Appunti vertex

La libreria Vertex di Java è un framework per la creazione di applicazioni reattive concorrenti ed altamente scalabili su JVM: programmazione reattiva, che si basa su eventi e flussi di dati asincroni perciò consente la gestione efficiente di operazioni non bloccanti, così evitando la perdita di eventi (e di dati).

L'unita di base di vertex viene chiamata Verticle che rappresenta un thread separato che può essere distribuito e scalato su altri vertici. Per esmpio su ogni mainVerticle si può richiamarne un'altro (anche per modulare progetti più complessi).

public class MainVerticle extends AbstractVerticle {

//i metodi start e stop (stop può essere omesso) definiscono il comportamento del server all'avvio e al termine del Verticle (praticamete il comportamento del thread).Ricordarsi di estendere sempre AbstractVerticle

@Override

public void start() {

// Codice che viene eseguito quando il main veticle viene avviato

System.out.println("MainVerticle started");

// Deploy del verticle aggiuntivo significa avviare un'altro verticle all'interno del main (esso può assistere il main verticle nelle sue funzioni oppure inizializzare anche un'altro server in ascolto su un'altra porta[8080,8888....])

vertx.deployVerticle(new AnotherVerticle());

}

@Override

public void stop() {

// Codice da eseguire quando il Verticle viene arrestato

System.out.println("MainVerticle stopped");

}

public static void main(String[] args) {

// Creare l'istanza di Vertx

Vertx vertx = Vertx.vertx();

// Main verticle viene avviato

vertx.deployVerticle(new MainVerticle());

}

}

class AnotherVerticle extends AbstractVerticle {

@Override

public void start() {

// Codice da eseguire quando il vertice interno viene avviato

System.out.println("AnotherVerticle started");

}

}

Se il verticle viene implementato in un altro file dovrà essere specificato il percorso file all'interno della chiamata:

Mentre sopra bastava "vertx.deployVerticle(new AnotherVerticle());" adesso servirà

vertx.deployVerticle(server.AnotherVerticle); contando come nome del package "server" e il nome del file "AnotherVerticle.java" il nome del file.

Un altro componenete fondamentale è il Router che si occupa di gestire il routing delle richieste in unìapplicazione vertex basata su http.

Esso permette di definire diverse destinazioni (o operazioni) per diversi tipi di richeste (come GET e POST), ogni routing associa un percorso URL a un handler:

un handler può essere un verticle specifico o una funzione che accetta un oggetto di tipo RoutingContext come parametro.

Il RoutingContext fornisce informazioni sulla richiesta http in corso e metodi per interagire con essa.

In pratica il router in vertex fornisce l'infrastruttura per gestire il routing delle richieste http e WebSocket nel server.

public void start() {

// Creare un'istanza di Vertx

Vertx vertx = Vertx.vertx();

// Creare un'istanza del server HTTP

HttpServer server = vertx.createHttpServer();

// Creare un router

Router router = Router.router(vertx);

// Definire una regola di routing per le richieste GET a "/api/utenti/data"

router.get("/api/utenti/data").handler(this::handleGetData);

// Definire una regola di routing per le richieste POST a "/api/utenti/data"

router.post("/api/utenti/data").handler(this::handlePostData);

// Associa il router al server HTTP

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

}

// Gestore per le richieste GET a "/api/utenti/data"

private void handleGetData(RoutingContext routingContext) {

// Logica per gestire la richiesta GET

routingContext.response()

.putHeader("content-type", "application/json")

.end("Elenco dei dati...");

}

// Gestore per le richieste POST a "/api/utenti/data"

private void handlePostData(RoutingContext routingContext) {

// Logica per gestire la richiesta POST

routingContext.response()

.putHeader("content-type", "application/json")

.end("Nuovo dato viene creato...");

}

Qui sopra viene usato routing per definire e differenziare il comportamento del server nella gestione delle richieste GET e POST.

L'utimo componenete fondamentale è l'Event Bus: un canale di comunicazione asincrono che consente ai verticle di comunicare tra loro in modo distribuito. I verticle possono inviare messaggi sull'event bus e sottoscriversi per ricevere messaggi relativi a determinati argomenti.

public class MainVerticle extends AbstractVerticle {

@Override

public void start() {

// Creare un'istanza di Vertx

Vertx vertx = Vertx.vertx();

// Deploy dei vertici

vertx.deployVerticle(new ProducerVerticle());

vertx.deployVerticle(new ConsumerVerticle());

}

public static void main(String[] args) {

// Deploy del vertice principale

Vertx.vertx().deployVerticle(new MainVerticle());

}

}

class ProducerVerticle extends AbstractVerticle {

@Override

public void start() {

// Invia un messaggio sull'EventBus utilizzando il metodo send(), specificando l'indirizzo dell'EventBus come destinazione e il contenuto del messaggio

vertx.eventBus().send("esempio.indirizzo", "Messaggio di esempio");

}

}

class ConsumerVerticle extends AbstractVerticle {

@Override

public void start() {

// Registra un handler per i messaggi inviati all'indirizzo "esempio.indirizzo"

vertx.eventBus().consumer("esempio.indirizzo", message -> {

// Riceve il messaggio e lo stampa sulla console

System.out.println("Messaggio ricevuto: " + message.body());

});

}

}

Questo è un'esempio semplice del funzionamento ed utilizzo dell'eventbus nella comunicazione tra 2 o più verticle.

Un altro aspetto fondamentale da tenere in considerazione sarebbe il Cluster Manager che supporta la creazione e gestione di cluster di verticle distribuiti su più server (per server si intende ogni istaza aperta di Vertx non dei verticle), per cluster si intendono più istanze di vertex su macchine diverse o su una stessa macchina, ma non ci interessa.