# **Mybatis从入门到精通**

# 1 mybatis入门

## 1.1 Mybatis简介

## 1.2创建Maven项目

项目结构如下，结构比较简单

book1-mybatis-demo01

├── pom.xml

└── src

└── main

├── java

│   └── com

│   └── mybatis

│   └── book

│   └── study

│   ├── mapper

│   │   └── CountryMapper.java

│   └── model

│   └── Country.java

└── resources

├── log4j.properties

├── mappers

│   └── CountryMapper.xml

└── mybatis-config.xml

pom文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.mybatis.book.study</groupId>  <artifactId>book1-mybatis-demo01</artifactId>  <version>1.0-SNAPSHOT</version>    <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>    <dependencies>   <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>   <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.7.12</version>  </dependency>   <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.7.12</version>  </dependency>   <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>   <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.3.0</version>  </dependency>  </dependencies>   <build>  <plugins>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1,8</target>  </configuration>  </plugin>  </plugins>  </build>   </project> |

## 1.3简单配置让mybatis跑起来

### 1.3.1准备数据库

创建数据库mybatis-study

创建表country和处处海湖数据

|  |
| --- |
| create table country (  id int not null auto\_increment,  countryname varchar(255) null,  countrycode varchar(255) null,  PRIMARY key (id)  );  INSERT into `country` (`countryname`,`countrycode`) values ('中国','CN'),('美国','US'),('俄罗斯','RU'),('英国','GB'),('法国','FR'); |

### 1.3.2配置mybatis

mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"> <configuration>  <settings>  <!-- 输出的日志-->  <setting name="logImpl" value="LOG4J"/>  </settings>   <typeAliases>  <!-- 实体对应的包-->  <package name="com.mybatis.book.study.model"/>  </typeAliases>   <!-- 数据源配置-->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC">  <property name="" value=""/>  </transactionManager>  <dataSource type="UNPOOLED">  <property name="driver" value="com.mysql.jdbc.Driver"/>  <property name="url" value="jdbc:mysql://localhost:3306/mybatis-study"/>  <property name="username" value="root"/>  <property name="password" value="mysql"/>  </dataSource>  </environment>  </environments>    <mappers>  <!-- 映射文件所在位置-->  <mapper resource="mappers/CountryMapper.xml"></mapper>  </mappers>  </configuration>  <typeAliases>配置包的别名，主要是在mapper.xml文件的映射实体可以省略包名  <mappers>中配置了一个包含完整类路径的CountryMapper.xml |

### 1.3.3 创建实体和映射文件

在model包下创建实体Country

|  |
| --- |
| package com.mybatis.book.study.model;  public class Country {  private Long id;   private String countryname;   private String countrycode;   public Long getId() {  return id;  }   public void setId(Long id) {  this.id = id;  }   public String getCountryname() {  return countryname;  }   public void setCountryname(String countryname) {  this.countryname = countryname;  }   public String getCountrycode() {  return countrycode;  }   public void setCountrycode(String countrycode) {  this.countrycode = countrycode;  } } |

在src/main/resources/下创建mappers文件夹，在该文件夹下创建CountryMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"> <mapper namespace="com.mybatis.book.study.mapper.CountryMapper">   <select id="selectAll" resultType="Country">  select id,countryname,countrycode from country  </select>  </mapper> |

### 1.3.4 配置Log4j

在src/main/resources/下创建log4j.properties

|  |
| --- |
| # 全局配置 log4j.rootLogger=ERROR, stdout # mybatis日志配置 log4j.logger.com.mybatis.book.study.mapper=TRACE #控制台输出配置 log4j.appender.stdout=org.apache.log4j.ConsoleAppender log4j.appender.stdout.layout=org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n |

### 1.3.5 编写测试用例

在src/test/java 中创建com.mybatis.book.study.CountryMapperTest

|  |
| --- |
| package com.mybatis.book.study;  import com.mybatis.book.study.model.Country; import org.apache.ibatis.io.Resources; import org.apache.ibatis.session.SqlSession; import org.apache.ibatis.session.SqlSessionFactory; import org.apache.ibatis.session.SqlSessionFactoryBuilder; import org.junit.BeforeClass; import org.junit.Test;  import java.io.IOException; import java.io.Reader; import java.util.List;  public class CountryMapperTest {   private static SqlSessionFactory *sqlSessionFactory*;   @BeforeClass  public static void init() {  try {  Reader reader = Resources.*getResourceAsReader*("mybatis-config.xml");  *sqlSessionFactory* = new SqlSessionFactoryBuilder().build(reader);  reader.close();  } catch (IOException e) {  e.printStackTrace();  }  }   @Test  public void testSelectAll() {  SqlSession sqlSession = *sqlSessionFactory*.openSession();  List<Country> countryList = sqlSession.selectList("selectAll");  printCountryList(countryList);  }   private void printCountryList(List<Country> countryList) {  for (Country country : countryList) {  System.*out*.printf("%-4d%4s%4s\n", country.getId(), country.getCountryname(), country.getCountrycode());  }  }  }  运行输出如下  DEBUG [main] - ==> Preparing: select id,countryname,countrycode from country  DEBUG [main] - ==> Parameters:  TRACE [main] - <== Columns: id, countryname, countrycode  TRACE [main] - <== Row: 1, 中国, CN  TRACE [main] - <== Row: 2, 美国, US  TRACE [main] - <== Row: 3, 俄罗斯, RU  TRACE [main] - <== Row: 4, 英国, GB  TRACE [main] - <== Row: 5, 法国, FR  DEBUG [main] - <== Total: 5  1 中国 CN  2 美国 US  3 俄罗斯 RU  4 英国 GB  5 法国 FR |

# **2 Mybatis XML方式的基本配置**

## 2.1 一个简单的权限控制需求

### 2.1.1 创建数据库表

|  |
| --- |
| create table sys\_user(  id bigint not null auto\_increment comment '用户Id',  user\_name varchar(50) comment '用户名',  user\_password varchar(50) comment '密码',  user\_email varchar(50) comment '邮箱',  user\_info text comment '简介',  head\_img blob comment '头像',  create\_time datetime comment '创建时间',  primary key (id)  );  alter table sys\_user comment '用户表';  create table sys\_role(  id bigint not null auto\_increment comment '角色Id',  role\_name varchar(50) comment '角色名',  enabled int comment '有效标志',  create\_by bigint comment '创建人',  create\_time datetime comment '创建时间',  primary key (id)  );  alter table sys\_role comment '角色表';  create table sys\_privilege(  id bigint not null auto\_increment comment '权限id',  privilege\_name varchar(50) comment '权限名称',  privilege\_url varchar(200) comment '权限url',  primary key (id)  );  alter table sys\_privilege comment '权限表';  create table sys\_user\_role (  user\_id bigint comment '用户Id',  role\_id bigint comment '角色Id'  );  alter table sys\_user\_role comment '用户角色关联表';  create table sys\_role\_privilege(  role\_id bigint comment '角色ID',  privilege\_id bigint comment '权限Id'  );  alter table sys\_role\_privilege comment '角色权限关联表';  INSERT INTO `sys\_user` values ('1','admin','123456','admin@mybatis.tk','管理员',null,'2019-10-25 16:00:00');  INSERT INTO `sys\_user` values ('1001','test','123456','test@mybatis.tk','测试用户',null,'2019-10-25 16:00:00');  insert into `sys\_role` values ('1','管理员','1','1','2019-10-25 16:00:00');  insert into `sys\_role` values ('2','普通用户','1','1','2019-10-25 16:00:00');  insert into `sys\_user\_role` values ('1','1');  insert into `sys\_user\_role` values ('1','2');  insert into `sys\_user\_role` values ('1001','2');  insert into `sys\_privilege` values ('1','用户管理','/users');  insert into `sys\_privilege` values ('2','角色管理','/roles');  insert into `sys\_privilege` values ('3','系统日志','/logs');  insert into `sys\_privilege` values ('4','人员维护','/persons');  insert into `sys\_privilege` values ('5','单位维护','/companies');  insert into `sys\_role\_privilege` values ('1','1');  insert into `sys\_role\_privilege` values ('1','2');  insert into `sys\_role\_privilege` values ('1','3');  insert into `sys\_role\_privilege` values ('2','4');  insert into `sys\_role\_privilege` values ('3','5'); |

### 2.1.2 创建实体类

在model包下创建实体类

SysUser

|  |
| --- |
| package com.mybatis.book.study.model;  import java.util.Date;  public class SysUser {   private Long id;  private String userName;  private String userPassword;  private String userEmail;  private String userInfo;  private byte[] headImg;  private Date createTime;   public Long getId() {  return id;  }   public void setId(Long id) {  this.id = id;  }   public String getUserName() {  return userName;  }   public void setUserName(String userName) {  this.userName = userName;  }   public String getUserPassword() {  return userPassword;  }   public void setUserPassword(String userPassword) {  this.userPassword = userPassword;  }   public String getUserEmail() {  return userEmail;  }   public void setUserEmail(String userEmail) {  this.userEmail = userEmail;  }   public String getUserInfo() {  return userInfo;  }   public void setUserInfo(String userInfo) {  this.userInfo = userInfo;  }   public byte[] getHeadImg() {  return headImg;  }   public void setHeadImg(byte[] headImg) {  this.headImg = headImg;  }   public Date getCreateTime() {  return createTime;  }   public void setCreateTime(Date createTime) {  this.createTime = createTime;  } } |

SysRole

|  |
| --- |
| package com.mybatis.book.study.model;  import java.util.Date;  public class SysRole {   private Long id;  private String roleName;  private Integer enabled;  private Long createBy;  private Date createTime;   public Long getId() {  return id;  }   public void setId(Long id) {  this.id = id;  }   public String getRoleName() {  return roleName;  }   public void setRoleName(String roleName) {  this.roleName = roleName;  }   public Integer getEnabled() {  return enabled;  }   public void setEnabled(Integer enabled) {  this.enabled = enabled;  }   public Long getCreateBy() {  return createBy;  }   public void setCreateBy(Long createBy) {  this.createBy = createBy;  }   public Date getCreateTime() {  return createTime;  }   public void setCreateTime(Date createTime) {  this.createTime = createTime;  } } |

SysRolePrivilege

|  |
| --- |
| package com.mybatis.book.study.model;  public class SysRolePrivilege {   private Long roleId;  private Long privilegeId;   public Long getRoleId() {  return roleId;  }   public void setRoleId(Long roleId) {  this.roleId = roleId;  }   public Long getPrivilegeId() {  return privilegeId;  }   public void setPrivilegeId(Long privilegeId) {  this.privilegeId = privilegeId;  } } |

SysUserRole

|  |
| --- |
| package com.mybatis.book.study.model;  public class SysUserRole {   private Long userId;  private Long roleId;   public Long getUserId() {  return userId;  }   public void setUserId(Long userId) {  this.userId = userId;  }   public Long getRoleId() {  return roleId;  }   public void setRoleId(Long roleId) {  this.roleId = roleId;  } } |

SysPrivilege

|  |
| --- |
| package com.mybatis.book.study.model;  public class SysPrivilege {   private Long id;  private String privilegeName;  private String privilegeUrl;   public Long getId() {  return id;  }   public void setId(Long id) {  this.id = id;  }   public String getPrivilegeName() {  return privilegeName;  }   public void setPrivilegeName(String privilegeName) {  this.privilegeName = privilegeName;  }   public String getPrivilegeUrl() {  return privilegeUrl;  }   public void setPrivilegeUrl(String privilegeUrl) {  this.privilegeUrl = privilegeUrl;  }   } |

## 2.2 使用XML方式

MyBatis使用Java的动态代理可以直接通过接口调用相应的方法，不需要提供接口的实现类，更不需要在实现类中使用SqlSession以通过命名空间间接滴啊用。

同时在mapper下创建对应的mapper接口：UserMapper.java,RoleMapper.java,PrivilegeMapper.java,UserRoleMapper.java,RolePrivilege.java

在src/main/resources/mappers/创建xml映射文件：UserMapper.xml,RoleMapper.xml,PrivilegeMapper.xml,UserRoleMapper.xml,RolePrivilege.xml

在mybatis-config.xml文件中 将映射文件加入：

|  |
| --- |
| <mappers>  <!-- 映射文件所在位置-->  <mapper resource="mappers/CountryMapper.xml"></mapper>  <mapper resource="mappers/UserMapper.xml"></mapper>  <mapper resource="mappers/RoleMapper.xml"></mapper>  <mapper resource="mappers/PrivilegeMapper.xml"></mapper>  <mapper resource="mappers/UserRoleMapper.xml"></mapper>  <mapper resource="mappers/RolePrivilegeMapper.xml"></mapper>  </mappers> |

## 2.3 select用法

项目的结构

book1-mybatis-demo01

├── pom.xml

└── src

├── main

│   ├── java

│   │   └── com

│   │   └── mybatis

│   │   └── book

│   │   └── study

│   │   ├── mapper

│   │   │   ├── CountryMapper.java

│   │   │   ├── PrivilegeMapper.java

│   │   │   ├── RoleMapper.java

│   │   │   ├── RolePrivilegeMapper.java

│   │   │   ├── UserMapper.java

│   │   │   └── UserRoleMapper.java

│   │   └── model

│   │   ├── Country.java

│   │   ├── SysPrivilege.java

│   │   ├── SysRole.java

│   │   ├── SysRoleExtend.java

│   │   ├── SysRolePrivilege.java

│   │   ├── SysUser.java

│   │   └── SysUserRole.java

│   └── resources

│   ├── log4j.properties

│   ├── mappers

│   │   ├── CountryMapper.xml

│   │   ├── PrivilegeMapper.xml

│   │   ├── RoleMapper.xml

│   │   ├── RolePrivilegeMapper.xml

│   │   ├── UserMapper.xml

│   │   └── UserRoleMapper.xml

│   └── mybatis-config.xml

└── test

└── java

└── com

└── mybatis

└── book

└── study

├── BaseMapperTest.java

├── CountryMapperTest.java

└── UserMapperTest.java

UserMapper的代码

|  |
| --- |
| package com.mybatis.book.study.mapper;  import com.mybatis.book.study.model.SysRole; import com.mybatis.book.study.model.SysRoleExtend; import com.mybatis.book.study.model.SysUser;  import java.util.List;  public interface UserMapper {   */\*\*  \* 通过id查询用户  \** ***@param*** *id  \** ***@return*** *\*/* SysUser selectById(Long id);   */\*\*  \* 查询全部用户  \** ***@return*** *\*/* List<SysUser> selectAll();   */\*\*  \* 根据用户id获取角色信息  \** ***@param*** *userId  \** ***@return*** *\*/* List<SysRole> selectRolesByUserId(Long userId);    */\*\*  \* 根据用户id获取角色信息  \* 返回带userName列  \** ***@param*** *userId  \** ***@return*** *\*/* List<SysRoleExtend> selectRolesByUserIdExtend(Long userId);    */\*\*  \* 根据用户id获取角色信息  \* 返回带SysUser多个列  \** ***@param*** *userId  \** ***@return*** *\*/* List<SysRole> selectRolesByUserIdWithSysUser(Long userId);  } |

UserMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"> <mapper namespace="com.mybatis.book.study.mapper.UserMapper">     <resultMap id="userMap" type="com.mybatis.book.study.model.SysUser">  <id property="id" column="id"/>  <result property="userName" column="user\_name"/>  <result property="userPassword" column="user\_password"/>  <result property="userEmail" column="user\_email"/>  <result property="userInfo" column="user\_info"/>  <result property="headImg" column="head\_img" jdbcType="BLOB"/>  <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"/>  </resultMap>    <select id="selectById" resultMap="userMap">  select \* from sys\_user where id = #{id}  </select>    <!-- selectById和selectAll的区别：selectById中使用了resultMap来设置结果映射，  而selectAll中则通过resultType直接指定了返回结果的类型。可以发现，如果使用resultType  来设置返回结果的类型，需要在SQL中为所有列名和属性名不一致的类设置别名，通过哟设置别名使  最终的查询结果列和resultType指定对象的属性名保持一致，进而实现自动映射-->  <select id="selectAll" resultType="com.mybatis.book.study.model.SysUser">  select id,  user\_name userName,  user\_password userPassword,  user\_email userEmail,  user\_info userInfo,  head\_img headImg,  create\_time createTime  from sys\_user  </select>   <select id="selectRolesByUserId" resultType="com.mybatis.book.study.model.SysRole">  select  r.id,  r.role\_name roleName,  r.enabled,  r.create\_by createBy,  r.create\_time createTime  from sys\_user u  inner join sys\_user\_role ur on u.id = ur.user\_id  inner join sys\_role r on ur.role\_id = r.id  where u.id = #{userId}  </select>   <select id="selectRolesByUserIdExtend" resultType="com.mybatis.book.study.model.SysRoleExtend">  select  r.id,  r.role\_name roleName,  r.enabled,  r.create\_by createBy,  r.create\_time createTime,  u.user\_name userName  from sys\_user u  inner join sys\_user\_role ur on u.id = ur.user\_id  inner join sys\_role r on ur.role\_id = r.id  where u.id = #{userId}  </select>   <select id="selectRolesByUserIdWithSysUser" resultType="com.mybatis.book.study.model.SysRole">  select  r.id,  r.role\_name roleName,  r.enabled,  r.create\_by createBy,  r.create\_time createTime,  u.user\_name as "user.userName",  u.user\_email as "user.userEmail"  from sys\_user u  inner join sys\_user\_role ur on u.id = ur.user\_id  inner join sys\_role r on ur.role\_id = r.id  where u.id = #{userId}  </select>   </mapper>  <resultMap>属性:  id:必填，并且唯一  type:必填,用于配置查询列所映射到的Java对象类型。  extends:选填，可以配置当前的resultMap继承自其他的resultMap,属性值为继承resultMap的id。  autoMapping：选填，可选值为true或false，用于配置是否启用非映射字段（没有在resultMap中配置的字段）的自动映射功能，该配置可以覆盖全局的autoMappingBehavior配置。  constructor：配置使用构造方法注入结果，包含以下两个子标签。  idArg：id参数，标记结果作为id（唯一值），可以帮助提高整体性能  arg：注入到构造方法的一个普通结果  id：一个id结果，标记结果作为id，可以帮助提高整体性能  result;注入到Java对象属性的普通结果  association：一个复杂的类型关联，许多结果将包成这种类型  collection：复杂类型集合  discriminator：根据结果值来决定使用哪个结果映射  Case：基于某些值的结果映射 |

Mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"> <configuration>  <settings>  <!-- 输出的日志-->  <setting name="logImpl" value="LOG4J"/>  <!-- true 可以自动将以下划线方式命名的数据库列映射到Java对象的驼峰式命名属性中-->  <setting name="mapUnderscoreToCamelCase" value="true"/>  </settings>   <typeAliases>  <!-- 实体对应的包-->  <package name="com.mybatis.book.study.model"/>  </typeAliases>   <!-- 数据源配置-->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC">  <property name="" value=""/>  </transactionManager>  <dataSource type="UNPOOLED">  <property name="driver" value="com.mysql.jdbc.Driver"/>  <property name="url" value="jdbc:mysql://localhost:3306/mybatis-study"/>  <property name="username" value="root"/>  <property name="password" value="mysql"/>  </dataSource>  </environment>  </environments>    <mappers>  <!-- 映射文件所在位置-->  <mapper resource="mappers/CountryMapper.xml"></mapper>  <mapper resource="mappers/UserMapper.xml"></mapper>  <mapper resource="mappers/RoleMapper.xml"></mapper>  <mapper resource="mappers/PrivilegeMapper.xml"></mapper>  <mapper resource="mappers/UserRoleMapper.xml"></mapper>  <mapper resource="mappers/RolePrivilegeMapper.xml"></mapper>  </mappers>  </configuration> |

测试类调整

BaseMapperTest

|  |
| --- |
| package com.mybatis.book.study;  import com.mybatis.book.study.model.Country; import org.apache.ibatis.io.Resources; import org.apache.ibatis.session.SqlSession; import org.apache.ibatis.session.SqlSessionFactory; import org.apache.ibatis.session.SqlSessionFactoryBuilder; import org.junit.BeforeClass; import org.junit.Test;  import java.io.IOException; import java.io.Reader; import java.util.List;  public class BaseMapperTest {   private static SqlSessionFactory *sqlSessionFactory*;   @BeforeClass  public static void init() {  try {  Reader reader = Resources.*getResourceAsReader*("mybatis-config.xml");  *sqlSessionFactory* = new SqlSessionFactoryBuilder().build(reader);  reader.close();  } catch (IOException e) {  e.printStackTrace();  }  }   public static SqlSessionFactory getSqlSessionFactory() {  return *sqlSessionFactory*;  }   public static SqlSession getSqlSession() {  return *sqlSessionFactory*.openSession();  } } |

为了测试resultType返回的结果中有两张表中的列，使用两种方式，一种方式继承，一种组合

SysRoleExtend(继承)

|  |
| --- |
| package com.mybatis.book.study.model;  public class SysRoleExtend extends SysRole {   private String userName;   public String getUserName() {  return userName;  }   public void setUserName(String userName) {  this.userName = userName;  } } |

SysRole（组合，将SysUser作为SysRole的属性）

|  |
| --- |
| package com.mybatis.book.study.model;  import java.util.Date;  public class SysRole {   private Long id;  private String roleName;  private Integer enabled;  private Long createBy;  private Date createTime;   private SysUser user;   public Long getId() {  return id;  }   public void setId(Long id) {  this.id = id;  }   public String getRoleName() {  return roleName;  }   public void setRoleName(String roleName) {  this.roleName = roleName;  }   public Integer getEnabled() {  return enabled;  }   public void setEnabled(Integer enabled) {  this.enabled = enabled;  }   public Long getCreateBy() {  return createBy;  }   public void setCreateBy(Long createBy) {  this.createBy = createBy;  }   public Date getCreateTime() {  return createTime;  }   public void setCreateTime(Date createTime) {  this.createTime = createTime;  }   public SysUser getUser() {  return user;  }   public void setUser(SysUser user) {  this.user = user;  } } |

UserMapperTest

|  |
| --- |
| package com.mybatis.book.study;  import com.mybatis.book.study.mapper.UserMapper; import com.mybatis.book.study.model.SysRole; import com.mybatis.book.study.model.SysRoleExtend; import com.mybatis.book.study.model.SysUser; import org.apache.ibatis.session.SqlSession; import org.junit.Assert; import org.junit.Test;  import java.util.List;  import static org.junit.Assert.\*;  public class UserMapperTest extends BaseMapperTest {    @Test  public void testSelectById() {  SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  SysUser sysUser = userMapper.selectById(1L);  *assertNotNull*(sysUser);  *assertEquals*("admin", sysUser.getUserName());   } finally {  sqlSession.close();  }  }    @Test  public void testSelectAll() {  SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  List<SysUser> userList = userMapper.selectAll();  *assertNotNull*(userList);  *assertTrue*(userList.size() > 0);  } finally {  sqlSession.close();  }  }    @Test  public void testSelectRolesByUserId() {  SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  List<SysRole> roles = userMapper.selectRolesByUserId(1L);  *assertNotNull*(roles);  *assertTrue*(roles.size() > 0);  } finally {  sqlSession.close();  }  }    @Test  public void testSelectRolesByUserIdExtend() {  SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  List<SysRoleExtend> roles = userMapper.selectRolesByUserIdExtend(1L);  *assertNotNull*(roles);  *assertTrue*(roles.size() > 0);  } finally {  sqlSession.close();  }  }    @Test  public void testSelectRolesByUserIdWithSysUser() {  SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  List<SysRole> roles = userMapper.selectRolesByUserIdWithSysUser(1L);  *assertNotNull*(roles);  *assertTrue*(roles.size() > 0);  } finally {  sqlSession.close();  }  }    }  testSelectById 输出如下：  DEBUG [main] - ==> Preparing: select \* from sys\_user where id = ?  DEBUG [main] - ==> Parameters: 1(Long)  TRACE [main] - <== Columns: id, user\_name, user\_password, user\_email, user\_info, head\_img, create\_time  TRACE [main] - <== Row: 1, admin, 123456, admin@mybatis.tk, <<BLOB>>, <<BLOB>>, 2019-10-25 16:00:00.0  DEBUG [main] - <== Total: 1  testSelectAll 输出如下:  DEBUG [main] - ==> Preparing: select id, user\_name userName, user\_password userPassword, user\_email userEmail, user\_info userInfo, head\_img headImg, create\_time createTime from sys\_user  DEBUG [main] - ==> Parameters:  TRACE [main] - <== Columns: id, userName, userPassword, userEmail, userInfo, headImg, createTime  TRACE [main] - <== Row: 1, admin, 123456, admin@mybatis.tk, <<BLOB>>, <<BLOB>>, 2019-10-25 16:00:00.0  TRACE [main] - <== Row: 1001, test, 123456, test@mybatis.tk, <<BLOB>>, <<BLOB>>, 2019-10-25 16:00:00.0  DEBUG [main] - <== Total: 2  testSelectRolesByUserId 输出如下:  DEBUG [main] - ==> Preparing: select r.id, r.role\_name roleName, r.enabled, r.create\_by createBy, r.create\_time createTime from sys\_user u inner join sys\_user\_role ur on u.id = ur.user\_id inner join sys\_role r on ur.role\_id = r.id where u.id = ?  DEBUG [main] - ==> Parameters: 1(Long)  TRACE [main] - <== Columns: id, roleName, enabled, createBy, createTime  TRACE [main] - <== Row: 1, 管理员, 1, 1, 2019-10-25 16:00:00.0  TRACE [main] - <== Row: 2, 普通用户, 1, 1, 2019-10-25 16:00:00.0  DEBUG [main] - <== Total: 2  testSelectRolesByUserIdExtend输出如下:  DEBUG [main] - ==> Preparing: select r.id, r.role\_name roleName, r.enabled, r.create\_by createBy, r.create\_time createTime, u.user\_name userName from sys\_user u inner join sys\_user\_role ur on u.id = ur.user\_id inner join sys\_role r on ur.role\_id = r.id where u.id = ?  DEBUG [main] - ==> Parameters: 1(Long)  TRACE [main] - <== Columns: id, roleName, enabled, createBy, createTime, userName  TRACE [main] - <== Row: 1, 管理员, 1, 1, 2019-10-25 16:00:00.0, admin  TRACE [main] - <== Row: 2, 普通用户, 1, 1, 2019-10-25 16:00:00.0, admin  DEBUG [main] - <== Total: 2  testSelectRolesByUserIdWithSysUser输出如下:  DEBUG [main] - ==> Preparing: select r.id, r.role\_name roleName, r.enabled, r.create\_by createBy, r.create\_time createTime, u.user\_name as "user.userName", u.user\_email as "user.userEmail" from sys\_user u inner join sys\_user\_role ur on u.id = ur.user\_id inner join sys\_role r on ur.role\_id = r.id where u.id = ?  DEBUG [main] - ==> Parameters: 1(Long)  TRACE [main] - <== Columns: id, roleName, enabled, createBy, createTime, user.userName, user.userEmail  TRACE [main] - <== Row: 1, 管理员, 1, 1, 2019-10-25 16:00:00.0, admin, admin@mybatis.tk  TRACE [main] - <== Row: 2, 普通用户, 1, 1, 2019-10-25 16:00:00.0, admin, admin@mybatis.tk  DEBUG [main] - <== Total: 2 |

## 2.4 insert用法

### 2.4.1简单的insert方法

在UserMapper.java增加

*/\*\*  
 \* 新增用户  
 \** ***@param*** *sysUser  
 \** ***@return*** *\*/*int insert(SysUser sysUser);

在UserMapper.xml 增加

|  |
| --- |
| <!--  insert标签属性：  id:命名空间中的唯一标识，  parameterType:即将传入的语句的参数的完全限定类名或别名。这个属性时可选的  flushCache:默认值是true，任何时候只要语句被调用，都会清空一级缓存和二级缓存  timeout：设置在抛出异常之前，驱动程序等待数据库返回请求结果的秒数  statementType：对于STATEMENT，PREPARED、CALLABLE，Mybatis会分表使用对应的Statement，PreparedStatement、CallableStatement，默认是PREPARED  useGeneratedKeys：默认值为false，如果设置为true，Mybatis会使用JDBC的getGeneratedKeys方法来取出由数据库内部生成的主键，  keyProperty：Mybatis通过getGeneratedKeys获取主键值后将要复制属性名。如果希望得到多个数据库自动生成的列，属性值也可以是以逗号分隔的属性名称列表  keyColumn：仅对INSERT和UPDATE有用。通过生成的键值设置表中的列名，这个设置仅在某些数据库（PostgreSQL）中是必须的，当主键列不是表中的第一列时需要设置  databaseId：如果配置了databaseIdProvider，Mybatis会加载所有的不带dataBaseId的或匹配当前databaseId的语句，如果同时存在带databaseId和不带databaseId的语句。   说明：BLOB对应的类型是ByteArrayInputStream，就是二级制数据流；date，time，datetime对应的JDBC类型分别为DATE(年月日)、TIME(时分秒)、TIMESTAMP(年月日时分秒)  --> <insert id="insert">  insert into sys\_user (id,user\_name,user\_password,user\_email,user\_info,head\_img,create\_time)  values (  #{id},#{userName} ,#{userPassword} ,#{userEmail} ,#{userInfo} ,#{headImg,jdbcType=BLOB},  #{createTime,jdbcType=TIME}  ); </insert>  insert标签属性：  id:命名空间中的唯一标识，  parameterType:即将传入的语句的参数的完全限定类名或别名。这个属性时可选的  flushCache:默认值是true，任何时候只要语句被调用，都会清空一级缓存和二级缓存  timeout：设置在抛出异常之前，驱动程序等待数据库返回请求结果的秒数  statementType：对于STATEMENT，PREPARED、CALLABLE，Mybatis会分表使用对应的Statement，PreparedStatement、CallableStatement，默认是PREPARED  useGeneratedKeys：默认值为false，如果设置为true，Mybatis会使用JDBC的getGeneratedKeys方法来取出由数据库内部生成的主键，  keyProperty：Mybatis通过getGeneratedKeys获取主键值后将要复制属性名。如果希望得到多个数据库自动生成的列，属性值也可以是以逗号分隔的属性名称列表  keyColumn：仅对INSERT和UPDATE有用。通过生成的键值设置表中的列名，这个设置仅在某些数据库（PostgreSQL）中是必须的，当主键列不是表中的第一列时需要设置  databaseId：如果配置了databaseIdProvider，Mybatis会加载所有的不带dataBaseId的或匹配当前databaseId的语句，如果同时存在带databaseId和不带databaseId的语句。  **说明：BLOB对应的类型是ByteArrayInputStream，就是二级制数据流；date，time，datetime对应的JDBC类型分别为DATE(年月日)、TIME(时分秒)、TIMESTAMP(年月日时分秒)** |

UserMapperTest 增加

|  |
| --- |
| UserMapperTest 增加  @Test public void testInsert() {   SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  SysUser user = new SysUser();  user.setUserName("test1");  user.setUserPassword("123456");  user.setUserEmail("test@mybatis.tk");  user.setUserInfo("test info");  user.setHeadImg(new byte[]{1, 2, 3});  user.setCreateTime(new Date());   int result = userMapper.insert(user);  *assertEquals*(result, 1);  // 这种情况下，未返回id  *assertNull*(user.getId());   } finally {  sqlSession.close();  } } |

### 2.4.2使用JDBC方式返回主键自增的值

在UserMapper.java增加

|  |
| --- |
| */\*\*  \* 新增用户-使用useGenerateKeys方式  \** ***@param*** *sysUser  \** ***@return*** *\*/* int insert2(SysUser sysUser); |

在UserMapper.xml 增加

|  |
| --- |
| <!--  useGeneratedKeys设置为tre后，mybatis会使用JDBC的getGeneratedKeys方法来取出数据库内部生成的主键  获得主键后将其复制给keyProperty配置的id属性。当需要设置多个属性时，使用逗号隔开。  通常还需要设置keyColum属性，按顺序指定数据库的列，这里列的值会和keyProperty配置的属性一一对应 --> <insert id="insert2" useGeneratedKeys="true" keyProperty="id">  insert into sys\_user (user\_name,user\_password,user\_email,user\_info,head\_img,create\_time)  values (  #{userName} ,#{userPassword} ,#{userEmail} ,#{userInfo} ,#{headImg,jdbcType=BLOB},  #{createTime,jdbcType=TIMESTAMP}  ); </insert>  useGeneratedKeys设置为tre后，mybatis会使用JDBC的getGeneratedKeys方法来取出数据库内部生成的主键  获得主键后将其复制给keyProperty配置的id属性。当需要设置多个属性时，使用逗号隔开。  通常还需要设置keyColum属性，按顺序指定数据库的列，这里列的值会和keyProperty配置的属性一一对应 |

UserMapperTest 增加

|  |
| --- |
| @Test public void testInsert2() {   SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  SysUser user = new SysUser();  user.setUserName("test1");  user.setUserPassword("123456");  user.setUserEmail("test@mybatis.tk");  user.setUserInfo("test info");  user.setHeadImg(new byte[]{1, 2, 3});  user.setCreateTime(new Date());   int result = userMapper.insert2(user);  *assertEquals*(result, 1);  // 返回id  *assertNotNull*(user.getId());   } finally {  sqlSession.rollback();  sqlSession.close();  } } |

### 2.4.3使用selectKey方式返回主键自增的值

在UserMapper.java增加

|  |
| --- |
| */\*\*  \* 新增用户-使用selectKey方式  \** ***@param*** *sysUser  \** ***@return*** *\*/* int insert3(SysUser sysUser); |

在UserMapper.xml 增加

|  |
| --- |
| <insert id="insert3">  insert into sys\_user (user\_name,user\_password,user\_email,user\_info,head\_img,create\_time)  values (  #{userName} ,#{userPassword} ,#{userEmail} ,#{userInfo} ,#{headImg,jdbcType=BLOB},  #{createTime,jdbcType=TIMESTAMP}  );   <!--  order 属性的设置和使用的数据库有关。在MySQL数据库中，order属性设置的值是AFTER，因为当前记录的主键值在  insert语句执行成功后才能获取到，在Oracle数据库中，order值设置为BEFORE，Oracle中更需要先从序列获取值，  然后将值作为主键插入到数据库中  -->  <selectKey keyColumn="id" resultType="long" keyProperty="id" order="AFTER" >  select LAST\_INSERT\_ID()  </selectKey> </insert>  不同数据库的selectKey中回写主键的SQL  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();  SYSBASE:SELECT @@IDENTITY;  DB2\_MF:SELECT IDENTITY\_VAL\_LOCAL() FROM SYSIBM.SYSDUMMY1;  INFORMIX:select dbinfo(‘sqlca.sqlerrdl’) from systables where tabid=1 |

UserMapperTest 增加

|  |
| --- |
| @Test public void testInsert3() {   SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  SysUser user = new SysUser();  user.setUserName("test1");  user.setUserPassword("123456");  user.setUserEmail("test@mybatis.tk");  user.setUserInfo("test info");  user.setHeadImg(new byte[]{1, 2, 3});  user.setCreateTime(new Date());   int result = userMapper.insert3(user);  *assertEquals*(result, 1);  // 返回id  *assertNotNull*(user.getId());   } finally {  sqlSession.rollback();  sqlSession.close();  } } |

## 2.5 update用法

在UserMapper.java增加

|  |
| --- |
| */\*\*  \* 根据主键个更新  \** ***@param*** *sysUser  \** ***@return*** *\*/* int updateById(SysUser sysUser); |

在UserMapper.xml 增加

|  |
| --- |
| <update id="updateById" >  update sys\_user  set user\_name = #{userName} ,  user\_password = #{userPassword} ,  user\_email = #{userEmail} ,  user\_info = #{userInfo} ,  head\_img = #{headImg,jdbcType=BLOB},  create\_time = #{createTime,jdbcType=TIMESTAMP}  where id = #{id} </update> |

UserMapperTest 增加

|  |
| --- |
| @Test public void testUpdateById() {   SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);   SysUser user = userMapper.selectById(1L);  *assertEquals*("admin", user.getUserName());  user.setUserName("admin\_test");  user.setUserEmail("test@mybatis.tk");   int result = userMapper.updateById(user);  *assertEquals*(result, 1);  user = userMapper.selectById(1L);  *assertEquals*("admin\_test", user.getUserName());  } finally {  sqlSession.rollback();  sqlSession.close();  } } |

## 2.6 delete用法

在UserMapper.java增加

|  |
| --- |
| */\*\*  \* 通过主键删除  \** ***@param*** *id  \** ***@return*** *\*/* int deleteById(Long id); |

在UserMapper.xml 增加

|  |
| --- |
| <delete id="deleteById" >  delete from sys\_user where id = #{id} </delete> |

UserMapperTest 增加

|  |
| --- |
| @Test public void testDeleteById() {   SqlSession sqlSession = *getSqlSession*();  try {  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);   SysUser user1 = userMapper.selectById(1L);  *assertNotNull*(user1);   *assertEquals*(1,userMapper.deleteById(1L));  *assertNull*(userMapper.selectById(1L));    } finally {  sqlSession.rollback();  sqlSession.close();  } } |