<http://blog.csdn.net/tyyytcj/article/details/76281836>

Spring Data是Spring提供的对数据库的访问方式而Spring Data JPA是Spring Data的一部分，是JPA的Spring 实现方案。通过Spring Data JPA可以很方便的利用spring-framew构建基于数据库的应用程序。   
Spring Data JPA是JPA的spring实现因此包含了很多详细的应用方式，特别是各种类型的查询检索方式，本文只是简单的涉及数据库的增删改查也就是CRUD操作。

代码环境：maven+spring-boot

首先需要在pom.xml引入依赖项

<dependency>

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

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

</dependency>

* 1
* 2
* 3
* 4

至于数据库的配置就不再过多的赘述可参见我前面的文章<http://blog.csdn.net/tyyytcj/article/details/76100682>

## 实体映射

使用jpa首先需要建立实体类与数据库之间的映射关系

@Entity

@Table(name = "SECONDLEVEL", schema = "MATERIAL", catalog = "")

public class SecondlevelEntity {

private String code;

private String name;

private String sLinks;//操作链接

private String dataIntegrity;

private int dataIdCount;

@Id

@Column(name = "CODE", nullable = false, length = 10)

public String getCode() {

return code;映射

}

public void setCode(String code) {

this.code = code;

}

@Basic

@Column(name = "NAME", nullable = false, length = 30)

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Transient

public String getsLinks() {

return "<a href='javascript:void(0);' onclick='material\_UpdateLb(\"" + this.code + "\");'>编辑</a>" +

"&nbsp;&nbsp;<a href='javascript:void(0);' onclick='material\_DeleteLb(\"" + this.code + "\");'>删除</a>";

}

public void setsLinks(String sLinks) {

this.sLinks = sLinks;

}

@Transient

public String getDataIntegrity() {

return dataIntegrity;

}

public void setDataIntegrity(String dataIntegrity) {

this.dataIntegrity = dataIntegrity;

}

@Transient

public int getDataIdCount() {

return dataIdCount;

}

public void setDataIdCount(int dataIdCount) {

this.dataIdCount = dataIdCount;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46

@Entity 注解在数据源配置的时候由spring自动进行扫描并创建相应的实体映射   
@Table 注解用于标明与数据库对应的表名数据库   
@Id 注解说明该属性作为该表的主键   
@Column 注解说明该属性对应的列名   
@Basic 表示一个简单的属性到数据库表的字段的映射   
@Transient 表示注解的该属性并非与该表中的任何一个字段进行映射

# 构建Repository

完成实体映射后将构建DAO接口，spring data jpa采用接口的方式声明相应的数据库操作方法并且提供了部分基础的数据库操作方法包括save、findOne、findAll、update、delete等方法，通过查询方法还提供了分页查询的功能

//spring data jpa 提供了多个接口并封装了部分数据库操作方法，具体可参见下面代码

List<T> findAll();

List<T> findAll(Sort sort);

List<T> findAll(Iterable<ID> iterable);

<S extends T> List<S> save(Iterable<S> iterable);

void flush();

<S extends T> S saveAndFlush(S s);

void deleteInBatch(Iterable<T> iterable);

void deleteAllInBatch();

T getOne(ID id);

<S extends T> List<S> findAll(Example<S> example);

<S extends T> List<S> findAll(Example<S> example, Sort sort);

Page<T> findAll(Pageable pageable);

<S extends T> S save(S s);

T findOne(ID id);

boolean exists(ID id);

long count();

void delete(ID id);

void delete(T t);

void delete(Iterable<? extends T> iterable);

void deleteAll();

<S extends T> S findOne(Example<S> example);

<S extends T> Page<S> findAll(Example<S> example, Pageable pageable);

<S extends T> long count(Example<S> example);

<S extends T> boolean exists(Example<S> example);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

结合自己的业务需求生成自己的Repository并集成框架本身提供的接口就能拥有框架本身提供的数据库访问功能，针对具体的应用情况声明了特定的查询方法，针对各种简单查询复杂查询的描述请参见spring data jpa官方手册<http://docs.spring.io/spring-data/jpa/docs/1.11.4.RELEASE/reference/html/>

@Repository

public interface SecondlevelDao extends MyBaseRepository<SecondlevelEntity, String> {

/\*\*

\* 分页获取信息

\* @param pageable

\* @return

\*/

@Override

Page<SecondlevelEntity> findAll(Pageable pageable);

/\*\*

\* 分页获取信息，并添加特定的查询参数条件

\* @param code

\* @param name

\* @param pageable

\* @return

\*/

Page<SecondlevelEntity> findByCodeOrName(String code, String name, Pageable pageable);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

## 增/删/改/查

增、删、改、查都是在业务类中完成具体操作，下面贴出业务类代码并进行描述

@Service

@Transactional(readOnly = true)

@CacheConfig

public class SecondlevelServiceImpl implements SecondlevelService {

@Autowired

private SecondlevelDao secondlevelDao;

private final Logger logger = LoggerFactory.getLogger(this.getClass());

/\*\*

\* 不含查询条件的分页获取数据

\*

\* @param pageNo

\* @param pageSize

\* @return

\*/

@Override

@Cacheable(value = "secondlevels", key = "#a0")

public Page<SecondlevelEntity> findAll(int pageNo, int pageSize) throws Exception {

try {

return secondlevelDao.findAll(new PageRequest(pageNo, pageSize, new Sort(Sort.Direction.ASC, "code")));

} catch (Exception e) {

logger.error(e.getLocalizedMessage());

return null;

}

}

/\*\*

\* 新增数据

\*

\* @param secondlevelEntity

\* @throws Exception

\*/

@Override

@Transactional

@CacheEvict(cacheNames="secondlevels",allEntries = true)

public void save(SecondlevelEntity secondlevelEntity) throws Exception {

try {

secondlevelDao.save(secondlevelEntity);

} catch (Exception e) {

logger.error(e.getLocalizedMessage());

}

}

/\*\*

\* 更新数据

\*

\* @param secondlevelEntity

\* @throws Exception

\*/

@Override

@Transactional

@Modifying

@CacheEvict(cacheNames="secondlevels",allEntries = true)

public void update(SecondlevelEntity secondlevelEntity) throws Exception {

try {

SecondlevelEntity secondlevelEntity1 = secondlevelDao.findOne(secondlevelEntity.getCode());

secondlevelEntity1.setName(secondlevelEntity.getName());

} catch (Exception e) {

logger.error(e.getLocalizedMessage());

}

}

/\*\*

\* 删除指定数据

\*

\* @param code

\* @throws Exception

\*/

@Override

@Transactional

@Modifying

@CacheEvict(cacheNames="secondlevels",allEntries = true)

public void delete(String code) throws Exception {

try {

SecondlevelEntity s=this.findOne(code);

System.out.println(s.getCode());

secondlevelDao.delete(s);

} catch (Exception e) {

logger.error(e.getLocalizedMessage());

}

}

}

@Service 注解该类为业务逻辑类由spring容器进行管理   
@Transactional(readOnly = true) 指定默认的事务类别（只读）   
@CacheConfig 配置了spring内置缓存来缓存常用查询数据

findAll() 方法一个简单的查询所有条目的方法（有分页功能）   
@Cacheable(value = “secondlevels”, key = “#a0”) 配置了缓存并指定方法第一个参数为缓存的key

save() 方法用于保存新增数据配置了多个注解   
@Transactional 配置方法的自定义事务级别   
@Modifying 定义事务为修改   
@CacheEvict(cacheNames=”secondlevels”,allEntries = true) 清除所有缓存使新增的数据能够被查到

update() 方法用于更新实体信息，spring data jpa对于数据的更新还有query update这种基于类sql的方式，这里我们不采用这种方式，我们采用的方式更简单update方法参数中包含了待更新实体的唯一标识，通过标识将原本的实体查询出来再将需要修改的属性值传递到查询中的实体就完成了数据库数据的更新十分简单   
@Transactional 配置方法的自定义事务级别   
@Modifying 定义事务为修改   
@CacheEvict(cacheNames=”secondlevels”,allEntries = true) 清除所有缓存使修改的数据得到更新

delete() 方法用于删除指定数据，由于delete方法删除的是实体，如果传递的参数不是实体那么需要先将待删除实体查询出来再进行删除

@Transactional 配置方法的自定义事务级别   
@Modifying 定义事务为修改   
@CacheEvict(cacheNames=”secondlevels”,allEntries = true) 清楚所有缓存使删除的数据更新

至此spring data jpa的简单的增、删、改、查介绍完毕！   
至于详细的查询请参考spring data jpa的官方文档

版权声明：本文为博主原创文章，未经博主允许不得转载。