# 五、SpringBoot与整合其他技术

## 4.1springboot整合jsp

## 导入包

<!--jsp页面使用jstl标签 -->  
<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>jstl</artifactId>  
</dependency>  
  
<!--用于编译jsp -->  
<dependency>  
 <groupId>org.apache.tomcat.embed</groupId>  
 <artifactId>tomcat-embed-jasper</artifactId>  
 <scope>provided</scope>  
</dependency>

## 配置application.properties

spring.mvc.view.prefix=/WEB-INF/views/  
spring.mvc.view.suffix=.jsp

## 在main下新建包，和resources同级

## webapp/WEB-INF/views

## 解决没有新建jsp页面问题

## 

## 5.1 SpringBoot整合Mybatis

### 5.1.1 添加Mybatis的起步依赖

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>2.1.1</version>  
</dependency>

**5.1.2 添加数据库驱动坐标**

<!--spring提供的连接池-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-jdbc</artifactId>  
</dependency>  
<!--数据库驱动-->  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

**注意：springboot默认提供的连接池是HIKariCP，这个比c3p0性能高很多**

**5.1.3 添加数据库连接信息**

在application.properties中添加数据量的连接信息,注意时区问题。

|  |  |
| --- | --- |
| 1 | #DB Configuration: |
| 2 | spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver |
| 3 | spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai |
|  | useUnicode=true&characterEncoding=utf8 |
| 4 | spring.datasource.username=root |
| 5 | spring.datasource.password=123456 |

### 创建user表

在test数据库中创建user表

|  |  |
| --- | --- |
| 1 | -- ---------------------------- |
| 2 | -- Table structure for `user` |
| 3 | -- ---------------------------- |
| 4 | DROP TABLE IF EXISTS `user`; |
| 5 | CREATE TABLE `user` ( |
| 6 | `id` int(11) NOT NULL AUTO\_INCREMENT, |
| 7 | `username` varchar(50) DEFAULT NULL, |
| 8 | `password` varchar(50) DEFAULT NULL, |
| 9 | `name` varchar(50) DEFAULT NULL, |
| 10 | PRIMARY KEY (`id`) |
| 11 | ) ENGINE=InnoDB AUTO\_INCREMENT=10 DEFAULT CHARSET=utf8; |
| 12 |  |
| 13 | -- ---------------------------- |
| 14 | -- Records of user |
| 15 | -- ---------------------------- |
| 16 | INSERT INTO `user` VALUES ('1', 'zhangsan', '123', '张三'); |
| 17 | INSERT INTO `user` VALUES ('2', 'lisi', '123', '李四'); |

### 创建实体Bean

**@Data**

public class User {

// 主键

private Long id;

// 用户名

private String username;

// 密码

private String password;

// 姓名

private String name;

}

* + 1. **编写Mapper**

1. @Mapper
2. public interface UserMapper {
3. public List<User> queryUserList(); 4 }

注意：@Mapper标记该类是一个mybatis的mapper接口，可以被spring boot自动扫描到spring上下文中，也可以这里不配置@Mapper，在主启动类上配置@MapperScan(basePackages = "com.zy.springbootmybatis.mapper")，效果一样的。

### 配置Mapper映射文件

在src\main\resources\mapper路径下加入UserMapper.xml配置文件"

|  |  |
| --- | --- |
| 1 | <?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.itheima.mapper.UserMapper">  <select id="queryUserList" resultType="user"> select \* from user  </select>  </mapper> |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |

### 在application.properties中添加mybatis的信息

|  |  |
| --- | --- |
| 1 | #spring集成Mybatis环境 |
| 2 | #pojo别名扫描包 |
| 3 | mybatis.type-aliases-package=com.itheima.domain |
| 4 | #加载Mybatis映射文件 |
| 5 | mybatis.mapper-locations=classpath:mapper/\*Mapper.xml |

* + 1. **编写测试Controller**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

@Controller

public class MapperController {

@Autowired

private UserMapper userMapper;

@RequestMapping("/queryUser")

@ResponseBody

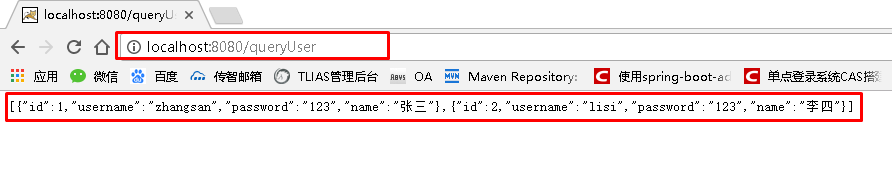
public List<User> queryUser(){

List<User> users = userMapper.queryUserList(); return users;

}

}

**5.1.10 测试**



方式二：Mapper无xml方式：

Mybatis. mapper-locations: classpath:mappering/\*Mapper.xml 就不要了。

Mapper中直接在接口方法上写sql语句。

//@Mapper  
public interface UserMapper {  
 @Select("select \* from user where id=#{id}")  
 User findOneUser(int id);  
}

* 1. **SpringBoot整合Junit**

**5.2.1 添加Junit的起步依赖**

1 <!--测试的起步依赖-->

1. <dependency>
2. <groupId>org.springframework.boot</groupId>
3. <artifactId>spring-boot-starter-test</artifactId>
4. <scope>test</scope>
5. </dependency>

**5.2.2 编写测试类**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

package com.itheima.test;

import com.itheima.MySpringBootApplication;

import com.itheima.domain.User;

import com.itheima.mapper.UserMapper; import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.context.SpringBootTest; import org.springframework.test.context.junit4.SpringRunner;

import java.util.List;

@RunWith(SpringRunner.class)

@SpringBootTest(classes = MySpringBootApplication.class) public class MapperTest {

@Autowired

private UserMapper userMapper;

@Test

public void test() {

List<User> users = userMapper.queryUserList(); System.out.println(users);

}

}

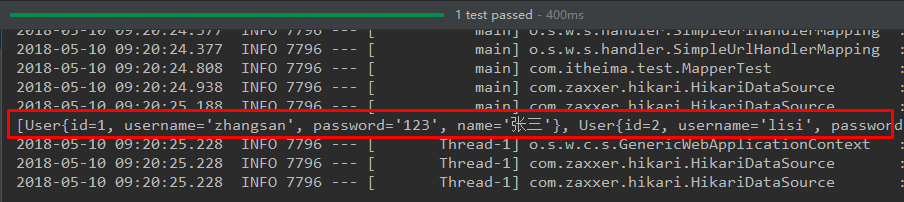
其中，

SpringRunner继承自SpringJUnit4ClassRunner，使用哪一个Spring提供的测试测试引擎都可以

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | public | final | class | SpringRunner | extends | SpringJUnit4ClassRunner |

@SpringBootTest的属性指定的是引导类的字节码对象

### 5.2.3 控制台打印信息



* 1. **SpringBoot整合Spring Data JPA**

**5.3.1 添加Spring Data JPA的起步依赖**

1. <!-- springBoot JPA的起步依赖 -->
2. <dependency>
3. <groupId>org.springframework.boot</groupId>
4. <artifactId>spring-boot-starter-data-jpa</artifactId>
5. </dependency>

**5.3.2 添加数据库驱动依赖**

1 <!-- MySQL连接驱动 -->

1. <dependency>
2. <groupId>mysql</groupId>
3. <artifactId>mysql-connector-java</artifactId>
4. </dependency>

**5.3.3 在application.properties中配置数据库和jpa的相关属性**

|  |  |
| --- | --- |
| 1 | #DB Configuration: |
| 2 | spring.datasource.driverClassName=com.mysql.jdbc.Driver |
| 3 | spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test? |
|  | useUnicode=true&characterEncoding=utf8 |
| 4 | spring.datasource.username=root |
| 5 | spring.datasource.password=root |
| 6 |  |
| 7 | #JPA Configuration: |
| 8 | spring.jpa.database=MySQL |
| 9 | spring.jpa.show-sql=true |
| 10 | spring.jpa.generate-ddl=true |
| 11 | spring.jpa.hibernate.ddl-auto=update |
| 12 | spring.jpa.hibernate.naming\_strategy=org.hibernate.cfg.ImprovedNamingStrategy |

**5.3.4 创建实体配置实体**

1

@Entity

2

3

4

5

6

7

8

9

10

11

12

13

14

15

public class User {

// 主键

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

// 用户名

private

// 密码

private

// 姓名

private

String

username;

String

password;

String

name;

//此处省略setter和getter方法... ...

}

**5.3.5 编写UserRepository**

1. public interface UserRepository extends JpaRepository<User,Long>{
2. public List<User> findAll(); 3 }

**5.3.6 编写测试类**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

@RunWith(SpringRunner.class)

@SpringBootTest(classes=MySpringBootApplication.class) public class JpaTest {

@Autowired

private UserRepository userRepository;

@Test

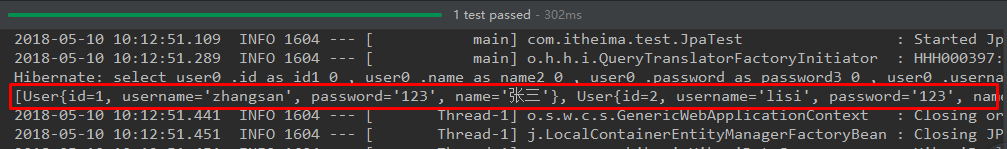
public void test(){

List<User> users = userRepository.findAll(); System.out.println(users);

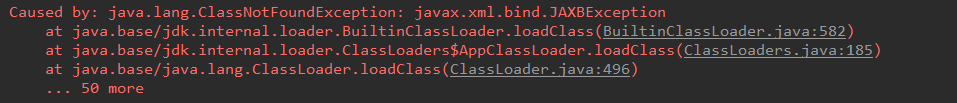
}

}

**5.3.7 控制台打印信息**



注意：如果是jdk9，执行报错如下：



原因：jdk缺少相应的jar

解决方案：手动导入对应的maven坐标，如下：

1 <!--jdk9需要导入如下坐标-->

1. <dependency>
2. <groupId>javax.xml.bind</groupId>
3. <artifactId>jaxb-api</artifactId>
4. <version>2.3.0</version>
5. </dependency>

## 5.4 SpringBoot整合Redis

### 5.4.1 添加redis的起步依赖

1 <!-- 配置使用redis启动器 -->

1. <dependency>
2. <groupId>org.springframework.boot</groupId>
3. <artifactId>spring-boot-starter-data-redis</artifactId>
4. </dependency>

**5.4.2 配置redis的连接信息**

1. #Redis
2. spring.redis.host=127.0.0.1
3. spring.redis.port=6379

**5.4.3 注入RedisTemplate测试redis操作**

1

2

3

4

5

6

7

8

9

10

11

12

13

@RunWith(SpringRunner.class)

@SpringBootTest(classes = SpringbootJpaApplication.class) public class RedisTest {

@Autowired

private UserRepository userRepository;

@Autowired

private RedisTemplate<String, String> redisTemplate;

@Test

public void test() throws JsonProcessingException {

//从redis缓存中获得指定的数据

|  |  |  |  |
| --- | --- | --- | --- |
| 14 | } | } | String userListData = redisTemplate.boundValueOps("user.findAll").get();  //如果redis中没有数据的话 if(null==userListData){  //查询数据库获得数据  List<User> all = userRepository.findAll();  //转换成json格式字符串  ObjectMapper om = new ObjectMapper(); userListData = om.writeValueAsString(all);  //将数据存储到redis中，下次在查询直接从redis中获得数据，不用在查询数据库  redisTemplate.boundValueOps("user.findAll").set(userListData); System.out.println("===============从数据库获得数据===============");  }else{  System.out.println("===============从redis缓存中获得数据===============");  }  System.out.println(userListData); |
| 15 |
| 16 |
| 17 |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |
| 33 |