JAVA主流日志梳理

JAVA主流日志梳理

引入

只要你是个程序员,你就不可能说不和日志打交道,无论是开发,学习和运维,日志已经是每个环节都必须的存在,然而,作为JAVA程序员,我们很不开心,因为JAVA中的日志体系实在是比较混乱,下面我将尝试用我已知的知识梳理一下这混乱的JAVA主流日志体系,wish me luck!

历史故事

我尝试用时间轴来解释为啥JAVA的日志体系会这么复杂吧:

Log4j - JDK1.3及以前

在这个时代的前辈们是没有日志的,所谓的日志打印也是依靠于`System.out.println()`和`System.err.println()`来实现的.在这种情况下,最好的结果就是消息等必要信息通过`System.out.println()`输出,异常的报错信息通过`System.err.println()`来输出,这种情况显示是不理想的,最简单的例子:现在我的程序已经上线运行了,我希望在我开发的环境中去除日志,这种情况下几乎做不到.

这个时代的JDK没有做的事情,Log4j来做了,它的设计理念一出世就征服了大家,它给出了当时最完美的定义`Logger`,`Appender`,`level`等等,它的思想到现在还被沿用着.

JUL - JDK1.4

SUN公司看着Log4j的崛起,坐不住了,JAVA是谁的地盘你知不知道,就这样出现了`java.util.logging`系列API,但是SUN爸爸的这个东西吧,更感觉是临时交任务出的,并没有细心打磨的感觉,不仅是没有Log4j功能完善,最重要的是没有提供类似Log4j中`Appenders`的`Handlers`,而是需要调用者自己去实现,更为重要的是就算我自己写,我也只有两个输出目的地`console`和 `file`.

试想一下,当时是你,你会选择哪个API来学习和使用?

JCL - 日志门面commons-logging的出现

要不说你大爷还是你大爷呢,面对市面上唯二的选择,还是有一部分人相信SUN公司会持续维护JUL的API的.在这种情况下,作为普通开发者,引入第三方的JAR依赖的时候我就不得不选择和我一样的日志API的依赖,这显然不合适,于是大部分人采用的是既配置JUL又配置Log4j的方式来完成程序编写的.

在这种情况下commons-logging出现了,严格来说它并不是一个日志框架,它所做的事情是如果你用JUL那么我就把你在JUL中需要打印的内容告诉Log4j让它来打印或者是反之把log4j的内容换成jul来输出.这样你就只需要一个了,你还可以自己选择用哪一个,是不是听起来很棒棒?

人无完人,commons-logging也不是完美的,它在对兼容JUL和log4j的配置这个问题上处理的并不是很好,且可能会导致出现NoClassDefFoundError的错误.对于这样的一个不定时的炸弹,很多人还是不能接受的.

SLF4j - 可能是最好的日志框架

在面对commons-logging的动态绑定可能出现NoClassDefFoundError错误的时候,作为log4j项目作者的Ceki Gülcü觉得是时候做出改变了,他发起了SLF4j项目,采用静态绑定的方式既兼容了市面上的log4j,jul和jcl又解决之前可能发生的NoClassDefFoundError.此外还提供了命名规则为XXX-over-slf4j.jar的桥接器接口,让你可以把其他非SLF4j的日志通过桥接器转换为它可以支持的日志实现.

logback - SLF4j的亲儿子

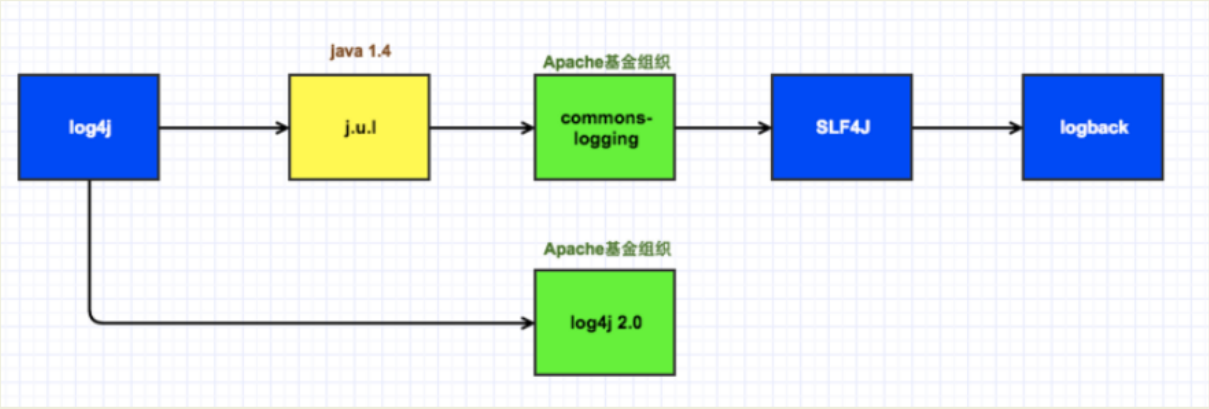
俗话说,送佛送到西,帮人帮到底,当然也可能是Ceki Gülcü觉得无聊,反正门面接口都搞完了,要不再搞一套实现?于是他又亲自操刀根据SLF4j的现状,写出来了logback,而且他还说了

> *我的logback性能最好了!*

log4j 2 - 开源社区的礼物

Ceki Gülcü在完成来了logback以后还惦记着之前搞的log4j,于是又去升级了一下大儿子,只不过这次不止是它一个人在战斗,在Apache基金会接纳了log4j 2以后,借助于开源社区的力量,它的强大似乎超越了logback,当然更高的性能和功能为它的配置也带来了更多的不便.

时间轴



我的选择

看了上面的那么多,现在对于JAVA主流日志为啥这么混乱大概有个了解了吧?

那么问题来了,你是系统架构者,面对这些技术,你会怎么选?为什么这个选?我的选择是SLF4j+logback+xxxx-over-slf4j.jar
说一下理由:

1. 日志门面选择SLF4j

- SLF4j彻底解决了NoClassDefFoundError的炸弹
- SLF4j支持参数化的日志输出
- SLF4j下的日志实现更换非常便捷

2. 日志实现选择logback

- logback相对于log4j性能接近有10倍的提升
- log4j 2虽然相对于log4j性能有接近18倍的提升,但是配置负责很多,且目前的业务场景下,不需要log4j 2的其他强大功能

3. 桥接器的加入

- 作为企业级应用程序的开发者,我们不可能不依赖于第三方框架和类库,加入桥接器会让我们的系统可以完整的输出依赖中的日志

实践参考

排除JCL等其他的日志框架

这里需要在引用到其他日志框架的MAVEN依赖中排除JCL等JAR包,我们使用exclusions标签来实现这个功能,举个栗子,我们使用了commons组件中的commons-beanutils组件,看名字也觉得它和commons-logging同属于Apache基金会,所以它使用commons-logging也是无可厚非的,但是我不想用啊,于是我的pom文件中,它的依赖就变成了这样:

```
<dependency>
    <groupId>commons-beanutils</groupId>
    <artifactId>commons-beanutils</artifactId>
    <version>${commons-beanutils.version}</version>
    <exclusions>
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

添加桥接器

在我们排除了其他的日志依赖以后就会出现一个问题,本来有日志的地方因为没有日志框架或实现导致日志出不来了,这个时候,我们需要桥接器来完成日志的桥接

常用的桥接器有:

- jul-to-slf4j.jar

- jcl-over-slf4j
- log4j-over-slf4j

logback的配置

到这里我们排除了其他日志,又通过桥接器把本来其他日志输出换成了logback,下面我们就需要配置logback本身了,下面我们来自slf4j+logback的配置及使用博文的配置文件:

```
<?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 子节点为 appender、logger、root
-->
<configuration scan="true" scanPeriod="60 seconds" debug="false">

    <!--用于区分不同应用程序的记录-->
    <contextName>appName</contextName>

    <!--日志文件所在目录，如果是tomcat，如下写法日志文件会在则为${TOMCAT_HOME}/bin/logs/目录下-->
    <property name="LOG_HOME" value="logs"/>

    <!--输出日志到控制台-->
    <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度 %logger输出日志的logger名 %msg：日志消息，%n是换行符 -->
            <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{36} : %msg%n</pattern>
            <!--解决乱码问题-->
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!--滚动文件 消息文件-->
    <appender name="infoFile" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- ThresholdFilter:临界值过滤器，过滤掉 TRACE 和 DEBUG 级别的日志 -->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>INFO</level>
        </filter>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!--输入的日志文件名称-->
            <fileNamePattern>${LOG_HOME}/log.%d{yyyy-MM-dd}.log</fileNamePattern>
            <!--保存最近30天的日志-->
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder>
```

```
        <charset>UTF-8</charset>
        <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{36} : %msg%n</pattern>
    </encoder>
</appender>

<!—滚动文件 异常文件—>
<appender name="errorFile" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>error</level>
    </filter>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${LOG_HOME}/error.%d{yyyy-MM-dd}.log</fileNamePattern>
        <maxHistory>30</maxHistory>
    </rollingPolicy>
    <encoder>
        <charset>UTF-8</charset>
        <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{36} : %msg%n</pattern>
    </encoder>
</appender>

<!—将日志输出到logstack—>
<!—<appender name="logstash" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
    <destination>47.93.173.81:7002</destination>
    <encoder class="net.logstash.logback.encoder.LogstashEncoder">
        <charset>UTF-8</charset>
    </encoder>
    <keepAliveDuration>5 minutes</keepAliveDuration>
</appender>—>

<!—这里如果是info，spring、mybatis等框架则不会输出：TRACE < DEBUG < INFO <  WARN < ERROR—>
<!—root是所有logger的祖先，均继承root，如果某一个自定义的logger没有指定level，就会寻找父logger看有没有指定级别，直到找到root。—>
<root level="debug">
    <appender-ref ref="stdout"/>
    <appender-ref ref="infoFile"/>
    <appender-ref ref="errorFile"/>
    <appender-ref ref="logstash"/>
</root>

<!—
为某个包单独配置logger

比如定时任务，写代码的包名为：com.seentao.task
步骤如下：
1、定义一个appender，取名为task（随意，只要下面logger引用就行了）
appender的配置按照需要即可
```

```
    2、定义一个logger:
    <logger name="com.seentao.task" level="DEBUG" additivity="false">
      <appender-ref ref="task" />
    </logger>
    注意：additivity必须设置为false，这样只会交给task这个appender，否则其他appender也会打印com.seentao.task里的log信息。

    3、这样，在com.seentao.task的logger就会是上面定义的logger了。
    private static Logger logger = LoggerFactory.getLogger(Class1.class);
    -->

</configuration>
```

maxwellyue这位兄弟的注释还是很完善的,我们一般使用的就是这两个,一是输出到控制台,而是输出到日志文件,上面都有,而且在向文件输出日志的时候,他还做了一个事,他把消息和异常的日志分开了,这种做法我很赞同,这对于那种任务调度或者是后台异步操作很多的系统是很有必要的,你只需要关注出问题的这部分就可以了,这个时候日志文件的价值才能体现出来.

关于日志文件中各个节点的具体说明和解析,因为我自己也不太了解,期望大家参考其他的博文.

日志等级了解

在看代码之前,我们先看一眼在logback中日志等级的划分,优先级由低到高排列

日志等级	作用范围
trace	最为详细的日志信息,一般是开发中加入,正式上线前消除的内容 也可以选择不去除,由日志文件的配置来进行过滤
debug	调试级别的日志,比如"查询到了三条数据"之类的日志信息
info	信息级别的日志,比如某个计算任务开始的时候打一句日志
warn	当需要给出告警的时候 例如根据一个id去表中查询数据的时候并没有查询到数据 这种消息应该被重视
error	我一般发生异常的时候会采用这个级别的日志 但是其实应该是遇到那种程序不能继续执行的异常才会使用这个级别的日志 因为有的时候异常也是正常的业务逻辑

从描述中可以看到,日志登记越高,越需要引起管理人员的注意,好在物依稀为贵,等级越高的日志越少才是正常的现象.

如何在代码中打印日志

说了那么多,终于要来看代码了,其实代码很简单的:

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class LogTest {

    private static Logger logger = LoggerFactory.getLogger(CustomerControllerTest.class);

    @Test
    public void test() {
        logger.trace("trace日志");
        logger.debug("debug日志");
        logger.info("消息日志");
        logger.warn("告警日志");
        logger.error("报错了", new RuntimeException());

    }
}
```

这里一定要注意不要引错了包哦!正常来说现在的项目里其他的日志框架及实现应该不存在才对.

特别注意

前面我们提到了桥接器,我们用的都是把其他的日志桥接到SLF4j上,那么相应的就有把SLF4j的日志桥接到其他日志框架上的桥接器,我们要注意不要把两者都引入其中,这样会导致日志的循环出不来的.

参考文章(无排名)

- 那些年我们用过的日志框架
- java日志简介
- slf4j+logback的配置及使用
- logback的使用和logback.xml详解

分类: 个人总结

紫川弘
关注 - 0
粉丝 - 0
+加关注

« 上一篇: JAVA 8 日期工具类