

Industrial strength Java/JEE Career Companion to open more doors

search here ...

Go

Home

Java FAQs

600+ Java Q&As

Career

Tutorials

Member

Why?

Can u Debug?

Java 8 ready?

Top X

Productivity Tools

Judging Experience?

Home › Interview › Java Architecture Interview Q&A › ♥♦ 01: 30+ Writing low latency applications in Java interview Q&As

♥♦ 01: 30+ Writing low latency applications in Java interview Q&As

Posted on November 25, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

32
Like
Share

Tweet

11
G+1

76

Share

Have you seen job advertisements requiring Java candidates to work in **real-time**, **low latency**, and **high throughput** systems? Wondering what questions you will be asked?

You will be quizzed on the low latency application you had recently worked on especially the outcomes in terms of the latencies, response times, and throughput along with the challenges you faced.

Q1. What do you understand by the terms **real-time application (RTA)**, **latency** & **throughput**?

A1. **Real-time application (aka RTA)** is an application

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

- ✚ Ice Breaker Interview
- ✚ Core Java Interview C
- ✚ JEE Interview Q&A (3
- ✚ Pressed for time? Jav
- ✚ SQL, XML, UML, JSC
- ✚ Hadoop & BigData Int
- ✚ Java Architecture Inte
- ♥♦ 01: 30+ Writing
- 001A: ♦ 7+ Java int
- 001B: ♦ Java archil
- 01: ♥♦ 40+ Java W
- 02: ♥♦ 13 Tips to w
- 03: ♦ What should l
- 04: ♦ How to go ab
- 05: ETL architectur
- 1. Asynchronous pi
- 2. Asynchronous pi
- ✚ Scala Interview Q&As
- ✚ Spring, Hibernate, & I
- ✚ Testing & Profiling/Sa
- ✚ Other Interview Q&A 1

where the content is pushed through “as it happens” within a specified time frame. These time frames are defined as SLAs (Service Level Agreements). For example, in a **Straight-Through Processing (STP)** solution, you have **real-time** trades flow between your front/middle office, and traders/stock exchange.

Real-time systems can be further divided into **1) Hard real-time** & **2) soft real-time**. **Hard real-time** is when an action is performed at the wrong time will have possibly no or negative effect. In other words, you must absolutely hit every deadline. It is not acceptable to say 90% of the time we hit the response time of 100ms. Only some systems have this requirement -> medical apps (e.g. pacemaker), defense systems, nuclear systems, avionics, etc.

Soft real-time is when an action is performed either too early or too late will still have a positive effect. If it had performed the task on time, it would have had greater value in terms of better customer experience, meeting the SLAs, etc. Soft real-time system can be a trading application with high through-put & low latency without any hard response time guarantees. No catastrophe happens when response times fail say 5% of time when 100K requests are sent. Most system fall into this category like financial applications (e.g. placing trades, matching trades, etc), event processing, telecom, etc.

















Q. Can JVM be used for **real-time** applications in the sense that it's guaranteed to react within a certain amount of time?

A. The answer is no for the standard JVMs, but the special JVMs that support “**Real-Time Specification for Java (RTSJ)**” extensions can process in hard real-time. Standard JVMs achieve “**soft real-time**” mainly due to automatic garbage collection and GC pauses associated with it. The RTSJ provides a subclass of RTT (i.e. Real-Time-Thread) called NoHeapRealtimeThread (NHRT). Instances of this subclass are protected from GC induced pauses. NHRTs are NOT allowed to use the heap. NHRTs use the scoped memory and immortal memory features to allocate memory on a more predictable basis.

 [Free Java Interview](#)


16 Technical Key Areas

[open all](#) | [close all](#)

-  [Best Practice \(6\)](#)
-  [Coding \(26\)](#)
-  [Concurrency \(6\)](#)
-  [Design Concepts \(7\)](#)
-  [Design Patterns \(11\)](#)
-  [Exception Handling \(3\)](#)
-  [Java Debugging \(21\)](#)
-  [Judging Experience \(1\)](#)
-  [Low Latency \(7\)](#)
-  [Memory Management \(1\)](#)
-  [Performance \(13\)](#)
-  [QoS \(8\)](#)
-  [Scalability \(4\)](#)
-  [SDLC \(6\)](#)
-  [Security \(13\)](#)
-  [Transaction Management \(1\)](#)

80+ step by step Java Tutorials

[open all](#) | [close all](#)

-  [Setting up Tutorial \(6\)](#)
-  [Tutorial - Diagnosis \(2\)](#)
-  [Akka Tutorial \(9\)](#)
-  [Core Java Tutorials \(2\)](#)
-  [Hadoop & Spark Tutorials \(1\)](#)
-  [JEE Tutorials \(19\)](#)
-  [Scala Tutorials \(1\)](#)
-  [Spring & Hibernate Tutorials \(1\)](#)
-  [Tools Tutorials \(19\)](#)
-  [Other Tutorials \(45\)](#)

Q. Do you favor hard or soft real-time Java development guidelines in general?

A. soft real-time is favored unless there is a specific need for hard real-time as soft real-time offers much better **developer productivity & application maintenance**.

Latency is the time required to perform some action or to produce some result. Latency is measured in units of time like seconds, milli seconds, micro seconds, nanoseconds, etc. What defines a “low” latency depends on the context – low latency over the internet might be 200ms whereas low latency in a trading application (e.g. pricing or order matching engines) using FIX or custom protocols over TCP/IP might be 2μs. Trading systems need to target 100 nano-seconds to 100 ms.

Throughput is the number of such actions executed or results produced per unit of time. This is measured in units of time like requests per second. The term “memory bandwidth” is sometimes used to specify the throughput of memory systems.

Hadoop Distributed File System (**HDFS**) & **Map-reduce** are about throughput. For example, processing a 100GB file across a 20 node cluster in 2 minutes or processing a 1TB file across a 100 node cluster in same 2 minutes to get the required **throughput**. HDFS **scales out** very well and uses cheaper commodity hardware. You can start with 20 nodes and then horizontally scale to 100+ nodes as higher throughput is required to analyze terabytes of data.

Q2. What is the difference between the terms **latency** and **response times**?”

A2. Latency is the time elapsed between when a request was sent to the server and when the first byte of the response has started to be received.

Response time is the time elapsed between when a request was sent to the server and when the response has been fully received. In a web application the browser needs to

100+ Java pre-interview coding tests

[open all](#) | [close all](#)

- [Can you write code? \(](#)
- [♦ Complete the given](#)
- [Converting from A to I](#)
- [Designing your classe](#)
- [Java Data Structures](#)
- [Passing the unit tests](#)
- [What is wrong with th](#)
- [Writing Code Home A](#)
- [Written Test Core Jav](#)
- [Written Test JEE \(1\)](#)

How good are your?

[open all](#) | [close all](#)

- [Career Making Know-](#)
- [Job Hunting & Resum](#)

load the assets like the DOM tree, images, CSS, and the JavaScript scripts.

So, the response time will always be \geq latency. In other words,

```
1  
2 response time = latency + processing time (e.g. r  
3
```

Low latency is a sum of many things, and two most important ones are 1. **Network Latency**, which is the time taken on the network to send/receive a message/event & 2. **Processing Latency**, which is the time taken by your application to act on a message/event.

If you are building a “trade order matching” engine in Java, the “**network latency**” is the time taken in say micro seconds to receive an order matching request to the engine from a client app plus the time taken for the client app to receive the first byte of the response message from the engine. The “**processing latency**” is the time elapsed in micro or milli seconds for the engine to match the order and build the response to be sent back to the client app.

Q3. Is a latency of over 20ms seconds considered fast or slow in HFT (High Frequency Trading) application?”

A3. Anything over 20ms will be considered slow. The HFT trades are conducted using algorithms to buy, sell, and match huge volume of trades. These are ultra low latency applications once used to be written in “C”, and now a days increasingly being written in Java.

Q4. What throughput will you be aiming for in HFT (High Frequency Trading) applications?”

A4. 50k to 200k orders or transactions per second. You will have multiple servers to process the requests. The architecture needs to be **scalable** to cater for growing demands. You can learn more at [Scalability interview questions & answers](#)

Q5. What latency will you be targeting for your applications?

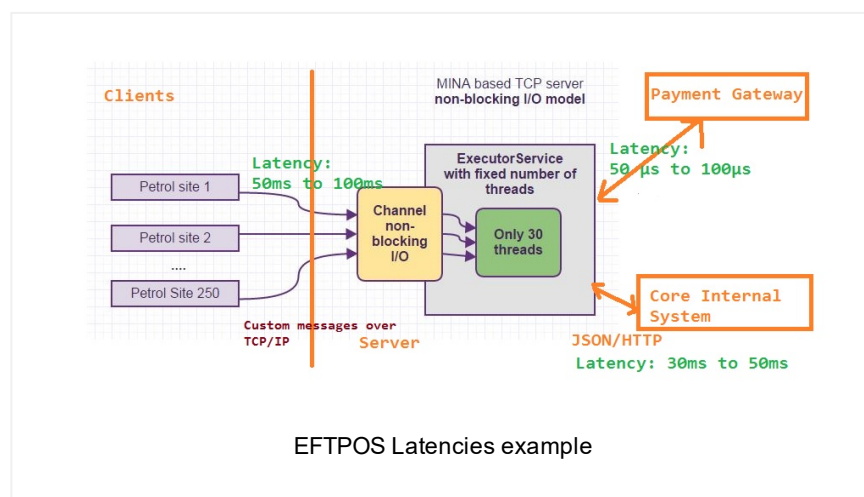
A5 It depends on the context of the application. For example,

#1. Trading system placing buy/sell equity or FX orders to the market will target a latency of under 20ms.

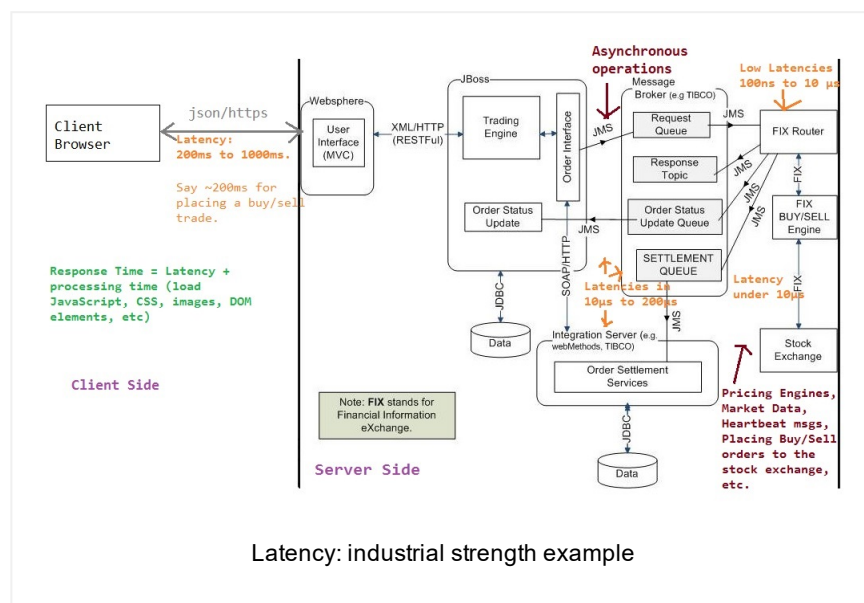
#2. A standard web application will target a latency of 200ms to 800ms.

#3. A gaming application or a more complex web application will target a latency of 500ms to 1000ms.

Example 1: An EFTPOS system



Example 2: An Online Trading System



Q6. How will you go about improving the latency for a more complex web site?

A6.

#1. Processing the requests asynchronously by submitting to a queue and getting the results later on via a client pull or server push.

#2. Reducing the complexity of the page by dividing the tasks with the view of better user experience. This also means smaller payloads transferred between the clients & the servers for better network latency.

#3 Producing less garbage by violating the OO concepts by favoring primitive data types, and applying the flyweight design pattern to improve reuse.

#4. Profiling the application with the tools like **VisualVM** to identify and improve the bottlenecks in terms of CPU, memory usage, Garbage Collection pauses, etc.

#5. [13 tips in detail to write low latency applications in Java.](#)

Q7. What do you understand by the terms real-time systems, latency, and scalability?

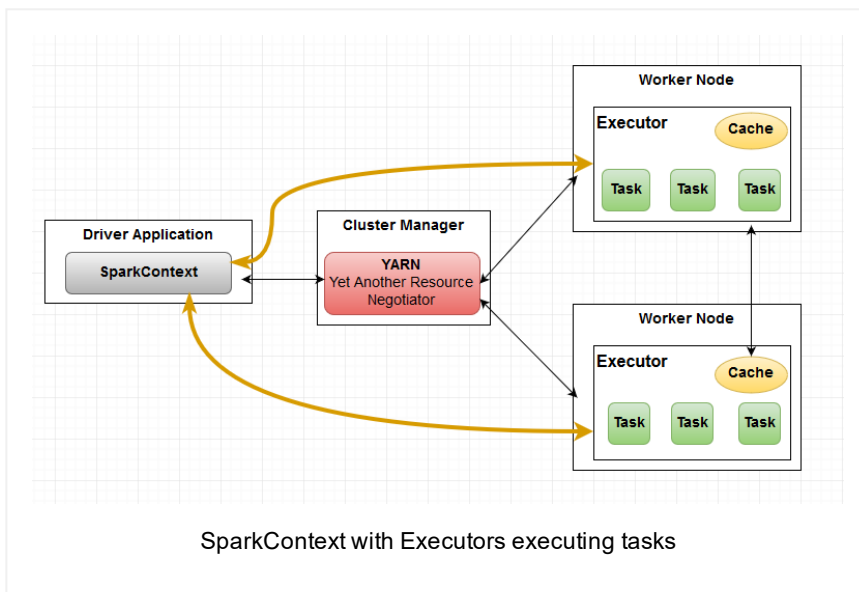
A7. Real-time and **low-latency** are distinctly separate subjects although often related. Real-time is about being more predictable than fast. Low latency systems need to be fast to meet SLAs (Service Level Acceptances) in sub milliseconds (e.g. micro seconds).

Scalability means the ability of the system to handle growing demands by adding more CPUs, memory, nodes, servers, etc. [Scalability interview questions & answers](#)

Q8. What are some of the considerations in writing low latency applications in Java?

A8.

- 1) Writing concurrent programs with Java multi-threading features such as executors, futures, completable futures, fork/join, concurrent data structures, etc.
- 2) Understanding the Java memory model & tuning memory & garbage collection in Java.
- 3) Using event based and non-blocking paradigms. For example, using frameworks like Apache **MINA**, **Netty**, Grizzly, and **Akka**.
- 4) MINA & Netty are lower level frameworks than Akka, and have NIO (New Java IO) as its core. NIO is an event driven non blocking paradigm.
- 5) Akka is a higher level general purpose framework compared to MINA & Netty for building event-driven, scalable, and fault-tolerant applications. Akka is written in Scala, with language bindings provided for both Scala and Java. Akka uses the **Actor model** to hide all the thread-related code and gives you really simple and helpful interfaces to easily implement a scalable and fault-tolerant system.
- 6) Even though you need to have a good handle on writing concurrent programs in Java & interviewers like to quiz/test you on it, favor a framework like Akka as writing complex concurrent programs is not a trivial task, and you need to deal with threads, locks, race conditions & debugging. Writing concurrent programs without frameworks can be error-prone and can lead to code that is difficult to read, test, and maintain.
- 7) If you are working in the BigData space, have a look at Apache Spark, which is based on Scala & **Akka Toolkit**. Here is an example of a Spark master “Driver Application” creating tasks & scheduling them to be run on the “Spark Executors”. Only two executors are shown here, but typical clusters will have 100+ nodes & executors.



Executors are worker nodes' processes in charge of running individual tasks in a given Spark job. Spark Executors are launched at the beginning of a Spark application and typically run for the entire lifetime of an application. Once they have finished running the tasks they send the results to the "Driver Application". "**Spark Executors**" also provide in-memory storage for RDDs that are cached.

Q9. What is an **actor model** in Akka toolset, which is also known as the **reactor design pattern**?

A9. "actor model" is a design pattern for writing concurrent and scalable code that runs on distributed systems. This is an **even driven** (i.e. message passing) model that involves sending & receiving events among actors.

1) Instead of invoking an object directly, you construct a message and send it to a destination object called an **actor**.

2) Each thread is an actor with a specific job to do. The actor engine stores the message in a queue.

3) When a thread becomes available, the actor engine running the actor delivers that message to its destination actor object.

4) When the actor completes its task, it sends a message back to the originating object, which is also considered an actor.

5) You can orchestrate which messages get passed to which actors under what conditions.

The **akka-camel** module allows “Untyped Actors” to receive and send messages over a great variety of protocols such as HTTP, SOAP, TCP, FTP, SMTP or JMS and APIs such as java & Scala .

Wondering what to learn or brush up on?

You need to have a good grasp of multi-threading, Java Memory Model, GC, Profiling, Non-blocking I/O (aka NIO), Big-O notation, lock free data structures like concurrent lists/maps, and strategies to produce less garbage. Here are tutorials & Q&As to build your skills to develop **LOW LATENCY** applications in Java.

1. [Simple Akka tutorial in Java step by step.](#)
2. [13 tips to write low latency applications in Java.](#)
3. [Reactive Programming \(RP\) in Java Interview Q&A.](#)
4. [Java GC tuning for low latency applications](#)
5. [Capture throughput & latencies with “Metrics Core” tutorial](#)
6. [Understanding Big O notations through Java examples](#)
7. [Java primitives & objects – memory consumption interview Q&A](#)
8. [ExecutorService Vs Fork/Join & Future Vs CompletableFuture Interview Q&As](#)
9. [Home assignment – Create a simple framework where work items can be submitted](#)

Better throughput on BigData

1. [Hadoop overview & architecture interview Q&As](#)
2. [Apache Spark interview Q&As](#)
3. [Apache Spark Tutorials](#)

Popular Member Posts

♦ [11 Spring boot interview questions & answers](#)

850 views

♦ [Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers](#)

768 views

001A: ♦ [7+ Java integration styles & patterns interview questions & answers](#)

397 views

[18 Java scenarios based interview Questions and Answers](#)

387 views

♦ [7 Java debugging interview questions & answers](#)

307 views

01b: ♦ [13 Spring basics Q8 – Q13 interview questions & answers](#)

305 views

01: ♦ [15 Ice breaker questions asked 90% of the time in Java job interviews with hints](#)

297 views

♦ [10 ERD \(Entity-Relationship Diagrams\) Interview Questions and Answers](#)

293 views

♦ [Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers](#)

246 views

001B: ♦ [Java architecture & design concepts interview questions & answers](#)

203 views

Bio

Latest Posts



Arulkumaran Kumaraswamipillai



Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. [join my LinkedIn Group](#). [Reviews](#)



About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site. **945+** paid members. [join my LinkedIn Group](#). [Reviews](#)

◀ Top 7+ Java productivity tools that make your life easier as a Java developer

Java 7 fork and join tutorial with a diagram and an example ▶

Posted in Java Architecture Interview Q&A, Java Key Area Essentials, Judging Experience Interview Q&A, Low Latency, Performance, Scalability

Leave a Reply

Logged in as [geethika](#). [Log out?](#)

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ [Java generics in no time](#) ☀ [Top 6 tips to transforming your thinking from OOP to FP](#) ☀ [How does a HashMap internally work? What is a hashing function?](#)
☀ [10+ Java String class interview Q&As](#) ☀ [Java auto un/boxing benefits & caveats](#) ☀ [Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect](#)

Non-Technical Know Hows

☀ [6 Aspects that can motivate you to fast-track your career & go places](#) ☀ [Are you reinventing yourself as a Java developer?](#) ☀ [8 tips to safeguard your Java career against offshoring](#) ☀ [My top 5 career mistakes](#)

Prepare to succeed

☀ [Turn readers of your Java CV go from “Blah blah” to “Wow”?](#) ☀ [How to prepare for Java job interviews?](#) ☀ [16 Technical Key Areas](#) ☀ [How to choose from multiple Java job offers?](#)

Select Category ▼

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.