# 简介

Spring Boot对常见的持久层框架都提供了自动化配置，例如JdbcTemplate、JAP等，MyBatis的自动化配置时MyBatis官方提供的。

# 整合JdbcTemplate

JdbcTemplate是Spring提供的一套JDBC模板引擎，它虽然没有MyBatis那么灵活，但比JDBC要方便很多。

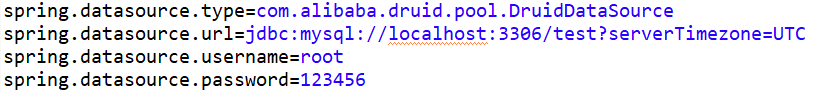
## 配置

1）添加依赖，如下：



druid是阿里巴巴的一个数据库连接池，性能优异。

2）在application.properties中配置数据库的基本信息，如下：



type表示数据源类型；

url表示数据库地址；

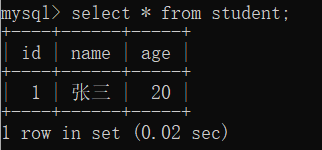
username表示用户名；

password表示密码；

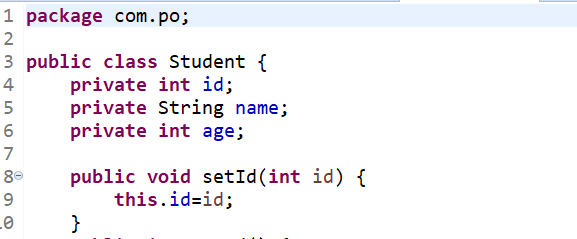
## 示例

使用JdbcTemplate示例如下：

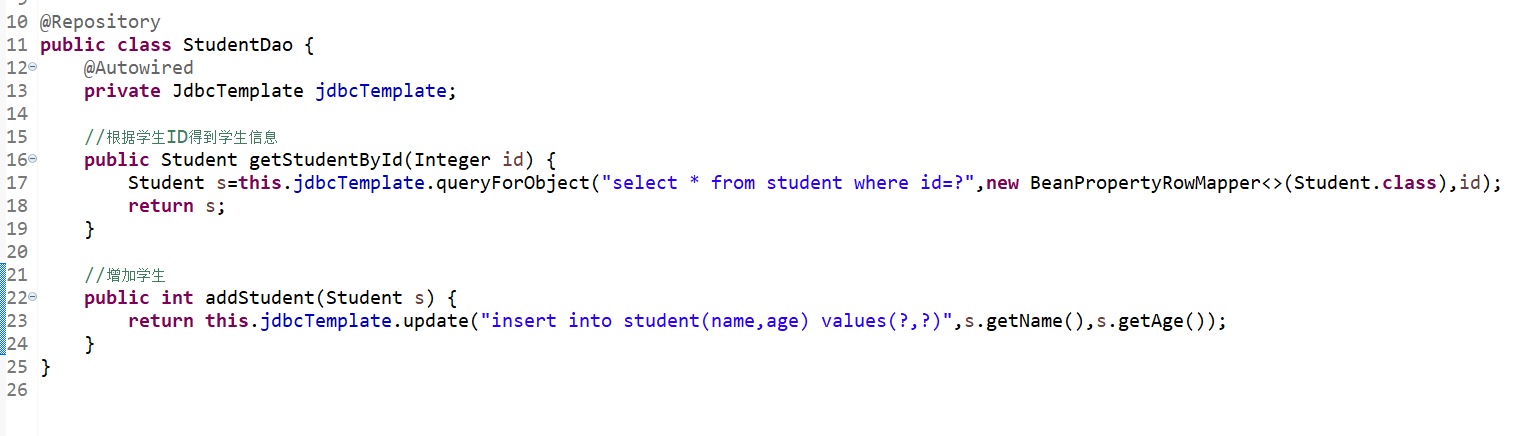
1. 在test库中存在表student，表中数据如下：



1. 创建student表对应的实体类，如下：



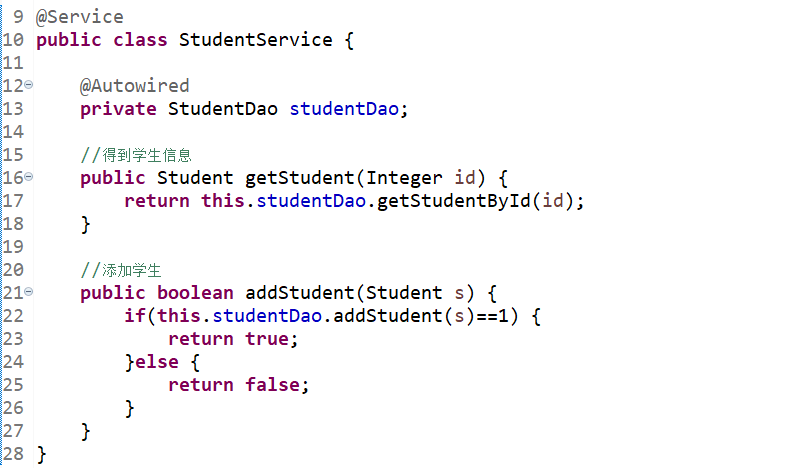
1. 创建数据库访问层，如下：



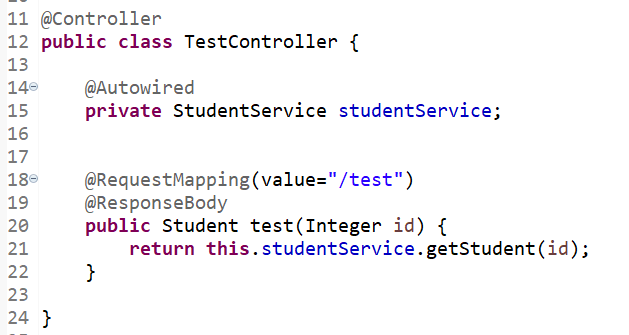
由于添加了spring-jdbc相关的依赖，所以JdbcTemplate会被注册到Spring容器中，在这里直接注入JdbcTemplate使用即可；

在JdbcTemplate中，增删改操作主要使用update和batchUpdate方法完成，查询操作主要使用queryhe1queryForObject完成，其余方法请自行查阅；

1. 创建Service，如下：

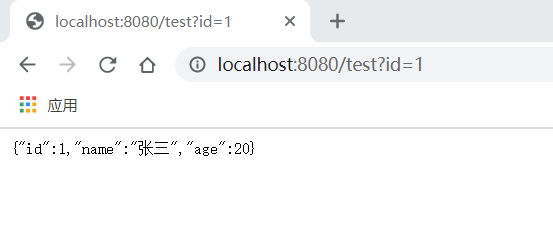


1. 创建Controller，如下：

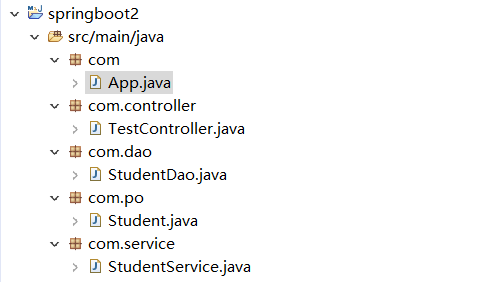


为了演示方便，将查询到的数据使用JSON的形式进行返回，而没有用到视图层技术。

1. 启动项目，打开浏览器访问，如下：



1. 示例项目结构如下：



关于JdbcTemplate更多的用法请查阅相关资料。

# 整合MyBatis

MyBatis是一款非常优秀的持久层框架，原名iBatis。在Spring中，MyBatis官方提供了一套自动化配置方案，可以做到MyBatis开箱即用。

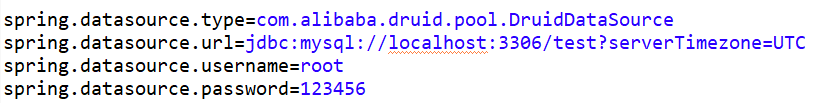
## 配置

1. 添加相关依赖，如下：



druid是阿里巴巴的一个数据库连接池，性能优异；

1. 在application.properties配置数据库连接信息，如下（以下只是基本配置）：



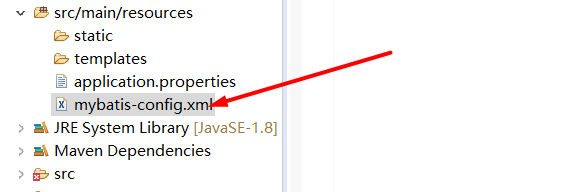
type表示数据源类型；

url表示数据库地址；

username表示用户名；

password表示密码；

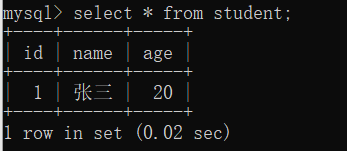
1. 在classpath下创建Mybatis配置文件，并在application.properties配置文件中指明其位置，如下：



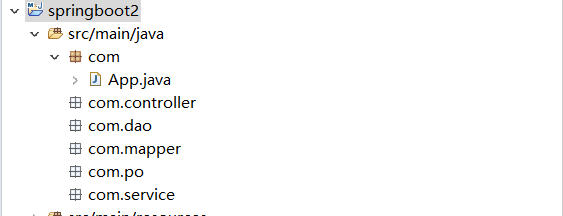


## 示例

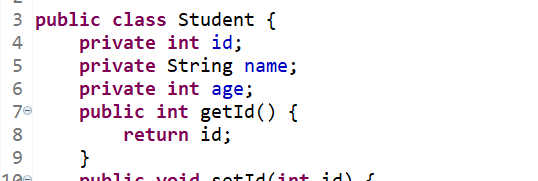
1. test库中存在表student，student表中数据如下：



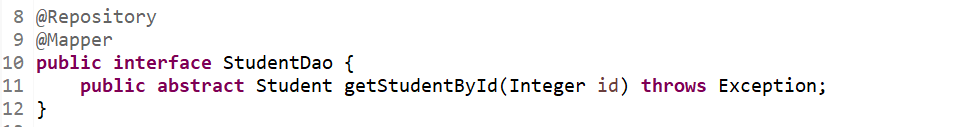
1. 示例项目目录结构如下：



1. 为student表创建实体类，如下

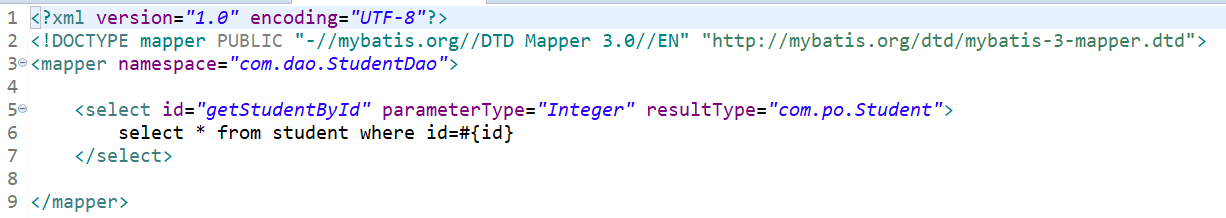


1. 创建映射器接口，如下：



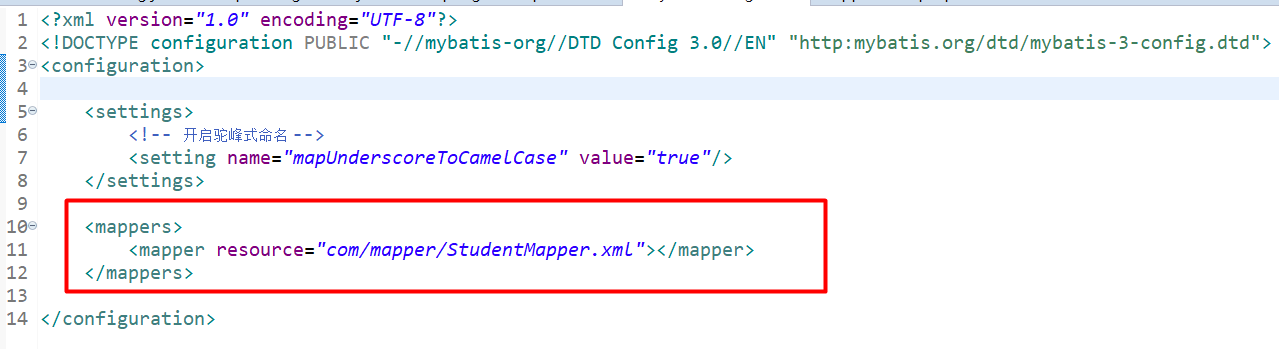
该接口中定义了一个方法，用于根据学生Id取得学生的信息；

1. 创建映射器Mapper文件，如下：

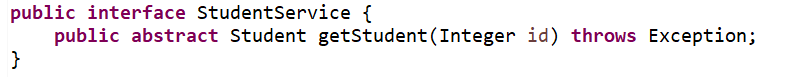


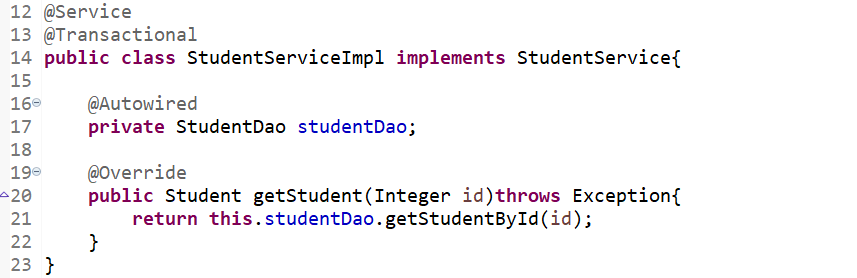
映射器接口和映射器Mapper文件共同组成Mybatis的映射器；

1. 在Mybatis配置文件中指明mapper文件所在位置，如下：

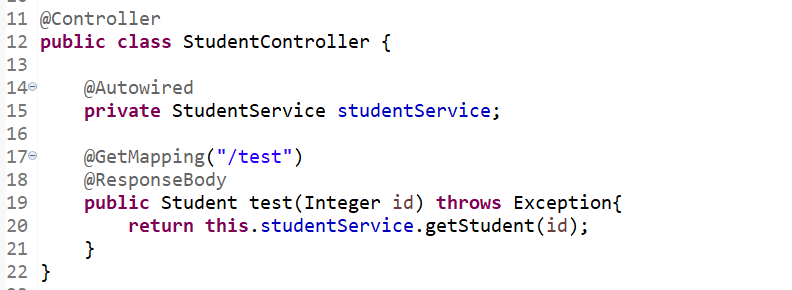


1. 创建业务层，包括接口和实现类，如下：



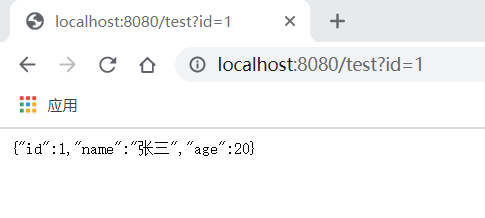


1. 创建控制层，如下：



为了方便，没有使用视图层技术，而是以JSON的形式将数据显示在页面上。

9）启动项目，打开浏览器访问，如下：



Spring Boot整合Mybatis成功。

# 整合Spring Data JPA

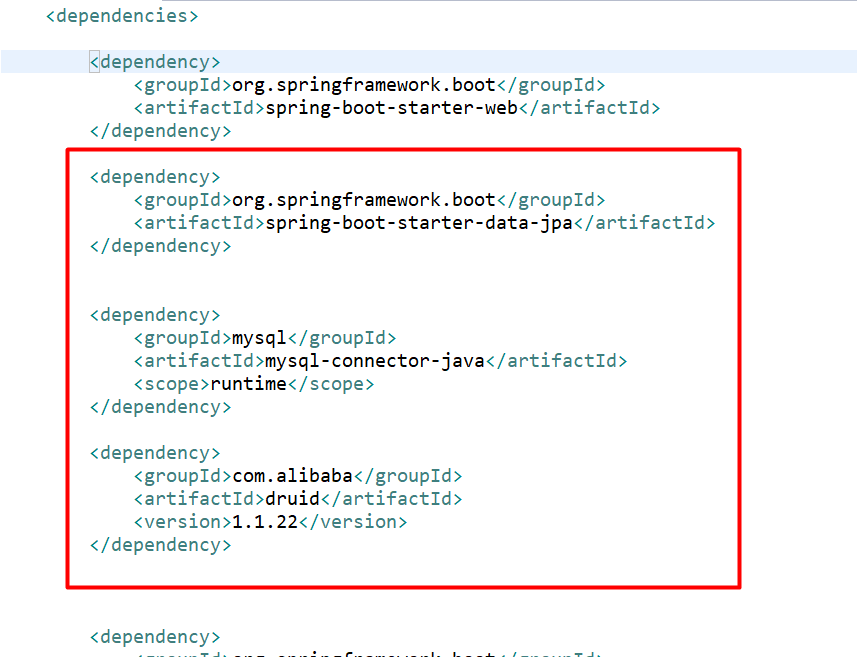
JPA和Spring Data是两个范畴的概念。

作为Java开发人员，应该对Hibernate不陌生，Hibernate是一个ORM（关系对象映射）框架，而JPA是一种ORM规范，即JPA指定了ORM规范，因此从功能上来说，JPA相当于Hibernate的一个子集；

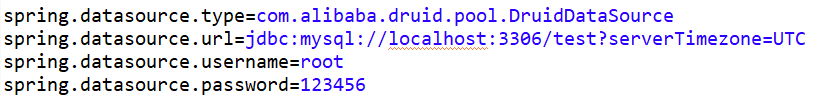
Spring Data是Spring的一个子项目，致力于简化数据库访问，通过规范的方法名称来分析开发者的意图，进而减少数据库访问层的代码量，Spring Data不仅支持关系型数据库，也支持非关系型数据库。Spring Data JPA可以有效简化关系型数据库访问代码。

## 配置

1. 添加依赖，如下：



1. 在application.properteis中配置数据库基本信息，如下：



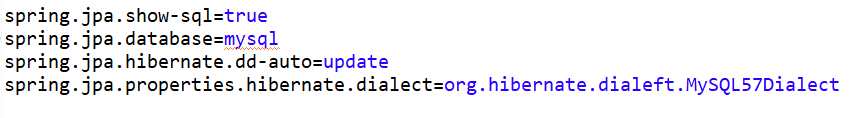
type表示数据源类型；

url表示数据库地址；

username表示用户名；

password表示密码；

1. 在application.properties中配置JAP，如下：



show-sql表示是否在控制台打印JPA执行过程生成的SQL；

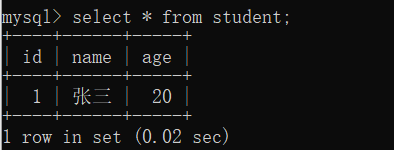
database表示JPA对应的数据库时MySQL；

hibername.dd-auto表示在项目启动时根据实体类更新数据库中的标识，值可以是create、create-drop、validate、update、no；

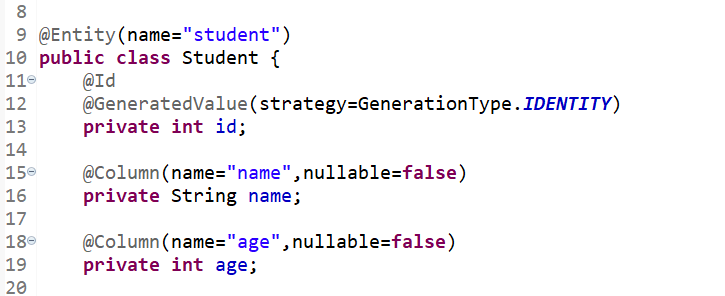
hibernate.dialect表示使用的数据库方言是MySQL57Dialect；

## 示例

1. 库test存在表student，内容如下：



1. 创建Student实体类，如下：



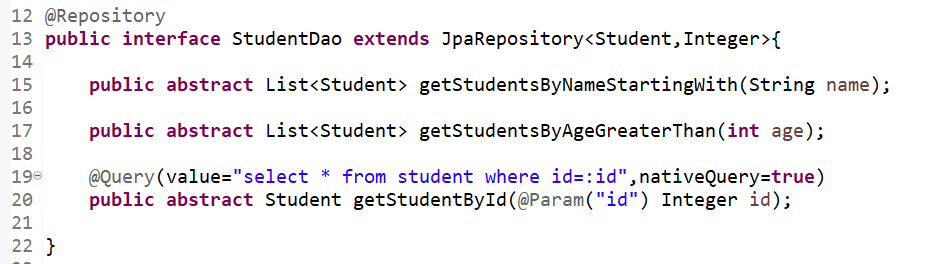
@Entity表示该类是一个实体类，name属性用于指定表；

@Id标注的属性表示该属性是一个主键；@GeneratedValue标注的属性表示该主键自动生成；

@Column标注的属性对应数据表的一个字段，name参数指定字段名，nullable参数表示其是否能为空；

@Transient标注的属性表示在生成数据库中的数据库表时，忽略该属性，即不生成对应的字段（在这里没有标注，只是讲一些这个注解的作用）。

1. 创建StudentDao接口，如下：



StudentDao接口继承了JpaRepository类，JpaRepository类中提供了一些基本的数据操作方法，有基本的增删查改、分页查询、排序查询等；

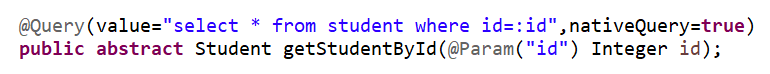
getStudentsByNameStartingWith方法表示查询以某个字符开始的所有学生；

getStudentsByAgeGreaterThan方法表示查询查询年龄大于某个值的所有学生；

1. 在SpringData JPA中，只要方法的定义符合既定规范，Spring Data就能分析出开发者的意图，从而避免开发者定义SQL。而所谓的既定规范，就是方法的命名规范。支持的命名规范如下：

|  |  |  |
| --- | --- | --- |
| KeyWorld | 方法命名举例 | 对应的SQL |
| And | findByNameAndAge | where name=? and age=? |
| Or | findByNameOrAge | where name=? or age=? |
| Is | findByAgeIs | where age=? |
| Equals | findByIdEquals | where id=? |
| Between | FindByAgeBetween | where age between? and ? |
| LessThan | findByAgeLessThan | where age<? |
| LessThanEquals | findByAgeLessThanEquals | where age<=? |
| GreaterThan | findByAgeGreaterThan | where where age>? |
| GreaterThanEquals | findByAgeGreaterThanEquals | where age>=? |
| After | findByAgeAfter | where age>? |
| Before | findByAgeBefore | where age<? |
| IsNull | findByNameIsNull | where name is null |
| IsNotNull,NotNull | findByNameNotNull | where name is not null |
| In | findByAgeIn | where age in(?) |
| NotIn | findByAgeNotIn | where age not in(?) |
| NotLike | findByNameNotLike | where name not like ? |
| Like | findByNameLike | where name like ? |
| StartingWith | findByNameStartingWith | where name like ‘?%’ |
| EncodingWith | findByNameEncodingWith | where name like ‘%?’ |
| Containing,Contains | findByNameContaining | where name like ‘%?%’ |
| OrderBy | findByAgeGreaterThanOrderByIdDesc | where age>? order by id desc |
| True | findByFlagTue | where flag=true |
| False | findByFlagFalse | where flag=false |

既定方法命名规范不一定满足所有的开发需求需求，因此Spring Data JPA也支持 自定义JPQL或者原生SQL，如下：

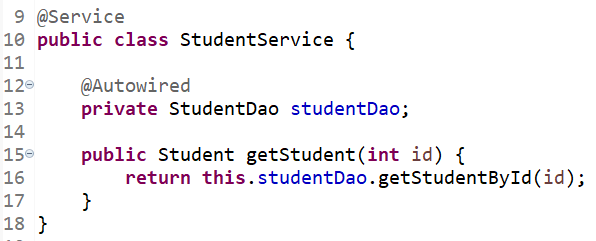


@Param将形参id的值绑定到SQL语句的:id上；

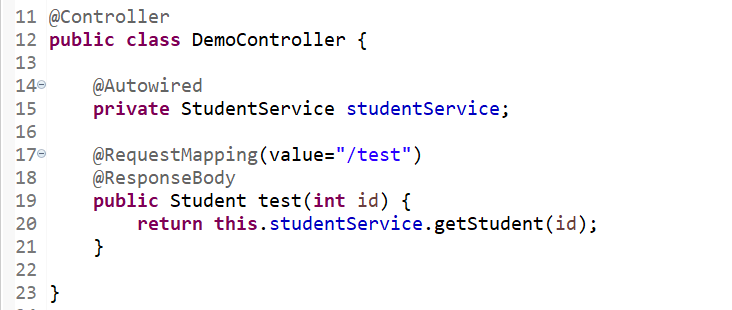
这里使用的是JPQL语句，JPQL是一种面向对象表达式语言，可以将SQL语法和简单查询语义绑定在一起，使用这种语言编写的查询是可移植的，可以被编译成所有主流数据库服务器上的SQL，JPQL与原生SQL语句类似，使用:属性名的形式进行参数绑定；

如果StudentDao接口中的方法涉及到修改操作，就要添加@Modifying注解并添加事务。

1. 创建StudentService，如下：

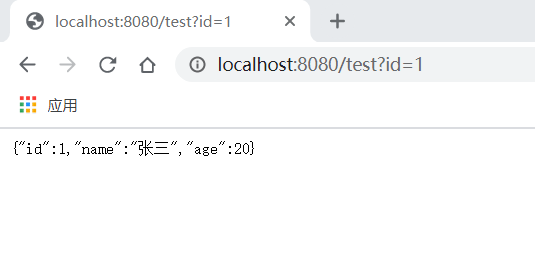


1. 创建控制器，如下：



为了演示方便，直接返回JSON数据，而没有用视图技术；

7）启动项目，打开浏览器访问，如下：



整合成功。当然，以上只是JPA的基本用法，更多详细用法请查阅相关资料。

# 多数据源

多数据源指的是一个项目使用了多个数据库、或者是一个项目使用了多种数据库技术。一般来说，采用MyCat等分布式数据库中间件是比较好的解决方案，这样可以把数据库读写分离、分库分表、备份等操作交给中间件去做，Java代码只需要专注于业务即可。但这并不意味着无法使用Java代码解决类似的问题。

在Spring、Mybatis中都可以配置多数据源，Spring Boot继承其衣钵，只不过配置方式有所变化。

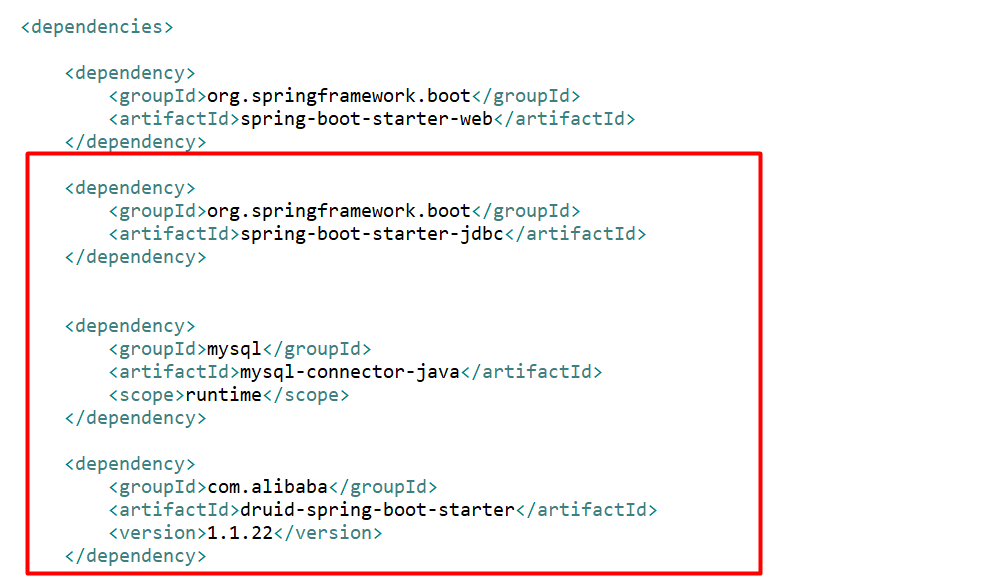
## JdbcTemplate多数据源

### 配置

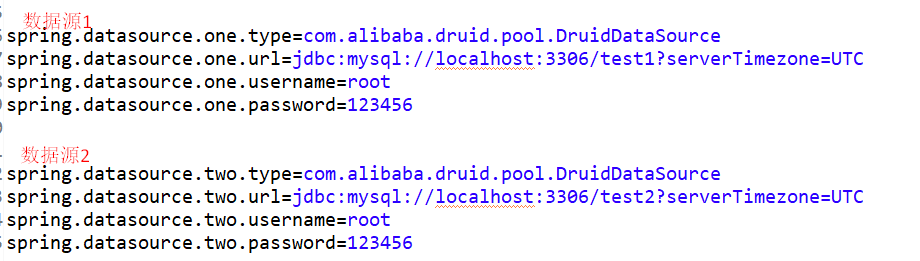
配置步骤大致分为四步：

添加依赖 --> 配置application.properties —> 配置DataSource --> 配置JdbcTemplate

1. 添加依赖，如下：

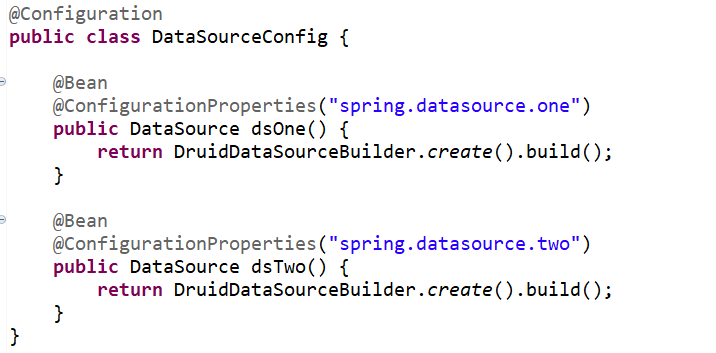


1. 在application.properties配置数据库连接信息，如下：



在这里配置了两个数据源，都是本机上的MySQL数据库，区别主要是数据库不同，一个是test1,数据库，一个是test2数据库；

1. 创建数据源配置类，如下：

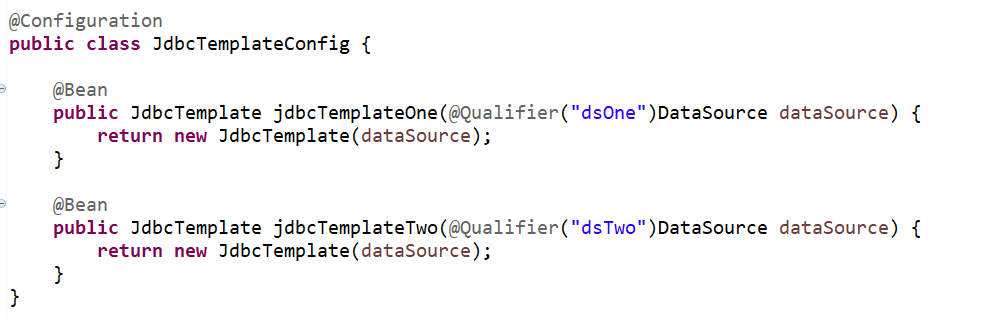


在这里配置了两个数据源：dsOne和dsTwo，默认方法名即实例名；

通过@ConfigurationProperties将application.properties配置文件中配置的多个数据源绑定为实例；

1. 如果只使用了spring-jdbc依赖，那么开发者没有提供JdbcTemplate实例时，Spring Boot默认会提供一个JdbcTemplate实例。

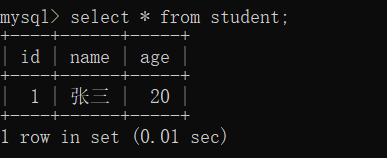
如果配置多数据源，需要由开发者提供JdbcTemplate实例，通过自定义类完成，如下：

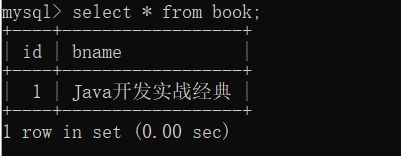


@Qualifier用于将不同名称的DataSource实例注入到JdbcTemplate中，得到JdbcTemplate。

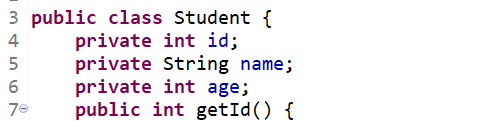
### 示例

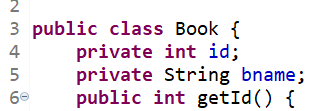
1. 在test1数据库中存在表student、在test2数据库中存在表book，数据如下：



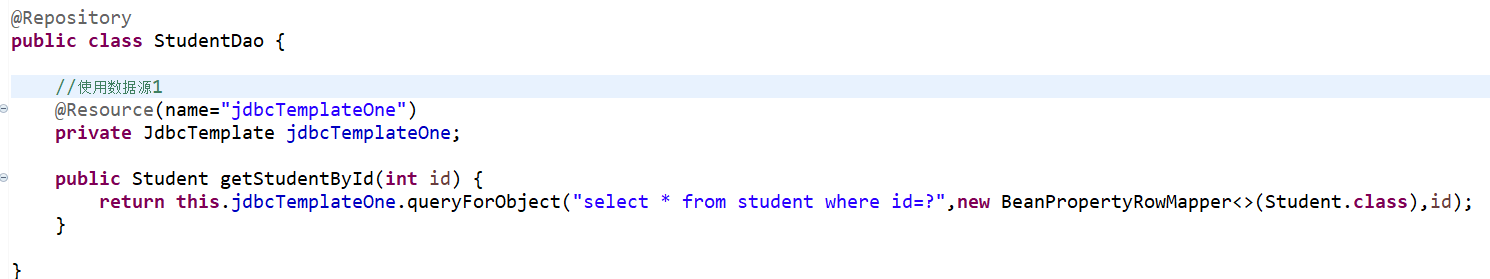


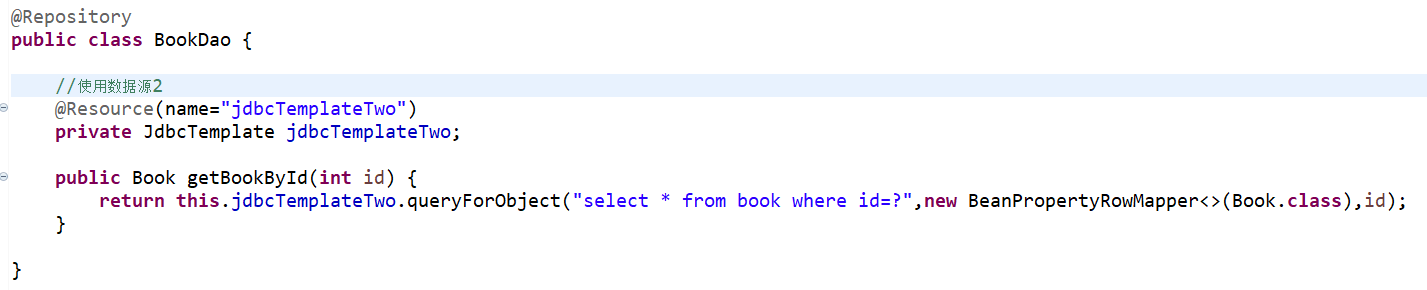
1. 分别为student表和book表创建实体类，如下：





1. StudentDao类和BookDao类，如下：





StudentDao使用数据源1，BookDao使用数据源2；

接着就可以创建业务层，调用持久层了。

## Mybatis多数据源

### 配置

其配置步骤大致如下：

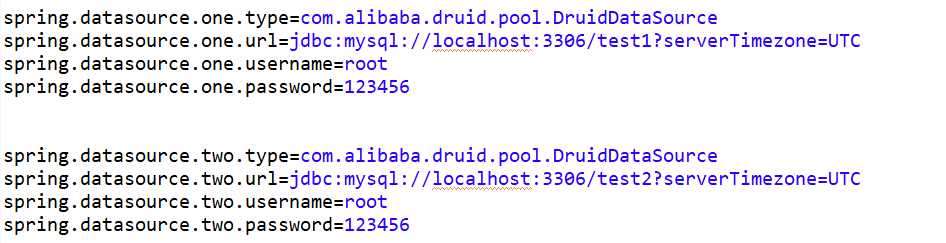
添加依赖 —> 在application.properties中配置多数据源 --> 创建数据源配置类

🡪 为每个数据源配置MyBatis设置 --> 创建mybatis-config.xml配置类并在application.properties中指明其位置

1. 添加依赖，如下：

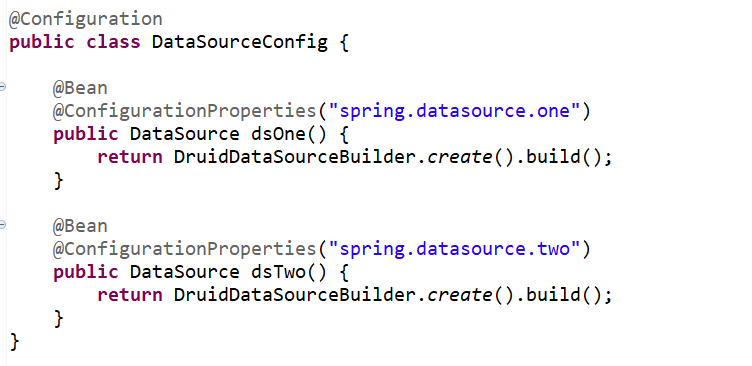


1. 在application.properties配置文件中配置数据库连接信息，如下：



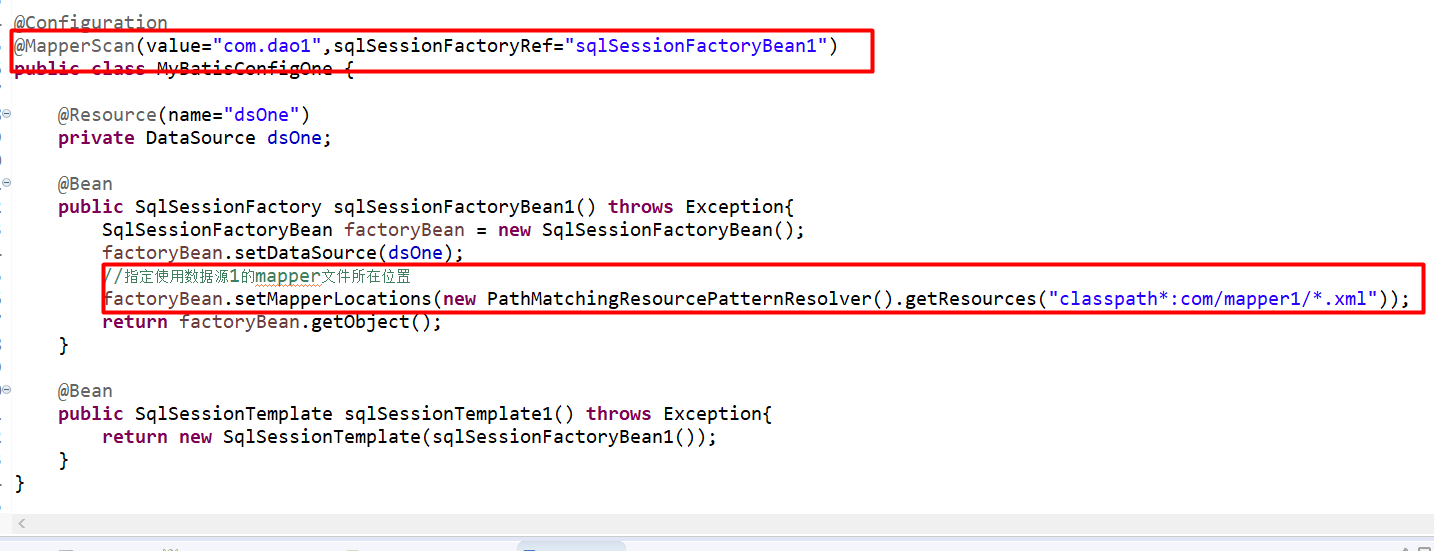
在这里配置了两个数据库连接信息，数据库地址都是本机上的MySQL，只不过数据库不同，一个是test1数据库，一个是test2数据库；

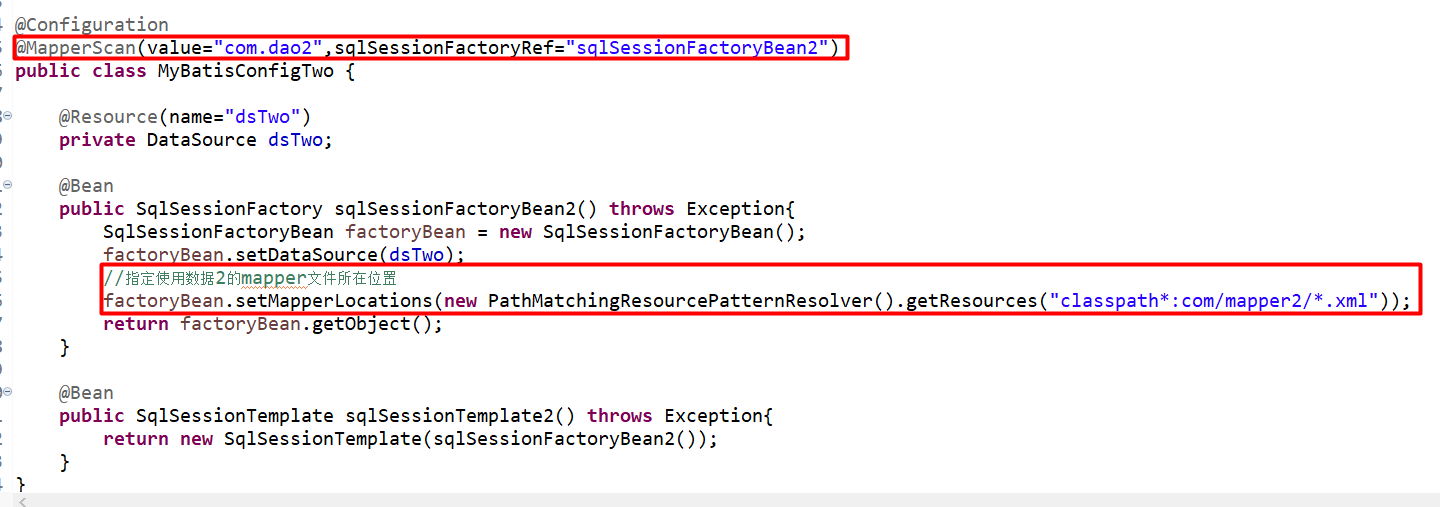
1. 创建数据源配置类，如下：



通过@ConfigurationProperties获取application.properties配置文件中配置的数据库信息，再通过@Configuration与@Bean搭配，将其实例注入到容器中，Bean的id即方法名；

1. 为每一个数据源创建一个MyBatis配置类，如下：





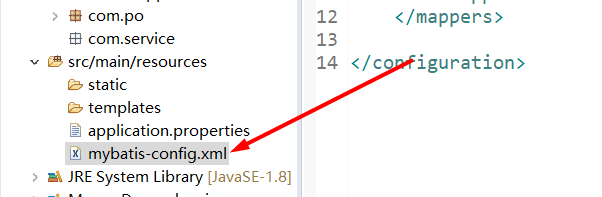
@MapperScan注解的value参数指定映射器接口所在的位置（包），同时指定SqlSessionFactory的实例名，则该位置下的Mapper将使用SqlSessionFactory实例；

SqlSession实例也就是@MapperScan注解的sqlSessionFactoryRef参数指定的实例；

SqlSessionTemplate实例是一个线程安全类，主要用来管理Mybatis中的SqlSession操作；

在指明映射器接口所在包的同时，还需要指明该接口所对应mapper文件所在的位置；

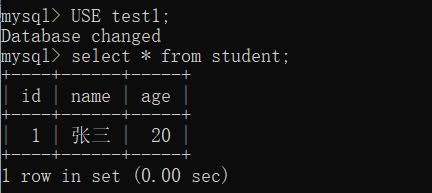
1. 在classpath路径下创建Mybatis配置文件，并在application.properties中指明其位置，如下：

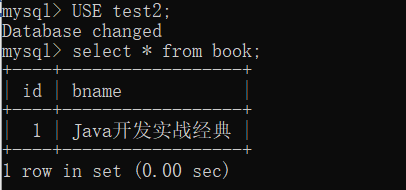




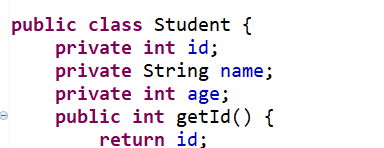
### 示例

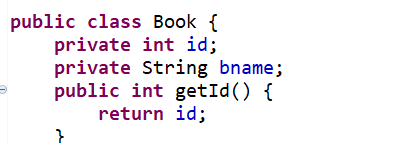
1. test1库中存在student表，test2库中存在book表，表中数据如下：



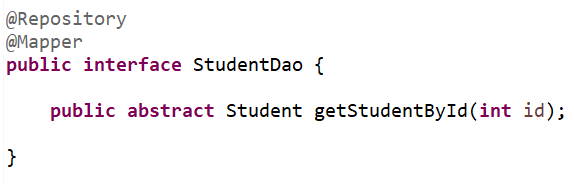


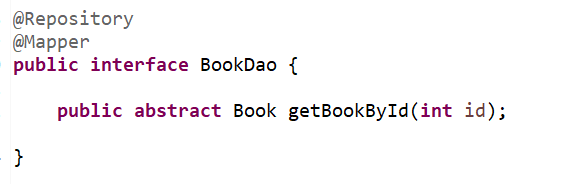
1. 为student表、book表创建实体类，如下：





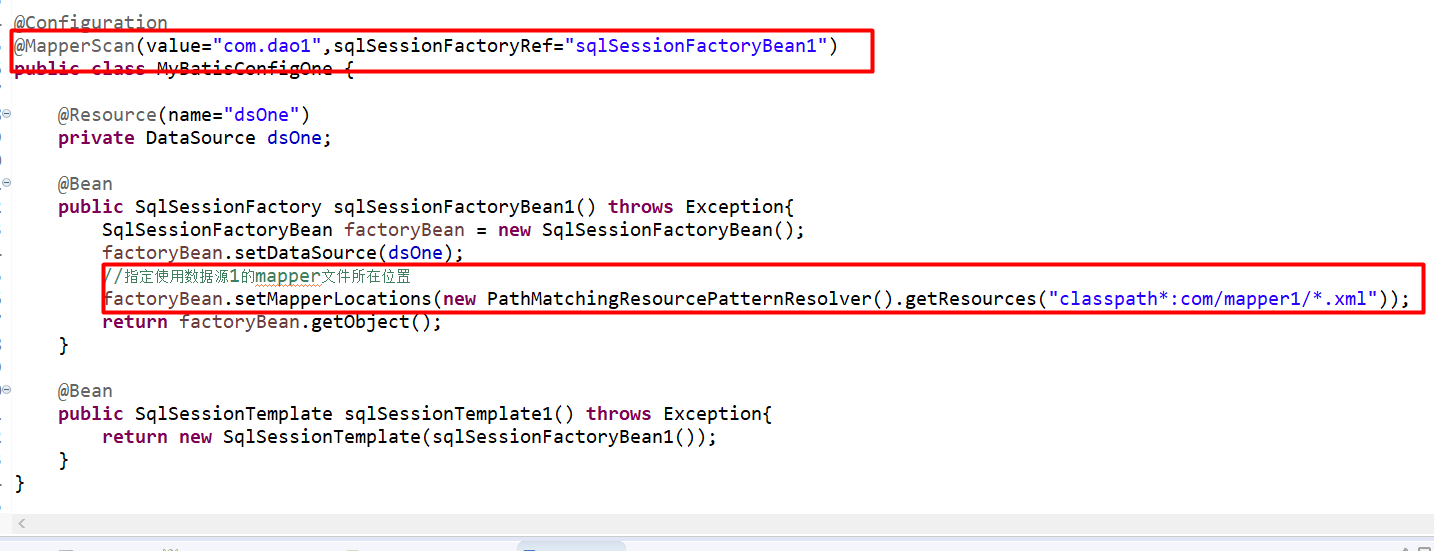
1. 在com.dao1包中创建StudentDao接口、在com.dao2包中创建BookDao接口，如下：

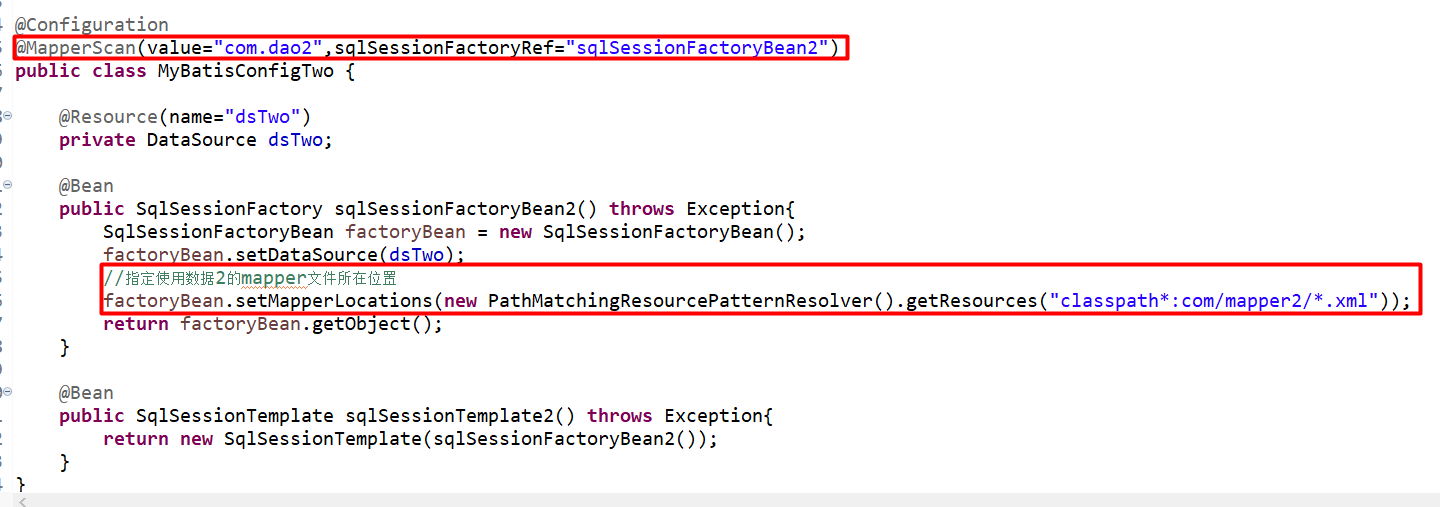




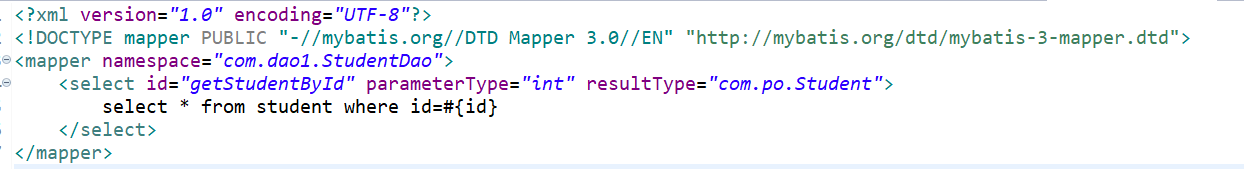
如果觉得为每个映射器接口都加上一个@Mapper注解比较麻烦，则可以在项目启动类上标注一个@MapperScan(“com.dao”) 注解

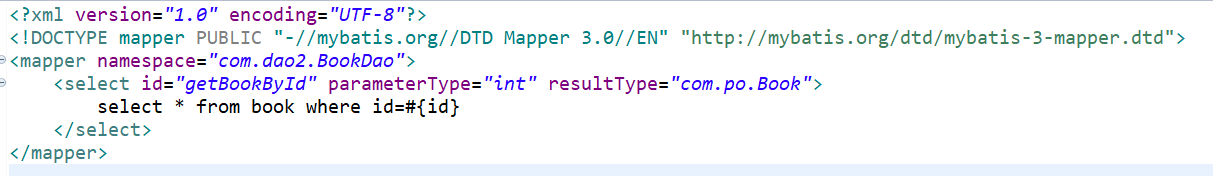
映射器接口所在包的位置必须与数据源Mybatis配置类中定义的一致，表示位于该包中的映射器接口使用哪个数据源，如下：





1. 在com.mapper1包中创建StudentMapper.xml文件，在com.mapper2包中创建BookMapper.xml文件，如下：





接着，业务层就可以调用持久层了；

## JPA多数据源

关于JPA多数据源的配置，由于技术具有实时性，按照教材书上的步骤并不能配置JPA多数据源，所以此处后期有待完善。

### 配置

### 示例

# 小结

JdbcTemplate应用并不广泛，MyBatis灵活性较好；Spring Data JPA能够快速实现一个RESTful风格的应用（将在第7章介绍）。