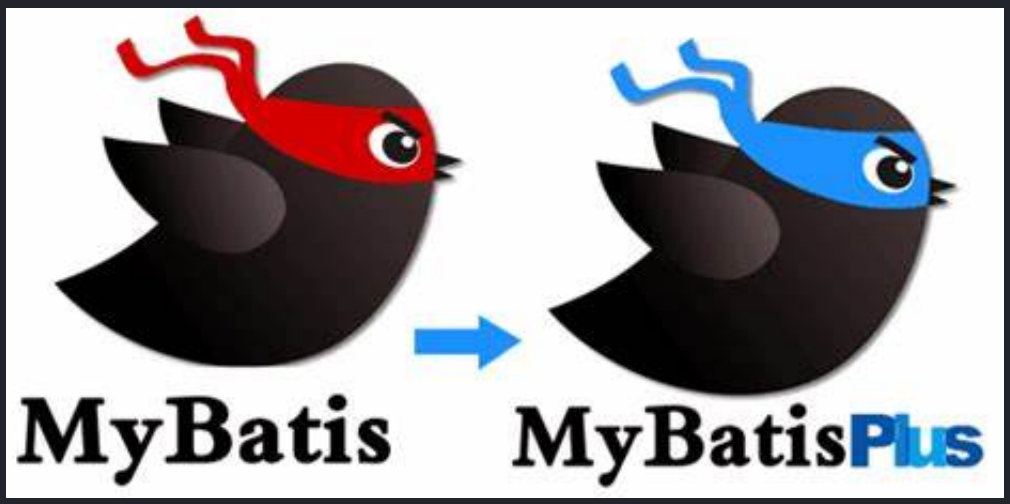
第一篇 整合SpringBoot快速开始增删改查

官网文档地址：https://mybatis.plus/guide/



一、Spring Boot整合Mybatis Plus

通过maven坐标引入依赖

```
<!-- mybatis plus -->
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>3.4.2</version>
</dependency>
<!-- mysql -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<!-- lombok -->
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

application配置数据源及日志输出级别

```
# 配置数据源
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/db_soft1921?
useUnicode=true&characterEncoding=utf8&serverTimezone=Asia
/Shanghai
    username: root
    password: root

# 配置Mybatis
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
  mapper-locations: classpath*:/mapper/*Mapper.xml
  type-aliases-package: top.mqxu.boot.mbp.domain
```

第三步：配置Mybatis的Mapper类文件的包扫描路径

```
@SpringBootApplication
@MapperScan(basePackages = {"top.mqxu.boot.mbp.mapper"})
public class SpringBootMybatisplusApplication {
    public static void main(String[] args) {

 SpringApplication.run(SpringBootMybatisplusApplication.cl
ass, args);
    }
}
```

二、编码构建实体和Mapper

编写实体类`User.java`

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class User {
    private Long id;
    private String name;
    private Integer age;
    private String email;
}
```

编写Mapper类`UserMapper.java`

```
public interface UserMapper extends BaseMapper<User> {

}
```

三、CRUD基础使用案例

3.1.增加一条记录

```
 User user =
User.builder().name("Rose").age(19).email("rose@baomidou.c
om").build();

 int row = userMapper.insert(user);
 assertEquals(1, row);

System.out.println("雪花算法id: " + user.getId());
```

写了上面的java代码，MP将会自动的根据java代码构造下面的SQL去数据库执行。注意：主键默认采用雪花算法

```
# 主键自动填充了雪花算法
INSERT INTO user ( id, name, age)
VALUES ( ?, ?, ? )
```

3.2.根据主键删除一条记录

```
int rows = userMapper.deleteById(1170243901535006722L);
System.out.println("影响记录数: " + rows);
```

1170243901535006722L是数据插入的时候根据雪花算法生成的id

```
DELETE FROM user
WHERE id=?
```

3.3.根据条件删除记录

```
//构造条件
Map<String, Object> map = new HashMap<>();
map.put("name", "Jack");
map.put("age", 20);
//执行删除
int rows = userMapper.deleteByMap(map);
assertEquals(1, rows);
System.out.println("影响记录数: " + rows);
```

3.4.根据主键查询一条数据

```
User user = userMapper.selectById(1089911557332887553L);
System.out.println(user);
```

3.5.根据ids批量查找数据

```
List<Long> ids = Arrays.asList(
    1087982257332887553L,
    1094590409767661570L,
    1094592041087729666L
);
List<User> list = userMapper.selectBatchIds(ids);
list.forEach(System.out::println);
```

对应的sql语句为

```
SELECT id,name,age,email
FROM user
WHERE id IN ( ? , ? , ? )
```

3.6.根据指定参数查询

```
Map<String, Object> map = new HashMap<>();
//map的key指代的是mysql表中的列名，并非java实体的属性名
map.put("name", "Jone");

List<User> list = userMapper.selectByMap(map);
list.forEach(System.out::println);
```

```
SELECT id,name,age,email
FROM user
WHERE name = ?
```

3.7.指定查询结果字段

1.

```
QueryWrapper<User> query = new QueryWrapper<>();
query.select("name", "age")    //指定查询结果字段
  .in("age", Arrays.asList(18, 19, 20))
  .last("limit 2");
List<User> list = userMapper.selectList(query);
list.forEach(System.out::println);
```

```
SELECT name,age
FROM user
WHERE age IN (?,?,?)
LIMIT 2
```

2.

```
QueryWrapper<User> query = new QueryWrapper<>();
query.like("name", "J%")    //like是MP的条件构造器，表示"模糊查
询"
  .lt("age", 30)            //lt是MP的条件构造器，表示"小于"关
系
  .select("name", "age");
List<Map<String, Object>> maps =
userMapper.selectMaps(query);
maps.forEach(System.out::println);
```

```
SELECT name,age
FROM user
WHERE name LIKE ?
AND age < ?
```

3.8.通过主键id修改数据

```
User user = new User();
user.setId(1088248199570832385L);
user.setAge(18);
user.setEmail("mybatis-plus@163.com");
int rows = userMapper.updateById(user);
System.out.println("影响记录数: " + rows);
```

```
UPDATE user
SET age=?, email=?
WHERE id=?
```

3.9.根据UpdateWrapper自定义条件修改数据

```java
UpdateWrapper<User> update = new UpdateWrapper<>();
update.eq("name", "Jack").eq("age", 28);    //eq是MP的条件构造器，表示"等于"关系

User user = new User();
user.setAge(29);
user.setEmail("hadoopcn2@163.com");
int rows = userMapper.update(user, update);
System.out.println("影响记录数：" + rows);
```

```sql
UPDATE user
SET age=?, email=?
WHERE name = ?
AND age = ?
```

四、附录---测试SQL：

```sql
DROP TABLE IF EXISTS user;

CREATE TABLE user
(
    id BIGINT(20) NOT NULL COMMENT '主键ID',
    name VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',
    age INT(11) NULL DEFAULT NULL COMMENT '年龄',
    email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',
    PRIMARY KEY (id)
);
```

其对应的数据库 Data 脚本如下：

```sql
INSERT INTO user (id, name, age, email) VALUES
(1, 'Jone', 18, 'test1@baomidou.com'),
(2, 'Jack', 20, 'test2@baomidou.com'),
(3, 'Tom', 28, 'test3@baomidou.com'),
(4, 'Sandy', 21, 'test4@baomidou.com'),
(5, 'Billie', 24, 'test5@baomidou.com');
```