3 ID服务器

~/TNV/src/03_id/01_globals.h

```
// ID服务器
// 声明全局变量
//
#pragma once

#include <vector>
#include <lib_acl.hpp>
#include "01_types.h"
//
// 配置信息
//
extern char* cfg_maddrs; // MySQL地址表
extern acl::master_str_tbl cfg_str[]; // 字符串配置表

extern int cfg_mtimeout;  // MySQL读写超时
extern int cfg_maxoffset; // 最大偏移
extern acl::master_int_tbl cfg_int[]; // 整型配置表

extern std::vector<std::string> g_maddrs; // MySQL地址表
extern std::string g_hostname; // 主机名
extern std::vector<id_pair_t> g_ids; // ID表
extern pthread_mutex_t g_mutex; // 互斥锁
```

~/TNV/src/03_id/02_globals.cpp

```
// ID服务器
// 定义全局变量
//
#include "01_globals.h"
//
// 配置信息
//
char* cfg_maddrs; // MySQL地址表
acl::master_str_tbl cfg_str[] = { // 字符串配置表
    {"mysql_addrs", "127.0.0.1", &cfg_maddrs},
    {0, 0, 0}};

int cfg_mtimeout;  // MySQL读写超时
int cfg_maxoffset; // 最大偏移
acl::master_int_tbl cfg_int[] = { // 整型配置表
    {"mysql_rw_timeout",  30, &cfg_mtimeout,  0, 0},
    {"idinc_max_step",   100, &cfg_maxoffset, 0, 0},
    {0, 0, 0, 0, 0}};

std::vector<std::string> g_maddrs; // MySQL地址表
std::string g_hostname; // 主机名
std::vector<id_pair_t> g_ids; // ID表
pthread_mutex_t g_mutex = PTHREAD_MUTEX_INITIALIZER; // 互斥锁
```

~/TNV/src/03_id/03_db.h

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

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

    // 连接数据库
    int connect(void);

    // 获取ID当前值，同时产生下一个值
    int get(char const* key, int inc, long* value) const;

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

~/TNV/src/03_id/04_db.cpp

```
// ID服务器
// 实现数据库访问类
//
#include "01_globals.h"
#include "03_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) {
    // 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;
```

```
25
26      // 遍历MySQL地址表，尝试连接数据库
27      for (std::vector<std::string>::const_iterator maddr =
28          g_maddrs.begin(); maddr != g_maddrs.end(); ++maddr)
29          if ((m_mysql = mysql_real_connect(mysql, maddr->c_str(),
30              "root", "123456", "tnv_idsdb", 0, NULL, 0)))
31              return OK;
32
33      logger_error("connect database fail: %s",
34          mysql_error(m_mysql = mysql));
35      return ERROR;
36  }
37
38  // 获取ID当前值，同时产生下一个值
39  int db_c::get(char const* key, int inc, long* value) const {
40      // 关闭自动提交
41      mysql_autocommit(m_mysql, 0);
42
43      // 查询数据库
44      acl::string sql;
45      sql.format("SELECT id_value FROM t_id_gen WHERE id='%s';", key);
46      if (mysql_query(m_mysql, sql.c_str())) {
47          logger_error("query database fail: %s, sql: %s",
48              mysql_error(m_mysql), sql.c_str());
49          mysql_autocommit(m_mysql, 1);
50          return ERROR;
51      }
52
53      // 获取查询结果
54      MYSQL_RES* res = mysql_store_result(m_mysql);
55      if (!res) {
56          logger_error("result is null: %s, sql: %s",
57              mysql_error(m_mysql), sql.c_str());
58          mysql_autocommit(m_mysql, 1);
59          return ERROR;
60      }
61
62      // 获取结果记录
63      MYSQL_ROW row = mysql_fetch_row(res);
64      if (row) { // 有记录
65          // 更新旧记录
66          sql.format("UPDATE t_id_gen SET id_value="
67              "id_value+%d WHERE id='%s';", inc, key);
68          if (mysql_query(m_mysql, sql.c_str())) {
69              logger_error("update database fail: %s, sql: %s",
70                  mysql_error(m_mysql), sql.c_str());
71              mysql_autocommit(m_mysql, 1);
72              return ERROR;
73          }
74          // 提交数据库
75          mysql_commit(m_mysql);
76          // 库中当前值
77          *value = atol(row[0]);
78      }
79      else { // 无记录
80          // 插入新记录
```

```
        sql.format("INSERT INTO t_id_gen SET id='%s', id_value='%d';",
            key, inc);
        if (mysql_query(m_mysql, sql.c_str())) {
            logger_error("insert database fail: %s, sql: %s",
                mysql_error(m_mysql), sql.c_str());
            mysql_autocommit(m_mysql, 1);
            return ERROR;
        }
        // 提交数据库
        mysql_commit(m_mysql);
        // 缺省当前值
        *value = 0;
    }

    // 打开自动提交
    mysql_autocommit(m_mysql, 1);

    return OK;
}
```

~/TNV/src/03_id/05_service.h

```
// ID服务器
// 声明业务服务类
//
#pragma once

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

private:
    // 处理来自存储服务器的获取ID请求
    bool get(acl::socket_stream* conn, long long bodylen) const;

    // 根据ID的键获取其值
    long get(char const* key) const;
    // 从数据库中获取ID值
    long fromdb(char const* key) const;

    // 应答ID
    bool id(acl::socket_stream* conn, long value) const;
    // 应答错误
    bool error(acl::socket_stream* conn, short errnumb,
        char const* format, ...) const;
};
```

~/TNV/src/03_id/06_service.cpp

```cpp
// ID服务器
// 实现业务服务类
//
#include "02_proto.h"
#include "03_util.h"
#include "01_globals.h"
#include "03_db.h"
#include "05_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|
    // 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) {
        case CMD_ID_GET:
            // 处理来自存储服务器的获取ID请求
            result = get(conn, bodylen);
            break;

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

    return result;
}

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

// 处理来自存储服务器的获取ID请求
bool service_c::get(acl::socket_stream* conn, long long bodylen) const {
    // |包体长度|命令|状态|ID键|
    // |    8   |  1 |  1 |64+1|
    // 检查包体长度
    long long expected = ID_KEY_MAX + 1; // 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];
    if (conn->read(body, bodylen) < 0) {
        logger_error("read fail: %s, bodylen: %lld, from: %s",
            acl::last_serror(), bodylen, conn->get_peer());
        return false;
    }

    // 根据ID的键获取其值
    long value = get(body);
    if (value < 0) {
        error(conn, -1, "get id fail, key: %s", body);
        return false;
    }

    logger("get id ok, key: %s, value: %ld", body, value);

    return id(conn, value);
}

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

// 根据ID的键获取其值
long service_c::get(char const* key) const {
    // 互斥锁加锁
    if ((errno = pthread_mutex_lock(&g_mutex))) {
        logger_error("call pthread_mutex_lock fail: %s",
            strerror(errno));
        return -1;
    }

    long value = -1;

    // 在ID表中查找ID
    std::vector<id_pair_t>::iterator id;
    for (id = g_ids.begin(); id != g_ids.end(); ++id)
        if (!strcmp(id->id_key, key))
            break;
    if (id != g_ids.end()) { // 找到该ID
        if (id->id_offset < cfg_maxoffset) { // 该ID的偏移未及上限
            value = id->id_value + id->id_offset;
            ++id->id_offset;
        }
        else if ((value = fromdb(key)) >= 0) { // 从数据库中获取ID值
            // 更新ID表中的ID
            id->id_value = value;
            id->id_offset = 1;
        }
    }
    else if ((value = fromdb(key)) >= 0) { // 从数据库中获取ID值
        // 在ID表中添加ID
        id_pair_t id;
        strcpy(id.id_key, key);
        id.id_value = value;
        id.id_offset = 1;
        g_ids.push_back(id);
```

```
    }

    // 互斥锁解锁
    if ((errno = pthread_mutex_unlock(&g_mutex))) {
        logger_error("call pthread_mutex_unlock fail: %s",
            strerror(errno));
        return -1;
    }

    return value;
}

// 从数据库中获取ID值
long service_c::fromdb(char const* key) const {
    db_c db; // 数据库访问对象

    // 连接数据库
    if (db.connect() != OK)
        return -1;

    long value = -1;

    // 获取ID当前值，同时产生下一个值
    if (db.get(key, cfg_maxoffset, &value) != OK)
        return -1;

    return value;
}

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

// 应答ID
bool service_c::id(acl::socket_stream* conn, long value) const {
    // |包体长度|命令|状态|ID值|
    // |    8   |  1 |  1 |  8 |
    // 构造响应
    long long bodylen = BODYLEN_SIZE;
    long long resplen = HEADLEN + bodylen;
    char resp[resplen] = {};
    llton(bodylen, resp);
    resp[BODYLEN_SIZE] = CMD_ID_REPLY;
    resp[BODYLEN_SIZE+COMMAND_SIZE] = 0;
    lton(value, resp + HEADLEN);

    // 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;
}

// 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);
    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_ID_REPLY;
    resp[BODYLEN_SIZE+COMMAND_SIZE] = STATUS_ERROR;
    ston(errnumb, resp + HEADLEN);
    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/03_id/07_server.h

```
// ID服务器
// 声明服务器类
//
#pragma once

#include <lib_acl.hpp>
//
// 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
    // 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);

    // 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);
    // 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);
};
```

~/TNV/src/03_id/08_server.cpp

```
// ID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 "05_service.h"
#include "07_server.h"

// 进程切换用户后被调用
void server_c::proc_on_init(void) {
    // MySQL地址表
    if (!cfg_maddrs || !*cfg_maddrs)
        logger_fatal("mysql addresses is null");
    split(cfg_maddrs, g_maddrs);
    if (g_maddrs.empty())
        logger_fatal("mysql addresses is empty");

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

    // 最大偏移不能太小
    if (cfg_maxoffset < 10)
        logger_fatal("invalid maximum offset: %d < 10", cfg_maxoffset);

    // 打印配置信息
    logger("cfg_maddrs: %s, cfg_mtimeout: %d, cfg_maxoffset: %d",
        cfg_maddrs, cfg_mtimeout, cfg_maxoffset);
}
```

```
34
35  // 子进程意图退出时被调用
36  // 返回true，子进程立即退出，否则
37  // 若配置项ioctl_quick_abort非0，子进程立即退出，否则
38  // 待所有客户机连接都关闭后，子进程再退出
39  bool server_c::proc_exit_timer(size_t nclients, size_t nthreads) {
40      if (!nclients || !nthreads) {
41          logger("nclients: %lu, nthreads: %lu", nclients, nthreads);
42          return true;
43      }
44
45      return false;
46  }
47
48  // 线程获得连接时被调用
49  // 返回true，连接将被用于后续通信，否则
50  // 函数返回后即关闭连接
51  bool server_c::thread_on_accept(acl::socket_stream* conn) {
52      logger("connect, from: %s", conn->get_peer());
53      return true;
54  }
55
56  // 与线程绑定的连接可读时被调用
57  // 返回true，保持长连接，否则
58  // 函数返回后即关闭连接
59  bool server_c::thread_on_read(acl::socket_stream* conn) {
60      // 接收包头
61      char head[HEADLEN];
62      if (conn->read(head, HEADLEN) < 0) {
63          if (conn->eof())
64              logger("connection has been closed, from: %s",
65                  conn->get_peer());
66          else
67              logger_error("read fail: %s, from: %s",
68                  acl::last_serror(), conn->get_peer());
69          return false;
70      }
71
72      // 业务处理
73      service_c service;
74      return service.business(conn, head);
75  }
76
77  // 线程读写连接超时时被调用
78  // 返回true，继续等待下一次读写，否则
79  // 函数返回后即关闭连接
80  bool server_c::thread_on_timeout(acl::socket_stream* conn) {
81      logger("read timeout, from: %s", conn->get_peer());
82      return true;
83  }
84
85  // 与线程绑定的连接关闭时被调用
86  void server_c::thread_on_close(acl::socket_stream* conn) {
87      logger("client disconnect, from: %s", conn->get_peer());
88  }
```

~/TNV/src/03_id/09_main.cpp

```cpp
// ID服务器
// 定义主函数
//
#include "01_globals.h"
#include "07_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("127.0.0.1:22000", "../etc/id.cfg");

    return 0;
}
```

~/TNV/src/03_id/Makefile

```
PROJ   = ../../bin/id
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/id.cfg

```
service id {
    # MySQL地址表
    mysql_addrs = 127.0.0.1
    # MySQL读写超时
    mysql_rw_timeout = 30
    # 最大偏移
    idinc_max_step = 100
}
```

~/TNV/sql/id.sql

```
DROP DATABASE IF EXISTS tnv_idsdb;
CREATE DATABASE tnv_idsdb;
USE tnv_idsdb;

CREATE TABLE `t_id_gen` (
  `id` varchar(64) NOT NULL DEFAULT '',
  `id_value` bigint(20) DEFAULT NULL,
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```