链滴

Springboot 使用 Logback 自动同步指定级别日志到 MariaDB 数据库

作者：zxniuniu

原文链接：https://ld246.com/article/1585146625743

来源网站：链滴

为方便问题排查与分析，有时候需要把符合条件的指定级别日志自动持久化到数据库中。

添加pom依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

创建logback配置文件logback-spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 日志级别从低到高：TRACE < DEBUG < INFO < WARN < ERROR < FATAL -->
<!-- 如果设置为WARN，则低于WARN的信息都不会输出 -->
<!-- scan:当此属性设置为true时，配置文档如果发生改变，将会被重新加载，默认值为true -->
<!-- scanPeriod:设置监测配置文档是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫
。 当scan为true时，此属性生效。默认的时间间隔为1分钟。 -->
<!-- debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。
认值为false。 -->
```

```
<configuration scan="true" scanPeriod="60 seconds" debug="false">
    <!-- name的值是变量的名称，value的值时变量定义的值。通过定义的值会被插入到logger上下
中。定义后，可以使“${}”来使用变量。 -->
    <property name="log.path" value="logs/" />

    <!-- 日志输出格式 -->
    <property name="LOG_PATTERN" value="%d{MM-dd HH:mm:ss} %-5level %logger{100}[
method,%line] ● %msg%n" />

    <!-- 彩色日志依赖的渲染类 -->
    <conversionRule conversionWord="clr" converterClass="org.springframework.boot.logging
logback.ColorConverter" />
    <conversionRule conversionWord="wex" converterClass="org.springframework.boot.loggi
g.logback.WhitespaceThrowableProxyConverter" />
    <conversionRule conversionWord="wEx" converterClass="org.springframework.boot.loggi
g.logback.ExtendedWhitespaceThrowableProxyConverter" />
    <property name="COLOR_PATTERN" value="${COLOR_PATTERN:-%clr(%d{yyyy-MM-dd H
:mm:ss.SSS}){faint} %clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(-){faint} %clr([%15.15t]){faint} %clr
%-40.40logger{39}){cyan} %clr(●){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wE
}}" />

    <!-- 控制台输出 -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <!-- 此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此
别的日志信息 -->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>DEBUG</level>
        </filter>
        <encoder>
            <!--<pattern>${LOG_PATTERN}</pattern>-->
            <pattern>${COLOR_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!--DEBUG 级别的日志-->
    <appender name="DEBUG_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 正在记录的日志文档的路径及文档名 -->
        <file>${log.path}/sys-debug.log</file>

        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>
                ${log.path}/sys-debug-%d{yyyy-MM-dd}.%i.log
            </fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAn
TimeBasedFNATP">
                <maxFileSize>
                    100MB
                </maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文档保留天数 -->
            <maxHistory>180</maxHistory>
        </rollingPolicy>
```

```
        <!--日志文档输出格式 -->
        <encoder>
            <pattern>${LOG_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>

        <!-- 此日志文档只记录ERROR级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>DEBUG</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!--INFO 级别的日志-->
    <appender name="INFO_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 正在记录的日志文档的路径及文档名 -->
        <file>${log.path}/sys-info.log</file>

        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>
                ${log.path}/sys-info-%d{yyyy-MM-dd}.%i.log
            </fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAn
TimeBasedFNATP">
                <maxFileSize>
                    100MB
                </maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文档保留天数 -->
            <maxHistory>180</maxHistory>
        </rollingPolicy>

        <!--日志文档输出格式 -->
        <encoder>
            <pattern>${LOG_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>

        <!-- 此日志文档只记录ERROR级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>INFO</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!--WARN 级别的日志-->
    <appender name="WARN_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 正在记录的日志文档的路径及文档名 -->
        <file>${log.path}/sys-warn.log</file>
```

```
        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>
                ${log.path}/sys-warn-%d{yyyy-MM-dd}.%i.log
            </fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAn
TimeBasedFNATP">
                <maxFileSize>
                    100MB
                </maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文档保留天数 -->
            <maxHistory>180</maxHistory>
        </rollingPolicy>

        <!--日志文档输出格式 -->
        <encoder>
            <pattern>${LOG_PATTERN}</pattern>
            <charset>UTF-8</charset> <!-- 此处设置字符集 -->
        </encoder>

        <!-- 此日志文档只记录ERROR级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>WARN</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!--ERROR 级别的日志-->
    <appender name="ERROR_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 正在记录的日志文档的路径及文档名 -->
        <file>${log.path}/sys-error.log</file>

        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>
                ${log.path}/sys-error-%d{yyyy-MM-dd}.%i.log
            </fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAn
TimeBasedFNATP">
                <maxFileSize>
                    100MB
                </maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文档保留天数 -->
            <maxHistory>180</maxHistory>
        </rollingPolicy>

        <!--日志文档输出格式 -->
        <encoder>
            <pattern>${LOG_PATTERN}</pattern>
            <charset>UTF-8</charset> <!-- 此处设置字符集 -->
        </encoder>
```

```
    <!-- 此日志文档只记录ERROR级别的 -->
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
      <level>ERROR</level>
      <onMatch>ACCEPT</onMatch>
      <onMismatch>DENY</onMismatch>
    </filter>
</appender>

<!--日志异步到数据库（指定连接信息） -->
<!--<appender name="DB" class="ch.qos.logback.classic.db.DBAppender">
    <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">
      <driverClass>com.mysql.cj.jdbc.Driver</driverClass>
      <url>jdbc:mysql://127.0.0.1:3306/databaseName</url>
      <user>root</user>
      <password>db_passowrd</password>
    </connectionSource>
</appender> -->
<!--日志异步到数据库（采用DbConnectionSource利用已有的连接信息）-->
<appender name="DB" class="ch.qos.logback.classic.db.DBAppender">
    <connectionSource class="cn.xianwu.framework.datasource.DbConnectionSource" />
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
      <level>WARN</level>
    </filter>
</appender>
<appender name="ASYNC_DB" class="ch.qos.logback.classic.AsyncAppender">
    <appender-ref ref="DB" />
    <includeCallerData>true</includeCallerData>
</appender>

<root level="DEBUG">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="DEBUG_FILE" />
    <appender-ref ref="INFO_FILE" />
    <appender-ref ref="WARN_FILE" />
    <appender-ref ref="ERROR_FILE" />
    <!-- 异步到数据库-->
    <appender-ref ref="ASYNC_DB" />
</root>

<!-- 系统模块日志级别控制 -->
<logger name="org" level="WARN" additivity="false" />

<logger name="ch.qos" level="WARN" additivity="false" />
<logger name="net" level="WARN" additivity="false" />

<logger name="io" level="WARN" additivity="false" />
<logger name="springfox" level="WARN" additivity="false" />

<logger name="com.alibaba" level="WARN" additivity="false" />

<logger name="cn.xianwu" level="DEBUG" />
<logger name="org.mybatis" level="DEBUG" />
<logger name="cn.xianwu.framework.shiro.web.session" level="ERROR" />
```

```
</configuration>
```

数据库连接信息加载

其中，最重要的是数据库的配置部分，你可以使用DriverManagerConnectionSource来指定数据库连接信息等，但由于使用了Springboot中的application-druid.yml中已经配置了数据库的连接信息所以重新定义了一个DbConnectionSource以利用数据库连接信息。

```
@Component
public class DbConnectionSource extends DriverManagerConnectionSource {
    public DbConnectionSource() throws Exception {
        Map<?, ?> map = YamlUtils.loadYaml("application-druid.yml");

        String publicKey = (String)YamlUtils.getProperty(map, "publickey");
        Map<?, ?> datasource = (Map)YamlUtils.getProperty(map, "spring.datasource");
        setDriverClass((String)YamlUtils.getProperty(datasource, "driverClassName"));

        Map<?, ?> master = (Map)YamlUtils.getProperty(datasource, "druid.master");
        setUrl((String)YamlUtils.getProperty(master, "url"));
        setUser((String)YamlUtils.getProperty(master, "username"));

        // 解密密码
        String password = (String)YamlUtils.getProperty(master, "password");
        if(StringUtils.isEmpty(publicKey)){
            setPassword(password);
        }else{
            setPassword(ConfigTools.decrypt(publicKey, password));
        }
    }
}

public class YamlUtils {
    public static Map<?, ?> loadYaml(String fileName) throws FileNotFoundException {
        InputStream in = YamlUtils.class.getClassLoader().getResourceAsStream(fileName);
        return StringUtils.isNotEmpty(fileName) ? (LinkedHashMap<?, ?>) new Yaml().load(in) : n
ll;
    }

    public static Object getProperty(Map<?, ?> map, Object qualifiedKey) {
        if (map != null && !map.isEmpty() && qualifiedKey != null) {
            String input = String.valueOf(qualifiedKey);
            if (!"".equals(input)) {
                if (input.contains(".")) {
                    int index = input.indexOf(".");
                    String left = input.substring(0, index);
                    String right = input.substring(index + 1, input.length());
                    return getProperty((Map<?, ?>) map.get(left), right);
                } else if (map.containsKey(input)) {
                    return map.get(input);
                } else {
                    return null;
                }
            }
        }
```

```
        return null;
    }
}
```

数据库连接信息application-druid.yml配置如下：

```
# https://github.com/alibaba/druid/wiki/如何在Spring-Boot中配置数据库密码加密?
# https://github.com/alibaba/druid/wiki/使用ConfigFilter
# 公钥
publickey: MFwwDQYJKoZIhvcNAQEABCDDSwAwSAJBANc0H+pBCEFSdn/JfNYABCDEFZcfe1/
mFWLJ7x+F2kCexIJTzngF37bjHbEZPf5j6/1gZnFNNyVJrzZ7vdt1ocUCAwEAAQ==

# 数据源配置
spring:
    datasource:
        type: com.alibaba.druid.pool.DruidDataSource
        driverClassName: com.mysql.cj.jdbc.Driver
        druid:
            # 主库数据源
            master:
                url: jdbc:mysql://172.16.220.14:3306/db?useUnicode=true&characterEncoding=utf
&zeroDateTimeBehavior=convertToNull&useSSL=true&serverTimezone=GMT%2B8
                username: root
                password: JwabcdeSSHvAAkPWYWleVQfxruuSOICmBzqHKGiyEsE5t2QawAIOPaQYD
qB2345667UTHplmHvAl9/C6zX61uQ==
            # 从库数据源
            slave:
                # 从数据源开关/默认关闭
                enabled: true
                url: jdbc:mysql://172.16.220.13:3306/db?useUnicode=true&characterEncoding=utf
&zeroDateTimeBehavior=convertToNull&useSSL=true&serverTimezone=GMT%2B8
                username: root
                password: JwabcdeSSHvAAkPWYWleVQfxruuSOICmBzqHKGiyEsE5t2QawAIOPaQYD
qB2345667UTHplmHvAl9/C6zX61uQ==
            # 初始连接数
            initialSize: 5
            # 最小连接池数量
            minIdle: 10
            # 最大连接池数量
            maxActive: 20
            # 配置获取连接等待超时的时间
            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
            timeBetweenEvictionRunsMillis: 60000
            # https://github.com/alibaba/druid/wiki/怎么保存Druid的监控记录
            # 配置timeBetweenLogStatsMillis>0之后，DruidDataSource会定期把监控数据输出到日
中
            timeBetweenLogStatsMillis: 300000
            # 配置一个连接在池中最小生存的时间，单位是毫秒
            minEvictableIdleTimeMillis: 300000
            # 配置一个连接在池中最大生存的时间，单位是毫秒
            maxEvictableIdleTimeMillis: 900000
            # 配置检测连接是否有效
            validationQuery: SELECT 1 FROM DUAL
```

```
        testWhileIdle: true
        testOnBorrow: false
        testOnReturn: false
        # 配置 connection-properties，启用加密，配置公钥。
        connection-properties: "config.decrypt=true;config.decrypt.key=${publickey}"
```

公钥生成可参见上方连接地址：https://github.com/alibaba/druid/wiki/如何在Spring-Boot中配数据库密码加密?

此处需要说明的一点是，springboot中的logback.xml文件需要配置为logback-spring.xml以便sprinboot能够正常加载，同时，此处也不能使用自动注入的DataSource，因为Spring的Logging加载的早，此时DataSource还未初始化，具体可参见LoggingApplicationListener。

数据库表结构

```
-- ----------------------------
-- Table structure for logging_event
-- ----------------------------
DROP TABLE IF EXISTS `logging_event`;
CREATE TABLE `logging_event` (
  `timestmp` bigint(20) NOT NULL,
  `formatted_message` text CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `logger_name` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `level_string` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `thread_name` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `reference_flag` smallint(6) NULL DEFAULT NULL,
  `arg0` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `arg1` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `arg2` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `arg3` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `caller_filename` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `caller_class` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `caller_method` varchar(254) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `caller_line` char(4) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `event_id` bigint(20) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`event_id`) USING BTREE
) ENGINE = MyISAM AUTO_INCREMENT = 2 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for logging_event_exception
-- ----------------------------
DROP TABLE IF EXISTS `logging_event_exception`;
```

```
CREATE TABLE `logging_event_exception`  (
  `event_id` bigint(20) NOT NULL,
  `i` smallint(6) NOT NULL,
  `trace_line` varchar(5000) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL

  PRIMARY KEY (`event_id`, `i`) USING BTREE
) ENGINE = MyISAM CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FOR
AT = Dynamic;

-- ----------------------------
-- Table structure for logging_event_property
-- ----------------------------
DROP TABLE IF EXISTS `logging_event_property`;
CREATE TABLE `logging_event_property`  (
  `event_id` bigint(20) NOT NULL,
  `mapped_key` varchar(245) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NU
L,
  `mapped_value` text CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT
NULL,
  PRIMARY KEY (`event_id`, `mapped_key`) USING BTREE
) ENGINE = MyISAM CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FOR
AT = Dynamic;
```

结论

配置完成，WARN以上级别的日志会自动异步到数据库，以便后期问题排查，以及分析。