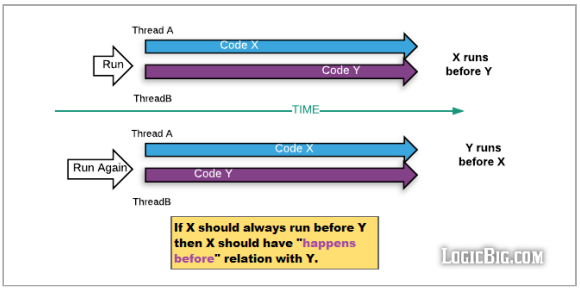
Previous Page | Next Page

Java - Understanding Happens-before relationship

[Updated: May 19, 2018, Created: May 26, 2016]



Happens-before relationship is a guarantee that action performed by one thread is visible to another action in different thread.

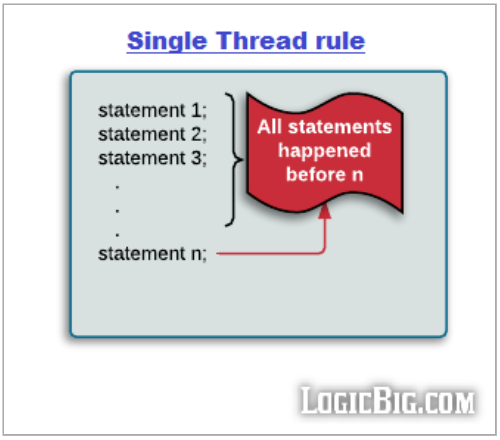
Happens-before defines a partial ordering on all actions within the program. To guarantee that the thread executing action Y can see the results of action X (whether or not X and Y occur in different threads), there must be a happens-before relationship between X and Y. In the absence of a happens-before ordering between two operations, the JVM is free to reorder them as it wants (JIT compiler optimization).

Happens-before is not just reordering of actions in 'time' but also a guarantee of ordering of read and write to memory . Two threads performing write and read to memory can be consistent to each other actions in terms of clock time but might not see each others changes consistently (Memory Consistency Errors) unless they have happens-before relationship.

How to establish happens-before relation?

Followings are the rules for happens-before:

- **Single thread rule**: Each action in a single thread happens-before every action in that thread that comes later in the program order.



- **Monitor lock rule**: An unlock on a monitor lock (exiting synchronized method/block) happens-before every subsequent acquiring on the

Recent Tutorials

Java 11 - java.lang.Class Changes

Java 11 - Nested-Based Access Control (JEP 181)

Java 11 - Local-Variable Syntax for Lambda Parameters (JEP 323)

Java 11 - java.nio.file.Path Changes

Java 11 - java.nio.file.Files Changes

Spring Cloud - Load Balancing And Running Multiple Instances of a Microservice

Spring Cloud - Using RestTemplate as a Load Balancer Client with NetFlix Ribbon

Java 11 - String Changes

Java 11 - Collection Changes

Java 11 - Removal of Java EE modules (JEP 320)

Java 11 - Getting started with HTTP Client API

TypeScript - Function Overloading

TypeScript - 'this' Parameter in functions

TypeScript - Using --noImplicitThis flag

TypeScript - Rest Parameters

TypeScript - Optional And Default Parameters in Functions

TypeScript - Function Types

Spring Data JPA - Using @Procedure with JPA named queries

Spring Data JPA - Using @Procedure annotation to call database stored procedure

JPA - Using @NamedStoredProcedureQuery to call database stored procedures

JPA - Calling Database Function

Jackson JSON - Using @JsonManagedReference and @JsonBackReference for circular references

Jackson JSON - Using @JsonIdentityInfo to handle circular references

Jackson JSON - Using @JsonTypeId to override polymorphic type information

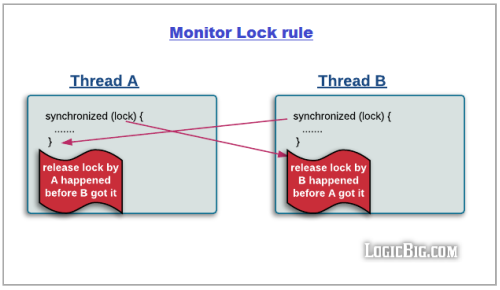
Spring Data JPA - Dynamic Projections

TypeScript - Interface Extending Classes

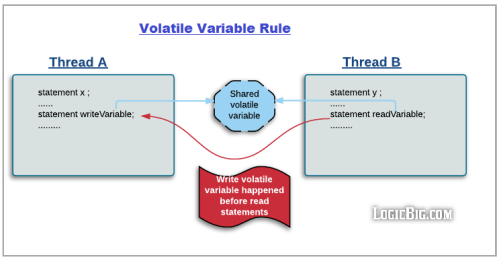
TypeScript - Interface Hybrid Types

TypeScript - Interface Extending Interfaces

same monitor lock.



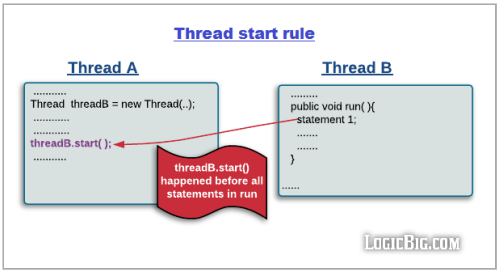
- **Volatile variable rule**: A write to a volatile field happens-before every subsequent read of that same field. Writes and reads of volatile fields have similar memory consistency effects as entering and exiting monitors (synchronized block around reads and writes), but without actually aquiring monitors/locks.



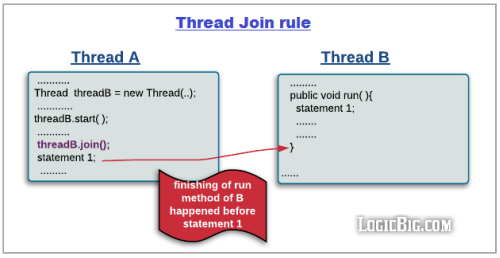
- **Thread start rule**: A call to Thread.start() on a thread happens-before every action in the started thread. Say thread A spawns a new thread B by calling threadA.start(). All actions performed in thread B's run method will see thread A's calling threadA.start() method and before that (only in thread A)

TypeScript - Class Implementing Interfaces
JPA - Calling HSQLDB Stored Procedure involving Cursor to get result set
JPA - Calling Stored Procedure With Ref Cursor Output Parameter
Installing Oracle Jdbc Driver to local Maven Repository
Connecting Oracle Database in JPA
JPA - Calling Stored Procedures
Hibernate - Creating Custom ImportSqlCommandExtractor to load scripts containing stored procedures/functions
Spring Cloud - Getting Started Example
Jackson JSON - Using @JsonRootName to customize POJO name to be serialized
Jackson JSON - Using @JsonValue to serialize a single value returned by a method or field
Jackson JSON - Using @JsonRawValue to serialize property as it is
TypeScript - Using Interfaces to describe Indexable Types
TypeScript - Using Interfaces to describe Functions
TypeScript - Interfaces with Read-Only Properties
TypeScript - Interfaces with Optional Properties
TypeScript - Type Assertions
TypeScript - Using Interfaces to describe Object Properties
TypeScript - Type Inference
Spring Data JPA - Class Based Projections
JPA - Using KEY, VALUE and ENTRY keywords to access Map Entity Relationship in JPQL
JPQL - Using keywords KEY, VLUE, ENTRY to navigate Map element collections
JPA Criteria API - Path Navigation
Fetch Joins in Criteria API
Java Swing - Using OverlayLayout to arrange components over the top of each other
Jackson JSON - Using @JsonPropertyOrder annotation to define serialized properties ordering

happened before them.



- **Thread join rule**: All actions in a thread happen-before any other thread successfully returns from a join on that thread. Say thread A spawns a new thread B by calling threadA.start() then calls threadA.join(). Thread A will wait at join() call until thread B's run method finishes. After join method returns, all subsequent actions in thread A will see all actions performed in thread B's run method happened before them.



- **Transitivity**: If A happens-before B, and B happens-before C, then A happens-before C.

Jackson JSON - Using @JsonEnumDefaultValue to mark enum element as default

Jackson JSON - Using @JsonAnySetter to deserialize unmapped JSON properties

Jackson JSON - Using @JsonAnyGetter Annotation to serialize any arbitrary properties

TypeScript - Parameter Properties

TypeScript - Readonly Modifier

TypeScript - Abstract Classes

TypeScript - Access Modifier: public, private and protected access

TypeScript - Inheritance

TypeScript - Class Syntax, Constructor, Member Variables, Methods and Getters/Setters

Spring Data JPA - Invoking Bean Methods from Projections' SpEL expressions

Spring Data JPA - Projections Using Default Methods

See Also

- A general description of concurrency and multithreading
- Happens-before specs, JLS 17.4.5
- Thread Livelock
- Thread Starvation and Fairness
- Thread Communication using wait/notify
- Deadlock
- Synchronized Blocks
- Intrinsic Locks and Synchronization
- Thread interference, Race Condition and Synchronization
- Thread states
- Thread Interrupts
- Thread Joining
- Java Concurrency quick examples

Java 11 Tutorials

- Java 11 Features

Java 10 Tutorials

- Java 10 Features

Java 9 Tutorials

- Java 9 Module System
- Java 9 Misc Features
- Java 9 JShell

Recent Tutorials

- Java 11 - java.lang.Class Changes
- Java 11 - Nested-Based Access Control (JEP 181)
- Java 11 - Local-Variable Syntax for Lambda Parameters (JEP 323)
- Java 11 - java.nio.file.Path Changes
- Java 11 - java.nio.file.Files Changes
- Spring Cloud - Load Balancing And Running Multiple Instances of a Microservice
- Spring Cloud - Using RestTemplate as a Load Balancer Client with NetFlix Ribbon
- Java 11 - String Changes
- Java 11 - Collection Changes
- Java 11 - Removal of Java EE modules (JEP 320)
- Java 11 - Getting started with HTTP Client API
- TypeScript - Function Overloading
- TypeScript - 'this' Parameter in functions

Share

Previous Page Next Page