Tedu.cn
达内教育

《分布式流媒体》实训项目

C/C++教学体系

TNV DAY13

直播课

目录

业务服务类(service_c)

三级方法

知识讲解

- 生成文件路径：genpath
  - 从存储路径表中随机抽取一个存储路径
  - 以存储路径为键从ID服务器获取与之对应的值作为文件ID
  - 先将文件ID转换为512进制，再根据它生成文件路径
  - 返回成功
- 将ID转换为512进制：id512
  - $14956534716 = (11122233344)_{512} = 111\times512^3 + 222\times512^2 + 333\times512^1 + 444\times512^0$

三级方法

知识讲解

- 用文件ID生成文件路径：id2path
  - 检查存储路径
  - 生成文件路径中的各个分量
  - 一级子目录：111
    - 二级子目录：222
    - 三级子目录：333
    - 当前时间戳： time(NULL)
    - 文件名后缀：444
  - 格式化完整的文件路径
  - 返回成功

三级方法

知识讲解

- 接收并保存文件：save
  - 实例化文件操作对象
  - 打开文件
  - 依次将接收到的数据块写入文件
    - 初始化未接收字节数为文件大小，分配接收写入缓冲区
    - 若还有未接收数据则循环执行
      - 接收数据，写入文件，未收递减
  - 关闭文件
  - 实例化数据库访问对象
  - 连接数据库
  - 设置文件ID和路径及大小的对应关系
  - 返回成功

三级方法

知识讲解

- 读取并发送文件：send
  - 实例化文件操作对象
  - 打开文件
  - 设置偏移
  - 构造响应头
  - 发送响应头
  - 依次将从文件中读取到的数据块作为响应体的一部分发送出去
    - 初始化未读取字节数为下载大小，分配读取发送缓冲区
    - 若还有未读取数据则循环执行
      - 读取文件，发送数据，未读递减
  - 关闭文件
  - 返回成功

底层方法

知识讲解

- 应答成功：ok
  - 构造响应
  - 发送响应
- 应答错误：error
  - 错误描述
  - 构造响应
  - 发送响应

包头		
包体长度	命令(102)	状态
8	1	1

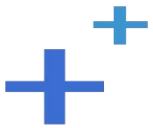
包头			包体	
包体长度	命令(102)	状态	错误号	错误描述
8	1	1	2	<=1024

跟踪客户机线程类(tracker_c)

属性和构造

知识讲解

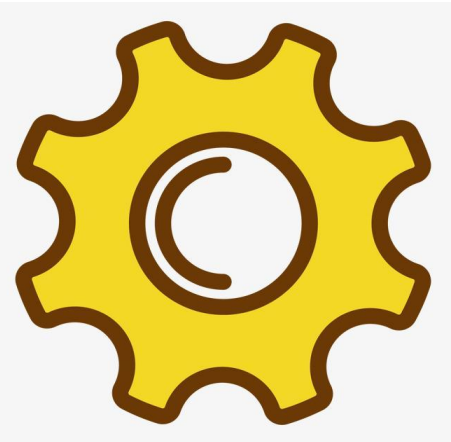
- 成员变量
  - 是否终止：m_stop
  - 跟踪服务器地址：m_taddr
- 构造函数：tracker_c
  - 初始化m_stop为false
  - 初始化m_taddr为跟踪服务器地址



方法

知识讲解

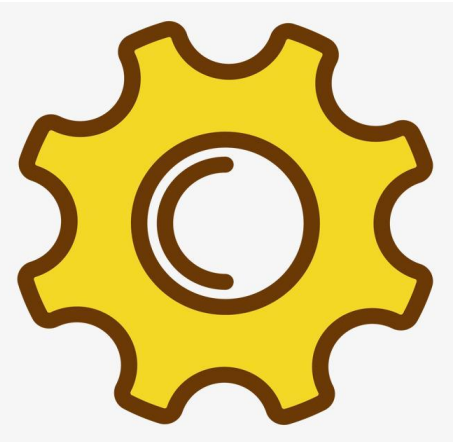
- 终止线程：stop
  - 将m_stop置为true
- 线程过程：run
  - 连接跟踪服务器，失败重连
  - 向跟踪服务器发送加入包，失败重连
  - 若线程不终止，则持续循环
    - 获取当前时间
    - 若当前时间距离上次心跳时间已足够久
      - 向跟踪服务器发送心跳包，失败重连
      - 更新上次心跳时间为当前时间
  - 退出循环，关闭连接，线程终止



方法

知识讲解

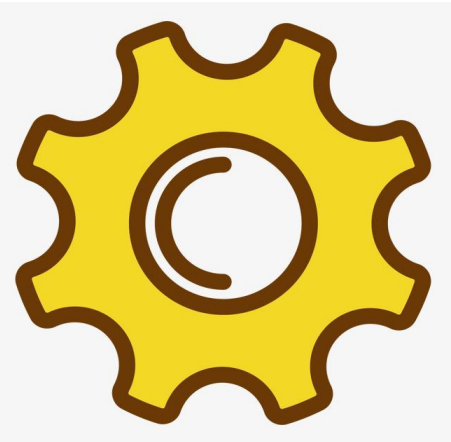
- 向跟踪服务器发送加入包：join
  - 构造请求
  - 发送请求
  - 接收包头
  - 解析包头
  - 检查命令
  - 检查状态
    - 跟踪服务器应答成功
      - 返回成功
    - 跟踪服务器应答错误
      - 检查包体长度并接收包体
      - 解析包体，获取错误号及错误描述并打印日志
      - 返回失败



方法

知识讲解

- 向跟踪服务器发送心跳包：beat
  - 构造请求
  - 发送请求
  - 接收包头
  - 解析包头
  - 检查命令
  - 检查状态
    - 跟踪服务器应答成功
      - 返回成功
    - 跟踪服务器应答错误
      - 检查包体长度并接收包体
      - 解析包体，获取错误号及错误描述并打印日志
      - 返回失败



附录：程序清单

TNV/src/04_storage/12_service.cpp

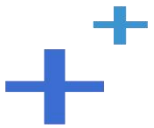
知识讲解

```
//生成文件路径
int service_c::genpath(char* filepath) const {
        //从存储路径表中随机抽取一个存储路径
        srand(time(NULL));
        int nspaths = g_spaths.size();
        int nrand = rand() % nspaths;
        std::string spath = g_spaths[nrand];

        //以存储路径为键从ID服务器获取与之对应的值作为文件ID
        id_c id;
        long fileid = id.get(spath.c_str());
        if (fileid < 0)
                return ERROR;

        //先将文件ID转换为512进制，再根据它生成文件路径
```

TNV/src/04_storage/12_service.cpp

知识讲解

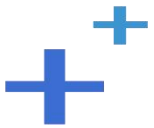
```
        return id2path(spath.c_str(), id512(fileid), filepath);
}

// 将ID转换为512进制
long service_c::id512(long id) const {
        long result = 0;

        for (int i = 1; id; i *= 1000) {
                result += (id % 512) * i;
                id /= 512;
        }

        return result;
}
```



TNV/src/04_storage/12_service.cpp

知识讲解

```
//用文件ID生成文件路径
int service_c::id2path(char const* spath, long fileid,
        char* filepath) const {
        //检查存储路径
        if (!spath || !strlen(spath)) {
                logger_error("storage path is null");
                return ERROR;
        }

        //生成文件路径中的各个分量
        unsigned short subdir1 = (fileid / 1000000000) % 1000; //一级子目录
        unsigned short subdir2 = (fileid / 1000000) % 1000;    //二级子目录
        unsigned short subdir3 = (fileid / 1000) % 1000;       //三级子目录
        time_t         curtime = time(NULL);                   //当前时间戳
        unsigned short postfix = (fileid / 1) % 1000;          //文件名后缀
```



TNV/src/04_storage/12_service.cpp

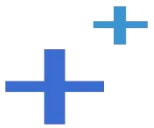
知识讲解

```
        //格式化完整的文件路径
        if (spath[strlen(spath)-1] == '/')
                snprintf(filepath, PATH_MAX + 1, "%s%03X/%03X/%03X/%1X_%03X",
                        spath, subdir1, subdir2, subdir3, curtime, postfix);
        else
                snprintf(filepath, PATH_MAX + 1, "%s/%03X/%03X/%03X/%1X_%03X",
                        spath, subdir1, subdir2, subdir3, curtime, postfix);

        return OK;
}

//接收并保存文件
int service_c::save(acl::socket_stream* conn, char const* appid,
        char const* userid, char const* fileid, long long filesize,
        char const* filepath) const {
```

TNV/src/04_storage/12_service.cpp

知识讲解

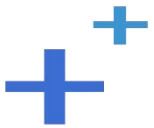
```
file_c file; //文件操作对象

//打开文件
if (file.open(filepath, file_c::O_WRITE) != OK)
        return ERROR;

//依次将接收到的数据块写入文件
long long remain = filesize; //未接收字节数
char rcvwr[STORAGE_RCVWR_SIZE]; //接收写入缓冲区
while (remain) { //还有未接收数据
        //接收数据
        long long bytes = std::min(remain, (long long)sizeof(rcvwr));
        long long count = conn->read(rcvwr, bytes);
        if (count < 0) {
                logger_error("read fail: %s, bytes: %lld, from: %s",
```



TNV/src/04_storage/12_service.cpp

知识讲解

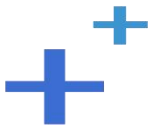
```
                        acl::last_serror(), bytes, conn->get_peer());
                file.close();
                return SOCKET_ERROR;
            }
            //写入文件
            if (file.write(rcvwr, count) != OK) {
                file.close();
                return ERROR;
            }
            //未收递减
            remain -= count;
        }

        //关闭文件
        file.close();
```



TNV/src/04_storage/12_service.cpp

知识讲解

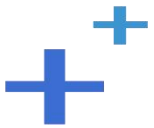
```
    db_c db; //数据库访问对象

    //连接数据库
    if (db.connect() != OK)
            return ERROR;

    //设置文件ID和路径及大小的对应关系
    if (db.set(appid, userid, fileid, filepath, filesize) != OK) {
            error(conn, -1, "insert database fail, fileid: %s", fileid);
            file.del(filepath);
            return ERROR;
    }

    return OK;
}
```



TNV/src/04_storage/12_service.cpp

知识讲解

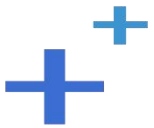
```
//读取并发送文件
int service_c::send(acl::socket_stream* conn, char const* filepath,
        long long offset, long long size) const {
        file_c file; //文件操作对象

        //打开文件
        if (file.open(filepath, file_c::O_READ) != OK)
                return ERROR;

        //设置偏移
        if (offset && file.seek(offset) != OK) {
                file.close();
                return ERROR;
        }
```



TNV/src/04_storage/12_service.cpp

知识讲解

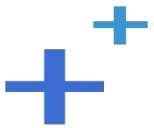
```
//|包体长度|命令|状态|文件内容|
//|  8  | 1| 1|内容大小|
//构造响应头
long long bodylen = size;
long long headlen = HEADLEN;
char head[headlen] = {};
llton(bodylen, head);
head[BODYLEN_SIZE] = CMD_STORAGE_REPLY;
head[BODYLEN_SIZE+COMMAND_SIZE] = 0;

//发送响应头
if (conn->write(head, headlen) < 0) {
        logger_error("write fail: %s, headlen: %lld, to: %s",
                acl::last_serror(), headlen, conn->get_peer());
        file.close();
```



TNV/src/04_storage/12_service.cpp

知识讲解

```
            return SOCKET_ERROR;
        }

        //依次将从文件中读取到的数据块作为响应体的一部分发送出去
        long long remain = size; //未读取字节数
        char rdsnd[STORAGE_RDSND_SIZE]; //读取发送缓冲区
        while (remain) { //还有未读取数据
                //读取文件
                long long count = std::min(remain, (long long)sizeof(rdsnd));
                if (file.read(rdsnd, count) != OK) {
                        file.close();
                        return ERROR;
                }
                //发送数据
                if (conn->write(rdsnd, count) < 0) {
```



TNV/src/04_storage/12_service.cpp

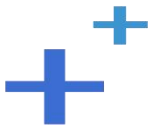
知识讲解

```
                logger_error("write fail: %s, count: %lld, to: %s",
                        acl::last_serror(), count, conn->get_peer());
                file.close();
                return SOCKET_ERROR;
            }
            //未读递减
            remain -= count;
        }

        //关闭文件
        file.close();

        return OK;
    }
```

TNV/src/04_storage/12_service.cpp

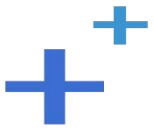
知识讲解

```
////////////////////////////////////////////////////////////////////////

//应答成功
bool service_c::ok(acl::socket_stream* conn) const {
        //|包体长度|命令|状态|
        //|    8    |  1 |  1 |
        //构造响应
        long long bodylen = 0;
        long long resplen = HEADLEN + bodylen;
        char resp[resplen] = {};
        llton(bodylen, resp);
        resp[BODYLEN_SIZE] = CMD_STORAGE_REPLY;
        resp[BODYLEN_SIZE+COMMAND_SIZE] = 0;
```



TNV/src/04_storage/12_service.cpp

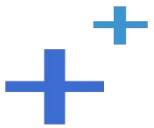

```
        //发送响应
        if (conn->write(resp, resplen) < 0) {
                logger_error("write fail: %s, resplen: %lld, to: %s",
                        acl::last_serror(), resplen, conn->get_peer());
                return false;
        }

        return true;
}

//应答错误
bool service_c::error(acl::socket_stream* conn, short errnumb,
        char const* format, ...) const {
        //错误描述
        char errdesc[ERROR_DESC_SIZE];
```



TNV/src/04_storage/12_service.cpp

知识讲解

```
va_list ap;
va_start(ap, format);
vsnprintf(errdesc, ERROR_DESC_SIZE, format, ap);
va_end(ap);
logger_error("%s", errdesc);
acl::string desc;
desc.format("[%s] %s", g_hostname.c_str(), errdesc);
memset(errdesc, 0, sizeof(errdesc));
strncpy(errdesc, desc.c_str(), ERROR_DESC_SIZE - 1);
size_t desclen = strlen(errdesc);
desclen += desclen != 0;

// |包体长度|命令|状态|错误号|错误描述|
// |    8    |  1 |  1 |    2   | <=1024 |
// 构造响应
```



TNV/src/04_storage/12_service.cpp

知识讲解

```
long long bodylen = ERROR_NUMB_SIZE + desclen;
long long resplen = HEADLEN + bodylen;
char resp[resplen] = {};
llton(bodylen, resp);
resp[BODYLEN_SIZE] = CMD_STORAGE_REPLY;
resp[BODYLEN_SIZE+COMMAND_SIZE] = STATUS_ERROR;
ston(errnumb, resp + HEADLEN);
if (desclen)
        strcpy(resp + HEADLEN + ERROR_NUMB_SIZE, errdesc);

//发送响应
if (conn->write(resp, resplen) < 0) {
        logger_error("write fail: %s, resplen: %lld, to: %s",
                acl::last_serror(), resplen, conn->get_peer());
        return false;
}

return true;
}
```

TNV/src/04_storage/13_tracker.h

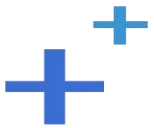
知识讲解

```
//存储服务器
//声明跟踪客户机线程类
//
#pragma once

#include <lib_acl.hpp>
//
//跟踪客户机线程类
//
class tracker_c: public acl::thread {
public:
        //构造函数
        tracker_c(char const* taddr);

        //终止线程
```

TNV/src/04_storage/13_tracker.h

知识讲解

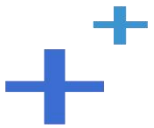
```
        void stop(void);

protected:
        //线程过程
        void* run(void);

private:
        //向跟踪服务器发送加入包
        int join(acl::socket_stream* conn) const;
        //向跟踪服务器发送心跳包
        int beat(acl::socket_stream* conn) const;

        bool         m_stop;  //是否终止
        acl::string m_taddr; //跟踪服务器地址
};
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

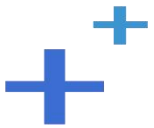
```
//存储服务器
//实现跟踪客户机线程类
//
#include <unistd.h>
#include "02_proto.h"
#include "03_util.h"
#include "01_globals.h"
#include "13_tracker.h"

//构造函数
tracker_c::tracker_c(char const* taddr): m_stop(false), m_taddr(taddr) {
}

//终止线程
void tracker_c::stop(void) {
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

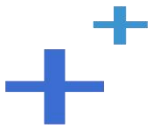
```
        m_stop = true;
}

//线程过程
void* tracker_c::run(void) {
        acl::socket_stream conn;

        while (!m_stop) {
                //连接跟踪服务器
                if (!conn.open(m_taddr, 10, 30)) {
                        logger_error("connect tracker fail, taddr: %s",
                                m_taddr.c_str());
                        sleep(2);
                        continue; //失败重连
                }
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

```
//向跟踪服务器发送加入包
if (join(&conn) != OK) {
        conn.close();
        sleep(2);
        continue; //失败重连
}

time_t last = time(NULL); //上次心跳
while (!m_stop) {
        time_t now = time(NULL); //现在
        //现在离上次心跳已足够久，再跳一次
        if (now - last >= cfg_interval) {
                //向跟踪服务器发送心跳包
                if (beat(&conn) != OK)
                        break; //失败重连
```



TNV/src/04_storage/14_tracker.cpp

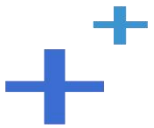
知识讲解

```
                        last = now;
                    }
                    sleep(1);
                }

                conn.close();
            }

            return NULL;
        }

        //向跟踪服务器发送加入包
        int tracker_c::join(acl::socket_stream* conn) const {
            //|包体长度|命令|状态|storage_join_body_t|
            //|  8  | 1| 1|   包体长度    |
```

TNV/src/04_storage/14_tracker.cpp

知识讲解

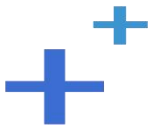
```
//构造请求
long long bodylen = sizeof(storage_join_body_t);
long long requlen = HEADLEN + bodylen;
char requ[requlen] = {};
llton(bodylen, requ);
requ[BODYLEN_SIZE] = CMD_TRACKER_JOIN;
requ[BODYLEN_SIZE+COMMAND_SIZE] = 0;
storage_join_body_t* sjb = (storage_join_body_t*)(requ + HEADLEN);
strcpy(sjb->sjb_version, g_version);              //版本
strcpy(sjb->sjb_groupname, cfg_gpname);           //组名
strcpy(sjb->sjb_hostname, g_hostname.c_str());    //主机名
ston(cfg_bindport, sjb->sjb_port);                //端口号
lton(g_stime, sjb->sjb_stime);                    //启动时间
lton(time(NULL), sjb->sjb_jtime);                 //加入时间
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

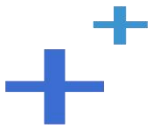
```
//发送请求
if (conn->write(requ, requlen) < 0) {
        logger_error("write fail: %s, requlen: %lld, to: %s",
                acl::last_serror(), requlen, conn->get_peer());
        return SOCKET_ERROR;
}

//接收包头
char head[HEADLEN];
if (conn->read(head, HEADLEN) < 0) {
        logger_error("read fail: %s, from: %s",
                acl::last_serror(), conn->get_peer());
        return SOCKET_ERROR;
}
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

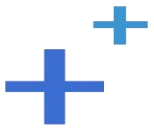
```
//|包体长度|命令|状态|
//|    8    |  1 |  1 |
//解析包头
if ((bodylen = ntoll(head)) < 0) { //包体长度
        logger_error("invalid body length: %lld < 0", bodylen);
        return ERROR;
}
int command = head[BODYLEN_SIZE]; //命令
int status = head[BODYLEN_SIZE+COMMAND_SIZE]; //状态
logger("bodylen: %lld, command: %d, status: %d",
        bodylen, command, status);

//检查命令
if (command != CMD_TRACKER_REPLY) {
        logger_error("unknown command: %d", command);
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

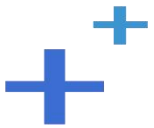
```
        return ERROR;
    }

    //应答成功
    if (!status) return OK;

    //|包体长度|命令|状态|错误号|错误描述|
    //|    8    |  1 |  1 |   2   | <=1024 |
    //检查包体长度
    long long expected = ERROR_NUMB_SIZE + ERROR_DESC_SIZE;
    if (bodylen > expected) {
        logger_error("invalid body length: %lld > %lld", bodylen, expected);
        return ERROR;
    }
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

```
//接收包体
char body[bodylen];
if (conn->read(body, bodylen) < 0) {
        logger_error("read fail: %s, bodylen: %lld, from: %s",
                acl::last_serror(), bodylen, conn->get_peer());
        return SOCKET_ERROR;
}

//解析包体
short errnumb = ntos(body);
char const* errdesc = "";
if (bodylen > ERROR_NUMB_SIZE)
        errdesc = body + ERROR_NUMB_SIZE;

logger_error("join fail, errnumb: %d, errdesc: %s",
```



TNV/src/04_storage/14_tracker.cpp

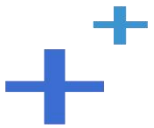
知识讲解

```
                errnumb, errdesc);
        return ERROR;
}

//向跟踪服务器发送心跳包
int tracker_c::beat(acl::socket_stream* conn) const {
        //|包体长度|命令|状态|storage_beat_body_t|
        //|  8  | 1| 1|  包体长度   |
        //构造请求
        long long bodylen = sizeof(storage_beat_body_t);
        long long requlen = HEADLEN + bodylen;
        char requ[requlen] = {};
        llton(bodylen, requ);
        requ[BODYLEN_SIZE] = CMD_TRACKER_BEAT;
        requ[BODYLEN_SIZE+COMMAND_SIZE] = 0;
```

TNV/src/04_storage/14_tracker.cpp

知识讲解

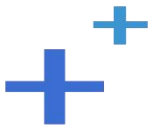
```
storage_beat_body_t* sbb = (storage_beat_body_t*)(requ + HEADLEN);
strcpy(sbb->sbb_groupname, cfg_gpname);          //组名
strcpy(sbb->sbb_hostname, g_hostname.c_str()); //主机名

//发送请求
if (conn->write(requ, requlen) < 0) {
        logger_error("write fail: %s, requlen: %lld, to: %s",
                acl::last_serror(), requlen, conn->get_peer());
        return SOCKET_ERROR;
}

//接收包头
char head[HEADLEN];
if (conn->read(head, HEADLEN) < 0) {
        logger_error("read fail: %s, from: %s",
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

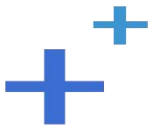
```
                acl::last_serror(), conn->get_peer());
        return SOCKET_ERROR;
}

//|包体长度|命令|状态|
//|    8    |  1 |  1 |
//解析包头
if ((bodylen = ntoll(head)) < 0) { //包体长度
        logger_error("invalid body length: %lld < 0", bodylen);
        return ERROR;
}
int command = head[BODYLEN_SIZE]; //命令
int status = head[BODYLEN_SIZE+COMMAND_SIZE]; //状态
logger("bodylen: %lld, command: %d, status: %d",
        bodylen, command, status);
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

```
//检查命令
if (command != CMD_TRACKER_REPLY) {
        logger_error("unknown command: %d", command);
        return ERROR;
}

//应答成功
if (!status) return OK;

//|包体长度|命令|状态|错误号|错误描述|
//|    8   |  1 |  1 |   2  | <=1024 |
//检查包体长度
long long expected = ERROR_NUMB_SIZE + ERROR_DESC_SIZE;
if (bodylen > expected) {
        logger_error("invalid body length: %lld > %lld",
```



TNV/src/04_storage/14_tracker.cpp

知识讲解

```
                bodylen, expected);
        return ERROR;
}

//接收包体
char body[bodylen];
if (conn->read(body, bodylen) < 0) {
        logger_error("read fail: %s, bodylen: %lld, from: %s",
                acl::last_serror(), bodylen, conn->get_peer());
        return SOCKET_ERROR;
}

//解析包体
short errnumb = ntos(body);
char const* errdesc = "";
if (bodylen > ERROR_NUMB_SIZE)
        errdesc = body + ERROR_NUMB_SIZE;

logger_error("beat fail, errnumb: %d, errdesc: %s", errnumb, errdesc);
return ERROR;
}
```

复习课见