SPRING FRAMEWORK 3.0

Dmitry Noskov

Data Access

Spring Data Access

- DAO support
- transaction management
- JDBC

- not included
  - ORM
  - marshalling XML

DAO support

jdbc namespace

Spring data access exceptions

JDBC namespace

```
<beans  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jdbc="http://www.springframework.org/schema/jdbc"
        xsi:schemaLocation="
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/jdbc
            http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd">



</beans>
```

Embedded database

```
<jdbc:embedded-database id="dataSource" type="HSQL|H2|Derby">
  <jdbc:script location="classpath:db-schema.sql"/>
  <jdbc:script location="classpath:test-data.sql"/>
</jdbc:embedded-database>


<bean class="org.exhanger.repository.jdbc.AccountJdbcRepository">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

Populate other DataSource

```
<bean id="dataSource" class="oracle.jdbc.pool.OracleDataSource">
  <property name="URL" value="${jdbc.url}"/>
  <property name="user" value="${jdbc.user}"/>
  <property name="password" value="${jdbc.password}"/>
</bean>

<jdbc:initialize-database data-source="dataSource" >
  <jdbc:script location="classpath:db-schema.sql"/>
  <jdbc:script location="classpath:test-data.sql"/>
</jdbc:initialize-database>
```

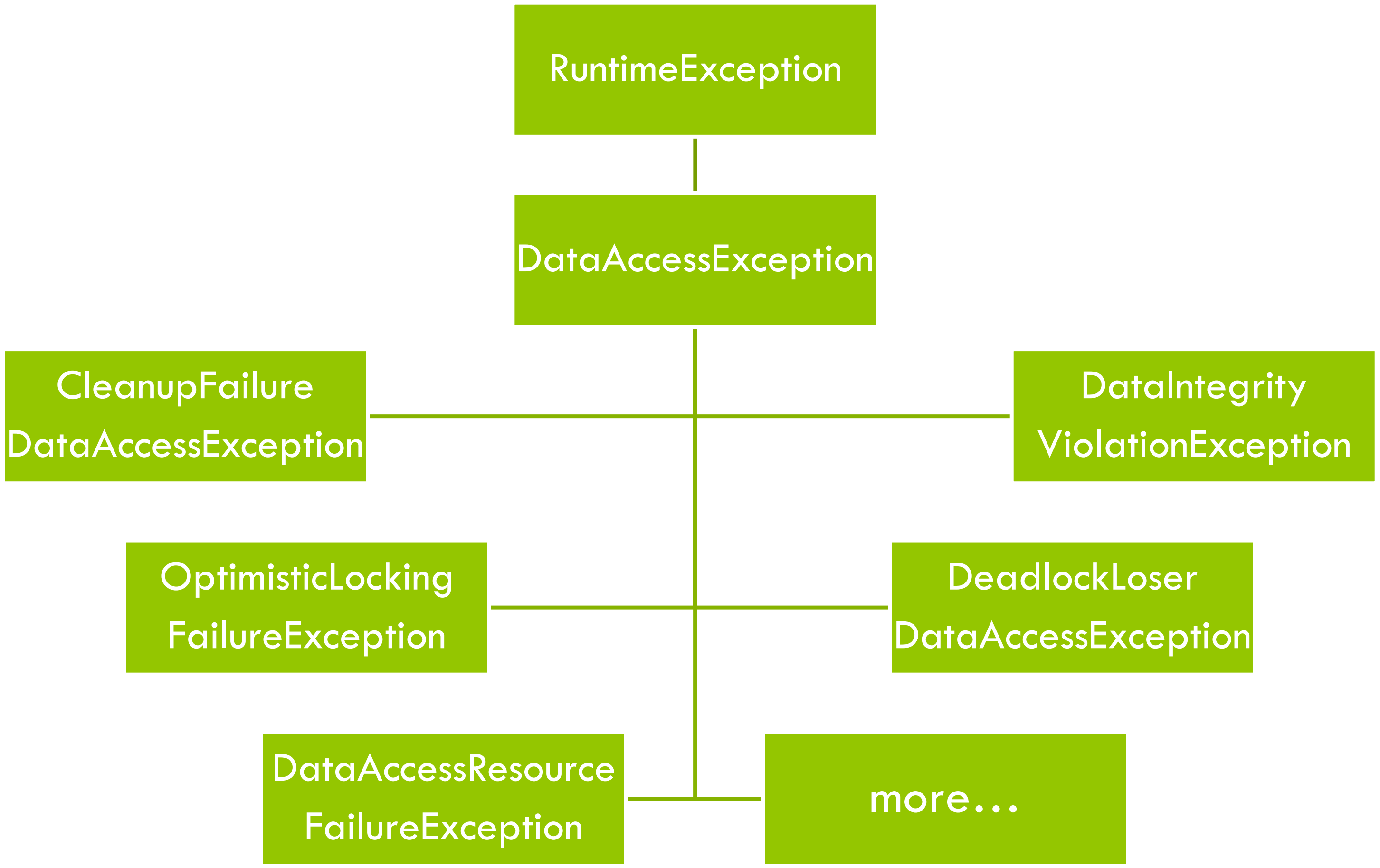
Problem of handling exceptions

- exceptions
  - checked
  - unchecked
- SQLException
  - technology-specific
  - depends on DBMS
  - couple the DAO and API

Spring DataAccessException's

- unchecked
- hide data access technology
  - JDBC
  - Hibernate
  - etc.
- hide DBMS error codes

DataAccessException hierarchy


RuntimeException
DataAccessException
CleanupFailure DataAccessException
DataIntegrity ViolationException
OptimisticLocking FailureException
DeadlockLoser DataAccessException
DataAccessResource FailureException
more…

@Repository

```
@Repository
public class AccountJdbcRepository implements AccountRepository {

  @Autowired
  private DataSource dataSource;

  private JdbcTemplate jdbcTemplate;

  @Autowired
  public AccountJdbcRepository(DataSource dataSource) {
    super
    this.jdbcTemplate = new JdbcTemplate(dataSource);
  }
}
```

Transaction management

Overview

Transaction models

Spring PlatformTransactionManager’s

Unit of work

```
public Long process(Account account, Order o) {
  String owner = o.getOwner();
  String from = o.getFrom();
  String to = o.getTo();

  accountService.reduceBalance(account, to, o.getValue() * o.getRate());
  accountService.increaseBalance(account, from, o.getValue());

  accountService.reduceBalance(owner, from, o.getValue());
  accountService.increaseBalance(owner, to, o.getValue() * o.getRate());

  orderService.completeOrder(order.getId());

  return auditService.createRecord(account, o);
}
```

ACID principles

- atomicity
  - "all or nothing" rule for unit of work
- consistency
  - only consistent data will be saved
- isolation
  - isolating transactions for each other
- durability
  - commited changes are permanent

Transaction models

- local
  - work across single transactional resource
  - resource-specific
  - easier to use
- global
  - work across multiple transactional resources

Local transactions

- can be managed by DBMS, etc.
- depends on connection

Classic database transaction

```
public class AccountJdbcRepository implements AccountRepository {
  private DataSource dataSource;

  public void modifyAccount(Account account) {
    String sql = "...";
    Connection conn = null;

    try {
      conn = dataSource.getConnection();
      conn.setAutoCommit(false);
      PreparedStatement ps = conn.prepareStatement(sql);
      s.executeUpdate();
      conn.commit();
    } catch (SQLException e) {
      conn.rollback();
    }
}
```

Problems with local transactions

- connection management code is error-prone
- transaction demarcation belongs at the service layer
  - multiple data access methods may be called within a transaction
  - connection must be managed at a higher level

Spring transaction management

- declarative
  - XML
  - annotations
- programmatic

Namespace

```
<beans 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">

</beans>
```

Basic configuration

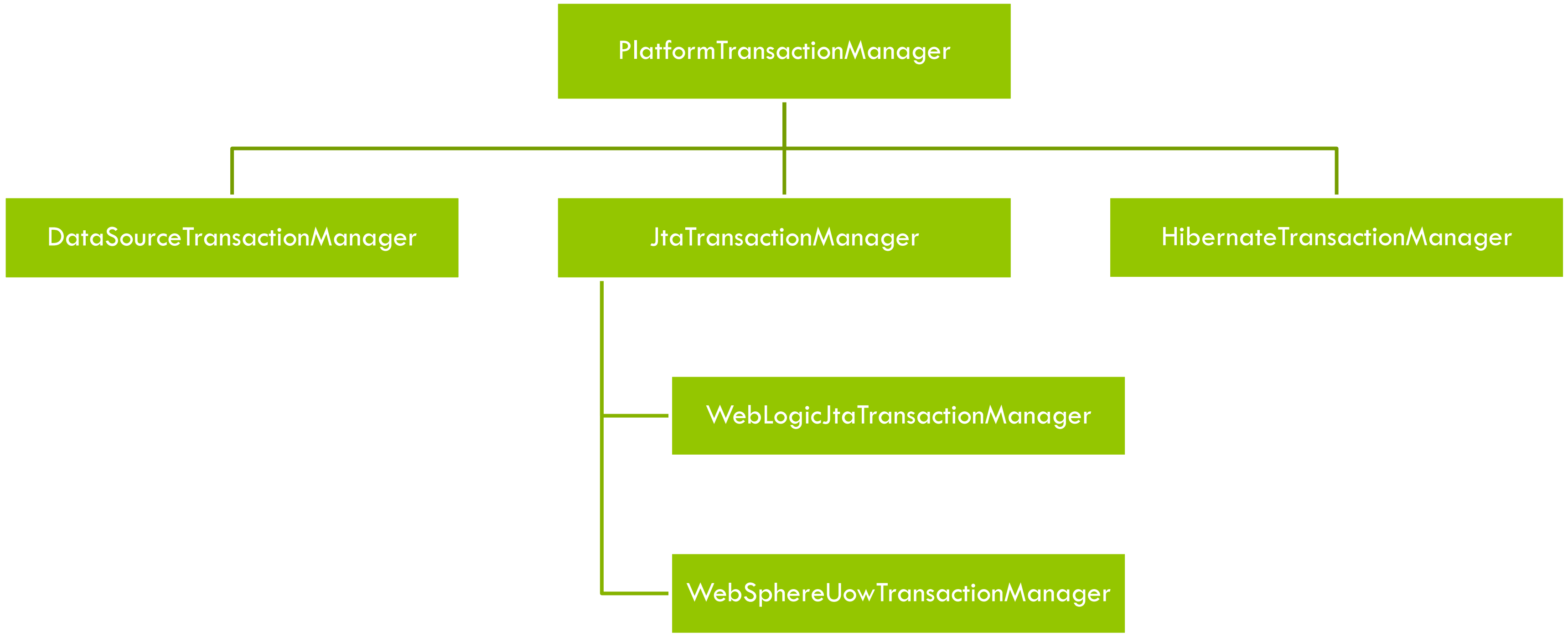
```
<beans>

  <tx:annotation-driven transaction-manager="transactionManager"
                        mode="proxy|aspectj"
                        order="0" <!-- Ordered.LOWEST_PRECEDENCE -->
                        proxy-target-class="false"/>

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

  <!-- <tx:jta-transaction-manager/> -->
</beans>
```

PlatformTransactionManager

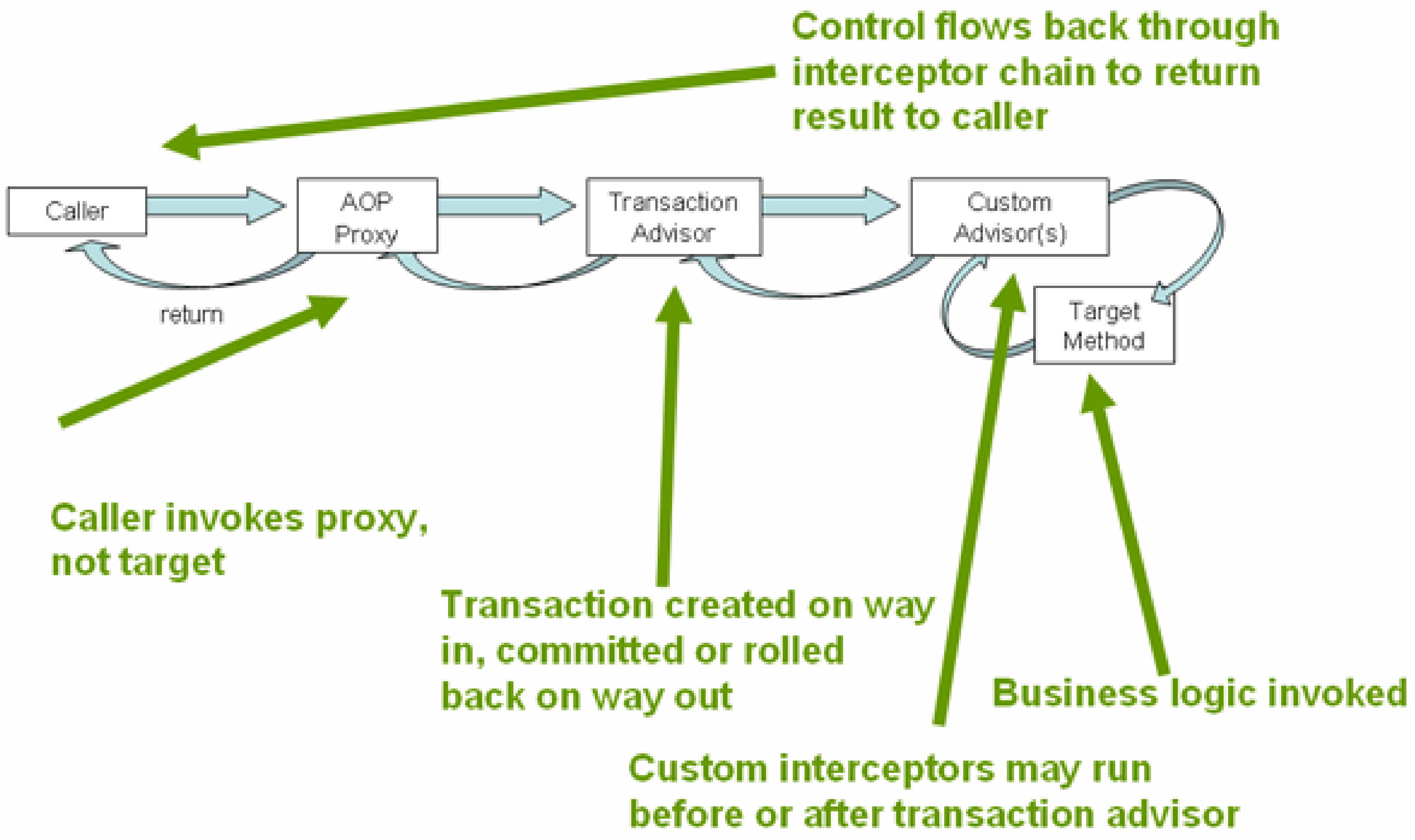

PlatformTransactionManager
DataSourceTransactionManager
JtaTransactionManager
HibernateTransactionManager
WebLogicJtaTransactionManager
WebSphereUowTransactionManager

@Transactional

```
@Transactional
public class AccountServiceImpl implements AccountService {

  @Transactional
  public Account getAccount(Long accountId) {
    return repository.getAccount(accountId);
  }

  @Transactional
  public Long createAccount(Account account) {
    return repository.createAccount(account);
  }
}
```


Control flows back through interceptor chain to return result to caller
Caller
AOP Proxy
Transaction Advisor
Custom Advisor(s)
Target Method
return
Caller invokes proxy, not target
Transaction created on way in, committed or rolled back on way out
Business logic invoked
Custom interceptors may run before or after transaction advisor

How it works

Transactional attributes

- propagation
- isolation
- rollback rules
- read-only
- timeout

Transaction propagation(1)

- REQUIRED
  - support a current transaction, create a new one if none exists

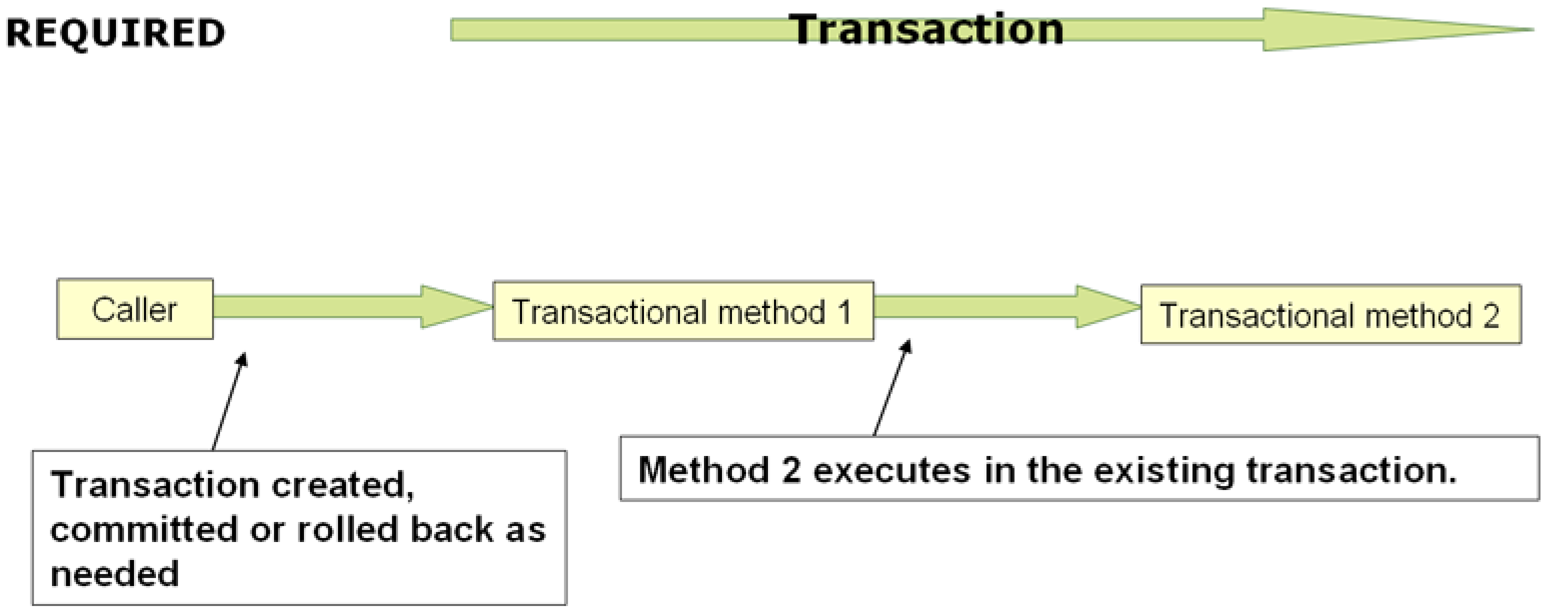
```
@Transactional(propagation = Propagation.REQUIRED)
public Long process(Account account, Order order){}
```

- REQUIRED_NEW
  - create a new transaction, suspend the current if one exists

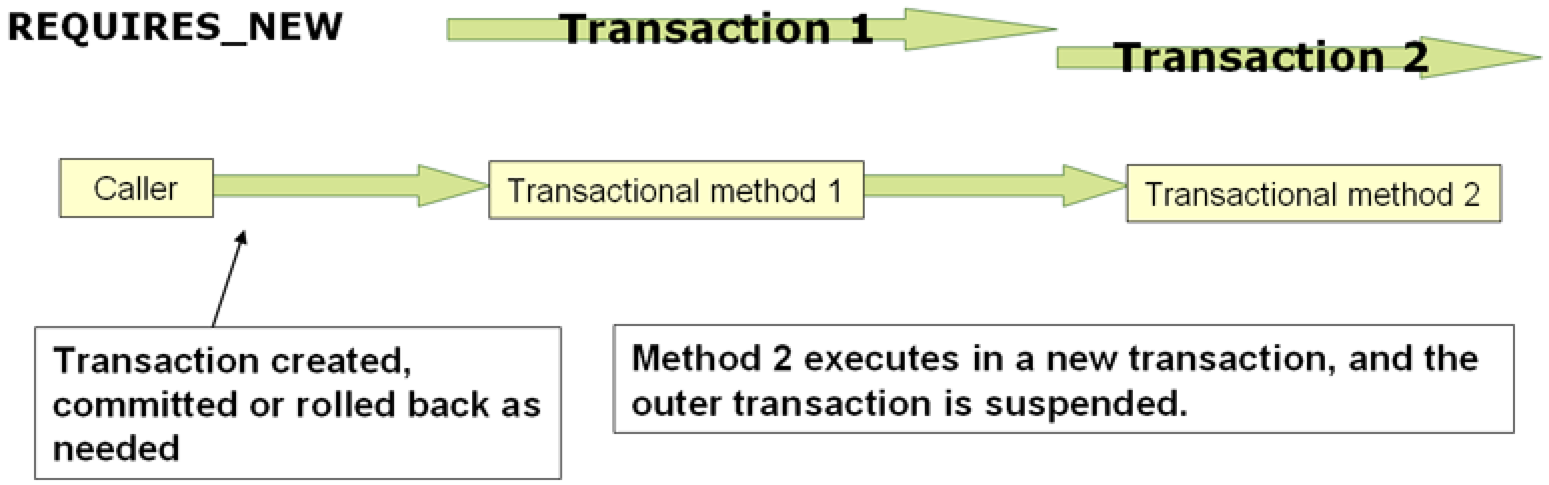
```
@Transactional(propagation = Propagation.REQUIRED_NEW)
public Long process(Account account, Order order){}
```

- NESTED
  - single physical transaction with multiple savepoints

REQUIRED


REQUIRED
Transaction
Caller
Transactional method 1
Transactional method 2
Transaction created, committed or rolled back as needed
Method 2 executes in the existing transaction.

REQUIRED_NEW



Transaction created, committed or rolled back as needed

Method 2 executes in a new transaction, and the outer transaction is suspended.

Transaction propagation(2)

- SUPPORTS
  - support a current transaction, or execute non-transactionally
- MANDATORY
  - support a current transaction, throw an exception if none exists
- NOT_SUPPORTED
  - execute non-transactionally, suspend the current transaction
- NEVER
  - execute non-transactionally, throw an exception if a transaction exists

Isolation levels

- levels
  - READ_UNCOMMITED
  - READ_COMMITED
  - REPEATABLE_READ
  - SERIALIZABLE
- DBMS
  - have differences in implementation
  - may not support something

READ_UNCOMMITED

- lowest isolation level
- dirty reads are allowed
- example

```
@Transactional(isolation = Isolation.READ_UNCOMMITTED)
public Order getOrder(Long orderId) {
  return repository.getOrder(orderId);
}
```

READ_COMMITED

- default strategy for most DBMSs
- only commited data can be accessed
- example

```
@Transactional(isolation = Isolation.READ_COMMITTED)
public Order getOrder(Long orderId) {
  return repository.getOrder(orderId);
}
```

Highest isolation levels

- REPEATABLE_READ
  - prevents non-repeatable reads
- SERIALIZABLE
  - prevents phantom reads

Rollback rules

```
@Service
public class AccountServiceImpl implements AccountService {

  @Transactional(rollbackFor = CheckedException.class)
  public Long createAccount(Account account) {
    return repository.createAccount(account);
  }

  @Transactional(noRollbackFor = SendNotificationException.class)
  public void modifyAccount(Account account) {
    repository.modifyAccount(account);
  }
}
```

Read-only transactions

- optimize resource for read-only access
- example

```
@Transactional(readOnly = true)
public Account getAccount(Long accountId) {
  return repository.getAccount(accountId);
}
```

Trsnsaction timeout

- example

```
@Transactional(timeout = 60)
public List<Order> getActiveOrders(String from, String to) {
}
```

Multiple transaction managers

```
<tx:annotation-driven/>
<bean id="transactionManager1" class="...">
  <qualifier value="txManager1"/>
</bean>
<bean id="transactionManager2" class="...">
  <qualifier value="txManager2"/>
</bean>

@Transactional(value = "txManager1")
public BigDecimal method1 (Order order) {}

@Transactional(value = "txManager2")
public Account merge(Account account) {}
```

Custom shortcut annotations

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("txManager1")
public @interface OrderTx {}

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("txManager2")
public @interface AccountTx {}

Using:
@OrderTx
public BigDecimal method1 (Order order) {}

@AccountTx
public Account merge(Account account) {}
```

Programmatic transactions

- TransactionTemplate
- PlatformTransactionManager

TransactionTemplate(1)

```
public class AccountServiceImpl implements AccountService {
  private final TransactionTemplate template;

  private AccountServiceImpl(PlatformTransactionManager manager) {
    template = new TransactionTemplate(manager);
  }

  public Object method1(final Object obj) {
    return template.execute(new TransactionCallback<Object>() {
      public Object doInTransaction(TransactionStatus status) {
        return operation(obj);
      }
  });
}
```

TransactionTemplate(2)

```
public class AccountServiceImpl implements AccountService {
  private final TransactionTemplate template;

  private AccountServiceImpl(PlatformTransactionManager manager) {
    template = new TransactionTemplate(manager);
  }

  public void method2(final Object obj) {
   template.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus st){
      operation(obj);
    }
   });
  }
```

TransactionTemplate(3)

```
public class AccountServiceImpl implements AccountService {
  private final TransactionTemplate template;

  public void method3(final Object obj) {
   template.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus s){
      try {
        operation(obj);
      } catch (RuntimeException e) {
        status.setRollbackOnly();
      }
    }
  );
}
```

PlatformTransactionManager

```java
public interface PlatformTransactionManager {
  /** Return a currently active transaction or create a new one,
     according to the specified propagation behavior. */
  TransactionStatus getTransaction(TransactionDefinition definition);

  /** Commit the given transaction, with regard to its status. If
  the transaction has been marked rollback-only programmatically,
  perform a rollback. */
  void commit(TransactionStatus status);

  /** Perform a rollback of the given transaction. */
  void rollback(TransactionStatus status);
}
```

XML-based

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="*" propagation="REQUIRED"
               isolation="READ_COMMITTED"
               rollback-for="java.lang.Throwable" />
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="txOperation"
       expression="execution(* org.training.api..*ServiceImpl.*(..))"/>
  <aop:advisor pointcut-ref="txOperation"
               advice-ref="txAdvice"
               order="" />
</aop:config>
```

Spring JDBC

Strong JDBC

```
public List<Account> getAccountsByType(Long type) {
  List<Account> accounts = new ArrayList<Account>();
  Connection connection = null;
  String sql = "select account_id, account_owner from account";
  try {
    connection = dataSource.getConnection();
    PreparedStatement ps = connection.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
    while (rs.next()) {
      accounts.add(new Account(rs.getLong(1), rs.getString(2)));
    }
  } catch (SQLException e) {/*handle exception*/}
  finally {
    try { connection.close();} catch (SQLException e) {/*handle*/}
  }
  return accounts;
}
```

Target code

```
public List<Account> getAccountsByType(Long type) {
  List<Account> accounts = new ArrayList<Account>();
  Connection connection = null;
  String sql = "select account_id, account_owner from account";
  try {
    connection = dataSource.getConnection();
    PreparedStatement ps = connection.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
    while (rs.next()) {
      accounts.add(new Account(rs.getLong(1), rs.getString(2)));
    }
  } catch (SQLException e) {/*handle exception*/}
  finally {
    try { connection.close();} catch (SQLException e) {/*handle*/}
  }
  return accounts;
}
```

Who does what

Action	Spring	You
Define connection parameters		X
Connection management	X	
SQL		X
Statement management	X	
ResultSet management	X	
Row Data Retrieval		X
Exception handling	X	
Handle transactions	X	

Spring JDBC abstractions

JdbcTemplate

NamedParameterJdbcTemplate

SimpleJdbcTemplate

SimpleJdbcInsert and SimpleJdbcCall

JdbcTemplate

Creating the JdbcTemplate

```
@Repository
public class AccountJdbcRepository implements AccountRepository {

  private JdbcTemplate jdbcTemplate;

  public AccountJdbcRepository(DataSource dataSource) {
    super();
    jdbcTemplate = new JdbcTemplate(dataSource);
  }
}
```

Simple types

```
int intCount =
    jdbcTemplate.queryForInt("select count(*) from accounts");

int longCount =
    jdbcTemplate.queryForLong (
        "select count(*) from accounts
         where account_owner like ?", "name");

String name =
    jdbcTemplate.queryForObject(
        "select account_owner from accounts
         where account_id = ?", new Object[]{10}, String.class);
```

Generic maps

```
public Map<String, Object> getAccount(Long accountId) {
  String sql = "select * from accounts where account_id = ?";
  return jdbcTemplate.queryForMap(sql, accountId);
}


public List<Map<String, Object>> getAccounts() {
  return jdbcTemplate.queryForList("select * from accounts");
}
```

Domain object(1)

```
Account account =
  jdbcTemplate.queryForObject(
    "select account_id, account_owner from accounts
     where account_id = ?",
        new Object[]{11},
        new RowMapper<Account>(){
          public Account mapRow(ResultSet rs, int rowNum){
            Account account = new Account();
            account.setId(rs.getLong("account_id"));
            account.setOwner (rs.getString("account_owner"));
            return account;
           }
        });
```

Domain object(2)

```
List<Account> accounts =
  jdbcTemplate.query(
    "select account_id, account_name from m_accounts
     where account_id = ?",
     new Object[]{11},
     new RowMapper<Account>(){
       public Account mapRow(ResultSet rs, int rowNum) {
         Account account = new Account();
         account.setId(rs.getLong("account_id"));
         account.setName(rs.getString("account_name"));

         return account;
       }
     });
```

Callback handler

```
public void writeAccountsToFile(File f) {
  jdbcTemplate.query("select * from account", new FileWriter(f));
}

public class FileWriter implements RowCallbackHandler {
  public FileWriter(File file) {
   super();
  }

  public void processRow(ResultSet rs) throws SQLException {
    //write result set to file
  }
}
```

ResultSet extractor

```
public Account getAccount(long accountId) {
  String sql = "select * from accounts where account_id = ?";
  return jdbcTemplate.query(sql,
                            new Object[]{accountId},
                            new Extractor());
}

public class Extractor implements ResultSetExtractor<Account> {
  public Account extractData(ResultSet rs) throws SQLException {
    //write result set to single domain object
    return account;
  }
}
```

Callback interfaces

- RowMapper
  - **each row** to domain object
- RowCallbackHandler
  - **another** output stream
- ResultSetExtructor
  - **multiple rows** to domain object

Insert / Update / Delete

```
jdbcTemplate.update(
    "insert into accounts(account_id, account_owner) values(?, ?)",
    17, "Account Name");

jdbcTemplate.update(
    "update accounts set account_owner = ? where account_id = ?",
    "New Account Name", 18);

jdbcTemplate.update(
    "delete from accounts where account_id = ?", 19);
```

NamedParameterJdbcTemplate

Creating the NamedJdbcTemplate

```
@Repository
public class AccountJdbcRepository implements AccountRepository {

  private NamedParameterJdbcTemplate namedJdbcTemplate;

  public AccountJdbcRepository(DataSource dataSource) {
    super();
    namedJdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
  }
}
```

Naming parameters(1)

```
public Long getCountByName(String name) {
  Map<String, ?> params = Collections.singletonMap("name", name);
  String sql =
      "select count(*) from accounts where owner_name like :name";

  return namedJdbcTemplate.queryForLong(sql, params);
}
```

Naming parameters(2)

- map parameter source

```
SqlParameterSource params = new MapSqlParameterSource("name","%ac%");
namedJdbcTemplate.queryForInt(
    "select count(*) from accounts where owner_name like :name",
    params);
```

- bean parameter source

```
Account acc = …;
SqlParameterSource params = new BeanPropertySqlParameterSource(acc);

namedJdbcTemplate.queryForInt(
  "select count(*) from m_accounts
   where account_name like :name and account_type = :type",
   params);
```

Batch operations

Batch by jdbcTemplate

```
public void batchUpdate(final List<Account> accounts) {
  jdbcTemplate.batchUpdate(
    "update accounts set account_name = ?, account_type = ?
     where account_id = ?",
    new BatchPreparedStatementSetter() {
      public void setValues(PreparedStatement ps, int i) {
        ps.setString(1, accounts.get(i).getName());
        ps.setString(2, accounts.get(i).getType());
        ps.setLong(3, accounts.get(i).getId());
      }
      public int getBatchSize() {
        return accounts.size();
      }
    });
}
```

Batch by namedJdbcTemplate

```
public void batchUpdate2(final List<Account> accounts) {
  SqlParameterSource[] batch =
      SqlParameterSourceUtils.createBatch(accounts.toArray());

  namedJdbcTemplate.batchUpdate(
    "update m_accounts set account_name = :name, account_type = :type
     where account_id = :id",
    batch);
}
```

SimpleJdbcInsert

```
simpleJdbcInsert =
  new SimpleJdbcInsert(dataSource).withTableName("accounts");

Map<String, Object> params = new HashMap<String, Object>();
params.put("account_id", 12);
params.put("account_name", "name");
params.put("account_type", "type");

simpleJdbcInsert.execute(params);
```

Summary

Spring Data Access

- promote layered architecture principles
- loose coupling between API and DAO layers
- supports top data access technologies
- flexible transaction management

Benefits of declarative transactions

- consistent model across different transaction APIs
  - JTA
  - JDBC
  - JPA
  - Hibernate
- supports for declarative transaction management
- simple API for programmatic transactions

Declarative transactions

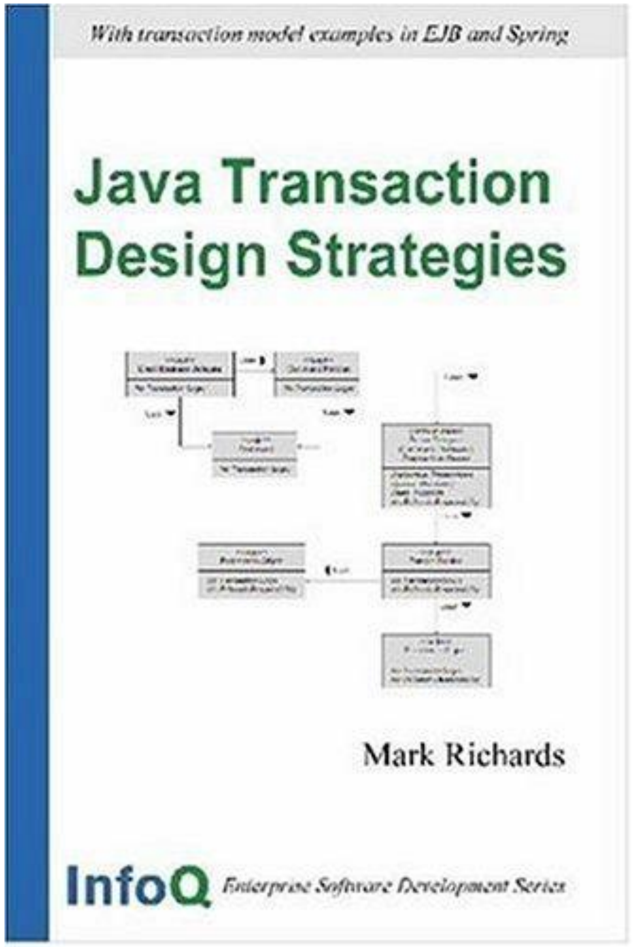
- any environment
- any classes
- rollback rules
- customize transactional behavior

- not support transaction contexts across remote calls

Links

- Spring reference
  - http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/spring-data-tier.html
- wiki: ACID
  - http://en.wikipedia.org/wiki/ACID
- wiki: isolation
  - http://en.wikipedia.org/wiki/Isolation_(database_systems)
- JTA
  - http://en.wikipedia.org/wiki/Java_Transaction_API

Books


With transaction model examples in EJB and Spring
Java Transaction Design Strategies
Mark Richards
InfoQ Enterprise Software Development Series

Questions

The end

noskov.d@Gmail.com

my Linked in profile

http://www.linkedin.com/in/noskovd

slideshare

http://www.slideshare.net/analizator/presentations