# 日志组件slf4j介绍及配置详解

## 1 基本介绍

每一个Java程序员都知道日志对于任何一个Java应用程序尤其是服务端程序是至关重要的，而很多程序员也已经熟悉各种不同的日志库，如java.util.logging、Apache log4j、logback。但如果你还不知道SLF4J（Simple logging facade for Java）的话，那么是时候在你的项目中学习使用SLF4J了。

SLF4J不同于其他日志类库，与其它日志类库有很大的不同。SLF4J(Simple logging Facade for Java)不是一个真正的日志实现，而是一个抽象层（ abstraction layer），它允许你在后台使用任意一个日志类库。如果是在编写供内外部都可以使用的API或者通用类库，那么你真不会希望使用你类库的客户端必须使用你选择的日志类库。

如果一个项目已经使用了log4j，而你加载了一个类库，比方说 Apache Active MQ——它依赖于于另外一个日志类库logback，那么你就需要把它也加载进去。但如果Apache Active MQ使用了SLF4J，你可以继续使用你的日志类库而无需忍受加载和维护一个新的日志框架的痛苦。

总的来说，SLF4J使你的代码独立于任意一个特定的日志API，这是对于API开发者的很好的思想。虽然抽象日志类库的思想已经不是新鲜的事物，而且Apache commons logging也已经在使用这种思想了，但SLF4J正迅速成为Java世界的日志标准。让我们再看几个使用SLF4J而不是log4j、logback或者java.util.logging的理由。

## 2 SLF4J对比Log4J，logback和java.util.Logging的优势

正如我之前说的，在你的代码中使用SLF4J写日志语句的主要出发点是使得你的程序独立于任何特定的日志类库，依赖于特定类库可能需要使用不同于你已有的配置，并且导致更多维护的麻烦。除此之外，还有一个SLF4J API的特性是使得我坚持使用SLF4J而抛弃我长期间钟爱的Log4j的理由，是被称为占位符(place holder)，在代码中表示为“{}”的特性。占位符是一个非常类似于在String的format()方法中的%s，因为它会在运行时被某个提供的实际字符串所替换。这不仅降低了你代码中字符串连接次数，而且还节省了新建的String对象。通过使用SLF4J，你可以在运行时延迟字符串的建立，这意味着只有需要的String对象才被建立。而如果你已经使用log4j，那么你已经对于在if条件中使用debug语句这种变通方案十分熟悉了，但SLF4J的占位符就比这个好用得多。

这是你在Log4j中使用的方案，但这并不有趣，而且降低了代码可读性，因为它增加了不必要的繁琐重复代码(boiler-plate code)：

if (logger.isDebugEnabled()) {

logger.debug("Processing trade with id: " + id + " symbol: " + symbol);

}

另一方面，如果你使用SLF4J的话，你可以得到更简洁格式的结果，就像以下展示的一样：

logger.debug("Processing trade with id: {} and symbol : {} ", id, symbol);

在SLF4J，我们不需要字符串连接而且不会导致暂时不需要的字符串消耗。取而代之，我们在一个以占位符和参数传递实际值构成的模板格式下写日志信息。你可能会在想万一我有很多个参数怎么办？嗯，那么你可以选择使用变量参数版本的日志方法或者以Object数组传递。这是一个相当方便和高效方法的打日志方法。记住，在生产最终日志信息字符串之前，这个方法会检查一个特定的日志级别是不是打开了，这不仅降低了内存消耗而且预先降低了CPU去处理字符串连接命令的时间。这里是使用SLF4J日志方法的代码，来自于slf4j-log4j12-1.6.1.jar中的Log4j的适配器类Log4jLoggerAdapter。

public void debug(String format, Object arg1, Object arg2) {

if (logger.isDebugEnabled()) {

FormattingTuple ft = MessageFormatter.format(format, arg1, arg2);

logger.log(FQCN, Level.DEBUG, ft.getMessage(), ft.getThrowable());

}

}

同时，我们应该知道打日志是对应用程序性能有着很大影响，在生产环节上我们建议只进行必要的日志记录。

# SLF4J简介与使用(整合log4j)

## 一、概念

SLF4J的全称是Simple Logging Facade for Java，即简单日志门面。SLF4J并不是具体的日志框架，而是作为一个简单门面服务于各类日志框架，如java.util.logging, logback和log4j。

SLF4J提供了统一的记录日志的接口，对不同日志系统的具体实现进行了抽象化，只要按照其提供的方法记录即可，最终日志的格式、记录级别、输出方式等通过绑定具体的日志系统来实现。

使用SLF4J的好处在于，你只需要按统一的方式写记录日志的代码，如：

public class LoggerTest {

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

public static void main(String[] args) {

logger.info("Current Time: {}", System.currentTimeMillis());

}

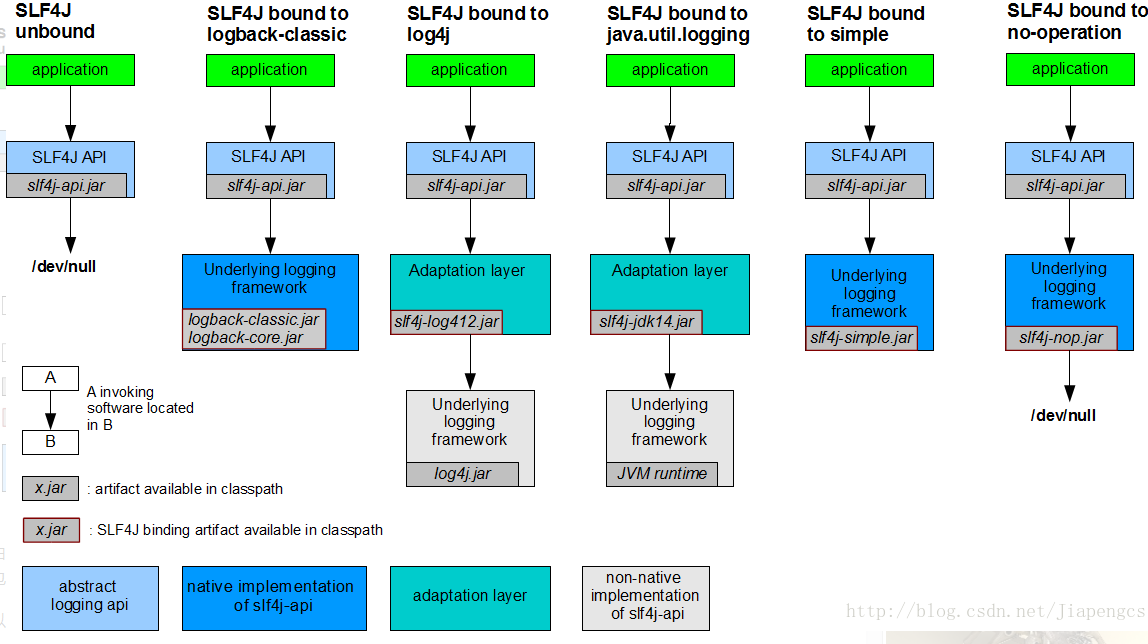
}

SLF4J支持*{}*作为占位符，等价于C语言中的*%s*，而不必再进行字符串的拼接，效率有显著的提升（见后面运行结果）。

而无需关心日志是通过哪个日志系统，以什么风格输出的。因为它们取决于部署项目时绑定的日志系统。   
例如，在项目中使用了SLF4J记录日志，并且绑定了log4j，则日志会以log4j的风格输出；后期需要改为以logback的风格输出日志，只需要将log4j替换成logback即可，不用修改项目中的代码。

## 二、依赖

SLF4J绑定各类日志框架的原理图：



由上图可知，使用SLF4J依赖于slf4j-api-1.8.0-alpha2.jar，部署时还依赖于要绑定的日志系统的jar包和相应的适配器jar包。

以绑定log4j为例，需要导入以下包：

* slf4j-api-1.8.0-alpha2.jar
* log4j-1.2.17.jar
* slf4j-log4j12-1.8.0-alpha2.jar

如果使用Maven，则只需添加适配器jar包依赖即可：

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.8.0-alpha2</version>

</dependency>

## 三、使用示例

这里以SLF4J + log4j为例。

### 1.在pom.xml中添加依赖（或者手动导入上述3个jar包）：

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.8.0-alpha2</version>

</dependency>

### 2.配置log4j

在类路径下创建log4j.properties配置文件，这样log4j会自动加载配置文件。

# rootLogger参数分别为：根Logger级别，输出器stdout，输出器log

log4j.rootLogger = info,stdout,log

# 输出信息到控制台

log4j.appender.stdout = org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout = org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern = %d [%-5p] %l %rms: %m%n

# 输出DEBUG级别以上的日志到D://logs/debug.log

log4j.appender.log = org.apache.log4j.DailyRollingFileAppender

log4j.appender.log.DatePattern = '.'yyyy-MM-dd

log4j.appender.log.File = D://debug.log

log4j.appender.log.Encoding = UTF-8

#log4j.appender.log.Threshold = INFO

log4j.appender.log.layout = org.apache.log4j.PatternLayout

log4j.appender.log.layout.ConversionPattern = %d [%-5p] (%c.%t): %m%n

将log4j.properties放在类路径下是最简单的做法，当然也可以通过PropertyConfigurator在代码中加载或者通过web.xml加载。

### 3.测试代码

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggerTest {

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

public static void main(String[] args) {

logger.info("Current Time: {}", System.currentTimeMillis());

logger.info("Current Time: " + System.currentTimeMillis());

logger.info("Current Time: {}", System.currentTimeMillis());

logger.trace("trace log");

logger.warn("warn log");

logger.debug("debug log");

logger.info("info log");

logger.error("error log");

}

}

### 4.运行结果

2017-06-16 23:11:05,490 [INFO ] com.jiapengcs.demos.slf4j.LoggerTest.main(LoggerTest.java:14) 0ms: Current Time: 1497625865488

2017-06-16 23:11:05,493 [INFO ] com.jiapengcs.demos.slf4j.LoggerTest.main(LoggerTest.java:15) 3ms: Current Time: 1497625865493

2017-06-16 23:11:05,493 [INFO ] com.jiapengcs.demos.slf4j.LoggerTest.main(LoggerTest.java:16) 3ms: Current Time: 1497625865493

2017-06-16 23:11:05,495 [WARN ] com.jiapengcs.demos.slf4j.LoggerTest.main(LoggerTest.java:18) 5ms: warn log

2017-06-16 23:11:05,495 [INFO ] com.jiapengcs.demos.slf4j.LoggerTest.main(LoggerTest.java:20) 5ms: info log

2017-06-16 23:11:05,495 [ERROR] com.jiapengcs.demos.slf4j.LoggerTest.main(LoggerTest.java:21) 5ms: error log

通常输出日志开销非常大，从上述结果可见，SLF4J通过{}作为占位符的方式输出字符串，相比字符串拼接的方式，效率有显著的提升。

### 5.更换日志系统

看到这里，你可能会有疑问：既然都用了log4j，为什么还要用SLF4J来写记录日志的代码呢，不是多此一举吗？

答案是否定的。假设我们不再需要log4j，而是希望改为使用java自带logging记录日志，我们需要做的仅仅是将pom.xml的依赖项slf4j-log4j12改为slf4j-jdk14即可，无需对上述测试代码做任何修改。

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-jdk14</artifactId>

<version>1.8.0-alpha2</version>

</dependency>

是的，就是这么简单。再次运行测试代码：

六月 16, 2017 10:32:30 下午 com.jiapengcs.demos.slf4j.LoggerTest main

信息: Current Time: 1497623550843

六月 16, 2017 10:32:30 下午 com.jiapengcs.demos.slf4j.LoggerTest main

信息: Current Time: 1497623550874

六月 16, 2017 10:32:30 下午 com.jiapengcs.demos.slf4j.LoggerTest main

信息: Current Time: 1497623550875

六月 16, 2017 10:32:30 下午 com.jiapengcs.demos.slf4j.LoggerTest main

警告: warn log

六月 16, 2017 10:32:30 下午 com.jiapengcs.demos.slf4j.LoggerTest main

信息: info log

六月 16, 2017 10:32:30 下午 com.jiapengcs.demos.slf4j.LoggerTest main

严重: error log

我们发现，此时日志已经变为以logging的方式输出。

## 四、总结

SLF4J的使用非常简单，甚至连官网上都说鉴于它太轻量，文档篇幅不长。

Given the small size of SLF4J, its documentation is not very lengthy.

在《阿里巴巴Java开发手册(正式版)》中，**日志规约**一项第一条就强制要求使用SLF4J：

1.【强制】应用中不可直接使用日志系统（Log4j、Logback）中的API，而应依赖使用日志框架SLF4J中的API，使用门面模式的日志框架，有利于维护和各个类的日志处理方式统一。

所以从现在开始使用SLF4J吧！

* [官方文档](https://www.slf4j.org/manual.html)
* [本文示例源代码](https://github.com/jiapengcs/demos/tree/master/src/main)

# 使用Slf4j集成Log4j2构建项目日志系统的完美解决方案

## 一、背景

　　最近因为公司项目性能需要，我们考虑把以前基于的log4j的日志系统重构成基于Slf4j和log4j2的日志系统，因为，使用slf4j可以很好的保证我们的日志系统具有良好的兼容性，兼容当前常见几种日志系统，而使用log4j2而不是log4j是因为Log4j 1.x 在高并发情况下出现死锁导致cpu使用率异常飙升，而Log4j2.0基于LMAX Disruptor的异步日志在多线程环境下性能会远远优于Log4j 1.x和logback（官方数据是10倍以上）。

　　关于slf4j的原理以及优点，请参见我的另一篇博客：[slf4j介绍以及实现原理窥探](http://www.cnblogs.com/hafiz/p/5486858.html)

## 二、搭建步骤

　　1.依赖管理

　　　　1).删除项目中存在的Log4j1.x所必须的log4j和slf4j-log4j12等依赖。

　　　　　　可以到项目的根目录，执行：mvn dependency:tree > tree.log，之后使用 cat tree.log | grep log4j命令进行查找。

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</exclusion>

<exclusion>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

</exclusion>

</exclusions>

  2).添加以下slf4j和log4j2的依赖.

1 <!-- slf4j核心包-->

2 <dependency>

3 <groupId>org.slf4j</groupId>

4 <artifactId>slf4j-api</artifactId>

5 <version>1.7.13</version>

6 </dependency>

7 <dependency>

8 <groupId>org.slf4j</groupId>

9 <artifactId>jcl-over-slf4j</artifactId>

10 <version>1.7.13</version>

11 <scope>runtime</scope>

12 </dependency>

13

14 <!--核心log4j2jar包-->

15 <dependency>

16 <groupId>org.apache.logging.log4j</groupId>

17 <artifactId>log4j-api</artifactId>

18 <version>2.4.1</version>

19 </dependency>

20 <dependency>

21 <groupId>org.apache.logging.log4j</groupId>

22 <artifactId>log4j-core</artifactId>

23 <version>2.4.1</version>

24 </dependency>

25 <!--用于与slf4j保持桥接-->

26 <dependency>

27 <groupId>org.apache.logging.log4j</groupId>

28 <artifactId>log4j-slf4j-impl</artifactId>

29 <version>2.4.1</version>

30 </dependency>

31 <!--web工程需要包含log4j-web，非web工程不需要-->

32 <dependency>

33 <groupId>org.apache.logging.log4j</groupId>

34 <artifactId>log4j-web</artifactId>

35 <version>2.4.1</version>

36 <scope>runtime</scope>

37 </dependency>

38

39 <!--需要使用log4j2的AsyncLogger需要包含disruptor-->

40 <dependency>

41 <groupId>com.lmax</groupId>

42 <artifactId>disruptor</artifactId>

43 <version>3.2.0</version>

44 </dependency>

2.web.xml中设置log4j2的监听器和过滤器（servlet3.0及以上版本不需要该步操作）

1 <!--对于log4j2，Servlet2.5以前的版本需要-->

2 <listener>

3 <listener-class>org.apache.logging.log4j.web.Log4jServletContextListener</listener-class>

4 </listener>

5 <filter>

6 <filter-name>log4jServletFilter</filter-name>

7 <filter-class>org.apache.logging.log4j.web.Log4jServletFilter</filter-class>

8 </filter>

9 <filter-mapping>

10 <filter-name>log4jServletFilter</filter-name>

11 <url-pattern>/\*</url-pattern>

12 <dispatcher>REQUEST</dispatcher>

13 <dispatcher>FORWARD</dispatcher>

14 <dispatcher>INCLUDE</dispatcher>

15 <dispatcher>ERROR</dispatcher>

16 </filter-mapping>

注意：log4j2不再支持properties文件了，只支持xml，json或是yaml，不指定位置的情况下默认在src/main/resources下查找。

　　　　　如果需要自定义位置，需要在上面的web.xml中添加以下代码

1 <context-param>

2 <param-name>log4jConfiguration</param-name>

3 <param-value>/WEB-INF/classes/log4j2.xml</param-value>

4 </context-param>

3.log4j2.xml

1 <?xml version="1.0" encoding="UTF-8"?>

2

3 <Configuration status="off" monitorInterval="1800">

4

5 <properties>

6 <property name="LOG\_HOME">/opt/logs/hafiz/log4j2Demo/logs</property>

7 <property name="ERROR\_LOG\_FILE\_NAME">error</property>

8 </properties>

9

10

11 <Appenders>

12 <Console name="Console" target="SYSTEM\_OUT">

13 <PatternLayout pattern="%d %-5p (%F:%L) - %m%n" />

14 </Console>

15

16 <RollingRandomAccessFile name="ErrorLog"

17 fileName="${LOG\_HOME}/${ERROR\_LOG\_FILE\_NAME}.log"

18 filePattern="${LOG\_HOME}/${ERROR\_LOG\_FILE\_NAME}.log.%d{yyyy-MM-dd}.gz">

19 <PatternLayout

20 pattern="%d %-5p (%F:%L) - %m%n"/>

21 <Policies>

22 <TimeBasedTriggeringPolicy/>

23 <SizeBasedTriggeringPolicy size="100 MB"/>

24 </Policies>

25 <DefaultRolloverStrategy max="20"/>

26 </RollingRandomAccessFile>

27

28 </Appenders>

29

30 <Loggers>

31 <!-- 3rdparty Loggers -->

32 <logger name="org.springframework.core" level="info">

33 </logger>

34 <logger name="org.springframework.beans" level="info">

35 </logger>

36 <logger name="org.springframework.context" level="info">

37 </logger>

38 <logger name="org.springframework.web" level="info">

39 </logger>

40

41 <logger name="com.hafiz.www.controller" level="error" includeLocation="true" additivity="false">

42 <appender-ref ref="ErrorLog"/>

43 <appender-ref ref="Console"/>

44 </logger>

45

46 <root level="info" includeLocation="true">

47 <appender-ref ref="Console"/>

48 </root>

49 </Loggers>

50 </Configuration>

4.测试类UserController.java

1 package com.hafiz.www.controller;

2

3 import com.hafiz.www.po.UserEntity;

4 import com.hafiz.www.service.UserService;

5 import org.slf4j.Logger;

6 import org.slf4j.LoggerFactory;

7 import org.springframework.beans.factory.annotation.Autowired;

8 import org.springframework.stereotype.Controller;

9 import org.springframework.web.bind.annotation.RequestMapping;

10 import org.springframework.web.bind.annotation.RequestMethod;

11 import org.springframework.web.bind.annotation.ResponseBody;

12

13 import java.util.ArrayList;

14 import java.util.List;

15

16 /\*\*

17 \* Desc:用户信息控制器

18 \* Created by hafiz.zhang on 2016/8/27.

19 \*/

20 @Controller

21 @RequestMapping("/user")

22 public class UserController {

23

24 private static final Logger logger = LoggerFactory.getLogger(UserController.class);

25

26 @Autowired

27 private UserService userService;

28

29 @RequestMapping(value = "/all", method = RequestMethod.GET)

30 @ResponseBody

31 public List<UserEntity> getAllUsers(){

32 logger.info("[GET] /user/all getAllUsers");

33 List<UserEntity> list = userService.getAllUsers();

34 logger.debug("This is log of level of debug");

35 logger.trace("log4j2 Demo");

36 logger.error("哎呀，出错啦~");

37 return list;

38 }

39 }

注意：在JVM启动参数中增加 -DLog4jContextSelector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector 开启异步日志.

小问题：

　　因为我之前是log4j 1.x，并且我在web.xml里面指定了log4j.properties文件的位置，第一次运行项目，忘了修改其值为log4j2.xml，出现了"Context [] startup failed due to previous errors"的错误，导致项目启动不起来，web.xml中修改为如下方式以后就正常了。

1 <context-param>

2 <param-name>log4jConfiguration</param-name>

3 <param-value>/WEB-INF/classes/log4j2.xml</param-value>

4 </context-param>