MyBatis Plus 学习记录和总结

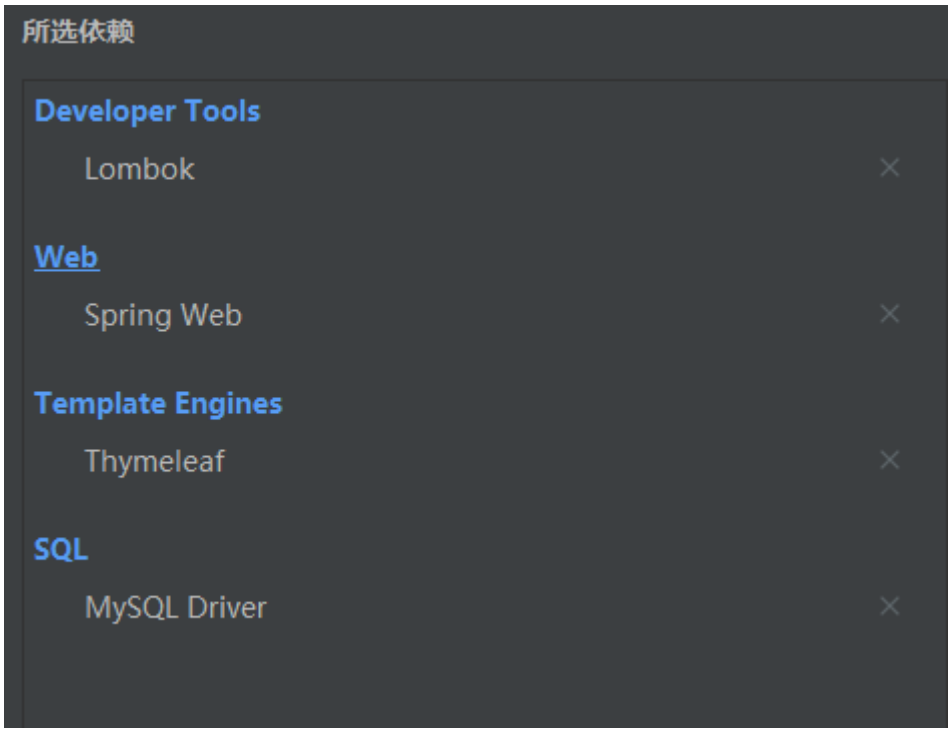
MyBatis Plus （简称 MP）是一个国产的框架，是一个基于 MyBatis 的增强工具。在 MyBatis 的基础上只做增强不做改变，为简化开发，提高效率而生。

MyBatis Plus 官网：https://mp.baomidou.com/

注：一定要去官网读一下 MP 的相关文档，了解一下他的功能和特性。

一、快速上手

- 创建 Spring Boot 官方创建组件创建工程，选择相应组件:



- pom.xml 中手动加入 MyBatis Plus 的依赖

```
<!-- Mybatis Plus -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-
starter</artifactId>
    <version>3.3.2</version>
</dependency>
```

- 新建 application.yml， 配置数据库数据源

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: admin
    password: 123
    url: jdbc:mysql://localhost:3305/mybatis_demo?
useUnicode=true&characterEncoding=utf8&serverTimezo
ne=UTC&allowMultiQueries=true
```

- 创建 User 实体类

```
package com.gloryh.mybatisplus.entity;

import lombok.Data;

/**
 * User 实体类
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
@Data
public class User {
    private Integer id;
    private String username;
    private String password;
    private Integer age;
}
```

- 创建 mapper 接口

```
package com.gloryh.mybatisplus.mapper;

import
com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.gloryh.mybatisplus.entity.User;

/**
 * User 实体类 Mapper 接口
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
public interface UserMapper extends
BaseMapper<User> {
}
```

- 对应的 SpringApplication 启动类添加 Spring Boot 的入口注解 和 Mapper 扫描注解

```
package com.gloryh.mybatisplus;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootAp
plication;

@SpringBootApplication
@MapperScan("com.gloryh.mybatisplus.mapper")
public class MybatisplusApplication {

    public static void main(String[] args) {

 SpringApplication.run(MybatisplusApplication.class
, args);
    }

}
```

- 测试

```
package com.gloryh.mybatisplus.mapper;

import org.junit.jupiter.api.Test;
import
org.springframework.beans.factory.annotation.Autowi
red;
import
org.springframework.boot.test.context.SpringBootTes
t;


/**
 * UserMapper 接口 测试类
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
@SpringBootTest
class UserMapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    void Test(){

 userMapper.selectList(null).forEach(System.out::pr
intln);
    }
}
```

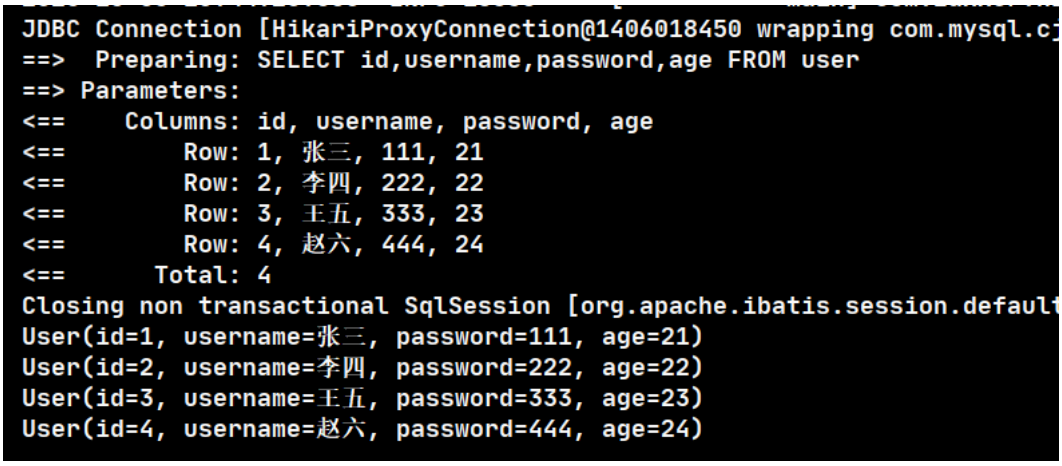
测试结果：

```
User(id=1, username=张三, password=111, age=21)
User(id=2, username=李四, password=222, age=22)
User(id=3, username=王五, password=333, age=23)
User(id=4, username=赵六, password=444, age=24)
```

- 如果想要看到 SQL 语句，需要在配置及文件中进行配置

```yaml
mybatis-plus:
  configuration:
    log-impl:
org.apache.ibatis.logging.stdout.StdOutImpl
```

查看命令行测试结果：



二、常用注解

- @TableName

  映射数据库表名，使用该注解可以使得类名和数据库表名在不一致时也可以完成映射。

- @TableId

  映射数据表主键名，用法与前者一致。

  > value 属性

  映射主键字段名

  > type

  设置主键的类型，生成策略。

  他提供一个枚举类型（三个已被淘汰）：

```
AUTO(0),
NONE(1),
INPUT(2),
ASSIGN_ID(3),
ASSIGN_UUID(4),
/** @deprecated */
@Deprecated
```

```
ID_WORKER(3),
/** @deprecated */
@Deprecated
ID_WORKER_STR(3),
/** @deprecated */
@Deprecated
UUID(4);
```

值	描述
AUTO	数据库自增
NONE	MP set主键，雪花算法实现
INPUT	需要开发者手动赋值
ASSIGN_ID	MP 分配 ID，类型：Long、Integer、String
ASSIGN_UUID	分配 UUID，主键也必须为 String 类型

- @TabledField

  映射数据表非主键字段名，用法与前者一致。

  > value

  映射非主键字段名

  > exist

  该属性用于表示是否为数据库对应表内的字段，如果实体类成员变量在数据库中没有对应的字段，可以使用该属性的 false 关键字 忽略 和数据表中及进行映射。

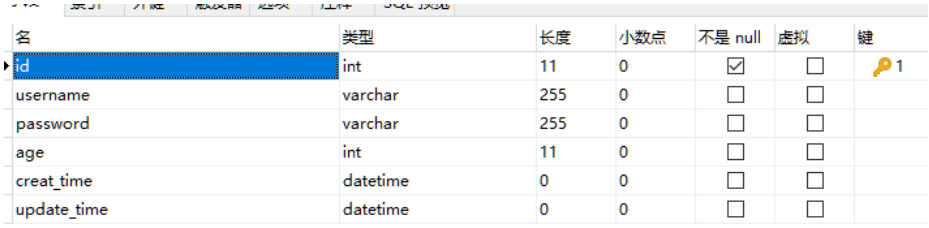
  > select

  该属性用于是否获取数据库中对应的字段的值，false则不获取

  > fill

  该属性用于表示 fill 是否自动填充，将对象存入数据库的时候，由 MyBatis Plus 自动给某些字段赋值 ， 例如订单信息里的创建时间和修改时间，就可以自动填充。

    - 1、在数据表中添加字段 creat_time 、update_time



名	类型	长度	小数点	不是 null	虚拟	键
id	int	11	0	☑	☐	1
username	varchar	255	0	☐	☐	
password	varchar	255	0	☐	☐	
age	int	11	0	☐	☐	
creat_time	datetime	0	0	☐	☐	
update_time	datetime	0	0	☐	☐	

- 2、在实体类中完成映射并处理好触发时机（MP 会自动将驼峰命名转化为下划线+小写的方式与数据库字段对应）

```
package com.gloryh.mybatisplus.entity;

import
com.baomidou.mybatisplus.annotation.*;
import lombok.Data;

import java.util.Date;

/**
 * User 实体类
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
@Data
@TableName("user")
public class User {
    @TableId(type = IdType.AUTO)
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    @TableField(fill = FieldFill.INSERT)
    private Date creatTime;
    @TableField(fill =
FieldFill.INSERT_UPDATE)
    private Date updateTime;
}
```

- 3、创建自动填充处理器

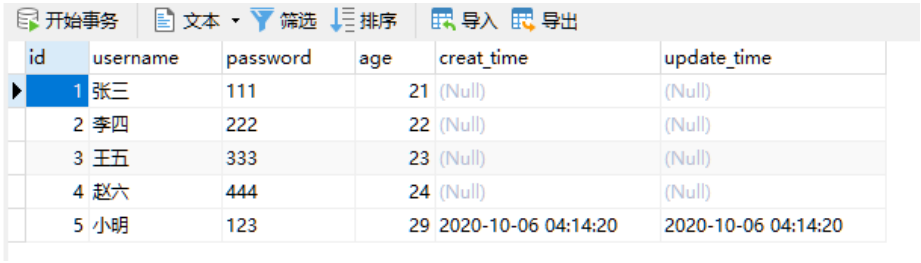
```
package com.gloryh.mybatisplus.handler;
```

```
import
com.baomidou.mybatisplus.core.handlers.MetaO
bjectHandler;
import
org.apache.ibatis.reflection.MetaObject;
import
org.springframework.stereotype.Component;

import java.util.Date;

/**
 * 自动填充处理器
 *
 * @author 黄光辉
 * @since 2020/10/6
 **/
@Component
public class MyMetaObjectHandler implements
MetaObjectHandler {
    @Override
    public void insertFill(MetaObject
metaObject) {
        //插入时进行填充

 this.setFieldValByName("creatTime",new
Date(),metaObject);

 this.setFieldValByName("updateTime",new
Date(),metaObject);
    }

    @Override
    public void updateFill(MetaObject
metaObject) {
        //更新时进行填充

 this.setFieldValByName("updateTime",new
Date(),metaObject);
    }
}
```

- 4、测试

```
@Test
void save(){
    User user =new User();
    user.setUsername("小明");
    user.setPassword("123");
    user.setAge(29);
    userMapper.insert(user);
}
```

运行：



id	username	password	age	creat_time	update_time
1	张三	111	21	(Null)	(Null)
2	李四	222	22	(Null)	(Null)
3	王五	333	23	(Null)	(Null)
4	赵六	444	24	(Null)	(Null)
5	小明	123	29	2020-10-06 04:14:20	2020-10-06 04:14:20

修改自测。

> @Version

标记乐观锁，通过 version 字段 来保证数据的安全性，当修改数据时，会以 version 作为条件，当条件成立时才会修改成功。

- 1、在数据表中添加 version 字段，默认值为 1
- 2、实体类添加 version 成员变量，并添加`@Version`注解

```
package com.gloryh.mybatisplus.entity;

import
com.baomidou.mybatisplus.annotation.*;
import lombok.Data;

import java.util.Date;

/**
 * User 实体类
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
```

```
@Data
@TableName("user")
public class User {
    @TableId(type = IdType.AUTO)
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    @TableField(fill = FieldFill.INSERT)
    private Date creatTime;
    @TableField(fill =
FieldFill.INSERT_UPDATE)
    private Date updateTime;
    @Version
    private Integer version;
}
```

- 3、注册一个配置类

```
package com.gloryh.mybatisplus.config;

import
com.baomidou.mybatisplus.extension.plugins.O
ptimisticLockerInterceptor;
import
org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Confi
guration;

/**
 * 配置类
 *
 * @author 黄光辉
 * @since 2020/10/6
 **/
@Configuration
public class MyBatisPlusConfig {

    @Bean
```

```
    public OptimisticLockerInterceptor
optimisticLockerInterceptor(){
        //返回一个乐观锁对象
        return new
OptimisticLockerInterceptor();
    }
}
```

- 4、测试

  不冲突：

```
@Test
void update() {
    User user = userMapper.selectById(5);
    user.setUsername("马七");
    userMapper.updateById(user);
}
```

  运行：

  执行的语句：

```
UPDATE user SET username=?, password=?,
age=?, creat_time=?, update_time=?,
version=? WHERE id=? AND version=?
```

  通过语句可以发现我们的version被作为了查询条件同时被进行了修改，这就体现了乐观所得思想。

  冲突时：

```
@Test
void update() {
    User user = userMapper.selectById(5);
    user.setUsername("马七");
    User user1 = userMapper.selectById(5);
    user1.setUsername("马八");
    userMapper.updateById(user1);
    userMapper.updateById(user);
}
```

  运行：

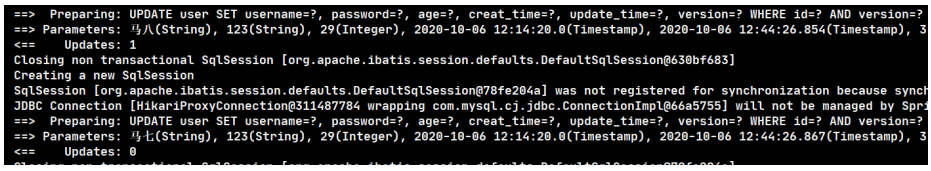
根据先后顺序，运行完成后 username 为 马七，version 为 4.

但是，查看结果：

2	李四	222	22	(Null)	(Null)	1
3	王五	333	23	(Null)	(Null)	1
4	赵六	444	24	(Null)	(Null)	1
5	马八	123	29	2020-10-06 04:14:20	2020-10-06 04:44:27	3

此时 username 为 马八，version 为 3，这就说明第二条语句未执行，或执行完 不对数据进行改变。

查看控制台语句：



可以看到一条执行影响一行，另一条执行影响0行，这是因为在第一局执行完，version 的值 被修改为 3，但是在第二条数据修改时，version 在数据库中已经为 3，但修改条件告诉他 要修改的 version 为 2，所以第二条语句的影响行数 为 0 行。

> @EnumValue

通用枚举类注解，将数据库字段映射成书体类的枚举型成员变量。

- 给实体类加一个状态成员变量 status ,枚举类型（值为1 代表工作状态，值为0代表休息状态）。

  创建枚举类：

```
package com.gloryh.mybatisplus.enums;

import
com.baomidou.mybatisplus.annotation.EnumValu
e;

/**
 * 用户实体类 status成员变量 枚举类型
 *
 * @author 黄光辉
 * @since 2020/10/8
 **/
```

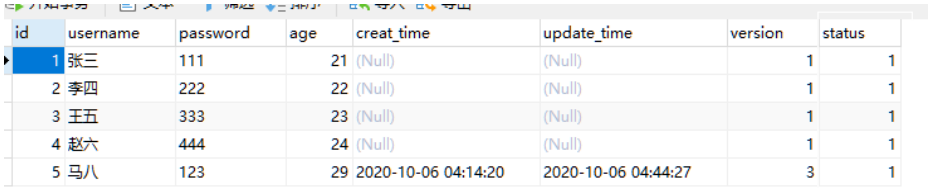
```
public enum  StatusEnum {
    WORK(1,"上班状态"),
    REST(0,"休息状态");

    StatusEnum(Integer code, String msg) {
        this.code = code;
        this.msg = msg;
    }

    @EnumValue
    private Integer code;
    private String msg;

}
```

在实体类中添加该枚举成员变量（命名要与数据库字段一致或使用注释进行映射）：

```
package com.gloryh.mybatisplus.entity;

import
com.baomidou.mybatisplus.annotation.*;
import
com.gloryh.mybatisplus.enums.StatusEnum;
import lombok.Data;

import java.util.Date;

/**
 * User 实体类
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
@Data
@TableName("user")
public class User {
    @TableId(type = IdType.AUTO)
    private Integer id;
    private String username;
    private String password;
```

```
    private Integer age;
    @TableField(fill = FieldFill.INSERT)
    private Date creatTime;
    @TableField(fill =
FieldFill.INSERT_UPDATE)
    private Date updateTime;
    @Version
    private Integer version;

    private StatusEnum status;
}
```

- 数据库中字段对应。

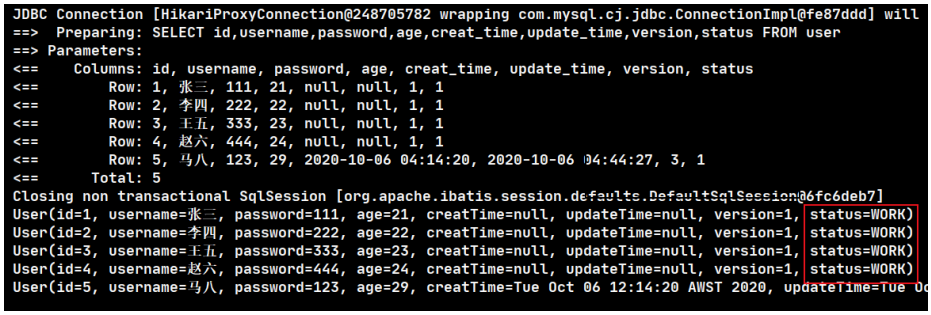


id	username	password	age	creat_time	update_time	version	status
1	张三	111	21	(Null)	(Null)	1	1
2	李四	222	22	(Null)	(Null)	1	1
3	王五	333	23	(Null)	(Null)	1	1
4	赵六	444	24	(Null)	(Null)	1	1
5	马八	123	29	2020-10-06 04:14:20	2020-10-06 04:44:27	3	1

- 配置文件 application.yml 中进行包配置

```
mybatis-plus:
  configuration:
    log-impl:
org.apache.ibatis.logging.stdout.StdOutImpl
  type-enums-package:
    com.gloryh.mybatisplus.enums
```

- 测试（查询全部）



```
JDBC Connection [HikariProxyConnection@248705782 wrapping com.mysql.cj.jdbc.ConnectionImpl@fe87ddd] will
==>  Preparing: SELECT id,username,password,age,creat_time,update_time,version,status FROM user
==> Parameters:
<==    Columns: id, username, password, age, creat_time, update_time, version, status
<==        Row: 1, 张三, 111, 21, null, null, 1, 1
<==        Row: 2, 李四, 222, 22, null, null, 1, 1
<==        Row: 3, 王五, 333, 23, null, null, 1, 1
<==        Row: 4, 赵六, 444, 24, null, null, 1, 1
<==        Row: 5, 马八, 123, 29, 2020-10-06 04:14:20, 2020-10-06 04:44:27, 3, 1
<==      Total: 5
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6fc6deb7]
User(id=1, username=张三, password=111, age=21, creatTime=null, updateTime=null, version=1, status=WORK)
User(id=2, username=李四, password=222, age=22, creatTime=null, updateTime=null, version=1, status=WORK)
User(id=3, username=王五, password=333, age=23, creatTime=null, updateTime=null, version=1, status=WORK)
User(id=4, username=赵六, password=444, age=24, creatTime=null, updateTime=null, version=1, status=WORK)
User(id=5, username=马八, password=123, age=29, creatTime=Tue Oct 06 12:14:20 AWST 2020, updateTime=Tue Oc
```

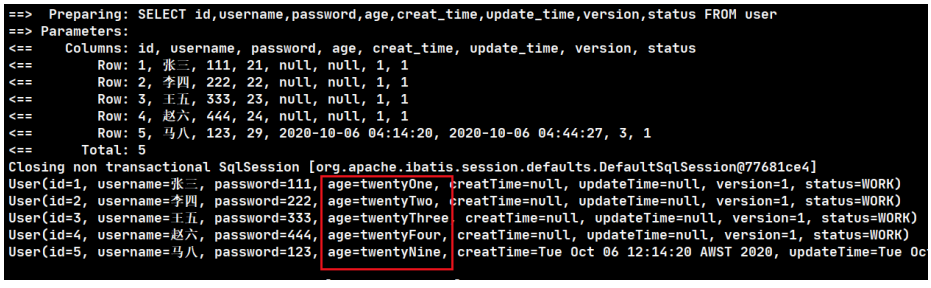
  可以看到，已完成 枚举类 的映射。

除此之外还可以使用接口的方式完成枚举类的映射（以 age 字段为例）。

- 创建枚举类，实现 IEnum 接口及其 getValue 方法

```java
package com.gloryh.mybatisplus.enums;

import
com.baomidou.mybatisplus.core.enums.IEnum;

/**
 * 年龄枚举类
 *
 * @author 黄光辉
 * @since 2020/10/8
 **/
public enum AgeEnum implements
IEnum<Integer> {
    twentyOne(21, "二十一"),
    twentyTwo(22, "二十二"),
    twentyThree(23, "二十三"),
    twentyFour(24, "二十四"),
    twentyNine(29, "二十九");

    private Integer code;
    private String msg;

    AgeEnum(Integer code, String msg) {
        this.code = code;
        this.msg = msg;
    }

    @Override
    public Integer getValue() {
        return this.code;
    }
}
```

- 实体类中修改 age 成员变量 为 AgeEnum 类型
- 查看结果

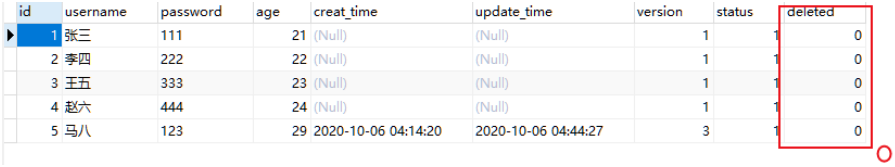


完成映射。

> @TableLogic

映射逻辑删除（假删除）。

- 数据库添加字段 deleted （值默认为 0，代表可以正常使用，值为 1 代表假删除，不可以正常使用）



id	username	password	age	creat_time	update_time	version	status	deleted
1	张三	111	21	(Null)	(Null)	1	1	0
2	李四	222	22	(Null)	(Null)	1	1	0
3	王五	333	23	(Null)	(Null)	1	1	0
4	赵六	444	24	(Null)	(Null)	1	1	0
5	马八	123	29	2020-10-06 04:14:20	2020-10-06 04:44:27	3	1	0

- 实体类添加对应的成员变量，并使用`@TableLogic`进行注释

```
package com.gloryh.mybatisplus.entity;

import
com.baomidou.mybatisplus.annotation.*;
import com.gloryh.mybatisplus.enums.AgeEnum;
import
com.gloryh.mybatisplus.enums.StatusEnum;
import lombok.Data;

import java.util.Date;

/**
 * User 实体类
 *
 * @author 黄光辉
 * @since 2020/10/5
 **/
@Data
@TableName("user")
public class User {
    @TableId(type = IdType.AUTO)
    private Integer id;
    private String username;
    private String password;
    private AgeEnum age;
    @TableField(fill = FieldFill.INSERT)
    private Date creatTime;
```

```
    @TableField(fill =
FieldFill.INSERT_UPDATE)
    private Date updateTime;
    @Version
    private Integer version;
    private StatusEnum status;
    @TableLogic
    private Integer deleted;
}
```

- 在 配置文件 application.yml 中添加对应配置

```
mybatis-plus:
  configuration:
    log-impl:
org.apache.ibatis.logging.stdout.StdOutImpl
  type-enums-package:
    com.gloryh.mybatisplus.enums
  global-config:
    db-config:
      logic-not-delete-value: 0
      logic-delete-value: 1
```

- 测试

```
@Test
void delete() {
    userMapper.deleteById(1);
}
```

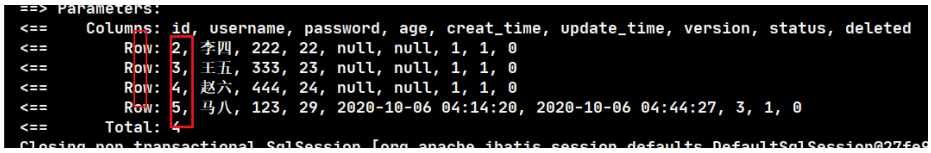
查看数据库（数据并没有被删除，但是对应 deleted 字段被修改为假删除状态）：

id	username	password	age	creat_time	update_time	version	status	deleted
1	张三	111	21	(Null)	(Null)	1	1	1
2	李四	222	22	(Null)	(Null)	1	1	0
3	王五	333	23	(Null)	(Null)	1	1	0
4	赵六	444	24	(Null)	(Null)	1	1	0
5	马八	123	29	2020-10-06 04:14:20	2020-10-06 04:44:27	3	1	0

查看命令行（并没有执行删除操作，而是执行的update操作）：

```
JDBC Connection [HikariProxyConnection@962969081 wrapping com.mysql.
==>  Preparing: UPDATE user SET deleted=1 WHERE id=? AND deleted=0
==> Parameters: 1(Integer)
<==    Updates: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defa
```

查询所有用户信息（查不到 id = 1 的信息，代表逻辑删除完成）：



三、CRUD 操作

1、查询

> SelectList 操作

```
@Test
void selectList() {
    //使用 QueryWrapper 进行条件查询
    QueryWrapper wrapper =new QueryWrapper<>();
    //null 代表不加任何条件查询，通常用语查询全部

 userMapper.selectList(null).forEach(System.out::println);
    //查询 数据库内 字段为 username ，值为 李四 的数据行
    wrapper.eq("username","李四");
    System.out.println(userMapper.selectList(wrapper));
    //多条件查询
    Map<String, Object> map = new HashMap<>();
    map.put("username","李四" );
    map.put("age", "22");
    wrapper.allEq(map);
    System.out.println(userMapper.selectList(wrapper));
    //年龄大于20(gt 为大于 ,lt 为小于, ne 为不等于 ,ge 为大于等于,le 为小于等与)
    wrapper.gt("age",20);
    System.out.println(userMapper.selectList(wrapper));
    //模糊查询like,同时提供左模糊（likeLeft）和右模糊（likeRight）的查询方法
    //username 中 有 李 字 的
    wrapper.like("username","李");
    System.out.println(userMapper.selectList(wrapper));
    //联合查询 inSql(查询 id 小于 4 且 age 大于 23 的 user)
    wrapper.inSql("id","SELECT id FROM user WHERE id < 4");
```

```
    wrapper.inSql("age","SELECT age FROM user WHERE age > 23");
    System.out.println(userMapper.selectList(wrapper));
    //排序（orderByASC 升序 orderByDesc 降序）
    wrapper.orderByDesc("age");
    System.out.println(userMapper.selectList(wrapper));
    //年龄大于 22 的 ，按年龄升序
    wrapper.orderByAsc("age");
    wrapper.having("age>22");
    System.out.println(userMapper.selectList(wrapper));
}
```

> selectById 操作

```
@Test
void selectById(){
    //单个主键查询
    System.out.println(userMapper.selectById(2));
    //多个主键查询

 userMapper.selectBatchIds(Arrays.asList(2,3,4,5)).forEach(System.out::println);
}
```

> selectByMap 操作

```
@Test
void selectByMap(){
    //类似于Wrapper 的 eq 操作，只能做等值判断
    Map<String, Object> map =new HashMap<>();
    map.put("name","李四");
    System.out.println(userMapper.selectByMap(map));
}
```

> selectCount 操作

```
@Test
void selectCount(){
    //统计操作，返回一个 Integer
    System.out.println(userMapper.selectCount(null));
}
```

> selectMaps 操作

```
@Test
void selectMaps(){
    //将查询的结果封装到 Map 中

 userMapper.selectMaps(null).forEach(System.out::println);
}
```

> selectPage 操作

第一步，进行分页配置

```
package com.gloryh.mybatisplus.config;

import
com.baomidou.mybatisplus.extension.plugins.OptimisticLocke
rInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.PaginationInter
ceptor;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;

/**
 * 配置类
 *
 * @author 黄光辉
 * @since 2020/10/6
 **/
@Configuration
public class MyBatisPlusConfig {

    /**
     * 乐观锁
     * @return 乐观锁对象
     */
    @Bean
    public OptimisticLockerInterceptor
optimisticLockerInterceptor(){
        //返回一个乐观锁对象
```

```
        return new OptimisticLockerInterceptor();
    }

    /**
     * 分页配置
     * @return 分页配置对象
     */
    @Bean
    public PaginationInterceptor paginationInterceptor(){
        return new PaginationInterceptor();
    }
}
```

第二步，使用 selectPage 方法

```
@Test
void selectPage(){
    //分页查询，要添加相关配置
    //每页2条，查询第一页
    Page<User> page =new Page<>(1,2);
    Page<User> result=userMapper.selectPage(page,null);
    System.out.println("每页最多条数"+result.getSize());
    System.out.println("数据库内总条数"+result.getTotal());
    //遍历当前页数据
    result.getRecords().forEach(System.out::println);
}
```

> selectMapsPage 操作

```
@Test
void selectMapsPage(){
    //即 将分页的结果封装到 Map 集合中。
    //每页2条，查询第一页
    Page<Map<String, Object>> page = new Page<>(1,2);
    Page<Map<String, Object>> result
=userMapper.selectMapsPage(page,null);
    //遍历当前页数据
    result.getRecords().forEach(System.out::println);
}
```

> selectObjs 操作

```
@Test
void selectObjs() {
    //返回一个 Object 集合 ，但集合内 只有 id

 userMapper.selectObjs(null).forEach(System.out::println);
}
```

> selectOne 操作

```
@Test
void selectOne(){
    //只获取一条数据库记录,前提是查询到的结果集只有一条记录
    QueryWrapper wrapper = new QueryWrapper();
    wrapper.eq("id",3);
    System.out.println(userMapper.selectOne(wrapper));
}
```

2、添加

```
@Test
void save() {
    User user = new User();
    user.setUsername("小明");
    user.setAge(AgeEnum.twentyOne);
    user.setPassword("123");
    userMapper.insert(user);
}
```

3、删除

```
@Test
void delete() {
    QueryWrapper wrapper =new QueryWrapper();
    //按 id 删除
    userMapper.deleteById(1);
    //按 id 批量删除
    userMapper.deleteBatchIds(Arrays.asList(1,5));
    // 按条件删除（年龄为21）
    wrapper.eq("age",21);
    userMapper.delete(wrapper);
```

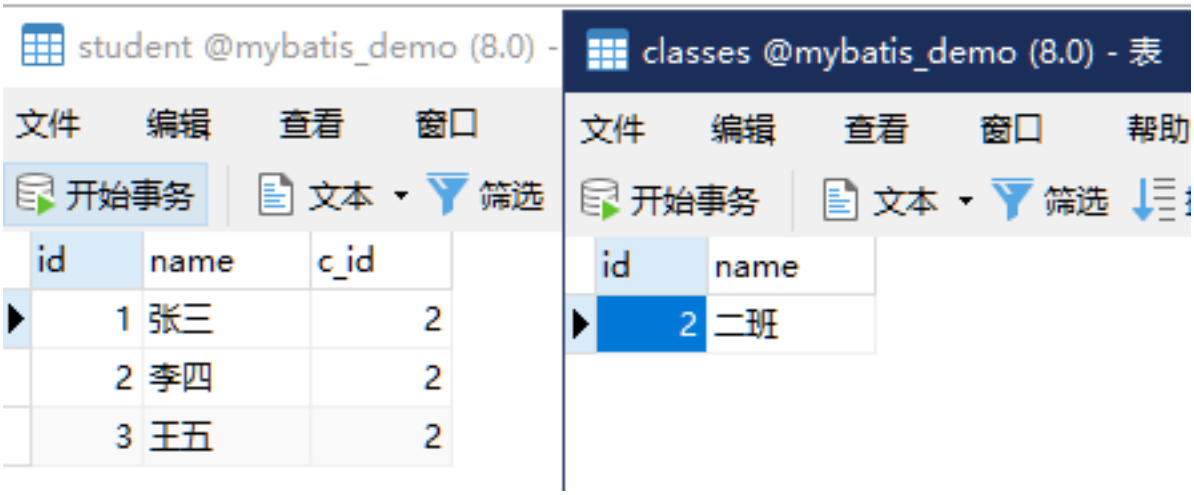
```
    //按条件（讲条件装进Map中）删除（年龄为21）
    Map<String, Object> map =new HashMap<>();
    map.put("age",21);
    userMapper.deleteByMap(map);

}
```

4、修改

```
@Test
void update() {
    User user = userMapper.selectById(5);
    //查询到后，通过 set 方法 再进行updateById进行修改。
    user.setUsername("马八");
    userMapper.updateById(user);
    ////查询到后,通过 set 方法修改后，,如果需要在加入新的条件，使用
QueryWrapper 加入条件， 再进行update进行修改。
    user=userMapper.selectById(5);
    user.setUsername("马八");
    QueryWrapper wrapper =new QueryWrapper();
    wrapper.eq("age",22);
    userMapper.update(user,wrapper);
}
```

5、自定义 SQL语句（多表关联查询）

用到的表：classes 班级表 和 student 学生表，关系为一对多（多对多操作基本类似）



对应班级实体类：

```java
package com.gloryh.mybatisplus.entity;

import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.Data;

/**
 * 班级实体类
 *
 * @author 黄光辉
 * @since 2020/10/8
 **/
@Data
@TableName("classes")
public class Classes {
    @TableId
    private Integer id;
    private String name;
}
```

对应的学生VO：

```java
package com.gloryh.mybatisplus.entity;

import lombok.Data;

/**
 * 学生信息表对应VO
 *
 * @author 黄光辉
 * @since 2020/10/8
 **/
@Data
public class StudentVO {
    private Integer id;
    private String name;
    private Integer cId;
    private String className;
}
```

用到的数据库查询语句：

```
SELECT * FROM student s,classes c WHERE s.c_id = c.id AND
c.id = 2;
```

查到的结果：

id	name	c_id	id(1)	name(1)
1	张三	2	2	二班
2	李四	2	2	二班
3	王五	2	2	二班

我们只需要得到 classes的name 和 p 里面的所有字段，所以数据库语句改为：

```
SELECT s.*,c.name as class_name FROM student s,classes c
WHERE s.c_id = c.id AND c.id = 2;
```

查到的结果：

id	name	c_id	class_name
1	张三	2	二班
2	李四	2	二班
3	王五	2	二班

同时，由于 MP 会根据驼峰命名规则自动将大写变成小写并在其前面追加下划线，这里的`c.name as class_name`就可以完成和 VO 中 className 成员变量的映射。

将结果集映射到 VO 中：

- 定义一个相关 Mapper 进行映射，并自定义映射方式

```
package com.gloryh.mybatisplus.mapper;

import
com.baomidou.mybatisplus.core.mapper.BaseMapper;
```

```
import com.gloryh.mybatisplus.entity.Classes;
import com.gloryh.mybatisplus.entity.StudentVO;
import org.apache.ibatis.annotations.Select;

import java.util.List;

/**
 * Classes 实体类 Mapper 接口
 *
 * @author 黄光辉
 * @since 2020/10/8
 **/
public interface ClassesMapper extends
BaseMapper<Classes> {
    @Select("SELECT s.*,c.name as class_name FROM
student s,classes c WHERE s.c_id = c.id AND c.id =
#{id}")
    List<StudentVO> students(Integer id);
}
```

- 测试类调用方法

```
package com.gloryh.mybatisplus.mapper;

import org.junit.jupiter.api.Test;
import
org.springframework.beans.factory.annotation.Autowi
red;
import
org.springframework.boot.test.context.SpringBootTes
t;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Classes 方法实现 测试类
 *
 * @author 黄光辉
 * @since 2020/10/8
 **/
@SpringBootTest
```

```
class ClassesMapperTest {

    @Autowired
    private ClassesMapper classesMapper;

    @Test
    void students() {

 classesMapper.students(2).forEach(System.out::println);
    }

}
```

- 运行查看结果：

  ![image-20201008153509399](C:\Users\admin\AppData\Roaming\Typora\typora-user-images\image-20201008153509399.png

四、MP 的自动生成

MP 可以根据数据表自动生成对应的实体类、Mapper、Service、ServletImpl、Controller

- pom.xml 中导入需要的依赖

```
<!-- MP 自动生成所需依赖 -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.4.0</version>
</dependency>

<!--MP自动生成所需模板相关依赖-->
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity-engine-core</artifactId>
    <version>2.2</version>
</dependency>
```

模板不仅限于 velocity，只是默认为 velocity，除此之外还有 Freemarker、Beetl

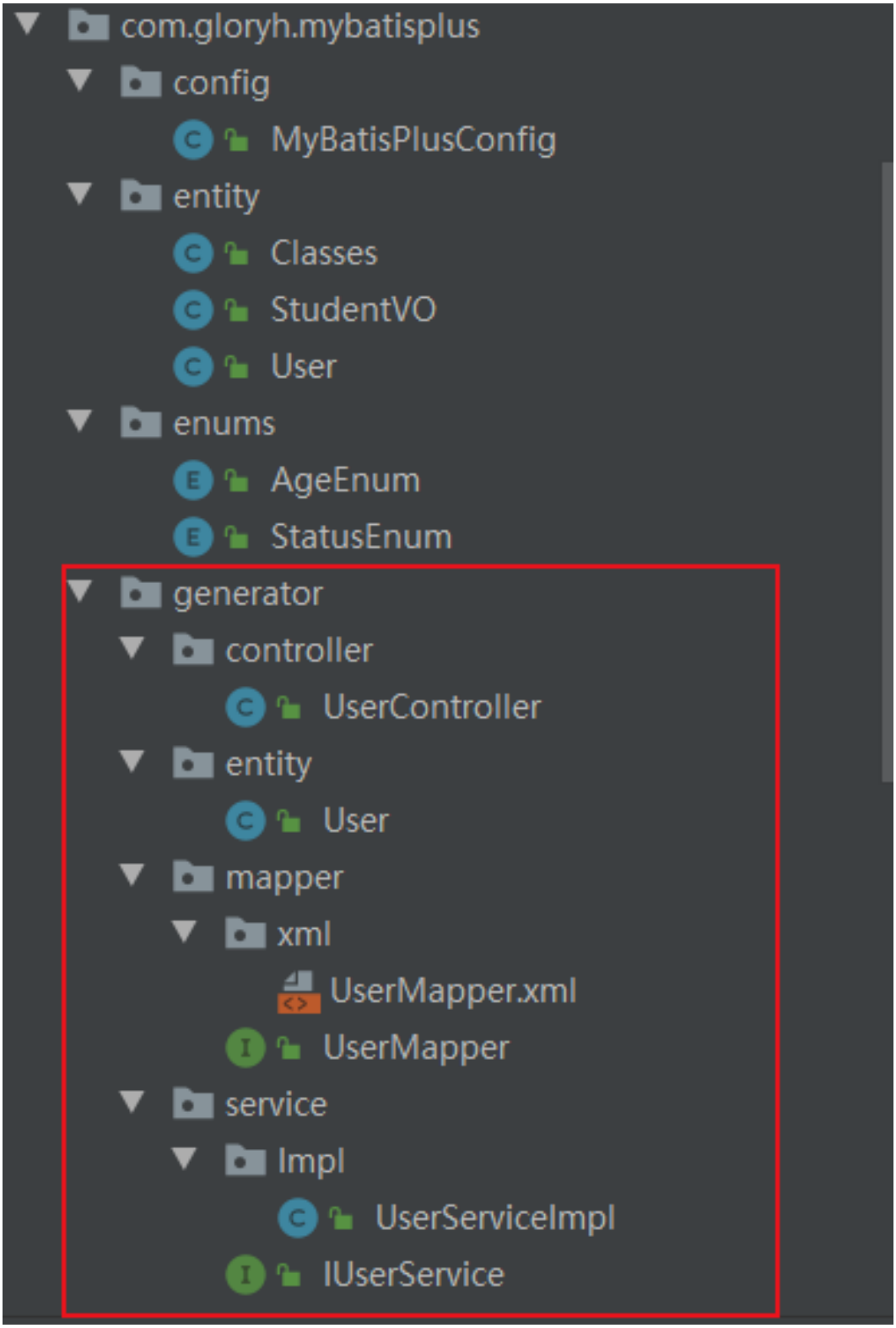
- 编写启动类

```
package com.gloryh.mybatisplus;

import com.baomidou.mybatisplus.annotation.DbType;
import
com.baomidou.mybatisplus.generator.AutoGenerator;
import
com.baomidou.mybatisplus.generator.config.DataSourc
eConfig;
import
com.baomidou.mybatisplus.generator.config.GlobalCon
fig;
import
com.baomidou.mybatisplus.generator.config.PackageCo
nfig;
import
com.baomidou.mybatisplus.generator.config.StrategyC
onfig;
import
com.baomidou.mybatisplus.generator.config.rules.Nam
ingStrategy;

/**
```

```
 * MP 自动生成启动类
 *
 * @author 黄光辉
 * @since 2020/10/10
 **/
public class Main {
    public static void main(String[] args) {
        //创建一个 generator 对象
        AutoGenerator autoGenerator = new
AutoGenerator();
        //配置数据源
        DataSourceConfig dataSourceConfig = new
DataSourceConfig();
        dataSourceConfig.setDbType(DbType.MYSQL);

 dataSourceConfig.setUrl("jdbc:mysql://localhost:33
06/test?useUnicode=true&characterEncoding=UTF-8");
        dataSourceConfig.setUsername("root");
        dataSourceConfig.setPassword(hgh");

 dataSourceConfig.setDriverName("com.mysql.cj.jdbc.
Driver");
        //装入 generator 对象

 autoGenerator.setDataSource(dataSourceConfig);
        //全局配置
        GlobalConfig globalConfig = new
GlobalConfig();

 globalConfig.setOutputDir(System.getProperty("user
.dir")+"/src/main/java");
        //创建完成后，是否打开文件夹
        globalConfig.setOpen(false);
        //设置创建完成后的作者名
        globalConfig.setAuthor("黄光辉");
        //装入generator对象

 autoGenerator.setGlobalConfig(globalConfig);
        //配置各类文件要存放的包信息
        PackageConfig packageConfig =new
PackageConfig();
```

```
        //父包名

 packageConfig.setParent("com.gloryh.mybatisplus");
        //所有文件要存放的包名
        packageConfig.setModuleName("generator");
        //设置各类型文件的包名
        packageConfig.setEntity("entity");
        packageConfig.setMapper("mapper");
        packageConfig.setController("controller");
        packageConfig.setService("service");

 packageConfig.setServiceImpl("service.Impl");
        //装入 generator 对象

 autoGenerator.setPackageInfo(packageConfig);
        //配置策略
        StrategyConfig strategyConfig =new
StrategyConfig();
        //是否给创建好的实体类添加Lombok注解
        strategyConfig.setEntityLombokModel(true);
        //创建实体类成员变量属性时使用驼峰式命名

 strategyConfig.setColumnNaming(NamingStrategy.unde
rline_to_camel);
        //装入generator对象
        autoGenerator.setStrategy(strategyConfig);

        //根据 generator 配置执行自动生成代码
        autoGenerator.execute();
    }
}
```

- 运行，查看结果



完成代码自动生成。

- 测试接口是否可用

  第一步，在启动类中加入刚刚生成的mapper扫描注解

```
package com.gloryh.mybatisplus;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
```

```
import
org.springframework.boot.autoconfigure.SpringBootAp
plication;

@SpringBootApplication
@MapperScan("com.gloryh.mybatisplus.generator.mappe
r")
public class MybatisplusApplication {

    public static void main(String[] args) {

 SpringApplication.run(MybatisplusApplication.class
, args);
    }

}
```

第二步，单元测试方法

```
package com.gloryh.mybatisplus.mapper;

import
com.gloryh.mybatisplus.generator.mapper.UserMapper;
import org.junit.jupiter.api.Test;
import
org.springframework.beans.factory.annotation.Autowi
red;
import
org.springframework.boot.test.context.SpringBootTes
t;

/**
 * 测试自动生成的mapper接口
 *
 * @author 黄光辉
 * @since 2020/10/11
 **/
@SpringBootTest
class UserMapperTest1 {

    @Autowired
```

```
    private UserMapper userMapper;

    @Test
    void test(){

 userMapper.selectList(null).forEach(System.out::println);
    }
}
```

第三步，applocatiuon.yml 修改配置文件的数据库信息

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: ddd
    url: jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=UTF-8

mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
  type-enums-package:
    com.gloryh.mybatisplus.enums
  global-config:
    db-config:
      logic-not-delete-value: 0
      logic-delete-value: 1
```

运行：

```
JDBC Connection [HikariProxyConnection@1722102020 wrapping com.mysql.cj.jdbc.ConnectionImpl@6d229b1c]
==>  Preparing: SELECT id,user_name,user_age FROM user
==> Parameters:
<==    Columns: id, user_name, user_age
<==        Row: 1, 夏天, 12
<==        Row: 2, 冬天, 22
<==      Total: 2
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@541179e7]
```

五、部署阿里云

> 部署准备

- 第一步，编写前后端交互方法

```
package
com.gloryh.mybatisplus.generator.controller;


import
com.gloryh.mybatisplus.generator.service.IUs
erService;
import
org.springframework.beans.factory.annotation
.Autowired;
import
org.springframework.web.bind.annotation.GetM
apping;
import
org.springframework.web.bind.annotation.Requ
estMapping;

import
org.springframework.stereotype.Controller;
import
org.springframework.web.servlet.ModelAndView
;

/**
 * <p>
 *  前端控制器
 * </p>
 *
 * @author 黄光辉
 * @since 2020-10-11
 */
@Controller
@RequestMapping("/generator/user")
public class UserController {

    @Autowired
    private IUserService userService;

    @GetMapping("/index")
```

```
    public ModelAndView index(){
        ModelAndView modelAndView=new
ModelAndView();
        modelAndView.setViewName("index");

 modelAndView.addObject("list",userService.l
ist());
        return modelAndView;
    }

}
```

- 第二步，创建对应的前端页面

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<table>
    <tr>
        <td>用户id：</td>
        <td>用户姓名：</td>
        <td>用户年龄：</td>
    </tr>
    <tr th:each="user:${list}">
        <td th:text="${user.id}"></td>
        <td th:text="${user.userName}"></td>
        <td th:text="${user.userAge}"></td>
    </tr>
</table>
</body>
</html>
```

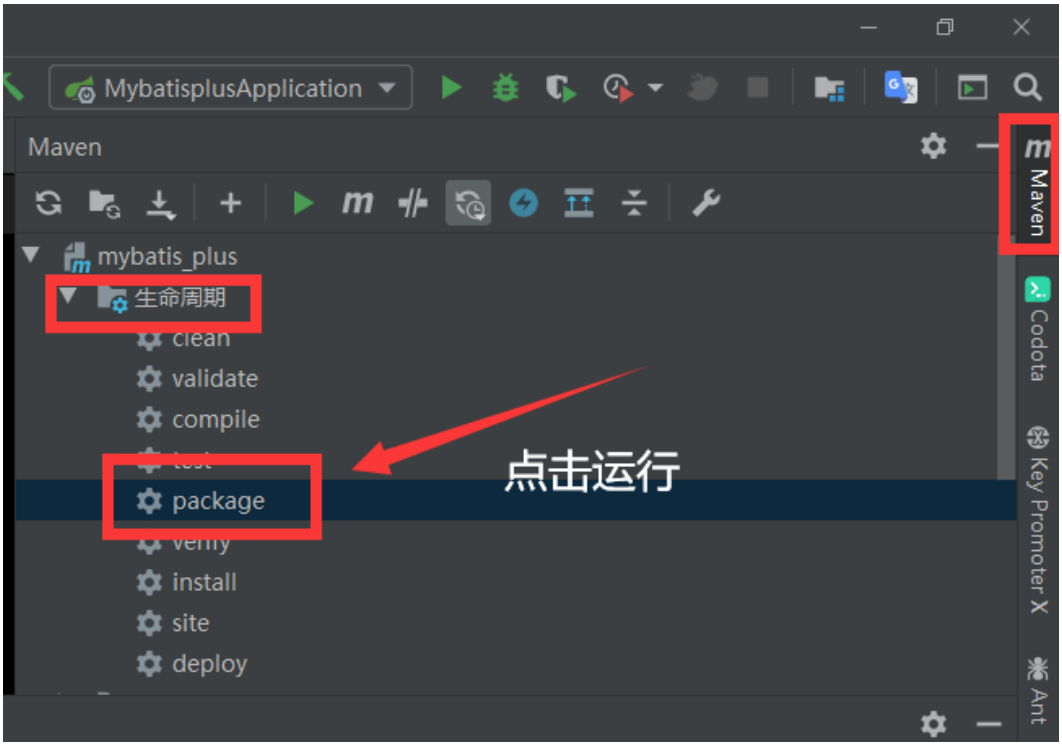
- application.yml 中进行thymeleaf配置

```
spring:
  datasource:
```

```
    driver-class-name:
com.mysql.cj.jdbc.Driver
    username: root
    password: ddd
    url: jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8
  thymeleaf:
    suffix: .html
    prefix: classpath:/templates/
mybatis-plus:
  configuration:
    log-impl:
org.apache.ibatis.logging.stdout.StdOutImpl
  type-enums-package:
    com.gloryh.mybatisplus.enums
  global-config:
    db-config:
      logic-not-delete-value: 0
      logic-delete-value: 1
```

> 打包部署

- 选择 Maven =》生命周期 =》package 命令，运行后出现 BUILD SUCCESS 提示代表打包成功。



```
020-10-12 08:40:58.571  INFO 13852 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : S
020-10-12 08:40:58.571  INFO 13852 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : H
020-10-12 08:40:59.787  INFO 13852 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : H
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ mybatisplus ---
[INFO] Building jar: E:\Program Files\IDEA Workspace\mybatisplus\target\mybatisplus-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.4.RELEASE:repackage (repackage) @ mybatisplus ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time:  20.369 s
[INFO] Finished at: 2020-10-12T08:41:01+08:00
[INFO] ------------------------------------------------------------------------
```

提示成功

- 上传至服务器，后运行 控制台命令

```
[root@gloryh ~]# ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[root@gloryh ~]# cd Downloads
[root@gloryh Downloads]# ls
mybatisplus-0.0.1-SNAPSHOT.jar
[root@gloryh Downloads]# java -jar mybatisplus-0.0.1-SNAPSHOT.jar
```

文件名

命令

- 提示，成功

```
ApplicationContext : Root WebApplicationContext: initialization completed in 367
2 ms
Logging initialized using 'class org.apache.ibatis.logging.stdout.StdOutImpl' ad
apter.
Registered plugin: 'com.baomidou.mybatisplus.extension.plugins.OptimisticLockerI
nterceptor@62379589'
Registered plugin: 'AbstractSqlParserHandler(sqlParserList=null, sqlParserFilter
=null)'
Property 'mapperLocations' was not specified.
 _ _   |_  _ _|_. ___ _ |    _
| | |\/|_)(_| | |_\  |_)||_|_\
     /               |
                        3.4.0
2020-10-12 08:53:12.280  INFO 17225 --- [           main] o.s.s.concurrent.Threa
dPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2020-10-12 08:53:12.533  INFO 17225 --- [           main] o.s.b.a.w.s.WelcomePag
eHandlerMapping    : Adding welcome page template: index
2020-10-12 08:53:13.740  INFO 17225 --- [           main] o.s.b.w.embedded.tomca
t.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2020-10-12 08:53:13.776  INFO 17225 --- [           main] c.g.mybatisplus.Mybati
splusApplication   : Started MybatisplusApplication in 9.595 seconds (JVM runni
g for 11.521)
2020-10-12 08:53:48.940  INFO 17225 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[lo
calhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
```

- 测试

用户id：	用户姓名：	用户年龄：
1	夏天	12
2	冬天	22