在 WebLogic Server 8.1 中处理 LOG

------- 赖歆
laix@telthink.com
2004 年 8 月 26 日

目录

1 前言

原本是打算写 WebService 的，结果这段时间做 TAMIS 的传送模块时，把 WebService 的思想融合到其中，于是就不再将 WebService 独立出来写。

相对我以前大量图片的风格来说这篇文章的文字描述可能会多一些，因为涉及了许多概念，大家不要看睡着了。好的，现在来看看 Log 吧。

在应用程序中添加日志记录总的来说基于三个目的：

监视代码中变量的变化情况，周期性的记录到文件中供其他应用进行统计分析工作；

跟踪代码运行时轨迹，作为日后审计的依据；

担当集成开发环境中的调试器的作用，向文件或控制台打印代码的调试信息。

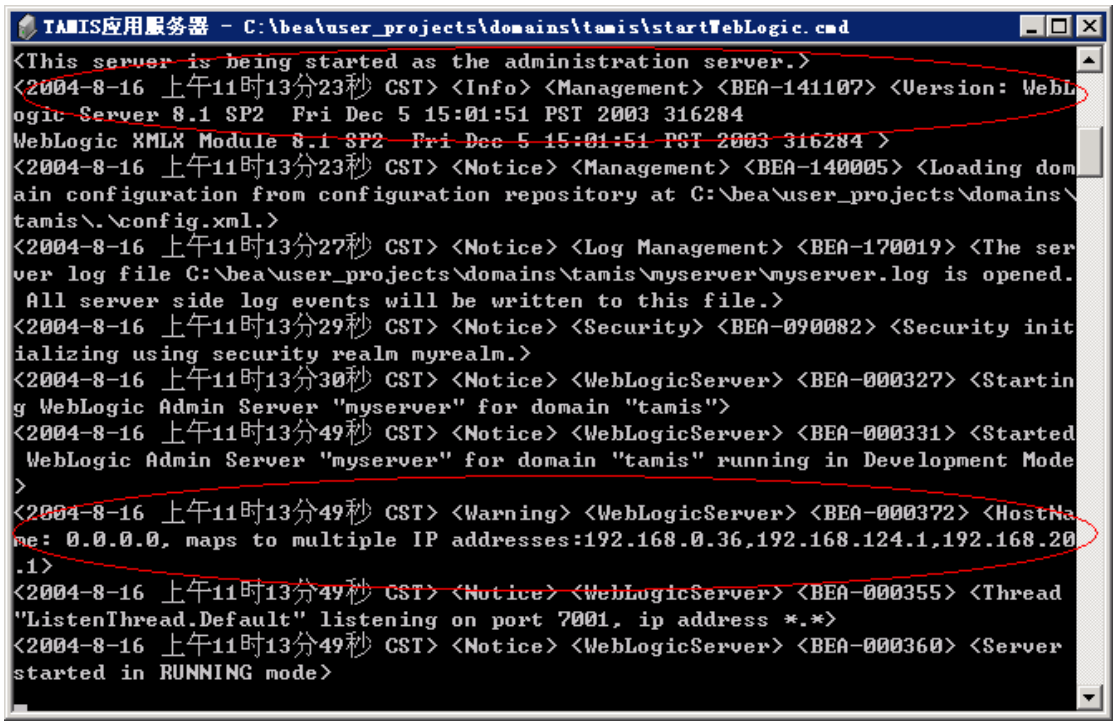
Log 在一个系统中的位置是十分重要的。为什么呢？因为有些系统错误是很容易重现的，比如逻辑上的失误；但有一些错误是很隐蔽的，比如新手最常遇到的 String 的 Null 问题。如果没有完善的日志，我们将很难从日益庞大的代码中快速的定位错误。最普通的做法就是在代码中嵌入许多的打印语句，这些打印语句可以输出到控制台或文件中，比较好的做法就是构造一个日志操作类来封装此类操作，而不是让一系列的打印语句充斥了代码的主体。

那如何建立一个相对完善的日志系统，而尽量不产生系统开销，以及不增加开发人员的工作呢？下面的文章将结合我们现在使用的 WebLogic Server 8.1 及正在开发的项目 TAMIS 对这个问题进行讨论。

2 分析日志系统

2.1 Weblogic Server 8.1 中使用的日志

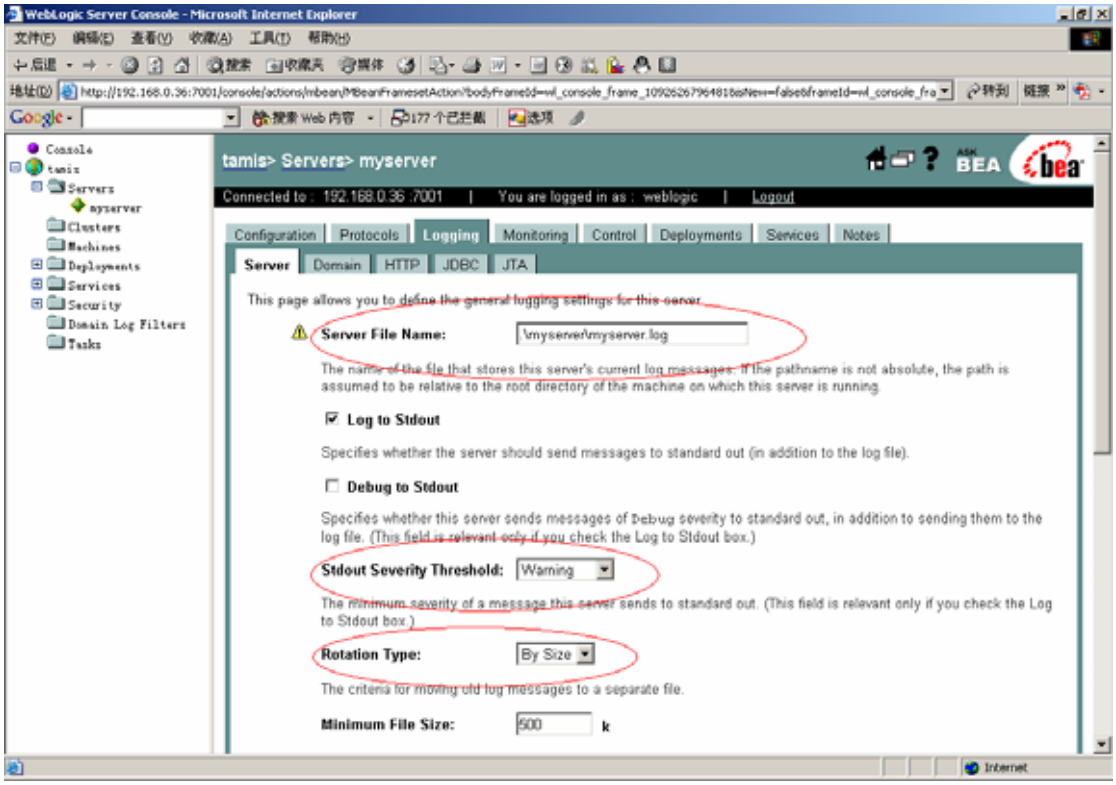
首先，在 Window 下运行 WebLogic Server 8.1，如果不使用重定向方式的话，你将看到一个如下图的 Console 日志窗口

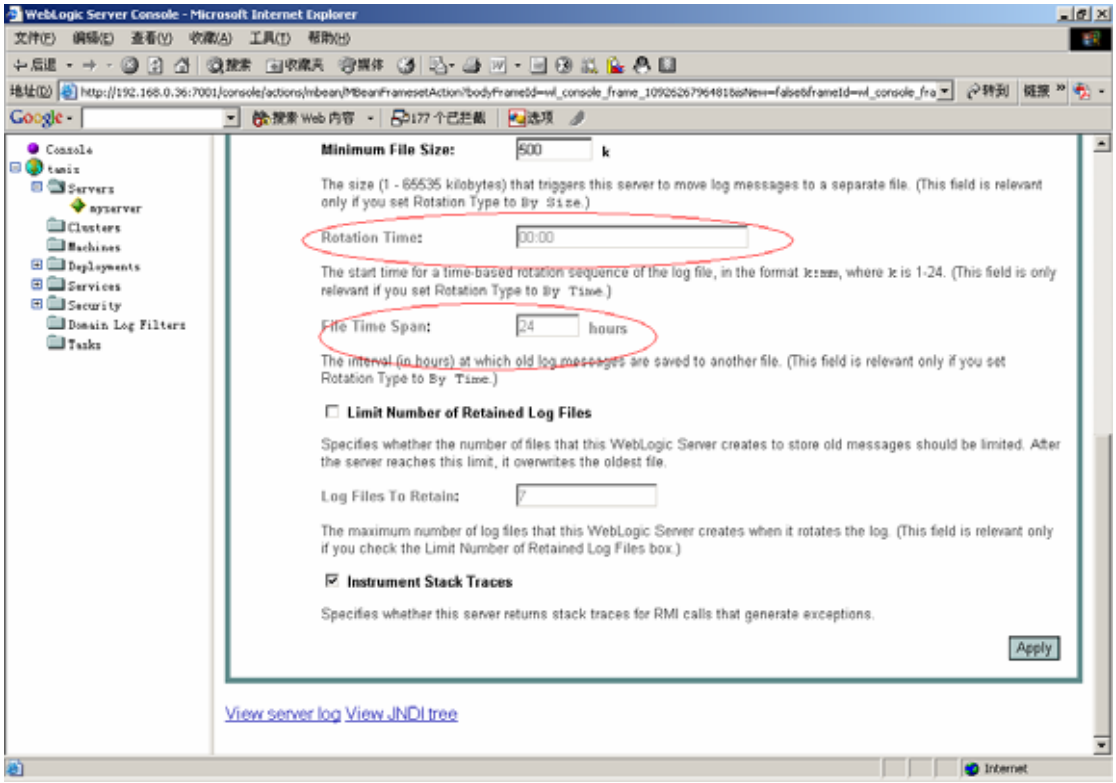


从这个图中，我们可以看到 WebLogic Server 8.1 默认输出的日志至少包含了以下几个元素：

日期时间戳、日志等级、日志所属模块、日志正文

再看 WebLogic Server 8.1 的 Console 模块中对 LOG 的配置，如下图：





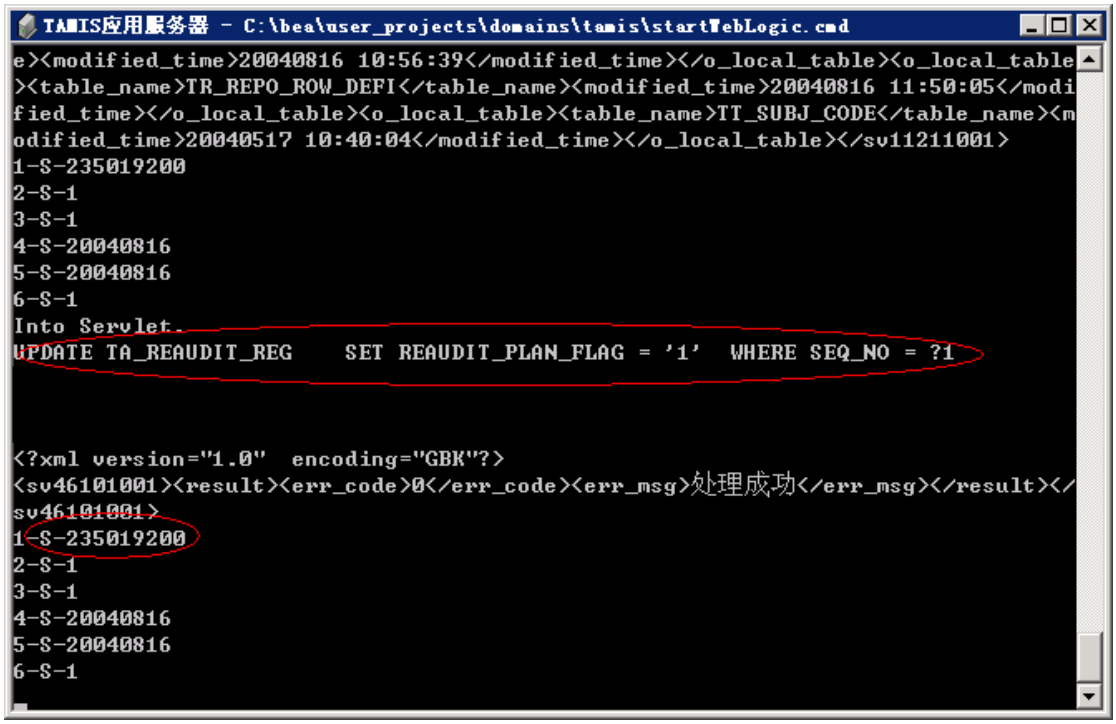
从图中我们看到WebLogic Server 8.1对日志输出用一个Servlet提供了配置功能。在这里我们可以定制日志的输出文件名、日志最低的输出级别、日志文件的建立规则（如：文件大小超过 X 就新建、按时间新建），最后通过 Apply 可以动态的对日志输出进行重分配。

2.2 TAMIS 目前使用的日志

目前开发的 TAMIS 系统是我们部门首个全面使用 J2EE 技术的项目，除了在稽查业务上的整合提升外，其在部门技术转型的意义更为突出。这里我们不对这个项目的其它技术进行分析，仅仅对它的日志进行分析。

在项目初期，我们对日志处理的技术积累并不充分，充其量仅仅留下一个接口而已，这里我们把它做为反面教材

如下图：



我们通过一个公用的 wwcom 中的静态函数进行 println 操作进行记录，输出到 Console 日志窗口中。可以发现，我们记录了 SQL 语句、部分变量值，当然如果有 exception 的话也会出现在这儿。

如果说时间戳、日志等级、模块信息等可以通过生成字符串来完成，但是问题远不至于此。我们输出到的日志，不符合 WebLogic Server 8.1 的日志标准（它有一个自已的日志类），因此不会记录到 WebLogicr Server 的日志文件中，而仅仅是通过这个窗口显示，当然我们可以自已用重定向“ > ”来将整个 console 日志输出到一个文件中，但是就没办法做到分成多个文件，更谈不上什么动态配置。

时间戳、所属模块和日志正文不需多做说明。引进日志等级可以用于动态的

过滤日志，在开发阶段以及试运行阶段，我们将需要 DEBUG 的信息，每个变量的值。而系统正式使用时，这样做将给系统的运行效率带来影响，以前我们在用 TUXEDO 做 TAX-21 时，是通过编译开关来控制，可以一但我们临时又需要查看 DEBUG 的信息呢？这时通过动态配置日志等级将我们带来极大的方便。**上面没有提到的是**，作为一个 WEB 应用来说，同一个业务同一时间可能会有多个线程进同时调用，因此还需要输出用于区别线程间的信息，否则，无法完整跟踪一个业务的运行全过程。显然我们的日志系统必须解决这个问题。

重新对我们的 wwcom 中的 log 函数进行改造迫在眉捷。因为同时将有多个进、线程对日志文件进行写入操作，与我们在前台只有单用户不同的是如果不加限制地写入将会造成文件的死锁。传统的思路是用 synchronization 申明为同步方式，这样大家排队写入，我做过测试，这个方法在解决问题的同时，带来了另一个比较严重的问题，因为是同步，在排队过程中给系统的效率带来了挺大的影响。通过查找资料，发现现在比较流行通过缓冲的方式进行异步写入，这样就不会对系统带来太大的影响。解决完这个问题后，其它问题的解决就没有什么值得注意的了。

操刀写了一段程序后，发现我所要做的工作，工作量实在是太大了，远远超出了我的工作安排。于是开始寻找合适的 Log 包。最先发现的是 JDK1.4 包中带的 java.util.logging，接着发现 Jakarta 开源项目 org.apache.log4j。对两个进行初步了解后，发现原先 org.apache.log4j 要做为 JDK1.4 的 LOG 标准，结果 logging 已进入了开发阶段，SUN 的项目负责不愿意再改写一些主要的 API。对两者进行简单的比较，对我们的系统有影响的有以下三点：

1. Log4J 更加成熟，从 1999 年 10 月开始至今已经有 3 年的时间，并且已经在许多项目中有着成熟的应用。而 JDK 中的 logging 包是在 1.4 之后才引入的，并且不能运行于 JDK 1.3 之前的版本。Log4J 则可以良好地运行于 JDK 1.1 之后的所有版本。（**从一定程度上解决了兼容性问题**）
2. Log4J 已经被移植到多种环境下，包括 log4c（C）、log4cpp（C++）、log4perl（Perl）、log4net（.net）等。在这些环境下，可以感受到几乎一致的配置和使用方式。这是 JDK 中的 logging API 所不能比拟的。 （**哈哈，要是我们当时 TAX-21 就用它就好了**）
3. Log4J 还具有更加强力的格式化系统，可以使记录输出时实现简单的模式。但是，它不会增加类而导致格式化工具的扩展。众多的附加程序和处理器使得 Log4J 数据包成为一个绝佳的选择，所有你所需要的都可能加以实现。 （**这个功能真的是很强大，后面会提到**）

随便说一下，再继续研究发现 WebLogic Server 8.1 的日志系统，发现它的日志包也建立是在 log4j 的基础上。同时发现它还大量使用了 jakarta 的 cli、lang、net 等等开源包，一方面说明了开源项目在商业应用，另一方面也看出我们强调的可重用组件开发的在商业应用所占的比例也是越来越大了（大到整个 J2EE 框架标准，小到这些包）。

忽然想到以前听谁开玩笑说的，BEA 的开发实力不能称上一流的软件公司，也就是欧洲的宏智。哈哈，仅仅是个玩笑，下面让我们来一起研究 Log4j 吧

3 Log4j 的概念

左边这个就是 Log4j 的标志，颜色搭配的不错吧，第一次看见 JAVA 的灰色咖啡杯变成这样的。

Log4j 是 Apache 的一个开放源代码项目，通过使用 Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI 组件、甚至是套接口服务器、NT 的事件记录器、UNIX Syslog 守护进程等；我们也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

此外，通过 Log4j 其他语言接口，您可以在 C、C++、.Net、PL/SQL 程序中使用 Log4j，其语法和用法与在 Java 程序中一样，使得多语言分布式系统得到一个统一一致的日志组件模块。而且，通过使用各种第三方扩展，您可以很方便地将 Log4j 集成到 J2EE、JINI 甚至是 SNMP 应用中。

log4j 有三种主要的组件：

记录器、存放器、布局

3.1 记录器（Logger）

日志记录器(Logger)是日志处理的核心组件。它有两个重要的属性：

关系、级别

3.1.1 关系

log4j 允许程序员定义多个记录器，每个记录器有自己的名字，记录器之间通过名字来表明隶属关系（或家族关系）。列如，记录器 a.b,与记录器 a.b.c 之间是父子关系，而记录器 a 与 a.b.c 之间是祖先与后代的关系，父子关系是祖先与后代关系的特例。通过这种关系，可以描述不同记录器之间的逻辑关系。

有一个记录器叫根记录器，它永远存在，且不能通过名字检索或引用，可以通过 Logger.getRootLogger() 方法取得它，而一般记录器通过

Logger.getLogger(String name)方法。

要注意两点：

1、当一个记录器不存在时，getLogger()将实例化一个记录器并返回引用；当一个记录器实例化后，再一次用相同的名字调用 getLogger()则只是返回对它的引用而不是再进行实例化。

2、与自然界中祖先先于后代出现不同，一个记录器的祖先可以比后代记录出现的晚，但会自动根据名字之间的关系建立这种家族关系。

有很多方法可以创建一个日志记录器（Logger），root 是自动构建的，我们在使用时只需构建各个子孙记录器，比较常用的用法，就是根据类名实例化一个静态的全局日志记录器:

```
static Logger mylogger = Logger.getLogger(test.class.getName());
```

然后就可以用以下语句记录日志，注意 Logger 并不关注日志的流向，而仅仅只是记录日志：

```
mylogger.info("select * from pc_area_code")
```

这里的 info 方法就涉及了记录器另一个重要属性 **级别**

3.1.2 级别

log4j 具有 5 种级别(Level) (不包括自定义级别 Level) ，从小到大为：

static Level DEBUG

DEBUG Level 指出细粒度信息事件对调试应用程序是非常有帮助的。

static Level INFO

INFO level 表明 消息在粗粒度级别上突出强调应用程序的运行过程。

static Level WARN

WARN level 表明会出现潜在错误的情形。

static Level ERROR

ERROR level 指出虽然发生错误事件，但仍然不影响系统的继续运行。

static Level FATAL

FATAL level 指出每个严重的错误事件将会导致应用程序的退出。

另外，还有两个可用的特别的日志记录级别:

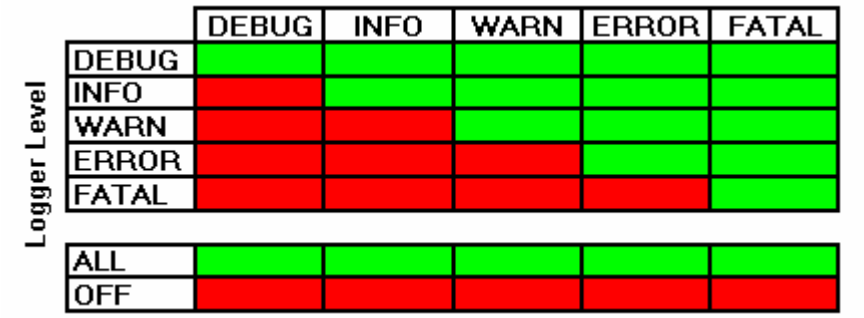
static Level ALL

ALL Level 是最低等级的，用于打开所有日志记录。

static Level OFF

OFF Level 是最高等级的，用于关闭所有日志记录。

如下表所示：



不同的记录器可以赋以不同的级别，如果某个成员没有被明确值，就自动继承最近的一个有级别长辈的级别值。根记录器总有级别值，它的级别是在初始化时建立的。在子孙记录器都不进行级别设置时，就等于都从根记录器中继承，这个恰恰也是我们常用的方式。这里要注意：

不要混淆关系和级别的关系，就象一个家庭中，成员间存在辈份关系，但不同的成员的身高可能不一样，且身高与辈份无关。也就是如果手动赋值，可能根记录器只能记录 WARN 级以上的日志，而子孙记录器可以记录所有的日志。

级别设置语句如下：

```
mylogger.setLevel((Level)Level.DEBUG);
```

3.2 存放器（Appender）

Appender 控制着将日志输出到哪里去，目前它可以支持 console, files, GUI components, remote socket servers, JMS, NT Event Loggers, remote UNIX Syslog daemons 等。当前我们最常用的应该是 console 和 files。

一个记录器可以有多个存放器，可以通过方法 addAppender 来增加存放器。每个记录器有一个继承开关,其开关决定记录器是/否继承其父记录器的存放器，注意，如果继承则只继承其父记录器，而不考虑更远的祖先的情况。在这里我们

一般还是采用和记录器构建一样的策略，都不手动赋值，统一从 root 中继承。

```
mylogger.addAppender(appender);
```

3.2.1 ConsoleAppender

使用用户指定的布局(layout) 输出日志事件到 System.out 或者 System.err。

```
ConsoleAppender appender = new ConsoleAppender(new PatternLayout());
```

默认的目标是 System.out。

3.2.2 FileAppender

把日志事件写入一个文件

```
FileAppender appender = new FileAppender(Layout layout, String filename)
```

实例化一个 FileAppender 并且打开变量"filename"指定的文件。

或者

```
FileAppender appender = new FileAppender(Layout layout, String filename, boolean append)
```

实例化一个 FileAppender 并且打开变量"filename"指定的文件。这个构造函数还可以选择是否对指定的文件进行追加的方式输出。如果没有指定值，那么默认的方式就是追加。

3.2.3 DailyRollingFileAppender

这个也是文件的一种，如果把日志统一写到一个文件中，时间长了，文件就越来越大会给我们带来不便。这时就应当考虑使用 DailyRollingFileAppender 了，它把 Log 信息输出到按照日期来区分的文件中。可以每天产生一个 log 文件，，每个 log 文件只记录当天的 log 信息。

```
DailyRollingFileAppender appender =
new DailyRollingFileAppender(org.apache.log4j.Layout layout,
                             java.lang.String filename,
                             java.lang.String datePattern)
```

我们可以通过最后一个 String 参数 dataPattern 来指定产生 log 的规则。常用的有以下几个：

'.'yyyy-MM	每个月初新建
'.'yyyy-ww	每周初新建，周初是周一还是周日由本地系统决定

'.'yyyy-MM-dd	每天午夜新建
'.'yyyy-MM-dd-HH	每小时新建
'.'yyyy-MM-dd-HH-mm	每分钟新建

3.3 布局（Layout）

布局负责格式化输出的 log 信息。Appender 必须使用一个与之相关联的 Layout，这样它才能知道怎样格式化它的输出。

当前，log4j 具有三种类型的 Layout:

1、HTMLLayout 格式化日志输出为 HTML 表格。

2、PatternLayout 根据指定的 转换模式格式化日志输出，或者如果没有指定任何转换模式，就使用默认的转换模式**（这种就像 C 语言里的 printf，就是一个格式化打印，也是我们常用的）**。

3、SimpleLayout 以一种非常简单的方式格式化日志输出，它打印级别 Level，然后跟着一个破折号“ -“ ，最后才是日志消息。

下表列出常用的 PatternLayout 的打印格式，更详细的可以看 log4j 自带的文档：

%m	输出代码中指定的消息
%p	输出优先级，即 DEBUG，INFO，WARN，ERROR，FATAL
%r	输出自应用启动到输出该 log 信息耗费的毫秒数
%c	输出所属的类目，通常就是所在类的全名
%t	输出产生该日志事件的线程名
%n	输出一个回车换行符，Windows 平台为“ \r\n ”，Unix 平台为“ \n ”
%d	输出日志时间点的日期或时间，默认格式为 ISO8601，也可以在其后指定格式 比如：%d{yyy MMM dd HH:mm:ss,SSS} 输出类似：2002 年 10 月 18 日 22：10：28，921
%l	输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数 举例：Testlog4.main(TestLog4.java:10)

3.4 外部配置文件

外部配置文件方式在我们日常使用中占着最大比重。如同上提到的，我们总是不手动对记录器、存放器进行分配，而是统一从 root 继承下来。那么 root 的相关值呢？对，可以从外部配置文件中读取，在初始化时完成这个操作（具体的构建对于我们来说是透明的，我们仅仅是一个初始化方法），这样很多可选项不必硬编码在软件中。使用外部配置文件的优点就是修改可选项不需要重新编译程序。唯一的缺点就是，由于用到 io 指令，速度稍微有些减慢。这个缺点也不是什么大问题，因为只在初始化时做一次罢了。

外部配置文件有两种存在方式，对应初始化也有两种方法。

XML 文件、文本文件

因为涉及的概念不多，这里我先不详细写。现在 XML 比较流行，我们又是初用它，在下面的实作中就用 XML 结合例子来说吧。

4 Log4j 的下载、安装及简单应用

4.1 下载

jakarta-log4j-1.2.8 (full package, includes binaries and source files)

http://apache.linuxforum.net/dist/logging/log4j/1.2.8/jakarta-log4j-1.2.8.zip

4.2 在 Jbuilder9 中安装

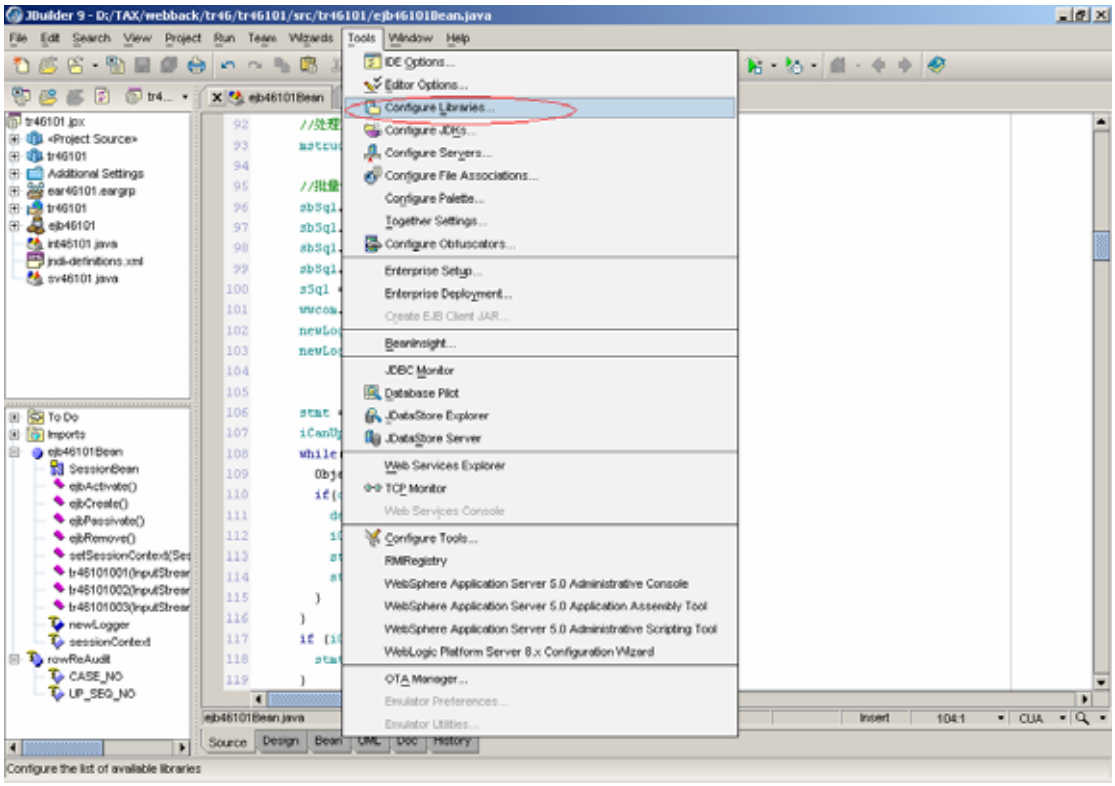
以前已多次讲过安装包的方法，这里就不再详细写了，简易步骤如下：

步骤一：

解压，找到 jakarta-log4j-1.2.8\dist\lib\log4j-1.2.8.jar

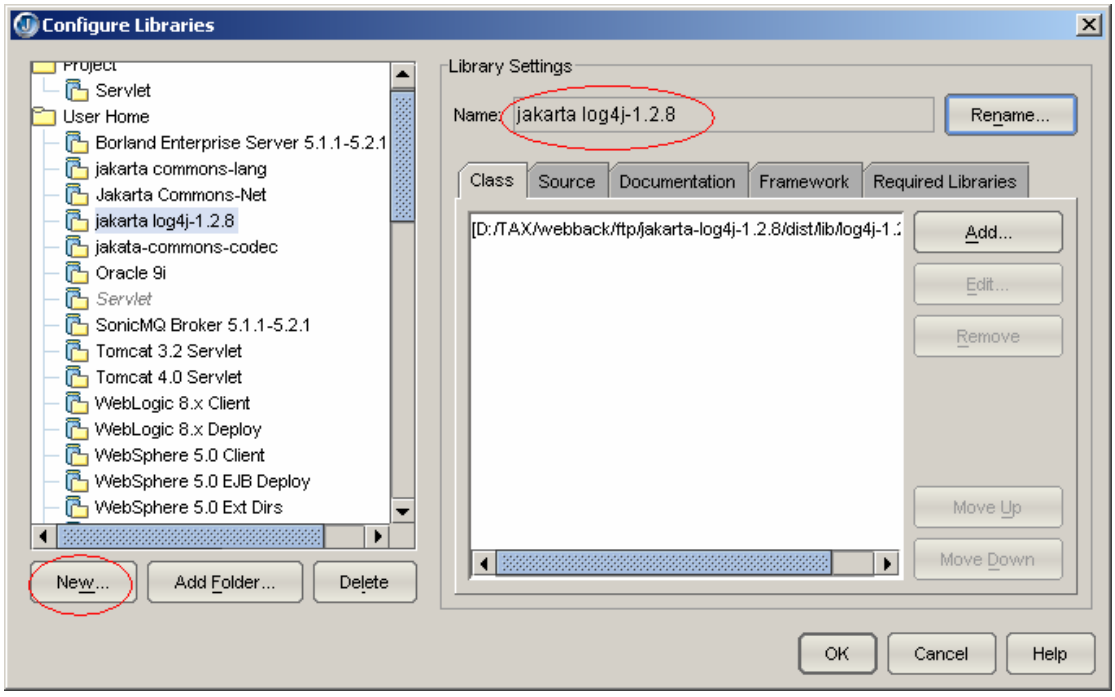
步骤二：

启动 Jbuilder9，进入 Tool->Configure Libraires…



步骤三：

New 一个包，取名为 jakarta log4j-1.2.8，选入包体、源码、文档，然后 OK



好了，这样就完成安装了，下面写些例子来看看吧

4.3 简单应用

4.3.1 XML 配置文件分析

先来看看这个 XML 文件：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
 <appender name="A1" class="org.apache.log4j.FileAppender">
                <param name="File"      value="A1.log" />
                <param name="Append" value="true" />
                <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%t %-5p %c{2} - %m%n"/>
                </layout>
 </appender>
 <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
       <layout class="org.apache.log4j.PatternLayout">
                        <param name="ConversionPattern"
                    value="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
       </layout>
 </appender>
 <category name="org.apache.log4j.xml">
    <priority value="debug" />
          <appender-ref ref="A1" />
 </category>
 <root>
      <priority value ="info" />
           <appender-ref ref="STDOUT" />
           <appender-ref ref="A1" />
 </root>
</log4j:configuration>
```

这个文件可以在 jakarta-log4j-1.2.8\src\java\org\apache\log4j\xml\examples 中找到，文件名为 sample2.xml，红色部分是我修改的，蓝色部分不用理会，category 是 log4j 以前的核心，现在已被 logger 所替代了，但是为了兼容所以没有去除，对于我们来说就没用了。

XML 的概念我就不多说了，对于 log4j 来说大家看看 log4j.dtd 就可以了解，log4j.dtd 文件大家可以在 jakarta-log4j-1.2.8\src\java\org\apache\log4j\xml 中找到。

分段一：

```
<appender name="A1" class="org.apache.log4j.FileAppender">
        <param name="File"      value="A1.log" />
        <param name="Append" value="true" />
        <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%t %-5p %c{2} - %m%n"/>
        </layout>
</appender>
```

这个描述了一个存放器

```
<appender name="A1" class="org.apache.log4j.FileAppender">
```

名字是 A1，是一个 FileAppender

```
        <param name="File"      value="A1.log" />
```

存放的文件名是 A1.log

```
        <param name="Append" value="true" />
```

使用追加方式而不是覆盖

```
        <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%t %-5p %c{2} - %m%n"/>
        </layout>
```

布局使用 PatternLayout。

分段二：

```
<appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
                <param name="ConversionPattern"
            value="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
```

```
        </layout>
    </appender>
```

这个描述了另一个存放器，名字是 STDOUT，是一个 ConsoleAppender，就是 System.out 了；布局也一样使用 PatternLayout。

分段三：

```
    <root>
        <priority value ="info" />
            <appender-ref ref="STDOUT" />
            <appender-ref ref="A1" />
    </root>
```

这个描述了 root 的信息

```
        <priority value ="info" />
```

说明 root 的最小级别是 info，也就是说，debug 级别的日志将不会记录

```
            <appender-ref ref="STDOUT" />
            <appender-ref ref="A1" />
```

说明对应着两个存放器 STDOUT、A1，也说是说，日志将同时输出到这两个地方。

4.3.2 程序代码分析

```
package log4jtest;


import org.apache.log4j.xml.DOMConfigurator;
import org.apache.log4j.Logger;


public class apptest {
   static Logger mylogger = Logger.getLogger(apptest.class.getName());
   public apptest() {
      DOMConfigurator.configure("sample2.xml");
      int i = -1;
      mylogger.debug("Message " + ++i);
```

```
        mylogger.warn ("Message " + ++i);
        mylogger.error("Message " + ++i);
        Exception e = new Exception("Just testing");
        mylogger.debug("Message " + ++i, e);
    }
    public static void main(String[] args) {
        apptest apptest1 = new apptest();
    }
}
```

程序段一：

```
    static Logger mylogger = Logger.getLogger(apptest.class.getName());
```

定义一个静态的全局变量，提供给下面程序使用

程序段二：

```
        DOMConfigurator.configure("sample2.xml");
```

从 XML 配置文件中读取信息初始化 root，这时 sample2.xml 必须与 jpx 在同一目录下，DOMConfigureator.configure 的方式还有多种，大家可以看它的文档

程序段三：

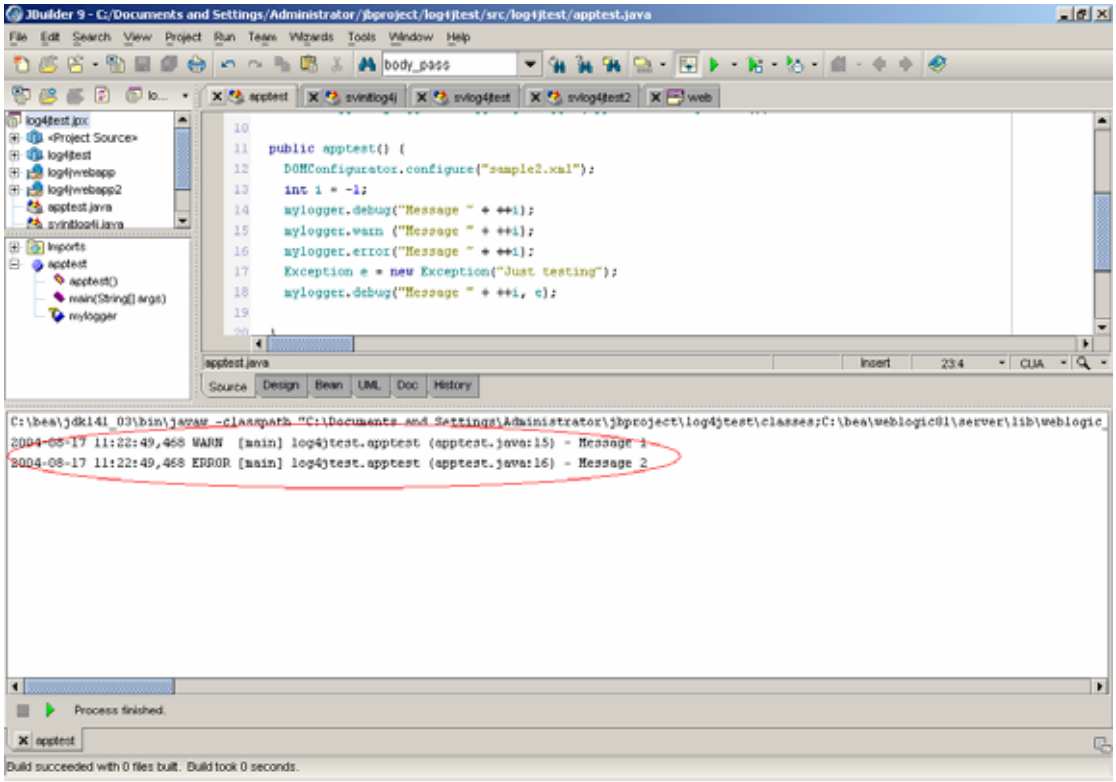
```
        mylogger.debug("Message " + ++i);
        mylogger.warn ("Message " + ++i);
        mylogger.error("Message " + ++i);
        Exception e = new Exception("Just testing");
        mylogger.debug("Message " + ++i, e);
```

这个就是具体的日志输出了

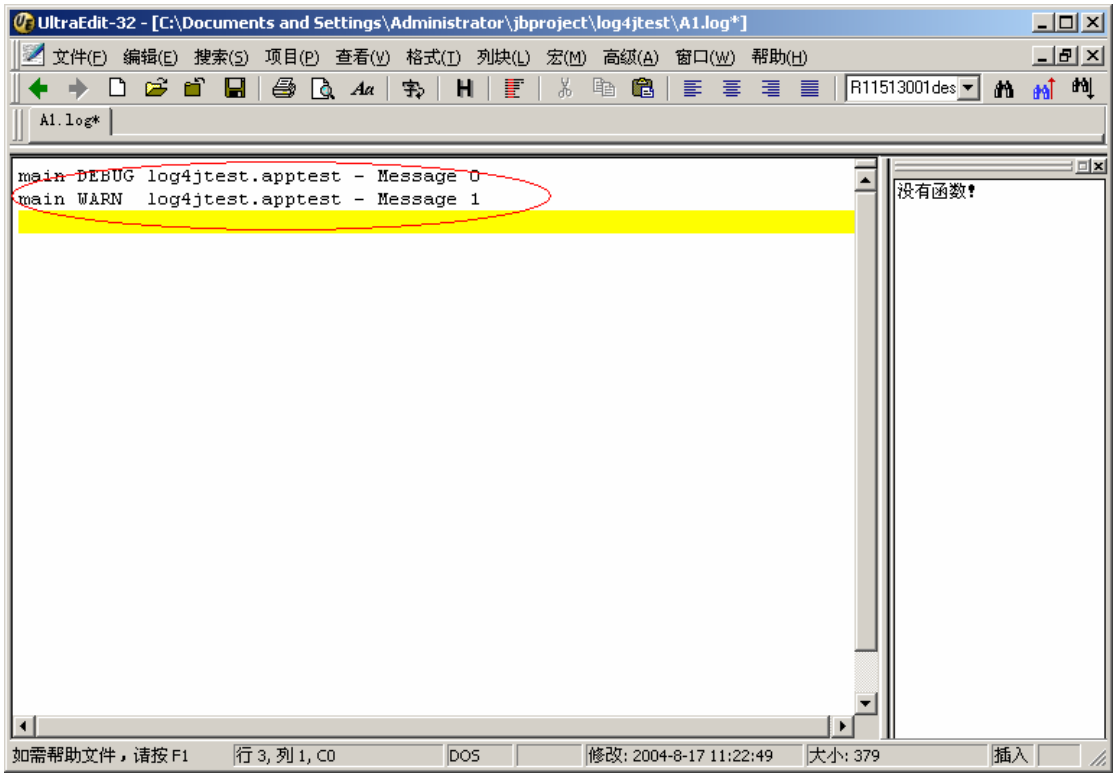
输出一：

由于我们上面定义的 root 的级别是 info，所以程序中的 mylogger.debug 不会记录在日志中。大家会发现在 Console 和文件中的日志信息不一样，那就是 PatternLayout 的作用了。

Console 输出



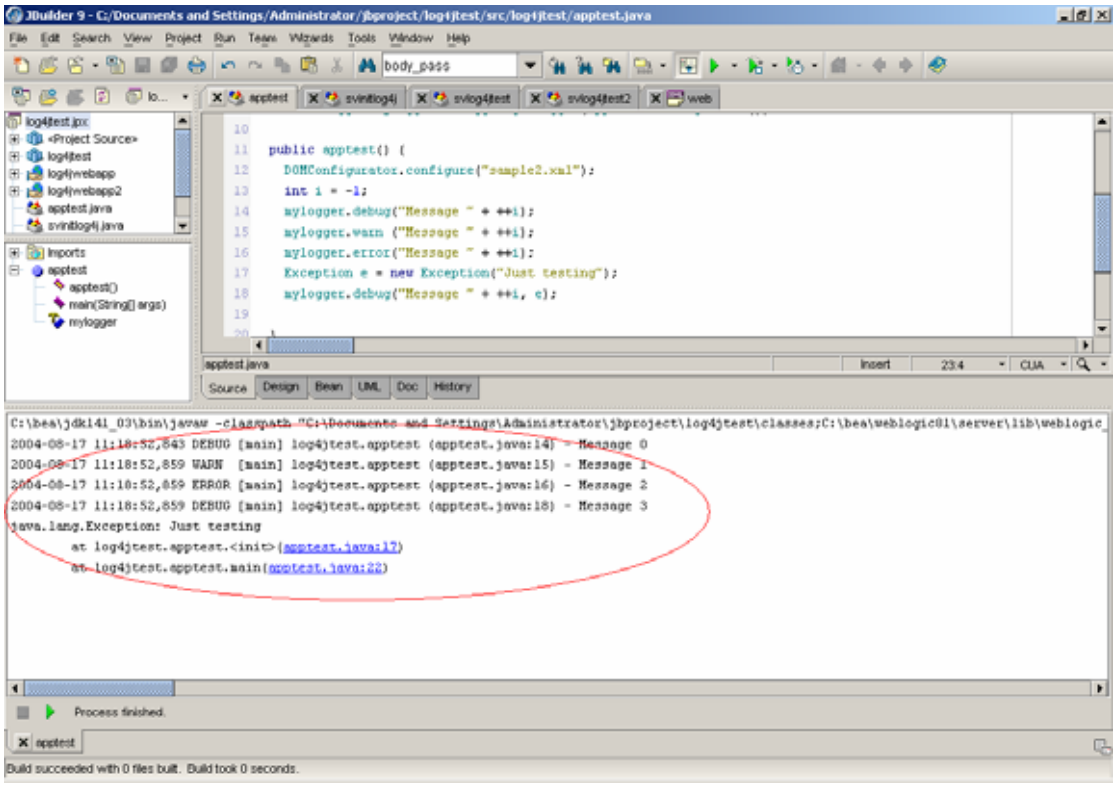
文件输出



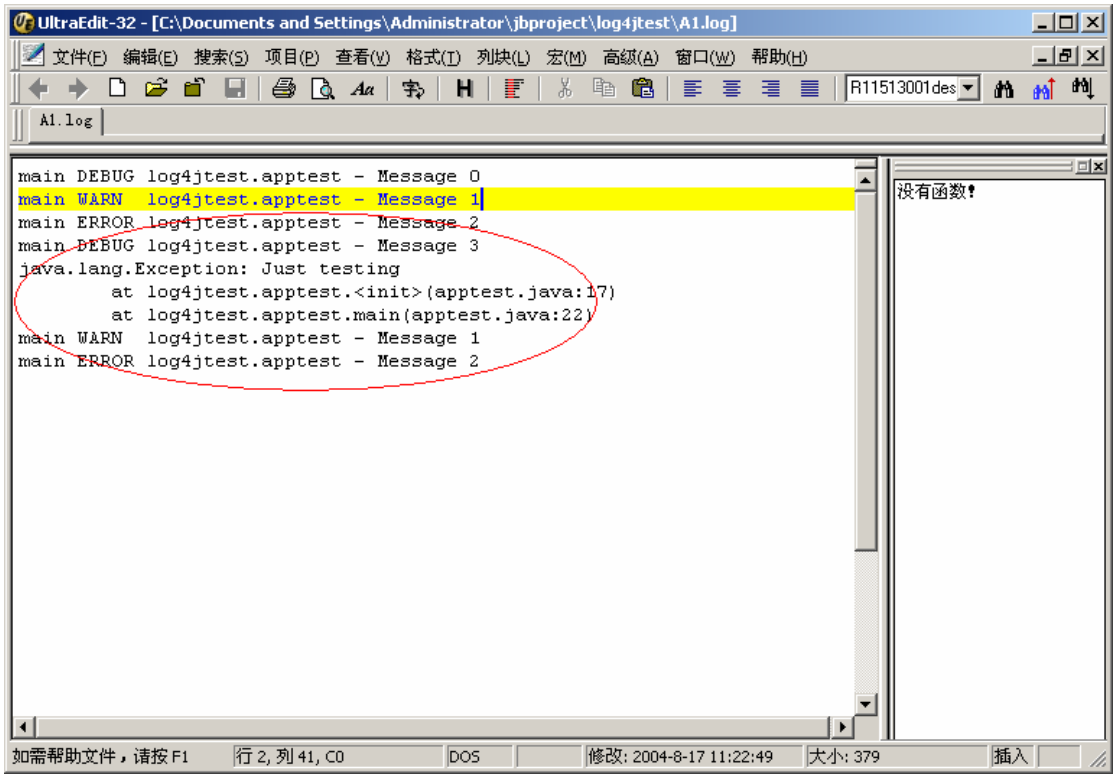
输出二：

将 XML 中 root 的级别改为 debug 后，所有的日志都记录了

Consolse 输出



文件输出



做完这个小例子，大家应该可以发现，了解了 log4j 的概念后，实际操作起来 log4j 还是非常易上手的。

5 Log4j 在 TAMIS 中的使用

上面的单机小程序中，我在程序的最前部进行 log4j 初始化操作，那么对于 TAMIS 这样的较大型的 J2EE 应用来说，我该如何正确的进行初始化操作呢？

这个涉及到 JAVA 的一个概念：类加载。这里我不详细的说它了，如果大家需要，告诉我，我再写。简单来说，一个子类需要一个外加包时，它会先看其基类是否有加载这个包，如果没有再找自已有没加载，仍没有就报异常了。这样就有两种方式加载了，一个是放在子类，各自加载，这样可能个性化；另一个就是放在基类，这样比较统一，但是自然个性化就没有了。TAMIS 采用的是在基类加载的方式，包括了 WWCOM 包也是这样。

步骤一：

将 log4j 包加入 WebLogic Server8.1 的 ClassPath 中

找到 user_projects\domains\tamis\ startWebLogic.cmd 文件

打开，找到 set CLASSPATH，在最后加上 x:\xxx\ log4j-1.2.8.jar

启动 WebLogic Server 8.1

步骤二：

那具体的做法是将一个 Servlet 在 WebLogic Server 8.1 启动时就加载，由它进行 root 的初始化工作。这个 Servlet 的代码很简单，如下：

```
package log4jtest;


import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import org.apache.log4j.xml.DOMConfigurator;


public class svinitlog4j extends HttpServlet {
   private static final String CONTENT_TYPE = "text/html; charset=GBK";
   //Initialize global variables
```

```
    public void init() throws ServletException {
        DOMConfigurator.configure("c:/sample2.xml");
    }
    //Process the HTTP Get request
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    }
    //Process the HTTP Post request
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
    //Clean up resources
    public void destroy() {
    }
}
```

我们真正需要它起作用的就是蓝色的那一句，其它的都是 Jbuilder 自已生成的。注意到这意话中初始化的参数是 c:/sample2.xml，也就是 sample2.xml 必须存在于 WebLogic Servlet 8.1 那台机器的 c:\下。DOMConfigurator.configure 的构造方法还有多个（包括用 URL），我使用的是最简单明了的一种。

步骤三：

要使这个 Servlet 在启动时就加载，必须修改 web.xml，具体如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>svinitlog4j</servlet-name>
    <servlet-class>log4jtest.svinitlog4j</servlet-class>
    <load-on-startup>1</load-on-startup>
```

```
    </servlet>
    <servlet-mapping>
      <servlet-name>svinitlog4j</servlet-name>
      <url-pattern>/svinitlog4j</url-pattern>
    </servlet-mapping>
  </web-app>
```

同样，起作用的也是蓝色那一句，其它的也是 Jbuilder 自动生成的。

将这个 War 发布后，WebLogic Server 8.1 中 root 就初始化好了。当我们想要改变 log 的输出时，先将 c:/sample2.xml 进行修改，然后只需重新发布这个 Servlet 就可以了。

步骤四：

编写 Servlet 代码进行测试

步骤五：

编写 EJB 代码进行测试

具体代码不写在这儿，通过以前对 J2EE 的讲解以及上面对 log4j 的讲解，我想大家应该可以写得出来了，只在这儿再贴出关键部分代码

```
    static Logger mylogger = Logger.getLogger(apptest.class.getName());
    mylogger.debug("Message " + ++i);
    mylogger.warn ("Message " + ++i);
    mylogger.error("Message " + ++i);
    Exception e = new Exception("Just testing");
    mylogger.debug("Message " + ++i, e);
```

6 参考资料

http://logging.apache.org/log4j/docs/manual.html

http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/log4j/log4j.html

7 下期预告

结合 WebService 以及 TAMIS 的传输系统，讨论在分布式应用中，进行大数据（二进制）传输时要注意的地方