



登临 Hamming V2 SDK

运行时日志使用指南

DL-DG/SW-065A-01

2024-12-27

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼栋5号1101室，江苏，中国

<http://www.denglin.ai>

email : support@denglin.ai

更新历史

版本	更新描述
01	第一次发布。

目录

目录

1 简介

2 日志控制工具 dlutil

3 NNE 日志说明

3.1 日志等级定义

3.2 日志分区定义

3.3 API Dump定义

3.4 动态控制日志

3.4.1 日志分区开关

3.4.2 API_DUMP 开关

3.4.3 使用示例

3.5 环境变量

3.5.1 日志分区总开关

3.5.2 日志分区小开关

3.5.3 API_DUMP 开关

4 HC 日志说明

4.1 HC_MSG_LEVEL 相关

4.2 HC 日志使用 dlutil 控制

4.3 DL_HC_API_DUMP

4.4 HC_DUMP_KLBKSB

4.5 HC_ASSERT_ON_ERROR、HC_ABORT_ON_ASSERTION_FAILURE

5 HAL2 日志说明

5.1 日志等级定义

5.2 控制方式说明

5.3 环境变量控制

5.4 dlutil 配置

5.5 控制方式总结

6 DLEOL 动态控制日志说明

6.1 DLEOL 日志配置

6.2 DLEOL 日志默认环境变量

6.3 dlutil 动态控制 DLEOL 日志

1 简介

登临 Hamming™ V2 SDK 是应用运行在登临硬件平台的软件栈。在应用调试或部署过程中，如果出现预期之外的情况包括程序宕机、GPU挂起、进程崩溃或者运行结果错误等，都可以通过SDK运行时日志工具来获取各个模块的运行情况。

运行时日志功能从范围角度看支持分模块控制，模块内分层控制以及分区控制，便于在应用过程中只输出必要的日志来降低对应用执行速度的影响。

从时间线角度，运行时日志可以通过日志控制工具在应用进程执行过程中动态开关不同范围的日志输出，也可以通过环境变量在进程运行前提前打开日志开关。

日志输出支持多种机制。

本文按照下面的结构组织：

- 日志控制工具dlutil的介绍，该工具适用于所有SDK运行时模块控制。
- SDK运行时各个模块的开关介绍，包括环境变量和对应的动态日志开关。其中NNE模块是网络推理引擎模块，HAL2和HC是底层公共驱动模块，DLEOL是算子库模块。

2 日志控制工具 dlutil

dlutil 用于动态控制目标进程的 Property 和启动 logcat 实例。

```
[dlutil] parse arguments failed.
Usage: dlutil <sub-command> [OPTIONS]
Sub-commands:
  logcat  Logcat related options.
  propctl Property control related options.
Options for logcat:
  --size <buffer size in kB>  Optional, buffer size to use. Default is 4MB, minimum is
16kB.
  --attach <pid1,pid2,...>  Optional, attach to one or more processes with the given
PIDs, separated by ','.
  --pid <pid1,pid2,...>      Just a alias for --attach.
  --id <id>                   Optional, unique ID to identify the logcat instance, default is the
current process ID.
  --output <file>             Optional, output file to write the logs to. - for stdout, + for
stderr, x for null.
Options for propctl:
  --pid <pid>                 Target process id
  <name>.<key>[=<value>] ...
Property control commands.
  --help                      Print this message and exit.
```

下面对dlutil的应用举例。

示例1：改变目标进程(1234) hc2，nne 的 Log Level

```
dlutil propctl --pid 1234 hc2.log_level=2 nne.log_level=3
```

示例2：查询目标进程(1234) hc2，nne 的 Log Level

```
dlutil propctl --pid 1234 hc2.log_level nne.log_level
```

示例3：启动 logcat 实例后，将应用（nnexec）的 log 重定向到此 logcat 实例

```
dlutil logcat --id 4567 //
# [logcat] Logcat URI: logcat://4567 Buffer Size: 4096 kbytes Output: -
DL_LOG_SINK=logcat://4567 nnexec args...
```

示例4：启动 logcat 实例，配置其输出文件 4567.log，应用（nnexec）的 log 重定向到此 logcat 实例

```
dlutil logcat --id 4567 --output 4567.log
# # [logcat] Logcat URI: logcat://4567 Buffer Size: 4096 kbytes Output: 4567.log
DL_LOG_SINK=logcat://4567 nnexec args...
```

示例5：启动 logcat 实例后，将多个应用（nnexec）的 log 重定向到此 logcat 实例

```
dlutil logcat --id 4567
# # [logcat] Logcat URI: logcat://4567 Buffer Size: 4096 kbytes Output: -
DL_LOG_SINK=logcat://4567 nnexec args... # 实例 0
DL_LOG_SINK=logcat://4567 nnexec args... # 实例 1
DL_LOG_SINK=logcat://4567 nnexec args... # 实例 2
```

示例6：启动 logcat 实例，同时将多个应用（运行中）的 log 重定向到此 logcat 实例

--attach 可以用 --pid 代替

```
nnexec args... # 实例 0 - pid 5432
nnexec args... # 实例 1 - pid 5433
nnexec args... # 实例 2 - pid 5434
dlutil logcat --id 4567 --attach 5432,5433,5434
```

示例7：启动 logcat 实例后，将 MPI 应用中所有进程的 log 重定向到此 logcat 实例

```
dlutil logcat --id 4567 //
DL_LOG_SINK=logcat://4567 mpi run ...
```

示例8：改变目标进程相关模块的 log level，并将其 log 输出重定向到 logcat 实例

```
nnexec args... # 实例0 - pid 5432
.....
dlutil propctl --pid 5432 hc2.log_level=0 nne.log_level=0 logcat --attach 5432
# # [logcat] Logcat URI: logcat://<pid> Buffer Size: 4096 kbytes Output: -
```

示例9：Sink 为 stdout/stderr 可以开启 color print

```
DL_LOG_COLOR=1 nnexec args...
```

3 NNE 日志说明

3.1 日志等级定义

级别	含义
0	不打印任何信息，默认是此级别。
1	Fatal级别，此级别负责打印fatal信息，如果有fatal 信息则会打印。
2	Error级别，此级别负责打印error信息，通常出现异常时，可以设置此级别以观察是否有报错。
3	Warn级别，此级别负责打印告警信息，程序结果可能是正确的，但是有风险。
4	Info级别，此级别负责打印info信息，通常是一些提示信息，以及API参数检查报错。
5	Debug级别，负责打印debug信息，用于debug问题。
6	Trace级别，信息最详细，也是用于辅助debug。

3.2 日志分区定义

NNE日志分为三个分区：build、define graph和runtime，其中每个分区的日志有独立的日志等级开关。同时定义了API DUMP开关，用来管理所有API的输入和输出参数的日志，不受分区控制。

分区	含义
BUILD阶段	构建出Engine之前的所有API，如OptimizationProfile、Builder、Network、Parser、Deserizlize和绝大部分Engine的API。
DEFINE_GRAPH阶段	构图期，Engine的CreateExecutionContext*，以及和user const相关的Engine API
RUNTIME阶段	ExecutionContext的所有API

3.3 API Dump定义

用来打印public函数的API调用行为。

3.4 动态控制日志

通过dlutil工具可以动态控制日志。dlutil 工具的使用请参考章节 [2 日志控制工具 dlutil](#)。

3.4.1 日志分区开关

开关的值表示分区的日志等级，分为0~6，等级6最详细，0则不打印。

NNE.MSG_LEVEL_BUILD

用来控制build阶段日志打印。

NNE.MSG_LEVEL_DEFINE_GRAPH

用来控制构图阶段日志打印。

NNE.MSG_LEVEL_RUNTIME

用来控制runtime阶段日志打印。

3.4.2 API_DUMP 开关

NNE.API_DUMP

与日志分区开关不同，API Dump只有0表示关闭；非0表示开启，用来打印所有调用的API信息。

3.4.3 使用示例

在目标程序运行后，通过另一个dlutil进程来修改和查看，各个日志分区的等级只能独立修改。

示例1

```
dlutil propctl --pid 3588463 NNE.MSG_LEVEL_BUILD=6 NNE.MSG_LEVEL_DEFINE_GRAPH=6  
NNE.MSG_LEVEL_RUNTIME=6 NNE.API_DUMP=1
```

其中 3588463 替换成真实的进程ID；=后面的数值，对于MSG_LEVEL，即前述日志等级；API_DUMP则表示打开或者关闭。

表示打开 build/define graph/runtime 3个分区的日志，同时打开API Dump的日志。

示例2

如果一个开关后面没有=X，则表示查看这个开关当前的数值，例如：

```
dlutil propctl --pid 3588463 NNE.MSG_LEVEL_BUILD NNE.MSG_LEVEL_DEFINE_GRAPH  
NNE.MSG_LEVEL_RUNTIME=6 NNE.API_DUMP=1
```

表示查看 BUILD、DEFINE_GRAPH 两个日志分区开关的等级，并设置RUNTIME日志分区的等级且打开API_DUMP。

3.5 环境变量

表示日志分区的环境变量共有4个：NNE_MSG_LEVEL，NNE_MSG_LEVEL_BUILD，NNE_MSG_LEVEL_DEFINE_GRAPH，NNE_MSG_LEVEL_RUNTIME。

表示API DUMP的环境变量为DL_NNE_API_DUMP，与日志分区的环境变量互相独立不受影响。

3.5.1 日志分区总开关

NNE_MSG_LEVEL

NNE_MSG_LEVEL可以一次设置三个分区的值。

说明：

当分区自身和NNE_MSG_LEVEL都设置了环境变量值时，取两者中值较大的。

3.5.2 日志分区小开关

由NNE_MSG_LEVEL_BUILD, NNE_MSG_LEVEL_DEFINE_GRAPH, NNE_MSG_LEVEL_RUNTIME控制，开关的值表示分区的日志等级，分为0~6，6等级最详细，0则不打印。含义如下：

NNE_MSG_LEVEL_BUILD

用来控制build阶段日志打印。

NNE_MSG_LEVEL_DEFINE_GRAPH

用来控制构图阶段日志打印。

NNE_MSG_LEVEL_RUNTIME

用来控制runtime阶段日志打印。

3.5.3 API_DUMP 开关

NNE_API_DUMP

与日志分区开关不同，API dump只有0表示关闭，非0表示开启，用来打印所有调用的API信息。

4 HC 日志说明

4.1 HC_MSG_LEVEL 相关

HC日志输出到终端：

```
// log level
typedef enum {
    kDLLogOff  = 0,
    kDLLogFatal = 1,
    kDLLogError = 2,
    kDLLogWarn  = 3,
    kDLLogInfo  = 4,
    kDLLogDebug = 5,
    kDLLogTrace = 6,
} dLLogLevels;
```

打开所有的HC 日志：

```
# log全开
HC_MSG_LEVEL=6
# Warn及以上
HC_MSG_LEVEL=3
```

分别独立控制对应模块的日志：

```
HC_CTX_MSG_LEVEL    # CONTEXT division
HC_STR_MSG_LEVEL    # STREAM division
HC_QUE_MSG_LEVEL    # QUEUE division
HC_MEM_MSG_LEVEL    # MEMORY division
HC_GRA_MSG_LEVEL    # GRAPH division
HC_CUK_MSG_LEVEL    # CUKERNEL division
HC_EVT_MSG_LEVEL    # EVENT division
# 比如，仅开GRAPH相关的log
HC_GRA_MSG_LEVEL=6
```

4.2 HC 日志使用 dlutil 控制

使用dlutil logcat查看HC日志：

```
dlutil logcat --id 4567
HC_MSG_LEVEL=6 DL_LOG_SINK=logcat://4567 nnexec args...
```

使用dlutil logcat将HC日志保存文件：

```
dlutil logcat --id 4567 --output hc.log
HC_MSG_LEVEL=6 DL_LOG_SINK=logcat://4567 nnexec args...
```

使用dlutil propctl动态改变HC日志等级。HC支持的dlutil propctl prop：

```
lib name:
HC
key:
MSG_LEVEL          # TOP division, all hc log
CTX_MSG_LEVEL      # CONTEXT division
STR_MSG_LEVEL      # STREAM division
QUE_MSG_LEVEL      # QUEUE division
MEM_MSG_LEVEL      # MEMORY division
GRA_MSG_LEVEL      # GRAPH division
CUK_MSG_LEVEL      # CUKERNEL division
EVT_MSG_LEVEL      # EVENT division

API_DUMP           # dump api
DUMP_KLB_KSB       # dump k1b ksb
ASSERT_ON_ERROR    # hc assert_on_error
ABORT_ON_ASSERTION_FAILURE # hc abort on assertion failure
```

开启HC日志：

```
dlutil propctl --pid 5432 HC.MSG_LEVEL=6
```

动态改变HC MSG_LEVEL，并将其日志输出重定向到logcat实例：

```
dlutil propctl --pid 5432 HC.MSG_LEVEL=6 logcat --attach 5432
```

动态改变HC日志等级，并将其日志输出重定向文件：

```
dlutil propctl --pid 5432 HC.MSG_LEVEL=6 logcat --attach 5432 --output hc.log
```

4.3 DL_HC_API_DUMP

```
DL_HC_API_DUMP=1    # 在当前目录生成DL_API_DUMP_*.txt文件
```

动态修改：

```
dlutil propctl --pid 5432 HC.API_DUMP=1
```

4.4 HC_DUMP_KLBKSB

```
HC_DUMP_KLBKSB=1      # 将dump信息通过hc log输出
HC_DUMP_KLBKSB=2      # 在当前目录生成KLB_KSB_PID_*.txt文件
```

动态修改：

```
dlutil propctl --pid 5432 HC.DUMP_KLB_KSB=1
dlutil propctl --pid 5432 HC.DUMP_KLB_KSB=2
```

4.5 HC_ASSERT_ON_ERROR、HC_ABORT_ON_ASSERTION_FAILURE

```
HC_ASSERT_ON_ERROR=1      # hc assert_on_error
HC_ABORT_ON_ASSERTION_FAILURE=1 # hc abort on assertion failure
```

动态修改：

```
dlutil propctl --pid 5432 HC.ASSERT_ON_ERROR=1
dlutil propctl --pid 5432 HC.ABORT_ON_ASSERTION_FAILURE=1
```

5 HAL2 日志说明

5.1 日志等级定义

HAL2 日志级别 (log level) 分为以下几类：

日志级别	名称	描述
0	OFF	全关闭
1	FATAL	严重错误，指 HAL 发现但无法继续处理的问题。
2	ERROR	错误，指软件发现的问题，但不能自动纠正，需要上层软件或人工处理。
3	WARN	警告，指软件发现的问题，已自动纠正，程序可正常运行。
4	INFO	提示信息（没有环境变量或property时的默认级别）
5	DEBUG	debug 打印。
6	TRACE	API trace, internal trace等。

注意：

可通过环境变量或 property 控制各 level message 输出到文件或 dlutil sink。
在此基础上，Info~Fatal级别的日志一定会打印输出。

5.2 控制方式说明

HAL2的日志有两种控制方式。其中环境变量的方式需要在运行前开启，而 dlutil property的方式可以在运行过程中动态打开。

说明：

这两种控制方式仅能控制输出到文件或 dlutil sink 的消息。其中 Info - Fatal 级别一定会同时打印到 console 上，不受环境变量或 property 影响。

5.3 环境变量控制

`HAL_MSG_LEVEL` 指定输出级别（低于指定级别的 message 也会打印）。

```
# enable trace
export HAL_MSG_LEVEL=6

# default 为 0 (OFF)
export HAL_MSG_LEVEL=0
```

另外，`DL_HAL_VERBOSE` 不区分 message level，指定是否（全部）输出，以及可指定输出目的地。

```
export DL_HAL_VERBOSE=value

# default 为 0，等同于：
export DL_HAL_VERBOSE=0
```

如果没有指定输出目的地，则输出到 dlutil sink。

Value	Description	Output Destination
0	Disable Trace/Debug level。相当于 MSG_LEVEL=0	console stdout
false		
1	Enable Trace/Debug level。相当于 MSG_LEVEL=6	dlutil sink Info ~ Fatal Level 会同时打印到 console stdout
true		
stdout	Enable Trace/Debug level	console stdout
stderr	Enable Trace/Debug level	console stdout
FILE_NAME	Enable Trace/Debug level	文件 FILE_NAME

5.4 dlutil 配置

运行过程中可动态开启 HAL2 trace log：

```
# 5 = Trace
dlutil propctl --pid 5432 HAL.MSG_LEVEL=6

# default 为 info level，等同于：
dlutil propctl --pid 5432 HAL.MSG_LEVEL=4
```

环境变量优先于 dlutil 配置，当 DL_HAL_VERBOSE 设为 0（false）时，相当于设置的 dlutil hal2.log_level 为 2。

5.5 控制方式总结

env DL_HAL_VERBOSE	dlutil 配置	结果 (Trace level)	结果 (Info ~ Fatal level)
0/false	log_level	dlutil sink, according to "log_level"	dlutil sink + console output

env DL_HAL_VERBOSE	dlutil 配置	结果 (Trace level)	结果 (Info ~ Fatal level)
true	无效 (已打开所有 level)	dlutil sink	dlutil sink + console output
stdout	无效 (不再去往dlutil sink)	console output	console output
stderr	无效 (不再去往 dlutil sink)	console output	console output
FILE_NAME	无效 (不再去往 dlutil sink)	file output	FILE_NAME + console output

6 DLEOL 动态控制日志说明

DLEOL可通过dlutil工具动态控制日志。

6.1 DLEOL 日志配置

DLEOL.key = value，其中dleol定义key可选字段：

```
MSG_LEVEL
GRAPH_MSG_LEVEL
OP_MSG_LEVEL
PASS_MSG_LEVEL
PATTERN_MSG_LEVEL
TENSOR_MSG_LEVEL
COMMON_FUNC_MSG_LEVEL
```

- MSG_LEVEL 控制其余所有 XXX_MSG_LEVEL。
- key 之间相互独立，即各个日志分区的等级独立修改。
- value 即当前division的日志级别，有效范围 0 ~ 5（5表示最高等级，小于0取0，大于5取5）控制log粒度。
value 默认值为0，表示LogOff。

6.2 DLEOL 日志默认环境变量

Properties的 value 可通过 DLEOL_MSG_LEVEL 设置初始值：

```
// terminal 1
export DLEOL_MSG_LEVEL=5
dleol_test --gtest_filter=xxx_case # pid = 1234

// terminal 2
dlutil propctl --pid 1234 DLEOL.GRAPH_MSG_LEVEL \
    DLEOL.OP_MSG_LEVEL \
    ...
// then will print: DLEOL.GRAPH_MSG_LEVEL=5, DLEOL.OP_MSG_LEVEL=5 ....
```

6.3 dlutil 动态控制 DLEOL 日志

动态更新（dleol 进程1234）section的日志等级。

```
dlutil propctl --pid 1234 DLEOL.GRAPH_MSG_LEVEL=5 \
    DLEOL.OP_MSG_LEVEL=5 \
    ...
# log 默认打印到终端，可使用dlutil logcat重定向
```

查询（dleol 进程1234）当前section的日志等级。

```
dlutil propctl --pid 1234 DLEOL.GRAPH_MSG_LEVEL \  
DLEOL.OP_MSG_LEVEL \  
...
```

设置日志等级 (log level) 并使用logcat实例。

```
dleo1_test args... # 实例0 - pid 5432  
.....  
dlutil propctl --pid 5432 DLEOL.GRAPH_MSG_LEVEL=5 \  
DLEOL.OP_MSG_LEVEL=5 \  
#DLEOL.xxx_SEC_LEVEL=5 \  
logcat --output dleo1_5432.log --attach 5432
```