链滴

Springboot 自定义自动记录接口日志到数据库注解

作者：Pirates

原文链接：https://ld246.com/article/1602830618868

来源网站：链滴

1.新建Log表

```
CREATE TABLE `log` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `user_name` varchar(50) NOT NULL COMMENT '用户名',
  `ip` varchar(64) DEFAULT NULL COMMENT 'IP',
  `description` varchar(255) DEFAULT NULL COMMENT '操作描述',
  `param` varchar(255) DEFAULT NULL COMMENT '参数值',
  `browser` varchar(255) DEFAULT NULL COMMENT '浏览器',
  `time` bigint(20) DEFAULT NULL COMMENT '执行时间',
  `type` varchar(255) DEFAULT NULL COMMENT '日志类型',
  `method` varchar(255) DEFAULT NULL COMMENT '执行方法',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `exception_detail` text COMMENT '异常详细信息',
  `update_time` datetime DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=529 DEFAULT CHARSET=utf8mb4;
```

2.添加依赖

```
    <!--aop-->
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-aop</artifactId>
       </dependency>
       <!--  UserAgentUtils，浏览器信息工具类  -->
       <dependency>
          <groupId>eu.bitwalker</groupId>
          <artifactId>UserAgentUtils</artifactId>
          <version>1.21</version>
       </dependency>
       <!--ip2region，这是根据ip查地址的工具，有兴趣自己可以了解-->
       <!-- <dependency>-->
       <!-- <groupId>org.lionsoul</groupId>-->
       <!-- <artifactId>ip2region</artifactId>-->
       <!-- <version>1.7.2</version>-->
       <!-- </dependency>-->
```

3.需要用到的工具类

```
import com.ljfchtcc.learn.error.LearnException;
import com.ljfchtcc.learn.result.ResultCode;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetails;

/**
 * <h3>learn</h3>
 * <p>SecurityUtils</p>
 *
```

```
 * @author : LiYu
 * @date : 2020-09-04 09:49
 **/
public class SecurityUtils {
    /**
     * 获取系统用户名称
     *
     * @return 系统用户名称
     */
    public static String getCurrentUsername() {
        final Authentication authentication = SecurityContextHolder.getContext().getAuthenticat
on();
        if (authentication.getPrincipal() == null) {
            throw new LearnException(ResultCode.UNAUTHORIZED, "当前登录状态过期");
        }
        try{
            UserDetails userDetails = (UserDetails) authentication.getPrincipal();
            return userDetails.getUsername();
        }catch (Exception e){
            throw new LearnException(ResultCode.UNAUTHORIZED, "当前未登录");
        }
    }
    /**
     * 取得当前用户登录IP, 如果当前用户未登录则返回空字符串.
     * 此方法无用
     */
    public static String getCurrentUserIp() {
        final Authentication authentication = SecurityContextHolder.getContext().getAuthenticat
on();

        if (authentication == null) {
            throw new LearnException(ResultCode.UNAUTHORIZED, "当前登录状态过期");
        }
        Object details = authentication.getDetails();
        if (!(details instanceof WebAuthenticationDetails)) {
            return "";
        }
        WebAuthenticationDetails webDetails = (WebAuthenticationDetails) details;
        return webDetails.getRemoteAddress();
    }
}

import eu.bitwalker.useragentutils.Browser;
import eu.bitwalker.useragentutils.UserAgent;

import javax.servlet.http.HttpServletRequest;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * <h3>learn</h3>
 * <p>LogUtils</p>
```

```
 *
 * @author : LiYu
 * @date : 2020-09-04 09:50
 **/
public class LogUtils {
    private static final char SEPARATOR = '_';

    private static final String UNKNOWN = "unknown";
    /**
     * 获取ip地址
     */
    public static String getIp(HttpServletRequest request) {
        String ip = request.getHeader("x-forwarded-for");
        if (ip == null || ip.length() == 0 || UNKNOWN.equalsIgnoreCase(ip)) {
            ip = request.getHeader("Proxy-Client-IP");
        }
        if (ip == null || ip.length() == 0 || UNKNOWN.equalsIgnoreCase(ip)) {
            ip = request.getHeader("WL-Proxy-Client-IP");
        }
        if (ip == null || ip.length() == 0 || UNKNOWN.equalsIgnoreCase(ip)) {
            ip = request.getRemoteAddr();
        }
        String comma = ",";
        String localhost = "127.0.0.1";
        if (ip.contains(comma)) {
            ip = ip.split(",")[0];
        }
        if  (localhost.equals(ip))  {
            // 获取本机真正的ip地址
            try {
                ip = InetAddress.getLocalHost().getHostAddress();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            }
        }
        return ip;
    }

    /**
     * 获取浏览器信息
     * @param request
     * @return
     */
    public static String getBrowser(HttpServletRequest request){
        UserAgent userAgent = UserAgent.parseUserAgentString(request.getHeader("User-Agen
")));
        Browser browser = userAgent.getBrowser();
        return browser.getName();
    }

    /**
     * 获取堆栈信息
     */
    public static String getStackTrace(Throwable throwable){
```

```
        StringWriter sw = new StringWriter();
        try (PrintWriter pw = new PrintWriter(sw)) {
            throwable.printStackTrace(pw);
            return sw.toString();
        }
    }
}

import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import javax.servlet.http.HttpServletRequest;
import java.util.Objects;

/**
 * <h3>learn</h3>
 * <p>RequestHolder</p>
 *
 * @author : LiYu
 * @date : 2020-09-04 09:53
 **/
public class RequestHolder {
    /**
     * 获取HttpServletRequest对象
     * @return
     */
    public static HttpServletRequest getHttpServletRequest() {
        return ((ServletRequestAttributes) Objects.requireNonNull(RequestContextHolder.getRe
uestAttributes())).getRequest();
    }
}
```

4.相对性的实体类

```
/**
 * <p>
 *
 * </p>
 *
 * @author LiYu
 * @since 2020-09-04
 */
@Accessors(chain = true)
@Data
@EqualsAndHashCode(callSuper = false)
@ApiModel(value="Log对象", description="")
public class Log extends BaseEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "用户名")
    private String userName;
```

```
    @ApiModelProperty(value = "IP")
    private String ip;

    @ApiModelProperty(value = "操作描述")
    private String description;

    @ApiModelProperty(value = "参数值")
    private String param;

    @ApiModelProperty(value = "浏览器")
    private String browser;

    @ApiModelProperty(value = "执行时间")
    private Long time;

    @ApiModelProperty(value = "日志类型")
    private String type;

    @ApiModelProperty(value = "执行方法")
    private String method;

    @ApiModelProperty(value = "异常详细信息")
    private String exceptionDetail;

    public Log( String type,Long time) {
        this.type = type;
        this.time = time;
    }
}
```

5.自定义操作日志的注解类

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * <h3>learn</h3>
 * <p>自定义操作日志的注解类</p>
 *
 * @author : LiYu
 * @date : 2020-09-04 09:53
 **/
@Target(ElementType.METHOD)//注解放置的目标位置,METHOD是可注解在方法级别上
@Retention(RetentionPolicy.RUNTIME)//注解在哪个阶段执行
public @interface LearnLog {
    String value() default "";
}
```

6.新建切面类

```
import com.ljfchtcc.learn.db.entity.Log;
import com.ljfchtcc.learn.db.service.ILogService;
import com.ljfchtcc.learn.util.LogUtils;
import com.ljfchtcc.learn.util.SecurityUtils;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;

/**
 * <h3>learn</h3>
 * <p>LogAspect</p>
 *
 * @author : LiYu
 * @date : 2020-09-04 09:56
 **/
@Component
@Aspect
@Slf4j
public class LogAspect {
    @Autowired
    private ILogService logService;

    ThreadLocal<Long> currentTime = new ThreadLocal<>();
    /**
     * 设置操作日志切入点 记录操作日志 在注解的位置切入代码
     */
    @Pointcut("@annotation(com.ljfchtcc.learn.log.LearnLog)")
    public void logPoinCut() {
    }

    /**
     * 配置环绕通知,使用在方法logPointcut()上注册的切入点
     *
     * @param joinPoint join point for advice
     */
    @Around("logPoinCut()")
    public Object saveSysLog(ProceedingJoinPoint joinPoint)throws Throwable{
        Object result;
        currentTime.set(System.currentTimeMillis());//记录方法的执行时间
        result = joinPoint.proceed();
        Log log = new Log("INFO",System.currentTimeMillis() - currentTime.get());
        currentTime.remove();
        HttpServletRequest request = RequestHolder.getHttpServletRequest();
        logService.save(SecurityUtils.getCurrentUsername(), LogUtils.getBrowser(request), LogUti
s.getIp(request),joinPoint, log);
        return result;
```

```
    }

    /**
     * 配置异常通知
     *
     * @param joinPoint join point for advice
     * @param e exception
     */
    @AfterThrowing(pointcut = "logPoinCut()", throwing = "e")
    public void logAfterThrowing(JoinPoint joinPoint, Throwable e) {
        Log log = new Log("ERROR",System.currentTimeMillis() - currentTime.get());
        currentTime.remove();
        log.setExceptionDetail(LogUtils.getStackTrace(e));
        HttpServletRequest request = RequestHolder.getHttpServletRequest();
        logService.save(SecurityUtils.getCurrentUsername(), LogUtils.getBrowser(request), LogUti
s.getIp(request), (ProceedingJoinPoint)joinPoint, log);
    }

}
```

7.相应方法以及接口

```
/**
 * <p>
 *  服务类
 * </p>
 *
 * @author LiYu
 * @since 2020-09-04
 */
public interface ILogService extends IService<Log> {

    /**
     * 保存日志
     * @param userName
     * @param browser
     * @param ip
     * @param joinPoint
     * @param log
     */
    void save(String userName, String browser, String ip, ProceedingJoinPoint joinPoint, Log lo
);

    /**
     * 删除所有错误日志
     */
    void delAllByError();

    /**
     * 删除所有INFO日志
     */
    void delAllByInfo();
}
```

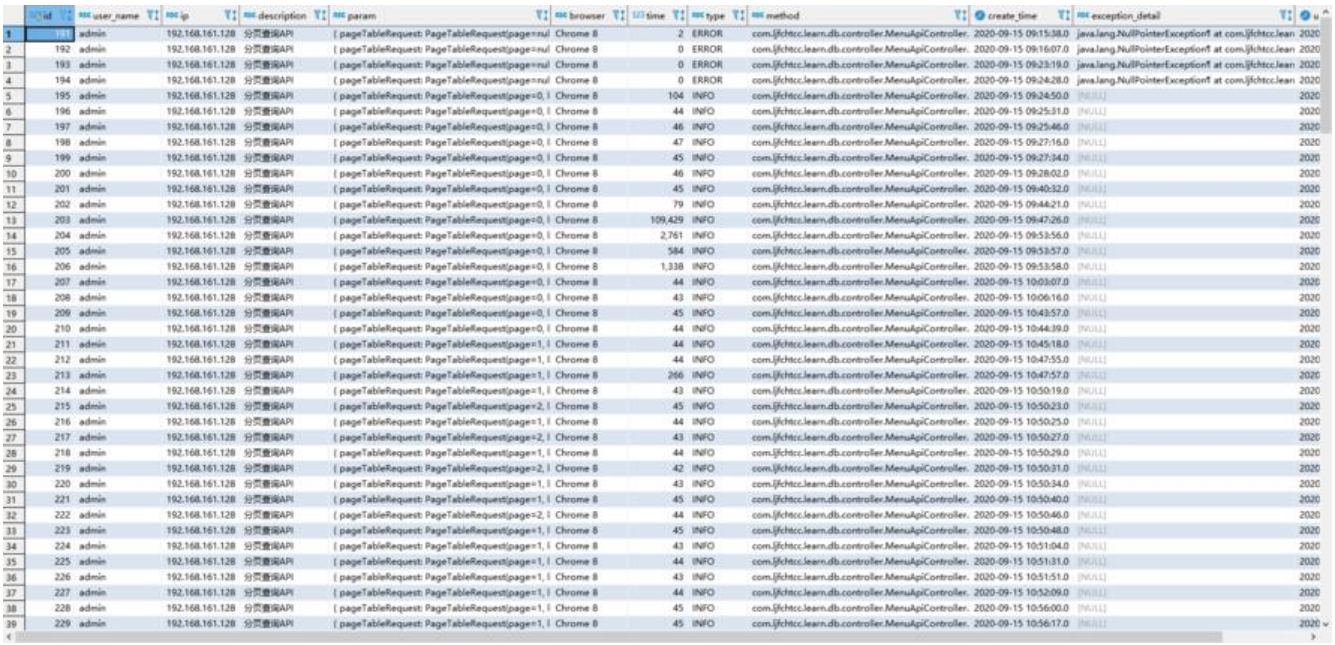
```
import cn.hutool.json.JSONObject;
import com.ljfchtcc.learn.db.entity.Log;
import com.ljfchtcc.learn.db.mapper.LogMapper;
import com.ljfchtcc.learn.db.service.ILogService;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.ljfchtcc.learn.log.LearnLog;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.lang.reflect.Method;

/**
 * <p>
 *  服务实现类
 * </p>
 *
 * @author LiYu
 * @since 2020-09-04
 */
@Service
public class LogServiceImpl extends ServiceImpl<LogMapper, Log> implements ILogService {
    @Autowired
    private LogMapper logMapper;

    @Override
    public void save(String userName, String browser, String ip, ProceedingJoinPoint joinPoint,
og log) {
        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        Method method = signature.getMethod();
        LearnLog learnLog = method.getAnnotation(LearnLog.class);
        // 方法路径
        String methodName = joinPoint.getTarget().getClass().getName()+"."+signature.getNam
()+"()";
        StringBuilder params = new StringBuilder("{");
        //参数值
        Object[] argValues = joinPoint.getArgs();
        //参数名称
        String[] argNames = ((MethodSignature)joinPoint.getSignature()).getParameterNames();
        if(argValues != null){
            for (int i = 0; i < argValues.length; i++) {
                params.append(" ").append(argNames[i]).append(": ").append(argValues[i]);
            }
        }
        // 描述
        if (log != null) {
            log.setDescription(learnLog.value());
        }
        assert log != null;
        log.setIp(ip);
        String loginPath = "login";
        if(loginPath.equals(signature.getName())){
```

```
            try {
                assert argValues != null;
                userName = new JSONObject(argValues[0]).get("userName").toString();
            }catch (Exception e){
                e.printStackTrace();
            }
        }
        log.setMethod(methodName);
        log.setUserName(userName);
        log.setParam(params.toString() + " }");
        log.setBrowser(browser);
        logMapper.insert(log);
    }

    @Override
    @Transactional(rollbackFor = Exception.class)
    public void delAllByError() {
        logMapper.delAllByInfo("ERROR");
    }

    @Override
    @Transactional(rollbackFor = Exception.class)
    public void delAllByInfo() {
        logMapper.delAllByInfo("INFO");
    }
}
```

8.使用

在接口处添加@LearnLog("接口备注")即可

9.最终效果



id	user_name	ip	description	param	browser	time	type	method	create_time	exception_detail	v
191	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=nul	Chrome 8	2	ERROR	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:15:38.0	java.lang.NullPointerException\n\t at com.ljfchtcc.lean	2020
192	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=nul	Chrome 8	0	ERROR	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:16:07.0	java.lang.NullPointerException\n\t at com.ljfchtcc.lean	2020
193	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=nul	Chrome 8	0	ERROR	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:23:19.0	java.lang.NullPointerException\n\t at com.ljfchtcc.lean	2020
194	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=nul	Chrome 8	0	ERROR	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:24:28.0	java.lang.NullPointerException\n\t at com.ljfchtcc.lean	2020
195	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	104	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:24:50.0	[NULL]	2020
196	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:25:31.0	[NULL]	2020
197	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	46	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:25:46.0	[NULL]	2020
198	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	47	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:27:16.0	[NULL]	2020
199	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:27:34.0	[NULL]	2020
200	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	46	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:28:02.0	[NULL]	2020
201	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:40:32.0	[NULL]	2020
202	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	79	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:44:21.0	[NULL]	2020
203	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	109,429	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:47:26.0	[NULL]	2020
204	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	2,761	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:53:56.0	[NULL]	2020
205	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	584	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:53:57.0	[NULL]	2020
206	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	1,338	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 09:53:58.0	[NULL]	2020
207	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:03:07.0	[NULL]	2020
208	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	43	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:06:16.0	[NULL]	2020
209	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:43:57.0	[NULL]	2020
210	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=0, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:44:39.0	[NULL]	2020
211	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:45:18.0	[NULL]	2020
212	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:47:55.0	[NULL]	2020
213	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	266	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:47:57.0	[NULL]	2020
214	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	43	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:19.0	[NULL]	2020
215	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=2, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:23.0	[NULL]	2020
216	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:25.0	[NULL]	2020
217	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=2, l	Chrome 8	43	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:27.0	[NULL]	2020
218	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:29.0	[NULL]	2020
219	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=2, l	Chrome 8	42	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:31.0	[NULL]	2020
220	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	43	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:34.0	[NULL]	2020
221	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:40.0	[NULL]	2020
222	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=2, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:46.0	[NULL]	2020
223	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:50:48.0	[NULL]	2020
224	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	43	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:51:04.0	[NULL]	2020
225	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:51:31.0	[NULL]	2020
226	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	43	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:51:51.0	[NULL]	2020
227	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	44	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:52:09.0	[NULL]	2020
228	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:56:00.0	[NULL]	2020
229	admin	192.168.161.128	分页查询API	{ pageTableRequest: PageTableRequest(page=1, l	Chrome 8	45	INFO	com.ljfchtcc.learn.db.controller.MenuApiController.	2020-09-15 10:56:17.0	[NULL]	2020

原文链接：Springboot 自定义自动记录接口日志到数据库注解