教程：

JPA学习笔记1——JPA基础： <https://blog.csdn.net/chjttony/article/details/6086298>

JPA学习笔记2——JPA高级： <https://blog.csdn.net/chjttony/article/details/6086305>

案例：

<https://gitee.com/zx19890628/spring-boot-example/tree/master/lab_011_db_jpa>

# spring data jpa

## JPA是什么？

JPA(Java Persistence API)是Sun官方提出的Java持久化规范。它为Java开发人员提供了一种对象/关联映射工具来管理Java应用中的关系数据。

他的出现主要是为了简化现有的持久化开发工作和整合ORM技术，结束现在Hibernate，TopLink，JDO等ORM框架各自为营的局面。值得注意的是，

JPA是在充分吸收了现有Hibernate，TopLink，JDO等ORM框架的基础上发展而来的，具有易于使用，伸缩性强等优点。从目前的开发社区的反应上看，

JPA受到了极大的支持和赞扬，其中就包括了Spring与EJB3.0的开发团队。

注意:JPA是一套规范，不是一套产品，那么像Hibernate,TopLink,JDO他们是一套产品，如果说这些产品实现了这个JPA规范，那么我们就可以叫他们为JPA的实现产品。

## **s**pring data jpa

Spring Data JPA 是 Spring 基于 ORM 框架、JPA 规范的基础上封装的一套JPA应用框架，可使开发者用极简的代码即可实现对数据的访问和操作。

它提供了包括增删改查等在内的常用功能，且易于扩展！学习并使用 Spring Data JPA 可以极大提高开发效率！

spring data jpa让我们解脱了DAO层的操作，基本上所有CRUD都可以依赖于它来实现

## 配置POM

<!-- jpa -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

## 配置数据源

spring.jpa.properties.hibernate.hbm2ddl.auto

create：先删除，再创建

update：如果表不存在就创建，不一样就更新，一样就什么都不做。

create-drop：初始化时创建表，SessionFactory执行close()时删除表。

validate：验证表结构是否一致，如果不一致，就抛异常。

spring.datasource.url=jdbc:mysql://localhost:3306/user?characterEncoding=UTF8

spring.datasource.username=root

spring.datasource.password=4100107223

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.properties.hibernate.hbm2ddl.auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.show-sql= true

## 编写dao

public interface UserDao extends JpaRepository<User, Long> {

//自定义sql执行

@Query("from User where userName=:userName")

User findByUserName(@Param("userName") String userName);

//jpa 自定义sql生成执行

List<User> findByUserNameOrEmail(String userName, String email);

Page<User> findAll(Pageable pageable);

@Transactional(timeout = 10)// 对事物的支持,查询超时的设置等

@Modifying // 根据需要添加

@Query("update User set userName = ?1 where id = ?2")

int modifyByIdAndUserId(String userName, Long id);

}

## 测试功能

@RunWith(SpringRunner.class)

@SpringBootTest

public class UserDaoTest {

@Autowired

private UserDao userDao;

@Test

public void test() throws Exception {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String date = sdf.format(new Date());

userDao.save(new User("张三", "23333", "zs@163.com", "zs", date));

userDao.save(new User("李四", "23333", "zs@163.com", "ls", date));

userDao.save(new User("王五", "23333", "zs@163.com", "ww", date));

userDao.save(new User("赵六", "23333", "zs@163.com", "zl", date));

List<User> findAll = userDao.findAll();

System.out.println(findAll.size());

User user = userDao.findByUserName("张三");

System.out.println(user);

List<User> findByUserNameOrEmail = userDao.findByUserNameOrEmail("赵六", "zs@163.com");

System.out.println(findByUserNameOrEmail.size());

}

@Test

public void testPageQuery() throws Exception {

int page = 1;

int size = 2;

Sort sort = new Sort(Direction.DESC, "id");

Pageable pageable = new PageRequest(page, size, sort);

Page<User> all = userDao.findAll(pageable);

for (User user : all) {

System.out.println(user);

}

}

@Test

public void testUpdate() throws Exception {

int i = this.userDao.modifyByIdAndUserId("张珊三", 1l);

System.out.println(i);

}

}

## 基本查询

基本查询也分为两种，一种是spring data默认已经实现，一种是根据查询的方法来自动解析成SQL。

### 预先生成方法

spring data jpa 默认预先生成了一些基本的CURD的方法，例如：增、删、改等等

1.继承JpaRepository

public interface UserRepository extends JpaRepository<User, Long> {

}

2.使用默认方法

@Test

public void testBaseQuery() throws Exception {

User user=new User();

userRepository.findAll();

userRepository.findOne(1l);

userRepository.save(user);

userRepository.delete(user);

userRepository.count();

userRepository.exists(1l);

// ...

}

### 自定义简单查询

自定义的简单查询就是根据方法名来自动生成SQL，主要的语法是

findXXBy,readAXXBy,queryXXBy,countXXBy, getXXBy后面跟属性名称：

User findByUserName(String userName);

也使用一些加一些关键字And、 Or

User findByUserNameOrEmail(String username, String email);

修改、删除、统计也是类似语法

Long deleteById(Long id);

Long countByUserName(String userName)

基本上SQL体系中的关键词都可以使用，例如：LIKE、 IgnoreCase、 OrderBy。

List<User> findByEmailLike(String email);

User findByUserNameIgnoreCase(String userName);

List<User> findByUserNameOrderByEmailDesc(String email);

具体的关键字，使用方法和生产成SQL如下表所示

Keyword Sample JPQL snippet

And findByLastnameAndFirstname … where x.lastname = ?1 and x.firstname = ?2

Or findByLastnameOrFirstname … where x.lastname = ?1 or x.firstname = ?2

Is,Equals findByFirstnameIs,findByFirstnameEquals … where x.firstname = ?1

Between findByStartDateBetween … where x.startDate between ?1 and ?2

LessThan findByAgeLessThan … where x.age < ?1

LessThanEqual findByAgeLessThanEqual … where x.age ⇐ ?1

GreaterThan findByAgeGreaterThan … where x.age > ?1

GreaterThanEqual findByAgeGreaterThanEqual … where x.age >= ?1

After findByStartDateAfter … where x.startDate > ?1

Before findByStartDateBefore … where x.startDate < ?1

IsNull findByAgeIsNull … where x.age is null

IsNotNull,NotNull findByAge(Is)NotNull … where x.age not null

Like findByFirstnameLike … where x.firstname like ?1

NotLike findByFirstnameNotLike … where x.firstname not like ?1

StartingWith findByFirstnameStartingWith … where x.firstname like ?1 (parameter bound with appended %)

EndingWith findByFirstnameEndingWith … where x.firstname like ?1 (parameter bound with prepended %)

Containing findByFirstnameContaining … where x.firstname like ?1 (parameter bound wrapped in %)

OrderBy findByAgeOrderByLastnameDesc … where x.age = ?1 order by x.lastname desc

Not findByLastnameNot … where x.lastname <> ?1

In findByAgeIn(Collection ages) … where x.age in ?1

NotIn findByAgeNotIn(Collection age) … where x.age not in ?1

TRUE findByActiveTrue() … where x.active = true

FALSE findByActiveFalse() … where x.active = false

IgnoreCase findByFirstnameIgnoreCase … where UPPER(x.firstame) = UPPER(?1)

### 复杂查询

在实际的开发中我们需要用到分页、删选、连表等查询的时候就需要特殊的方法或者自定义SQL

1.分页查询

分页查询在实际使用中非常普遍了，spring data jpa已经帮我们实现了分页的功能，在查询的方法中，需要传入参数Pageable,当查询中有多个参数的时候Pageable建议做为最后一个参数传入

Page<User> findALL(Pageable pageable);

Page<User> findByUserName(String userName,Pageable pageable);

Pageable 是spring封装的分页实现类，使用的时候需要传入页数、每页条数和排序规则

@Test

public void testPageQuery() throws Exception {

int page=1,size=10;

Sort sort = new Sort(Direction.DESC, "id");

Pageable pageable = new PageRequest(page, size, sort);

userRepository.findALL(pageable);

userRepository.findByUserName("testName", pageable);

}

2.限制查询

有时候我们只需要查询前N个元素，或者支取前一个实体。

ser findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);

3.自定义SQL查询

其实Spring data 觉大部分的SQL都可以根据方法名定义的方式来实现，但是由于某些原因我们想使用自定义的SQL来查询，spring data也是完美支持的；在SQL的查询方法上面使用@Query注解，如涉及到删除和修改在需要加上@Modifying.也可以根据需要添加 @Transactional 对事物的支持，查询超时的设置等

@Modifying

@Query("update User u set u.userName = ?1 where c.id = ?2")

int modifyByIdAndUserId(String userName, Long id);

@Transactional

@Modifying

@Query("delete from User where id = ?1")

void deleteByUserId(Long id);

@Transactional(timeout = 10)

@Query("select u from User u where u.emailAddress = ?1")

User findByEmailAddress(String emailAddress);

### 多表查询

1.多表查询在spring data jpa中有两种实现方式，第一种是利用hibernate的级联查询来实现，第二种是创建一个结果集的接口来接收连表查询后的结果，这里主要第二种方式。

首先需要定义一个结果集的接口类。

public interface HotelSummary {

City getCity();

String getName();

Double getAverageRating();

default Integer getAverageRatingRounded() {

return getAverageRating() == null ? null : (int) Math.round(getAverageRating());

}

}

查询的方法返回类型设置为新创建的接口

@Query("select h.city as city, h.name as name, avg(r.rating) as averageRating " "from Hotel h left outer join h.reviews r where h.city = ?1 group by h")

Page<HotelSummary> findByCity(City city, Pageable pageable);

@Query("select h.name as name, avg(r.rating) as averageRating "- "from Hotel h left outer join h.reviews r group by h")

Page<HotelSummary> findByCity(Pageable pageable);

使用

Page<HotelSummary> hotels = this.hotelRepository.findByCity(new PageRequest(0, 10, Direction.ASC, "name"));

for(HotelSummary summay:hotels){

System.out.println("Name" +summay.getName());

}

在运行中Spring会给接口（HotelSummary）自动生产一个代理类来接收返回的结果，代码汇总使用getXX的形式来获取

## 多数据源的支持

同源数据库的多源支持，总的来说，spring data jpa不是非常适合使用配置对个数据库

日常项目中因为使用的分布式开发模式，不同的服务有不同的数据源，常常需要在一个项目中使用多个数据源，因此需要配置sping data jpa对多数据源的使用，一般分一下为三步：

### 配置多个数据库application.properties

spring.datasource.primary.url=jdbc:mysql://localhost:3306/test1

spring.datasource.primary.username=root

spring.datasource.primary.password=root

spring.datasource.primary.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.secondary.url=jdbc:mysql://localhost:3306/test2

spring.datasource.secondary.username=root

spring.datasource.secondary.password=root

spring.datasource.secondary.driver-class-name=com.mysql.jdbc.Driver

### 创建配置类

@Configuration

public class DataSourceConfig {

@Bean(name = "primaryDataSource")

@Qualifier("primaryDataSource")

@ConfigurationProperties(prefix="spring.datasource.primary")

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "secondaryDataSource")

@Qualifier("secondaryDataSource")

@Primary

@ConfigurationProperties(prefix="spring.datasource.secondary")

public DataSource secondaryDataSource() {

return DataSourceBuilder.create().build();

}

}

### jdbcTemplate使用

@Bean(name = "primaryJdbcTemplate")

public JdbcTemplate primaryJdbcTemplate(

@Qualifier("primaryDataSource") DataSource dataSource) {

return new JdbcTemplate(dataSource);

}

@Bean(name = "secondaryJdbcTemplate")

public JdbcTemplate secondaryJdbcTemplate(

@Qualifier("secondaryDataSource") DataSource dataSource) {

return new JdbcTemplate(dataSource);

}

### spring data jpa使用

@Configuration

@EnableTransactionManagement

@EnableJpaRepositories(

entityManagerFactoryRef="entityManagerFactoryPrimary",

transactionManagerRef="transactionManagerPrimary",

basePackages= { "com.didispace.domain.p" }) //设置Repository所在位置

public class PrimaryConfig {

@Autowired @Qualifier("primaryDataSource")

private DataSource primaryDataSource;

@Primary

@Bean(name = "entityManagerPrimary")

public EntityManager entityManager(EntityManagerFactoryBuilder builder) {

return entityManagerFactoryPrimary(builder).getObject().createEntityManager();

}

@Primary

@Bean(name = "entityManagerFactoryPrimary")

public LocalContainerEntityManagerFactoryBean entityManagerFactoryPrimary (EntityManagerFactoryBuilder builder) {

return builder

.dataSource(primaryDataSource)

.properties(getVendorProperties(primaryDataSource))

.packages("com.didispace.domain.p") //设置实体类所在位置

.persistenceUnit("primaryPersistenceUnit")

.build();

}

@Autowired

private JpaProperties jpaProperties;

private Map<String, String> getVendorProperties(DataSource dataSource) {

return jpaProperties.getHibernateProperties(dataSource);

}

@Primary

@Bean(name = "transactionManagerPrimary")

public PlatformTransactionManager transactionManagerPrimary(EntityManagerFactoryBuilder builder) {

return new JpaTransactionManager(entityManagerFactoryPrimary(builder).getObject());

}

}

## 异构数据库多源支持

比如我们的项目中，即需要对mysql的支持，也需要对mongodb的查询等。

实体类声明@Entity 关系型数据库支持类型、声明@Document 为mongodb支持类型，不同的数据源使用不同的实体就可以了

interface PersonRepository extends Repository<Person, Long> {

…

}

@Entity

public class Person {

…

}

interface UserRepository extends Repository<User, Long> {

…

}

@Document

public class User {

…

}

但是，如果User用户既使用mysql也使用mongodb呢，也可以做混合使用

interface JpaPersonRepository extends Repository<Person, Long> {

…

}

interface MongoDBPersonRepository extends Repository<Person, Long> {

…

}

@Entity

@Document

public class Person {

…

}

也可以通过对不同的包路径进行声明，比如A包路径下使用mysql,B包路径下使用mongoDB

@EnableJpaRepositories(basePackages = "com.neo.repositories.jpa")

@EnableMongoRepositories(basePackages = "com.neo.repositories.mongo")

interface Configuration { }

## 其它

### 使用枚举

使用枚举的时候，我们希望数据库中存储的是枚举对应的String类型，而不是枚举的索引值，需要在属性上面添加@Enumerated(EnumType.STRING) 注解

@Enumerated(EnumType.STRING)

@Column(nullable = true)

private UserType type;

### 不需要和数据库映射的属性

正常情况下我们在实体类上加入注解@Entity，就会让实体类和表相关连如果其中某个属性我们不需要和数据库来关联只是在展示的时候做计算，只需要加上@Transient属性既可。

@Transient

private String userName;

## 初始化数据

我们在做测试的时候经常需要初始化导入一些数据，如何来处理呢？

会有两种选择，一种是使用Jpa，另外一种是Spring JDBC。两种方式各有区别下面来详细介绍。

### 使用Jpa

在使用spring boot时，如果jpa设置spring.jpa.hibernate.ddl-auto的属性设置为create or create-drop的时候，spring boot 启动时默认会扫描classpath下面（项目中一般是resources目录）是否有import.sql，如果有机会执行import.sql脚本。

### 使用Spring JDBC

使用Spring JDBC 需要在配置文件中添加以下配置

spring:

datasource:

schema: database/data.sql

sql-script-encoding: utf-8

jpa:

hibernate:

ddl-auto: none

schema ：设置脚本的路径

sql-script-encoding：设置脚本的编码

spring boot项目启动的时候会自动执行脚本。

### 不同点

第一种方式启动的时候Jpa会自动创建表，import.sql只负责创建表单后的初始化数据。

第二种方式启动的时候不会创建表，需要在初始化脚本中加班判断表是否存在，不存在创建表再初始化脚本的步骤。

在生产中，这两种模式都建议慎用！

### ddl-auto 四个值的解释

create： 每次加载hibernate时都会删除上一次的生成的表，然后根据你的model类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。

create-drop ：每次加载hibernate时根据model类生成表，但是sessionFactory一关闭,表就自动删除。

update：最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据 model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会。

validate ：每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。

none : 什么都不做。