5 Java Concurrency interview Q&A

Posted on September 9, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Concurrency is very important in any modern system, and this is one topic many software engineers struggle to have a good grasp. The complexity in concurrency programming stems from the fact that the threads often need to operate on the common data. Each thread has its own sequence of execution, but accesses common data.

Q1. How will you go about identifying and debugging Java concurrency issues like thread starvation, dead lock, and contention?

A1. Firstly, concurrency issues mainly surface under load. So, you need to write JMeter scripts to reproduce the load and scenario under which the concurrency issues surface.

Debugging concurrency issues and fixing any thread starvation, dead lock, and contention require skills and experience to identify and reproduce these hard to resolve issues. Here are some techniques to detect concurrency issues.

- **Manually reviewing the code** for any obvious thread-safety issues. There are static analysis tools like Sonar, ThreadCheck, etc for catching concurrency bugs at compile-time by analyzing their byte code.
- **List all possible causes and add extensive log statements** and write test cases to prove or disprove your theories.
- **Thread dumps are very useful for diagnosing synchronization problems** such as deadlocks. The trick is to take 5 or 6 sets of thread dumps at an interval of 5 seconds between each to have a log file that has 25 to 30 seconds worth of runtime action. For thread dumps, use kill -3 in Unix and CTRL+BREAK in Windows. There are tools like Thread Dump Analyzer (TDA), Samurai, etc. to derive useful information from the thread dumps to find where the problem is. For example, Samurai colors idle threads in grey, blocked threads in red, and running threads in green. You must pay more attention to those red threads.
- There are tools like **JDB** (i.e. Java DeBugger) where a “watch” can be set up on the suspected variable. When ever the application is modifying that variable, a thread dump will be printed.
- There are **dynamic analysis tools** like jstack and **JConsole**, which is a JMX compliant GUI tool to get a thread dump on the fly. The JConsole GUI tool does have handy features like “detect deadlock” button to perform deadlock detection operations and ability to inspect the threads and objects in error states. Similar tools are available for other languages as well.

Q2. What are some of the best practices to keep in mind relating to writing concurrent programs?

A2.

- **Favor immutable objects** as they are inherently thread-safe.
- If you need to use mutable objects, and share them among threads, then a key element of thread-safety is locking access to shared data while it is being operated on by a thread. For example, in Java you can use the synchronized keyword.
- Generally try to keep your locking for as shorter duration as possible to minimize any thread contention issues if you have many threads running. Putting a big, fat lock right at the start of the function and unlocking it at the end of the function is useful on functions that are rarely called, but can adversely impact performance on frequently called functions. Putting one or many larger locks in the function around the data that actually need protection is a finer grained approach that works better than the coarse grained approach, especially when there are only a few places in the function that actually need protection and there are larger areas that are thread-safe and can be carried out concurrently.
- **Use proven concurrency libraries (e.g. java.util.concurrency)** as opposed to writing your own. Well written concurrency libraries provide concurrent access to reads, while restricting concurrent writes.
- **Favor "optimistic concurrency control"** over pessimistic concurrency control.

Q3. What are common production issues, and how will you go about resolving them?

A3. There could be general run time production issues that either slow down or make a system to hang. In these situations, the general approach for troubleshooting would be to analyze the thread dumps to isolate the threads which are causing the slow-down or hang. For example, a Java thread dump gives you a snapshot of all threads running inside a Java Virtual Machine. There are graphical tools like Samurai to help you analyze the thread dumps more effectively.

Git & SVN (6)
JMeter (2)
JSF (2)
Maven (3)
Testing & Profiling/Sa
Other Interview Q&A t
Free Java Interviev

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 I
Low Latency (7)
Memory Managemen
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managen

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 Tuto

- **Application seems to consume 100% CPU and throughput has drastically reduced** – Get a series of thread dumps, say 7 to 10 at a particular interval, say 5 -8 seconds and analyze these thread dumps by inspecting closely the “runnable” threads to ensure that if a particular thread is progressing well. If a particular thread is executing the same method through all the thread dumps, then that particular method could be the root cause. You can now continue your investigation by inspecting the code.
- **Application consumes very less CPU and response times are very poor due to heavy I/O operations** like file or database read/write operations – Get a series of thread dumps and inspect for threads that are in “blocked” status. This analysis can also be used for situations where the application server hangs due to running out of all runnable threads due to a deadlock or a thread is holding a lock on an object and never returns it while other threads are waiting for the same lock.

The solution to the above problems could vastly vary from fixing the thread safety issue(s) to reducing the size of synchronization granularity, and from implementing appropriate caching strategies to setting the appropriate connection timeouts, etc discussed under performance and scalability key areas.

Q4. Can you explain the terms “optimistic concurrency control” and “pessimistic concurrency control”?
A4. Most web applications allow a user to query a database, retrieve a local copy of the queried data, make changes to that local copy, and then finally send the updates and the unchanged values back to the database. When two or more users are concurrently updating the same row in the database, it is possible to cause “lost update” issues. There are 2 ways to handle concurrent updates causing “lost update issues”.

1. Pessimistic concurrency control involves pessimistically locking the database row(s) with the appropriate database

isolation levels or "**select … for update nowait**" SQL. Even when there is no conncurrent access, the row(s) will be locked. This can adversely impact performance and scalability.

2. Optimistic concurrency control assumes that concurrent updates occur very rarely, and deals with concurrent updates when they occur by detecting them, and prompting the users to retry via eror messages like "Another user has changed the data since your last request. Please try again.". Version numbers or timestamps are used to detect concurrent updates. In optimistic concurrency control, the SQL query will be something like "**update … where id = ? and timestamp = ?**" or "**update … where id = ? and vesrion = ?**" . Hibernate provides support for optimistic concurrency control with "**version**" or "**timestamp**" columns.

Q5. What are the 3 common causes of concurrency issues in a multi-threaded application?
A5.

1) **Atomicity** means an operation will either be completed or not done at all. Other threads will not be able to see the operation "in progress" — it will never be viewed in a partially complete state.
2) **Visibility** determines when the effects of one thread can be seen by another.
3) **Ordering** determines when actions in one thread can be seen to occur out of order with respect to another.

Concurrency issues in Java can be fixed with a number of different ways

1) Carefully using **synchronized** keyword at block level, method level, or class level.
2) Using the combination of "**volatile**" for variables and **block level synchronized** keywords.
3) Using the Atomic classes like ***AtomicInteger***, as atomic classes are inherently thread-safe.
4) Favoring immutable objects as they are inherently thread-safe. Once constructed, immutable objects cannot be

modified.

5) Using the explicit locks, concurrent collection classes like **ConcurrentHashmap**, Semaphores, CountDownLatch, CyclicBarrier, etc provided by the **java.util.concurrent** package.

Popular Posts

♦ 11 Spring boot interview questions & answers

828 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

768 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

389 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

296 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

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

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

202 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**

‹ ♦ Find 2 numbers from an array that add up to the target EIP ›

Posted in Concurrency, Java Key Area Essentials, member-paid

Tags: Architect FAQs

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

↑