基于事件驱动的Web App

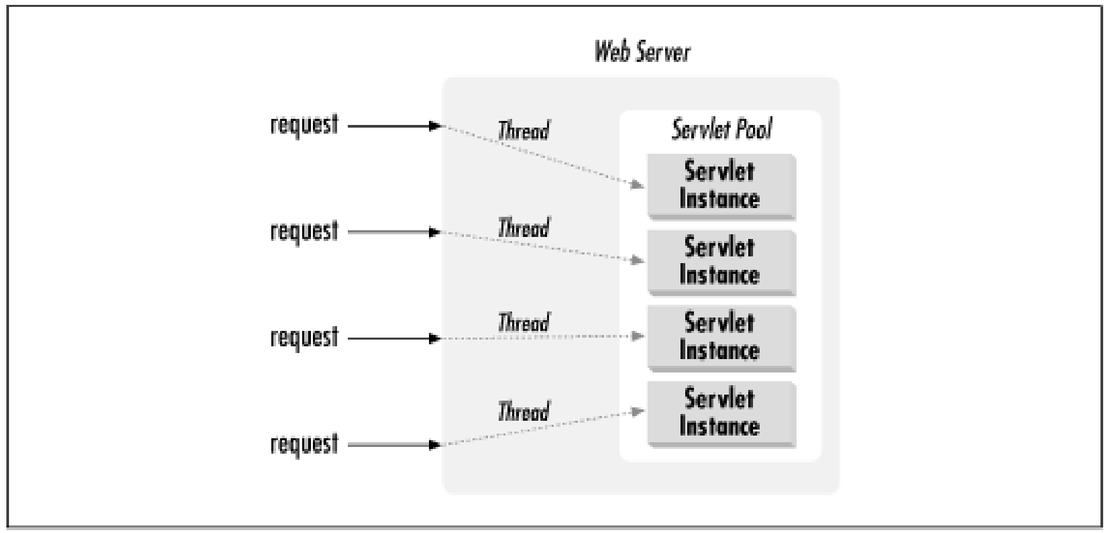
——使用Vert.x构建微服务

党凡

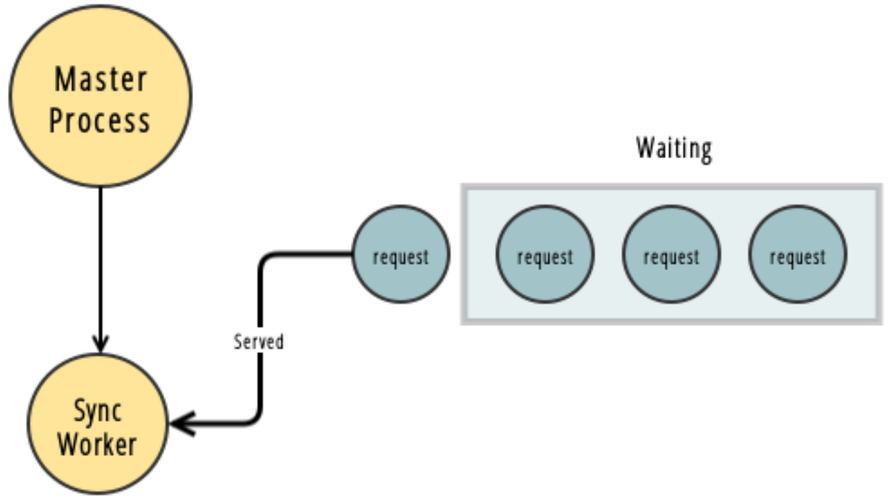
01

基础概念

同步模型



每个请求一个线程



多个请求共享一个线程，排队执行

设置

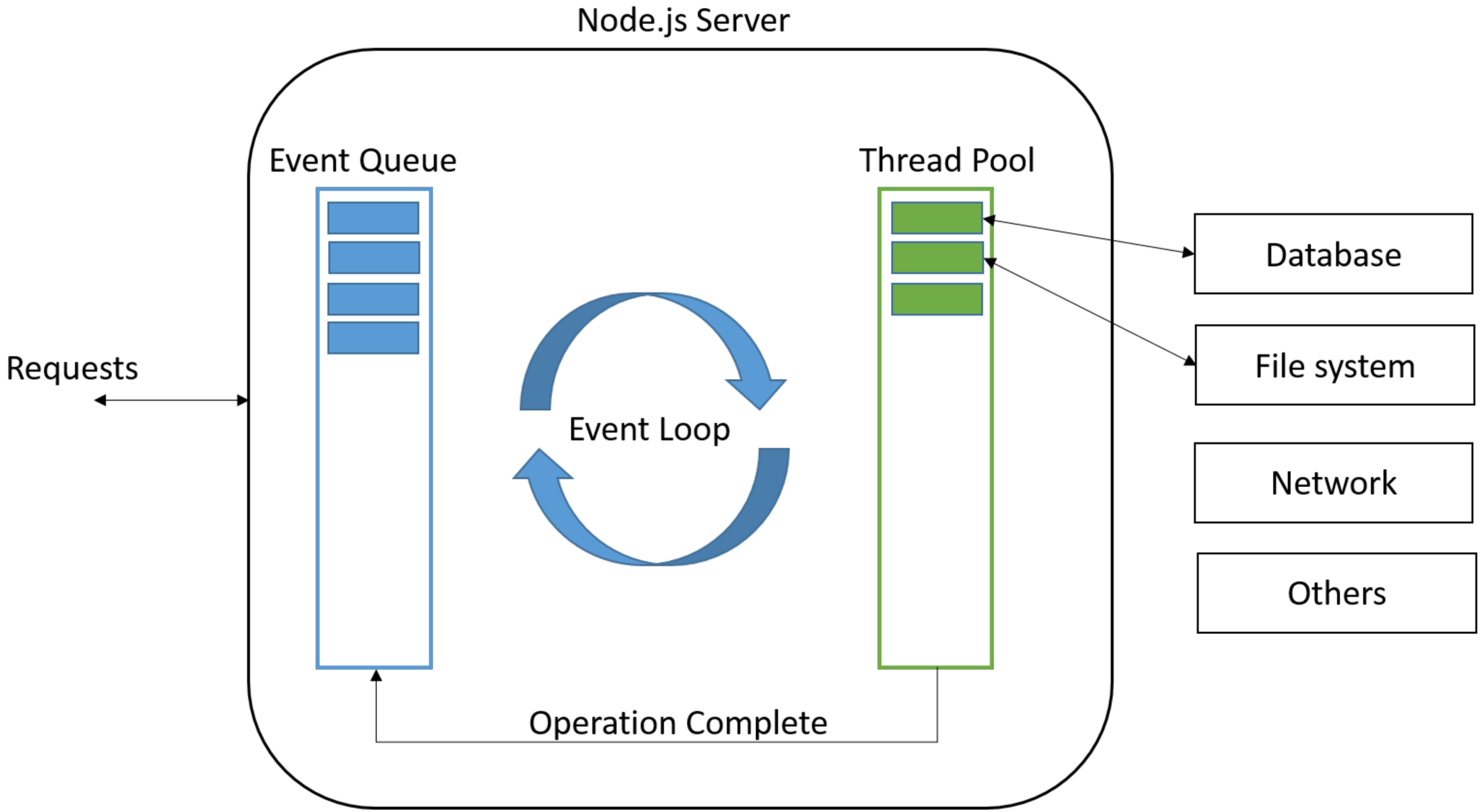
Gunicorn + Django 是哪一种线程模型?

A 每个请求一个线程

B 多个请求共享一个线程，排队执行

提交

异步模型（Node.js）


Node.js Server
Event Queue
Thread Pool
Requests
Event Loop
Database
File system
Network
Others
Operation Complete

Node.js

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

1. 不适合CPU密集型应用
2. 单进程运行（通常场景）
3. 调试困难（异步会吞异常）
4. Callback hell
5. 动态类型

你主要使用哪些语言?

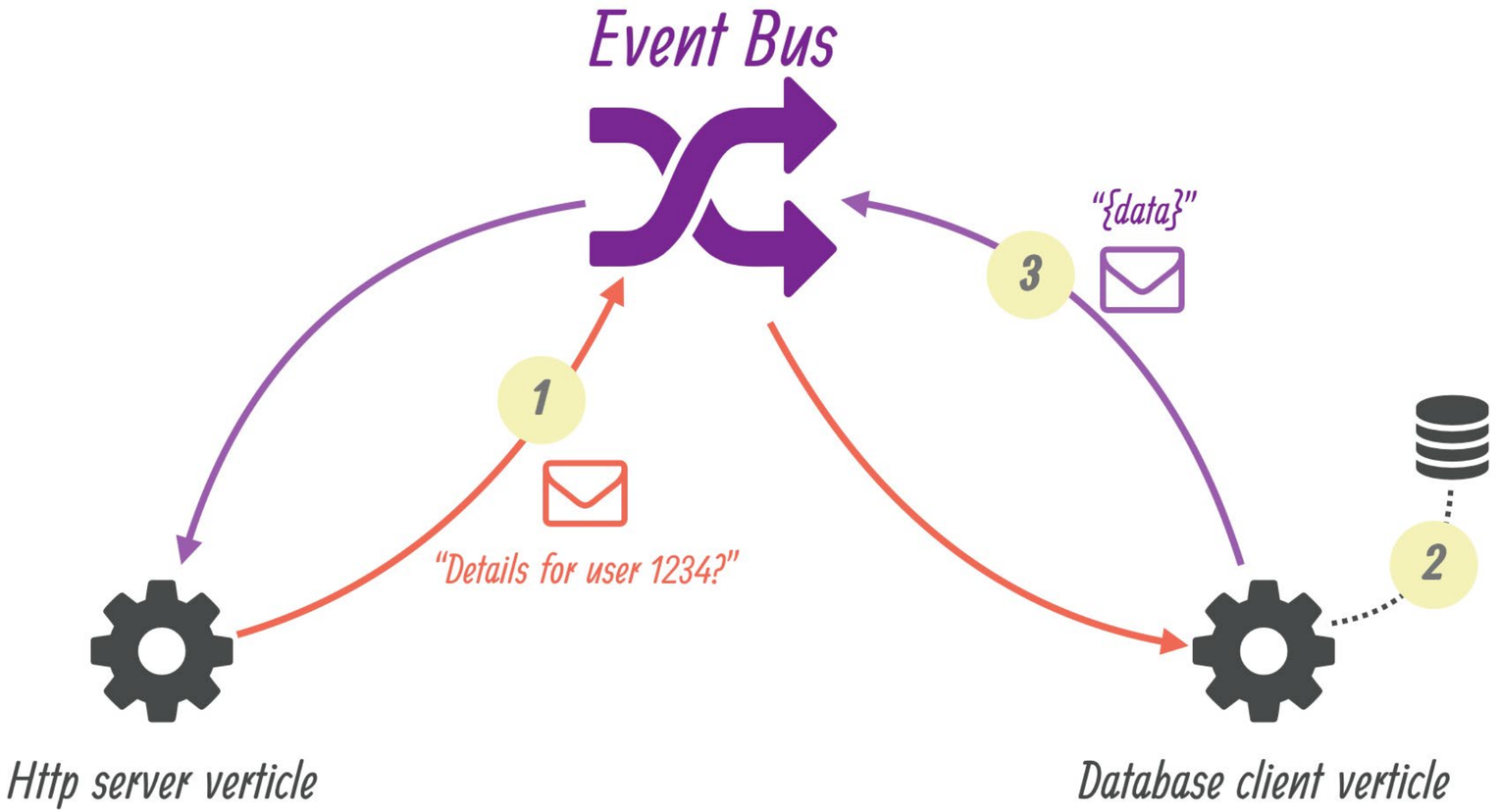
- A Python
- B Java
- C JavaScript
- D Scala
- E Kotlin
- F C#
- G Golang
- H Rust

提交

Eclipse Vert.x 是什么

Vert.x

Event Bus 是不同的 Verticle 异步消息通信的工具



Verticle 是由 Vert.x 部署和运行的代码块。

Verticle

```
class MainVerticle : AbstractVerticle() {
  override fun start(startFuture: Future<Void>) {
    vertx
      .createHttpServer()
      .requestHandler { req : HttpServerRequest! →
        WebClient.create(vertx)
          .getAbs( absoluteURI: "http://ifconfig.co/json")
          .send { res : AsyncResult<HttpResponse<Buffer!>!>! →
            if (res.succeeded()) {
              val body : JsonObject! = res.result().bodyAsJsonObject()
              req.response()
                .putHeader( name: "content-type", value: "text/plain")
                .end( chunk: "My IP: ${body.getString( key: "ip")}")
            } else {
              req.response().setStatusCode(500).end( chunk: "oops")
            }
          }
      }
      .listen( port: 8888) { http : AsyncResult<HttpServer!>! →
        if (http.succeeded()) {
          startFuture.~~complete~~()
          println("HTTP server started on port 8888")
        } else {
          startFuture.~~fail~~(http.cause());
        }
      }
  }
}
```

Verticle

```
class MainVerticle : AbstractVerticle() {
  override fun start(startFuture: Future<Void>) {
    vertx
      .createHttpServer()
      .requestHandler { req : HttpServerRequest! →
        WebClient.create(vertx)
          .getAbs( absoluteURI: "http://ifconfig.co/json")
          .send { res : AsyncResult<HttpResponse<Buffer!>!>! →
            if (res.succeeded()) {
              val body : JsonObject! = res.result().bodyAsJsonObject()
              req.response()
                .putHeader( name: "content-type", value: "text/plain")
                .end( chunk: "My IP: ${body.getString( key: "ip")}")
            } else {
              req.response().setStatusCode(500).end( chunk: "oops")
            }
          }
      }
      .listen( port: 8888) { http : AsyncResult<HttpServer!>! →
        if (http.succeeded()) {
          startFuture.complete()
          println("HTTP server started on port 8888")
        } else {
          startFuture.fail(http.cause());
        }
      }
  }
}
```

```
override suspend fun start() {
  vertx
    .createHttpServer()
    .requestCoroutineHandler { req : HttpServerRequest →
      val client : WebClient! = WebClient.create(vertx)
      val res : HttpResponse<Buffer!> = client.getAbs( absoluteURI: "http://ifconfig.co/json").sendAwait()
      val body : JsonObject! = res.bodyAsJsonObject()
      req.response()
        .putHeader( name: "content-type", value: "text/plain")
        .end( chunk: "My IP: ${body.getString( key: "ip")}")
    }
    .listenAwait( port: 8888)
}
```

Verticle

1. Vert.x 基于事件驱动，所以——
   总是等回调！

2. 不要写阻塞式代码
   Thread.sleep

3. 使用Vert.x的API
   读写socket
   收发消息
   ……

Event Bus

```
override suspend fun start() {
  vertx.eventBus().consumer<String>( address: "test") { it: Message<String!>!
    it.reply("hi!")
  }

  vertx.setPeriodic( delay: 1000) { it: Long!
    vertx.eventBus().request<String>( address: "test",  message: "") { it: AsyncResult<Message<String!>!>!
      println("${System.currentTimeMillis()} ${it.result().body()}")
    }
  }
}
```

Event Bus

```
override suspend fun start() {
  vertx.eventBus().consumer<String>( address: "test") { it: Message<String!>!
    it.reply("hi!")
  }

  vertx.setPeriodic( delay: 1000) { it: Long!
    vertx.eventBus().request<String>( address: "test",  message: "") { it: AsyncResult<Message<String!>!>!
      println("${System.currentTimeMillis()} ${it.result().body()}")
    }
  }
}
```

1. 点对点
2. 请求-回复
3. 订阅-广播

02

Web基础

Router

```
override suspend fun start() {
  val router : Router! = Router.router(vertx)
  router.route( path: "/test").coroutineHandler { ctx : RoutingContext →
    ctx.request().headers().forEach { println(it) }
    ctx.request().params().forEach { println(it) }
    ctx.response()
      .putHeader( name: "Content-Type", value: "application/json")
      .endAwait(Json.encode(jsonObjectOf( ...fields: "status" to "ok")))
  }
  router.getWithRegex( regex: ".*bar").coroutineHandler { ctx : RoutingContext →
    ctx.response().end( chunk: "ok")
  }

  vertx
    .createHttpServer()
    .requestHandler(router)
    .listenAwait( port: 8888)
}
```

Router

1. 路径匹配
    a. 精确匹配: "/path"
    b. 前缀匹配: "/path/*"
    c. 路径参数: "/path/:param1"
    d. 正则表达式: "/.*bar"

2. HTTP方法

3. MIME type

以下哪些是合法的 MIME type?

A text/json

B application/octet-stream

C application/json

D application/protobuf

提交

Sub-Router

```
private fun getSubRouter() : Router!  = Router.router(vertx).apply { this: Router!
  get("/test").coroutineHandler { ctx : RoutingContext  →
    ctx.response().end( chunk: "hi!")
  }
}

override suspend fun start() {
  val router : Router!  = Router.router(vertx)
  router.mountSubRouter( mountPoint: "/api", getSubRouter())

  vertx
    .createHttpServer()
    .requestHandler(router)
    .listenAwait( port: 8888)
}
```

数据库

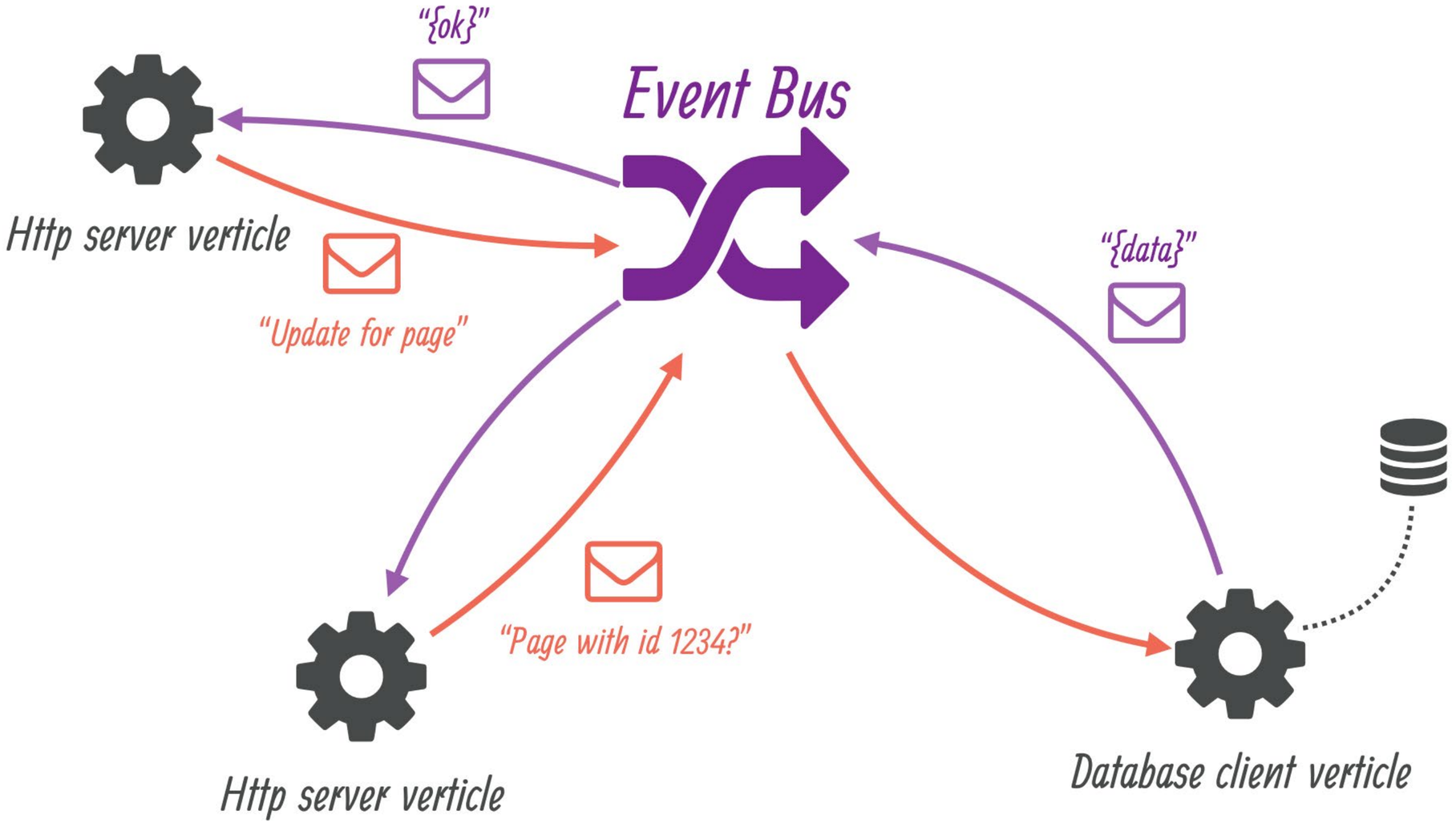
```
override suspend fun start() {
  val redis : RedisAPI! = RedisAPI.api(Redis.createClient(vertx).connectAwait())
  val mysql : MySQLPool! = MySQLPool.pool(vertx, connectionUri: "mysql://root:root@localhost/test")

  val router : Router! = Router.router(vertx)
  router.get("/redis").coroutineHandler { ctx : RoutingContext →
    val resp : Response? = redis.getAwait( arg0: "test-key")
    if (resp == null) {
      ctx.response().setStatusCode(404).end()
    } else {
      ctx.response().end(resp.toString())
    }
  }
  router.get("/mysql").coroutineHandler { ctx : RoutingContext →
    val conn : SqlConnection = mysql.getConnectionAwait()
    val resp : RowSet<Row!>! = conn.query( sql: "select 114514").executeAwait()
    ctx.response().end(resp.first().getInteger( pos: 0).toString())
  }

  vertx
    .createHttpServer()
    .requestHandler(router)
    .listenAwait( port: 8888)
}
```

03

构建微服务


"{ok}"
Event Bus
Http server verticle
"Update for page"
"{data}"
"Page with id 1234?"
Http server verticle
Database client verticle

数据服务

```
@VertxGen
@ProxyGen
interface DataService {

  @Fluent
  fun findData(id: Long, requestHandler: Handler<AsyncResult<String?>>): DataService

}

class DataServiceImpl : DataService {

  override fun findData(id: Long, requestHandler: Handler<AsyncResult<String?>>): DataService {
    if (id == 1L) {
      requestHandler.handle(Future.succeededFuture( result: "Harry"))
    } else {
      requestHandler.handle(Future.succeededFuture( result: null))
    }
    return this
  }

}
```

```
override suspend fun start() {
  ServiceBinder(vertx)
    .setAddress("data-service")
    .register(DataService::class.java, DataServiceImpl())
  val service = DataServiceVertxEBProxy(vertx,
    address: "data-service",
    deliveryOptionsOf(sendTimeout = 5000L))

  val router : Router! = Router.router(vertx)
  router.get("/test").coroutineHandler { ctx : RoutingContext →
    val id : Long = ctx.request().params().get("id").toLong()
    val resp: String? = awaitResult { service.findData(id, it) }
    if (resp == null) {
      ctx.response().setStatusCode(404).end( chunk: "not found")
    } else {
      ctx.response().end(resp)
    }
  }

  vertx
    .createHttpServer()
    .requestHandler(router)
    .listenAwait( port: 8888)
}
```

服务发现

```
class ServiceVerticle : CoroutineVerticle() {
  override suspend fun start() {
    val addr = "data-service"
    ServiceBinder(vertx)
      .setAddress(addr)
      .register(DataService::class.java, DataServiceImpl())
    val discovery : ServiceDiscovery! = ServiceDiscovery.create(vertx)
    val record : Record! = EventBusService.createRecord( name: "data",
      addr,
      DataService::class.java)
    discovery.publishAwait(record)
  }
}
```

```
override suspend fun start() {
  val discovery : ServiceDiscovery! = ServiceDiscovery.create(vertx)

  val router : Router! = Router.router(vertx)
  router.get("/test").coroutineHandler { ctx : RoutingContext →
    val record : Record = discovery.getRecordAwait { it.name == "data" }!!
    val reference : ServiceReference! = discovery.getReference(record)
    val service : DataService! = reference.getAs(DataService::class.java)
    val id : Long = ctx.request().params().get("id").toLong()
    val resp: String? = awaitResult { service.findData(id, it) }
    if (resp == null) {
      ctx.response().setStatusCode(404).end( chunk: "not found")
    } else {
      ctx.response().end(resp)
    }
    reference.release()
  }

  vertx
    .createHttpServer()
    .requestHandler(router)
    .listenAwait( port: 8888)
}
```

04

其他议题

投票

最多可选4项

设置

如何提高代码质量?

A 单元测试/集成测试

B Code Review

C Tester Driven Development

D CI

提交

测试

1. 集成测试为主

2. 必要的单元测试

测试

```groovy
@Stepwise
class LiveRouterSpec extends Specification {
  static Vertx vertx
  static RedisAPI redis
  static MySQLPool mysql
  static [redacted]LiveService service

  void setupSpec() { ... }

  void cleanupSpec() { ... }

  def "should get [redacted] params"() {
    given: "An instance of AsyncConditions"
    def async = new AsyncConditions(1)

    when: "Invoke service"
    service.get[redacted]Param() { res ->
      async.evaluate {
        assert res.result().getString("app_token") == "xxxxxxx"
      }
    }

    then: "Expect the result to be completed in the specified time"
    async.await(2)
  }
```

初始化数据

清理数据

定义一个测试

等待 1 个异步完成

断言

最多运行2秒

学习资源

1. 核心部分：https://vertx.io/docs/vertx-core/kotlin/
2. Web部分：https://vertx.io/docs/vertx-core/kotlin/
3. Wiki示例（从开发到重构）：https://vertx.io/docs/guide-for-java-devs/
4. 微服务示例：https://vertx.io/blog/vert-x-blueprint-tutorials/

Golang

1. 没有异常机制

```
if (err != nil) {}
db.Exec("DELTE FROM item WHERE id = 2")
```

2. 包管理/GOPATH

3. Duck typing

4. 不支持泛型

5. 没有可空类型（nullable）

Q&A