Tedu.cn
达内教育

《分布式流媒体》实训项目

C/C++教学体系

TNV DAY11

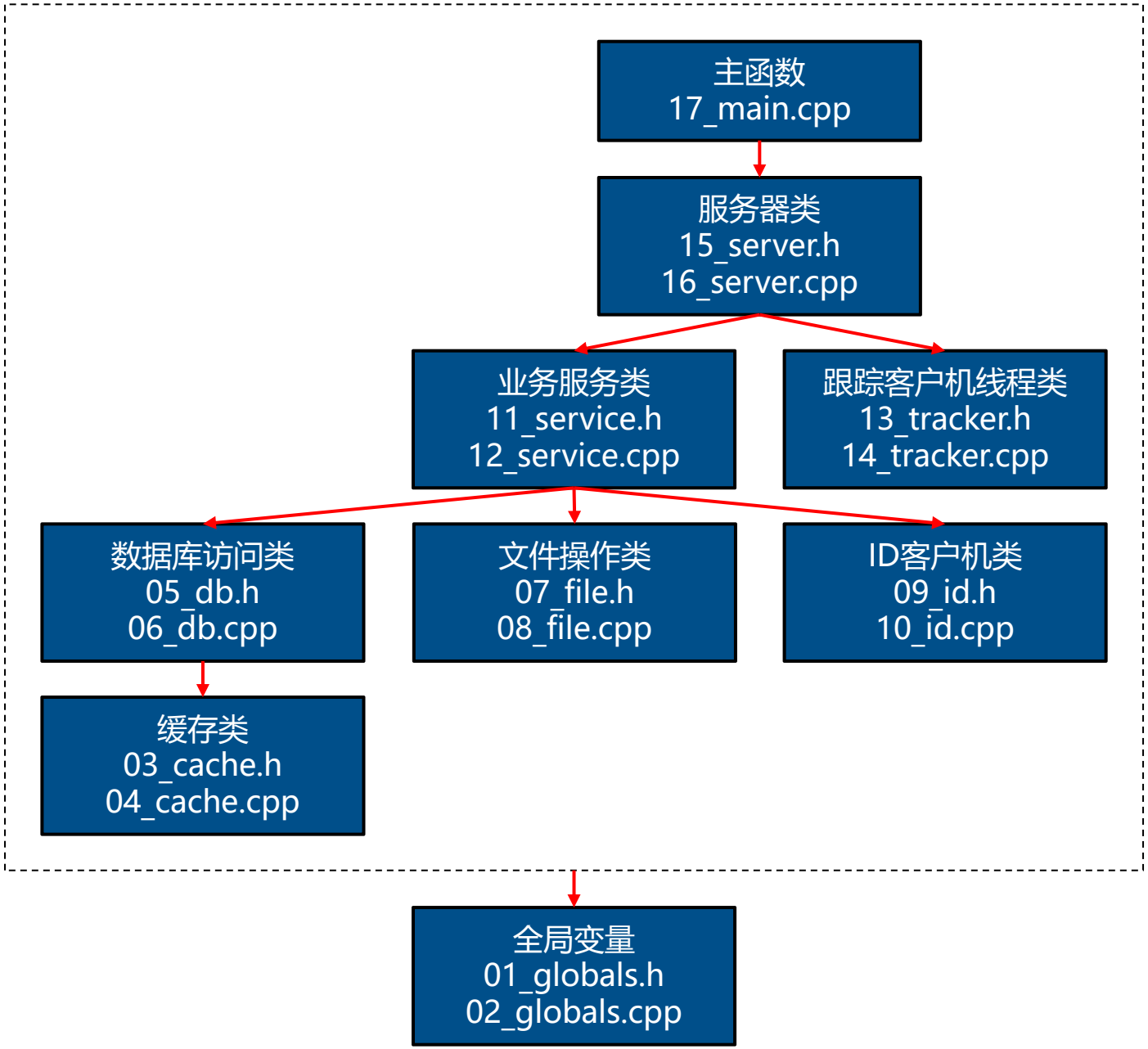
直播课

目录

存储服务器详细设计

组织结构

- 04_storage

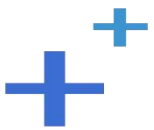


知识讲解

开发计划

知识讲解

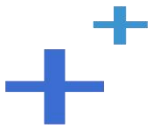
序号	内容	文档/代码	时间
38	声明全局变量	04_storage/01_globals.h	3小时
39	定义全局变量	04_storage/02_globals.cpp	
40	声明缓存类	04_storage/03_cache.h	
41	实现缓存类	04_storage/04_cache.cpp	
42	声明数据库访问类	04_storage/05_db.h	
43	实现数据库访问类	04_storage/06_db.cpp	
44	声明文件操作类	04_storage/07_file.h	3小时
45	实现文件操作类	04_storage/08_file.cpp	
46	声明ID客户机类	04_storage/09_id.h	
47	实现ID客户机类	04_storage/10_id.cpp	



开发计划

知识讲解

序号	内容	文档/代码	时间
48	声明业务服务类	04_storage/11_service.h	6小时
49	实现业务服务类	04_storage/12_service.cpp	
50	声明跟踪客户机线程类	04_storage/13_tracker.h	3小时
51	实现跟踪客户机线程类	04_storage/14_tracker.cpp	
52	声明服务器类	04_storage/15_server.h	
53	实现服务器类	04_storage/16_server.cpp	
54	定义主函数	04_storage/17_main.cpp	
55	构建脚本	04_storage/Makefile	
56	配置文件	etc/storage.cfg	
57	建表脚本	sql/storage.sql	



全局变量

配置信息

知识讲解

- 字符串配置表：cfg_str
  - 隶属组名：cfg_gpname
  - 存储路径表：cfg_spaths
  - 跟踪服务器地址表：cfg_taddrs
  - ID服务器地址表：cfg_iaddrs
  - MySQL地址表：cfg_maddrs
  - Redis地址表：cfg_raddrs

配置信息

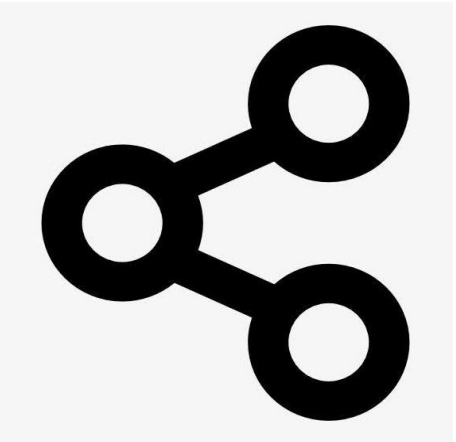
知识讲解

- 整型配置表：cfg_int
  - 绑定端口号：cfg_bindport
  - 心跳间隔秒数：cfg_interval
  - MySQL读写超时：cfg_mtimeout
  - Redis连接池最大连接数：cfg_maxconns
  - Redis连接超时：cfg_ctimeout
  - Redis读写超时：cfg_rtimeout
  - Redis键超时：cfg_ktimeout

共享信息

知识讲解

- 配置共享信息
  - 存储路径表：g_spaths
  - 跟踪服务器地址表：g_taddrs
  - ID服务器地址表：g_iaddrs
  - MySQL地址表：g_maddrs
  - Redis地址表：g_raddrs
- 其它共享信息
  - Redis连接池：g_rconns
  - 主机名：g_hostname
  - 版本：g_version
  - 启动时间：g_stime

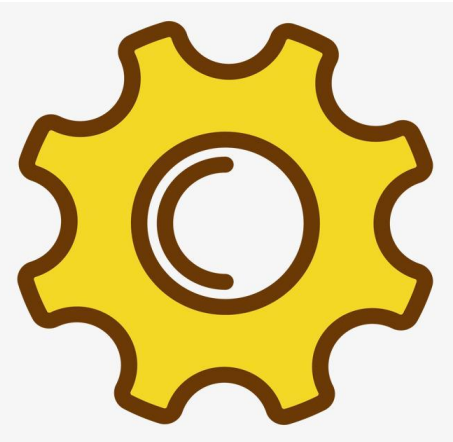


缓存类(cache_c)

方法

知识讲解

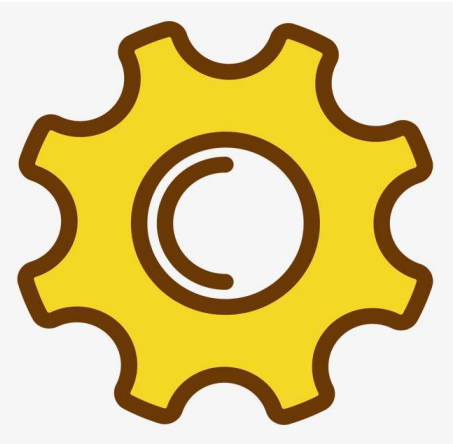
- 根据键获取其值：get
  - 构造键
  - 检查Redis连接池
  - 从连接池中获取一个Redis连接
  - 持有此连接的Redis对象即为Redis客户机
  - 借助Redis客户机根据键获取其值
  - 检查空值
  - 返回成功



方法

知识讲解

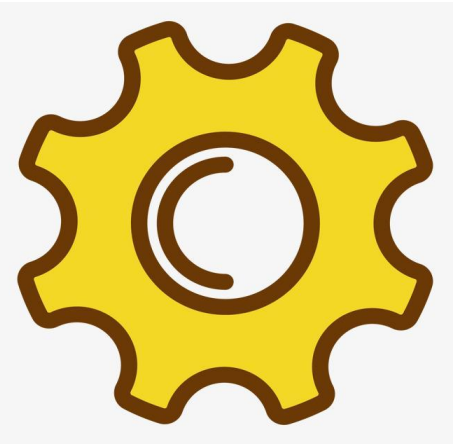
- 设置指定键的值：set
  - 构造键
  - 检查Redis连接池
  - 从连接池中获取一个Redis连接
  - 持有此连接的Redis对象即为Redis客户机
  - 借助Redis客户机设置指定键的值
  - 返回成功



方法

知识讲解

- 删除指定键值对：del
  - 构造键
  - 检查Redis连接池
  - 从连接池中获取一个Redis连接
  - 持有此连接的Redis对象即为Redis客户机
  - 借助Redis客户机删除指定键值对
  - 返回成功

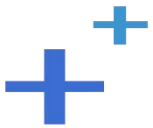


数据库访问类(db_c)

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

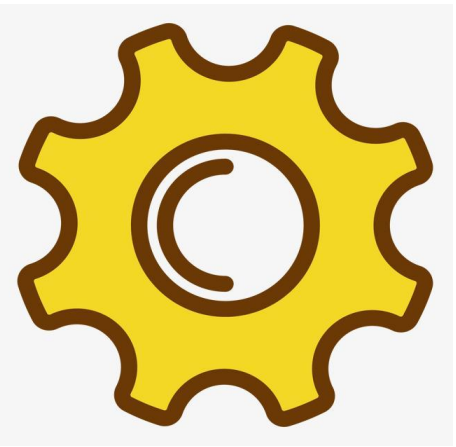
- 连接数据库：connect
  - 遍历MySQL地址表，尝试连接数据库
- 根据文件ID获取其对应的路径及大小：get
  - 先尝试从缓存中获取与文件ID对应的路径及大小，缓存中没有再查询数据库
  - 获取查询结果
  - 获取结果记录
  - 将文件ID和路径及大小的对应关系保存在缓存中
  - 返回成功

id	appid	userid	status	file_path	file_size	create_time	update_time
5f9e3fa7085e5aa25387c8cf7b5f29000117	tnvideo	tnv001	0	../data/000/000/000/5F9E3FA7_000	734529655	2020-11-01 12:55:14	2020-11-01 12:55:14

方法

知识讲解

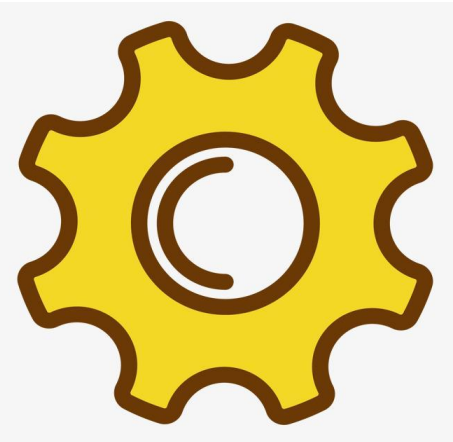
- 设置文件ID和路径及大小的对应关系：set
  - 根据用户ID获取其对应的表名
  - 插入一条记录
  - 检查插入结果
  - 返回成功
- 删除文件ID：del
  - 先从缓存中删除文件ID
  - 再从数据库中删除文件ID
  - 检查删除结果
  - 返回成功



方法

知识讲解

- 根据用户ID获取其对应的表名：table_of_user
  - 计算用户ID的哈希值，取其低31位对3取模再加1
  - 将上述计算结果以左补0两位十进制整数的形式，作为文件信息表名的后缀
    - t_file_01
    - t_file_02
    - t_file_03
- 计算哈希值：hash
  - 将哈希值初始化为0
  - 对被哈希信息从0开始逐字节计算
    - 奇数字节：哈希值 ^= ~(哈希值<<11^字节^哈希值>>5)
    - 偶数字节：哈希值 ^= 哈希值<<7^字节^哈希值>>3
  - 返回哈希值

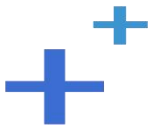


附录：程序清单

TNV/src/04_storage/01_globals.h

知识讲解

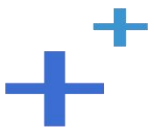
```
//存储服务器
//声明全局变量
//
#pragma once

#include <vector>
#include <string>
#include <lib_acl.hpp>
//
//配置信息
//
extern char* cfg_gpname; //隶属组名
extern char* cfg_spaths; //存储路径表
extern char* cfg_taddrs; //跟踪服务器地址表
extern char* cfg_iaddrs; //ID服务器地址表
```



TNV/src/04_storage/01_globals.h

知识讲解

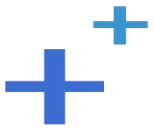
```
extern char* cfg_maddrs; //MySQL地址表
extern char* cfg_raddrs; //Redis地址表
extern acl::master_str_tbl cfg_str[]; //字符串配置表

extern int cfg_bindport; //绑定端口号
extern int cfg_interval; //心跳间隔秒数
extern int cfg_mtimeout; //MySQL读写超时
extern int cfg_maxconns; //Redis连接池最大连接数
extern int cfg_ctimeout; //Redis连接超时
extern int cfg_rtimeout; //Redis读写超时
extern int cfg_ktimeout; //Redis键超时
```



TNV/src/04_storage/01_globals.h

知识讲解

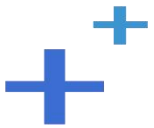
```
extern acl::master_int_tbl cfg_int[]; //整型配置表

extern std::vector<std::string> g_spaths; //存储路径表
extern std::vector<std::string> g_taddrs; //跟踪服务器地址表
extern std::vector<std::string> g_iaddrs; //ID服务器地址表
extern std::vector<std::string> g_maddrs; //MySQL地址表
extern std::vector<std::string> g_raddrs; //Redis地址表
extern acl::redis_client_pool*  g_rconns; //Redis连接池
extern std::string g_hostname; //主机名
extern char const* g_version; //版本
extern time_t g_stime; //启动时间
```



TNV/src/04_storage/02_globals.cpp

知识讲解

```
//存储服务器
//定义全局变量
//
#include "01_globals.h"
//
//配置信息
//
char* cfg_gpname; //隶属组名
char* cfg_spaths; //存储路径表
char* cfg_taddrs; //跟踪服务器地址表
char* cfg_iaddrs; //ID服务器地址表
char* cfg_maddrs; //MySQL地址表
char* cfg_raddrs; //Redis地址表
acl::master_str_tbl cfg_str[] = { //字符串配置表
        {"tnv_group_name",     "group001",          &cfg_gpname},
```



TNV/src/04_storage/02_globals.cpp

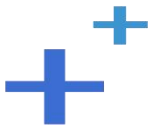
```
    {"tnv_store_paths",    "../data",          &cfg_spaths},
    {"tnv_tracker_addrs", "127.0.0.1:21000", &cfg_taddrs},
    {"tnv_ids_addrs",      "127.0.0.1:22000", &cfg_iaddrs},
    {"mysql_addrs",        "127.0.0.1",        &cfg_maddrs},
    {"redis_addrs",        "127.0.0.1:6379",   &cfg_raddrs},
    {0, 0, 0}};

int cfg_bindport; //绑定端口号
int cfg_interval; //心跳间隔秒数
int cfg_mtimeout; //MySQL读写超时
int cfg_maxconns; //Redis连接池最大连接数
int cfg_ctimeout; //Redis连接超时
int cfg_rtimeout; //Redis读写超时
int cfg_ktimeout; //Redis键超时
acl::master_int_tbl cfg_int[] = { //整型配置表
```

知识讲解

TNV/src/04_storage/02_globals.cpp

```
{"tnv_storage_port",          23000, &cfg_bindport, 0, 0},
{"tnv_heart_beat_interval",      10, &cfg_interval, 0, 0},
{"mysql_rw_timeout",             30, &cfg_mtimeout, 0, 0},
{"redis_max_conn_num",          600, &cfg_maxconns, 0, 0},
{"redis_conn_timeout",           10, &cfg_ctimeout, 0, 0},
{"redis_rw_timeout",             10, &cfg_rtimeout, 0, 0},
{"redis_key_timeout",            60, &cfg_ktimeout, 0, 0},
{0, 0, 0, 0, 0}};
```

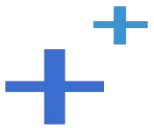
知识讲解



TNV/src/04_storage/02_globals.cpp

知识讲解

```
std::vector<std::string> g_spaths; //存储路径表
std::vector<std::string> g_taddrs; //跟踪服务器地址表
std::vector<std::string> g_iaddrs; //ID服务器地址表
std::vector<std::string> g_maddrs; //MySQL地址表
std::vector<std::string> g_raddrs; //Redis地址表
acl::redis_client_pool*  g_rconns; //Redis连接池
std::string g_hostname; //主机名
char const* g_version = "1.0"; //版本
time_t g_stime; //启动时间
```



TNV/src/04_storage/03_cache.h

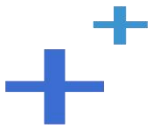
知识讲解

```
//存储服务器
//声明缓存类
//
#pragma once

#include <lib_acl.hpp>
//
//缓存类
//
class cache_c {
public:
        //根据键获取其值
        int get(char const* key, acl::string& value) const;

        //设置指定键的值
        int set(char const* key, char const* value, int timeout = -1) const;

        //删除指定键值对
        int del(char const* key) const;
};
```

TNV/src/04_storage/04_cache.cpp

知识讲解

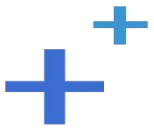
```
//存储服务器
//实现缓存类
//
#include "01_types.h"
#include "01_globals.h"
#include "03_cache.h"

//根据键获取其值
int cache_c::get(char const* key, acl::string& value) const {
        //构造键
        acl::string storage_key;
        storage_key.format("%s:%s", STORAGE_REDIS_PREFIX, key);

        //检查Redis连接池
        if (!g_rconns) {
```



TNV/src/04_storage/04_cache.cpp

知识讲解

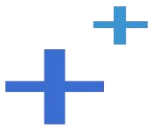
```
            logger_warn("redis connection pool is null, key: %s",
                    storage_key.c_str());
            return ERROR;
    }

    // 从连接池中获取一个Redis连接
    acl::redis_client* rconn = (acl::redis_client*)g_rconns->peek();
    if (!rconn) {
            logger_warn("peek redis connection fail, key: %s",
                    storage_key.c_str());
            return ERROR;
    }

    // 持有此连接的Redis对象即为Redis客户机
    acl::redis redis;
```



TNV/src/04_storage/04_cache.cpp

知识讲解

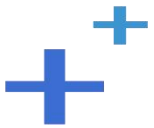
```
redis.set_client(rconn);

//借助Redis客户机根据键获取其值
if (!redis.get(storage_key.c_str(), value)) {
        logger_warn("get cache fail, key: %s", storage_key.c_str());
        g_rconns->put(rconn, false);
        return ERROR;
}

//检查空值
if (value.empty()) {
        logger_warn("value is empty, key: %s", storage_key.c_str());
        g_rconns->put(rconn, false);
        return ERROR;
}
```



TNV/src/04_storage/04_cache.cpp

知识讲解

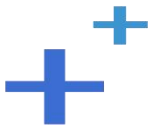
```
        logger("get cache ok, key: %s, value: %s",
                storage_key.c_str(), value.c_str());
        g_rconns->put(rconn, true);

        return OK;
}

//设置指定键的值
int cache_c::set(char const* key, char const* value,
        int timeout /* = -1 */) const {
        //构造键
        acl::string storage_key;
        storage_key.format("%s:%s", STORAGE_REDIS_PREFIX, key);

        //检查Redis连接池
```



TNV/src/04_storage/04_cache.cpp

知识讲解

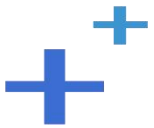
```
if (!g_rconns) {
        logger_warn("redis connection pool is null, key: %s",
                storage_key.c_str());
        return ERROR;
}

// 从连接池中获取一个Redis连接
acl::redis_client* rconn = (acl::redis_client*)g_rconns->peek();
if (!rconn) {
        logger_warn("peek Redis connection fail, key: %s",
                storage_key.c_str());
        return ERROR;
}

// 持有此连接的Redis对象即为Redis客户机
```



TNV/src/04_storage/04_cache.cpp

知识讲解

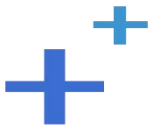
```
acl::redis redis;
redis.set_client(rconn);

// 借助Redis客户机设置指定键的值
if (timeout < 0)
        timeout = cfg_ktimeout;
if (!redis.setex(storage_key.c_str(), value, timeout)) {
        logger_warn("set cache fail, key: %s, value: %s, timeout: %d",
                storage_key.c_str(), value, timeout);
        g_rconns->put(rconn, false);
        return ERROR;
}
logger("set cache ok, key: %s, value: %s, timeout: %d",
        storage_key.c_str(), value, timeout);
g_rconns->put(rconn, true);
```



TNV/src/04_storage/04_cache.cpp

知识讲解

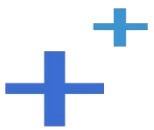
```
        return OK;
}

//删除指定键值对
int cache_c::del(char const* key) const {
        //构造键
        acl::string storage_key;
        storage_key.format("%s:%s", STORAGE_REDIS_PREFIX, key);

        //检查Redis连接池
        if (!g_rconns) {
                logger_warn("redis connection pool is null, key: %s",
                        storage_key.c_str());
                return ERROR;
        }
```



TNV/src/04_storage/04_cache.cpp

知识讲解

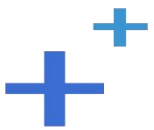
```
// 从连接池中获取一个Redis连接
acl::redis_client* rconn = (acl::redis_client*)g_rconns->peek();
if (!rconn) {
        logger_warn("peek Redis connection fail, key: %s",
                storage_key.c_str());
        return ERROR;
}

// 持有此连接的Redis对象即为Redis客户机
acl::redis redis;
redis.set_client(rconn);
```



TNV/src/04_storage/04_cache.cpp

知识讲解

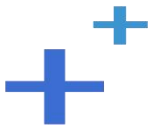
```
        // 借助Redis客户机删除指定键值对
        if (!redis.del_one(storage_key.c_str())) {
                logger_warn("delete cache fail, key: %s", storage_key.c_str());
                g_rconns->put(rconn, false);
                return ERROR;
        }
        logger("delete cache ok, key: %s", storage_key.c_str());
        g_rconns->put(rconn, true);

        return OK;
}
```



TNV/src/04_storage/05_db.h

知识讲解

```
//存储服务器
//声明数据库访问类
//
#pragma once

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



TNV/src/04_storage/05_db.h

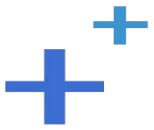
知识讲解

```
~db_c(void);

//连接数据库
int connect(void);

//根据文件ID获取其对应的路径及大小
int get(char const* appid, char const* userid, char const* fileid,
        std::string& filepath, long long* filesize) const;
//设置文件ID和路径及大小的对应关系
int set(char const* appid, char const* userid, char const* fileid,
        char const* filepath, long long filesize) const;
//删除文件ID
```

TNV/src/04_storage/05_db.h

```
        int del(char const* appid, char const*userid,
                char const* fileid) const;

private:
        //根据用户ID获取其对应的表名
        std::string table_of_user(char const* userid) const;

        //计算哈希值
        unsigned int hash(char const* buf, size_t len) const;

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

知识讲解



TNV/src/04_storage/06_db.cpp

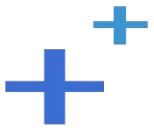
知识讲解

```
//存储服务器
//实现数据库访问类
//
#include "01_types.h"
#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
```

TNV/src/04_storage/06_db.cpp

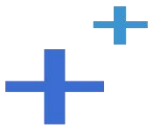
知识讲解

```
db_c::~db_c(void) {
        //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)
```



TNV/src/04_storage/06_db.cpp

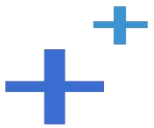

```
                if ((m_mysql = mysql_real_connect(mysql, maddr->c_str(),
                        "root", "123456", "tnv_storagedb", 0, NULL, 0)))
                    return OK;

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

//根据文件ID获取其对应的路径及大小
int db_c::get(char const* appid, char const* userid, char const* fileid,
        std::string& filepath, long long* filesize) const {
        //先尝试从缓存中获取与文件ID对应的路径及大小
        cache_c cache;
        acl::string key;
```



TNV/src/04_storage/06_db.cpp

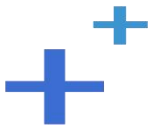
知识讲解

```
key.format("uid:fid:%s:%s", userid, fileid);
acl::string value;
if (cache.get(key, value) == OK) {
        std::vector<acl::string> size_path = value.split2(";");
        if (size_path.size() == 2) {
                filepath = size_path[1].c_str();
                *filesize = atoll(size_path[0].c_str());
                if (!filepath.empty() && *filesize > 0) {
                        logger("from cache, appid: %s, userid: %s, "
                                "fileid: %s, filepath: %s, filesize: %lld",
                                appid, userid, fileid, filepath.c_str(), *filesize);
                        return OK;
                }
        }
}
```



TNV/src/04_storage/06_db.cpp

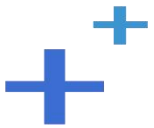

```
//缓存中没有再查询数据库
std::string tablename = table_of_user(userid);
if (tablename.empty()) {
        logger_error("tablename is empty, appid: %s, "
                "userid: %s, fileid: %s", appid, userid, fileid);
        return ERROR;
}
acl::string sql;
sql.format("SELECT file_path, file_size FROM %s WHERE id='%s';",
        tablename.c_str(), fileid);
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/04_storage/06_db.cpp

知识讲解

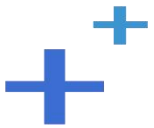
```
//获取查询结果
MYSQL_RES* res = mysql_store_result(m_mysql);
if (!res) {
        logger_error("result is null: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}

//获取结果记录
MYSQL_ROW row = mysql_fetch_row(res);
if (!row) {
        logger_error("result is empty: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}
```



TNV/src/04_storage/06_db.cpp

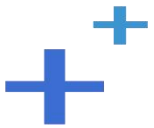
知识讲解

```
        filepath = row[0];
        *filesize = atoll(row[1]);
        logger("from database, appid: %s, userid: %s, "
               "fileid: %s, filepath: %s, filesize: %lld",
               appid, userid, fileid, filepath.c_str(), *filesize);

        //将文件ID和路径及大小的对应关系保存在缓存中
        value.format("%lld;%s", *filesize, filepath.c_str());
        cache.set(key, value.c_str());

        return OK;
}

//设置文件ID和路径及大小的对应关系
int db_c::set(char const* appid, char const* userid, char const* fileid,
```



TNV/src/04_storage/06_db.cpp

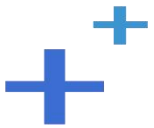

```
char const* filepath, long long filesize) const {
//根据用户ID获取其对应的表名
std::string tablename = table_of_user(userid);
if (tablename.empty()) {
        logger_error("tablename is empty, appid: %s, "
                "userid: %s, fileid: %s", appid, userid, fileid);
        return ERROR;
}

//插入一条记录
acl::string sql;
sql.format("INSERT INTO %s SET id='%s', appid='%s', "
        "userid='%s', status=0, file_path='%s', file_size=%lld;",
        tablename.c_str(), fileid, appid, userid, filepath, filesize);
if (mysql_query(m_mysql, sql.c_str())) {
```



TNV/src/04_storage/06_db.cpp

知识讲解

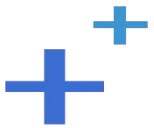
```
            logger_error("insert database fail: %s, sql: %s",
                    mysql_error(m_mysql), sql.c_str());
            return ERROR;
        }

        //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/04_storage/06_db.cpp

知识讲解

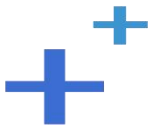
```
//删除文件ID
int db_c::del(char const* appid, char const*userid,
        char const* fileid) const {
        //先从缓存中删除文件ID
        cache_c cache;
        acl::string key;
        key.format("uid:fid:%s:%s", userid, fileid);
        if (cache.del(key) != OK)
                logger_warn("delete cache fail: appid: %s, "
                        "userid: %s, fileid: %s", appid, userid, fileid);

        //再从数据库中删除文件ID
        std::string tablename = table_of_user(userid);
        if (tablename.empty()) {
                logger_error("tablename is empty, appid: %s, "
```



TNV/src/04_storage/06_db.cpp

知识讲解

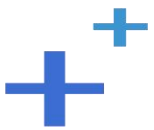
```
                "userid: %s, fileid: %s", appid, userid, fileid);
        return ERROR;
}
acl::string sql;
sql.format("DELETE FROM %s WHERE id='%s';",
        tablename.c_str(), fileid);
if (mysql_query(m_mysql, sql.c_str())) {
        logger_error("delete database fail: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
        return ERROR;
}

//检查删除结果
MYSQL_RES* res = mysql_store_result(m_mysql);
if (!res && mysql_field_count(m_mysql)) {
```



TNV/src/04_storage/06_db.cpp

知识讲解

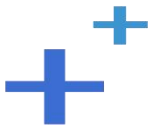
```
                logger_error("delete database fail: %s, sql: %s",
                        mysql_error(m_mysql), sql.c_str());
                return ERROR;
        }

        return OK;
}

//根据用户ID获取其对应的表名
std::string db_c::table_of_user(char const* userid) const {
        char tablename[10];

        sprintf(tablename, "t_file_%02d",
                (hash(userid, strlen(userid)) & 0x7FFFFFFF) % 3 + 1);
```



TNV/src/04_storage/06_db.cpp

知识讲解

```
        return tablename;
}

//计算哈希值
unsigned int db_c::hash(char const* buf, size_t len) const {
        unsigned int h = 0;

        for (size_t i = 0; i < len; ++i)
                h ^= i&1 ? ~(h<<11^buf[i]^h>>5) : h<<7^buf[i]^h>>3;

        return h;
}
```



复习课见