通过之前的范例，我们已经为trace-a和trace-b引入了Spring Cloud Sleuth的基础模块spring-cloud-starter-sleuth，实现了为各微服务的日志信息中添加跟踪信息的功能。但是，由于日志文件都离散的存储在各个服务实例的文件系统之上，仅仅通过查看日志文件来分析我们的请求链路依然是一件相当麻烦的差事，所以我们还需要一些工具来帮助我们集中的收集、存储和搜索这些跟踪信息。引入基于日志的分析系统是一个不错的选择，比如：ELK平台，它可以轻松的帮助我们来收集和存储这些跟踪日志，同时在需要的时候我们也可以根据Trace ID来轻松地搜索出对应请求链路相关的明细日志。

ELK平台主要有由ElasticSearch、Logstash和Kiabana三个开源免费工具组成：

- Elasticsearch是个开源分布式搜索引擎，它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。
- Logstash是一个完全开源的工具，他可以对你的日志进行收集、过滤，并将其存储供以后使用。
- Kibana 也是一个开源和免费的工具，它Kibana可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助您汇总、分析和搜索重要数据日志。

Spring Cloud Sleuth在与ELK平台整合使用时，实际上我们只要实现与负责日志收集的Logstash完成数据对接即可，所以我们需要为Logstash准备json格式的日志输出。由于Spring Boot应用默认使用了logback来记录日志，而Logstash自身也有对logback日志工具的支持工具，所以我们可以直接通过在logback的配置中增加对logstash的appender，就能非常方便的将日志转换成以json的格式存储和输出了。

下面我们来详细介绍一下在之前的范例trace-b工程基础上，如何实现面向Logstash的日志输出配置：

- 在pom.xml依赖中引入logstash-logback-encoder依赖，具体如下：

```
<dependency>
  <groupId>net.logstash.logback</groupId>
  <artifactId>logstash-logback-encoder</artifactId>
  <version>4.6</version>
</dependency>
```

- 在工程/resource目录下创建bootstrap.properties配置文件，将spring.application.name=trace-b配置移动到该文件中去。由于logback-spring.xml的加载在application.properties之前，所以之前的配置logback-spring.xml无法获取到spring.application.name属性，因此这里将该属性移动到最先加载的bootstrap.properties配置文件中。
- 在工程/resource目录下创建logback配置文件logback-spring.xml，具体内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/defaults.xml"/>

    <springProperty scope="context" name="springAppName" source="spring.application.name"/>
    <!-- 日志在工程中的输出位置 -->
    <property name="LOG_FILE" value="${BUILD_FOLDER:-build}/${springAppName}"/>
    <!-- 控制台的日志输出样式 -->
    <property name="CONSOLE_LOG_PATTERN"
              value="%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} %clr(${LOG_LEVEL_PATTERN:-%5p})
%clr([${springAppName:-},%X{X-B3-TraceId:-},%X{X-B3-SpanId:-},%X{X-Span-Export:-}]){yellow} %clr(${PID:-}){magenta} %clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint}
%m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}"/>

    <!-- 控制台Appender -->
    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>INFO</level>
        </filter>
        <encoder>
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
            <charset>utf8</charset>
```

```
        </encoder>
    </appender>

    <!-- 为logstash输出的json格式的Appender -->
    <appender name="logstash" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_FILE}.json</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_FILE}.json.%d{yyyy-MM-dd}.gz</fileNamePattern>
            <maxHistory>7</maxHistory>
        </rollingPolicy>
        <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <timestamp>
                    <timeZone>UTC</timeZone>
                </timestamp>
                <pattern>
                    <pattern>
                        {
                        "severity": "%level",
                        "service": "${springAppName:-}",
                        "trace": "%X{X-B3-TraceId:-}",
                        "span": "%X{X-B3-SpanId:-}",
                        "exportable": "%X{X-Span-Export:-}",
                        "pid": "${PID:-}",
                        "thread": "%thread",
                        "class": "%logger{40}",
                        "rest": "%message"
                        }
                    </pattern>
                </pattern>
            </providers>
        </encoder>
    </appender>

    <root level="INFO">
        <appender-ref ref="console"/>
        <appender-ref ref="logstash"/>
    </root>
</configuration>
```

对logstash支持主要通过名为logstash的appender实现，内容并不复杂，主要是对日志信息的格式化处理，上面为了方便调试查看我们先将json日志输出到文件中。
完成上面的改造之后，我们再将快速入门的示例运行起来，并发起对trace-1的接口访问。此时我们可以在trace-b的工程目录下发现有一个build目录，下面分别创建了以各自应用名称命名的json文件，该文件就是在logback-spring.xml中配置的名为logstash的appender输出的日志文件，其中记录了类似下面格式的json日志：


```
{"@timestamp":"2019-04-18T05:57:17.554+00:00","severity":"INFO","service":"trace-b","trace":"5a37a88fae82f659","span":"f465705303d185d6","exportable":"false","pid":"19996","thread":"http-nio-2412-exec-1","class":"1.s.TracebApplication$$EnhancerBySpringCGLIB$$7f19680a","rest":"===<call trace-b, TraceId=5a37a88fae82f659, SpanId=f465705303d185d6>==="}
{"@timestamp":"2019-04-18T05:57:17.568+00:00","severity":"DEBUG","service":"trace-b","trace":"5a37a88fae82f659","span":"f465705303d185d6","exportable":"false","pid":"19996","thread":"http-
```

nio-2412-exec-1","class":"o.s.web.servlet.DispatcherServlet","rest":"Null ModelAndView returned to DispatcherServlet with name 'dispatcherServlet': assuming HandlerAdapter completed request handling"}