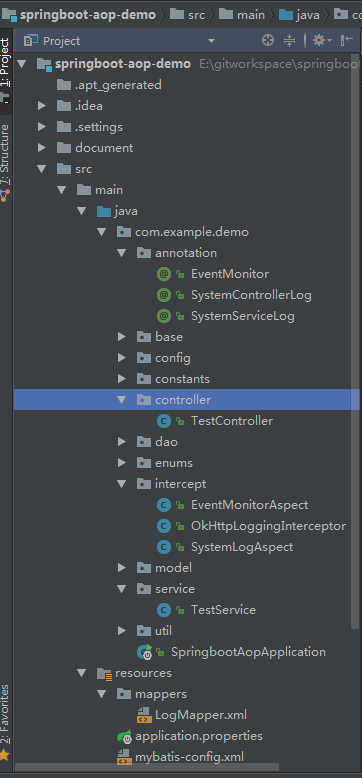
# springboot-aop-demo说明

## 项目结构：



## Pom.xml引入的jar包

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-configuration-processor</artifactId>  
 <optional>true</optional>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-autoconfigure</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjrt</artifactId>  
 <version>1.8.13</version>  
 </dependency>  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.8.13</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.google.guava</groupId>  
 <artifactId>guava</artifactId>  
 <version>RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>org.assertj</groupId>  
 <artifactId>assertj-core</artifactId>  
 <version>2.6.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.squareup.okhttp3</groupId>  
 <artifactId>okhttp</artifactId>  
 <version>3.10.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>commons-lang</groupId>  
 <artifactId>commons-lang</artifactId>  
 <version>2.6</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.45</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.3.1</version>  
 </dependency>  
 <!-- mysql 驱动 -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.38</version>  
 </dependency>  
 <!-- 数据库连接池 -->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.0.5</version>  
 </dependency>  
  
 <dependency>  
 <groupId>tk.mybatis</groupId>  
 <artifactId>mapper-spring-boot-starter</artifactId>  
 <version>1.1.4</version>  
 </dependency>  
  
 <dependency>  
 <groupId>io.github.openfeign</groupId>  
 <artifactId>feign-core</artifactId>  
 <version>9.6.0</version>  
 </dependency>  
  
</dependencies>  
  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

## 配置文件application.properties

spring.application.name=aop-demo  
server.port=8080  
  
server.tomcat.uri-encoding=UTF-8  
spring.http.encoding.charset=UTF-8  
spring.http.encoding.enabled=true  
spring.http.encoding.force=true  
spring.messages.encoding=UTF-8  
  
# ================================================================================================  
  
spring.devtools.restart.exclude=static/\*\*  
spring.devtools.restart.additional-exclude=static/\*\*,public/\*\*  
  
spring.session.store-type=none  
  
spring.profiles.active=dev  
  
# ================================================================================================  
  
# 数据库基本配置  
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test  
spring.datasource.username=root  
spring.datasource.password=111111  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
  
spring.jpa.database=MYSQL  
# 显示后台处理的SQL语句  
spring.jpa.show-sql=true  
# 自动检查实体和数据库表是否一致，如果不一致则会进行更新数据库表  
# spring.jpa.hibernate.ddl-auto=update  
  
#连接池的配置信息  
## 初始化大小，最小，最大  
spring.druid.initialSize=5  
spring.druid.minIdle=5  
spring.druid.maxActive=20  
## 配置获取连接等待超时的时间  
spring.druid.maxWait=60000  
# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒  
spring.druid.timeBetweenEvictionRunsMillis=60000  
# 配置一个连接在池中最小生存的时间，单位是毫秒  
spring.druid.minEvictableIdleTimeMillis=300000  
spring.druid.validationQuery=SELECT 1 FROM DUAL  
spring.druid.testWhileIdle=true  
spring.druid.testOnBorrow=false  
spring.druid.testOnReturn=false  
spring.druid.poolPreparedStatements=true  
spring.druid.maxPoolPreparedStatementPerConnectionSize=20  
# 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙  
spring.druid.filters=stat,wall,log4j  
# 通过connectProperties属性来打开mergeSql功能；慢SQL记录  
spring.druid.connectionProperties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000  
  
mybatis.config-location=classpath:mybatis-config.xml   
mybatis.mapper-locations=classpath\*:mappers/\*Mapper.xml  
mybatis.type-aliases-package=com.example.demo.model  
  
# ============================================ 日志 ====================================================  
logging.level.com.example.demo.dao=DEBUG  
logging.pattern.console=%d{yyyy/MM/dd-HH:mm:ss} [%thread] %-5level %logger- %msg%n  
logging.pattern.file=%d{yyyy/MM/dd-HH:mm} [%thread] %-5level %logger- %msg%n  
logging.file=E:/testspace/log.log

## 数据库配置mybatis-config.xml

<?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>  
  
 <properties>  
 <property name="dialect" value="mysql"/>  
 </properties>  
  
 <settings>  
 <!-- 全局映射器启用缓存 -->  
 <setting name="cacheEnabled" value="true" />  
 <!-- 查询时,关闭关联对象及时加载以提高性能 -->  
 <setting name="lazyLoadingEnabled" value="false" />  
 <!-- 设置关联对象加载的形态,此处为按需加载字段(加载字段由SQL指定),不会加载关联表的所有字段,以提高性能 -->  
 <setting name="aggressiveLazyLoading" value="false" />  
 <!-- 对于位置的SQL查询,允许返回不同的结果集以达到通用的效果 -->  
 <setting name="multipleResultSetsEnabled" value="true" />  
 <!-- 允许使用列标签代替列明 -->  
 <setting name="useColumnLabel" value="true" />  
 <!-- 允许使用自定义的主键值(比如由程序生成的UUID 32位编码作为键值), 数据表的pk生成策略将被覆盖 -->  
 <setting name="useGeneratedKeys" value="true" />  
 <!-- 给予被嵌套的resultMap以字段-属性的映射支持 -->  
 <setting name="autoMappingBehavior" value="PARTIAL" />  
 <!-- 对于批量更新操作缓存SQL以提高性能 -->  
 <setting name="defaultExecutorType" value="REUSE" />  
 <!-- 数据库超过25000秒仍未响应则超时 -->  
 <setting name="defaultStatementTimeout" value="25000" />  
 <setting name="safeRowBoundsEnabled" value="false" />  
 <setting name="mapUnderscoreToCamelCase" value="false" />  
 <setting name="localCacheScope" value="SESSION" />  
 <setting name="jdbcTypeForNull" value="OTHER" />  
 <setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode ,toString" />  
 <!-- 打印查询语句 -->  
 <!--<setting name="logImpl" value="STDOUT\_LOGGING" />-->  
 </settings>  
  
 <typeHandlers>  
 <typeHandler handler="org.apache.ibatis.type.InstantTypeHandler" />  
 <typeHandler handler="org.apache.ibatis.type.LocalDateTimeTypeHandler" />  
 <typeHandler handler="org.apache.ibatis.type.LocalDateTypeHandler" />  
 <typeHandler handler="org.apache.ibatis.type.LocalTimeTypeHandler" />  
 <typeHandler handler="org.apache.ibatis.type.OffsetDateTimeTypeHandler" />  
 <typeHandler handler="org.apache.ibatis.type.OffsetTimeTypeHandler" />  
 <typeHandler handler="org.apache.ibatis.type.ZonedDateTimeTypeHandler" />  
 </typeHandlers>  
  
  
</configuration>

## Config：

### 数据源：

@Configuration  
@ConfigurationProperties(prefix = "spring.datasource")  
public class DruidConfig {  
  
 private String url;  
 private String username;  
 private String password;  
  
 @Bean  
 public ServletRegistrationBean servletRegistration() {  
 ServletRegistrationBean servletRegistration = new ServletRegistrationBean(new StatViewServlet()); //添加初始化参数：initParams  
 servletRegistration.addUrlMappings("/druid/\*");  
 //白名单  
 servletRegistration.addInitParameter("allow", url);  
 //IP黑名单 (存在共同时，deny优先于allow) : 如果满足deny的话提示:Sorry, you are not permitted to view this page.  
 //servletRegistration.addInitParameter("deny", "192.168.1.73");  
 //登录查看信息的账号密码.  
 servletRegistration.addInitParameter("loginUsername", username);  
 servletRegistration.addInitParameter("loginPassword", password);  
 //是否能够重置数据.  
 servletRegistration.addInitParameter("resetEnable", "false");  
 return servletRegistration;  
 }  
  
 @Bean  
 public FilterRegistrationBean filterRegistrationBean() {  
 FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean(new WebStatFilter());  
 //添加过滤规则.  
 filterRegistrationBean.addUrlPatterns("/\*");  
 //添加不需要忽略的格式信息.  
 filterRegistrationBean.addInitParameter("exclusions", "\*.js,\*.gif,\*.jpg,\*.png,\*.css,\*.ico,/druid/\*");  
 return filterRegistrationBean;  
 }  
  
 public String getUrl() {  
 return url;  
 }  
  
 public void setUrl(String url) {  
 this.url = url;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
}

## 启动入口

@SpringBootApplication  
@ComponentScan({"com.example.demo"})  
@MapperScan(basePackages = "com.example.demo.dao")//mapper接口的路径  
public class SpringbootAopApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(SpringbootAopApplication.class, args);  
 }  
}

## AOP

### 注解

方法执行时间的注解

@Retention(RetentionPolicy.*RUNTIME*)  
@Target({ElementType.*METHOD*})  
public @interface EventMonitor {  
 String name() default ""; //方法名  
 String desc() default ""; //方法描述  
  
}

ControllerLog注解

@Target({ElementType.*PARAMETER*, ElementType.*METHOD*})  
@Retention(RetentionPolicy.*RUNTIME*)  
@Documented  
public @interface SystemControllerLog {  
 String description() default "";//描述    
 String moduleType() default "";//模块代码    
 String operateValue() default "";//操作类型    
 boolean firstParamName() default false;  
}

ServiceLog注解

@Target({ElementType.*METHOD*,ElementType.*PARAMETER*})  
@Retention(RetentionPolicy.*RUNTIME*)  
@Documented  
public @interface SystemServiceLog {  
 String description() default "";//描述    
 String moduleType() default "";//模块代码    
 String operateValue() default "";//操作类型    
 boolean firstParamName() default false;  
}

### 注解切入

方法执行时间的注解 切入

@Aspect  
@Component  
public class EventMonitorAspect {  
  
 private Logger logger = LoggerFactory.*getLogger*("monitor");  
  
 @Pointcut("@annotation(com.example.demo.annotation.EventMonitor)")  
 public void monitor() {  
 }  
  
 @Around("monitor()")  
 public Object around(ProceedingJoinPoint joinPoint) throws Throwable {  
 MethodSignature signature = (MethodSignature) joinPoint.getSignature();  
 Method method = signature.getMethod();  
 EventMonitor monitor = method.getAnnotation(EventMonitor.class);  
  
 String monitorName = monitor.name();  
 String monitorDesc = monitor.desc();  
  
 long start = System.*currentTimeMillis*();  
 int status = 200;  
 try {  
 Object result = joinPoint.proceed();  
 return result;  
 } catch (Throwable e) {  
 status = 500;  
 throw e;  
 } finally {  
 logger.info("{\"eventName\":\"{}\",\"eventDesc\":\"{}\",\"time\":{},\"status\":{}}", monitorName, monitorDesc, System.*currentTimeMillis*() - start, status);  
 }  
 }  
}

日志注解切入

@Aspect  
@Component  
public class SystemLogAspect {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(SystemLogAspect.class);  
  
 // 队列  
 private static BlockingQueue<Log> *queue* = new LinkedBlockingQueue<Log>();  
  
 // 缓存线程池  
 private static ExecutorService *threadPool* = Executors.*newFixedThreadPool*(3);  
  
 // 任务数  
 private static int *taskSize* = 6;  
  
 // 线程是否已启动  
 boolean isStartThread = false;  
  
 // 用来启动或停止线程  
 static boolean *run* = true;  
  
 @Autowired  
 private LogMapper logMapper;  
  
 // Service层切点  
 @Pointcut("@annotation(com.example.demo.annotation.SystemServiceLog)")  
 public void serviceAspect() {  
 }  
  
 // Controller层切点  
 @Pointcut("@annotation(com.example.demo.annotation.SystemControllerLog)")  
 public void controllerAspect() {  
 }  
  
 public static BlockingQueue<Log> getQueue() {  
 return *queue*;  
 }  
  
 public static void setQueue(BlockingQueue<Log> queue) {  
 SystemLogAspect.*queue* = queue;  
 }  
  
 public static boolean isRun() {  
 return *run*;  
 }  
  
 public static void setRun(boolean run) {  
 SystemLogAspect.*run* = run;  
 }  
  
 */\*\*  
 \* 返回通知 用于拦截Controller层记录用户的操作  
 \*  
 \** ***@param*** *joinPoint 切点  
 \** ***@param*** *result 返回值  
 \** ***@see*** *[类、类#方法、类#成员]  
 \*/* @AfterReturning(value = "controllerAspect()", returning = "result")  
 public void afterReturn(JoinPoint joinPoint, Object result) {  
 // 请求的IP  
 User user = WebUtils.*getSessionValue*(CommonConstant.*SESSION\_USER*);  
 WebResult webResult = new WebResult();  
 webResult.setCode(CommonConstant.*BACK\_SUCCESS*);  
 try {  
 if (WebResult.class.isInstance(result)) {  
 webResult = (WebResult) result;  
 }  
 InnnerBean innnerBean = *getControllerMethodDescription*(joinPoint);  
 Object[] arguments = innnerBean.getArguments();  
 String remark = innnerBean.getDescription();  
  
 Log log = new Log.Builder().type(LogTypes.type.*operate*)  
 .moduleType(innnerBean.getModuleType())  
 .operateCode(joinPoint.getSignature().getName())  
 .operateValue(innnerBean.getOperateValue())  
 .remark(remark)  
 .operateStatus(webResult.getCode().equals(CommonConstant.*BACK\_SUCCESS*) ? LogTypes.operateStatus.*Y* : LogTypes.operateStatus.*N*)// 返回值1操作成功，否则失败  
 .method((joinPoint.getTarget().getClass().getName() + "." + joinPoint.getSignature().getName() + "()"))  
 .param(arguments.toString())  
 .loginName(user.getName())  
 .fullName(user.getName())  
 .build();  
 // 放入队列  
 *queue*.put(log);  
 if (!isStartThread) {  
 for (int i = 0; i < *taskSize*; i++) {  
 *threadPool*.execute(new saveLogThread());  
 }  
 isStartThread = true;  
 }  
 } catch (Exception e) {  
 *logger*.error("异常信息:{}", e.toString());  
 }  
 }  
  
 */\*\*  
 \* 异常通知 用于拦截service层记录异常日志  
 \*  
 \** ***@param*** *joinPoint  
 \** ***@param*** *e  
 \** ***@see*** *[类、类#方法、类#成员]  
 \*/* @AfterThrowing(pointcut = "serviceAspect()", throwing = "e")  
 public void doAfterThrowing(JoinPoint joinPoint, Throwable e) {  
 // 读取session中的用户  
 User user = WebUtils.*getSessionValue*(CommonConstant.*SESSION\_USER*);  
 String params = "";  
  
 try {  
 if (joinPoint.getArgs() != null && joinPoint.getArgs().length > 0) {  
 for (int i = 0; i < joinPoint.getArgs().length; i++) {  
 params += JSON.*toJSONString*(joinPoint.getArgs()[i].toString()) + ";";  
 }  
 }  
  
 InnnerBean innnerBean = *getServiceMthodDescription*(joinPoint);  
 String loginName = "";  
  
 Log log =  
 new Log.Builder().type(LogTypes.type.*exception*)  
 .moduleType(innnerBean.getModuleType())  
 .operateCode(joinPoint.getSignature().getName())  
 .operateValue(innnerBean.getOperateValue())  
 .remark(innnerBean.getDescription())  
 .operateStatus(LogTypes.operateStatus.*N*)  
 .method(  
 (joinPoint.getTarget().getClass().getName() + "." + joinPoint.getSignature().getName() + "()"))  
 .param(params)  
 .exceptionDetail(e.toString())  
 .build();  
 // 放入队列  
 *queue*.put(log);  
 if (!isStartThread) {  
 new Thread(new saveLogThread()).start();  
 isStartThread = true;  
 }  
 } catch (Exception ex) {  
 *logger*.error("异常信息:{}", ex.toString());  
 } finally {  
 *logger*.error("异常方法:{" + joinPoint.getTarget().getClass().getName() + "}异常代码:{"  
 + joinPoint.getSignature().getName() + "}异常信息:{" + e.toString() + "}参数:{" + params + "}");  
 }  
  
 }  
  
 */\*\*  
 \* 获取注解中对方法的描述信息 用于service层注解  
 \*  
 \** ***@param*** *joinPoint 切点  
 \** ***@return*** *方法描述  
 \** ***@throws*** *Exception  
 \** ***@see*** *[类、类#方法、类#成员]  
 \*/* @SuppressWarnings("rawtypes")  
 public static InnnerBean getServiceMthodDescription(JoinPoint joinPoint)  
 throws Exception {  
 String targetName = joinPoint.getTarget().getClass().getName();  
 String methodName = joinPoint.getSignature().getName();  
 Object[] arguments = joinPoint.getArgs();  
 Class targetClass = Class.*forName*(targetName);  
 Method[] methods = targetClass.getMethods();  
 String moduleType = "";  
 String operateValue = "";  
 String description = "";  
 InnnerBean innnerBean = new InnnerBean(moduleType, operateValue, description);  
 for (Method method : methods) {  
 if (method.getName().equals(methodName)) {  
 Class[] clazzs = method.getParameterTypes();  
 if (clazzs.length == arguments.length) {  
 SystemServiceLog annotation = method.getAnnotation(SystemServiceLog.class);  
 moduleType = annotation.moduleType();  
 operateValue = annotation.operateValue();  
 description = annotation.description();  
 innnerBean = new InnnerBean(moduleType, operateValue, description);  
 break;  
 }  
 }  
 }  
 innnerBean.setArguments(arguments);  
 return innnerBean;  
 }  
  
 */\*\*  
 \* 获取注解中对方法的描述信息 用于Controller层注解  
 \*  
 \** ***@param*** *joinPoint 切点  
 \** ***@return*** *方法描述  
 \** ***@throws*** *Exception  
 \** ***@see*** *[类、类#方法、类#成员]  
 \*/* @SuppressWarnings("rawtypes")  
 public static InnnerBean getControllerMethodDescription(JoinPoint joinPoint)  
 throws Exception {  
 String targetName = joinPoint.getTarget().getClass().getName();  
 String methodName = joinPoint.getSignature().getName();  
 Object[] arguments = joinPoint.getArgs();  
 Class targetClass = Class.*forName*(targetName);  
 Method[] methods = targetClass.getMethods();  
 String moduleType = "";  
 String operateValue = "";  
 String description = "";  
 boolean firstParamName = false;  
 InnnerBean innnerBean = new InnnerBean(moduleType, operateValue, description);  
 for (Method method : methods) {  
 if (method.getName().equals(methodName)) {  
 Class[] clazzs = method.getParameterTypes();  
 if (clazzs.length == arguments.length) {  
 SystemControllerLog annotation = method.getAnnotation(SystemControllerLog.class);  
 moduleType = annotation.moduleType();  
 operateValue = annotation.operateValue();  
 description = annotation.description();  
 firstParamName = annotation.firstParamName();  
 innnerBean = new InnnerBean(moduleType, operateValue, description);  
 innnerBean.setFirstParamName(firstParamName);  
 break;  
 }  
 }  
 }  
 innnerBean.setArguments(arguments);  
 return innnerBean;  
 }  
  
 */\*\*  
 \* 内部类封装注入信息  
 \*  
 \** ***@see*** *[相关类/方法]  
 \** ***@since*** *[产品/模块版本]  
 \*/* static class InnnerBean {  
 private String moduleType;// 模块代码  
  
 private String description;// 描述  
  
 private String operateValue;// 操作类型  
  
 private boolean firstParamName;  
  
 private Object[] arguments;  
  
 public InnnerBean(String moduleType, String operateValue, String description) {  
 super();  
 this.moduleType = moduleType;  
 this.description = description;  
 this.operateValue = operateValue;  
 }  
  
 public String getOperateValue() {  
 return operateValue;  
 }  
  
 public void setOperateValue(String operateValue) {  
 this.operateValue = operateValue;  
 }  
  
 public String getModuleType() {  
 return moduleType;  
 }  
  
 public void setModuleType(String moduleType) {  
 this.moduleType = moduleType;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 public Object[] getArguments() {  
 return arguments;  
 }  
  
 public void setArguments(Object[] arguments) {  
 this.arguments = arguments;  
 }  
  
 public boolean isFirstParamName() {  
 return firstParamName;  
 }  
  
 public void setFirstParamName(boolean firstParamName) {  
 this.firstParamName = firstParamName;  
 }  
 }  
  
 */\*\*  
 \* 异步保存日志  
 \*  
 \** ***@see*** *[相关类/方法]  
 \** ***@since*** *[产品/模块版本]  
 \*/* class saveLogThread implements Runnable {  
 Lock lock = new ReentrantLock();  
  
 @Override  
 public void run() {  
 try {  
 while (*run*) {  
 while (*queue*.size() != 0) {  
 // 如果对插入顺序无要求，此处不需要同步可提升效率  
 lock.lock();  
 Log log = *queue*.take();  
 logMapper.insert(log);  
 lock.unlock();  
 }  
 Thread.*sleep*(3000);  
 }  
 } catch (InterruptedException e) {  
 *logger*.error("saveLogThread被唤醒：" + e.toString());  
 } catch (Exception e) {  
 *logger*.error("saveLogThread异常：" + e.toString());  
 }  
 }  
 }  
  
}

### 注解使用

Controller上使用

@RestController  
public class TestController {  
  
 @Autowired  
 private TestService testService;  
  
 private static String *json* = "{\"password\":\"111111\",\"createTime\":\"2017-04-19 17:21:31.0\",\"name\":\"aaa\",\"id\":\"1\"}";  
  
 @GetMapping("/testEventMonitor")

@EventMonitor(name="testEventMonitor",desc = "测试EventMonitor")  
 @SystemControllerLog(moduleType= LogTypes.operateValue.*select*,operateValue= LogTypes.operateValue.*login*,description="controllerLog")  
 @GetMapping("/controllerLog")  
 public void controllerLog() {  
 Thread.*sleep*(100);

WebUtils.*setSessionValue*(CommonConstant.*SESSION\_USER*,JSONObject.*parseObject*(*json*,User.class));  
 }  
  
 @GetMapping("/serviceLog")  
 public void serviceLog() throws Exception{  
 WebUtils.*setSessionValue*(CommonConstant.*SESSION\_USER*,JSONObject.*parseObject*(*json*,User.class));  
 testService.serviceMethod();  
 }  
}

Service上使用

@Service  
public class TestService {  
  
 @SystemServiceLog(moduleType= LogTypes.operateValue.*select*,operateValue=LogTypes.operateValue.*login*,description=" serviceLog ")  
 public void serviceMethod() throws Exception{  
 System.*out*.println("================ test SystemServiceLog ==================");  
 // 模拟异常  
 throw new Exception("test SystemServiceLog ");  
 }  
}

浏览器访问

Controller访问日志 、方法执行时间（控制台打印）

http://127.0.0.1:8080/serviceLog

Service异常日志

http://127.0.0.1:8080/serviceLog

记录的日志

