匠人学院
jiangren.com.au

Java Spring Boot
——Spring Data JPA

什么是ORM？

- 对象关系映射（Object Relational Mapping，简称 ORM）模式是一种为了解决面向对象与关系数据库存在的互不匹配的现象技术。简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系数据库中
- 当开发一个应用程序的时候（不使用 O/R Mapping），你可能会写不少数据访问层的代码，用来从数据库保存、删除、读取对象信息等。在 DAL 中写了很多的方法来读取对象数据、改变状态对象等任务，而这些代码写起来总是重复的。 针对这些问题 ORM 提供了解决方案，简化了将程序中的对象持久化到关系数据库中的操作。
- ORM 框架的本质是简化编程中操作数据库的编码，在 Java 江湖中发展到现在基本上就剩两家，一个是宣称可以不用写一句 SQL 的 Hibernate，一个是以动态 SQL 见长的 MyBatis，两者各有特点。在企业级系统开发中可以根据需求灵活使用，发现一个有趣的现象：传统企业大都喜欢使用 Hibernate，而互联网行业通常使用 MyBatis。

什么是JPA

- 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，它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，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。

Spring 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。

快速上手

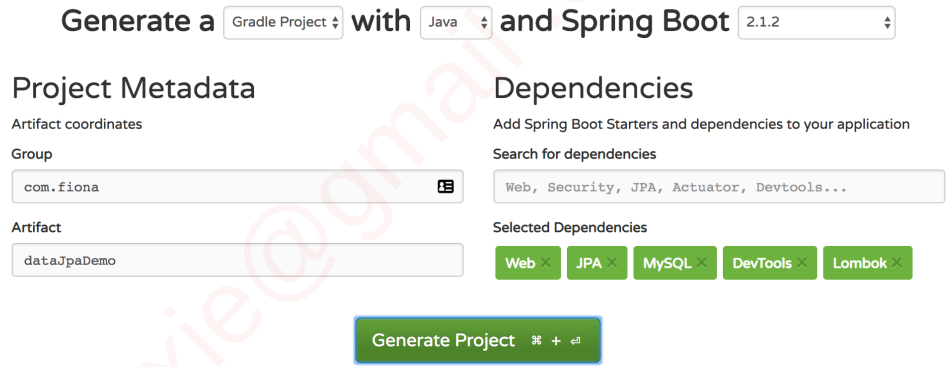
- 添加依赖：

```
compile('org.springframework.boot:spring-boot-starter-data-jpa')
compile('mysql:mysql-connector-java')
compile('org.apache.commons:commons-lang3:3.1')
```

- 添加配置文件：

```
Spring.datasource.url=jdbc:mysql://localhost:3306/test
Spring.datasource.username=root
Spring.datasource.password=root
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
```



快速上手

添加配置文件： hibernate.hbm2ddl.auto 参数的作用主要用于：自动创建|更新|验证数据库表结构，有四个值：

- ✓ 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。

dialect 主要是指定生成表名的存储引擎为 InnoDB；

show-sql 是否打印出自动生产的 SQL，方便调试的时候查看。

快速上手

```
@Entity
@Data
public class User {

  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue
  private Long id;
  @Column(nullable = false, unique = true)
  private String userName;
  @Column(nullable = false)
  private String passWord;
  @Column(nullable = false, unique = true)
  private String email;
  @Column(nullable = true, unique = true)
  private String nickName;
  @Column(nullable = false)
  private String regTime;

}
```

dao 只要继承 JpaRepository 类就可以，几乎可以不用写方法，还有可以根据方法名来自动的生产 SQL，比如 findByUserName 会自动生产一个以 userName 为参数的查询方法，比如 findAll 自动会查询表里面的所有数据，比如自动分页等。

```
package com.fiona.datajpademo.repository;

import com.fiona.datajpademo.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
  User findByUserName(String userName);
  User findByUserNameOrEmail(String username, String email);
}
```

Entity 中不映射成列的字段得加 @Transient 注解，不加注解也会映射成列。

测试一下

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserRepositoryTest {

  @Resource
  private UserRepository userRepository;

  @Test
  public void test() {
    Date date = new Date();
    DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG);
    String formattedDate = dateFormat.format(date);

    userRepository.save(new User("aa", "aa@gmail.com", "aa", "aa123456", formattedDate));
    userRepository.save(new User("bb", "bb@gmail.com", "bb", "bb123456", formattedDate));
    userRepository.save(new User("cc", "cc@gmail.com", "cc", "cc123456", formattedDate));

    Assert.assertEquals(3, userRepository.findAll().size());
    Assert.assertEquals("bb", userRepository.findByUserNameOrEmail("bb",
"cc@gmail.com").getNickName());
    userRepository.delete(userRepository.findByUserName("aa1"));
  }
}
```

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。

1. 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，如增、删、改等。

（1）继承 JpaRepository

```
public interface UserRepository
extends JpaRepository<User, Long>
{

}
```

```
@Test
 public void testBaseQuery() {
    Date date = new Date();
    DateFormat dateFormat =
DateFormat.getDateTimeInstance(DateFormat.LONG,
DateFormat.LONG);
    String formattedDate = dateFormat.format(date);
    User user = new User("aa", "aa@gmail.com", "aa", "aa123456",
formattedDate)
    userRepository.findAll();
//    userRepository.findOne(11)
    userRepository.save(user);
    userRepository.delete(user);
    userRepository.count();
//    userRepository.exists(1l);
 }
```

基本查询

JpaRepository extends PagingAndSortingRepository which in turn extends CrudRepository.

Their main functions are:

- CrudRepository mainly provides CRUD functions.
- PagingAndSortingRepository provides methods to do pagination and sorting records.
- JpaRepository provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.

Because of the inheritance mentioned above, JpaRepository will have all the functions of CrudRepository and PagingAndSortingRepository. So if you don't need the repository to have the functions provided by JpaRepository and PagingAndSortingRepository , use CrudRepository.

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。也可以加一些关键字 And、Or

修改、删除、统计也是类似语法：

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

```
List<User> findByEmailLike(String email);
User findByUserNameIgnoreCase(String userName);
List<User> findByUserNameOrderByEmailDesc(String
email);
```

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<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> 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

在实际的开发中需要用到分页、筛选、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

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:

```
@Query("select u from User u")
Page<User> findALL(Pageable pageable);

Page<User> findByNickName(String nickName, Pageable pageable);
```

```
@Test
public void testPageQuery(){
  int page=1,size=10;
  Sort sort = new Sort(Sort.Direction.DESC, "id");
  Pageable pageable = PageRequest.of(page, size, sort);
  userRepository.findALL(pageable);
  userRepository.findByNickName("testName", pageable);
}
```

复杂查询

限制查询

有时候需要查询前 N 个元素，或者只取前一个实体。

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

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

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

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

复杂查询

匠人学院
jiangren.com.au

自定义 SQL 查询

使用 Spring Data 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 也可以完美支持；在 SQL 的查询方法上面使用 **@Query** 注解，如涉及到删除和修改需要加上 **@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。

```
@Transactional(timeout = 10)
@Modifying
@Query("update User set userName = ?1 where id = ?2")
int modifyById(String  userName, Long id);

@Transactional
@Modifying
@Query("delete from User where id = ?1")
void deleteById(Long id);

@Query("select u from User u where u.email = ?1")
User findByEmail(String email);
```

复杂查询

多表查询

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，这里主要使用第二种方式。

创建一个用户详情类：

```
@Entity
public class UserDetails {
  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue
  private Long id;
  @Column(nullable = false, unique = true)
  private String userId;
  @Column(nullable = true)
  private String address;
  @Column(nullable = true)
  private String hobby;

  // need to add getter and setter
}
```

定义一个结果集的接口类，接口类的内容来自于用户表和用户详情表。

```
public interface UserInfo {
  String getUserName();
  String getEmail();
  String getAddress();
  String getHobby();
}
```

复杂查询

多表查询

查询的方法返回类型为 UserInfo：

特别注意这里的 SQL 是 HQL，需要写类的名和属性

```
@Query("select u.userName as userName, u.email as email, d.address as address , d.hobby as hobby from
User u , UserDetail d " +
   "where u.id=d.userId  and  d.hobby = ?1 ")
List<UserInfo> findUserInfo(String hobby);
```

```
public class UserDetailsRepositoryTest {
  @Resource
  private UserDetailsRepository userDetailRepository;

  @Test
  public void testUserInfo() {
    List<UserInfo> userInfos = userDetailRepository.findUserInfo("打球");
    for (UserInfo userInfo : userInfos) {
      System.out.println("address " + userInfo.getAddress());
    }
  }
}
```

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

复杂查询

多参数

```
@Query("SELECT u FROM UserEntity u WHERE u.tenantId = :tenantId " +
   "AND u.customerId = :customerId AND u.authority = :authority " +
   "AND LOWER(u.searchText) LIKE LOWER(CONCAT(:searchText, '%'))" +
   "AND u.id > :idOffset ORDER BY u.id")
List<UserEntity> findUsersByAuthority(@Param("tenantId") String tenantId,
                                      @Param("customerId") String customerId,
                                      @Param("idOffset") String idOffset,
                                      @Param("searchText") String searchText,
                                      @Param("authority") Authority authority,
                                      Pageable pageable);
```

也可以使用Native Sql，nativeQuery=true

```
@Query(
   value = "SELECT * FROM USERS u WHERE u.status = 1",
   nativeQuery = true)
Collection<User> findAllActiveUsersNative();
```

More

使用枚举的时候，如果选择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 UserStatus status;
```

不需要和数据库映射的属性

正常情况下实体类上加入注解 @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  fullName;

@Transient
private int age;
```

连接多个数据源

在项目开发中，常常需要在一个项目中使用多个数据源，因此需要配置 Spring Data JPA 对多数据源的使用，一般分为以下三步：

- 配置多数据源
- 不同源的 repository 放入不同包路径
- 声明不同的包路径下使用不同的数据源、事务支持

application.properties

```
#primary
spring.primary.datasource.url=jdbc:mysql://localhost:3306/test1
spring.primary.datasource.username=root
spring.primary.datasource.password=root
spring.primary.datasource.driver-class-name=com.mysql.jdbc.Driver
#secondary
spring.secondary.datasource.url=jdbc:mysql://localhost:3306/test2
spring.secondary.datasource.username=root
spring.secondary.datasource.password=root
spring.secondary.datasource.driver-class-name=com.mysql.jdbc.Driver
```

```
@Configuration
public class DataSourceConfig {

    @Bean(name = "primaryDataSource")
    @Qualifier("primaryDataSource")
    @ConfigurationProperties(prefix="spring.primary.datasource")
    public DataSource primaryDataSource() {
        return DataSourceBuilder.create().build();
    }

    @Bean(name = "secondaryDataSource")
    @Qualifier("secondaryDataSource")
    @Primary
    @ConfigurationProperties(prefix="spring.secondary.datasource")
    public DataSource secondaryDataSource() {
        return DataSourceBuilder.create().build();
    }

}
```

Database Migration

为什么需要？

Tools：

Flyway vs. Liquibase

https://flywaydb.org/documentation/gradle/

https://www.liquibase.org/

SQL注入

SQL注入是比较常见的网络攻击方式之一，它不是利用操作系统的BUG来实现攻击，而是针对程序员编程时的疏忽，通过SQL语句，实现无帐号登录，甚至篡改数据库。
比如在一个登录界面，要求输入用户名和密码：
可以这样输入实现免帐号登录：

用户名： ‘or 1 = 1 --
密 码：

点登陆,如若没有做特殊处理,那么这个非法用户可以登陆进去。
从理论上说，后台认证程序中会有如下的SQL语句：

**String sql = "select * from user_table where username=
' "+userName+" ' and password=' "+password+" '";**
当输入了上面的用户名和密码，上面的SQL语句变成：
**SELECT * FROM user_table WHERE username=
''or 1 = 1 -- and password=''**

SQL注入

1.永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.永远不要使用动态拼装SQL，使用参数化的SQL或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

5.应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装，把异常信息存放在独立的表中。

匠人学院
jiangren.com.au

Q & A