4. Spring Boot中使用JdbcTemplate

个人觉得JdbcTemplate相较于MyBaits，Hibernate等数据库框架更容易上手，对SQL的操作也更为直观方便，所以在项目中也是一个不错的选择。在Spring Boot开启JdbcTemplate很简单，只需要引入 `spring-boot-starter-jdbc` 依赖即可。JdbcTemplate封装了许多SQL操作，具体可查阅官方文档 https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html。

引入依赖

spring-boot-starter-jdbc：

XML

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

数据库驱动为ojdbc6，数据源采用Druid。具体可参考https://mrbird.cc/Spring-Boot%E4%B8%AD%E4%BD%BF%E7%94%A8Mybatis.html。

代码编写

数据准备：

PL/SQL

```
CREATE TABLE "SCOTT"."STUDENT" (
  "SNO" VARCHAR2(3 BYTE) NOT NULL ,
  "SNAME" VARCHAR2(9 BYTE) NOT NULL ,
  "SSEX" CHAR(2 BYTE) NOT NULL
);

INSERT INTO "SCOTT"."STUDENT" VALUES ('001', 'KangKang', 'M ');
INSERT INTO "SCOTT"."STUDENT" VALUES ('002', 'Mike', 'M ');
INSERT INTO "SCOTT"."STUDENT" VALUES ('003', 'Jane', 'F ');
```

这里主要演示在Dao的实现类里使用JdbcTemplate，所以其它模块代码的编写就不展示了，具体可参考文末的源码。

StudentDaoImp类代码：

Java

```java
@Repository("studentDao")
public class StudentDaoImp implements StudentDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public int add(Student student) {
        // String sql = "insert into student(sno,sname,ssex) values(?,?,?)";
        // Object[] args = { student.getSno(), student.getName(), student.getSex() };
        // int[] argTypes = { Types.VARCHAR, Types.VARCHAR, Types.VARCHAR };
        // return this.jdbcTemplate.update(sql, args, argTypes);
        String sql = "insert into student(sno,sname,ssex) values(:sno,:name,:sex)";
        NamedParameterJdbcTemplate npjt = new NamedParameterJdbcTemplate(this.jdbcTemplate.getDataSource());
        return npjt.update(sql, new BeanPropertySqlParameterSource(student));
    }

    @Override
    public int update(Student student) {
        String sql = "update student set sname = ?,ssex = ? where sno = ?";
        Object[] args = { student.getName(), student.getSex(), student.getSno() };
        int[] argTypes = { Types.VARCHAR, Types.VARCHAR, Types.VARCHAR };
        return this.jdbcTemplate.update(sql, args, argTypes);
    }

    @Override
    public int deleteBysno(String sno) {
        String sql = "delete from student where sno = ?";
        Object[] args = { sno };
        int[] argTypes = { Types.VARCHAR };
        return this.jdbcTemplate.update(sql, args, argTypes);
    }

    @Override
    public List<Map<String, Object>> queryStudentsListMap() {
        String sql = "select * from student";
        return this.jdbcTemplate.queryForList(sql);
```

```
    }

    @Override
    public Student querystudentBySno(String sno) {
        String sql = "select * from student where sno = ?";
        Object[] args = { sno };
        int[] argTypes = { Types.VARCHAR };
        List<Student> studentList = this.jdbcTemplate.query(sql, args, argTypes, new StudentMapper());
        if (studentList != null && studentList.size() > 0) {
            return studentList.get(0);
        } else {
            return null;
        }
    }
}
```

在引入 `spring-boot-starter-jdbc` 驱动后，可直接在类中注入JdbcTemplate。由上面代码可发现，对于保存操作有两种不同的方法，当插入的表字段较多的情况下，推荐使用 `NamedParameterJdbcTemplate` 。

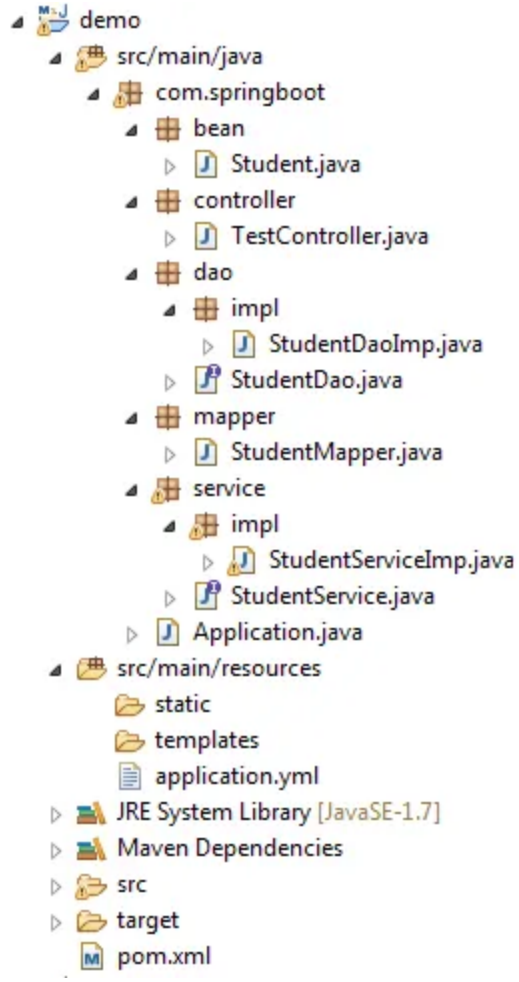
对于返回结果，可以直接使用 `List<Map<String, Object>>` 来接收，这也是个人比较推荐使用的方式，毕竟比较简单方便；也可以使用库表对应的实体对象来接收，不过这时候我们就需要手动创建一个实现了 `org.springframework.jdbc.core.RowMapper` 的对象，用于将实体对象属性和库表字段一一对应：

Java

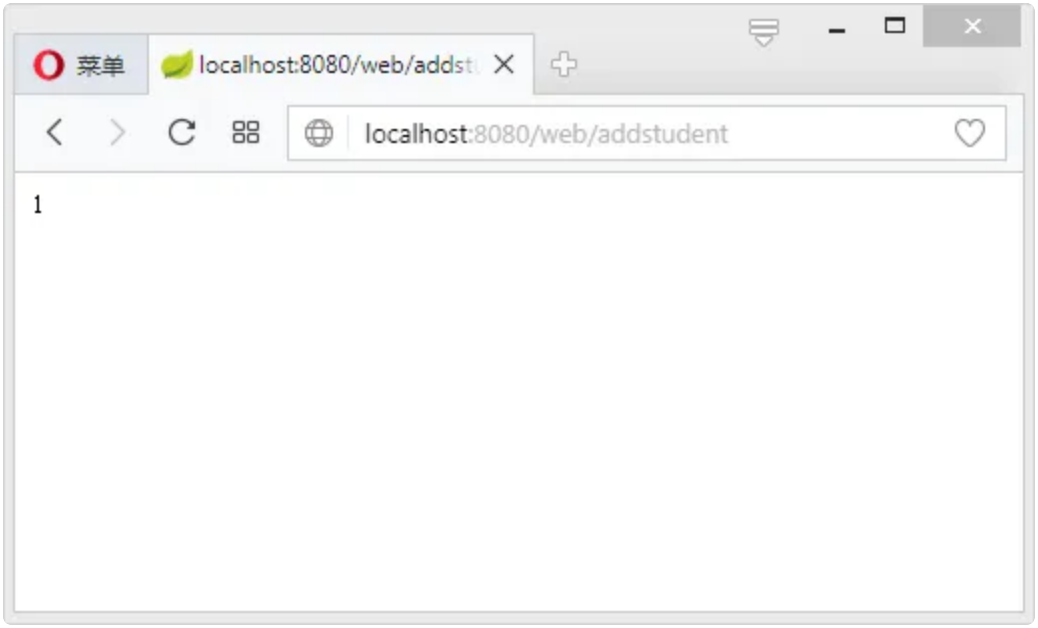
```
public class StudentMapper implements RowMapper<Student>{
    @Override
    public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
        Student student = new Student();
        student.setSno(rs.getString("sno"));
        student.setName(rs.getString("sname"));
        student.setSex(rs.getString("ssex"));
        return student;
    }
}
```

测试

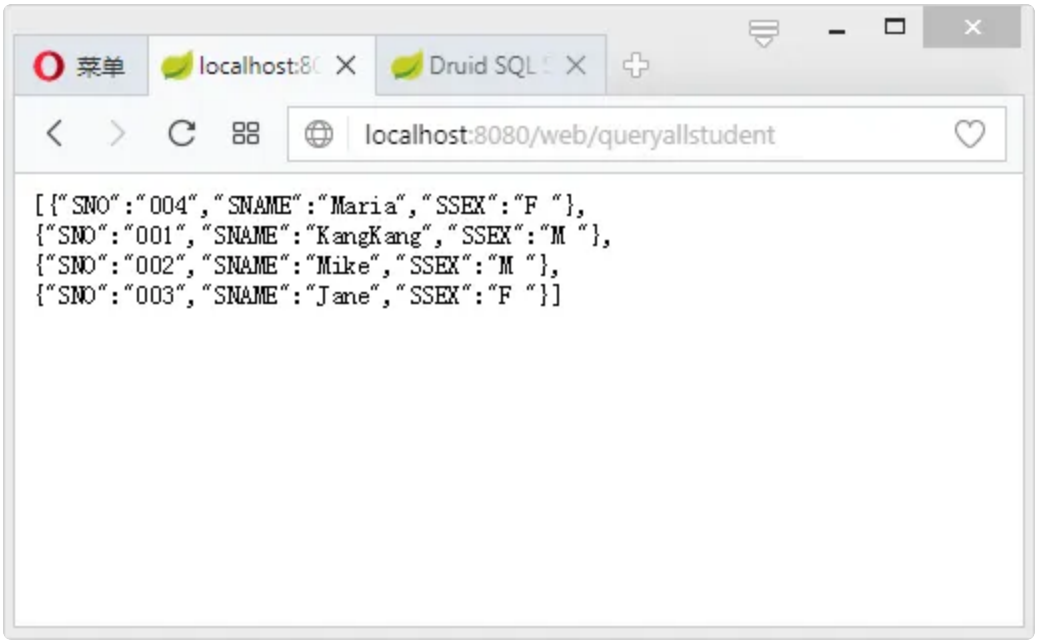
最终项目目录如下图所示：



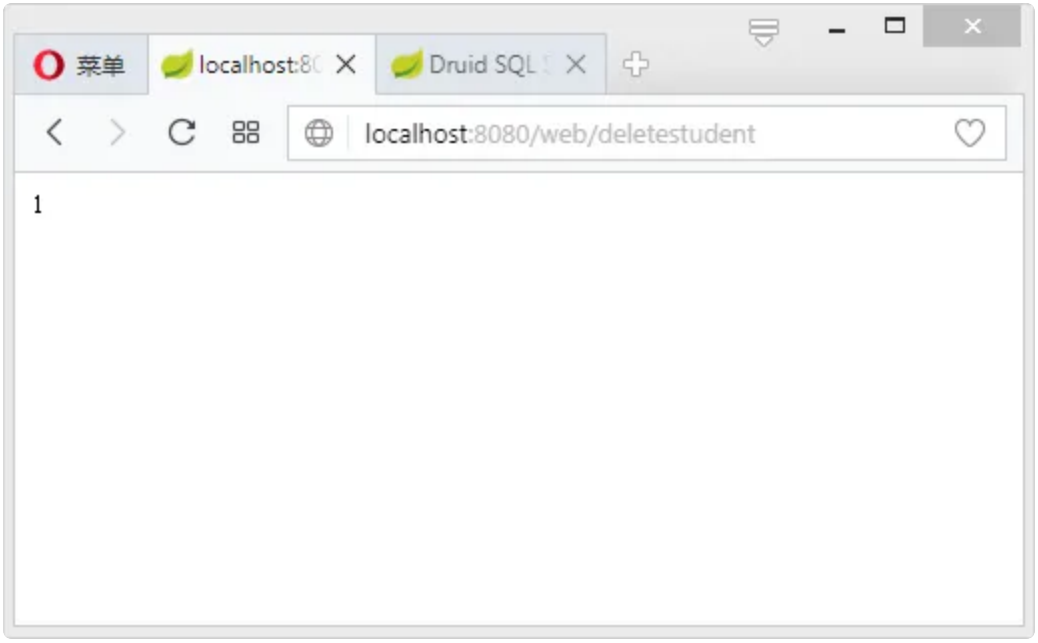
启动项目，测试插入数据http://localhost:8080/web/addstudent?sno=004&name=Maria&sex=F：



查询所有学生数据http://localhost:8080/web/queryallstudent:



测试删除http://localhost:8080/web/deletestudent?sno=004：



https://github.com/wuyouzhuguli/Spring-Boot-Demos/tree/master/04.Spring-Boot-JdbcTemplate