



Reactive Quarkus With Coroutine and MongoDB

蘇芃翰 (豬排)

電獺股份有限公司共同創辦人
兼副執行長

介紹一下我們公司 創業歷程影響技術選型



大部分人認識的電獺



3C 界的 AKB48

電獺是技術研發公司

- 2012 年成立, 2013 年推出「鄉民晚報」App(台版「今日頭條」)
- 試著分析鄉晚讀者累積的閱讀行為大數據
- 2016 推出 AotterTrek 廣告聯播網, 成立子公司 電豹 Aseal 專營廣告市場開發
- 台灣所有的 PTT 閱讀器 App 皆為聯播網渠道, 並擴及各大新聞媒體網站
- 如今是台灣最大 App 原生廣告 SSP, 江湖俗稱「鄉民聯播網」

技術挑戰！

- 鄉民有個習性...
- 地震了！先上八卦版！



2022/09/18 24 小時內四次地震
廣告主機負載圖

鄉民侯風地動儀

限制下的技術選型

- 新創很窮不能用鈔能力解決
- 團隊只會 Java
- 為了管理方便也「只能會 Java」
- 高效能極輕量的框架
- JVM 生態系
- Reactive Programming

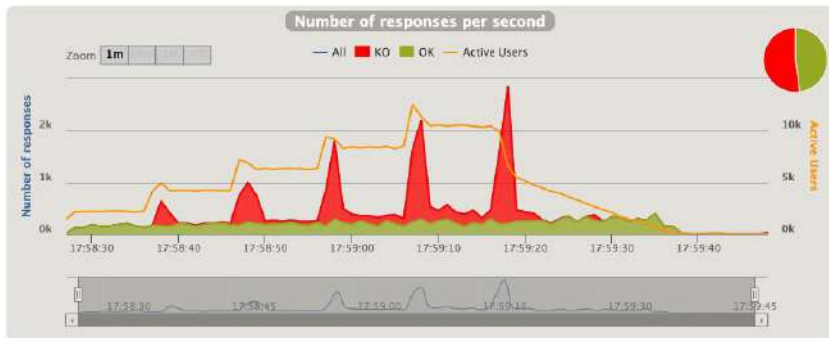
追榜的過程中，我們注意到了 Vert.x



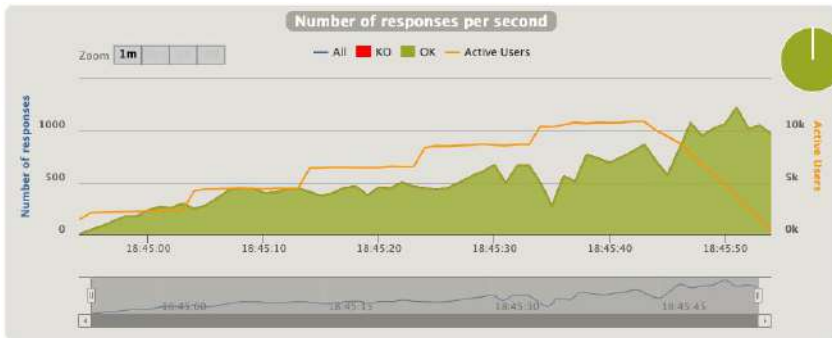
2017-2019 期間常衝上榜

Vert.x 是 jvm 版的 Node.js 真的有點猛

spring boot



vert.x



<https://github.com/b2etw/reactive-coroutine-performance-test>

2021 COSCUP Vincent 大大與我一起做的小測試

但有個小問題...

Vert.x 官方預設擁抱 callback style 的寫法

```
public void start() {  
    vertx.createHttpServer()  
        .requestHandler(handler)  
        .listen(8080, ar -> {  
            if (ar.succeeded()) {  
                logger.info("HTTP server started")  
            } else {  
                logger.warn("fail", ar.cause())  
            }  
        })  
    };  
}
```

真實世界應用情境，寫 Java 弄不好會變這樣

請 ChatGPT4 老師幫我們 demo:



You

請幫我撰寫 vertx 的模擬程式碼，撰寫一個 api 端點，做到以下：

1. 呼叫遠端 api 端點，取得 userId
2. 用 userId 查詢本地資料庫，取得 favList
3. 將 favList 回應給 api caller

```

@Override
public void start() {
    HttpServer server = vertx.createHttpServer();
    WebClient client = WebClient.create(vertx);

    Router router = Router.router(vertx);

    router.get("/api/favorites").handler(routingContext -> {
        // Step 1: Call remote API to get userId
        client.getAbs("http://example.com/api/userId").send(ar -> {
            if (ar.succeeded()) {
                String userId = ar.result().bodyAsString();

                // Step 2: Use userId to query local database for favList
                getFavoritesFromDatabase(userId).onComplete(dbResponse -> {
                    if (dbResponse.succeeded()) {
                        String favList = dbResponse.result();

                        // Step 3: Respond to API caller with favList
                        routingContext.response()
                            .putHeader("content-type", "application/json")
                            .end(favList);
                    } else {
                        routingContext.fail(dbResponse.cause());
                    }
                });
            } else {
                routingContext.fail(ar.cause());
            }
        });
    });

    server.requestHandler(router).listen(8080);
}

```

換 CompletableFuture 試試



You

change to completablefuture and use reactive chaining style

```
@Override
public void start() {
    HttpServer server = vertx.createHttpServer();
    WebClient client = WebClient.create(vertx);

    Router router = Router.router(vertx);

    router.get("/api/favorites").handler(routingContext -> {
        getUserId(client)
            .thenCompose(this::getFavoritesFromDatabase)
            .thenAccept(favList -> sendResponse(routingContext, favList))
            .exceptionally(error -> handleException(routingContext, error));
    });

    server.requestHandler(router).listen(8080);
}

private CompletableFuture<String> getUserId(WebClient client) {
    CompletableFuture<String> future = new CompletableFuture<>();
    client.getAbs("http://example.com/api/userId").send(ar -> {
        if (ar.succeeded()) {
            future.complete(ar.result().bodyAsString());
        } else {
            future.completeExceptionally(ar.cause());
        }
    });
    return future;
}
```

換 RxJava 試試



You

now change to rxjava

```

@Override
public void start() {
    HttpServer server = vertx.createHttpServer();
    WebClient client = WebClient.create(vertx);

    Router router = Router.router(vertx);

    router.get("/api/favorites").handler(routingContext -> {
        // Step 1: Call remote API to get userId
        getUserId(client)
            .flatMap(this::getFavoritesFromDatabase)
            .subscribe(
                favList -> sendResponse(routingContext, favList),
                error -> handleException(routingContext, error)
            );
    });

    server.requestHandler(router).listen(8080);
}

private Single<String> getUserId(WebClient client) {
    return Single.create(emitter ->
        client.getAbs("http://example.com/api/userId").send(ar -> {
            if (ar.succeeded()) {
                emitter.onSuccess(ar.result().bodyAsString());
            } else {
                emitter.onError(ar.cause());
            }
        })
    );
}

```

Java Reactive Programming 痛苦深淵

- reactive chaining 需要花時間理解
 - 到底我的 code 會不會跑, 還是要 subscribe 才會動
 - [chaining operator](#) 寫到心累
- 在 lamda expression 中噴錯了, 會無聲死亡...
 - 得小心翼翼處理錯誤
- JVM 中難道就沒有隔壁棚的 async, await 可以用嗎?

kotlin coroutine
來拯救勒~~



改寫成 kotlin coroutine

Vert.x 在很早期 (coroutine 還沒正式 release 前) 官方就出了 coroutine support 的 plugin



You

now use kotlin and coroutine to rewrite the code

```
override suspend fun start() {  
    val server = vertx.createHttpServer()  
    val client = WebClient.create(vertx)  
  
    val router = Router.router(vertx)  
    router.get("/api/favorites").coroutineHandler { ctx ->  
        try {  
            val userId = getUserId(client, ctx)  
            val favList = getFavoritesFromDatabase(userId)  
            sendResponse(ctx, favList)  
        } catch (e: Exception) {  
            handleException(ctx, e)  
        }  
    }  
  
    server.requestHandler(router).listen(8080)  
}  
  
private suspend fun getUserId(client: WebClient, ctx: RoutingContext): String {  
    val response = client.getAbs("http://example.com/api/userId").sendAwait()  
    return response.bodyAsString()  
}
```

有沒有很感動！

- 全攤平了！
- 不用去記憶一大堆 chaining operator 代表的意義
- 這樣就 reactive 了真的假的！
- 還可以 try catch!

More idiomatic!

```
override suspend fun start() {  
    val server = vertx.createHttpServer()  
    val client = WebClient.create(vertx)  
  
    val router = Router.router(vertx)  
    router.get("/api/favorites").coroutineHandler { ctx ->  
        try {  
            getUserId(client, ctx)  
                .let { getFavoritesFromDatabase(it) }  
                .let { sendResponse(ctx, it) }  
        } catch (e: Exception) {  
            handleException(ctx, e)  
        }  
    }  
  
    server.requestHandler(router).listen(8080)  
}  
  
private suspend fun getUserId(client: WebClient, ctx: RoutingContext): String = client  
    .getAbs("http://example.com/api/userId")  
    .sendAwait()  
    .bodyAsString()
```

我 OK! 先跳坑囉～！

- 找到了這個寫法之後，我們才敢在 production 上使用 Vert.x
- 2019 年，我們就有第一個 Vert.x-kotlin-coroutine 上 production
- 到這邊總結一下
 - 我們很窮，所以選 Vert.x
 - 我們想把 Vert.x 寫到方便維護，所以選 kotlin coroutine
 - 為了導入 coroutine，所以去學 kotlin
 - 然後就全公司後端都開始用 kotlin 了
 - 誰叫 java 動那麼慢，好啦他包袱多，不怪他

但 Vert.x 直接寫大專案，有點辛苦

- Vert.x 認為自己是 toolkit, 不是 framework
- 官方沒有提供 DI、CronJob 等等常見的工具
- router 要自己寫，風格類似 nodejs, java 人比較不習慣

Quarkus
來拯救勒~~



Quarkus 是個完整的 framework

- Vert.x 外面包一層, 性能一樣威猛
- 生態系整合完善
- 啊就很像 spring boot 的開發者體驗
- Red Hat 在罩的開源專案
- Say no more, ChatGPT4 老師請幫我再改寫一次:

Quarkus kotlin coroutine。如此優雅！

```
@Path("/api")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
class MyResource {

    @Inject
    lateinit var client: WebClient

    @GET
    @Path("/favorites")
    suspend fun getFavorites() = getUserId().let { getFavoritesFromDatabase(it) }

    private suspend fun getUserId() = client
        .getAbs("http://example.com/api/userId").send().await().indefinitely().bodyAsString()
}
```

提醒大家，最一開始是長這樣的

```
@Override
public void start() {
    HttpServer server = vertx.createHttpServer();
    WebClient client = WebClient.create(vertx);

    Router router = Router.router(vertx);

    router.get("/api/favorites").handler(routingContext -> {
        // Step 1: Call remote API to get userId
        client.getAbs("http://example.com/api/userId").send(ar -> {
            if (ar.succeeded()) {
                String userId = ar.result().bodyAsString();

                // Step 2: Use userId to query local database for favList
                getFavoritesFromDatabase(userId).onComplete(dbResponse -> {
                    if (dbResponse.succeeded()) {
                        String favList = dbResponse.result();

                        // Step 3: Respond to API caller with favList
                        routingContext.response()
                            .putHeader("content-type", "application/json")
                            .end(favList);
                    } else {
                        routingContext.fail(dbResponse.cause());
                    }
                });
            } else {
                routingContext.fail(ar.cause());
            }
        });
    });

    server.requestHandler(router).listen(8080);
}
```

假設今天你被推坑成功了

- 有幾個雷先講
- Quarkus 為了支援 java user, reactive 方面用了 Mutiny 函式庫
 - Uni 類似 RxJava Single
 - Multi 類似 RxJava Flowable
- Quarks 正在逐漸加強對 coroutine 的支援, 但 Mutiny 依舊還是 first class citizen
 - 有時候一些 built-in annotation 碰上 suspend function 會怪怪的
 - 舉個例子

```
@Path("/api")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
class MyResource {

    @Inject
    lateinit var client: WebClient

    @GET
    @Path("/favorites")
    @CacheResult // 這個會爆炸
    suspend fun getFavorites() = getUserId().let { getFavoritesFromDatabase(it) }

    private suspend fun getUserId() = client
        .getAbs("http://example.com/api/userId").send().await().indefinitely().bodyAsString()
}
```

```
@Path("/api")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
class MyResource {

    @Inject
    lateinit var client: WebClient

    @GET
    @Path("/favorites")
    @CacheResult // 這樣就不會爆炸
    fun getFavorites(): Uni<String> = getUserId()
        .onItem().transformToUni { getFavoritesFromDatabase(it) }

    private fun getUserId(): Uni<String> = client.getAbs("http://example.com/api/userId")
        .send()
        .onItem().transform { it.bodyAsString() }

}
```

但熟悉的 reactive chaining 又冒回來了...

介紹實用小撇步，繼續快樂 coroutine

Kotlin extension function 好好用

```
fun <T> Vertx.uni(fn: suspend () -> T): Uni<T> =  
    CoroutineScope(vertx.dispatcher())  
        .async { fn() }  
        .asUni()
```

```

@Path("/api")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
class MyResource {

    @Inject
    lateinit var client: WebClient

    @Inject
    lateinit var vertx: Vertx

    @GET
    @Path("/favorites")
    @CacheResult // 這個不會爆炸
    fun getFavorites() = vertx.uni { getUserId().let { getFavoritesFromDatabase(it) } }

    private suspend fun getUserId() = client
        .getAbs("http://example.com/api/userId").send().await().indefinitely().bodyAsString()
}

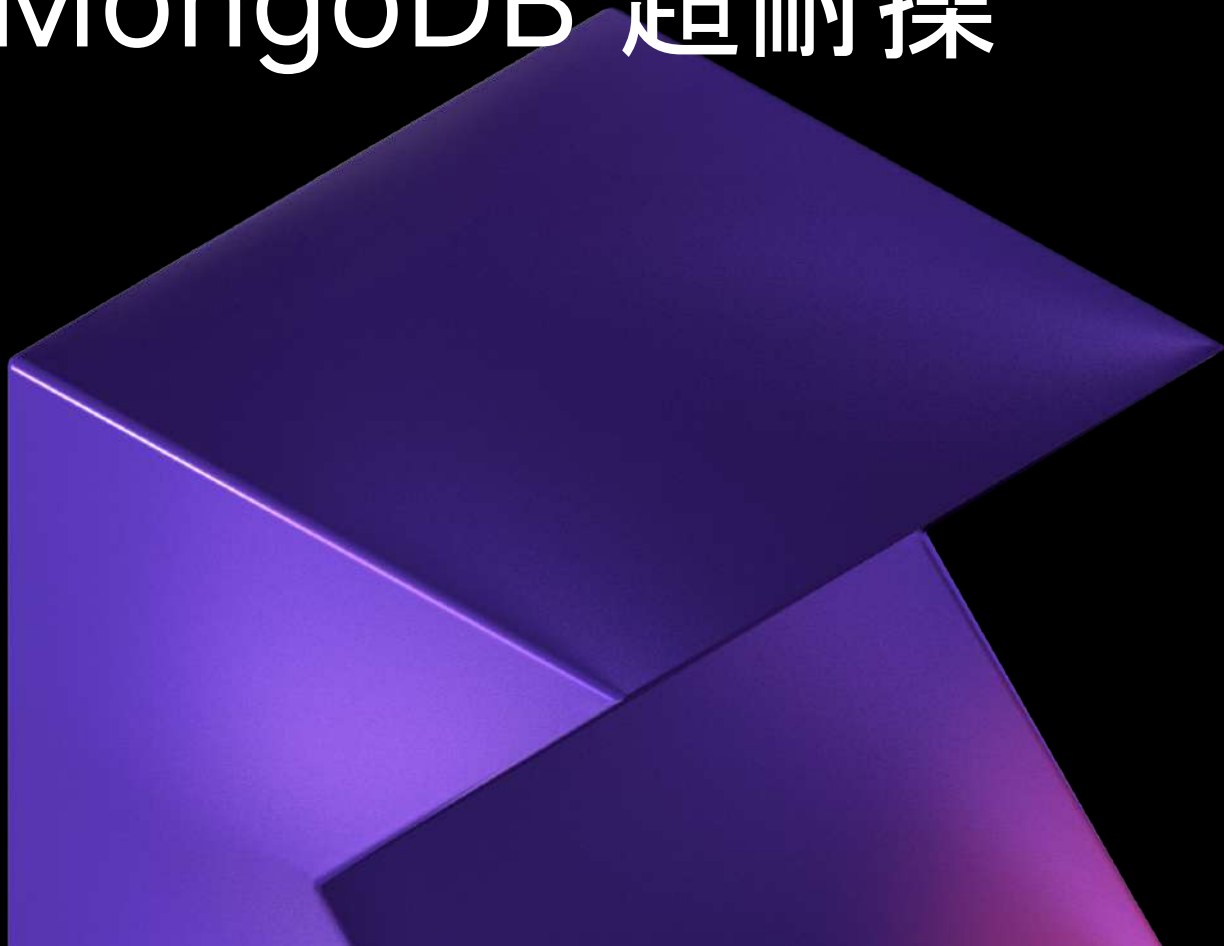
```

reactive chaining 掰掰慢走不送！

接下來...

- 好叻, 現在我的 service 跑起來很威猛了
- 還有一個地方可能會是效能瓶頸
- 嘿對, 資料庫那端

MongoDB 超耐操



廣告投遞系統的課題 - 資料庫端

- 高寫入的系統, 先抓每分鐘 100 萬次 db access 好了
- 我們常需要回答這類的問題:
 - 某個廣告, 在某個版位, 今天截至目前為止, 產生了多少曝光數、點擊數
- 最直覺的寫法: 把每個事件都存起來
- 然後? 做 grouping / aggregation
 - High cardinality 的資料, 叫 DB 做 distinct count 非常耗資源!
 - 怎麼辦勒?
 - Spark? Flink? Data warehouse solution?
 - 沒錢啦!

Why MongoDB

- 開源免錢效能高，自架主機沒煩惱，還有雲端可選擇，彈性自由方便好
- 耐操，操掛了通常是我的錯，不是 MongoDB 的錯
 - 例如忘記下 index
 - 或是沒有用 bulk operation 增進效能
 - 調教好之後 MongoDB 沒讓我失望過
- 官方直接提供 java reactive stream 版本的 driver
 - 可以橋接成包含 coroutine 在內的各種 reactive 函式庫
 - 與 vert.x 的相性很吻合，真正實踐「一路暢通」non blocking 到底

來練習實戰這題

- 某個廣告，在某個版位，今天截至目前為止，產生了多少曝光數、點擊數
 - ◉ ~~每個事件存起來，之後再 grouping~~
 - 當事件來的時候，在 DB 中找到該粒度的資料，並且將事件 count +1
 - 誒等等，找到該粒度的資料，並且將事件 count +1，細思極恐
 - Danger! Racing condition warning!
 - RDBMS 會怎麼做？
 - Yes, lock
 - Dead lock 惡夢, wait for lock 好久, Transaction ctrl 要熟悉...

MongoDB Atomic Update

- In MongoDB, a write operation is atomic on the level of a single document, even if the operation modifies multiple embedded documents within a single document.

某個廣告，在某個版位，今天截至目前為止，產生了多少曝光數、點擊數

```
db.DailyStatistic.updateOne(  
  {  
    "adUnitId": adUnitId,  
    "placeUid": placeUid,  
    "date": date  
  },  
  {  
    $inc: {"impression": 1}  
  },  
  {  
    upsert: true  
  }  
)
```

這三個 field 要下
unique index

This is atomic

如果噴錯，可以樂觀
retry 一次

Quarkus kotlin coroutine 寫法

這裡使用 quarkus 包好的 mongo driver

```
suspend fun updateDailyStatistic(adUnitId: String, placeUid: String, date: Date) = col.updateOne(
    Filters.and(
        Filters.eq("adUnitId", adUnitId),
        Filters.eq("placeUid", placeUid),
        Filters.eq("date", date)
    ),
    Updates.inc("impression", 1),
    UpdateOptions().upsert(true)
) // 到這裡回應的是 Uni<Unit>
.awaitSuspending() // 轉成 coroutine suspend function
```

不用擔心 lock, racing condition 等, 速度又非常快

Kotlin 語法寫 MongoDB 應用很方便

假設有份 collection 專放文章 (Post), 我們來寫一個 query posts function, 吃 authorId, category, state 三個參數。如果該參數有帶值, 則加入查找條件。如果所有參數都沒帶值, 則查詢所有資料

Java 實作

```
public List<Post> query(String authorId, String category, String state) {  
    List<Bson> filters = new ArrayList<>();  
  
    if (authorId != null && !authorId.isBlank()) {  
        filters.add(Filters.eq("authorId", authorId));  
    }  
    if (category != null && !category.isBlank()) {  
        filters.add(Filters.eq("category", category));  
    }  
    if (state != null && !state.isBlank()) {  
        filters.add(Filters.eq("state", state));  
    }  
  
    Bson queryFilter = filters.isEmpty() ? new Document() : Filters.and(filters);  
  
    return col.find(queryFilter).asList()  
}
```

Kotlin 實作

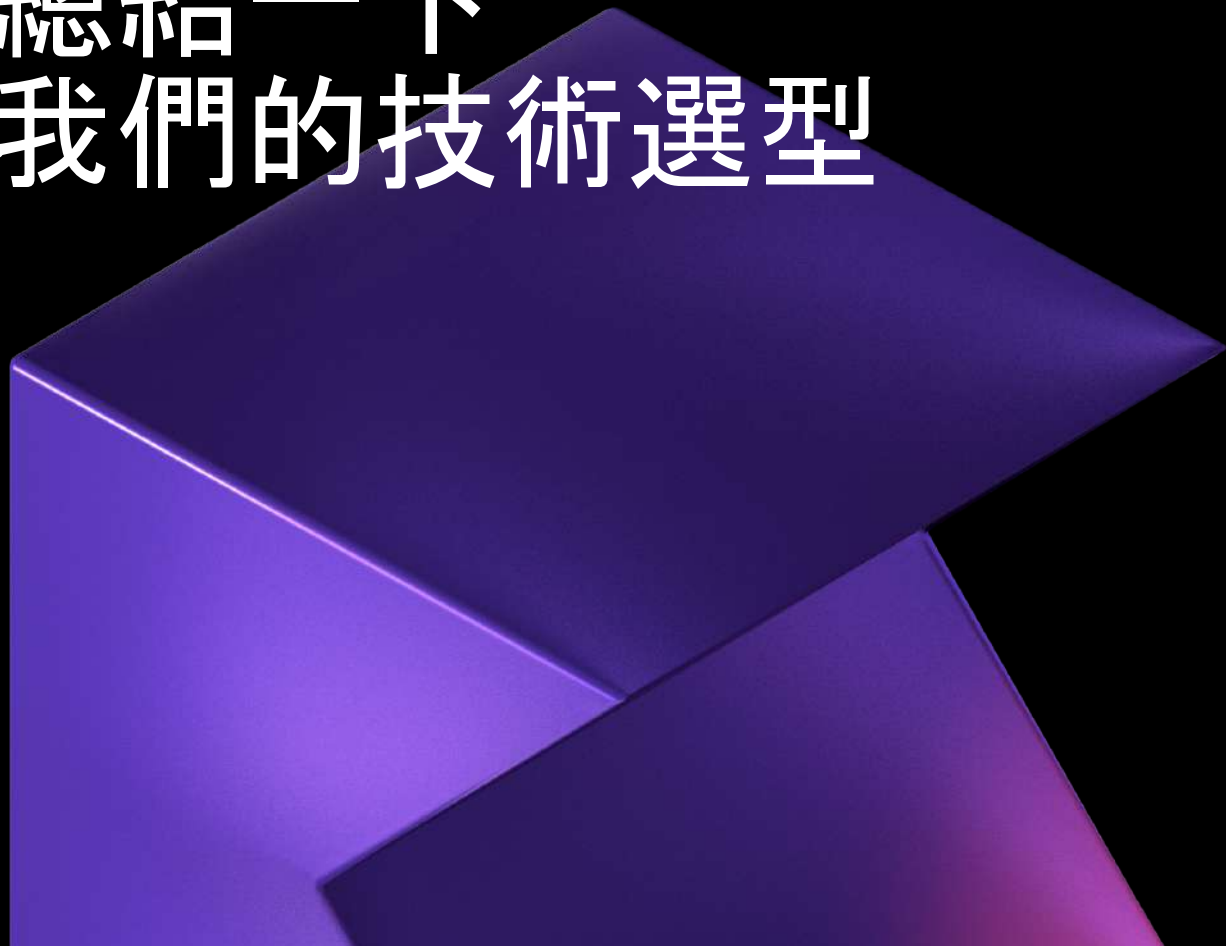
```
suspend fun query(authorId: String?, category: String?, state: String?): List<Post> = listOfNotNull(
    authorId?.takeIf { it.isNotBlank() }?.let { Filters.eq("authorId", it) },
    category?.takeIf { it.isNotBlank() }?.let { Filters.eq("category", it) },
    state?.takeIf { it.isNotBlank() }?.let { Filters.eq("state", it) }
)
    .takeIf { it.isNotEmpty() }
    ?.let { col.findAsList(Filters.and(it)) }
    ?: col.findAsList()
```

還能更漂亮 - ext function

```
fun String?.toBsonEqOrNull(field: String): Bson? = this
    ?.takeIf { it.isNotBlank() }
    ?.let { Filters.eq(field, it) }
```

```
suspend fun query(authorId: String?, category: String?, state: String?): List<Post> = listOfNotNull(
    authorId.toBsonEqOrNull("authorId"),
    category.toBsonEqOrNull("category"),
    state.toBsonEqOrNull("state")
)
    .takeIf { it.isNotEmpty() }
    ?.let { col.findAsList(Filters.and(it)) }
    ?: col.findAsList()
```

總結一下 我們的技術選型



因為窮，又只會 Java

- 選 Vert.x 節省主機開銷
- 選 kotlin coroutine 才能寫出好維護的 vert.x 專案，少踩坑就是省錢
- 導入 Quarkus 讓工程師有類似 spring boot 的開發體驗，省時間就是省錢
- 選用 MongoDB 搭建 full reactive stack, 同時也很省錢

電獺貢獻的 Quarkus 學習資源

- 電獺開源兩個小專案，希望可以對大家有幫助：
 - 教學專案：[Step-by-step 完成一個 部落格系統](#)
 - tech stack: kotlin-coroutine-mongodb
 - resource/service/repository 分層
 - Fontend
 - qute: public pages, server side render
 - vue-cli: admin console, client side render SPA
 - credits mainly to 國哲, thank you Joyce and Vera
- 應用範例：[The missing documentations](#)

Member credits - 感謝貢獻的團隊成員

- 廣告系統後端 / Infra
 - Ryan, SmallMouth, Vera, 國哲, YC
- 廣告系統前端 / web sdk
 - Steven, Cynthia, John
- App SDK
 - Anthony, SW, Robert, Charlotte
- 專案組
 - YC, Joyce, Dion
- Special Thanks
 - Vincent 大大

Thanks!

Have a nice Kotlin



@kotlin | Developed by JetBrains