Java Interview Questions and Answers

Java/J2ee interview Questions and Answers

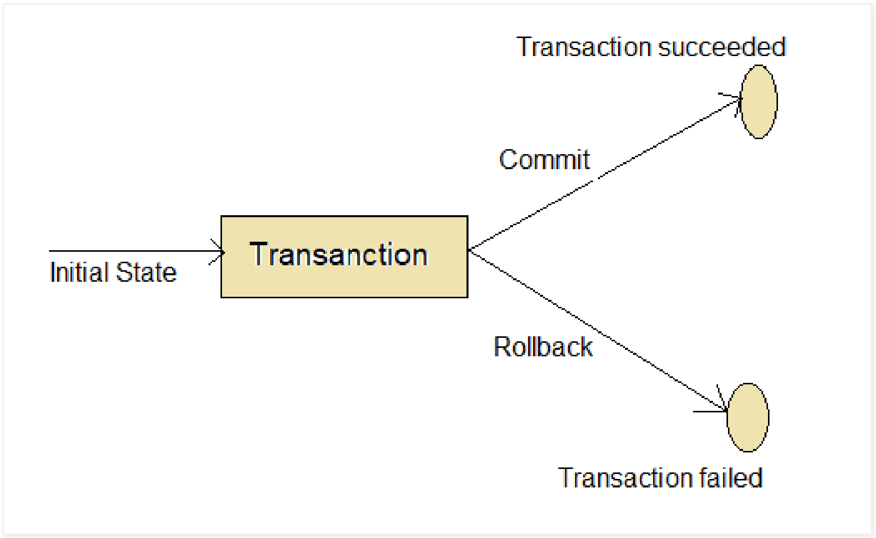
Tuesday, 23 January 2018

Transaction Management in Spring

In previous post, explained the ***CRUD operation of Spring with Hibernate***, this article we can discuss about Spring Transaction Management with simple examples.

A transaction is a logical unit of work that contains one or more statements. A transaction is an atomic unit. The effects of all the statements in a transaction can be either all committed or all rolled back.

See the below diagram to show the transaction management,



Transaction management is an important part of enterprise applications to ensure data integrity and consistency.

The concept of transactions can be described as **ACID(*Atomicity*, *Consistency*, *Isolation*, *Durability*)** property:

- **Atomicity:**

Atomicity requires that each transaction is “all or nothing” which means either the sequence of a transaction is successful or unsuccessful.

If one part of a transaction fails then entire transaction fails.

- **Consistency:**

The database value should be consistent from state to state while any data written to the database for any combination of constraints, cascade, triggers etc.

- **Isolation:**

Each transaction must be executed in isolation in concurrent environment to prevent data corruption.

- **Durability:**

Once a transaction has been committed, the results of this transaction have to be made permanent even in the event of power loss, crashes, or errors.

Spring supports two types of transaction management:

- **Programmatic Transaction Management**
- **Declarative Transaction Management**

Programmatic transaction management:

Programmatic means you have transaction management code surrounding your business code. This gives extreme flexibility, but is difficult to maintain and, well, boilerplate.

Example:--

```
onlineBooking() {
T1.start();
    checkAvailability();
T1.Commit();

T2.start();
    selectItems();
payment();
itemConfirmation();
T2.commit();
}
```

I will discuss this in next upcoming post in details.

Declarative transaction management:

Transaction management is separated from business code and only annotations or XML based configurations are used to manage the transactions.

I will discuss this in upcoming post in details.

Next:

Spring transaction propagation

Propagation is the ability to decide how the business methods should be encapsulated in both logical or physical transactions.

REQUIRED

The same transaction, if already exists, will be used in the current bean method execution context
if the transaction does not exist already then a new transaction will be created, if multiple methods configured with REQUIRED behavior then they will share the same transaction.

REQUIRES_NEW

This behavior means a new transaction will always be created irrespective of whether a transaction exists or not each transaction runs independently

NESTED

This behavior makes nested Spring transactions to use the same physical transaction
but sets savepoints between nested invocations so inner transactions may also rollback
independently of outer transactions.

MANDATORY

This propagation states that an existing opened transaction must already exist. If not an exception will be thrown by the container.

NEVER

This behavior states that an existing opened transaction must not already exist.
If a transaction exists an exception will be thrown by the container. This is totally opposite to *Mandatory* propagation.

NOT_SUPPORTED

The NOT_SUPPORTED behavior will execute outside of the scope of any transaction.

If an opened transaction already exists it will be paused.

SUPPORTS

The SUPPORTS behavior will execute in the scope of a transaction if an opened transaction already exists.If there isn't an already opened transaction the method will execute anyway but in a non-transaction way.

Spring transaction isolation level

Isolation level defines how the changes made to some data repository by one transaction affect other simultaneous concurrent transactions, and also how and when that changed data becomes available to other transactions. When we define a transaction using the Spring framework we are also able to configure in which isolation level that same transaction will be executed.

READ_UNCOMMITTED

This isolation level states that a transaction may read data that is still uncommitted by other transactions.

READ_COMMITTED

This isolation level states that a transaction can't read data that is not yet committed by other transactions.

REPEATABLE_READ

This isolation level states that if a transaction reads one record from the database multiple times the result of all those reading operations must always be the same.

SERIALIZABLE

This isolation level is the most restrictive of all isolation levels. Transactions are executed with locking at all levels (read, range and write locking) so they appear as if they were executed in a serialized way.

Related Posts:--

1) What is IOC Container in Spring? Difference between BeanFactory and ApplicationContext
2) Spring MVC with Hibernate CRUD Example
3) Spring Annotations and examples
4) Spring Configuration Metadata (XML, Annotation and Java)
5) Spring @Qualifier Annotation with example
6) What is Autowiring in Spring ? Explain Autowiring modes and limitations with examples

Posted by Anil Nivargi at 1/23/2018 11:30:00 pm G+

Labels: j2ee interview questions with answers, Spring interview questions and answers

Location:India Bengaluru, Karnataka, India

2 comments:

Unknown 24 January 2018 at 17:46

Thanks for sharing this good blog.This is very important and imformative blog for Java .It's very interesting and useful for students
Java Online Training Hyderabad

Reply

Raj Sharma 23 June 2018 at 12:55

This information you provided in the blog that is really unique I love it!! Thanks for sharing such a great blog Keep posting..
Spring Interview Questions and Answers

Reply