Java-Success.com

Industrial strength Java/JEE Career Companion to open more doors

Register | Login | Logout | Contact Us

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 › Pressed for time? Java/JEE Interview FAQs › Java Key Area Essentials › ♥ Top 10 causes of performance issues in Java

♥ Top 10 causes of performance issues in Java

Posted on October 28, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

0 Like Share Tweet 4 G+1 Share

Cause #1: The JVM spends **more time performing garbage collection** due to

- improper Garbage Collection (GC) configuration. E.g. Young generation being too small.
- Heap size is too small (use -Xmx). The application footprint is larger than the allocated heap size.
- Wrong use of libraries. For example, XML based report generation using DOM parser as opposed to StAX for large reports generated concurrently by multiple users.
- Incorrectly creating and discarding objects without astutely reusing them with a flyweight design pattern

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

open all | close all

or proper caching strategy.

- Other OS activities like swap space or networking activity during GC can make GC pauses last longer.
- Any explicit *System.gc( )* from your application or third party modules.

Run your JVM with GC options such as

- -verbose:gc (print the GC logs)
- -Xloggc: (comprehensive GC logging)
- -XX:+PrintGCDetails (for more detailed output)
- -XX:+PrintTenuringDistribution (tenuring thresholds)

to understand the GC patterns.

Cause #2: Bad use of application algorithms, strategies, and queries. For example

- SQL queries with Cartesian joins.
- SQL queries invoking materialized views
- Regular expressions with back tracking algorithms.
- Inefficient Java coding and algorithms in frequently executed methods leading to death by thousand cuts.
- Excessive data caching or inappropriate cache refresh strategy.
- Overuse of pessimistic locking as opposed to favoring optimistic locking.

Cause #3: Memory leaks due to

- Long living objects having reference to short living objects, causing the memory to slowly grow. For example, singleton classes referring to short lived objects. This prevents short-lived objects being garbage collected.
- Improper use of thread-local variables. The thread-local variables will not be removed by the garbage collector as long as the thread itself is alive. So, when threads are pooled and kept alive forever, the object might never be removed by the garbage collector.
- Using mutable static fields to hold data caches, and not explicitly clearing them. The mutable static fields and collections need to be explicitly cleared.

- Objects with circular or bidirectional references.
- JNI memory leaks.

Cause #4: Poor integration with external systems without proper design & testing.

- Not properly deciding between synchronous vs asynchronous calls to internal and external systems. Long running tasks need to be performed asynchronously.
- Not properly setting service timeouts and retries or setting service time out values to be too high.
- Not performing non happy path testing and not tuning external systems to perform efficiently.
- Unnecessarily making too many network round trips.

Cause #5: Improper use of Java frameworks and libraries.

- Using Hibernate without properly understanding lazy loading versus eager fetching and other tuning capabilities.
- Not inspecting the SQLs internally generated by your ORM tools like Hibernate.
- Using the deprecated libraries like *Vector* or *Hashtable* as opposed to the new concurrent libraries that allow concurrent reads.
- Using blocking I/O where the the Java NIO (New I/O) with non blocking capability is favored.
- Database deadlocks due bad schema design or application logic.
- Spinning out your own in efficient libraries as opposed to favoring proven frameworks.

Cause #6: Multi-threading issues due to to deadlocks, thread starvation, and thread contention.

- Using coarse grained locks over fine grained locks.
- Not favoring concurrent utility classes like *ConcurrentHashMap*, *CopyOnWriteArrayList*, etc.
- Not favoring lock free algorithms.

Cause #7: Not managing and recycling your non memory resources properly.

16 Technical Key Areas

80+ step by step Java Tutorials

- Not pooling your valuable non memory resources like sockets, connections, file handles, threads, etc.
- Not properly releasing your resources back to its pool after use can lead to resource leaks and performance issues.
- Use of too many threads leading to more CPU time being used for context switching.
- Hand rolling your own pooling without favoring proven libraries.
- Using a third-party library with resource leaks.
- Load balancers leaking sockets.

Cause #8: Bad infrastructure designs and bugs.

- Databases tables not properly partitioned.
- Not enough physical memory on the box.
- Not enough hard disk space.
- Bad network latency.
- Too many nodes on the server.
- Load balancers not working as intended and not performing outage testing.
- Not tuning application servers properly.
- Not performing proper capacity planning.
- router, switch, and DNS server failures.

Cause #9: Excessive logging and not using proper logging libraries with capabilities to control log levels like debug, info, warning, etc. *System.out.println(….)* are NOT ALLOWED. Favor asynchronous logging in mission critical and high volume applications like trading systems.

Cause #10: Not conducting performance tests, not monitoring the systems, and lack of documentation and performance focus from the start of the project.

- Not performing performance test plans with tools like JMeter with production like data prior to each deployment.
- Not monitoring the systems for CPU usage, memory usage, thread usage, garbage collection patterns, I/O, etc on an on going basis.
- Not defining SLA’s (Service Level Agreements) from the start in the non-functional specification.

100+ Java pre-interview coding tests

How good are your …..?

- Not maintaining proper performance benchmarks to be compared against with the successive releases and performance tests.
- Not looking for performance issues in peer code reviews or not having code reviews at all.

Popular Posts

♦ 11 Spring boot interview questions & answers

827 views

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

767 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**

‹ ♥ JasperReport generating a PDF file – Part 4

♥ Top 11 slacknesses or mistakes that can come back and bite you as an experienced Java developer or architect ›

Posted in Java Key Area Essentials, Performance

Tags: Free Content, Free FAQs, TopX

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