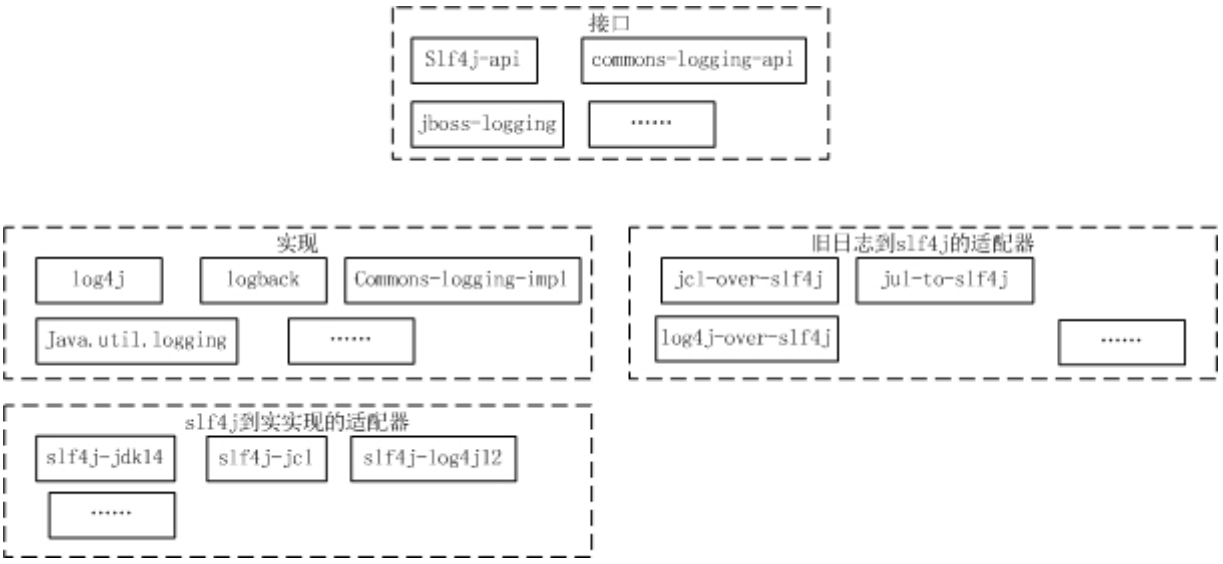
log4j+logback+slf4j+commons-logging的关系与调试 - 上校

背景

由于现在开源框架日益丰富，好多开源框架使用的日志组件不尽相同。存在着在一个项目中，不同的版本，不同的框架共存。导致日志输出异常混乱。虽然也不至于对系统造成致命伤害，但是明显可以看出，架构不够精良，追求极致略有不足。

其中有一些标准通用接口，标准实现，各种桥接器的存在，下面就让笔者树立一下这些框架之间的关系。



从上图中，我们可以看到4部分。

接口：将所有日志实现适配到了一起，用统一的接口调用。

实现：目前主流的日志实现

旧日志到slf4j的适配器：如果使用了slf4j，但是只想用一种实现，想把log4j的日志体系也从logback输出，这个是很有用的。

slf4j到实现的适配器：如果想制定slf4j的具体实现，需要这些包。

slf4J与旧日志框架的关系

slf4j等于commons-logging，是各种日志实现的通用入口，会根据classpath中存在下面哪一个Jar来决定具体的日志实现库。

logback-classic(默认的logback实现)

slf4j-jcl.jar(apache commons logging)

slf4j-logj12.jar(log4j 1.2.4)

slf4j-jdk14(java.util.logging)

将所有使用旧式日志API的第三方类库或旧代码的日志调用转到slfj

jcl-over-slf4j.jar/jcl104-over-slf4j：apache commons logging 1.1.1/1.0.4，直接替换即可。

log4j-over-slf4j.jar：log4j，直接替换即可。

jul-to-slf4j：jdk logging，需要在程序开始时调用SLF4JBridgeHandler.install()来注册listener 参考JulOverSlf4jProcessor，可在applicationContext.xml中定义该bean来实现初始化。注意原有的log4j.properites将失效，logback网站上提供转换器，支持从log4j.properties 转换到logback.xml 。

优势：转移到logback的理由

slf4j支持参数化的logger.error("帐号ID：{}不存在", userId);告别了if(logger.isDebugEnable()) 时代。

另外logback的整体性能比log4j也较佳，hibernate等项目已经采用了slf4j:"某些关键操作，比如判定是否记录一条日志语句的操作，其性能得到了显著的提高。这个操作在LOGBack中需要3纳秒，而在Log4J中则需要30纳秒。 LOGBack创建记录器（logger）的速度也更快：13毫秒，而在Log4J中需要23毫秒。更重要的是，它获取已存在的记录器只需94纳秒，而Log4J需要2234纳秒，时间减少到了1/23。"

slf4j和logback的使用

1.如果日志的参数超过3个，需要写成

Object[] params = {newVal, below, above}; logger.debug("Value {} was inserted between {} and {}.", params);

commons-logging 和slf4j的代码比较：commons-logging 示例代码：

```
1. import org.apache.commons.logging.Log;
2. import org.apache.commons.logging.LogFactory;
3. public class TestLog {
4.     Log log = LogFactory.getLog(TestLog.class);
5.     public void print() {
6.         if (log.isDebugEnabled()) {
7.             log.debug(sql);
8.             log.debug("My name is " + name + "， I am " + age + " years old.");
9.         }
10.    }
11. }
```

slf4j的示例代码：

```
1. import org.slf4j.Logger;
2. import org.slf4j.LoggerFactory;
3. public class TestLogBySlf4J {
4.     Logger logger = LoggerFactory.getLogger(TestLogBySlf4J.class);
5.     public void print() {
6.         logger.debug(sql);
7.         logger.debug("My name is {}， I am {} years old.", name, age);
8.     }
9. }
```

2.因为内部已优化，作者认为slf4j的logger不需要定义为static。
3.可设置缓存后批量写日志文件(但服务器如果重启，可能会丢失未写到磁盘的记录)
4.MDC，用Filter，将当前用户名等业务信息放入MDC中，在日志format定义中即可使用该变量。
5.JMS Appender用于告警, DB Appender用于业务日志等

生产环境情况以及优雅方案

我厂的项目由于使用了众多的开源架构，所以导致项目中的日志体系非常混乱。经常出现日志包冲突的情况。例如：commons-logging-1.0.4，commons-logging-1.1.3，log4j，logback，jboss-logging，java.util.logging.....不同的版本，不同的实现。之前笔者至少要配置log4j，logback，commons-logging三个配置文件，才能完成日志的输出。研究了日志体系以后，我厂的maven的pom.xml如下

```
1.     <!-- log -->
2. <dependency>
3.     <groupId>org.slf4j</groupId>
4.     <artifactId>slf4j-api</artifactId>
5.     <version>${org.slf4j-version}</version>
6. </dependency>
7. <dependency>
8.     <groupId>org.slf4j</groupId>
9.     <artifactId>jcl-over-slf4j</artifactId>
10.    <version>${org.slf4j-version}</version>
11. </dependency>
```

```
12. <dependency>
13.     <groupId>org.slf4j</groupId>
14.     <artifactId>log4j-over-slf4j</artifactId>
15.     <version>${org.slf4j-version}</version>
16. </dependency>
17. <dependency>
18.     <groupId>org.slf4j</groupId>
19.     <artifactId>jul-to-slf4j</artifactId>
20.     <version>${org.slf4j-version}</version>
21. </dependency>
22. <dependency>
23.     <groupId>org.jboss.logging</groupId>
24.     <artifactId>jboss-logging</artifactId>
25.     <version>3.1.4.GA</version>
26. </dependency>
27. <!--
28. <dependency>
29.     <groupId>commons-logging</groupId>
30.     <artifactId>commons-logging</artifactId>
31.     <version>1.1.3</version>
32. </dependency>
33. <dependency>
34.     <groupId>log4j</groupId>
35.     <artifactId>log4j</artifactId>
36.     <version>1.2.17</version>
37. </dependency>
38.  -->
```

加了几个重要的实现适配器，log4j-over-slf4j，log4j-over-slf4j，jul-to-slf4j。然后jboss-logging部分也进行了统一的版本控制。同时删除及exclude掉所有log4j，commons-logging的各个版本。适配器和具体日志实现，不能共存，否则适配器不生效。这样的话，我们只有logback配置文件即可，因为log4j的输出已经委托给了slf4j（通过log4j-over-slf4j），而slf4j的默认实现是logback。

其中JUL需要额外执行一行初始化代码

```
1. SLF4JBridgeHandler.install();// jul to slf4j
```

常见错误调试

【错误1】只是一个典型错误，主要是错误的第二行红色部分

log4j:WARN No appenders could be found for logger (com.mchange.v2.log.MLog).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

出现此错误是由于没有log4j配置文件导致。在classpath下添加log4j.xml，并做一些配置即可。配置完以后，启动输出

2014-02-25 09:30:31:743[INFO][com.mchange.v2.log.MLog.<clinit>(MLog.java:80)] - MLog clients using log4j logging.
2014-02-25 09:30:31:780[INFO] [com.mchange.v2.c3p0.C3P0Registry.banner(C3P0Registry.java:204)] - Initializing c3p0-0.9.1.2 [built 21-May-2007 15:04:56; debug? true; trace: 10]

即表示回复正常

。例如我的配置文件如下

log4j.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
3. <log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
4.     <appender name="console" class="org.apache.log4j.ConsoleAppender">
5.         <layout class="org.apache.log4j.PatternLayout">
6.             <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss:SSS} [%-5p][%l] - %m%n" />
7.         </layout>
8.     </appender>
9.     <logger name="org.springframework" additivity="false">
10.         <level value="info" />
11.         <appender-ref ref="console" />
12.     </logger>
13.     <root>
14.         <level value="info" />
15.         <appender-ref ref="console" />
```

```
16.     </root>
17. </log4j:configuration>
```