Configuring Spring and JTA without full Java EE

JOSH LONG (/TEAM/JLONG) AUGUST 15, 2011

2 COMMENTS (/BLOG/2011/08/15/CONFIGURING-SPRING-AND-JTA-WITHOUT-FULL-JAVA-EE#DISQUS_THREAD)

Spring has rich support for transaction management through its `PlatformTransactionManager` interface and the hierarchy of implementations. Spring's transaction support provides a consistent interface for the transactional semantics of numerous APIs. Broadly, transactions can be split into two categories: local transactions and global transactions. Local transactions are those that affect only one transaction resource. Most often, these resources have their own transactional APIs, even if the notion of a transaction is not explicitly surfaced. Often it's surfaced as the concept of a session, a unit of work with demarcating APIs to tell the resource when buffered work should be committed. Global transactions are those that span one or more transactional resources, and enlist them all in a single transaction.

A few common examples of local transactions are in the JMS and JDBC APIs. In JMS, a user can create a transacted Session, send and receive messages and, when the work on the message has completed, call `Session.commit()` to tell the server that it can finalize the work. In the database world, JDBC `Connection`s auto commit queries by default. This is fine for one-off statements, but usually it is preferable to collect several related statements into a batch and then commit all, or none, of them. In JDBC, you do this by first setting the `Connection`'s `setAutoCommit()` method to false, and then explicitly calling `Connection.commit()` at the end of the batch. Both of these APIs, and numerous others, provide the notion of a transactional unit-of-work which may be committed, finalized, flushed or otherwise made permanent at the client's discretion. The APIs are wildly different, but the concepts are the same.

Global transactions a different beast altogether. You should use them anytime you want to let multiple resources participate in a transaction. This is sometimes a requirement: perhaps you want to send a JMS message and write to the database? Or, perhaps you want to use two different persistence contexts with JPA? In global transaction setups, a third party transaction monitor enlists multiple transactional resources in a transaction, prepares them for the commit – in this stage the resource usually does the equivalent of a dry-run commit – and then, finally, commit each of the resources. These steps underlie most global transaction implementations and is called two-phase commit (2PC). If one commit fails (because of causes not present when they prepared for the commit, like a network outage), then the transaction monitor asks each of the resources to undo, or rollback, the last transaction.