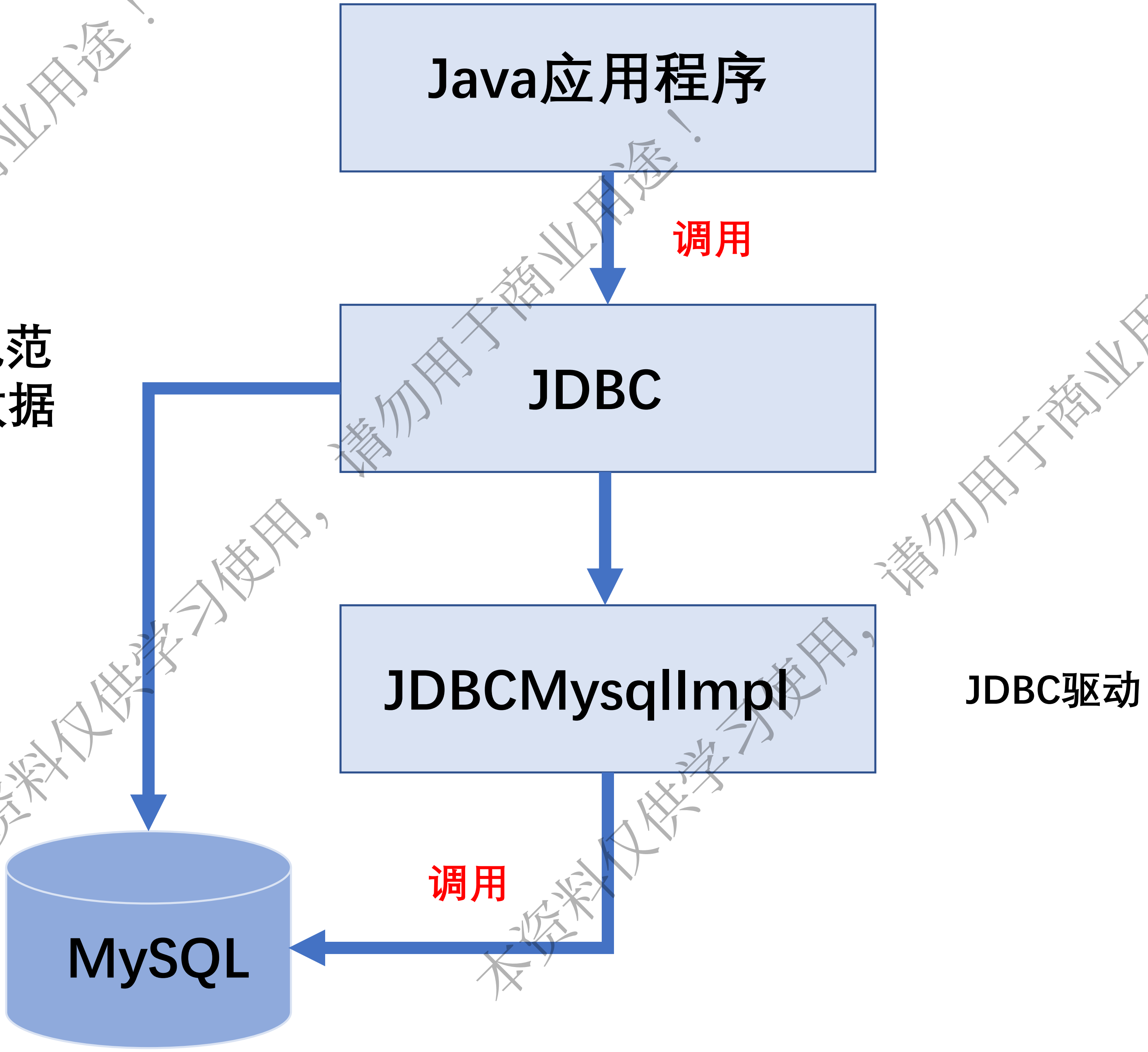
# SpringBoot与数据访问

- 1.JDBC

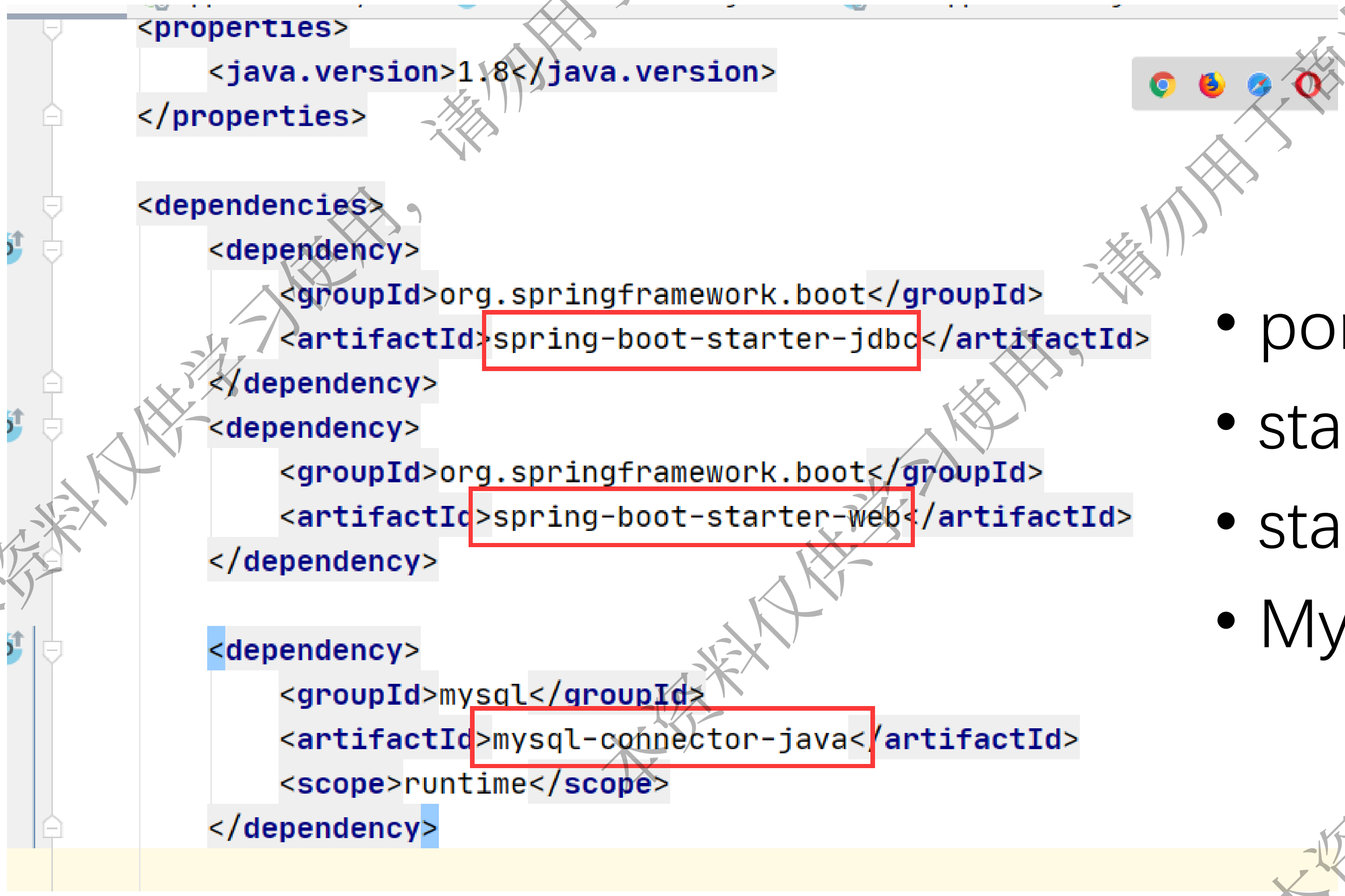
- 2.整合MyBatis
- ① 注解版
- ② 配置文件版

- 3.整合JPA

**Java数据库连接：用来规范客户端程序如何来访问数据库的应用程序接口**

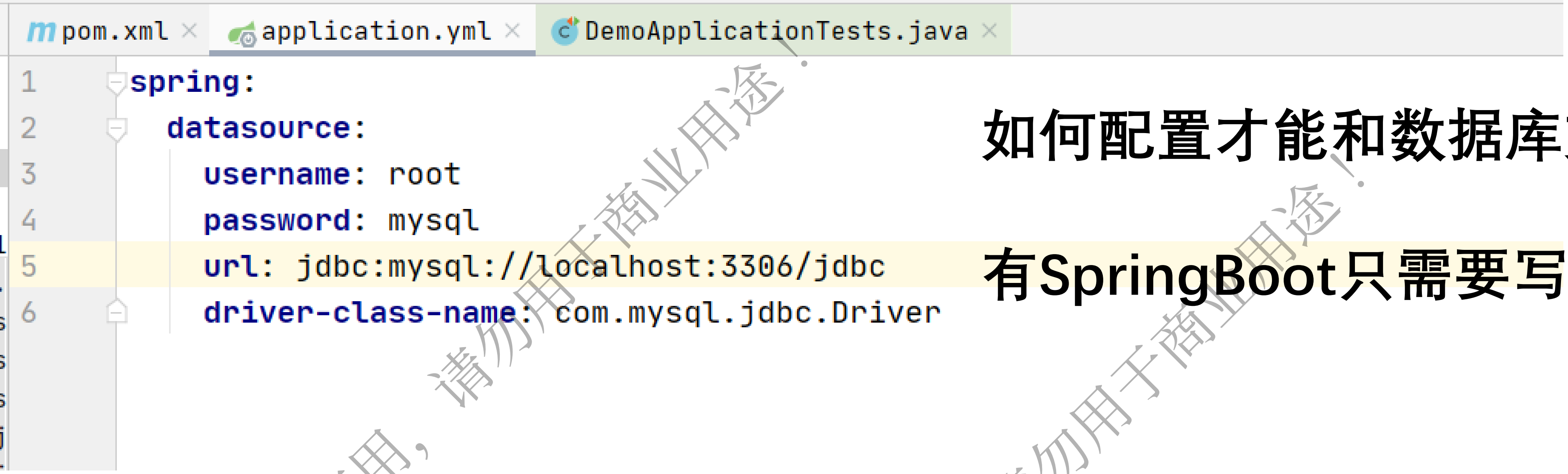


# 1.JDBC



```xml
<properties>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
```

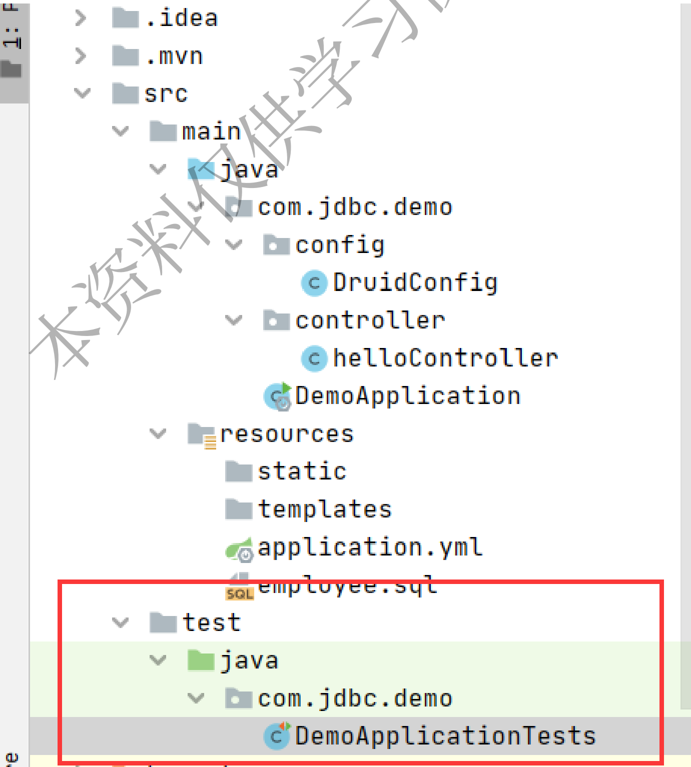
- pom文件中引入的是:
- starter-jdbc
- starter-web
- MySQL驱动

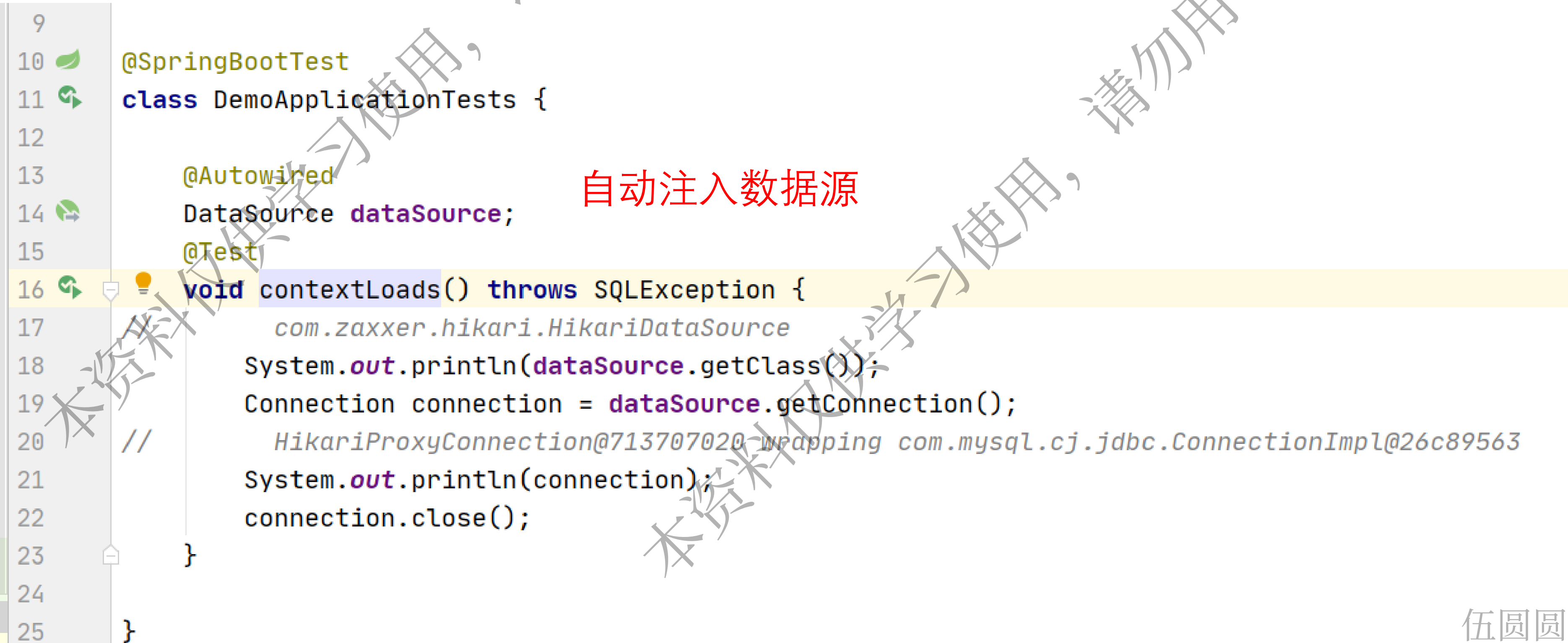


```yaml
spring:
  datasource:
    username: root
    password: mysql
    url: jdbc:mysql://localhost:3306/jdbc
    driver-class-name: com.mysql.jdbc.Driver
```

## 如何配置才能和数据库交互？

## 有SpringBoot只需要写相关的配置项

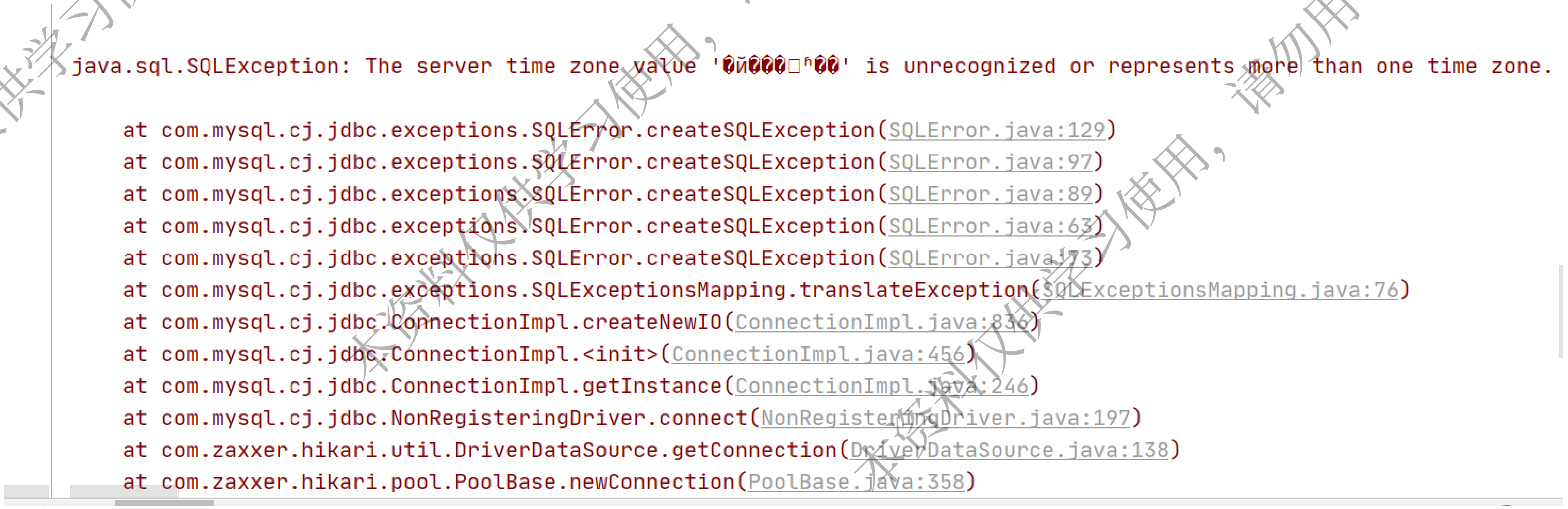




```java
@SpringBootTest
class DemoApplicationTests {

    @Autowired
    DataSource dataSource;
    @Test
    void contextLoads() throws SQLException {
//        com.zaxxer.hikari.HikariDataSource
        System.out.println(dataSource.getClass());
        Connection connection = dataSource.getConnection();
//        HikariProxyConnection@713707020 wrapping com.mysql.cj.jdbc.ConnectionImpl@26c89563
        System.out.println(connection);
        connection.close();
    }

}
```

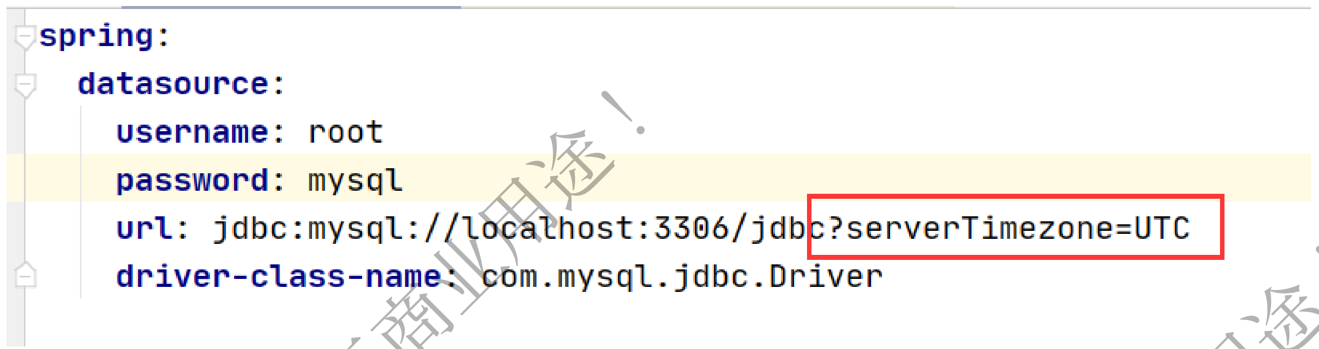
自动注入数据源

java.sql.SQLException: The **server time zone** value '�й���□ᶠ��' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the 'serverTimezone' configuration property) to use a more specifc time zone value if you want to utilize time zone support.



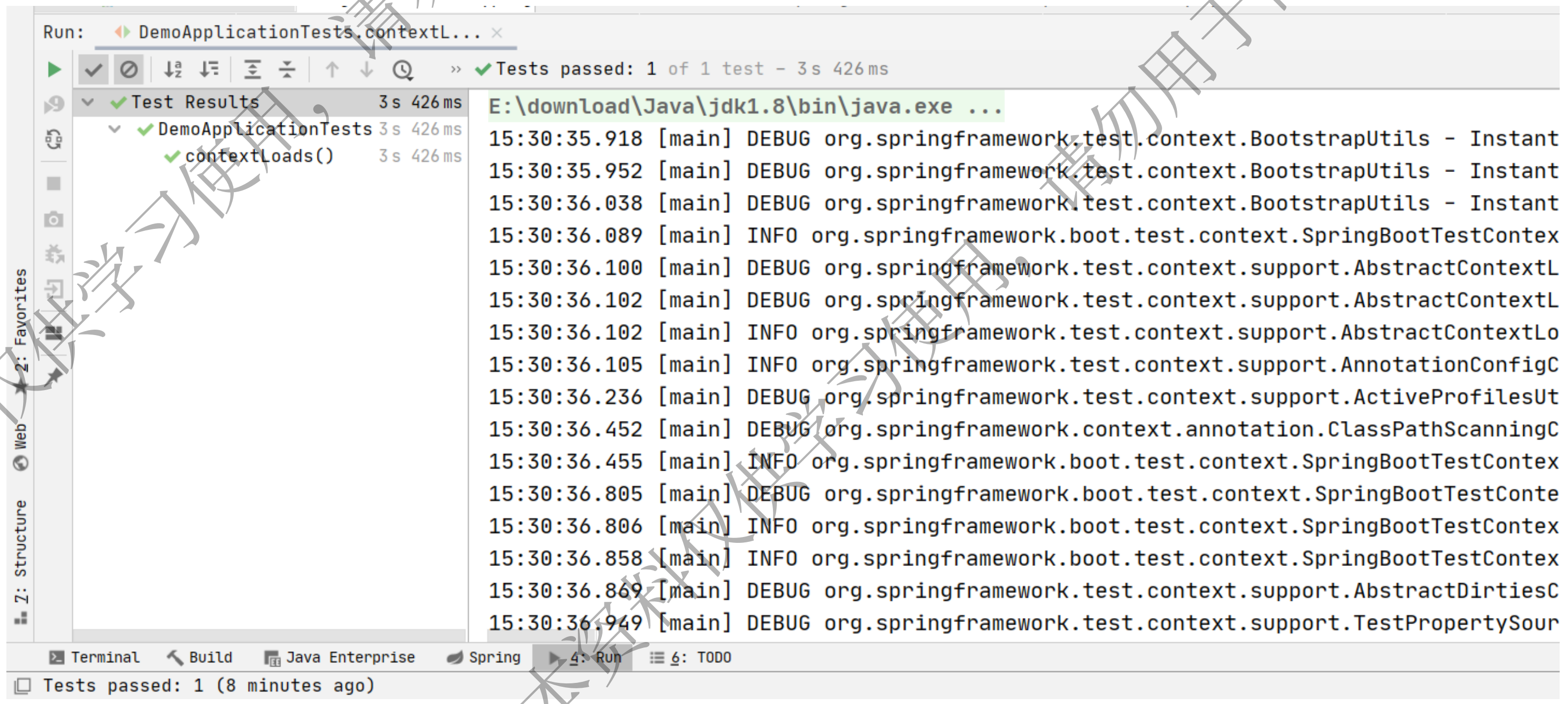
```
java.sql.SQLException: The server time zone value '�й���□ʰ��' is unrecognized or represents more than one time zone.

    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:129)
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:97)
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:89)
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:63)
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:73)
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:76)
    at com.mysql.cj.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:836)
    at com.mysql.cj.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:456)
    at com.mysql.cj.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:246)
    at com.mysql.cj.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:197)
    at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:138)
    at com.zaxxer.hikari.pool.PoolBase.newConnection(PoolBase.java:358)
```

- 原因:
- MySQL服务器时区（继承自系统时区）的格式与MySQL连接器所期望的格式不同



```yaml
spring:
  datasource:
    username: root
    password: mysql
    url: jdbc:mysql://localhost:3306/jdbc?serverTimezone=UTC
    driver-class-name: com.mysql.jdbc.Driver
```

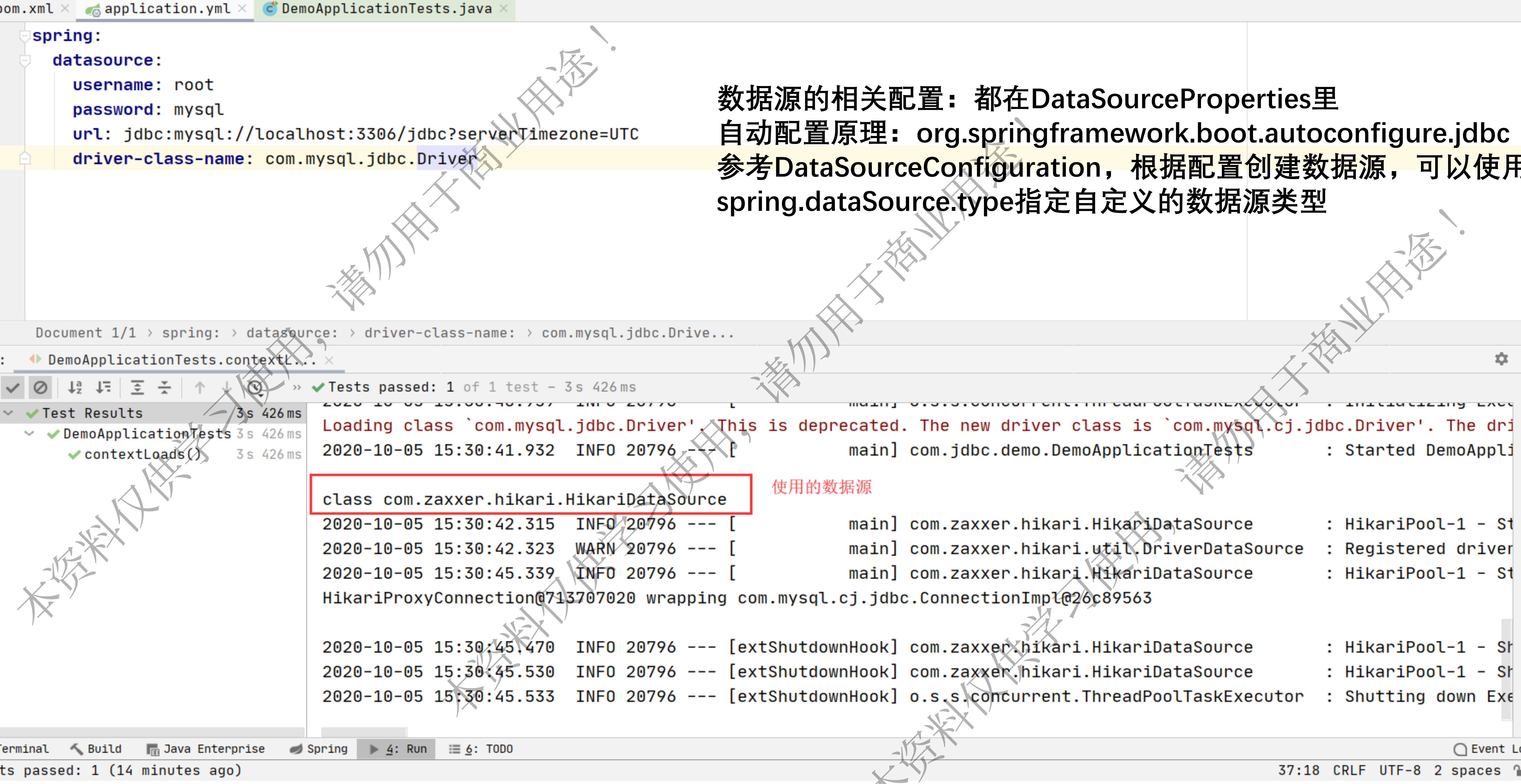
- 解决办法:
- 在JDBC连接url后添加时区属性: serverTimezone=UTC
- UTC是统一标准世界时间



Run: DemoApplicationTests.contextL...
Tests passed: 1 of 1 test – 3 s 426 ms
Test Results 3 s 426 ms
DemoApplicationTests 3 s 426 ms
contextLoads() 3 s 426 ms
Terminal
Build
Java Enterprise
Spring
4: Run
6: TODO
2: Favorites
Web
Z: Structure

```
E:\download\Java\jdk1.8\bin\java.exe ...
15:30:35.918 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instant
15:30:35.952 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instant
15:30:36.038 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instant
15:30:36.089 [main] INFO org.springframework.boot.test.context.SpringBootTestContex
15:30:36.100 [main] DEBUG org.springframework.test.context.support.AbstractContextL
15:30:36.102 [main] DEBUG org.springframework.test.context.support.AbstractContextL
15:30:36.102 [main] INFO org.springframework.test.context.support.AbstractContextLo
15:30:36.105 [main] INFO org.springframework.test.context.support.AnnotationConfigC
15:30:36.236 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUt
15:30:36.452 [main] DEBUG org.springframework.context.annotation.ClassPathScanningC
15:30:36.455 [main] INFO org.springframework.boot.test.context.SpringBootTestContex
15:30:36.805 [main] DEBUG org.springframework.boot.test.context.SpringBootTestConte
15:30:36.806 [main] INFO org.springframework.boot.test.context.SpringBootTestContex
15:30:36.858 [main] INFO org.springframework.boot.test.context.SpringBootTestContex
15:30:36.869 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesC
15:30:36.949 [main] DEBUG org.springframework.test.context.support.TestPropertySour
```

Tests passed: 1 (8 minutes ago)



om.xml
application.yml
DemoApplicationTests.java

```yaml
spring:
  datasource:
    username: root
    password: mysql
    url: jdbc:mysql://localhost:3306/jdbc?serverTimezone=UTC
    driver-class-name: com.mysql.jdbc.Driver
```

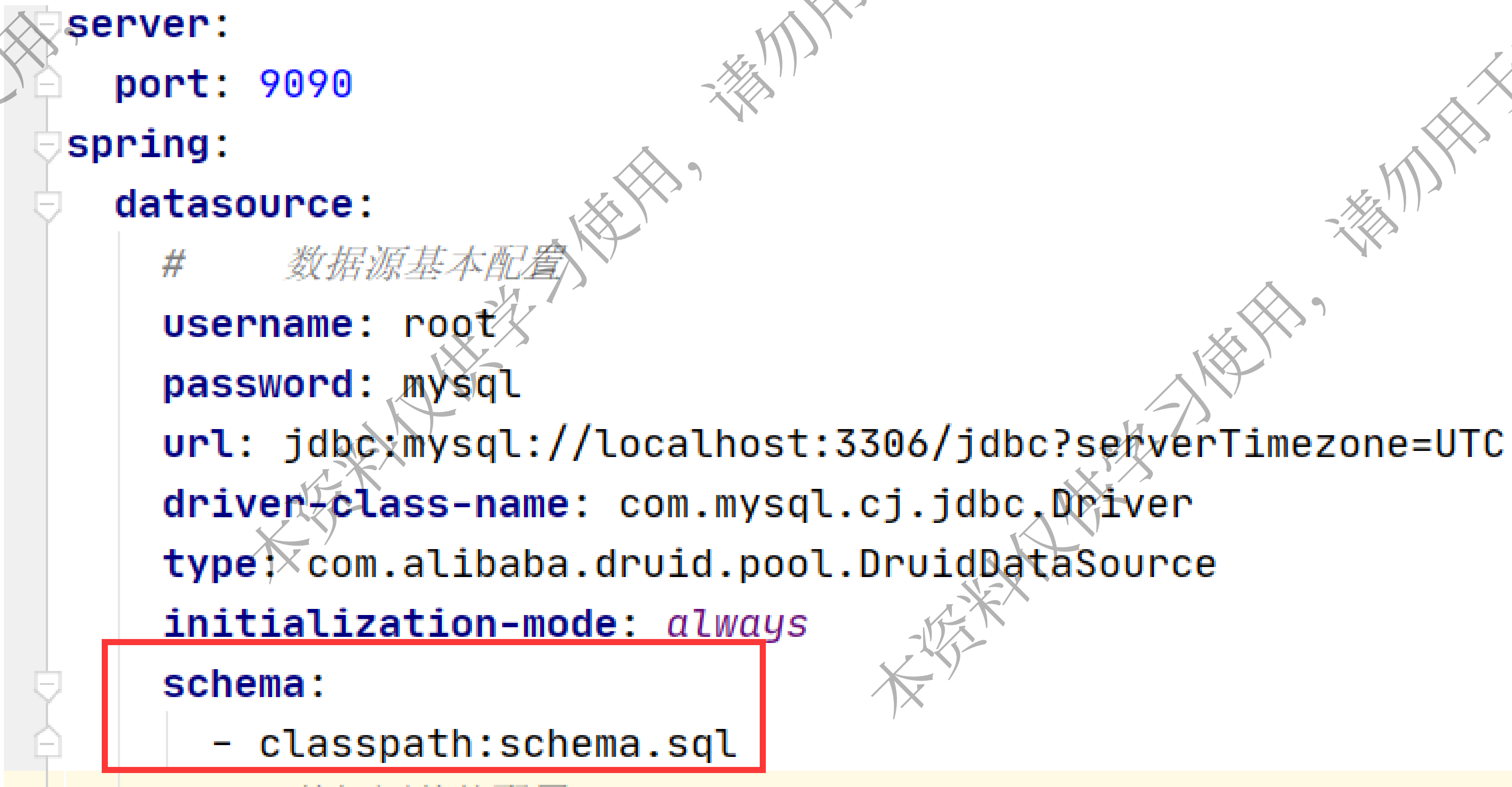
数据源的相关配置：都在DataSourceProperties里
自动配置原理：org.springframework.boot.autoconfigure.jdbc
参考DataSourceConfiguration，根据配置创建数据源，可以使用
spring.dataSource.type指定自定义的数据源类型

Document 1/1 › spring: › datasource: › driver-class-name: › com.mysql.jdbc.Drive...
DemoApplicationTests.contextL...
Tests passed: 1 of 1 test – 3 s 426 ms
Test Results 3 s 426 ms
DemoApplicationTests 3 s 426 ms
contextLoads() 3 s 426 ms

```
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The dri
2020-10-05 15:30:41.932  INFO 20796 --- [           main] com.jdbc.demo.DemoApplicationTests       : Started DemoAppli

class com.zaxxer.hikari.HikariDataSource
2020-10-05 15:30:42.315  INFO 20796 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - St
2020-10-05 15:30:42.323  WARN 20796 --- [           main] com.zaxxer.hikari.util.DriverDataSource  : Registered driver
2020-10-05 15:30:45.339  INFO 20796 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - St
HikariProxyConnection@713707020 wrapping com.mysql.cj.jdbc.ConnectionImpl@26c89563

2020-10-05 15:30:45.470  INFO 20796 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Sh
2020-10-05 15:30:45.530  INFO 20796 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Sh
2020-10-05 15:30:45.533  INFO 20796 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down Exe
```

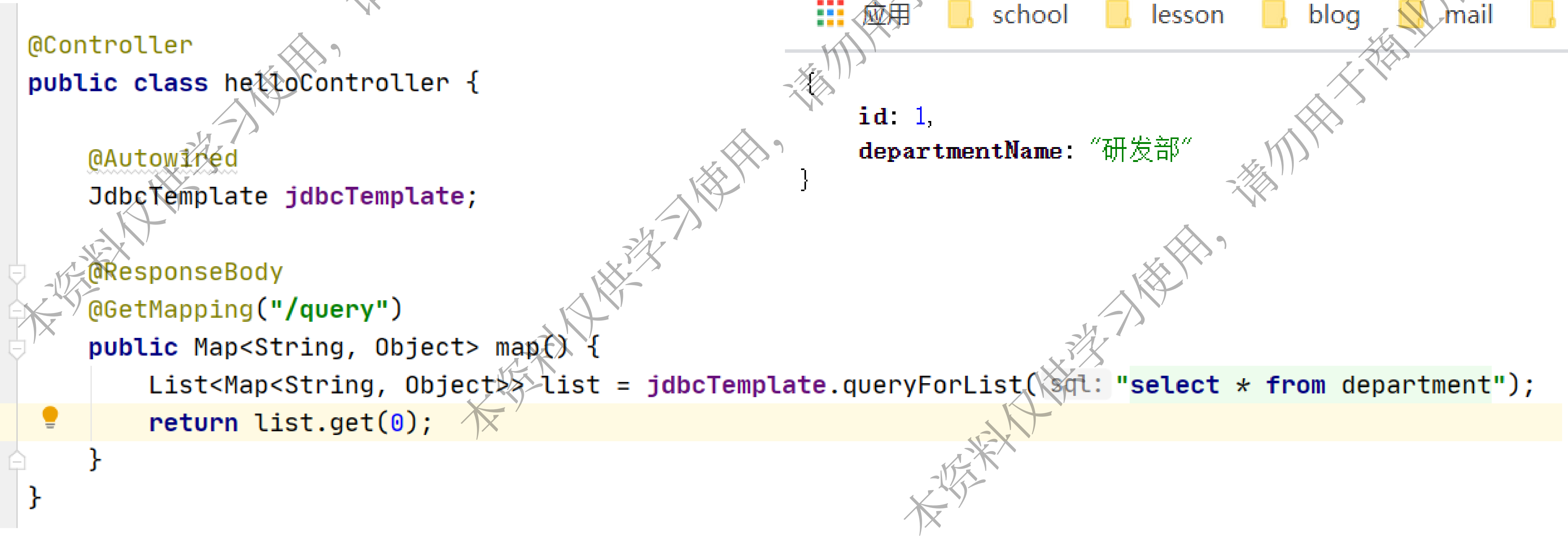
使用的数据源

- 操作数据库:
- DataSourceInitializer: ApplicationListener
- 作用:
- ① runSchemaScripts()运行建表语句
- ② runDataScripts()运行插入数据的SQL语句

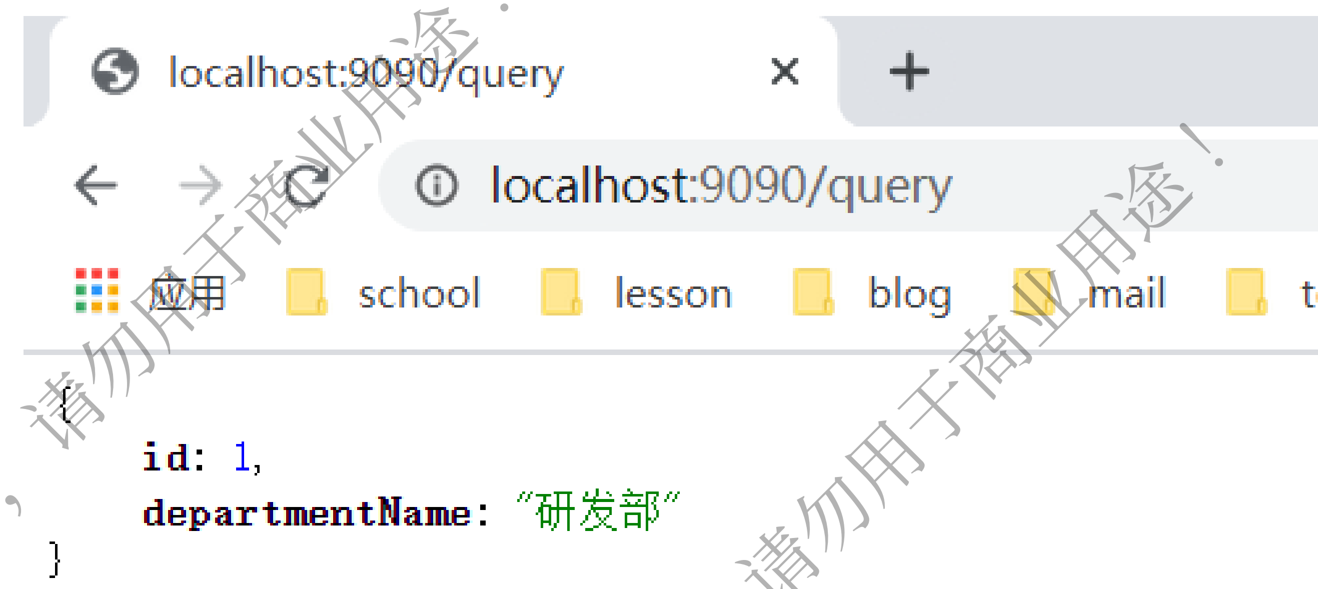


```yaml
server:
  port: 9090
spring:
  datasource:
    #      数据源基本配置
    username: root
    password: mysql
    url: jdbc:mysql://localhost:3306/jdbc?serverTimezone=UTC
    driver-class-name: com.mysql.cj.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource
    initialization-mode: always
    schema:
      - classpath:schema.sql
```

- **自动配置了JdbcTemplate操作数据库**



```
@Controller
public class helloController {

    @Autowired
    JdbcTemplate jdbcTemplate;

    @ResponseBody
    @GetMapping("/query")
    public Map<String, Object> map() {
        List<Map<String, Object>> list = jdbcTemplate.queryForList( sql: "select * from department");
        return list.get(0);
    }
}
```



```
{
    id: 1,
    departmentName: "研发部"
}
```

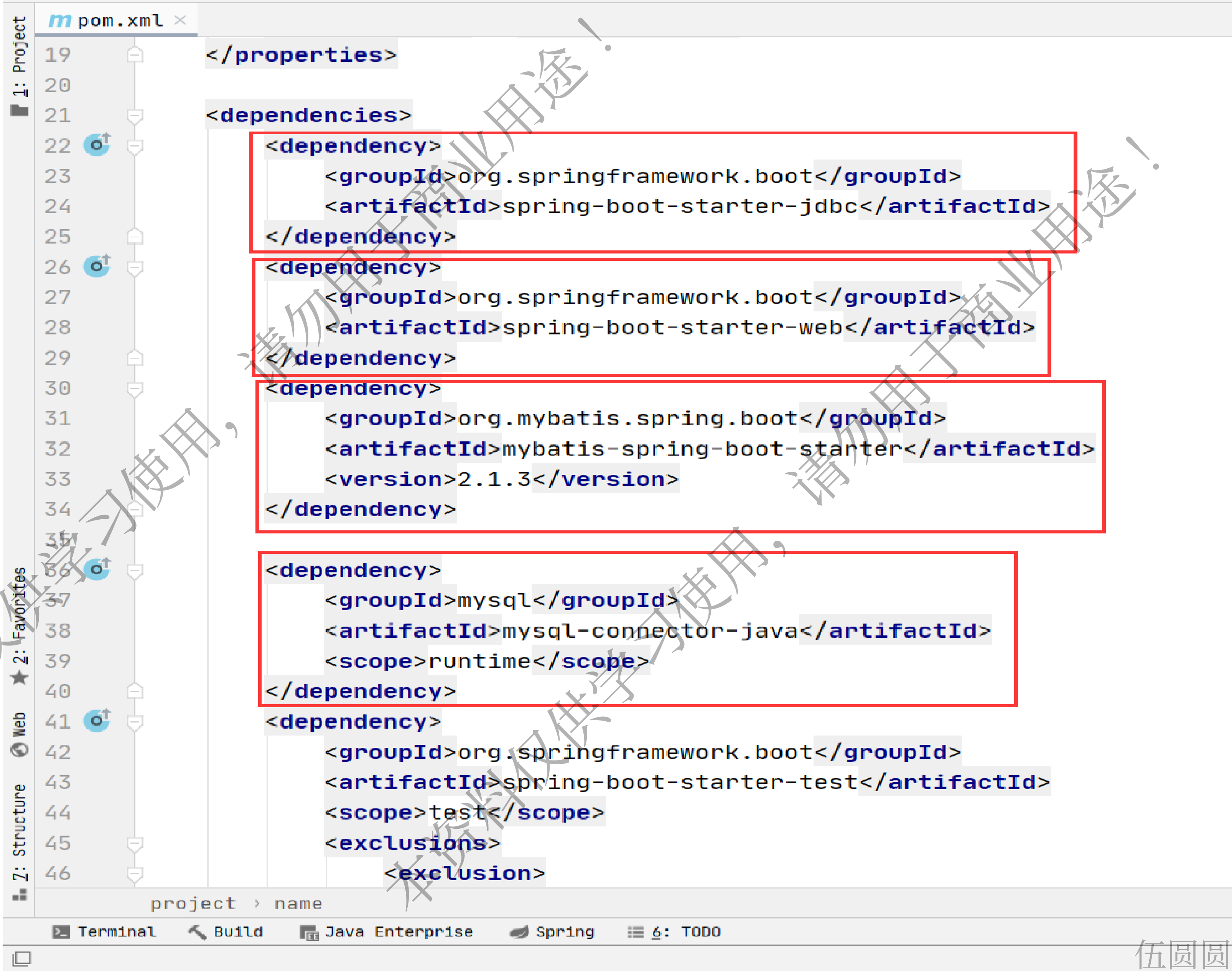
## 2.整合MyBatis

- MyBatis是一个持久层框架，对jdbc的操作数据库过程进行封装

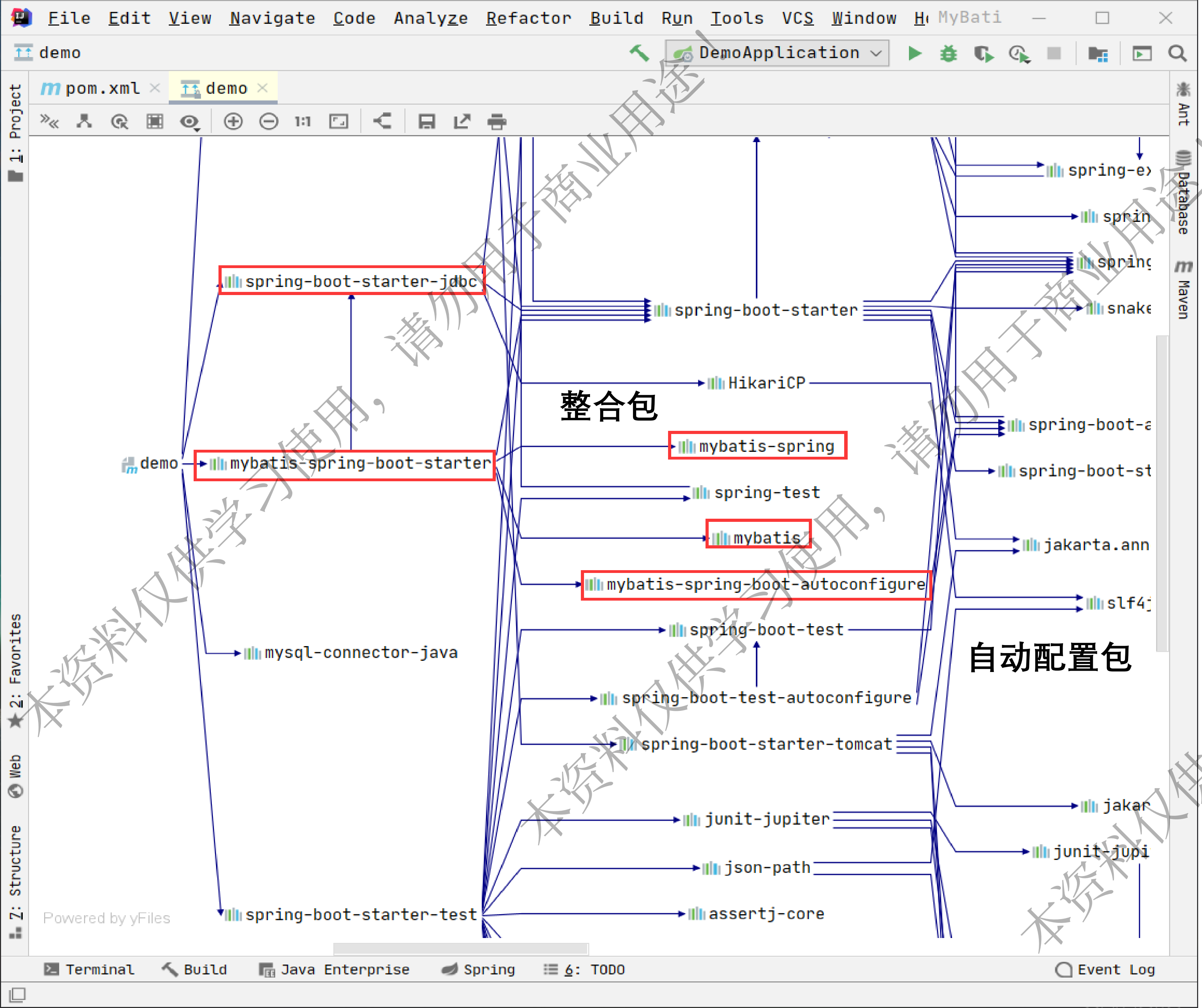
- 为什么不用jdbc?
- SQL语句硬编码到Java代码中不利于维护；
- 数据库频繁开启和关闭造成数据库的资源浪费

- MyBatis使用数据库连接池来管理数据库的连接，将SQL语句配置在XML文件中避免代码将查询的结果集自动映射为Java对象

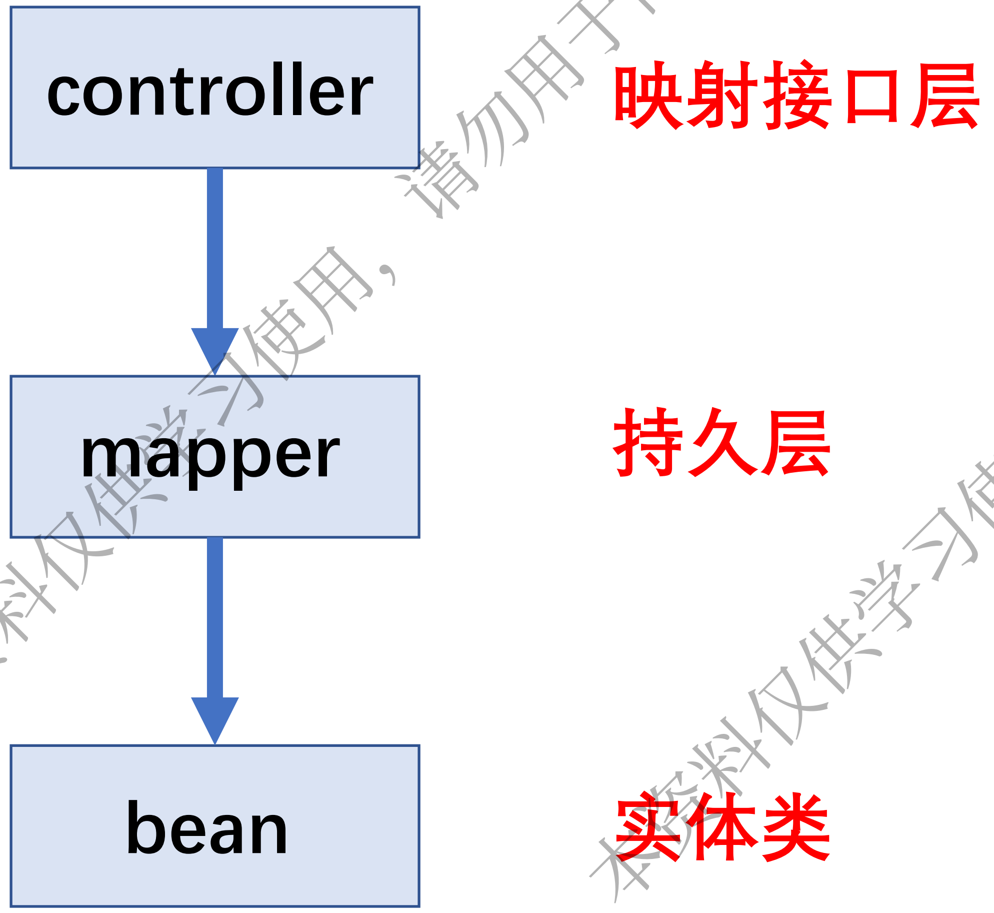
通过插件mybatis-spring-boot-starter
在SpringBoot中集成MyBatis，
不用关心原生配置方式的细节，
直接使用默认配置就能实现最基本的功能

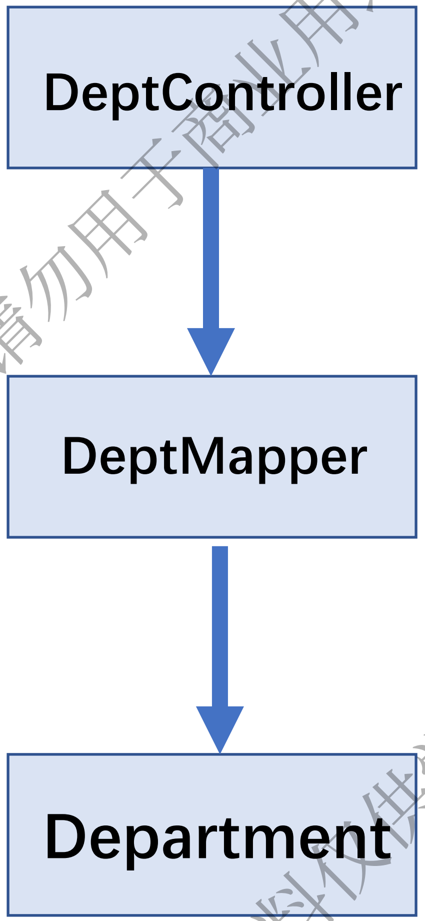


```
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.1.3</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window H MyBati
demo
DemoApplication
pom.xml
demo
1: Project
2: Favorites
Web
Z: Structure
Ant
Database
Maven
demo
spring-boot-starter-jdbc
mybatis-spring-boot-starter
mysql-connector-java
spring-boot-starter-test
spring-boot-starter
HikariCP
整合包
mybatis-spring
spring-test
mybatis
mybatis-spring-boot-autoconfigure
spring-boot-test
spring-boot-test-autoconfigure
spring-boot-starter-tomcat
junit-jupiter
json-path
assertj-core
spring-ex
sprin
spring
snake
spring-boot-a
spring-boot-st
jakarta.ann
slf4j
自动配置包
jakar
junit-jupi
Powered by yFiles
Terminal
Build
Java Enterprise
Spring
6: TODO
Event Log

- 使用MyBatis建立三层结构

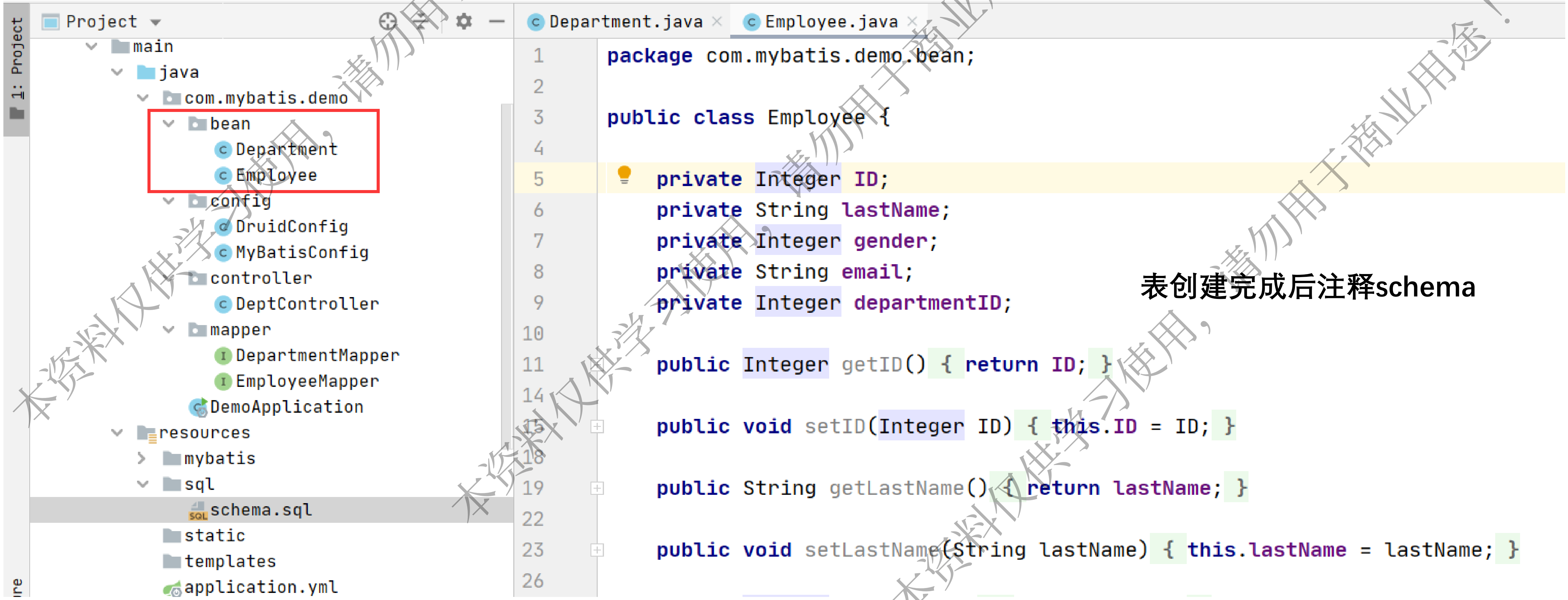




controller通过接收前端传过来的参数继续业务操作，返回一个指定的路径或数据表

将实体类映射到数据库中的表并进行操作

- 基础环境搭建
- 创建JavaBean封装表的数据



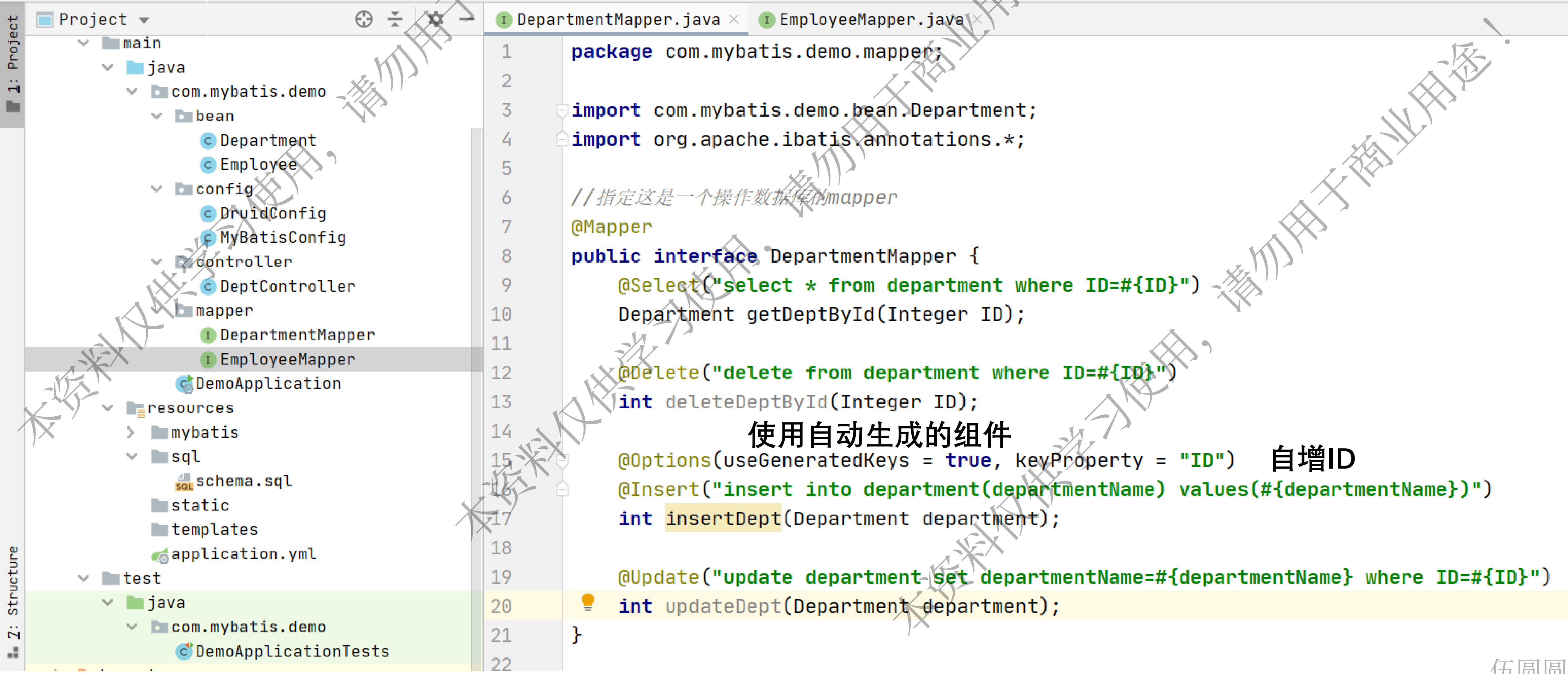
```java
package com.mybatis.demo.bean;

public class Employee {

    private Integer ID;
    private String lastName;
    private Integer gender;
    private String email;
    private Integer departmentID;

    public Integer getID() { return ID; }

    public void setID(Integer ID) { this.ID = ID; }

    public String getLastName() { return lastName; }

    public void setLastName(String lastName) { this.lastName = lastName; }
```

表创建完成后注释schema

- 基础注解：
- MyBatis主要提供以下CRUD注解：

- @Select
- @Insert
- @Update
- @Delete

- ① 注解版MyBatis（DepartmentMapper）
- 定义接口映射器

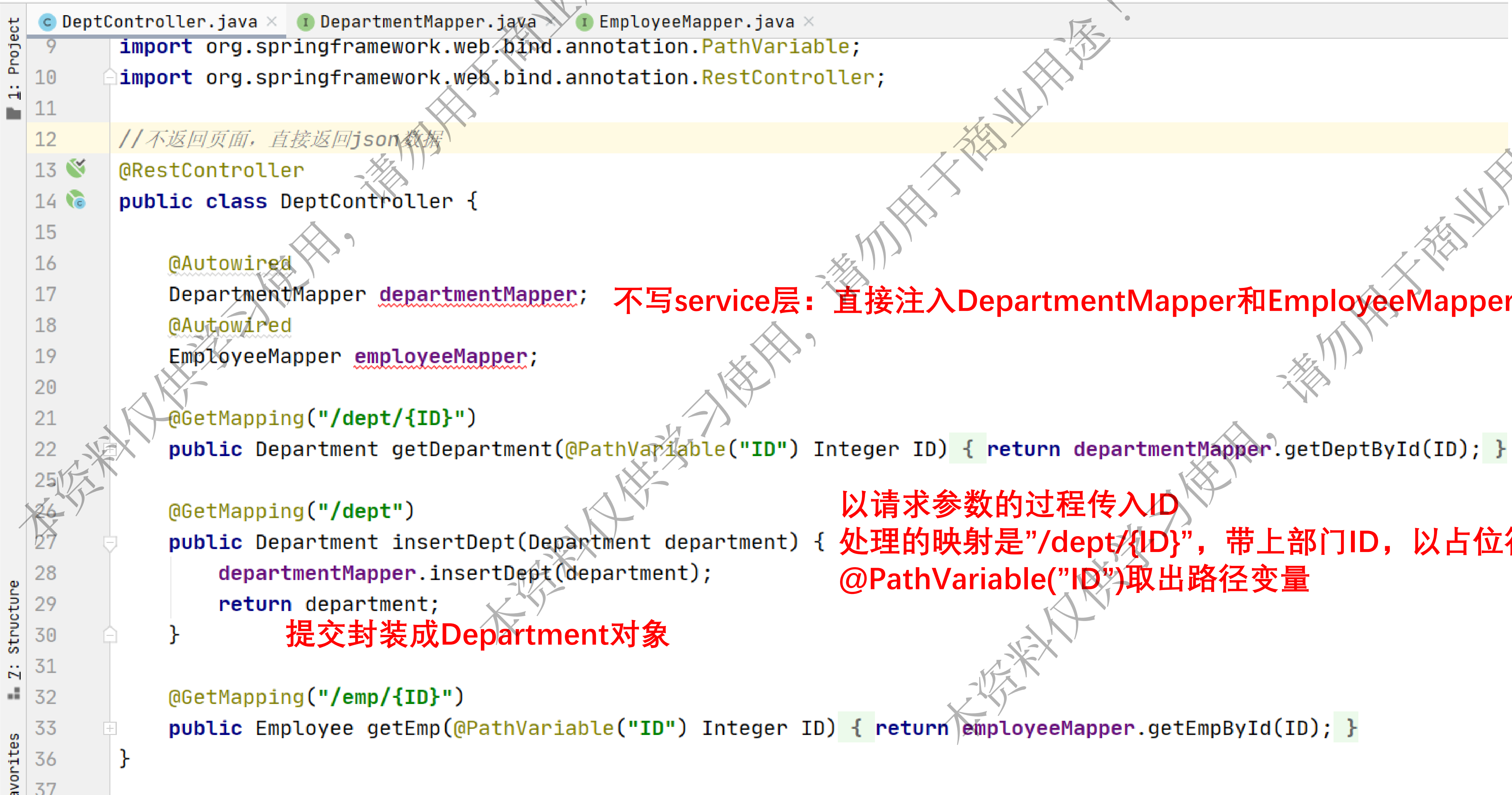


```java
package com.mybatis.demo.mapper;

import com.mybatis.demo.bean.Department;
import org.apache.ibatis.annotations.*;

//指定这是一个操作数据库的mapper
@Mapper
public interface DepartmentMapper {
    @Select("select * from department where ID=#{ID}")
    Department getDeptById(Integer ID);

    @Delete("delete from department where ID=#{ID}")
    int deleteDeptById(Integer ID);

    @Options(useGeneratedKeys = true, keyProperty = "ID")
    @Insert("insert into department(departmentName) values(#{departmentName})")
    int insertDept(Department department);

    @Update("update department set departmentName=#{departmentName} where ID=#{ID}")
    int updateDept(Department department);
}
```

使用自动生成的组件

自增ID

- 使用接口映射器



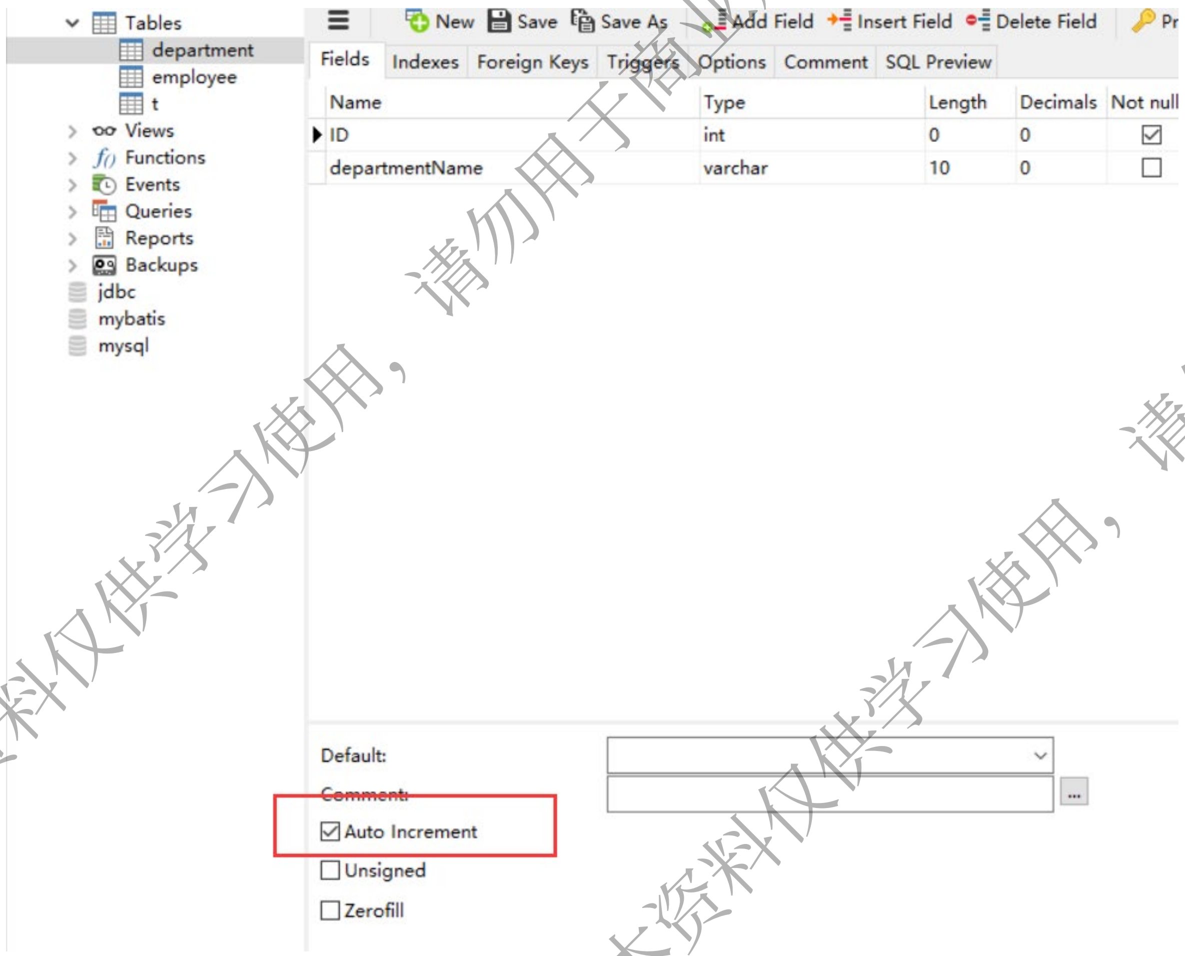
```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

//不返回页面，直接返回json数据
@RestController
public class DeptController {

    @Autowired
    DepartmentMapper departmentMapper;
    @Autowired
    EmployeeMapper employeeMapper;

    @GetMapping("/dept/{ID}")
    public Department getDepartment(@PathVariable("ID") Integer ID) { return departmentMapper.getDeptById(ID); }

    @GetMapping("/dept")
    public Department insertDept(Department department) {
        departmentMapper.insertDept(department);
        return department;
    }

    @GetMapping("/emp/{ID}")
    public Employee getEmp(@PathVariable("ID") Integer ID) { return employeeMapper.getEmpById(ID); }
}
```

不写service层：直接注入DepartmentMapper和EmployeeMapper

以请求参数的过程传入ID
处理的映射是"/dept/{ID}"，带上部门ID，以占位符的方式
@PathVariable("ID")取出路径变量

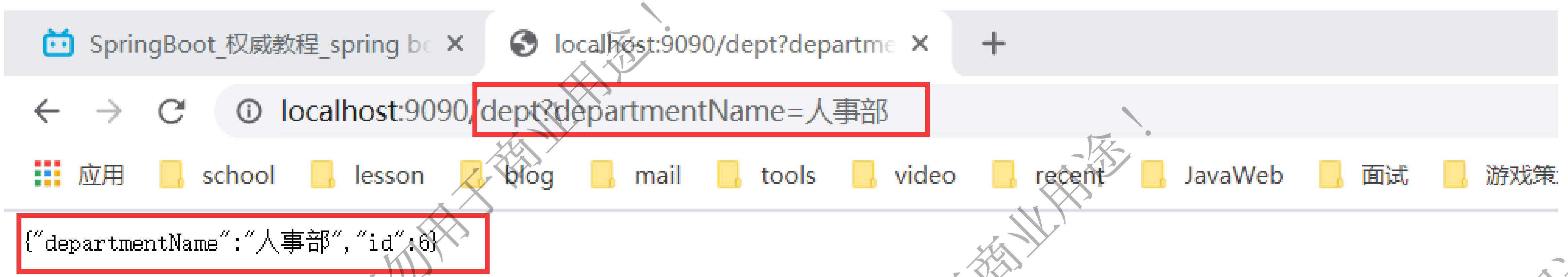
提交封装成Department对象

- **将主键ID设为int：Auto Increasement自增**

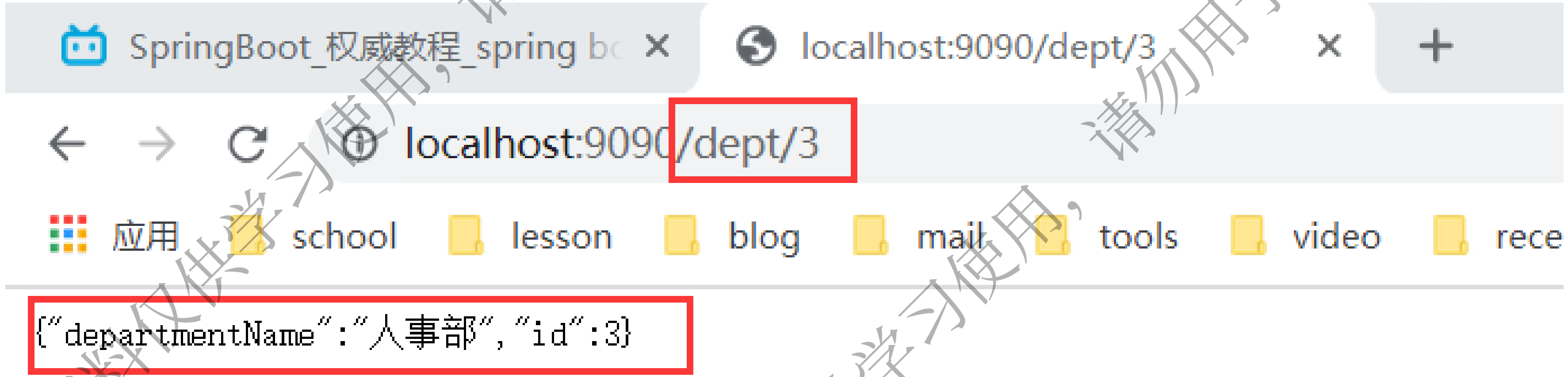


| Name | Type | Length | Decimals | Not null |
|---|---|---|---|---|
| ID | int | 0 | 0 | ☑ |
| departmentName | varchar | 10 | 0 | ☐ |

Default:
Comment:
☑ Auto Increment
☐ Unsigned
☐ Zerofill

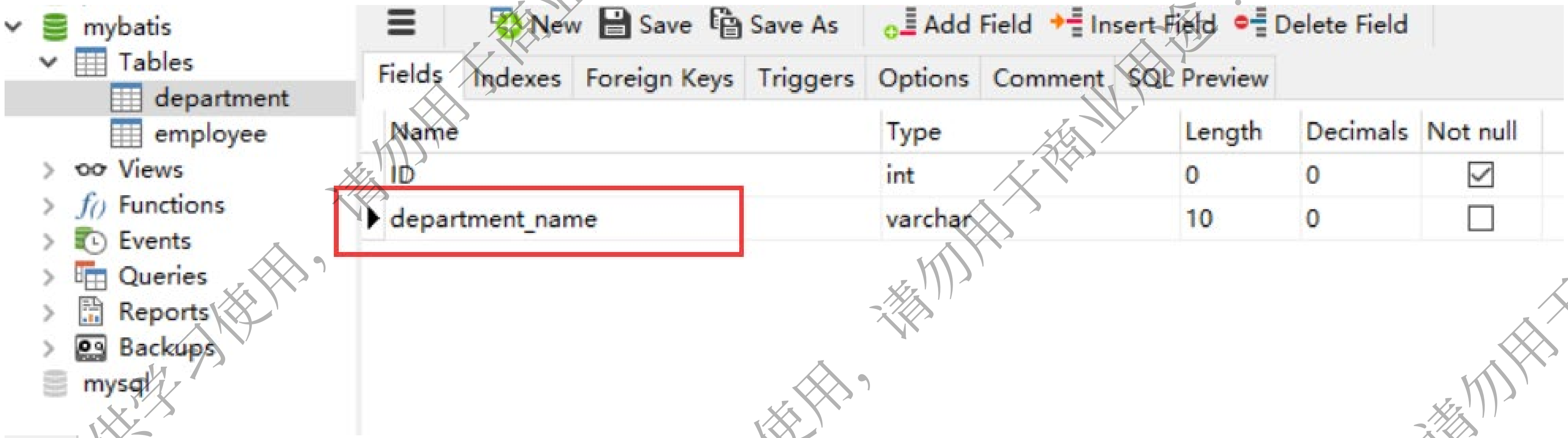

localhost:9090/dept?departmentName=人事部

```
{"departmentName":"人事部","id":6}
```

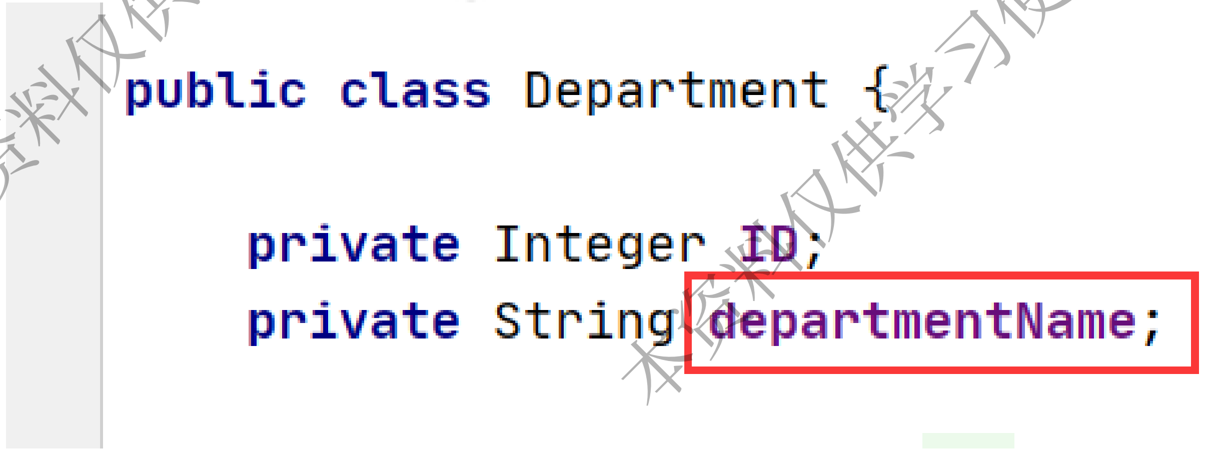

localhost:9090/dept/3

```
{"departmentName":"人事部","id":3}
```

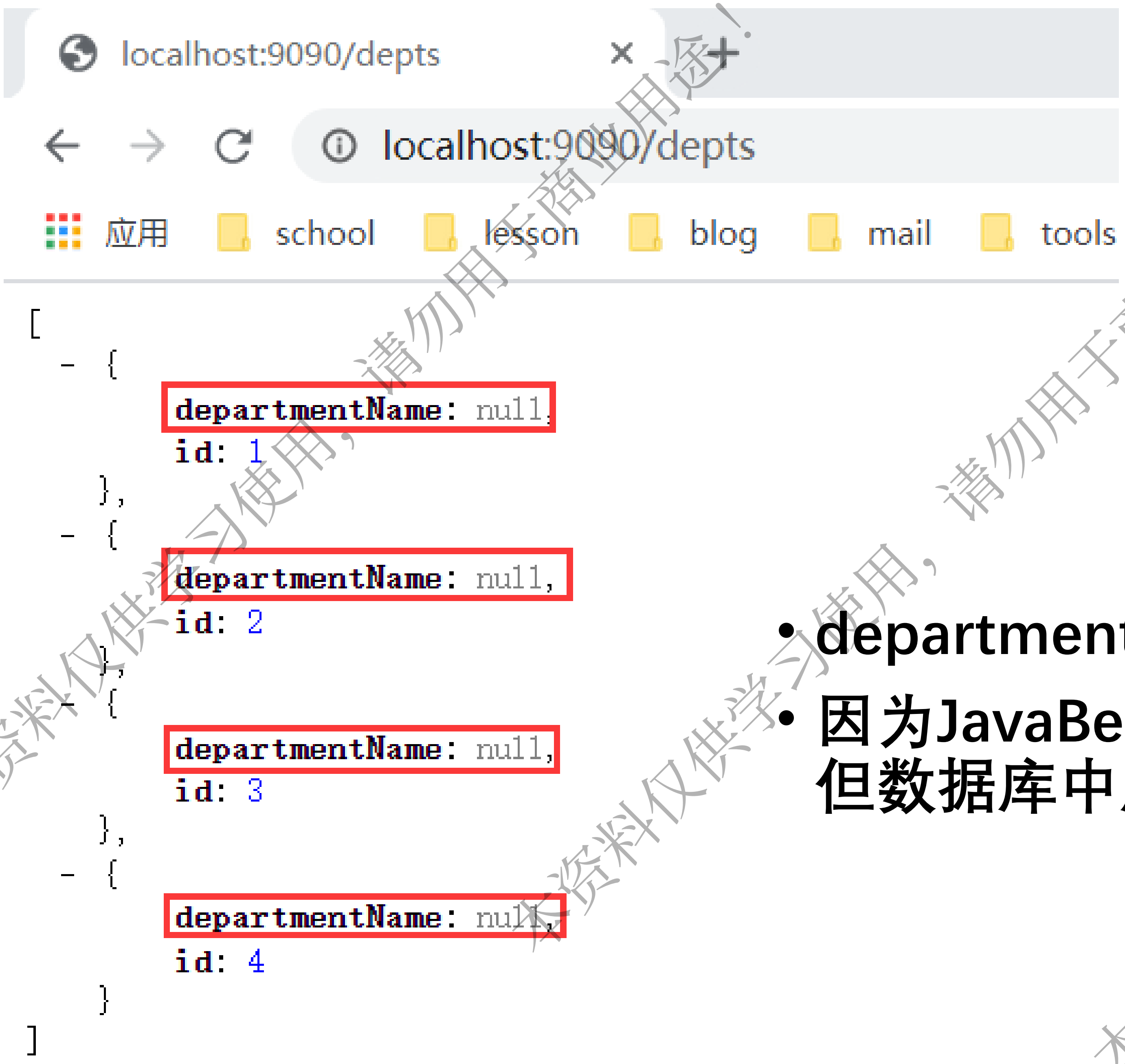
## • 数据库名和类名的相关属性名不同时：



| Name | Type | Length | Decimals | Not null |
|---|---|---|---|---|
| ID | int | 0 | 0 | ✓ |
| department_name | varchar | 10 | 0 | ☐ |

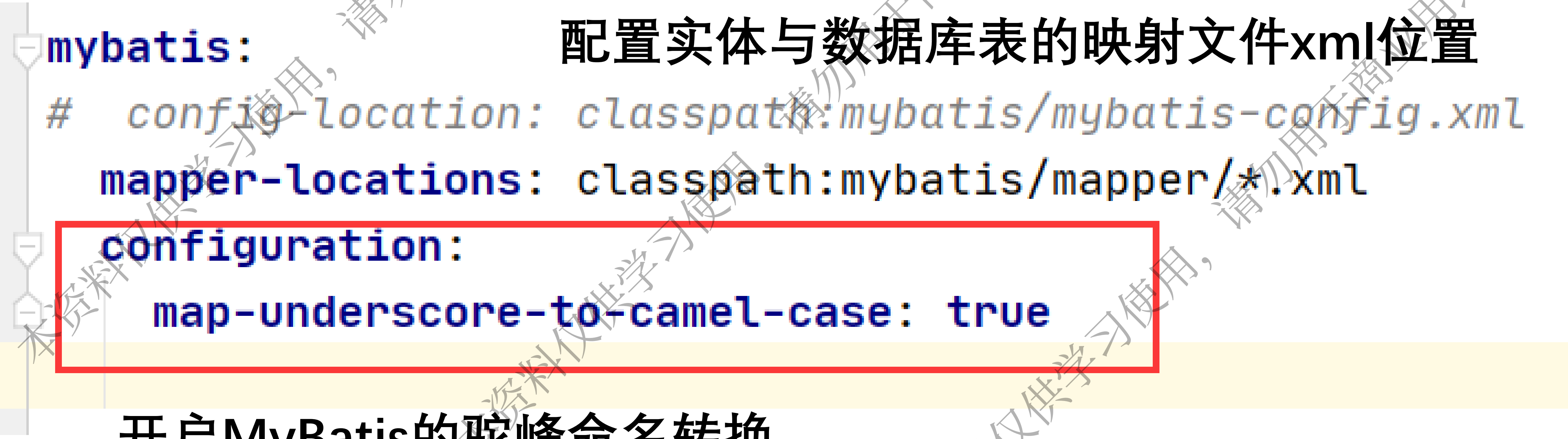


```
public class Department {

    private Integer ID;
    private String departmentName;
```



```
[
  - {
        departmentName: null,
        id: 1
    },
  - {
        departmentName: null,
        id: 2
    },
  - {
        departmentName: null,
        id: 3
    },
  - {
        departmentName: null,
        id: 4
    }
]
```

- departmentName封装不上
- 因为JavaBean属性名叫departmentName 但数据库中属性名叫department_name

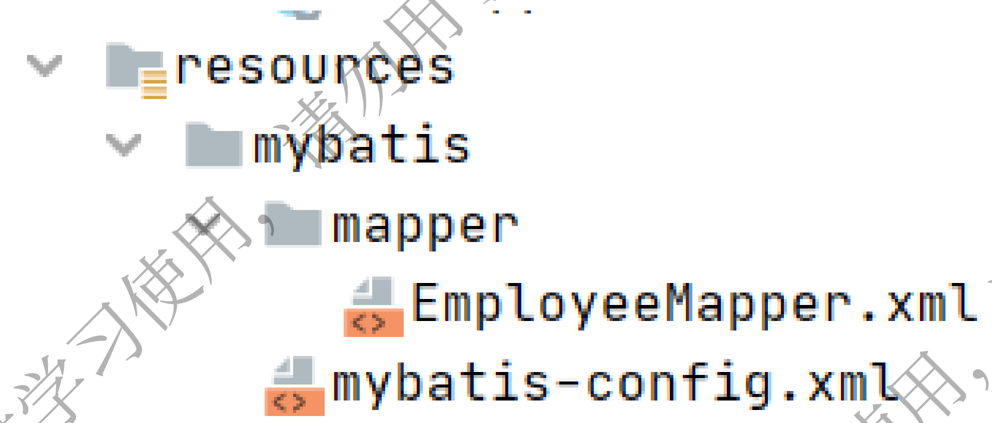


```
mybatis:
#  config-location: classpath:mybatis/mybatis-config.xml
  mapper-locations: classpath:mybatis/mapper/*.xml
  configuration:
    map-underscore-to-camel-case: true
```

配置实体与数据库表的映射文件xml位置

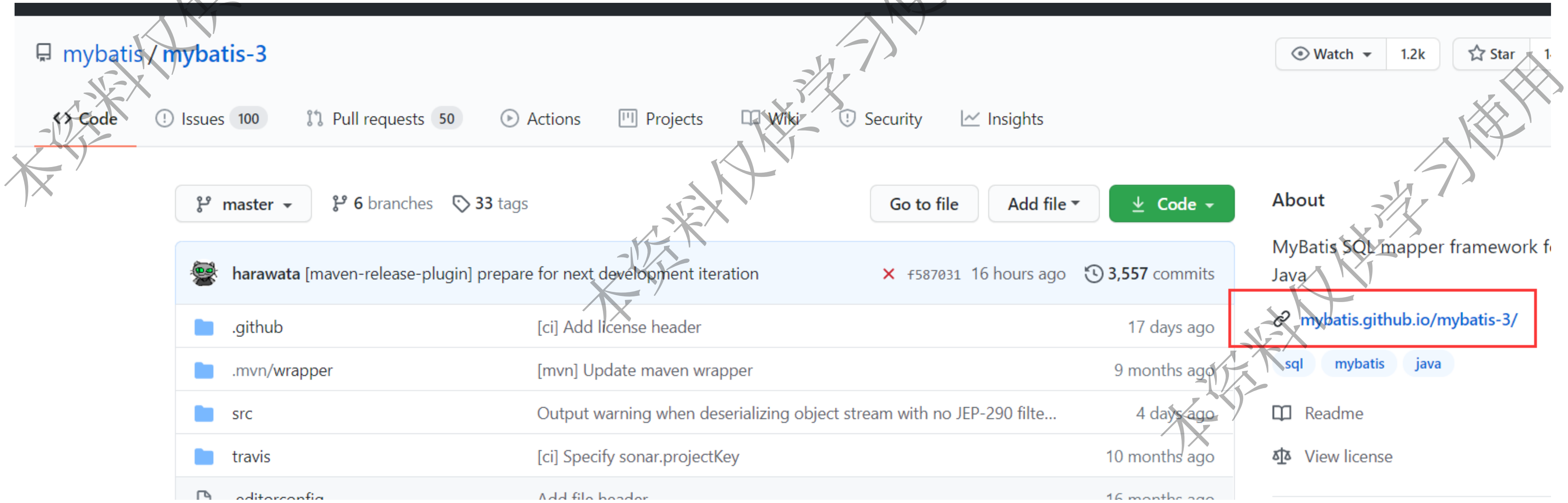
开启MyBatis的驼峰命名转换

- ② 配置版MyBatis（EmployeeMapper）
- 在resources下创建mybatis.mapper（存所有的SQL映射文件）



```
resources
  mybatis
    mapper
      EmployeeMapper.xml
  mybatis-config.xml
```

**映射文件参照MyBatis的官方文档**
**MyBatis代码都托管到了GitHub上**



mybatis / mybatis-3

Watch 1.2k | Star

Code | Issues 100 | Pull requests 50 | Actions | Projects | Wiki | Security | Insights

master | 6 branches | 33 tags | Go to file | Add file | Code

harawata [maven-release-plugin] prepare for next development iteration ✕ f587031 16 hours ago 3,557 commits

| | | |
|---|---|---|
| .github | [ci] Add license header | 17 days ago |
| .mvn/wrapper | [mvn] Update maven wrapper | 9 months ago |
| src | Output warning when deserializing object stream with no JEP-290 filte... | 4 days ago |
| travis | [ci] Specify sonar.projectKey | 10 months ago |
| .editorconfig | Add file header | 16 months ago |

About

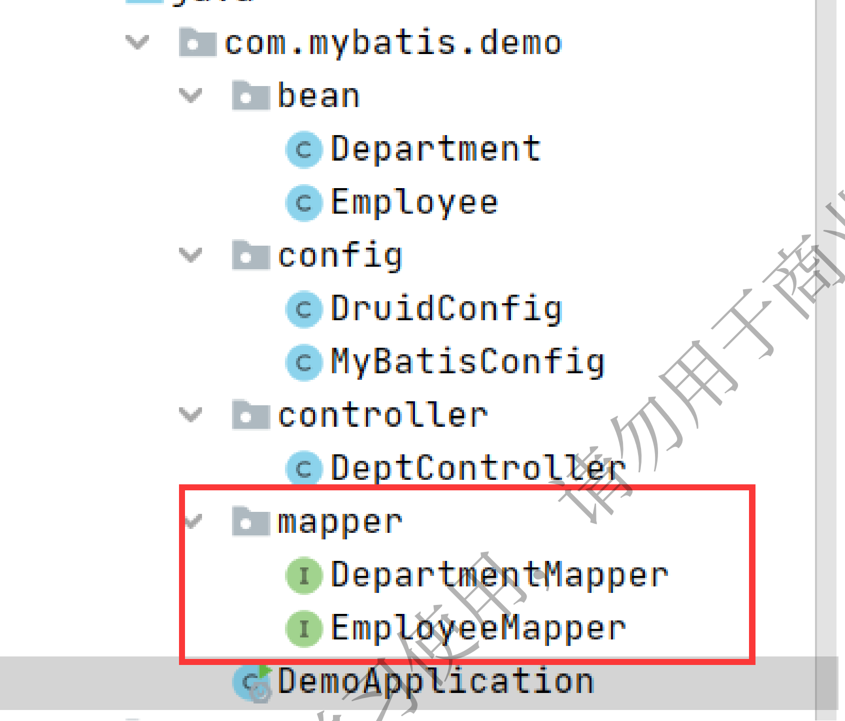
MyBatis SQL mapper framework f
Java

mybatis.github.io/mybatis-3/

sql | mybatis | java

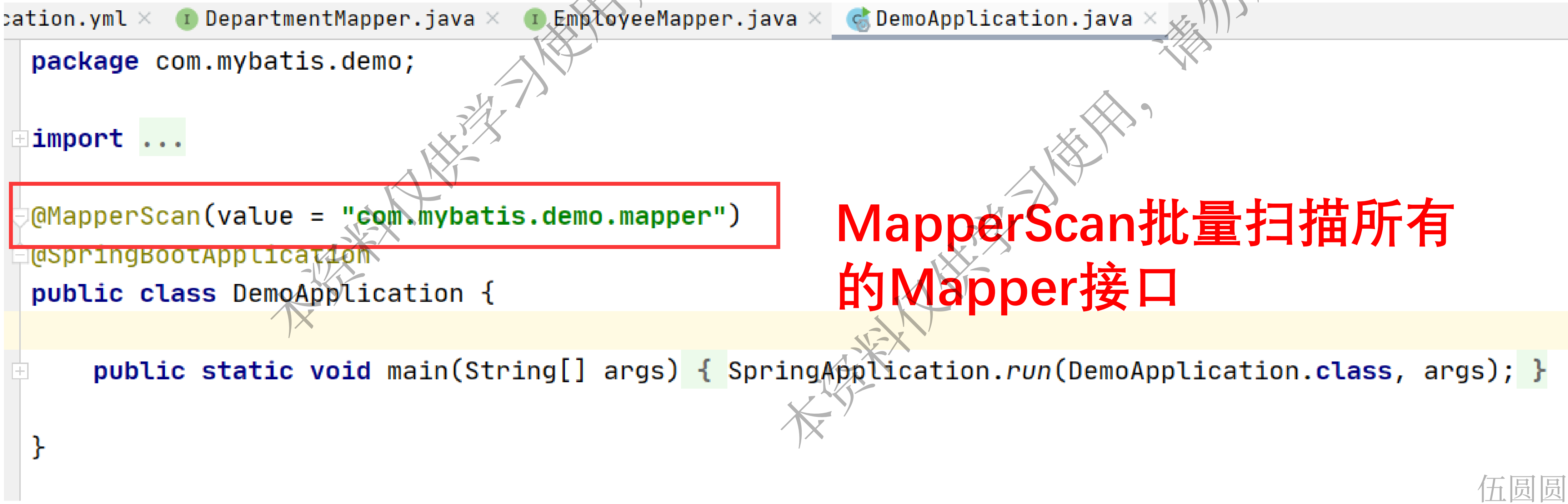
Readme

View license



```
com.mybatis.demo
  bean
    Department
    Employee
  config
    DruidConfig
    MyBatisConfig
  controller
    DeptController
  mapper
    DepartmentMapper
    EmployeeMapper
  DemoApplication
```

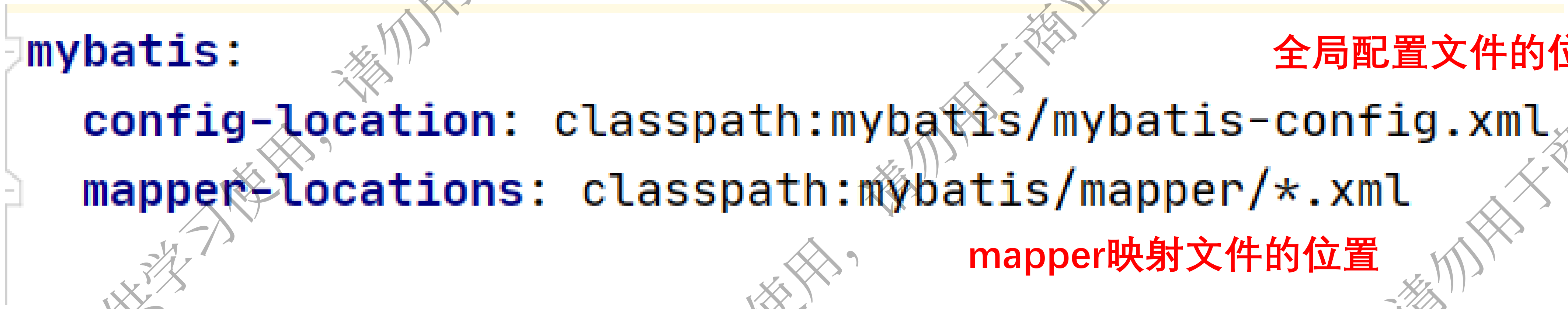
如果mapper特别多，每一个mapper上都要标注一个@Mapper注解



```java
package com.mybatis.demo;

import ...

@MapperScan(value = "com.mybatis.demo.mapper")
@SpringBootApplication
public class DemoApplication {

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

}
```

MapperScan批量扫描所有的Mapper接口

- 在application.yml中配置：让MyBatis知道配置文件的存在

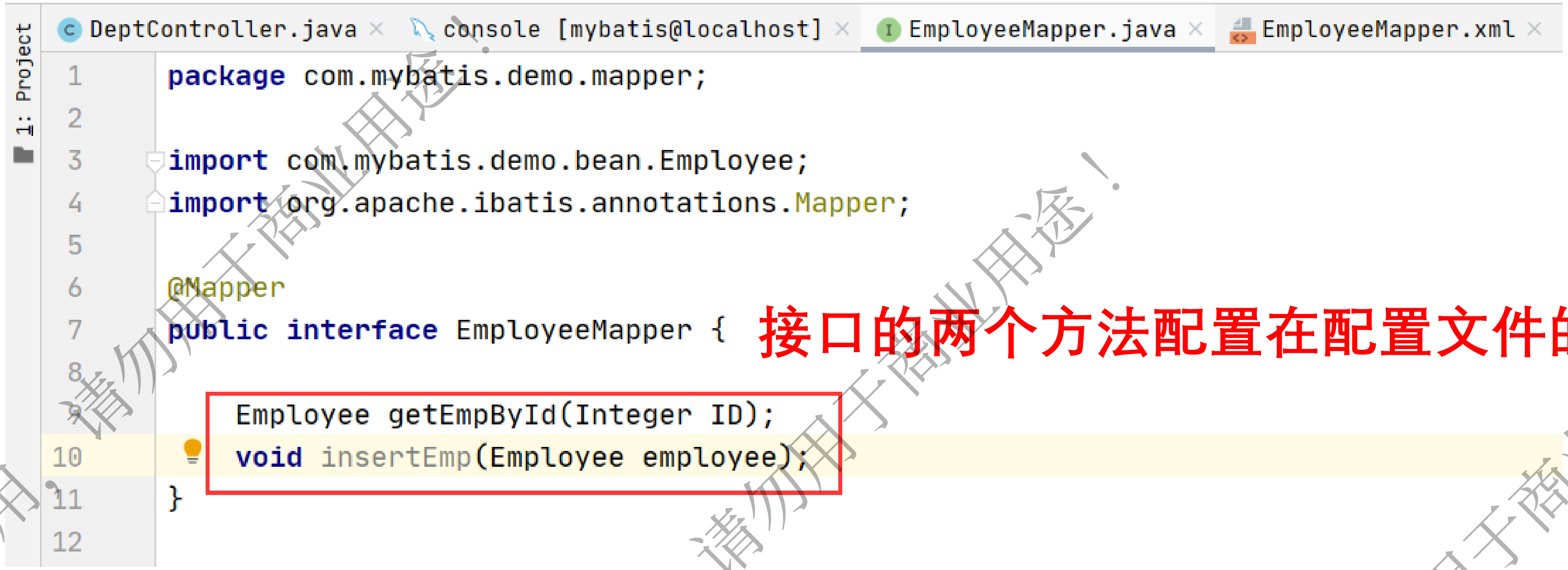


```yaml
mybatis:
  config-location: classpath:mybatis/mybatis-config.xml
  mapper-locations: classpath:mybatis/mapper/*.xml
```

全局配置文件的位置

mapper映射文件的位置

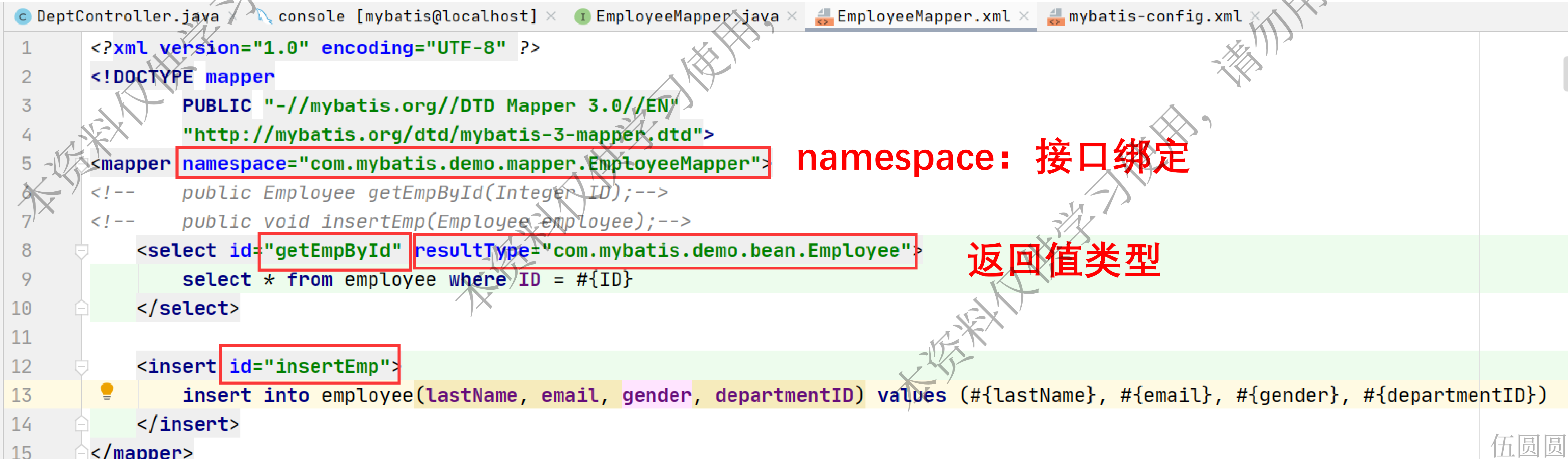
- 接口绑定



DeptController.java × console [mybatis@localhost] × EmployeeMapper.java × EmployeeMapper.xml ×

```java
package com.mybatis.demo.mapper;

import com.mybatis.demo.bean.Employee;
import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface EmployeeMapper {

    Employee getEmpById(Integer ID);
    void insertEmp(Employee employee);
}
```

接口的两个方法配置在配置文件的映射里

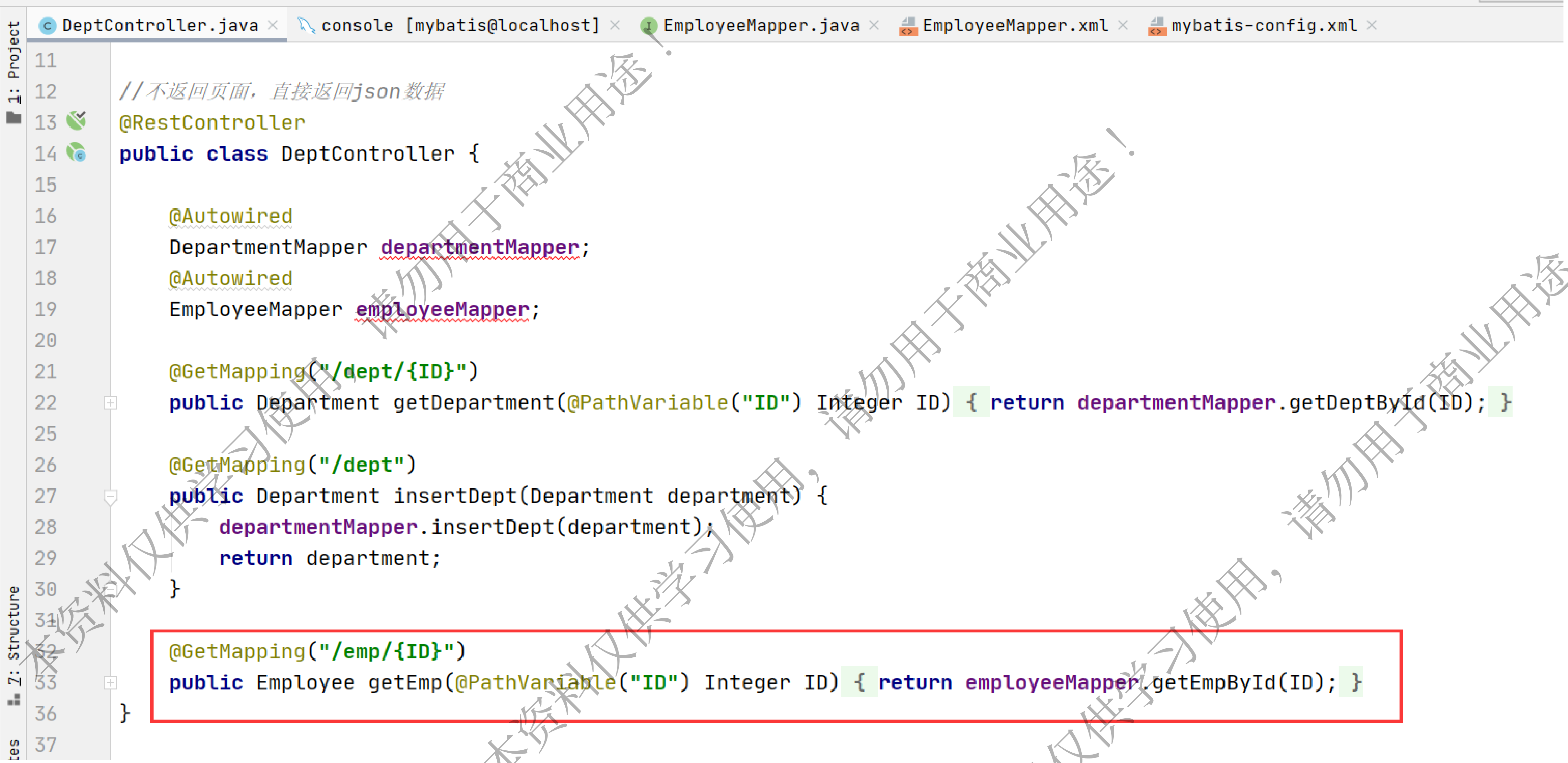


DeptController.java × console [mybatis@localhost] × EmployeeMapper.java × EmployeeMapper.xml × mybatis-config.xml ×

```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="com.mybatis.demo.mapper.EmployeeMapper">
<!--    public Employee getEmpById(Integer ID);-->
<!--    public void insertEmp(Employee employee);-->
    <select id="getEmpById" resultType="com.mybatis.demo.bean.Employee">
        select * from employee where ID = #{ID}
    </select>

    <insert id="insertEmp">
        insert into employee(lastName, email, gender, departmentID) values (#{lastName}, #{email}, #{gender}, #{departmentID})
    </insert>
</mapper>
```

namespace：接口绑定

返回值类型

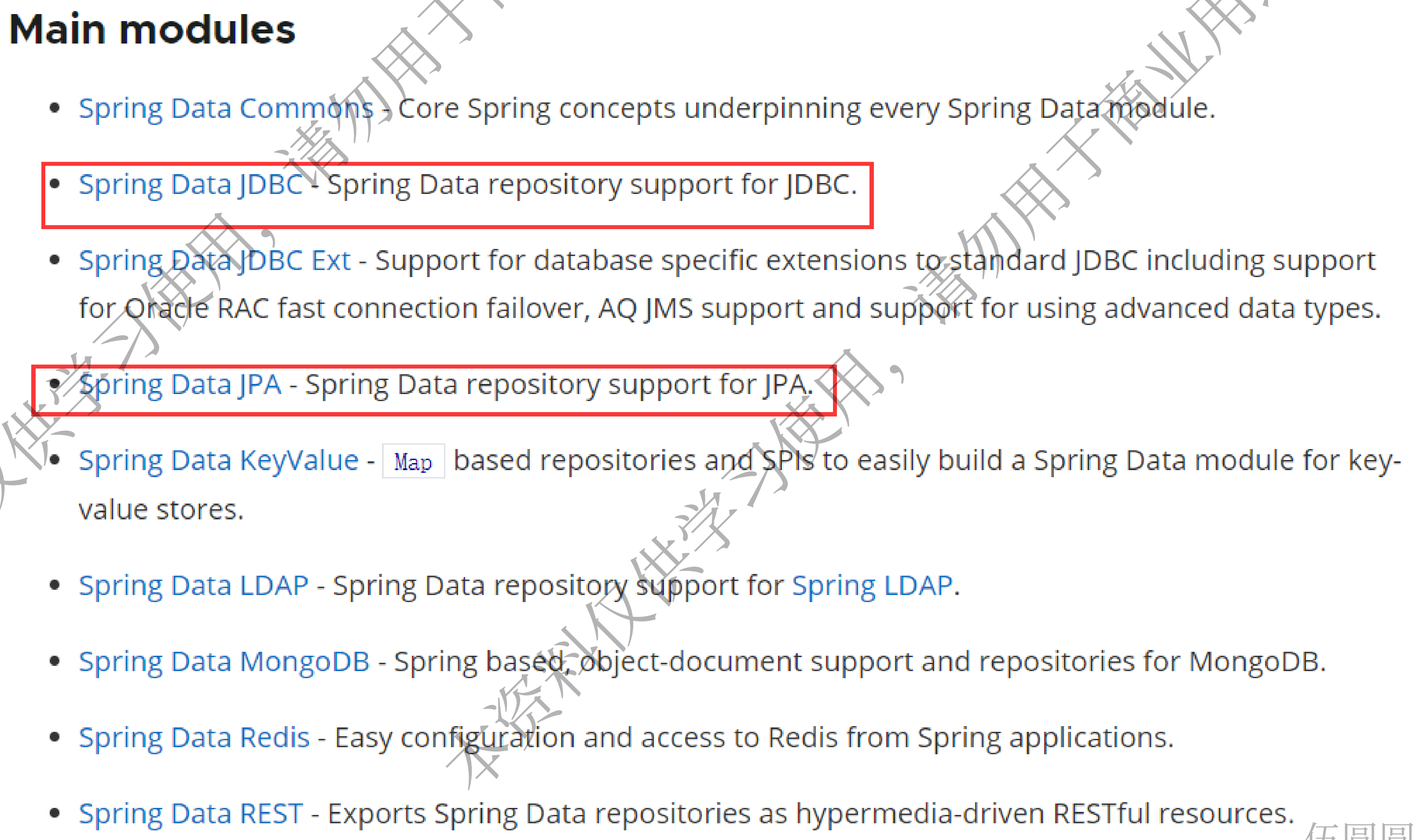

DeptController.java × console [mybatis@localhost] × EmployeeMapper.java × EmployeeMapper.xml × mybatis-config.xml ×

```java

//不返回页面，直接返回json数据
@RestController
public class DeptController {

    @Autowired
    DepartmentMapper departmentMapper;
    @Autowired
    EmployeeMapper employeeMapper;

    @GetMapping("/dept/{ID}")
    public Department getDepartment(@PathVariable("ID") Integer ID) { return departmentMapper.getDeptById(ID); }

    @GetMapping("/dept")
    public Department insertDept(Department department) {
        departmentMapper.insertDept(department);
        return department;
    }

    @GetMapping("/emp/{ID}")
    public Employee getEmp(@PathVariable("ID") Integer ID) { return employeeMapper.getEmpById(ID); }
}

```

## •3.JPA（Java持久层API　描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中）

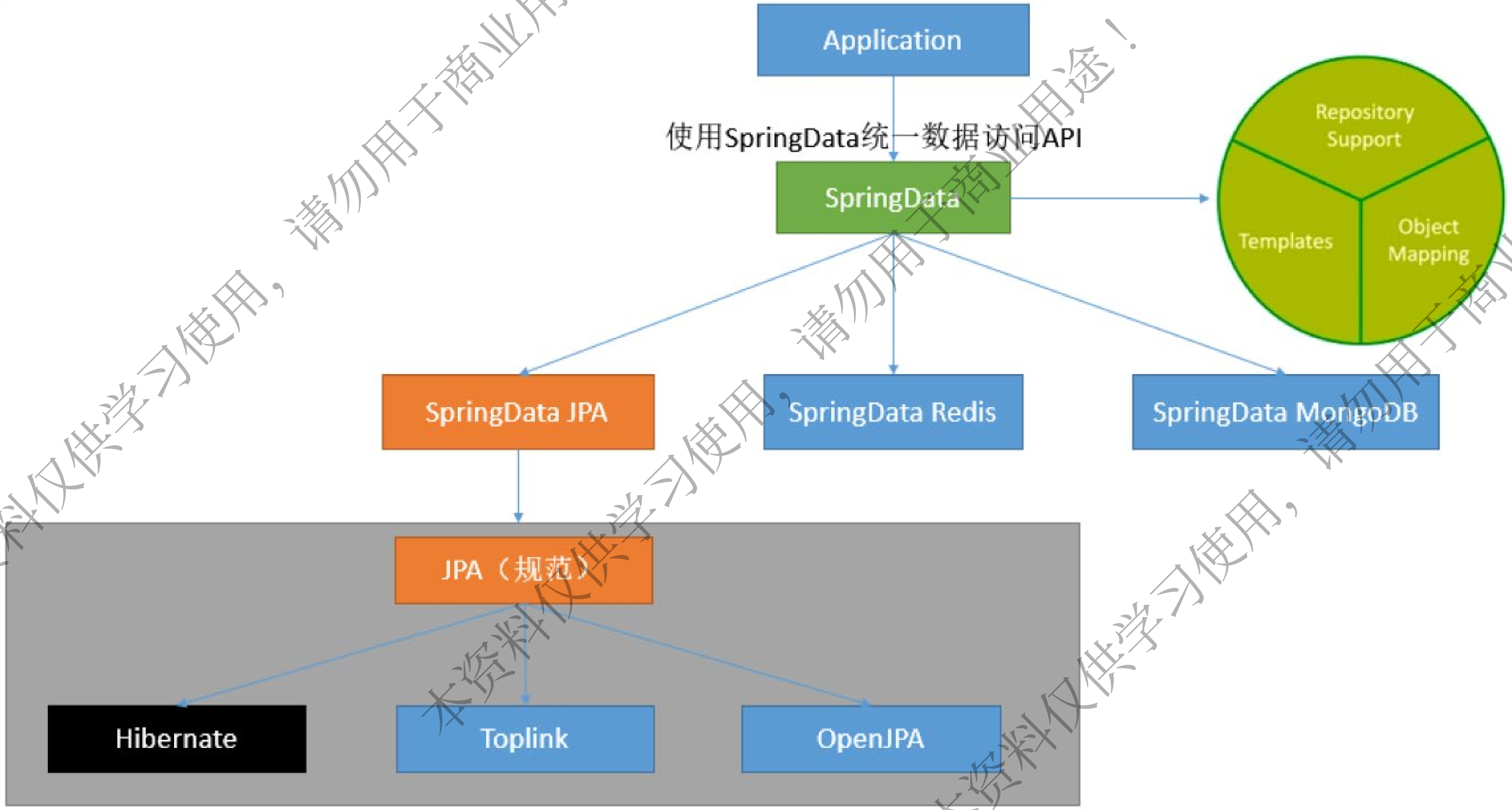
Spring Data
Spring Boot底层默认进行数据访问采用的技术
<u>简化数据访问</u>

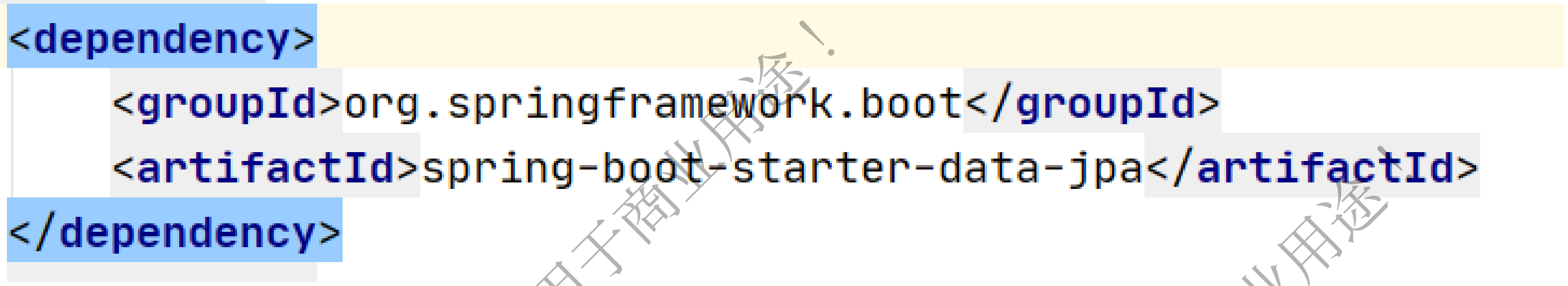
特点:
① 提供统一的API对数据访问层进行操作
② 有统一的Repository接口
Repository<T, ID extends Serializable>



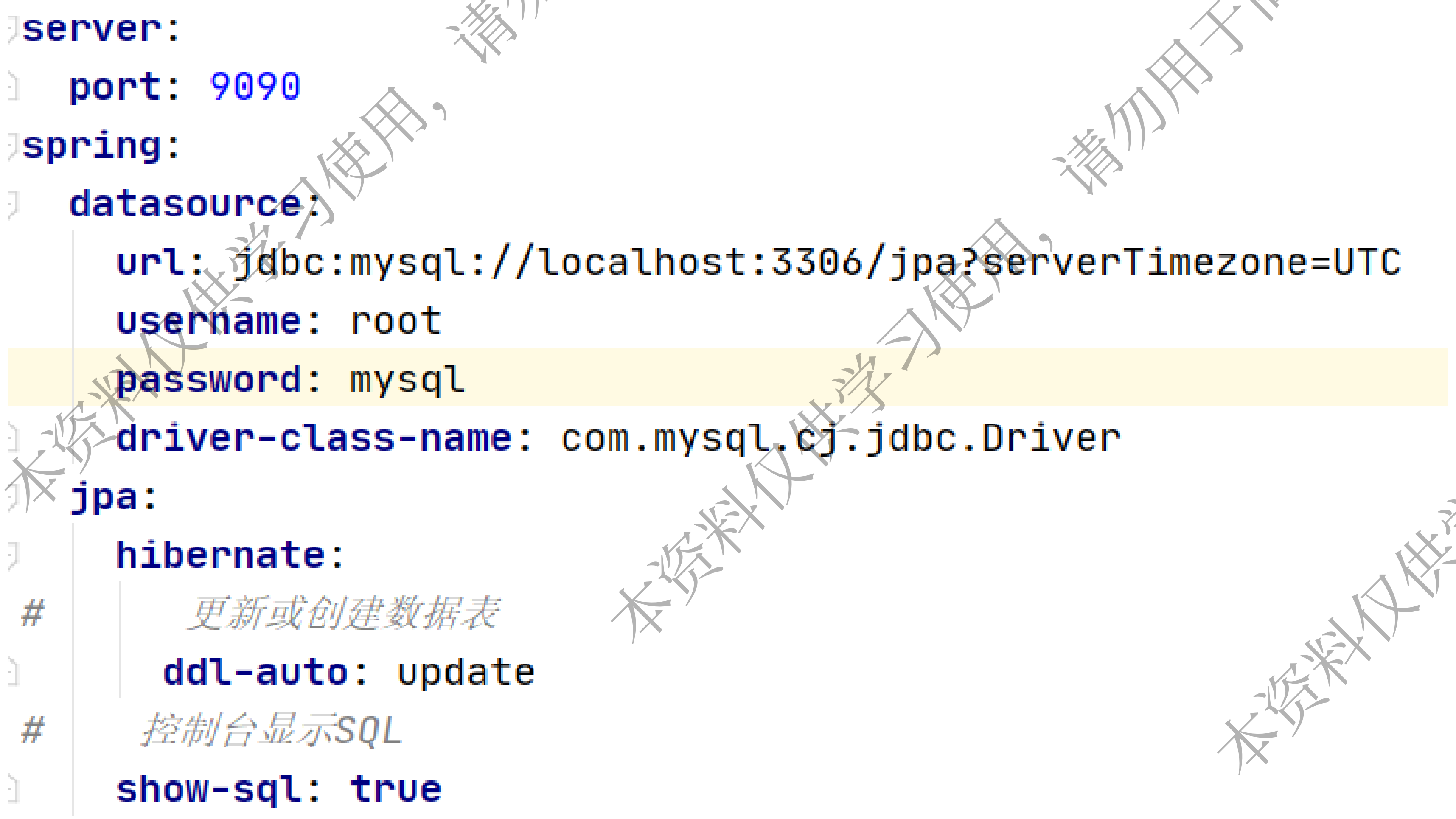
**Main modules**

- Spring Data Commons - Core Spring concepts underpinning every Spring Data module.
- Spring Data JDBC - Spring Data repository support for JDBC.
- Spring Data JDBC Ext - Support for database specific extensions to standard JDBC including support for Oracle RAC fast connection failover, AQ JMS support and support for using advanced data types.
- Spring Data JPA - Spring Data repository support for JPA.
- Spring Data KeyValue - `Map` based repositories and SPIs to easily build a Spring Data module for key-value stores.
- Spring Data LDAP - Spring Data repository support for Spring LDAP.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Easy configuration and access to Redis from Spring applications.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.


Application
使用SpringData统一数据访问API
SpringData
Repository Support
Templates
Object Mapping
SpringData JPA
SpringData Redis
SpringData MongoDB
JPA（规范）
Hibernate
Toplink
OpenJPA

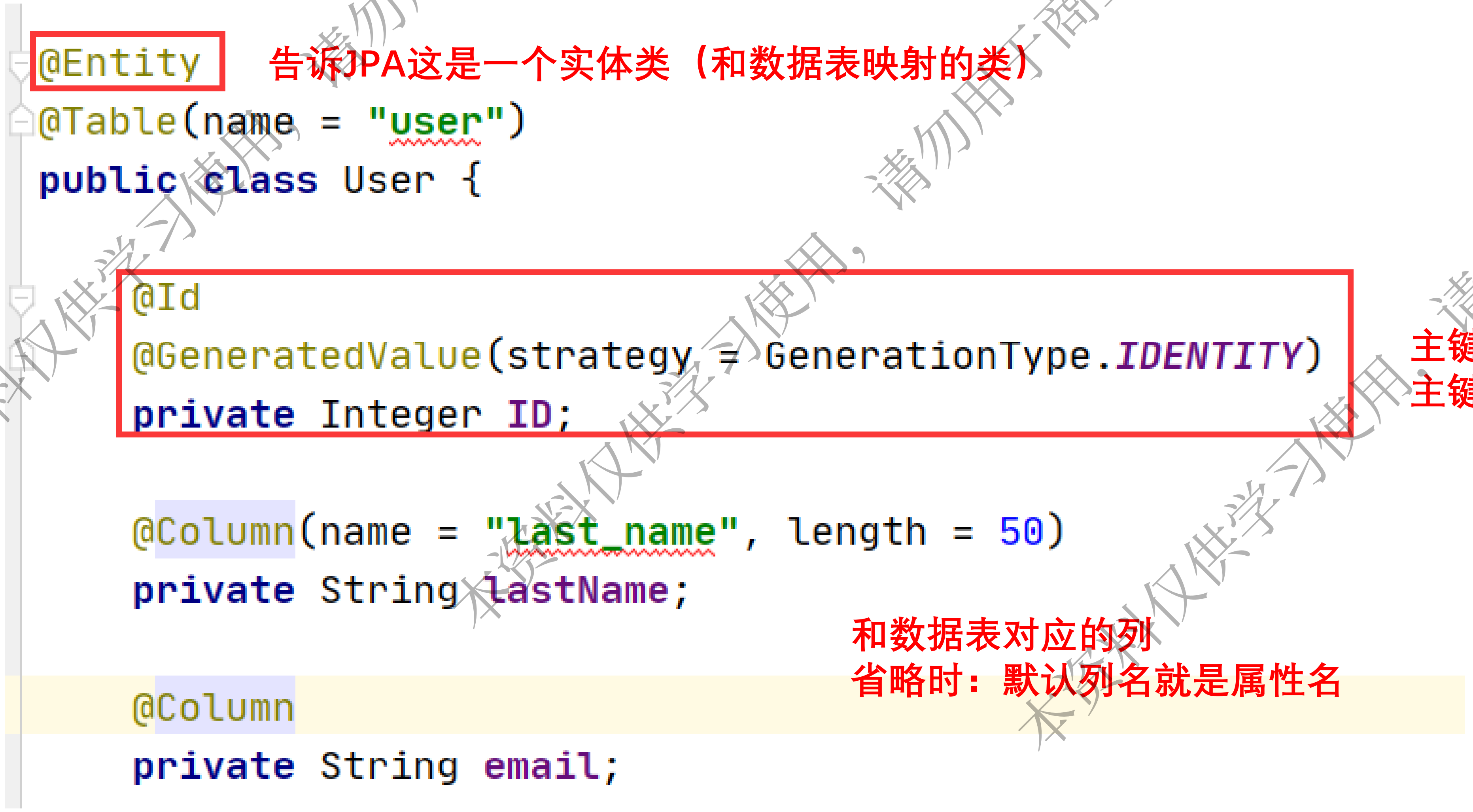


```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```



```yaml
server:
  port: 9090
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/jpa?serverTimezone=UTC
    username: root
    password: mysql
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
#      更新或创建数据表
      ddl-auto: update
#    控制台显示SQL
    show-sql: true
```

1.编写一个实体类和数据表进行映射，并配置好映射关系



```
@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer ID;

    @Column(name = "last_name", length = 50)
    private String lastName;

    @Column
    private String email;
```

告诉JPA这是一个实体类（和数据表映射的类）

主键
主键自增

和数据表对应的列
省略时：默认列名就是属性名

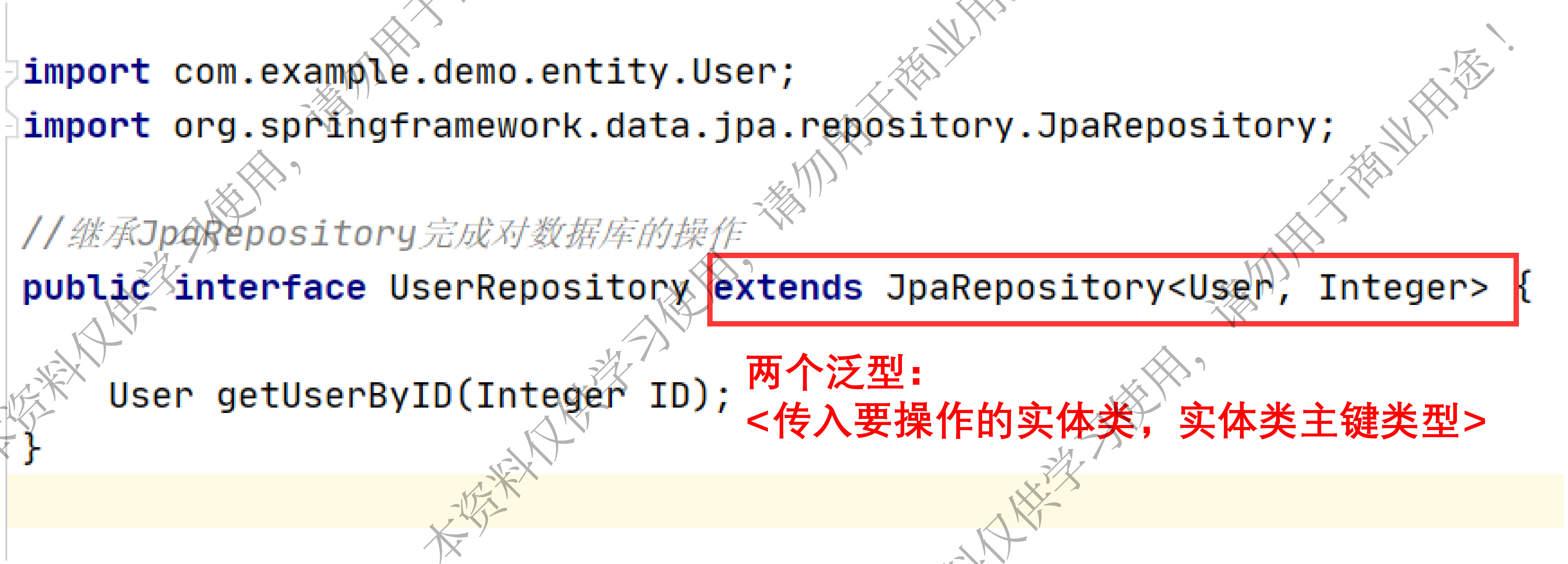
- @Entity：表明该类为一个实体类，默认对应数据库中的表名为实体名的小写

- @Entity
- @Table(name = "user", schema = "test")
- name：数据库表名；catalog：数据库目录；schema：数据库模式

- @Column：定义了将成员属性映射到关系表中哪一列和该列的结构信息

- @Id：注释指定表的主键

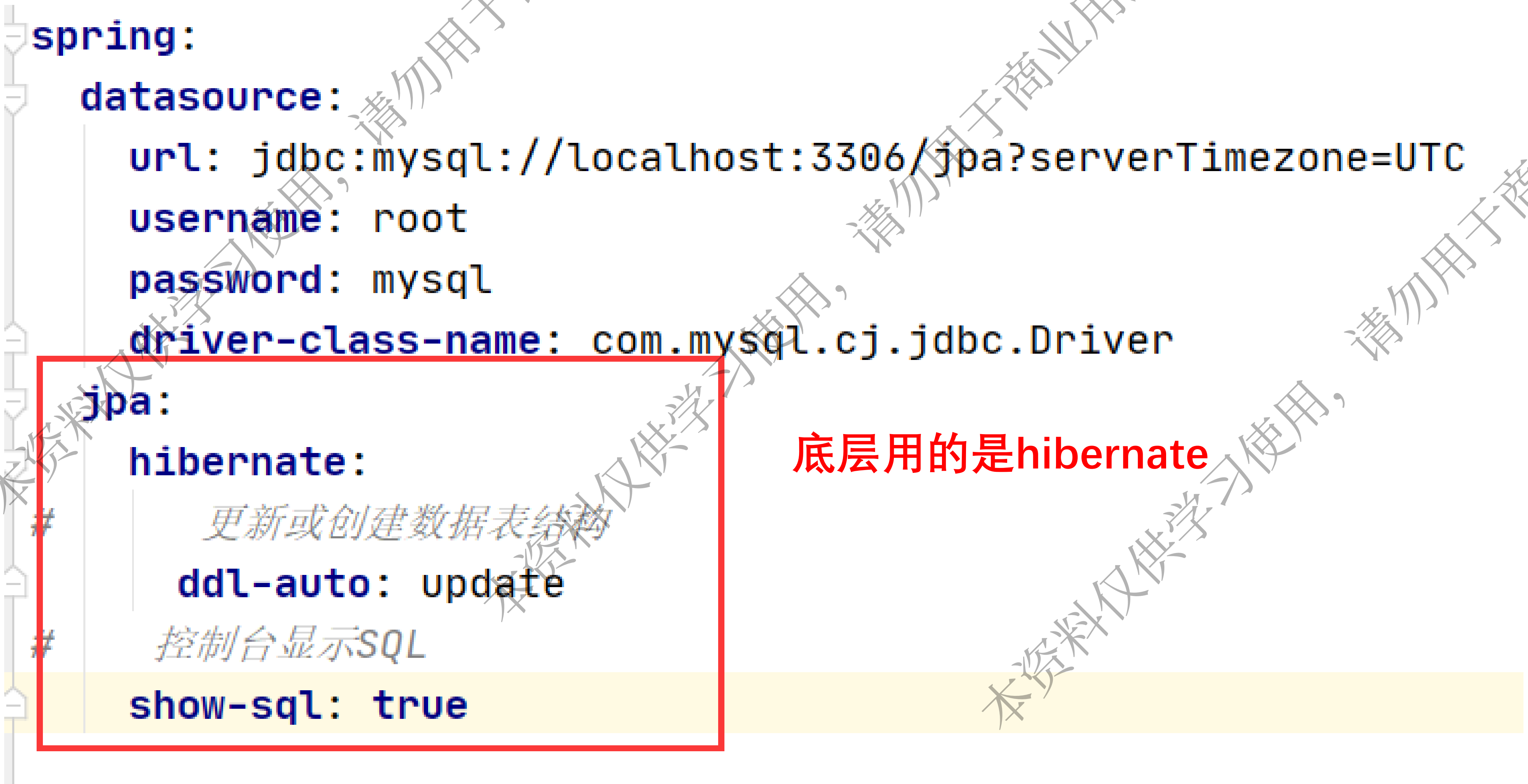
- 2.编写一个接口（Repository）操作实体类对应的数据表



```java
import com.example.demo.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

//继承JpaRepository完成对数据库的操作
public interface UserRepository extends JpaRepository<User, Integer> {

    User getUserByID(Integer ID);
}
```

两个泛型：
<传入要操作的实体类，实体类主键类型>

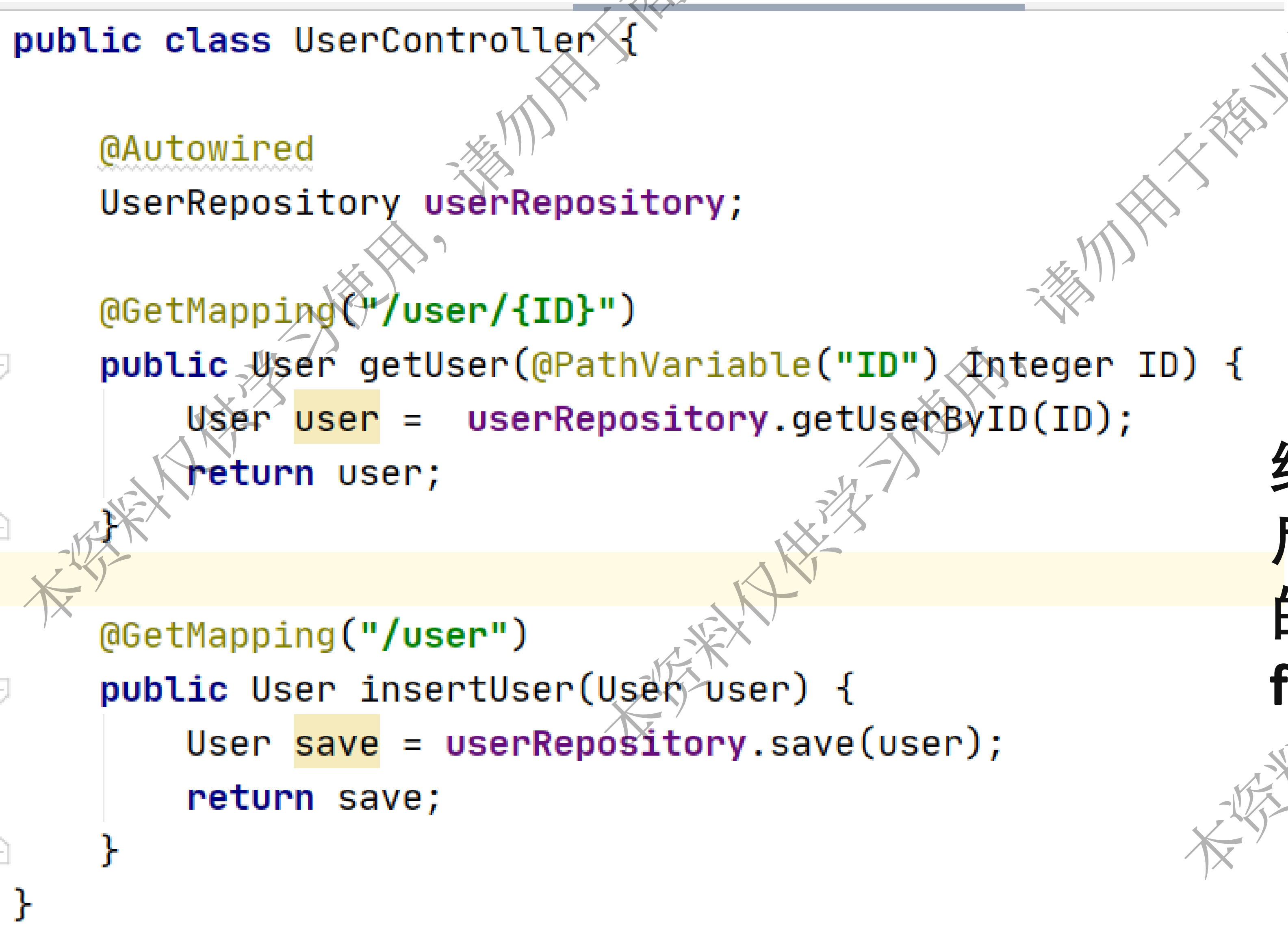
## 基本配置



```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/jpa?serverTimezone=UTC
    username: root
    password: mysql
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
#       更新或创建数据表结构
      ddl-auto: update
#     控制台显示SQL
    show-sql: true
```

底层用的是hibernate

## 如何进行增删改查?



```
public class UserController {

    @Autowired
    UserRepository userRepository;

    @GetMapping("/user/{ID}")
    public User getUser(@PathVariable("ID") Integer ID) {
        User user =  userRepository.getUserByID(ID);
        return user;
    }

    @GetMapping("/user")
    public User insertUser(User user) {
        User save = userRepository.save(user);
        return save;
    }
}
```

**继承 JpaRepository 接口之后自动具备了 一系列常用的数据操作方法：findAll、findOne 、save等**