**SpringBoot整合MyBatis操作数据库**

**MyBatis 简介**

MyBatis 的前身是 Apache 社区的一个开源项目 iBatis，于 2010 年更名为 MyBatis。MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集，使得开发人员更加关注 SQL 本身和业务逻辑，不用再去花费时间关注整个复杂的 JDBC 操作过程。

MyBatis 的优点如下：

* 封装了 JDBC 大部分操作，减少开发人员工作量；
* 相比一些自动化的 ORM 框架，“半自动化”使得开发人员可以自由的编写 SQL 语句，灵活度更高；
* Java 代码与 SQL 语句分离，降低维护难度；
* 自动映射结果集，减少重复的编码工作；
* 开源社区十分活跃，文档齐全，学习成本不高。

**mybatis-springboot-starter 介绍**

Spring Boot 的核心特性包括简化配置并快速开发，当我们需要整合某一个功能时，只需要引入其特定的场景启动器 ( starter ) 即可，比如 web 模块整合、jdbc 模块整合，我们在开发时只需要在 pom.xml 文件中引入对应的场景依赖即可。Spring 官方并没有提供 MyBatis 的场景启动器，但是 MyBatis 官方却紧紧的抱住了 Spring 的大腿，他们提供了 MyBatis 整合 Spring Boot 项目时的场景启动器，也就是 **mybatis-springboot-starter**，大家通过命名方式也能够发现其中的区别，Spring 官方提供的启动器的命名方式为 **spring-boot-starter-\***，与它还是有一些差别的，接下来我们来介绍一下 **mybatis-springboot-starter** 场景启动器。

其官网地址为 [mybatis-spring-boot](http://www.mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/index.html" \t "_blank)，官网对 **mybatis-springboot-starter** 的介绍如下所示：

The MyBatis-Spring-Boot-Starter help you build quickly MyBatis applications on top of the Spring Boot.

MyBatis-Spring-Boot-Starter 可以帮助开发者快速创建基于 Spring Boot 的 MyBatis 应用程序，那么使用 **MyBatis-Spring-Boot-Starter**可以做什么呢？

* 构建独立的 MyBatis 应用程序
* 零模板
* 更少的 XML 配置代码甚至无 XML 配置

**Spring Boot 整合 MyBatis 过程**

项目结构目录如下：

lou-spring-boot

├── pom.xml

├── README.md

└── src

├── main

│ ├── java

│ │ └── com

│ │ └── lou

│ │ └── springboot

│ │ ├── Application.java

│ │ ├── controller

│ │ │ ├── HelloController.java

│ │ │ ├── JdbcController.java

│ │ │ └── MyBatisController.java

│ │ ├── dao

│ │ │ └── UserDao.java

│ │ └── entity

│ │ └── User.java

│ └── resources

│ ├── application.properties

│ ├── mapper

│ │ └── UserDao.xml

│ └── templates

└── test

└── java

└── com

└── lou

└── springboot

└── ApplicationTests.java

**添加依赖**

*<*dependency*>  
 <*groupId*>*org.mybatis.spring.boot*</*groupId*>  
 <*artifactId*>*mybatis-spring-boot-starter*</*artifactId*>  
 <*version*>*1.3.2*</*version*>  
</*dependency*>*

**application.yaml配置**

Spring Boot 整合 MyBatis 时几个比较需要注意的配置参数：

* **mybatis.config-location**

配置 mybatis-config.xml 路径，mybatis-config.xml 中配置 MyBatis 基础属性，如果项目中配置了 mybatis-config.xml 文件需要设置该参数

* **mybatis.mapper-locations**

配置 Mapper 文件对应的 XML 文件路径

* **mybatis.type-aliases-package**

配置项目中实体类包路径

**启动类增加 Mapper 扫描**

在启动类中添加对 Mapper 包扫描 @MapperScan，Spring Boot 启动的时候会自动加载包路径下的 Mapper 接口：

@SpringBootApplication

@MapperScan("com.lou.springboot.dao") //添加 @Mapper 注解

public class Application {

public static void main(String[] args) {

System.out.println("启动 Spring Boot...");

SpringApplication.run(Application.class, args);

}

}

当然也可以直接在每个 Mapper 接口上面添加 @Mapper 注解，但是如果 Mapper 接口数量较多，在每个 Mapper 加注解是挺繁琐的，建议使用扫描注解

**Spring Boot 整合 MyBatis 实例**

**新建实体类和 Mapper 接口**

在 entity 包下新建 User 类，将bg\_users中的字段映射到该实体类中：

public class User *{* private Integer id;  
 private String name;  
 private String password;  
  
 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 String getPassword*() {* return password;  
 *}* public void setPassword*(*String password*) {* this.password = password;  
 *}  
}*

在 dao 包中新建 UserDao 接口，并定义增删改查四个接口：

public interface UserDao *{* //findall  
 public List*<*User*>* findAll*()*;  
  
 //增加  
 public boolean insertUser*(*User user*)*;  
   
 //删除  
 public boolean deleteUser*(*Integer id*)*;  
   
 //修改  
 public boolean updUser*(*User user*)*;  
  
*}*

**创建 Mapper 接口的映射文件**

在 resources/mapper 目录下新建 Mapper 接口的映射文件 UserDao.xml，之后进行映射文件的编写。

1. 首先，定义映射文件与 Mapper 接口的对应关系，比如该示例中，需要将 UserDao.xml 的与对应的 UserDao 接口类之间的关系定义出来：

<mapper namespace="com.springbootjdbc.dao.UserDao">

1. 之后，配置表结构和实体类的对应关系：

<resultMaptype="com.springbootjdbc.entity.User"id="UserResult">

1. <result property="id" column="id"/>
2. <result property="name" column="name"/>
3. <result property="password" column="password"/>

</resultMap>

1. 最后，针对对应的接口方法，编写具体的 SQL 语句，最终的 UserDao.xml 文件如下：
2. <?xml version="1.0" encoding="UTF-8"?>
3. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4. <mapper namespace="com.lou.springboot.dao.UserDao">
5. <resultMap type="com.lou.springboot.entity.User" id="UserResult">
6. <result property="id" column="id"/>
7. <result property="name" column="name"/>
8. <result property="password" column="password"/>
9. </resultMap>
10. <select id="findAllUsers" resultMap="UserResult">
11. select id,name,password from tb\_user
12. order by id desc
13. </select>
14. <insert id="insertUser" parameterType="com.lou.springboot.entity.User">
15. insert into tb\_user(name,password)
16. values(#{name},#{password})
17. </insert>
18. <update id="updUser" parameterType="com.lou.springboot.entity.User">
19. update tb\_user
20. set
21. name=#{name},password=#{password}
22. where id=#{id}
23. </update>
24. <delete id="delUser" parameterType="int">
25. delete from tb\_user where id=#{id}
26. </delete>

</mapper>

**新建 MyBatisController**

为了对 MyBatis 进行功能测试，在 controller 包下新建 MyBatisController 类，并新增 4 个方法分别接收对于 tb\_user 表的增删改查请求，代码如下：  
@RestController  
public class MyBatisController *{* @Resource  
 UserDao userDao;  
   
 @GetMapping*(*"/users/mybatis/queryAll"*)* public List*<*User*>* queryAll*()  
 {* return userDao.findAll*()*;  
 *}* @GetMapping*(*"/users/mybatis/insert"*)* public boolean insert*(*String name,String password*)  
 {* if *(*StringUtils.*isEmpty(*name*)*||StringUtils.*isEmpty(*password*))* return false;  
 User user = new User*()*;  
 user.setName*(*name*)*;  
 user.setPassword*(*password*)*;  
 return userDao.insertUser*(*user*)*;  
 *}* @GetMapping*(*"/users/mybatis/delete"*)* public boolean delete*(*Integer id*)  
 {* if *(*id==null||id<1*)* return false;  
 return userDao.deleteUser*(*id*)*;  
 *}* @GetMapping*(*"/users/mybatis/alter"*)* public boolean alter*(*Integer id,String name,String password*)  
 {* if *(*id==null||id<1||StringUtils.*isEmpty(*name*)*||StringUtils.*isEmpty(*password*))* return false;  
 User user = new User*()*;  
 user.setId*(*id*)*;  
 user.setName*(*name*)*;  
 user.setPassword*(*password*)*;  
 return userDao.updUser*(*user*)*;  
 *}  
}*

**测试**