

**Ascend 310**  
**V100R001**

# Mind Studio API 参考

文档版本	01
发布日期	2019-02-25



版权所有 © 华为技术有限公司 2019。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

## 华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<http://www.huawei.com>

客户服务邮箱：[support@huawei.com](mailto:support@huawei.com)

客户服务电话：4008302118

# 目 录

<b>1 IDE Daemon 模块相关接口.....</b>	<b>1</b>
1.1 创建远程文件句柄.....	1
1.2 远程写文件.....	2
1.3 关闭文件.....	2
1.4 调用示例.....	3
<b>2 Log 模块相关接口.....</b>	<b>4</b>
2.1 日志初始化接口.....	4
2.2 打印 error 级别日志.....	5
2.3 打印 warning 级别日志.....	7
2.4 打印 info 级别日志.....	8
2.5 打印 debug 级别日志.....	8
2.6 打印 event 级别日志.....	9
2.7 获取日志级别.....	9

# 1 IDE Daemon 模块相关接口

IDE Daemon模块提供接口给HiAI Engine模块调用，通过该接口可以实现将DVPP预处理结果数据发送到Mind Studio侧，Mind Studio将结果数据写到文件中。

[1.1 创建远程文件句柄](#)

[1.2 远程写文件](#)

[1.3 关闭文件](#)

[1.4 调用示例](#)

## 1.1 创建远程文件句柄

### 函数原型

**IDE\_SESSION ideOpenFile(connInfo\_t \*connInfo, const char \*fileName)**

### 函数功能

创建远程文件句柄。

### 参数说明

参数	输入/输出	说明
<b>connInfo</b>	输入	连接信息。
<b>fileName</b>	输入	保存的文件路径，包含文件名。

### 返回值

- NULL：打开远程文件失败。
- 非NULL：打开远程文件成功。

## 1.2 远程写文件

### 函数原型

**ideError\_t ideWriteFile(IDE\_SESSION session, const void \*data, uint32\_t len)**

### 函数功能

远程写文件。

### 参数说明

参数	输入/输出	说明
<b>session</b>	输入	会话句柄。
<b>data</b>	输入	需要写入文件的数据。
<b>len</b>	输入	长度。

### 返回值

- IDE\_DAEMON\_NONE\_ERROR：写文件成功。
- IDE\_DAEMON\_UNKNOW\_ERROR：写文件失败。

## 1.3 关闭文件

### 函数原型

**ideError\_t ideCloseFile(IDE\_SESSION session)**

### 函数功能

关闭文件。

### 参数说明

参数	输入/输出	说明
<b>session</b>	输入	会话句柄。

### 返回值

- IDE\_DAEMON\_NONE\_ERROR：关闭本次会话成功，结束写文件操作。
- IDE\_DAEMON\_UNKNOW\_ERROR：关闭本次会话失败，结束写文件操作。

## 1.4 调用示例

IDE Daemon模块相关接口的调用顺序依次为：创建远程文件句柄、远程写文件、关闭文件。

接口调用示例如下：

```
//包含头文件ide_daemon_api.h;
//代码仓的具体路径为: inc/toolchain/ide_daemon_api.h
#include"ide_daemon_api"

#define PORT (22118)
char* str = "Hello worlds!";

connInfo_t connInfo = {0};
memcpy_s(connInfo.ip, IDE_DAEMON_IP_LEN, "127.0.0.1", strlen("127.0.0.1"));
connInfo.port = PORT;
connInfo.deviceID = 0;

IDE_SESSION handle = ideOpenFile(&connInfo, "/tmp/test.bin");

ideError_t ret = ideWriteFile(handle, str, strlen(str));

ideCloseFile(handle);
```

# 2 Log 模块相关接口

Log模块提供接口给用户态应用模块（例如：接口入参处指定的模块）调用，实现日志内容统一收集。

[2.1 日志初始化接口](#)

[2.2 打印error级别日志](#)

[2.3 打印warning级别日志](#)

[2.4 打印info级别日志](#)

[2.5 打印debug级别日志](#)

[2.6 打印event级别日志](#)

[2.7 获取日志级别](#)

## 2.1 日志初始化接口

### 函数原型

```
void dlog_init();
```

### 函数功能

初始化日志以获取本地配置文件。多次调用也只执行一次。

### 参数说明

无。

### 返回值

无。

### 调用示例

```
//调用接口前，需要先导入动态库libslog.so  
dlog_init();
```

## 2.2 打印 error 级别日志

### 函数原型

```
void dlog_error(int module_id, const char *fmt, ...);
```

### 函数功能

打印error级别日志。



## 参数说明

表 2-1 参数说明

参数	输入/输出	说明
<b>module_id</b>	输入	<pre>enum { DLOG = 0, // Dlog SLOG, // Slog IDEDD, // IDE daemon device IDEDH, // IDE daemon host LOGAGTH, // log agent host HCCL, // HCCL FMK, // Framework HIAIENGINE, // Matrix DVPP, // DVPP RUNTIME, // Runtime CCE, // CCE #ifdef OS_TYPE == LINUX HDC, // HDC #else HDCL, // HDCL windows has a def with the same name HDC, so change HDC to HDCL #endif DRV, // Driver MDCCONTROL, // Mdc control MDCFUSION, // Mdc fusion MDCLOCATION, // Mdc location MDCPERCEPTION, // Mdc perception MDCMOP, MDCFSM, MDCCOMMON, MDCMONITOR, MDCBSWP, // MDC basesoftware platform MDCDEFAULT, // MDC UNDEFINE MDCSC, // MDC spatial cognition MDCBP, MDCTF, MLL, DEVMM, // Dlog memory managent KERNEL, // Kernel MDCSMCMD, // sm_control_cmd }</pre>

参数	输入/输出	说明
		MDCSCREEN, // parking_spot_screen LIBMEDIA, // Libmedia CCECPU, // ai cpu ASCENDDK, // AscendDK ROS, // ROS INVALID_MODULE_ID };
fmt	输入	要打印的内容。

### 返回值

无。

### 调用示例

```
//调用接口前，需要先导入动态库libslog.so  
dlog_error(SLOG, "malloc failed in sklogd");  
char *msg= "hello";  
dlog_error(KERNEL, "message is %s", msg);
```

## 2.3 打印 warning 级别日志

### 函数原型

**void dlog\_warn(int module\_id, const char \*fmt, ...);**

### 函数功能

打印warn级别日志。

### 参数说明

参数	输入/输出	说明
module_id	输入	请参见表2-1。
fmt	输入	要打印的内容。

### 返回值

无。

### 调用示例

```
//调用接口前，需要先导入动态库libslog.so  
dlog_warn(SLOG, "malloc failed in sklogd");
```

```
char *msg= "hello";  
dlog_warn(KERNEL, "message is %s", msg);
```

## 2.4 打印 info 级别日志

### 函数原型

```
void dlog_info(int module_id, const char *fmt, ...);
```

### 函数功能

打印info级别日志。

### 参数说明

参数	输入/输出	说明
module_id	输入	请参见 <a href="#">表2-1</a> 。
fmt	输入	要打印的内容。

### 返回值

无。

### 调用示例

```
//调用接口前，需要先导入动态库libslog.so  
dlog_info(SLOG, "malloc failed in sklogd");  
char *msg= "hello";  
dlog_info(KERNEL, "message is %s", msg);
```

## 2.5 打印 debug 级别日志

### 函数原型

```
void dlog_debug(int module_id, const char *fmt, ...);
```

### 函数功能

打印debug级别日志。

### 参数说明

参数	输入/输出	说明
module_id	输入	请参见 <a href="#">表2-1</a> 。
fmt	输入	要打印的内容。

## 返回值

无。

## 调用示例

```
//调用接口前，需要先导入动态库libslog.so  
dlog_debug(SLOG, "malloc failed in sklogd");  
char *msg= "hello";  
dlog_debug(KERNEL, "message is %s", msg);
```

# 2.6 打印 event 级别日志

## 函数原型

```
void dlog_event(int module_id, const char *fmt, ...);
```

## 函数功能

打印event级别日志。

## 参数说明

参数	输入/输出	说明
module_id	输入	请参见 <a href="#">表2-1</a> 。
fmt	输入	要打印的内容。

## 返回值

无。

## 调用示例

```
//调用接口前，需要先导入动态库libslog.so  
dlog_event(SLOG, "malloc failed in sklogd");  
char *msg= "hello";  
dlog_event(KERNEL, "message is %s", msg);
```

# 2.7 获取日志级别

## 函数原型

```
int dlog_getlevel(int module_id, int* enable_event);
```

## 函数功能

获取日志级别。调用打印日志接口时，可先获取当前日志级别，以判断是否能成功打印。

## 参数说明

参数	输入/输出	说明
<b>module_id</b>	输入	请参见表2-1。
<b>enable_event</b>	输入	1: enable 0: disable

## 返回值

返回该module的日志级别

```
DLOG_NULL    (-1) // don't output log
DLOG_ERROR   0    // error conditions
DLOG_WARN    1    // warning conditions
DLOG_INFO    2    // informational
DLOG_DEBUG   3    // debug-level messages
```

## 调用示例

```
//调用接口前，需要先导入动态库libslog.so
int enable_event;
int sloglevel = dlog_getlevel(SLOG, &enable_event);
```