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** **int**erface UserInfoMapper **extends** Mapper**<**UserInfo**>** **{**

*//其他必须手写的接口...*

**}**

一旦继承了Mapper<T>,继承的Mapper就拥有了以下通用的方法:

*//根据实体类不为null的字段进行查询,条件全部使用=号and条件*

List**<**T**>** select**(**T record**);**

*//根据实体类不为null的字段查询总数,条件全部使用=号and条件*

**int** **selectCount**(T record**);**

*//根据主键进行查询,必须保证结果唯一*

*//单个字段做主键时,可以直接写主键的值*

*//联合主键时,key可以是实体类,也可以是Map*

T **selectByPrimaryKey**(Object key**);**

*//插入一条数据*

*//支持Oracle序列,UUID,类似Mysql的INDENTITY自动增长(自动回写)*

*//优先使用传入的参数值,参数值空时,才会使用序列、UUID,自动增长*

**int** **insert**(T record**);**

*//插入一条数据,只插入不为null的字段,不会影响有默认值的字段*

*//支持Oracle序列,UUID,类似Mysql的INDENTITY自动增长(自动回写)*

*//优先使用传入的参数值,参数值空时,才会使用序列、UUID,自动增长*

**int** **insertSelective**(T record**);**

*//根据实体类中字段不为null的条件进行删除,条件全部使用=号and条件*

**int** **delete**(T key**);**

*//通过主键进行删除,这里最多只会删除一条数据*

*//单个字段做主键时,可以直接写主键的值*

*//联合主键时,key可以是实体类,也可以是Map*

**int** **deleteByPrimaryKey**(Object key**);**

*//根据主键进行更新,这里最多只会更新一条数据*

*//参数为实体类*

**int** **updateByPrimaryKey**(T record**);**

*//根据主键进行更新*

*//只会更新不是null的数据*

**int** **updateByPrimaryKeySelective**(T record**);**

*//根据Exmaple条件查询总数*

**int** **selectCountByExample**(Object example**);**

*//根据Exmaple条件删除*

**int** **deleteByExample**(Object example**);**

*//根据Exmaple条件查询*

List**<**T**>** selectByExample**(**Object example**);**

*//根据Exmaple条件更新非空(null)字段*

**int** **updateByExampleSelective**(@Param**(**"record"**)** T record**,** @Param**(**"example"**)** Object example**);**

*//根据Exmaple条件更新全部字段*

**int** **updateByExample**(@Param**(**"record"**)** T record**,** @Param**(**"example"**)** Object example**);**

## 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:

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. 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).
8. 实体类可以继承使用,可以参考测试代码中的com.github.abel533.model.UserLogin2类.
9. 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.

除了上面提到的这些,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。

这三种方式不能同时使用,同时存在时按照 序列>UUID>主键自增的优先级进行选择.下面是具体配置方法:

1. 使用序列可以添加如下的注解:
2. *//可以用于数字类型,字符串类型(需数据库支持自动转型)的字段*
3. @SequenceGenerator**(**name**=**"Any"**,**sequenceName**=**"seq\_userid"**)**
4. @Id
5. **private** Integer id**;**

该字段不会回写。这种情况对应类似如下的XML：

<insert id="insertAuthor">

insert into Author

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

values

(seq\_userid.nextval, #{username, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})

</insert>

1. 使用UUID时:
2. *//可以用于任意字符串类型长度超过32位的字段*
3. @GeneratedValue**(**generator **=** "UUID"**)**
4. **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>

1. 使用主键自增:
2. *//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)*
3. @Id
4. @GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**
5. **private** Integer id**;**

增加这个注解后，\_\_会回写ID\_\_。

通过设置@GeneratedValue的generator参数可以支持更多的获取主键的方法，例如在Oracle中使用序列：

@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>

1. 主键自增还有一种简单的写法：
2. *//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)*
3. @GeneratedValue**(**generator **=** "JDBC"**)**
4. **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>

## 3. 将继承的Mapper接口添加到Mybatis配置中

例如本项目测试中的配置:

<mappers>

<mapper class="com.github.abel533.mapper.CountryMapper" />

<mapper class="com.github.abel533.mapper.Country2Mapper" />

<mapper class="com.github.abel533.mapper.CountryTMapper" />

<mapper class="com.github.abel533.mapper.CountryUMapper" />

<mapper class="com.github.abel533.mapper.CountryIMapper" />

<mapper class="com.github.abel533.mapper.UserInfoMapper" />

<mapper class="com.github.abel533.mapper.UserLoginMapper" />

<mapper class="com.github.abel533.mapper.UserLogin2Mapper" />

</mappers>

**附:Spring配置相关**

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="org.mybatis.spring.mapper.MapperScannerConfigurer">

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

</bean>

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/UseMapperInSpring4.md)。

## 4. 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.