**第一个springboot项目**

1. 打开https://start.spring.io/
2. 填写相关的项目信息，下载生成的压缩包
3. 使用idea打开，会根据pom.xml自动下载依赖包
4. 引入web项目相关依赖 ，复制以下内容到 pom.xml中

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId></dependency>

<dependency>

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

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

<scope>test</scope></dependency>

引入后，idea会自动下载对应依赖包（如果没有自动下载，可以尝试重新启动idea）

1. 新建package ,名称为controller, 在包下新建类 HelloWorldController.java，复制如下代码

@RestController

@RequestMapping("/hello")

public class HelloWorldController {

@RequestMapping("/say")

public String say(){

return "Hello World";

}

}

1. 修改 web服务对应的端口

在 resources\application.properties 添加 server.port=8999

1. 以FirstSampleApplication 下的main函数作为项目的入口，启动项目
2. 在浏览器中输入 <http://localhost:8999/hello/say> 来验证项目是否成功

参考：https://www.cnblogs.com/fishpro/p/spring-boot-study-helloworld.html

**集成mysql**

1. pom中添加mysql 相关依赖

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<!--添加JDBC依赖-->

<dependency>

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

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

</dependency>

1. Properties 文件中添加 数据库连接配置

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://192.168.125.200:3306/testdb?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8

spring.datasource.username=root

spring.datasource.password=1010

并在对应的数据库服务器中新建 testdb 数据库，并新建表

CREATE TABLE `student` (

`id` int NOT NULL,

`name` varchar(45) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`age` int DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

1. 新建访问数据库的 service接口和实现

新建 包service，在其下新建imp 子包

在 service 下新建 IUserService 接口，

public interface IUserService {

public List<Map<String,Object>> getUserName();

}

在 imp 子包下新建 UserServiceImp 实现上面的接口

@Service

public class UserServiceImp implements IUserService {

@Autowired

private JdbcTemplate jdbcTemplate;

@Override

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

List<Map<String,Object>> result =jdbcTemplate.queryForList("select \* from student");

return result;

}

}

1. 在controller中添加 IUserService的引用

@Autowired

IUserService service;

5.新建方法

@RequestMapping("/queryUserName")

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

return service.getUserName();

}

6.通过 <http://localhost:8999/hello/queryUserName> 访问

参考 https://blog.csdn.net/ymeddmn/article/details/95992463

**集成mongodb**

1.添加依赖

<dependency>

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

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

2.properties文件中添加连接的配置 (前提是对应的用户在对应的数据库里有权限)

spring.data.mongodb.uri=mongodb://xl1:1010@192.168.125.200:27017/testdb

并通过客户端工具在mongodb服务器上建立表和库

db.testtable.insert({"id":1,"name":"aaa","age":10}) 标准

1. 在上一节中的service的实现中

通过

@Autowired

private MongoTemplate mongoTemplate;

注入 MongoTemplate

1. 在上一节中的service的接口和实现中添加

@Override

public String queryMongodb()

{

Query query = new Query(Criteria.where("id").is(1));

// 查询一条满足条件的数据

Map result = mongoTemplate.findOne(query, Map.class, "testtable");

return result.toString();

}

1. 在上一节的controller中添加 方法

@RequestMapping("/queryMongodb")

public String queryMongodb() {

return service.queryMongodb();

}

通过 <http://localhost:8999/hello/queryMongodb> 访问数据，返回结果

**集成redis**

1.添加依赖

<dependency>

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

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

2.properties文件中添加连接的配置 (需要预先设置好密码)

spring.redis.host=192.168.125.200

spring.redis.password=1010

spring.redis.port=6379

3.在上一节中的service的实现中

通过

@Autowired

private RedisTemplate<String, String> redisTemplate;

注入 redisTemplate

1. 在上一节中的service的接口和实现中添加

@Override

public String redisTest()

{

redisTemplate.opsForValue().set("aa","bb");

String getValue = redisTemplate.opsForValue().get("aa");

return getValue;

}

1. 在上一节的controller中添加 方法

@RequestMapping("/redisTest")

public String redisTest() {

return service.redisTest();

}

通过 http://localhost:8999/hello/redisTest 访问数据，返回结果

**集成mybatis**

1. 接第一个个springboot项目

1.添加依赖

<dependency>

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

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

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

<version>8.0.13</version>

</dependency>

2.properties文件中添配置 (对应的package 和连接字符串要和代码数据库匹配)

mybatis.type-aliases-package=com.example.mybatisdemo (此行可以不要)

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://192.168.125.200:3306/testdb?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8

spring.datasource.username=xl

spring.datasource.password=1010

3.在启动类中加注解

@MapperScan("com.example.mybatisdemo.mapper")

新建package mapper (此package需和上一行配置的完全对应，层级不能多也不能少) 并新建StudentMapper.java 的接口，并添加方法

public interface StudentMapper {

int getCount();

}

4.在properties文件中添加 配置项

mybatis.config-location=classpath:mybatis/mybatis-config.xml

mybatis.mapper-locations=classpath:mybatis/mapper/\*.xml

5.在resources目录下新建 mybatis目录并新建mybatis-config.xml 和mapper子目录，并新建student.xml文件，和上一步的配置对应

mybatis-config.xml的内容如下

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<typeAliases>

<typeAlias alias="Integer" type="java.lang.Integer" />

<typeAlias alias="Long" type="java.lang.Long" />

<typeAlias alias="HashMap" type="java.util.HashMap" />

<typeAlias alias="LinkedHashMap" type="java.util.LinkedHashMap" />

<typeAlias alias="ArrayList" type="java.util.ArrayList" />

<typeAlias alias="LinkedList" type="java.util.LinkedList" />

</typeAliases>

</configuration>

student.xml 的内容如下（需要和上面的StudentMapper 接口对应）

<?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.example.mybatisdemo.mapper.StudentMapper">

<select id="getCount" resultType="java.lang.Integer">

SELECT count(\*)

FROM student

</select>

</mapper>

1. 新建 package service 以及子package imp

新建TestService 接口机器 实现类 TestServiceImp

public interface TestService {

int getTestCount();

}

@Service

public class TestServiceImp implements TestService {

@Autowired

private StudentMapper studentMapper;

@Override

public int getTestCount( ) {

return studentMapper.getCount();

}

}

6.新建package controller,并在其中新建TestController 类

@RestController

@RequestMapping("/test")

public class TestController {

@Autowired

private TestService testService;

@RequestMapping("/getCount")

public int getCount() {

return testService.getTestCount();

}

}

通过 localhost:port//test/getCount 来访问

7.返回实体列表

新建package bean 以及与表对应的实体类Student.java

public class Student {

public int id;

public String name;

public int 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 int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

在StudentMapper.java中添加方法 List<Student> getAll();

在student.xml文件中添加 对应的实体映射和方法映射

<resultMap id="studentMap" type="com.example.mybatisdemo.bean.Student">

<id property="id" column="id"/>

<result property="name" column="name "/>

<result property="age" column="age"/>

</resultMap>

<select id="getAll" resultMap="studentMap">

SELECT \*

FROM student

</select>

在 TestService 和 TestServiceImp 中添加对应的方法和方法实现

List<Student> getAll();

@Override

public List<Student> getAll()

{

return studentMapper.getAll();

}

在Contoller中添加对应的方法

@RequestMapping("/getAll")

public List<Student> getAll() {

return testService.getAll();

}

通过 <http://localhost:8929/test/getAll> 访问

**集成mybatis plus**

0.接第一个springboot项目

1.添加依赖

<dependency>

<groupId>com.baomidou</groupId>

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

<version>3.4.1</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

<version>8.0.13</version>

</dependency>

2.properties文件中添配置

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://192.168.125.200:3306/testdb?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8

spring.datasource.username=xl

spring.datasource.password=1010

并在对应的数据库中执行脚本，生成测试表和测试数据

CREATE TABLE user

(

id BIGINT(20) NOT NULL COMMENT '主键ID',

name VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',

age INT(11) NULL DEFAULT NULL COMMENT '年龄',

email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',

PRIMARY KEY (id)

);

INSERT INTO user (id, name, age, email) VALUES

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

(2, 'Jack', 20, 'test2@baomidou.com'),

(3, 'Tom', 28, 'test3@baomidou.com'),

(4, 'Sandy', 21, 'test4@baomidou.com'),

(5, 'Billie', 24, ['test5@baomidou.com');](mailto:'test5@baomidou.com');)

1. 使用 mybatis plus 中 内置的方法 来访问数据库

在启动类中加注解

@MapperScan("com.example.mybatisdemo.mapper")

新建 package bean , 在其中新建 实体类 User.java

public class User {

private Long id;

private String name;

private Integer age;

private String email;

public Long getId() {

return id;

}

public void setId(Long 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;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

新建package mapper (此package需和上一行配置的完全对应，层级不能多也不能少) 并新建UserMapper .java 的接口

public interface UserMapper extends BaseMapper<User> {

}

在HelloWorldController中添加

@Autowired

private UserMapper userMapper;

@RequestMapping("/plus")

public List<User> plus(){

List<User> curList = userMapper.selectList(null);

return curList;

}

1. 使用xml 中定义的方法来访问数据库 （和mybatis中的一致）

在properties文件中添加 配置项

mybatis-plus.mapper-locations=classpath:mybatis/mapper/\*.xml

在resources目录下新建 mybatis目录 和mapper子目录，并新建user.xml文件，

user.xml 的内容如下（需要和上面的UserMapper 接口对应）

<?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.example.mybatisdemo.mapper.StudentMapper">

<select id="getCount" resultType="java.lang.Integer">

SELECT count(\*)

FROM student

</select>

</mapper>

在 UserMapper.java中添加 int getCount(); 方法

在HelloWorldController.java中添加 方法

@RequestMapping("/plusxml")

public int plusxml(){

return userMapper.getCount();

}

使用 <http://localhost:8099/hello/plusxml> 访问

1. 基于xml 做实体映射的查询

在user.xml中添加

<resultMap id="userMap" type="com.example.mybatisdemo.bean.User">

<id property="id" column="id"/>

<result property="name" column="name "/>

<result property="age" column="age"/>

<result property="email" column="email"/>

</resultMap>

<select id="getAllByXml" resultMap="userMap">

SELECT \*

FROM user

</select>

在UserMapper.java 中添加方法

List<User> getAllByXml();

在 HelloWorldController.java中添加

@RequestMapping("/getAllByXml")

public List<User> getAllByXml(){

return userMapper.getAllByXml();

}

使用 [http://localhost:8099/hello/getAllByXml](http://localhost:8099/hello/plusxml) 访问

1. 基于mapper的crud

在HelloWorldController.java中添加 如下方法

@RequestMapping("/mapperTest")

public int mapperTest() {

User inserUser = new User();

inserUser.setId(80L);

inserUser.setAge(100);

inserUser.setName("bbsad");

inserUser.setEmail("aa@bb.com");

userMapper.insert(inserUser);

User updateUser = new User();

updateUser.setId(10L);

updateUser.setAge(1000);

userMapper.updateById(updateUser);

UpdateWrapper<User> updateWrapper = new UpdateWrapper<>();

updateWrapper.eq("age", "1000");

User user = new User();

user.setAge(18);

Integer rows = userMapper.update(user, updateWrapper);

UpdateWrapper<User> updateWrapper1 = new UpdateWrapper<>();

updateWrapper1.eq("age", 18).set("age", 35);

rows = userMapper.update(null, updateWrapper1);

//lamada 方式update

LambdaUpdateWrapper<User> lambdaUpdateWrapper = new LambdaUpdateWrapper<>();

lambdaUpdateWrapper.eq(User::getAge, 35).set(User::getAge, 34);

rows = userMapper.update(null, lambdaUpdateWrapper);

QueryWrapper<User> deleteQw = new QueryWrapper<User>();

deleteQw.between("age",30,35);

userMapper.delete(deleteQw);

return 1;

}

使用 [http://localhost:8099/hello/getAllByXml](http://localhost:8099/hello/plusxml) 访问

6.分页查询

添加依赖

<dependency>

<groupId>com.github.jsqlparser</groupId>

<artifactId>jsqlparser</artifactId>

<version>4.0</version>

</dependency>

添加配置类 MybatisConfig.java

@Configuration

@ConditionalOnClass(value = {PaginationInterceptor.class})

public class MybatisConfig {

@Bean

public PaginationInterceptor paginationInterceptor() {

PaginationInterceptor paginationInterceptor = new PaginationInterceptor();

return paginationInterceptor;

}

}

在HelloWorldController.java 中添加如下方法

@RequestMapping("/pageTest")

public int pageTest()

{

QueryWrapper<User> ew = new QueryWrapper<User>();

ew.orderByDesc("id");

ew.between("age",20,30);

IPage<User> firstPage = new Page<>(1, 2);//参数一是当前页，参数二是每页个数

IPage<User> secondPage = new Page<>(2, 2);//参数一是当前页，参数二是每页个数

IPage<User> curPage = userMapper.selectPage(new Page<>(1, 2), ew);

long pages = curPage.getPages();

List<User> firstList = userMapper.selectPage(firstPage, ew).getRecords();

List<User> secondList = userMapper.selectPage(secondPage, ew).getRecords();

return 2;

}

使用 [http://localhost:8099/hello/pageTest](http://localhost:8099/hello/plusxml) 访问

**mybatis plus 代码自动生成**

0.接第一个springboot项目

1. 添加自动生成相关的依赖，默认使用freemarker作为模板

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-generator</artifactId>

<version>3.4.1</version>

</dependency>

<dependency>

<groupId>org.freemarker</groupId>

<artifactId>freemarker</artifactId>

<version>2.3.30</version>

</dependency>

2.新建 CodeGenerator.java ，此文件中包含main函数，根据对应的连接字符串和配置，来生成代码

import com.baomidou.mybatisplus.core.toolkit.StringPool;

import com.baomidou.mybatisplus.generator.AutoGenerator;

import com.baomidou.mybatisplus.generator.InjectionConfig;

import com.baomidou.mybatisplus.generator.config.\*;

import com.baomidou.mybatisplus.generator.config.po.TableInfo;

import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;

import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;

import java.util.ArrayList;

import java.util.List;

public class CodeGenerator {

public static void main(String[] args) {

// 代码生成器

AutoGenerator mpg = new AutoGenerator();

// 全局配置

GlobalConfig gc = new GlobalConfig();

String projectPath = System.getProperty("user.dir");

gc.setOutputDir(projectPath + "/src/main/java");

gc.setAuthor("jobob");

gc.setOpen(false);

// gc.setSwagger2(true); 实体属性 Swagger2 注解

mpg.setGlobalConfig(gc);

// 数据源配置

DataSourceConfig dsc = new DataSourceConfig();

dsc.setUrl("jdbc:mysql://192.168.125.200:3306/testdb?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8");

// dsc.setSchemaName("public");

dsc.setDriverName("com.mysql.cj.jdbc.Driver");

dsc.setUsername("xl");

dsc.setPassword("1010");

mpg.setDataSource(dsc);

// 包配置

PackageConfig pc = new PackageConfig();

// pc.setModuleName(scanner("模块名"));

pc.setParent("com.example.mybatisdemo");

mpg.setPackageInfo(pc);

// 自定义配置

InjectionConfig cfg = new InjectionConfig() {

@Override

public void initMap() {

// to do nothing

}

};

// 如果模板引擎是 freemarker

String templatePath = "/templates/mapper.xml.ftl";

// 如果模板引擎是 velocity

// String templatePath = "/templates/mapper.xml.vm";

// 自定义输出配置

List<FileOutConfig> focList = new ArrayList<>();

// 自定义配置会被优先输出

focList.add(new FileOutConfig(templatePath) {

@Override

public String outputFile(TableInfo tableInfo) {

// 自定义输出文件名 ， 如果你 Entity 设置了前后缀、此处注意 xml 的名称会跟着发生变化！！

return projectPath + "/src/main/resources/mapper/" + pc.getModuleName()

+ "/" + tableInfo.getEntityName() + "Mapper" + StringPool.DOT\_XML;

}

});

/\*

cfg.setFileCreate(new IFileCreate() {

@Override

public boolean isCreate(ConfigBuilder configBuilder, FileType fileType, String filePath) {

// 判断自定义文件夹是否需要创建

checkDir("调用默认方法创建的目录，自定义目录用");

if (fileType == FileType.MAPPER) {

// 已经生成 mapper 文件判断存在，不想重新生成返回 false

return !new File(filePath).exists();

}

// 允许生成模板文件

return true;

}

});

\*/

cfg.setFileOutConfigList(focList);

mpg.setCfg(cfg);

// 配置模板

TemplateConfig templateConfig = new TemplateConfig();

// 配置自定义输出模板

//指定自定义模板路径，注意不要带上.ftl/.vm, 会根据使用的模板引擎自动识别

// templateConfig.setEntity("templates/entity2.java");

// templateConfig.setService();

// templateConfig.setController();

templateConfig.setXml(null);

mpg.setTemplate(templateConfig);

// 策略配置

StrategyConfig strategy = new StrategyConfig();

strategy.setNaming(NamingStrategy.underline\_to\_camel);

strategy.setColumnNaming(NamingStrategy.underline\_to\_camel);

// strategy.setSuperEntityClass("你自己的父类实体,没有就不用设置!");

// strategy.setEntityLombokModel(true);

strategy.setRestControllerStyle(true);

// 公共父类

// strategy.setSuperControllerClass("你自己的父类控制器,没有就不用设置!");

// 写于父类中的公共字段

// strategy.setSuperEntityColumns("id");

strategy.setInclude("user".split(","));

strategy.setControllerMappingHyphenStyle(true);

strategy.setTablePrefix(pc.getModuleName() + "\_");

mpg.setStrategy(strategy);

mpg.setTemplateEngine(new FreemarkerTemplateEngine());

mpg.execute();

}

}

触雷配置数据源意外，主要有以下几项的配置

pc.setParent("com.example.mybatisdemo"); //根namespace

strategy.setInclude("user".split(",")); //要生成的表

return projectPath + "/src/main/resources/mapper/" //生成的xml 的地址

1. 添加配置和依赖（此步骤和普通 mybatis项目步骤一样）

主类中添加 @MapperScan("com.example.mybatisdemo.mapper")

properties文件中添加

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://192.168.125.200:3306/testdb?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8

spring.datasource.username=xl

spring.datasource.password=1010

mybatis-plus.mapper-locations=classpath:mapper/\*.xml

依赖中添加

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

<version>8.0.13</version>

</dependency>

<dependency>

<groupId>com.baomidou</groupId>

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

<version>3.4.1</version>

</dependency>

<dependency>

<groupId>com.github.jsqlparser</groupId>

<artifactId>jsqlparser</artifactId>

<version>4.0</version>

</dependency>

最后一个是分页用

4.在UserController.java 中添加测试方法

@Autowired

private IUserService userService;

@RequestMapping("/serviceTest")

public int serviceTest(){

User inserUser = new User();

inserUser.setId(310L);

inserUser.setAge(100);

inserUser.setName("bbsad");

inserUser.setEmail("aa@bb.com");

userService.save (inserUser);

User updateUser = new User();

updateUser.setId(10L);

updateUser.setAge(1000);

userService.updateById(updateUser);

UpdateWrapper<User> updateWrapper = new UpdateWrapper<>();

updateWrapper.eq("age", "1000");

User user = new User();

user.setAge(18);

boolean rows = userService.update(user, updateWrapper);

UpdateWrapper<User> updateWrapper1 = new UpdateWrapper<>();

updateWrapper1.eq("age", 18).set("age", 35);

rows = userService.update(null, updateWrapper1);

//lamada 方式update

LambdaUpdateWrapper<User> lambdaUpdateWrapper = new LambdaUpdateWrapper<>();

lambdaUpdateWrapper.eq(User::getAge, 35).set(User::getAge, 34);

rows = userService.update(null, lambdaUpdateWrapper);

QueryWrapper<User> deleteQw = new QueryWrapper<User>();

deleteQw.between("age",30,35);

userService.remove(deleteQw);

return 1;

}

@RequestMapping("/pageTest")

public int pageTest()

{

QueryWrapper<User> ew = new QueryWrapper<User>();

ew.orderByDesc("id");

ew.between("age",20,30);

IPage<User> firstPage = new Page<>(1, 2);//参数一是当前页，参数二是每页个数

IPage<User> secondPage = new Page<>(2, 2);//参数一是当前页，参数二是每页个数

IPage<User> curPage = userService.page(new Page<>(1, 2), ew);

long pages = curPage.getPages();

List<User> firstList = userService.page(firstPage, ew).getRecords();

List<User> secondList = userService.page(secondPage, ew).getRecords();

return 2;

}

通过对应的url 访问

**集成log4j2**

0.接第一个springboot项目

1. 添加依赖

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.5</version>

</dependency>

1. 在resource目录下添加log4j2.xml 文件，完成日志输出的配置，示例如下：

<?xml version="1.0" encoding="UTF-8"?>

<configuration status="error">

<!-- 先定义所有的appender -->

<appenders>

<!-- 这个输出控制台的配置 -->

<Console name="Console" target="SYSTEM\_OUT">

<!-- 控制台只输出level及以上级别的信息（onMatch），其他的直接拒绝（onMismatch） -->

<ThresholdFilter level="trace" onMatch="ACCEPT" onMismatch="DENY"/>

<!-- 这个都知道是输出日志的格式 -->

<PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %t %class{36} %L %M - %msg%xEx%n"/>

</Console>

<!-- 文件会打印出所有信息，这个log每次运行程序会自动清空，由append属性决定，这个也挺有用的，适合临时测试用 -->

<!-- append为TRUE表示消息增加到指定文件中，false表示消息覆盖指定的文件内容，默认值是true -->

<File name="log" fileName="logs/test.log" append="false">

<PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %t %class{36} %L %M - %msg%xEx%n"/>

</File>

<!-- 添加过滤器ThresholdFilter,可以有选择的输出某个级别以上的类别 onMatch="ACCEPT" onMismatch="DENY"意思是匹配就接受,否则直接拒绝 -->

<File name="ERROR" fileName="logs/error.log">

<ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="DENY"/>

<PatternLayout pattern="%d{yyyy.MM.dd 'at' HH:mm:ss z} %-5level %t %class{36} %L %M - %msg%xEx%n"/>

</File>

<!-- 这个会打印出所有的信息，每次大小超过size，则这size大小的日志会自动存入按年份-月份建立的文件夹下面并进行压缩，作为存档 -->

<RollingFile name="RollingFile" fileName="logs/web.log"

filePattern="logs/$${date:yyyy-MM}/web-%d{MM-dd-yyyy}-%i.log.gz">

<PatternLayout pattern="%d{yyyy-MM-dd 'at' HH:mm:ss z} %-5level %t %class{36} %L %M - %msg%xEx%n"/>

<SizeBasedTriggeringPolicy size="2MB"/>

</RollingFile>

</appenders>

<!-- 然后定义logger，只有定义了logger并引入的appender，appender才会生效 -->

<loggers>

<!-- 建立一个默认的root的logger -->

<root level="trace">

<appender-ref ref="RollingFile"/>

<appender-ref ref="Console"/>

<appender-ref ref="ERROR" />

<appender-ref ref="log"/>

</root>

</loggers>

</configuration>

1. 在代码中调用

设置日志变量

private static Logger logger = LogManager.getLogger(HelloWorldController.class.getName());

输出日志的代码

logger.trace("trace");

logger.info("info");

logger.error("error");

参考 <https://www.cnblogs.com/KylinBlog/p/7841217.html>

**读取yml 和property文件**

1.在resource目录下新建 application.yml , 按格式进行配置

通过以下方式将变量读取到变量中, 支持 map list 等数据结构

@Value("${server.port}")

public String port;

@Value("${dd.bb}")

public String bb;

2.yml文件的多环境

可以新建application-dev.yml application-test.yml application-prod.yml 文件 进行不同环境下的配置

在 application.yml 文件中 添加如下配置，确定当前激活的是什么环境

spring:

# 环境 dev:开发环境|test:测试环境|prod:生产环境

profiles:

active: dev #激活的配置文件

**定时任务**

固定线程的任务

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.springframework.context.annotation.Configuration;

import org.springframework.scheduling.annotation.EnableScheduling;

import org.springframework.scheduling.annotation.Scheduled;

import java.time.LocalDateTime;

@Configuration //1.主要用于标记配置类，兼备Component的效果。

@EnableScheduling // 2.开启定时任务

public class SaticScheduleTask {

private static Logger logger = LogManager.getLogger(SaticScheduleTask.class.getName());

//3.添加定时任务

@Scheduled(cron = "0/5 \* \* \* \* ?")

//或直接指定时间间隔，例如：5秒

//@Scheduled(fixedRate=5000)

private void configureTasks() {

logger.trace("执行静态定时任务时间: " + LocalDateTime.now());

}

}

2.异步任务，动态分配线程

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.springframework.scheduling.annotation.Async;

import org.springframework.scheduling.annotation.EnableAsync;

import org.springframework.scheduling.annotation.EnableScheduling;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

import java.time.LocalDateTime;

//@Component注解用于对那些比较中立的类进行注释；

//相对与在持久层、业务层和控制层分别采用 @Repository、@Service 和 @Controller 对分层中的类进行注释

@Component

@EnableScheduling // 1.开启定时任务

@EnableAsync // 2.开启多线程

public class MultithreadScheduleTask {

private static Logger logger = LogManager.getLogger(MultithreadScheduleTask.class.getName());

@Async

@Scheduled(fixedDelay = 1000) //间隔1秒

public void first() throws InterruptedException {

logger.trace("第一个定时任务开始 : " + LocalDateTime.now().toLocalTime() + "\r\n线程 : " + Thread.currentThread().getName());

Thread.sleep(1000 \* 10);

}

@Async

@Scheduled(fixedDelay = 2000)

public void second() {

logger.trace("第二个定时任务开始 : " + LocalDateTime.now().toLocalTime() + "\r\n线程 : " + Thread.currentThread().getName());

}

}