系统开发 log 日志使用规范

目的

在系统开发过程中，项目经理都非常强调编码中 log 的重要性，并且强迫每个程序员都要求写 log。但是为啥要写 log，怎么写 log，却很少有人去说去讲，让大家明白写 log 的目的性，有利于大家理解这样的行为。

现在我们就来讨论一下写 log 是目的是什么?

在代码中嵌入 log 代码信息，主要记录下列信息：

1、 记录系统运行异常信息。

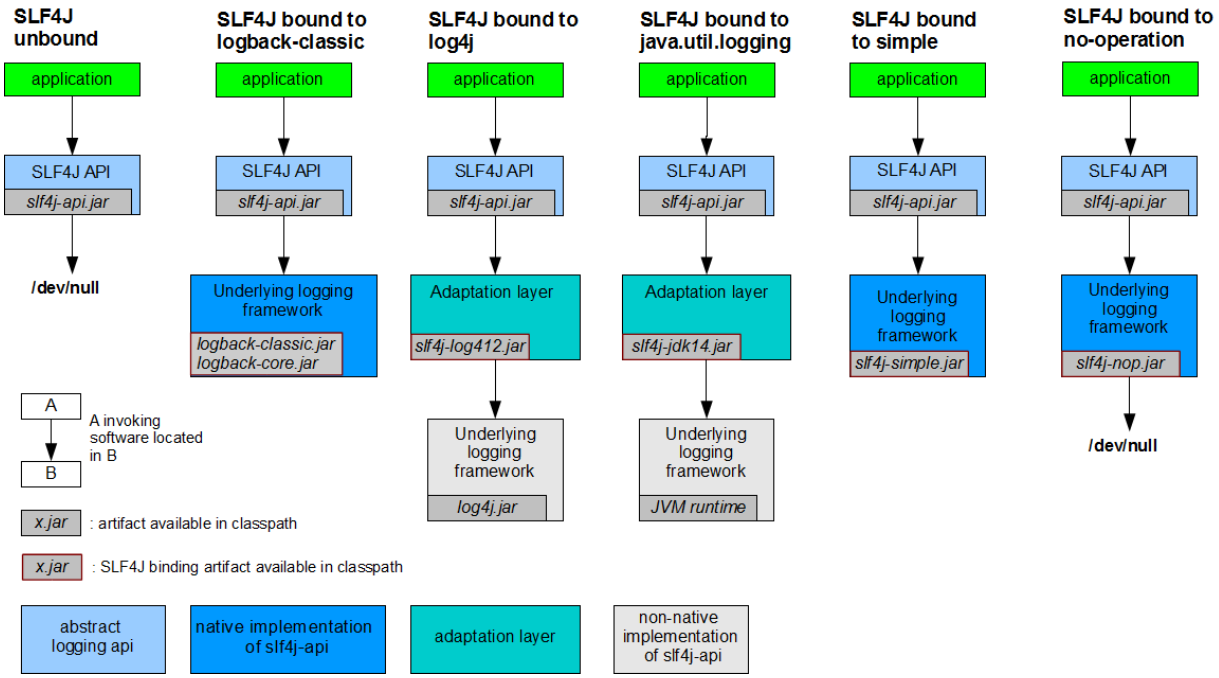
2、 记录系统运行状态信息。

3、 记录系统运行性能指标。

通过对上述信息分析和诊断，我们能采取正确的手段来提高系统质量 和 提升系统性能。

Java 日志组件选型

针对 spring 容器进行日志选型：选择 slf4j + 自己想用的实现 log 类（推荐采用 log4j）。



选择 slf4j 的理由：

1、 可以和多种实现融合，具体实现 log 类，只要替换响应的 jar，对应用程序不要做任何修改。
2、 编码简化了，不需要判断是否需要输入的 if-else 语句，通过可变参量格式化输出，方便书写
3、 当系统发布正常运行时，需要关闭 log 时，只要把对应的实现 jar 删除即可。
4、 解决项目中多个 log 组件冲突问题，通过引入一个 slf4j 来实现所有的日志组件自由切换。

在系统开发项目总需要引入以下文件：
Slf4j-api.jar
Slf4j-log4j.jar
Log4j.jar
Log4j.properties

日志类型

主要分三大类：
安全类信息：记录系统边界交互行为和信息
业务类信息：记录系统内部业务处理行为和信息
性能类信息：记录系统硬件对业务处理的支撑能力

日志级别

一般分五级：
ERROR（错误）：此信息输出后，主体系统核心模块正常工作，需要修复才能正常工作。
WARN（警告）：此信息输出后，系统一般模块存在问题，不影响系统运行。
INFO（通知）：此信息输出后，主要是记录系统运行状态等关联信息
DEBUG（调试）：最细粒度的输出，除却上面各种情况后，你希望输出的相关信息，都可以在这里输出。
TRACE（跟踪）：最细粒度的输出，除却上面各种情况后，你希望输出的相关信息，都可以在这里输出。（我们系统中不采用此级别）

日志记录准则

	ERROR	WARN	INFO	DEBUG
安全类信息		合法拒绝	正常	其他
业务类信息	重要模块异	一般模块异常	正常	其他

	常			
性能类信息		超越指标信息	正常	其他

日志记录代码规范

这里的细则以标准的三层架构来分析，包括表示层/接口层、业务层、存储层。

需要写日志的 java 类请加入以下代码：（实现 log 日志功能）

```
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class HelloWorld {
   public static void main(String[] args) {
      Logger logger = LoggerFactory.getLogger(HelloWorld.class);
       logger.info("Hello World");
   }
}
```

建议使用 slf4j 的高版本，使用可变参量的特性。

安全类信息记录　　业务类信息记录　　性能类信息记录

表示层/接口层类

```
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoginAction extends BaseAction
     {
/**
     * 员工登录
     * @return
     * @throws Exception
     */
    public void empLogin(){
```

```
Logger logger = LoggerFactory.getLogger(LoginAction.class);

……
/*
 * 可以采用syslog的中产生的id，id是由外部用户或者定时器调用产生
 */
long logoID=System.currentTimeMillis();
String user="";
String method="";

    logger.info("事务{}用户{}调用{}",logoID,user,method );

……
try
{
    logger.info("事务{}用户{}调用{}调用登陆服务
",logoID,user,method );
        //调用登陆服务
        logger.info("事务{}用户{}调用{}调用登陆服务完成
",logoID,user,method );
}
catch (Exception e)
{
        logger.error("事务{}用户{}调用{}调用登陆服务异常
",logoID,user,method );
        logger.error("事务id="+logoID, e);
}

if(null == loginEmp)
{
    logger.warn("事务{}用户{}调用{}登陆失败
",logoID,user,method );
        //this.session.put("globalURL", this.request.getContextPath() +
"/index.jsp");
```

```
            outJson("{\"success\":false}");
        }else{
                logger.info("事务{}用户{}调用{}用户验证通过
",logoID,user,method );
            this.session.put("user", loginEmp);
            //取得登录员工可以查看的机构id
            String ids = "";
            try
            {
                logger.info("事务{}用户{}调用{}调用权限服务
",logoID,user,method );
                //调用权限服务
                logger.info("事务{}用户{}调用{}调用权限服务完成
",logoID,user,method );
            }
            catch (Exception e)
            {
                    logger.error("事务{}用户{}调用{}调用权限服务异常
",logoID,user,method );
                logger.error("事务id="+logoID, e);
                e.printStackTrace();
            }

            //取得登录员工的机构 上级银行基本的Id
            Long orgId = 0L;
            String orgName = "";
            try
            {
                logger.info("事务{}用户{}调用{}调用获取机构服务
",logoID,user,method );
                // 调用获取机构服务
                logger.info("事务{}用户{}调用{}调用获取机构服务
```

```
",logoID,user,method );
            }
            catch (Exception e)
            {
                logger.error("事务{}用户{}调用{}调用获取机构服务异
常",logoID,user,method );
                logger.error("事务 id="+logoID, e);
                e.printStackTrace();
            }

            this.session.put("bankCode", orgId);
            this.session.put("bankName", orgName);
            this.session.put("indexOrgIds", ids);

if(MD5.MD5To32(loginEmp.getEmpNo()).equals(loginEmp.getEmpPwd())){
                logger.info("事务{}用户{}调用{}密码验证通过
",logoID,user,method );
                outJson("{\"success\":true,\"url\":\" "+
request.getContextPath() +"/jsp/main/system/updatepwd.jsp\"}");
            }else{
                logger.info("事务{}用户{}调用{}密码错误
",logoID,user,method );
                outJson("{\"success\":true,\"url\":\" "+
request.getContextPath() +"/jsp/main/\"}");
            }
            //return SUCCESS;
                logger.debug("事务{}用户{}调用{}完
成,user{},bankcode{},bankName{},indexOrgIds{}",logoID,user,method,use
r,orgId,orgName,ids );
                logger.info("事务{}用户{}调用{}完成
```

```
",logoID,user,method );
        }
    }
```

业务层类

参考表示层的记录方法

存储层类

由于大量采用成熟组件，自身带有日志，无需重复记录

附件

Log4j.properies 标准模板

```
log4j.rootLogger=info, stdout, R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p %d{yy-MM-dd
HH:mm:ss} %c:%L - %m%n

log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=logs\\hcmm_core.log
log4j.appender.R.DatePattern='.'yyyy-MM-dd

log4j.appender.R.layout=org.apache.log4j.PatternLayout
# [日志级别|时间] [线程名|类名|方法名|行号] message
log4j.appender.R.layout.ConversionPattern=[%p|%d{yyyy-MM-dd HH:mm:ss}]
[%t|%C|%M|%L] - %m%n
# ERROR >WARN > INFO > DEBUG, 级别越高，信息输出越少。
log4j.logger.org.hibernate=DEBUG
log4j.logger.org.hibernate.cache=info
log4j.logger.org.hibernate=warn
log4j.logger.org.hibernate.cache=warn

log4j.logger.org.hibernate.SQL=info
```

```
log4j.logger.org.springframework=info
log4j.logger.com.wri.hy.hcmm_baseServer=info
```

log4j 配置规则

输出等级控制：

等级可分为 OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL，如果配置 OFF 则不打出任何信息，如果配置为 INFO 这样只显示 INFO, WARN, ERROR 的 log 信息，而 DEBUG 信息不会被显示。

输出类型控制：

org.apache.log4j.ConsoleAppender（控制台），

org.apache.log4j.FileAppender（文件），

org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件），

org.apache.log4j.RollingFileAppender(文件大小到达指定尺寸的时候产生一个新的文件）

org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）

输出信息模式：

org.apache.log4j.HTMLLayout（以 HTML 表格形式布局），

org.apache.log4j.PatternLayout（可以灵活地指定布局模式），

org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串），

org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等等信息）

自定义模式：

输出的信息