Concurrency With Java 8

SEMINAR

contents

- course contents

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

Concurrency With Java 8

SEMINAR

course contents

- Producer Consumer(Basic Hand-Off) Day1
- Common Issues with thread
- Java Memory Model(JMM)
- Applied Threading techniques
- Building Blocks for Highly Concurrent Design
- Highly Concurrent Data Structures-Part1
- Java8 Day2
- Java8 Concurrency

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

Concurrency With Java 8

- ***Producer Consumer(Basic Hand-Off)***
  - Why wait-notify require Synchronization
    - notifyAll used as work around
    - Structural modification to hidden queue by wait-notify
    - locking handling done by OS
    - use cases for notify-notifyAll
    - Hidden queue
    - design issues with synchronization
- ***Common Issues with thread***
  - Uncaught Exception Handler
  - problem with stop
  - Dealing with InterruptedStatus
- ***Java Memory Model(JMM)***
  - Sequential Consistency would disallow common optimizations
  - Volatile
  - Real Meaning and effect of synchronization
  - The changes in JMM
  - Final
  - Shortcomings of the original JMM
    - Finals not really final
    - Prevents effective compiler optimizations
    - Processor executes operations out of order
    - Compiler is free to reorder certain instructions
    - Cache reorders writes
    - Old JMM surprising and confusing
  - New JMM and goals of JSR-133
    - Simple,intuitive and, feasible
    - Out-of-thin-air safety
    - High performance JVM implementations across architectures

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- Minimal impact on existing code
- Initialization safety
- Preserve existing safety guarantees and type-safety

- ***Applied Threading techniques***
  - Safe Construction techniques
  - Thread Local Storage
  - Thread safety levels
  - UnSafe Construction techniques
- ***Building Blocks for Highly Concurrent Design***
  - Reentrant Lock
    - ReentrantReadWriteLock
    - ReentrantLock
  - CAS
    - Wait-free Queue implementation
    - Optimistic Design
    - Wait-free Stack implementation
    - Hardware based locking
    - ABA problem
      - Markable reference
      - weakCompareAndSet
      - Stamped reference
  - Lock Striping
    - Lock Striping on LinkNodes
    - Lock Striping on table
  - Indentifying scalability bottlenecks in java.util.Collection
    - segregating them based on Thread safety levels
  - Lock Implementation
    - Multiple user conditions and wait queues
    - Lock Polling techniques
    - Based on CAS
    - Design issues with synchronization

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- ***Highly Concurrent Data Structures-Part1***
  - Weakly Consistent Iterators vs Fail Fast Iterators
  - ConcurrentHashMap
    - Structure
    - remove/put/resize lock
    - Almost immutability
    - Using volatile to detect interference
    - Read does not block in common code path
- ***Java8***
  - Introducing streams
    - What are streams?
    - Streams vs collections
    - Getting started with streams
    - Stream operations
  - Working with streams
    - Putting it all into practice
    - Mapping
    - Finding and matching
    - Building streams
    - Reducing
    - Numeric streams
    - Filtering and slicing
  - Collecting data with streams
    - Reducing and summarizing
    - The Collector interface
    - Grouping
    - Developing your own collector for better performance
    - Collectors in a nutshell
    - Partitioning
  - (Java8's new and notable)Advanced Collections and Collectors

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- Strings
- Reduction as a Collector
- Enter the Collector
- Collection Niceties
- Method References
- Into Other Collections
- Refactoring and Custom Collectors
- Composing Collectors
- Grouping the Data
- To Values
- Partitioning the Data
- Element Ordering

Lambda Expressions

- Functional Interfaces
- Type Inference
- How to Spot a Lambda in a Haystack
- Your First Lambda Expression
- Using Values

Lambda Libraries

- Binary Interface Compatibility
- Default Methods and Subclassing
- Using Lambda Expressions in Code
- Tradeoffs
- @FunctionalInterface
- Static Methods on Interfaces
- Primitives
- Overload Resolution
- The Three Rules
- Multiple Inheritance
- Default Methods

Lambda:Design and Architectural Principles

- Strategy Pattern

- Command Pattern
- Observer Pattern
- The Dependency Inversion Principle
- The Single Responsibility Principle
- The Open/Closed Principle
- Lambda-Enabled SOLID Principles
- Lambda-Enabled Domain-Specific Languages
- Template Method Pattern
- Lambda-Enabled Design Patterns
- A DSL in Java

- ***Java8 Concurrency***
  - Parallel data processing and performance
    - The fork/join framework
    - Parallel streams
    - Spliterator
  - Lambda-Enabled Concurrency
    - Reactive Programming
    - Callbacks
    - When and Where
    - Message Passing Architectures
    - Completable Futures
    - Futures
    - Why Use Nonblocking I/O?
    - The Pyramid of Doom
  - Data Parallelism
    - Parallel Stream Operations
    - Why Is Parallelism Important?
    - Performance
    - Parallel Array Operations
    - Simulations
    - Parallelism Versus Concurrency
    - Caveats

- Java8 Atomics and Locks
  - StampedReference
  - DoubleAdder
  - LongAdder
  - DoubleAccumulator
  - LongAccumulator