# **[如何使用通用Mapper](https://www.cnblogs.com/softidea/p/6049285.html)**

集成方法请看上面的文档，集成后，可以继续阅读本页文档。

## **1. 继承通用的Mapper<T>,必须指定泛型<T>**

例如下面的例子:

public interface UserInfoMapper extends Mapper<UserInfo> {

//其他必须手写的接口...

}

一旦继承了Mapper<T>,继承的Mapper就拥有了Mapper<T>所有的通用方法。

2. 泛型(实体类)<T>的类型必须符合要求

实体类按照如下规则和数据库表进行转换,注解全部是JPA中的注解:

**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**。

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.

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.

可以使用@Column(name = "fieldName")指定不符合第3条规则的字段名

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.

**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.**

**默认情况下,实体类中如果不存在包含**@Id**注解的字段,所有的字段都会作为主键字段进行使用(这种效率极低).**

**实体类**可以**继承**使用,可以参考测试代码中的tk.mybatis.mapper.model.UserLogin2类.

由于基本类型,如int作为实体类字段时会有默认值0,而且无法消除,所以实体类中建议不要使用基本类型.

@NameStyle注解，用来配置对象名/字段和表名/字段之间的转换方式，该注解优先于全局配置style，可选值：

* 1. normal:使用实体类名/属性名作为表名/字段名
  2. camelhump:**这是默认值**，驼峰转换为下划线形式
  3. uppercase:转换为大写
  4. lowercase:转换为小写

通过[使用Mapper专用的MyBatis生成器插件](http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/7.UseMBG.md" \t "https://www.cnblogs.com/softidea/p/_blank)可以直接生成符合要求带注解的实体类。

### **重点强调@Transient注解**

许多人由于不仔细看文档，频繁在这个问题上出错。

如果你的实体类中包含了不是数据库表中的字段，你需要给这个字段加上@Transient注解，这样通用Mapper在处理单表操作时就不会将标注的属性当成表字段处理！

## **3.主键策略(仅用于**insert**方法)**

通用Mapper还提供了序列(支持Oracle)、UUID(任意数据库,字段长度32)、**主键自增**(类似Mysql,Hsqldb)三种方式，其中序列和UUID可以配置多个，**主键自增只能配置一个**。

由于MySql自增主键最常用，所以这里从最简单的配置方式开始。

### **1.@GeneratedValue(generator = "JDBC")**

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

这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段）。  
这种情况对应的xml类似下面这样:

<insert id="insertAuthor" useGeneratedKeys="true" keyProperty="id">

insert into Author (username,password,email,bio)

values (#{username},#{password},#{email},#{bio})</insert>

### **2.@**GeneratedValue**(strategy = GenerationType.IDENTITY)**

这个注解适用于**主键自增**的情况，支持下面这些数据库：

* **DB2**: VALUES IDENTITY\_VAL\_LOCAL()
* **MYSQL**: SELECT **LAST\_INSERT\_ID**()
* **SQLSERVER**: SELECT SCOPE\_IDENTITY()
* **CLOUDSCAPE**: VALUES IDENTITY\_VAL\_LOCAL()
* **DERBY**: VALUES IDENTITY\_VAL\_LOCAL()
* **HSQLDB**: CALL IDENTITY()
* **SYBASE**: SELECT @@IDENTITY
* **DB2\_MF**: SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1
* **INFORMIX**: select dbinfo('sqlca.sqlerrd1') from systables where tabid=1
* **JDBC**:这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段）。

使用GenerationType.IDENTITY需要在全局配置中配置IDENTITY的参数值，并且需要根据数库配置ORDER属性。

举例如下：

//不限于@**Id**注解的字段,但是**一个实体类中只能存在一个**(继承关系中也**只能存在一个**)@Id@GeneratedValue(strategy = GenerationType.IDENTITY)private Integer id;

对应的XML形式为：

<insert id="insertAuthor">

<selectKey keyProperty="id" resultType="int" order="AFTER">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into Author

(id, username, password, email,bio, favourite\_section)

values

(#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})</insert>

注意<**selectKey**>中的内容就是IDENTITY参数值对应数据库的SQL

### **3.@GeneratedValue(generator = "UUID")**

//可以用于任意字符串类型长度超过32位的字段@GeneratedValue(generator = "UUID")private String username;

该字段不会回写。这种情况对应类似如下的XML：

<insert id="insertAuthor">

<bind name="username\_bind" value='@java.util.UUID@randomUUID().toString().replace("-", "")' />

insert into Author

(id, username, password, email,bio, favourite\_section)

values

(#{id}, #{username\_bind}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})</insert>

**注意：这种方式不能回写，如果想要回写，请看**[通用 Mapper UUID 简单示例](http://git.oschina.net/free/Mapper/blob/master/wiki/mapper3/10.Mapper-UUID.md" \t "https://www.cnblogs.com/softidea/p/_blank)

### **4.Oracle使用序列**

@Id@GeneratedValue(strategy = GenerationType.IDENTITY,generator = "select SEQ\_ID.nextval from dual")private Integer id;

使用Oracle序列的时候，还需要配置:

<property name="ORDER" value="BEFORE"/>

因为在插入数据库前，需要先获取到序列值，否则会报错。  
这种情况对于的xml类似下面这样：

<insert id="insertAuthor"><selectKey keyProperty="id" resultType="int" order="BEFORE">

select SEQ\_ID.nextval from dual</selectKey>

insert into Author

(id, username, password, email,bio, favourite\_section)

values

(#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})</insert>

## **4. 将继承的Mapper接口添加到Mybatis配置中**

### **非Spring项目中在mybatis配置文件中配置，如:**

<mappers>

<mapper class="tk.mybatis.mapper.mapper.CountryMapper" />

<mapper class="tk.mybatis.mapper.mapper.UserInfoMapper" />

<mapper class="tk.mybatis.mapper.mapper.UserLoginMapper" /></mappers>

### **Spring配置方式**

如果你在Spring中配置Mapper接口,不需要像上面这样一个个配置,只需要有下面的这个扫描Mapper接口的这个配置即可:

<bean class="tk.mybatis.spring.mapper.**MapperScannerConfigurer**">

<property name="**basePackage**" value="com.isea533.mybatis.mapper"/></bean>

另外因为通用接口都有顶层的接口，所以你还可以用下面的方式进行配置：

<bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="**basePackage**" value="com.\*\*.mapper"/>

<property name="**markerInterface**" value="tk.mybatis.mapper.common.Mapper"/></bean>

这样配置后，直接继承了Mapper接口的才会被扫描，basePackage可以配置的范围更大。

如果想在Spring4中使用泛型注入，还需要包含Mapper<T>所在的包，具体请看 [在Spring4中使用通用Mapper](http://git.oschina.net/free/Mapper2/blob/master/wiki/mapper/4.Spring4.md" \t "https://www.cnblogs.com/softidea/p/_blank)。

## **5. 代码中使用**

例如下面这个简单的例子:

SqlSession sqlSession = MybatisHelper.getSqlSession();try {

//**获取Mapper**

UserInfoMapper mapper = **sqlSession.getMapper**(UserInfoMapper.class);

UserInfo userInfo = new UserInfo();

userInfo.setUsername("abel533");

userInfo.setPassword("123456");

userInfo.setUsertype("2");

userInfo.setEmail("abel533@gmail.com");

//新增一条数据

Assert.assertEquals(1, mapper.insert(userInfo));

//ID回写,不为空

Assert.assertNotNull(userInfo.getId());

//6是当前的ID

Assert.assertEquals(6, (int)userInfo.getId());

//通过主键删除新增的数据

Assert.assertEquals(1,mapper.deleteByPrimaryKey(userInfo));} finally {

sqlSession.close();}

另一个例子:

SqlSession sqlSession = MybatisHelper.getSqlSession();try {

//获取Mapper

CountryMapper mapper = sqlSession.getMapper(CountryMapper.class);

//查询总数

Assert.assertEquals(183, mapper.selectCount(new Country()));

//查询100

Country country = mapper.selectByPrimaryKey(100);

//根据主键删除

Assert.assertEquals(1, mapper.deleteByPrimaryKey(100));

//查询总数

Assert.assertEquals(182, mapper.selectCount(new Country()));

//插入

Assert.assertEquals(1, mapper.insert(country));} finally {

sqlSession.close();}

**附:Spring使用相关**

直接在需要的地方注入**Mapper继承的接口**即可,和一般情况下的使用没有区别.

## **6.其他**

如果你的实体是继承Map的，你可能需要将数据库查询的结果从大写下划线形式转换为驼峰形式，你可以搭配下面的拦截器使用：

**CameHumpInterceptor - Map结果的Key转为驼峰式**