Java 实体映射工具 MapStruct

<https://blog.csdn.net/zhige_me/article/details/80699784>

DO（业务实体对象），DTO（数据传输对象）。

在一个成熟的工程中，尤其是现在的分布式系统中，应用与应用之间，还有单独的应用细分模块之后，DO 一般不会让外部依赖，这时候需要在提供对外接口的模块里放 DTO 用于对象传输，也即是 DO 对象对内，DTO对象对外，DTO 可以根据业务需要变更，并不需要映射 DO 的全部属性。

这种 对象与对象之间的互相转换，就需要有一个专门用来解决转换问题的工具，毕竟每一个字段都 get/set 会很麻烦。

MapStruct 就是这样的一个属性映射工具，只需要定义一个 Mapper 接口，MapStruct 就会自动实现这个映射接口，避免了复杂繁琐的映射实现。MapStruct官网地址

工程中引入 gradle 依赖

zygl\adapters\persistence\build.gradle

dependencies {

implementation 'org.mapstruct:mapstruct:' + "${structMapVersion}"

annotationProcessor 'org.mapstruct:mapstruct-processor:' + "${structMapVersion}"

}

基本映射

这里定义两个 DO 对象 Person 和 User，其中 user 是 Person 的一个属性 ，一个 DTO 对象 PersonDTO

@NoArgsConstructor

@AllArgsConstructor

@Data

public class Person {

private Long id;

private String name;

private String email;

private Date birthday;

private User user;

}

@NoArgsConstructor

@AllArgsConstructor

@Data

public class User {

private Integer age;

}

@NoArgsConstructor

@AllArgsConstructor

@Data

public class PersonDTO {

private Long id;

private String name;

/\*\*

\* 对应 Person.user.age

\*/

private Integer age;

private String email;

/\*\*

\* 与 DO 里面的字段名称(birthDay)不一致

\*/

private Date birth;

/\*\*

\* 对 DO 里面的字段(birthDay)进行拓展,dateFormat 的形式

\*/

private String birthDateFormat;

/\*\*

\* 对 DO 里面的字段(birthDay)进行拓展,expression 的形式

\*/

private String birthExpressionFormat;

}

写一个 Mapper 接口 PersonConverter，其中两个方法，一个是单实体映射，另一个是List映射

若源对象属性与目标对象属性名字一致，会自动映射对应属性，不一样的需要指定，也可以用 format 转成自己想要的类型，也支持表达式的方式，可以看到像 id、name、email这些名词一致的我并没有指定 source-target，而birthday-birth指定了，转换格式的 birthDateFormat 加了dateFormat 或者 birthExpressionFormat 加了 expression，如果某个属性你不想映射，可以加个 ignore=true

@Mapper

public interface PersonConverter {

PersonConverter INSTANCE = Mappers.getMapper(PersonConverter.class);

@Mappings({

@Mapping(source = "birthday", target = "birth"),

@Mapping(source = "birthday", target = "birthDateFormat", dateFormat = "yyyy-MM-dd HH:mm:ss"),

@Mapping(target = "birthExpressionFormat", expression = "java(org.apache.commons.lang3.time.DateFormatUtils.format(person.getBirthday(),\"yyyy-MM-dd HH:mm:ss\"))"),

@Mapping(source = "user.age", target = "age"),

@Mapping(target = "email", ignore = true)

})

PersonDTO domain2dto(Person person);

List<PersonDTO> domain2dto(List<Person> people);

}

编译MapStruct之后，手工编译或者启动 IDE 的时候 IDE 也会帮我们编译， 会自动在 target/classes 下生成对应的实现类

手工编译命令

mvn compile

注意！！！下面这个 PersonConverterImpl 是自动生成的，不是自己写的！

public class PersonConverterImpl implements PersonConverter {

public PersonConverterImpl() {

}

public PersonDTO domain2dto(Person person) {

if (person == null) {

return null;

} else {

PersonDTO personDTO = new PersonDTO();

personDTO.setBirth(person.getBirthday());

if (person.getBirthday() != null) {

personDTO.setBirthDateFormat((new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")).format(person.getBirthday()));

}

Integer age = this.personUserAge(person);

if (age != null) {

personDTO.setAge(age);

}

personDTO.setId(person.getId());

personDTO.setName(person.getName());

personDTO.setBirthExpressionFormat(DateFormatUtils.format(person.getBirthday(), "yyyy-MM-dd HH:mm:ss"));

return personDTO;

}

}

public List<PersonDTO> domain2dto(List<Person> people) {

if (people == null) {

return null;

} else {

List<PersonDTO> list = new ArrayList(people.size());

Iterator var3 = people.iterator();

while(var3.hasNext()) {

Person person = (Person)var3.next();

list.add(this.domain2dto(person));

}

return list;

}

}

private Integer personUserAge(Person person) {

if (person == null) {

return null;

} else {

User user = person.getUser();

if (user == null) {

return null;

} else {

Integer age = user.getAge();

return age == null ? null : age;

}

}

}

}

写一个单元测试类 PersonConverterTest 测试一下，看看效果

public class PersonConverterTest {

@Test

public void test() {

Person person = new Person(1L,"zhige","zhige.me@gmail.com",new Date(),new User(1));

PersonDTO personDTO = PersonConverter.INSTANCE.domain2dto(person);

assertNotNull(personDTO);

assertEquals(personDTO.getId(), person.getId());

assertEquals(personDTO.getName(), person.getName());

assertEquals(personDTO.getBirth(), person.getBirthday());

String format = DateFormatUtils.format(personDTO.getBirth(), "yyyy-MM-dd HH:mm:ss");

assertEquals(personDTO.getBirthDateFormat(),format);

assertEquals(personDTO.getBirthExpressionFormat(),format);

List<Person> people = new ArrayList<>();

people.add(person);

List<PersonDTO> personDTOs = PersonConverter.INSTANCE.domain2dto(people);

assertNotNull(personDTOs);

}

}

多对一

MapStruct 可以将几种类型的对象映射为另外一种类型，比如将多个 DO 对象转换为 DTO

例子

两个 DO 对象 Item 和 Sku，一个 DTO 对象 SkuDTO

@NoArgsConstructor

@AllArgsConstructor

@Data

public class Item {

private Long id;

private String title;

}

@NoArgsConstructor

@AllArgsConstructor

@Data

public class Sku {

private Long id;

private String code;

private Integer price;

}

@NoArgsConstructor

@AllArgsConstructor

@Data

public class SkuDTO {

private Long skuId;

private String skuCode;

private Integer skuPrice;

private Long itemId;

private String itemName;

}

创建 ItemConverter（映射）接口，MapStruct 就会自动实现该接口

@Mapper

public interface ItemConverter {

ItemConverter INSTANCE = Mappers.getMapper(ItemConverter.class);

@Mappings({

@Mapping(source = "sku.id",target = "skuId"),

@Mapping(source = "sku.code",target = "skuCode"),

@Mapping(source = "sku.price",target = "skuPrice"),

@Mapping(source = "item.id",target = "itemId"),

@Mapping(source = "item.title",target = "itemName")

})

SkuDTO domain2dto(Item item, Sku sku);

}

创建测试类，讲 Item 和 Sku 两个 DO对象，映射成一个 DTO 对象 SkuDTO

public class ItemConverterTest {

@Test

public void test() {

Item item = new Item(1L, "iPhone X");

Sku sku = new Sku(2L, "phone12345", 1000000);

SkuDTO skuDTO = ItemConverter.INSTANCE.domain2dto(item, sku);

assertNotNull(skuDTO);

assertEquals(skuDTO.getSkuId(),sku.getId());

assertEquals(skuDTO.getSkuCode(),sku.getCode());

assertEquals(skuDTO.getSkuPrice(),sku.getPrice());

assertEquals(skuDTO.getItemId(),item.getId());

assertEquals(skuDTO.getItemName(),item.getTitle());

}

}

可以添加自定义方法

// 形式如下

default PersonDTO personToPersonDTO(Person person) {

//hand-written mapping logic

}

// 比如在 PersonConverter 里面加入如下

default Boolean convert2Bool(Integer value) {

if (value == null || value < 1) {

return Boolean.FALSE;

} else {

return Boolean.TRUE;

}

}

default Integer convert2Int(Boolean value) {

if (value == null) {

return null;

}

if (Boolean.TRUE.equals(value)) {

return 1;

}

return 0;

}

// 测试类 PersonConverterTest 加入

assertTrue(PersonConverter.INSTANCE.convert2Bool(1));

assertEquals((int)PersonConverter.INSTANCE.convert2Int(true),1);

#### 如果已经有了接收对象，更新目标对象

// 比如在 PersonConverter 里面加入如下，@InheritConfiguration 用于继承刚才的配置

@InheritConfiguration(name = "domain2dto")

void update(Person person, @MappingTarget PersonDTO personDTO);

// 测试类 PersonConverterTest 加入如下

Person person = new Person(1L,"zhige","zhige.me@gmail.com",new Date(),new User(1));

PersonDTO personDTO = PersonConverter.INSTANCE.domain2dto(person);

assertEquals("zhige", personDTO.getName());

person.setName("xiaozhi");

PersonConverter.INSTANCE.update(person, personDTO);

assertEquals("xiaozhi", personDTO.getName());

Spring 注入的方式

// 刚才一直写的例子是默认的方式

PersonConverter INSTANCE = Mappers.getMapper(PersonConverter.class);

还有一种常用的方式，是和常用的框架 Spring 结合，在 @Mapper 后面加入 componentModel="spring"

@Mapper(componentModel="spring")

public interface PersonConverter {

@Mappings({

@Mapping(source = "birthday", target = "birth"),

@Mapping(source = "birthday", target = "birthDateFormat", dateFormat = "yyyy-MM-dd HH:mm:ss"),

@Mapping(target = "birthExpressionFormat", expression = "java(org.apache.commons.lang3.time.DateFormatUtils.format(person.getBirthday(),\"yyyy-MM-dd HH:mm:ss\"))"),

@Mapping(source = "user.age", target = "age"),

@Mapping(target = "email", ignore = true)

})

PersonDTO domain2dto(Person person);

}

这时候测试类改一下，我用的 spring boot 的形式

@RunWith(SpringRunner.class)

@SpringBootTest(classes = BaseTestConfiguration.class)

public class PersonConverterTest {

//这里把转换器装配进来

@Autowired

private PersonConverter personConverter;

@Test

public void test() {

Person person = new Person(1L,"zhige","zhige.me@gmail.com",new Date(),new User(1));

PersonDTO personDTO = personConverter.domain2dto(person);

assertNotNull(personDTO);

assertEquals(personDTO.getId(), person.getId());

assertEquals(personDTO.getName(), person.getName());

assertEquals(personDTO.getBirth(), person.getBirthday());

String format = DateFormatUtils.format(personDTO.getBirth(), "yyyy-MM-dd HH:mm:ss");

assertEquals(personDTO.getBirthDateFormat(),format);

assertEquals(personDTO.getBirthExpressionFormat(),format);

}

}

我 test 路径下加入了一个配置类

@EnableAutoConfiguration

@Configuration

@ComponentScan

public class BaseTestConfiguration {

}

MapStruct 注解的关键词

@Mapper 只有在接口加上这个注解， MapStruct 才会去实现该接口

@Mapper 里有个 componentModel 属性，主要是指定实现类的类型，一般用到两个

default：默认，可以通过 Mappers.getMapper(Class) 方式获取实例对象

spring：在接口的实现类上自动添加注解 @Component，可通过 @Autowired 方式注入

@Mapping：属性映射，若源对象属性与目标对象名字一致，会自动映射对应属性

source：源属性

target：目标属性

dateFormat：String 到 Date 日期之间相互转换，通过 SimpleDateFormat，该值为 SimpleDateFormat 的日期格式

ignore: 忽略这个字段

@Mappings：配置多个@Mapping

@MappingTarget 用于更新已有对象

@InheritConfiguration 用于继承配置

本文只是写了一些常用的比较简单的一些功能，更详细的可以去阅读官方文档：