High performance reactive applications with Vert.x

Tim Fox

Red Hat

Bio

- Employed By Red Hat to lead the Vert.x project
- Worked in open source exclusively for the past 9 years
- Some projects I've been involved with: Vert.x (creator), RabbitMQ, HornetQ (creator), JBoss AS, Mobicents...

Overview

- Lightweight, reactive, application platform
- Superficially similar to Node.js - but not a clone!
- Inspired also from Erlang/OTP
- Polyglot
- High performance (see latest TechEmpower benchmarks!)
- Simple but not simplistic

Polyglot

Full implementation:

Java JS JRuby Groovy Jython

Almost there:

Scala php

Core Asychronous APIs

- Core is small and static
- TCP/SSL clients and servers
- HTTP/HTTPS clients and servers
- Websockets, SockJS
- File system
- Event bus
- DNS (new)
- UDP (new)
- etc

Why Asynchronous?

- Modern servers need to handle high levels of concurrency – web servers, websockets, IoT etc
- OS threads are still a precious resource
- Need to service many connections with small number of threads
- Blocked OS threads means they can't do other work

Verticle

- Execution unit of Vert.x
- Can be written in any language
- Single threaded – less scope for race conditions
- Verticles communicate by message passing
- Hmmm.. sounds like the Actor Model?

Demo

Event Bus

- The nervous system of Vert.x
- Verticles send messages over the event bus
- Point to point. Publish/Subscribe. Request/Response
- Pass strings, buffers, primitive types or JSON
- JSON messages are preferred for structured data

Clustered Event Bus

- Lightweight peer-to-peer messaging system
- Connects multiple Vert.x JVM instances
- Applications are loosely coupled components distributed across your network
- No monolithic "application server"
- Micro-services

Event bus in the Browser

- Event bus extends to *client side* JavaScript too
- Uses the same API on the client
- Powerful distributed event space spanning both client and server nodes
- Ideal for modern "real-time" web applications
- Use whatever client side toolkit you prefer

Demo

Modules

- Modules encapsulate code and resources
- One or more modules per application
- Must include a mod.json descriptor file
- Modules contain zero or more verticles
- Can be runnable or non-runnable
- Module class-loaders provide isolation

Demo

An ecosystem of modules

- Sharing modules encourages reuse
- Modules can be pushed to any Maven or Bintray repository
- Encourage an ecosystem of modules
- Modules are the lego bricks to create your application

It's all about the modules

MongoDB
Redis
MySQL/PostgreSQL
SMTP
JDBC
Jersey
Promises
Guice
Spring
Vertigo
Metrics

Facebook
Yoke
Kafka
BSON
work-queue
NoDyn
GCM
SocketIO
Sessions
RxJava
etc...

Fat jars

- Build module into self contained "fat" executable jar
- Convenient for devops
- Fairly small overhead ~ 4.7 MB

Demo

High Availability

- Automatic failover of deployed modules
- Nodes can be logically grouped
- Network partition detection (quorum)

Demo

Developing with Vert.x

- Vert.x is IDE and build system agnostic
- Can just use a text editor if you like
- Maven archetype
- Gradle template
- Debug and test in IDE
- Module auto-redeploy during development

Demo

Summary

- Write apps as set of loosely coupled components that live *anywhere* where you want – no app server.
- Polyglot – use the language(s) you want
- Simple concurrency – wave goodbye to most race conditions
- Modules – a library of lego bricks to build apps with
- High availability
- Ease of development

Project Info

- Independent Community Project
- The main project is an Eclipse Foundation project
- All code is on GitHub
- 100% open source (ASL 2.0 + Creative Commons)
- One of the most popular Java projects on GitHub

Get involved!

- Loads more to do
- Very active and growing community
- Find us on GitHub
- Google group: vertx
- IRC channel: #vertx on freenode.net

Q & A