Tedu.cn
达内教育

《分布式流媒体》实训项目

C/C++教学体系

TNV DAY06

直播课

目录

业务服务类(service_c)

三级方法

知识讲解

- 将存储服务器加入组表：join
  - 互斥锁加锁
  - 在组表中查找待加入存储服务器所隶属的组
    - 若找到该组
      - 若待加入存储服务器已在该组的存储服务器列表中
        - 更新该列表中的相应记录
      - 若待加入存储服务器不在该组的存储服务器列表中
        - 将待加入存储服务器加入该列表
    - 若没有该组
      - 将待加入存储服务器所隶属的组加入组表
      - 将待加入存储服务器加入该组的存储服务器列表
  - 互斥锁解锁
  - 返回成功

三级方法

- 将存储服务器标为活动：beat
  - 互斥锁加锁
  - 在组表中查找待标记存储服务器所隶属的组
    - 若找到该组
      - 若待标记存储服务器已在该组的存储服务器列表中
        - 更新该列表中的相应记录
      - 若待标记存储服务器不在该组的存储服务器列表中
        - 处理失败
    - 若没有该组
      - 处理失败
  - 互斥锁解锁
  - 返回处理结果

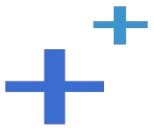
知识讲解

三级方法

知识讲解

- 响应客户机存储服务器地址列表：saddrs
  - 应用ID是否合法
  - 应用ID是否存在
  - 根据用户ID获取其对应的组名
  - 根据组名获取存储服务器地址列表
  - 构造响应
  - 发送响应
  - 返回成功

包头			包体	
包体长度	命令(100)	状态	组名	存储服务器地址列表
8	1	1	包体长度	



三级方法

知识讲解

- 根据用户ID获取其对应的组名：group_of_user
  - 实例化数据库访问对象
  - 连接数据库
  - 根据用户ID获取其对应的组名
  - 组名为空表示该用户没有组，为其随机分配一个
    - 获取全部组名
    - 随机抽取组名
    - 设置用户ID和组名的对应关系
  - 返回成功

三级方法

知识讲解

- 根据组名获取存储服务器地址列表：saddrs_of_group
  - 互斥锁加锁
  - 根据组名在组表中查找特定组
    - 若找到该组
      - 若该组的存储服务器列表非空
        - 在该组的存储服务器列表中，从随机位置开始最多抽取三台活动存储服务器
        - 若没有处于活动状态的存储服务器
          - 处理失败
      - 若该组的存储服务器列表为空
        - 处理失败
    - 若没有该组
      - 处理失败
  - 互斥锁解锁
  - 返回处理结果

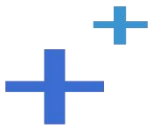
底层方法

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(100)	状态
8	1	1

包头			包体	
包体长度	命令(100)	状态	错误号	错误描述
8	1	1	2	<=1024

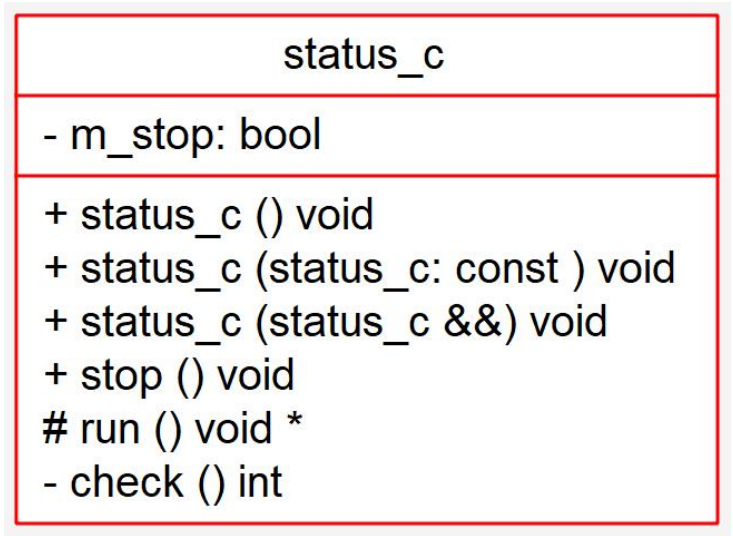


状态检查线程类(status_c)

属性和构造

知识讲解

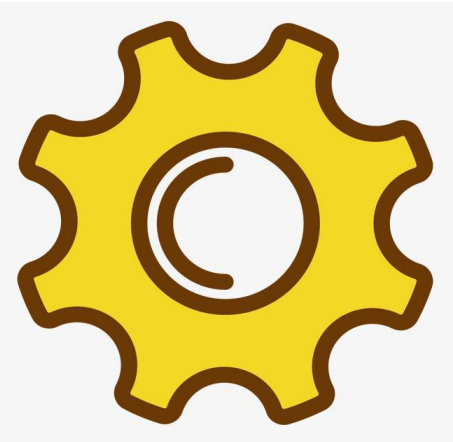
- 成员变量
  - 是否终止：m_stop
- 构造函数：status_c
  - 初始化m_stop为false



方法

知识讲解

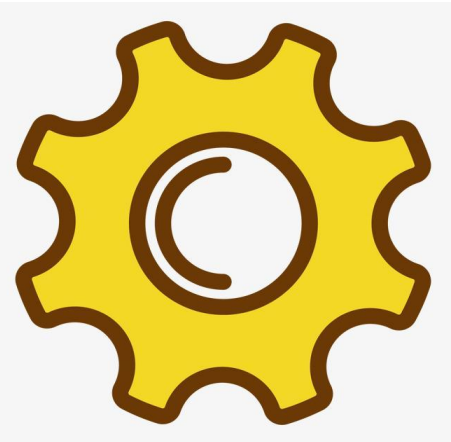
- 终止线程：stop
  - 将m_stop置为true
- 线程过程：run
  - 若线程不终止，则持续循环
    - 获取当前时间
    - 若当前时间距离上次检查时间已足够久
      - 检查存储服务器状态
      - 更新上次检查时间为当前时间
  - 退出循环，线程终止



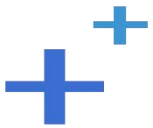
方法

- 线程过程：run

```
for (time_t last = time(NULL); !m_stop; sleep(1)) {
        time_t now = time(NULL); // 现在

        // 若现在距离最近一次检查存储服务器状态已足够久
        if (now - last >= cfg_interval) {
                check(); // 检查存储服务器状态
                last = now; // 更新最近一次检查时间
        }
}
```



知识讲解

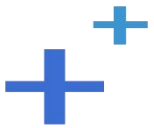


附录：程序清单

TNV/src/02_tracker/08_service.cpp

知识讲解

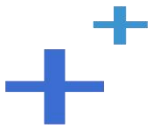
```
//将存储服务器加入组表
int service_c::join(storage_join_t const* sj, char const* saddr) const {
        //互斥锁加锁
        if ((errno = pthread_mutex_lock(&g_mutex))) {
                logger_error("call pthread_mutex_lock fail: %s",
                        strerror(errno));
                return ERROR;
        }

        //在组表中查找待加入存储服务器所隶属的组
        std::map<std::string, std::list<storage_info_t> >::iterator
                group = g_groups.find(sj->sj_groupname);
        if (group != g_groups.end()) { //若找到该组
                //遍历该组的存储服务器列表
                std::list<storage_info_t>::iterator si;
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
for (si = group->second.begin();
        si != group->second.end(); ++si)
        //若待加入存储服务器已在该列表中
        if (!strcmp(si->si_hostname, sj->sj_hostname) &&
                !strcmp(si->si_addr, saddr)) {
                //更新该列表中的相应记录
                strcpy(si->si_version, sj->sj_version); //版本
                si->si_port   = sj->sj_port;            //端口号
                si->si_stime  = sj->sj_stime;           //启动时间
                si->si_jtime  = sj->sj_jtime;           //加入时间
                si->si_btime  = sj->sj_jtime;           //心跳时间
                si->si_status = STORAGE_STATUS_ONLINE;  //状态
                break;
        }
//若待加入存储服务器不在该列表中
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
        if (si == group->second.end()) {
                //将待加入存储服务器加入该列表
                storage_info_t si;
                strcpy(si.si_version,  sj->sj_version);  //版本
                strcpy(si.si_hostname, sj->sj_hostname); //主机名
                strcpy(si.si_addr,     saddr);          //IP地址
                si.si_port   = sj->sj_port;             //端口号
                si.si_stime  = sj->sj_stime;            //启动时间
                si.si_jtime  = sj->sj_jtime;            //加入时间
                si.si_btime  = sj->sj_jtime;            //心跳时间
                si.si_status = STORAGE_STATUS_ONLINE;   //状态
                group->second.push_back(si);
        }
}
else { //若没有该组
```

TNV/src/02_tracker/08_service.cpp

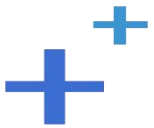
知识讲解

```
        //将待加入存储服务器所隶属的组加入组表
        g_groups[sj->sj_groupname] = std::list<storage_info_t>();
        //将待加入存储服务器加入该组的存储服务器列表
        storage_info_t si;
        strcpy(si.si_version,   sj->sj_version);  //版本
        strcpy(si.si_hostname,  sj->sj_hostname); //主机名
        strcpy(si.si_addr,      saddr);           //IP地址
        si.si_port    = sj->sj_port;              //端口号
        si.si_stime   = sj->sj_stime;             //启动时间
        si.si_jtime   = sj->sj_jtime;             //加入时间
        si.si_btime   = sj->sj_jtime;             //心跳时间
        si.si_status = STORAGE_STATUS_ONLINE;     //状态
        g_groups[sj->sj_groupname].push_back(si);
    }
```

TNV/src/02_tracker/08_service.cpp

知识讲解

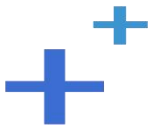
```
        //互斥锁解锁
        if ((errno = pthread_mutex_unlock(&g_mutex))) {
                logger_error("call pthread_mutex_unlock fail: %s",
                        strerror(errno));
                return ERROR;
        }

        return OK;
}

//将存储服务器标为活动
int service_c::beat(char const* groupname, char const* hostname,
        char const* saddr) const {
        //互斥锁加锁
        if ((errno = pthread_mutex_lock(&g_mutex))) {
```



TNV/src/02_tracker/08_service.cpp

知识讲解

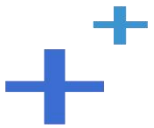
```
        logger_error("call pthread_mutex_lock fail: %s",
                strerror(errno));
        return ERROR;
}

int result = OK;

//在组表中查找待标记存储服务器所隶属的组
std::map<std::string, std::list<storage_info_t> >::iterator
        group = g_groups.find(groupname);
if (group != g_groups.end()) { //若找到该组
        //遍历该组的存储服务器列表
        std::list<storage_info_t>::iterator si;
        for (si = group->second.begin();
                si != group->second.end(); ++si)
```



TNV/src/02_tracker/08_service.cpp

知识讲解

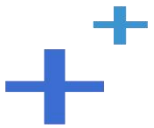
```
                //若待标记存储服务器已在该列表中
                if (!strcmp(si->si_hostname, hostname) &&
                        !strcmp(si->si_addr, saddr)) {
                        //更新该列表中的相应记录
                        si->si_btime  = time(NULL);             //心跳时间
                        si->si_status = STORAGE_STATUS_ACTIVE; //状态
                        break;
                }
        //若待标记存储服务器不在该列表中
        if (si == group->second.end()) {
                logger_error("storage not found, groupname: %s, "
                        "hostname: %s, saddr: %s", groupname, hostname, saddr);
                result = ERROR;
        }
}
```



TNV/src/02_tracker/08_service.cpp

知识讲解

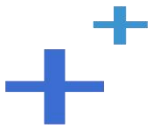
```
        else { //若没有该组
                logger_error("group not found, groupname: %s", groupname);
                result = ERROR;
        }

        //互斥锁解锁
        if ((errno = pthread_mutex_unlock(&g_mutex))) {
                logger_error("call pthread_mutex_unlock fail: %s",
                        strerror(errno));
                return ERROR;
        }

        return result;
}
```



TNV/src/02_tracker/08_service.cpp

知识讲解

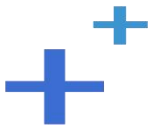
```
//响应客户机存储服务器地址列表
int service_c::saddrs(acl::socket_stream* conn,
        char const* appid, char const* userid) const {
        //应用ID是否合法
        if (valid(appid) != OK) {
                error(conn, -1, "invalid appid: %s", appid);
                return ERROR;
        }

        //应用ID是否存在
        if (std::find(g_appids.begin(), g_appids.end(),
                appid) == g_appids.end()) {
                error(conn, -1, "unknown appid: %s", appid);
                return ERROR;
        }
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
//根据用户ID获取其对应的组名
std::string groupname;
if (group_of_user(appid, userid, groupname) != OK) {
        error(conn, -1, "get groupname fail");
        return ERROR;
}

//根据组名获取存储服务器地址列表
std::string saddrs;
if (saddrs_of_group(groupname.c_str(), saddrs) != OK) {
        error(conn, -1, "get storage address fail");
        return ERROR;
}

logger("appid: %s, userid: %s, groupname: %s, saddrs: %s",
```



TNV/src/02_tracker/08_service.cpp

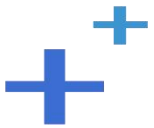
知识讲解

```
        appid, userid, groupname.c_str(), saddrs.c_str());

//|包体长度|命令|状态|组名|存储服务器地址列表|
//|  8  | 1| 1|    包体长度     |
//构造响应
long long bodylen = STORAGE_GROUPNAME_MAX + 1 + saddrs.size() + 1;
long long resplen = HEADLEN + bodylen;
char resp[resplen] = {};
llton(bodylen, resp);
resp[BODYLEN_SIZE] = CMD_TRACKER_REPLY;
resp[BODYLEN_SIZE+COMMAND_SIZE] = 0;
strncpy(resp + HEADLEN, groupname.c_str(), STORAGE_GROUPNAME_MAX);
strcpy(resp + HEADLEN + STORAGE_GROUPNAME_MAX + 1, saddrs.c_str());

//发送响应
```

TNV/src/02_tracker/08_service.cpp

知识讲解

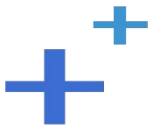
```
        if (conn->write(resp, resplen) < 0) {
                logger_error("write fail: %s, resplen: %lld, to: %s",
                        acl::last_serror(), resplen, conn->get_peer());
                return ERROR;
        }

        return OK;
}

//根据用户ID获取其对应的组名
int service_c::group_of_user(char const* appid,
        char const* userid, std::string& groupname) const {
        db_c db; //数据库访问对象

        //连接数据库
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
if (db.connect() != OK)
        return ERROR;

//根据用户ID获取其对应的组名
if (db.get(userid, groupname) != OK)
        return ERROR;

//组名为空表示该用户没有组，为其随机分配一个
if (groupname.empty()) {
        logger("groupname is empty, appid: %s, userid: %s, allocate one",
                appid, userid);

        //获取全部组名
        std::vector<std::string> groupnames;
        if (db.get(groupnames) != OK)
```



TNV/src/02_tracker/08_service.cpp

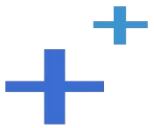
知识讲解

```
            return ERROR;
        if (groupnames.empty()) {
            logger_error("groupnames is empty, appid: %s, userid: %s",
                    appid, userid);
            return ERROR;
        }

        //随机抽取组名
        srand(time(NULL));
        groupname = groupnames[rand() % groupnames.size()];

        //设置用户ID和组名的对应关系
        if (db.set(appid, userid, groupname.c_str()) != OK)
            return ERROR;
    }
```

TNV/src/02_tracker/08_service.cpp

知识讲解

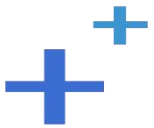
```
        return OK;
}

//根据组名获取存储服务器地址列表
int service_c::saddrs_of_group(char const* groupname,
        std::string& saddrs) const {
        //互斥锁加锁
        if ((errno = pthread_mutex_lock(&g_mutex))) {
                logger_error("call pthread_mutex_lock fail: %s",
                        strerror(errno));
                return ERROR;
        }

        int result = OK;
```



TNV/src/02_tracker/08_service.cpp

知识讲解

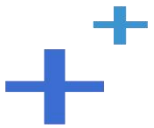
```
//根据组名在组表中查找特定组
std::map<std::string, std::list<storage_info_t> >::const_iterator
        group = g_groups.find(groupname);
if (group != g_groups.end()) { //若找到该组
        if (!group->second.empty()) { //若该组的存储服务器列表非空
                //在该组的存储服务器列表中，从随机位置开
                //始最多抽取三台处于活动状态的存储服务器
                srand(time(NULL));
                int nsis = group->second.size();
                int nrand = rand() % nsis;
                std::list<storage_info_t>::const_iterator si =
                        group->second.begin();
                int nacts = 0;
                for (int i = 0; i < nsis + nrand; ++i, ++si) {
                        if (si == group->second.end())
```



TNV/src/02_tracker/08_service.cpp

知识讲解

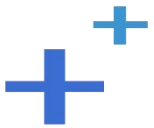
```
                si = group->second.begin();
            logger("i: %d, nrand: %d, addr: %s, port: %u, "
                "status: %d", i, nrand, si->si_addr, si->si_port,
                si->si_status);
            if (i >= nrand && si->si_status ==
                STORAGE_STATUS_ACTIVE) {
                char saddr[256];
                sprintf(saddr, "%s:%d", si->si_addr, si->si_port);
                saddrs += saddr;
                saddrs += ";";
                if (++nacts >= 3)
                    break;
            }
        }
        if (!nacts) { //若没有处于活动状态的存储服务器
```



TNV/src/02_tracker/08_service.cpp

知识讲解

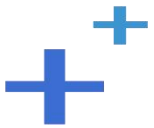
```
                                logger_error("no active storage in group %s",
                                        groupname);
                                result = ERROR;
                        }
                }
                else { //若该组的存储服务器列表为空
                        logger_error("no storage in group %s", groupname);
                        result = ERROR;
                }
        }
        else { //若没有该组
                logger_error("not found group %s", groupname);
                result = ERROR;
        }
```



TNV/src/02_tracker/08_service.cpp

知识讲解

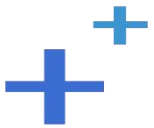
```
        //互斥锁解锁
        if ((errno = pthread_mutex_unlock(&g_mutex))) {
                logger_error("call pthread_mutex_unlock fail: %s",
                        strerror(errno));
                return ERROR;
        }

        return result;
}

////////////////////////////////////////////////////////////////////////

//应答成功
bool service_c::ok(acl::socket_stream* conn) const {
        //|包体长度|命令|状态|
```



TNV/src/02_tracker/08_service.cpp

知识讲解

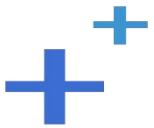
```
// |    8    |  1 |  1 |
//构造响应
long long bodylen = 0;
long long resplen = HEADLEN + bodylen;
char resp[resplen] = {};
llton(bodylen, resp);
resp[BODYLEN_SIZE] = CMD_TRACKER_REPLY;
resp[BODYLEN_SIZE+COMMAND_SIZE] = 0;

//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;
}
```



TNV/src/02_tracker/08_service.cpp

知识讲解

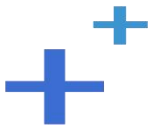
```
        return true;
}

//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];
        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);
```



TNV/src/02_tracker/08_service.cpp

知识讲解

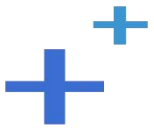
```
memset(errdesc, 0, sizeof(errdesc));
strncpy(errdesc, desc.c_str(), ERROR_DESC_SIZE - 1);
size_t desclen = strlen(errdesc);
desclen += desclen != 0;

//|包体长度|命令|状态|错误号|错误描述|
//|    8    |  1 |  1 |   2   | <=1024 |
//构造响应
long long bodylen = ERROR_NUMB_SIZE + desclen;
long long resplen = HEADLEN + bodylen;
char resp[resplen] = {};
llton(bodylen, resp);
resp[BODYLEN_SIZE] = CMD_TRACKER_REPLY;
resp[BODYLEN_SIZE+COMMAND_SIZE] = STATUS_ERROR;
ston(errnumb, resp + HEADLEN);
```



TNV/src/02_tracker/08_service.cpp

知识讲解

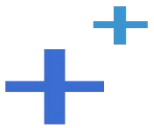
```
        if (desclen)
                strcpy(resp + HEADLEN + ERROR_NUMB_SIZE, errdesc);

        //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/02_tracker/09_status.h

知识讲解

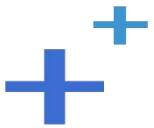
```
//跟踪服务器
//声明存储服务器状态检查线程类
//
#pragma once

#include <lib_acl.hpp>
//
//存储服务器状态检查线程类
//
class status_c: public acl::thread {
public:
        //构造函数
        status_c(void);

        //终止线程
```



TNV/src/02_tracker/09_status.h

知识讲解

```
        void stop(void);

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

private:
        //检查存储服务器状态
        int check(void) const;

        bool m_stop; //是否终止
};
```



TNV/src/02_tracker/10_status.cpp

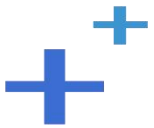
知识讲解

```
//跟踪服务器
//实现存储服务器状态检查线程类
//
#include <unistd.h>
#include "01_globals.h"
#include "09_status.h"

//构造函数
status_c::status_c(void): m_stop(false) {
}

//终止线程
void status_c::stop(void) {
        m_stop = true;
}
```

TNV/src/02_tracker/10_status.cpp

知识讲解

```
//线程过程
void* status_c::run(void) {
        for (time_t last = time(NULL); !m_stop; sleep(1)) {
                time_t now = time(NULL); //现在

                //若现在距离最近一次检查存储服务器状态已足够久
                if (now - last >= cfg_interval) {
                        check(); //检查存储服务器状态
                        last = now; //更新最近一次检查时间
                }
        }

        return NULL;
}
```



复习课见