前言

使用Mybatis的开发者，大多数都会遇到一个问题，就是要写大量的SQL在xml文件中，除了特殊的业务逻辑SQL之外，还有大量结构类似的增删改查SQL。而且，当数据库表结构改动时，对应的所有SQL以及实体类都需要更改。这工作量和效率的影响或许就是区别增删改查程序员和真正程序员的屏障。这时，通用Mapper便应运而生……

什么是通用Mapper

通用Mapper就是为了解决单表增删改查，基于Mybatis的插件。开发人员不需要编写SQL，不需要在DAO中增加方法，只要写好实体类，就能支持相应的增删改查方法。

如何使用

以MySQL为例，假设存在这样一张表：

```
CREATE TABLE `test_table` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT '',
  `create_time` datetime DEFAULT NULL,
  `create_user_id` varchar(32) DEFAULT NULL,
  `update_time` datetime DEFAULT NULL,
  `update_user_id` varchar(32) DEFAULT NULL,
  `is_delete` int(8) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

主键是`id`，自增。下面以这张表为例介绍如何使用通用Mapper。

Maven依赖

```
<!-- 通用Mapper -->
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper</artifactId>
    <version>3.3.9</version>
</dependency>
```

SpringMVC配置

```
<!-- 通用 Mapper -->
<bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage"
value="cn.com.bluemoon.bd.service.spider.dao"/>
    <property name="properties">
        <value>
            mappers=tk.mybatis.mapper.common.Mapper
        </value>
    </property>
</bean>
```

注意这里使用`tk.mybatis.spring.mapper.MapperScannerConfigure`替换原来Mybatis的`org.mybatis.spring.mapper.MapperScannerConfigurer`。

可配参数介绍：

- `UUID`：设置生成UUID的方法，需要用OGNL方式配置，不限制返回值，但是必须和字段类型匹配
- `IDENTITY`：取回主键的方式，可以配置的内容看下一篇如何使用中的介绍
- `ORDER`：`<seletKey>`中的order属性，可选值为BEFORE和AFTER
- `catalog`：数据库的catalog，如果设置该值，查询的时候表名会带catalog设置的前缀
- `schema`：同catalog，catalog优先级高于schema
- `seqFormat`：序列的获取规则,使用{num}格式化参数，默认值为{0}.nextval，针对Oracle，可选参数一共4个，对应0,1,2,3分别为SequenceName，ColumnName, PropertyName，TableName
- `notEmpty`：insert和update中，是否判断字符串类型!=''，少数方法会用到
- `style`：实体和表转换时的规则，默认驼峰转下划线，可选值为normal用实体名和字段名;camelhump是默认值，驼峰转下划线;uppercase转换为大写;lowercase转换为小写
- `enableMethodAnnotation`：可以控制是否支持方法上的JPA注解，默认false。

大多数情况下不会用到这些参数，有特殊情况可以自行研究。

实体类的写法

记住一个原则：实体类的字段数量 >= 数据库表中需要操作的字段数量。默认情况下，实体类中的所有字段都会作为表中的字段来操作，如果有额外的字段，必须加

上 `@Transient` 注解。

```
@Table(name = "test_table")
public class TestTableVO implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(generator = "JDBC")
    private Long id;

    @Transient
    private String userId;

    private String name;

    private Timestamp createTime;

    private String createUserId;

    private Timestamp updateTime;

    private String updateUserId;

    private Integer isDelete;

    // 省略get、set…

}
```

说明：

1. 表名默认使用类名,驼峰转下划线(只对大写字母进行处理),如 `UserInfo` 默认对应的表名为 `user_info` 。
2. 表名可以使用 `@Table(name = "tableName")` 进行指定,对不符合第一条默认规则的可以通过这种方式指定表名.
3. 字段默认和 `@Column` 一样,都会作为表字段,表字段默认为Java对象的Field名字驼峰转下划线形式.
4. 可以使用 `@Column(name = "fieldName")` 指定不符合第3条规则的字段名
5. 使用 `@Transient` 注解可以忽略字段,添加该注解的字段不会作为表字段使用.
6. 建议一定是有一个 `@Id` 注解作为主键的字段,可以有多个 `@Id` 注解的字段作为联合主键.

7. 如果是MySQL的自增字段，加上`@GeneratedValue(generator = "JDBC")`即可。如果是其他数据库，可以参考官网文档。

`DAO`的写法

在传统的Mybatis写法中，`DAO`接口需要与`Mapper`文件关联，即需要编写`SQL`来实现`DAO`接口中的方法。而在通用Mapper中，`DAO`只需要继承一个通用接口，即可拥有丰富的方法：

继承通用的Mapper，必须指定泛型

```
public interface TestTableDao extends Mapper<TestTableVO> {
}
```

一旦继承了Mapper，继承的Mapper就拥有了Mapper所有的通用方法：

Select
方法：`List<T> select(T record);`
说明：根据实体中的属性值进行查询，查询条件使用等号

方法：`T selectByPrimaryKey(Object key);`
说明：根据主键字段进行查询，方法参数必须包含完整的主键属性，查询条件使用等号

方法：`List<T> selectAll();`
说明：查询全部结果，select(null)方法能达到同样的效果

方法：`T selectOne(T record);`
说明：根据实体中的属性进行查询，只能有一个返回值，有多个结果是抛出异常，查询条件使用等号

方法：`int selectCount(T record);`
说明：根据实体中的属性查询总数，查询条件使用等号

Insert
方法：`int insert(T record);`
说明：保存一个实体，null的属性也会保存，不会使用数据库默认值

方法：`int insertSelective(T record);`
说明：保存一个实体，null的属性不会保存，会使用数据库默认值

Update
方法：`int updateByPrimaryKey(T record);`

说明：根据主键更新实体全部字段，null值会被更新

方法：`int updateByPrimaryKeySelective(T record);`
说明：根据主键更新属性不为null的值

Delete
方法：`int delete(T record);`
说明：根据实体属性作为条件进行删除，查询条件使用等号

方法：`int deleteByPrimaryKey(Object key);`
说明：根据主键字段进行删除，方法参数必须包含完整的主键属性

Example方法
方法：`List<T> selectByExample(Object example);`
说明：根据Example条件进行查询
重点：这个查询支持通过`Example`类指定查询列，通过`selectProperties`方法指定查询列

方法：`int selectCountByExample(Object example);`
说明：根据Example条件进行查询总数

方法：`int updateByExample(@Param("record") T record, @Param("example") Object example);`
说明：根据Example条件更新实体`record`包含的全部属性，null值会被更新

方法：`int updateByExampleSelective(@Param("record") T record, @Param("example") Object example);`
说明：根据Example条件更新实体`record`包含的不是null的属性值

方法：`int deleteByExample(Object example);`
说明：根据Example条件删除数据

代码中使用

在`service`中注入`dao`，即可使用。

```
@Autowired
private TestTableDao testTableDao;
```

下面演示大概的写法：

新增

```
```

```
TestTableVO vo = new TestTableVO();
// 省略为vo设置属性...
int row = testTableDao.insertSelective(vo);
```

修改

```
TestTableVO vo = new TestTableVO();
// 省略为vo设置属性...
int row = testTableDao.updateByPrimaryKeySelective(vo);
```

查询单个

```
TestTableVO vo = new TestTableVO();
vo.setId(123L);
TestTableVO result = testTableDao.selectOne(vo);
```

条件查询

```
// 创建Example
Example example = new Example(TestTableVO.class);
// 创建Criteria
Example.Criteria criteria = example.createCriteria();
// 添加条件
criteria.andEqualTo("isDelete", 0);
criteria.andLike("name", "%abc123%");
List<TestTableVO> list = testTableDao.selectByExample(example);
```

总结

通用Mapper的原理是通过反射获取实体类的信息，构造出相应的SQL，因此我们只需要维护好实体类即可，对于应付复杂多变的需求提供了很大的便利。上文叙述的只是通用Mapper的简单用法，在实际项目中，还是要根据业务，在通用Mapper的基础上封装出粒度更大、更通用、更好用的方法。

附 Spring Boot 配置

Maven

```
```

```xml
<!--mybatis-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.3.1</version>
</dependency>
<!--mapper-->
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper-spring-boot-starter</artifactId>
    <version>1.1.4</version>
</dependency>
```

application.properties 配置

```properties
#mapper
#mappers 多个接口时逗号隔开
mapper.mappers=tk.mybatis.mapper.common.Mapper
mapper.not-empty=false
mapper.identity=MYSQL
```

参考

- https://gitee.com/free/Mapper
- https://github.com/abel533/MyBatis-Spring-Boot