Transaction Management with Spring

Bruce Campbell
• This is a training **NOT** a presentation
• Please ask questions
• Prerequisites
  – A computer
Outline

• Transaction Management
  – ACID
  – Demo Vanilla JDBC

• Spring Transaction Management
  – Java stack configuration
  – Programmatic
  – Demo of Transaction Template
  – Declarative
  – Local vs Global & JTA
  – Lab
Transaction Management
ACID Transactions

- ACID is a set of properties that define the reliability of a set of changes (to a database)
  - Atomic - all or nothing
  - Consistent - results are valid
  - Isolated - indicates when changes become visible
  - Durable - changes stick no matter what
ACID by Example

- Transfer funds between accounts
- This is a 2-step process
  - First, remove the money from my account (debit)
  - Second, add the money to your account (credit)
• Atomic
  – If we group both steps together
  – and control that group such that either
    • both steps succeed or
    • nothing gets changed
  – then we’ve achieved atomicity

Two-Step Process
1. Debit my account
2. Credit your account
• **Consistent**: results are valid
  – all keys and constraints remain in-tact
  – my account is debited the same amount that your account is credited - $10
Isolated: when are changes visible?

- If the change is isolated then neither of the changes should be visible through another connection until we indicate that the process is done (commit).
- **Durable**: changes stick
  – even if the database crashes

Two-Step Process
1. Debit my account
2. Credit your account
Transactions

- ACID is a set of properties that define the reliability of a set of changes (to a database)
  - **Atomic** - all or nothing
  - **Consistent** - results are valid
  - **Isolated** - indicates when changes become visible
  - **Durable** - changes stick no matter what

The Atomic and Isolated properties require demarcation of scope... i.e. the boundaries of a transaction
Demo
Transaction in Vanilla JDBC
Spring Transaction Management
Spring Transaction Management

- Spring provides a consistent abstraction for transaction management
  - declarative and programatic methods
  - consistent between JDBC, JPA/Hibernate etc.
  - nicely integrated with other Spring stuff
  - intuitive and easy to use
Spring Transaction Management

• Configuration in the Java Stack
  – ApplicationContext.xml
    • `<stack-db:transaction-manager />`
      – Configures a transaction manager named transactionManager
        » JpaTransactionManager for JPA
        » DataSourceTransactionManager otherwise
    • `<tx:annotation-driven />`
      – Configures the application to use @Transactional annotations
to demarcate transaction boundaries declaratively
Spring Transaction Management

- Two means of doing transactions in Spring programmatically
  - PlatformTransactionManager
    - not covered in this training
  - TransactionTemplate...
• TransactionTemplate
  – callback approach similar to JdbcTemplate
  – handles much of the boilerplate code
  – releases transactional resources
  – but couples your code to Spring’s transaction framework
  – per Spring, use it only when necessary
Demo

Spring TransactionTemplate
• Spring **declarative** transaction management
  – Simple
  – Non-invasive
  – handles much of the boilerplate code
  – releases transactional resources
  – Most common method
  – Recommended method (v.s. programatic)
Spring Transaction Management

• Declarative transactions can be configured via xml or annotations - we’ll cover annotations
• Specify transactional behavior on a per-method basis
• Spring takes care of the rest (using AOP)
• Mark the contents of a method to be within a transaction with @Transactional
  – Use on the class definition or public method definition
  – avoid the interface
  – Method annotations override the class annotation
@Service("exampleService")
@.Transactional(readOnly=true)
public class ExampleServiceImpl implements ExampleService {
    ...
    @Transactional
    public void createExample(Example example) {
        ...
    }
    public List<Example> getAllExamples() {
        ...
    }
}
Flow of Control Without a Transaction

1. Target Method Invoked

2. Result returned to caller
Flow of Control with a Spring Transaction

1. Caller invokes Proxy
2. Transaction created
3. Target Method Invoked
4. Transaction committed or rolled back
5. Result returned to caller
Spring Transaction Management

• Properties of @Transactional
  – value: name of the tx manager to use
  – propagation
  – isolation
  – readOnly
  – timeout
  – rollbackFor & rollbackForClassname
  – noRollbackFor & noRollbackForClassname
Spring Transaction Management

• Propagation property: defines what should happen when @Transactional is encountered and a transaction is already in progress
  – Propagation.REQUIRED - join the current transaction (default)
  – Propagation.REQUIRES_NEW - suspend the current transaction and start a new one
  – Propagation.NESTED - sets a new save point so the inner transaction can roll back without effecting the outer transaction(s)
  – See the Spring docs for others
Spring Transaction Management

- Isolation property: degree to which this transaction is isolated from the work of other transactions
  - DEFAULT - use the underlying datastore’s value (the default if not specified)
  - READ_COMMITTED - dirty reads are not allowed
  - SERIALIZABLE - dirty reads, non-repeatable reads and phantom reads not allowed
  - Others not supported by Oracle RDBMS
Spring Transaction Management

- readOnly property
  - true or false
  - default is read/write (readOnly=false)
  - “Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.”
Spring Transaction Management

- timeout property
  - in seconds
  - How long this transaction runs before timing out and being rolled back
  - defaults to the default timeout of the underlying transaction system
• `rollbackFor` & `noRollbackFor` properties
  – array of exception classes that cause rollback - must derive from throwable

• `rollbackForClassname` & `noRollbackForClassname` properties
  – same as above but provide a string of the class name
• Rollback Rules
  – runtime exceptions will cause a rollback
  – checked exceptions must be declared
Spring Transaction Management

- rollbackFor & noRollbackFor properties
  - array of exception classes that cause rollback - must derive from throwable

- rollbackForClassname & noRollbackForClassname properties
  - same as above but provide a string of the class name
Spring Transaction Management

• Things to consider
  – Oracle implicitly commits after DDL statements
  – Nested transactions do not work with JPA
  – In proxy mode of spring AOP, only external method calls coming in through the proxy are intercepted. A method within the target object calling another method of the same class will not lead to transaction creation\(^1\)
Flow of Control with a Spring Transaction

1. Caller invokes Proxy
2. Transaction created
3. Target Method Invoked
4. Transaction committed or rolled back

Spring Transaction Management

Subsequent Method Calls

AOP Proxy

Target Method

Transaction Advisor
Local vs. Global Transactions
Transactions

- Local Transactions
  - span one resource
  - a.k.a. resource local

- Example: Transferring funds between bank accounts within a single database using the same database connection
Transactions

- Global Transaction
  - span multiple resources
  - a.k.a. XA Transaction, distributed transaction
  - require two phase commits
  - complicated to configure properly for recovery

- Example: Transferring funds between banks
  - using a database connection to debit one account
  - and a web service to credit the other account
Transactions

• JTA (Java Transaction API)
  – a specification that defines standard interfaces between a transaction manager and the parties involved in a distributed transaction
  – http://www.oracle.com/technetwork/java/javaee/jta/
Transactions

• Spring transaction management can interface with JTA
  – Websphere’s, Atomikos, etc.

• The Stack team recommends local transactions
  – If you think you need XA transactions please talk to a member of the stack team
• Declarative Transaction Lab
• https://tech.lds.org/wiki/Database_Development_2#Lab_2
• Summary
  – modify createExample() to perform 2 tasks
  – put the method into a transaction
  – verify
Solution
Transaction Lab
• Transactions help ensure data integrity
• Spring’s Declarative Transaction Management
  – Simple
  – Non-invasive
  – handles much of the boilerplate code
  – releases transactional resources
  – So use it
Questions?