Spring Data JPA学习笔记

1.简介

- SpringData：其实SpringData就是Spring提供了一个操作数据的框架。而SpringData JPA只是SpringData框架下的一个基于JPA标准操作数据的模块。
- SpringData JPA：基于JPA的标准数据进行操作。简化操作持久层的代码。只需要编写接口即可。
- JPA：JPA是Java Persistence API的简称，中文名为Java持久层API，是JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。JPA的出现主要是为了简化持久层开发以及整合ORM技术，结束Hibernate、TopLink、JDO等ORM框架各自为营的局面。JPA是在吸收现有ORM框架的基础上发展而来，易于使用，伸缩性强。总的来说，JPA包括以下3方面的技术：

（1）**ORM映射元数据**： 支持XML和注解两种元数据的形式，元数据描述对象和表之间的映射关系。

（2）**API**： 操作实体对象来执行CRUD操作。

（3）**查询语言**： 通过面向对象而非面向数据库的查询语言（`JPQL`）查询数据，避免程序的SQL语句紧密耦合。

2.Spring Boot 整合 Spring Data JPA

2.1导入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

2.2 相关配置

```yaml
spring:
  #数据库相关配置
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
    username: root
    password: mysql123
  #Spring Data JPA相关配置
  jpa:
    database: MySQL
    hibernate:
      ddl-auto: update
      show-sql: true
```

解释：

`jpa:hibernate:ddl-auto`：hibernate的配置属性，其主要作用是：自动创建、更新、验证数据库表结构。该参数的几种配置如下：

- `create`：每次运行程序时，都会根据model类重新创建表，所以是导致数据库中数据丢失的重要原因之一。
- `create-drop`：每次运行程序时根据model类创建表，待程序结束时自动清空表。
- `upadte`：每次运行程序，没有表时会创建表，如果对象发生改变会更新表结构，原有数据不会清空，只会更新（推荐使用）
- `validate`：运行程序时会校验数据与数据库的字段类型是否相同，字段不同会报错。

`jpa:hibernate:show-sql`：日志中是否显示sql语句。

2.3 实体类-User

```java
@Entity
@Table(name = "user")
@Data//提供类的get、set、equals、hashCode、canEqual、toString方法
@AllArgsConstructor//提供类的全参构造
@NoArgsConstructor//提供类的无参构造
public class User {

    @Id//主键
    @GeneratedValue(strategy = GenerationType.IDENTITY)//自增主键
    private String id;

    @Column(name = "username")
    private String username;

    @Column(name = "password")
    private String password;

    @Column(name = "email")
```

```
    private String email;

}
```

注解@GeneratedValue 解释：

作用：在JPA中,@GeneratedValue注解存在的意义主要就是为一个实体生成一个唯一标识的主键(JPA要求每一个实体Entity,必须有且只有一个主键),@GeneratedValue提供了主键的生成策略。

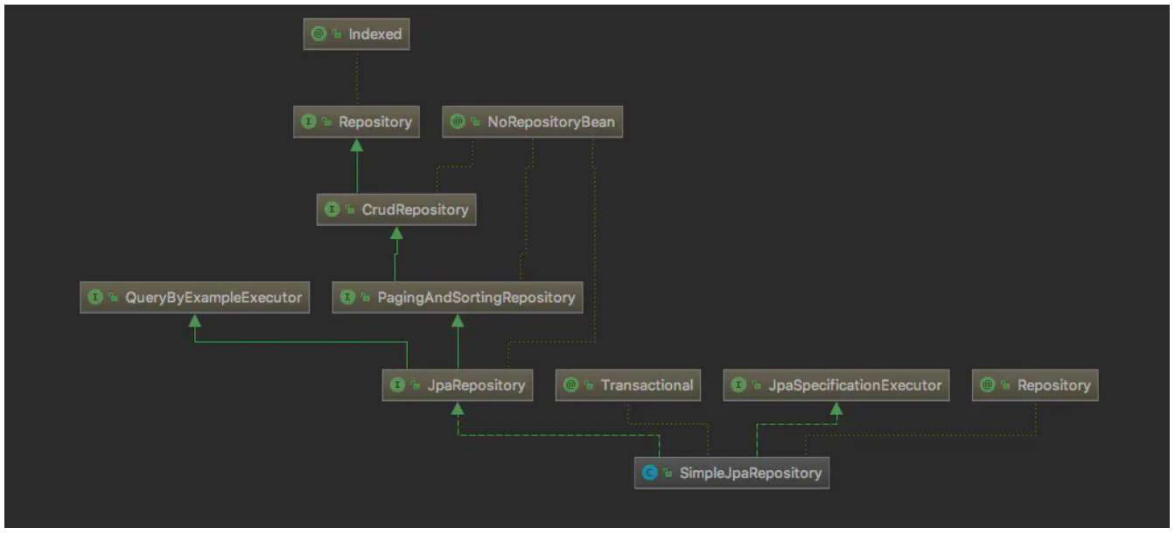
JPA提供的四种标准用法为TABLE,SEQUENCE,IDENTITY,AUTO.

```
//AUTO:主键由程序控制(默认选项)
@GeneratedValue(strategy = GenerationTye.AUTO)

//IDENTITY:主键由数据库生成, 采用数据库自增长, Oracle不支持这种方式
@GeneratedValue(strategy = GenerationTye.IDENTITY)

//SEQUENCE:通过数据库的序列产生主键, MYSQL不支持
@GeneratedValue(strategy = GenerationTye.SEQUENCE)

//Table:提供特定的数据库产生主键, 该方式更有利于数据库的移植
@GeneratedValue(strategy = GenerationTye.Table)
```

2.4 Dao层

```
/**Dao层继承JpaRepository<T,ID>
T:entity层的数据类名
ID:实体类主键的类型 */
public interface PersonRepository extends JpaRepository<Person, Long> {
    Optional<User> findByEmail(String Email);

    Optional<User> findByPhone(String phone);

    User findByUsername(String username);//根据用户名查找user

    List<User> findByUsernameIgnoreCase(String username);//忽略用户名大小写

    List<User> findByUsernameLike(String username);//模糊搜索

    User findByUsernameAndPassword(String username, String password);//And

    List<User> findByIdIn(List<String> ids);//列表查询in

    List<User> findByIdInOrderByUsername(List<String> ids);//排序

    void deleteByIdIn(List<String> ids);//列表删除

    Long countByUsernameLike(String username);//计数
}
```

接口关系类图如下：



2.4.1 方法名自动解析

Spring Data Jpa通过解析方法名创建查询，框架在进行方法名解析时，会先把方法名多余的前缀find...By, read...By, query...By, count...By以及get...By截取掉，然后对剩下部分进行解析，第一个By会被用作分隔符来指示实际查询条件的开始。 我们可以在`实体属性`上定义条件，并将它们与And和Or连接起来，从而创建大量查询。支持关键字如下:

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	findByFirstname,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	… findByFirstnameNotLike
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> ages)	… 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)

- Spring Data Jpa支持使用`first`、`top`以及`Distinct`关键字来限制查询结果，如：

```
User findFirstByUsernameOrderByUsernameAsc(String username);

List<User> findTop10ByUsername(String username, Sort sort);

List<User> findTop10ByUsername(String username, Pageable pageable);
```

- save()函数既可以新增，又可以修改：

新增：userService.save(user);

修改：先使用findById()获得user，对user进行修改，然后再save()。

2.4.2 自定义查询@Query

Spring Data JPA中除了使用内置的规则方法，也可以自己写sql语句。只需在声明的方法上面标注@Query注解，同时提供一个 JPQL 查询语句即可。

（1）基于位置的参数绑定

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

@Query("select u from User u where u.username = ?1 and u.password = ?2")
User getByUsernameAndPassword(String username, String password);

@Query("select u from User u where u.username like %?1%")
List<User> getByUsernameLike(String username);
```

（2）基于注解@Param的参数绑定

```
@Query("select u from User u where u.id = :id")
User getById(@Param("id") String userId);

@Query("select u from User u where u.username = :username or u.email = :email")
User getByUsernameOrEmail(@Param("username") String username, @Param("email")
String email);
```

（3）原生查询

原生查询指涉及到的表名和属性名为数据库中的真实表名和属性名，而非原生查询，查询语句中的表名则是对应的项目中实体类的类名。

```
@Query(value = "select * from tb_user u where u.email = ?1", nativeQuery = true)
User queryByEmail(String email);

@Query(value = "select * from tb_user u where u.email = :email", nativeQuery =
true)
User queryByEmail(@Param("email") String email);
```

（4）自定义修改、删除操作

单独使用@Query注解只是查询，如涉及到修改、删除则需要再加上`@Modifying`注解，如：

```
@Transactional
@Modifying
@Query("update User u set u.password = ?2 where u.username = ?1")
int updatePasswordByUsername(String username, String password);

@Transactional
@Modifying
@Query("delete from User where username = ?1")
void deleteByUsername(String username);
```

2.5 Service层

Service层负责实现主要的业务逻辑。

UserService接口：

```
public interface UserService {

    /**用户登录*/
    public bool Login_Password(String username, String password);

}
```

UserServiceImpl：

```
@Service
public class UserServiceImpl  implements UserService {
```

```
    @Autowired
    private UserRepository userRepository;

    public bool Login_Password(String username, String password){
        Optional<User> u = userRepository.findByUserName(username);
        //Optional<T>的isPresent()可以判断查询是否得到结果
        if(u.isPresent()){
            User user = u.get();
            if(user.getPassword.equals(password))
                return true;
            else
                return false;
        }
        else
            return false;

    }

}
```

2.6 Controller层

```
@RestController
@RequestMapping("/api/user")
@Slf4j
public class UserController {

    @Autowired
    private UserService userService;

    private JSONObject jsonObject;

    //用户登录接口
    @PostMapping("/login")
    public ResponseUtils Login(@RequestBody JSONObject jsonObject) {

        String username = jsonObject.getString("username");
        String password = jsonObject.getString("password");

        if(userService.Login(username,password))
            return ResponseUtils.response(200,"用户登录成功", jsonObject);
        else
            return ResponseUtils.response(200,"用户登录失败", jsonObject);
    }
}
```

此时便可以测试 http://localhost:8080/api/user/login 接口是否正确。