

## 课程介绍

- ActiveRecord
- Oracle 主键Sequence
- Mybatis-Plus的插件
- Sql 注入器实现自定义全局操作
- 自动填充功能
- 逻辑删除
- 通用枚举
- 代码生成器
- MybatisX 快速开发插件

## 1、ActiveRecord

ActiveRecord (简称AR) 一直广受动态语言 (PHP、Ruby 等) 的喜爱，而Java 作为准静态语言，对于 ActiveRecord 往往只能感叹其优雅，所以我们也 AR 道路上进行了一定的探索，希望大家能够喜欢。

### 什么是ActiveRecord ?

ActiveRecord也属于ORM (对象关系映射) 层，由Rails最早提出，遵循标准的ORM模型：表映射到记录，记录映射到对象，字段映射到对象属性。配合遵循的命名和配置惯例，能够很大程度的快速实现模型的操作，而且简洁易懂。

ActiveRecord的主要思想是：

- 每一个数据库表对应创建一个类，类的每一个对象实例对应于数据库中表的一行记录；通常表的每个字段在类中都有相应的Field；
- ActiveRecord同时负责把自己持久化，在ActiveRecord中封装了对数据库的访问，即CURD；
- ActiveRecord是一种领域模型(Domain Model)，封装了部分业务逻辑；

### 1.1、开启AR之旅

在MP中，开启AR非常简单，只需要将实体对象继承Model即可。

```
1 package cn.itcast.mp.pojo;
2
3 import com.baomidou.mybatisplus.annotation.IdType;
4 import com.baomidou.mybatisplus.annotation.TableField;
5 import com.baomidou.mybatisplus.annotation.TableId;
6 import com.baomidou.mybatisplus.annotation.TableName;
7 import com.baomidou.mybatisplus.extension.activerecord.Model;
8 import lombok.AllArgsConstructor;
9 import lombok.Data;
10 import lombok.NoArgsConstructor;
11
12 @Data
13 @NoArgsConstructor
14 @AllArgsConstructor
15 public class User extends Model<User> {
```



```
16
17     private Long id;
18     private String userName;
19     private String password;
20     private String name;
21     private Integer age;
22     private String email;
23
24 }
```

## 1.2、根据主键查询

```
1  @RunWith(SpringRunner.class)
2  @SpringBootTest
3  public class UserMapperTest {
4
5      @Autowired
6      private UserMapper userMapper;
7
8      @Test
9      public void testAR() {
10         User user = new User();
11         user.setId(2L);
12         User user2 = user.selectById();
13
14         System.out.println(user2);
15     }
16
17 }
```

## 1.3、新增数据

```
1  @RunWith(SpringRunner.class)
2  @SpringBootTest
3  public class UserMapperTest {
4
5      @Autowired
6      private UserMapper userMapper;
7
8      @Test
9      public void testAR() {
10         User user = new User();
11         user.setName("刘备");
12         user.setAge(30);
13         user.setPassword("123456");
14         user.setUserName("liubei");
15         user.setEmail("liubei@itcast.cn");
16
17         boolean insert = user.insert();
18
19         System.out.println(insert);
20     }
21 }
```

```
20     }  
21  
22 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] ==> Preparing: INSERT INTO  
  tb_user ( user_name, password, name, age, email ) VALUES ( ?, ?, ?, ?, ? )  
2 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] ==> Parameters: liubei(String),  
  123456(String), 刘备(String), 30(Integer), liubei@itcast.cn(String)  
3 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] <== Updates: 1
```

id	user_name	password	name	age	email
2	lisi	123456	李四	20	test2@itcast.cn
3	wangwu	123456	王五	28	test3@itcast.cn
4	zhaoliu	123456	赵六	21	test4@itcast.cn
5	sunqi	123456	孙七	24	test5@itcast.cn
6	caocao	123456	曹操	20	test@itcast.cn
7	caocao	123456	曹操	20	test@itcast.cn
8	liubei	123456	刘备	30	liubei@itcast.cn

## 1.5、更新操作

```
1 @RunWith(SpringRunner.class)  
2 @SpringBootTest  
3 public class UserMapperTest {  
4  
5     @Autowired  
6     private UserMapper userMapper;  
7  
8     @Test  
9     public void testAR() {  
10         User user = new User();  
11         user.setId(8L);  
12         user.setAge(35);  
13  
14         boolean update = user.updateById();  
15         System.out.println(update);  
16     }  
17  
18 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Preparing: UPDATE
  tb_user SET age=? WHERE id=?
2 [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Parameters: 35(Integer),
  8(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] <== Updates: 1
```

id	user_name	password	name	age	email
2	lisi	123456	李四	20	test2@itcast.cn
3	wangwu	123456	王五	28	test3@itcast.cn
4	zhaoliu	123456	赵六	21	test4@itcast.cn
5	sunqi	123456	孙七	24	test5@itcast.cn
6	caocao	123456	曹操	20	test@itcast.cn
7	caocao	123456	曹操	20	test@itcast.cn
8	liubei	123456	刘备	35	liubei@itcast.cn

## 1.6、删除操作

```
1 @RunWith(SpringRunner.class)
2 @SpringBootTest
3 public class UserMapperTest {
4
5     @Autowired
6     private UserMapper userMapper;
7
8     @Test
9     public void testAR() {
10         User user = new User();
11         user.setId(7L);
12
13         boolean delete = user.deleteById();
14         System.out.println(delete);
15     }
16
17 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Preparing: DELETE FROM
  tb_user WHERE id=?
2 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Parameters: 7(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] <== Updates: 1
```

## 1.7、根据条件查询

```
1 @RunWith(SpringRunner.class)
2 @SpringBootTest
3 public class UserMapperTest {
```



```
4
5     @Autowired
6     private UserMapper userMapper;
7
8     @Test
9     public void testAR() {
10         User user = new User();
11         QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
12         userQueryWrapper.le("age", "20");
13
14         List<User> users = user.selectList(userQueryWrapper);
15         for (User user1 : users) {
16             System.out.println(user1);
17         }
18     }
19
20 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE age <= ?
2 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters: 20(String)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] <== Total: 2
4
5 User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
address=null)
6 User(id=6, userName=caocao, password=123456, name=曹操, age=20, email=test@itcast.cn,
address=null)
```

## 2、Oracle 主键Sequence

在mysql中，主键往往是自增长的，这样使用起来是比较方便的，如果使用的是Oracle数据库，那么就不能使用自增长了，就得使用Sequence 序列生成id值了。

### 2.1、部署Oracle环境

为了简化环境部署，这里使用Docker环境进行部署安装Oracle。

```
1 #拉取镜像
2 docker pull sath89/oracle-12c
3
4 #创建容器
5 docker create --name oracle -p 1521:1521 sath89/oracle-12c
6
7 #启动
8 docker start oracle && docker logs -f oracle
9
10 #下面是启动过程
11 Database not initialized. Initializing database.
12 Starting tnslnsr
```



```
13 Copying database files
14 1% complete
15 3% complete
16 11% complete
17 18% complete
18 26% complete
19 37% complete
20 Creating and starting Oracle instance
21 40% complete
22 45% complete
23 50% complete
24 55% complete
25 56% complete
26 60% complete
27 62% complete
28 Completing Database Creation
29 66% complete
30 70% complete
31 73% complete
32 85% complete
33 96% complete
34 100% complete
35 Look at the log file "/u01/app/oracle/cfgtoollogs/dbca/xe/xe.log" for further details.
36 Configuring Apex console
37 Database initialized. Please visit http://#container:8080/em
    http://#container:8080/apex for extra configuration if needed
38 Starting web management console
39
40 PL/SQL procedure successfully completed.
41
42 Starting import from '/docker-entrypoint-initdb.d':
43 ls: cannot access /docker-entrypoint-initdb.d/*: No such file or directory
44 Import finished
45
46 Database ready to use. Enjoy! ;)
47
48 #通过用户名密码即可登录
49 用户名和密码为： system/oracle
```

下面使用navicat12进行连接并操作Oracle，使用资料中提供的安装包，可以试用14天。

需要注意的是：由于安装的Oracle是64位版本，所以navicat也是需要使用64为版本，否则连接不成功。



Oracle - 新建连接

常规

高级

数据库

SSH

  
Navicat

  
数据库

连接名:

192.168.31.81

连接类型:

Basic

主机:

192.168.31.81

端口:

1521

服务名:

xe

☒ 服务名

☐ SID

用户名:

system

密码:

●●●●●●

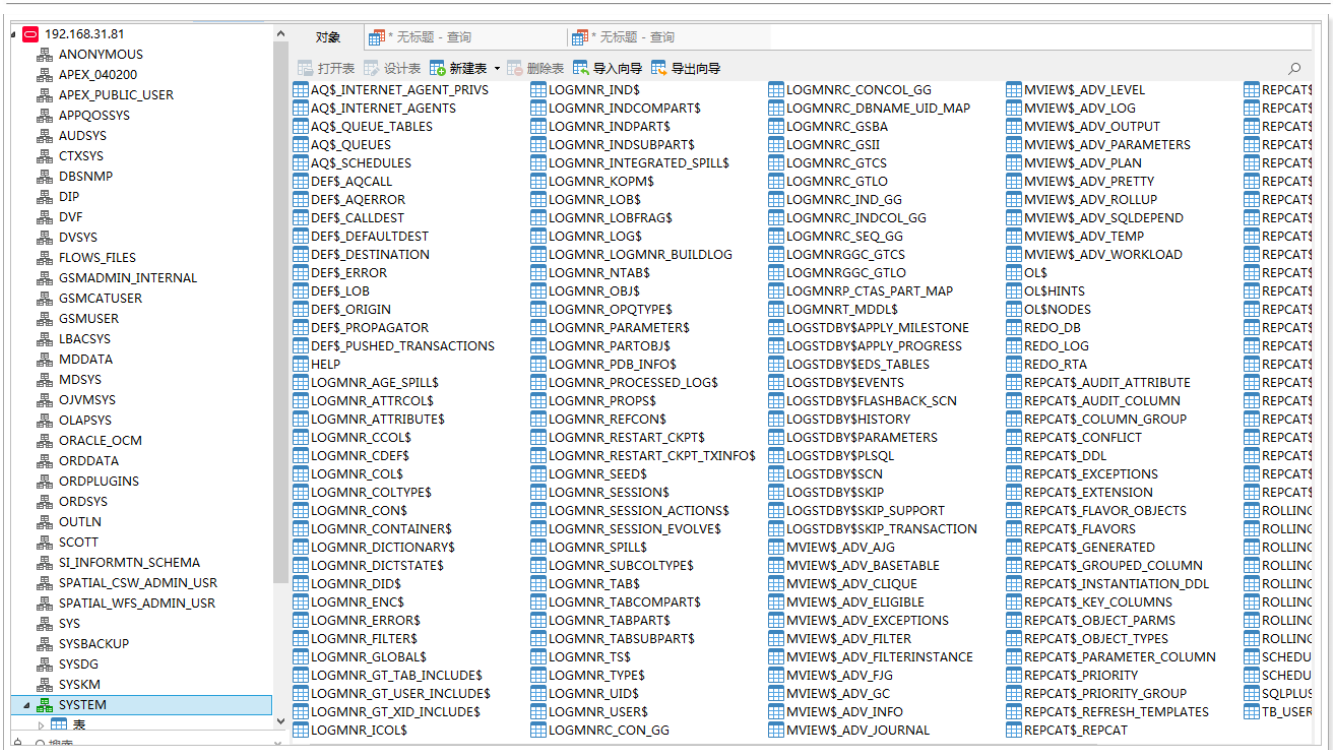
☒ 保存密码

测试连接

确定

取消

连接成功：



## 2.2、创建表以及序列

```

1  --创建表，表名以及字段名都要大写
2  CREATE TABLE "TB_USER" (
3      "ID" NUMBER(20) VISIBLE NOT NULL ,
4      "USER_NAME" VARCHAR2(255 BYTE) VISIBLE ,
5      "PASSWORD" VARCHAR2(255 BYTE) VISIBLE ,
6      "NAME" VARCHAR2(255 BYTE) VISIBLE ,
7      "AGE" NUMBER(10) VISIBLE ,
8      "EMAIL" VARCHAR2(255 BYTE) VISIBLE
9  )
10
11 --创建序列
12 CREATE SEQUENCE SEQ_USER START WITH 1 INCREMENT BY 1
    
```

## 2.3、jdbc驱动包

由于版权原因，我们不能直接通过maven的中央仓库下载oracle数据库的jdbc驱动包，所以我们需要将驱动包安装到本地仓库。

```

1  #ojdbc8.jar文件在资料中可以找到
2
3  mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc8 -Dversion=12.1.0.1 -
    Dpackaging=jar -Dfile=ojdbc8.jar
    
```

安装完成后的坐标：



```
1 <dependency>
2   <groupId>com.oracle</groupId>
3   <artifactId>ojdbc8</artifactId>
4   <version>12.1.0.1</version>
5 </dependency>
```

## 2.4、修改application.properties

对于application.properties的修改，需要修改2个位置，分别是：

```
1 #数据库连接配置
2 spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
3 spring.datasource.url=jdbc:oracle:thin:@192.168.31.81:1521:xe
4 spring.datasource.username=system
5 spring.datasource.password=oracle
6
7 #id生成策略
8 mybatis-plus.global-config.db-config.id-type=input
```

## 2.5、配置序列

使用Oracle的序列需要做2件事情：

第一，需要配置MP的序列生成器到Spring容器：

```
1 package cn.itcast.mp;
2
3 import com.baomidou.mybatisplus.extension.incrementer.OracleKeyGenerator;
4 import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
5 import org.mybatis.spring.annotation.MapperScan;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8
9 @Configuration
10 @MapperScan("cn.itcast.mp.mapper") //设置mapper接口的扫描包
11 public class MybatisPlusConfig {
12
13     /**
14      * 分页插件
15      */
16     @Bean
17     public PaginationInterceptor paginationInterceptor() {
18         return new PaginationInterceptor();
19     }
20
21     /**
22      * 序列生成器
23      */
24     @Bean
25     public OracleKeyGenerator oracleKeyGenerator(){
26         return new OracleKeyGenerator();
27     }
28 }
```



```
27     }  
28 }
```

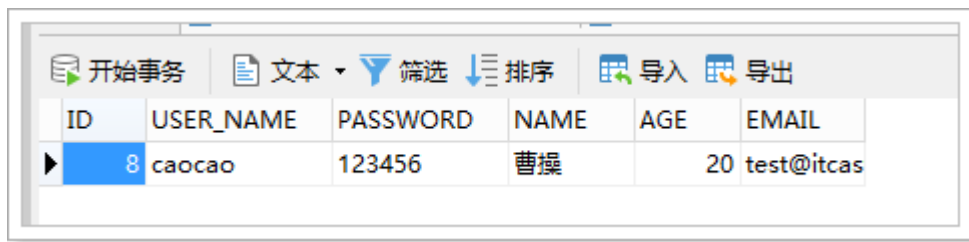
第二，在实体对象中指定序列的名称：

```
1 @KeySequence(value = "SEQ_USER", clazz = Long.class)  
2 public class User{  
3     .....  
4 }
```

## 2.6、测试

```
1 package cn.itcast.mp;  
2  
3 import cn.itcast.mp.mapper.UserMapper;  
4 import cn.itcast.mp.pojo.User;  
5 import org.junit.Test;  
6 import org.junit.runner.RunWith;  
7 import org.springframework.beans.factory.annotation.Autowired;  
8 import org.springframework.boot.test.context.SpringBootTest;  
9 import org.springframework.test.context.junit4.SpringRunner;  
10  
11 import java.util.List;  
12  
13 @RunWith(SpringRunner.class)  
14 @SpringBootTest  
15 public class UserMapperTest {  
16  
17     @Autowired  
18     private UserMapper userMapper;  
19  
20     @Test  
21     public void testInsert(){  
22         User user = new User();  
23         user.setAge(20);  
24         user.setEmail("test@itcast.cn");  
25         user.setName("曹操");  
26         user.setUsername("caocao");  
27         user.setPassword("123456");  
28  
29         int result = this.userMapper.insert(user); //返回的result是受影响的行数，并不是自增  
    后的id  
30         System.out.println("result = " + result);  
31  
32         System.out.println(user.getId()); //自增后的id会回填到对象中  
33     }  
34  
35     @Test  
36     public void testSelectById(){  
37         User user = this.userMapper.selectById(8L);  
38         System.out.println(user);  
39     }
```

```
40  
41 }
```



ID	USER_NAME	PASSWORD	NAME	AGE	EMAIL
8	caocao	123456	曹操	20	test@itcas

## 3、插件

### 3.1、mybatis的插件机制

MyBatis 允许你在已映射语句执行过程中的某一点进行拦截调用。默认情况下，MyBatis 允许使用插件来拦截的方法调用包括：

1. Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
2. ParameterHandler (getParameterObject, setParameters)
3. ResultSetHandler (handleResultSets, handleOutputParameters)
4. StatementHandler (prepare, parameterize, batch, update, query)

我们看到了可以拦截Executor接口的部分方法，比如update，query，commit，rollback等方法，还有其他接口的一些方法等。

总体概括为：

1. 拦截执行器的方法
2. 拦截参数的处理
3. 拦截结果集的处理
4. 拦截Sql语法构建的处理

拦截器示例：

```
1 package cn.itcast.mp.plugins;  
2  
3 import org.apache.ibatis.executor.Executor;  
4 import org.apache.ibatis.mapping.MappedStatement;  
5 import org.apache.ibatis.plugin.*;  
6  
7 import java.util.Properties;  
8  
9 @Intercepts({@Signature(  
10     type= Executor.class,  
11     method = "update",  
12     args = {MappedStatement.class, Object.class}})})  
13 public class MyInterceptor implements Interceptor {  
14  
15     @Override  
16     public Object intercept(Invocation invocation) throws Throwable {  
17         //拦截方法，具体业务逻辑编写的位置  
18         return invocation.proceed();  
19     }  
20 }
```

```
19     }
20
21     @Override
22     public Object plugin(Object target) {
23         //创建target对象的代理对象,目的是将当前拦截器加入到该对象中
24         return Plugin.wrap(target, this);
25     }
26
27     @Override
28     public void setProperties(Properties properties) {
29         //属性设置
30     }
31 }
```

注入到Spring容器：

```
1     /**
2      * 自定义拦截器
3      */
4     @Bean
5     public MyInterceptor myInterceptor(){
6         return new MyInterceptor();
7     }
```

或者通过xml配置，mybatis-config.xml：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <plugins>
7         <plugin interceptor="cn.itcast.mp.plugins.MyInterceptor"></plugin>
8     </plugins>
9 </configuration>
```

## 3.2、执行分析插件

在MP中提供了对SQL执行的分析的插件，可用作阻断全表更新、删除的操作，注意：该插件仅适用于开发环境，不适用于生产环境。

SpringBoot配置：



```
1  @Bean
2  public SqlExplainInterceptor sqlExplainInterceptor(){
3      SqlExplainInterceptor sqlExplainInterceptor = new SqlExplainInterceptor();
4
5      List<ISqlParser> sqlParserList = new ArrayList<>();
6      // 攻击 SQL 阻断解析器、加入解析链
7      sqlParserList.add(new BlockAttackSqlParser());
8      sqlExplainInterceptor.setSqlParserList(sqlParserList);
9
10     return sqlExplainInterceptor;
11 }
```

测试：

```
1  @Test
2  public void testUpdate(){
3      User user = new User();
4      user.setAge(20);
5
6      int result = this.userMapper.update(user, null);
7      System.out.println("result = " + result);
8  }
```

结果：



```
1 Caused by: com.baomidou.mybatisplus.core.exceptions.MybatisPlusException: Prohibition
  of table update operation
2     at
  com.baomidou.mybatisplus.core.toolkit.ExceptionUtils.mpe(ExceptionUtils.java:49)
3     at com.baomidou.mybatisplus.core.toolkit.Assert.isTrue(Assert.java:38)
4     at com.baomidou.mybatisplus.core.toolkit.Assert.notNull(Assert.java:72)
5     at
  com.baomidou.mybatisplus.extension.parsers.BlockAttackSqlParser.processUpdate(BlockAtt
  ackSqlParser.java:45)
6     at
  com.baomidou.mybatisplus.core.parser.AbstractJsqlParser.processParser(AbstractJsqlPars
  er.java:92)
7     at
  com.baomidou.mybatisplus.core.parser.AbstractJsqlParser.parser(AbstractJsqlParser.java
  :67)
8     at
  com.baomidou.mybatisplus.extension.handlers.AbstractSqlParserHandler.sqlParser(Abstrac
  tSqlParserHandler.java:76)
9     at
  com.baomidou.mybatisplus.extension.plugins.SqlExplainInterceptor.intercept(SqlExplainI
  nterceptor.java:63)
10    at org.apache.ibatis.plugin.Plugin.invoke(Plugin.java:61)
11    at com.sun.proxy.$Proxy70.update(Unknown Source)
12    at
  org.apache.ibatis.session.defaults.DefaultSqlSession.update(DefaultSqlSession.java:197
  )
13    ... 41 more
```

可以看到，当执行全表更新时，会抛出异常，这样有效防止了一些误操作。

### 3.3、性能分析插件

性能分析拦截器，用于输出每条 SQL 语句及其执行时间，可以设置最大执行时间，超过时间会抛出异常。

该插件只用于开发环境，不建议生产环境使用。

配置：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <plugins>
7         <!-- SQL 执行性能分析，开发环境使用，线上不推荐。 maxTime 指的是 sql 最大执行时长 -->
8         <plugin
9             interceptor="com.baomidou.mybatisplus.extension.plugins.PerformanceInterceptor">
10             <property name="maxTime" value="100" />
11             <!--SQL是否格式化 默认false-->
12             <property name="format" value="true" />
13         </plugin>
14     </plugins>
15 </configuration>
```

执行结果：

```
1 Time: 11 ms - ID: cn.itcast.mp.mapper.UserMapper.selectById
2 Execute SQL :
3     SELECT
4         id,
5         user_name,
6         password,
7         name,
8         age,
9         email
10    FROM
11        tb_user
12    WHERE
13        id=7
```

可以看到，执行时间为11ms。如果将maxTime设置为1，那么，该操作会抛出异常。

```
1 Caused by: com.baomidou.mybatisplus.core.exceptions.MybatisPlusException: The SQL
  execution time is too large, please optimize !
2     at com.baomidou.mybatisplus.core.toolkit.ExceptionUtils.mpe(ExceptionUtils.java:49)
3     at com.baomidou.mybatisplus.core.toolkit.Assert.isTrue(Assert.java:38)
4     .....
```

## 3.4、乐观锁插件

### 3.4.1、主要适用场景

意图：

当要更新一条记录的时候，希望这条记录没有被别人更新

乐观锁实现方式：

- 取出记录时，获取当前version
- 更新时，带上这个version
- 执行更新时，set version = newVersion where version = oldVersion
- 如果version不对，就更新失败

### 3.4.2、插件配置

spring xml:

```
1 <bean class="com.baomidou.mybatisplus.extension.plugins.OptimisticLockerInterceptor"/>
```

spring boot:



```
1 @Bean
2 public OptimisticLockerInterceptor optimisticLockerInterceptor() {
3     return new OptimisticLockerInterceptor();
4 }
```

### 3.4.3、注解实体字段

需要为实体字段添加@Version注解。

第一步，为表添加version字段，并且设置初始值为1：

```
1 ALTER TABLE `tb_user`
2 ADD COLUMN `version` int(10) NULL AFTER `email`;
3
4 UPDATE `tb_user` SET `version`='1';
```

第二步，为用户实体对象添加version字段，并且添加@Version注解：

```
1 @Version
2 private Integer version;
```

### 3.4.4、测试

测试用例：

```
1 @Test
2 public void testUpdate(){
3     User user = new User();
4     user.setAge(30);
5     user.setId(2L);
6     user.setVersion(1); //获取到version为1
7
8     int result = this.userMapper.updateById(user);
9     System.out.println("result = " + result);
10 }
```

执行日志：





```
1  [main] [com.baomidou.mybatisplus.extension.parsers.BlockAttackSqlParser]-[DEBUG]
   Original SQL: UPDATE tb_user SET age=?,
2
3  version=? WHERE id=? AND version=?
4  [main] [com.baomidou.mybatisplus.extension.parsers.BlockAttackSqlParser]-[DEBUG]
   parser sql: UPDATE tb_user SET age = ?, version = ? WHERE id = ? AND version = ?
5  [main] [org.springframework.jdbc.datasource.DataSourceUtils]-[DEBUG] Fetching JDBC
   Connection from DataSource
6  [main] [org.mybatis.spring.transaction.SpringManagedTransaction]-[DEBUG] JDBC
   Connection [HikariProxyConnection@540206885 wrapping
   com.mysql.jdbc.JDBC4Connection@27e0f2f5] will not be managed by Spring
7  [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Preparing: UPDATE
   tb_user SET age=?, version=? WHERE id=? AND version=?
8  [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Parameters:
   30(Integer), 2(Integer), 2(Long), 1(Integer)
9  [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] <== Updates: 1
10 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional
   SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@30135202]
11 result = 1
```

可以看到，更新的条件中有version条件，并且更新的version为2。

如果再次执行，更新则不成功。这样就避免了多人同时更新时导致数据的不一致。

### 3.4.5、特别说明

- 支持的数据类型只有:int,Integer,long,Long,Date,Timestamp,LocalDateTime
- 整数类型下 `newVersion = oldVersion + 1`
- `newVersion` 会回写到 `entity` 中
- 仅支持 `updateById(id)` 与 `update(entity, wrapper)` 方法
- 在 `update(entity, wrapper)` 方法下, `wrapper` 不能复用!!!

## 4、Sql 注入器

我们已经知道，在MP中，通过AbstractSqlInjector将BaseMapper中的方法注入到了Mybatis容器，这样这些方法才可以正常执行。

那么，如果我们需要扩充BaseMapper中的方法，又该如何实现呢？

下面我们以扩展findAll方法为例进行学习。

### 4.1、编写MyBaseMapper

```
1 package cn.itcast.mp.mapper;  
2  
3 import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
4  
5 import java.util.List;  
6  
7 public interface MyBaseMapper<T> extends BaseMapper<T> {  
8  
9     List<T> findAll();  
10  
11 }
```

其他的Mapper都可以继承该Mapper，这样实现了统一的扩展。

如：

```
1 package cn.itcast.mp.mapper;  
2  
3 import cn.itcast.mp.pojo.User;  
4  
5 public interface UserMapper extends MyBaseMapper<User> {  
6  
7     User findById(Long id);  
8 }
```

## 4.2、编写MySqlInjector

如果直接继承AbstractSqlInjector的话，原有的BaseMapper中的方法将失效，所以我们选择继承DefaultSqlInjector进行扩展。

```
1 package cn.itcast.mp.sqlInjector;  
2  
3 import com.baomidou.mybatisplus.core.injector.AbstractMethod;  
4 import com.baomidou.mybatisplus.core.injector.DefaultSqlInjector;  
5  
6 import java.util.List;  
7  
8 public class MySqlInjector extends DefaultSqlInjector {  
9  
10     @Override  
11     public List<AbstractMethod> getMethodList() {  
12         List<AbstractMethod> methodList = super.getMethodList();  
13  
14         methodList.add(new FindAll());  
15  
16         // 再扩充自定义的方法  
17         list.add(new FindAll());  
18  
19         return methodList;  
20     }  
21 }
```

## 4.3、编写FindAll

```
1 package cn.itcast.mp.sqlInjector;
2
3 import com.baomidou.mybatisplus.core.enums.SqlMethod;
4 import com.baomidou.mybatisplus.core.injector.AbstractMethod;
5 import com.baomidou.mybatisplus.core.metadata.TableInfo;
6 import org.apache.ibatis.mapping.MappedStatement;
7 import org.apache.ibatis.mapping.SqlSource;
8
9 public class FindAll extends AbstractMethod {
10
11     @Override
12     public MappedStatement injectMappedStatement(Class<?> mapperClass, Class<?>
modelClass, TableInfo tableInfo) {
13         String sqlMethod = "findAll";
14         String sql = "select * from " + tableInfo.getTableName();
15         SqlSource sqlSource = languageDriver.createSqlSource(configuration, sql,
modelClass);
16         return this.addSelectMappedStatement(mapperClass, sqlMethod, sqlSource,
modelClass, tableInfo);
17     }
18
19 }
```

## 4.4、注册到Spring容器

```
1 /**
2  * 自定义SQL注入器
3  */
4 @Bean
5 public MySQLInjector mysqlInjector(){
6     return new MySQLInjector();
7 }
```

## 4.5、测试

```
1 @Test
2 public void testFindAll(){
3     List<User> users = this.userMapper.findAll();
4     for (User user : users) {
5         System.out.println(user);
6     }
7 }
```

输出的SQL：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.findAll]-[DEBUG] ==> Preparing: select * from
  tb_user
2 [main] [cn.itcast.mp.mapper.UserMapper.findAll]-[DEBUG] ==> Parameters:
3 [main] [cn.itcast.mp.mapper.UserMapper.findAll]-[DEBUG] <==      Total: 10
```

至此，我们实现了全局扩展SQL注入器。

## 5、自动填充功能

有些时候我们可能会有这样的需求，插入或者更新数据时，希望有些字段可以自动填充数据，比如密码、version等。在MP中提供了这样的功能，可以实现自动填充。

### 5.1、添加@TableField注解

```
1 @TableField(fill = FieldFill.INSERT) //插入数据时进行填充
2 private String password;
```

为password添加自动填充功能，在新增数据时有效。

FieldFill提供了多种模式选择：

```
1 public enum FieldFill {
2     /**
3      * 默认不处理
4      */
5     DEFAULT,
6     /**
7      * 插入时填充字段
8      */
9     INSERT,
10    /**
11     * 更新时填充字段
12     */
13    UPDATE,
14    /**
15     * 插入和更新时填充字段
16     */
17    INSERT_UPDATE
18 }
```

### 5.2、编写MyMetaObjectHandler

```
1 package cn.itcast.mp.handler;
2
3 import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
4 import org.apache.ibatis.reflection.MetaObject;
5 import org.springframework.stereotype.Component;
6
7 @Component
8 public class MyMetaObjectHandler implements MetaObjectHandler {
```



```
9
10 @Override
11 public void insertFill(MetaObject metaObject) {
12     Object password = getFieldValByName("password", metaObject);
13     if(null == password){
14         //字段为空，可以进行填充
15         setFieldValByName("password", "123456", metaObject);
16     }
17 }
18
19 @Override
20 public void updateFill(MetaObject metaObject) {
21
22 }
23 }
```

### 5.3、测试

```
1 @Test
2 public void testInsert(){
3     User user = new User();
4     user.setName("关羽");
5     user.setUserName("guanyu");
6     user.setAge(30);
7     user.setEmail("guanyu@itast.cn");
8     user.setVersion(1);
9
10    int result = this.userMapper.insert(user);
11    System.out.println("result = " + result);
12 }
```

结果：

id	user_name	password	name	age	email	version
2	lisi	123456	李四	30	test2@itcast.cn	2
3	wangwu	123456	王五	20	test3@itcast.cn	1
4	zhaoliu	123456	赵六	20	test4@itcast.cn	1
5	sunqi	123456	孙七	20	test5@itcast.cn	1
6	caocao	123456	曹操	20	test@itcast.cn	1
8	liubei	123456	刘备	20	liubei@itcast.cn	1
9	caocao	123456	曹操	20	test@itcast.cn	1
14	guanyu	123456	关羽	30	guanyu@itast.cn	1

## 6、逻辑删除

开发系统时，有时候在实现功能时，删除操作需要实现逻辑删除，所谓逻辑删除就是将数据标记为删除，而并非真正的物理删除（非DELETE操作），查询时需要携带状态条件，确保被标记的数据不被查询到。这样做的目的就是避免数据被真正的删除。

MP就提供了这样的功能，方便我们使用，接下来我们一起学习下。

## 6.1、修改表结构

为tb\_user表增加deleted字段，用于表示数据是否被删除，1代表删除，0代表未删除。

```
1 ALTER TABLE `tb_user`  
2 ADD COLUMN `deleted` int(1) NULL DEFAULT 0 COMMENT '1代表删除，0代表未删除' AFTER  
   `version`;
```

同时，也修改User实体，增加deleted属性并且添加@TableLogic注解：

```
1 @TableLogic  
2 private Integer deleted;
```

## 6.2、配置

application.properties：

```
1 # 逻辑已删除值(默认为 1)  
2 mybatis-plus-global-config.db-config.logic-delete-value=1  
3 # 逻辑未删除值(默认为 0)  
4 mybatis-plus-global-config.db-config.logic-not-delete-value=0
```

## 6.3、测试

```
1 @Test  
2 public void testDeleteById(){  
3     this.userMapper.deleteById(2L);  
4 }
```

执行的SQL：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Preparing: UPDATE  
   tb_user SET deleted=1 WHERE id=? AND deleted=0  
2 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Parameters: 2(Long)  
3 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] <== Updates: 1
```

id	user_name	password	name	age	email	version	deleted
2	lisi	123456	李四	30	test2@itcast.cn	2	1
3	wangwu	123456	王五	20	test3@itcast.cn	1	0
4	zhaoliu	123456	赵六	20	test4@itcast.cn	1	0
5	sunqi	123456	孙七	20	test5@itcast.cn	1	0
6	caocao	123456	曹操	20	test@itcast.cn	1	0
8	liubei	123456	刘备	20	liubei@itcast.cn	1	0
9	caocao	123456	曹操	20	test@itcast.cn	1	0
14	guanyu	123456	关羽	30	guanyu@itast.cn	1	0

测试查询：

```
1 @Test
2 public void testSelectById(){
3     User user = this.userMapper.selectById(2L);
4     System.out.println(user);
5 }
```

执行的SQL：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Preparing: SELECT
id,user_name,password,name,age,email,version,deleted FROM tb_user WHERE id=? AND
deleted=0
2 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Parameters: 2(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] <== Total: 0
```

可见，已经实现了逻辑删除。

## 7、通用枚举

解决了繁琐的配置，让 mybatis 优雅的使用枚举属性！

### 7.1、修改表结构

```
1 ALTER TABLE `tb_user`
2 ADD COLUMN `sex` int(1) NULL DEFAULT 1 COMMENT '1-男, 2-女' AFTER `deleted`;
```

### 7.2、定义枚举

```
1 package cn.itcast.mp.enums;
2
3 import com.baomidou.mybatisplus.core.enums.IEnum;
4 import com.fasterxml.jackson.annotation.JsonValue;
5
6 public enum SexEnum implements IEnum<Integer> {
7
8     MAN(1, "男"),
```



```
9      WOMAN(2, "女");
10
11      private int value;
12      private String desc;
13
14      SexEnum(int value, String desc) {
15          this.value = value;
16          this.desc = desc;
17      }
18
19      @Override
20      public Integer getValue() {
21          return this.value;
22      }
23
24      @Override
25      public String toString() {
26          return this.desc;
27      }
28  }
```

## 7.3、配置

```
1  # 枚举包扫描
2  mybatis-plus.type-enums-package=cn.itcast.mp.enums
```

## 7.4、修改实体

```
1  private SexEnum sex;
```

## 7.5、测试

测试插入数据：

```
1  @Test
2  public void testInsert(){
3      User user = new User();
4      user.setName("貂蝉");
5      user.setUserName("diaochan");
6      user.setAge(20);
7      user.setEmail("diaochan@itast.cn");
8      user.setVersion(1);
9      user.setSex(SexEnum.WOMAN);
10
11      int result = this.userMapper.insert(user);
12      System.out.println("result = " + result);
13  }
```

SQL：



```

1 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] ==> Preparing: INSERT INTO
  tb_user ( user_name, password, name, age, email, version, sex ) VALUES ( ?, ?, ?, ?, ?,
  ?, ? )
2 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] ==> Parameters:
  diaochan(String), 123456(String), 貂蝉(String), 20(Integer), diaochan@itast.cn(String),
  1(Integer), 2(Integer)
3 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] <== Updates: 1
    
```

id	user_name	password	name	age	email	version	deleted	sex
2	lisi	123456	李四	30	test2@itcast.cn	2	0	2
3	wangwu	123456	王五	20	test3@itcast.cn	1	0	1
4	zhaoliu	123456	赵六	20	test4@itcast.cn	1	0	1
5	sunqi	123456	孙七	20	test5@itcast.cn	1	0	1
6	caocao	123456	曹操	20	test@itcast.cn	1	0	1
8	liubei	123456	刘备	20	liubei@itcast.cn	1	0	1
9	caocao	123456	曹操	20	test@itcast.cn	1	0	1
14	guanyu	123456	关羽	30	guanyu@itast.cn	1	0	1
15	diaochan	123456	貂蝉	20	diaochan@itast.cn	1	0	2

查询：

```

1 @Test
2 public void testSelectById(){
3     User user = this.userMapper.selectById(2L);
4     System.out.println(user);
5 }
    
```

结果：

```

1 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Preparing: SELECT
  id,user_name,password,name,age,email,version,deleted,sex FROM tb_user WHERE id=? AND
  deleted=0
2 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Parameters: 2(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] <== Total: 1
4
5 User(id=2, userName=lisi, password=123456, name=李四, age=30, email=test2@itcast.cn,
  address=null, version=2, deleted=0, sex=女)
    
```

从测试可以看出，可以很方便的使用枚举了。

查询条件时也是有效的：



```
1  @Test
2  public void testSelectBySex() {
3      QueryWrapper<User> wrapper = new QueryWrapper<>();
4      wrapper.eq("sex", SexEnum.WOMAN);
5      List<User> users = this.userMapper.selectList(wrapper);
6      for (User user : users) {
7          System.out.println(user);
8      }
9  }
```

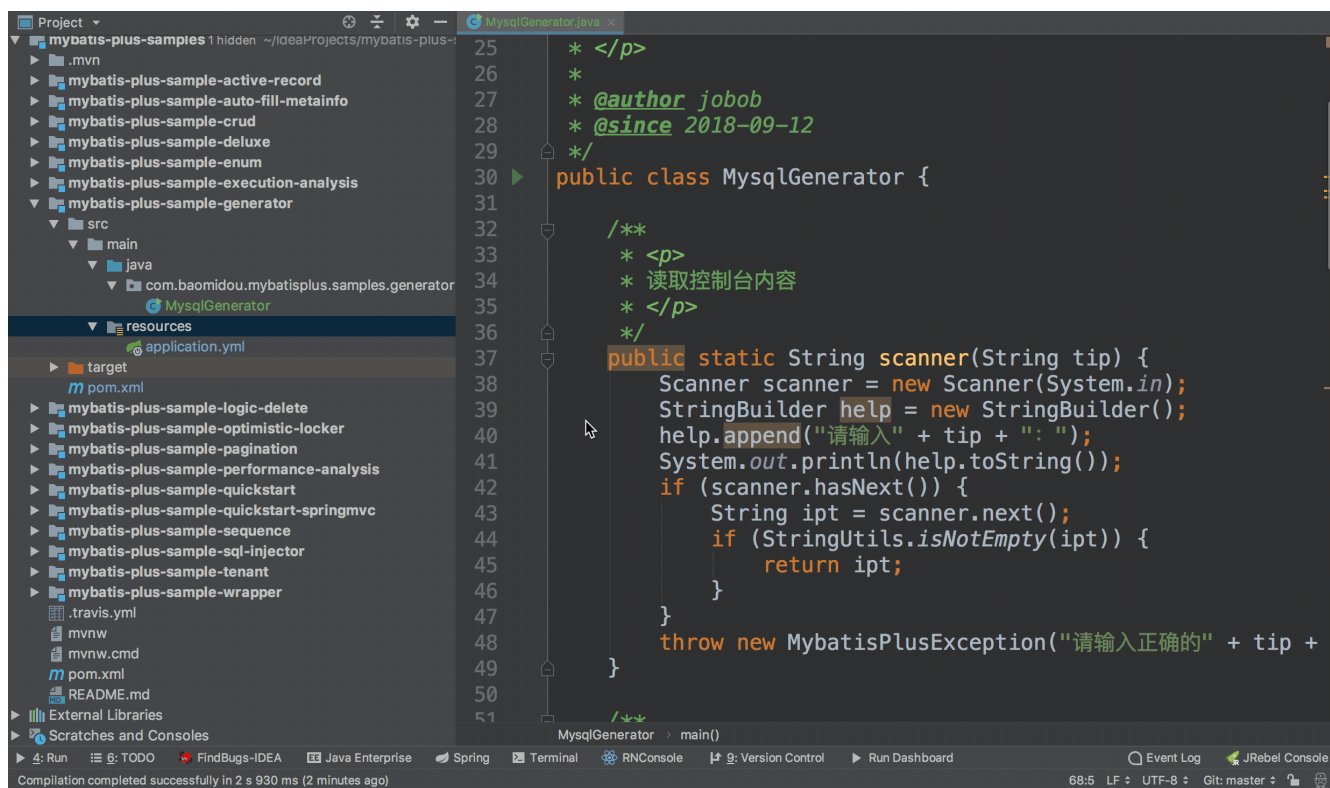
SQL :

```
1  [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Preparing: SELECT
id,user_name,password,name,age,email,version,deleted,sex FROM tb_user WHERE deleted=0
AND sex = ?
2  [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters: 2(Integer)
3  [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] <==      Total: 3
```

## 8、代码生成器

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Entity、Mapper、Mapper XML、Service、Controller 等各个模块的代码，极大的提升了开发效率。

效果：



### 8.1、创建工程

pom.xml :



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.1.4.RELEASE</version>
11    </parent>
12
13    <groupId>cn.itcast.mp</groupId>
14    <artifactId>itcast-mp-generator</artifactId>
15    <version>1.0-SNAPSHOT</version>
16
17    <dependencies>
18        <dependency>
19            <groupId>org.springframework.boot</groupId>
20            <artifactId>spring-boot-starter-test</artifactId>
21            <scope>test</scope>
22        </dependency>
23
24        <!--mybatis-plus的springboot支持-->
25        <dependency>
26            <groupId>com.baomidou</groupId>
27            <artifactId>mybatis-plus-boot-starter</artifactId>
28            <version>3.1.1</version>
29        </dependency>
30        <dependency>
31            <groupId>com.baomidou</groupId>
32            <artifactId>mybatis-plus-generator</artifactId>
33            <version>3.1.1</version>
34        </dependency>
35        <dependency>
36            <groupId>org.springframework.boot</groupId>
37            <artifactId>spring-boot-starter-freemarker</artifactId>
38        </dependency>
39        <!--mysql驱动-->
40        <dependency>
41            <groupId>mysql</groupId>
42            <artifactId>mysql-connector-java</artifactId>
43            <version>5.1.47</version>
44        </dependency>
45        <dependency>
46            <groupId>org.slf4j</groupId>
47            <artifactId>slf4j-log4j12</artifactId>
48        </dependency>
49
50    </dependencies>
51
52    <build>
```



```
53     <plugins>
54         <plugin>
55             <groupId>org.springframework.boot</groupId>
56             <artifactId>spring-boot-maven-plugin</artifactId>
57         </plugin>
58     </plugins>
59 </build>
60
61 </project>
```

## 8.2、代码

```
1 package cn.itcast.mp.generator;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.Scanner;
6
7 import com.baomidou.mybatisplus.core.exceptions.MybatisPlusException;
8 import com.baomidou.mybatisplus.core.toolkit.StringPool;
9 import com.baomidou.mybatisplus.core.toolkit.StringUtils;
10 import com.baomidou.mybatisplus.generator.AutoGenerator;
11 import com.baomidou.mybatisplus.generator.InjectionConfig;
12 import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
13 import com.baomidou.mybatisplus.generator.config.FileOutConfig;
14 import com.baomidou.mybatisplus.generator.config.GlobalConfig;
15 import com.baomidou.mybatisplus.generator.config.PackageConfig;
16 import com.baomidou.mybatisplus.generator.config.StrategyConfig;
17 import com.baomidou.mybatisplus.generator.config.TemplateConfig;
18 import com.baomidou.mybatisplus.generator.config.po.TableInfo;
19 import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
20 import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;
21
22 /**
23  * <p>
24  * mysql 代码生成器演示例子
25  * </p>
26  */
27 public class MysqlGenerator {
28
29     /**
30      * <p>
31      * 读取控制台内容
32      * </p>
33      */
34     public static String scanner(String tip) {
35         Scanner scanner = new Scanner(System.in);
36         StringBuilder help = new StringBuilder();
37         help.append("请输入" + tip + " : ");
38         System.out.println(help.toString());
39         if (scanner.hasNext()) {
40             String ipt = scanner.next();
```



```
41         if (StringUtils.isEmpty(ipt)) {
42             return ipt;
43         }
44     }
45     throw new MybatisPlusException("请输入正确的" + tip + "!");
46 }
47
48 /**
49  * RUN THIS
50  */
51 public static void main(String[] args) {
52     // 代码生成器
53     AutoGenerator mpg = new AutoGenerator();
54
55     // 全局配置
56     GlobalConfig gc = new GlobalConfig();
57     String projectPath = System.getProperty("user.dir");
58     gc.setOutputDir(projectPath + "/src/main/java");
59     gc.setAuthor("itcast");
60     gc.setOpen(false);
61     mpg.setGlobalConfig(gc);
62
63     // 数据源配置
64     DataSourceConfig dsc = new DataSourceConfig();
65     dsc.setUrl("jdbc:mysql://127.0.0.1:3306/mp?
66 useUnicode=true&useSSL=false&characterEncoding=utf8");
67     // dsc.setSchemaName("public");
68     dsc.setDriverName("com.mysql.jdbc.Driver");
69     dsc.setUsername("root");
70     dsc.setPassword("root");
71     mpg.setDataSource(dsc);
72
73     // 包配置
74     PackageConfig pc = new PackageConfig();
75     pc.setModuleName(scanner("模块名"));
76     pc.setParent("cn.itcast.mp.generator");
77     mpg.setPackageInfo(pc);
78
79     // 自定义配置
80     InjectionConfig cfg = new InjectionConfig() {
81         @Override
82         public void initMap() {
83             // to do nothing
84         }
85     };
86     List<FileOutConfig> focList = new ArrayList<>();
87     focList.add(new FileOutConfig("/templates/mapper.xml.ftl") {
88         @Override
89         public String outputFile(TableInfo tableInfo) {
90             // 自定义输入文件名称
91             return projectPath + "/itcast-mp-
92 generator/src/main/resources/mapper/" + pc.getModuleName()
```



```

91         + "/" + tableInfo.getEntityName() + "Mapper" +
StringPool.DOT_XML;
92     }
93     });
94     cfg.setFileOutConfigList(focList);
95     mpg.setCfg(cfg);
96     mpg.setTemplate(new TemplateConfig().setXml(null));
97
98     // 策略配置
99     StrategyConfig strategy = new StrategyConfig();
100    strategy.setNaming(NamingStrategy.underline_to_camel);
101    strategy.setColumnNaming(NamingStrategy.underline_to_camel);
102    //
strategy.setSuperEntityClass("com.baomidou.mybatisplus.samples.generator.common.BaseEntity");
103    strategy.setEntityLombokModel(true);
104    //
strategy.setSuperControllerClass("com.baomidou.mybatisplus.samples.generator.common.BaseController");
105    strategy.setInclude(scanner("表名"));
106    strategy.setSuperEntityColumns("id");
107    strategy.setControllerMappingHyphenStyle(true);
108    strategy.setTablePrefix(pc.getModuleName() + "_");
109    mpg.setStrategy(strategy);
110    // 选择 freemarker 引擎需要指定如下加，注意 pom 依赖必须有！
111    mpg.setTemplateEngine(new FreemarkerTemplateEngine());
112    mpg.execute();
113    }
114
115    }

```

## 8.3、测试

请输入模块名：

user

请输入表名：

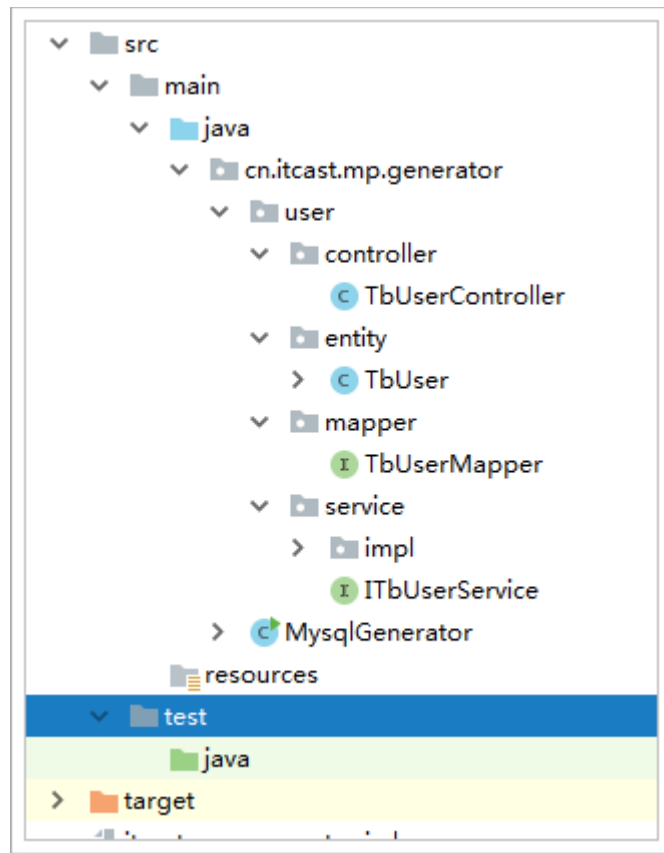
tb\_user

```

16:38:30.403 [main] DEBUG com.baomidou.mybatisplus.generator.AutoGenerator - =====准备生成
16:38:30.902 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录: [F:\code
16:38:30.902 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录: [F:\code
16:38:30.903 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录: [F:\code
16:38:30.904 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录: [F:\code
16:38:30.904 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 创建目录: [F:\code
log4j:WARN No appenders could be found for logger (freemarker.cache).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
16:38:31.149 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/en
16:38:31.156 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模板:/templates/ma
16:38:31.161 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模
16:38:31.164 [main] DEBUG com.baomidou.mybatisplus.generator.engine.AbstractTemplateEngine - 模

```

代码已生成：



实体对象：



```
@Accessors(chain = true)
public class TbUser implements Serializable {

    private static final long serialVersionUID = 1L;

    /**
     * 用户名
     */
    private String userName;

    /**
     * 密码
     */
    private String password;

    /**
     * 姓名
     */
    private String name;

    /**
     * 年龄
     */
    private Integer age;

    /**
     * 邮箱
     */
    private String email;
```

## 9、MybatisX 快速开发插件

MybatisX 是一款基于 IDEA 的快速开发插件，为效率而生。

安装方法：打开 IDEA，进入 File -> Settings -> Plugins -> Browse Repositories，输入 `mybatisx` 搜索并安装。

功能：

- Java 与 XML 调回跳转
- Mapper 方法自动生成 XML





```
1 package com.baomidou.springboot.mapper;
2
3 +import ...
4
5
6
7
8
9
10 /**
11  * User 表数据库控制层接口
12  */
13 public interface UserMapper extends SuperMapper<User> {
14
15     /**
16      * 自定义注入方法
17      */
18     int deleteAll();
19
20     @Select("select test_id as id, name, age, test_type from user")
21     List<User> selectListBySQL();
22
23 }
```