MyBatis

1 学习目标

1. 了解MyBatis的介绍和历史
2. **重点掌握**SpringBoot整合MyBatis
3. **重点掌握**MyBatis基于注解方式
4. **重点掌握**MyBatis基于XML方式

2 MyBatis介绍

- MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。
- MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
- MyBatis可以使用简单的XML或注解用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录.

3 MyBatis的历史

- 原是Apache的一个开源项目**iBatis**, 该项目最初由Clinton Begin创建。2005年，该项目被提交到了Apache Software Foundation。但是由于名称与IBM拥有的商标iSeries和DB2的i系列冲突，因此项目名称与2010年被更改为MyBatis。

4 SpringBoot整合MyBatis

4.1 项目准备

①在JSDSecondStage项目下,创建**MyBatisDemo**模块,并指定版本号为2.5.4

②为项目添加相关依赖

注意: 无需引入spring对jdbc的驱动,因为MyBatis会自动引入

```
<!--mysql数据库驱动依赖-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<!--引入相关mybatis依赖-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.2.0</version>
</dependency>
```

4.2 配置数据源

- 在application.yml文件中设置基础配置

```yaml
#设置连接数据库的url、username、password，这三部分不能省略
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/tedu?serverTimezone=Asia/Shanghai&characterEncoding=utf8&serverTimeZone=Asia/Shanghai
    username: root
    password: root
```

5 MyBatis基于注解方式

5.1 编写实体类

- `ORM(Object Relational Mapping)`：对象关系映射,指的是持久化数据和实体对象的映射模式，为了解决面向对象与关系型数据库存在的互不匹配的现象的技术。其具体的映射规则是:一张表对应一个类，表中的各个字段对应类中的属性，表中的一条数据对应类的一个对象
- MyBatis可以自动将查询的结果与对应实体类映射,所以需要准备与查询的表相关联的实体类,需要注意的是,实体类的属性需要和表字段保持一致
- 由于此处我们的入门案例是查询teacher表,所以创建Teacher类,并在该类中声明与teacher表字段相同的属性(属性声明时,最好开启驼峰规则)

① `Teacher`

```java
public class Teacher {
    private long id;
    private String name;
    private long age;
    private String title;
    private long manager;
    private long salary;
    private long comm;
    private String gender;
    private long subjectId;

    //get和set方法,toString方法
}
```

5.2 定义接口和方法

- MyBatis需要准备一个接口类,作为Mapper,该接口中的每一个方法可以绑定一条SQL,实现调用接口方法即可调用SQL的功能
- 在 MyBatis 中，Mapper 接口的命名一般要遵循一定的规范，以便于开发人员理解和维护代码。以下是一些常用的命名规范：
  - Mapper 接口的名称应该与对应的 SQL 语句相对应，可以根据表名或者业务功能命名，比如 UserMapper、OrderMapper 等。
  - Mapper 接口的方法名应该能够清晰地表示这个方法所执行的 SQL 语句，一般可以使用动词加上表名或者业务功能的方式来命名，比如 addUser、updateOrder 等。
  - Mapper 接口中的方法参数名应该与 SQL 语句中的参数名相对应，这样可以提高代码的可读性和可维护性。

① `TeacherMapper`

```java
//指定这是一个操作数据库的mapper
@Mapper
public interface TeacherMapper {
    @Select("SELECT * FROM teacher")
    public List<Teacher> getTeacherAll();
}
```

② `TestMyBatisAnno`

```java
/**
 * 用于测试基于MyBatis注解开发的入门案例
 */
@SpringBootTest
public class TestMyBatisAnno {
    @Autowired
    private LocationsMapper locationsMapper;

    @Test
    public void testSelectAll() {
        List<Teacher> teacherAll = teacherMapper.getTeacherAll();
        for (Teacher teacher : teacherAll) {
            System.out.println(teacher);
        }
    }
}
```

③ `application.yml开启驼峰规则`

```yaml
#开启驼峰规则
mybatis:
  configuration:
    map-underscore-to-camel-case: true
```

5.3 测试增删改查

① `TeacherMapper`

```java
//指定这是一个操作数据库的mapper
@Mapper
public interface TeacherMapper {
    @Select("SELECT * FROM teacher")
    public List<Teacher> getTeacherAll();

    @Insert("INSERT INTO teacher VALUES (6666,'光头师傅',22,'宗师',null,100000,50000,'男',0);")
    public int addTeacher();

    @Update("UPDATE teacher SET salary = 1000 WHERE name = '光头师傅'")
    public int updateTeacher();

    @Delete("DELETE FROM teacher WHERE name = '光头师傅'")
    public int deleteTeacher();
}
```

② `TestMyBatisAnno`

```java
@SpringBootTest
class TestMyBatisAnno {
    @Autowired
    private TeacherMapper teacherMapper;

    @Test
```

```
    public void testSelectAll() {
        List<Teacher> teacherAll = teacherMapper.getTeacherAll();
        for (Teacher teacher : teacherAll) {
            System.out.println(teacher);
        }
    }
    @Test
    public void testAddLocations() {
        int rows = teacherMapper.addTeacher();
        System.out.println(rows > 0 ? "新增成功!" : "新增失败!!");
    }

    @Test
    public void testUpdateLocations() {
        int rows = teacherMapper.updateTeacher();
        System.out.println(rows > 0 ? "修改成功!" : "修改失败!!");
    }

    @Test
    public void testDeleteLocations() {
        int rows = teacherMapper.deleteTeacher();
        System.out.println(rows > 0 ? "删除成功!" : "删除失败!!");
    }
}
```

5.4 @MapperScan的使用

- 由于Mapper接口是需要使用@Mapper注解的,但是随着后续的开发,可能这样的Mapper接口会变得很多,那么可能每个接口都要加@Mapper注解,就会变得很麻烦,所以可以在SpringBoot的主启动类上添加@MapperScan注解,并指定要扫描的mapper包,这样的话,就会自动去获取指定包下的接口了

① `MyBatisDemoApplication`

```
//指定mapper接口所在包,当主启动类执行时,会自动扫描指定包下的接口
@MapperScan(value = "cn.tedu.mapper")
@SpringBootApplication
public class MyBatisDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyBatisDemoApplication.class, args);
    }

}
```

6 MyBatis基于XML文件

6.1 编写实体类

- 由于此处我们的入门案例是查询jobs表,所以将Jobs类复制到项目中

6.2 定义接口和方法

- MyBatis需要准备一个接口类,作为Mapper,该接口中的每一个方法可以绑定一条SQL,实现调用接口方法即可调用SQL的功能

① `StudentMapper`

```java
package cn.tedu.mapper;

import com.pojo.Student;
import java.util.List;

public interface StudentMapper {
    public List<Student> getStudentAll();
}
```

6.3 定义SQL文件

- MyBatis基于XML文件的方式,是通过让SQL文件和指定的接口绑定,然后SQL语句统一书写在XML文件中
- 一般情况下,MyBatis的mapper.xml和mapper接口会同名,原因是为了方便开发人员进行代码的管理和维护。

① `StudentMapper.xml文件`

```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="cn.tedu.mapper.StudentMapper">

</mapper>
```

6.4 定义SQL

- 在SQL文件中,可以书写任意的SQL,只不过需要写在对应的标签中,并且id还要和对应的接口名相同
- 在SQL文件中通过**select**标签定义查询表所有记录的SQL语句
  - id的值必须要和绑定的接口中的方法名相同
  - **resultType**则表示查询的结果封装到那个实体类中,也就是返回值的类型,如果返回的是集合,则定义集合中元素的类型
- 在SQL文件中通过**insert**,**update**,**delete**标签定义增删改表中记录的SQL语句

① `StudentMapper.xml文件`

```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="cn.tedu.mapper.JobsMapper">
    <select id="getStudentAll" resultType="cn.tedu.pojo.Student">
        SELECT id, name, age, gender, job, birth, location_id, team_leader, class_id FROM student
    </select>
</mapper>
```

6.5 指定MyBatis中mapper文件扫描路径

- MyBatis在使用XML方式开发时,必须要在配置文件中指定mapper文件所在路径,否则会找不到资源

```yaml
mybatis:
  mapper-locations: classpath: mapper文件所在路径
```

---

① `application.yml`

```yaml
#设置连接数据库的url、username、password，这三部分不能省略
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/tedu?
serverTimezone=Asia/Shanghai&characterEncoding=utf8
    username: root
    password: root
#开启驼峰规则
mybatis:
  configuration:
    map-underscore-to-camel-case: true
  #指定mapper文件路径
  mapper-locations: classpath:mapper/*.xml
```

6.6 测试查询

① `TestMyBatisXml`

```java
/**
 * 用于测试基于MyBatis注解开发的入门案例
 */
@SpringBootTest
public class TestMyBatisXML {
    @Autowired
    private StudentMapper studentMapper;

    @Test
    public void testGetStudentAll() {
        List<Student> all = studentMapper.getStudentAll();
        for (Student student : all) {
            System.out.println(student);
        }
    }
}
```

6.7 测试增删改操作

① `StudentMapper接口`

```java
public interface StudentMapper {
    public List<Student> getStudentAll();

    public int insertStudent();

    public int updateStudent();

    public int deleteStudent();
}
```

② `StudentMapper.xml`

```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="tedu.mapper.StudentMapper">
    <select id="getStudentAll" resultType="tedu.pojo.Student">
        SELECT id,
               name,
```

```
                age,
                gender,
                job,
                birth,
                location_id,
                team_leader,
                class_id
        FROM student
    </select>
    <insert id="insertStudent">
        INSERT INTO student
        values ('7777', '猪八戒', 3000, '男', '分家大队长', NULL, 0, 0, 0)
    </insert>
    <update id="updateStudent">
        UPDATE student
        SET job='净坛使者'
        WHERE name = '猪八戒'
    </update>
    <delete id="deleteStudent">
        DELETE
        FROM student
        WHERE name = '猪八戒'
    </delete>
</mapper>
```

③ `TestMyBatisXml`

```
@SpringBootTest
class TestMyBatisXML {
    @Autowired
    private StudentMapper studentMapper;

    @Test
    public void testGetStudentAll() {
        List<Student> all = studentMapper.getStudentAll();
        for (Student student : all) {
            System.out.println(student);
        }
    }
    @Test
    public void testInsertStudent() {
        int rows = studentMapper.insertStudent();
        System.out.println(rows > 0 ? "新增成功!" : "新增失败!!");
    }
    @Test
    public void testUpdate() {
        int rows = studentMapper.updateStudent();
        System.out.println(rows > 0 ? "修改成功!" : "修改失败!!");
    }

    @Test
    public void testDelete() {
        int rows = studentMapper.deleteStudent();
        System.out.println(rows > 0 ? "删除成功!" : "删除失败!!");
    }
}
```