9-日志框架与全局日志管理

1.日志框架的体系结构

刚刚接触到日志的同学可能会被各种日志框架吓到，包括各种日志框架之间的jar总是发生冲突，另很多小伙伴头疼不已。本章将学习各种 Java 日志框架发展过程，以及他们之间的关系，以及如何选型。

一、五花八门的日志工具包

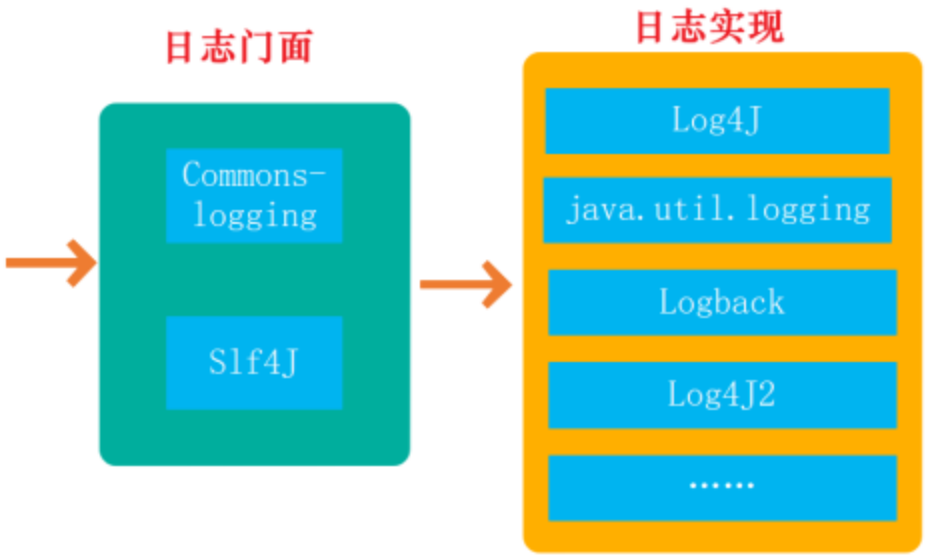
1.1. 日志框架

- JDK java.util.logging 包:java.util.logging 是 jdk1.4 发布的 java 日志包, 可以说是应用比较久远的日志工具包
- log4j： apache 的一个开源项目，提供了强有力的 java 日志支持，甚至他也提供了其他语言包括 C、C++、.Net、PL/SQL 的接口，从而实现多语言并存的分布式环境日志打印。目前已经停止更新，所以不推荐使用。
- Logback：由log4j创始人设计的另一个开源日志组件，作为Spring Boot默认的日志框架，应用比较广泛。
- log4j2 ： Apache Log4j2是对Log4j的升级，它比其前身Log4j1.x提供了重大改进，并提供了Logback中可用的许多改进，同时修复了Logback架构中的一些问题。它基于LMAX公司开发Disruptor（一个开源的无锁并发框架），改善了Log4j和Logback在架构设计方面的缺陷，具有超高的吞吐量和低延迟，性能比Log4j1.x和Logback更好。

1.2.日志门面

- commons-logging： Apache commons类库中的一员，他作为一个日志门面，能够自动选择使用 log4j 还是 JDK logging，但是他不依赖Log4j，JDK Logging的API。如果项目的classpath中包含了log4j的类库，就会使用log4j，否则就使用JDK Logging。
- SLF4J： 可以说是目前应用最为广泛的日志门面了，它提供了一个日志抽象层，允许你在后台使用任意一个日志类库。如：log4j、log4j2、logback

1.3日志门面存在的意义



为什么不直接使用日志框架，而是搞出一个日志门面?

日志门面（SLF4J）主要是为了给Java日志访问提供一套标准、规范的API框架，其主要意义在于提供接口，具体的实现可以交由其他日志框架来实现，例如log4j和logback等。 对于一般的Java项目而言，日志框架会选择slf4j-api作为门面，配上具体的实现框架（log4j、log4j2、logback等），中间使用桥接器完成桥接。

前面介绍的几种日志框架，每一种日志框架都有自己单独的API，要使用对应的框架就要使用其对应的API，这就大大的增加应用程序代码对于日志框架的耦合性要求。有了SLF4J这个门面之后，程序员永远都是面向SLF4J编程，可以实现简单快速地替换底层的日志框架而不会导致业务代码需要做相应的修改。

在使用 SLF4J 进行日志记录时，通常都需要在每个需要记录日志的类中定义 Logger 变量，如下所示：

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController
public class LogTestController {
    private static final Logger logger =
LoggerFactory.getLogger(LogTestController.class);

    @GetMapping("/test")
    public void test(){
        logger.trace("Trace 日志...");
        logger.debug("Debug 日志...");
        logger.info("Info 日志...");
        logger.warn("Warn 日志...");
        logger.error("Error 日志...");
    }
}
```

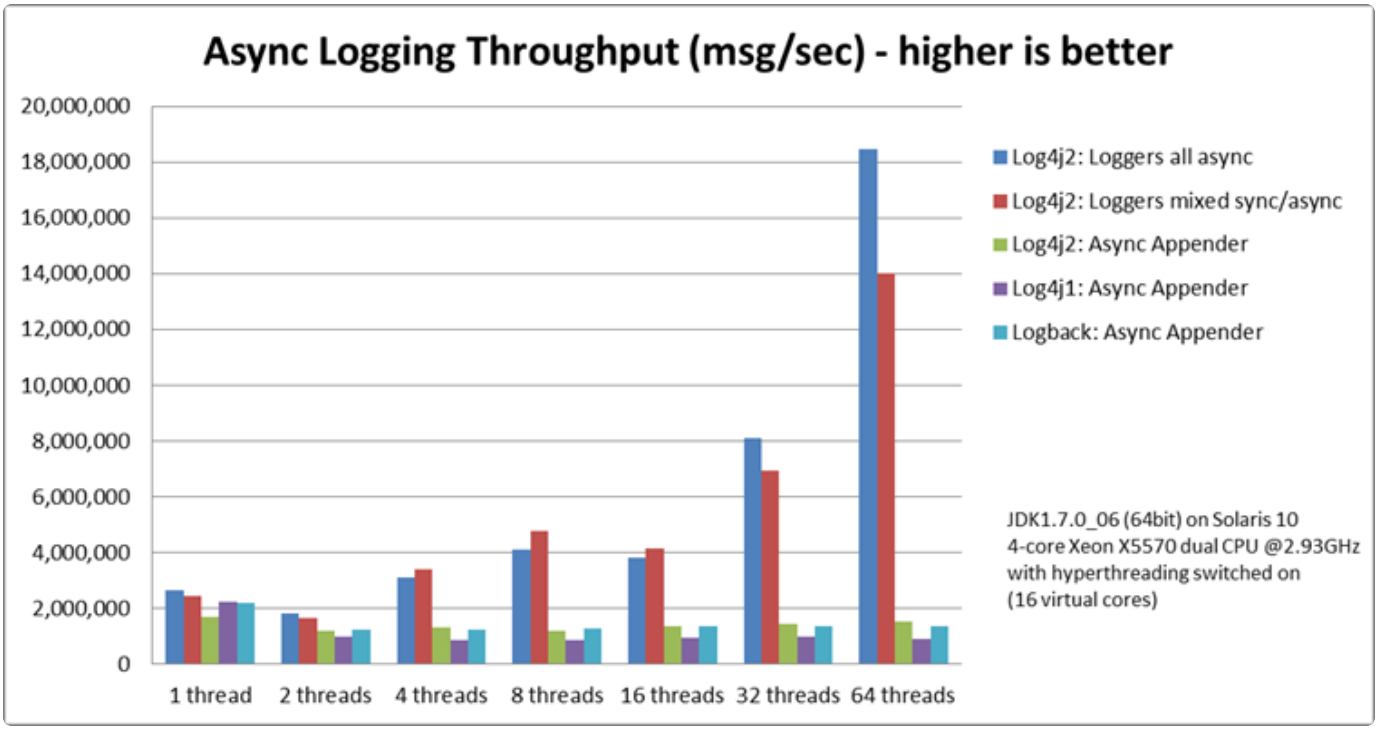
这显然属于重复性劳动，降低了开发效率，如果你在项目中引入了 Lombok，那么可以使用它提供的 @Slf4j 注解来自动生成上面那个变量，默认的变量名是 log，如果我们想采用惯用的 LOGGER 变量名，那么可以在工程的 main/java 目录中增加 lombok.config 文件，并在文件中增加 lombok.log.fieldName=LOGGER 的配置项即可。

二、日志框架选型

- Spring Boot 默认的日志记录框架使用的是 Logback
- 其中 Log4j 可以认为是一个过时的函数库，已经停止更新，不推荐使用，相比之下，性能和功能也是最差的。
- logback 虽然是 Spring Boot 默认的，但性能上还是不及 Log4j2，因此，在现阶段，日志记录首选 Log4j2。

SLF4J + Log4j2 是我们推荐的日志记录选型。

性能测试结果



参考:log4j2官网

三、日志级别

细说各日志框架整合配置前，我们先来大致了解下，最常见的日志的几个级别：ERROR, WARN, INFO, DEBUG和TRACE。像其他的，比如ALL、OFF和FATAL之类的开发过程中应该基本上是不会涉及的。所以以下从低到高一次介绍下常见的日志级别。

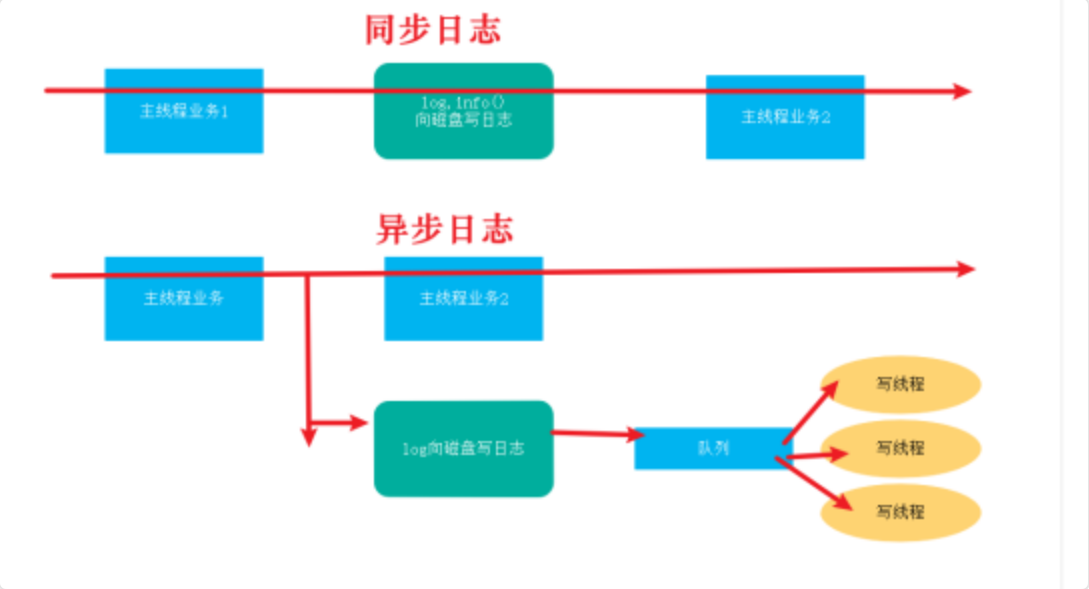
1. TRACE：追踪。一般上对核心系统进行性能调试或者跟踪问题时有用，此级别很低，一般上是不开启的，开启后日志会很快就打满磁盘的。
2. DEBUG:调试。这个大家应该不陌生了。开发过程中主要是打印记录一些运行信息之类的。
3. INFO:信息。这个是最常见的了，大部分默认就是这个级别的日志。一般上记录了一些交互信息，一些请求参数等等。可方便定位问题，或者还原现场环境的时候使用。此日志相对来说是比较重要的。
4. WARN:警告。这个一般上是记录潜在的可能会引发错误的信息。比如启动时，某某配置文件不存在或者某个参数未设置之类的。
5. ERROR:错误。这个也是比较常见的，一般上是在捕获异常时输出，虽然发生了错误，但不影响系统的正常运行。但可能会导致系统出错或是宕机等。

日志级别从小到大为trace<debug<info<warn<error<fatal，由于通常日志框架默认日志级别设置为INFO，因此trace和debug级别的日志都看不到。

```
2022-04-17 13:59:16.566  INFO c.z.b.l.controller.LogTestController    : Info 日志...
2020-04-17 13:59:16.566  WARN  c.z.b.l.controller.LogTestController   : Warn 日志...
2020-04-17 13:59:16.566  ERROR c.z.b.l.controller.LogTestController  : Error 日志...
```

四、常见术语概念解析

1. appender：主要控制日志输出到哪里，比如：文件、数据库、控制台打印等
2. logger: 用来设置某一个包或者具体某一个类的日志打印级别、以及指定appender
3. root：也是一个logger，是一个特殊的父logger。所有的子logger最终都会将输出流交给root，除非在子logger中配置了additivity="false"。
4. rollingPolicy：所有日志都放在一个文件是不好的，所以可以指定滚动策略，按照一定周期或文件大小切割存放日志文件。
5. RolloverStrategy：日志清理策略。通常是指日志保留的时间。
6. 异步日志：单独开一个线程做日志的写操作，达到不阻塞主线程的目的。



- 同步日志，主线程要等到日志写磁盘完成之后，才能继续向下执行
- 异步日志，主线程写日志只是将日志消息放入一个队列，之后就继续向下执行，这个过程是在内存层面完成的。之后由专门的线程从队列中获取日志数据写入磁盘，所以不阻塞主线程。主线程（核心业务代码）执行效率很高。

2.logback日志框架配置

logback既可以通过application配置文件进行日志的配置，又可以通过logback-spring.xml进行日志的配置。通常情况下，使用全局配置文件application.yml或properties进行配置就足够了，如果您的日志输出需求特别复杂而且需求比较个性化，可以考虑使用logback-spring.xml的配置方式。

一、application配置文件实现日志配置

我们可以在applicaition.properties(yml) 文件中进行日志的配置

YAML | 复制代码

```yaml
logging:
  level:
    root: info
    com.mqxu.boot.log.controller: debug
  file:
    path: /Users/mqxu/Desktop/logs
    name:  /Users/mqxu/Desktop/logs/boot.log
    max-size: 10MB
    max-history: 10
  pattern:
    console: '%red(%d{yyyy-MM-dd HH:mm:ss}) %green([%thread]) %highlight(%-5level) %boldMagenta(%logger{10}) - %cyan(%msg%n)'
    file: '%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level [%thread] %logger : %msg%n'
```

- logging.level.root=info指定整个系统的默认日志级别是info，日志级别统一化
- logging.level.com.zimug.boot.launch.controller=debug,指定某个特定的package的日志级别是debug，日志级别个性化。优先级角度，个性配置大于统一配置。
- logging.file.path将日志输出到指定目录，如果不指定logging.file.name，日志文件的默认名称是spring.log。配置了logging.file.name之后，logging.file.path配置失效。

- 无论何种设置，Spring Boot都会自动按天分割日志文件，也就是说每天都会自动生成一个新的log文件，而之前的会自动打成GZ压缩包。# 日志文件大小
- 可以设置logging.file.max-size=10MB分割的每个日志的文件最大容量，超过这个size之后日志继续分隔。
- 可以设置保留的日志时间logging.file.max-history=10，以天为单位
- logging.pattern.file输出到文件中的日志的格式
- logging.pattern.console控制台输出日志的格式，为了在控制台调试时候显示效果更清晰，为日志增加了颜色。red、green等等

日志格式占位符

配合这张图，看一下占位符和logging.pattern.console格式配置之间的关系

```
14:35:19 [restartedMain] INFO  o.s.b.a.w.s.WelcomePageHandlerMapping - Adding
age: class path resource [public/index.html]
14:35:21 [restartedMain] INFO  o.s.b.d.a.OptionalLiveReloadServer - LiveReload
s running on port 35729
14:35:21 [restartedMain] INFO  o.s.b.w.e.t.TomcatWebServer - Tomcat started on
8888 (http) with context path ''
14:35:21 [restartedMain] INFO  c.z.b.l.BootLaunchApplication - Started
hApplication in 9.084 seconds (JVM running for 10.133)
```

- %d{HH:mm:ss.SSS}：日志输出时间（red）
- %thread：输出日志的进程名字，这在Web应用以及异步任务处理中很有用 （green）
- %-5level：日志级别，并且使用5个字符靠左对齐 （highlight高亮蓝色）
- %logger：日志输出类的名字 （boldMagenta粗体洋红色）
- %msg：日志消息 （cyan蓝绿色）
- %n：平台的换行符

二、使用logback-spring.xml实现日志配置

2.1.需求

一般情况下，使用全局配置文件application.yml或properties进行配置就足够了，如果你的日志输出需求特别复杂，可以考虑使用logback-spring.xml的配置方式。

spring boot 用自带的logback打印日志，多环境打印：

1. 生产环境输出到控制台和文件,一天一个文件,保留30天.
2. 开发环境输出到控制台和打印sql(mybatis)输出，生产环境不打印这个信息

3. 测试环境只输出到控制台。不输出到文件

> 打印Mybatis SQL，只需要把使用到Mybatis的package的日志级别调整为DEBUG，就可以将SQL打印出来。

前提：项目已经支持application.yml的profile多环境配置

2.2.需求实现

因为logback是spring boot的默认日志框架，所以不需要引入maven依赖，直接上logback-spring.xml放在resources下面

XML | 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <!--引入默认的一些设置-->
  <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
  <!--web信息-->
  <logger name="org.springframework.web" level="info"/>

  <!--写入日志到控制台的appender,用默认的,但是要去掉charset,否则windows下tomcat下乱码-->
  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>${CONSOLE_LOG_PATTERN}</pattern>
    </encoder>
  </appender>

  <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->
  <property name="LOG_PATH" value="/Users/mqxu/Desktop/logs"/>
  <!--写入日志到文件的appender-->
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <!--日志文件输出的文件名,每天一个文件-->
      <FileNamePattern>${LOG_PATH}.%d{yyyy-MM-dd}.log</FileNamePattern>
      <!--日志文件保留天数-->
      <maxHistory>30</maxHistory>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
    </encoder>
    <!--日志文件最大的大小-->
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <MaxFileSize>10MB</MaxFileSize>
    </triggeringPolicy>
  </appender>

  <!--异步写日志到文件-->
  <appender name="asyncFileAppender" class="ch.qos.logback.classic.AsyncAppender">
    <discardingThreshold>0</discardingThreshold>
    <queueSize>500</queueSize>
    <appender-ref ref="FILE"/>
```

```
    </appender>

    <!--生产环境:打印控制台和输出到文件-->
    <springProfile name="prod">
      <root level="info">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="asyncFileAppender"/>
      </root>
    </springProfile>

    <!--开发环境:打印控制台-->
    <springProfile name="dev">
      <!-- 打印sql -->
      <logger name="com.mqxu.boot.log" level="DEBUG"/>
      <root level="DEBUG">
        <appender-ref ref="CONSOLE"/>
      </root>
    </springProfile>

    <!--测试环境:打印控制台-->
    <springProfile name="test">
      <root level="info">
        <appender-ref ref="CONSOLE"/>
      </root>
    </springProfile>
  </configuration>
```

异步日志配置：

- 异步日志queueSize 默认值256，异步日志队列的容量。
- discardingThreshold：当异步日志队列的剩余容量小于这个阈值，会丢弃TRACE, DEBUG or INFO级别的日志。如果不希望丢弃日志（即全量保存），那可以设置为0。但是当队列占满后，非阻塞的异步日志会变成阻塞的同步日志。所以在高并发低延迟要求的系统里面针对不重要的日志可以设置discardingThreshold丢弃策略，值大于0。

2.3.测试

上面配置完成之后，可以使用如下代码测试一下，是否满足了2.1节中提出的需求。

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController
public class LogTestController {
    private static final Logger logger =
LoggerFactory.getLogger(LogTestController.class);

    @GetMapping("/testlog")
    public void test(){
        logger.trace("Trace 日志...");
        logger.debug("Debug 日志...");
        logger.info("Info 日志...");
        logger.warn("Warn 日志...");
        logger.error("Error 日志...");
    }
}
```

3.log4j2日志框架配置

一、引入maven依赖

Spring Boot默认使用LogBack，但是我们没有看到显示依赖的jar包，其实是因为所在的jar包spring-boot-starter-logging都是作为spring-boot-starter-web或者spring-boot-starter依赖的一部分。如果这里要使用Log4j2，需要从spring-boot-starter-web中去掉spring-boot-starter-logging依赖，同时显示声明使用Log4j2的依赖jar包，具体如下:

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>

```

XML | 复制代码

二、添加配置文件log4j2-spring.xml

在resources目录下新建一个log4j2-spring.xml文件，放在src/main/resources目录下即可被Spring Boot应用识别。

XML | 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <properties>
    <!--日志输出位置-->
    <property name="LOG_HOME">/Users/mqxu/Desktop/logs</property>
  </properties>

  <Appenders>
    <!-- 将日志输出到控制台-->
    <Console name="CONSOLE" target="SYSTEM_OUT">
      <!--设置日志格式及颜色-->
      <PatternLayout
                     pattern="[%style{%d}{bright,green}][%highlight{%p}]
[%style{%t}{bright,blue}][%style{%C}{bright,yellow}]:
%msg%n%style{%throwable}{red}"
                     disableAnsi="false" noConsoleNoAnsi="false"/>
    </Console>

    <!-- 将日志输出到文件-->
    <RollingFile name="FILE-APPENDER"
                 fileName="${LOG_HOME}/log4j2-demo.log"
                 filePattern="${LOG_HOME}/log4j2-demo-%d{yyyy-MM-dd}-
%i.log">
      <!--设置日志格式-->
      <PatternLayout>
        <pattern>[%d][%p][%t][%C] %m%n</pattern>
      </PatternLayout>
      <Policies>
        <!-- 设置日志文件切分参数 -->
        <SizeBasedTriggeringPolicy size="100 MB"/>
        <TimeBasedTriggeringPolicy/>
      </Policies>
      <!--设置最大存档数-->
      <DefaultRolloverStrategy max="20"/>
    </RollingFile>
  </Appenders>

  <Loggers>
    <!-- 根日志设置 -->
    <Root level="debug">
      <AppenderRef ref="CONSOLE" level="debug"/>
      <AppenderRef ref="FILE-APPENDER" level="info"/>
    </Root>

    <!--spring日志-->
```

```
        <Logger name="org.springframework" level="info"/>
        <!-- mybatis日志 -->
        <Logger name="com.mybatis" level="warn"/>
      </Loggers>
    </configuration>
```

- 两个Appender，一个叫做CONSOLE用于输出日志到控制台，一个叫做FILE-APPENDER输出日志到文件
- PatternLayout用于指定输出日志的格式，[%d][%p][%t][%C] %m%n 这些占位符将结合下文测试结果为大家介绍
- Policies用于指定文件切分参数
  - TimeBasedTriggeringPolicy默认的size是1，结合filePattern定义%d{yyyy-MM-dd}，则每天生成一个文件（最小的时间切分粒度是小时）
  - <SizeBasedTriggeringPolicy size="100 MB"/> 当文件大小到100MB的时候，切分一个新的日志文件
- <DefaultRolloverStrategy max="20"/>表示文件最大的存档数量，多余的将被删除

三、自定义配置文件

但是我们通常会有这样一个需求，就是不同的环境使用不同的配置，比如：我们需要三个log4j2 xml文件：

- log4j2-dev.xml 开发环境日志配置
- log4j2-prod.xml 生产环境日志配置
- log4j2-test.xml 测试环境日志配置

但是Spring Boot并不知道log4j2-<profile>.xml这些配置文件是干什么的，所以需要通过在application.yml文件中显示声明才行。

举例：在application-dev.yml里面使用log4j2-dev.xml配置文件

YAML | 复制代码

```yaml
logging:
  config: classpath:log4j2-dev.xml
```

以此类推，在application-prod.yml里面使用log4j2-prod.xml配置文件，在application-test.yml里面使用log4j2-test.xml配置文件。

说一下占位符

```xml
<PatternLayout pattern="[%style{%d}{bright,green}][%highlight{%p}]
[%style{%t}{bright,blue}][%style{%C}{bright,yellow}]:
%msg%n%style{%throwable}{red}"
                    disableAnsi="false" noConsoleNoAnsi="false"/>
```

- %d： date时间
- %p： 日志级别
- %t： thread线程名称
- %C： class类文件名称
- %msg：日志信息
- %n换行
- %style{%throwable}{red} 加样式，异常信息红色显示

4.拦截器实现统一访问日志

一、需求

我们本节要实现的需求

- 针对当前系统的每一次接口访问，要记录是什么人访问的（用户名）、什么时间访问的、访问耗时多长时间、使用什么HTTP method方法访问的、访问结果如何等。可以称为审计日志。
- 将访问记录审计日志，输出到一个单独的日志文件access.log

二、定义访问日志内容记录实体类

Java | 复制代码

```java
package com.mqxu.boot.domain;

import com.mqxu.boot.util.FormatUtils;
import lombok.Data;

import java.util.Date;

/**
 * @description: 访问日志内容记录实体类
 * @author: mqxu
 * @date: 2022-04-04
 **/
@Data
public class AccessLog {
    /**
     * 访问者用户名
     */
    private String username;
    /**
     * 请求路径
     */
    private String uri;
    /**
     * 请求消耗时长
     */
    private Integer duration;
    /**
     * http 方法：GET、POST等
     */
    private String httpMethod;
    /**
     * http 请求响应状态码
     */
    private Integer httpStatus;
    /**
     * 访问者ip
     */
    private String ip;
    /**
     * 此条记录的创建时间
     */
    private Date createTime;

    @Override
    public String toString() {
```

```
46            return "{" + "username=" + this.username + "," + "uri=" +
      this.uri + "," + "duration=" + this.duration + "," + "httpMethod=" +
      this.httpMethod + "," + "httpStatus=" + this.httpStatus + "," + "ip=" +
      this.ip + "," + "createTime=" + FormatUtils.forTime(this.createTime) +
      "}";
47        }
48
49    }
```

三、自定义日志拦截器

通过自定义拦截器的方式，记录审计日志。

- 拦截器的preHandle方法，可以用于拦截请求处理开始。用于记录请求开始时间等信息保存到Http Request，用于后续计算请求时长。
- 拦截器的postHandle方法，可以用于拦截请求处理完成。可以从Request对象获取开始时间，计算本次请求总的处理时长等信息。

Java | 复制代码

```java
package com.mqxu.boot.interceptor;

import com.mqxu.boot.domain.AccessLog;
import com.mqxu.boot.util.AddressIpUtils;
import lombok.extern.slf4j.Slf4j;
import org.springframework.lang.Nullable;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Date;

/**
 * @description: 自定义日志拦截器
 * @author: mqxu
 * @date: 2022-04-04
 **/
@Component
@Slf4j
public class AccessLogInterceptor implements HandlerInterceptor {

    /**
     * 进入SpringMVC的Controller之前开始记录日志实体
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object o) throws Exception {
        //创建日志实体
        AccessLog accessLog = new AccessLog();
        // 设置IP地址
        accessLog.setIp(request.getRemoteAddr());
        //accessLog.setIp(AddressIpUtils.getIpAddress(request));
        //设置请求方法,GET,POST...
        accessLog.setHttpMethod(request.getMethod());
        //设置请求路径(端点)
        accessLog.setUri(request.getRequestURI());
        //设置请求开始时间
        request.setAttribute("sendTime", System.currentTimeMillis());
        //设置请求实体到request内，方便afterCompletion方法调用
        request.setAttribute("accessLog", accessLog);
        return true;
    }

```

```
45
46      @Override
47      public void postHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler, @Nullable ModelAndView
    modelAndView) throws Exception {
48          //获取本次请求日志实体
49          AccessLog accessLog = (AccessLog)
    request.getAttribute("accessLog");
50          //获取请求错误码，根据需求存入数据库，这里不保存
51          int status = response.getStatus();
52          accessLog.setHttpStatus(status);
53          // 设置访问者
54          accessLog.setUsername(request.getParameter("username"));
55          //当前时间
56          long currentTime = System.currentTimeMillis();
57          //请求开始时间
58          long sendTime =
    Long.parseLong(request.getAttribute("sendTime").toString());
59          //设置请求时间差
60          accessLog.setDuration(Integer.valueOf((currentTime - sendTime) +
    ""));
61          accessLog.setCreateTime(new Date());
62          //打印日志
63          log.info(String.valueOf(accessLog));
64      }
65
66      @Override
67      public void afterCompletion(HttpServletRequest request,
    HttpServletResponse response, Object o, Exception e) throws Exception {
68          log.info("afterCompletion... ");
69      }
70  }
```

LoggerFactory.getLogger("ACCESS-LOG")获取一个日志配置中的Logger的名字，用于打印日志输出，持久化到日志文件里。

四、拦截器注册

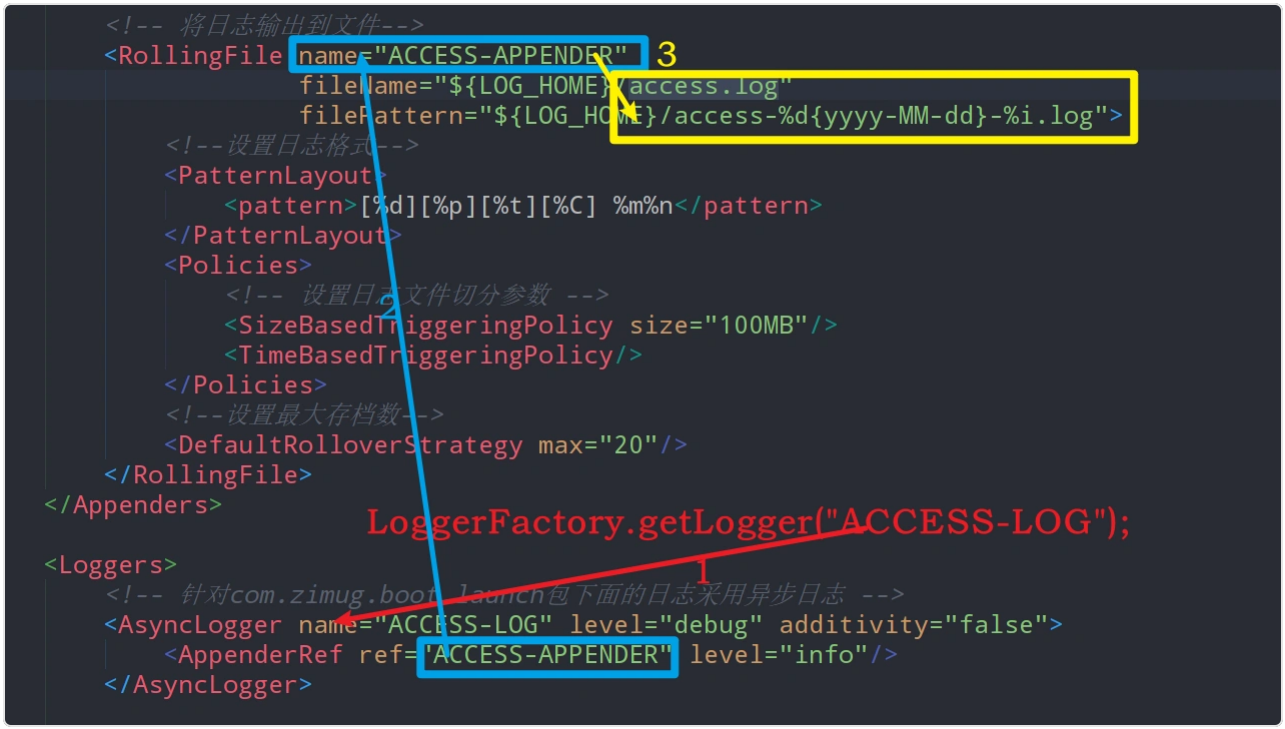
```java
@Configuration
public class MyWebMvcConfigurer implements WebMvcConfigurer {

    //设置排除路径，spring boot 2.*，注意排除掉静态资源的路径，不然静态资源无法访问
    private final String[] excludePath = {"/static"};

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new AccessLogInterceptor()).addPathPatterns("/**").excludePathPatterns(excludePath);
    }
}
```

五、"ACCESS-LOG"的日志Logger定义

配置参考以Log4J2配置为例



- LoggerFactory.getLogger("ACCESS-LOG") 代码去配置文件里面找一个name为ACCESS-LOG的Logger配置。
- 该Logger是一个AsyncLogger，指向的输出目标是ACCESS-APPENDER
- ACCESS-APPENDER是一个日志文件输出配置，日志文件是access-log.log

XML | 复制代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <properties>
        <!--日志输出位置-->
        <property name="LOG_HOME">/Users/mqxu/Desktop</property>
    </properties>
    <Appenders>
        <!-- 将日志输出到文件-->
        <RollingFile name="ACCESS-APPENDER"
                     fileName="${LOG_HOME}/access.log"
                     filePattern="${LOG_HOME}/access-%d{yyyy-MM-dd}-%i.log">
            <!--设置日志格式-->
            <PatternLayout>
                <pattern>[%d][%p][%t][%C] %m%n</pattern>
            </PatternLayout>
            <Policies>
                <!-- 设置日志文件切分参数 -->
                <SizeBasedTriggeringPolicy size="100MB"/>
                <TimeBasedTriggeringPolicy/>
            </Policies>
            <!--设置最大存档数-->
            <DefaultRolloverStrategy max="20"/>
        </RollingFile>
    </Appenders>

    <Loggers>
        <AsyncLogger name="ACCESS-LOG" level="debug" additivity="false">
            <AppenderRef ref="ACCESS-APPENDER" level="info"/>
        </AsyncLogger>
    </Loggers>
</configuration>
```