# 1 培训目标

本次Springboot培训，旨在让在座各位同仁，尤其是深海BSP项目组的开发人员，了解并且学习深海BSP项目中的各种技术的整合与应用，节约在熟悉项目代码的过程中由于技术问题所消耗的大量时间，希望经过此次springboot的培训，加深对项目技术的理解，然后将更多的时间投入到对系统业务的熟悉中去，而不是浪费在探究技术放面。

而且，这篇培训文档也能新进入团队的伙伴更快的投入到项目实际开发工作中。

# 2 springboot介绍

## 2.1 什么是springboot

官方截图。。。

## 2.2 Springboot有什么优点

1 快速构建项目。

2 对主流框架可以实现无配置集成。

3 项目可以独立运行，无需外部servlet容器。

4 极大的提高了项目的开发部署效率。

## 2.3 Springboot快速入门

1. 首先我们创建一个maven项目

项目名称springboot-study-demo

1. 设置springboot-study-demo的parent

说明：Springboot的项目必须要将parent设置为springboot的parent，该parent包含了大量默认的配置，大大简化了我们的开发。

<!-- 该parent包含了大量默认的配置，简化开发 -->

<parent>

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

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

<version>1.5.1.RELEASE</version>

</parent>

1. 导入spring boot的web支持

<dependency>

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

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

</dependency>

1. 导入spring-boot-maven插件

<build>

<plugins>

<!-- 导入spring-boot-maven插件 -->

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

<mainClass>com.cxn.MyApplication</mainClass>

</configuration>

</plugin>

</plugins>

</build>

1. 编写springboot的启动类

**package** com.cxn;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** MyApplication {

// springboot项目的启动入口

**public** **static** **void** main(String[] args) {

// SpringApplication.run(MyApplication.class, args);

SpringApplication app = **new** SpringApplication(MyApplication.**class**);

app.run(args);

}

}

1. 新建一个controller

**package** com.cxn.controller;

**import** org.springframework.stereotype.Controller;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

@Controller

**public** **class** HelloWorldController {

@RequestMapping(path = "/hello",method={RequestMethod.***GET***})

@ResponseBody

**public** String helloWorld(){

System.***out***.println("helloWorld!");

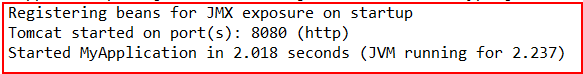
**return** "helloWorld!";

}

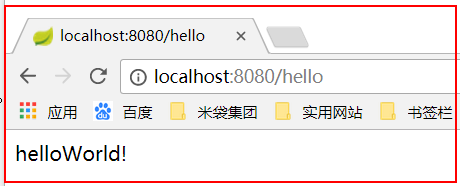
}

1. 启动测试

运行MyApplication的Main方法，开一下启动效果。



1. 测试



至此，一个最简单的springboot项目就搭建好了。

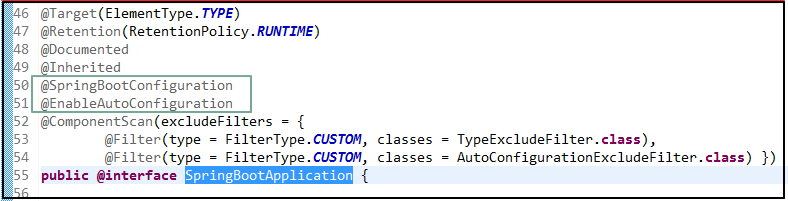
## 2.4 Springboot的核心

2.4.1 入口类和@SpringBootApplication

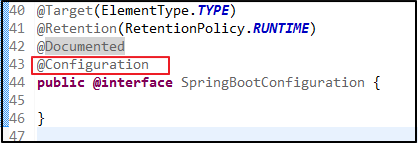
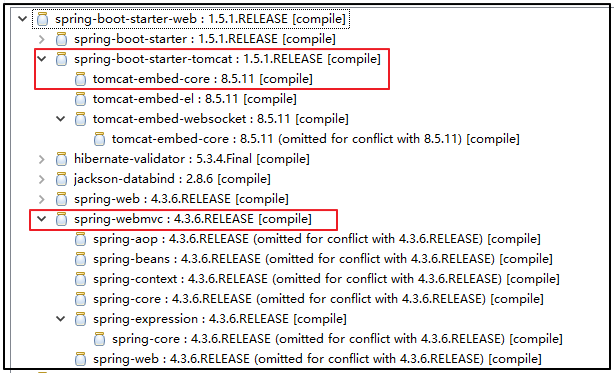
Springboot的项目都会有一个入口类，入口类中会有main方法，这是一个标准的java应用程序的入口方法。

在刚才的HelloWorld实例中，MyApplication就是springboot的入口类。

@SpringBootApplication注解是SpringBoot的核心注解，它其实是一个组合注解：

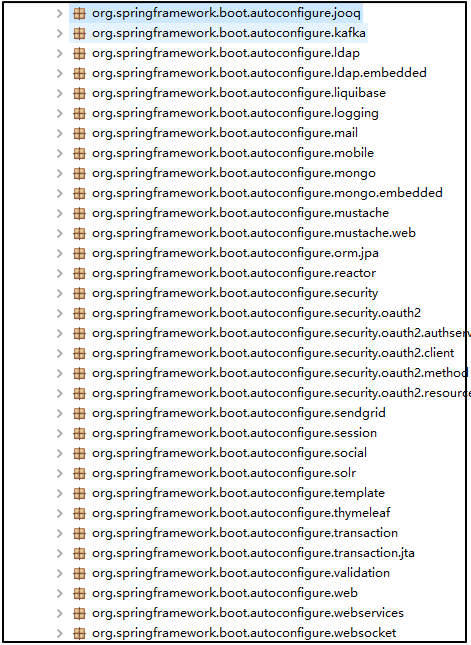
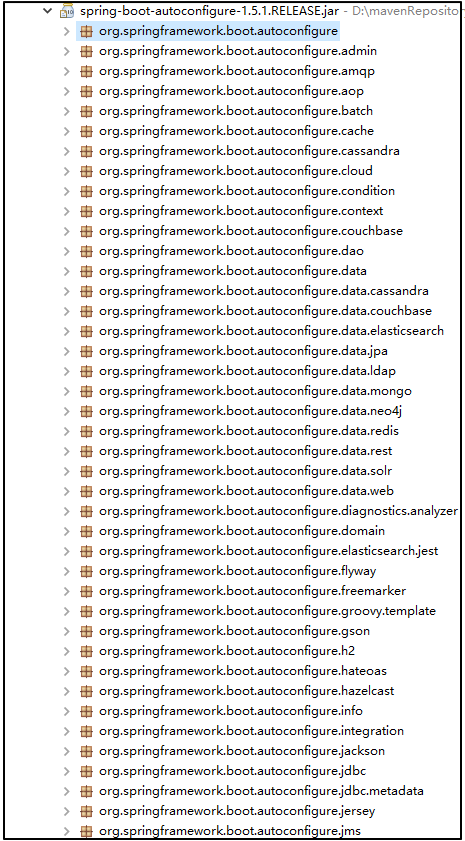


该注解主要组合了以下注解：

1. @SpringBootConfiguration：这是SpringBoot项目的配置注解，这也是一个组合注解：  
     
   在Spring Boot项目中推荐使用@ SpringBootConfiguration替代@Configuration
2. @EnableAutoConfiguration：启用自动配置，该注解会使SpringBoot根据项目中依赖的jar包自动配置项目的配置项：
   1. 如：我们添加了spring-boot-starter-web的依赖，项目中也就会引入SpringMVC的依赖，SpringBoot就会自动配置tomcat和SpringMVC  
      
3. @ComponentScan：默认扫描@SpringBootApplication所在类的同级目录以及它的子目录。

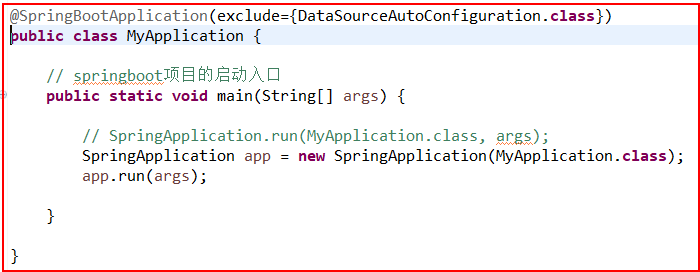
2.4.2 关闭自动配置

通过上述，我们得知，SpringBoot会根据项目中的jar包依赖，自动做出配置，SpringBoot支持的自动配置如下（非常多）：



如果我们不需要SpringBoot自动配置，想关闭某一项的自动配置，该如何设置呢？

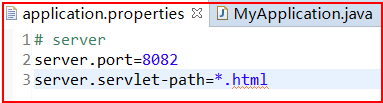
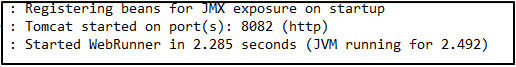
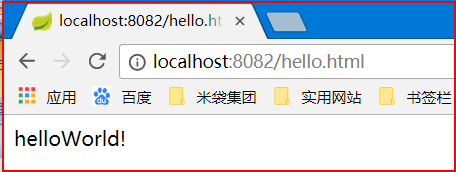
比如：我们不想自动配置数据源，想手动配置。



当然了，其他的配置就类似了。

2.4.3 全局配置文件

SpringBoot项目使用一个全局的配置文件application.properties或者是application.yml，在resources目录下或者类路径下的/config下，一般我们放到resources下。

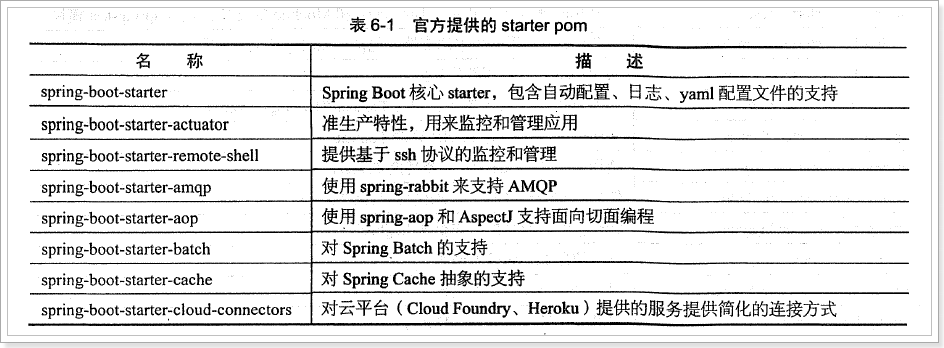
1. 修改tomcat的端口为8082和进入DispatcherServlet的规则为：\*.html  
   
2. 重新启动应用，查看效果：  
     
      
   测试：  
    

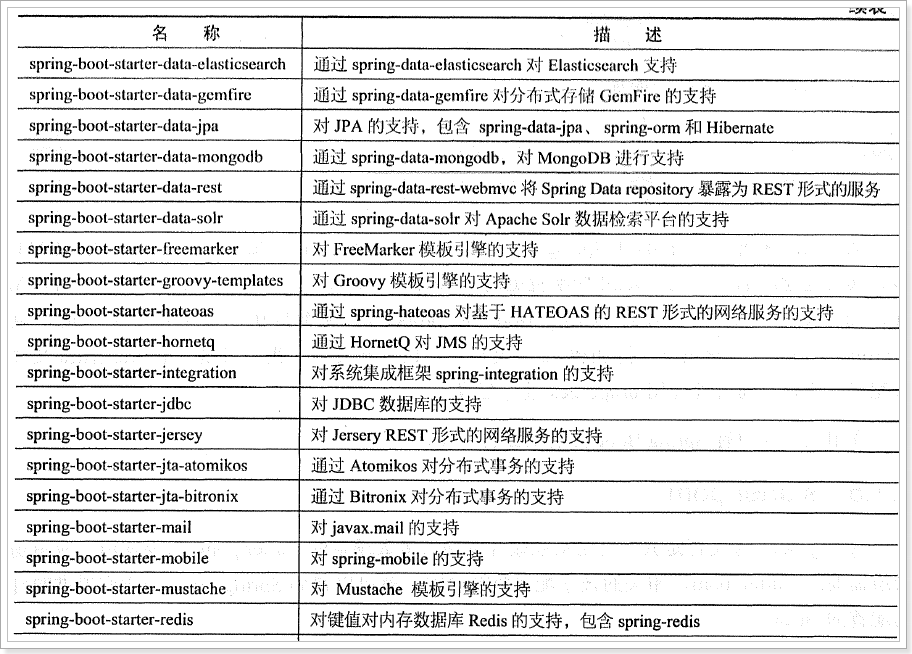
2.4.4 Starter pom

Spring Boot为我们提供了简化企业级开发绝大多数场景的starter pom,只要使用了应用场景所需要的starter pom,

相关的技术配置将会消除,这样就可以得到Spring Boot为我们提供的自动配置的Bean。

截图中展示的是官方提供的starter pom列表:







当然随着spring生态的不断壮大,还涌现出很多非官方的starter-pom,还可以自定义starter,有感兴趣的同事可以参考这篇博文:

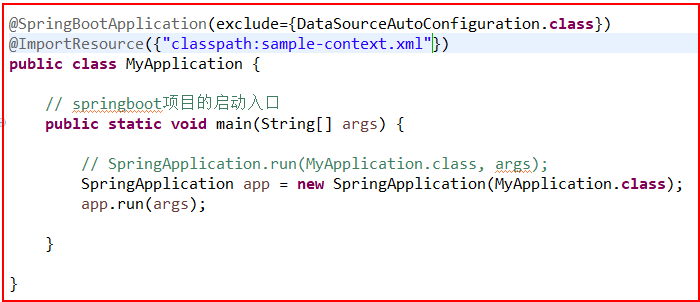
<http://www.jianshu.com/p/85460c1d835a>



Xml配置文件

Spring Boot 提倡零配置,即无xml配置,但在实际项目中,可能有一些特殊要求我们必须使用xml配置,这时我们可以通过Spring提供的@ImportResource来加载xml配置。

例如:



2.4.6日志

SpringBoot对各种日志框架都做了支持，我们可以通过配置来修改默认的日志的配置：

# logging

# logging.file=${user.home}/logs/springboot-study-demo.log

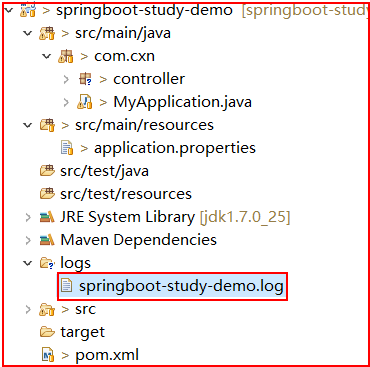
logging.file=logs/springboot-study-demo.log

logging.level.root=info

logging.pattern.console=[%d{yyyy-MM-ddhh:mm:ssSSS}]%t%5p%c-%m%n

logging.pattern.file=[%d{yyyy-MM-ddhh:mm:ssSSS}]%t%5p%c-%m%n

重新启动一下，刷新项目后在项目列表下看到日志：

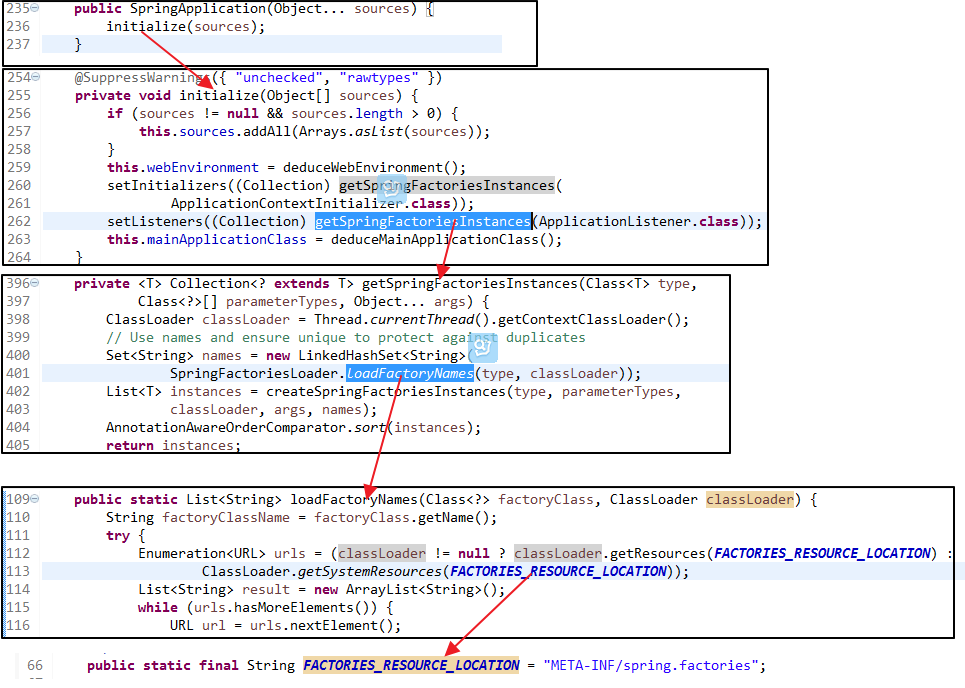


## 2.5 SpringBoot的自动配置的原理

SpringBoot在进行SpringApplication对象实例化时会加载META-INF/spring.factories文件，将该配置文件中的配置载入到Spring容器。

源码分析

org.springframework.boot.SpringApplication：



由此可见，读取该配置文件来加载内容。

# 3 SpringBoot整合第三方内容

## 3.1 Springboot连接数据库

导入pom依赖：

<!-- mysql驱动 -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

首先使用jdbc连接数据库

数据库建表语句：

CREATE TABLE `tbl\_study\_table1` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`text` text,

`flag` tinyint(1) DEFAULT NULL,

`create\_time` datetime DEFAULT NULL,

`update\_time` timestamp NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

新建StudyTableModel

**public** **class** StudyTableModel {

**private** Long id;

**private** String text;

**private** Integer flag;

**private** String createTime;

**private** String updateTime;

// 省略get set方法

}

新建StudyTableService接口

**package** com.cxn.service;

**import** com.cxn.model.StudyTableModel;

**public** **interface** StudyTableService {

**public** StudyTableModel getById(**long** id);

}

新建StudyTableServiceImpl实现类

**package** com.cxn.service.impl;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.springframework.stereotype.Service;

**import** com.cxn.model.StudyTableModel;

**import** com.cxn.service.StudyTableService;

@Service

**public** **class** StudyTableServiceImpl **implements** StudyTableService {

@Override

**public** StudyTableModel getById(**long** modelId) {

**try** {

// 1.注册驱动

Class.*forName*("com.mysql.jdbc.Driver");

// 2.获得连接

String url = "jdbc:mysql://localhost:3306/studytest\_db";

String user = "root";

String password = "123";

Connection conn = DriverManager.*getConnection*(url, user, password);

// 3.sql的承载对象

String sql = " select \* from tbl\_study\_table1 where id = " + modelId;

PreparedStatement preState = conn.prepareStatement(sql);

// 4.执行sql获得结果

ResultSet rs = preState.executeQuery();

// 5.处理结果

StudyTableModel model = **new** StudyTableModel();

**while** (rs.next()) {

**long** id = rs.getInt("id");

String text = rs.getString("text");

**int** flag = rs.getInt("flag");

String createTime = rs.getString("create\_time");

String updateTime = rs.getString("update\_time");

System.***out***.println(id + ":::" + text + ":::" + flag +":::" + createTime + ":::" + updateTime);

model.setId(id);

model.setFlag(flag);

model.setText(text);

model.setCreateTime(createTime);

model.setUpdateTime(updateTime);

}

// 6.释放资源

rs.close();

preState.close();

conn.close();

**return** model;

} **catch** (Exception e) {

e.printStackTrace();

**return** **null**;

}

}

}

在HelloWorldController中注入service和添加方法

@Autowired

**private** StudyTableService studyTableService;

@RequestMapping(path="/getById/{id}",method={RequestMethod.***GET***})

@ResponseBody

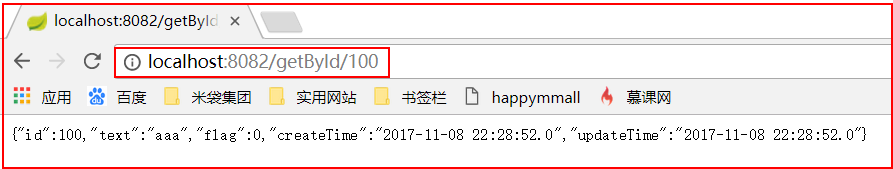
**public** StudyTableModel getById(@PathVariable("id") **long** id){

System.***out***.println("进入getById方法:" + id);

**return** studyTableService.getById(id);

}

进行测试：



拿到数据，数据库连接成功。

## 3.2 Springboot整合SwaggerUI

由于每次通过浏览器进行访问测试，不是很方便，postman访问也是一样，所以这里先行整合SwaggerUI，便于后续进行各种功能的测试。

导入pom依赖

<!-- 整合swaggerUI -->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.5.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.5.0</version>

</dependency>

新建包com.cxn.config

新建SwaggerConfig类

**package** com.cxn.config;

**import** org.springframework.boot.SpringBootConfiguration;

**import** org.springframework.context.annotation.Bean;

**import** springfox.documentation.builders.ApiInfoBuilder;

**import** springfox.documentation.builders.PathSelectors;

**import** springfox.documentation.builders.RequestHandlerSelectors;

**import** springfox.documentation.service.ApiInfo;

**import** springfox.documentation.service.Contact;

**import** springfox.documentation.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2

@SpringBootConfiguration

**public** **class** SwaggerConfig {

@Bean

**public** Docket controllerApi(){

Docket docket = **new** Docket(DocumentationType.***SWAGGER\_2***);

docket.groupName("group-one")

.apiInfo(apiInfo())

.select()

.apis(RequestHandlerSelectors.*basePackage*("com.cxn.controller"))

.paths(PathSelectors.*any*())// 可以使用正则regex("/user/.\*")

.build();

**return** docket;

}

// api题头信息

**private** ApiInfo apiInfo() {

ApiInfo apiInfo = **new** ApiInfoBuilder().title("SpringBoot-study-demo")

.description("这里是描述。。。")

.license("MIT")

.licenseUrl("http://opensource.org/licenses/MIT")

.contact(**new** Contact("电台台长", "192.168.99.66", "caoxunan121@163.com"))

.version("1.0")

.build();

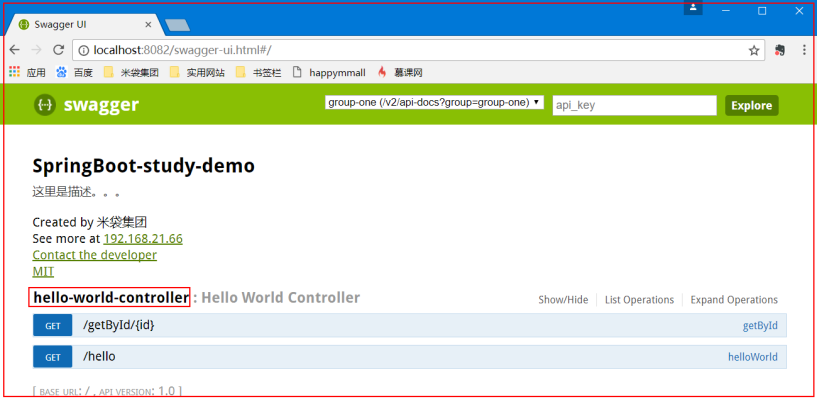
**return** apiInfo;

}

}

浏览器访问:

http://localhost:8082/swagger-ui.html#/



整合成功。

**3.3 Springboot整合mybatis及Durid线程池**

导入pom依赖

<!-- 德鲁伊线程池 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.28</version>

</dependency>

<!-- springboot整合mybatis -->

<dependency>

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

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

<version>1.2.0</version>

</dependency>

在resources下新建mybatis-conf.xml文件

<?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>

<settings>

<!-- 禁用 mybatis 二级缓存 -->

<setting name=*"cacheEnabled"* value=*"false"*/>

<!-- 禁用 mybatis 一级缓存 -->

<setting name=*"localCacheScope"* value=*"STATEMENT"*/>

<!-- 开启自动驼峰匹配 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"*/>

</settings>

</configuration>

向application.properties中添加数据源参数

# dataSource

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

spring.datasource.url=jdbc:mysql://localhost:3306/studytest\_db?characterEncoding=utf-8&autoReconnect=true&allowMultiQueries=true

spring.datasource.username=root

spring.datasource.password=123

# mybatis

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

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

新建druid配置类

import java.util.Arrays;

import javax.sql.DataSource;

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

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 com.alibaba.druid.filter.Filter;

import com.alibaba.druid.filter.stat.StatFilter;

import com.alibaba.druid.pool.DruidDataSource;

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

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

/\*\*

\* 德鲁伊配置类

\* @authorasus

\*

\*/

@Configuration

public class DuridConfig {

@Bean(name="statFilter")

public Filter statFilter(){

StatFilter stat=new StatFilter();

return stat;

}

@Bean(name = "dataSource")

public DataSource druidDataSource(@Value("${spring.datasource.driver-class-name}") String driver,

@Value("${spring.datasource.url}") String url,

@Value("${spring.datasource.username}") String username,

@Value("${spring.datasource.password}") String password ,

Filter ... filters) {

DruidDataSource druidDataSource = new DruidDataSource();

druidDataSource.setDriverClassName(driver);

druidDataSource.setUrl(url);

druidDataSource.setUsername(username);

druidDataSource.setPassword(password);

druidDataSource.setValidationQuery("select 1");

druidDataSource.setTestWhileIdle(true);

druidDataSource.setMinIdle(5);

druidDataSource.setMinEvictableIdleTimeMillis(300000);

druidDataSource.setTestOnBorrow(true);

druidDataSource.setRemoveAbandoned(true);

druidDataSource.setRemoveAbandonedTimeout(100000);

druidDataSource.setLogAbandoned(true);

if(filters!=null&&filters.length>0){

druidDataSource.setProxyFilters(Arrays.asList(filters));

}

return druidDataSource;

}

@Bean

public ServletRegistrationBean druidServlet() {

ServletRegistrationBean reg = new ServletRegistrationBean();

reg.setServlet(new StatViewServlet());

reg.addUrlMappings("/druid/\*");

//reg.addInitParameter("allow", "127.0.0.1");

//reg.addInitParameter("deny","");

reg.addInitParameter("loginUsername", "midai");

reg.addInitParameter("loginPassword", "midai");

return reg;

}

@Bean

public FilterRegistrationBean filterRegistrationBean() {

FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean();

filterRegistrationBean.setFilter(new WebStatFilter());

filterRegistrationBean.addUrlPatterns("/\*");

filterRegistrationBean.addInitParameter("exclusions", "\*.js,\*.gif,\*.jpg,\*.png,\*.css,\*.ico,/druid/\*");

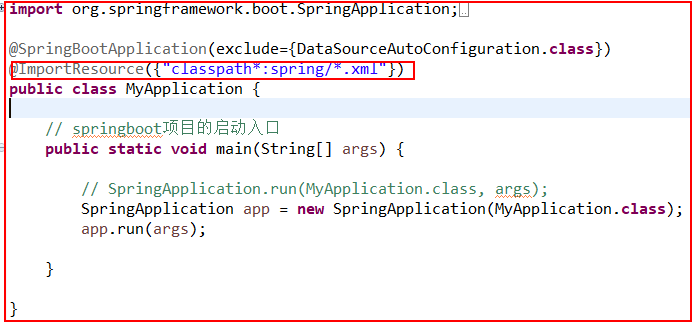
return filterRegistrationBean;

}

}

启动类添加注解：

@ImportResource({"classpath\*:spring/\*.xml"})



新建包com.cxn.mapper

新建接口StudyTableMapper

**package** com.cxn.mapper;

**import** org.apache.ibatis.annotations.Mapper;

**import** org.apache.ibatis.annotations.Select;

**import** com.cxn.model.StudyTableModel;

@Mapper

**public** **interface** StudyTableMapper {

@Select("select \* from tbl\_study\_table1 where id = #{id}")

**public** StudyTableModel getById(**long** id);

}

在StudyTableService中添加接口方法

**public** StudyTableModel getByIdUseMapper(**long** id);

修改实现类

@Override

**public** StudyTableModel getByIdUseMapper(**long** id) {

**return** studyTableMapper.getById(id);

}

在HelloWorldController中添加方法

@RequestMapping(path="/getByIdUseMapper/{id}",method={RequestMethod.***GET***})

@ResponseBody

**public** StudyTableModel getByIdUseMapper(@PathVariable("id") **long** id){

System.***out***.println("进入getById方法:" + id);

**return** studyTableService.getByIdUseMapper(id);

}

使用SwaggerUI进行测试，没有问题。



访问:http://localhost:8082/druid/login.html进入durid监控器页面

如果数据表列名与model属性不匹配的话，则新建映射关系文件

新建src/main/resources/mybatis/mapper/StudyTableMapper.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.cxn.mapper.StudyTableMapper"*>

<resultMap type=*"com.cxn.model.StudyTableModel"* id=*"StudyTableModelMap"*>

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

<result property=*"text"* column=*"text"*/>

<result property=*"flag"* column=*"flag"*/>

<result property=*"createTime"* column=*"create\_time"*/>

<result property=*"updateTime"* column=*"update\_time"*/>

</resultMap>

</mapper>

添加配置让Spring进行事务管理。。。。。。。。。。。。。。。

新建src/main/resources/spring/spring-transaction.xml

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-4.0.xsd"*>

<!-- ========================================分隔线========================================= -->

<!-- 配置Spring的事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 注解方式配置事物 -->

<!-- <tx:annotation-driven transaction-manager="transactionManager" /> -->

<!-- 拦截器方式配置事物 -->

<tx:advice id=*"transactionAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"add\*"* propagation=*"REQUIRED"* />

<tx:method name=*"append\*"* propagation=*"REQUIRED"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* />

<tx:method name=*"save\*"* propagation=*"REQUIRED"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* />

<tx:method name=*"modify\*"* propagation=*"REQUIRED"* />

<tx:method name=*"edit\*"* propagation=*"REQUIRED"* />

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* />

<tx:method name=*"del\*"* propagation=*"REQUIRED"* />

<tx:method name=*"remove\*"* propagation=*"REQUIRED"* />

<tx:method name=*"turn\*"* propagation=*"REQUIRED"* />

<tx:method name=*"batch\*"* propagation=*"REQUIRED"* />

<tx:method name=*"repair"* propagation=*"REQUIRED"* />

<tx:method name=*"delAndRepair"* propagation=*"REQUIRED"* />

<tx:method name=*"new\*"* propagation=*"REQUIRES\_NEW"* />

<tx:method name=*"not\*"* propagation=*"NOT\_SUPPORTED"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"load\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"search\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"datagrid\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"query\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"\*"* propagation=*"SUPPORTS"* />

</tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut id=*"transactionPointcut"*

expression=*"execution(\* com.cxn.service.impl.\*Impl.\*(..))"* />

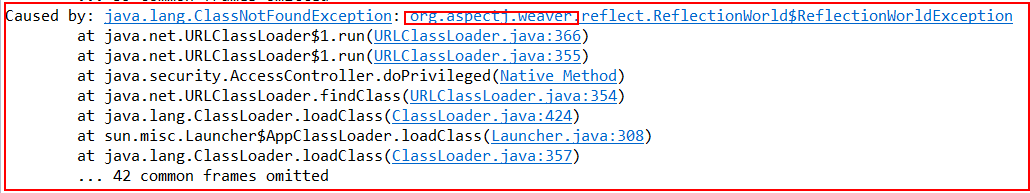
<aop:advisor pointcut-ref=*"transactionPointcut"*

advice-ref=*"transactionAdvice"* />

</aop:config>

</beans>

启动报错，



<!-- 导入依赖来支持spring管理事务 -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

</dependency>

再次启动，启动正常。

在Mapper中添加方法。

@Delete(" delete from tbl\_study\_table1 where id = #{id}")

**public** **long** deleteById(**long** id);

在StudyTableService中添加

**public** **long** deleteById(**long** id);

在StudyTableServiceImpl中添加

@Override

**public** **long** deleteById(**long** id) {

System.***out***.println("Before执行删除。。");

**long** result = studyTableMapper.deleteById(id);

**int** i = 10/0;

System.***out***.println("After执行删除。。");

**return** result;

}

在HelloWorldController中添加

@RequestMapping(path="/deleteById/{id}",method={RequestMethod.***GET***})

@ResponseBody

**public** String deleteById(@PathVariable("id") **long** id){

System.***out***.println("进入deleteById方法:" + id);

**long** resultNum = studyTableService.deleteById(id);

System.***out***.println("受影响条数:" + resultNum);

**return** "success";

}

使用swaggerUI进行测试：

首先调用getById方法，输入500，查询结果：



在调用删除方法，发现抛出异常：



再次查询：



依然查的到，说明spring已经对事务进行了管理。

## 3.4 Springboot整合reids

导入pom依赖

<!-- springboot 整合 redis -->

<dependency>

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

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

</dependency>

添加application.properties配置文件

#redis

spring.redis.host=192.168.57.101

spring.redis.port=6379

spring.redis.database=0

spring.redis.pool.maxActive=8

spring.redis.pool.maxWait=-1

spring.redis.pool.maxIdle=8

spring.redis.pool.minIdle=0

spring.redis.timeout=0

新建src/main/java/controller/RedisController

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

**import** org.springframework.data.redis.core.StringRedisTemplate;

**import** org.springframework.data.redis.core.ValueOperations;

**import** org.springframework.web.bind.annotation.PathVariable;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** com.cxn.model.StudyTableModel;

**import** com.cxn.service.StudyTableService;

**import** com.fasterxml.jackson.databind.ObjectMapper;

@RestController

@RequestMapping(path = "/redis")

**public** **class** RedisController {

@Autowired

**private** StudyTableService studyTableService;

@Autowired

**private** StringRedisTemplate stringRedisTemplate;

@RequestMapping(path="/findByCache/{id}",method={RequestMethod.***GET***})

**public** StudyTableModel findByCache(@PathVariable("id") **long** id) **throws** Exception{

System.***out***.println("进入方法:findByCache/"+id);

ValueOperations<String, String>ops = stringRedisTemplate.opsForValue();

String modelJson = ops.get(String.*valueOf*(id));

ObjectMapper mapper = **new** ObjectMapper();

**if** (modelJson == **null**) {

System.***out***.println("缓存中没有值,从数据库中查询~");

StudyTableModel model = studyTableService.getByIdUseMapper(id);

modelJson = mapper.writeValueAsString(model);

ops.set(String.*valueOf*(id), modelJson);

**return** model;

}

System.***out***.println("从redis中取出缓存~");

StudyTableModel tableModel = mapper.readValue(modelJson, StudyTableModel.**class**);

**return** tableModel;

}

@RequestMapping(path="/deleteByCache/{id}",method={RequestMethod.***GET***})

**public** String deleteByCache(@PathVariable("id") **long** id){

System.***out***.println("进入方法:findByCache/"+id);

**try** {

stringRedisTemplate.delete(String.*valueOf*(id));

} **catch** (Exception e) {

e.printStackTrace();

**return**"fail";

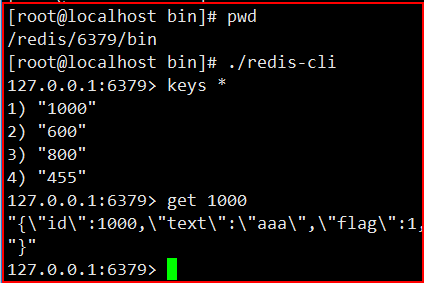
}

**return**"success";

}

}

使用SwaggerUI访问:(打开德鲁伊sql监控,打开redis客户端)

演示使用redisTemplate存储对象乱码问题及解答:

在RedisController中添加方法

@Autowired

**private** RedisTemplate<String, Object> redisTemplate;

@RequestMapping(path="/findByCache2/{id}",method={RequestMethod.***GET***})

**public** StudyTableModel findByCache2(@PathVariable("id") **long** id) **throws** Exception{

System.***out***.println(">>>>>>>>>>>>>>>>>>进入方法:findByCache2/"+id);

ValueOperations<String, Object>opsForValue = redisTemplate.opsForValue();

Object object = opsForValue.get(String.*valueOf*(id));

**if** (object == **null**) {

System.***out***.println("缓存中没有值~");

StudyTableModel model = studyTableService.getById(id);

opsForValue.set(String.*valueOf*(id), model);

**return** model;

}**else**{

System.***out***.println("从redis中取出缓存~");

StudyTableModel model1 = (StudyTableModel)object;

**return** model1;

}

}

@RequestMapping(path="/deleteByCache2/{id}",method={RequestMethod.***GET***})

**public** String deleteByCache2(@PathVariable("id") **long** id){

System.***out***.println(">>>>>>>>>>>>>>>>>>进入方法:deleteByCache2/"+id);

**try** {

redisTemplate.delete(String.*valueOf*(id));

} **catch** (Exception e) {

e.printStackTrace();

**return**"fail";

}

**return**"success";

}

新增RedisConfig

**import** org.springframework.boot.autoconfigure.condition.ConditionalOnClass;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.core.annotation.Order;

**import** org.springframework.data.redis.connection.RedisConnectionFactory;

**import** org.springframework.data.redis.core.RedisTemplate;

**import** org.springframework.data.redis.core.StringRedisTemplate;

**import** org.springframework.data.redis.serializer.StringRedisSerializer;

@Order(value=Integer.***MIN\_VALUE***+3)

@Configuration

@ConditionalOnClass(RedisConnectionFactory.**class**)

**public** **class** RedisConfig {

@Bean

**public** RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory){

RedisTemplate<String, Object>redisTemplate =**new** RedisTemplate<String, Object>();

redisTemplate.setConnectionFactory(redisConnectionFactory);

// 设置redisTemplate key的序列化器，默认使用的是原生Java，

// 因为在redis上存放Object对象是通过对象的序列化方式实现的，所以如果使用redisTemplate存放pojo类的话，必须实现serializable接口

// redisTemplate.setKeySerializer(new StringRedisSerializer());

redisTemplate.setEnableTransactionSupport(**false**);

System.***out***.println("redisTemplate初始化成功");

**return** redisTemplate;

}

@Bean

**public** StringRedisTemplate stringRedisTemplate(RedisConnectionFactory redisConnectionFactory){

StringRedisTemplate stringRedisTemplate =**new** StringRedisTemplate();

stringRedisTemplate.setConnectionFactory(redisConnectionFactory);

stringRedisTemplate.setKeySerializer(**new** StringRedisSerializer());

stringRedisTemplate.setEnableTransactionSupport(**false**);

**return** stringRedisTemplate;

}

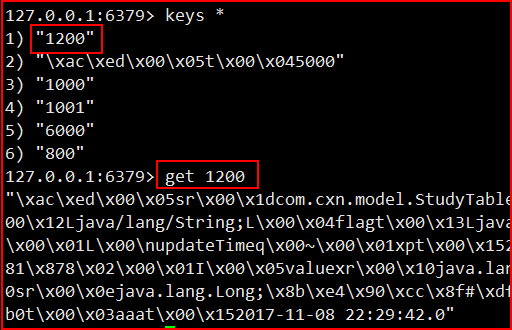
}

使用SwaggerUI进行访问：





发现存放的key变作了乱码形式，这是因为redisTemplate把key也序列化了，这个时候只要将redisTemplate的key的序列化方式指定一下即可，即：RedisConfig的注释放开。



## 3.5 Springboot整合MongoDB

<!-- mongoDB -->

<dependency>

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

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

</dependency>

application.properties中添加配置

#mongodb

#spring.data.mongodb.uri=mongodb://trans\_code:trans\_code@192.168.21.215:27017/trans\_code\_tasks

spring.data.mongodb.uri=mongodb://192.168.57.101:27017/trans\_code\_tasks

新建MongoControllers

**import** java.util.List;

**import** java.util.regex.Pattern;

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

**import** org.springframework.data.mongodb.core.MongoTemplate;

**import** org.springframework.data.mongodb.core.query.Criteria;

**import** org.springframework.data.mongodb.core.query.Query;

**import** org.springframework.data.mongodb.core.query.Update;

**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.RequestBody;

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

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

**import** com.cxn.model.User;

**import** com.mongodb.WriteResult;

@Controller

**public** **class** MongoController {

@Autowired

**private** MongoTemplate mongoTemplate;

@PostMapping(path="/saveUser")

@ResponseBody

**public** String saveUser(@RequestBody User user){

System.***out***.println(user);

mongoTemplate.save(user);

**return**"success";

}

@PostMapping(path="/removeUserByName")

@ResponseBody

**public** WriteResult removeUserByName(@RequestParam String username){

System.***out***.println("username:"+username);

Criteria criteria = **new** Criteria();

criteria.andOperator(Criteria.*where*("username").is(username));

WriteResult result = mongoTemplate.remove(**new** Query(criteria), User.**class**);

System.***out***.println("WriteResult:"+result);

**return** result;

}

@PostMapping(path="/updateUserByName")

@ResponseBody

**public** WriteResult updateUserByName(@RequestParam("username") String username, @RequestParam("updateName") String updateName){

System.***out***.println("username:"+username);

Criteria criteria = **new** Criteria();

criteria.andOperator(Criteria.*where*("username").is(username));

Update update = **new** Update();

update.set("username", updateName);

WriteResult result = mongoTemplate.updateFirst(**new** Query(criteria), update, User.**class**);

System.***out***.println("WriteResult:"+result);

// mongoTemplate.updateMulti(query, update, entityClass);

**return** result;

}

@GetMapping(path="/getUserListByName")

@ResponseBody

**public** List<User> getUserListByName(@RequestParam String username){

System.***out***.println("username:"+username);

Criteria criteria = **new** Criteria();

criteria.andOperator(Criteria.*where*("username").is(username));

Query query = **new** Query(criteria);

List<User>userList = mongoTemplate.find(query, User.**class**);

**return** userList;

}

@GetMapping(path="/getOneUserByName")

@ResponseBody

**public** User getOneUserByName(@RequestParam String username){

System.***out***.println("username:"+username);

Criteria criteria = **new** Criteria();

criteria.andOperator(Criteria.*where*("username").is(username));

User user = mongoTemplate.findOne(**new** Query(criteria), User.**class**);

System.***out***.println("User:"+user);

**return** user;

}

// 模糊查询

@GetMapping(path="/getUserByNameSample")

@ResponseBody

**public** List<User> getUserByNameSample(@RequestParam String username){

System.***out***.println("username:"+username);

Criteria criteria = **new** Criteria();

Pattern pattern = Pattern.*compile*("^.\*" + username + ".\*$", Pattern.***CASE\_INSENSITIVE***);

criteria.orOperator(Criteria.*where*("username").regex(pattern));

Query query = **new** Query(criteria);

query.skip(0); // 从哪条记录开始(第一条数据是0)

query.limit(12);// 取多少条记录

List<User>userList = mongoTemplate.find(query, User.**class**);

**long** count = mongoTemplate.count(query, User.**class**);

System.***out***.println(count);

**return** userList;

}

}

新建model

**public** **class** User {

**private** String username;

**private** String password;

**private** **int** age;

**private** **double** height;

// 省略get set方法

}

使用swaggerUi进行测试。

## 3.6 Springboot整合ActiveMQ

导入pom依赖

<!--引入该jar包,可以使用默认的ActiveMQ -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jms</artifactId>

</dependency>

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-all</artifactId>

<version>5.13.2</version>

</dependency>

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-pool</artifactId>

</dependency>

application.properties添加配置

#activeMQ

spring.activemq.broker-url=tcp://192.168.57.101:61616

spring.activemq.username=admin

spring.activemq.pad=admin

spring.activemq.in-memory=true

spring.activemq.pool.enabled=false

新增ActiveMQConfig

**import** java.util.Properties;

**import** org.apache.activemq.ActiveMQConnectionFactory;

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

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.jms.annotation.EnableJms;

@EnableJms

@Configuration

**public** **class** ActiveMQConfig {

@Value("${spring.activemq.broker-url}")

**private** String brokerUrl;

@Bean

**public** ActiveMQConnectionFactory activeMQConnectionFactory (){

Properties props = **new** Properties();

props.setProperty("prefetchPolicy.queuePrefetch", "1000");

props.setProperty("prefetchPolicy.queueBrowserPrefetch", "500");

props.setProperty("prefetchPolicy.durableTopicPrefetch", "100");

props.setProperty("prefetchPolicy.topicPrefetch", "32766");

ActiveMQConnectionFactory activeMQConnectionFactory =

**new** ActiveMQConnectionFactory(

ActiveMQConnectionFactory.***DEFAULT\_USER***,

ActiveMQConnectionFactory.***DEFAULT\_PASSWORD***,

ActiveMQConnectionFactory.***DEFAULT\_BROKER\_URL***);

activeMQConnectionFactory.setProperties(props);

activeMQConnectionFactory.setBrokerURL(brokerUrl);

**return** activeMQConnectionFactory;

}

}

新增消息发送方MyMessageProducer

**import** javax.jms.Destination;

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

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.jms.core.JmsMessagingTemplate;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** MyMessageProducer **implements** CommandLineRunner{

@Autowired// 也可以注入JmsTemplate，JmsMessagingTemplate对JmsTemplate进行了封装

**private** JmsMessagingTemplate jmsTemplate;

// 发送消息，destination是发送到的队列，message是待发送的消息

**public** **void** sendMessage(Destination destination, **final** String message){

jmsTemplate.convertAndSend(destination, message);

}

@Override

**public** **void** run(String... args) **throws** Exception {

System.***out***.println(">>>>>>>>>消息发送方启动。。。");

}

}

新增消息消费方MyMessageConsumer

**import** org.springframework.jms.annotation.JmsListener;

**import** org.springframework.messaging.handler.annotation.SendTo;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** MyMessageConsumer {

// 使用JmsListener配置消费者监听的队列，其中text是接收到的消息

@JmsListener(destination = "test.queue")

@SendTo("out.queue")// 实现双向队列

**public** String receiveQueue(String text) {

// **TODO** do something

System.***out***.println("Consumer--A-->>收到的报文为:"+text);

**return**"out>>" + text;

}

@JmsListener(destination = "out.queue")

**public** **void** receiveQueue1(String text) {

// **TODO** do something

System.***out***.println("Consumer--B-->>收到的报文为:"+text);

}

}

新建ActivemqController

**import** javax.jms.Destination;

**import** org.apache.activemq.command.ActiveMQQueue;

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

**import** org.springframework.web.bind.annotation.PathVariable;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** com.cxn.activemq.MyMessageProducer;

@RestController

@RequestMapping(path="/activemq")

**public** **class** ActivemqController {

@Autowired

**private** MyMessageProducer messageProducer;

@RequestMapping(path="/send/{message}",method={RequestMethod.***GET***})

**public** String testSendMessage(@PathVariable String message){

Destination destination = **new** ActiveMQQueue("test.queue");

// 发送消息

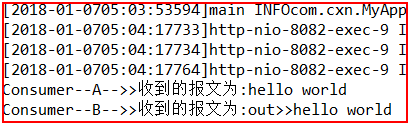
messageProducer.sendMessage(destination , message);

**return**"success";

}

}

使用swaggerUI测试：

## 3.7 Springboot整合Elastic-Job

导入pom依赖

<!-- elastic-job 定时任务框架 -->

<dependency>

<groupId>com.dangdang</groupId>

<artifactId>elastic-job-lite-core</artifactId>

<version>2.1.5</version>

</dependency>

<dependency>

<groupId>com.dangdang</groupId>

<artifactId>elastic-job-lite-spring</artifactId>

<version>2.1.5</version>

</dependency>

<!-- zookeeper -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.4.9</version>

</dependency>

新建src/main/java/com/cxn/job/ScheduleJob

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** com.dangdang.ddframe.job.api.ShardingContext;

**import** com.dangdang.ddframe.job.api.simple.SimpleJob;

**public** **class** ScheduleJob **implements** SimpleJob{

@Override

**public** **void** execute(ShardingContext shardingContext) {

System.***out***.println("开始执行定时任务 " + **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss E").format(**new** Date()));

}

}

新建src/main/resources/elastic-job.xml

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:reg=*"http://www.dangdang.com/schema/ddframe/reg"*

xmlns:job=*"http://www.dangdang.com/schema/ddframe/job"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.dangdang.com/schema/ddframe/reg*

*http://www.dangdang.com/schema/ddframe/reg/reg.xsd*

*http://www.dangdang.com/schema/ddframe/job*

*http://www.dangdang.com/schema/ddframe/job/job.xsd"*>

<!-- 配置注册中心 ，任务的信息都会在zk中存储 -->

<reg:zookeeper id=*"regCenter"* server-lists=*"192.168.57.101:2181"*

namespace=*"test-job"* base-sleep-time-milliseconds=*"1000"*

max-sleep-time-milliseconds=*"3000"* max-retries=*"3"*/>

<!-- 配置简单作业 -->

<!-- 分片为1，即不需要分片；支持覆盖，即会用本次的配置覆盖缓存在zk中的配置 -->

<job:simple id=*"testTask"* class=*"com.cxn.job.ScheduleJob"*

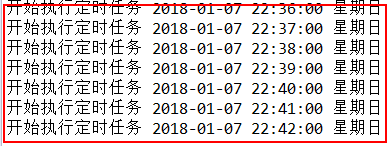
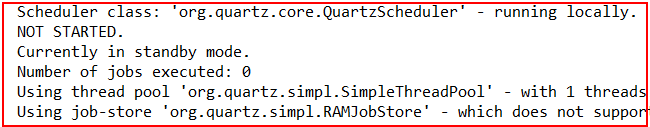
registry-center-ref=*"regCenter"* cron=*"0 0/1 \* \* \* ?"*

sharding-total-count=*"1"* overwrite=*"true"*>

</job:simple>

</beans>

启动项目：



出现图中字样，说明整合成功。

## 3.8 Springboot整合OSS

导入pom依赖

<!-- 阿里云OSS -->

<dependency>

<groupId>com.aliyun.oss</groupId>

<artifactId>aliyun-sdk-oss</artifactId>

<version>2.0.6</version>

</dependency>

新建OssController

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.springframework.stereotype.Controller;

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

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

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

**import** com.aliyun.oss.OSSClient;

**import** com.aliyun.oss.model.OSSObjectSummary;

**import** com.aliyun.oss.model.ObjectListing;

@Controller

@RequestMapping(path="/oss")

**public** **class** OssController {

@GetMapping(path="/getNameList")

@ResponseBody

**public** List<String> getNameList(){

String endpoint = "\*\*\*\*";

String accessId = "\*\*\*\*";

String accessKey = "\*\*\*\*";

// 初始化一个OSSClient

OSSClient client = **new** OSSClient(endpoint, accessId, accessKey);

String bucketName = "osstest";

// 获取指定bucket下的所有Object信息

ObjectListing listing = client.listObjects(bucketName);

List<String>nameList = **new** ArrayList<>();

// 遍历所有Object

**for** (OSSObjectSummary objectSummary : listing.getObjectSummaries()) {

System.***out***.println(objectSummary.getKey());

nameList.add(objectSummary.getKey());

}

**return** nameList;

}

}

其它具体实现需要自己查看官方API

<http://aliyun_portal_storage.oss.aliyuncs.com/oss_api/oss_javahtml/index.html>

## 3.9 Springboot整合Dubbo

在springboot-study-api中新建service接口

**public** **interface** DubboService {

**public** String getResult();

}

然后分别在service和web中引入api模块。过程略。

新建maven module项目springboot-study-service作为服务提供方，

分别在服务提供方和服务调用方导入pom依赖

<!-- dubbo依赖 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.8.4</version>

</dependency>

<!-- dubbo传输采用kryo序列化 -->

<dependency>

<groupId>com.esotericsoftware.kryo</groupId>

<artifactId>kryo</artifactId>

<version>2.24.0</version>

</dependency>

<dependency>

<groupId>de.javakaffee</groupId>

<artifactId>kryo-serializers</artifactId>

<version>0.26</version>

</dependency>

<dependency>

<groupId>org.javassist</groupId>

<artifactId>javassist</artifactId>

</dependency>

<dependency>

<groupId>org.jboss.netty</groupId>

<artifactId>netty</artifactId>

<version>3.2.10.Final</version>

</dependency>

<!-- dubbo依赖的zk客户端 -->

<dependency>

<groupId>com.101tec</groupId>

<artifactId>zkclient</artifactId>

<version>0.10</version>

<exclusions>

<exclusion>

<artifactId>netty</artifactId>

<groupId>io.netty</groupId>

</exclusion>

<exclusion>

<artifactId>slf4j-log4j12</artifactId>

<groupId>org.slf4j</groupId>

</exclusion>

</exclusions>

</dependency>

服务提供方application.properties

# server

server.port=8083

server.servlet-path=/

## Dubbo provider config

# application

dubbo.application.name=server-provider

# registry

dubbo.registry.address=zookeeper://192.168.57.101:2181

dubbo.registry.file=cache/server-provider.cache

# protocol

dubbo.protocol.port=28091

#annotation

dubbo.annotation.package=com.cxn

#provider

dubbo.provider.timeout=30000

dubbo.provider.retries=0

新建src/main/java/com/cxn/service/impl/DubboServiceImpl

**import** com.alibaba.dubbo.config.annotation.Service;

**import** com.cxn.service.DubboService;

@Service(version="1.0.0",protocol="dubbo")

**public** **class** DubboServiceImpl **implements** DubboService {

@Override

**public** String getResult() {

**return** "remote success";

}

}

新建服务提供者的src/main/resources/spring/dubbo.xml

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

<!-- - Copyright 1999-2011 Alibaba Group. - - Licensed under the Apache License,

Version 2.0 (the "License"); - you may not use this file except in compliance

with the License. - You may obtain a copy of the License at - - http://www.apache.org/licenses/LICENSE-2.0

- - Unless required by applicable law or agreed to in writing, software -

distributed under the License is distributed on an "AS IS" BASIS, - WITHOUT

WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. - See the

License for the specific language governing permissions and - limitations

under the License. -->

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://code.alibabatech.com/schema/dubbo*

*http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<dubbo:application name=*"${dubbo.application.name}"* organization=*"dubbox"* logger=*"slf4j"*/>

<dubbo:registry address=*"${dubbo.registry.address}"* file=*"${dubbo.registry.file}"* />

<dubbo:annotation package=*"${dubbo.annotation.package}"* />

<dubbo:protocol name=*"dubbo"* serialization=*"kryo"* port=*"${dubbo.protocol.port}"* />

<!-- dubbo服务缺省设置 -->

<dubbo:provider timeout=*"${dubbo.provider.timeout}"* retries=*"${dubbo.provider.retries}"* />

<dubbo:consumer check=*"false"* />

</beans>

追加服务消费者springboot-study-web的application.properties

## Dubbo consumer config

# application

dubbo.application.name=server-customer

# registry

dubbo.registry.address=zookeeper://192.168.57.101:2181

dubbo.registry.file=cache/server-customer.cache

# protocol

dubbo.protocol.port=28092

#annotation

dubbo.annotation.package=com.cxn

#provider

dubbo.provider.timeout=600000

dubbo.provider.retries=0

新建消费者/src/main/resources/spring/dubbo.xml

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

<!-- - Copyright 1999-2011 Alibaba Group. - - Licensed under the Apache License,

Version 2.0 (the "License"); - you may not use this file except in compliance

with the License. - You may obtain a copy of the License at - - http://www.apache.org/licenses/LICENSE-2.0

- - Unless required by applicable law or agreed to in writing, software -

distributed under the License is distributed on an "AS IS" BASIS, - WITHOUT

WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. - See the

License for the specific language governing permissions and - limitations

under the License. -->

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://code.alibabatech.com/schema/dubbo*

*http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<dubbo:application name=*"${dubbo.application.name}"* organization=*"dubbox"* logger=*"slf4j"*/>

<dubbo:registry address=*"${dubbo.registry.address}"* file=*"${dubbo.registry.file}"* />

<dubbo:annotation package=*"${dubbo.annotation.package}"* />

<dubbo:protocol name=*"dubbo"* serialization=*"kryo"* port=*"${dubbo.protocol.port}"* />

<!-- dubbo服务缺省设置 -->

<dubbo:provider timeout=*"${dubbo.provider.timeout}"* retries=*"${dubbo.provider.retries}"* />

<dubbo:consumer check=*"false"* />

</beans>

消费方新建src/main/java/com/cxn/controller/DubboController

**import** org.springframework.stereotype.Controller;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** com.alibaba.dubbo.config.annotation.Reference;

**import** com.cxn.service.DubboService;

@Controller

@RequestMapping(path="/dubbo")

**public** **class** DubboController {

@Reference(version = "1.0.0")

**private** DubboService dubboService;

@ResponseBody

@RequestMapping(path="/test",method={RequestMethod.***GET***})

**public** String testDubbo(){

System.***out***.println(">>>>>>>>>>>>>"+dubboService);

**return** dubboService.getResult();

}

}

使用swagger-ui访问测试



## 3.10 Springboot整合邮件email

结合模板引擎一起

## 3.11 Springboot整合模板引擎velocity/freemarker

新建springboot-study-api中service

**import** java.io.File;

**import** java.util.Map;

**public** **interface** EmailService {

/\*\*

\* 发送简单邮件

\* **@param** sendTo 收件人地址

\* **@param** titel 邮件标题

\* **@param** content 邮件内容

\*/

**public** **void** sendSimpleMail(String sendTo, String titel, String content);

/\*\*

\* 发送带附件邮件

\* **@param** sendTo 收件人地址

\* **@param** titel 邮件标题

\* **@param** content 邮件内容

\* **@param** attachments<文件名，附件> 附件列表

\*/

**public** **void** sendAttachmentsMail(String sendTo, String titel, String content, Map<String, File> attachments);

/\*\*

\* 发送模板邮件 使用velocity

\* **@param** sendTo 收件人地址

\* **@param** titel 邮件标题

\* **@param** content<key, 内容> 邮件内容

\* **@param** attachments<文件名，附件> 附件列表

\*/

**public** **void** sendTemplateMail(String sendTo, String titel, Map<String, Object> content, Map<String, File> attachments);

/\*\*

\* 发送html样式邮件

\* **@param** sendTo 收件人地址

\* **@param** titel 邮件标题

\* **@param** html html格式邮件内容

\*/

**public** **void** sendHtmlMail(String sendTo, String titel, String html);

/\*\*

\* 发送带静态资源的邮件

\* **@param** sendTo 收件人地址

\* **@param** titel 邮件标题

\*/

**public** **void** sendInlineMail(String sendTo, String titel);

/\*\*

\* 发送模板邮件 使用 freemarker

\* **@param** sendTo 收件人地址

\* **@param** titel 邮件标题

\* **@param** content<key, 内容> 邮件内容

\* **@param** attachments<文件名，附件> 附件列表

\*/

**void** sendTemplateMailFreemarker(String sendTo, String titel, Map<String, Object> content, Map<String, File> attachments);

}

springboot-study-service导入pom依赖

<!-- 邮件发送 -->

<dependency>

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

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

</dependency>

<!-- 模板引擎 -->

<dependency>

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

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

<version>1.4.1.RELEASE</version>

</dependency>

<dependency>

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

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

</dependency>

然后追加application.properties

# mail

spring.mail.host=smtp.163.com

spring.mail.username=caoxunan121@163.com

spring.mail.password=password

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

spring.mail.properties.mail.smtp.starttls.required=true

# FREEMARKER (FreeMarkerAutoConfiguration)

spring.freemarker.charset=UTF-8

spring.freemarker.content-type=text/html

spring.freemarker.suffix=.ftl

spring.freemarker.template-loader-path=classpath:/template/

spring.freemarker.settings.default\_encoding=UTF-8

新建EmailService的实现类

**import** java.io.File;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** javax.mail.internet.MimeMessage;

**import** org.apache.velocity.app.VelocityEngine;

**import** org.apache.velocity.runtime.RuntimeConstants;

**import** org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader;

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

**import** org.springframework.core.io.FileSystemResource;

**import** org.springframework.mail.SimpleMailMessage;

**import** org.springframework.mail.javamail.JavaMailSender;

**import** org.springframework.mail.javamail.MimeMessageHelper;

**import** org.springframework.ui.freemarker.FreeMarkerTemplateUtils;

**import** org.springframework.ui.velocity.~~VelocityEngineUtils~~;

**import** org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;

**import** com.alibaba.dubbo.config.annotation.Service;

**import** com.cxn.config.EmailConfig;

**import** com.cxn.service.EmailService;

**import** freemarker.template.Template;

@SuppressWarnings("deprecation")

@Service(version="1.0.0",protocol="dubbo")

**public** **class** EmailServiceImpl **implements** EmailService {

@Autowired

**private** EmailConfig emailConfig;

@Autowired

**private** JavaMailSender mailSender;

@Autowired

**private** FreeMarkerConfigurer freeMarkerConfigurer;

**private** **static** VelocityEngine *velocityEngine*;

**static** {

*velocityEngine* = **new** VelocityEngine();

*velocityEngine*.setProperty("input.encoding", "UTF-8");

*velocityEngine*.setProperty("output.encoding", "UTF-8");

*velocityEngine*.setProperty(RuntimeConstants.***RESOURCE\_LOADER***, "classpath");

*velocityEngine*.setProperty("classpath.resource.loader.class", ClasspathResourceLoader.**class**.getName());

*velocityEngine*.init();

}

@Override

**public** **void** sendSimpleMail(String sendTo, String titel, String content) {

SimpleMailMessage message = **new** SimpleMailMessage();

message.setFrom(emailConfig.getEmailFrom());

message.setTo(sendTo);

message.setSubject(titel);

message.setText(content);

mailSender.send(message);

}

@Override

**public** **void** sendHtmlMail(String sendTo, String titel, String html) {

MimeMessage message = **null**;

**try** {

message = mailSender.createMimeMessage();

MimeMessageHelper helper = **new** MimeMessageHelper(message, **true**);

helper.setFrom(emailConfig.getEmailFrom());

helper.setTo(sendTo);

helper.setSubject(titel);

helper.setText(html, **true**);

} **catch** (Exception e) {

e.printStackTrace();

}

mailSender.send(message);

}

@Override

**public** **void** sendAttachmentsMail(String sendTo, String titel, String content, Map<String, File> attachments) {

MimeMessage mimeMessage = mailSender.createMimeMessage();

**try** {

MimeMessageHelper helper = **new** MimeMessageHelper(mimeMessage, **true**);

helper.setFrom(emailConfig.getEmailFrom());

helper.setTo(sendTo);

helper.setSubject(titel);

helper.setText(content);

**if** (**null** != attachments) {

**for** (Entry<String, File> entry : attachments.entrySet()) {

helper.addAttachment(entry.getKey(), **new** FileSystemResource(entry.getValue()));

}

}

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

mailSender.send(mimeMessage);

}

@Override

**public** **void** sendInlineMail(String sendTo, String titel) {

MimeMessage mimeMessage = mailSender.createMimeMessage();

**try** {

MimeMessageHelper helper = **new** MimeMessageHelper(mimeMessage, **true**);

helper.setFrom(emailConfig.getEmailFrom());

helper.setTo(sendTo);

helper.setSubject("主题：带静态资源的邮件");

//第二个参数指定发送的是HTML格式,同时cid:是固定的写法

helper.setText("<html><body>带静态资源的邮件内容 图片:<img src='cid:picture' /></body></html>", **true**);

FileSystemResource file = **new** FileSystemResource(**new** File("src/main/resources/template/github.jpg"));

helper.addInline("picture",file);

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

mailSender.send(mimeMessage);

}

@Override

**public** **void** sendTemplateMail(String sendTo, String titel, Map<String, Object> content, Map<String, File> attachments) {

MimeMessage mimeMessage = mailSender.createMimeMessage();

**try** {

MimeMessageHelper helper = **new** MimeMessageHelper(mimeMessage, **true**);

helper.setFrom(emailConfig.getEmailFrom());

helper.setTo(sendTo);

helper.setSubject(titel);

String text = ~~VelocityEngineUtils~~.~~mergeTemplateIntoString~~(*velocityEngine*, "template/email.vm", "UTF-8", content);

helper.setText(text, **true**);

**if** (**null** != attachments) {

**for** (Entry<String, File> entry : attachments.entrySet()) {

helper.addAttachment(entry.getKey(), **new** FileSystemResource(entry.getValue()));

}

}

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

mailSender.send(mimeMessage);

}

@Override

**public** **void** sendTemplateMailFreemarker(String sendTo, String titel, Map<String, Object> content, Map<String, File> attachments) {

MimeMessage mimeMessage = mailSender.createMimeMessage();

**try** {

MimeMessageHelper helper = **new** MimeMessageHelper(mimeMessage, **true**);

helper.setFrom(emailConfig.getEmailFrom());

helper.setTo(sendTo);

helper.setSubject(titel);

//修改 application.properties 文件中的读取路径

// FreeMarkerConfigurer configurer = new FreeMarkerConfigurer();

// configurer.setTemplateLoaderPath("classpath:templates");

//读取 html 模板

Template template = freeMarkerConfigurer.getConfiguration().getTemplate("email.vm");

String html = FreeMarkerTemplateUtils.*processTemplateIntoString*(template, content);

helper.setText(html, **true**);

**if** (**null** != attachments) {

File file1 = **new** File("src/main/resources/template/李小龙.gif");

File file2 = **new** File("src/main/resources/template/github.jpg");

attachments.put("图片1.jpg", file1);

attachments.put("图片2.jpg", file2);

**for** (Entry<String, File> entry : attachments.entrySet()) {

helper.addAttachment(entry.getKey(), **new** FileSystemResource(entry.getValue()));

}

}

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

mailSender.send(mimeMessage);

}

}

新建src/main/java/com/cxn/config/EmailConfig

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

**import** org.springframework.stereotype.Component;

@Component

**public** **class** EmailConfig {

/\*\*

\* 发件邮箱，通过set方法将只注入bean中

\*/

@Value("${spring.mail.username}")

**private** String emailFrom;

**public** String getEmailFrom() {

**return** emailFrom;

}

**public** **void** setEmailFrom(String emailFrom) {

**this**.emailFrom = emailFrom;

}

}

springboot-study-web新增DemoEmailController

**import** java.io.File;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.TreeMap;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** io.swagger.annotations.ApiOperation;

**import** com.alibaba.dubbo.config.annotation.Reference;

**import** com.cxn.service.EmailService;

@RestController

@RequestMapping(value="/email")

**public** **class** DemoEmailController {

@Reference(version="1.0.0")

**private** EmailService emailService;

@ApiOperation(value="测试普通邮件发送", notes="普通邮件发送")

@RequestMapping(value = "/sendSimpleMail", method = RequestMethod.***GET***)

**public** String sendSimpleMail() **throws** Exception {

String sendTo = "caoxunan121@163.com";

String titel = "测试邮件标题";

String content = "测试邮件内容";

emailService.sendSimpleMail(sendTo, titel, content);

**return** "success";

}

@ApiOperation(value="html样式邮件发送", notes="html样式邮件发送")

@RequestMapping(value = "/sendHtmlEmail", method = RequestMethod.***GET***)

**public** String sendHtmlEmail(){

String sendTo = "caoxunan121@163.com";

String titel = "标题：发送Html内容";

// 拼接html代码

StringBuffer sb = **new** StringBuffer();

sb.append("<h1>大标题-h1</h1>")

.append("<p style='color:#F00'>红色字</p>")

.append("<p style='text-align:right'>右对齐</p>");

String content = sb.toString();

emailService.sendHtmlMail(sendTo, titel, content);

**return** "success";

}

@ApiOperation(value="发送带附件的邮件", notes="发送带附件的邮件")

@RequestMapping(value = "/sendAttachmentsMail", method = RequestMethod.***GET***)

**public** String sendAttachmentsMail(){

String sendTo = "caoxunan121@163.com";

String titel = "标题：发送带附件的邮件";

String content = "邮件内容。。。。。";

Map<String, File> attachments = **new** TreeMap<>();

emailService.sendAttachmentsMail(sendTo, titel, content, attachments);

**return** "success";

}

@ApiOperation(value="发送带静态资源的邮件", notes="发送带静态资源的邮件")

@RequestMapping(value = "/sendInlineMail", method = RequestMethod.***GET***)

**public** String sendInlineMail(){

String sendTo = "caoxunan121@163.com";

String titel = "标题：发送带附件的邮件";

emailService.sendInlineMail(sendTo, titel);

**return** "success";

}

@ApiOperation(value="发送模板邮件", notes="发送模板邮件")

@RequestMapping(value = "/sendTemplateMailVelocity", method = RequestMethod.***GET***)

**public** String sendTemplateMailVelocity(){

String sendTo = "caoxunan121@163.com";

String titel = "标题：发送模板邮件";

Map<String, Object> content = **new** HashMap<>();

content.put("orderId", "JJ-CD-180104-001");

content.put("runState", "Velocity");

emailService.sendTemplateMail(sendTo, titel, content , **null**);

**return** "success";

}

@ApiOperation(value="发送模板邮件", notes="发送模板邮件")

@RequestMapping(value = "/sendTemplateMailFreemarker", method = RequestMethod.***GET***)

**public** String sendTemplateMailFreemarker(){

String sendTo = "caoxunan121@163.com";

String titel = "标题：发送模板邮件";

Map<String, Object> content = **new** HashMap<>();

content.put("orderId", "JJ-CD-180104-001");

content.put("runState", "Freemarker");

emailService.sendTemplateMailFreemarker(sendTo, titel, content , **null**);

**return** "success";

}

}

使用swagger-UI进行测试邮件功能：



追加web的application.properties配置：(根据需要决定放在web层或service层)

# FREEMARKER (FreeMarkerAutoConfiguration)

spring.freemarker.charset=UTF-8

spring.freemarker.content-type=text/html

spring.freemarker.suffix=.ftl

spring.freemarker.template-loader-path=classpath:/template/

spring.freemarker.settings.default\_encoding=UTF-8

将service中的template/email.vm复制到web的template文件夹下。

新增TemplateController

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.Writer;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.apache.velocity.Template;

**import** org.apache.velocity.VelocityContext;

**import** org.apache.velocity.app.VelocityEngine;

**import** org.apache.velocity.runtime.RuntimeConstants;

**import** org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader;

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

**import** org.springframework.stereotype.Controller;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;

@Controller

@RequestMapping(path="/template")

**public** **class** TemplateController {

@Autowired

**private** FreeMarkerConfigurer freeMarkerConfigurer;

@RequestMapping(path="/velocity",method = {RequestMethod.***GET***})

@ResponseBody

**public** String generateStaticFileVelocity(){

// 初始化模板引擎

VelocityEngine ve = **new** VelocityEngine();

ve.setProperty(RuntimeConstants.***RESOURCE\_LOADER***, "classpath");

ve.setProperty("classpath.resource.loader.class", ClasspathResourceLoader.**class**.getName());

ve.init();

// 获取模板文件

Template t = ve.getTemplate("template/email.vm", "UTF-8");

// 设置变量

VelocityContext context = **new** VelocityContext();

context.put("orderId", "JJ-CD-180104-001");

context.put("runState", "Velocity");

// 输出

Writer writer;

**try** {

writer = **new** FileWriter(**new** File("email-velocity.html"));

t.merge(context,writer);

writer.flush();

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

**throw** **new** RuntimeException(e.getMessage());

}

**return** "success";

}

@RequestMapping(path="/freemaker",method = {RequestMethod.***GET***})

@ResponseBody

**public** String generateStaticFileFreemarker(){

**try** {

// 加载模板文件

freemarker.template.Template template = freeMarkerConfigurer.getConfiguration().getTemplate("email.vm");

// 设置输出文件的位置

FileWriter fileWriter = **new** FileWriter(**new** File("email-freemarker.html"));

Map<String, Object> context = **new** HashMap<>();

context.put("orderId", "JJ-CD-180104-001");

context.put("runState", "Freemarker");

// 开始输出

template.process(context, fileWriter);

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException(e.getMessage());

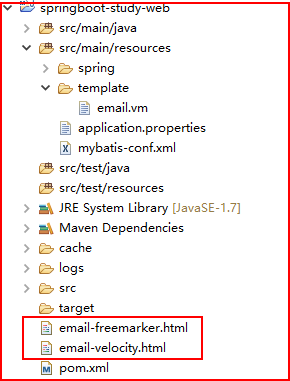
}

**return** "success";

}

}

使用swagger-ui进行测试：



# 4 Spring部分技术在项目中的应用

## 4.1 Springboot使用前置拦截器处理登陆请求

新建src/main/java/com/cxn/config/WebConfig

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.web.servlet.config.annotation.InterceptorRegistry;

**import** org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

**import** com.cxn.interceptor.LoginInterceptor;

@Configuration

**public** **class** WebConfig **extends** WebMvcConfigurerAdapter{

/\*\*

\* 自定义静态资源

\* **@param** registry

\*/

@Override

**public** **void** addResourceHandlers(ResourceHandlerRegistry registry) {

//将所有/static/\*\* 访问都映射到classpath:/static/ 目录下

registry.addResourceHandler("/static/\*\*").addResourceLocations("classpath:/static/");

}

/\*\*

\* 拦截器(用户登录验证)

\* **@param** registry

\*/

@Override

**public** **void** addInterceptors(InterceptorRegistry registry) {

// addPathPatterns 用于添加拦截规则

// excludePathPatterns 用户排除拦截

LoginInterceptor loginInterceptor=**new** LoginInterceptor();

registry.addInterceptor(loginInterceptor).addPathPatterns("/\*\*");

**super**.addInterceptors(registry);

}

}

新建src/main/java/interceptor/LoginInterceptor

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.web.servlet.HandlerInterceptor;

**import** org.springframework.web.servlet.ModelAndView;

**public** **class** LoginInterceptor **implements** HandlerInterceptor{

@Override

**public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {

System.***out***.println(">>>>>>>>>进入登陆拦截器<<<<<<<<<<");

System.***out***.println(request.getRequestURI());

// **TODO** Auto-generated method stub

**if**( request.getRequestURI().contains("/swagger-resources/")||request.getRequestURI().contains("/login")){

// 如果是登陆请求 或 满足条件请求的话，直接放行，

System.***out***.println(">>>>>>>>>请求满足条件<<<<<<<<<<");

**return** **true**;

}

System.***out***.println(">>>>>>>>>请求不满足条件<<<<<<<<<<");

String key = request.getHeader("coffee-oa-token");

/\*if(key ==null||"null".equals(key)){

response.setCharacterEncoding("utf-8");

response.setStatus(org.apache.http.HttpStatus.SC\_FORBIDDEN);

ErrorMessage error=new ErrorMessage();

error.setErrorMsg("请求头部必须包含token信息");

ObjectMapper om=new ObjectMapper();

String str=om.writer().writeValueAsString(error);

response.getWriter().write(str);

return false;

}

\*/

**if** ("demo".equals(key)) {

System.***out***.println("do something~");

}**else**{

System.***out***.println("do other things~");

}

**return** **true**;

}

@Override

**public** **void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)

**throws** Exception {

// **TODO** Auto-generated method stub

}

}

新建src/main/java/com/cxn/controller/SystemController

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.stereotype.Controller;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** com.cxn.model.User;

**import** io.swagger.annotations.ApiOperation;

@Controller

@RequestMapping("/system")

**public** **class** SystemController {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(SystemController.**class**);

@ApiOperation(value = "用户登陆")

@RequestMapping(path="/login",method={RequestMethod.***GET***})

**public** ResponseEntity<User> login(@RequestParam("username") String username, @RequestParam("password") String password){

***logger***.debug(">>>>>>>>>>coming in login method.<<<<<<<<<");

// 1.经过springMVC的前置拦截器后进入方法

// 2.取出用户名去数据库查询用户

// 3.校验密码

// 4.密码正确，利用username+System.currentTimeMillis(),在取md5作为token

// 5.将token作为key，当前用户的信息CurrentUser作为value存入redis

User user = **new** User();

user.setUsername(username);

user.setPassword(password);

user.setAge(25);

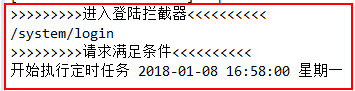
user.setHeight(170);

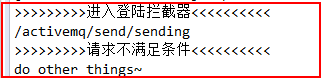
**return** ResponseEntity.*ok*(user);

}

}

使用swaggerUI测试:





## 4.2 Springboot使用自定义注解 + AOP的使用

导入pom依赖

<!-- 开启AOP -->

<dependency>

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

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

<exclusions>

<exclusion>

<artifactId>logback-classic</artifactId>

<groupId>ch.qos.logback</groupId>

</exclusion>

<exclusion>

<artifactId>log4j-over-slf4j</artifactId>

<groupId>org.slf4j</groupId>

</exclusion>

</exclusions>

</dependency>

追加application.properties

#aop

spring.aop.auto=true

spring.aop.proxy-target-class=true

新建src/main/java/com/cxn/aspect/LogAspect

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** org.aspectj.lang.JoinPoint;

**import** org.aspectj.lang.annotation.After;

**import** org.aspectj.lang.annotation.AfterReturning;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.aspectj.lang.annotation.Before;

**import** org.aspectj.lang.annotation.Pointcut;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** LogAspect {

/\*\*

execution(\* com.midai.service.impl..\*.\*(..))

解释如下：

符号 含义

execution（）

表达式的主体；

第一个”\*“符号

表示返回值的类型任意；

com.sample.service.impl AOP所切的服务的包名，即，我们的业务部分

包名后面的”..“ 表示当前包及子包

第二个”\*“ 表示类名，\*即所有类。此处可以自定义，下文有举例

.\*(..) 表示任何方法名，括号表示参数，两个点表示任何参数类型

execution(<修饰符模式>?<返回类型模式><方法名模式>(<参数模式>)<异常模式>?) 除了返回类型模式、方法名模式和参数模式外，其它项都是可选的。

\*/

@Before("execution(\* com.cxn.controller..\*.\*(..))")

**public** **void** logServiceAccessBefore(JoinPoint joinPoint){

System.***out***.println("Before1: " + joinPoint);

**if** (joinPoint.getArgs().length > 0) {

System.***out***.println("日志记录:用户" +joinPoint.getArgs()[0] + "在" + **new** SimpleDateFormat("yyyy-MM-dd hh:mm;ss").format(**new** Date()) + "调用了"+ joinPoint.getSignature()+"方法" );

}**else**{

System.***out***.println("日志记录:在" + **new** SimpleDateFormat("yyyy-MM-dd hh:mm;ss").format(**new** Date()) + "调用了"+ joinPoint.getSignature()+"方法" );

}

}

@After("execution(\* com.cxn.controller..\*.\*(..))")

**public** **void** logServiceAccessAfter(JoinPoint joinPoint){

System.***out***.println("After: " + joinPoint);

}

@AfterReturning("execution(\* com.cxn.controller..\*.\*(..))")

**public** **void** logServiceAccessAfterReturning(JoinPoint joinPoint) {

System.***out***.println("Completed: " + joinPoint);

}

/\*\*

\* around 开启后会覆盖before

\* **@param** joinPoint

\*/

// @Around("execution(\* com.midai.controller..\*.\*(..))")

**public** **void** logServiceAccessAround(JoinPoint joinPoint){

System.***out***.println("Around: " + joinPoint);

}

//声明一个切入点

@Pointcut("execution(\* com.cxn.controller..\*.\*(..))")

**public** **void** logServiceAccessPointCut(){

}

@Before("logServiceAccessPointCut()")

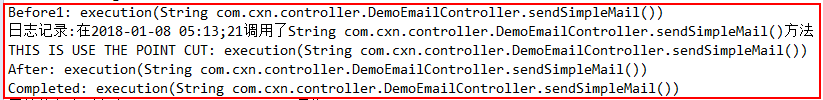
**public** **void** losgUseThePointCut(JoinPoint joinPoint){

System.***out***.println("THIS IS USE THE POINT CUT: " + joinPoint);

}

}

使用swagger-UI进行访问测试：



AOP 切方法，成功。

具体的语法可以查看切点表达式，博文介绍：

http://blog.csdn.net/qwe6112071/article/details/50951720

新建src/main/java/com/cxn/annotation/DemoActivity

**import** java.lang.annotation.Documented;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

@Retention(RetentionPolicy.***RUNTIME***)

@Target({ElementType.***METHOD***})

@Documented

**public** **@interface** DemoActivity {

**public** String task() **default** "#{taskId}";

}

新建src/main/java/com/cxn/annotation/DemoActivityParam

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

@Retention(RetentionPolicy.***RUNTIME***)

@Target({ElementType.***PARAMETER***})

**public** **@interface** DemoActivityParam {

}

LogAspect先将原有规则注释掉(控制台看着乱),然后添加方法：

@SuppressWarnings({ "rawtypes", "unused" })

@Before("@annotation(activity)")

// @Before("@annotation(com.cxn.annotation.DemoActivity)")

**public** **void** logServiceAccessBefore(JoinPoint joinPoint, DemoActivity activity){

System.***out***.println("Before: " + joinPoint);

System.***out***.println("activity:" + activity);

Object[] args = joinPoint.getArgs();

**if** (args != **null**) {

Method m = **null**;

Object target = joinPoint.getTarget();

String methodName = joinPoint.getSignature().getName();

Class[] parameterTypes = ((MethodSignature) joinPoint.getSignature()).getMethod().getParameterTypes();

**try** {

m = target.getClass().getMethod(methodName, parameterTypes);

} **catch** (Exception e) {

e.printStackTrace();

}

Annotation[][] an = m.getParameterAnnotations();

**for** (**int** i = 0; i < an.length; i++) {

Annotation[] n = an[i];

**for** (**int** k = 0; k < n.length; k++) {

**if** (n[k] **instanceof** DemoActivityParam) {

DemoActivityParam p = (DemoActivityParam) n[k];

// 如果p中有定义的值，可以取出来

**if** (args[i] **instanceof** String) {

// 得到自定义注解对应参数的值

String test = (String) args[i];

System.***out***.println("自定义注解@DemoActivityParam对应参数的值为:" + test);

**break**;

}

}

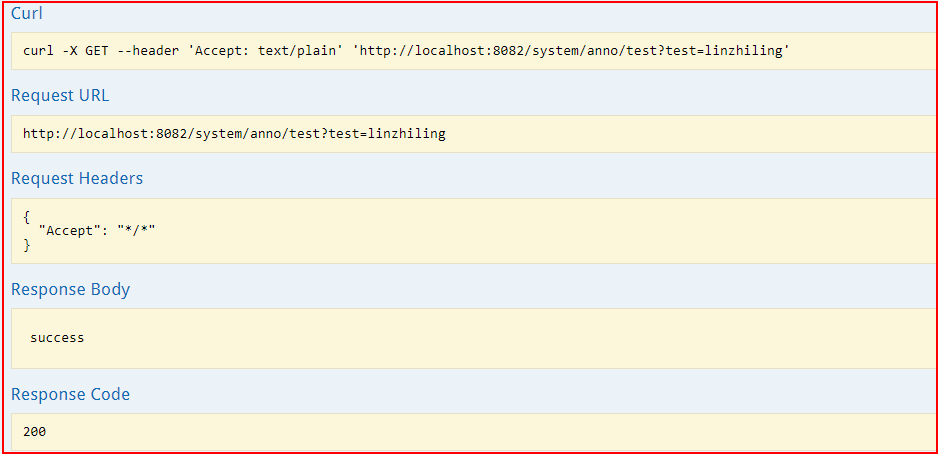
}

}

}

}

使用swagger-UI访问测试：



1515405203(1)

# 5.自定义springboot-starter

## 5.1创建一个maven项目

Ⅰ：pom文件：

注意：**spring-boot-autoconfigure**必须引入，根据版本自己选择

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.cxn</groupId>

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

<version>1.0.0</version>

<dependencies>

**<dependency>**

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

**<artifactId>****spring-boot-autoconfigure</artifactId>**

**<version>1.5.1.RELEASE</version>**

**</dependency>**

<!-- 根据功能需要，添加其它的依赖 -->

<!-- <dependency> -->

<!-- <groupId>org.apache.commons</groupId> -->

<!-- <artifactId>commons-lang3</artifactId> -->

<!-- <version>3.5</version> -->

<!-- </dependency> -->

</dependencies>

</project>

Ⅱ：创建DemoService接口

**package** com.cxn.demo.service;

**public** **interface** DemoService {

**public** String jointString(String value);

}

Ⅲ：创建实现类DemoServiceImpl

**package** com.cxn.demo.service.impl;

**import** com.cxn.demo.service.DemoService;

**public** **class** DemoServiceImpl **implements** DemoService{

**private** String param1;

**private** String param2;

@Override

**public** String jointString(String value) {

**return** value + param1 + param2;

}

**public** String getParam1() {

**return** param1;

}

**public** **void** setParam1(String param1) {

**this**.param1 = param1;

}

**public** String getParam2() {

**return** param2;

}

**public** **void** setParam2(String param2) {

**this**.param2 = param2;

}

}

Ⅳ：创建DemoProperties用来从配置文件中读取配置参数

**package** com.cxn.demo.config;

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

@ConfigurationProperties(prefix=DemoProperties.***DEMO\_PREFIX***)

**public** **class** DemoProperties {

// 意思是读取配置文件中前缀为DEMO\_PREFIX的配置

**public** **static** **final** String ***DEMO\_PREFIX*** = "com.cxn.demo.starter";

// 自定义属性

**private** String param1;

**private** String param2;

**public** String getParam1() {

**return** param1;

}

**public** **void** setParam1(String param1) {

**this**.param1 = param1;

}

**public** String getParam2() {

**return** param2;

}

**public** **void** setParam2(String param2) {

**this**.param2 = param2;

}

}

Ⅴ：创建DemoAutoConfiguration自动配置类（关键）

**package** com.cxn.demo.config;

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

**import** org.springframework.boot.autoconfigure.condition.ConditionalOnClass;

**import** org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;

**import** org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;

**import** org.springframework.boot.context.properties.EnableConfigurationProperties;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** com.cxn.demo.service.DemoService;

**import** com.cxn.demo.service.impl.DemoServiceImpl;

@Configuration

@ConditionalOnClass(DemoService.**class**)

@EnableConfigurationProperties(DemoProperties.**class**)

**public** **class** DemoAutoConfiguration {

@Autowired

**private** DemoProperties properties;

@Bean

// 自动构建bean的条件

// 条件一：在spring中没有DemoService.class文件的时候

@ConditionalOnMissingBean(DemoService.**class**)

// 条件二：配置文件中配置了指定的参数-->DemoProperties.DEMO\_PREFIX.enabled=true才会触发自动装配

@ConditionalOnProperty(prefix = DemoProperties.***DEMO\_PREFIX***, value = "enabled", havingValue = "true")

**public** DemoService demoService(){

// 可以new一个对象然后手动赋值，也可以通过构造方法的方式注入值

DemoServiceImpl demoService = **new** DemoServiceImpl();

demoService.setParam1(properties.getParam1());

demoService.setParam2(properties.getParam2());

**return** demoService;

}

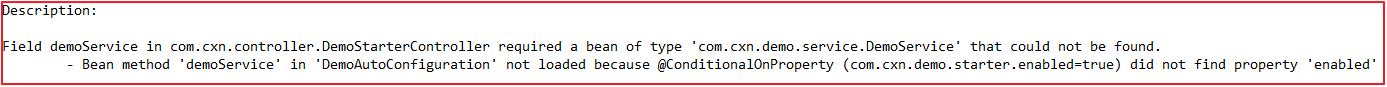
}

@ConditionalXxxx是作为条件存在的，只有满足了我们指定的条件spring才会自动创建该Bean.

这里指定的条件是当DemoService不存在的时候，并且配置文件中满足了配置要求：

com.cxn.demo.starter.enabled=true的条件下才会自动配置该bean。

如果项目中注入了该Bean,但是配置文件没有配置com.cxn.demo.starter.enabled=true的话，启动项目会报错：



当然，如果没有使用该Bean，即DemoService,也没有xxx=true的话，也不会报错，因为springboot-starter会按需决定是否装载该bean.

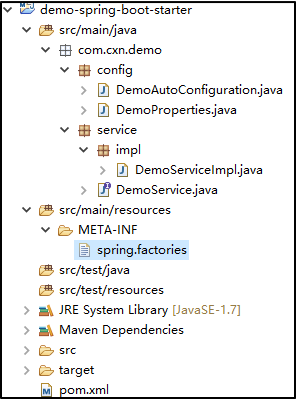
Ⅵ：在resource下新建文件夹META-INF然后新建spring.factories文件

添加如下配置信息：

org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.cxn.demo.config.DemoAutoConfiguration

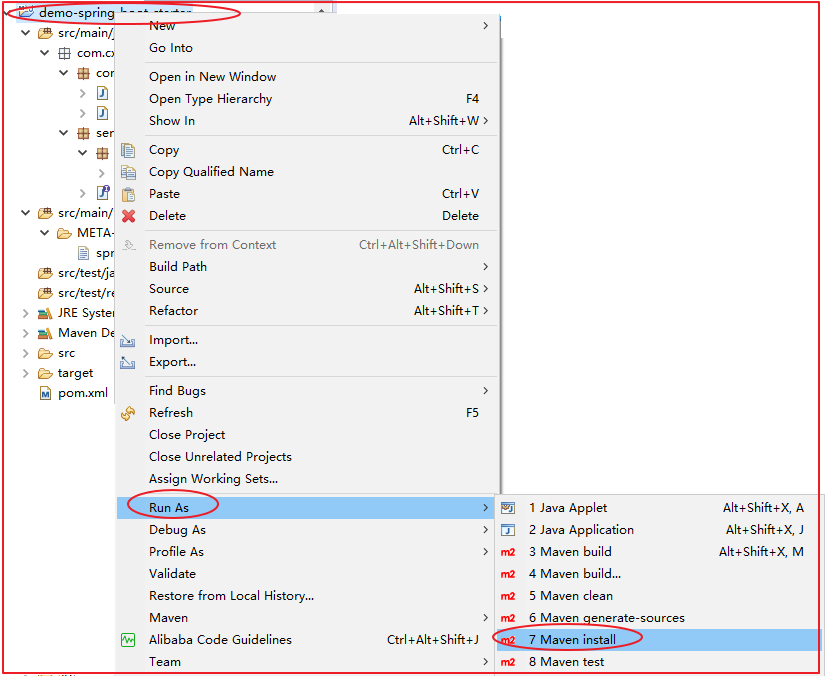
因为springboot启动后回去扫面所有jar包META-INF文件夹下的spring.factories文件，从而得到自动配置信息。

最终项目目录结构如图：

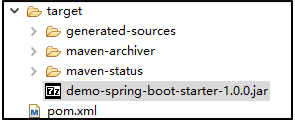


## 5.2 将自定义starter安装到maven 仓库

自定义starter创建完成之后，可以通过右击项目，然后Run As -->Maven install命令，将maven项目打成jar包



打包后jar包会存放在target文件夹下:



而后可以通过eclipse将jar包打入本地仓库，或者登陆公司私服仓库，指定groupId和ArtifactId以及版本号后将jar包导入，就可以在其它maven项目中使用该starter了。

## 5.3 测试

导入pom.xml文件

<!-- 自定义starter -->

<dependency>

<groupId>com.cxn</groupId>

<artifactId>demo-starter</artifactId>

<version>1.0.0</version>

</dependency>

新建一个DemoStarterController

**package** com.cxn.controller;

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

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

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

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

**import** com.cxn.demo.service.DemoService;

@RestController

@RequestMapping(value="/demoStarter")

**public** **class** DemoStarterController {

@Autowired

**private** DemoService demoService;

@RequestMapping(value="/test", method=RequestMethod.***GET***)

@ResponseBody

**public** String testDemoStarter(@RequestParam("value") String value){

String jointString = demoService.jointString(value);

System.***out***.println(jointString);

**return** jointString;

}

}

添加配置文件

com.cxn.demo.starter.param1=+demoTest

com.cxn.demo.starter.param2=+testDemo

com.cxn.demo.starter.enabled=true

使用swagger-UI进行测试：测试成功

http://localhost:8082/swagger-ui.html#!/demo-starter-controller/testDemoStarterUsingGET

