Tedu.cn
达内教育

《分布式流媒体》实训项目

C/C++教学体系

TNV DAY14

直播课

目录

服务器类(server_c)

属性

- 成员变量
  - 跟踪客户机线程集：m_trackers

知识讲解

server_c
+ thread_on_accept (conn: int *) bool + thread_on_close (conn: int *) void + thread_on_read (conn: int *) bool + thread_on_timeout (conn: int *) bool # proc_exit_timer (nclients: size_t , nthreads: size_t) bool # proc_on_exit () void # proc_on_init () void

进程级回调方法

知识讲解

- 进程启动时被调用：proc_on_init
- 检查隶属组名和绑定端口号
- 检查并拆分存储路径表、跟踪服务器地址表和ID服务器地址表
- 检查并拆分MySQL地址表
- 检查并拆分Redis地址表
- 遍历Redis地址表，尝试创建连接池
  - 若Redis连接池创建成功
    - 则设置Redis连接超时和读写超时
- 获取主机名
- 记录启动时间
- 创建并启动连接每台跟踪服务器的客户机线程
- 打印配置信息

进程级回调方法

知识讲解

- 进程意图退出时被调用：proc_exit_timer
- 返回true，进程立即退出，否则若配置项ioctl_quick_abort非0，进程立即退出，否则待所有客户机连接都关闭后，进程再退出
- 终止所有跟踪客户机线程
- 检查客户机数和客户线程数
  - 若其中至少有一个为零
    - 则立即退出
  - 否则
    - 待所有客户机连接都关闭后再退出，除非配置项ioctl_quick_abort非零

进程级回调方法

知识讲解

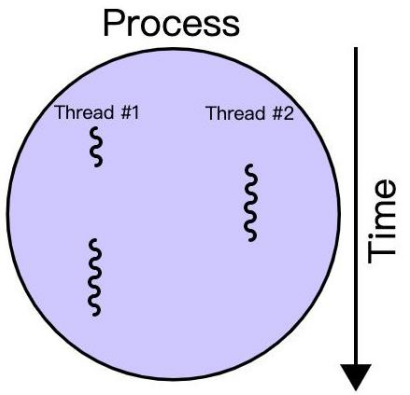
- 进程即将退出时被调用：proc_on_exit
  - 回收并销毁所有跟踪客户机线程
  - 销毁Redis连接池

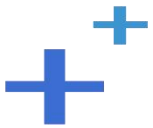
```
// 进程退出前被调用
void server_c::proc_on_exit(void) {
    for (std::list<tracker_c*>::iterator tracker = m_trackers.begin();
        tracker != m_trackers.end(); ++tracker) {
        // 回收跟踪客户机线程
        if (!(*tracker)->wait(NULL))
            logger_error("wait thread #%lu fail", (*tracker)->thread_id());
        // 销毁跟踪客户机线程
        delete *tracker;
    }

    m_trackers.clear();

    // 销毁Redis连接池
    if (g_rconns) {
        delete g_rconns;
        g_rconns = NULL;
    }
}
```

线程级回调方法

知识讲解

- 线程获得连接时被调用：thread_on_accept
  - 返回true，连接将被用于后续通信，否则函数返回后即关闭连接
  - 打印日志
- 线程连接可读时被调用：thread_on_read
  - 返回true，保持长连接，否则函数返回后即关闭连接
  - 接收包头
  - 业务处理

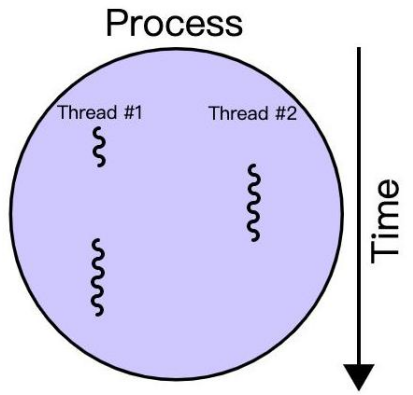




线程级回调方法

知识讲解

- 线程读写超时时被调用：thread_on_timeout
  - 返回true，继续等待下一次读写，否则函数返回后即关闭连接
  - 打印日志
  - 返回true以保持连接
- 线程连接关闭时被调用：thread_on_close
  - 打印日志

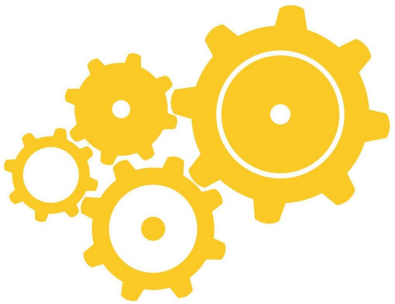


主函数(main)

主函数(main)

知识讲解

- 初始化ACL库
  - acl::acl_cpp_init();
  - acl::log::stdout_open(true);
- 创建并运行服务器
  - server_c& server = acl::singleton2<server_c>::get_instance();
  - server.set_cfg_str(cfg_str);
  - server.set_cfg_int(cfg_int);
  - server.run_alone("127.0.0.1:23000", "../etc/storage.cfg");



构建脚本(Makefile)

构建脚本(Makefile)

知识讲解

```
PROJ = ../../bin/storage
OBJS = $(patsubst %.cpp, %.o, $(wildcard ../01_common/*.cpp *.cpp))
CC = g++
LINK = g++
RM = rm -rf
CFLAGS = -c -Wall -I/usr/include/acl-lib/acl_cpp `mysql_config --
cflags` -I../01_common
LIBS = -pthread -lacl_all `mysql_config --libs`
```

构建脚本(Makefile)

知识讲解

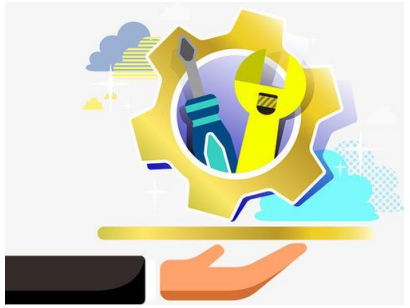
```
all: $(PROJ)
$(PROJ): $(OBJS)
        $(LINK) $^ $(LIBS) -o $@
.cpp.o:
        $(CC) $(CFLAGS) $^ -o $@
clean:
        $(RM) $(PROJ) $(OBJS)
```

配置文件(storage.cfg)

配置文件(storage.cfg)

知识讲解

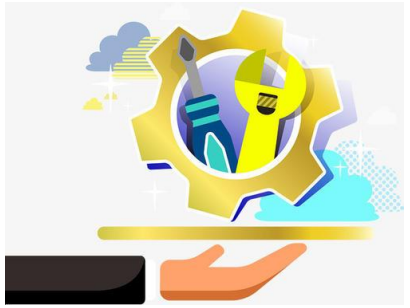
- 隶属组名：tnv_group_name = group001
- 存储路径表：tnv_store_paths = ../data
- 跟踪服务器地址表：tnv_tracker_addrs = 127.0.0.1:21000
- ID服务器地址表：tnv_ids_addrs = 127.0.0.1:22000
- MySQL地址表：mysql_addrs = 127.0.0.1
- Redis地址表：redis_addrs = 127.0.0.1:6379
- 绑定端口号：tnv_storage_port = 23000



配置文件(storage.cfg)

知识讲解

- 心跳间隔秒数：tnv_heart_beat_interval = 10
- MySQL读写超时：mysql_rw_timeout = 30
- Redis连接池最大连接数：redis_max_conn_num = 600
- Redis连接超时：redis_conn_timeout = 10
- Redis读写超时：redis_rw_timeout = 10
- Redis键超时：redis_key_timeout = 60

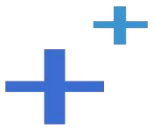


附录：程序清单

TNV/src/04_storage/15_server.h

知识讲解

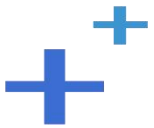
```
//存储服务器
//声明服务器类
//
#pragma once

#include <list>
#include <lib_acl.hpp>
#include "13_tracker.h"
//
//服务器类
//
class server_c: public acl::master_threads {
protected:
        //进程切换用户后被调用
        void proc_on_init(void);
```



TNV/src/04_storage/15_server.h

知识讲解

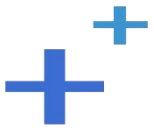
```
// 子进程意图退出时被调用
// 返回true，子进程立即退出，否则
// 若配置项ioctl_quick_abort非0，子进程立即退出，否则
// 待所有客户机连接都关闭后，子进程再退出
bool proc_exit_timer(size_t nclients, size_t nthreads);
// 进程退出前被调用
void proc_on_exit(void);

// 线程获得连接时被调用
// 返回true，连接将被用于后续通信，否则
// 函数返回后即关闭连接
bool thread_on_accept(acl::socket_stream* conn);
// 与线程绑定的连接可读时被调用
// 返回true，保持长连接，否则
// 函数返回后即关闭连接
```



TNV/src/04_storage/15_server.h

知识讲解

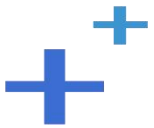
```
        bool thread_on_read(acl::socket_stream* conn);
        //线程读写连接超时时被调用
        //返回true，继续等待下一次读写，否则
        //函数返回后即关闭连接
        bool thread_on_timeout(acl::socket_stream* conn);
        //与线程绑定的连接关闭时被调用
        void thread_on_close(acl::socket_stream* conn);

private:
        std::list<tracker_c*> m_trackers; //跟踪客户机线程集
};
```



TNV/src/04_storage/16_server.cpp

知识讲解

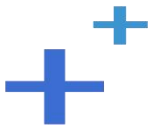
```
//存储服务器
//实现服务器类
//
#include <unistd.h>
#include "02_proto.h"
#include "03_util.h"
#include "01_globals.h"
#include "11_service.h"
#include "15_server.h"

//进程切换用户后被调用
void server_c::proc_on_init(void) {
        //隶属组名
        if (strlen(cfg_gpname) > STORAGE_GROUPNAME_MAX)
                logger_fatal("groupname too big %lu > %d",
```



TNV/src/04_storage/16_server.cpp

知识讲解

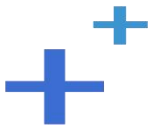
```
                strlen(cfg_gpname), STORAGE_GROUPNAME_MAX);

//绑定端口号
if (cfg_bindport <= 0)
        logger_fatal("invalid bind port %d <= 0", cfg_bindport);

//存储路径表
if (!cfg_spaths || !strlen(cfg_spaths))
        logger_fatal("storage paths is null");
split(cfg_spaths, g_spaths);
if (g_spaths.empty())
        logger_fatal("storage paths is null");

//跟踪服务器地址表
if (!cfg_taddrs || !strlen(cfg_taddrs))
```



TNV/src/04_storage/16_server.cpp

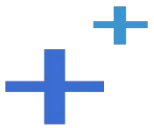
```
        logger_fatal("tracker addresses is null");
split(cfg_taddrs, g_taddrs);
if (g_taddrs.empty())
        logger_fatal("tracker addresses is null");

// ID服务器地址表
if (!cfg_iaddrs || !strlen(cfg_iaddrs))
        logger_fatal("id addresses is null");
split(cfg_iaddrs, g_iaddrs);
if (g_iaddrs.empty())
        logger_fatal("id addresses is null");

// MySQL地址表
if (!cfg_maddrs || !strlen(cfg_maddrs))
        logger_fatal("mysql addresses is null");
```

知识讲解



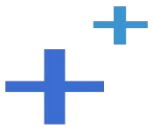
TNV/src/04_storage/16_server.cpp


```
split(cfg_maddrs, g_maddrs);
if (g_maddrs.empty())
        logger_fatal("mysql addresses is null");

// Redis地址表
if (!cfg_raddrs || !strlen(cfg_raddrs))
        logger_error("redis addresses is null");
else {
        split(cfg_raddrs, g_raddrs);
        if (g_raddrs.empty())
                logger_error("redis addresses is empty");
        else {
                // 遍历Redis地址表，尝试创建连接池
                for (std::vector<std::string>::const_iterator raddr =
                        g_raddrs.begin(); raddr != g_raddrs.end(); ++raddr)
```



TNV/src/04_storage/16_server.cpp

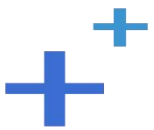

```
                if ((g_rconns = new acl::redis_client_pool(
                        raddr->c_str(), cfg_maxconns))) {
                        // 设置Redis连接超时和读写超时
                        g_rconns->set_timeout(cfg_ctimeout, cfg_rtimeout);
                        break;
                }
        if (!g_rconns)
                logger_error("create redis connection pool fail, cfg_raddrs: %s",
                        cfg_raddrs);
    }
}

// 主机名
char hostname[256+1] = {};
if (gethostname(hostname, sizeof(hostname) - 1))
```



TNV/src/04_storage/16_server.cpp

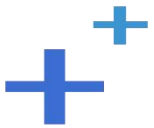
知识讲解

```
        logger_error("call gethostname fail: %s", strerror(errno));
g_hostname = hostname;

//启动时间
g_stime = time(NULL);

//创建并启动连接每台跟踪服务器的客户机线程
for (std::vector<std::string>::const_iterator taddr = g_taddrs.begin();
        taddr != g_taddrs.end(); ++taddr) {
        tracker_c* tracker = new tracker_c(taddr->c_str());
        tracker->set_detachable(false);
        tracker->start();
        m_trackers.push_back(tracker);
}
```



TNV/src/04_storage/16_server.cpp

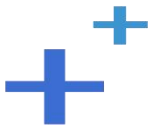

```
        //打印配置信息
        logger("cfg_gpname: %s, cfg_spaths: %s, cfg_taddrs: %s, "
               "cfg_iaddrs: %s, cfg_maddrs: %s, cfg_raddrs: %s, "
               "cfg_bindport: %d, cfg_interval: %d, cfg_mtimeout: %d, "
               "cfg_maxconns: %d, cfg_ctimeout: %d, cfg_rtimeout: %d, "
               "cfg_ktimeout: %d",
               cfg_gpname, cfg_spaths, cfg_taddrs,
               cfg_iaddrs, cfg_maddrs, cfg_raddrs,
               cfg_bindport, cfg_interval, cfg_mtimeout,
               cfg_maxconns, cfg_ctimeout, cfg_rtimeout,
               cfg_ktimeout);
}

//子进程意图退出时被调用
//返回true，子进程立即退出，否则
```



TNV/src/04_storage/16_server.cpp

知识讲解

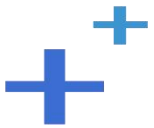
```
//若配置项ioctl_quick_abort非0，子进程立即退出，否则
//待所有客户机连接都关闭后，子进程再退出
bool server_c::proc_exit_timer(size_t nclients, size_t nthreads) {
        for (std::list<tracker_c*>::iterator tracker = m_trackers.begin();
                tracker != m_trackers.end(); ++tracker)
                //终止跟踪客户机线程
                (*tracker)->stop();

        if (!nclients || !nthreads) {
                logger("nclients: %lu, nthreads: %lu", nclients, nthreads);
                return true;
        }

        return false;
}
```



TNV/src/04_storage/16_server.cpp

知识讲解

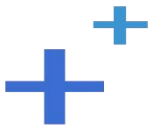
```
//进程退出前被调用
void server_c::proc_on_exit(void) {
        for (std::list<tracker_c*>::iterator tracker = m_trackers.begin();
                tracker != m_trackers.end(); ++tracker) {
                //回收跟踪客户机线程
                if (!(*tracker)->wait(NULL))
                        logger_error("wait thread #%lu fail", (*tracker)->thread_id());
                //销毁跟踪客户机线程
                delete *tracker;
        }

        m_trackers.clear();

        //销毁Redis连接池
        if (g_rconns) {
```



TNV/src/04_storage/16_server.cpp

知识讲解

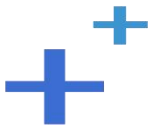
```
            delete g_rconns;
            g_rconns = NULL;
        }
    }

    // 线程获得连接时被调用
    // 返回true，连接将被用于后续通信，否则
    // 函数返回后即关闭连接
    bool server_c::thread_on_accept(acl::socket_stream* conn) {
        logger("connect, from: %s", conn->get_peer());
        return true;
    }

    // 与线程绑定的连接可读时被调用
    // 返回true，保持长连接，否则
```



TNV/src/04_storage/16_server.cpp

知识讲解

```
// 函数返回后即关闭连接
bool server_c::thread_on_read(acl::socket_stream* conn) {
        // 接收包头
        char head[HEADLEN];
        if (conn->read(head, HEADLEN) < 0) {
                if (conn->eof())
                        logger("connection has been closed, from: %s",
                                conn->get_peer());
                else
                        logger_error("read fail: %s, from: %s",
                                acl::last_serror(), conn->get_peer());
                return false;
        }

        // 业务处理
```



TNV/src/04_storage/16_server.cpp

知识讲解

```
        service_c service;
        return service.business(conn, head);
}

//线程读写连接超时时被调用
//返回true，继续等待下一次读写，否则
//函数返回后即关闭连接
bool server_c::thread_on_timeout(acl::socket_stream* conn) {
        logger("read timeout, from: %s", conn->get_peer());
        return true;
}

//与线程绑定的连接关闭时被调用
void server_c::thread_on_close(acl::socket_stream* conn) {
        logger("client disconnect, from: %s", conn->get_peer());
}
```

TNV/src/04_storage/17_main.cpp

知识讲解

```
//存储服务器
//定义主函数
//
#include "01_globals.h"
#include "15_server.h"

int main(void) {
        //初始化ACL库
        acl::acl_cpp_init();
        acl::log::stdout_open(true);

        //创建并运行服务器
        server_c& server = acl::singleton2<server_c>::get_instance();
        server.set_cfg_str(cfg_str);
        server.set_cfg_int(cfg_int);
        server.run_alone("127.0.0.1:23000", "../etc/storage.cfg");

        return 0;
}
```

TNV/src/04_storage/Makefile

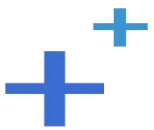
知识讲解

```
PROJ   = ../../bin/storage
OBJS   = $(patsubst %.cpp, %.o, $(wildcard ../01_common/*.cpp *.cpp))
CC     = g++
LINK   = g++
RM     = rm -rf
CFLAGS = -c -Wall -I/usr/include/acl-lib/acl_cpp `mysql_config --cflags` -I../01_common
LIBS   = -pthread -lacl_all `mysql_config --libs`

all: $(PROJ)

$(PROJ): $(OBJS)
        $(LINK) $^ $(LIBS) -o $@
.cpp.o:
        $(CC) $(CFLAGS) $^ -o $@
clean:
        $(RM) $(PROJ) $(OBJS)
```

TNV/etc/storage.cfg

知识讲解

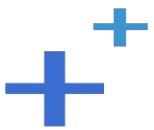
```
service storage {
        # 隶属组名
        tnv_group_name = group001
        # 存储路径表
        tnv_store_paths = ../data
        # 跟踪服务器地址表
        tnv_tracker_addrs = 127.0.0.1:21000
        # ID服务器地址表
        tnv_ids_addrs = 127.0.0.1:22000
        # MySQL地址表
        mysql_addrs = 127.0.0.1
        # Redis地址表
        redis_addrs = 127.0.0.1:6379
        # 绑定端口号
```



TNV/etc/storage.cfg

知识讲解

```
    tnv_storage_port = 23000
    # 心跳间隔秒数
    tnv_heart_beat_interval = 10
    # MySQL读写超时
    mysql_rw_timeout = 30
    # Redis连接池最大连接数
    redis_max_conn_num = 600
    # Redis连接超时
    redis_conn_timeout = 10
    # Redis读写超时
    redis_rw_timeout = 10
    # Redis键超时
    redis_key_timeout = 60
}
```



复习课见