**Os-SET底软平台和**

**Open-SET开放运行平台**

**设计和使用说明**

星 级：

版本/修订：V1.2

编 制

审 核

|  |
| --- |
| 2021-XX-XX 文件发布日期 2021-XX-XX文件生效日期 |
|  |

**文档修订历史**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **版本** | **作者** | **版本变化对象** | **变化内容描述** | **审核人** | **批准人** | **修订日期** |
| V1.0 | 孙加伟 |  | 1、初始版本 |  |  | 2021/11/30 |
| V1.1 | 孙加伟 |  | 1. 支持模块化架构 2. O(pen)-SET初版 |  |  | 2022/7/4 |
| V1.2 | 孙加伟 |  | 1、对接SSET平台 |  |  | 2022/9/2 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

目 录

[1 引言 1](#_Toc113133975)

[1.1 编写目的 1](#_Toc113133976)

[1.2 背景 1](#_Toc113133977)

[2 OSET通用平台介绍 1](#_Toc113133978)

[3 Os-SET底软平台 2](#_Toc113133979)

[3.1 平台结构 2](#_Toc113133980)

[3.2 编译 2](#_Toc113133981)

[3.2.1 编译方法 2](#_Toc113133982)

[3.2.2 动态库说明 3](#_Toc113133983)

[3.3 调用方法 3](#_Toc113133984)

[3.4 环境要求 4](#_Toc113133985)

[3.5 安装包版本说明 4](#_Toc113133986)

[3.5.1 安装和卸载 4](#_Toc113133987)

[3.6 单元测试框架 5](#_Toc113133988)

[4 Open-SET开放平台 9](#_Toc113133989)

[4.1 平台结构 9](#_Toc113133990)

[4.2 部署方式 10](#_Toc113133991)

[4.3 对接SSET平台 11](#_Toc113133992)

[4.4 通用开发接口 11](#_Toc113133993)

[4.5 Esl远程调用库 12](#_Toc113133994)

[4.6 OM APP范例 12](#_Toc113133995)

[4.6.1 在线模块查看 14](#_Toc113133996)

[4.6.2 OM模块加载 15](#_Toc113133997)

[4.6.3 OM模块卸载 15](#_Toc113133998)

[4.6.4 OM模块重载 15](#_Toc113133999)

# 引言

## 编写目的

本文档的目的是从OSET平台需求出发，设计V1.1版本的整体方案，可分解出进行概要设计和详细设计的模块和子模块，定义各模块和子模块的功能，模块和子模块的关系，满足OSET V1.1版本的系统需求和软件设计需求。

本文档用于指导OSET V1.1版本的概要设计和详细设计文档，以及后续的集成测试。

## 背景

软件系统名称：OSET通用平台

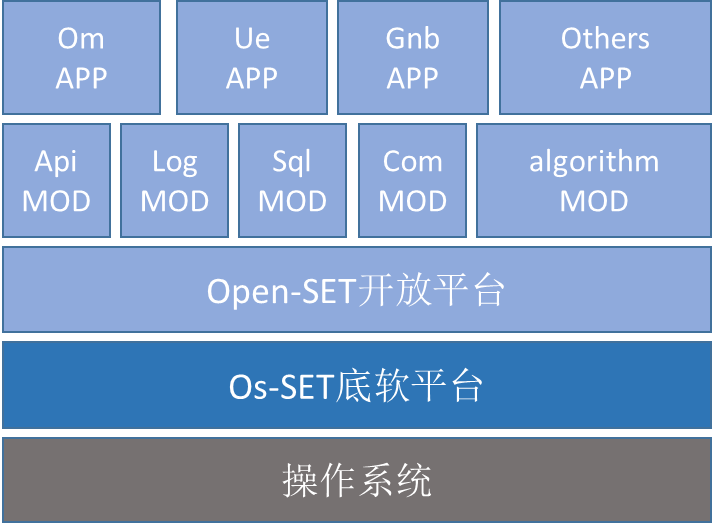
软件开发者：SSET协议平台组

开发语言：C/C++、Python等

硬件平台：X86服务器

软件平台：Linux

# OSET通用平台介绍



为了实现各种不同体制协议的仿真和模拟，实现协议栈的模块化加载和迭代，减少操作系统对上层业务的影响，提高软件开发的统一性和可重用性，开发了OSET通用平台。

整个通用平台由Os-SET底软平台和Open-SET开放运行平台两部分构成。Os-SET底层平台提供各种系统底层接口，实现底层系统和上层业务的隔离。Open-SET平台是依托于Os-SET底软平台开发的一款开放运行平台，整个平台全部采用模块化架构，支持模块的实时加载卸载，支持运行业务的实时弹缩，提供esl远程调用库，支持python、java的跨语言直接调用，尽量避免使用中间件。此外整个平台都围绕开放原则开发，定义了简单通用的函数接口和模板，方便第三方开发和迭代。

# Os-SET底软平台

## 平台结构



Core目录为核心目录，核心功能均在此目录开发，全部采用动态库提供给第三方。

平台目录结构如下

├─cmake ***Cmake编译引用库***

├─core ***OSET底软目录***

│ ├─base ***OSET base库***

│ ├─esl  ***Esl远程调用库***

│ ├─mods ***API模块库***

├─doc ***说明文档***

├─tests ***单元测试***

├─tools ***常用工具***

├─CMakeLists.txt  ***cmake编译***

## 编译

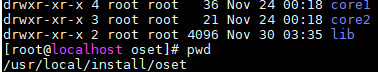
### 编译方法

整个OSET工程可以独立安装，也可以作为外部工程被第三方工程cmake引用，视情况而定。独立安装时，编译使用Cmake外部编译，保持目录清洁。

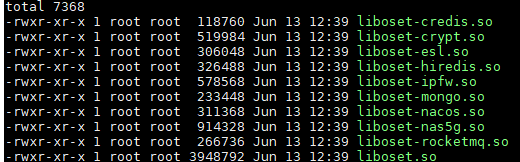
1. ***cmake .. && make && make install***

*说明：cmake .. -DCMAKE\_INSTALL\_PREFIX=路径地址*

*//若不指定路径，编译的文件会默认放在/usr/local/install/oset，建议使用默认目录。*



### 动态库说明



***Liboset.so oset base动态库（日志、IO、线程、进程、内存管理、strings等）***

***Liboset-xxx.so oset module接口库***

## 调用方法

默认不指定安装路径，安装在系统盘*/usr/local/install/oset*目录下。

* cmake工程：
* Base库引用：

*FindOSET.cmake*

*include\_directories(${OSET\_INCLUDE\_DIRS})*

*link\_directories(${OSET\_LIBRARIES})*

* module库引用：

*FindOSET-XXX.cmake*

*include\_directories(${OSET\_XXX\_INCLUDE\_DIRS })*

*link\_directories(${OSET\_XXX\_LIBRARIES})*

* *源码调用：*

头文件引用+接口函数调用

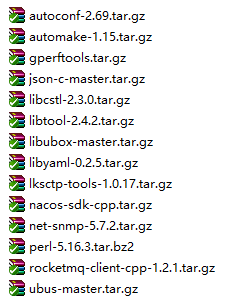
## 环境要求

1. 支持系统

Ubuntu20.04 64位（推荐）

CentOS 7.4 64位

1. 底软依赖库



执行SSET/tools/ sset\_os\_oset\_depends.sh脚本，自动安装依赖。

## 安装包版本说明

* Ubuntu20.04



* Centos7.4



***软件版本号（major.minor.release）-辅助版本号 (.api)***

### 安装和卸载



**安装新版本之前必须卸载旧版本！！！**

#### Ubuntu20.04

查看安装包信息 dpkg -s oset

安装 dpkg -i oset\_ *X.X.X-X* \_amd64.deb

卸载 dpkg --purge oset

查看安装情况 dpkg –l | grep oset

帮助命令使用 dpkg --help

#### Centos7.6

查看安装包信息 rpm -qpi oset- *X.X.X-X*.x86\_64.rpm

安装 rpm -ivh --nodeps oset- *X.X.X-X*.x86\_64.rpm （忽略依赖问题）

卸载 rpm -evh oset

查看安装情况 rpm –qa | grep oset

帮助命令使用 rpm --help

## 单元测试框架

蓝色部分为底软初始化和相关配置；红色部分为abts框架。

*int main(int argc, const char \*const argv[])*

*{*

*int rv, i, opt;*

*oset\_getopt\_t options;*

*struct {*

*char \*log\_level;*

*char \*domain\_mask;*

*} optarg;*

*const char \*argv\_out[argc+2]; /\* '-e error' is always added \*/*

*abts\_suite \*suite = NULL;*

*oset\_pkbuf\_config\_t config;*

*rv = abts\_main(argc, argv, argv\_out); //UT框架入参初始化*

*if (rv != OSET\_OK) return rv;*

*memset(&optarg, 0, sizeof(optarg));*

*oset\_getopt\_init(&options, (char\*\*)argv\_out);*

*while ((opt = oset\_getopt(&options, "e:m:")) != -1) {*

*switch (opt) {*

*case 'e':*

*optarg.log\_level = options.optarg;*

*break;*

*case 'm':*

*optarg.domain\_mask = options.optarg;*

*break;*

*case '?':*

*default:*

*fprintf(stderr, "%s: should not be reached\n", OSET\_FUNC);*

*return OSET\_ERROR;*

*}*

*}*

*oset\_core\_initialize();*

*oset\_pkbuf\_default\_init(&config);*

*oset\_pkbuf\_default\_create(&config);*

*atexit(terminate);*

*rv = oset\_log\_config\_domain(optarg.domain\_mask, optarg.log\_level);*

*if (rv != OSET\_OK) return rv;*

*for (i = 0; alltests[i].func; i++) //加载所有测试用例函数*

*suite = alltests[i].func(suite);*

*return abts\_report(suite); //生成单元测试报告*

*}*

*const struct testlist {*

*abts\_suite \*(\*func)(abts\_suite \*suite);*

*} alltests[] = { //加载测试用例函数*

*{test\_list},*

*{test\_pool},*

*{test\_strings},*

*{test\_time},*

*{test\_conv},*

*{test\_log},*

*{test\_pkbuf},*

*{test\_memory},*

*{test\_rbtree},*

*{test\_timer},*

*{test\_thread},*

*{test\_socket},*

*{test\_queue},*

*{test\_poll},*

*{test\_tlv},*

*{test\_hash},*

*{test\_uuid},*

*{NULL},*

*};*

*abts\_suite \*test\_list(abts\_suite \*suite)// 加载测试用例case*

*{*

*suite = ADD\_SUITE(suite)*

*abts\_run\_test(suite, list\_test1, NULL);*

*abts\_run\_test(suite, list\_test2, NULL);*

*abts\_run\_test(suite, list\_test3, NULL);*

*abts\_run\_test(suite, list\_test4, NULL);*

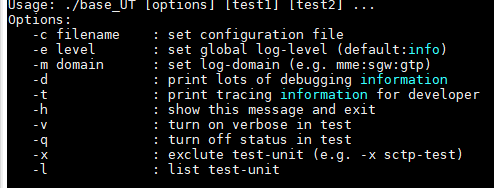
*abts\_run\_test(suite, list\_test5, NULL);*

*abts\_run\_test(suite, list\_test6, NULL);*

*return suite;*

*}*

**1、使用说明：**



*-l 显示测试用例函数*

*-x 测试指定文件之外的其余用例*

*-v 打出错case详细信息（默认打开）*

*-q 关闭进度显示*

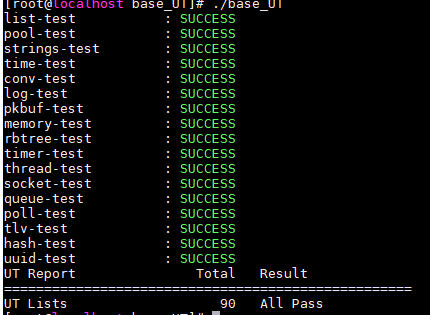
*-t 默认打开trace级别日志*

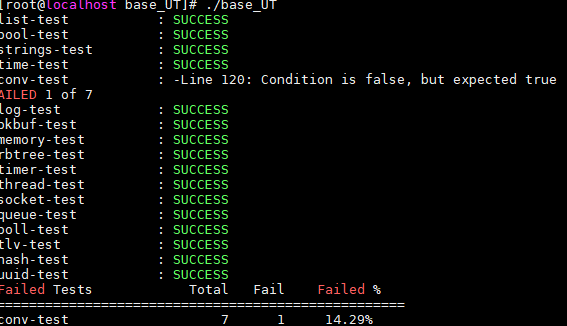
*-d 默认打开debug级别日志*

*-m log模块 -e debug （设置部分模块打印日志级别，-m –e配合使用）*

***整个单元测试框架默认使用error级别日志打印。***

***2、运行结果***





1. **断言集说明**

* 判断整形相等或不等

ABTS\_INT\_EQUAL / ABTS\_INT\_NEQUAL

* 判断size相等（size\_t类型）

ABTS\_SIZE\_EQUAL

* 判断字符串相等或不等

ABTS\_STR\_EQUAL / ABTS\_ STR \_NEQUAL

* 判断指针相等或不为空

ABTS\_ PTR \_EQUAL / ABTS\_PTR\_NOTNULL

* 判断条件为真

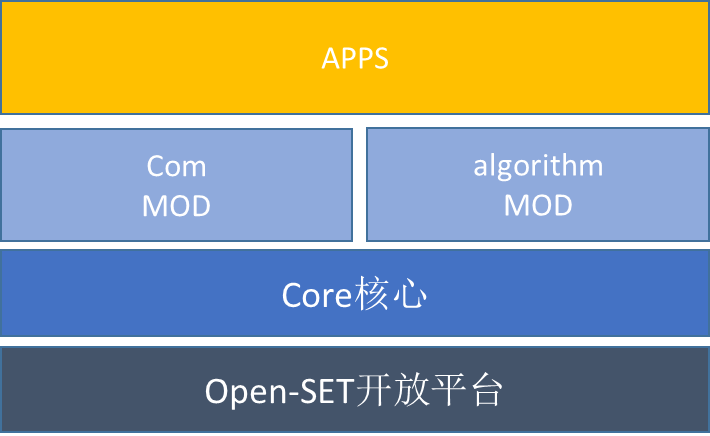
ABTS\_TRUE

* 出错及断言

ABTS\_FAIL / ABTS\_ASSERT

# Open-SET开放平台

## 平台结构



Open-SET平台由三大部分构成：

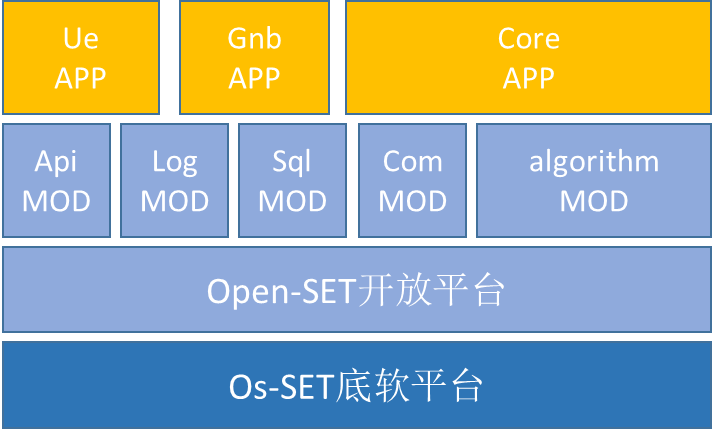
* Core核心：core核心负责整个平台的running，是整个平台最重要的运行框架，负责Mod和App的管理和调度。
* Mod：Module是平台的中间层，分为common mod和algorithm mod两大类，common模块提供各种常用的API接口，供给上层业务调用，是Os-SET的实例化，如日志模块、CLI模块、Redis模块等；algorithm模块是SSET常用的算法模块，提供各种算法API接口。
* App：APP是平台最上层的业务，由业务逻辑和Mod API组合而成。

整个平台都是模块化设计，MOD和APP两大核心可以任意加载、卸载、重载，非常灵活。平台的运行依托于Os-SET底软平台，否则无法运行。

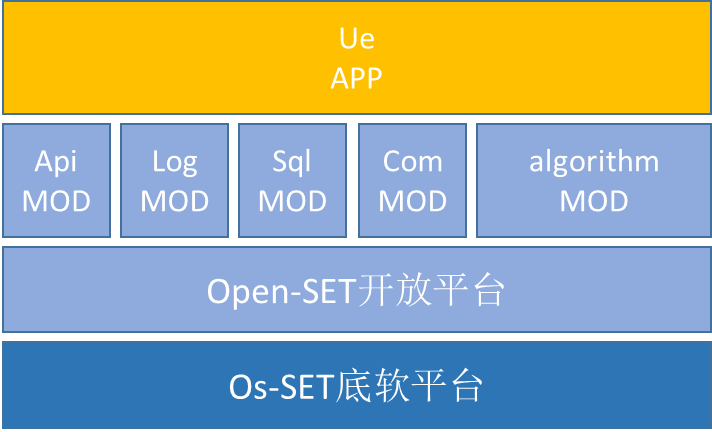
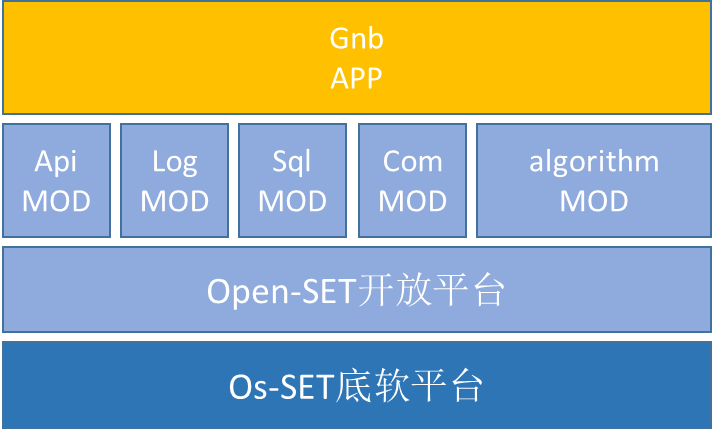
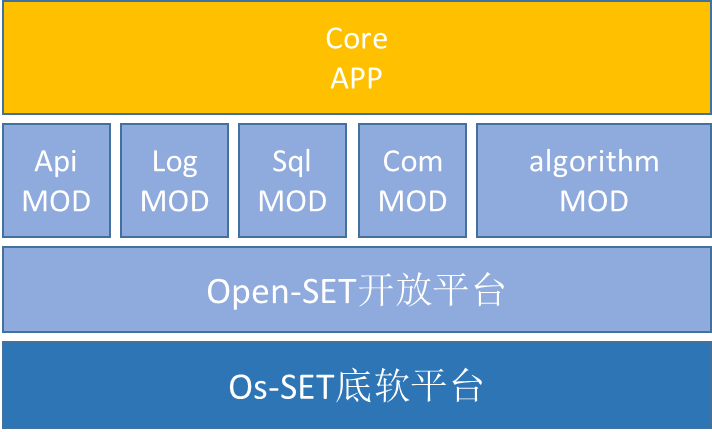
## 部署方式

部署的方式取决于业务，而不取决于平台。整个框架非常灵活，取决于应用场景。

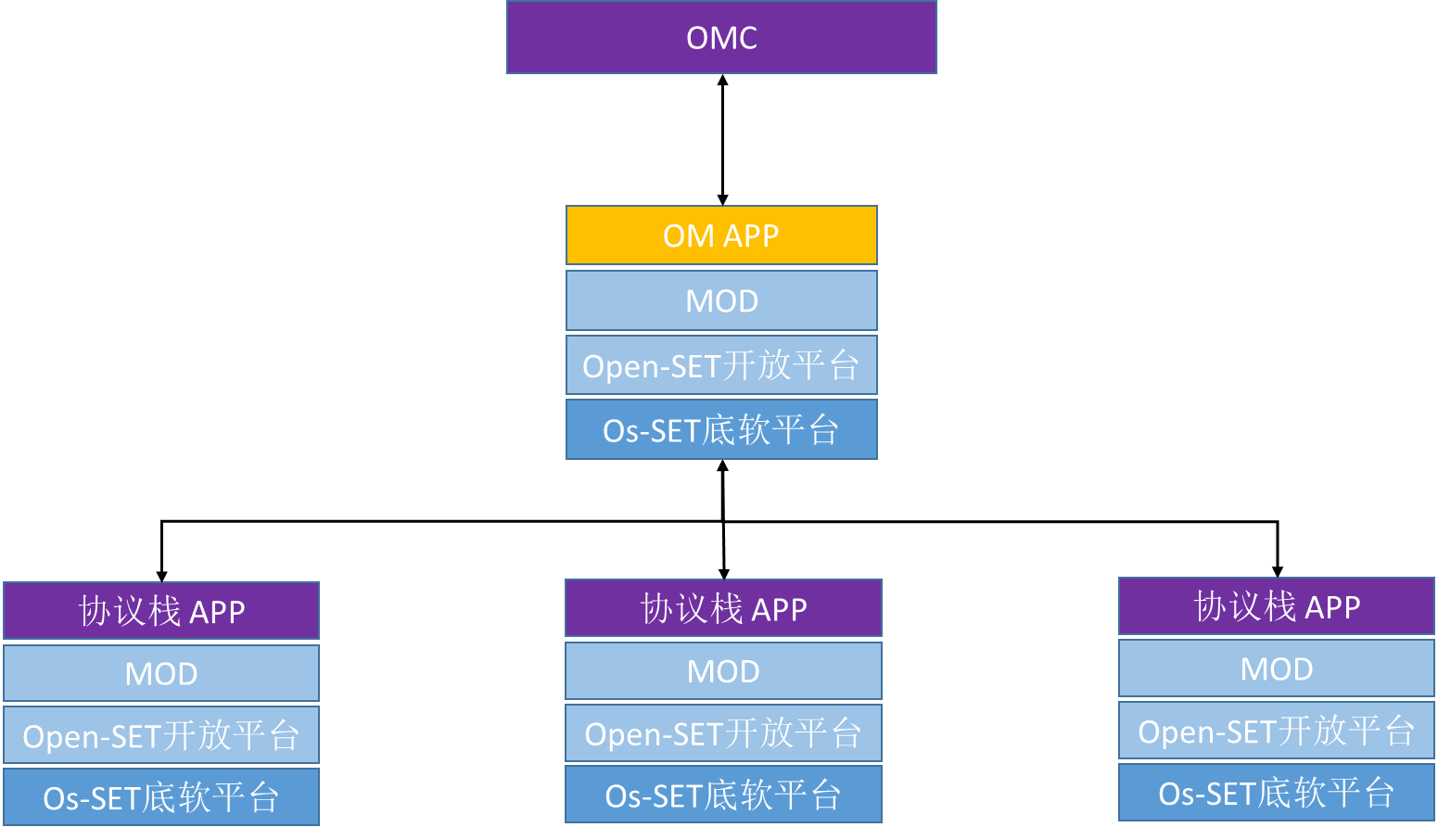
* 单机部署



* 分布式部署

## 对接SSET平台



SSET平台作为分布式系统，协议栈业务的接入通过OM模块实现。

## 通用开发接口

通用开发接口是MOD和APP模块化的关键，也是第三方开发需要遵守的规范，整个接口非常简单和规范。

*OSET\_MODULE\_LOAD\_FUNCTION(mod\_xxx\_load)*

*{*

*加载xxx模块*

*}*

*OSET\_MODULE\_SHUTDOWN\_FUNCTION(mod\_xxx\_shutdown)*

*{*

*卸载xxx模块*

*}*

*OSET\_MODULE\_RUNTIME\_FUNCTION(mod\_xxx\_runtime)*

*{*

*实时运行xxx模块*

*}*

*OSET\_MODULE\_DEFINITION(libmod\_xxx,*

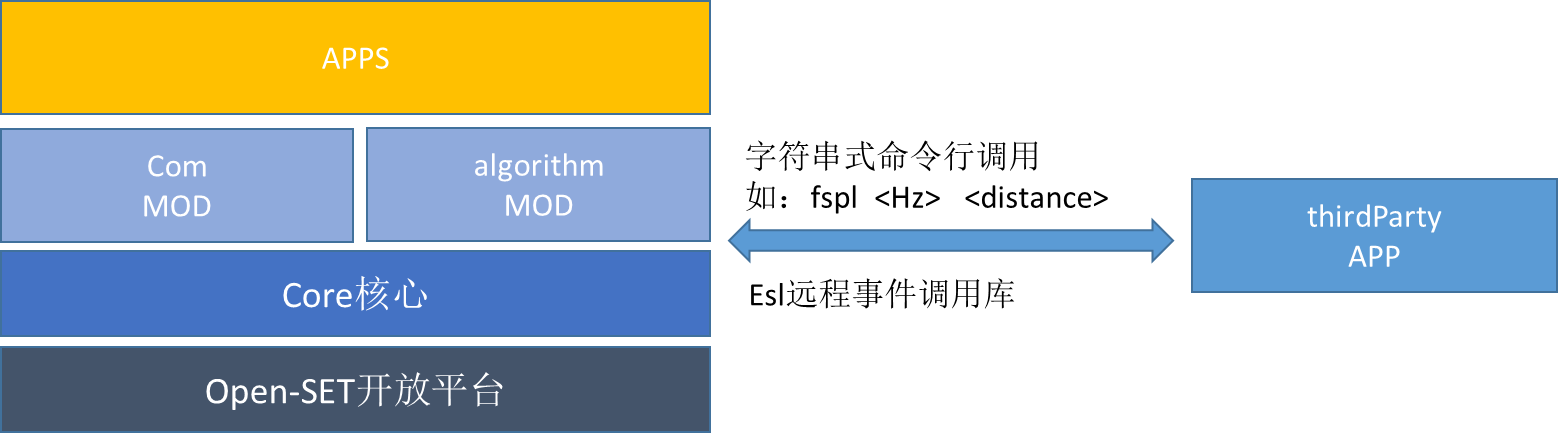
*mod\_xxx\_load,*

*mod\_xxx\_shutdown,*

*mod\_xxx\_runtime); //注册宏*

接入Open-SET平台只需要以上四个函数，非常简单。

## Esl远程调用库

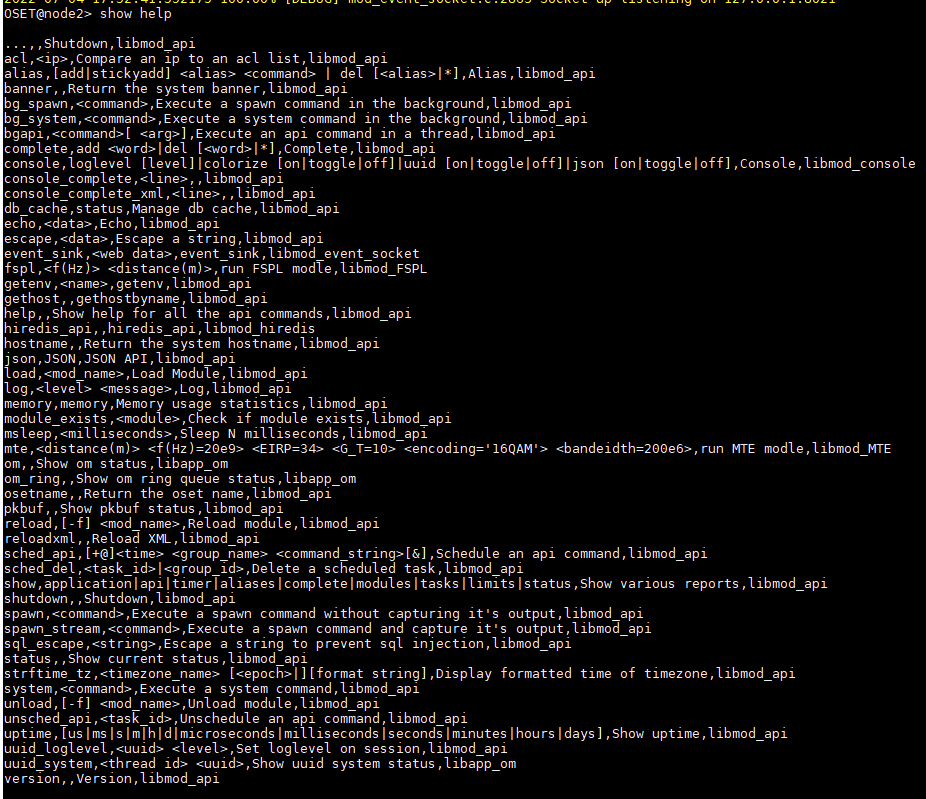


远程调用库是为了方便分布式场景下的远程调用，可以远程调用Open-SET平台上加载的任意模块API。使用方法和本地方法一致，都是通过命令行方的字符串命令调用，至于底层实现方法，用户不需要关心。

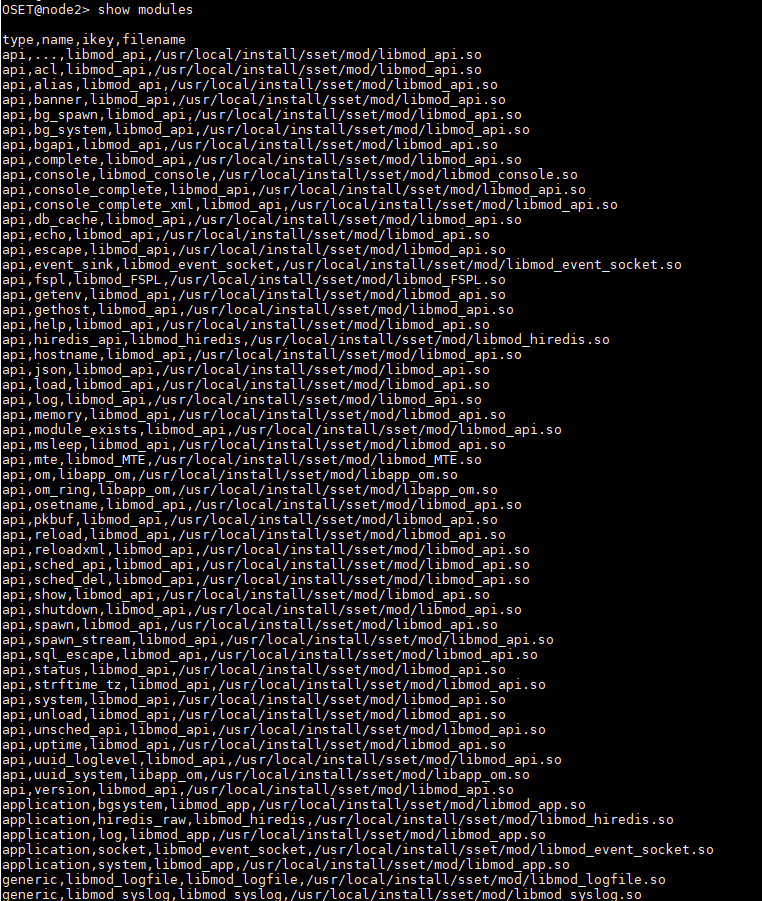
## OM APP范例

整个协议栈的OM网管模块就是基于Open-SET平台开发，新模块的编码可以参考OM。

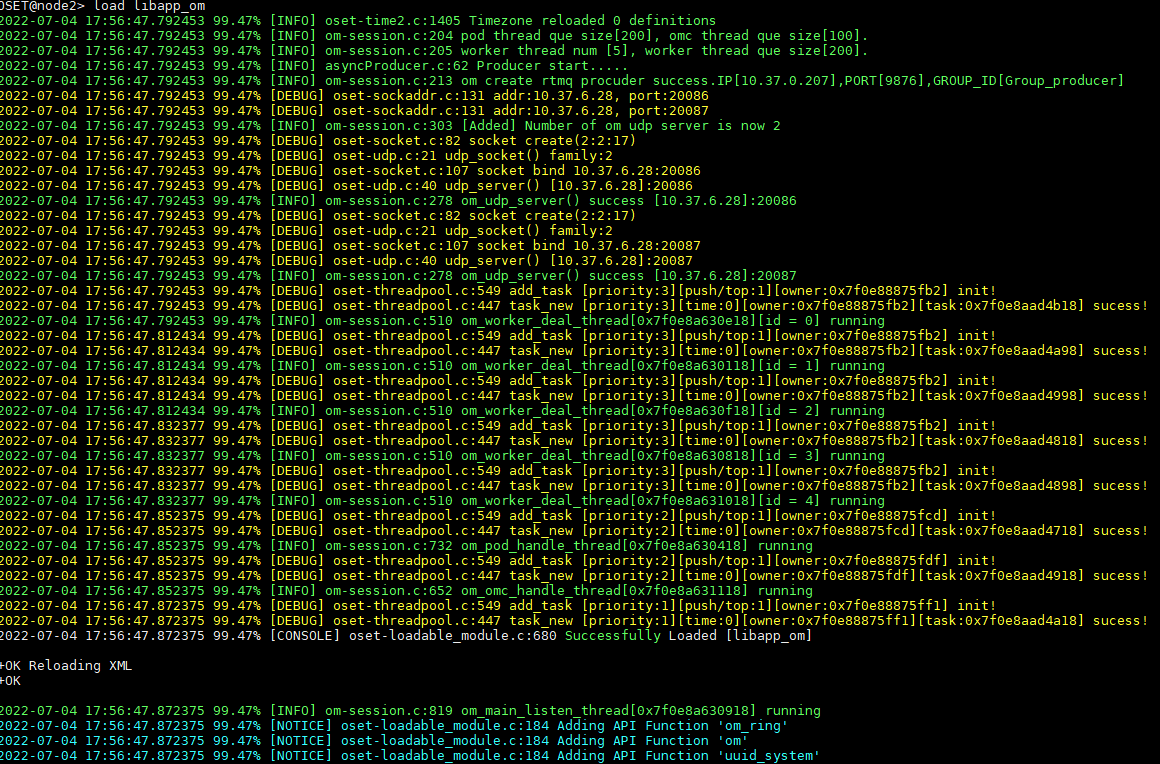
平台提供本地和远程两种CLI交互方式，通过show help命令可以查看使用方法。



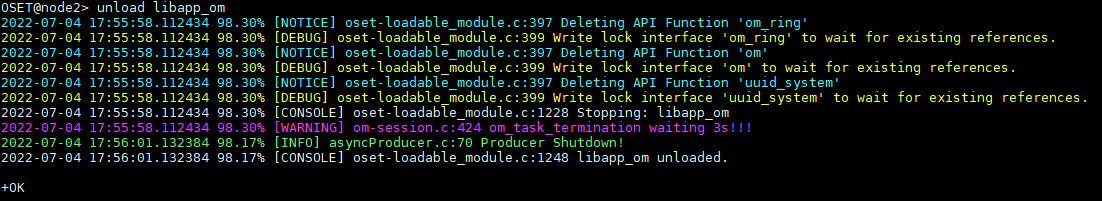
### 在线模块查看



### OM模块加载



### OM模块卸载



### OM模块重载

重载命令就是卸载+加载命令的合集，reload +模块名字。