链滴

MyBatis 日志模块源码分析

作者：zwxbest

原文链接：https://ld246.com/article/1545053651618

来源网站：链滴

相关设计模式

适配器模式

```
@startuml
namespace  mybatis {
interface Log
class LogFactory
LogFactory .down.> Log
Jdk14LoggingImpl .up.|> Log
Slf4jImpl .up.|> Log
StdOutImpl .up.|> Log
NoLoggingImpl .up.|> Log
JakartaCommonsLoggingImpl .up.|> Log
Log4jImpl .up.|> Log
Log4j2Impl .up.|> Log
}

namespace slf4j_api {
interface Logger
}
namespace log4j_api_2 {
interface Logger
}
namespace java.lang {
class System
}
namespace java.util.logging {
class Logger
}
namespace jcl_over_slf4j {
interface Log
}
namespace log4j {
interface Logger
}
mybatis.Slf4jImpl o-down-  slf4j_api.Logger
mybatis.StdOutImpl o-down- java.lang.System
mybatis.Jdk14LoggingImpl o-down- java.util.logging.Logger
mybatis.JakartaCommonsLoggingImpl o-down- jcl_over_slf4j.Log
mybatis.Log4jImpl o-down- log4j.Logger
mybatis.Log4j2Impl o-down-  log4j_api_2.Logger
@enduml
```

目标接口为Log接口，需要适配的类为实际提供日志支持的类，比如log4j_api_2 .Logger，适配器实现了Log接口，并由mybatis.Log4j2Impl组成，两者之间是聚合关系。

门面模式

```
@startuml
namespace slf4j_api {
```

```
interface Logger
interface LocationAwareLogger
LocationAwareLogger -up-|> Logger
}
namespace log4j_slf4j_impl {
class Log4jLogger
}
log4j_slf4j_impl.Log4jLogger .up.|> slf4j_api.LocationAwareLogger

namespace Client {
}
Client -down-> slf4j_api.Logger
@enduml
```

slf4j_api.Logger充当门面，Client调用slf4j_api.Logger获得服务。

具体的服务支持由og4j_slf4j_impl.Log4jLogger 或jul_over_slf4j.Log4jLogger等等提供支持

代理模式

```
@startuml
interface Connection
ConnectionImpl .up.|> Connection
note left of ConnectionImpl:不需要创建此类
ConnectionProxy .up.|> Connection
ConnectionProxy  o-up- "1" ConnectionImpl
@enduml
```

ConnectionProxy作为Connection接口实现类的代理类。

日志选择

org.apache.ibatis.logging.LogFactory静态初始化块中

```
tryImplementation(new Runnable() {
  @Override
  public void run() {
    useSlf4jLogging();
  }
});
tryImplementation(new Runnable() {
  @Override
  public void run() {
    useCommonsLogging();
  }
});
tryImplementation(new Runnable() {
  @Override
  public void run() {
    useLog4J2Logging();
  }
});
tryImplementation(new Runnable() {
  @Override
```

```
  public void run() {
    useLog4JLogging();
  }
});
tryImplementation(new Runnable() {
  @Override
  public void run() {
    useJdkLogging();
  }
});
tryImplementation(new Runnable() {
  @Override
  public void run() {
    useNoLogging();
  }
});

private static void tryImplementation(Runnable runnable) {
  if (logConstructor == null) {
    try {
      runnable.run();
    } catch (Throwable t) {
      // ignore
  }
  }
}
```

一共调用了6次tryImplementation，每次调用传入不同的门面，按顺序找到第1个实现了门面逻的子系统，写入logConstructor，不再往下找。

这里虽然创建了Runnable，但并没有启动线程，因为tryImplementation调用的是run而不是start

以log4j2为例

只贴出流程中需要的代码，其他的用...表示省略

```
public Slf4jImpl(String clazz) {
  Logger logger = LoggerFactory.getLogger(clazz);
  ...
}

public static Logger getLogger(String name) {
    ILoggerFactory iLoggerFactory = getILoggerFactory();
     ...
}
```

iLoggerFactory打上条件断点，条件是name.contains("LogFactory")。

在expression中输出class org.apache.logging.slf4j.Log4jLoggerFactory，可看到name为org.apache.logging.slf4j.Log4jLoggerFactory，在log4j-slf4j-impl中。

选择了slf4j门面，log4j-slf4j-impl为子系统

日志打印逻辑

```
@startuml
abstract class BaseJdbcLogger
interface InvocationHandler
StatementLogger -up-|> BaseJdbcLogger
ConnectionLogger -up-|> BaseJdbcLogger
ResultSetLogger -up-|> BaseJdbcLogger
PreparedStatementLogger -up-|> BaseJdbcLogger
StatementLogger .down.|> InvocationHandler
ConnectionLogger .down.|> InvocationHandler
ResultSetLogger .down.|> InvocationHandler
PreparedStatementLogger .down.|> InvocationHandler
@enduml
```

在代理类的invoke方法中打印log

打印完整日志

```
import lombok.extern.log4j.Log4j2;
import org.apache.commons.collections.CollectionUtils;
import org.apache.ibatis.executor.Executor;
import org.apache.ibatis.mapping.BoundSql;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.ParameterMapping;
import org.apache.ibatis.plugin.*;
import org.apache.ibatis.reflection.MetaObject;
import org.apache.ibatis.session.Configuration;
import org.apache.ibatis.session.ResultHandler;
import org.apache.ibatis.session.RowBounds;
import org.apache.ibatis.type.TypeHandlerRegistry;

import java.text.DateFormat;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;
import java.util.regex.Matcher;
import org.springframework.stereotype.Component;

@Component
@Log4j2
@Intercepts({
    @Signature(type = Executor.class, method = "update", args = {
        MappedStatement.class, Object.class}),
    @Signature(type = Executor.class, method = "query", args = {
        MappedStatement.class, Object.class, RowBounds.class,
        ResultHandler.class})})
public class MyBatisLogInterceptor implements Interceptor {

    @Override
  public Object intercept(Invocation invocation) throws Throwable {
        try {
```

```
            MappedStatement mappedStatement = (MappedStatement) invocation
                .getArgs()[0];
            Object parameter = null;
            if (invocation.getArgs().length > 1) {
                parameter = invocation.getArgs()[1];
            }
            String sqlId = mappedStatement.getId();
            BoundSql boundSql = mappedStatement
                .getBoundSql(parameter);
            Configuration configuration = mappedStatement.getConfiguration();
            String sql = getSql(configuration, boundSql, sqlId);
            log.info(sql);
        } catch (Exception e) {
            log.error(e.getMessage(), e);
        }
        return invocation.proceed();
    }

    private static String getSql(Configuration configuration, BoundSql boundSql, String sqlId) {
        String sql = showSql(configuration, boundSql);
        return sqlId.concat(":").concat(sql);
    }

    /**
     * 如果参数是String，则添加单引号
     * 如果是日期，则转换为时间格式器并加单引号
     * 对参数是null和不是null的情况作了处理
     */
    private static String getParameterValue(Object obj) {
        String value = null;
        if (obj instanceof String) {
            value = "'" + obj.toString() + "'";
        } else if (obj instanceof Date) {
            DateFormat formatter = DateFormat
                .getDateTimeInstance(DateFormat.DEFAULT, DateFormat.DEFAULT, Locale.CHINA);
            value = "'" + formatter.format(new Date()) + "'";
        } else {
            if (obj != null) {
                value = obj.toString();
            } else {
                value = "";
            }

        }
        return value;
    }

    // 进行?的替换
    private static String showSql(Configuration configuration, BoundSql boundSql) {
        Object parameterObject = boundSql.getParameterObject();
        List parameterMappings = boundSql
            .getParameterMappings();
        String sql = boundSql.getSql().replaceAll("[\\s]+", " ");
        if (CollectionUtils.isNotEmpty(parameterMappings) && parameterObject != null) {
```

```
            TypeHandlerRegistry typeHandlerRegistry = configuration
                .getTypeHandlerRegistry();
            if (typeHandlerRegistry.hasTypeHandler(parameterObject.getClass())) {
                sql = sql.replaceFirst("\\?",
                    Matcher.quoteReplacement(getParameterValue(parameterObject)));
            } else {
                MetaObject metaObject = configuration.newMetaObject(
                    parameterObject);
                for (ParameterMapping parameterMapping : parameterMappings) {
                    String propertyName = parameterMapping.getProperty();
                    if (metaObject.hasGetter(propertyName)) {
                        Object obj = metaObject.getValue(propertyName);
                        sql = sql
                            .replaceFirst("\\?", Matcher.quoteReplacement(getParameterValue(obj)));
                    } else if (boundSql.hasAdditionalParameter(propertyName)) {
                        Object obj = boundSql.getAdditionalParameter(propertyName);
                        sql = sql
                            .replaceFirst("\\?", Matcher.quoteReplacement(getParameterValue(obj)));
                    } else {
                        sql = sql.replaceFirst("\\?", "missing");
                    }
                }
            }
        }
        return sql;
    }

    @Override
  public Object plugin(Object target) {
        return Plugin.wrap(target, this);
    }

    @Override
  public void setProperties(Properties properties) {

    }
}
```

关于拦截器、JDK动态代理、CGLIB，有时间再开一篇。