainCoroutineDispatcher() {
    companion object {
        private val mainQueue: ffrt_queue_t? = ffrt_get_main_queue()
    }

    override fun dispatch(context: CoroutineContext, block: Runnable) {
        memScoped {
            val attr = alloc<ffrt_task_attr_t> {
                ffrt_task_attr_init(ptr)
            }
            ffrt_task_attr_set_queue_priority(
                attr.ptr,
                ffrt_queue_priority_high
            )
            ffrt_queue_submit(
                 mainQueue,
                 createQueueTaskPtr(block),
                 attr.ptr
             )
            ffrt_task_attr_destroy(attr.ptr)
        }
    }
}
```

Kotlin Native 与 ArkTS 交互



ArkTS 使用 Native 代码

Harmony 工程

```ts
export const add: (a: number, b: number) => number;
```

TS 接口定义

```ts
import { hilog } from '@kit.PerformanceAnalysisKit';
import testNapi from 'libentry.so';

@Entry
@Component
struct Index {
  @State message: string = 'Hello World';

  build() {
    Row() {
      Column() {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(() => {
            hilog.info(0x0000, 'testTag', 'Test NAPI 2 + 3 = %{public}d', testNapi.add(2, 3));
          })
      }
      .width('100%')
    }
    .height('100%')
  }
}
```

使用

ArkTS 使用 Native 代码

```cpp
static napi_value Add(napi_env env, napi_callback_info info)
{
    size_t argc = 2;
    napi_value args[2] = {nullptr};
    napi_get_cb_info(env, info, &argc, args , nullptr, nullptr);
    napi_valuetype valuetype0;
    napi_typeof(env, args[0], &valuetype0);
    napi_valuetype valuetype1;
    napi_typeof(env, args[1], &valuetype1);
    double value0;
    napi_get_value_double(env, args[0], &value0);
    double value1;
    napi_get_value_double(env, args[1], &value1);
    napi_value sum;
    napi_create_double(env, NativeAdd(value0, value1), &sum);
    return sum;
}
```

Add 方法实现

```cpp
EXTERN_C_START
static napi_value Init(napi_env env, napi_value exports)
{
    napi_property_descriptor desc[] = {
        { "add", nullptr, Add, nullptr, nullptr, nullptr, napi_default, nullptr }
    };
    napi_define_properties(env, exports, sizeof(desc) / sizeof(desc[0]), desc);
    return exports;
}
EXTERN_C_END

static napi_module demoModule = {
    .nm_version = 1,
    .nm_flags = 0,
    .nm_filename = nullptr,
    .nm_register_func = Init,
    .nm_modname = "entry",
    .nm_priv = ((void*)0),
    .reserved = { 0 },
};

extern "C" __attribute__((constructor)) void RegisterEntryModule(void)
{
    napi_module_register(&demoModule);
}
```

Napi 注册

Kotlin Native 生成的 .h

```h
#ifndef KONAN_LIBNEURONS_H
#define KONAN_LIBNEURONS_H
#ifdef __cplusplus
extern "C" {
#endif
#ifdef __cplusplus
typedef bool            libneurons_KBoolean;
#else
typedef _Bool           libneurons_KBoolean;
#endif
typedef unsigned short     libneurons_KChar;
typedef signed char        libneurons_KByte;
typedef short              libneurons_KShort;
typedef int                libneurons_KInt;
typedef long long          libneurons_KLong;
typedef unsigned char      libneurons_KUByte;
typedef unsigned short     libneurons_KUShort;
typedef unsigned int       libneurons_KUInt;
typedef unsigned long long libneurons_KULong;
typedef float              libneurons_KFloat;
typedef double             libneurons_KDouble;
typedef float __attribute__ ((__vector_size__ (16))) libneurons_KVector128;
typedef void*              libneurons_KNativePtr;
struct libneurons_KType;
typedef struct libneurons_KType libneurons_KType;
```

```h
typedef struct {
  struct {
    struct {
      struct {
        struct {
        } DBDelegate;
        struct {
          libneurons_KType* (*_type)(void);
          const char* (*get_buildId)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          const char* (*get_channel)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          libneurons_KBoolean (*get_debug)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          const char* (*get_fawkesAppKey)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          libneurons_KLong (*get_fts)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          libneurons_KInt (*get_internalVersionCode)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          const char* (*get_oid)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          libneurons_KInt (*get_platform)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          const char* (*get_sessionId)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          libneurons_KInt (*get_uid)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          const char* (*get_version)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
          libneurons_KInt (*get_versionCode)(libneurons_kref_yylx_bilibili_kn_neurons_AppDelegate thiz);
        } AppDelegate;
```

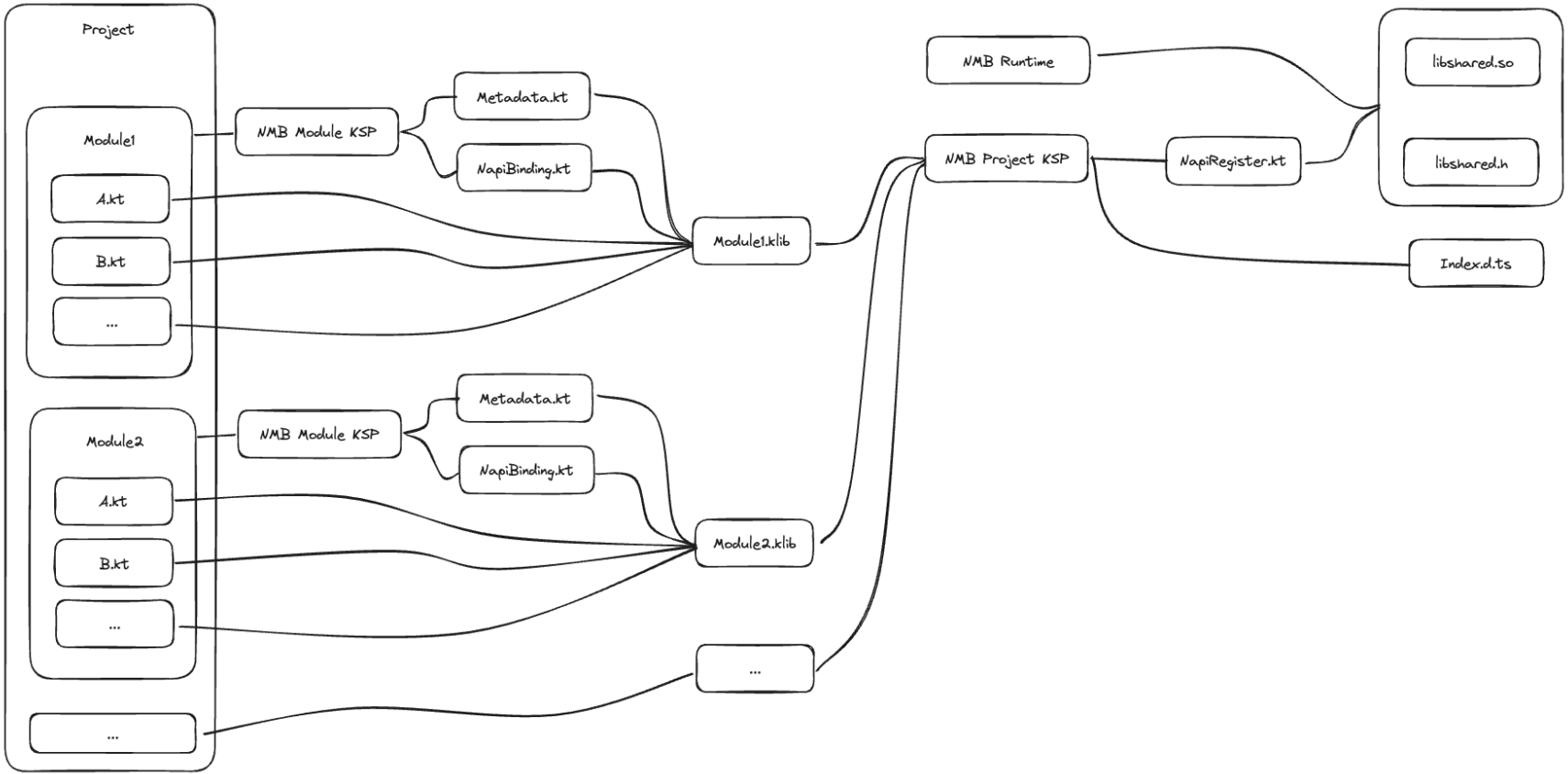
Kotlin Native 与 ArkTS 互操作问题

- Kotlin Native 生成的头文件比较难用
- 大量 Napi Binding 代码需要手动实现，费时且容易出错
- .d.ts 的声明文件需要手写

NMB (Napi Module Binding)

- NMB 是自动生成 Kotlin Napi Binding 的 KSP
- 自动化生成 .d.ts 声明
- 支持 Kotlin 内置类型的导出
- 支持自定义 class、interface、object、enum、function 等多种类型的导出
- suspend function 与 Promise 的自动转换
- 简化 Napi Register 的模版代码
- 支持模块化导出

NMB 工作流程


Project
Module1
A.kt
B.kt
...
NMB Module KSP
Metadata.kt
NapiBinding.kt
Module1.klib
Module2
A.kt
B.kt
...
NMB Module KSP
Metadata.kt
NapiBinding.kt
Module2.klib
...
...
NMB Runtime
NMB Project KSP
NapiRegister.kt
libshared.so
libshared.h
Index.d.ts

NMB Example

```kotlin
package yylx.bilibili.nmb.example

import kotlinx.coroutines.delay
import yylx.bilibili.kn.nmb.api.NMBExport
import yylx.bilibili.kn.nmb.api.NMBName

@NMBExport
class KtClass {
    val str = "I'm Kotlin"
    suspend fun wait(ms: Long) {
        delay(ms)
    }
}

@NMBExport
interface KtInterface {
    var whoAreU: String
}

@NMBExport
@NMBName( name: "ktPlus")
fun plus(a: Int, b: Int): Int  = a + b
```

Kotlin Code

```typescript
export namespace NMBExample {
  export class KtClass implements __NMBExport {
    readonly __doNotImplementDirectly: __DoNotUse;
    protected constructor(_: __DoNotUse);
    static create(): KtClass;
    readonly str: string;
    wait(ms: bigint): Promise<void>;
    equals(other: unknown | undefined): boolean;
    hashCode(): number;
    toString(): string;
  }
  export interface KtInterface extends __NMBExport {
    whoAreU: string;
  }
  export namespace KtInterface {
    export interface Impl {
      whoAreU: string;
    }
    export function create(impl: Impl): KtInterface;
  }
  export const ktPlus: (a: number, b: number) => number;
}
```

Index.d.ts

NMB Example

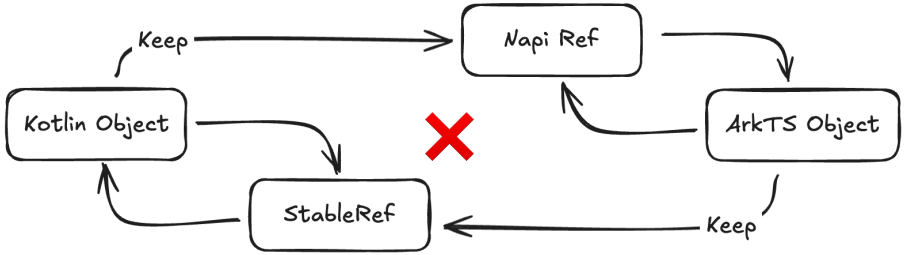
```cpp
#include "libexample_api.h"

extern "C" __attribute__((constructor)) void RegisterEntryModule(void)
{
    NMBInit();
}
```

Napi 注册

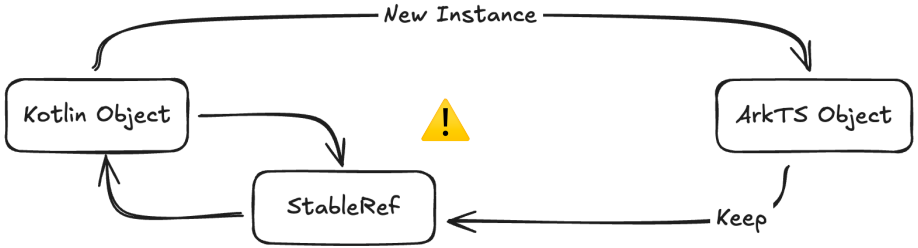
```
import { NMBExample } from 'libentry.so';

@Entry
@Component
struct Index {
  @State message: string = 'Hello World';

  build() {
    Row() {
      Column() {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(async () => {
            const kt = NMBExample.KtClass.create()
            await kt.wait(1000n)
            console.log(`${kt.str} from Kotlin Native:`, NMBExample.ktPlus(1, 2));
          })
      }
      .width('100%')
    }
    .height('100%')
  }
}
```

使用

NMB 内存管理

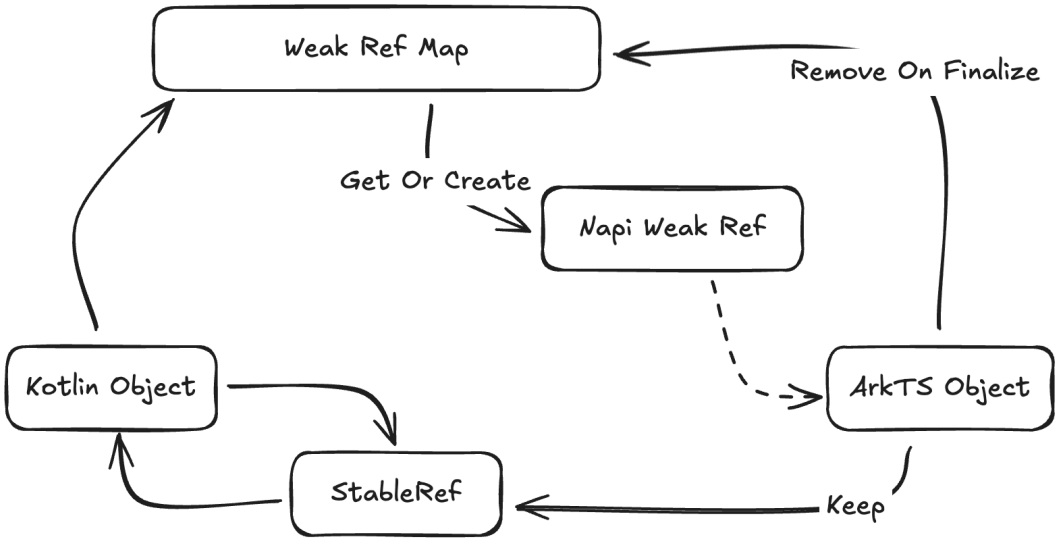


- 循环引用
- Kotlin、ArkTS 两边都无法 GC



- 同一个 Kotlin 对象可能对应多个不同的 ArkTS 对象
- 频繁创建对象带来的额外开销

NMB 内存管理



- 一一对应
- 不入侵原有类型结构
- Lifecycle: Kotlin Obj >= ArkTS Obj

NMB 支持类型

Kotlin	Int、Float、Double	Long	Boolean	String	Flow、StateFlow、MutableStateFlow
NMBExport	number	bigint	boolean	string	NMBFlow、NMBStateFlow、NMBMutableStateFlow

Kotlin	ByteArray	List、MutableList	Map、MutableMap	Set、MutableSet
NMBExport	NMBByteArray	NMBList、NMBMutableList	NMBMap、NMBMutableMap	NMBSet、NMBMutableSet

Kotlin	Custom Class	Custom Interface	Custom Enum	Custom Object	Custom Function
NMBExport	Custom Class	Custom Interface	Custom Enum	Custom Object	Custom Function

Kotlin Native + NMB + ArkUI

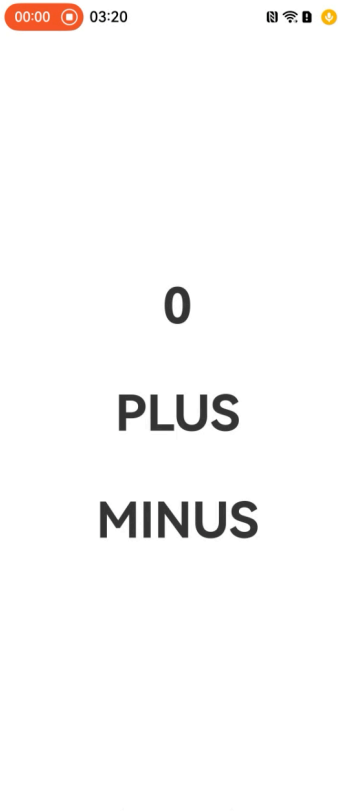
```kotlin
// commonMain
// State
data class State(val count: Int = 0)

// Action
@NMBExport
@NMBExport.NotImplementable
sealed interface Action
@NMBExport
data class Plus(val value: Int): Action
@NMBExport
data class Minus(val value: Int): Action

// Reducer
object Reducer {
    fun reduce(state: State, action: Action): State {
        return when (action) {
            is Plus -> state.copy(count = state.count + action.value)
            is Minus -> state.copy(count = state.count - action.value)
        }
    }
}

// Store
class Store(init: State) {
    private val mutableStateFlow = MutableStateFlow(init)
    val stateFlow by lazy { mutableStateFlow.asStateFlow() }

    fun dispatch(action: Action) {
        mutableStateFlow.value = Reducer.reduce(mutableStateFlow.value, action)
    }
}
```

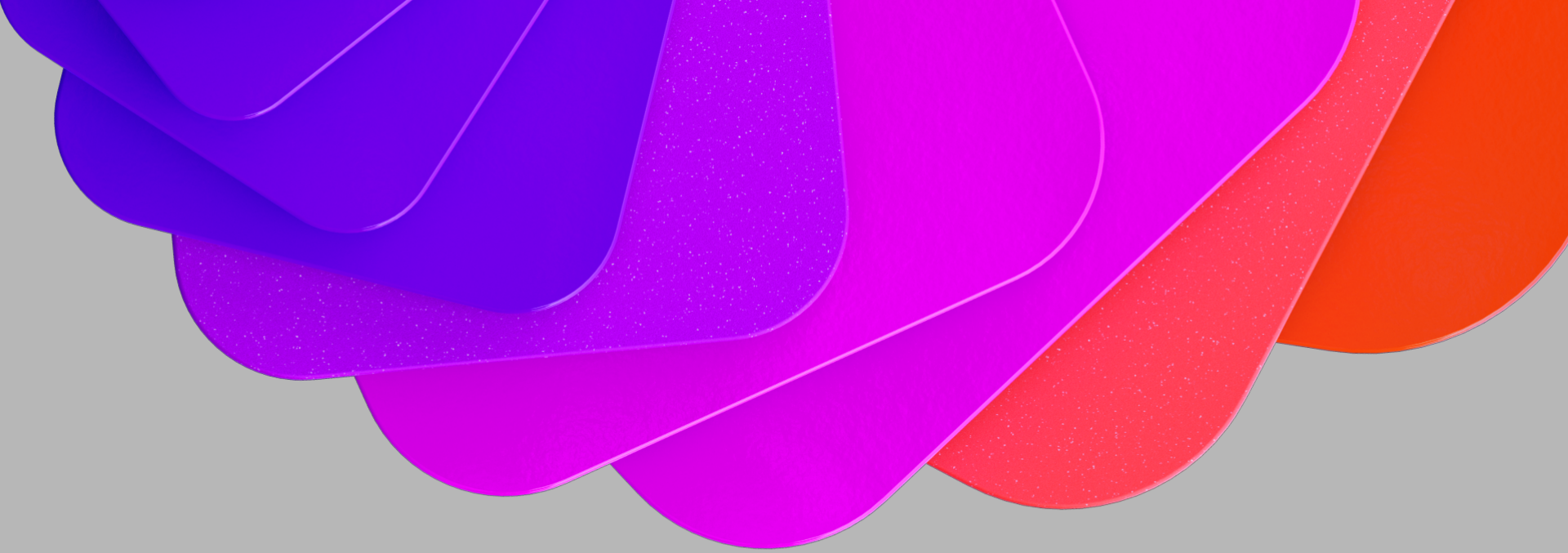
```kotlin
// harmonyMain
@NMBExport
interface ArkUIModelLike {
    var count: Int
}

@NMBExport
@NMBExport.NotImplementable
interface Dispatcher {
    fun dispatch(action: Action)
}

@NMBExport
fun createDispatcher(uiModel: ArkUIModelLike): Dispatcher {
    val store = Store(State(uiModel.count))
    MainScope().launch {
        store.stateFlow.distinctUntilChangedBy {
            it.count
        }.collect {
            uiModel.count = it.count
        }
    }
    return object : Dispatcher {
        override fun dispatch(action: Action) {
            store.dispatch(action)
        }
    }
}
```

Kotlin Native + NMB + ArkUI

```
@Entry
@ComponentV2
struct Index {
  @Local model: Model = new Model()
  private dispatcher = NMBExample.createDispatcher(NMBExample.ArkUIModelLike.create(this.model))

  build() {
    Row() {
      Column() {
        Text(this.model.count.toString())
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
        Blank().height(50)
        Text("PLUS")
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(() => {
            this.dispatcher.dispatch(NMBExample.Plus.create(1))
          })
        Blank().height(50)
        Text("MINUS")
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(() => {
            this.dispatcher.dispatch(NMBExample.Minus.create(1))
          })
      }.width('100%')
    }.height('100%')
  }
}
```

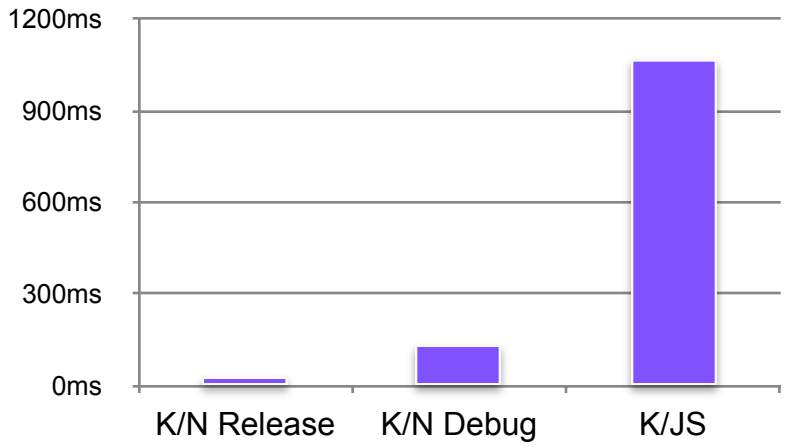
```
@ObservedV2
class Model implements NMBExample.ArkUIModelLike.Impl {
  @Trace count: number;

  constructor(count: number = 0) {
    this.count = count
  }
}
```





Kotlin Native vs. Kotlin JS

Performance



使用 kotlinx.serialization 反序列化 json 耗时

```
// commonMain
private val json = Json {
    ignoreUnknownKeys = true
    isLenient = true
}

fun decode(rawString: String): Model {
    return json.decodeFromString(rawString)
}

// harmonyMain
@NMBExport
@NMBName("jsonDecodeTest")
fun jsonDecodeTest(rawString: String) {
    decode(rawString)
}

// jsMain
@OptIn(ExperimentalJsExport::class)
@JsExport
@JsName("jsonDecodeTest")
fun jsonDecodeTest(rawString: String) {
    decode(rawString)
}
```

多线程并发能力

Kotlin Native

- 支持 kotlinx.coroutines，并具备真正的多线程能力
- 可以通过 kotlinx.coroutines 异步处理耗时任务，降低 UI 线程负载
- Kotlin 代码可以轻松实现多线程并发逻辑，降低编码负担

Kotlin JS

- 支持 kotlinx.coroutines，不具备多线程能力
- kotlinx.coroutines 的异步任务都执行在 UI 线程，占用 UI 绘制时间
- Kotlin 代码难以利用 ArkTS 提供的多线程并发能力，需要更多额外的代码来降低 UI 线程负载

生态问题

- HarmonyOS 环境与 Browser、Node 环境并不能完美兼容
- 大量 jsMain 的平台实现是基于Browser、Node 环境，环境差异导致的功能无法正常使用，没有办法在编码阶段暴露
- nativeMain 的平台实现大多基于 POSIX 接口，与 HarmonyOS 环境完美兼容，同时 harmonyMain 的平台适配逻辑部分可以复用 linuxMain
- ArkTS 不是 JS，虽然 Ark Runtime 可以运行 JS，但 JS 没有办法享受到 Ark Runtime 额外的性能优化
- 未来的 ArkTS 2.0 将会更加明显

木桶效应

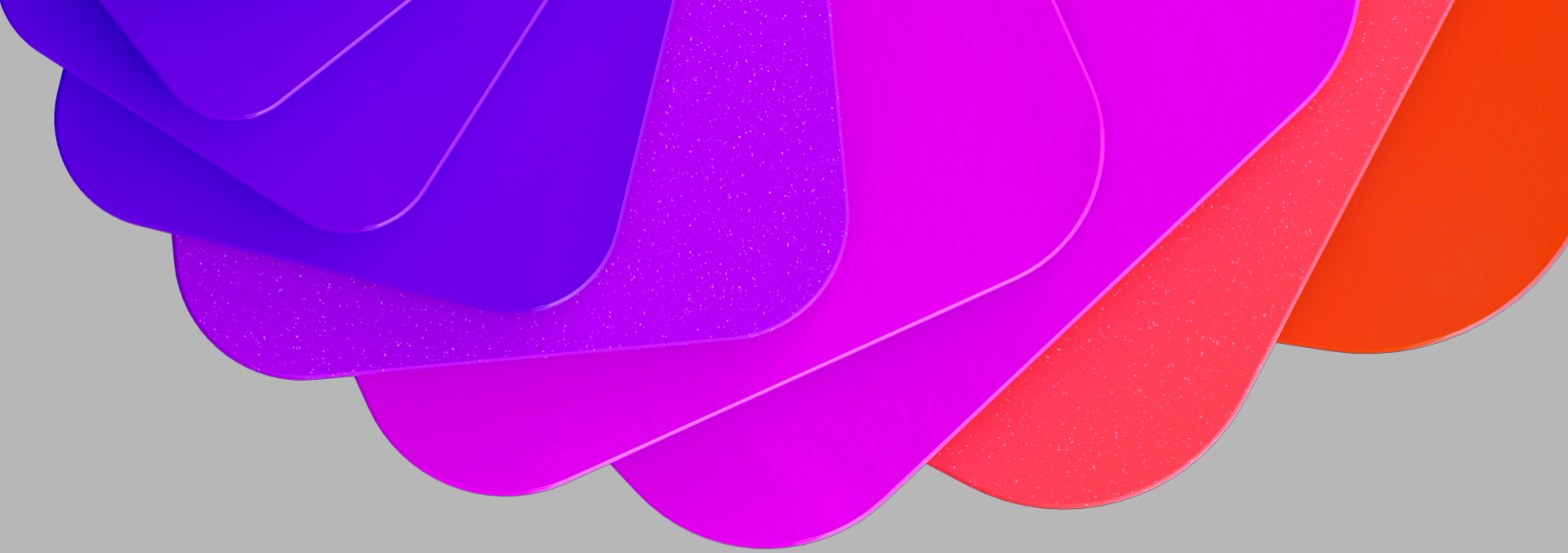
- 原本 B 站的 KMP 项目只有 iOS（Native）、Android（JVM）两类 Target
- 在加入 ohos（JS）后，由于之前提到的这些问题，导致有一部分原本实现在 commonMain 的逻辑被迫需要拆分，对于逻辑代码复用产生了负效应
- 额外的增加 Kotlin JS，无形的增加了跨端研发的门槛
- 公共逻辑实现只能取 JVM、Native、JS 三者最小集

Compose Multiplatform

- 为保障极致的用户体验，bilibili APP 在主场景中，都使用平台原生的 UI 框架进行开发
- Jetpack Compose 作为 Google 主推的 Android UI 框架，在 B 站已有一定基础建设
- Compose Multiplatform 作为跨平台的 UI 框架，对研发效率提升显著
- 使用 Compose Multiplatform 可以复用大量已有建设，解决多端 UI 不一致的现状
- 使用 Kotlin Native 将 Compose Multiplatform 适配到 Harmony 平台的成本远比使用 Kotlin JS 小的多得多
- Kotlin Native 能带来更好的渲染性能
- MVI 作为 B 站移动端主推的架构，天然适配 Compose Multiplatform

Kotlin Native vs. Kotlin JS

HarmonyOS	多线程	性能	平台能力	调试	代码复用	想象空间
Kotlin Native	⭐⭐⭐	⭐⭐⭐	⭐⭐	⭐	⭐⭐⭐	⭐⭐⭐
Kotlin JS	❌	⭐	⭐⭐⭐	⭐	⭐⭐	⭐



Join Us

bilibili 内推

加入我们 一起生产快乐吧！

专属内推码 69KP15

我是Vicky的饲养员
扫码投递岗位
加入哔哩哔哩

Thanks!
Have a
Nice Kotlin

Kotlin

zangzhicong@bilibili.com

问答环节

Kotlin

zangzhicong@bilibili.com