JAVA应用开发日志解决方案

Author:	Maolin.Chen
Creation Date:	May 30,2010
Last Updated:	June 23, 2010
Document Ref:	GMCC/HODF/[XXX]
Version:	1.0

文档控制

更改记录

日期	作者	版本	更改参考
2010-5-30	Maolin.Chen	1.0	无前版本

内容目录

1. 简介

在应用系统的维护过程中，应用程序运行时所记录的日志对维护工作起着至关重要的作用，应用日志常常可以让我们方便快速的定位到故障或BUG的所在。就JAVA应用开发而言，目前的流行的日志解决方案有LOG4J、LOG4J+JCL、LOG4J+SLF4J等，本文就以实用为原则主要介绍这三种日志框架方案的应用。

1.1. 下载所需组件

1. LOG4J 1.2.16

   http://www.apache.org/dyn/closer.cgi/logging/log4j/1.2.16/apache-log4j-1.2.16.zip

2. Jakarta Commons Logging(JCL)
   http://apache.etoak.com/commons/logging/binaries/commons-logging-1.1.1-bin.zip

3. SLF4J
   http://www.slf4j.org/

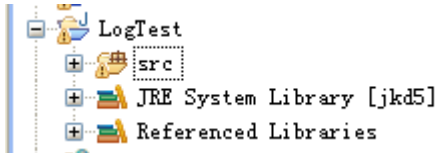
4. 本文使用MyEclipse6.5作为集成开发环境，并且假设读者熟悉MyEclipse6.5的基本操作。

2. JAVA日志之屠龙刀LOG4J

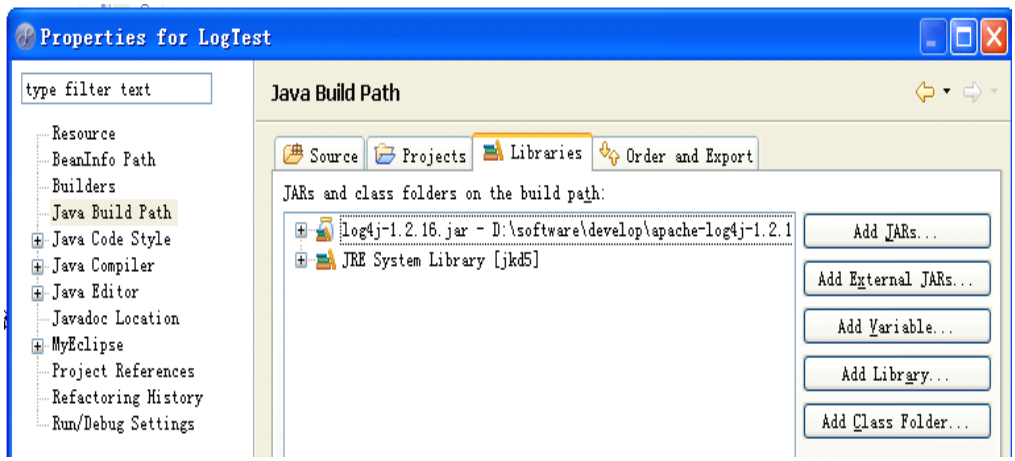
无论是开源项目还是商业项目，LOG4J在事实上都成为了大多数程序员的选择，因为其配置简单，使用方便，功能强大，能生成各种格式或各种类型的日志，并且可以将日志写到文件、数据库、网络等目的地，下面我们就来介绍一下如何使用这一框架。

2.1. 创建项目

1. 打开MyEclipse6.5，创建一个名为**LogTest**的**Java Project**。

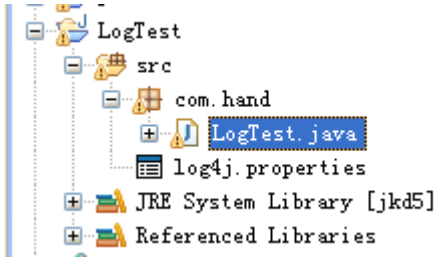


2. 右键**LogTest**项目，选择**properties→Java Build Path→Libraries→Add External Jar**,选择你下载的LOG4J JAR包，添加进本项目。



2.2. 创建测试类

1. 新建一个类，类名叫**LogTest.java**,包名 **com.hand**



2. 在LogTest类里写如下内容

```
package com.hand;
import org.apache.log4j.Logger;
public class LogTest {
   public static void main(String []args){
//获取Logger实例，参数为本类
      Logger logger=Logger.getLogger(LogTest.class);
      logger.debug("debuging");//输出一段DEBUG信息
      logger.info("info..."); //输出一段INFO信息
```

```
        logger.error("error...");//输出一段ERROR错误信息
    }
}
```

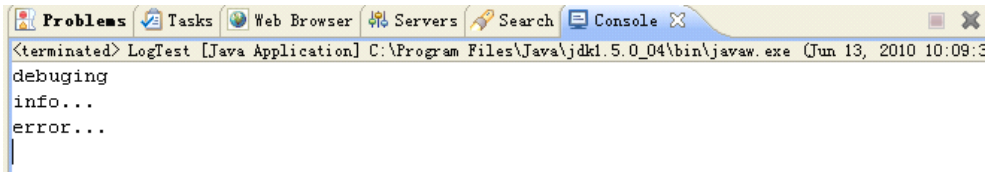
就这样，我们就完成日志输出的代码编写部分。接下来，我们只需要对LOG4J做一点点的配置，日志就能够呈现到我们面前了。

2.3. 配置方式

1. 在SRC目录下创建一个文件，名为**log4j.properties** ，内容如下：

```
#此属性指定日志等级等于或低于INFO的日志信息输出到名为stdout的目的地
log4j.rootCategory=DEBUG, stdout
#此属性执行stdout这个输出目的地类型为控制台
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
#此属性指定输出日志的布局类，这里采用LOG4J默认的布局类
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
```

具备以上三个属性，我们就能够将日志输出到控制台了，打开LogTest.java ，右键该类，点击 Run As →Java Application，在控制台我们就可以看到如下输出，

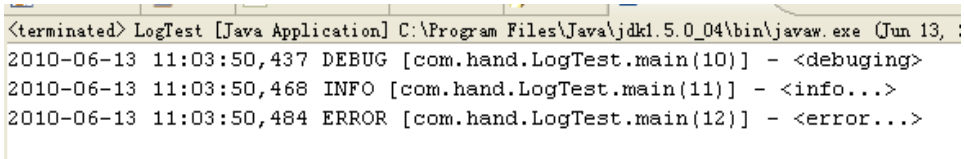


有的童鞋就会说啦，这个样的输出我就用SOP（不是杜拉拉里的SOP，此处为System.out.println）就可以办到，还花那么大力气来搞什么LOG4J，下面我们要玩的东西就会告诉你这个世界上有很多SOP办不到的事情。

2. 上面输出的信息仅仅包含信息本身，可实际情况下，我们更多时候需要一同输出信息产生的时间，信息是在哪个地方（类）产生的等，要实现这样的效果，我们只需在log4j.properties文件里添加一行：

```
log4j.appender.stdout.layout.ConversionPattern= %d %p [%C.%M(%L)] -
<%m>%n
```

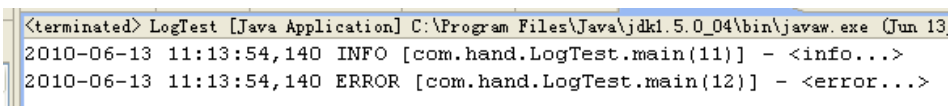
重新运行LogTest.java,



详细的信息便呈现出来了。

3. 问题又来了，假设我们程序已经开发完了，需要部署到正式环境了，这个时候如果我们日志输出级别还是保持在DEBUG级别的话，那么我们的日志信息量以及日志文件个头将会变得很大很大，不便于我们查阅和维护，因此我们在部署到正式环境的时候需要将日志级别调低一点。修改log4j.properties

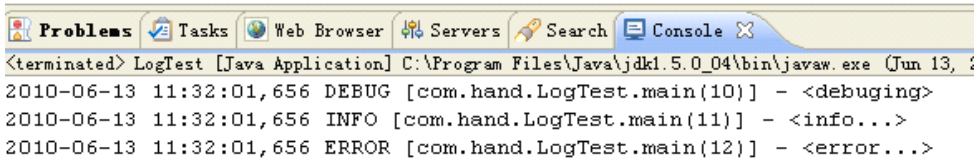
```
log4j.rootCategory=INFO, stdout
```



这样我们的日志里就不显示DEBUG信息了，具体各级别的含义请参考附件。

4. 这样一来，我们又产生问题了，假如某个类或者包非常重要，我想让这个类或者包输出DEBUG信息，怎么办呢，LOG4J为我们提供了解决方法，只需在log4j.properties文件里添加一行：

```
#log4j.logger加上包名或类名
log4j.logger.com.hand.LogTest=DEBUG
```



看，DEBUG信息又回来了。

5. 到目前为止，我们都只是把日志输出到控制台，可是大多数情况我们需要把日志输出到日志文件，这要怎么配置呢，很简单，修改log4j.properties

```
#在前面的基础上，修改log4j.rootCategory属性添加一个目的地，起个名儿叫logfile
log4j.rootCategory=INFO, stdout,logfile
#
log4j.appender.logfile=org.apache.log4j.FileAppender
log4j.appender.logfile.File=LogTest.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern= %d %p [%C.%M(%L)] -
<%m>%n
```

这样一来日志就会自动输出到LogTest.log文件

6. 如果老是向同一个日志文件写日志，那么一段时间后这个日志文件会变得非常大，不便于我们我们维护，因此我们希望按照日志文件大小，比如日志文件到达100M后就自动生成一个新的日志文件。LOG4J也提供了简单的配置方式，修改log4j.properties

```
log4j.appender.logfile= org.apache.log4j.RollingFileAppender
log4j.appender.logfile.File=LogTest.log
log4j.appender.logfile.Append=true
log4j.appender.logfile.MaxFileSize=100MB
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern= %d %p [%C.%M(%L)] -
<%m>%n
```

有时候我们又希望按照日期，每天生成一个日志文件：

```
log4j.appender.logfile= org.apache.log4j.DailyRollingFileAppender
log4j.appender.logfile.File=LogTest.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern= %d %p [%C.%M(%L)] -
<%m>%n
```

常用的配置我们介绍得差不多了，其他诸如输出到数据库、SOCKET等方式可参阅网上相关文章，这里就不再赘述了。

LOG4J功能如此之强悍，霸气，视天下之兵器如朽木，因此我将之称为JAVA日志之屠龙刀，也算是名副其实。

3. JAVA日志之倚天剑JCL

既然LOG4J如此之强大，为什么我们还要提JCL呢？正所谓萝卜白菜各有所爱，不一定所有的开发人员或者客户都喜欢用LOG4J，因此有时候我们不能在程序直接使用LOG4J，尤其是在类库的开发中，这个时候JCL就可以粉墨登场了，它本身并不是一个很优秀的日志解决方案，但是它擅长于利用其他优秀日志框架来为自己服务，可以说，JCL使开发人员不用关心日志服务最终由什么框架实现。

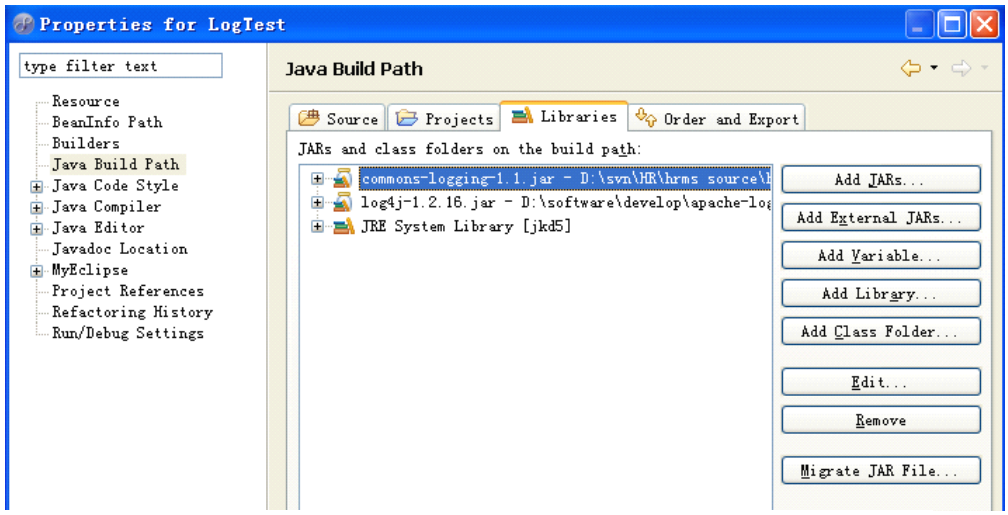
官方给出的说明是：

Jakarta Commons Logging (JCL)提供的是一个日志(Log)接口(interface)，同时兼顾轻量级和不依赖于具体的日志实现工具。它提供给中间件/日志工具开发者一个简单的日志操作抽象，允许程序开发人员使用不同的具体日志实现工具。

因此，在使用JCL时，往往都要和某个日志框架搭配使用，这里我们还是选择最经典的方案，JCL+LOG4J。

3.1. 配置项目

我们还是利用上文创建的**LogTest**项目，参考上文添加LOG4J包的方法，给项目添加JCL包



3.2. 修改测试类

修改LogTest.java

```
package com.hand;
import org.apache.commons.logging.*;
/**
 * @author Administrator
 *
 */
public class LogTest {
    private static final Log logger = LogFactory.getLog(LogTest.class);
    public static void main(String []args){
        //Logger logger=Logger.getLogger(LogTest.class);

        logger.debug("debuging");
        logger.info("info...");
        logger.error("error...");
    }
}
```

这里需要说明的是JCL最大的一个特性：

COMMON-LOGGIN的API中，Log(基本记录器)和LogFactory(负责创建Log实例)是两个基类。该API直接提供对下列底层日志记录工具的支持：Jdk14Logger，Log4JLogger，LogKitLogger，NoOpLogger （直接丢弃所有日志信息），还有一个SimpleLog。

有必要详细说明一下调用LogFactory.getLog()时发生的事情。**调用该函数会启动一个发现过程，即找出必需的底层日志记录功能的实现，具体的发现过程在下面列出：**

⑴ Commons的Logging首先在CLASSPATH中查找**commons-logging.properties**文件。这个属性文件至少定义org.apache.commons.logging.Log属性，它的值应该是上述任意Log接口实现的完整限定名称。如果找到 org.apache.commons.logging.Log属相，则使用该属相对应的日志组件。结束发现过程。

⑵ 如果上面的步骤失败（文件不存在或属相不存在），Commons的Logging接着检查**系统属性 org.apache.commons.logging.Log**。如果找到org.apache.commons.logging.Log系统属性，则使用该系统属性对应的日志组件。结束发现过程。

⑶ 如果找不到org.apache.commons.logging.Log系统属性，Logging接着在CLASSPATH中寻找**log4j的类**。如果找到了，Logging就假定应用要使用的是log4j。不过这时log4j本身的属性仍要通过log4j.properties文件正确配置。结束发现过程。

⑷ 如果上述查找均不能找到适当的Logging API，但应用程序正运行在JRE 1.4或更高版本上，则默认使用**JRE 1.4的日志**记录功能。结束发现过程。

⑸ 最后，如果上述操作都失败（JRE 版本也低于1.4），则应用将使用**内建的SimpleLog**。SimpleLog把所有日志信息直接输出到System.err。结束发现过程。

本例中，JCL在第（1）步和第（2）步均查找失败，在第（3）步是找到了LOG4J的JAR包，于是JCL就使用LOG4J作为日志处理方案。

如果后来的开发人员不想使用LOG4J，可直接添加相应的JAR包或类，然后在**commons-logging.properties**文件中指定即可，而不用去修改程序代码。

4. JAVA日志之乾坤大挪移SLF4J

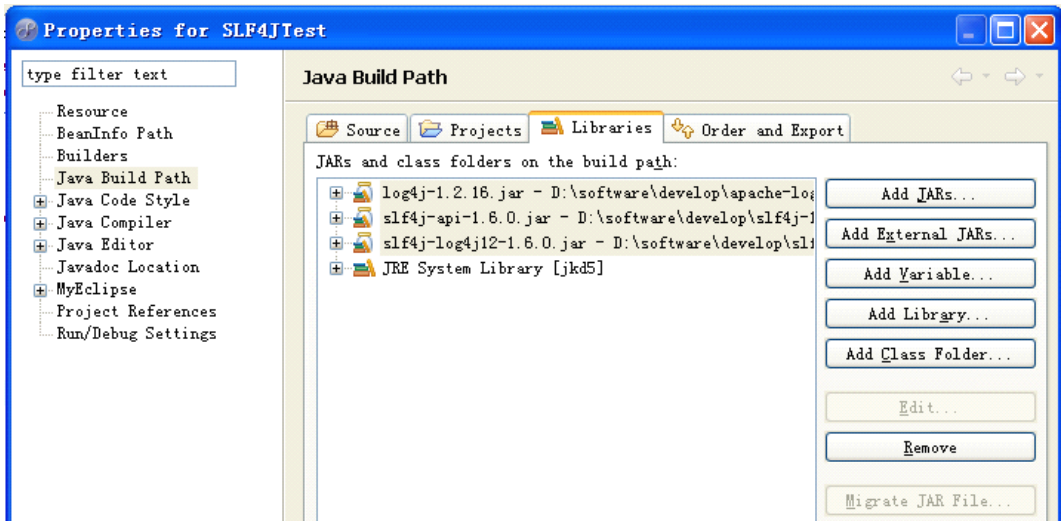
Simple Logging Facade for Java (SLF4J)的性质和功能跟JCL差不多，都是类似于接口的，但是之所以我们会提使用到SLF4J主要是因为SLF4J具备以下优点：SLF4J 受类加载器的影响较小，不易产生内存溢出的问题，性能得到了改善，更主要是顺应了潮流的发展--可方便部署到 OSGI 环境中。正是基于这些优点，过去大的如 SSH 三雄(Spring、Struts、Hibernate)，还有 WAS 应用服务器，小的就不计其数以前用的通用日志框架都清一色的 JCL，日志实现会选用 Log4j，而现在 Hibernate、Tapesty、DbUnit、Jetty V6 等纷纷变节，都采用了 SLF4J。

本文就介绍SLF4J+LOG4J这种比较受推崇的方案。

4.1. 创建项目

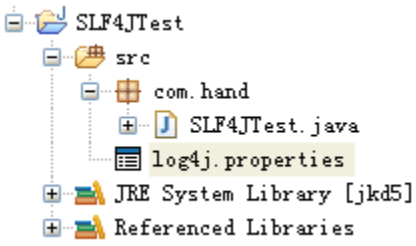
1. 打开MyEclipse6.5，创建一个名为**SLF4JTest**的**Java Project**。

2. 右键**SLF4JTest**项目，选择**properties→Java Build Path→Libraries→Add External Jar**,选择你下载的**slf4j-api-1.6.0.jar** 基本API包，因为我们要与LOG4J结合使用，因此我们还需要添加SLF4J提供的LOG4J支持包**slf4j-log4j12-1.6.0.jar**，以及LOG4J自己的原生包**log4j-1.2.16.jar**。



4.2. 创建测试类

1. 新建一个类，类名叫**SLF4JTest.java** ,包名 **com.hand**



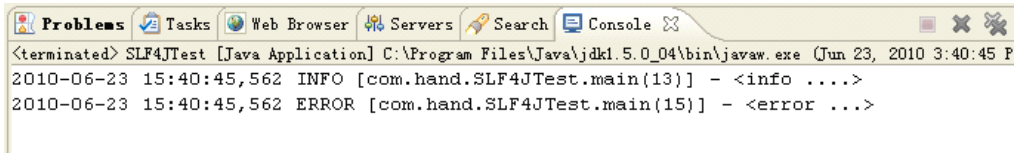
2. 在**SLF4JTest**类里写如下内容

```
package com.hand;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
public class SLF4JTest {

   public static void main(String[] args) {
       Logger logger =
LoggerFactory.getLogger(SLF4JTest.class);
       logger.info("info ....");
       logger.debug("debug ...");
       logger.error("error ...");

   }

}
```

接下来需要配置log4j.properties，为了简便起见，我们直接拷贝前文LogTest项目的log4j.properties文件到SRC目录下。

2. 运行本类，得到输出：



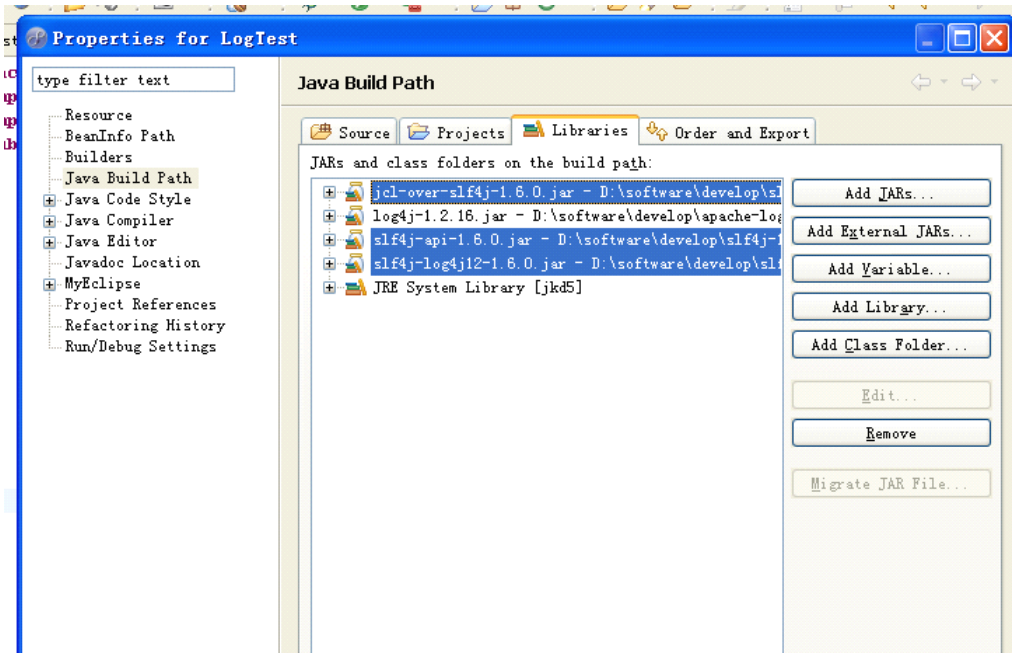
一个SLF4J的基本测试就完成了。

4.3. 遗留系统解决方案

如同前文我们LogTest项目一样，假设我的程序已经使用了JCL，现在我发现其实SLF4J更适合我的系统环境，我想把日志“接口”换成SLF4J，怎么办？SLF4J为我们提供了一个很方便的方案。

以LogTest项目为例：

右键**LogTest**项目，选择**properties→Java Build Path→Libraries**，删除commons-logging-1.1.jar，添加SLF4J提供的jcl-over-slf4j-1.6.0.jar，slf4j-api-1.6.0.jar,slf4j-log4j12-1.6.0.jar 三个包。



这里需要说明的是，jcl-over-slf4j-1.6.0.jar包是用来替换JCL的原生包的，它提供JCL一模一样的API，但是底层却是桥接到SLF4J去了。对于程序来说，这完全是透明的，我们不需要修改任何程序，仍然可以按照JCL的API去编写程序，非常的方便。

任何疑问和建议请联系

chenmaolin88@163.com

5. 未结与已结问题

未结问题

序号	问题	解决方案	负责人	目标日期	实际日期

已结问题

序号	问题	解决方案	负责人	目标日期	实际日期