Spring Transaction

Objectives

- Explore and Implement Spring JDBC
- Define Transaction and Transaction Properties
- Introduction to Spring Transaction
- Spring Tranasction Management Support
- Enabling Spring Declarative Tranasction Support

Introduction to Spring JDBC

- **The JdbcTemplate**
  - Spring JDBC provides a central class which takes care of creation and release of resources such as creating and closing of connection object etc.
  - It internally uses JDBC API, but eliminates a lot of problems of JDBC API.
  - It is part of **org.springframework.jdbc.core.JdbcTemplate**.
  - It handles the exception and provides the informative exception messages by the help of exception classes defined in the **org.springframework.dao** package.

Introduction to Spring JDBC(Contd.)

- **Problems of JDBC API**
  - We need to write a lot of code before and after executing the query, such as creating connection, statement, closing Resultset, Connection etc.
  - We need to perform exception handling code on the database logic.
  - We need to handle transaction.
  - Repetition of all these codes from one to another database logic is a time consuming task.

Methods of Spring JdbcTemplate class

No.	Method	Description
1)	public int update(String query)	is used to insert, update and delete records.
2)	public int update(String query,Object... args)	is used to insert, update and delete records using PreparedStatement using given arguments.
3)	public void execute(String query)	is used to execute DDL query.
4)	public List query(String sql, RowMapper rse)	is used to fetch records using RowMapper.

Introduction to Spring JDBC(Contd..)

- **DriverManagerDataSource**
  - In order to work with **JdbcTemplate** class we need to register a bean which is type of **DriverManagerDataSource**
  - The **DriverManagerDataSource bean** contains the information about the database such as **driver class name**, **connection URL**, **username** and **password.**
  - There is a property named datasource in the **JdbcTemplate** class of **DriverManagerDataSource type**
  - So, we need to provide the reference of **DriverManagerDataSource** object in the **JdbcTemplate** class for the datasource property.

What is Transaction?

- **Transaction**
    - A transaction is a unit of work in which either all operations must execute or none of them.

- **There are four important Properties to understand about Transaction.**
    - **Atomic** - This property makes sure that either all operations within a transaction must be successful or none of them.
    - **Consistent**- This property makes sure that data should be in consistent state once the transaction is completed.
    - **Isolated-** this property allows multiple users to access the same set of data and each user's processing should be isolated from others.
    - **Durable** – Result of the transaction should be permanent once the transaction is completed to avoid any loss of data.

Spring Transaction Management

- Spring provides extensive support for transaction management and help developers to focus more on business logic rather than worrying about the integrity of data incase of any system failures.

- Note: Boilerplate code or boilerplate is the sections of code that have to be included in many places with little or no alteration.

Spring Transaction Management Support

- **Spring provides support for both programmatic and declarative transactions management**
  - **Programmatic Transactions** –
    - With programmatic transactions , transaction management code like, commit when everything is successful or rolling back if anything goes wrong is clubbed with the business logic.
    - Programmatic transaction is mixed with your business logic hence it is tightly coupled and you have to boiler-plate code in your application.
  - **Declarative Transactions**-
    - Declarative transactions separates transaction management code from business logic. Spring supports declarative transactions using transaction advice (using AOP).
    - Spring declarative transaction management addresses these concerns by using Aspect Oriented Programming to achieve loose coupling and avoid boiler-plate code in our application.

Choosing Transaction Manager

- **Choosing Transaction Manager**
  - Spring supports several transaction managers which delegate the transaction management responsibilities to platform specific implementations.
  - Plarform Transaction manager is the parent of all transaction manager implementations.

- **Different types of Transaction Managers**
  - DataSource Transaction Manager
  - Hibernate Transaction Manager
  - Jdo Transaction Manager
  - JTA Transaction manager

Different types of Transaction Managers

- **DataSource Transaction manager** -
  - We can use DataSourceTransactionManager for simple JDBC persistence mechanism.
- **Sample configuration of DataSourceTransactionManager looks like below**

```
<bean id="transactionManager"
 class="org.springframework.jdbc.datasource.DataSourceTransactionManager>
<property name="dataSource" ref= "datasource" />
</bean>
```

- **Hibernate Transaction manager** –
  - Hibernate transaction manager should be used when our application is using Hibernate.
- **Sample configuration of HibernateTransactionManager looks like below**

```
<bean id="transactionManager"
     class="org.springframework.orm.hibernate3.HibernateTransactionManager>
    <property name="sessionFactory" ref= "sessionFactory" />
    </bean>
```

Different types of Transaction Managers(contd..)

infogain
www.infogain.com

- **Jdo Transaction manager**
  - **Use below configuration to use Java data object transaction manager .**

```
<bean id=”transactionManager”
  class=”org.springframework.orm.jdo.JdoTransactionManager>
 <property name=”persistanceManagerFactory” ref= “persistanceManagerFactory” />
</bean>
```

- **JTA Transaction manager**
  - If you have to use transaction across multiple data sources than we need to use **Java Transactions API** transactions .
  - Internally JTA implementation handles transaction responsibility.

- **Use below configuration to configure JTA transaction manager.**

```
  <bean id=”transactionManager”
 class=”org.springframework.transaction.jta.JtaTransactionManager>
<property name=”transactonManagerName” ref= “java:/TransactionManager” />
 </bean>
```

Enabling Spring Declarative Transaction Management

- **To use the annotation style transaction you have to add some bean configuration in your xml file i.e:**
  - **<tx:annotation-driven/>**: Automatically adds transaction support which eventually wraps your code in transaction scope
  - **Initializing DataSourceTransactionManager bean**

- **Important Points About Spring Bean Configuration File:**
  - **tx:annotation-driven : E**lement is used to tell Spring context that we are using annotation based transaction management configuration.
  - **transaction-manager** : Attribute is used to provide the transaction manager bean name.
  - **proxy-target-class attribute** is used to tell Spring context to use class based proxies, without it you will get runtime exception with message such as Exception in thread "main" **org.springframework.beans.factory.BeanNotOfRequiredTypeException**:

Spring Declarative transaction management(contd...)

www.infogain.com

- Since we are using JDBC, we are creating transactionManager bean of type **org.springframework.jdbc.datasource.DataSourceTransactionManager.** This is important we should use proper transaction manager implementation class based on our transaction API use.
- **dataSource bean** is used to create the **DataSource** object and we are required to provide the database configuration properties such as **driverClassName, url, username and password**.
- We are injecting **dataSource** into **customerDAO** bean. Similarly we are injecting **customerDAO** bean into **customerManager** bean definition.

Spring Declarative transaction management(contd...)

- **@Transactional Annotation**
  - It is used to add declarative transactions management.
  - It can be used at method level or class level. @Transactional at class level wraps all method in transaction scope.
  - The @Transactional annotation has several properties like readOnly, isolation,rollbackFor, noRollbackFor etc that can be used to control how one transaction behaves and communicate with other transactions.

infogain

Thank You

infogain

www.infogain.com

Infogain Corporation, HQ
485 Alberto Way Los Gatos,
CA 95032 USA
Phone: 408-355-6000
Fax: 408-355-7000

Pune
7th Floor, Bhalerao Towers, CTS No.1669 -
1670, Behind Hotel Pride,
Shivaji Nagar, Pune - 411005
Phone : +91-20-66236700

Infogain Irvine
41 Corporate Park,
Suite 390 Irvine, CA 2606 USA
Phone: 949-223-5100
Fax: 949-223-5110

Infogain Austin
Stratum Executive Center Building D
11044 Research Boulevard Suite 200
Austin, Texas 78759

Noida
A-16, Sector 60, Noida Gautam Budh agar,
201301 (U.P.) India
Phone: +91-120-2445144
Fax: +91-120-2580406

Dubai
P O Box 500588 Office No.105,
Building No. 4, Dubai Outsource Zone,
Dubai, United Arab Emirates
Tel: +971-4-458-7336