Tedu.cn
达内教育

《分布式流媒体》实训项目

C/C++教学体系

Tedu.cn
达内教育

TNV DAY05

直播课

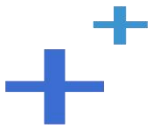
目录

数据库访问类(db_c)

属性、构造和析构

知识讲解

- 成员变量
  - MySQL对象：m_mysql
- 构造函数：db_c
  - 创建MySQL对象
- 析构函数：~db_c
  - 销毁MySQL对象

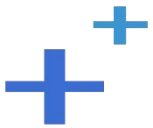


方法

知识讲解

- 连接数据库：connect
  - 遍历MySQL地址表，尝试连接数据库
- 根据用户ID获取其对应的组名：get
  - 先尝试从缓存中获取与用户ID对应的组名
  - 缓存中没有再查询数据库
  - 获取查询结果
  - 获取结果记录
  - 将用户ID和组名的对应关系保存在缓存中
  - 返回成功

id	appid	userid	group_name	create_time	update_time
1	tnvideo	tnv001	group001	2020-11-01 12:55:03	2020-11-01 12:55:03

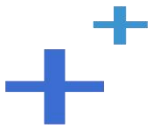


方法

知识讲解

- 设置用户ID和组名的对应关系：set
  - 插入一条记录
  - 检查插入结果
  - 返回成功
- 获取全部组名：get
  - 查询全部组名
  - 获取查询结果
  - 获取结果记录
  - 返回成功

id	group_name	create_time	update_time
1	group001	2020-11-01 12:53:56	2020-11-01 12:53:56



业务服务类(service_c)

一级方法

知识讲解

- 业务处理：business
  - 解析包头
    - 包体长度
    - 命令
    - 状态
  - 根据命令执行具体业务处理
    - 处理来自存储服务器的加入包
    - 处理来自存储服务器的心跳包
    - 处理来自客户机的获取存储服务器地址列表请求
    - 处理来自客户机的获取组列表请求
  - 返回处理结果

二级方法

知识讲解

- 处理来自存储服务器的加入包：join
  - 检查包体长度
  - 接收包体
  - 解析包体
    - 版本
    - 组名
    - 主机名
    - 端口号
    - 启动时间
    - 加入时间
  - 将存储服务器加入组表
  - 响应成功报文

包头			包体					
包体长度	命令(10)	状态	版本	组名	主机名	端口号	启动时间	加入时间
8	1	1	包体长度					

二级方法

知识讲解

- 处理来自存储服务器的心跳包：beat
  - 检查包体长度
  - 接收包体
  - 解析包体
    - 组名
    - 主机名
  - 将存储服务器标为活动
  - 响应成功报文

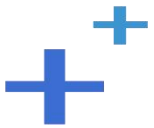
包头			包体	
包体长度	命令(11)	状态	组名	主机名
8	1	1	包体长度	

二级方法

- 处理来自客户机的获取存储服务器地址列表请求：saddrs
  - 检查包体长度
  - 接收包体
  - 解析包体
    - 应用ID
    - 用户ID
    - 文件ID
  - 响应客户机存储服务器地址列表

包头			包体		
包体长度	命令(12)	状态	应用ID	用户ID	文件ID
8	1	1	16	256	128

知识讲解



二级方法

知识讲解

- 处理来自客户机的获取组列表请求：groups
  - 互斥锁加锁
  - 遍历组表中的每一个组
    - 遍历该组中的每一台存储服务器
      - 格式化存储服务器字符串，并将其拼接到单组字符串中
      - 累加活动存储服务器数
    - 将活动存储服务器数格式化到单组字符串中，并将后者拼接到全组字符串中
  - 删除全组字符串的结尾换行符
  - 互斥锁解锁
  - 构造响应
  - 发送响应
  - 返回成功

包头		
包体长度	命令(13)	状态
8	1	1

包头			包体
包体长度	命令(100)	状态	组列表
8	1	1	包体长度

附录：程序清单

TNV/src/02_tracker/05_db.h

知识讲解

```
//跟踪服务器
//声明数据库访问类
//
#pragma once

#include <string>
#include <vector>
#include <mysql.h>
//
//数据库访问类
//
class db_c {
public:
        //构造函数
        db_c(void);
```

TNV/src/02_tracker/05_db.h

知识讲解

```
        //析构函数
        ~db_c(void);

        //连接数据库
        int connect(void);

        //根据用户ID获取其对应的组名
        int get(char const* userid, std::string& groupname) const;
        //设置用户ID和组名的对应关系
        int set(char const* appid, char const* userid,
                char const* groupname) const;
        //获取全部组名
        int get(std::vector<std::string>& groupnames) const;

private:
        MYSQL* m_mysql; //MySQL对象
};
```

TNV/src/02_tracker/06_db.cpp

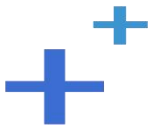
知识讲解

```
//跟踪服务器
//实现数据库访问类
//
#include "01_globals.h"
#include "03_cache.h"
#include "05_db.h"

//构造函数
db_c::db_c(void): m_mysql(mysql_init(NULL)) { //创建MySQL对象
        if (!m_mysql)
                logger_error("create dao fail: %s", mysql_error(m_mysql));
}

//析构函数
db_c::~db_c(void) {
```

TNV/src/02_tracker/06_db.cpp

知识讲解

```
        //销毁MySQL对象
        if (m_mysql) {
                mysql_close(m_mysql);
                m_mysql = NULL;
        }
}

//连接数据库
int db_c::connect(void) {
        MYSQL* mysql = m_mysql;

        //遍历MySQL地址表，尝试连接数据库
        for (std::vector<std::string>::const_iterator maddr =
                g_maddrs.begin(); maddr != g_maddrs.end(); ++maddr)
                if ((m_mysql = mysql_real_connect(mysql, maddr->c_str(),
```



TNV/src/02_tracker/06_db.cpp

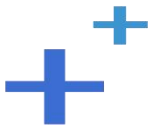
知识讲解

```
                "root", "123456", "tnv_trackerdb", 0, NULL, 0)))
                return OK;

        logger_error("connect database fail: %s",
                mysql_error(m_mysql = mysql));
        return ERROR;
}

//根据用户ID获取其对应的组名
int db_c::get(char const* userid, std::string& groupname) const {
        //先尝试从缓存中获取与用户ID对应的组名
        cache_c cache;
        acl::string key;
        key.format("userid:%s", userid);
        acl::string value;
```

TNV/src/02_tracker/06_db.cpp

知识讲解

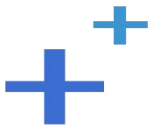
```
if (cache.get(key, value) == OK) {
        groupname = value.c_str();
        return OK;
}

//缓存中没有再查询数据库
acl::string sql;
sql.format("SELECT group_name FROM t_router WHERE userid='%s';",
        userid);
if (mysql_query(m_mysql, sql.c_str())) {
        logger_error("query database fail: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}
```



TNV/src/02_tracker/06_db.cpp

知识讲解

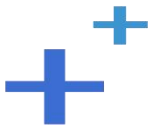
```
//获取查询结果
MYSQL_RES* res = mysql_store_result(m_mysql);
if (!res) {
        logger_error("result is null: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}

//获取结果记录
MYSQL_ROW row = mysql_fetch_row(res);
if (!row)
        logger_warn("result is empty: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
else {
        groupname = row[0];
```



TNV/src/02_tracker/06_db.cpp

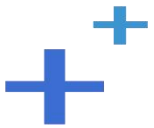
知识讲解

```
                //将用户ID和组名的对应关系保存在缓存中
                cache.set(key, groupname.c_str());
        }

        return OK;
}

//设置用户ID和组名的对应关系
int db_c::set(char const* appid, char const* userid,
        char const* groupname) const {
        //插入一条记录
        acl::string sql;
        sql.format("INSERT INTO t_router SET "
                "appid='%s', userid='%s', group_name='%s';",
                appid, userid, groupname);
```



TNV/src/02_tracker/06_db.cpp

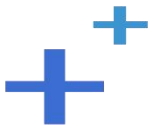

```
if (mysql_query(m_mysql, sql.c_str())) {
        logger_error("insert database fail: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}

//检查插入结果
MYSQL_RES* res = mysql_store_result(m_mysql);
if (!res && mysql_field_count(m_mysql)) {
        logger_error("insert database fail: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}

return OK;
```



TNV/src/02_tracker/06_db.cpp

知识讲解

```
}

//获取全部组名
int db_c::get(std::vector<std::string>& groupnames) const {
        //查询全部组名
        acl::string sql;
        sql.format("SELECT group_name FROM t_groups_info;");
        if (mysql_query(m_mysql, sql.c_str())) {
                logger_error("query database fail: %s, sql: %s",
                        mysql_error(m_mysql), sql.c_str());
                return ERROR;
        }

        //获取查询结果
        MYSQL_RES* res = mysql_store_result(m_mysql);
```



TNV/src/02_tracker/06_db.cpp

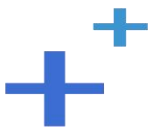
知识讲解

```
    if (!res) {
        logger_error("result is null: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
    }

    //获取结果记录
    int nrows = mysql_num_rows(res);
    for (int i = 0; i < nrows; ++i) {
        MYSQL_ROW row = mysql_fetch_row(res);
        if (!row)
            break;
        groupnames.push_back(row[0]);
    }

    return OK;
}
```

TNV/src/02_tracker/07_service.h

知识讲解

```
//跟踪服务器
//声明业务服务类
//
#pragma once

#include <lib_acl.hpp>
#include "01_types.h"
//
//业务服务类
//
class service_c {
public:
        //业务处理
        bool business(acl::socket_stream* conn, char const* head) const;
```



TNV/src/02_tracker/07_service.h

知识讲解

```
private:
        //处理来自存储服务器的加入包
        bool join(acl::socket_stream* conn, long long bodylen) const;
        //处理来自存储服务器的心跳包
        bool beat(acl::socket_stream* conn, long long bodylen) const;
        //处理来自客户机的获取存储服务器地址列表请求
        bool saddrs(acl::socket_stream* conn, long long bodylen) const;
        //处理来自客户机的获取组列表请求
        bool groups(acl::socket_stream* conn) const;

        //将存储服务器加入组表
        int join(storage_join_t const* sj, char const* saddr) const;

        //将存储服务器标为活动
        int beat(char const* groupname, char const* hostname,
                char const* saddr) const;
```

TNV/src/02_tracker/07_service.h

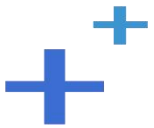
知识讲解

```
        //响应客户机存储服务器地址列表
        int saddrs(acl::socket_stream* conn,
                char const* appid, char const* userid) const;
        //根据用户ID获取其对应的组名
        int group_of_user(char const* appid,
                char const* userid, std::string& groupname) const;
        //根据组名获取存储服务器地址列表
        int saddrs_of_group(char const* groupname,
                std::string& saddrs) const;

        //应答成功
        bool ok(acl::socket_stream* conn) const;
        //应答错误
        bool error(acl::socket_stream* conn, short errnumb,
                char const* format, ...) const;
};
```

TNV/src/02_tracker/08_service.cpp

知识讲解

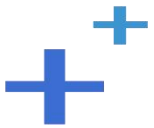
```
//跟踪服务器
//实现业务服务类
//
#include <algorithm>
#include "02_proto.h"
#include "03_util.h"
#include "01_globals.h"
#include "05_db.h"
#include "07_service.h"

//业务处理
bool service_c::business(acl::socket_stream* conn,
        char const* head) const {
        //|包体长度|命令|状态| 包体 |
        //|  8  | 1| 1|包体长度|
```



TNV/src/02_tracker/08_service.cpp

知识讲解

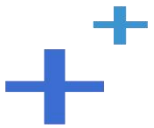
```
//解析包头
long long bodylen = ntoll(head); //包体长度
if (bodylen < 0) {
        error(conn, -1, "invalid body length: %lld < 0", bodylen);
        return false;
}
int command = head[BODYLEN_SIZE]; //命令
int status = head[BODYLEN_SIZE+COMMAND_SIZE]; //状态
logger("bodylen: %lld, command: %d, status: %d",
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

bool result;

//根据命令执行具体业务处理
switch (command) {
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
case CMD_TRACKER_JOIN:
        //处理来自存储服务器的加入包
        result = join(conn, bodylen);
        break;

case CMD_TRACKER_BEAT:
        //处理来自存储服务器的心跳包
        result = beat(conn, bodylen);
        break;

case CMD_TRACKER_SADDRS:
        //处理来自客户机的获取存储服务器地址列表请求
        result = saddrs(conn, bodylen);
        break;
```



TNV/src/02_tracker/08_service.cpp

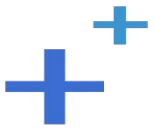
知识讲解

```
                case CMD_TRACKER_GROUPS:
                        //处理来自客户机的获取组列表请求
                        result = groups(conn);
                        break;

                default:
                        error(conn, -1, "unknown command: %d", command);
                        return false;
        }

        return result;
}

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

TNV/src/02_tracker/08_service.cpp

知识讲解

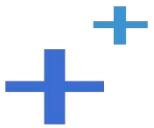
```
//处理来自存储服务器的加入包
bool service_c::join(acl::socket_stream* conn,
        long long bodylen) const {
        //|包体长度|命令|状态|storage_join_body_t|
        //|  8  | 1 | 1 |  包体长度   |
        //检查包体长度
        long long expected = sizeof(storage_join_body_t); //期望包体长度
        if (bodylen != expected) {
                error(conn, -1, "invalid body length: %lld != %lld",
                        bodylen, expected);
                return false;
        }

        //接收包体
        char body[bodylen];
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
if (conn->read(body, bodylen) < 0) {
        logger_error("read fail: %s, bodylen: %lld, from: %s",
                acl::last_serror(), bodylen, conn->get_peer());
        return false;
}

//解析包体
storage_join_t sj;
storage_join_body_t* sjb = (storage_join_body_t*)body;
//版本
strcpy(sj.sj_version, sjb->sjb_version);
//组名
strcpy(sj.sj_groupname, sjb->sjb_groupname);
if (valid(sj.sj_groupname) != OK) {
        error(conn, -1, "invalid groupname: %s", sj.sj_groupname);
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
        return false;
}
//主机名
strcpy(sj.sj_hostname, sjb->sjb_hostname);
//端口号
sj.sj_port = ntos(sjb->sjb_port);
if (!sj.sj_port) {
        error(conn, -1, "invalid port: %u", sj.sj_port);
        return false;
}
//启动时间
sj.sj_stime = ntol(sjb->sjb_stime);
//加入时间
sj.sj_jtime = ntol(sjb->sjb_jtime);
logger("storage join, version: %s, groupname: %s, "
```

TNV/src/02_tracker/08_service.cpp

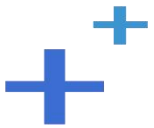
```
                "hostname: %s, port: %u, stime: %s, jtime: %s",
                sj.sj_version, sj.sj_groupname,
                sj.sj_hostname, sj.sj_port,
                std::string(ctime(&sj.sj_stime)).c_str(),
                std::string(ctime(&sj.sj_jtime)).c_str());

        //将存储服务器加入组表
        if (join(&sj, conn->get_peer()) != OK) {
                error(conn, -1, "join into groups fail");
                return false;
        }

        return ok(conn);
}
```

知识讲解

TNV/src/02_tracker/08_service.cpp

知识讲解

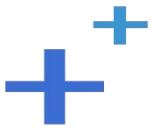
```
//处理来自存储服务器的心跳包
bool service_c::beat(acl::socket_stream* conn,
        long long bodylen) const {
        //|包体长度|命令|状态|storage_beat_body_t|
        //|  8  | 1 | 1 |  包体长度   |
        //检查包体长度
        long long expected = sizeof(storage_beat_body_t); //期望包体长度
        if (bodylen != expected) {
                error(conn, -1, "invalid body length: %lld != %lld",
                        bodylen, expected);
                return false;
        }

        //接收包体
        char body[bodylen];
```



TNV/src/02_tracker/08_service.cpp

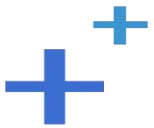
知识讲解

```
if (conn->read(body, bodylen) < 0) {
        logger_error("read fail: %s, bodylen: %lld, from: %s",
                acl::last_serror(), bodylen, conn->get_peer());
        return false;
}

//解析包体
storage_beat_body_t* sbb = (storage_beat_body_t*)body;
//组名
char groupname[STORAGE_GROUPNAME_MAX+1];
strcpy(groupname, sbb->sbb_groupname);
//主机名
char hostname[STORAGE_HOSTNAME_MAX+1];
strcpy(hostname, sbb->sbb_hostname);
logger("storage beat, groupname: %s, hostname: %s",
```



TNV/src/02_tracker/08_service.cpp

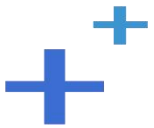

```
                groupname, hostname);

        //将存储服务器标为活动
        if (beat(groupname, hostname, conn->get_peer()) != OK) {
                error(conn, -1, "mark storage as active fail");
                return false;
        }

        return ok(conn);
}

//处理来自客户机的获取存储服务器地址列表请求
bool service_c::saddrs(acl::socket_stream* conn,
        long long bodylen) const {
        //|包体长度|命令|状态|应用ID|用户ID|文件ID|
```



TNV/src/02_tracker/08_service.cpp

知识讲解

```
// |    8    | 1 |  1 |  16  |  256 |  128 |
//检查包体长度
long long expected = APPID_SIZE + USERID_SIZE + FILEID_SIZE;
if (bodylen != expected) {
        error(conn, -1, "invalid body length: %lld != %lld",
                bodylen, expected);
        return false;
}

//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 false;
```



TNV/src/02_tracker/08_service.cpp

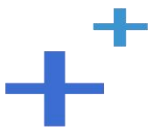
知识讲解

```
    }

    //解析包体
    char appid[APPID_SIZE];
    strcpy(appid, body);
    char userid[USERID_SIZE];
    strcpy(userid, body + APPID_SIZE);
    char fileid[FILEID_SIZE];
    strcpy(fileid, body + APPID_SIZE + USERID_SIZE);

    //响应客户机存储服务器地址列表
    if (saddrs(conn, appid, userid) != OK)
            return false;

    return true;
}
```

TNV/src/02_tracker/08_service.cpp

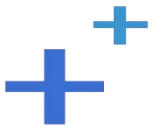
知识讲解

```
//处理来自客户机的获取组列表请求
bool service_c::groups(acl::socket_stream* conn) const {
        //互斥锁加锁
        if ((errno = pthread_mutex_lock(&g_mutex))) {
                logger_error("call pthread_mutex_lock fail: %s",
                        strerror(errno));
                return false;
        }

        acl::string gps; //全组字符串
        gps.format("          COUNT OF GROUPS: %lu\n", g_groups.size());

        //遍历组表中的每一个组
        for (std::map<std::string, std::list<storage_info_t> >::
                const_iterator group = g_groups.begin();
```



TNV/src/02_tracker/08_service.cpp

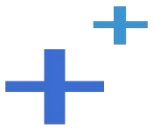

```
group != g_groups.end(); ++group) {
acl::string grp; //单组字符串
grp.format("                GROUPNAME: %s\n"
        "       COUNT OF STORAGES: %lu\n"
        "COUNT OF ACTIVE STORAGES: %s\n",
        group->first.c_str(),
        group->second.size(),
        "%d");

int act = 0; //活动存储服务器数

//遍历该组中的每一台存储服务器
for (std::list<storage_info_t>::const_iterator si =
        group->second.begin(); si != group->second.end(); ++si) {
        acl::string stg; //存储服务器字符串
```



TNV/src/02_tracker/08_service.cpp

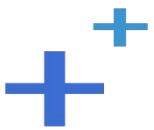
知识讲解

```
stg.format("                    VERSION: %s\n"
           "                   HOSTNAME: %s\n"
           "                    ADDRESS: %s:%u\n"
           "                 STARTUP TIME: %s"
           "                    JOIN TIME: %s"
           "                    BEAT TIME: %s"
           "                      STATUS: ",
           si->si_version,
           si->si_hostname,
           si->si_addr, si->si_port,
           std::string(ctime(&si->si_stime)).c_str(),
           std::string(ctime(&si->si_jtime)).c_str(),
           std::string(ctime(&si->si_btime)).c_str());

switch (si->si_status) {
```



TNV/src/02_tracker/08_service.cpp


```
            case STORAGE_STATUS_OFFLINE:
                stg += "OFFLINE";
                break;
            case STORAGE_STATUS_ONLINE:
                stg += "ONLINE";
                break;
            case STORAGE_STATUS_ACTIVE:
                stg += "ACTIVE";
                ++act;
                break;
            default:
                stg += "UNKNOWN";
                break;
    }
```



TNV/src/02_tracker/08_service.cpp

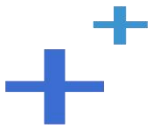
知识讲解

```
                grp += stg + "\n";
            }

            gps += grp.format(grp, act);
        }

        gps = gps.left(gps.size() - 1);

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

TNV/src/02_tracker/08_service.cpp

知识讲解

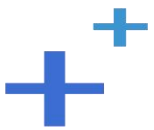
```
//|包体长度|命令|状态| 组列表 |
//|  8  | 1 | 1 |包体长度|
//构造响应
long long bodylen = gps.size() + 1;
long long resplen = HEADLEN + bodylen;
char resp[resplen] = {};
llton(bodylen, resp);
resp[BODYLEN_SIZE] = CMD_TRACKER_REPLY;
resp[BODYLEN_SIZE+COMMAND_SIZE] = 0;
strcpy(resp + HEADLEN, gps.c_str());

//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());
```



TNV/src/02_tracker/08_service.cpp

```
            return false;
        }

        return true;
    }

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

知识讲解



复习课见