# 潜力无限

## Kotlin 用于服务端与 WebAssembly

贾彦伟

# Kotlin Everywhere

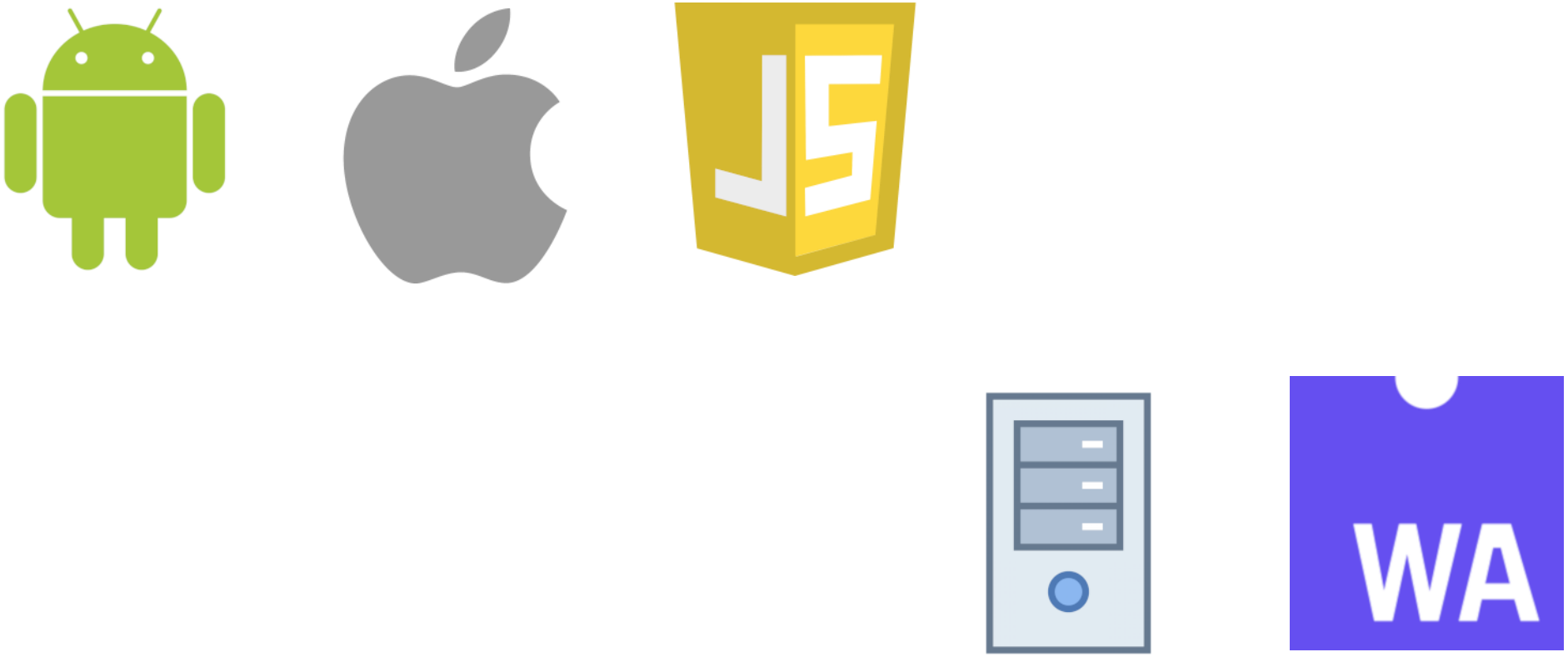
# Kotlin Everywhere

服务端

# Kotlin Everywhere

WA WebAssembly

# Kotlin Everywhere



# 潜力无限

# 服务端现状

不易推行

# WebAssembly 现状


第三梯队

# 服务端的潜力与优势

1. 既有生态
2. 现代特性

# WebAssembly 潜力与优势

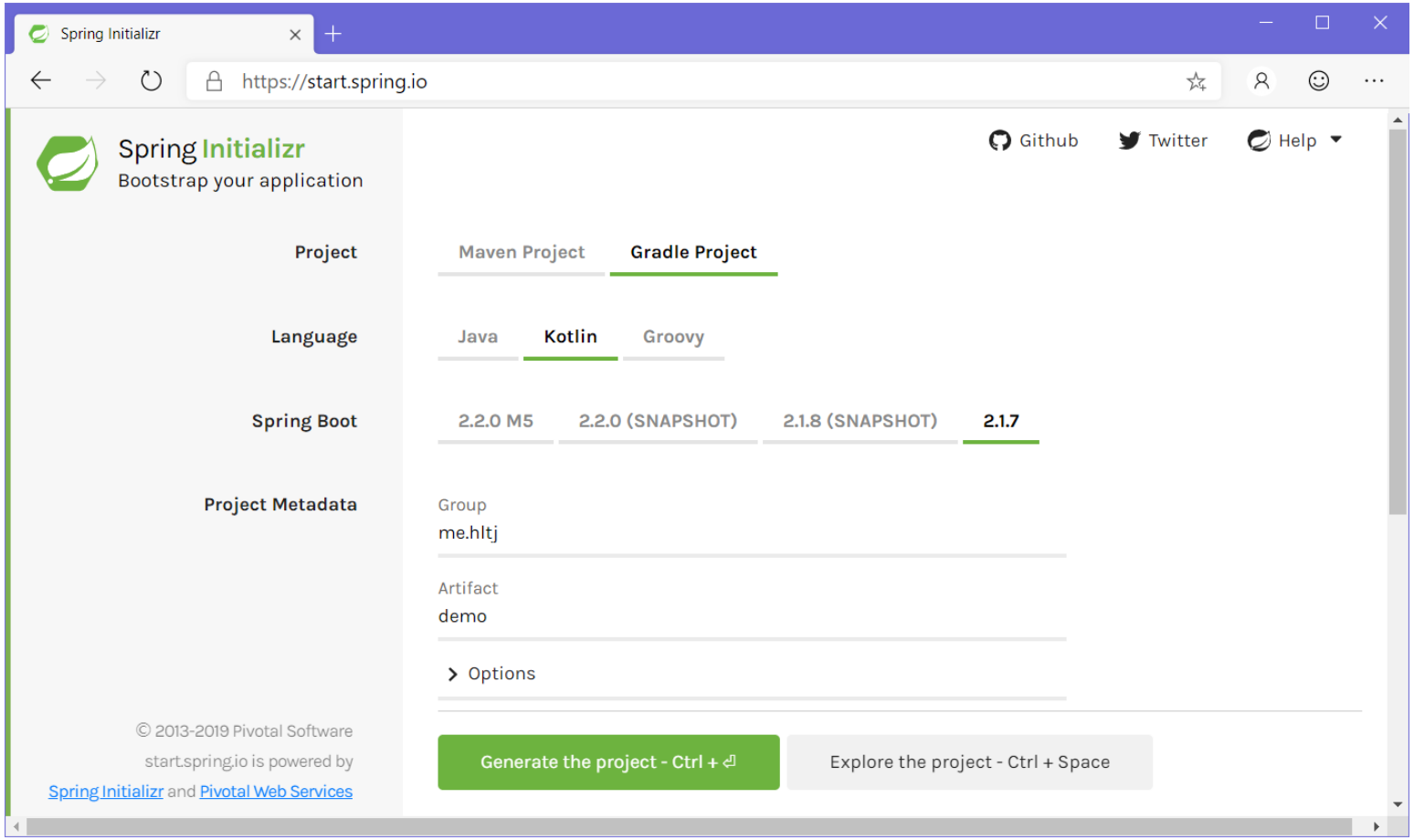

1. 平台潜力
2. 官方重视

# 服务端潜力与优势

与 Java 生态
无缝融合

充分利用
Kotlin 现代特性

# 服务端 Java 生态： Spring



# 服务端 Java 生态： Spring

```
@RestController
class MyController {
    companion object {
        private val logger = LoggerFactory.getLogger(MyController::class.java)
    }

    @RequestMapping( ...value: "/", method = [GET])
    fun sample(): String {
        MDC.put( key: "request_id", newRequestId())
        logger.info("enter")

        HttpAsyncClients.createDefault().use { it: CloseableHttpAsyncClient!
            it.start()

            it.execute(HttpGet( uri: "http://127.0.0.1:8081/"), object: FutureCallback<HttpResponse> {
                override fun cancelled() { logger.warn("canceled") }

                override fun completed(result: HttpResponse?) { logger.info("completed") }

                override fun failed(ex: Exception?) { logger.error("failed") }

            }).get()?.let { result ->  return result.statusLine.reasonPhrase }
        }

        return "-- nothing --"
    }
}
```

# Java 生态：Spring

- 可空性
  - @RequestParam name: String
  - @RequestParam name: String?

  - @Autowired lateinit var foo: Foo
  - @Autowired lateinit var foo: Foo?

# 服务端 Java 生态：Spring

- DSL

```
GenericApplicationContext context = new GenericApplicationContext();
context.registerBean(Foo.class);
context.registerBean(Bar.class, () -> new
        Bar(context.getBean(Foo.class))
);
```

# 服务端 Java 生态：Spring

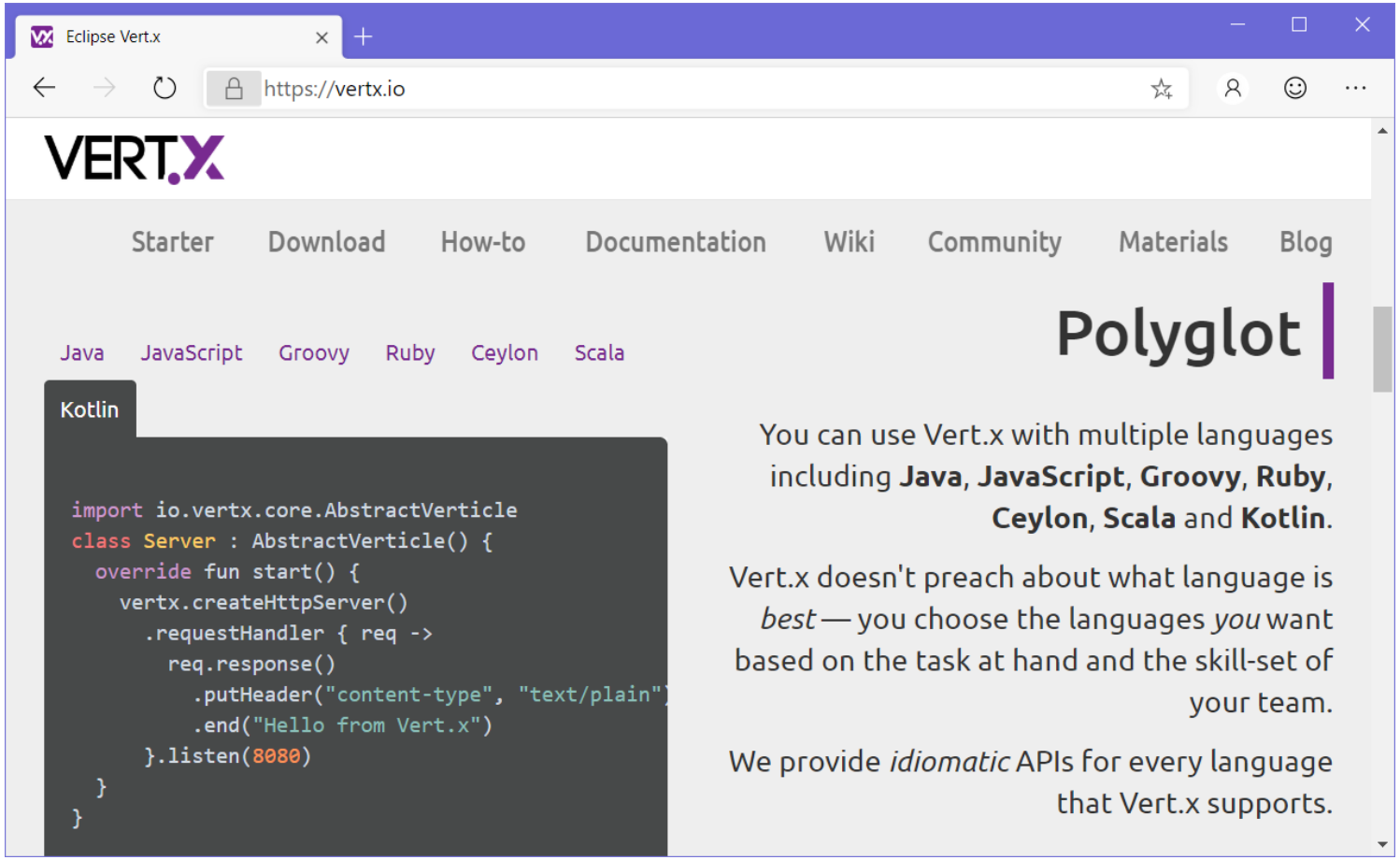
- DSL

```
beans {
    bean<Foo>()
    bean { Bar(ref()) }
}
```

# Kotlin 专用： Spring Fu

```kotlin
val app = application(WebApplicationType.SERVLET) {
    beans {
        bean<SampleService>()
        bean<SampleHandler>()
    }
    webMvc {
        port = if (profiles.contains("test")) 8181 else 8080
        router {
            val handler = ref<SampleHandler>()
            GET("/", handler::hello)
            GET("/api", handler::json)
        }
        converters {
            string()
            jackson()
        }
    }
}

fun main() {
    app.run()
}
```

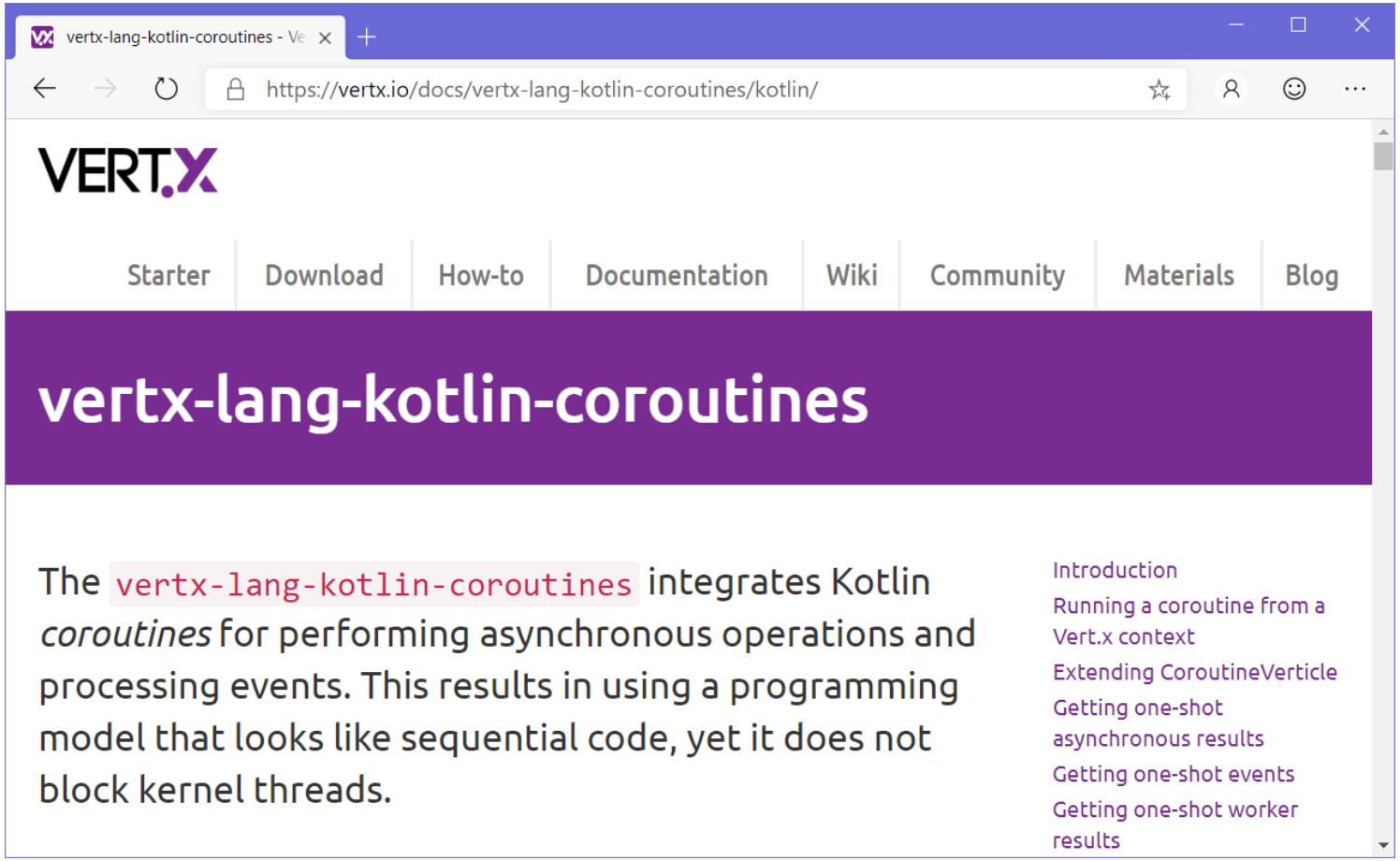
# 服务端 Java 生态： Vert.x



# 服务端 Java 生态： Vert.x

```
class MyVerticle : AbstractVerticle() {
    companion object {
        private val logger = LoggerFactory.getLogger(MyVerticle::class.java)
    }

    override fun start() {
        vertx.createHttpServer()
            .requestHandler { it: HttpServerRequest!
                MDC.put( key: "request-id", newRequestId())

                logger.info("request arrived")

                WebClient.create(Vertx.vertx()).getAbs( absoluteURI: "http://localhost:8081/").send { ar ->
                    val body = ar.result().bodyAsString()
                    logger.info("the upstream body is: {}", body)

                    it.response().end( chunk: "the upstream body size is: ${body.length}")
                }
            }.listen( port: 8080)
    }
}
```

# 服务端 Java 生态： Vert.x



# 服务端 Java 生态： Vert.x

```
suspend fun awaitingFuture() {
    val httpServerFuture = Future.future<HttpServer>()
    vertx.createHttpServer()
        .requestHandler { req -> req.response().end("Hello!") }
        .listen(8000, httpServerFuture)

    val httpServer = httpServerFuture.await()
    println("HTTP server port: ${httpServer.actualPort()}")

    val result = CompositeFuture.all(httpServerFuture, httpServerFuture).await()
    if (result.succeeded()) {
        println("The server is now running!")
    } else {
        result.cause().printStackTrace()
    }
}
```

# 原生 Kotlin 服务框架： Ktor


用 Kotlin 开发互联应用

# 原生 Kotlin 服务框架： Ktor

```
fun main(args: Array<String>) {
    val server = embeddedServer(Netty, port = 8080) {
        routing {
            get("/") {
                call.respondText("Hello World!", ContentType.Text.Plain)
            }
            get("/demo") {
                call.respondText("HELLO WORLD!")
            }
        }
    }
    server.start(wait = true)
}
```

# 原生 Kotlin 服务框架： Ktor

```
get("/") {
    val data = IndexData(listOf(1, 2, 3))
    call.respondHtml {
        head {
            link(rel = "stylesheet", href = "/static/styles.css")
        }
        body {
            ul {
                for (item in data.items) {
                    li { +"$item" }
                }
            }
        }
    }
}
```

# 原生 Kotlin 服务框架： Ktor

```
routing {
    webSocket("/chat") { // this: DefaultWebSocketSession
        while (true) {
            val frame = incoming.receive() // suspend
            when (frame) {
                is Frame.Text -> {
                    val text = frame.readText()
                    outgoing.send(Frame.Text(text)) // suspend
                }
            }
        }
    }
}
```

# 原生 Kotlin 服务框架： Ktor

```
get("/public/forum/posts") {
    val posts = httpClient.get<List<Post>>("http://localhost:8080/atom/forum/posts")

    val authorIds = posts.map { post -> post.authorId }.joinToString(separator = ",")

    val users = httpClient.get<List<User>>("http://localhost:8080/atom/user/users?ids=$authorIds")
        .map { user -> user.id to user }.toMap()

    call.respond(posts.map { post ->
        SimplePost(
            id = post.id,
            title = post.title,
            authorName = users[post.authorId]?.name ?: "无名氏"
        )
    })
}
```

# 原生 Kotlin 服务框架： Ktor

```
get("/public/forum/posts/{id}") {
    val postId = call.parameters["id"]

    val deferredPost = async {
        httpClient.get<List<Post>>("http://localhost:8080/atom/forum/posts?ids=$postId")[0]
    }

    val deferredReplies = async {
        httpClient.get<List<Reply>>("http://localhost:8080/atom/forum/replies?post_id=$postId")
    }

    val post = deferredPost.await()
    val deferredUser = async {
        httpClient.get<List<User>>("http://localhost:8080/atom/user/users?ids=${post.authorId}")[0]
    }

    val user = deferredUser.await()
    call.respond(
        DetailedPost(
            id = post.id,
            author = SimpleUser(user.id, user.name),
            title = post.title,
            content = post.content,
            replies = deferredReplies.await()
        )
    )
}
```

# 原生 Kotlin 服务框架： Ktor

## Kthumbor - a thumbnail service

language Kotlin | License AGPL v3 | build passing | hits 207

## Kthumbor 缩略图服务

语言 Kotlin | 授权许可 AGPL v3 | 构建 passing | hits 207

Kthumbor is an HTTP thumbnail service application implemented with Kotlin & Ktor.

Kthumbor 是用 Kotlin 与 Ktor 实现的 HTTP 缩略图服务程序。

### Build Status 构建状态

| | Java 12 | Java 11 | Java 8 |
|---|---|---|---|
| Ubuntu 18 | build passing | build passing | - |
| Ubuntu 16 | build passing | build passing | build passing |
| macOS | build passing | build passing | build passing |
| Windows | build passing | build passing | build passing |

# 原生 Kotlin SQL 库

JB team | TC build failed | Download 0.17.2 | license Apache License 2.0

## Exposed - Kotlin SQL Library

*Exposed* is a prototype for a lightweight SQL library written over JDBC driver for Kotlin language. It does have two layers of database access: typesafe SQL wrapping DSL and lightweight data access objects

# 原生 Kotlin SQL 库

编辑本页

Ktorm

KOTLIN ORM LIB WITH SQL DSL

build passing | Maven Central v2.5 | license Apache 2 | code quality A | awesome kotlin

## Ktorm 是什么？

Ktorm 是直接基于纯 JDBC 编写的高效简洁的轻量级 Kotlin ORM 框架，它提供了强类型而且灵活的 SQL DSL 和方便的序列 API，以减少我们操作数据库的重复劳动。当然，所有的 SQL 都是自动生成的。Ktorm 基于 Apache 2.0 协议开放源代码，源码托管在 GitHub，如果对你有帮助的话，请留下你的 star：vincentlauvlwj/Ktorm Stars 200

# 原生 Kotlin 异步 SQL


chat on gitter | Download 1.0.6 | Maven Central v1.0.6 | build passing | license Apache V.2 | codecov 80% | awesome kotlin

jasync-sql is a Simple, Netty based, asynchronous, performant and reliable database drivers for PostgreSQL and MySQL written in Kotlin.

# 更多资源

https://github.com/KotlinBy/awesome-kotlin

## Web Back ⇑

- ktorio/ktor - Web backend framework for Kotlin. Easy to use, fun and asynchronous.
- TinyMission/kara - Web framework written in Kotlin.
- http4k/http4k - Toolkit for serving and consuming HTTP services in a functional and consistent way.
- jean79/yested - A Kotlin framework for building web applications in Javascript.
- hhariri/wasabi - An HTTP Framework built with Kotlin for the JVM.
- Kotlin/kotlinx.html - Kotlin DSL for HTML.
- MarioAriasC/KotlinPrimavera - Spring support libraries for Kotlin.
- kohesive/kovert - An invisible, super easy and powerful REST and Web framework over Vert.x or Undertow.
- pgutkowski/KGraphQL - A GraphQL implementation written in Kotlin
- taskworld/kraph - GraphQL request string builder written in Kotlin
- sepatel/tekniq - Full-feature HTTP DSL Framework, HTTP Client, JDBC DSL, Loading Cache and Configuration
- vert-x3/vertx-lang-kotlin - This module provides Kotlin language bindings including DSL and extension functions for vert.x 3
- jooby-project/jooby - Modular micro web framework for Java and Kotlin
- gimlet2/kottpd - REST framework in pure Kotlin, inspired by spark-java
- kwebio/core - kweb.io Build rich live-updating web apps in pure server-side Kotlin.

# WebAssembly 潜力与优势

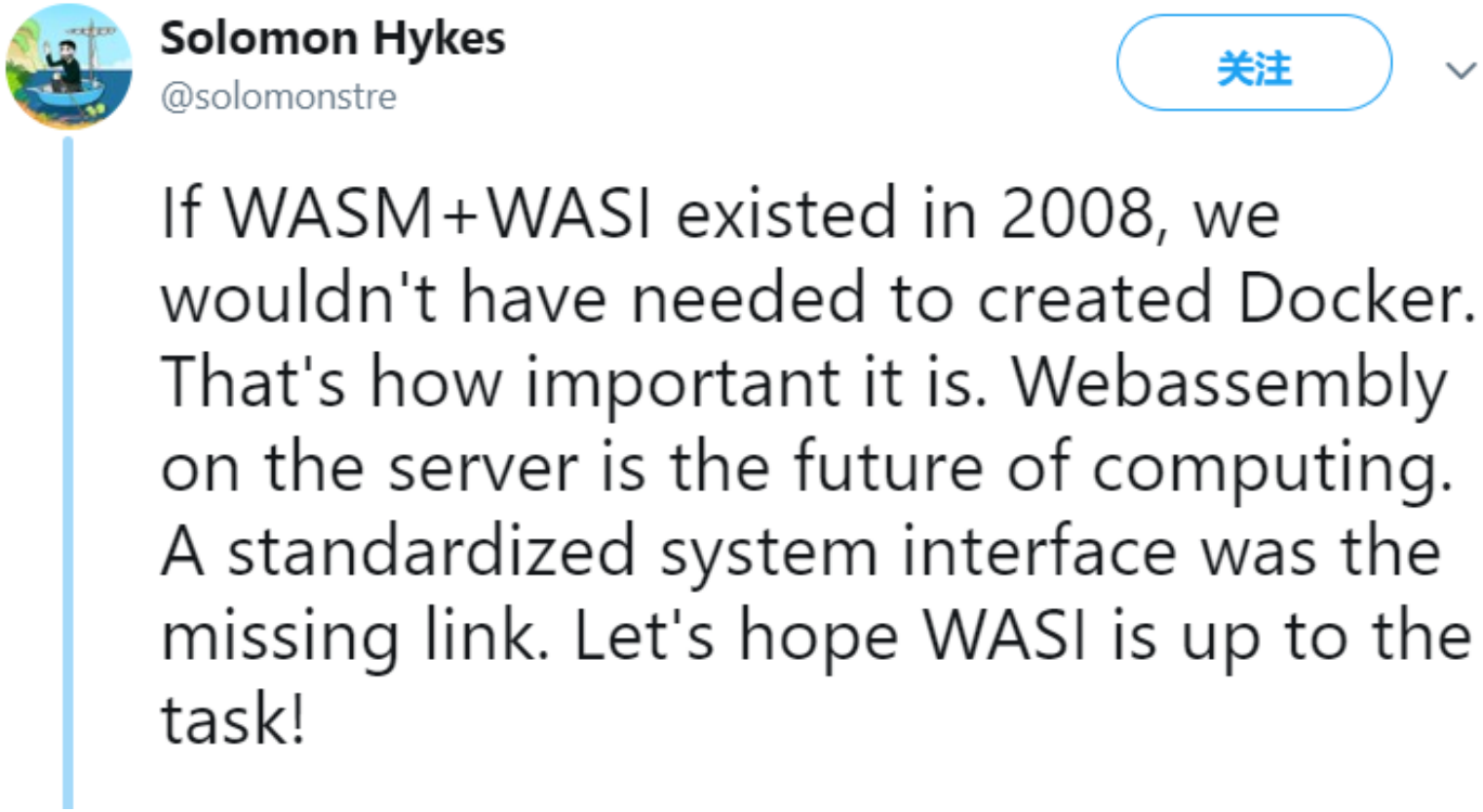
WebAssembly
自身潜力

JetBrains
官方重视

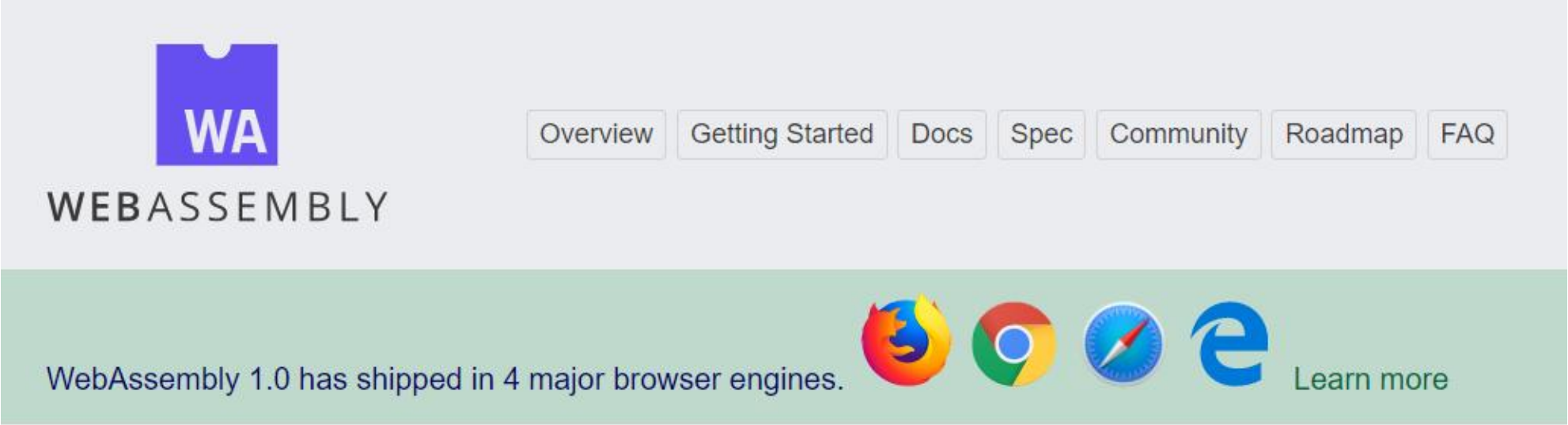
# WebAssembly：平台潜力

The Next Big Platform

# WebAssembly： the Next Big Platform



# WebAssembly



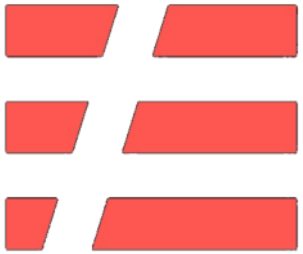

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

# WebAssembly

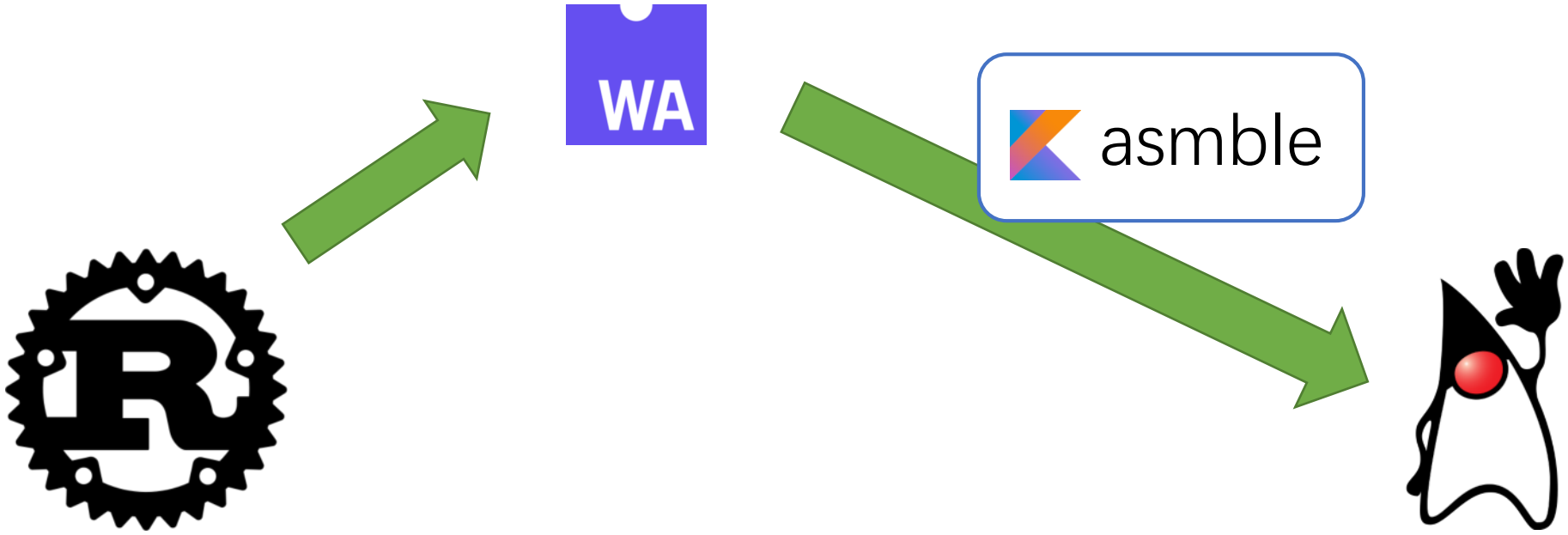
```
(module
    (func $sum (export "sum") (param $i i32) (result i32)
     (local $c i32)
        get_local $i
        i32.const 1
        i32.le_s
        if
            get_local $i
            set_local $c
        else
            get_local $i
            i32.const 1
            i32.sub
            call $sum
            get_local $i
            i32.add
            set_local $c
        end
        get_local $c
    )
)
```

# WebAssembly： the Next Big Platform





# Java 如何调用 Rust 代码



# WebAssembly：官方重视

这个猜测很可能是剧透

# Kotlin WebAssembly

JetBrains / **kotlin**

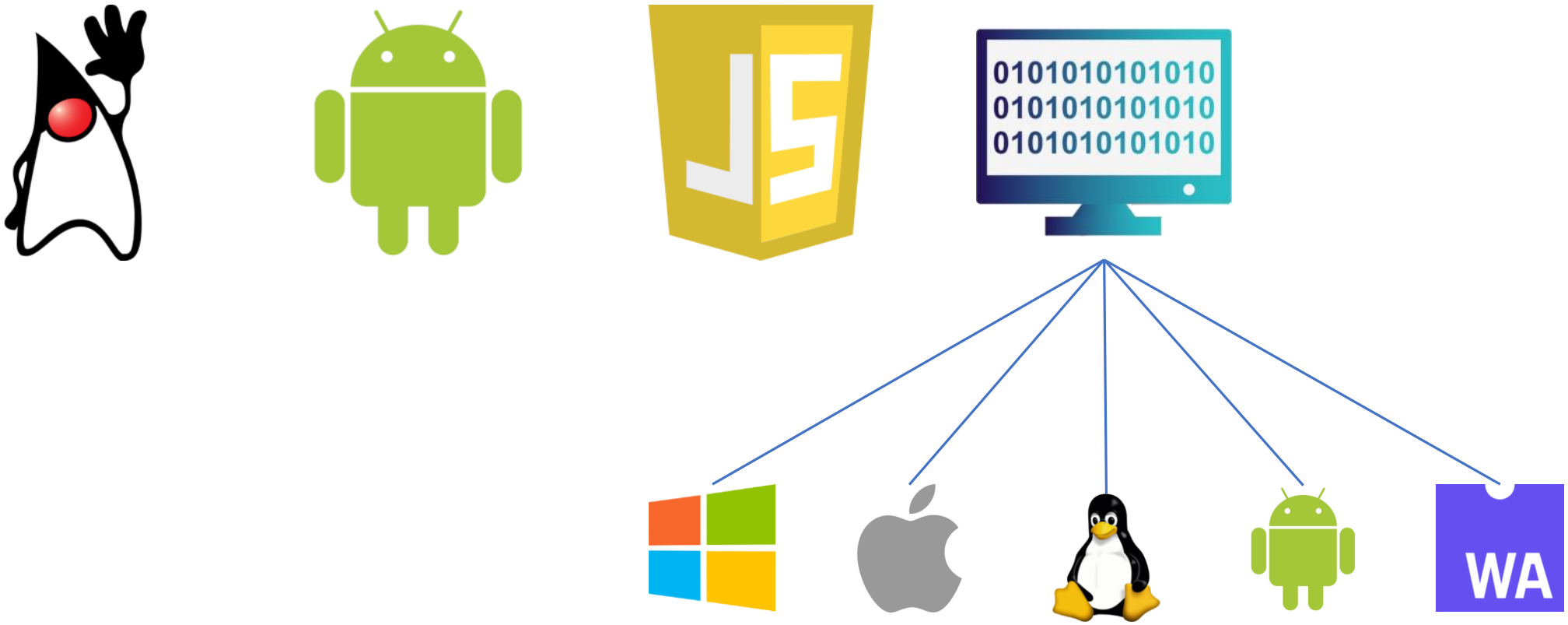
Code | Pull requests 95 | Security | Insights

## [WASM] Initial infrastructure

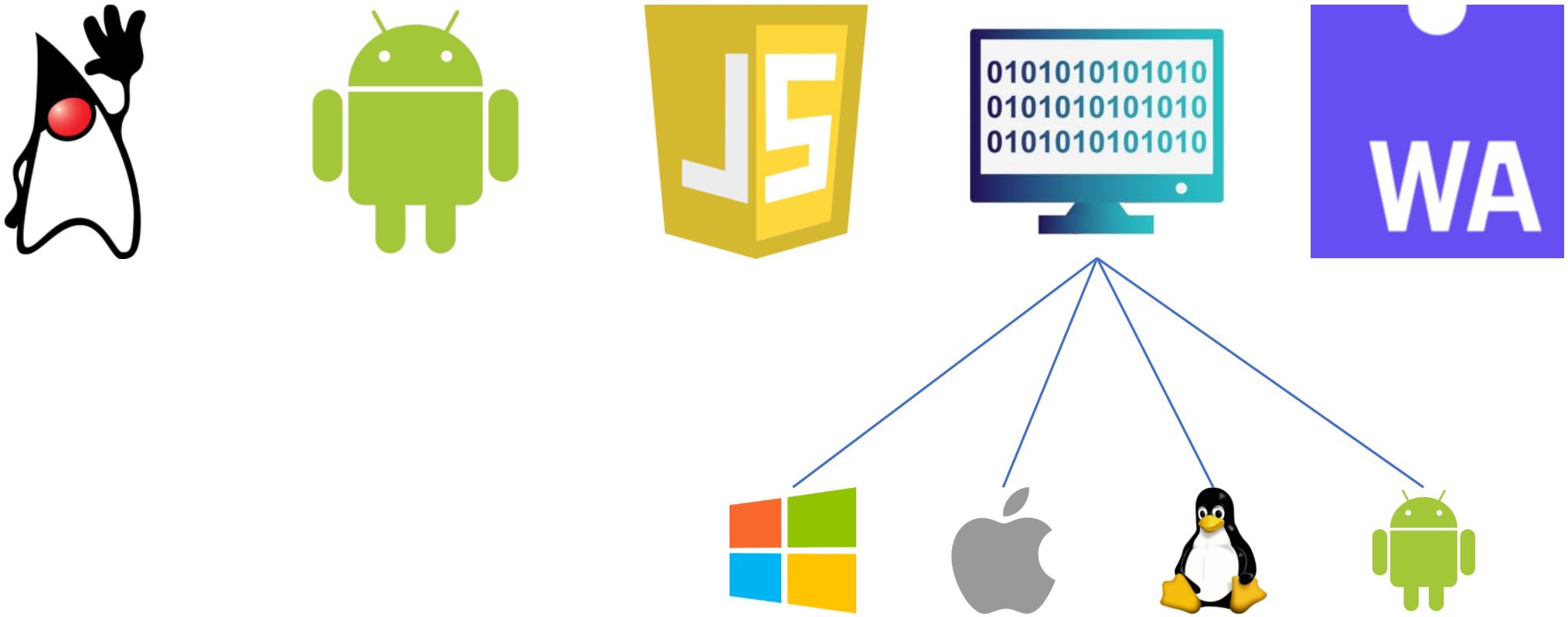
```
- New module ":compiler:backend.wasm"
    - Initial compiler infra (driver, phaser, context)
    - Subset of Wasm AST
    - Skeleton of IR -> Wasm AST
    - Wasm AST -> WAT transformer

- Testing infra

- SpiderMonkey jsshell tool
```

master (#562) build-1.3.60-dev-1562 ... build-1.3.60-dev-1389

# Kotlin WebAssembly



# Kotlin WebAssembly



# Kotlin WebAssembly 示例

```kotlin
@JsName( name: "fibonacci")
tailrec fun fibonacci(n: Long, acc1: Long = 1, acc2: Long = 1): Long =
    if (n <= 1L ) acc2 else fibonacci( n: n - 1L, acc2,  acc2: acc1 + acc2)

fun main() {
   println(fibonacci( n: 10))
   println(fibonacci( n: 100000))
}
```

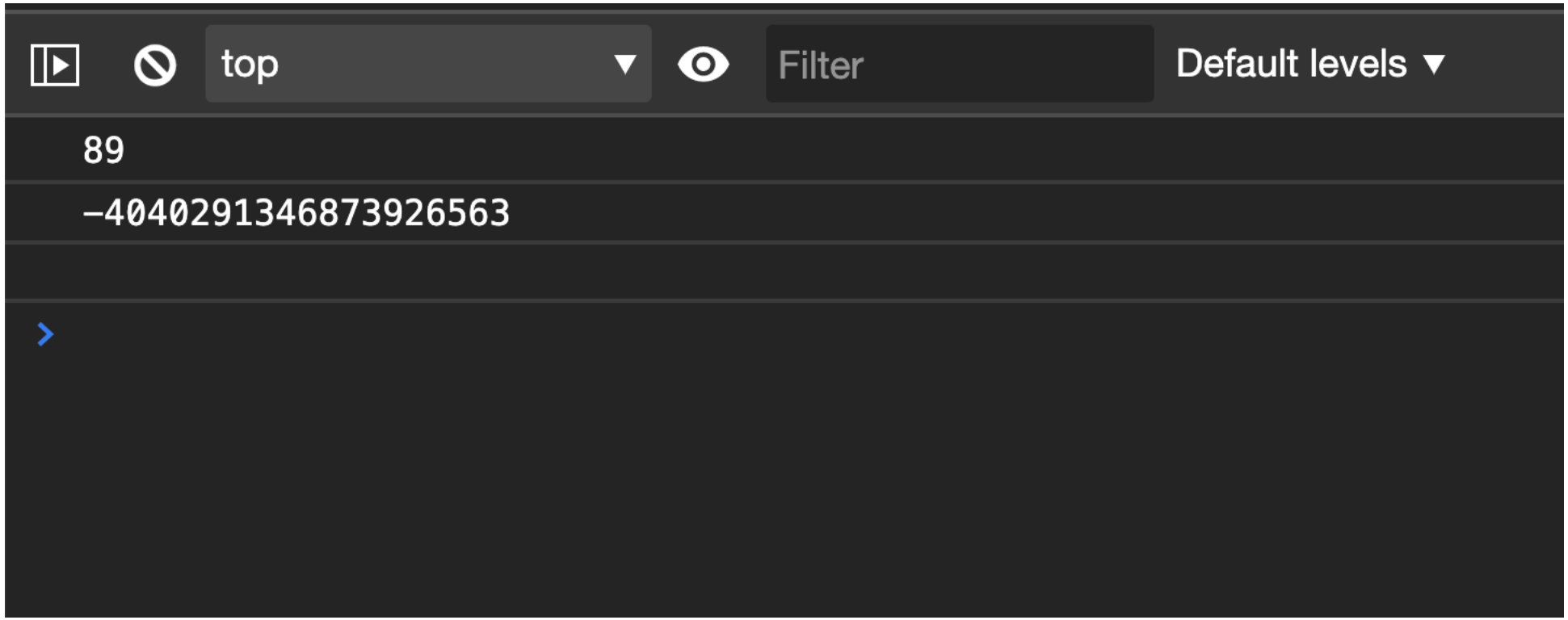
# Kotlin WebAssembly 示例

```
top ▼   Filter   Default levels ▼

89

▶Uncaught RangeError: Maximum call stack size exceeded
    at Kotlin.Long.compare (kotlin.js:410)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)
    at fibonacci (fibjs.js:15)

>
```

# Kotlin WebAssembly 示例

```
@CName("fibonacci")
tailrec fun fibonacci(n: Long, acc1: Long = 1, acc2: Long = 1): Long =
    if (n <= 1L ) acc2 else fibonacci( n: n - 1L, acc2,  acc2: acc1 + acc2)

fun main() {
    println(fibonacci( n: 10))
    println(fibonacci( n: 100000))
}
```

# Kotlin WebAssembly 示例



# Q & A

| 公众号《灰蓝时光》 | 微博：灰蓝天际 |
| --- | --- |
| | |