IT-Academy

Интеграция Spring Hibernate

Автор: Юлий Слабко

Integrating Hibernate with Spring

IT-Academy

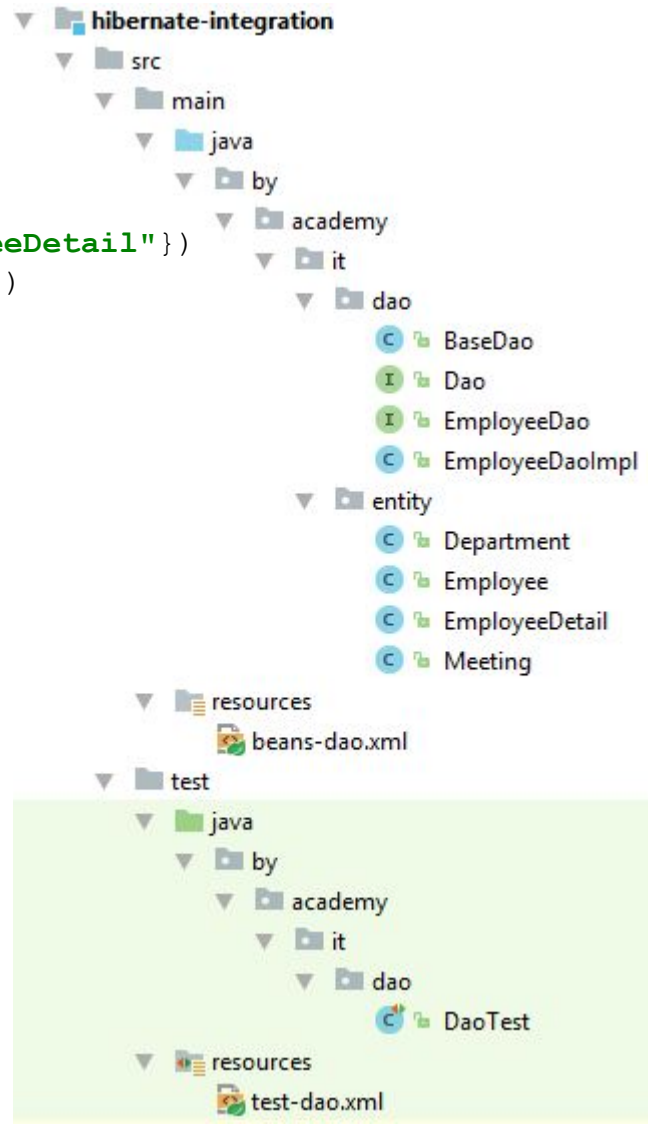
```
<spring.version>4.3.12.RELEASE</spring.version>
<hibernate.version>5.2.11.Final</hibernate.version>
<jdbc.version>5.1.34</jdbc.version>
<aspect.version>1.8.4</aspect.version>
<compiler.version>3.7.0</compiler.version>
<junit.version>4.12</junit.version>
```

```
<dependency>
    <groupId>org.springframework</ groupId>
    <artifactId>spring-context</ artifactId>
    <version>${spring.version}</ version>
</dependency>
<dependency>
    <groupId>org.springframework</ groupId>
    <artifactId>spring-orm</ artifactId>
    <version>${spring.version}</ version>
</dependency>
<dependency>
    <groupId>org.springframework</ groupId>
    <artifactId>spring-aop</ artifactId>
    <version>${spring.version}</ version>
</dependency>
<dependency>
    <groupId>org.springframework</ groupId>
    <artifactId>spring-test</ artifactId>
    <version>${spring.version}</ version>
    <scope>test</ scope>
</dependency>
<dependency>
    <groupId>org.aspectj</ groupId>
    <artifactId>aspectjweaver</ artifactId>
    <version>${aspect.version}</ version>
</dependency>
```

```
<dependency>
    <groupId>org.hibernate</ groupId>

<artifactId>hibernate-entitymanager</ artifactId>
    <version>${hibernate.version}</ version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</ artifactId>
    <version>${jdbc.version}</ version>
</dependency>
<dependency>
    <groupId>org.apache.commons</ groupId>
    <artifactId>commons-dbcp2</ artifactId>
    <version>2.1.1</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</ groupId>
    <artifactId>lombok</artifactId>
    <version>1.16.6</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</ version>
    <scope>test</ scope>
</dependency>
```

Integrating Hibernate with Spring

```java
@Data @NoArgsConstructor @AllArgsConstructor
@EqualsAndHashCode(exclude = {"department", "meetings", "employeeDetail"})
@ToString(exclude = {"department", "meetings", "employeeDetail"})
@Entity
public class Employee {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;
    private String lastName;
    @CreationTimestamp
    private LocalDateTime date;

    @OneToOne(mappedBy = "employee",
          cascade = CascadeType.PERSIST)
    private EmployeeDetail employeeDetail;

    @ManyToOne
    @JoinColumn(name = "DEPARTMENT_ID")
    private Department department;

    @ManyToMany(cascade = CascadeType.ALL)
    private List<Meeting> meetings = new ArrayList<>();
}
```



Integrating Hibernate with Spring

IT-Academy

```
@Data @NoArgsConstructor @AllArgsConstructor
@EqualsAndHashCode(exclude = {"employees"})
@ToString(exclude = {"employees"})
@Entity
public class EmployeeDetail implements Serializable {
    @Id @GenericGenerator(name = "one-one",
            strategy = "foreign",
            parameters = @Parameter(name = "property", value = "employee"))
    @GeneratedValue(generator = "one-one")
    private Long id;
    private String street;
    private String city;
    private String state;
    private String country;
    @OneToOne(fetch = FetchType.EAGER)
    @PrimaryKeyJoinColumn
    private Employee employee;
}
```

Integrating Hibernate with Spring

IT-Academy

```
@Data @NoArgsConstructor
@EqualsAndHashCode(exclude = {"employees"})
@ToString(exclude = {"employees"})
@Entity
public class Department implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long departmentId;
    private String departmentName;

    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
    private Set<Employee> employees = new HashSet<>(0);

    public Department(String name) {
        this.departmentName = name;
    }
}
```

Integrating Hibernate with Spring

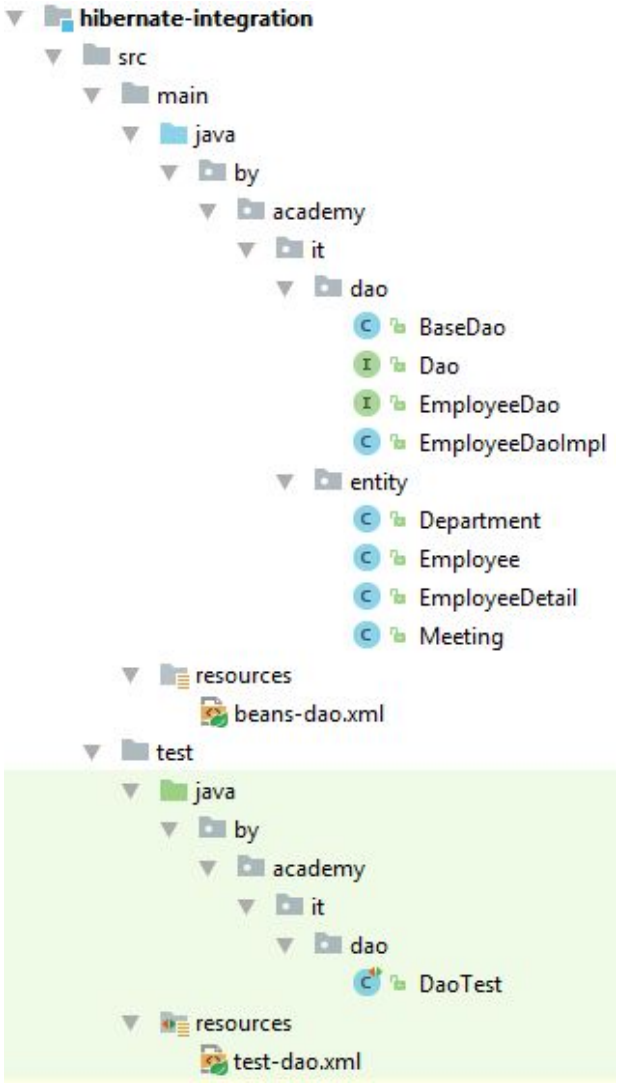
IT-Academy

```
@Data @NoArgsConstructor @AllArgsConstructor
@EqualsAndHashCode(exclude = {"employees"})
@ToString(exclude = {"employees"})
@Entity
public class Meeting implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long meetingId;
    private String subject;
    @CreationTimestamp
    private LocalDateTime startDate;
    @ManyToMany(mappedBy = "meetings", cascade = CascadeType.ALL)
    private List<Employee> employees = new ArrayList<>();

    public Meeting(String subject) {
        this.subject = subject;
    }
}
```

Integrating Hibernate with Spring

IT-Academy

```java
package by.academy.it.dao;

import java.io.Serializable;

public interface Dao<T> {
    T add(T t);

    T update(T t);

    T get(Serializable id);

    void delete(Serializable id);
}
```



Integrating Hibernate with Spring

IT-Academy

```
@Repository
public class BaseDao<T> implements Dao<T> {
    Class<T> clazz;
    ThreadLocal<EntityManager> em
     = new ThreadLocal<>();
    @Autowired
    private EntityManagerFactory factory;

    @Override
    public T add(T t) {
        begin();
        getEm().persist(t);
        commit();
        return t;
    }
    @Override
    public T get(Serializable id) {
        return getEm().find(clazz, id);
    }
    @Override
    public T update(T t) {
        begin();
        getEm().merge(t);
        commit();
        return t;
    }
```

```
    @Override
    public void delete(Serializable id) {
        begin();
        T t = getEm().find(clazz, id);
        getEm().remove(t);
        commit();
    }

    public EntityManager getEm() {
        if (em.get() == null) {
em.set(factory.createEntityManager());
        }
        return em.get();
    }
    public void begin() {
        getEm().getTransaction().begin();
    }
    public void commit() {
        getEm().getTransaction().commit();
    }
```

Integrating Hibernate with Spring

IT-Academy

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="by.academy.it.dao"/>

    <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
        <property name="url"
     value="jdbc:mysql://localhost:3306/spring_hibernate_integration?createDatabaseIfNotExist=true"/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="username" value="root"/>
        <property name="password" value="yuli"/>
        <property name="initialSize" value="5"/>
        <property name="maxTotal" value="20"/>
    </bean>
```

Integrating Hibernate with Spring

IT-Academy

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="persistenceUnitName" value="jpa-unit"/>
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
        </property>
        <property name="packagesToScan">
            <list>
                <value>by.academy.it.entity</ value>
            </ list>
        </property>
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQL55Dialect</ prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">false</prop>
                <prop key="hibernate.hbm2ddl.auto">create</prop>
            </ props>
        </property>
    </bean>
</beans>
```

Integrating Hibernate with Spring

IT-Academy

```java
package by.academy.it.dao;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import by.academy.it.entity.Employee;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:test-dao.xml")
public class DaoTest {

    @Autowired
    private EmployeeDao employeeDao;

    @Test
    public void saveTest() {
        Employee e = new Employee();
        e.setFirstName("Yuli");
        e.setLastName("Slabko");
        e = employeeDao.add(e);

        Assert.assertEquals("Yuli", employeeDao.get(e.getId()).getFirstName());
    }
}
```

Integrating Hibernate with Spring

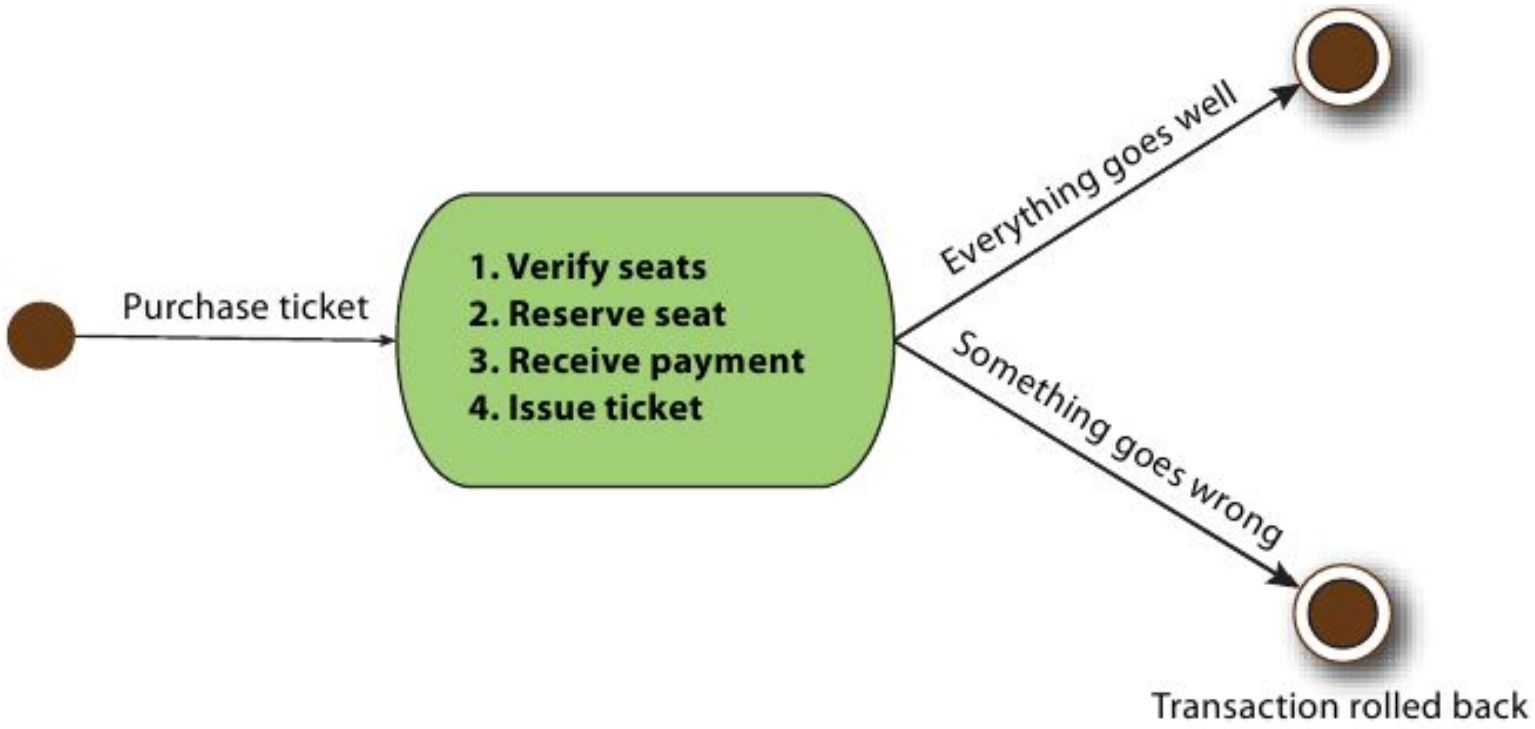
IT-Academy

Свойство конфигурации пула	Назначение
`initialSize`	Начальное число соединений при создании пула
`maxActive`	Максимально допустимое число одновременно открытых активных соединений. Значение 0 соответствует неограниченному числу соединений
`maxIdle`	Максимально допустимое число простаивающих соединений, которые не будут закрыты. Значение 0 соответствует неограниченному числу соединений
`maxOpenPreparedStatements`	Максимально допустимое количество скомпилированных запросов, которые могут быть помещены в пул запросов одновременно. Значение 0 соответствует неограниченному числу запросов
`maxWait`	Время ожидания пулом возврата соединения в пул (при отсутствии свободных соединений) до возбуждения исключения. Значение –1 соответствует бесконечному ожиданию
`minEvictableIdleTimeMillis`	Максимальное время простоя соединения, прежде чем оно может быть удалено из пула
`minIdle`	Минимальное число простаивающих соединений, которые могут оставаться в пуле без создания новых соединений
`poolPreparedStatements`	Признак поддержки пула скомпилированных запросов (логическое значение)

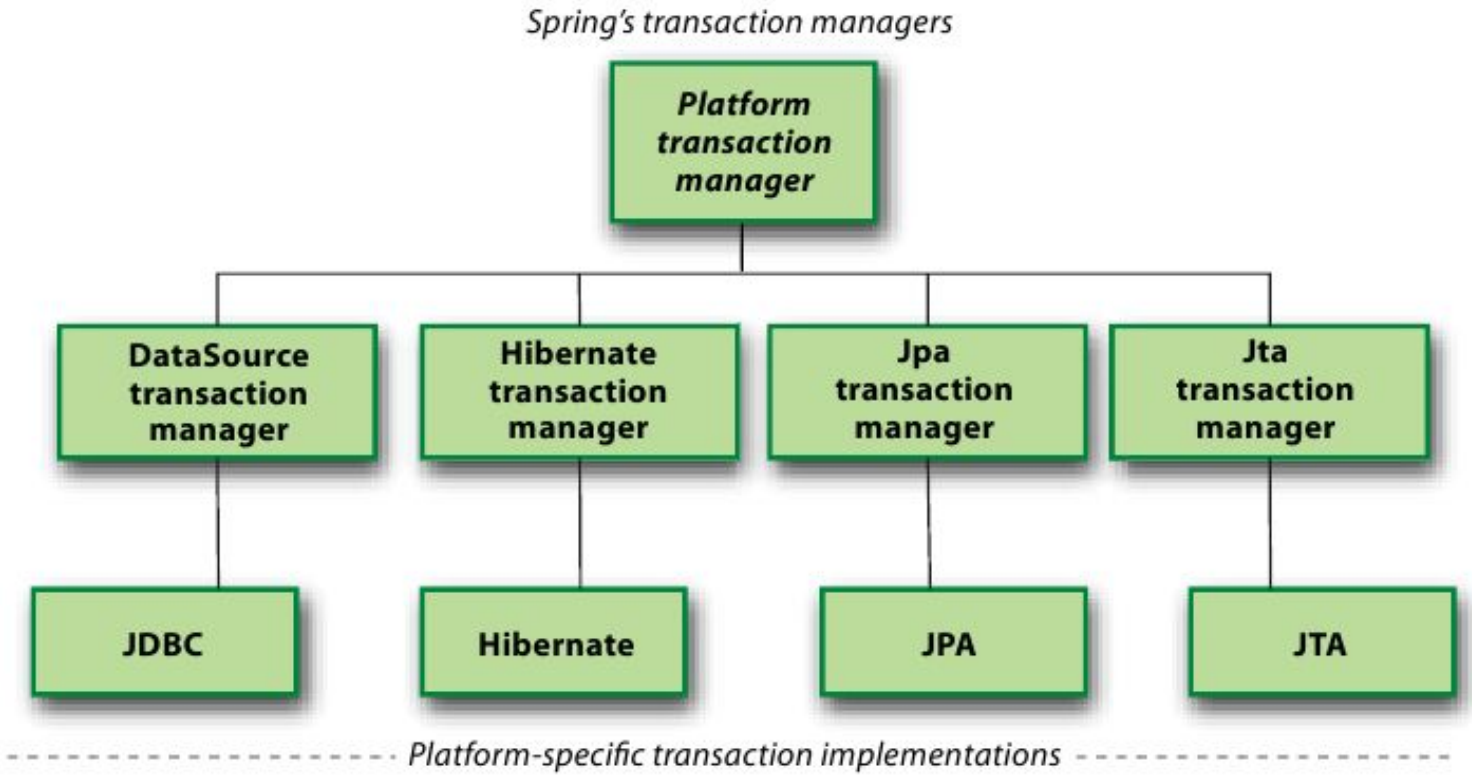
IT-Academy

ВАШИ ВОПРОСЫ?

Understanding transactions



Choosing a transaction manager



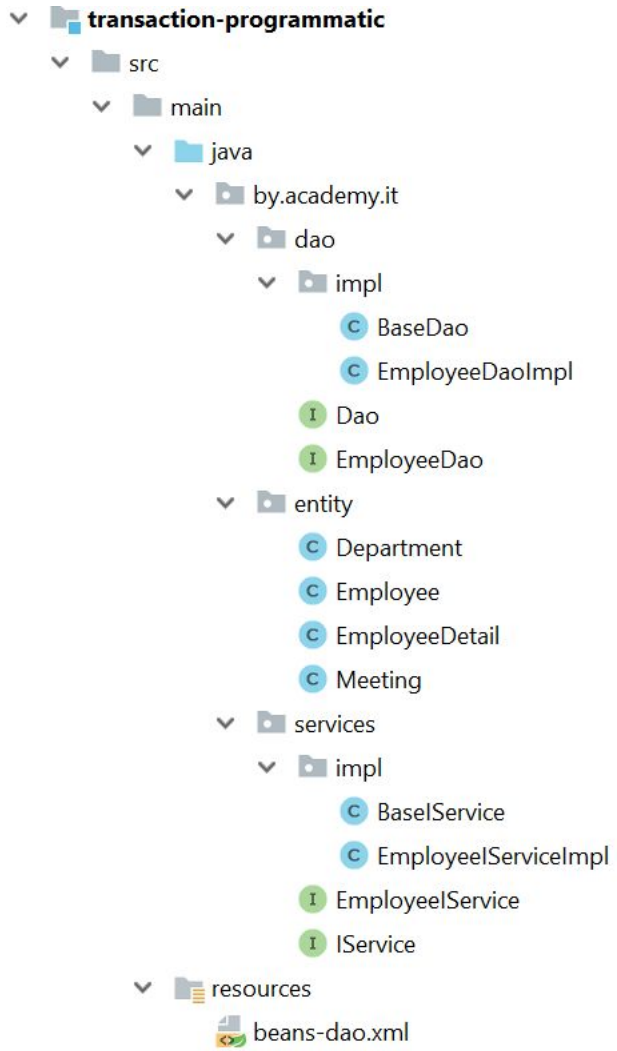
Choosing a transaction manager

IT-Academy

- ✔ org.springframework.jdbc.datasource.DataSourceTransactionManager
- ✔ org.springframework.transaction.jta.JtaTransactionManager
- ✔ org.springframework.orm.hibernate5.HibernateTransactionManager
- ✔ org.springframework.orm.jpa.JpaTransactionManager

Programming transactions in Spring

```java
@Repository
public class BaseDao<T> implements Dao<T> {
    Class<T> clazz;
    @PersistenceContext
    @Getter
    private EntityManager em;

    @Override
    public T add(T t) {
        em.persist(t);
        return t;
    }
    @Override
    public T get(Serializable id) {
        return em.find(clazz, id);
    }
    @Override
    public T update(T t) {
        em.merge(t);
        return t;
    }
    @Override
    public void delete(Serializable id) {
        T t = em.find(clazz, id);
        em.remove(t);
    }
}
```

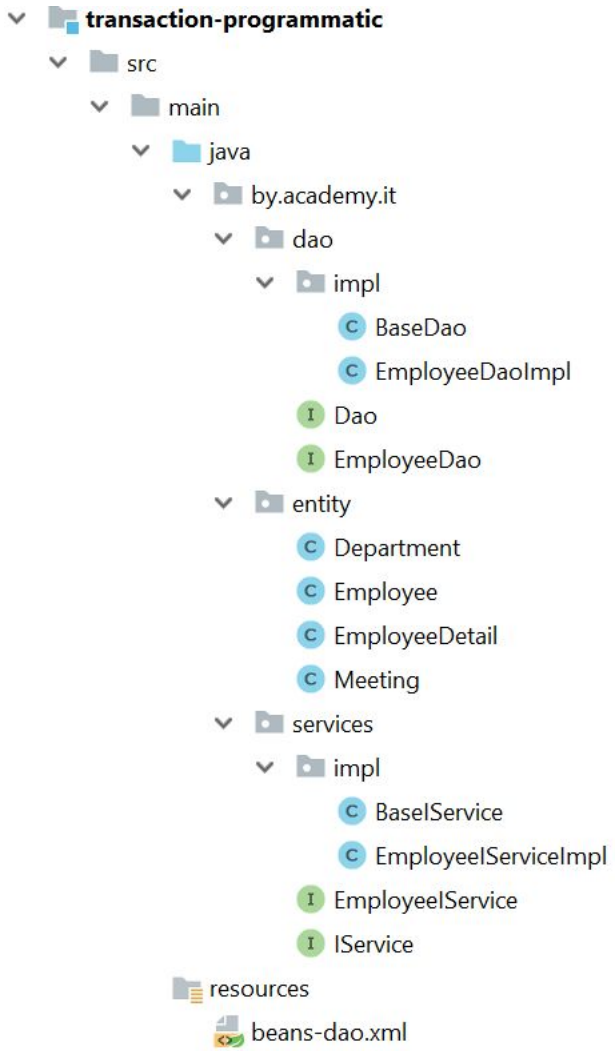


Programming transactions in Spring

```java
public interface Dao<T> {
    T add(T t);
    T update(T t);
    T get(Serializable id);
    void delete(Serializable id);
}
```

```java
public interface EmployeeDao extends Dao<Employee> {
    List<Employee> getEmployee();
}
```

```java
@Repository
public class EmployeeDaoImpl extends BaseDao<Employee>
implements EmployeeDao {
    public EmployeeDaoImpl() {
        super();
        clazz = Employee.class;
    }

    @Override
    public List<Employee> getEmployee() {
        return getEm()
     .createQuery("from Employee").getResultList();
    }
}
```



Programming transactions in Spring

IT-Academy

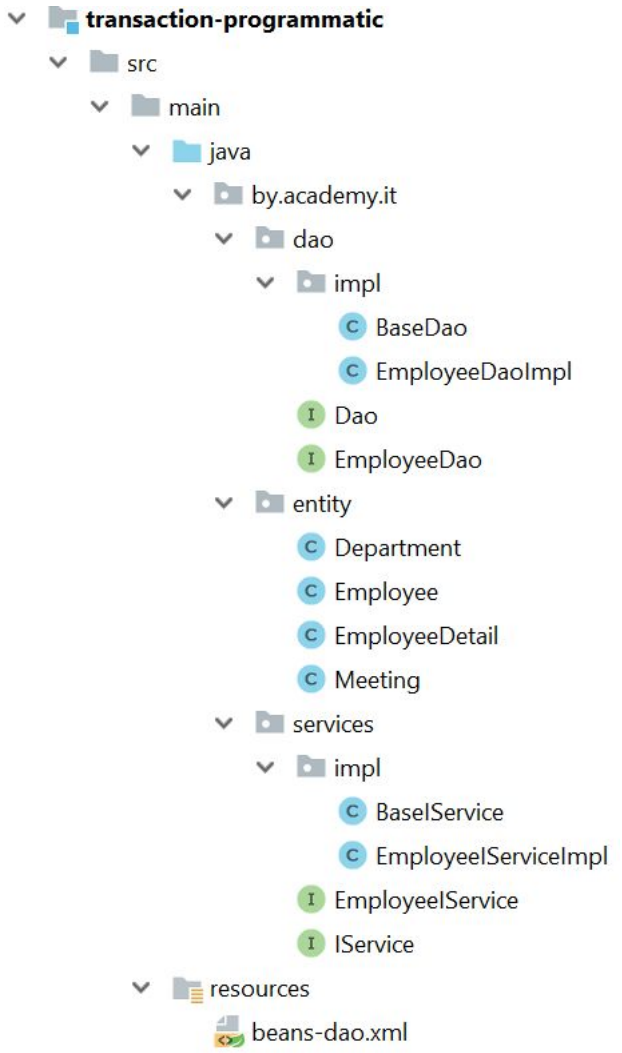
```
@Service
public class BaseService<T> implements IService<T> {
    @Autowired
    private Dao<T> baseDao;
    @Autowired
    TransactionTemplate  transactionTemplate;
    @Override
    public T add(T t) {
        return transactionTemplate.execute(new TransactionCallback<T>() {
            public T doInTransaction(TransactionStatus transactionStatus) {
                try {
                    return baseDao.add(t);
                } catch (Exception e) {
                    transactionStatus.setRollbackOnly();
                }
                return null;
            }
        });
    }
    @Override
    public T update(T t) {
        return null;
    }
    @Override
    public T get(Serializable id) {
        return baseDao.get(id);
    }
    @Override
    public void delete(Serializable id) {
        baseDao.delete(id);
    }
}
```

Programming transactions in Spring

```
public interface IService<T> {
    T add(T t);
    T update(T t);
    T get(Serializable id);
    void delete(Serializable id);
}
```

```
public interface EmployeeService extends IService<Employee> { }
```

```
@Service
public class EmployeeServiceImpl extends
    BaseIService<Employee> implements EmployeeIService {

}
```



Programming transactions in Spring

IT-Academy

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="by.academy.it.dao"/>
    <context:component-scan base-package="by.academy.it.services"/>

    <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
        <property name="url"
value="jdbc:mysql://localhost:3306/spring_hibernate_integration "/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="username" value="root"/>
        <property name="password" value="yuli"/>
        <property name="initialSize" value="5"/>
        <property name="maxTotal" value="20"/>
    </bean>
```

Programming transactions in Spring

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="persistenceUnitName" value="jpa-unit"/>
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
        </property>
        <property name="packagesToScan">
            <list>
                <value>by.academy.it.entity</value>
            </list>
        </property>
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQL55Dialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.hbm2ddl.auto">create</prop>
            </props>
        </property>
    </bean>
    <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory"/>
    </bean>
    <bean id="transactionTemplate"
class="org.springframework.transaction.support.TransactionTemplate">
        <property name="transactionManager" ref="txManager"/>
    </bean>
</beans>
```

Вопросы

IT-Academy

Declaring transactions. Conception

Управление возвращается через цепочку перехватчиков, чтобы вернуть результат в точку вызова

Объект вызывает прокси, а не метод целевого класса

Транзакция создается на входе, фиксируется или откатывается на выходе

Вызванная бизнес-логика

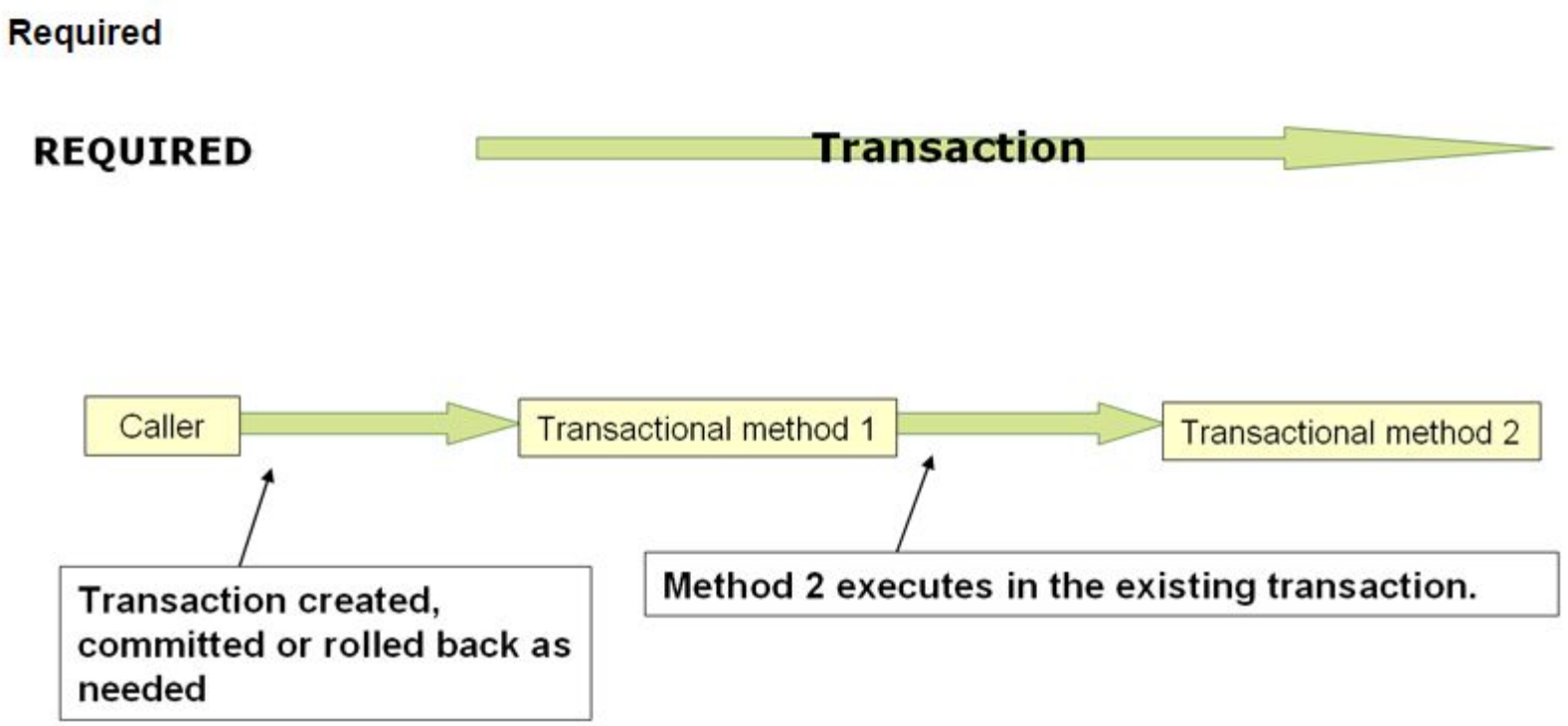
Пользовательские советы могут выполниться до транзакционного совета after

Declaring transactions

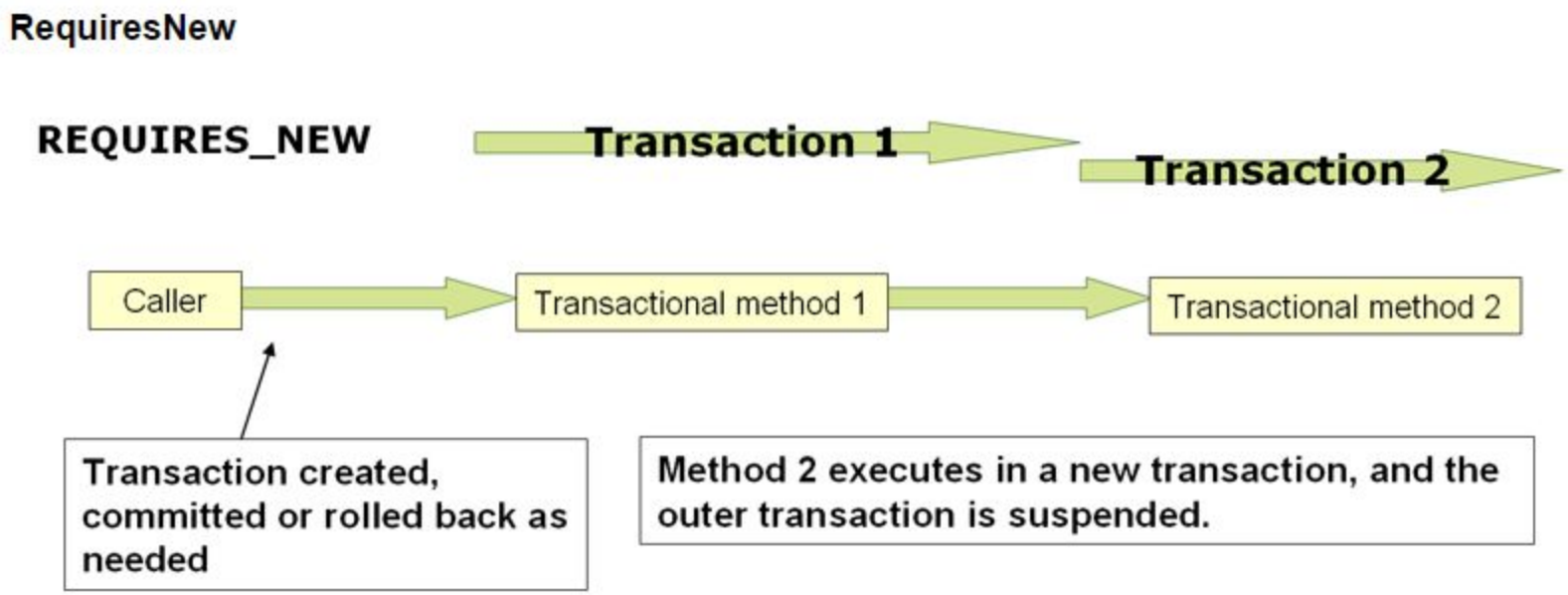
IT-Academy

Propagation behavior	What it means
PROPAGATION_MANDATORY	Indicates that the method must run within a transaction. If no existing transaction is in progress, an exception will be thrown.
PROPAGATION_NESTED	Indicates that the method should be run within a nested transaction if an existing transaction is in progress. The nested transaction can be committed and rolled back individually from the enclosing transaction. If no enclosing transaction exists, behaves like PROPAGATION_REQUIRED. Vendor support for this propagation behavior is spotty at best. Consult the documentation for your resource manager to determine if nested transactions are supported.
PROPAGATION_NEVER	Indicates that the current method shouldn't run within a transactional context. If an existing transaction is in progress, an exception will be thrown.
PROPAGATION_NOT_SUPPORTED	Indicates that the method shouldn't run within a transaction. If an existing transaction is in progress, it'll be suspended for the duration of the method. If using JTATransactionManager, access to TransactionManager is required.
PROPAGATION_REQUIRED	Indicates that the current method must run within a transaction. If an existing transaction is in progress, the method will run within that transaction. Otherwise, a new transaction will be started.
PROPAGATION_REQUIRES_NEW	Indicates that the current method must run within its own transaction. A new transaction is started and if an existing transaction is in progress, it'll be suspended for the duration of the method. If using JTATransactionManager, access to TransactionManager is required.
PROPAGATION_SUPPORTS	Indicates that the current method doesn't require a transactional context, but may run within a transaction if one is already in progress.

Declaring transactions. Propagation – “Required”



Declaring transactions. Propagation – “Required new”



Declaring transactions

IT-Academy

Isolation level	What it means
ISOLATION_DEFAULT	Use the default isolation level of the underlying data store.
ISOLATION_READ_UNCOMMITTED	Allows you to read changes that haven't yet been committed. May result in dirty reads, phantom reads, and nonrepeatable reads.
ISOLATION_READ_COMMITTED	Allows reads from concurrent transactions that have been committed. Dirty reads are prevented, but phantom and nonrepeatable reads may still occur.
ISOLATION_REPEATABLE_READ	Multiple reads of the same field will yield the same results, unless changed by the transaction itself. Dirty reads and nonrepeatable reads are prevented, but phantom reads may still occur.
ISOLATION_SERIALIZABLE	This fully ACID-compliant isolation level ensures that dirty reads, nonrepeatable reads, and phantom reads are all prevented. This is the slowest of all isolation levels because it's typically accomplished by doing full table locks on the tables involved in the transaction.

Declaring transactions in XML

IT-Academy

Attribute	Purpose
`isolation`	Specifies the transaction isolation level.
`propagation`	Defines the transaction's propagation rule.
`read-only`	Specifies that a transaction be read-only.

Rollback rules:
`rollback-for`
`no-rollback-for`
	`rollback-for` specifies checked exceptions for which a transaction should be rolled back and not committed.
	`no-rollback-for` specifies exceptions for which the transaction should continue and not be rolled back.

`timeout`	Defines a timeout for a long-running transaction.

Declaring transactions in XML

IT-Academy

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       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/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/tx
                http://www.springframework.org/schema/tx/spring-tx.xsd
                http://www.springframework.org/schema/aop
                http://www.springframework.org/schema/aop/spring-aop.xsd">
    <context:component-scan base-package="by.academy.it.dao"/>
    <context:component-scan base-package="by.academy.it.services"/>

    <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
        <property name="url"
          value="jdbc:mysql://localhost:3306/spring_hibernate_integration "/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="username" value="root"/>
        <property name="password" value="yuli"/>
        <property name="initialSize" value="5"/>
        <property name="maxTotal" value="20"/>
    </bean>
```

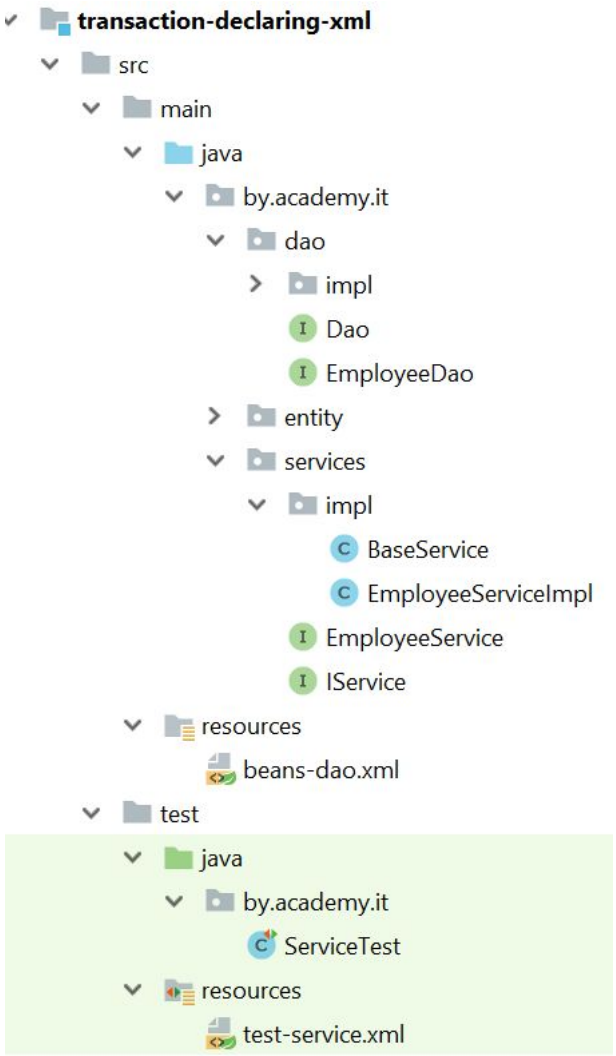
Declaring transactions in XML

IT-Academy

```
<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="*"/>
        <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:advisor pointcut="execution(* by.academy.it.services.IService.*(..))"
                 advice-ref="txAdvice"/>
</aop:config>
```

Declaring transactions in XML

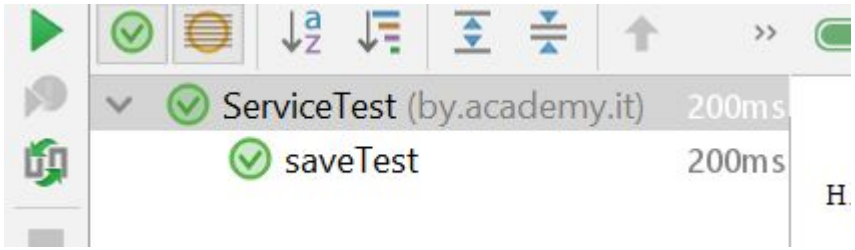
```
@Service
public class BaseService<T> implements
IService<T> {
    @Autowired
    private Dao<T> baseDao;
    @Override
    public T add(T t) {
        return baseDao.add(t);
    }
    @Override
    public T update(T t) {
        return null;
    }
    @Override
    public T get(Serializable id) {
        return baseDao.get(id);
    }
    @Override
    public void delete(Serializable id) {
        baseDao.delete(id);
    }
}
```



Declaring transactions in XML

IT-Academy

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:test-service.xml")
public class ServiceTest {

    @Autowired
    private EmployeeService employeeService;

    @Test
    public void saveTest() {
        Employee e = new Employee();
        e.setFirstName("Yulij I");
        e.setLastName("Slabko");
        e = employeeService.add(e);

        Assert.assertEquals("Yulij I", employeeService.get(e.getId()).getFirstName());
    }
}
```



Вопросы

IT-Academy

Defining annotation-driven transactions

IT-Academy

```
@Service
@Transactional
public class BaseService<T> implements IService<T> {
    @Autowired
    private Dao<T> baseDao;
    @Override
    public T add(T t) {
        return baseDao.add(t);
    }
    @Override
    public T update(T t) {
        return null;
    }
    @Override
    @Transactional(
            propagation = Propagation.SUPPORTS,
            readOnly = true,
            timeout = 60
    )
    public T get(Serializable id) {
        return baseDao.get(id);
    }
    @Override
    public void delete(Serializable id) {
        baseDao.delete(id);
    }
}
```

Defining annotation-driven transactions

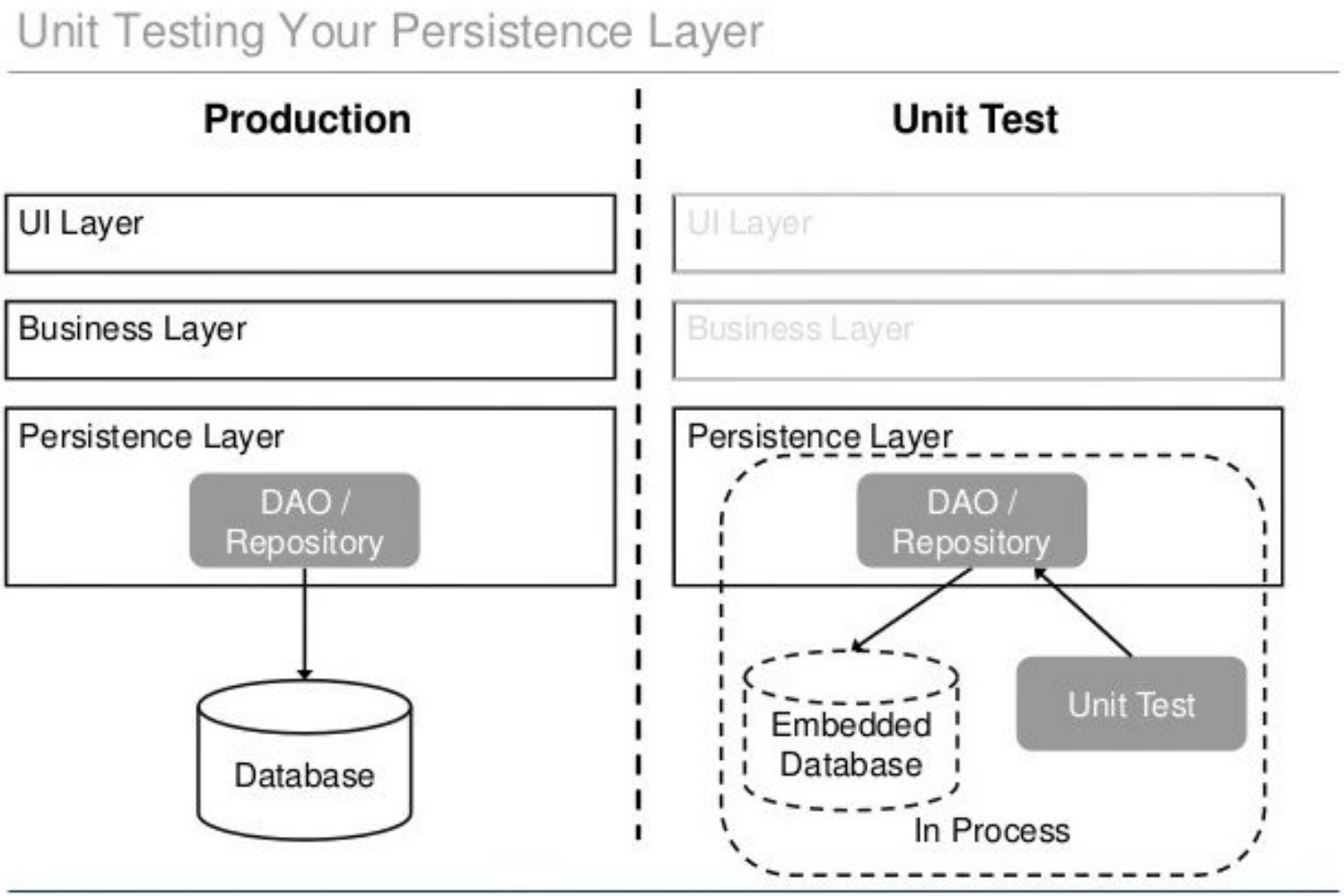
IT-Academy

```
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="persistenceUnitName" value="jpa-unit"/>
    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
    </property>
    <property name="packagesToScan">
        <list>
            <value>by.academy.it.entity</value>
        </list>
    </property>
    <property name="jpaProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQL55Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
            <prop key="hibernate.hbm2ddl.auto">create</prop>
        </props>
    </property>
</bean>
<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>
<tx:annotation-driven transaction-manager="txManager"/>
```

Вопросы

IT-Academy

Unit-testing persistence layer with Spring



DaoTest.java

IT-Academy

```java
@ContextConfiguration("/testContext.xml")
//@ContextConfiguration(classes = { HibernateTestConfiguration.class })
@FixMethodOrder(MethodSorters.NAME_ASCENDING) // Baaaaaaadd!!!!
@RunWith(SpringJUnit4ClassRunner.class)
@Transactional
public class DaoTest{
    @Autowired
    private PersonDao personDao;
    @Test
    @Transactional
    @Rollback(false)
    public void addPerson() {
        Person p = new Person(); p.setName("Yuli"); p.setSurname("Slabko"); p.setAge(30);
        Person persistent = personDao.add(p);
        assertNotNull(persistent.getId());
        persistent = personDao.get(persistent.getId());
        assertEquals("Person is not persisted", p, persistent);
    }

    @Test
    @Transactional
    public void deletePerson() {
        List<Person> list = personDao.getPersons();
        int size = list.size();
        if (list.size() > 0) {
            Person persistent = list.get(0);
            personDao.delete(persistent);
            assertNotSame(personDao.getPersons().size(), size);
        }
    }
}
```

testContext.xml

IT-Academy

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd">
    <context:annotation-config/>
    <aop:aspectj-autoproxy proxy-target-class="true" expose-proxy="false"/>
    <tx:annotation-driven transaction-manager="txManager"/>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="url" value="jdbc:mysql://localhost:3306/test_mvc_person?createDatabaseIfNotExist=true"/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="username" value="root"/>
        <property name="password" value="yuli"/>
        <property name="initialSize" value="2"/>
        <property name="maxActive" value="3"/>
    </bean>
```

testContext.xml

IT-Academy

```
    <bean id="sessionFactory"
          class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="packagesToScan">
            <list>
                <value>by.it.academy.pojos</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="debug">true</prop>
                <prop key="connection.isolation">2</prop>
                <prop key="hibernate.cglib.use_reflection_optimizer">true</prop>
                <prop key="hibernate.hbm2ddl.auto">create-drop</prop>
            </props>
        </property>
    </bean>
    <bean id="txManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="personDao" class="by.it.academy.dao.PersonDao">
        <constructor-arg ref="sessionFactory"/>
    </bean>
</beans>
```

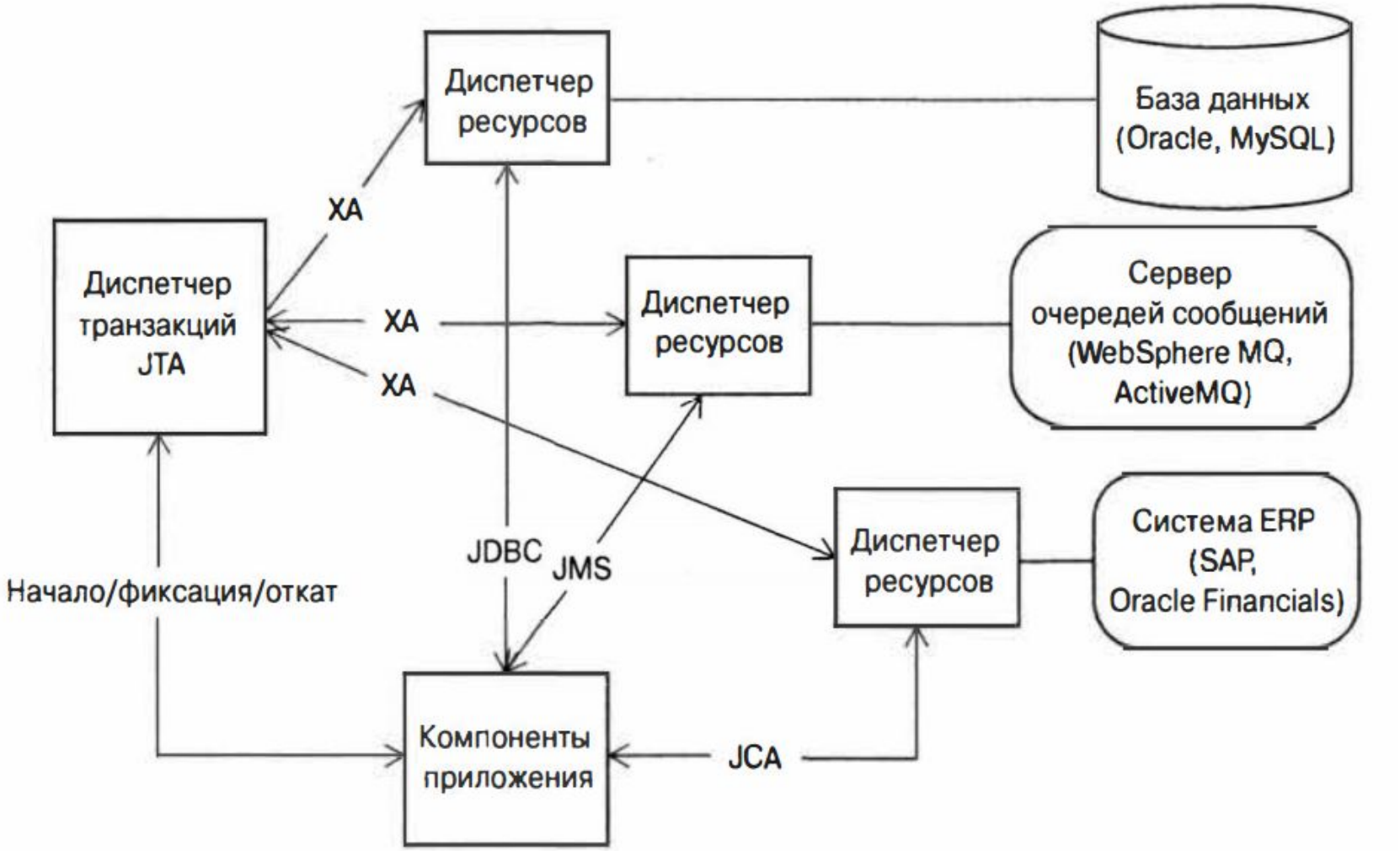
Results

IT-Academy

DaoTest (by.it 250ms
addPerson 103ms
deletePers 147ms

```
"C:\Program Files\Java\jdk1.8.0_77\bin\java" ...
Hibernate: alter table PERSON_PERSON_PROFILE drop foreign key FK_442i0iv2hl9ld24p8b142ceqk
Hibernate: alter table PERSON_PERSON_PROFILE drop foreign key FK_edovs1y432auh1y4j268pqam9
Hibernate: drop table if exists PERSON
Hibernate: drop table if exists PERSON_PERSON_PROFILE
Hibernate: drop table if exists PERSON_PROFILE
Hibernate: drop table if exists PRODUCT
Hibernate: create table PERSON (PERSON_ID integer not null auto_increment, AGE integer, LOGIN varchar(30) check (LOGIN<=30 AND LOGIN>=5), NAME
 varchar(255), PASSWORD varchar(255) check (PASSWORD<=30 AND PASSWORD>=6), STATE varchar(255) not null, SURNAME varchar(255), primary key
 (PERSON_ID))
Hibernate: create table PERSON_PERSON_PROFILE (PERSON_ID integer not null, PERSON_PROFILE_ID bigint not null, primary key (PERSON_ID,
 PERSON_PROFILE_ID))
Hibernate: create table PERSON_PROFILE (id bigint not null auto_increment, TYPE varchar(15) not null, primary key (id))
Hibernate: create table PRODUCT (ID integer not null auto_increment, MODEL varchar(255), NAME varchar(255), PRICE double precision, primary key
 (ID))
Hibernate: alter table PERSON_PROFILE add constraint UK_f0ffdpm077wfm8uwwip5yvl2s  unique (TYPE)
Hibernate: alter table PERSON_PERSON_PROFILE add constraint FK_442i0iv2hl9ld24p8b142ceqk foreign key (PERSON_PROFILE_ID) references
 PERSON_PROFILE (id)
Hibernate: alter table PERSON_PERSON_PROFILE add constraint FK_edovs1y432auh1y4j268pqam9 foreign key (PERSON_ID) references PERSON (PERSON_ID)
Hibernate: insert into PERSON (AGE, LOGIN, NAME, PASSWORD, STATE, SURNAME) values (?, ?, ?, ?, ?, ?)
Hibernate: select person0_.PERSON_ID as PERSON_I1_0_, person0_.AGE as AGE2_0_, person0_.LOGIN as LOGIN3_0_, person0_.NAME as NAME4_0_,
 person0_.PASSWORD as PASSWORD5_0_, person0_.STATE as STATE6_0_, person0_.SURNAME as SURNAME7_0_ from PERSON person0_
Hibernate: select personprof0_.PERSON_ID as PERSON_I1_0_0_, personprof0_.PERSON_PROFILE_ID as PERSON_P2_1_0_, personprof1_.id as id1_2_1_,
 personprof1_.TYPE as TYPE2_2_1_ from PERSON_PERSON_PROFILE personprof0_ inner join PERSON_PROFILE personprof1_ on personprof0_
 .PERSON_PROFILE_ID=personprof1_.id where personprof0_.PERSON_ID=?
Hibernate: delete from PERSON where PERSON_ID=?
Hibernate: select person0_.PERSON_ID as PERSON_I1_0_, person0_.AGE as AGE2_0_, person0_.LOGIN as LOGIN3_0_, person0_.NAME as NAME4_0_,
 person0_.PASSWORD as PASSWORD5_0_, person0_.STATE as STATE6_0_, person0_.SURNAME as SURNAME7_0_ from PERSON person0_
Hibernate: alter table PERSON_PERSON_PROFILE drop foreign key FK_442i0iv2hl9ld24p8b142ceqk
Hibernate: alter table PERSON_PERSON_PROFILE drop foreign key FK_edovs1y432auh1y4j268pqam9
Hibernate: drop table if exists PERSON
Hibernate: drop table if exists PERSON_PERSON_PROFILE
Hibernate: drop table if exists PERSON_PROFILE
Hibernate: drop table if exists PRODUCT

Process finished with exit code 0
```

Choosing a transaction manager



Вопросы

IT-Academy

IT-Academy

Спасибо за внимание