Transaction Management and Caching in Spring

Transaction Management

Transaction management in Spring allows you to ensure that a group of operations is executed as a single unit of work. Transactions help in managing consistency and atomicity,

especially in situations involving databases or distributed systems.

Declarative Transactions:

Spring provides support for declarative transaction management using the `@Transactional` annotation. The `@Transactional` annotation can be applied to classes or specific methods to demarcate transaction boundaries.

Programmatic Transactions:

Programmatic transactions allow manual control of transactions using `PlatformTransactionManager`. This approach is useful when you need more granular control over transaction behavior.

Transaction Propagation:

Propagation determines how transactions behave in relation to one another. Example of propagation types include:

- `REQUIRED`: If there is an existing transaction, join it; otherwise, create a new one.
- `REQUIRES_NEW`: Always create a new transaction, suspending any existing transaction.

Isolation Levels:

Isolation levels define how transactional operations interact with one another in terms of visibility. Examples include:

- `READ_COMMITTED`: Prevents dirty reads but allows non-repeatable reads.
- `SERIALIZABLE`: Highest isolation level, ensuring full consistency but reduces concurrency.

Caching in Spring

Caching improves application performance by reducing the need to repeatedly access a database or external service for frequently requested data.

Declarative Caching:

Caching in Spring is typically managed using annotations such as `@Cacheable`, `@CacheEvict`, and `@CachePut`.

- `@Cacheable`: Caches the result of a method call based on the parameters provided.
- `@CacheEvict`: Used to remove entries from the cache when the data is modified.
- `@CachePut`: Updates the cache with a new value after a method execution.

Cache Expiration:

Cache expiration policies can be configured to automatically remove stale data from the cache. This ensures that the cache remains up to date with the underlying data.

Cache Provider:

Spring supports various cache providers like EhCache, Redis, Hazelcast, etc.

You can select the cache provider that best suits your application requirements.