Spring Framework
Module 7 – Transactions

Evgeniy Krivosheev
Vyacheslav Yakovenko
Last update: Feb, 2012

Содержание

- ACID
- Типы транзакций
- Обзор API
- Уровни изоляции
- Распространение транзакции между вызовами (Propagation)
- Использование AOP для управления транзакциями
- Программное управление транзакциями
- Примеры
- Упражнения

Spring :: Tx :: ACID

LUXOFT TRAINING

- **Atomicity (Атомарность)** - выполняются или все операции, или ни одной;
- **Consistence (Непротиворечивость)** - после завершения транзакции (успешного или нет) данные остаются в непротиворечивом состоянии;
- **Isolation (Изолированность)** - транзакции позволяют нескольким пользователям одновременно работать с одними и теми же данными;
- **Durability (Длительность)** - как только транзакция завершилась, результат транзакции должен быть сохранен.

Spring :: Tx :: Типы

- Глобальные – управляются сервером приложений с использованием JTA (Java Transaction API);
- Локальные – специфичны для ресурса;

Spring :: Tx

- Модель поддержки транзакций, используемая в Spring Framework, подходит для разных транзакционных API (JDBC, Hibernate, JPA, JDO, etc.)

- Преимущества:
  - НЕ нужен сервер приложений;
  - Декларативное управление транзакциями.

Spring :: Tx API

Основная абстракция для работы с транзакциями в Spring, определяется интерфейсом:

- org.springframework.transaction.PlatformTransactionManager

```
public interface PlatformTransactionManager {
   TransactionStatus getTransaction(TransactionDefinition definition)
        throws TransactionException;
   void commit(TransactionStatus status)
        throws TransactionException;
   void rollback(TransactionStatus status)
        throws TransactionException;
}
```

Spring :: Tx API

Наиболее востребованные имплементации:

- DataSourceTransactionManager ;
- HibernateTransactionManager ;
- JmsTransactionManager ;
- JmsTransactionManager102 ;
- JpaTransactionManager ;
- OC4JJtaTransactionManager ;
- WebLogicJtaTransactionManager ;
- WebSphereUowTransactionManager ;

Spring :: Tx API

LUXOFT TRAINING

Т.к. PlatformTransactionManager:

- это интерфейс – для него легко можно сделать mock или stub, что позволяет облегчить тестирование приложения ;
- не завязан на JNDI, определяется в Spring IoC контейнере ;

Spring :: Tx API

В качестве интерфейса, для задания и получения свойств конкретной транзакции используется интерфейс:

- org.springframework.transaction.TransactionDefinition

Spring :: Tx API

В случае декларативного объявления транзакций, TransactionDefinition создается опосредованно через аннотации, например:

```
@Transactional(readOnly = true)
public class DefaultFooService implements FooService {

   public Foo getFoo(String fooName) {
        // do something
   }

   @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
   public void updateFoo(Foo foo) {
        // do something
   }
}
```

Spring :: Tx API

Однако, для того чтобы приложение смогло работать с транзакциями в декларативной манере, в контексте приложения необходимо добавить декларацию:

```
<tx:annotation-driven/>
```

И внедрить один или более бинов – менеджеров транзакций:

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

Spring :: Tx API

Параметры TransactionDefinition:

- **Isolation** – уровень изоляции транзакций ;
- **Propagation** – как передаются транзакции от метода к методу ;
- **Timeout** ;
- **Read-only** status ;

Spring :: Tx :: Уровни изоляции

LUXOFT TRAINING

- ISOLATION_DEFAULT
  - использует уровень изоляции базы данных
- ISOLATION_READ_UNCOMMITTED
  - Чтение незафиксированных изменений своей транзакции и конкурирующих транзакций
  - Возможны нечистые, неповторяемые чтения и фантомы
- ISOLATION_READ_COMMITTED
  - Чтение всех изменений своей транзакции и зафиксированных изменений конкурирующих транзакций
  - Нечистые чтения невозможны
  - Возможны неповторяемые чтения и фантомы

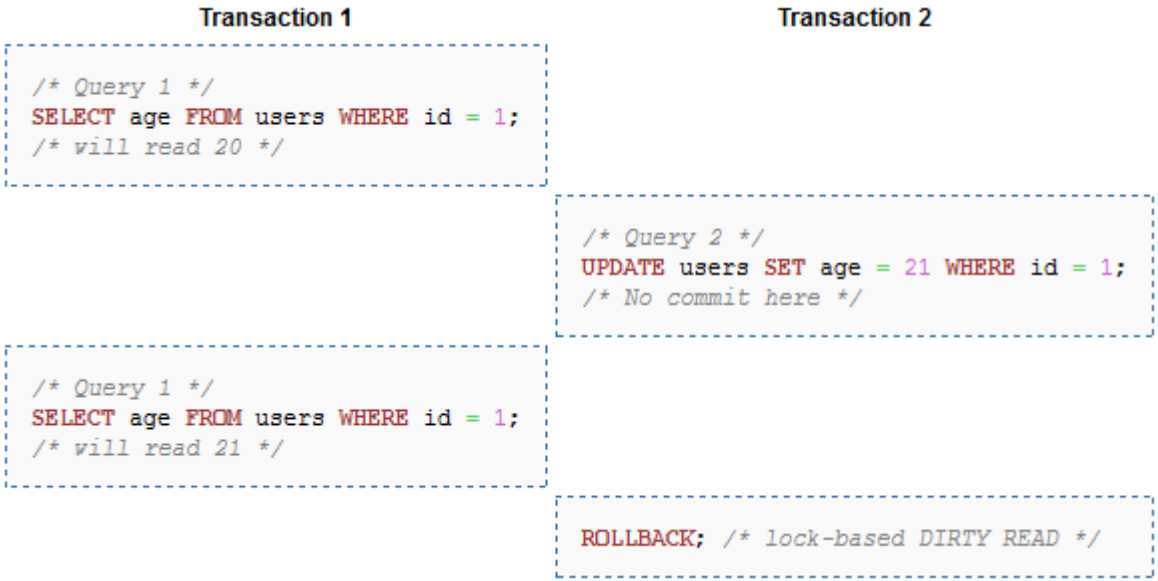
Spring :: Tx :: Уровни изоляции

LUXOFT TRAINING

- ISOLATION_REPEATABLE_READ
  - Чтение всех изменений своей транзакции, любые изменения, внесённые конкурирующими транзакциями после начала своей недоступны ;
  - Нечистые и неповторяемые чтения невозможны ;
  - Возможны фантомы ;
- ISOLATION_SERIALIZABLE
  - Идентичен ситуации при которой транзакции выполняются строго последовательно одна после другой ;
  - Запрещено чтение всех данных изменённых с начала транзакции, в том числе и своей транзакцией ;
  - Фантомы невозможны ;

Spring :: Tx :: Пример влияния уровня изоляции

- Dirty reads (Uncommitted Dependency)



* - Дополнительную информацию касательно неповторяемых чтений, фантомов и т.д. вы найдете в материалах слушателя.

Spring :: Tx :: Уровни изоляции

Spring поддерживает следующие уровни изоляции определенные в Enum `org.springframework.transaction.annotation.Isolation`:

- DEFAULT - Use the default isolation level of the underlying datastore. READ_COMMITTED - A constant indicating that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
- READ_UNCOMMITTED - A constant indicating that dirty reads, non-repeatable reads and phantom reads can occur.
- REPEATABLE_READ - A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
- SERIALIZABLE - A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented.

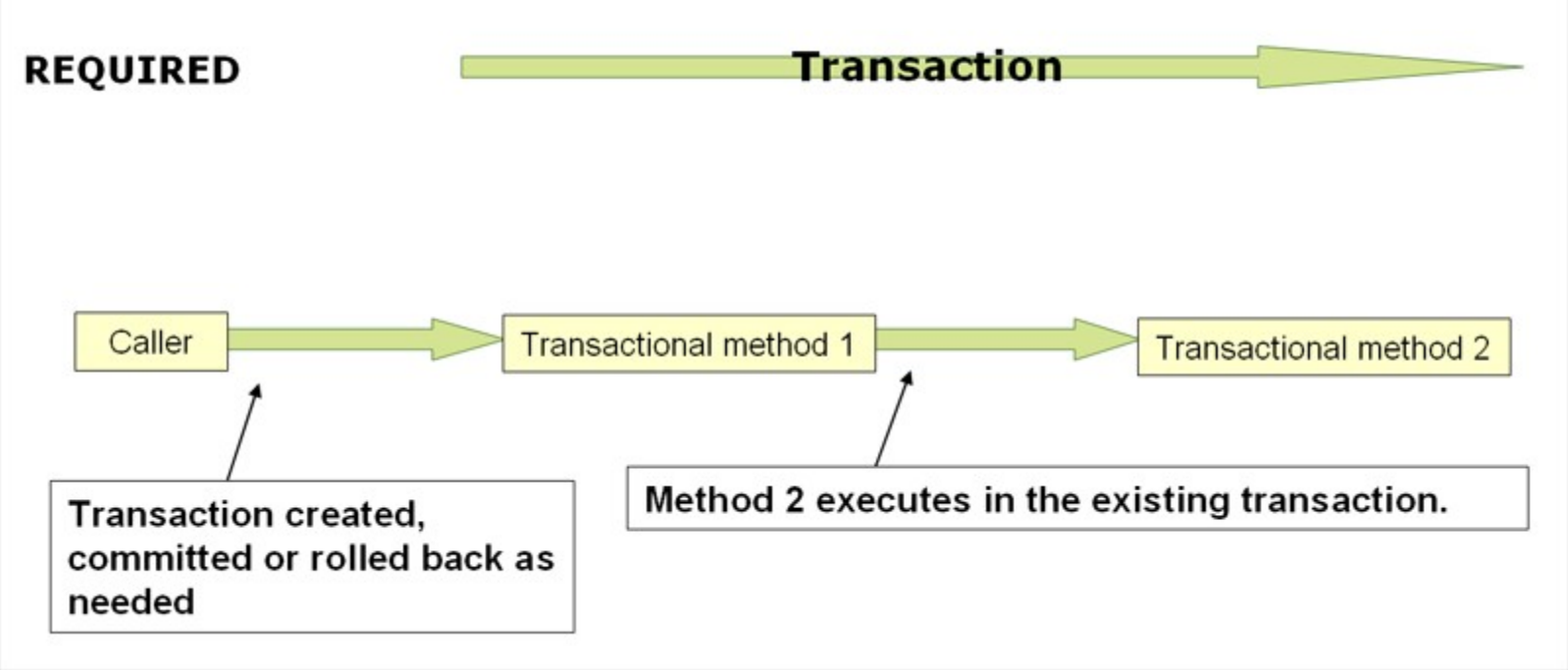
Spring :: Tx :: Propagation

Spring поддерживает следующие способы передачи транзакции от метода к методу в Enum `org.springframework.transaction.annotation.Propagation`:

- MANDATORY - Support a current transaction, throw an exception if none exists.
- NESTED - Execute within a nested transaction if a current transaction exists, behave like REQUIRED else.
- NEVER - Execute non-transactionally, throw an exception if a transaction exists.
- NOT_SUPPORTED - Execute non-transactionally, suspend the current transaction if one exists.
- REQUIRED - Support a current transaction, create a new one if none exists.
- REQUIRES_NEW - Create a new transaction, suspend the current transaction if one exists.
- SUPPORTS - Support a current transaction, execute non-transactionally if none exists.

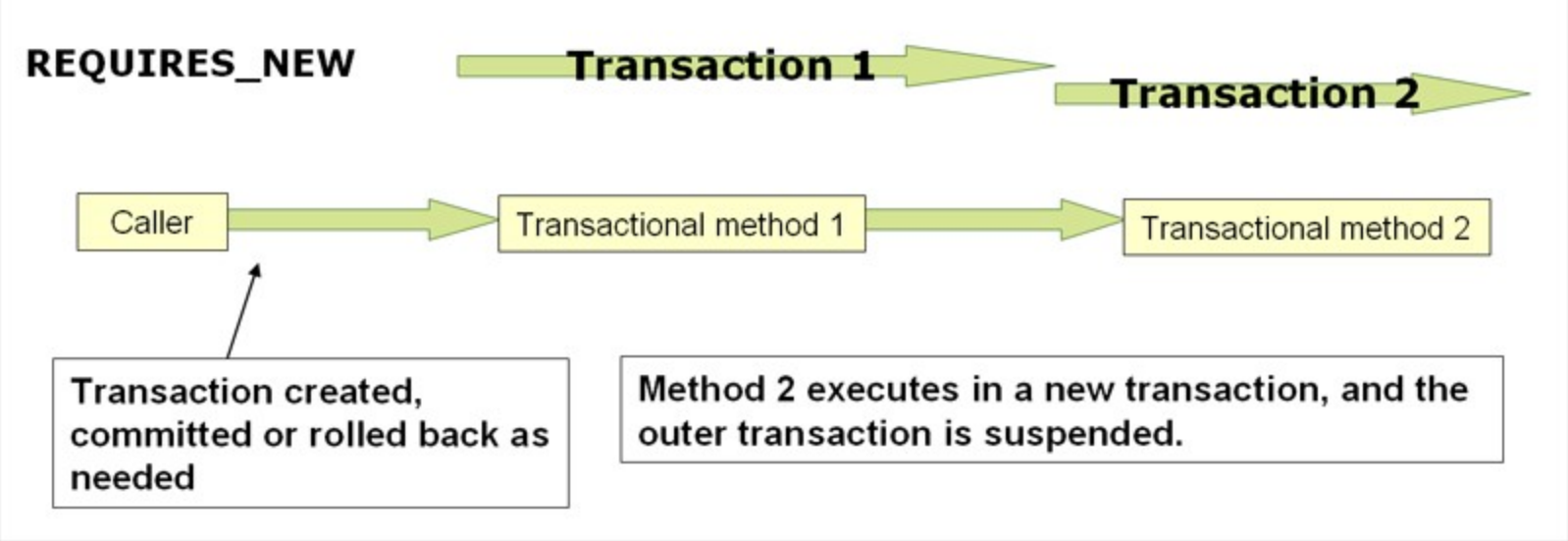
Spring :: Tx :: Propagation

- REQUIRED:



Spring :: Tx :: Propagation

- REQUIRES_NEW:



Spring :: Tx

- Timeout
  - задает промежуток времени, после которого транзакция автоматически откатится
- Read-only status
  - Транзакция со статусом read only не позволяет модифицировать данные;
  - Во многих случаях это может быть хорошей оптимизацией.

Spring :: Tx :: Примеры

Рассмотрим несколько примеров включения конкретных реализаций менеджеров транзакций в контекст приложения

Spring :: Tx :: Примеры

Рассмотрим несколько примеров включения конкретных реализаций менеджеров транзакций в контекст приложения

Spring :: Tx :: Примеры

DataSourceTransactionManager:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
   <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
   <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
   <property name="username" value="scott"/>
   <property name="password" value="tiger"/>
</bean>

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

Spring :: Tx :: Примеры

JtaTransactionManager в J2EE контейнере:

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/jpetstore"  />


<bean id="txManager"
   class="org.springframework.transaction.jta.JtaTransactionManager" />
```

Spring :: Tx :: Rollback

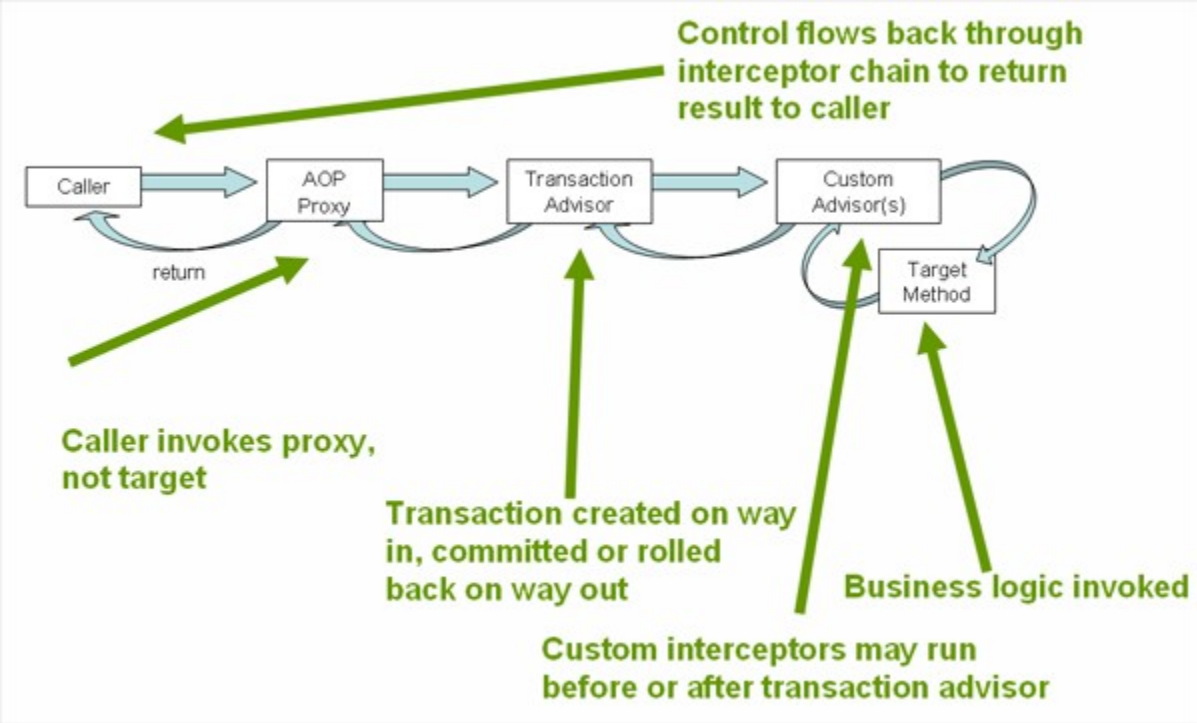
Правила для Rollback по умолчанию:

- Позволяют задать, какие исключения должны автоматически вызывать отмену транзакции ;
- По умолчанию, транзакции откатываются только для RuntimeException ;
- Для Exception исключений – нет ;

Есть возможность переопределить это поведение:

```
@Transactional(rollbackFor = IOException.class,
  noRollbackFor = RuntimeException.class)
public void doSomething() {
  …
}
```

Spring :: Tx + AOP



Spring :: Tx :: Rollback

LUXOFT TRAINING

@Transactional применяется к:

- Интерфейсам ;
- Классам ;
- Методам в интерфейсе ;
- public методам в классе ;

@Transactional лучше всегда применять к конкретным классам и их методам, а не к интерфейсам

Spring :: Tx :: Программное управление

Один из возможный вариантов: используя TransactionTemplate:

```
public class SimpleService implements Service {
    private TransactionTemplate transactionTemplate;
    public Object someServiceMethod() {
        return transactionTemplate.execute(
                new TransactionCallback() {
            public Object doInTransaction(
                    TransactionStatus status) {
                updateOperation1();
                return resultOfUpdateOperation2();
            }
        });
    }
}
```

Spring :: Tx :: Программное управление

LUXOFT TRAINING

В этом случае задать все свойства можно программно:

```
public void nonCallbackService() {
    transactionTemplate.setIsolationLevel(
            TransactionDefinition.ISOLATION_READ_COMMITTED);
    transactionTemplate.setReadOnly(false);
    transactionTemplate.setTimeout(100);
    transactionTemplate.setPropagationBehavior(
            TransactionDefinition.PROPAGATION_REQUIRED);
}
```

Spring :: Tx :: Программное управление

LUXOFT TRAINING

В этом случае задать все свойства можно программно:

- TransactionTemplate поддерживает callback-ориентированный подход;
- Реализовать TransactionCallback, метод doInTransaction();
- Передать его в метод execute() из TransactionTemplate;

```
public void callbackService() {
    transactionTemplate.execute(new TransactionCallback() {
        public Object doInTransaction(TransactionStatus status){
            updateOperation1();
            return resultOfUpdateOperation2();
        }
    });
}
```

Spring :: Tx :: Программное управление

LUXOFT TRAINING

- В большинстве случаев используется декларативное управление транзакциями
- Особенно, если в приложении много транзакционных операций
- Программное управление используется в случае:
  - В приложении мало транзакционных операций, в этом случае использование TransactionTemplate может быть приемлемо, но не очень желательно;
  - Необходимо явно задавать имя транзакции.

Spring :: Tx :: Пример конфигурирования

```
<aop:config>
    <aop:pointcut id="defaultServiceOperation” expression="execution(* x.y.service.*Service.*(..))"/>
    <aop:pointcut id="noTxServiceOperation« expression="execution(*
   x.y.service.ddl.DefaultDdlManager.*(..))"/>
    <aop:advisor pointcut-ref="defaultServiceOperation" advice-ref="defaultTxAdvice"/>
    <aop:advisor pointcut-ref="noTxServiceOperation" advice-ref="noTxAdvice"/>
</aop:config>
<bean id="fooService" class="x.y.service.DefaultFooService"/>
<bean id="anotherFooService" class="x.y.service.ddl.DefaultDdlManager"/>
<tx:advice id="defaultTxAdvice">
    <tx:attributes>
      <tx:method name="get*" read-only="true"/>
      <tx:method name="*"/>
    </tx:attributes>
</tx:advice>
<tx:advice id="noTxAdvice">
    <tx:attributes>
      <tx:method name="*" propagation="NEVER"/>
    </tx:attributes>
</tx:advice>
```

Упражнения

LUXOFT TRAINING

№ 8 : «Управление транзакциями в Spring»

- 30 мин – самостоятельная работа;

LUXOFT TRAINING

Вопросы!?