链滴

SpringBoot 是如何实现日志的？

作者：JavaFish

原文链接：https://ld246.com/article/1626435085177

来源网站：链滴

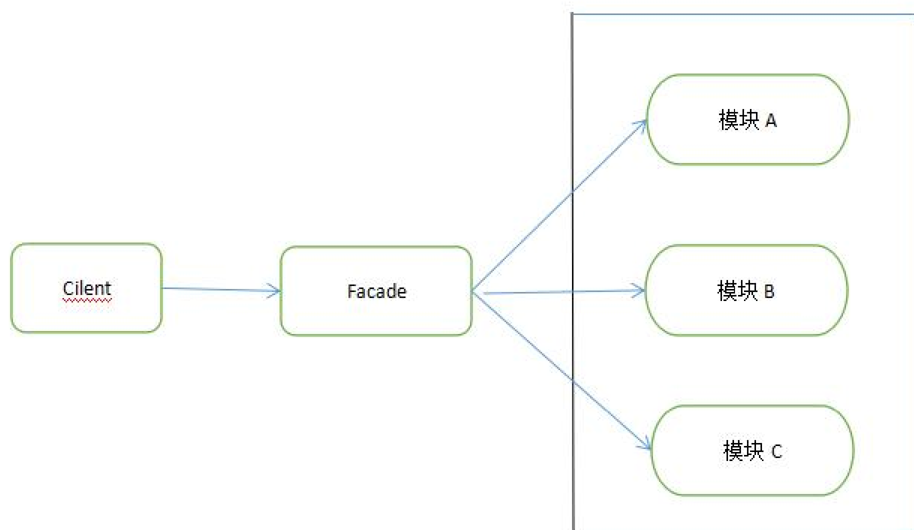
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

前言

休息日闲着无聊看了下 SpringBoot 中的日志实现，把我的理解跟大家说下。

门面模式

说到日志框架不得不说门面模式。门面模式，其核心为外部与一个子系统的通信必须通过一个统一的观对象进行，使得子系统更易于使用。用一张图来表示门面模式的结构为：



简单来说，该模式就是把一些复杂的流程封装成一个接口供给外部用户更简单的使用。这个模式，设计到3个角色。

1）.门面角色：外观模式的核心。它被客户角色调用，它熟悉子系统的功能。内部根据客户角色需求预定了几种功能的组合（模块）。

2）.子系统（模块）角色：实现了子系统的功能。它对客户角色和 Facade 是未知的。它内部可有系统内的相互交互，也可以由供外界调用的接口。

3）.客户角色：通过调用 Facede 来完成要实现的功能。

市面上的日志框架

日志门面	日志实现
JCL（Jakarta Commons Logging）、SLF4j（Simple Logging Facade for Java）、jboss-logging	Log4j 、JUL（java.util.logging）、Log4j2 、 Logback

简单说下，上表的日志门面对应了门面模式中的 Facede 对象，它们只是一个接口层，并不提供日志现；而日志实现则对应着各个子系统或者模块，日志记录的具体逻辑实现，就写在这些右边的框架里；那我们的应用程序就相当于客户端。

为什么要使用门面模式？

试想下我们开发系统的场景，需要用到很多包，而这些包又有自己的日志框架，于是就会出现这样的况：我们自己的系统中使用了 Logback 这个日志系统，我们的系统使用了 Hibernate，Hibernate 使用的日志系统为 jboss-logging，我们的系统又使用了 Spring，Spring 中使用的日志系统为 commons-logging。

这样，我们的系统就不得不同时支持并维护 Logback、jboss-logging、commons-logging 三种日框架，非常不便。解决这个问题的方式就是引入一个接口层，由接口层决定使用哪一种日志系统，而用端只需要做的事情就是打印日志而不需要关心如何打印日志，而上表的日志门面就是这种接口层。

鉴于此，我们选择日志时，就必须从上表左边的日志门面和右边的日志实现各选择一个框架，而 SprigBoot 底层默认选用的就是 SLF4j 和 Logback 来实现日志输出。

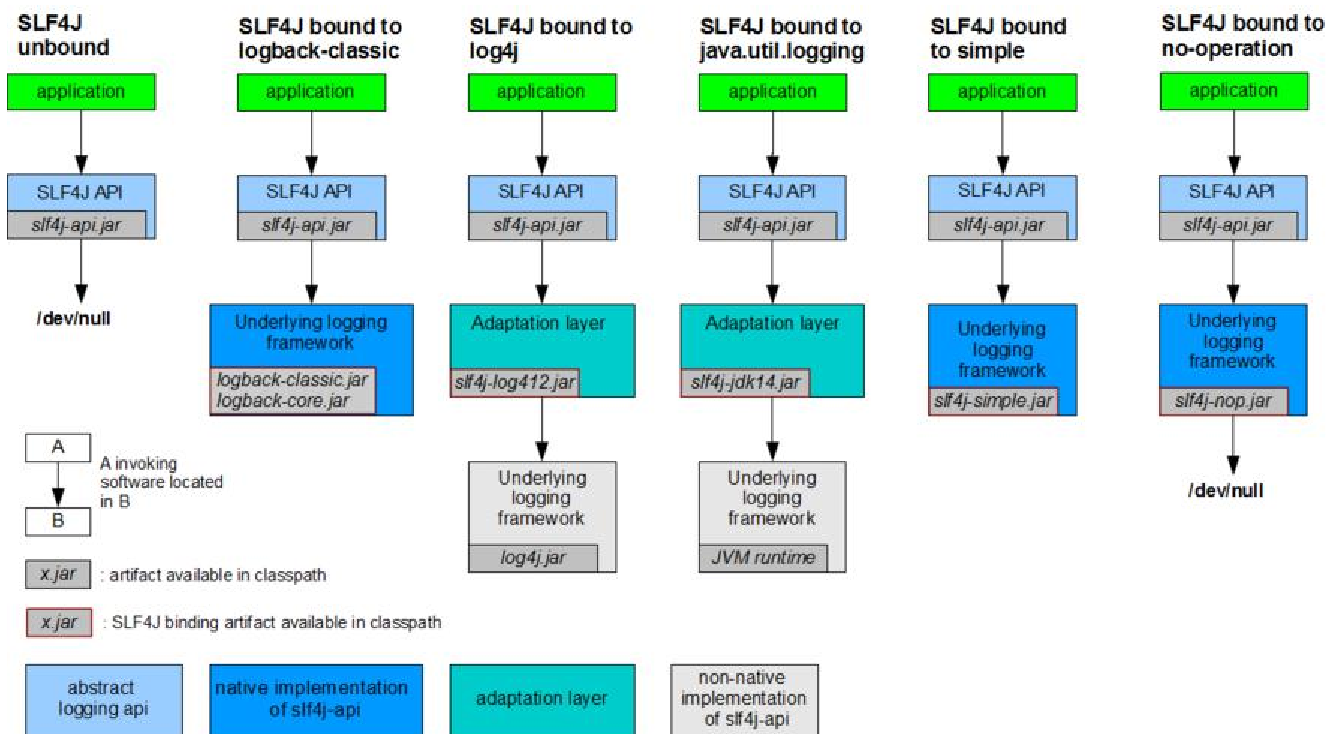
SLF4j 使用

官方文档给出这样一个例子：

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

public class HelloWorld {
  public static void main(String[] args) {
    // HelloWorld.class 就是你要打印的指定类的日志，
    // 如果你想在其它类中打印，那就把 HelloWorld.class 替换成目标类名.class 即可。
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);
    logger.info("Hello World");
  }
}
```

为了理解 slf4j 的工作原理，我翻了下它的官方文档，看到这么一张图：



原文链接：SpringBoot 是如何实现日志的？

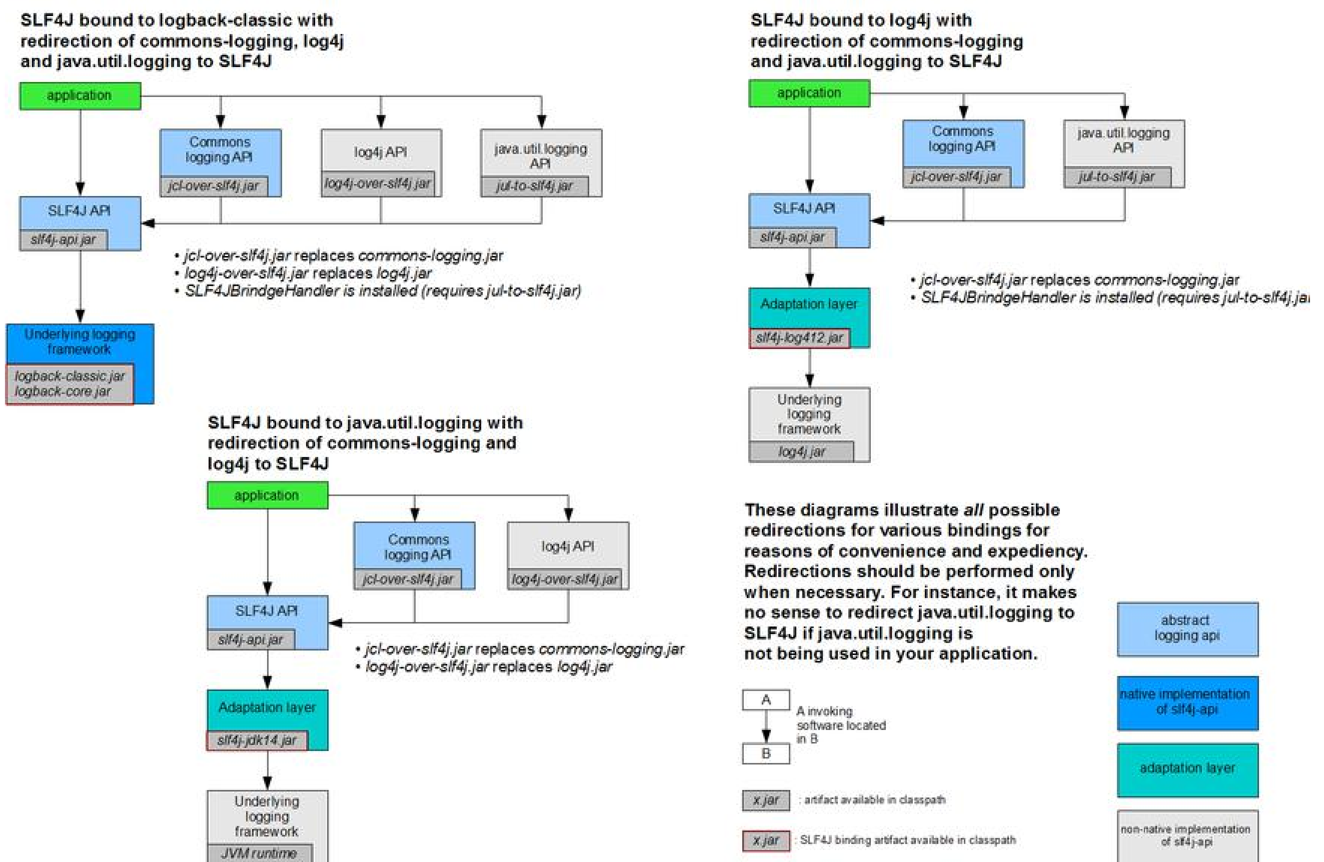
简单解释一下，上图 slf4j 有六种用法，一共五种角色，application 不用说，就是我们的系统；SLF4J API 就是日志接口层（门面）；蓝色和最下面灰色的就是具体日志实现（子系统）；而 Adaptation 是适配层。

解释下，上图第二，第三种用法。其中第二种就是 SpringBoot 的默认用法；而为什么会出现第三种因为 Log4J 出现得比较早，它根本不知道后面会有 SLF4J 这东西。Log4J 不能直接作为 SLF4J 的日实现，所以中间就出现了适配层。第四种同理。

这里提醒下，每一个日志的实现框架都有自己的配置文件。使用 slf4j 以后，**配置文件还是做成日实现框架自己本身的配置文件。比如，Logback 就使用 logback.xml、Log4j 就使用 Log4j.xml 文。

如何让系统中所有的日志都统一到 slf4j？

我继续浏览了下官网，看见这么一张图：



由上图可以看出，让系统中所有的日志都统一到 slf4j 的做法是：

1、**将系统中其他日志框架先排除出去**

2、**用中间包来替换原有的日志框架**

3、**我们导入 slf4j 其他的实现**

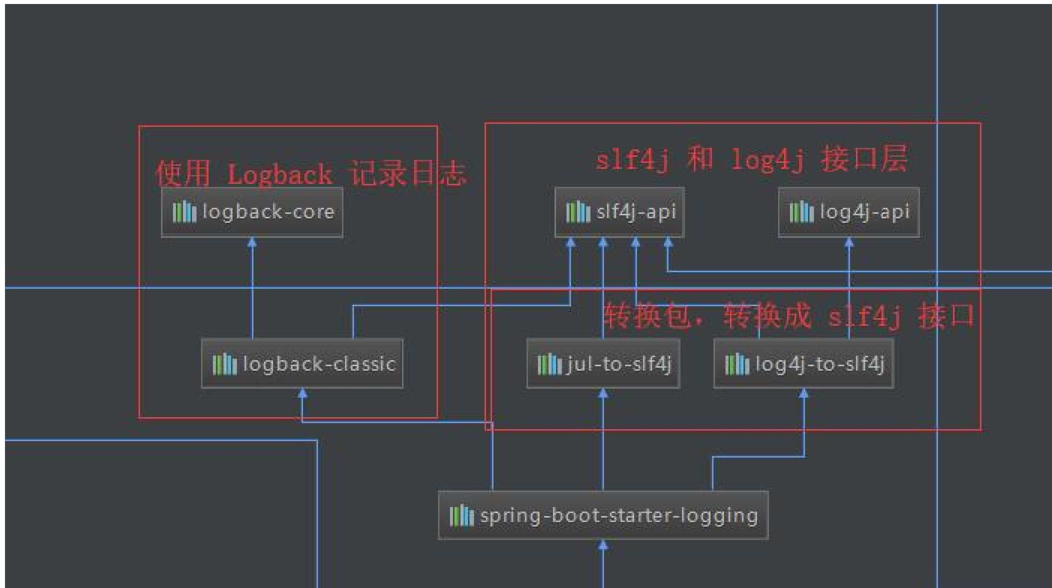
SpringBoot 中的日志关系

原文链接：SpringBoot 是如何实现日志的?

SpringBoot 使用以下依赖实现日志功能：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-logging</artifactId>
    <version>2.1.3.RELEASE</version>
    <scope>compile</scope>
</dependency>
```

spring-boot-starter-logging 有这么一张关系图：



可见，

1、SpringBoot2.x 底层也是使用 slf4j+logback 或 log4j 的方式进行日志记录；

2、SpringBoot 引入中间替换包把其他的日志都替换成了 slf4j；

3、 如果我们要引入其他框架、可以把这个框架的默认日志依赖移除掉。

比如 Spring 使用的是 commons-logging 框架，我们可以这样移除。

```
<dependency>
  <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
      <exclusions>
        <exclusion>
          <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
      </exclusions>
</dependency>
```

SpringBoot 能自动适配所有的日志，而且底层使用 slf4j+logback 的方式记录日志，引入其他框架时候，只需要把这个框架依赖的日志框架排除掉即可。

日志使用

1、默认配置（以 Log4j 框架为例），SpringBoot 默认帮我们配置好了日志：

```
    //记录器
    Logger logger = LoggerFactory.getLogger(getClass());
    @Test
    public void contextLoads() {
        //日志的级别；
        //由低到高   trace<debug<info<warn<error
        //可以调整输出的日志级别；日志就只会在这个级别以以后的高级别生效
        logger.trace("这是trace日志...");
        logger.debug("这是debug日志...");
        // SpringBoot 默认给我们使用的是 info 级别的，没有指定级别的就用SpringBoot 默认规定的
别；root 级别
        logger.info("这是info日志...");
        logger.warn("这是warn日志...");
        logger.error("这是error日志...");
    }
```

2、log4j.properties 修改日志默认配置

```
logging.level.com.nasus=debug

#logging.path=
# 不指定路径在当前项目下生成 springboot.log 日志
# 可以指定完整的路径；
#logging.file=Z:/springboot.log

# 在当前磁盘的根路径下创建 spring 文件夹和里面的 log 文件夹；使用 spring.log 作为默认文件
logging.path=/spring/log

#  在控制台输出的日志的格式
logging.pattern.console=%d{yyyy-MM-dd} [%thread] %-5level %logger{50} - %msg%n
# 指定文件中日志输出的格式
logging.pattern.file=%d{yyyy-MM-dd} === [%thread] === %-5level === %logger{50} ====
%msg%n
```

logging.file	logging.path	Example	Description
(none)	(none)		只在控制台输出
指定文件名	(none)	my.log	输出日志到my.log文件
(none)	指定目录	/var/log	输出到指定目录的 spring.log 文件中

3、指定配置

SpringBoot 会自动加载类路径下对应框架的配置文件，所以我们只需给类路径下放上每个日志框架己的配置文件即可，SpringBoot 就不会使用默认配置了。

框架	**命名方式**
Logback	logback-spring.xml, logback-spring.groovy, logback.xml or logback.groovy

原文链接：SpringBoot 是如何实现日志的?

Log4j2 log4j2-spring.xml or log4j2.xml

JDK (Java Util Logging) logging.properties

logback.xml：直接就被日志框架识别了。

logback-spring.xml：日志框架就不直接加载日志的配置项，由 SpringBoot 解析日志配置，可以用 SpringBoot 的高级 Profile 功能。

```
<springProfile name="staging">
   <!-- configuration to be enabled when the "staging" profile is active -->
    可以指定某段配置只在某个环境下生效
</springProfile>
```

例子 （以 Logback 框架为例）：

```
<appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
        <!--
        日志输出格式：
           %d表示日期时间，
           %thread表示线程名，
           %-5level：级别从左显示5个字符宽度
           %logger{50} 表示logger名字最长50个字符，否则按照句点分割。
           %msg：日志消息，
           %n是换行符
        -->
        <layout class="ch.qos.logback.classic.PatternLayout">
                <!--指定在 dev 环境下，控制台使用该格式输出日志-->
                <springProfile name="dev">
               <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} ----> [%thread] ---> %-5level %logger{
0} - %msg%n</pattern>
           </springProfile>
           <!--指定在非 dev 环境下，控制台使用该格式输出日志-->
           <springProfile name="!dev">
               <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} ==== [%thread] ==== %-5level %logg
r{50} - %msg%n</pattern>
           </springProfile>
        </layout>
    </appender>
```

如果使用 logback.xml 作为日志配置文件，而不是 logback-spring.xml，还要使用profile 功能，会以下错误：

```
no applicable action for [springProfile]
```

切换日志框架

了解了 SpringBoot 的底层日志依赖关系，我们就可以按照 slf4j 的日志适配图，进行相关的切换。

例如，切换成 slf4j+log4j ，可以这样做

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

```
    <exclusion>
      <artifactId>logback-classic</artifactId>
      <groupId>ch.qos.logback</groupId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
</dependency>
```

切换成 log4j2 ， 就可以这样做。

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

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

最后放上 logback-spring.xml 的详细配置，大家在自己项目可以参考配置。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
scan：当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true。
scanPeriod：设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒当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。默认值为false。
-->
<configuration scan="false" scanPeriod="60 seconds" debug="false">
    <!-- 定义日志的根目录 -->
    <property name="LOG_HOME" value="/app/log" />
    <!-- 定义日志文件名称 -->
    <property name="appName" value="nasus-springboot"></property>
    <!-- ch.qos.logback.core.ConsoleAppender 表示控制台输出 -->
    <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
        <!--
        日志输出格式：
            %d表示日期时间，
            %thread表示线程名，
            %-5level：级别从左显示5个字符宽度
            %logger{50} 表示logger名字最长50个字符，否则按照句点分割。
            %msg：日志消息，
```

```
            %n是换行符
        -->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <springProfile name="dev">
                <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} ----> [%thread] ---> %-5level %logger{
0} - %msg%n</pattern>
            </springProfile>
            <springProfile name="!dev">
                <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} ==== [%thread] ==== %-5level %logg
r{50} - %msg%n</pattern>
            </springProfile>
        </layout>
    </appender>

    <!-- 滚动记录文件，先将日志记录到指定文件，当符合某个条件时，将日志记录到其他文件 -->
    <appender name="appLogAppender" class="ch.qos.logback.core.rolling.RollingFileAppen
er">
        <!-- 指定日志文件的名称 -->
        <file>${LOG_HOME}/${appName}.log</file>
        <!--
        当发生滚动时，决定 RollingFileAppender 的行为，涉及文件移动和重命名
        TimeBasedRollingPolicy： 最常用的滚动策略，它根据时间来制定滚动策略，既负责滚动也负
出发滚动。
        -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!--
            滚动时产生的文件的存放位置及文件名称 %d{yyyy-MM-dd}：按天进行日志滚动
            %i：当文件大小超过maxFileSize时，按照i进行文件滚动
            -->
            <fileNamePattern>${LOG_HOME}/${appName}-%d{yyyy-MM-dd}-%i.log</fileNamePa
tern>
            <!--
            可选节点，控制保留的归档文件的最大数量，超出数量就删除旧文件。假设设置每天滚动，
            且maxHistory是365，则只保存最近365天的文件，删除之前的旧文件。注意，删除旧文件
，
            那些为了归档而创建的目录也会被删除。
            -->
            <MaxHistory>365</MaxHistory>
            <!--
            当日志文件超过maxFileSize指定的大小是，根据上面提到的%i进行日志文件滚动 注意此处
置SizeBasedTriggeringPolicy是无法实现按文件大小进行滚动的，必须配置timeBasedFileNamingA
dTriggeringPolicy
            -->
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAn
TimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
        <!-- 日志输出格式： -->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [ %thread ] - [ %-5level ] [ %logger{50} : %l
ne ] - %msg%n</pattern>
        </layout>
    </appender>
```

```
    <!--
        logger主要用于存放日志对象，也可以定义日志类型、级别
        name：表示匹配的logger类型前缀，也就是包的前半部分
        level：要记录的日志级别，包括 TRACE < DEBUG < INFO < WARN < ERROR
        additivity：作用在于children-logger是否使用 rootLogger配置的appender进行输出，
        false：表示只用当前logger的appender-ref，true：
        表示当前logger的appender-ref和rootLogger的appender-ref都有效
    -->
    <!-- hibernate logger -->
    <logger name="com.nasus" level="debug" />
    <!-- Spring framework logger -->
    <logger name="org.springframework" level="debug" additivity="false"></logger>



    <!--
    root 与 logger 是父子关系，没有特别定义则默认为root，任何一个类只会和一个logger对应，
    要么是定义的logger，要么是root，判断的关键在于找到这个logger，然后判断这个logger的app
nder和level。
    -->
    <root level="info">
        <appender-ref ref="stdout" />
        <appender-ref ref="appLogAppender" />
    </root>
</configuration>
```

参考文献

http://www.importnew.com/28494.html

https://www.cnblogs.com/lthIU/p/5860607.html

后语

如果本文对你哪怕有一丁点帮助，请帮忙点好看。你的好看是我坚持写作的动力。

另外，关注之后在发送 **1024** 可领取免费学习资料。

资料详情请看这篇旧文：Python、C++、Java、Linux、Go、前端、算法资料分享

一个优秀的废人，不止技术，还有财富

每周分享 Java 、理财干货喂饱你。

关注、转发、在看是一种美德！

长按可以订阅

原文链接：SpringBoot 是如何实现日志的?