ZSTACK

# 框架

|  |  |
| --- | --- |
| APP |  |
| UI |  |
| ALGO |  |
| VS |  |
| DEFAULT |  |
| NET |  |
| MISC |  |
|  |  |

# LOG

## Usage

<app> -d “default:3,app:3,ui:3,misc:3,net:3,avmips:3,algo:3”

|  |  |
| --- | --- |
| 0 | ERROR |
| 1 | WARNING |
| 2 | USER |
| 3 | DEBUG/Developer |

限制条件：

1. 不在0-3范围内的值会自动纠正为0或3
2. 如default关键字的长度不能超过31个
3. 如果没有指定调试信息，默认将”default:3”传入
4. 如果源文件中没有指定LOG\_MODULE，就分配给default模块

我之前有个想法，就是配置中不指定level，只要是user级别的都打印出来，而如果在调试配置中出现的，肯定是调试级别的。但是如果系统很大，LOG很多的情况下，我只想看某个模块，这个USER强制打印就不合理了。

## LEVEL

|  |  |
| --- | --- |
| ERROR | 系统会因此无法运行 |
| WARNING | 可以忽略，可以处理，系统不会因为缺胳膊少腿运行不了 |
| USER | 用户需要知道的 LOG，而非开发人员 |
| DEBUG | 开发者级别：这个模块的输入输出，在干什么 |

之前开发者级别有两个级别，分别是文件级和函数级，但是这两个区分起来过于繁琐，就统一为开发者级别。还有一个最主要的原因，因为模块的加入，我希望用两个bit代表每个module的调试级别。如果有5个级别，需要3个bit，不好对齐。

## MODULE

我本来想让每个文件都有LOG开关，想看哪个文件就通过参数指定看：

<FILE\_ID>:<LEVEL> 举例：”0:2,65:3,1:4”

因此还设计了一个工具，为每个源文件生成文件ID，放到公共头文件中，如下所示：

#define FILE\_1 (1)

#define FILE\_2 (2)

然后在源文件定义如下：

#define LOG\_FILE FILE\_1

这样我在使用log(level,…)的时候，就会自动变成log(LOG\_FILE, level, …)，但是如果zstacklib被编译成库，这些LOG\_FILE就会被固定下来，而如果其他文件发生变化，这个头文件需要被更新。总的来说，很麻烦。而我的设计原则是一眼能看懂，**简单的复用性才强**。

如果真的有BUG，Level提升到Developer等级，所有涉及的模块，相关LOG都会被观察到，这对于一个小的系统，也没啥问题。如果是AVMIPS级别的，我可能只想看Video相关的，这个时候需要在所有Video相关的文件中定义一个LOG\_MODULE的宏了。但是我的Application都是比较小的，涉及的模块非常少，可以暂时不定义，用DEFAULT即可。即当文件中没有定义LOG\_MODULE的时候：

#ifndef LOG\_MODULE

#define LOG\_MODULE LOG\_MODULE\_DEFAULT

#endif

综上，LOG应该区分模块，而非文件。模块与系统设计息息相关。参考[框架](#_框架)。

## API

extern void log\_init(char \*config);

#define log(lvl, argv, ...)

LOG的API是log(level, …)，不能再复杂了。参数level是必须的，这对每一条任务定性。

# API

按照字母顺序排列。

## log\_init

int log\_init(char \*config);

## log

void log(int level, …);