HCL TALENTCARE

HCL TALENTCARE

Spring with Hibernate

HCL

Overview

- Spring can simplify your Hibernate application. Spring's Hibernate integration uses the same generic transaction infrastructure and DAO exception hierarchy that it uses for JDBC, JDO, iBATIS, and TopLink. This makes it easy to mix and match persistence methodologies, if necessary.

Objectives

- After completing this chapter, you will be able to:
  - Recognize the benefits of integrating Spring with Hibernate.
  - Identify how to use HibernateTemplate to do database operations.
  - Use Hibernate contextual session.

Do You Know

- What is ORM?
- Name some famous ORM tools.

What is Hibernate?

HCL TALENTCARE

- It is the object/relational mapping framework for Java.
- It builds persistent objects following common Java idioms:
  - Association
  - Inheritance
  - Polymorphism
  - Composition
  - Collections API for “many” relationships.

Why Hibernate?

HCL TALENTCARE

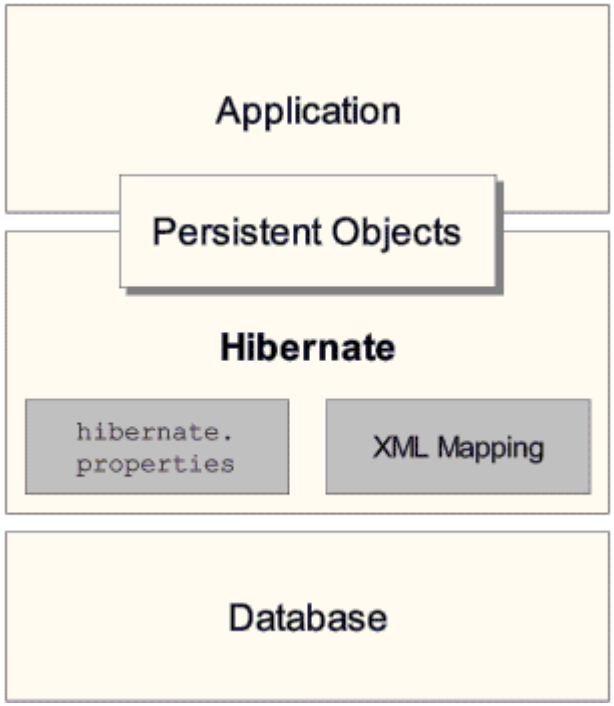
- Focus on domain object modeling.
- Performance.
  - High performance object caching.
- Sophisticated query facilities
  - Hibernate Query Language (HQL)
  - Criteria API and Query by Criteria
  - Query by Example.

HCL

High Level Architecture of Hibernate

HCL TALENTCARE

- Role of Hibernate in a Java Application



Hibernate Application

- A Hibernate application consists of three parts:
  - Hibernate Mapping (XML) File
  - Hibernate Properties File/xml file
  - Hibernate Java Library

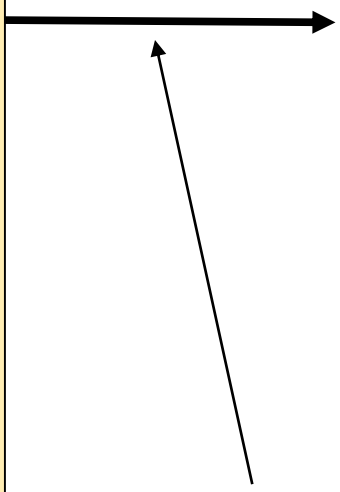
- And (of course):
  - Java Class Files (Persistent Objects)
  - Database Schema

Hibernate Mapping

HCL TALENTCARE

JavaObject

```
public class Course {


    private Long id;

    private String
title;

    private Date
beginDate;

    private Date
endDate;

    private int fee;

    …
}
```



SQL Table

```
ID          [long]
primary key,
TITLE[String]
BEGIN_DATE[date]
END_DATE[date]
FEE[int]
```

Magic Happens Here
(O/R Mapper – i.e. Hibernate)

HCL

Hibernate Mapping (XML) File:course.hbm.xml

HCL TALENTCARE

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
       "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping package="com.spring.course">
  <class name="Course" table="COURSE">
    <id column="ID" name="id" type="long">
      <generator class="increment"/>
    </id>
    <property name="title" type="string">
      <column length="100" name="TITLE" not-null="true"/>
    </property>
    <property column="BEGIN_DATE" name="beginDate"
type="date"/>
    <property column="END_DATE" name="endDate" type="date"/>
    <property column="FEE" name="fee" type="int"/>
  </class>
</hibernate-mapping>
```


HCL

Hibernate Configuration File

- Once the mapping files for the persistent objects have been completed, its now time to configure Hibernate. This will have all the information to connect Hibernate to a database.

Hibernate Configuration File

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">
 <hibernate-configuration>
     <session-factory> <!-- Database connection settings -->
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver
       </property>
        <property
name="connection.url">jdbc:derby://localhost:1527/coursedb;create
=true
       </property>
        <property name="connection.username">app</property>
        <property name="connection.password">app</property>
 <!—Derby dialect -->
       <property
name="dialect">org.hibernate.dialect.DerbyDialect</property>
       <mapping resource="com/spring/course/Course.hbm.xml"/>
</session-factory> </hibernate-configuration>
```

Main Program

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
…
public class Main{
  public static void main(String[] args) {
    Session session = null;
    Transaction tx =null;
    try{
      // This step will read hibernate.cfg.xml  and prepar
e hibernate for use
      SessionFactory sessionFactory = new
        Configuration().configure().buildSessionFactory();
       session =sessionFactory.openSession();
       tx = session.beginTransaction();
        //
Create new instance of Course and set values in it
        Course course = new Course();
```

Main Program (Contd.)

```
    course.settTitle("Spring 3.0");
       // other setter methods…
       tx.begin();
        session.save(course);
        System.out.println("Done");
        tx.commit();
    }catch(Exception e){
      System.out.println(e.getMessage());
     tx.rollback();
    }finally{
      // Actual contact insertion will happen at this step
      session.flush();
      session.close();
      }
     }
}
```

Interactive Activity

1. What is an ORM FRAMEWORK?

2. State whether the given statement is True or False: hibernate.cfg.xml configures the session object

Why Spring?

HCL TALENTCARE

- When using an Hibernate on its own, you need to build the session factory and manage it manually.
- Spring provides several factory beans for you to create a Hibernate session factory as a singleton bean in the IoC container.
- This factory can be shared between multiple beans via dependency injection.
- Moreover, this allows the session factory to integrate with other Spring data access facilities,such as data sources and transaction managers.

HCL

Configuring a Hibernate Session Factory in Spring

HCL TALENTCARE

- Configuring Hibernate Session Factory

```
import org.hibernate.SessionFactory;
public class HibernateCourseDao implements CourseDao {
    private SessionFactory sessionFactory;

   public void setSessionFactory(SessionFactory
sessionFactory) {
       this.sessionFactory = sessionFactory;
}
...
}
```


HCL

Configuring a Hibernate Session Factory in Spring (Contd.)

HCL TALENTCARE

```
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/
beans
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFact
oryBean">
         <property name="configLocation"
value="classpath:hibernate.cfg.xml" />
</bean>
<bean id="courseDao"
 class="com.spring.hibernate.HibernateCourseDao">
       <property name="sessionFactory"
ref="sessionFactory" />
</bean>
</beans>
```


HCL

Configuring a Hibernate Session Factory in Spring (Contd.)

HCL TALENTCARE

```
public class Main {

public static void main(String[] args) {

ApplicationContext context =
       new ClassPathXmlApplicationContext("beans-
hibernate.xml");
CourseDao courseDao = (CourseDao)
context.getBean("courseDao");
...
}
}
```

Configuring a Hibernate Session Factory in Spring (Contd.)

HCL TALENTCARE

- You can even ignore the Hibernate configuration file by merging all the configurations into LocalSessionFactoryBean.

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFact
oryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="mappingResources">
        <list>
        <value>com/spring/hibernate/Course.hbm.xml</value>
       </list>
     </property>
    <property name="hibernateProperties">
       <props>
           <prop
key="hibernate.dialect">org.hibernate.dialect.DerbyDialect
</prop>
           <prop key="hibernate.show_sql">true</prop>
           <prop
key="hibernate.hbm2ddl.auto">update</prop>
```

Activity

- State whether the given statement is True or False: Spring provides factory methods to create the Hibernate Session factory and inject it between multiple beans.

Hibernate Template (Contd.)

HCL TALENTCARE

- Spring defines the HibernateTemplate classes to provide template methods for different types of Hibernate operations to minimize the effort involved in using them.

```
public class HibernateCourseDao implements CourseDao {
    private HibernateTemplate hibernateTemplate;

    public void setHibernateTemplate(HibernateTemplate
hibernateTemplate) {
        this.hibernateTemplate = hibernateTemplate;
    }
    public void store(Course course) {
       hibernateTemplate.saveOrUpdate(course);
   }
   public void delete(Long courseId) {
       Course course = (Course)
hibernateTemplate.get(Course.class, courseId);
   hibernateTemplate.delete(course);
} … }
```

Hibernate Template (Contd.)

HCL TALENTCARE

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/
beans
   http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd
   http://www.springframework.org/schema/tx
   http://www.springframework.org/schema/tx/spring-tx-
3.0.xsd">
…
  <bean id="hibernateTemplate"
class="org.springframework.orm.hibernate3.HibernateTemplat
e">
  <property name="sessionFactory" ref="sessionFactory" />
  </bean>
  <bean name="courseDao"
     class="com.spring.course.HibernateCourseDao">
     <property name="hibernateTemplate"
ref="hibernateTemplate" />
  </bean>
```

Hibernate Template (Contd.)

HCL TALENTCARE

- If you want to get access to the underlying Hibernate session in order to perform native Hibernate, you can implement the **HibernateCallback** interface.

- This will give you a chance to use any implementation-specific features directly if there is not sufficient support already available from the template implementations.

```
hibernateTemplate.execute(new HibernateCallback() {
                public Object doInHibernate(Session session)
throws

HibernateException, SQLException {
// ... anything you can imagine doing can be done here.
// Cache invalidation, for example…
}
};
```

HibernateDaoSupport

```
Importorg.springframework.orm.hibernate3.support.Hibernate
DaoSupport;
public class HibernateCourseDao extends
HibernateDaoSupport implements   CourseDao {
   public void store(Course course) {
          getHibernateTemplate().saveOrUpdate(course);
   }
   public void delete(Long courseId) {
          Course course = (Course)
getHibernateTemplate().get(Course.class,courseId);
          getHibernateTemplate().delete(course);
}…
}
```

```
<bean name="courseDao”
class="com.spring.hibernate.HibernateCourseDao">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

Hibernate's Contextual Sessions (Contd.)

HCL TALENTCARE

- Using HibernateTemplate means your DAO (Data Access Object) has to depend on Spring's API.
- An alternative to Spring's HibernateTemplate is to use Hibernate's contextual sessions.
- In Hibernate 3, a session factory can manage contextual sessions for you and allows you to retrieve them by the **getCurrentSession()** method on org.hibernate.SessionFactory.
- Within a single transaction, you will get the same session for each getCurrentSession() method call. This ensures that there will be only one Hibernate session per transaction,

HCL

Hibernate's Contextual Sessions (Contd.)

HCL TALENTCARE

```
@Repository("courseDao")
public class HibernateCourseDao implements CourseDao {
    private SessionFactory sessionFactory;
    …
 @Transactional(readOnly = true)
    public List<Course> findAll() {
        Query query =
sessionFactory.getCurrentSession().createQuery("from

Course2");
        return query.list();
    }
```

Hibernate's Contextual Sessions (Contd.)

HCL TALENTCARE

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/
beans
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-
3.0.xsd">
...
<tx:annotation-driven />
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransac
tionManager">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>
<bean  class="org.springframework.dao.annotation.
PersistenceExceptionTranslationPostProcessor"/>
```


HCL

Test Your Understanding

1. Name the core class in Spring used to interact with Hibernate

2. Name the Spring class used to create an instance of Hibernate 3 SessionFactory.

Summary

- In this module, you have learned how to use Spring's support for Hibernate.
- You have learned how to configure SessionFactory and HibernateTemplate
- You have learned how to write DAO classes using HiberDaoSupport class.
- We have discussed how to use hibernate contextual session within DAO classes.

Source

- www.springframework.org

HCL TALENTCARE

Thank You

HCL

HCL TALENTCARE

HCL TALENTCARE

Spring Transaction Management

HCL

Overview

- Transaction management is an essential technique in enterprise application development to ensure data integrity and consistency. Without transaction management, your data and resources may be corrupted and left in an inconsistent state. Transaction management is particularly important for recovering from unexpected errors in a concurrent and distributed environment.

Objectives

- After completing this chapter you will be able to:
  - Recognize the importance of transaction management.
  - Identify how to implement declarative transaction in Spring.

What is a Transaction?

HCL TALENTCARE

- A transaction is a series of actions that are treated as a single unit of work.
- These actions should either be complete entirely or take no effect at all. If all the actions go well, the transaction should be committed permanently.
- In contrast, if any of them goes wrong, the transaction should be rolled back to the initial state as if nothing had happened.

HCL

Spring Transaction Management

HCL TALENTCARE

- Like the bean-managed transaction (BMT) and container-managed transaction (CMT) approaches in EJB, Spring supports both programmatic and declarative transaction management.

- The aim of Spring's transaction support is to provide an alternative to EJB transactions by adding transaction capabilities to POJO(Plain Old Java Object).

HCL

Programmatic Transaction Management

HCL TALENTCARE

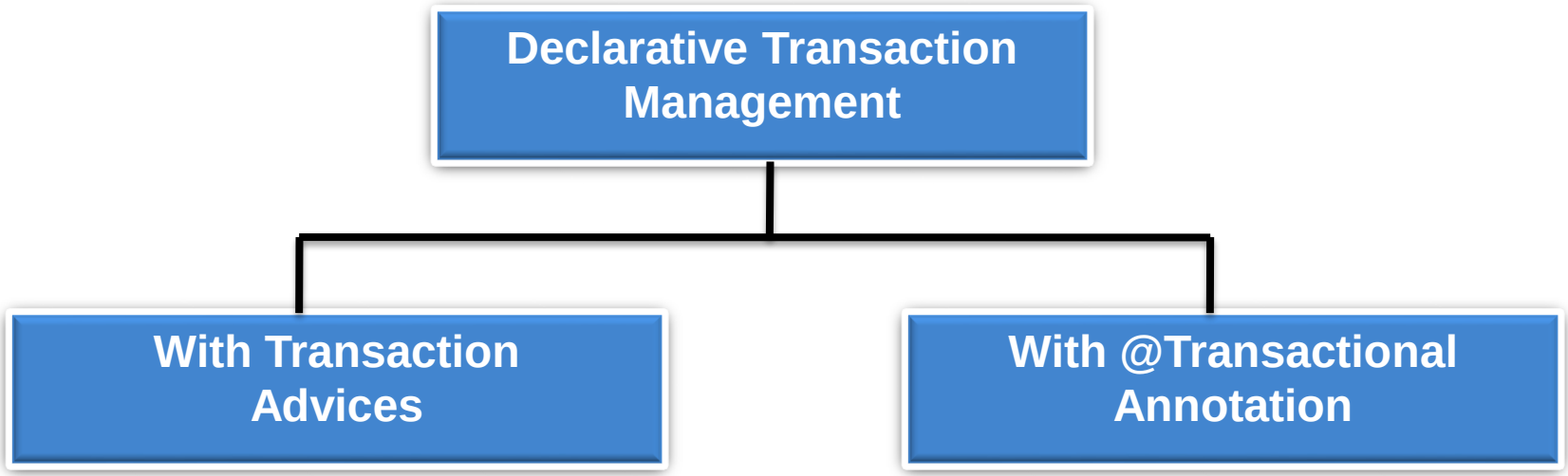
- Programmatic transaction management is achieved by embedding transaction management code in your business methods to control the commit and rollback of transactions.
- You usually commit a transaction if a method completes normally and roll back a transaction if a method throws certain types of exceptions.
- With programmatic transaction management, you can define your own rules to commit and roll back transactions.

HCL

Declarative Transaction Management

HCL TALENTCARE

- Spring supports declarative transaction management through the Spring AOP framework.
- It is achieved by separating transaction management code from your business methods via declarations.
- It is less flexible than programmatic transaction management



HCL

Spring Transaction Manager

HCL TALENTCARE

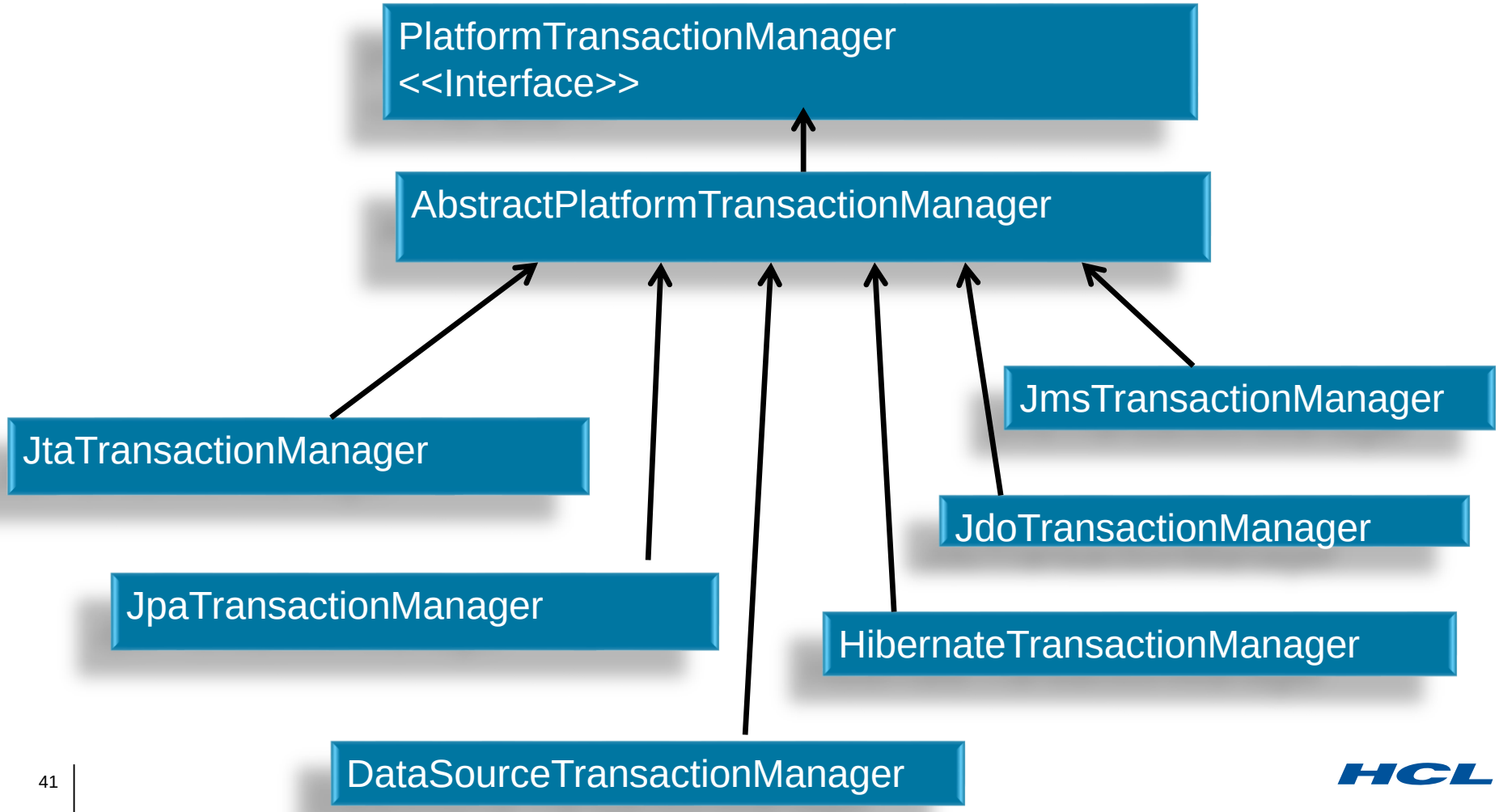
- Spring abstracts a general set of transaction facilities from different transaction management APIs.
- As an application developer, you can simply utilize Spring's transaction facilities without having to know much about the underlying transaction APIs.
- With these facilities, your transaction management code will be independent of any specific transaction technology.
- PlatformTransactionManager is a general interface for all Spring transaction managers.

HCL

Spring Transaction Manager (Contd.)

HCL TALENTCARE

The diagram shows the common implementations of the PlatformTransactionManager interface in Spring



HCL

Spring Transaction Manager (Contd.)

HCL TALENTCARE

- A transaction manager is declared in the Spring IoC container as a normal bean.

```
<bean id="transactionManager"
      class="org.springframework.jdbc.

      datasource.DataSourceTransactionManager">
<property name="dataSource" ref="dataSource"/>
</bean>
```


HCL

Activity

- State whether the given statement is True or False: Spring provides both programmatic and declarative transaction management

@Transactional Annotation

HCL TALENTCARE

- Using @Transactional for a business method

```
importorg.springframework.transaction.annotation.Transactional;
importorg.springframework.jdbc.core.support.JdbcDaoSupport;

public class JdbcBookShop extends JdbcDaoSupport implements
BookShop {
@Transactional
public void purchase(String isbn, String username) {
int price = getJdbcTemplate().queryForInt(
      "SELECT PRICE FROM BOOK WHERE ISBN = ?", new Object[] {
isbn });
getJdbcTemplate().update( "UPDATE BOOK_STOCK SET STOCK =
      STOCK - 1 "+ "WHERE ISBN = ?", new Object[] { isbn });
getJdbcTemplate().update( "UPDATE ACCOUNT SET BALANCE =
           BALANCE - ? "+ "WHERE USERNAME = ?", new Object[] {
price,  username }); }
}
```


HCL

@Transactional Annotation (Contd.)

HCL TALENTCARE

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/bea
ns
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">
<tx:annotation-driven transaction-
manager="transactionManager"/>
...
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransact
ionManager">
<property name="dataSource" ref="dataSource"/>
</bean> …
</beans>
```


HCL

Transaction Attributes

- Transaction attribute defines the rules for applying transaction policies on methods.
  - Propagation behavior
  - Isolation Level
  - Read-only status
  - Timeout
  - Rollback policy

Propagation Behavior

HCL TALENTCARE

- Propagation Behavior is used to define the transaction boundaries.
- When a transactional method is called by another method, it is necessary to specify how the transaction should be propagated.
- For example, the method may continue to run within the existing transaction, or it may start a new transaction and run within its own transaction.

HCL

Propagation Behavior Supported by Spring

HCL TALENTCARE

- PROPAGATION_REQUIRED : Join the current transaction. Create a new transaction, if none exists.
- PROPAGATION_SUPPORTS : Join the current transaction. Execute non-transactionally, if none exists.
- PROPAGATION_MANDATORY: Join the current transaction. Throw exception , if none exists.
- PROPAGATION_REQUIRES_NEW : Create a new transaction. Suspend the current transaction, if one exists.
- PROPAGATION_NOT_SUPPORTED: Execute non-transactionally. Suspend the current transaction, if one exists
- PROPAGATION_NEVER : Execute non-transactionally. Throw an exception, if a transaction exists
- PROPAGATION_NESTED : Execute within a nested transaction if a current transaction exists. Create a new transaction, if none exists

HCL

Propagation Behavior Supported by Spring (Contd.)

HCL TALENTCARE

```
import
org.springframework.transaction.annotation.Propagation;
import
org.springframework.transaction.annotation.Transactional;

public class BookShopCashier implements Cashier {
    …
    @Transactional(propagation = Propagation.REQUIRED)
    public void checkout(List<String> isbns, String
username) {
        for (String isbn : isbns) {
            bookShop.purchase(isbn, username);
        }
    }
…
}
```


HCL

Isolation Level

- When multiple transactions of the same application or different applications are operating concurrently on the same dataset, many unexpected problems may arise.
- The problems caused by concurrent transactions can be categorized into three types:
  - Dirty read
  - Nonrepeatable read
  - Phantom read

Isolation Levels Supported by Spring

HCL TALENTCARE

- DEFAULT: It uses the underlying datastore isolation policy. For most databases, the default isolation level is READ_COMMITTED.
- READ_UNCOMMITTED: It allows to read the data even when there are uncommitted rows. Thus it can lead dirty read or non repeatable read or phantom reads.
- READ_COMMITTED: Only committed rows are read but still it may cause non-repeatable read or phantom reads.
- REPEATABLE_READ: Prevents dirty read as well as non-repeatable issues but phantom reads could be there.
- SERIALIZABLE: This brings the ACID (Atomicity, concurrency, isolation and durability) compliance. However, since it is very strict and restricted, there might be a performance issue.

HCL

Isolation Levels Supported by Spring (Contd.)

HCL TALENTCARE

```
import org.springframework.transaction.annotation.Isolation;
…
public class JdbcBookShop extends JdbcDaoSupport implements
BookShop {
…
@Transactional(isolation = Isolation.REPEATABLE_READ)
public int checkStock(String isbn) {
       String threadName = Thread.currentThread().getName();
     System.out.println(threadName + " - Prepare to check
book stock");
     int stock = getJdbcTemplate().queryForInt("SELECT STOCK
FROM BOOK_STOCK WHERE ISBN = ?", new Object[] { isbn });
        System.out.println(threadName + " - Book stock is " +
stock);
        sleep(threadName);
        return stock; }}
```


HCL

Read Only

HCL TALENTCARE

- If a transaction only reads but does not update data, the database engine could optimize this transaction.
- The read-only flag is just a hint to enable a resource to optimize the transaction, and a resource might not necessarily cause a failure if a write is attempted.

```
public class JdbcBookShop extends JdbcDaoSupport
implements BookShop {
...
@Transactional(
isolation = Isolation.REPEATABLE_READ, readOnly =
true)
public int checkStock(String isbn) {
... }
}
```


HCL

Timeout

HCL TALENTCARE

- Since a transaction may acquire locks on rows and tables, a long transaction will tie up resources and have an impact on overall performance.
- The timeout transaction attribute (in seconds) indicates how long your transaction can survive before it is forced to roll back.
- This can prevent a long transaction from tying up resources.

```
public class JdbcBookShop extends JdbcDaoSupport
implements BookShop {
...
@Transactional(
isolation = Isolation.REPEATABLE_READ, readOnly = true,
timeout=30)
public int checkStock(String isbn) {
... }
}
```


HCL

Rollback

HCL TALENTCARE

- By default, only unchecked exceptions (i.e., of type RuntimeException and Error) will cause a transaction to roll back, while checked exceptions will not.
- The exceptions that cause a transaction to roll back or not can be specified by the rollback transaction attribute.
- Any exception not explicitly specified in this attribute will be handled by the default rollback rule.

Rollback (Contd.)

HCL TALENTCARE

```
public class JdbcBookShop extends JdbcDaoSupport
implements BookShop {
...
@Transactional(
propagation = Propagation.REQUIRES_NEW,
Isolation=Isolation.REPEATABLE_READ),
Timeout=30
rollbackFor = IOException.class,
noRollbackFor = ArithmeticException.class)
public void purchase(String isbn, String username) throws
Exception{
throw new ArithmeticException();
//throw new IOException();
 }
}
```


HCL

Activity

HCL TALENTCARE

- State whether the following statements are TRUE or FALSE
  - @Transactional Annotation can be used at the method level as well as class level but not with interfaces
  - @Transactional attributes can specify: isolation level, timeout for transactions, propagation behavior, read-only status and rollback policy
  - Propagation behavior specifies how a transaction should be propagated when a transactional method is called by another method.
  - By default only checked exceptions will cause a transaction to roll back

HCL

Test Your Understanding

HCL TALENTCARE

- 1) What is the default propagation behavior?
- 2) What transaction attribute is used to ensure that a long running transaction(which hinder performance) doesn't lock the critical resources?

HCL

Summary

- In this module we have discussed about transactions and its importance in business methods.
- We have also discussed about various PlatformTransactionManager available in Spring framework.
- We have explored how to implement declarative transaction management in Spring using @Transactional annotation.

Source

- www.springframework.org

HCL TALENTCARE

Thank You

HCL