SpringBoot 第四章

（SpringBoot 整合 SpringMVC+MyBatis）

需求分析：通过使用 SpringBoot+SpringMVC+MyBatis 整合实现一个对数据库中的 users 表的 CRUD 的操作

一、 创建项目

1 修改 pom 文件

```
    <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>1.5.10.RELEASE</version>
        </parent>
        <groupId>com.bjsxt</groupId>
        <artifactId>12-spring-boot-springmvc-mybatis</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <properties>
            <java.version>1.7</java.version>
            <thymeleaf.version>3.0.2.RELEASE</thymeleaf.version>

    <thymeleaf-layout-dialect.version>2.0.4</thymeleaf-layout-dialect.ve
rsion>
        </properties>

        <dependencies>
            <!-- springBoot 的启动器 -->
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
            </dependency>
            <!-- web 启动器 -->
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-thymeleaf</artifactId>
            </dependency>
            <!-- Mybatis 启动器 -->
```

```
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.1.1</version>
        </dependency>
        <!-- mysql 数据库驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- druid 数据库连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.9</version>
        </dependency>
    </dependencies>
</project>
```

2 添加 application.properties 全局配置文件

```
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/ssm
spring.datasource.username=root
spring.datasource.password=root

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

mybatis.type-aliases-package=com.bjsxt.pojo
```

3 数据库表设计

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

二、 添加用户

1 创建实体类

```
public class Users {
    private Integer id;
    private String name;
    private Integer  age;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }

}
```

2 创建 mapper 接口以及映射配置文件

```
import com.bjsxt.pojo.Users;

public interface UsersMapper {

    void insertUser(Users users);
}
```

```
<?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.bjsxt.mapper.UsersMapper">
    <insert id="insertUser" parameterType="users">
        insert into users(name,age) values(#{name},#{age})
    </insert>
</mapper>
```

3 创建业务层

```
@Service
@Transactional
public class UsersServiceImpl implements UsersService {

    @Autowired
    private UsersMapper usersMapper;

    @Override
    public void addUser(Users users) {
        this.usersMapper.insertUser(users);
    }
}
```

4 创建 Controller

```
@Controller
@RequestMapping("/users")
public class UsersController {

    @Autowired
    private UsersService usersService;

    /**
     * 页面跳转
     */
    @RequestMapping("/{page}")
    public String showPage(@PathVariable String page){
        return page;
    }

    /**
```

```
        * 添加用户
        */
       @RequestMapping("/addUser")
       public String addUser(Users users){
           this.usersService.addUser(users);
           return "ok";
       }
   }
```

5 编写页面

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>添加用户</title>
</head>
<body>

    <form th:action="@{/users/addUser}" method="post">
        用户姓名：<input type="text" name="name"/><br/>
        用户年龄：<input type="text" name="age"/><br/>
        <input type="submit" value="确定"/><br/>
    </form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>操作提示页面</title>
</head>
<body>
    操作成功!!!
</body>
</html>
```

6 启动类

```
@SpringBootApplication
@MapperScan("com.bjsxt.mapper") //@MapperScan 用户扫描MyBatis的Mapper
```

```
接口
    public class App {

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

三、 查询用户

1 在 mapper 接口中以及映射配置文件中添加相关代码

```
    List<Users> selectUsersAll();
```

```
    <select id="selectUsersAll" resultType="users">
            select id,name,age from users
        </select>
```

2 在业务层中添加查询方法

```
    @Override
        public List<Users> findUserAll() {
            return this.usersMapper.selectUsersAll();
        }
```

3 在 Controller 中添加方法

```
    /**
         * 查询全部用户
         */
        @RequestMapping("/findUserAll")
        public String findUserAll(Model model){
            List<Users> list = this.usersService.findUserAll();
            model.addAttribute("list", list);
            return "showUsers";
        }
```

4 添加页面

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>展示用户数据</title>
</head>
<body>
    <table border="1" style="width:300px;">
        <tr>
            <th>用户 ID</th>
            <th>用户姓名</th>
            <th>用户年龄</th>
        </tr>
        <tr th:each="user : ${list}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.name}"></td>
            <td th:text="${user.age}"></td>
        </tr>
    </table>
</body>
</html>
```

四、 用户更新

1 更新用户之前的查询，并将数据在页面中回显

1.1 修改 mapper 接口以及映射配置文件

```
Users selectUsersById(Integer id);
```

```
<select id="selectUsersById" resultType="users">
        select id,name,age from users where id = #{value}
    </select>
```

1.2 修改业务层代码

```
@Override
    public Users findUserById(Integer id) {
        return this.usersMapper.selectUsersById(id);
    }
```

1.3 修改 Controller

```
/**
 * 根据用户 id 查询用户
 */
@RequestMapping("/findUserById")
public String findUserById(Integer id,Model model){
    Users user = this.usersService.findUserById(id);
    model.addAttribute("user", user);
    return "updateUser";
}
```

1.4 添加页面 updateUsers.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form th:action="@{/users/editUser}" method="post">
            <input type="hidden" name="id" th:field="${user.id}"/>
            用户姓名：<input type="text" name="name"
th:field="${user.name}"/><br/>
            用户年龄：<input type="text" name="age"
th:field="${user.age}"/><br/>
            <input type="submit" value="确定"/><br/>
    </form>
</body>
</html>
```

1.5 修改 showUsers.html 页面添加操作功能

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>展示用户数据</title>
</head>
<body>
    <table border="1" style="width:300px;">
```

```
            <tr>
                <th>用户 ID</th>
                <th>用户姓名</th>
                <th>用户年龄</th>
                <th>操作</th>
            </tr>
            <tr th:each="user : ${list}">
                <td th:text="${user.id}"></td>
                <td th:text="${user.name}"></td>
                <td th:text="${user.age}"></td>
                <td>
                    <a th:href="@{/users/findUserById(id=${user.id})}">更新用户</a>
                </td>
            </tr>
        </table>
    </body>
    </html>
```

2 用户更新

2.1 修改 mapper 接口以及映射配置文件

```
    void updateUser(Users users);
```

```
    <update id="updateUser" parameterType="users">
            update users set name=#{name} ,age=#{age} where id=#{id}
        </update>
```

2.2 修改业务层代码

```
    @Override
        public void updateUser(Users users) {
            this.usersMapper.updateUser(users);
        }
```

2.3 修改 Controller

```
    /**
         * 更新用户
         */
```

```
    @RequestMapping("/editUser")
    public String editUser(Users users){
        this.usersService.updateUser(users);
        return "ok";
    }
```

五、 删除用户

1 修改 mapper 接口以及映射配置文件

```
void deleteUserById(Integer id);
```

```
<delete id="deleteUserById">
        delete from users where id = #{value}
    </delete>
```

2 修改业务层代码

```
@Override
    public void deleteUserById(Integer id) {
        this.usersMapper.deleteUserById(id);
    }
```

3 修改 Controller

```
/**
     * 删除用户
     */
    @RequestMapping("/delUser")
    public String delUser(Integer id){
        this.usersService.deleteUserById(id);
        return "redirect:/users/findUserAll";
    }
```

4 修改 showUsers.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>展示用户数据</title>
```

```
    </head>
    <body>
        <table border="1" style="width:300px;">
            <tr>
                <th>用户 ID</th>
                <th>用户姓名</th>
                <th>用户年龄</th>
                <th>操作</th>
            </tr>
            <tr th:each="user : ${list}">
                <td th:text="${user.id}"></td>
                <td th:text="${user.name}"></td>
                <td th:text="${user.age}"></td>
                <td>
                    <a th:href="@{/users/findUserById(id=${user.id})}">更
新用户</a>
                    <a th:href="@{/users/delUser(id=${user.id})}">删除用户
</a>
                </td>
            </tr>
        </table>
    </body>
    </html>
```