|  |  |  |  |
| --- | --- | --- | --- |
| **深圳国泰安教育技术股份有限公司** | **版本** | **密级** | **页数** |
| **V2.0** | **机密** | **共9页** |
| **文档编号: GTA\_CMMI\_TLC** | | |

**国泰安代码提交日志规范**

|  |  |  |  |
| --- | --- | --- | --- |
| **作者** | 刘兵 | **编制日期** | 2014-09-20 |
| **审核** | 黄文 | **审核日期** | 2015-10-26 |
| **批准** | 陈工孟 | **批准日期** | 2015-10-30 |



深圳国泰安教育技术股份有限公司

版权所有 侵权必究

**文档修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版号** | **修改日期** | **修改内容** | **修改人** |
| V1.0 | 2014-09-20 | 根据CMMI复审要求编制 | 刘兵 |
| V2.0 | 2015-10-30 | 正式发布CMMI4流程文档 | / |
|  |  |  |  |

**目录**

[1 目的 3](#_Toc380678778)

[2 适用人群 3](#_Toc380678779)

[3 使用范围 3](#_Toc380678780)

[3.1 日志系统统一使用log4家族日志系统。 3](#_Toc380678781)

[3.2 以下列出C++、C#、Java日志级别分类如下： 4](#_Toc380678782)

[3.3 日志格式 4](#_Toc380678783)

[3.4 各级别日志应包含的信息 4](#_Toc380678784)

[3.5 日志模板 5](#_Toc380678785)

[附录1 日志配置文件标志说明 6](#_Toc380678786)

[附件2 c#日志设计说明 8](#_Toc380678787)

# 目的

本文为了日志的统一化，便于后期维护人员查看日志信息，标准化当前A开发线而制定的日志标准。后续开发人员必须遵循此标准进行开发

# 适用人群

开发人员——本文为开发人员制订了日志标准，所有A开发线开发人员必须在此标准下进行开发。不得违反本规范

测试人员——可以根据本规范进行日志查询，定位问题

运维人员——在正式生产环境下，运维人员可以根据本规范进行问题定位，为用户答疑并提交系统问题。

# 使用范围

本规范的使用范围为A开发线开发团队

## 日志系统统一使用log4家族日志系统。

以下为具体选型：

|  |  |
| --- | --- |
| Log4家族 | |
| Log4C | 适合C |
| Log4CPlus | 适合C++ |
| Log4j | 适合Java |
| Log4Net | 适合DOTNet |
| Log4Perl | 适合Perl |
| Log4PHP | 适合PHP |
| Log4PLSQL | 适合PL/SQL |
| Log4LS | 适合LotusScript |
| Log4py | 适合python |
| qmmslog | 适合Qt/C++ 平台 |
| Log4r | 适合Ruby |
| JDK1.4's util.logging framework | 适合JDK1.4Util |

## 以下列出C++、C#、Java日志级别分类如下：

|  |  |
| --- | --- |
| 日志类别 | |
| TRACE | 跟踪信息(log4cplus独有，模拟堆栈信息) |
| DEBUG | 调试信息 |
| INFO | 一般信息 |
| WARN | 警告信息 |
| ERROR | 错误信息 |
| FATAL | 严重错误信息 |

## 日志格式

%-6t %-6p [%D{%m-%d %H:%M:%S.%q}] [%l] - %m %n

格式说明见附录1。

## 各级别日志应包含的信息

|  |  |  |
| --- | --- | --- |
| 日志类别 | | |
| TRACE | 跟踪信息(log4cplus独有，模拟堆栈信息) | C++输出进入和退出某函数 |
| DEBUG | 调试信息 |  |
| INFO | 一般信息 |  |
| WARN | 警告信息 |  |
| ERROR | 错误信息 |  |
| FATAL | 严重错误信息 |  |

## 日志模板

程序使用日志模板控制日志输出，以下列举C++、C#、Java日志模板样式。

log4cpus日志模板：

log4cplus.rootLogger = TRACE, RF\_Default

log4cplus.appender.RF\_Default = log4cplus::RollingFileAppender

log4cplus.appender.RF\_Default.File = ./log/DEFAULT.log

log4cplus.appender.RF\_Default.MaxFileSize = 100MB

log4cplus.appender.RF\_Default.MaxBackupIndex = 100

log4cplus.appender.RF\_Default.layout = log4cplus::PatternLayout

log4cplus.appender.RF\_Default.layout.ConversionPattern = %-6t %-6p [%D{%m-%d %H:%M:%S.%q}] [%l] - %m %n

log4Net日志模板

<?xml version="1.0" encoding="utf-8" ?>

<log4net>

<appender name="FileAppender" type="log4net.Appender.FileAppender">

<file value="ClientLog.txt" />

<appendToFile value="True" />

<param name="MaxFileSize" value="10240" />

<param name="MaxSizeRollBackups" value="100" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />

</layout>

</appender>

<appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender" >

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />

</layout>

</appender>

<root>

<level value="INFO" />

<appender-ref ref="FileAppender" />

<!--<appender-ref ref="ConsoleAppender" />-->

</root>

</log4net>

log4j日志模板

log4j.rootLogger = DEBUG, stdout, D, E,MAIL

### console ###

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=%5t-%5p-[%-d{yyyy-MM-dd HH:mm:ss,SSS}]-%l-%m%n ### debug file ###

log4j.appender.D = org.apache.log4j.DailyRollingFileAppender

log4j.appender.D.File=logs/debug.log

log4j.appender.D.DatePattern='.'yyyy-MM-dd

log4j.appender.D.Append = true

log4j.appender.D.Threshold = DEBUG

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

log4j.appender.D.layout.ConversionPattern=%5t-%5p-[%-d{yyyy-MM-dd HH:mm:ss,SSS}]-%l-%m%n

### error for sigle file ###

log4j.appender.E = org.apache.log4j.DailyRollingFileAppender

log4j.appender.E.File=logs/error.log

log4j.appender.E.DatePattern='.'yyyy-MM-dd

log4j.appender.E.Append = true

log4j.appender.E.Threshold = ERROR

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

log4j.appender.E.layout.ConversionPattern=%5t-%5p-[%-d{yyyy-MM-dd HH:mm:ss,SSS}]-%l-%m%n

### fatal for sigle file ###

log4j.appender.E = org.apache.log4j.DailyRollingFileAppender

log4j.appender.E.File=logs/fatal.log

log4j.appender.E.DatePattern='.'yyyy-MM-dd

log4j.appender.E.Append = true

log4j.appender.E.Threshold = FATAL

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

log4j.appender.E.layout.ConversionPattern=%5t-%5p-[%-d{yyyy-MM-dd HH:mm:ss,SSS}]-%l-%m%n

附录1 日志配置文件标志说明

（1）"%%"，转义为%, 即，std::string pattern = "%%" 时输出: "%"  
（2）"%c"，输出logger名称，比如std::string pattern ="%c" 时输出: "test\_logger.subtest"，  
     也可以控制logger名称的显示层次，比如"%c{1}"时输出"test\_logger"，其中数字表示层次。  
（3）"%D"，显示本地时间，当std::string pattern ="%D" 时输出:"2004-10-16 18:55:45"，%d显示标准时间，  
     所以当std::string pattern ="%d" 时输出 "2004-10-16 10:55:45" （因为我们是东8区，差8个小时啊）。  
     可以通过%d{...}定义更详细的显示格式，比如%d{%H:%M:%s}表示要显示小时:分钟：秒。大括号中可显示的  
     预定义标识符如下：  
       
%a -- 表示礼拜几，英文缩写形式，比如"Fri"  
%A -- 表示礼拜几，比如"Friday"  
%b -- 表示几月份，英文缩写形式，比如"Oct"  
%B -- 表示几月份，"October"  
%c -- 标准的日期＋时间格式，如 "Sat Oct 16 18:56:19 2004"  
%d -- 表示今天是这个月的几号(1-31)"16"  
%H -- 表示当前时刻是几时(0-23)，如 "18"  
%I -- 表示当前时刻是几时(1-12)，如 "6"  
%j -- 表示今天是哪一天(1-366)，如 "290"  
%m -- 表示本月是哪一月(1-12)，如 "10"  
%M -- 表示当前时刻是哪一分钟(0-59)，如 "59"  
%p -- 表示现在是上午还是下午， AM or PM  
%q -- 表示当前时刻中毫秒部分(0-999)，如 "237"  
%Q -- 表示当前时刻中带小数的毫秒部分(0-999.999)，如 "430.732"  
%S -- 表示当前时刻的多少秒(0-59)，如 "32"  
%U -- 表示本周是今年的第几个礼拜，以周日为第一天开始计算(0-53)，如 "41"  
%w -- 表示礼拜几，(0-6, 礼拜天为0)，如 "6"  
%W -- 表示本周是今年的第几个礼拜，以周一为第一天开始计算(0-53)，如 "41"  
%x -- 标准的日期格式，如 "10/16/04"  
%X -- 标准的时间格式，如 "19:02:34"  
%y -- 两位数的年份(0-99)，如 "04"  
%Y -- 四位数的年份，如 "2004"  
%Z -- 时区名，比如 "GMT"

（4）"%F"，输出当前记录器所在的文件名称，比如std::string pattern ="%F" 时输出: "main.cpp"  
（5）"%L"，输出当前记录器所在的文件行号，比如std::string pattern ="%L" 时输出: "51"  
（6）"%l"，输出当前记录器所在的文件名称和行号，比如std::string pattern ="%L" 时输出:  
     "main.cpp:51"  
（7）"%m"，输出原始信息，比如std::string pattern ="%m" 时输出: "teststr"，即上述代码中  
     LOG4CPLUS\_DEBUG的第二个参数，这种实现机制可以确保原始信息被嵌入到带格式的信息中。  
（8）"%n"，换行符，没什么好解释的  
（9）"%p"，输出LogLevel，比如std::string pattern ="%p" 时输出: "DEBUG"  
（10）"%t"，输出记录器所在的线程ID，比如std::string pattern ="%t" 时输出: "1075298944"  
（11）"%x"，嵌套诊断上下文NDC (nested diagnostic context) 输出，从堆栈中弹出上下文信息，NDC可以用对  
      不同源的log信息（同时地）交叉输出进行区分，关于NDC方面的详细介绍会在下文中提到。  
（12）格式对齐，比如std::string pattern ="%-10m"时表示左对齐，宽度是10，此时会输出"teststr   "，当  
      然其它的控制字符也可以相同的方式来使用，比如"%-12d"，"%-5p"等等（刚接触log4cplus文档时还以为  
      "%-5p"整个字符串代表LogLevel呢，呵呵）。

附件2 c#日志设计说明

c#开发人员在开发过程中，记录日志必须遵循如下规则：

1. ERROR与FATAL日志级别的日志，一定要记录当前发生异常的原因以及堆栈信息，即一定要输出Exception.ToString()，而非Exception.Message
2. 在进行异常处理过程中，根据异常引发的等级，对异常进行分级记录，即使引发的异常原因十分微小，也需要输出Info级别的日志，不能有任何捕获异常而不记录日志的行为
3. 性能日志，建议使用DEBUG级别
4. 针对业务数据引发的异常，在打印Error日志前，一定要打印当前引发异常的快照数据，便于在后续分析过程中针对问题进行处理
5. 一般的调试信息、性能日志，不要使用ERROR日志，因为ERROR及以上日志在终端可能会以邮件的形式进行发送
6. 底层模块中，不建议捕获异常，除非认定异常十分微小
7. 不要使用异常去进行业务逻辑处理