# L1、搭建环境

## 创建数据库

CREATE DATABASE IF NOT EXISTS study default charset utf8mb4 collate utf8mb4\_general\_ci;

DROP TABLE IF EXISTS user;

CREATE TABLE user (

id bigint(20) NOT NULL AUTO\_INCREMENT,

username varchar(255) NOT NULL,

age int(11) DEFAULT NULL,

gender tinyint(1) DEFAULT NULL,

birthday date DEFAULT NULL,

balance int(11) DEFAULT NULL,

register\_time datetime DEFAULT NULL ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (id)

) ENGINE=InnoDB AUTO\_INCREMENT=0 DEFAULT CHARSET=utf8mb4;

## pom.xml

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<!-- 包含spring-boot-starter-jdbc，可以使用JdbcTemplate -->

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.5</version>

<configuration>

<verbose>true</verbose>

<overwrite>true</overwrite>

</configuration>

</plugin>

</plugins>

</build>

## application.propertiesr

在src/main/resources下创建application.properties

debug=false

spring.datasource.url=jdbc:mysql://localhost:3306/study?useSSL=false&zeroDateTimeBehavior=convertToNull&useUnicode=true&characterEncoding=UTF-8&autoReconnect=true

spring.datasource.password=000000

spring.datasource.username=root

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

server.port=80

logging.file=log.log

logging.level.root=INFO

## generatorConfig.xml

在src/main/resources下创建generatorConfig.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN" "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd" >

<!-- 配置生成器 运行mvn mybatis-generator:generate -DskipTests -->

<generatorConfiguration>

<classPathEntry

location=*"F:\repo\mysql\mysql-connector-java\5.1.42\mysql-connector-java-5.1.42.jar"* />

<context id=*"mysql"* targetRuntime=*"MyBatis3Simple"*>

<property name=*"javaFileEncoding"* value=*"UTF-8"* />

<jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*

connectionURL=*"jdbc:mysql://localhost:3306/study?useSSL=false"*

userId=*"root"* password=*"000000"*>

</jdbcConnection>

<javaModelGenerator targetPackage=*"com.study.mybatis.dmo"*

targetProject=*"src/main/java"*>

</javaModelGenerator>

<sqlMapGenerator targetPackage=*"com.study.mybatis.dao"*

targetProject=*"src/main/resources"*>

</sqlMapGenerator>

<javaClientGenerator targetPackage=*"com.study.mybatis.dao"* type=*"XMLMAPPER"*

targetProject=*"src/main/java"*>

</javaClientGenerator>

<table tableName=*"user"* domainObjectName=*"UserDmo"* mapperName=*"UserDao"*>

<property name=*"constructorBased"* value=*"true"* />

<generatedKey column=*"id"* sqlStatement=*"MySql"* identity=*"true"* />

<!—默认是驼峰命名法，如果需要重命名可以添加 <columnOverride column="gender" property="sex" javaType="java.lang.Boolean"/> -->

</table>

</context>

</generatorConfiguration>

## DataConfig

**import** javax.sql.DataSource;

**import** org.mybatis.spring.SqlSessionFactoryBean;

**import** org.mybatis.spring.mapper.MapperScannerConfigurer;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.jdbc.datasource.DataSourceTransactionManager;

@Configuration

**public** **class** DataConfig {

@Bean

**public** MapperScannerConfigurer mapperScannerConfigurer() {

MapperScannerConfigurer configurer = **new** MapperScannerConfigurer();

configurer.setBasePackage("com.study.mybatis.dao");

configurer.setSqlSessionFactoryBeanName("sqlSessionFactoryBean");

**return** configurer;

}

@Bean

**public** SqlSessionFactoryBean sqlSessionFactoryBean(DataSource dataSource) {

SqlSessionFactoryBean bean = **new** SqlSessionFactoryBean();

bean.setDataSource(dataSource);

**return** bean;

}

}

## 事务

使用生成器自动生成的mapper，自动带了事务，可在web层直接调用

在业务层使用事务，需要在类上或者方法上添加注解

import org.springframework.transaction.annotation.Transactional;

@Transactional(rollbackFor = Throwable.class)

## 测试

在src/test/java下建立测试包和类

@RunWith(SpringRunner.**class**)

@SpringBootTest

//默认测试生效

@Rollback(**false**)

**public** **class** TestBo {

@Autowired

UserBo userBo;

//如果userBo中事务有异常，会回滚

@Test

**public** **void** userInsert() **throws** Exception {

UserDmo user = **new** UserDmo();

user.setUsername("mybatis");

user.setAge(30);

user.setBalance(100);

Calendar cal = Calendar.*getInstance*();

//月份从0开始，日期为1992-11-17

cal.set(1992, 10, 17);

user.setBirthday(cal.getTime());

user.setGender(**true**);

user.setRegisterTime(**new** Date());

**this**.userBo.insert(user);

}}

# L2、mapper.xml配置简单的crud操作

## 根元素

<?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.study.mybatis.dao.UserDao"*>

namespace=dao对象的完全限定名

## ResultMap

1. 用来查询时设置返回值。
2. type 为返回类型
3. 子元素id,restult中
4. column为数据表中的列名或别名，不区分大小写
5. jdbcType可选，mybatis可以自动识别，为数据表中列的类型，必须是org.apache.ibatis.type.JdbcType枚举中的值
6. properties为类中的属性名

<resultMap id=*"BaseResultMap"* type=*"com.study.mybatis.dmo.UserDmo"*>

<id column=*"id"* jdbcType=*"BIGINT"* property=*"id"* />

<result column=*"username"* jdbcType=*"VARCHAR"* property=*"username"* />

<result column=*"age"* jdbcType=*"INTEGER"* property=*"age"* />

<result column=*"gender"* jdbcType=*"BIT"* property=*"gender"* />

<result column=*"birthday"* jdbcType=*"DATE"* property=*"birthday"* />

<result column=*"balance"* jdbcType=*"INTEGER"* property=*"balance"* />

<result column=*"register\_time"* jdbcType=*"TIMESTAMP"* property=*"registerTime"* />

</resultMap>

## select

1. id为接口中的方法名
2. resultMap 设置返回类型，为外部的resultMap元素的id。
3. resultType 设置返回类型，值一般为Java自带的基本类型包装类的完全限定名或者map,hashmap,如果为map,返回值的map key为列名，如果类的属性名和数据库中的列名相同(不区分大小写)可以使用自定义的类名，值为对应的值。如果是集合情况，返回类型应该是集合中的元素类型，而不是集合本身。
4. resultMap和resultType不能同时使用。
5. parameterType 为参数类的完全限定名，是可选的，因为MyBatis可以通过TypeHandler自动识别
6. 在select语句中，#{参数名} 参数名可以是接口方法中的参数，也可以是参数对象中的属性。例如username为UserDmo中的属性。
7. 如果传多个参数，在参数前添加@param(参数值)；或者封装成Map

List<UserDmo> selectByExample(UserDmo example);

<select id=*"selectByExample"* resultMap=*"BaseResultMap"*>

select id, username, age, gender, birthday, balance,

register\_time

from user where username=#{username}

</select>

**import** org.apache.ibatis.annotations.Param;

**int** countByGenderAndAgeLessThan(@Param("gender") **boolean** gender, @Param("age") **int** age);

<select id=*"countByGenderAndAgeLessThan"* resultType=*"java.lang.Integer"*>

select count(1) from user where age &lt; #{age} and gender=#{gender}

</select>

## insert、update、delete

1. insert

**void** insert(UserDmo record);//或者返回int类型

<insert id=*"insert"*>

insert into user (username, age,gender,birthday, balance,register\_time)

values (#{username},#{age},#{gender},#{birthday},#{balance},#{registerTime})

</insert>

1. update

**int** updateByPrimaryKey(UserDmo record);

<update id=*"updateByPrimaryKey"* >

update user

set username = #{username},

age =#{age},

gender = #{gender},

birthday =#{birthday},

balance = #{balance},

register\_time = #{registerTime}

where id =#{id}

</update>

1. delete

**int** deleteByPrimaryKey(Long id);

<delete id=*"deleteByPrimaryKey"* >

delete from user where id = #{id}

</delete>

## #{}与${}

1、#传的参数会自动添加引号转为字符串，从而防止注入

2、$传的参数不会加引号，无法防止注入。一般用于传入表名，列名

例如

List<Map<String, Object>> findAll(@Param("table") String table, @Param("order") String order);

<select id=*"findAll"* resultType=*"map"*>

select \*

from ${table}

order by ${order} desc

</select>

# L3、缓存机制

1. 一级缓存
2. Mybatis的一级缓存是SqlSession级别的，作用域是SqlSession范围。
3. 当执行相同的sql查询语句(参数也相同)时，第一次会将查到的数据写到缓存(内存)，第二次查询就直接从缓存中获取数据，不再查询数据库。
4. 当SqlSession执行DML(insert,update,delete)操作，并提交后，Mybatis会清空SqlSession中的一级缓存，以保证缓存中存储的数据是最新信息。当执行insert后，对象会自动活得ID。当执行update后，对象会自动更新为新对象。
5. 一级缓存是默认开启的，不需要任何配置。
6. 二级缓存
7. 二级缓存是Mapper级别的，作用域是mapper的namespace。
8. 二级缓存由多个SqlSession共享。不同的SqlSession执行相同的namespace中的sql语句(参数也相同)，第一次会从数据库中查询并将查出来的数据写进缓存，第二次查询时直接从缓存中获取数据，不再查询数据库。
9. 使用二级缓存时，与查询结果映射的对象必须实现java.id.serializable接口。
10. 在一个mapper中，可以通过namespace.id引用到另一个mapper.xml中的对象。
11. Mybatis默认没有开启二级缓存，开启需要在application.properties中添加

mybatis.configuration.cache-enabled=true

1. 可在mapper的namespace中定义cache元素设置二级缓存，开启后，dmo对象必须实现序列化接口

<cache eviction=*"LRU"* flushInterval=*"60000"* size=*"512"* readOnly=*"false"* />

eviction:回收策略，默认是LRU。

* + LRU：最近最少使用的策略，移除最长时间不被使用的对象
  + FIFO：先进先出策略，按对象进入缓存的顺序移除他们
  + SOFT：软引用策略，移除基于垃圾回收期状态和软引用规则的对象
  + WEAK：弱引用策略，更积极地移除基于垃圾回收器状态和弱引用规则的对象

flushInterval:刷新间隔。单位为毫秒。默认情况下是不设置的，也没有刷新间隔。

size:缓存数目。默认是1024。

readOnly:默认为false。如果为true会给所有调用者返回对象的相同实例，因此对象不能被修改。如果为false会返回对象的拷贝，这样会更慢一些，但是安全。

# L4、映射关系

1. 准备数据库

DROP TABLE IF EXISTS clazz;

CREATE TABLE clazz (

id bigint(20) NOT NULL AUTO\_INCREMENT,

name varchar(255) DEFAULT NULL,

PRIMARY KEY (id)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4;

DROP TABLE IF EXISTS student;

CREATE TABLE student (

id bigint(20) NOT NULL AUTO\_INCREMENT,

name varchar(255) NOT NULL,

age int(11) DEFAULT NULL,

gender tinyint(1) DEFAULT NULL,

birthday date DEFAULT NULL,

clazz bigint(20) DEFAULT NULL,

PRIMARY KEY (id),

KEY clazz\_id (clazz),

CONSTRAINT student\_ibfk\_1 FOREIGN KEY (clazz) REFERENCES clazz (id)

) ENGINE=InnoDB AUTO\_INCREMENT=40 DEFAULT CHARSET=utf8mb4;

INSERT INTO clazz VALUES ('4', '1班');

INSERT INTO clazz VALUES ('5', '2班');

INSERT INTO clazz VALUES ('6', '3班');

INSERT INTO student VALUES ('35', '张三', '18', '1', '2017-07-03', '4');

INSERT INTO student VALUES ('36', '李四', '19', '1', '2017-07-03', '5');

INSERT INTO student VALUES ('37', '王二', '19', '1', '2017-07-03', '4');

INSERT INTO student VALUES ('38', '麻', '19', '1', '2017-07-03', '5');

1. 多对一

在Student类中添加属性

**private** ClazzDmo clazz;//getter,setter略

在mapper.xml中

<resultMap id=*"BaseResultMap"* type=*"com.study.mybatis.dmo.StudentDmo"*>

<id property=*"id"* column=*"id"* />

<result property=*"name"* column=*"name"* />

<result property=*"birthday"* column=*"birthday"* />

<result property=*"age"* column=*"age"* />

<result property=*"gender"* column=*"gender"* />

<association property=*"clazz"* column=*"clazz"*

javaType=*"com.study.mybatis.dmo.ClazzDmo"*>

<id column=*"cid"* property=*"id"* />

<result column=*"cname"* property=*"name"* />

</association>

</resultMap>

<select id=*"selectAll"* resultMap=*"BaseResultMap"*>

<include refid=*"select"* />

</select>

<sql id=*"select"*>

select s.id, s.name, s.age, s.gender, s.birthday, c.name

cname,c.id cid

from student s inner join clazz c on s.clazz=c.id

</sql>

1. 一对多

在ClazzDmo中添加属性

**private** List<StudentDmo> students; //getter,setter略

在mapper中添加

<resultMap id=*"BaseResultMap"* type=*"com.study.mybatis.dmo.ClazzDmo"*>

<id column=*"id"* property=*"id"* />

<result column=*"name"* property=*"name"* />

<collection property=*"students"* javaType=*"java.util.ArrayList"*

column=*"id"* select=*"com.study.mybatis.dao.StudentDao.selectStudentByClazz\_Id"*

fetchType=*"lazy"*>

<!-- column为主键表中的ID,用于嵌套查询SQL语句传入参数 -->

<id property=*"id"* column=*"id"* />

<result property=*"name"* column=*"name"* />

<result property=*"birthday"* column=*"birthday"* />

<result property=*"age"* column=*"age"* />

<result property=*"gender"* column=*"gender"* />

</collection>

</resultMap>

在namespace为*com.study.mybatis.dao.StudentDao*的mapper中添加

<select id=*"selectStudentByClazz\_Id"* resultMap=*"BaseResultMap"*>

<include refid=*"select"* />

where c.id=#{cid}

</select>

在application.properties中启用懒加载

mybatis.configuration.lazy-loading-enabled=true

mybatis.configuration.aggressive-lazy-loading=false