视频监控流式存储方案使用说明

（V 0.2）

文件编号	版本号	拟制人/修改人	拟制/修改日期	更改理由	主要更改内容（写要点即可）
	V0.1	吴贻刚	2015-9-1	无	新建
	V0.2	吴贻刚	2015-9-14	添加缩减第三方库体积建议	添加针对嵌入式设备编译第三方库时，体积缩小建议
注 1：每次更改归档文件时，需填写此表。 注 2：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。					

目录

一、概述

本方案是基于 OSS 为视频监控行业定制的一套解决方案，在使用本方案时，也不影响使用 OSS 的 API.

本解决方案 SDK 包括两部分：设备端 SDK+示例以及服务端 SDK+示例。客户端如何从 OSS 拉取视频流的示例包含在设备端 demo 中。

本版本依赖 OSS_C_SDk 库。

设备端 SDK+示例文件：oss_camera_c_sdk.zip ————C 语言版本

服务端 SDK+示例文件:

oss_camera_server_c_sdk.zip ————C 语言版本

oss_camera_server_java_sdk.zip ————Java 语言版本

本文的着重介绍 C 语言版本的 SDK 使用说明，Java 版本的使用说明见：oss_camera_server_java_sdk/src/site/apidocs/index.html

二、设备端&服务端 C_SDK 编译说明

1、安装 OSS_C_SDk,下载地址以及安装说明：

https://docs.aliyun.com/?spm=5176.383663.9.10.YN2pkt#/pub/oss/sdk/sdk-download&c

2、设备端 C _SDK（oss_camera_c_sdk）编译说明：

【依赖库】:

* OSS: [https://docs.aliyun.com/?#/pub/oss/sdk/sdk-download&c]
* APR/APR-UTIL: [https://apr.apache.org/]

针对嵌入式库体积优化，配置建议：

编译 apr 库

./configure CFLAGS='-Os' --prefix=/usr/local/apr/

编译 apr-util 库

./configure CFLAGS='-Os' --prefix=/usr/local/apr-util --with-apr=/usr/local/apr

* LIBXML2: [http://www.xmlsoft.org/]
  针对嵌入式库体积优化， 建议：
  参考：http://www.cokco.cn/thread-11777-1-1.html
  配置：./configure CFLAGS='-Os' --with-minimum --with-writer --with-tree --with-xpath

* CURL: [http://curl.haxx.se/]
  针对嵌入式库体积优化， 建议：
  参考：http://curl.haxx.se/docs/install.html
  配置：./configure CFLAGS='-Os' --prefix=/usr/local/libcurl --disable-ftp --disable-file --disable-ldap --disable-dict -disable-telnet --disable-tftp --disable-rtsp --disable-pop3 -disable-imap --disable-smtp --disable-gopher --disable-ares -disable-debug --without-ssl --without-zlib --without-libidn

* SSL: [https://www.openssl.org/]

【安装 cmake】
* download cmake from www.cmake.org [http://www.cmake.org/download/]
* install cmake accoding to guide [http://www.cmake.org/install/]

【用 cmake 生成 Makefile】
* cmake .

注意：如果依赖库未安装，可能会遇到编译错误，请按提示安装依赖库。例如：
```
APR_INCLUDE_DIR
    used as include directory in directory xxx
    used as include directory in directory xxx
APR_LIBRARY
     linked by target "oss_media_example" in directory xxx
APR_UTIL_INCLUDE_DIR
    used as include directory in directory xxx
    used as include directory in directory xxx
APR_UTIL_LIBRARY
     linked by target "oss_media_example" in directory xxx
OSS_LIBRARY
     linked by target "oss_media_example" in directory xxx
XML2_INCLUDE_DIR
    used as include directory in directory xxx
    used as include directory in directory xxx
```

【利用 ccmake or cmake -D 命令行参数进行编译】
* ccmake .
* cmake -DAPR_INCLUDE_DIR=xxx -DAPR_LIBRARY=xxx -DAPR_UTIL_INCLUDE_DIR=xxx -

DAPR_UTIL_LIBRARY=xxx -DXML2_INCLUDE_DIR=xxx -DOSS_LIBRARY=xxx .

【编译 OSS_MEDIA_C_SDK】
* make

【目录结构】:

```
build
├─── bin
│   └─── oss_media_example
├─── include
│   ├─── aos_buf.h
│   ├─── aos_define.h
│   ├─── aos_fstack.h
│   ├─── aos_http_io.h
│   ├─── aos_list.h
│   ├─── aos_log.h
│   ├─── aos_status.h
│   ├─── aos_string.h
│   ├─── aos_transport.h
│   ├─── aos_util.h
│   ├─── oss_api.h
│   ├─── oss_define.h
│   ├─── oss_media_file.h
│   └─── oss_util.h
├─── lib
│   └─── liboss_media.a
└─── res
```

3、服务端 C _SDK（oss_camera_server_c_sdk）编译说明：
【依赖库】:
* OSS: [https://docs.aliyun.com/?#/pub/oss/sdk/sdk-download&c]
* APR/APR-UTIL: [https://apr.apache.org/]
* LIBXML2: [http://www.xmlsoft.org/]
* SSL: [https://www.openssl.org/]
* CURL: [http://curl.haxx.se/]

【安装 cmake】
* download cmake from www.cmake.org [http://www.cmake.org/download/]
* install cmake accoding to guide [http://www.cmake.org/install/]

【用 cmake 生成 Makefile】
* cmake .

注意：如果依赖库未安装，可能会遇到编译错误，请按提示安装依赖库。例如：
APR_INCLUDE_DIR

used as include directory in directory xxx
used as include directory in directory xxx
APR_LIBRARY
linked by target "oss_media_example" in directory xxx
APR_UTIL_INCLUDE_DIR
used as include directory in directory xxx
used as include directory in directory xxx
APR_UTIL_LIBRARY
linked by target "oss_media_example" in directory xxx
OSS_LIBRARY
linked by target "oss_media_example" in directory xxx
XML2_INCLUDE_DIR
used as include directory in directory xxx
used as include directory in directory xxx

【利用 ccmake or cmake -D 命令行参数进行编译】
* ccmake .
* cmake -DAPR_INCLUDE_DIR=xxx -DAPR_LIBRARY=xxx -DAPR_UTIL_INCLUDE_DIR=xxx -DAPR_UTIL_LIBRARY=xxx -DXML2_INCLUDE_DIR=xxx -DOSS_LIBRARY=xxx .

【编译 OSS_MEDIA_C_SDK】
* make

【目录结构】:
build
├── bin
│ ├── oss_media_example
│ └── sts_sample
├── include
│ ├── aos_buf.h
│ ├── aos_define.h
│ ├── aos_fstack.h
│ ├── aos_http_io.h
│ ├── aos_list.h
│ ├── aos_log.h
│ ├── aos_status.h
│ ├── aos_string.h
│ ├── aos_transport.h
│ ├── aos_util.h
│ ├── libsts.h
│ ├── oss_api.h
│ ├── oss_define.h
│ ├── oss_media.h
│ └── oss_util.h

```
└──── lib
    ├──── liboss_media.a
    └──── libsts.a
```

三、设备端 C _SDK 结构体以及 API 说明

A、结构体说明

1、结构体名称：oss_media_file_stat_t

【用途】:

用来描述 oss media file 的状态；

【语法】:

```
typedef struct oss_media_file_stat_s {
    long length;      // 文件长度
    char *type;       // 文件的长度，普通文件/append 文件
} oss_media_file_stat_t;
```

2、结构体名称：oss_media_file_t;

【用途】:

oss media file 的属性；

OSS 相关知识：

https://docs.aliyun.com/?spm=5176.7114037.1996646101.7.IU4YDs&pos=4#/pub/oss

【语法】:

```
typedef struct oss_media_file_s {
    void     *ipc;                  // 网络摄像机信息（比如摄像机 ID 等）

    char     *host;                 // oss host
    int      port;                  // oss port
    char     *bucket;               // oss bucket name
    char     *filename;             //   oss 对象名称（文件名）
    char     *ak;                   // oss temporary access key
    char     *sk;                   // oss temporary access key secret
    char     *token;                // oss temporary access token
    time_t expiration;              // oss 临时访问过期时间

    void (* auth)(struct oss_media_file_s *file); // 用户安全访问函数

    char *mode;                     // 打开文件的模式 [r : read; w : write; a : append]
    char *_type;                    // 文件类型
    long   _length;                 //文件长度
    long   _read_offset;            // 读取偏移

    int    code;                    // OSS http code
```

```
    char message[32+1];                // OSS code message

} oss_media_file_t;
```

B、API 说明

1、oss_media_init

【目的】:

本 SDK 初始化函数

【语法】:

```
int oss_media_init();
```

【描述】:

在应用中必须首先调用该函数接口，且只调用一次；

【参数】:

无

【返回值】:

返回 AOSE_OK ，成功；否则失败.

【注意】:

无

2、oss_media_destroy

【目的】:

反初始化函数

【语法】:

```
void oss_media_destroy();
```

【描述】:

需在应用结束使用 OSS 时调用；

【参数】:

无

【返回值】:

无

【注意】:

与初始化函数一一对应

3、oss_media_file_create

【目的】:

在 OSS 中创建媒体文件

【语法】:

```
oss_media_file_t *oss_media_file_create();
```

【描述】:

模拟文件操作，在 OSS 中创建媒体文件，返回该文件的属性结构体

【参数】:

无

【返回值】:

oss_media_file_t *
【注意】:
无

4、oss_media_file_free
【目的】:
释放 OSS 中的媒体文件
【语法】:
void oss_media_file_free(oss_media_file_t *file);
【描述】:
模拟文件操作，根据指定的文件名，释放 OSS 中的媒体文件
【参数】:
输入参数为该文件对应的属性结构体：oss_media_file_t
【返回值】:
oss_media_file_t *
【注意】:
无

5、oss_media_file_open
【目的】:
打开 OSS 中的媒体文件
【语法】:
int oss_media_file_open(oss_media_file_t *file, char *mode);
【描述】:
模拟文件操作的 open 动作，根据结构体中指定的文件名，打开 OSS 中的媒体文件，有三种模式：
'r': file access mode of read.
'w': file access mode of write. 覆盖写，一般用来保存索引文件。
a': file access mode of append. 追加写，也就是流式写入，一般用来保存视频流文件。
但目前不支持两种及以上组合动作，比如 r+w 等；在 append 模式下，若要实现对同一文件边写边读的功能，请对同一文件名先按'w'模式打开；同时按'r'模式再打开一次，这样可以实现对同一文件一边 append 写入，一边从文件中读已经 append 完成的数据。
【参数】:
输入参数为该文件对应的属性结构体：oss_media_file_t
【返回值】:
0：成功；
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
目前不支持两种及以上组合动作，比如 r+w 等；

6、oss_media_file_close
【目的】:
关闭 OSS 中的媒体文件
【语法】:

int oss_media_file_close(oss_media_file_t *file);
【描述】:
模拟文件操作的 close 动作，根据结构体中指定的文件名，关闭 OSS 中的媒体文件，代表这个文件结束。
【参数】:
输入参数为该文件对应的属性结构体：oss_media_file_t
【返回值】:
0：成功；
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
若该文件是以'w'模式打开，也就是普通文件对象，则该文件 close 后就不能再进行写操作；若该文件是以'a'模式打开，也就是 append 文件对象，则该文件 close 后还可以'a'打开进行 append 写操作；

7、oss_media_file_stat
【目的】:
查看 OSS 中的媒体文件状态信息
【语法】:
int oss_media_file_stat(oss_media_file_t *file, oss_media_file_stat_t *stat);
【描述】:
模拟文件操作，，通过结构体 oss_media_file_t 中的文件名，查看 OSS 中的媒体文件状态信息。目前只能看到 oss_media_file_stat_t 中的结果：包括文件的长度和文件类型（普通文件，或者 append 文件）

【参数】:
输入参数为该文件对应的属性结构体：oss_media_file_t
输出参数：oss_media_file_stat_t
【返回值】:
0：成功；
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
无

8、oss_media_file_tell
【目的】:
当前指针相对与文件头的位置
【语法】:
long oss_media_file_tell(oss_media_file_t *file);
【描述】:
模拟文件操作，通过结构体 oss_media_file_t 中的文件名，返回 OSS 中的媒体文件当前指针相对于文件头的位置。
【参数】:
输入参数为该文件对应的属性结构体：oss_media_file_t
【返回值】:

大于-1：成功，返回文件指针当前位置
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
无

9、oss_media_file_seek
【目的】:
当前指针相对与文件头的位置
【语法】:
long oss_media_file_seek(oss_media_file_t *file, long offset);
【描述】:
模拟文件操作，通过结构体 oss_media_file_t 中的文件名，按 offset 偏移到相对于文件头的位置。
【参数】:
输入参数：
文件对应的属性结构体：oss_media_file_t
long offset ,相对于文件头的偏移位置
【返回值】:
大于-1：成功，返回文件指针相对文件头偏移的位置
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
一般用于’r’操作，不能用于 ‘w’操作；

10、oss_media_file_read
【目的】:
读文件内容到指定 buf;
【语法】:
int oss_media_file_read(oss_media_file_t *file, void *buf, int nbyte);
【描述】:
模拟文件操作，通过结构体 oss_media_file_t 中的文件名，读指定长度的文件内容到 buf 中。
【参数】:
输入参数：
文件对应的属性结构体：oss_media_file_t
void *buf , 读取出来的内容存放的 buf
int nbyte，读取文件的长度
【返回值】:
大于-1：成功，返回读取文件内容的长度
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
无

11、oss_media_file_write

【目的】:
将 buf 中的内容写入到文件;
【语法】:
int oss_media_file_read(oss_media_file_t *file, void *buf, int nbyte);
【描述】:
模拟文件操作，从 buf 中写指定长度的内容到 oss_media_file_t 结构体的文件中。
【参数】:
输入参数：
文件对应的属性结构体：oss_media_file_t
void *buf，写入内容存放的 buf
int nbyte，写入文件的长度
【返回值】:
大于-1：成功，返回写入文件内容的长度
-1：请查看结构体 oss_media_file_t 中错误码和错误信息（code/messaage）；
【注意】:
无

四、设备端 C_SDK 示例代码

1、从设备端写文件到 OSS:

```
static void oss_clean(char *filename) {
    aos_pool_t *pool;
    oss_request_options_t *opts;
    aos_string_t bucket;
    aos_string_t key;
    aos_status_t *status;
    aos_table_t *resp_headers;

    aos_pool_create(&pool, NULL);

    opts = oss_request_options_create(pool);
    opts->config = oss_config_create(pool);
    aos_str_set(&opts->config->host, g_host);
    opts->config->port = g_port;
    aos_str_set(&opts->config->id, g_gak);
    aos_str_set(&opts->config->key, g_gsk);
    opts->config->is_oss_domain = 1;
    opts->ctl = aos_http_controller_create(pool, 0);
    aos_str_set(&bucket, g_bucket);
    aos_str_set(&key, filename);
    resp_headers = aos_table_make(pool, 0);
```

```
    status = oss_delete_object(opts, &bucket, &key, &resp_headers);
    if (!aos_status_is_ok(status)) {
        aos_info_log("clean oss error. [code=%d, message=%s]", status->code,
status->error_code);
        aos_info_log("example exit.");
        aos_pool_destroy(pool);
        exit(-1);
    }

    aos_pool_destroy(pool);
}

static void example_auth(oss_media_file_t *file) {
    file->host=g_host;
    file->port=g_port;
    file->bucket=g_bucket;
    file->filename=g_filename;
    file->ak = g_ak;
    file->sk = g_sk;
    file->token = g_token;
    // expiration 5 sec.
    file->expiration = time(NULL) + 5;
    aos_info_log("auth: example_auth");
}

static void example_write() {
    oss_clean(g_filename);

    oss_media_file_t *file;
    // create
    file = oss_media_file_create();
    file->auth = example_auth;
    // open
    oss_media_file_open(file, "w");
    // write
    oss_media_file_write(file, g_val, strlen(g_val));
    oss_media_file_write(file, g_val, strlen(g_val));
    // close
    oss_media_file_close(file);
    // free
    oss_media_file_free(file);
}
```

2、追加视频流到 OSS（append 模式）

```
static void example_append() {
    oss_clean(g_filename);

    oss_media_file_t *file;
    // create
    file = oss_media_file_create();
    file->auth = example_auth;
    // open
    oss_media_file_open(file, "a");
    // write
    int i;
    for (i = 0; i < 8; i++) {
        oss_media_file_write(file, g_val, strlen(g_val));
    }
    // close
    oss_media_file_close(file);
    // free
    oss_media_file_free(file);
}
```

3、从 OSS 中 read 文件

```
static void example_read() {
    char *content;
    int ntotal, nread, nbuf=16;
    char buf[nbuf];

    oss_media_file_t *file;
    // create
    file = oss_media_file_create();
    file->auth = example_auth;
    // open
    oss_media_file_open(file, "r");
    // stat
    oss_media_file_stat_t stat;
    oss_media_file_stat(file, &stat);
    aos_info_log("file [name=%s, length=%d, type=%s]", file->filename, stat.length, stat.type);

    // read
    content = malloc(stat.length + 1);
    ntotal = 0;
    while ((nread=oss_media_file_read(file, buf, nbuf)) > 0) {
        memcpy(content + ntotal, buf, nread);
        ntotal += nread;
    }
```

```
    content[ntotal] = '\0';
    aos_info_log("file content is: \n%s", content);
    free(content);

    // close
    oss_media_file_close(file);
    // free
    oss_media_file_free(file);
}
```

4、文件 seek 操作

```
static void example_seek() {
    char *content;
    int ntotal, nread, nbuf=16;
    char buf[nbuf];

    oss_media_file_t *file;
    // create
    file = oss_media_file_create();
    file->auth = example_auth;
    // open
    oss_media_file_open(file, "r");
    // stat
    oss_media_file_stat_t stat;
    oss_media_file_stat(file, &stat);
    aos_info_log("file [name=%s, length=%d, type=%s]", file->filename, stat.length, stat.type);

    // tell
    aos_info_log("file [position=%d]", oss_media_file_tell(file));

    // seek
    oss_media_file_seek(file, stat.length / 2);

    // tell
    aos_info_log("file [position=%d]", oss_media_file_tell(file));

    // read
    content = malloc(stat.length / 2 + 2);
    ntotal = 0;
    while ((nread=oss_media_file_read(file, buf, nbuf)) > 0) {
        memcpy(content + ntotal, buf, nread);
        ntotal += nread;
    }
    content[ntotal] = '\0';
```

```
    aos_info_log("file content is: \n%s", content);
    free(content);

    // close
    oss_media_file_close(file);
    // free
    oss_media_file_free(file);

}
```

其他示例代码请见：…/oss_media_example.c.

五、服务端 C _SDK 结构体以及 API 说明

A、结构体说明

1、结构体名称：oss_media_config_t

【用途】:

用来配置 OSS 访问信息；

【语法】:

```
typedef struct oss_media_config_s {
    char *host;              // oss host
    int  port;               // oss port
    char *ak;                // oss access key
    char *sk;                // oss secret key
    char *role_arn;          // role arn
} oss_media_config_t;
```

2、结构体名称：oss_media_status_t;

【用途】:

用来返回 OSS 访问状态信息，包含（成功 OR 失败）的状态码和信息；

【语法】:

```
typedef struct oss_media_status_s {
    int code;                  // oss http status code
    char message[256+1]; // oss http code message
} oss_media_status_t;
```

3、结构体名称：oss_media_acl_t;;

【用途】:

oss bucket acl，oss bucket 访问控制列表

acl:

```
 *  private write + private read
 *  private write + public  read
 *  public write  + public  read
```

详见：
https://docs.aliyun.com/?spm=5176.7114037.1996646101.7.IU4YDs&pos=4#/pub/oss/api-reference/access-control&bucket-acl
【语法】:
typedef oss_acl_e oss_media_acl_t;

4、结构体名称：oss_media_lifecycle_rule_t;
【用途】:
* oss bucket lifecycle rule.
* oss file will be deleted automatic in specified days.
【语法】:
```
typedef struct oss_media_lifecycle_rule_s {
    char *name;              // rule name
    char *path;              // oss media path
    char *status;            // Enabled or Disabled
    int  days;
} oss_media_lifecycle_rule_t;
```

5、结构体名称：oss_media_lifecycle_rules_t
【用途】:
* oss bucket lifecycle rule sets.
* this struct describes a collection of oss bucket lifecycle rules.
【语法】:
```
typedef struct oss_media_lifecycle_rules_s {
    aos_pool_t *_pool;
    int size;                                   // rule array size
    oss_media_lifecycle_rule_t **rules;         // rule array
} oss_media_lifecycle_rules_t;
```

6、结构体名称：oss_media_files_t
【用途】:
* oss media files
* this struct describes a collection of oss media files.
【语法】:
```
typedef struct oss_media_files_s {
    char *path;              // file path
    char *marker;            // paging identifier
    int  max_size;           // max items per page

    char *next_marker;       // paging identifier
    int  size;               // iterms per page
    char **file_names;       // file name array
```

} oss_media_files_t;

7、结构体名称：oss_media_token_t

【用途】:

* oss media token
* this struct describes the security token for accessing oss media file.

使用前请先了解 STS 相关知识：

https://docs.aliyun.com/?spm=5176.7114037.1996646101.3.qn85ea&pos=2#/pub/ram/sts-user-guide/intro

https://docs.aliyun.com/?spm=5176.7114037.1996646101.1.qn85ea&pos=1#/pub/ram/sts-api-reference/intro

【语法】:

typedef struct STSData oss_media_token_t;

B、API 说明

1、oss_media_init

【目的】:

本 SDK 初始化函数

【语法】:

int oss_media_init();

【描述】:

在应用中必须首先调用该函数接口，且只调用一次；

【参数】:

无

【返回值】:

返回 AOSE_OK ，成功；否则失败；

【注意】:

无

2、oss_media_destroy

【目的】:

反初始化函数

【语法】:

void oss_media_destroy();

【描述】:

需在应用结束使用 OSS 时调用；

【参数】:

无

【返回值】:

无

【注意】:

与初始化函数一一对应

3、oss_media_file_create
【目的】:
在 OSS 中创建媒体文件
【语法】:
oss_media_files_t* oss_media_create_files();
【描述】:
模拟文件操作，在 OSS 中创建媒体文件，返回该文件的属性结构体

【参数】:
无
【返回值】:
oss_media_file_t *
【注意】:
无

4、oss_media_file_free
【目的】:
释放 OSS 中的媒体文件
【语法】:
void oss_media_free_files(oss_media_files_t *files);
【描述】:
模拟文件操作，根据指定的文件名，释放 OSS 中的媒体文件
【参数】:
输入参数为该文件对应的属性结构体：oss_media_file_t
【返回值】:
oss_media_file_t *
【注意】:
无

5、oss_media_create_lifecycle_rules
【目的】:
创建 lifecycle 规则结构体。
Lifecycle 功能介绍：
https://docs.aliyun.com/#/pub/oss/product-documentation/function&lifecycle
【语法】:
oss_media_lifecycle_rules_t *oss_media_create_lifecycle_rules(int size);
【描述】:
创建 lifecycle 规则结构体。为 bucket lifecycle 做准备。
【参数】:
Int Size 规则数组的长度
【返回值】:
oss_media_lifecycle_rules_t * 生产的规则数组的指针
【注意】:
无

6、oss_media_free_lifecycle_rules
【目的】:
释放指定的 lifecycle 规则结构体。
Lifecycle 相关知识：
https://docs.aliyun.com/#/pub/oss/product-documentation/function&lifecycle
【语法】:
void oss_media_free_lifecycle_rules(oss_media_lifecycle_rules_t *rules);
【描述】:
当创建 lifecycle 规则失效时，可以用此函数释放。
【参数】:
oss_media_lifecycle_rules_t *rules 当前要释放的规则；
【返回值】:
无
【注意】:
无

7、oss_media_create_bucket
【目的】:
创建 OSS bucket
Bucket 相关知识：https://docs.aliyun.com/#/pub/oss/product-documentation/function&bucket
【语法】:
int oss_media_create_bucket(oss_media_config_t *config, char *bucket_name, oss_media_acl_t acl, oss_media_status_t *status);
【描述】:
用户使用 OSS 存储对象，必须先创建 bucket ； Bucket 是 OSS 上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。
【参数】:
oss_media_config_t *config OSS 相关配置信息；
char *bucket_name bucket 名称；
oss_media_acl_t acl bucket 访问控制权限；
oss_media_status_t *status bucket 创建状态信息；
【返回值】:
0：成功
-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）；
【注意】:
无

8、oss_media_delete_bucket
【目的】:
删除已有 OSS bucket；
Bucket 相关知识：https://docs.aliyun.com/#/pub/oss/product-documentation/function&bucket
【语法】:
int oss_media_delete_bucket(oss_media_config_t *config, char *bucket_name,

oss_media_status_t *status);
【描述】:
删除已有 OSS bucket;
【参数】:
oss_media_config_t *config　OSS 相关配置信息;
char *bucket_name　　bucket 名称;
oss_media_status_t *status　bucket 创建状态信息;
【返回值】:
0：成功
-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）;
【注意】:
无

9、oss_media_create_bucket_lifecycle
【目的】:
为 OSS bucket 创建 lifecycle
Bucket 相关知识：https://docs.aliyun.com/#/pub/oss/product-documentation/function&bucket
Lifecycle 相关知识：https://docs.aliyun.com/#/pub/oss/product-documentation/function&lifecycle
【语法】:
int oss_media_create_bucket_lifecycle(oss_media_config_t *config, char *bucket_name,
oss_media_lifecycle_rules_t *rules, oss_media_status_t *status);
【描述】:
Bucket 的拥有者可以通过该接口来设置 Bucket 的 Lifecycle 配置。Lifecycle 开启后，OSS 将按照配置，定期自动删除与 Lifecycle 规则相匹配的 Object（文件）
【参数】:
oss_media_config_t *config　　OSS 相关配置信息;
char *bucket_name　　bucket 名称;
oss_media_lifecycle_rules_t *rules　　lifecycle 规则结构体;
oss_media_status_t *status　　bucket 创建状态信息;
【返回值】:
0：成功
-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）;
【注意】:
无

10、oss_media_get_bucket_lifecycle
【目的】:
获取 OSS bucket 相关的 lifecycle 信息
Bucket 相关知识：https://docs.aliyun.com/#/pub/oss/product-documentation/function&bucket
Lifecycle 相关知识：https://docs.aliyun.com/#/pub/oss/product-documentation/function&lifecycle
【语法】:
int oss_media_get_bucket_lifecycle(oss_media_config_t *config, char *bucket_name,
oss_media_lifecycle_rules_t *rules, oss_media_status_t *status);

【描述】:
Bucket 的拥有者可以通过该接口来查看 Bucket 的 Lifecycle 配置信息。
【参数】:
oss_media_config_t *config　　OSS 相关配置信息；
char *bucket_name　　bucket 名称；
oss_media_lifecycle_rules_t *rules　　lifecycle 规则结构体；
oss_media_status_t *status　　bucket 创建状态信息；
【返回值】:
0：成功
-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）；
【注意】:
无

11、oss_media_delete_file
【目的】:
删除 OSS bucket 中的指定文件；
【语法】:
int oss_media_delete_file(oss_media_config_t *config, char *bucket_name, char *file, oss_media_status_t *status);
【描述】:
删除 OSS 中的文件。
【参数】:
oss_media_config_t *config　　OSS 相关配置信息；
char *bucket_name　　bucket 名称；
char *file,　　文件名；
oss_media_status_t *status　　bucket 创建状态信息；
【返回值】:
0：成功
-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）；
【注意】:
无

12、oss_media_list_files
【目的】:
列出 OSS bucket 中的文件；
【语法】:
int oss_media_list_files(oss_media_config_t *config, char *bucket_name, oss_media_files_t *files, oss_media_status_t *status);
【描述】:
当使用者想要查看 OSS bucket 中的文件时，通过本接口列出 OSS bucket 中的文件，支持分页；
【参数】:
oss_media_config_t *config　　OSS 相关配置信息；

char *bucket_name	bucket 名称；
oss_media_files_t *files	文件信息
oss_media_status_t *status	bucket 创建状态信息；

【返回值】:

0：成功

-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）；

【注意】:

无

13、oss_media_get_token

【目的】:

获取访问 OSS 的临时 access token.；

【语法】:

```
int oss_media_get_token(oss_media_config_t *config, char *bucket_name, char *path,
                        char *mode, unsigned int expiration,
                        oss_media_token_t *token, oss_media_status_t *status);
```

【描述】:

为了访问 OSS 更加安全，阿里云提供临时访问 token 服务：STS。建议使用本 API 前先了解 STS 相关知识：

https://docs.aliyun.com/?spm=5176.7114037.1996646101.3.qn85ea&pos=2#/pub/ram/sts-user-guide/intro

https://docs.aliyun.com/?spm=5176.7114037.1996646101.1.qn85ea&pos=1#/pub/ram/sts-api-reference/intro

*

【参数】:

oss_media_config_t *config	OSS 相关配置信息；
char *bucket_name	bucket 名称；
char *path	文件路径
char *mode,	r': file access mode of read； w': file access mode of write. 'a': file access mode of append.
unsigned int expiration	访问过期时间，以秒为单位，最大 3600 秒
oss_media_token_t *token	获取到的临时访问 token；
oss_media_status_t *status	bucket 创建状态信息；

【返回值】:

0：成功

-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）；

【注意】:

无

14、oss_media_get_token_from_policy

【目的】:

通过访问策略（policy）获取 OSS 的临时 token；

【语法】:
int oss_media_get_token_from_policy(oss_media_config_t *config, char *policy,
unsigned int expiration,
oss_media_token_t *token, oss_media_status_t *status);

【描述】:
若用户需要更加灵活的 OSS 访问策略，请使用本 API.相对于 oss_media_get_token 这个 API，本 API 的访问策略会更灵活。
建议使用本 API 前先了解 STS 相关知识：
https://docs.aliyun.com/?spm=5176.7114037.1996646101.3.qn85ea&pos=2#/pub/ram/sts-user-guide/intro
https://docs.aliyun.com/?spm=5176.7114037.1996646101.1.qn85ea&pos=1#/pub/ram/sts-api-reference/intro
*
【参数】:

oss_media_config_t *config	OSS 相关配置信息；
char *policy	访问策略，用来描述授权策略的一种描述语言
unsigned int expiration	访问过期时间，以秒为单位，最大 3600 秒
oss_media_token_t *token	获取到的临时访问 token；
oss_media_status_t *status	bucket 创建状态信息；

【返回值】:
0：成功
-1：请查看结构体 oss_media_status_t 中错误码和错误信息（code/messaage）；
【注意】:
无

六、服务端 C _SDK 示例

1、创建 bucket

```
void example_create_bucket() {
    int r;
    oss_media_status_t status;
    r = oss_media_create_bucket(&config, g_bucket, OSS_ACL_PRIVATE, &status);
    aos_info_log("create bucket. [name=%s, ret=%d, code=%d, message=%s]", g_bucket, r,
status.code, status.message);
}
```

2、删除 bucket

```
void example_delete_bucket() {
    int r;
    oss_media_status_t status;
    r = oss_media_delete_bucket(&config, g_bucket, &status);
```

```
    aos_info_log("delete bucket. [name=%s, ret=%d, code=%d, message=%s]", g_bucket, r,
status.code, status.message);
}
```

3、设置 lifecycle（到期自动删除）

```
void example_create_bucket_lifecycle() {
    int r, i;
    oss_media_status_t status;
    oss_media_lifecycle_rules_t *rules;

    rules = oss_media_create_lifecycle_rules(2);
    oss_media_lifecycle_rule_t rule1;
    rule1.name = "example-1";
    rule1.path = "/example/1";
    rule1.status = "Enabled";
    rule1.days = 1;
    oss_media_lifecycle_rule_t rule2;
    rule2.name = "example-2";
    rule2.path = "/example/2";
    rule2.status = "Disabled";
    rule2.days = 2;

    rules->rules[0] = &rule1;
    rules->rules[1] = &rule2;

    r = oss_media_create_bucket_lifecycle(&config, g_bucket, rules, &status);
    aos_info_log("create bucket lifecycle. [name=%s, ret=%d, code=%d, message=%s]",
g_bucket, r, status.code, status.message);

    oss_media_free_lifecycle_rules(rules);
}
```

4、获取临时 token

```
void example_get_token() {
    int r;
    oss_media_status_t status;
    oss_media_token_t    token;

    r = oss_media_get_token(&config, g_bucket, "/*", "rwa", 3600, &token, &status);
    aos_info_log("get token. [ak=%s, sk=%s, token=%s, ret=%d, code=%d, message=%s]",
            token.tmpAccessKeyId, token.tmpAccessKeySecret, token.securityToken, r,
status.code, status.message);
}
```

```
void example_get_token_from_policy() {
    int r;
    oss_media_status_t status;
    oss_media_token_t    token;
    char *policy = "{\"Version\":\"1\",\"Statement\":[{\"Effect\":\"Allow\", \"Action\":\"*\",
\"Resource\":\"acs:oss:*:*:data-misc/*\"}]}";

    r = oss_media_get_token_from_policy(&config, policy, 3600, &token, &status);
    aos_info_log("get token. [ak=%s, sk=%s, token=%s, ret=%d, code=%d, message=%s]",
            token.tmpAccessKeyId, token.tmpAccessKeySecret, token.securityToken, r,
status.code, status.message);
}
```

更多示例代码，请见： …/ oss_media_example.c

七、移动端 OSS_SDK

1、IOS 端 OSS SDK：
https://docs.aliyun.com/?spm=5176.383663.9.8.Av1tEB#/pub/oss/sdk/sdk-download&ios

2、安卓端 OSS SDK
https://docs.aliyun.com/?spm=5176.383663.9.8.Av1tEB#/pub/oss/sdk/sdk-download&android