Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение науки и высшего образования**

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

**Институт информационных технологий и радиотехники (ИИТР)**

**Лабораторная работа № 4**

**по дисциплине**

**«Распределённые программные системы»**

Тема «Распределённые транзакции»

Выполнила:

ст. гр. ИСТ-117

Шумейко Д.С.

Принял:

Макаров С.А.

Владимир, 2020

**Цель работы:**

Получить сведения о распределенных транзакциях в Java EE-приложениях и закрепить их с помощью экспериментов.

**Репозиторий:** <https://github.com/daryashumeyko/rps_6semestr>

**Ход работы:**

Описание кнопок:

1. Выполнить добавление в обе БД успешно.  
Результат: Транзакция завершилась, данные в обе БД добавлены.

2. В первой БД – откат (setRollbackOnly()), а во второй успешно.  
Результат: данные не добавлены ни в одну БД, т.к. откат в добавлении в первую БД привел к откату всей транзакции.

3. В первой БД – успешное добавление, во второй – выбросить EJBException.  
Результат: данные не добавлены ни в одну БД, т.к. исключение привело к откату всей транзакции.

4. В первой БД – откат (setRollbackOnly()), а вторая завершается успешно, но метод выполняется вне контекста транзакции.  
Результат: данные не добавлены ни в одну БД, т.к. выполнение метода добавления во второй БД привело к выбрасыванию исключения TransactionRequiredException. Это случилось, потому что менеджеру сущностей обязательно нужна транзакция, чтобы брать оттуда контекст персистентности.

5. Метод добавления в первую БД выполняется в контексте новой транзакции, а в методе добавления в вторую БД – выбрасывается EJBException.  
Результат: Данные добавлены только в первую БД, т.к. она выполнялась в контексте новой транзакции. Исключение, выброшенное во время добавления во вторую транзакцию, привело только к откату общей транзакции. Новую транзакцию этот откат не затронул.

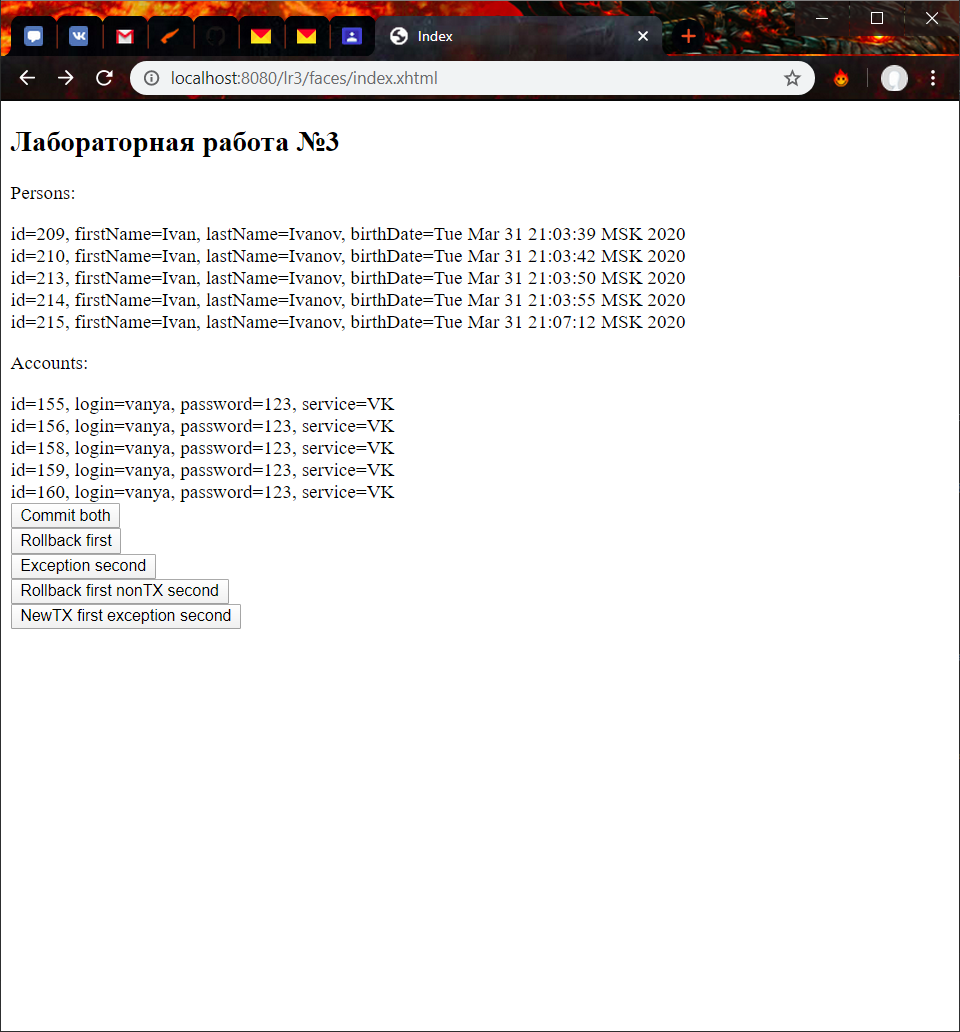


Рис.1.

package ru.rps.lr3.ejb.local.component;  
  
import java.util.List;  
import javax.ejb.EJBException;  
import javax.ejb.Stateless;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import javax.transaction.Transactional;  
  
import ru.rps.lr3.ejb.local.entity.Account;  
  
@Stateless  
public class AccountDAOLocalImpl implements AccountDAOLocal {  
  
 @PersistenceContext(unitName = "transactAccount")  
 private EntityManager em;  
  
 @Override  
 public void createSuccess(Account account) {  
 em.persist(account);  
 }  
  
 @Override  
 public void createException(Account account) throws EJBException {  
 em.persist(account);  
 throw new EJBException();  
 }  
  
 @Override  
 @Transactional(Transactional.TxType.*NOT\_SUPPORTED*)  
 public void createNonTX(Account account) {  
 em.persist(account);  
 }  
  
 @Override  
 public List<Account> getAll() {  
 return em.createQuery("select a from Account a", Account.class).getResultList();  
 }  
}

package ru.rps.lr3.ejb.remote.component;  
  
import java.util.List;  
import javax.annotation.Resource;  
import javax.ejb.SessionContext;  
import javax.ejb.Stateless;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import javax.transaction.Transactional;  
  
import ru.rps.lr3.ejb.remote.client.PersonDAORemote;  
import ru.rps.lr3.ejb.remote.entity.Person;  
  
@Stateless  
public class PersonDAORemoteImpl implements PersonDAORemote {  
  
 @PersistenceContext(unitName = "transactPerson")  
 private EntityManager em;  
  
 @Resource  
 private SessionContext sessionContext;  
  
 @Override  
 public void createSuccess(Person person) {  
 em.persist(person);  
 }  
  
 @Override  
 public void createRollback(Person person) {  
 em.persist(person);  
 sessionContext.setRollbackOnly();  
 }  
  
 @Override  
 @Transactional(Transactional.TxType.*REQUIRES\_NEW*)  
 public void createNewTX(Person person) {  
 em.persist(person);  
 }  
  
 @Override  
 public List<Person> getAll() {  
 return em.createQuery("select p from Person p", Person.class).getResultList();  
 }  
}

package ru.rps.lr3.ejb.remote.entity;  
  
import java.io.Serializable;  
import java.util.Date;  
import java.util.Objects;  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "person")  
public class Person implements Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 @Column  
 private String firstName;  
 @Column  
 private String lastName;  
 @Column  
 @Temporal(TemporalType.*DATE*)  
 private Date birthDate;  
  
 public Person() {  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public Date getBirthDate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(Date birthDate) {  
 this.birthDate = birthDate;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if(this == o) {  
 return true;  
 }  
 if(o == null || getClass() != o.getClass()) {  
 return false;  
 }  
 Person person = (Person) o;  
 return Objects.*equals*(getId(), person.getId()) &&  
 Objects.*equals*(getFirstName(), person.getFirstName()) &&  
 Objects.*equals*(getLastName(), person.getLastName()) &&  
 Objects.*equals*(getBirthDate(), person.getBirthDate());  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(getId(), getFirstName(), getLastName(), getBirthDate());  
 }  
  
 @Override  
 public String toString() {  
 return "id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", birthDate=" + birthDate;  
 }  
}

package ru.rps.lr3.web.component;  
  
import java.util.List;  
import javax.ejb.EJB;  
import javax.ejb.LocalBean;  
import javax.ejb.Stateless;  
import javax.transaction.Transactional;  
  
import ru.rps.lr3.ejb.local.component.AccountDAOLocal;  
import ru.rps.lr3.ejb.local.entity.Account;  
import ru.rps.lr3.ejb.remote.client.PersonDAORemote;  
import ru.rps.lr3.ejb.remote.entity.Person;  
  
@LocalBean  
@Stateless  
public class Experiments {  
 @EJB  
 private PersonDAORemote personDAO;  
 @EJB  
 private AccountDAOLocal accountDAO;  
  
 @Transactional  
 public void commitBoth(Person person, Account account) {  
 personDAO.createSuccess(person);  
 accountDAO.createSuccess(account);  
 }  
  
 @Transactional  
 public void rollbackFirst(Person person, Account account) {  
 personDAO.createRollback(person);  
 accountDAO.createSuccess(account);  
 }  
  
 @Transactional  
 public void exceptionSecond(Person person, Account account) {  
 personDAO.createSuccess(person);  
 accountDAO.createException(account);  
 }  
  
 @Transactional  
 public void rollbackFirstNonTXSecond(Person person, Account account) {  
 accountDAO.createNonTX(account);  
 personDAO.createRollback(person);  
 }  
  
 @Transactional  
 public void newTXFirstExceptionSecond(Person person, Account account) {  
 personDAO.createNewTX(person);  
 accountDAO.createException(account);  
 }  
  
 public List<Account> getAccounts() {  
 return accountDAO.getAll();  
 }  
  
 public List<Person> getPersons() {  
 return personDAO.getAll();  
 }  
}

package ru.rps.lr3.web.mb;  
  
import java.io.Serializable;  
import java.util.Date;  
import java.util.List;  
import javax.ejb.EJB;  
import javax.enterprise.context.RequestScoped;  
import javax.inject.Named;  
  
import ru.rps.lr3.ejb.remote.entity.Person;  
import ru.rps.lr3.ejb.local.entity.Account;  
import ru.rps.lr3.web.component.Experiments;  
  
@Named("experiment")  
@RequestScoped  
public class ExperimentController implements Serializable {  
  
 @EJB  
 private Experiments experiments;  
  
 public String commitBoth() {  
 Person person = new Person();  
 person.setFirstName("Ivan");  
 person.setLastName("Ivanov");  
 person.setBirthDate(new Date());  
// Person person = new Person("Ivan", "Ivanov", new Date());  
 Account account = new Account("vanya", "123", "VK");  
 experiments.commitBoth(person, account);  
 return "index";  
 }  
  
 public String rollbackFirst() {  
 Person person = new Person();  
 person.setFirstName("Ivan");  
 person.setLastName("Ivanov");  
 person.setBirthDate(new Date());  
  
 //Person person = new Person("Petr", "Petrov", new Date());  
 Account account = new Account("petya", "123", "FACEBOOK");  
 experiments.rollbackFirst(person, account);  
 return "index";  
 }  
  
 public String exceptionSecond() {  
 Person person = new Person();  
 person.setFirstName("Ivan");  
 person.setLastName("Ivanov");  
 person.setBirthDate(new Date());  
  
 //Person person = new Person("Sergey", "Sergeev", new Date());  
 Account account = new Account("sereja", "123", "TWITTER");  
 experiments.exceptionSecond(person, account);  
 return "index";  
 }  
  
 public String rollbackFirstNonTXSecond() {  
 Person person = new Person();  
 person.setFirstName("Ivan");  
 person.setLastName("Ivanov");  
 person.setBirthDate(new Date());  
  
 //Person person = new Person("Alexey", "Alexeev", new Date());  
 Account account = new Account("lexa", "123", "GOOGLE");  
 experiments.rollbackFirstNonTXSecond(person, account);  
 return "index";  
 }  
  
 public String newTXFirstExceptionSecond() {  
 Person person = new Person();  
 person.setFirstName("Ivan");  
 person.setLastName("Ivanov");  
 person.setBirthDate(new Date());  
  
 //Person person = new Person("Andrey", "Andreev", new Date());  
 Account account = new Account("andrew", "123", "MAIL");  
 experiments.newTXFirstExceptionSecond(person, account);  
 return "index";  
 }  
  
 public List<Account> getAccounts() {  
 return experiments.getAccounts();  
 }  
  
 public List<Person> getPersons() {  
 return experiments.getPersons();  
 }  
}

**Выводы:**

В ходе выполнения данной лабораторной работы были получены сведения о распределенных транзакциях в Java EE-приложениях и закреплены с помощью экспериментов.