日志服务

<http://www.ibm.com/developerworks/cn/web/wa-lo-usertrack/>

# 配置

## 新增拦截器

一般情况下，对来自浏览器的请求的拦截，是利用Filter实现的，这种方式可以实现Bean预处理、后处理。 Spring MVC的拦截器不仅可实现Filter的所有功能，还可以更精确的控制拦截精度。

LogIntercepter继承自HandlerInterceptor或HandlerInterceptorAdapter

public class LogInterceptor extends HandlerInterceptorAdapter {

**//在Controller前调用**

@Override

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

     super.preHandle(request, response, handler);

     LogHelper.begin(request);

     return true;

}

**//在Controller后，生成视图前调用**

@Override

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {

     if(!(handler instanceof HandlerMethod)){

          return ;

     }

     ((HandlerMethod)handler).getMethod();//FIXME处理返回值

     super.postHandle(request, response, handler, modelAndView);

     LogHelper.handled();

}

**//在DispatcherServlet完全处理完请求后被调用**

**可根据ex是否为null判断是否发生异常，进行日志记录**

@Override

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {

     super.afterCompletion(request, response, handler, ex);

     LogHelper.end(request, response, ex);

}

@Override

public void afterConcurrentHandlingStarted(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

     super.afterConcurrentHandlingStarted(request, response, handler);

     LogHelper.end(request, response, null);

}

}

## 配置拦截器

//mt.oa.mvc.xml中配置拦截器

<mvc:interceptors>

     <mvc:interceptor>

          <mvc:mapping path="/oa/\*\*"/>

          <mvc:mapping path="/api/\*\*"/>

          <mvc:mapping path="/service/\*\*"/>

          <bean class="com.meituan.oa.pf.mvc.interceptor.LogInterceptor"/>

     </mvc:interceptor>

</mvc:interceptors>

## Spring 配置Log4j

**/WEB-INF/web.xml：**

<context-param>

        <param-name>log4jConfigLocation</param-name>

        <param-value>/WEB-INF/classes/log4j-dev.properties</param-value>

</context-param>

<listener>

        <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

## 修改log4j配置文件

**/WEB-INF/classes/log4j-dev.properties**

**%x 表示会在每个日志行上打印当前 NDC|MDC 上下文**

logpath=/tmp/logs  //输出路径

**// log4j.rootLogger=[level]，appenderName， ……，level就是设置需要输出信息的级别，后面是appender的输出的目的地，appenderName就是指定日志信息输出到哪个地方。**

log4j.rootCategory=ALL,web  //将所有等级都输出到web上

#biz service log 将DEBUG输出到service上，Appender为DialyRollingFileAppender，文件为service.log，格式为PatternLayout

log4j.logger.SERVICE=DEBUG,service

log4j.appender.service=org.apache.log4j.DailyRollingFileAppender

log4j.appender.service.File=${logpath}/service/service.log

log4j.appender.service.layout=org.apache.log4j.PatternLayout

log4j.appender.service.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}|[%t]|[%C.%M()<%L>] {%m}%n

#db process log

log4j.logger.DB=DEBUG,db

log4j.appender.db=org.apache.log4j.DailyRollingFileAppender

log4j.appender.db.File=${logpath}/db/db.log

log4j.appender.db.layout=org.apache.log4j.PatternLayout

log4j.appender.db.layout.ConversionPattern=%n[%d{yyyy-MM-dd HH:mm:ss}][%t]%n{%m}%n

增加一个处理

## LogHelper

NDC采用了一个类似栈的机制来push和pop上下文信息

MDC内部使用了类似map的机制来存储信息

public class MDC {

//清空map所有的条目。

public static void clear();

//根据key值返回相应的对象

public static object get(String key);

//返回所有的key值.

public static Enumeration getKeys();

//把key值和关联的对象，插入map中

public static void put(String key, Object val),

//删除key对应的对象

public static  remove(String key)

}

向MDC记录请求信息，包括请求地址等

public static void begin(HttpServletRequest request)

计算请求效率

public static void end(HttpServletRequest request, HttpServletResponse response, Throwable t)

输出错误

public static void operror(String message, BuzException buzerror)

## ProfLogHelper

返回long数组

第一个值是当前时间

第二个值是当前使用内存

第三个值是当前线程的CPU时间

public static long[] begin()

返回long数组

第一个值是请求使用时间

第二个值是请求使用内存

第三个值是请求使用CPU时间

public static long[] cal(long[] start)

返回JSON字符串

{

rid：msg

time：使用时间

memory：占用内存量

cpu：占用cpu时间

}

public static String end(long[] start, Object msg)

# 读取日志

系统日志分为四类：

1. 操作日志：oplog.js
2. 性能日志：perflog.js
3. 技术日志：techlog.js
4. 界面日志：uilog.js

LogController相应url请求。

LogBizService，定义查询接口。

LogBizServiceImpl：利用MongoOperations查询日志。

# 写入日志

AbstractBusinessService利用

op\_cate：操作日志分类

client：用户信息

msg：具体操作内容

public void oplog(String op\_cate,ClientEnv client,String msg)