Tedu.cn
达内教育

《分布式流媒体》实训项目

C/C++教学体系

TNV DAY07

直播课

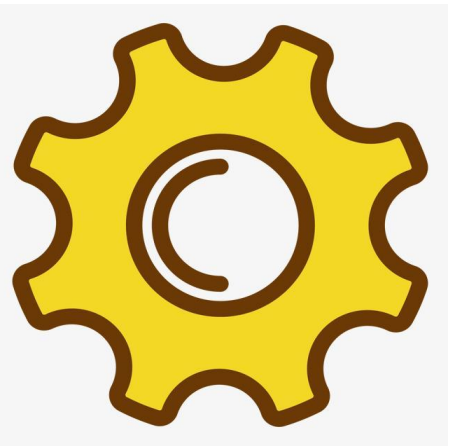
目录

状态检查线程类(status_c)

方法

- 检查存储服务器状态：check
  - 获取当前时间
  - 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
      - 若该存储服务器心跳停止太久
        - 则将其状态标记为离线
  - 互斥锁解锁
  - 返回成功

知识讲解

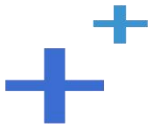


方法

- 检查存储服务器状态：check

```
time_t now = time(NULL); // 现在

// 遍历组表中的每一个组
for (std::map<std::string, std::list<storage_info_t> >::iterator
        group = g_groups.begin(); group != g_groups.end(); ++group)
        // 遍历该组中的每一台存储服务器
        for (std::list<storage_info_t>::iterator si =
                group->second.begin(); si != group->second.end(); ++si)
                // 若该存储服务器心跳停止太久
                if (now - si->si_btime >= cfg_interval)
                        // 则将其状态标记为离线
                        si->si_status = STORAGE_STATUS_OFFLINE;
```

知识讲解



服务器类(server_c)

属性

- 成员变量
  - 存储服务器状态检查线程：m_status

知识讲解

server_c
- m_status: status_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

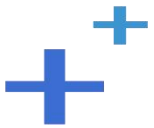
- 进程启动时被调用：proc_on_init
  - 检查并拆分应用ID表
  - 检查并拆分MySQL地址表
  - 检查并拆分Redis地址表
  - 遍历Redis地址表，尝试创建连接池
    - 若Redis连接池创建成功
      - 则设置Redis连接超时和读写超时
  - 获取主机名
  - 创建并启动存储服务器状态检查线程
  - 打印配置信息

进程级回调方法

- 进程启动时被调用：proc_on_init

```
split(cfg_raddrs, g_raddrs);

// 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)
        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);
```

知识讲解



进程级回调方法

- 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
  - 终止存储服务器状态检查线程
  - 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

知识讲解

进程级回调方法

- 进程意图退出时被调用：proc_exit_timer

```
// 终止存储服务器状态检查线程
m_status->stop();

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

return false;
```

知识讲解

进程级回调方法

知识讲解

- 进程即将退出时被调用：proc_on_exit
  - 回收存储服务器状态检测线程
  - 销毁存储服务器状态检查线程
  - 销毁Redis连接池

```
// 进程退出前被调用
void server_c::proc_on_exit(void) {
    // 回收存储服务器状态检测线程
    if (!m_status->wait(NULL))
        logger_error("wait thread #%lu fail", m_status->thread_id());

    // 销毁存储服务器状态检查线程
    if (m_status) {
        delete m_status;
        m_status = NULL;
    }

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

进程级回调方法

- 进程即将退出时被调用：proc_on_exit

```
// 回收存储服务器状态检测线程
if (!m_status->wait(NULL))
        logger_error("wait thread #%lu fail", m_status->thread_id());
// 销毁存储服务器状态检查线程
if (m_status) {
        delete m_status;
        m_status = NULL;
}
// 销毁Redis连接池
if (g_rconns) {
        delete g_rconns;
        g_rconns = NULL;
}
```

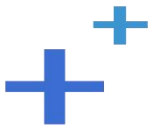
知识讲解

附录：程序清单

TNV/src/02_tracker/10_status.cpp

知识讲解

```
//检查存储服务器状态
int status_c::check(void) const {
        time_t now = time(NULL); //现在

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

        //遍历组表中的每一个组
        for (std::map<std::string, std::list<storage_info_t> >::iterator
                group = g_groups.begin(); group != g_groups.end(); ++group)
                //遍历该组中的每一台存储服务器
```



TNV/src/02_tracker/10_status.cpp

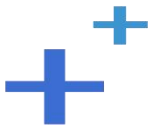
知识讲解

```
            for (std::list<storage_info_t>::iterator si =
                    group->second.begin(); si != group->second.end(); ++si)
                    //若该存储服务器心跳停止太久
                    if (now - si->si_btime >= cfg_interval)
                            //则将其状态标记为离线
                            si->si_status = STORAGE_STATUS_OFFLINE;

    //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;
}
```

TNV/src/02_tracker/11_server.h

知识讲解

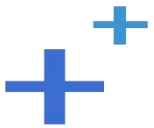
```
//跟踪服务器
//声明服务器类
//
#pragma once

#include <lib_acl.hpp>
#include "09_status.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);
        //子进程意图退出时被调用
```



TNV/src/02_tracker/11_server.h

知识讲解

```
// 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
bool thread_on_read(acl::socket_stream* conn);
```



TNV/src/02_tracker/11_server.h

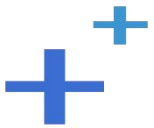
```
        //线程读写连接超时时被调用
        //返回true，继续等待下一次读写，否则
        //函数返回后即关闭连接
        bool thread_on_timeout(acl::socket_stream* conn);
        //与线程绑定的连接关闭时被调用
        void thread_on_close(acl::socket_stream* conn);

private:
        status_c* m_status; //存储服务器状态检查线程
};
```

知识讲解

TNV/src/02_tracker/12_server.cpp

知识讲解

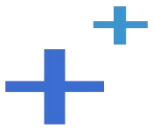
```
//跟踪服务器
//实现服务器类
//
#include <unistd.h>
#include "02_proto.h"
#include "03_util.h"
#include "01_globals.h"
#include "07_service.h"
#include "11_server.h"

//进程切换用户后被调用
void server_c::proc_on_init(void) {
        //应用ID表
        if (!cfg_appids || !strlen(cfg_appids))
                logger_fatal("application ids is null");
```



TNV/src/02_tracker/12_server.cpp

知识讲解

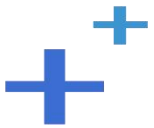
```
split(cfg_appids, g_appids);
if (g_appids.empty())
        logger_fatal("application ids is empty");

// MySQL地址表
if (!cfg_maddrs || !strlen(cfg_maddrs))
        logger_fatal("mysql addresses is null");
split(cfg_maddrs, g_maddrs);
if (g_maddrs.empty())
        logger_fatal("mysql addresses is empty");

// Redis地址表
if (!cfg_raddrs || !strlen(cfg_raddrs))
        logger_error("redis addresses is null");
else {
```



TNV/src/02_tracker/12_server.cpp

知识讲解

```
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)
                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",
```



TNV/src/02_tracker/12_server.cpp

知识讲解

```
                                        cfg_raddrs);
        }
}

//主机名
char hostname[256+1] = {};
if (gethostname(hostname, sizeof(hostname) - 1))
        logger_error("call gethostname fail: %s", strerror(errno));
g_hostname = hostname;

//创建并启动存储服务器状态检查线程
if ((m_status = new status_c)) {
        m_status->set_detachable(false);
        m_status->start();
}
```

TNV/src/02_tracker/12_server.cpp

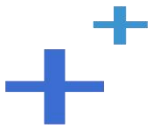
知识讲解

```
        //打印配置信息
        logger("cfg_appids: %s, cfg_maddrs: %s, cfg_raddrs: %s, "
               "cfg_interval: %d, cfg_mtimeout: %d, cfg_maxconns: %d, "
               "cfg_ctimeout: %d, cfg_rtimeout: %d, cfg_ktimeout: %d",
               cfg_appids, cfg_maddrs, cfg_raddrs,
               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
//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) {
        //终止存储服务器状态检查线程
```

TNV/src/02_tracker/12_server.cpp

知识讲解

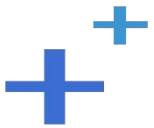
```
        m_status->stop();

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

        return false;
}

//进程退出前被调用
void server_c::proc_on_exit(void) {
        //回收存储服务器状态检测线程
        if (!m_status->wait(NULL))
                logger_error("wait thread #%lu fail", m_status->thread_id());
```



TNV/src/02_tracker/12_server.cpp

知识讲解

```
        //销毁存储服务器状态检查线程
        if (m_status) {
                delete m_status;
                m_status = NULL;
        }

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



复习课见