**Лабораторная работа №7**

1. Что такое транзакция?

Транзакция — минимальная логически осмысленная операция, которая имеет смысл и может быть совершена только полностью.

1. В чем разница между Spring Boot и Spring MVC?

Hibernate одна из самых популярных открытых реализаций последней версии спецификации (JPA 2.1). Даже скорее самая популярная, почти стандарт де-факто. То есть JPA только описывает правила и API, а Hibernate реализует эти описания, впрочем у Hibernate (как и у многих других реализаций JPA) есть дополнительные возможности, не описанные в JPA (и не переносимые на другие реализации JPA).

1. Что такое **Entity?**

это некий объектный "образ" сущности, хранимой в БД.

В простейшем случае, entity находит свое отображение в виде таблицы, но это не всегда так. Например, в случае описания сложных связей (многие ко многим, например), будет создана дополнительно промежуточная таблица связей.

**Entity** - это просто storage-класс, для полей которого с помощью **аннотаций** прописаны БД свойства (тип, связи, имя колонки в БД, допустимые значения и т.д.).

1. Конфигурационный файл Hibernate?

Hibernate требует заранее знать – где найти информацию о сопоставлении, которая определяет, как ваши классы Java связаны с таблицами базы данных. Hibernate также требует набора параметров конфигурации, связанных с базой данных и других связанных параметров. Вся такая информация обычно предоставляется в виде стандартного файла свойств Java с именем hibernate.properties или в виде файла XML с именем hibernate.cfg.xml.

Ниже приведен список важных свойств, вам необходимо будет настроить для баз данных в автономной ситуации:

hibernate.dialect - Это свойство заставляет Hibernate генерировать соответствующий SQL для выбранной базы данных

hibernate.connection.driver\_class - Класс драйвера JDBC.

hibernate.connection.url - URL JDBC для экземпляра базы данных.

hibernate.connection.username - Имя пользователя базы данных.

hibernate.connection.password - Пароль базы данных.

hibernate.connection.pool\_size - Ограничивает количество соединений, ожидающих в пуле соединений базы данных Hibernate.

hibernate.connection.autocommit - Позволяет использовать режим автоматической фиксации для соединения JDBC.

1. Аннотация @Autowired?

**@Autowired**. Принцип её работы очень прост.

Допустим у нас есть bean-зависимости:

@Configuration

public class Config {

// тип бина - ServiceDependency, имя - dependency

public ServiceDependency dependency() {

return new ServiceDependencyImpl()

}

}

И есть класс сервиса:

@Service

public class MyService {

private ServiceDependency dependency;

@Autowired

public MyService(ServiceDependency dependency) {

this.dependency = dependency;

}

}

И при создании контекста Spring автоматически определит, что для создания **MyService** требуется bean типа **ServiceDependency** (или наследник), найдёт его у себя, в рамках подставит зависимость **ServiceDependencyImpl** в bean **MyService**.

На самом деле, начиная со Spring 4.0 аннотацию ***@Autowired*** можно не ставить на конструктор, если он единственный в классе.

## Другие варианты использования аннотации

**@Autowired** можно ставить непосредственно на поле. Да-да, это будет работать и с private-полями:

@Service

public class MyService {

@Autowired

private ServiceDependency dependency;

// бизнес-методы

}

Также аннотацию можно ставить на сеттеры:

@Service

public class MyService {

private ServiceDependency dependency;

@Autowired

public void setDependency(ServiceDependency dependency) {

this.dependency = dependency;

}

}

Но можно так же ставить и на отдельные методы, например:

@Service

public class MyService {

private ServiceDependency dependency;

@Autowired

public void init(ServiceDependency dependency) {

this.dependency = dependency;

}

}

## Что ещё?

Предположим, что бинов типа **ServiceDependency** несколько (допустим dependency1 и dependency2). Тогда, чтобы задать конкретный bean, необходимо использовать аннотацию **@Qualifier**:

@Service

public class MyService {

private ServiceDependency dependency;

@Autowired

public void setDependency(

@Qualifier("dependency1") ServiceDependency dependency

) {

this.dependency = dependency;

}

}

## А если мы захотим использовать все бины?

Сделать это можно простым способом:

@Service

public class MyService {

private ServiceDependency[] dependencies;

@Autowired

public void setDependency(ServiceDependency[] dependencies) {

this.dependency = dependency;

}

}

И Spring вставит (удивительно) все бины, реализующие интерфейс **ServiceDependency**. То же самое верно и для типизированных коллекций. Как ни странно, но порядком следования в этой коллекции можно управлять с помощью аннотации **@Order**.

## Но самое замечательное, вот это:

@Autowired

private Map<String, StringCallable> map;

Spring автоматически подставит не только сами бины, но и имена бинов в качестве ключей. Это может быть эффективно использовано, например, в паттерне «стратегия».

1. Что такое JPQL (Java Persistence query language)?

**Java Persistence Query Language** (**JPQL**) - язык запросов, определенный JPA. JPQL подобен SQL, но воздействует на объекты, атрибуты и отношения вместо таблиц и столбцов. JPQL может использоваться для чтения (SELECT), а также массовых обновлений (UPDATE) и удалений (DELETE). JPQL может использоваться в NamedQuery (через аннотации или XML) или в динамических запросах, используя EntityManager createQuery() API.

1. Интерфейс JpaRepository?

JpaRepository – это интерфейс фреймворка Spring Data предоставляющий набор стандартных методов JPA для работы с БД.

1. Аннотация @Id, @GeneratedValue?

Аннотация @GeneratedValue для JPA. В JPA значение примечания @GeneratedValue главным образом заключается в создании уникально идентифицированного первичного ключа для объекта (JPA требует, чтобы каждый объект должен иметь один и только один первичный ключ). @GeneratedValue обеспечивает генерацию первичного ключа. стратегия. Аннотация @GeneratedValue имеет два атрибута, а именно, стратегию и генератор, где значением атрибута генератора является строка со значением по умолчанию "", которое объявляет имя генератора первичного ключа (соответствует генераторам первичного ключа с тем же именем @SequenceGenerator и @TableGenerator ).

С помощью аннотации @Id мы указываем первичный ключ (Primary Key) данного класса

1. Как проекте Maven подключить JPA?

Вначале создаем пустой проект maven.

Затем необходимо настроить pom.xml.

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>ru.javastudy</groupId>

<artifactId>jpa\_quickStart</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<hibernate-version>5.0.1.Final</hibernate-version>

<spring-framework-version>4.1.5.RELEASE</spring-framework-version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-framework-bom</artifactId>

<version>${spring-framework-version}</version>

<scope>import</scope>

<type>pom</type>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!--driver for connection to MYSql database -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.34</version>

</dependency>

<!-- Hibernate -->

<!-- for JPA, use hibernate-entitymanager instead of hibernate-core. Support @PersistenceContext -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>${hibernate-version}</version>

</dependency>

<!-- Spring core and orm -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

</dependency>

<!--support orm classes-->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

</dependency>

<!--support @Service, @Repository -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

</dependencies>

</project>

Добавление поддержки фреймворков Hibernate, JPA. Нажимаем на название проекта правой кнопкой и выбираем Add Framework support.Отмечаем галочкой Spring, JavaEE Persistence и Hibernate. Если зависимости подгрузились, то будет предложено выбрать библиотеку maven. После этого в проект добавятся необходимые xml — META-INF\persistence.xml и sping-config.xml.

Настройка spring-config.xml.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd" >

<!--@Transaction annotation support -->

<tx:annotation-driven transaction-manager="transactionManager"/>

<!--Scanning components in base-package (look for annotations) -->

<context:component-scan base-package="ru.javastudy"/>

<!--Activates various annotations to be detected in bean classes: Spring's @Required and @Autowired and so on-->

<context:annotation-config/>

<!--Обеспечивает работу с транзакциями в Spring -->

<bean id="transactionManager"

class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="emf"/>

</bean>

<!-- Datasource. Источник данных - база MySQL -->

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/javastudy" />

<property name="username" value="root" />

<property name="password" value="admin" />

</bean>

<!-- EntityManagerFactory -->

<bean id="emf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="dataSource" ref="dataSource" />

<!--Поставщик данных - hibernate-->

<property name="jpaVendorAdapter">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />

</property>

<!--поиск сущностей в этом пакете-->

<property name="packagesToScan"

value="ru.javastudy"/>

<!--детали конфигурации поставщика постоянства (hibernate) -->

<property name="jpaProperties">

<props>

<prop key="hibernate.dialect">

org.hibernate.dialect.H2Dialect

</prop>

<prop key="hibernate.max\_fetch\_depth">3</prop>

<prop key="hibernate.jdbc.fetch\_size">50</prop>

<prop key="hibernate.jdbc.batch\_size">10</prop>

<prop key="hibernate.show\_sql">true</prop>

</props>

</property>

</bean>

</beans>

1. Методы JpaRepository.

<S extends T> S save(S entity); – сохраняет переданную сущность.

Optional<T> findById(ID primaryKey); – возвращает сущность, идентифицированную данным ID.

Iterable<T> findAll(); – возвращает все сущности.

long count(); – возвращает количество сущностей.

void delete(T entity); – удаляет переданную сущность.

boolean existsById(ID primaryKey); – указывает, существует ли сущность с данным ID.

flush() – производит запись сохраненных данных непосредственно в БД.