Подсказки: Hibernate

Подсказки смотрите только после того, как сами **уже решили задачу или попробовали, но не получилось**

Конфигурация Hibernate
Привязка единой сессии к потоку исполнения
Модель пользователя с аннотациями Hibernate
Использование транзакций в сервисах

Конфигурация Hibernate

```
@Configuration
public class HibernateConfiguration {

    @Bean
    public SessionFactory sessionFactory() {
        org.hibernate.cfg.Configuration configuration = new org.hibernate.cfg.Configur
ation();

        configuration
                .addAnnotatedClass(Account.class)
                 .addAnnotatedClass(User.class)
                .addPackage("your.package")
                .setProperty("hibernate.connection.driver_class", "org.postgresql.Driv
er")
                .setProperty("hibernate.connection.url", "jdbc:postgresql://localhost:
5432/postgres")
                .setProperty("hibernate.connection.username", "postgres")
                .setProperty("hibernate.connection.password", "root")
                .setProperty("hibernate.show_sql", "true")
                .setProperty("hibernate.hbm2ddl.auto", "update");

        return configuration.buildSessionFactory();
    }

}
```

Привязка единой сессии к потоку исполнения

В многопоточных приложениях может возникнуть проблема с управлением сессиями Hibernate. Если сессия используется несколькими потоками одновременно, это может привести к неожиданным ошибкам и некорректной работе приложения. Необходимо обеспечить, чтобы каждая сессия была связана только с одним потоком исполнения.

Решение

Hibernate предоставляет механизм для управления сессиями в одном потоке с использованием параметра `hibernate.current_session_context_class`. Установка значения этого параметра в `thread` позволяет Hibernate

автоматически привязывать сессию к текущему потоку, что обеспечивает безопасное использование сессий в многопоточной среде.

1. **Настройка конфигурации Hibernate в Java:**

Создайте класс конфигурации Hibernate и установите свойство `hibernate.current_session_context_class` в `thread`.

```
configuration.setProperty("hibernate.current_session_context_class", "thread");
```

2. **Использование сессий в коде:**

Вам нужно будет использовать `getCurrentSession()` для получения текущей сессии в каждом потоке. Пример использования в сервисе:

```
import org.hibernate.Session;
import org.hibernate.Transaction;

public class UserService {

    public void createUser(User user) {
        Session session = null;
        Transaction transaction = null;
        try {
            session = sessionFactory.getCurrentSession();
            transaction = session.beginTransaction();
            session.save(user);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        } finally {
             if (session != null) {
                       session.close();
               }
        }
    }

    public User findUserById(Long id) {
        session = sessionFactory.getCurrentSession();
        return session.get(User.class, id);
    }
}
```

Теперь каждая сессия будет связана с текущим потоком, что обеспечит безопасное использование сессий в многопоточной среде.

Модель пользователя с аннотациями Hibernate

```
@Table(name = "users")
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "login", unique = true, updatable = false)
    private String login;
```

```
    @OneToMany(mappedBy = "user", fetch = FetchType.EAGER)
    private List<Account> accountList;

    // getters and setters
}
```

Использование транзакций в сервисах

Советуем вынести логику по работе с транзакциями в вспомогательный метод.
Также предусматривается вариант выполнения, когда один метод в транзакции вызывает другой транзакционный метод.
В этом случае второй метод должен увидеть существующую транзакцию и не закрывать ее при окончании работы. Эту транзакцию должен закрыть тот метод, который ее открыл.

```
public void deposit(Long toAccountId, Integer amount) {
    executeInTransaction(() -> {
            ...
    });
}

public<T> T executeInTransaction(Supplier<T> action) {
    var session = sessionFactory.getCurrentSession();
    Transaction transaction = session.getTransaction();

        // если транзакция уже не в статусе NOT_ACTIVE
        // просто продолжаем выполнение в этой транзакции
    if (!transaction.getStatus().equals(TransactionStatus.NOT_ACTIVE)) {
        return action.get();
    }

    try {
        session.beginTransaction();
        T returnValue = action.get();
        transaction.commit();
        return returnValue;
    } catch (Exception e) {
        transaction.rollback();
        throw e;
    } finally {
        session.close();
    }
}
```