日志

Mybatis 的内置日志工厂提供日志功能，内置日志工厂将日志交给以下其中一种工具作代理：

- SLF4J
- Apache Commons Logging
- Log4j 2
- Log4j
- JDK logging

MyBatis 内置日志工厂基于运行时自省机制选择合适的日志工具。它会使用第一个查找得到的工具（按上文列举的顺序查找）。如果一个都未找到，日志功能就会被禁用。

不少应用服务器（如 Tomcat 和 WebShpere）的类路径中已经包含 Commons Logging，所以在这种配置环境下的 MyBatis 会把它作为日志工具，记住这点非常重要。这将意味着，在诸如 WebSphere 的环境中，它提供了 Commons Logging 的私有实现，你的 Log4J 配置将被忽略。MyBatis 将你的 Log4J 配置忽略掉是相当令人郁闷的（事实上，正是因为在这种配置环境下，MyBatis 才会选择使用 Commons Logging 而不是 Log4J）。如果你的应用部署在一个类路径已经包含 Commons Logging 的环境中，而你又想使用其它日志工具，你可以通过在 MyBatis 配置文件 mybatis-config.xml 里面添加一项 setting 来选择别的日志工具。

```
<configuration>
  <settings>
    ...
    <setting name="logImpl" value="LOG4J"/>
    ...
  </settings>
</configuration>
```

logImpl 可选的值有：SLF4J、LOG4J、LOG4J2、JDK_LOGGING、COMMONS_LOGGING、STDOUT_LOGGING、NO_LOGGING，或者是实现了接口 `org.apache.ibatis.logging.Log` 的，且构造方法是以字符串为参数的类的完全限定名。（译者注：可以参考org.apache.ibatis.logging.slf4j.Slf4jImpl.java的实现）

你也可以调用如下任一方法来使用日志工具：

```
org.apache.ibatis.logging.LogFactory.useSlf4jLogging();
org.apache.ibatis.logging.LogFactory.useLog4JLogging();
org.apache.ibatis.logging.LogFactory.useJdkLogging();
org.apache.ibatis.logging.LogFactory.useCommonsLogging();
org.apache.ibatis.logging.LogFactory.useStdOutLogging();
```

如果你决定要调用以上某个方法，请在调用其它 MyBatis 方法之前调用它。另外，仅当运行时类路径中存在该日志工具时，调用与该日志工具对应的方法才会生效，否则 MyBatis 一概忽略。如你环境中并不存在 Log4J，你却调用了相应的方法，MyBatis 就会忽略这一调用，转而以默认的查找顺序查找日志工具。

关于 SLF4J、Apache Commons Logging、Apache Log4J 和 JDK Logging 的 API 介绍不在本文档介绍范围内。不过，下面的例子可以作为一个快速入门。关于这些日志框架的更多信息，可以参考以下链接：

- Apache Commons Logging (http://commons.apache.org/logging)
- Apache Log4j (http://logging.apache.org/log4j/)
- JDK Logging API (http://java.sun.com/j2se/1.4.1/docs/guide/util/logging/)

日志配置

你可以对包、映射类的全限定名、命名空间或全限定语句名开启日志功能来查看 MyBatis 的日志语句。

再次说明下，具体怎么做，由使用的日志工具决定，这里以 Log4J 为例。配置日志功能非常简单：添加一个或多个配置文件（如 log4j.properties），有时需要添加 jar 包（如 log4j.jar）。下面的例子将使用 Log4J 来配置完整的日志服务，共两个步骤：

步骤 1：添加 Log4J 的 jar 包

因为我们使用的是 Log4J，就要确保它的 jar 包在应用中是可用的。要启用 Log4J，只要将 jar 包添加到应用的类路径中即可。Log4J 的 jar 包可以在上面的链接中下载。

对于 web 应用或企业级应用，则需要将 `log4j.jar` 添加到 `WEB-INF/lib` 目录下；对于独立应用，可以将它添加到JVM 的 `-classpath` 启动参数中。

步骤 2：配置 Log4J

配置 Log4J 比较简单，假如你需要记录这个映射器接口的日志：

```
package org.mybatis.example;
public interface BlogMapper {
  @Select("SELECT * FROM blog WHERE id = #{id}")
  Blog selectBlog(int id);
}
```

在应用的类路径中创建一个名称为 `log4j.properties` 的文件，文件的具体内容如下：

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.org.mybatis.example.BlogMapper=TRACE
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

添加以上配置后，Log4J 就会记录 `org.mybatis.example.BlogMapper` 的详细执行操作，且仅记录应用中其它类的错误信息（若有）。

你也可以将日志的记录方式从接口级别切换到语句级别，从而实现更细粒度的控制。如下配置只对 `selectBlog` 语句记录日志：

```
log4j.logger.org.mybatis.example.BlogMapper.selectBlog=TRACE
```

与此相对，可以对一组映射器接口记录日志，只要对映射器接口所在的包开启日志功能即可：

```
log4j.logger.org.mybatis.example=TRACE
```

某些查询可能会返回庞大的结果集，此时只想记录其执行的 SQL 语句而不想记录结果该怎么办？为此，Mybatis 中 SQL 语句的日志级别被设为DEBUG（JDK 日志设为 FINE），结果的日志级别为 TRACE（JDK 日志设为 FINER)。所以，只要将日志级别调整为 DEBUG 即可达到目的：

```
log4j.logger.org.mybatis.example=DEBUG
```

要记录日志的是类似下面的映射器文件而不是映射器接口又该怎么做呢？

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

如需对 XML 文件记录日志，只要对命名空间增加日志记录功能即可：

```
log4j.logger.org.mybatis.example.BlogMapper=TRACE
```

要记录具体语句的日志可以这样做：

```
log4j.logger.org.mybatis.example.BlogMapper.selectBlog=TRACE
```

你应该注意到了，为映射器接口和 XML 文件添加日志功能的语句毫无差别。

注意 如果你使用的是 SLF4J 或 Log4j 2，MyBatis 将以 MYBATIS 这个值进行调用。

配置文件 `log4j.properties` 的余下内容是针对日志输出源的，这一内容已经超出本文档范围。关于 Log4J 的更多内容，可以参考Log4J 的网站。不过，你也可以简单地做做实验，看看不同的配置会产生怎样的效果。
