# **SpringBootDemo**

## **新建SpringBoot项目**

使用idea New Project -> Spring Initializr 在 Developer Tools 中选中 Web 下的Spring Web

## **改变默认端口**

打开 application.properties 添加配置 server.port=8092

**开始debug模式**

打开 application.properties 添加配置 debug=true

## **输出helloworld**

新建Controller package ，新建HelloController.java

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/hello")

public class HelloController {

@RequestMapping("/sayhello")

public String hello() {

return "Hello World";

}

}

输入 <http://localhost:8092/hello/sayhello> 访问

## **输出json**

新建 pojo package,并新加 User.java

public class User {

private int id;

private String name;

private String pwd;

public User() {

}

public User(int id, String name, String age) {

this.id = id;

this.name = name;

this.pwd = age;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPwd() {

return pwd;

}

public void setPwd(String pwd) {

this.pwd = pwd;

}

@Override

public String toString() {

return "User{" +

"id=" + id +

", name='" + name + '\'' +

", pwd=" + pwd +

'}';

}

}

在 HelloController.java 中添加如下方法

@RequestMapping("/hellojson")

@ResponseBody

public User hellojson() {

User curUser = new User(33,"合肥","bb");

return curUser ;

}

输入 <http://localhost:8092/hello/hellojson> 访问

## **部署为windows服务**

将springboot项目打包成 jar (如springboot-demo1-0.0.1-SNAPSHOT.jar)

在 <https://github.com/kohsuke/winsw/releases> 下载 WinSW.NET4.exe

将 WinSW.NET4.exe 重命名为你想要部署的服务的名称.exe(如 springbootdemo1.exe)

新建 springbootdemo1.xml

<service>

<id>springbootdemo1</id>

<name>springbootdemo1</name>

<description>This is springbootdemo1 service.</description>

<!-- java环境变量 -->

<env name="JAVA\_HOME" value="%JAVA\_HOME%"/>

<executable>java</executable>

<arguments>-jar "C:\springbootservice\springboot-demo1-0.0.1-SNAPSHOT.jar"</arguments>

<!-- 开机启动 -->

<startmode>Automatic</startmode>

<!-- 日志配置 -->

<logpath>%BASE%\log</logpath>

<logmode>rotate</logmode>

</service>

将 exe xml jar 三个文件放到同目录下

执行springbootdemo1.exe install 来安装服务（需要管理员权限）

执行springbootdemo1.exe uninstall 来卸载服务（需要管理员权限）

## **yaml配置直接给实体类直接赋值**

新建 Dog.java Person.java 两个实体类，用于接收yaml配置

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class Dog {

@Value("旺财")

private String name;

@Value("3")

private Integer age;

public Dog() {

}

public Dog(String name, Integer age) {

this.name = name;

this.age = age;

}

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;

}

@Override

public String toString() {

return "Dog{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

}

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.List;

import java.util.Map;

@Component

@ConfigurationProperties(prefix = "person")

///绑定配置文件

public class Person {

private String name;

private Integer age;

private Boolean happy;

private Date birth;

private Map<String,Object> maps;

private List<Object> lists;

private Dog dog;

public Person() {

}

public Person(String name, Integer age, Boolean happy, Date birth, Map<String, Object> maps, List<Object> lists, Dog dog) {

this.name = name;

this.age = age;

this.happy = happy;

this.birth = birth;

this.maps = maps;

this.lists = lists;

this.dog = dog;

}

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;

}

public Boolean getHappy() {

return happy;

}

public void setHappy(Boolean happy) {

this.happy = happy;

}

public Date getBirth() {

return birth;

}

public void setBirth(Date birth) {

this.birth = birth;

}

public Map<String, Object> getMaps() {

return maps;

}

public void setMaps(Map<String, Object> maps) {

this.maps = maps;

}

public List<Object> getLists() {

return lists;

}

public void setLists(List<Object> lists) {

this.lists = lists;

}

public Dog getDog() {

return dog;

}

public void setDog(Dog dog) {

this.dog = dog;

}

@Override

public String toString() {

return "Person{" +

"name='" + name + '\'' +

", age=" + age +

", happy=" + happy +

", birth=" + birth +

", maps=" + maps +

", lists=" + lists +

", dog=" + dog +

'}';

}

}

新建 application.yaml 文件来写入配置

person:

name: adc

age: 33

happy: false

birth: 2019/11/02

maps: {k1: v1,k2: v2}

lists: [code1,code2,code3]

dog:

name: 哈哈

age: 3

当需要读取配置的时候，只需要在任意地方定义变量

@Autowired

private Person person;

此时 person 变量会自动被赋值

在HelloController.java 中添加如下函数

@RequestMapping("/person")

@ResponseBody

public Person person() {

return person ;

}

输入 <http://localhost:8092/hello/person> 得到配置值的输出

## **整合自己的自动装配类（AutoConfig）**

参考 <https://www.cnblogs.com/hellokuangshen/p/12486631.html>

## **整合jdbc**

建表

CREATE TABLE `user` (

`id` int(11) NOT NULL,

`name` varchar(45) DEFAULT NULL,

`pwd` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

添加依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

在yaml中添加配置

spring:

datasource:

username: xl

password: 1010

url: jdbc:mysql://localhost:3306/mybatis?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8

driver-class-name: com.mysql.cj.jdbc.Driver

新建JdbcController.java

@RestController

@RequestMapping("/jdbc")

public class JdbcController {

@Autowired

DataSource dataSource;

@RequestMapping("/testConnection")

@ResponseBody

public void testJdbc() {

System.out.println(dataSource.getClass());

//获得连接

Connection connection = null;

try {

connection = dataSource.getConnection();

System.out.println("test connection" + connection);

//关闭连接

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

输入 <http://localhost:8092/jdbc/testConnection>进行验证

## **使用JdbcTemplate**

在 JdbcController.java 中添加

@Autowired

JdbcTemplate jdbcTemplate;

@GetMapping("/list")

public List<Map<String, Object>> userList(){

String sql = "select \* from user";

List<Map<String, Object>> maps = jdbcTemplate.queryForList(sql);

return maps;

}

@GetMapping("/add")

public String addUser(){

String sql = "insert into user(id, name,pwd)" +

" values ('44','24736743@qq.com','dfdfd')";

jdbcTemplate.update(sql);

//查询

return "addOk";

}

//修改用户信息

@GetMapping("/update/{id}")

public String updateUser(@PathVariable("id") int id){

String sql = "update user set name=?,pwd=? where id="+id;

Object[] objects = new Object[2];

objects[0] = "得分333";

objects[1] = "233";

jdbcTemplate.update(sql,objects);

return "updateOk";

}

//删除用户

@GetMapping("/delete/{id}")

public String delUser(@PathVariable("id") int id){

String sql = "delete from user where id=?";

jdbcTemplate.update(sql,id);

return "deleteOk";

}

输入 http://localhost:8092/jdbc/list

http://localhost:8092/jdbc/add

http://localhost:8092/jdbc/update/44

<http://localhost:8092/jdbc/delete/44> 验证

## **集成 Druid 数据源，如何实现日志和数据库监控**

加入依赖

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.21</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

修改yaml文件在spring: datasource: 下加入

type: com.alibaba.druid.pool.DruidDataSource # 自定义数据源

可以在yaml文件中添加 Druid 的额外配置

#Spring Boot 默认是不注入这些属性值的，需要自己绑定

#druid 数据源专有配置

initialSize: 5

minIdle: 5

maxActive: 20

maxWait: 60000

timeBetweenEvictionRunsMillis: 60000

minEvictableIdleTimeMillis: 300000

validationQuery: SELECT 1 FROM DUAL

testWhileIdle: true

testOnBorrow: false

testOnReturn: false

poolPreparedStatements: true

加入Druid 的配置类 DruidConfig.java

import com.alibaba.druid.pool.DruidDataSource;

import com.alibaba.druid.support.http.StatViewServlet;

import com.alibaba.druid.support.http.WebStatFilter;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.boot.web.servlet.FilterRegistrationBean;

import org.springframework.boot.web.servlet.ServletRegistrationBean;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

@Configuration

public class DruidConfig {

/\*

将自定义的 Druid数据源添加到容器中，不再让 Spring Boot 自动创建

绑定全局配置文件中的 druid 数据源属性到 com.alibaba.druid.pool.DruidDataSource从而让它们生效

@ConfigurationProperties(prefix = "spring.datasource")：作用就是将 全局配置文件中

前缀为 spring.datasource的属性值注入到 com.alibaba.druid.pool.DruidDataSource 的同名参数中

\*/

@ConfigurationProperties(prefix = "spring.datasource")

@Bean

public DataSource druidDataSource() {

return new DruidDataSource();

}

//配置 Druid 监控管理后台的Servlet；

//内置 Servlet 容器时没有web.xml文件，所以使用 Spring Boot 的注册 Servlet 方式

@Bean

public ServletRegistrationBean statViewServlet() {

ServletRegistrationBean bean = new ServletRegistrationBean(new StatViewServlet(), "/druid/\*");

// 这些参数可以在 com.alibaba.druid.support.http.StatViewServlet

// 的父类 com.alibaba.druid.support.http.ResourceServlet 中找到

Map<String, String> initParams = new HashMap<>();

initParams.put("loginUsername", "admin"); //后台管理界面的登录账号

initParams.put("loginPassword", "123456"); //后台管理界面的登录密码

//后台允许谁可以访问

//initParams.put("allow", "localhost")：表示只有本机可以访问

//initParams.put("allow", "")：为空或者为null时，表示允许所有访问

initParams.put("allow", "");

//deny：Druid 后台拒绝谁访问

//initParams.put("kuangshen", "192.168.1.20");表示禁止此ip访问

//设置初始化参数

bean.setInitParameters(initParams);

return bean;

}

//配置 Druid 监控 之 web 监控的 filter

//WebStatFilter：用于配置Web和Druid数据源之间的管理关联监控统计

@Bean

public FilterRegistrationBean webStatFilter() {

FilterRegistrationBean bean = new FilterRegistrationBean();

bean.setFilter(new WebStatFilter());

//exclusions：设置哪些请求进行过滤排除掉，从而不进行统计

Map<String, String> initParams = new HashMap<>();

initParams.put("exclusions", "\*.js,\*.css,/druid/\*,/jdbc/\*");

bean.setInitParameters(initParams);

//"/\*" 表示过滤所有请求

bean.setUrlPatterns(Arrays.asList("/\*"));

return bean;

}

}

实现了读取yaml配置

在JdbcController.java中添加

@RequestMapping("/testDruidConnection")

@ResponseBody

public void testDruidConnection() throws SQLException {

//看一下默认数据源

System.out.println(dataSource.getClass());

//获得连接

Connection connection = dataSource.getConnection();

System.out.println(connection);

DruidDataSource druidDataSource = (DruidDataSource) dataSource;

System.out.println("druidDataSource 数据源最大连接数：" + druidDataSource.getMaxActive());

System.out.println("druidDataSource 数据源初始化连接数：" + druidDataSource.getInitialSize());

//关闭连接

connection.close();

}

实现输出Druid 数据源的相关匹配值值

监控管理界面的配置

输入 <http://localhost:8092/druid> 进入页面

**整合MyBatis**

添加依赖

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.1.1</version>

</dependency>

配置yaml文件中的数据库链接（内容和上一节一样）

加入

mybatis:

mapper-locations: classpath:\*.xml

此配置为扫描 mapper文件的地址

新建实体（使用至之前已存在的User）

public class User {

private int id;

private String name;

private String pwd;

public User() {

}

public User(int id, String name, String age) {

this.id = id;

this.name = name;

this.pwd = age;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPwd() {

return pwd;

}

public void setPwd(String pwd) {

this.pwd = pwd;

}

@Override

public String toString() {

return "User{" +

"id=" + id +

", name='" + name + '\'' +

", pwd=" + pwd +

'}';

}

}

建立Mapper接口 UserMapper.java

import com.xl.springbootdemo1.pojo.User;

import org.apache.ibatis.annotations.Mapper;

import org.springframework.stereotype.Repository;

import java.util.List;

@Mapper

@Repository

public interface UserMapper {

List<User> getUserList();

User getUserById(int id);

int addUser(User user);

int updateUser(User user);

int deleteUser(int id);

}

在 resources目录下添加 映射文件 UserMapper.xml

<?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.xl.springbootdemo1.Mapper.UserMapper">

<select id="getUserList" resultType="com.xl.springbootdemo1.pojo.User">

select \* from mybatis.user

</select>

<select id="getUserById" parameterType="int" resultType="com.xl.springbootdemo1.pojo.User">

select \* from mybatis.user where id = #{id}

</select>

<insert id="addUser" parameterType="com.xl.springbootdemo1.pojo.User">

insert into mybatis.user (id,name,pwd) values (#{id},#{name},#{pwd});

</insert>

<update id="updateUser" parameterType="com.xl.springbootdemo1.pojo.User">

update mybatis.user set name = #{name} , pwd = #{pwd} where id = #{id} ;

</update>

<update id="deleteUser" parameterType="int">

delete from mybatis.user where id = #{id} ;

</update>

</mapper>

新建Controller MyBatisController.java

import com.xl.springbootdemo1.Mapper.UserMapper;

import com.xl.springbootdemo1.pojo.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController

@RequestMapping("/mybatis")

public class MyBatisController {

@Autowired

UserMapper departmentMapper;

@RequestMapping("/GetUserList")

public List<User> getUserList() {

return departmentMapper.getUserList();

}

@RequestMapping("/GetUserById")

public User GetUserById(int id) {

return departmentMapper.getUserById(id);

}

@RequestMapping("/AddUser")

public int AddUser(User user) {

return departmentMapper.addUser(user);

}

@RequestMapping("/UpdateUser")

public int updateUser(User user) {

return departmentMapper.updateUser(user);

}

@RequestMapping("/DeleteUser")

public int deleteUser(int id) {

return departmentMapper.deleteUser(id);

}

}

输入 http://localhost:8092/mybatis/GetUserById?id=3

http://localhost:8092/mybatis/GetUserList

http://localhost:8092/mybatis/AddUser?id=3335&name=abde&pwd=adfd

http://localhost:8092/mybatis/UpdateUser?id=3335&name=abde&pwd=adfdaaa

http://localhost:8092/mybatis/DeleteUser?id=3335

进行增删查改

可以使用上文的 <http://localhost:8092/druid> 查看数据库连接情况

**分页查询**

添加依赖

<!-- pagehelper -->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper-spring-boot-starter</artifactId>

<version>1.2.5</version>

</dependency>

在yaml文件中添加配置

pagehelper:

helperDialect: mysql

reasonable: true

supportMethodsArguments: true

params: count=countSql

在Controller中添加方法 实现分页查询

@RequestMapping("/GetUserListPage")

public List<User> getUserListPage(@RequestParam(defaultValue = "1") int pageNo, @RequestParam(defaultValue = "2") int pageSize) {

PageHelper.startPage(pageNo,pageSize);

return departmentMapper.getUserList();

}

输入 [http://localhost:8092/mybatis/GetUserListPage](http://localhost:8092/mybatis/GetUserListPage?pageNo=3)

<http://localhost:8092/mybatis/GetUserListPage?pageNo=3>

<http://localhost:8092/mybatis/GetUserListPage?pageNo=3&pageSize=3>

来查看结果

**执行定时任务**

在springboot 入口类上 加 @EnableScheduling

添加 service package，并新建 ScheduledService.java 类

@Service

public class ScheduledService {

//秒 分 时 日 月 星期

@Scheduled(cron = "0/10 \* \* \* \* ?")

public void hello(){

System.out.println("你被执行了" );

}

}

表示 每10秒钟 执行一次任务

具体的cron 表达式 ，可以搜cron 表达式在线生成器

**执行异步任务**

在springboot 入口类上 加 @EnableAsync

添加 service package，并新建 AsyncService.java 类

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Service;

@Service

public class AsyncService {

@Async

public void hello()

{

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("处理完成。。。。。");

}

}

在 HelloController中添加

@Autowired

private AsyncService curService;

@RequestMapping("/async")

@ResponseBody

public String asyncTest()

{

curService.hello();

System.out.println("返回");

return "返回";

}

输入 <http://localhost:8092/hello/async> 可以看到先显示返回，再打印出 处理完成

**集成jsp**

添加依赖

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

在yaml配置文件中添加配置

mvc:

view:

prefix: /WEB-INF/jsp/

suffix: .jsp

在src/main下创建webapp/WEB-INF/jsp文件夹。

在jsp文件夹下创建index.jsp

新建 index.jsp

<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-type" content="text/html; charset=UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>

<title>首页</title>

</head>

<body>

hello ${userName}

</body>

</html>

新建 IndexController.java

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class IndexController {

@GetMapping("/index")

public ModelAndView index() {

ModelAndView view = new ModelAndView("index");

view.addObject("userName", "index");

return view;

}

//真实访问地址 : 项目名/HelloController/hello

@RequestMapping("/index1")

public String sayHello(Model model){

//向模型中添加属性msg与值，可以在JSP页面中取出并渲染

model.addAttribute("userName","index1");

//web-inf/jsp/hello.jsp

return "index";

}

}

输入 <http://localhost:8092/index>

<http://localhost:8092/index>1 查看结果

文件上传

确保支持jsp

在yaml文件中添加上传相关配置

servlet:

multipart:

max-request-size: 100MB

max-file-size: 10MB

在webapp/WEB-INF/jsp目录下新建 支持单文件上传的 upload.jsp

<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-type" content="text/html; charset=UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>

<title>单文件上传</title>

</head>

<body>

<form method="post" action="/upload" enctype="multipart/form-data">

<input type="file" name="file"><br>

<input type="submit" value="提交">

</form>

</body>

</html>

在webapp/WEB-INF/jsp目录下新建支持多文件上传的 multiUpload.jsp

<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-type" content="text/html; charset=UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>

<title>多文件上传</title>

</head>

<body>

<form method="post" action="/multiUpload" enctype="multipart/form-data">

<input type="file" name="file"><br>

<input type="file" name="file"><br>

<input type="file" name="file"><br>

<input type="submit" value="提交">

</form>

</body>

</html>

新建 Controller UploadController.java

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.multipart.MultipartFile;

import org.springframework.web.multipart.MultipartHttpServletRequest;

import javax.servlet.http.HttpServletRequest;

import java.io.File;

import java.io.IOException;

import java.util.List;

@Controller

public class UploadController {

@GetMapping("/upload")

public String upload() {

return "upload";

}

@PostMapping("/upload")

@ResponseBody

public String upload(@RequestParam("file") MultipartFile file) {

if (file.isEmpty()) {

return "上传失败，请选择文件";

}

String fileName = file.getOriginalFilename();

String filePath = "/";

File dest = new File(filePath + fileName);

try {

file.transferTo(dest);

return "上传成功";

} catch (IOException e) {

}

return "上传失败！";

}

@GetMapping("/multiUpload")

public String multiUpload() {

return "multiUpload";

}

@PostMapping("/multiUpload")

@ResponseBody

public String multiUpload(HttpServletRequest request) {

List<MultipartFile> files = ((MultipartHttpServletRequest) request).getFiles("file");

String filePath = "/";

for (int i = 0; i < files.size(); i++) {

MultipartFile file = files.get(i);

if (file.isEmpty()) {

continue;

}

String fileName = file.getOriginalFilename();

File dest = new File(filePath + fileName);

try {

file.transferTo(dest);

} catch (IOException e) {

return "上传第" + (i++) + "个文件失败";

}

}

return "上传成功";

}

}

输入[http://localhost:8092/upload](http://localhost:8092/multiUpload)

<http://localhost:8092/multiUpload> 查看上传结果

通过修改保存逻辑来进行具体文件的保存

**多环境配置**

新建application-dev.yaml 和 application-prod.yaml 文件

在其中 将原来application.yaml 的内容copy到其中

在原 application.yaml 上加入

spring:

profiles:

active: test

表示启用application-dev.yaml 的配置

**多MyBatisPlus的使用**

新建 springboot 项目

建立表，插入数据

CREATE TABLE `userplus1` (

`id` bigint(20) PRIMARY KEY NOT NULL COMMENT '主键ID',

`name` varchar(30) DEFAULT NULL COMMENT '姓名',

`age` int(11) DEFAULT NULL COMMENT '年龄',

`email` varchar(50) DEFAULT NULL COMMENT '邮箱',

`manager\_id` bigint(20) DEFAULT NULL COMMENT '上级ID',

`create\_time` datetime DEFAULT NULL COMMENT '建立时间',

constraint manager\_fk foreign key(manager\_id)

references userplus1(id)

) ENGINE=InnoDB DEFAULT CHARSET=UTF8;

INSERT INTO userplus1 (id, name, age, email,manager\_id,create\_time) VALUES

(1, 'Jone', 18, 'test1@baomidou.com',1,'2020-01-01'),

(2, 'Jack', 20, 'test2@baomidou.com',1,'2020-01-01'),

(3, 'Tom', 28, 'test3@baomidou.com',2,'2020-01-01'),

(4, 'Sandy', 21, 'test4@baomidou.com',2,'2020-01-01'),

(5, 'Billie', 24, 'test5@baomidou.com',2,'2020-01-01');

添加依赖

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

添加 lombok 插件

Settings-Plugins-> 输入框中输入lombok,找到lombok plugins插件,然后点击Intalls安装

配置数据源

在yaml文件中添加配置

配置数据源

spring:

datasource:

username: xl

password: 1010

url: jdbc:mysql://localhost:3306/mybatis?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8

driver-class-name: com.mysql.cj.jdbc.Driver

添加映射文件 xml的路径

mybatis:

mapper-locations: classpath:\*.xml

添加实体

import lombok.Data;

import java.time.LocalDateTime;

@Data

public class userplus1 {

private Long id;

private String name;

private Integer age;

private String email;

private Long managerId;

private LocalDateTime createTime;

}

新建package 在package下添加UserPlusMapper.java

import com.baomidou.mybatisplus.core.mapper.BaseMapper;

import com.xl.mybatisplusdemo.model.userplus1;

public interface UserPlus1Mapper extends BaseMapper<userplus1> {

}

在启动类上加注解，指向上步新建的package

@MapperScan("com.xl.mybatisplusdemo.dao")

新建Controller

import com.xl.mybatisplusdemo.dao.UserPlus1Mapper;

import com.xl.mybatisplusdemo.model.userplus1;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController

@RequestMapping("/mybatisplus")

public class UserPlus1Controller {

@Autowired

UserPlus1Mapper userMapper;

@RequestMapping("/GetUserList")

public List<userplus1> getUserList() {

return userMapper.selectList(null);

}

}

输入 <http://localhost:8013/mybatisplus/GetUserList> 进行访问

注解

在实体类上添加 @TableName("userplus1"), 来指定映射的表

在实体的字段上添加 @TableId 指定当前属性对应的是主键

在实体的字段上添加 @TableField("name1") 指定当前属性对应的表的字段名

在实体的字段上添加 @TableField(exist = false) 指定当前属性不映射到表