金现代
JINXIANDAI

专题二

Spring Boot应用

——集成MyBatis、单元测试

目录

01_添加依赖

- 在pom.xml中添加依赖

```
<!--集成mybatis需要的依赖-->
<!--mybatis和SpringBoot整合模块-->
 <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
        <version>2.0.1</version>
</dependency>

<!--数据库驱动-->
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.30</version>
 </dependency>
```

02_配置数据源

- 在application.yml添加数据源及MyBatis相关配置

```
spring:
###数据库四大参数配置，spring会根据四大参数创建数据源########
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/mysql?useUnicode=true
    username: root
    password: mysql
    driver-class-name: com.mysql.jdbc.Driver
  mvc:
    static-path-pattern: /static/**        # 静态资源路径
    view:
      prefix: /WEB-INF/view                # 自定义页面文件路径，
#####mybatis相关配置项,必须配置映射文件的位置####
mybatis:
  mapper-locations: classpath:mapping/*.xml   ##扫描mapper映射文件
  type-aliases-package: com.jxdedu.demo.model  ##扫描实体类
```

03_实体类及业务代码创建

- 创建实体类、dao接口、mapper映射文件，注意double类型数据的处理

```
//实体类
public class Empl {
    private Integer empno;
    private String ename;
    private String job;
    private Double sal;
   //get和set方法
}
```

```
//dao接口
public interface EmplMapper {
    List<Empl> getAll();
}
```

```
    <!--如果结果集中有double类型，需要指明javaType和
jdbcType-->
    <resultMap id="baseMap" type="Empl">
        <id column="empno" property="empno"/>
        <result column="ename" property="ename"/>
        <result column="job" property="job"/>
        <result column="sal" property="sal"
                javaType="double" jdbcType="DOUBLE"/>
    </resultMap>
    <!--List<Empl> getAll();-->
    <select id="getAll" resultMap="baseMap">
        select empno,ename,job,sal from empl
    </select>
```

04_创建Controller

- 创建controller,调用dao层方法

```
@Controller
public class DemoController {
    @Autowired
    private  EmplMapper emplMapper;

    @RequestMapping("/")
    @ResponseBody
    public List<Empl> getAll(){
        return emplMapper.getAll();
    }
}
```

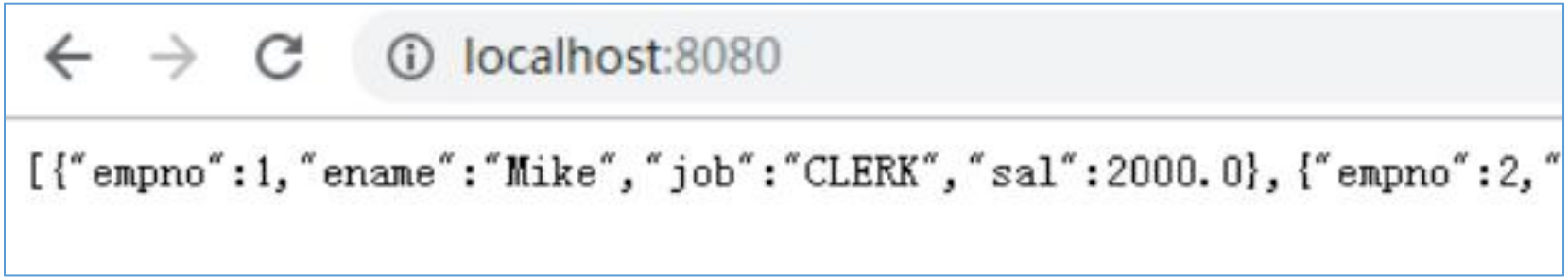
05_主程序配置

- 添加MapperScan注解，扫描dao层接口

```
@SpringBootApplication
//dao接口所在路径
@MapperScan("com.jxdedu.demo.dao")
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class,args);
    }
}
```

06_运行结果

- 启动主程序，访问http://localhost:8080



单元测试简介

- 什么是单元测试
  - 利用一小段代码来测试一个小的、有明确功能的代码是否正确
- 单元测试好处
  - 测试路径路径比较短，易于测试
  - 快速的定位bug，减少bug
  - 提高代码质量
- 测试要求
  - 要求测试访问无返回值、无参数、无静态特征

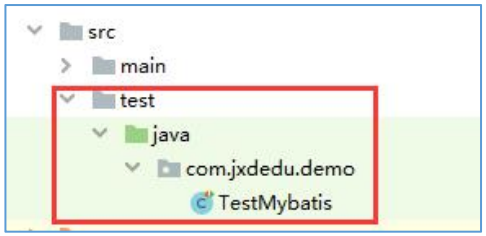
01_添加依赖

- 在pom.xml中添加依赖

```
<!--集成单元测试junit需要的依赖 -->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
 </dependency>
```

02_创建测试类

- 在test--->java目录下创建测试类



- 添加注解
  - **@RunWith(SpringRunner.class) 指定启动器**
  - **@SpringBootTest(classes = DemoApplication.class)**
    - **启动spring容器，参数为主程序的字节码**

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class TestMybatis {
    ……
}
```

03_创建测试方法

- 01_注入要测试的类，如注入dao层组件，我们可以测试增删改查方法
- 02_创建测试方法
  - 必须使用@Test注解

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class TestMybatis {
    //注入要测试的类
    @Autowired
    private EmplMapper emplMapper;    //代码中的红线忽略就可以,这是编译问题，这个时候检测不到这个组件
    @Test
    public  void test(){
        System.out.println(emplMapper.getAll().size());
    }
}
```

04_运行测试方法

- 点击测试方法前的运行图标进行测试，在控制台查看运行结果
- 这样在不启动主程序的情况下，就可以单独测试底层的增删改查方法