**Springboot笔记2**

# 过滤器和Servlet

## 过滤器

@ServletComponentScan --------------启动类要加上这个注解



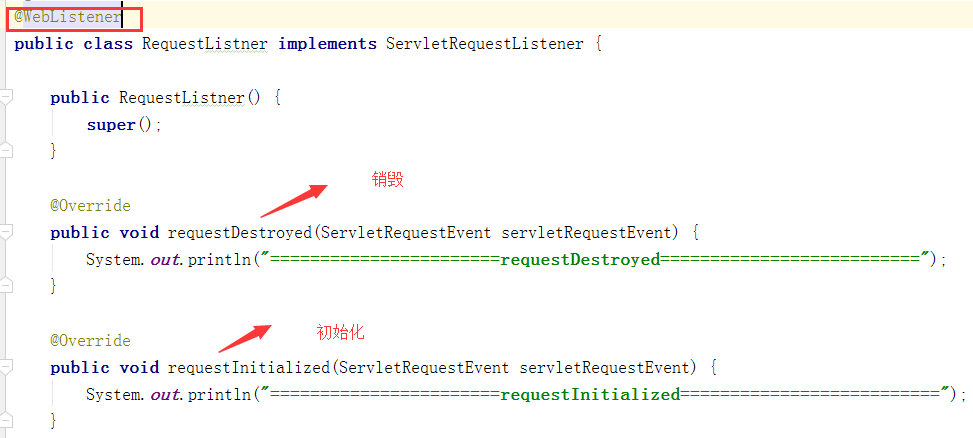
## Servlet



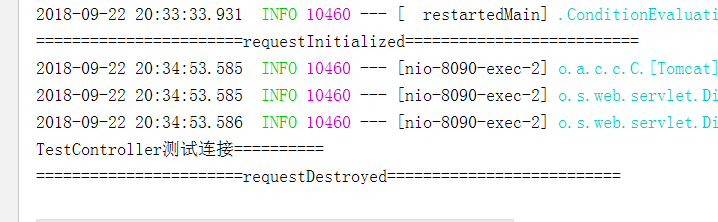
# Listener

常用的监听器 servletContextListener、httpSessionListener、servletRequestListener

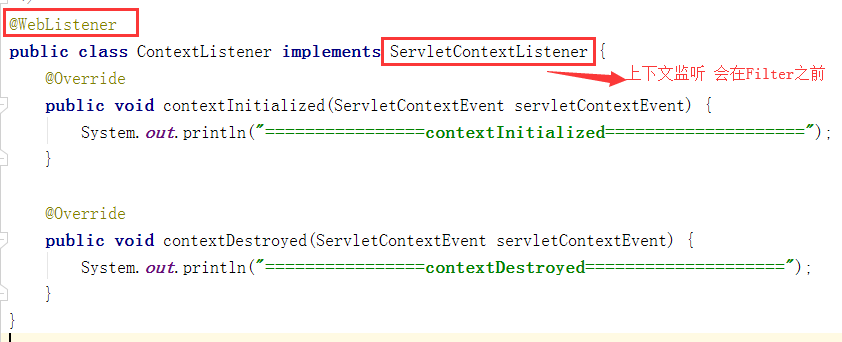
## servletRequestListener 请求

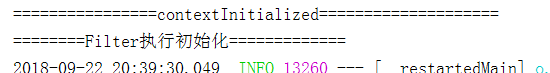


**说明：这是请求接口的时候进行监听。**



## ServletContextListener 上下文

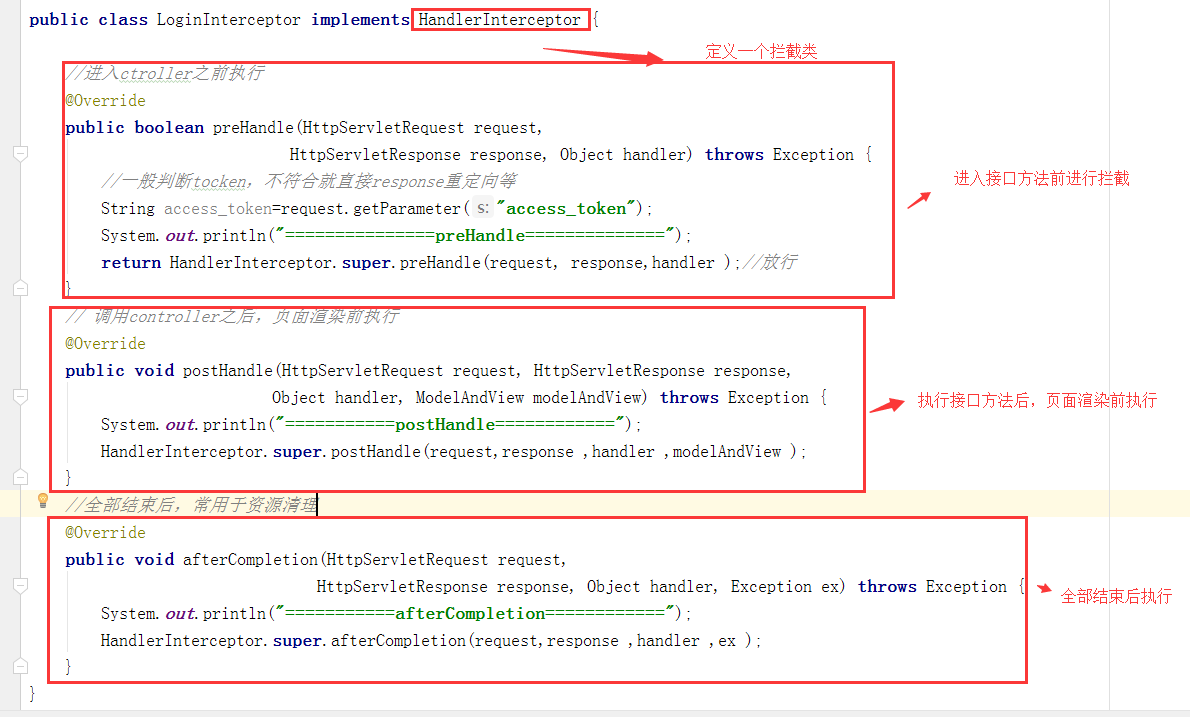




# 拦截器

## 拦截器配置

步骤1：mplements HandlerInterceptor



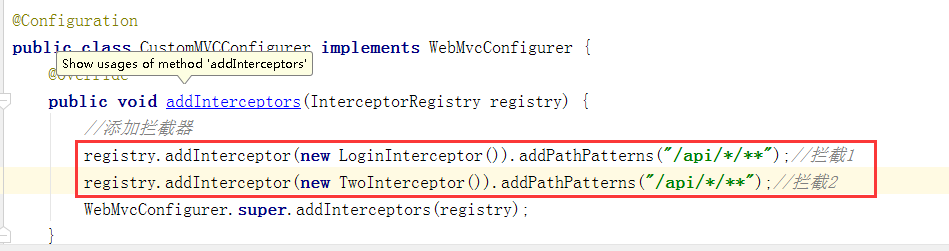
步骤2：implements WebMvcConfigurer



执行顺序：



## 多重拦截执行顺序



执行效果：



# 监听 过滤 拦截执行顺序

servletContextListener

Filter

servletRequestListener

Interceptor

上下文监听->过滤前->请求监听->拦截前->action执行

->拦截后->请求监听->过滤后->上下文监听

# Freemarker 模板引擎

## 配置及使用

步骤1：

<!-- 引入freemarker模板引擎的依赖 -->

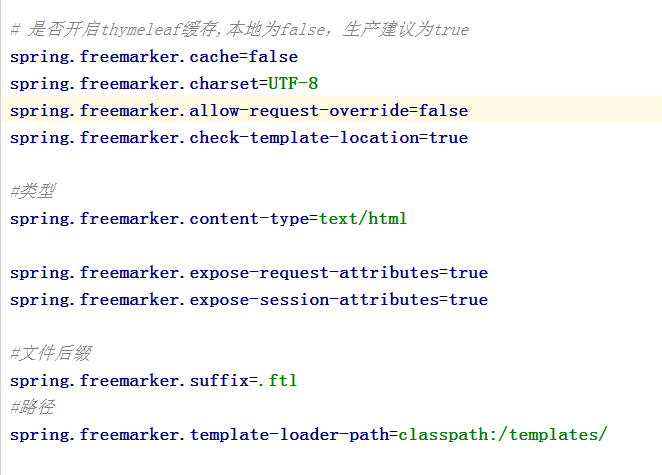
<dependency>

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

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

</dependency>

步骤2：



# 是否开启thymeleaf缓存,本地为false，生产建议为true

spring.freemarker.cache=false

spring.freemarker.charset=UTF-8

spring.freemarker.allow-request-override=false

spring.freemarker.check-template-location=true

#类型

spring.freemarker.content-type=text/html

spring.freemarker.expose-request-attributes=true

spring.freemarker.expose-session-attributes=true

#文件后缀

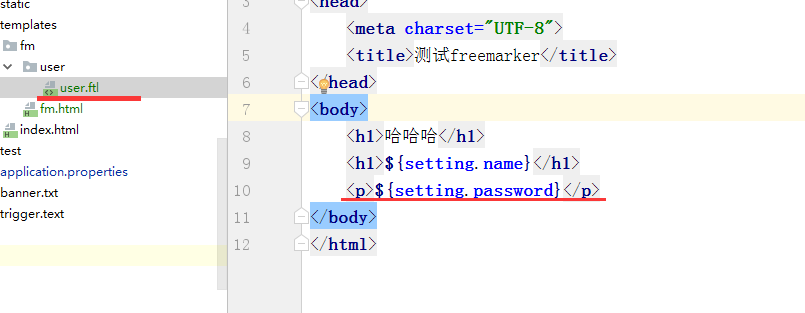
spring.freemarker.suffix=.ftl

#路径

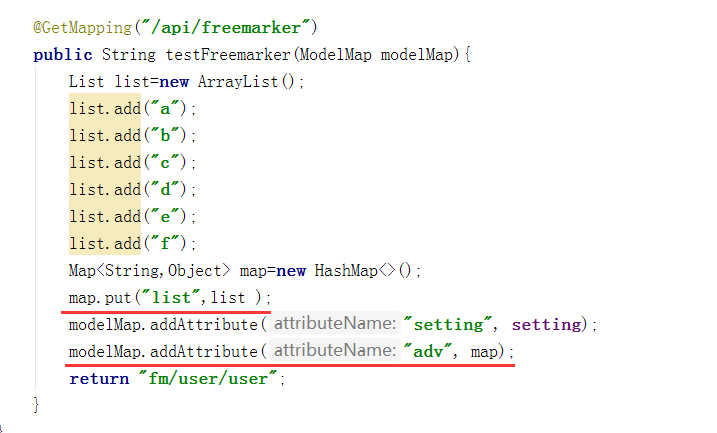
spring.freemarker.template-loader-path=classpath:/templates/

步骤3：





## Freemarker遍历





# Thymeleaf 模板引擎

步骤1：

<dependency>

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

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

<version>2.0.4.RELEASE</version>

</dependency>

步骤2:

#开发时关闭缓存,不然没法看到实时页面

spring.thymeleaf.cache=false

spring.thymeleaf.mode=HTML5

#前缀

spring.thymeleaf.prefix=classpath:/templates/tl/

#编码

spring.thymeleaf.encoding=UTF-8

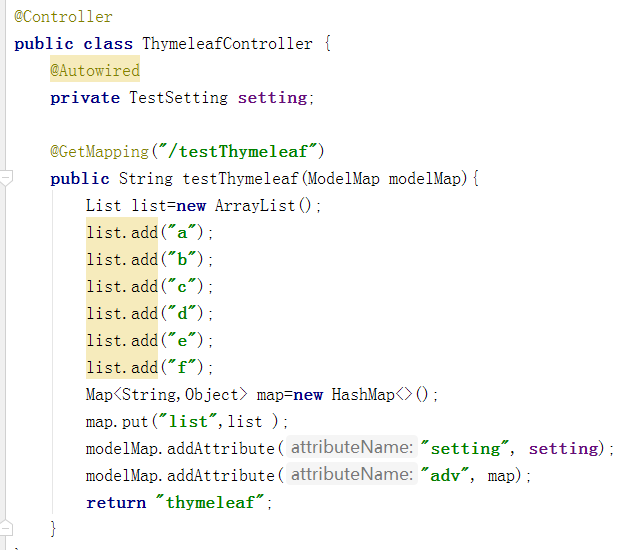
#类型

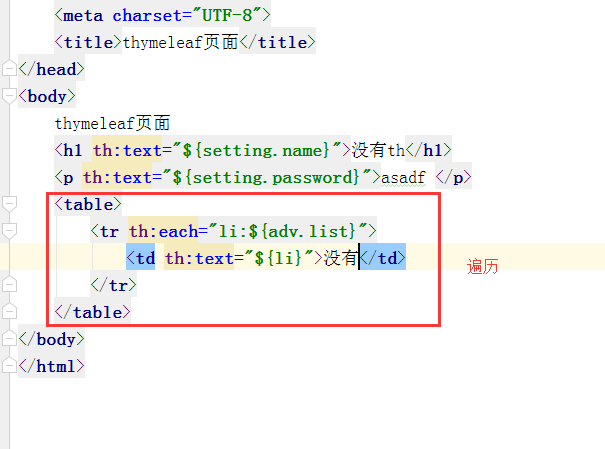
spring.thymeleaf.servlet.content-type=text/html

#名称的后缀

spring.thymeleaf.suffix=.html

步骤3：





# 整合mybatis

## 整合、插入、打印sql

**步骤1：**

依赖：

<dependency>

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

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

<version>1.3.2</version>

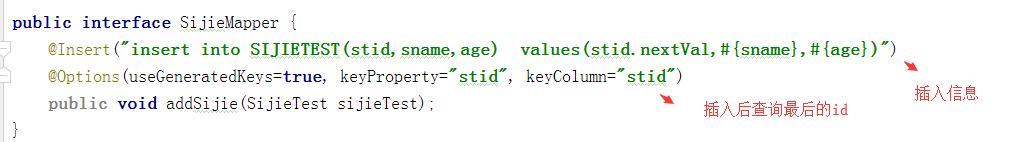
</dependency>

数据库：

**步骤2：**

**@MapperScan("com.example.demo.Dao")**

**步骤3：**

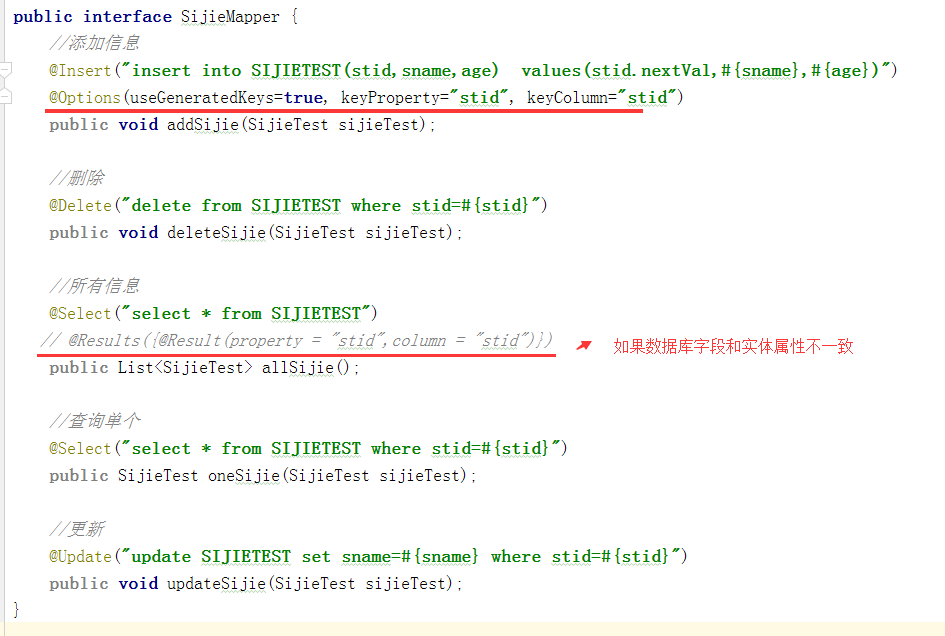


然后其他的业务层，控制层使用方法不变！

**#增加打印sql语句，一般用于本地开发测试**

mybatis.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl

## 增删改查



# 事务

讲解场景的隔离级别

Serializable： 最严格，串行处理，消耗资源大

Repeatable Read：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据

Read Committed：大多数主流数据库的默认事务等级

Read Uncommitted：保证了读取过程中不会读取到非法数据。

讲解常见的传播行为

PROPAGATION\_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务,最常见的选择。

PROPAGATION\_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW--新建事务，如果当前存在事务，把当前事务挂起, 两个事务之间没有关系，一个异常，一个提交，不会同时回滚

PROPAGATION\_NOT\_SUPPORTED--以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

PROPAGATION\_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常

@EnableTransactionManagement

@Transactional(propagation = Propagation.REQUIRED,isolation = Isolation.DEFAULT,timeout=36000,rollbackFor=Exception.class)

# redis的部署和应用

看案例。

# 加密在testcontroller

<**dependency**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-web</**artifactId**>  
 <**version**>5.1.0.RELEASE</**version**>  
</**dependency**>

@GetMapping(**"/passwordEncoder"**)  
 **public** String passwordEncoder(String string){  
 String test1=**passwordEncoder**.encode(string);  
 String test2=**"$2a$10$4so92gEv/fM70p4ynle3Ie.Le5cjWhBW6E3oeA4IE9lij7JCA0mra"**;  
 String result=**""**;  
 **if** (**passwordEncoder**.matches(string, test2)){  
 result=**"true"**;  
 }**else** {  
 result=**"false"**;  
 }  
  
 **return passwordEncoder**.encode(string);  
}

# 日志

logback-spring.xml

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

<configuration>

<appender name="consoleApp" class="ch.qos.logback.core.ConsoleAppender">

<layout class="ch.qos.logback.classic.PatternLayout">

<pattern>

%date{yyyy-MM-dd HH:mm:ss.SSS} %-5level[%thread]%logger{56}.%method:%L -%msg%n

</pattern>

</layout>

</appender>

<appender name="fileInfoApp" class="ch.qos.logback.core.rolling.RollingFileAppender">

<filter class="ch.qos.logback.classic.filter.LevelFilter">

<level>ERROR</level>

<onMatch>DENY</onMatch>

<onMismatch>ACCEPT</onMismatch>

</filter>

<encoder>

<pattern>

%date{yyyy-MM-dd HH:mm:ss.SSS} %-5level[%thread]%logger{56}.%method:%L -%msg%n

</pattern>

</encoder>

<!-- 滚动策略 -->

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- 路径 -->

<fileNamePattern>app\_log/log/app.info.%d.log</fileNamePattern>

</rollingPolicy>

</appender>

<appender name="fileErrorApp" class="ch.qos.logback.core.rolling.RollingFileAppender">

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>ERROR</level>

</filter>

<encoder>

<pattern>

%date{yyyy-MM-dd HH:mm:ss.SSS} %-5level[%thread]%logger{56}.%method:%L -%msg%n

</pattern>

</encoder>

<!-- 设置滚动策略 -->

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- 路径 -->

<fileNamePattern>app\_log/log/app.err.%d.log</fileNamePattern>

<!-- 控制保留的归档文件的最大数量，超出数量就删除旧文件，假设设置每个月滚动，

且<maxHistory> 是1，则只保存最近1个月的文件，删除之前的旧文件 -->

<MaxHistory>1</MaxHistory>

</rollingPolicy>

</appender>

<root level="INFO">

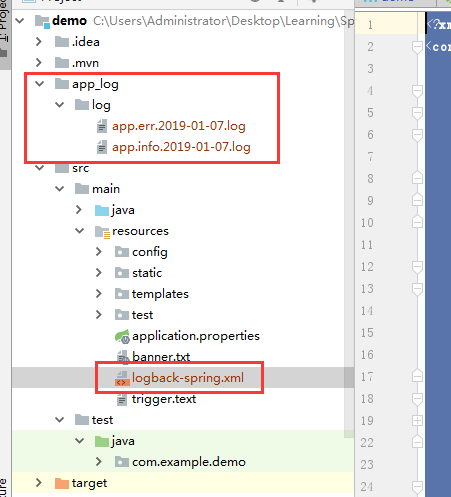
<appender-ref ref="consoleApp"/>

<appender-ref ref="fileInfoApp"/>

<appender-ref ref="fileErrorApp"/>

</root>

</configuration>



# 定时任务&异步任务

## 定时任务设置

简介：讲解什么是定时任务和常见定时任务区别

1、常见定时任务 Java自带的java.util.Timer类

timer:配置比较麻烦，时间延后问题

timertask:不推荐

2、Quartz框架

配置更简单

xml或者注解

3、SpringBoot使用注解方式开启定时任务

1）启动类里面 @EnableScheduling开启定时任务，自动扫描

2）定时任务业务类 加注解 @Component被容器扫描

3）定时执行的方法加上注解 @Scheduled(fixedRate=2000) 定期执行一次

## SpringBoot常用定时任务配置实战

简介：SpringBoot常用定时任务表达式配置和在线生成器

1、cron 定时任务表达式 @Scheduled(cron="\*/1 \* \* \* \* \*") 表示每秒

1）crontab 工具 https://tool.lu/crontab/

2、fixedRate: 定时多久执行一次（上一次开始执行时间点后xx秒再次执行；）

3、fixedDelay: 上一次执行结束时间点后xx秒再次执行

4、fixedDelayString: 字符串形式，可以通过配置文件指定

## 异步任务

简介：讲解什么是异步任务，和使用SpringBoot2.x开发异步任务实战

1、什么是异步任务和使用场景：适用于处理log、发送邮件、短信……等

下单接口->查库存 100

余额校验 150

风控用户100

....

2、启动类里面使用@EnableAsync注解开启功能，自动扫描

3、定义异步任务类并使用@Component标记组件被容器扫描,异步方法加上@Async

注意点：

1）要把异步任务封装到类里面，不能直接写到Controller

2）增加Future<String> 返回结果 AsyncResult<String>("task执行完成");

3）如果需要拿到结果 需要判断全部的 task.isDone()

4、通过注入方式，注入到controller里面，如果测试前后区别则改为同步则把Async注释掉

