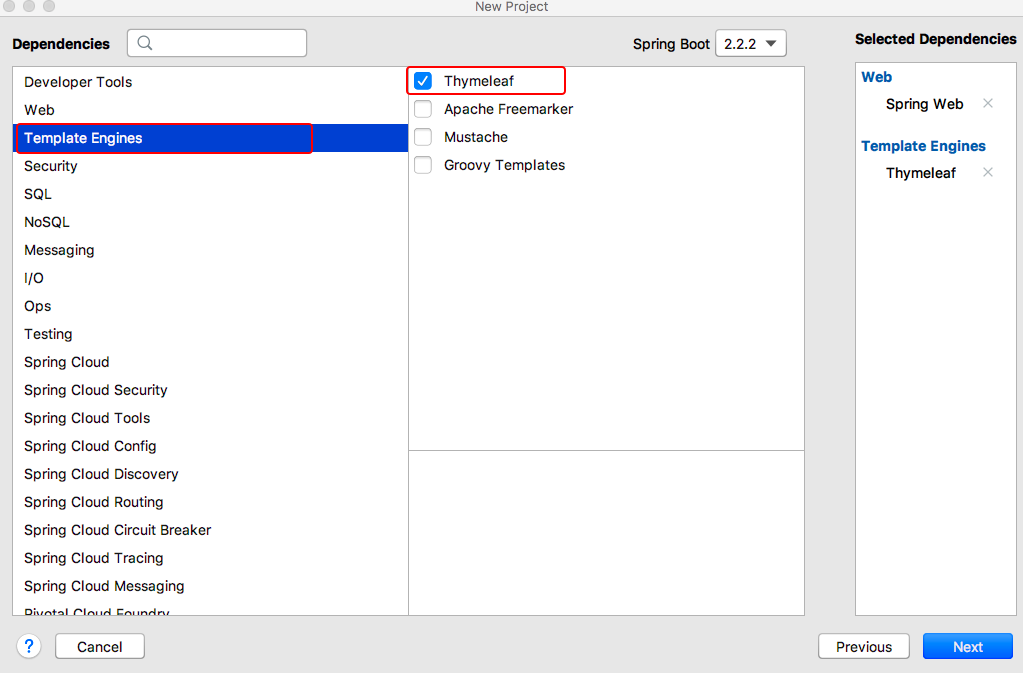
### JAVA第三阶段—DAY17-JAVA案例

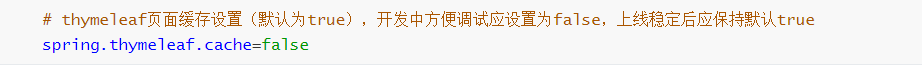
# 1 Spring Boot整合Thymeleaf

1创建Spring Boot项目，引入Thymeleaf依赖



2. 编写配置文件

打开application.properties全局配置文件，在该文件中对Thymeleaf模板页面的数据缓存进行设置



使用“spring.thymeleaf.cache=false”将Thymeleaf默认开启的缓存设置为了false，用来关闭模板页面缓存，在实际生产环境下建议开启缓存。

3. 创建web控制类

在项目中创建名为com.demo.controller的包，并在该包下创建一个用于前端模板页面动态数据替换效果测试的访问实体类LoginController

|  |
| --- |
| @Controller  public class LoginController {  /\*\*  \* 获取并封装当前年份跳转到登录页login.html  \*/  @RequestMapping("/toLoginPage")  public String toLoginPage(Model model){  model.addAttribute("currentYear", Calendar.getInstance().get(Calendar.YEAR));  return "login";  } |

toLoginPage()方法用于向登录页面login.html跳转，同时携带了当前年份信息currentYear。

4 创建模板页面并引入静态资源文件

在“classpath:/templates/”目录下引入一个用户登录的模板页面login.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns:th="http://www.thymeleaf.org">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1,shrink-to-fit=no">  <title>用户登录界面</title>  <link th:href="@{/login/css/bootstrap.min.css}" rel="stylesheet">  <link th:href="@{/login/css/signin.css}" rel="stylesheet">  </head>  <body class="text-center">  <!-- 用户登录form表单 -->  <form class="form-signin">  <img class="mb-4" th:src="@{/login/img/login.jpg}" width="72" height="72">  <h1 class="h3 mb-3 font-weight-normal">请登录</h1>  <input type="text" class="form-control"  th:placeholder="用户名" required="" autofocus="">  <input type="password" class="form-control"  th:placeholder="密码" required="">  <div class="checkbox mb-3">  <label>  <input type="checkbox" value="remember-me"> 记住我  </label>  </div>  <button class="btn btn-lg btn-primary btn-block" type="submit" >登录</button>  <p class="mt-5 mb-3 text-muted">© <span th:text="${currentYear}">2019</span>-<span th:text="${currentYear}+1">2020</span></p>  </form>  </body>  </html> |

通过“xmlns:th="http://www.thymeleaf.org"”引入了Thymeleaf模板标签；

使用“th:href”和“th:src”分别引入了两个外联的样式文件和一个图片；

使用“th:text”引入了后台动态传递过来的当前年份currentYear

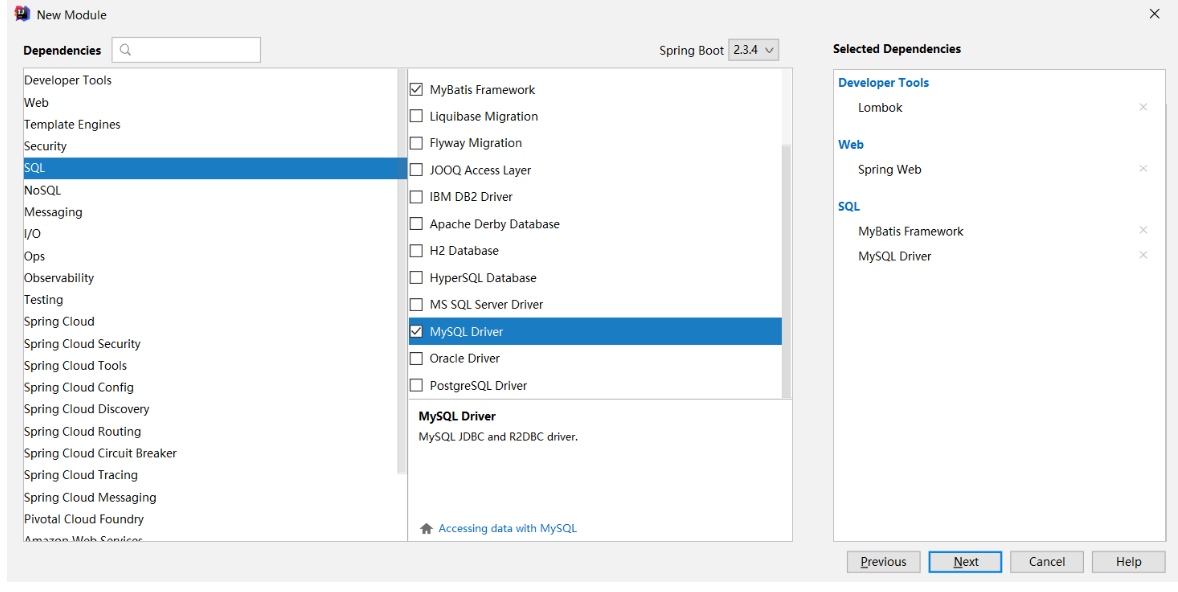
5．效果测试



可以看出，登录页面login.html显示正常，在文件4-3中使用“th:\*”相关属性引入的静态文件生效，并且在页面底部动态显示了当前日期2019-2020，而不是文件中的静态数字2019-2020。这进一步说明了Spring Boot与Thymeleaf整合成功，完成了静态资源的引入和动态数据的显示

# 2 SpringBoot实战演练

1 创建Spring boot工程



2 编辑pom.xml

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.3</version>  </dependency> |

3 User实体类编写

|  |
| --- |
| @Data  public class User implements Serializable {  private Integer id;  private String username;  private String password;  private String birthday;  private static final long serialVersionUID = 1L;  } |

4 UserMapper编写及Mapper文件

|  |
| --- |
| public interface UserMapper {  int deleteByPrimaryKey(Integer id);  int insert(User record);  int insertSelective(User record);  User selectByPrimaryKey(Integer id);  int updateByPrimaryKeySelective(User record);  int updateByPrimaryKey(User record);  List<User> queryAll();  } |

|  |
| --- |
| <?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.example.base.mapper.UserMapper">  <resultMap id="BaseResultMap" type="com.example.base.pojo.User">  <id column="id" jdbcType="INTEGER" property="id" />  <result column="username" jdbcType="VARCHAR" property="username" />  <result column="password" jdbcType="VARCHAR" property="password" />  <result column="birthday" jdbcType="VARCHAR" property="birthday" />  </resultMap>  <sql id="Base\_Column\_List">  id, username, password, birthday  </sql>  <select id="selectByPrimaryKey" parameterType="java.lang.Integer" resultMap="BaseResultMap">  select  <include refid="Base\_Column\_List" />  from user  where id = #{id,jdbcType=INTEGER}  </select>  <select id="queryAll" resultType="com.example.base.pojo.User">  select <include refid="Base\_Column\_List" />  from user  </select>  .......  </mapper> |

5 UserService接口及实现类编写

|  |
| --- |
| package com.demo.service.impl;  import com.demo.mapper.UserMapper;  import com.demo.pojo.User;  import com.demo.service.UserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import java.util.List;  @Service  public class UserServiceImpl implements UserService {  @Autowired  private UserMapper userMapper;  @Override  public List<User> queryAll() {  return userMapper.queryAll();  }  @Override  public User findById(Integer id) {  return userMapper.selectByPrimaryKey(id);  }  @Override  public void insert(User user) {  //userMapper.insert(user); //将除id所有的列都拼SQL  userMapper.insertSelective(user); //只是将不为空的列才拼SQL  }  @Override  public void deleteById(Integer id) {  userMapper.deleteByPrimaryKey(id);  }  @Override  public void update(User user) {  userMapper.updateByPrimaryKeySelective(user);  }  } |

6 UserController编写

|  |
| --- |
| package com.demo.controller;  import com.demo.pojo.User;  import com.demo.service.UserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.\*;  import org.yaml.snakeyaml.events.Event;  import java.util.List;  @RestController  @RequestMapping("/user")  public class UserController {  @Autowired  private UserService userService;  /\*\*  \* restful格式进行访问  \* 查询：GET  \* 新增: POST  \* 更新：PUT  \* 删除: DELETE  \*/  /\*\*  \* 查询所有  \* @return  \*/  @GetMapping("/query")  public List<User> queryAll(){  return userService.queryAll();  }  /\*\*  \* 通过ID查询  \*/  @GetMapping("/query/{id}")  public User queryById(@PathVariable Integer id){  return userService.findById(id);  }  /\*\*  \* 删除  \* @param id  \* @return  \*/  @DeleteMapping("/delete/{id}")  public String delete(@PathVariable Integer id){  userService.deleteById(id);  return "删除成功";  }  /\*\*  \* 新增  \* @param user  \* @return  \*/  @PostMapping("/insert")  public String insert(User user){  userService.insert(user);  return "新增成功";  }  /\*\*  \* 修改  \* @param user  \* @return  \*/  @PutMapping("/update")  public String update(User user){  userService.update(user);  return "修改成功";  }  } |

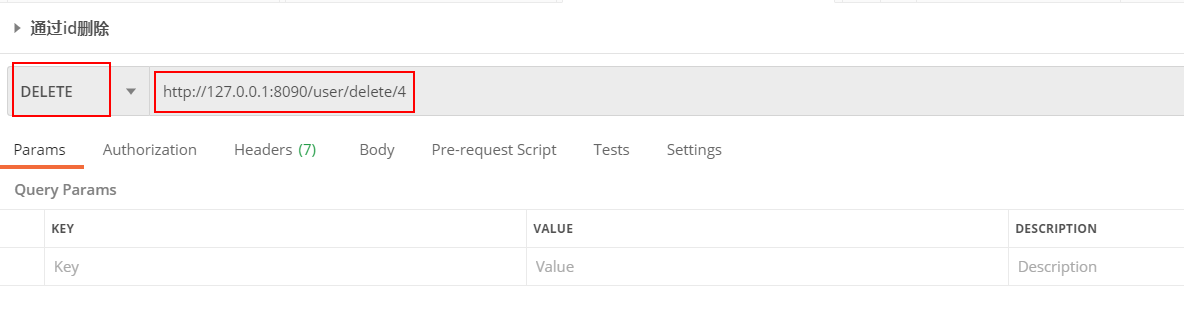
7 application.yml

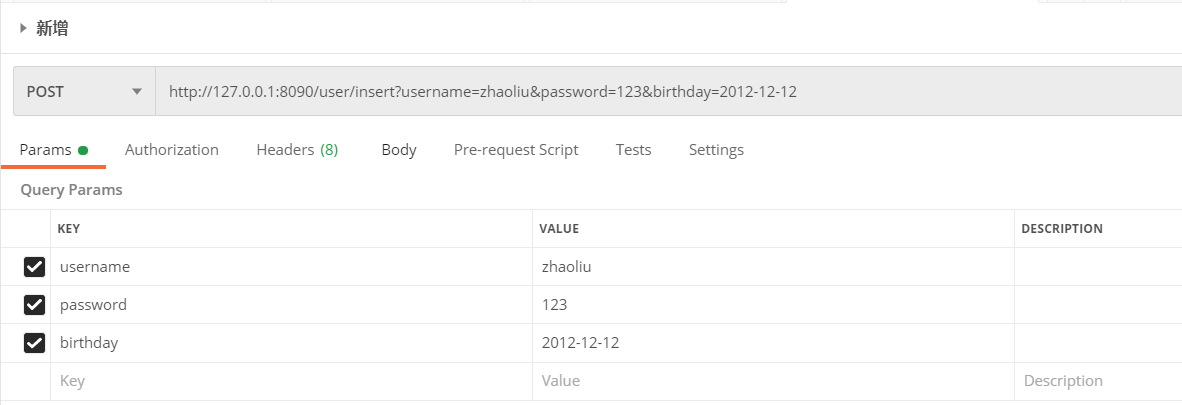
|  |
| --- |
| ##服务器配置  server:  port: 8090  servlet:  context-path: /  ##数据源配置  spring:  datasource:  name: druid  type: com.alibaba.druid.pool.DruidDataSource  url: jdbc:mysql://localhost:3306/springbootdata?characterEncoding=utf-8&serverTimezone=UTC  username: root  password: wu7787879  #整合mybatis  mybatis:  mapper-locations: classpath:mapper/\*Mapper.xml #声明Mybatis映射文件所在的位置 |

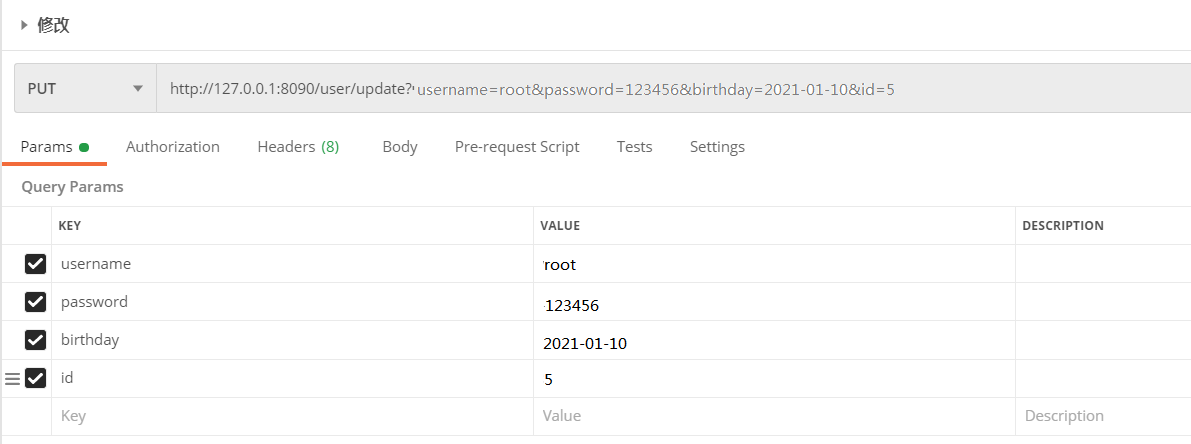
8 启动类

|  |
| --- |
| @SpringBootApplication  //使用的Mybatis,扫描com.demo.mapper  @MapperScan("com.demo.mapper")  public class Springbootdemo5Application {  public static void main(String[] args) {  SpringApplication.run(Springbootdemo5Application.class, args);  }  } |

9 使用Postman测试







# 3 Spring Boot项目部署

需求：将Spring Boot项目使用maven指令打成jar包并运行测试

分析：

需要添加打包组件将项目中的资源、配置、依赖包打到一个jar包中；可以使用maven的package；

部署：java -jar 包名

步骤实现：

（1）添加打包组件

   <build>  
       <plugins>  
          <!-- 打jar包时如果不配置该插件，打出来的jar包没有清单文件 -->  
           <plugin>  
               <groupId>org.springframework.boot</groupId>  
               <artifactId>spring-boot-maven-plugin</artifactId>  
           </plugin>  
       </plugins>  
   </build>

（2）部署运行

java -jar 包名