1 公共模块

~/TNV/src/01_common/01_types.h

```c
// 公共模块
// 定义所有模块都会用到的宏和数据类型
//
#pragma once

#include <netinet/in.h>
//
// 函数返回值
//
#define OK             0 // 成功
#define ERROR         -1 // 本地错误
#define SOCKET_ERROR -2 // 套接字通信错误
#define STATUS_ERROR -3 // 服务器状态异常
//
// 缓存
//
#define TRACKER_REDIS_PREFIX "tracker" // 跟踪服务器Redis前缀
#define STORAGE_REDIS_PREFIX "storage" // 存储服务器Redis前缀
//
// 存储服务器状态
//
typedef enum storage_status {
    STORAGE_STATUS_OFFLINE, // 存储服务器离线
    STORAGE_STATUS_ONLINE,  // 存储服务器在线
    STORAGE_STATUS_ACTIVE   // 存储服务器活动
}   storage_status_t;       // 存储服务器状态
//
// 存储服务器加入和信息
//
#define STORAGE_VERSION_MAX     6 // 版本最大字符数
#define STORAGE_GROUPNAME_MAX  16 // 组名最大字符数
#define STORAGE_HOSTNAME_MAX  128 // 主机名最大字符数
#define STORAGE_ADDR_MAX       16 // 地址最大字符数

typedef struct storage_join {
    char      sj_version[STORAGE_VERSION_MAX+1];     // 版本
    char      sj_groupname[STORAGE_GROUPNAME_MAX+1]; // 组名
    char      sj_hostname[STORAGE_HOSTNAME_MAX+1];   // 主机名
    in_port_t sj_port;                               // 端口号
    time_t    sj_stime;                              // 启动时间
    time_t    sj_jtime;                              // 加入时间
}   storage_join_t; // 存储服务器加入

typedef struct storage_info {
    char             si_version[STORAGE_VERSION_MAX+1];   // 版本
    char             si_hostname[STORAGE_HOSTNAME_MAX+1]; // 主机名
    char             si_addr[STORAGE_ADDR_MAX+1];         // IP地址
    in_port_t        si_port;                             // 端口号
    time_t           si_stime;                            // 启动时间
```

```
    time_t           si_jtime;                              // 加入时间
    time_t           si_btime;                              // 心跳时间
    storage_status_t si_status;                             // 状态
}   storage_info_t; // 存储服务器信息
//
// ID键值对
//
#define ID_KEY_MAX 64 // 键最大字符数

typedef struct id_pair {
    char id_key[ID_KEY_MAX+1]; // 键
    long id_value;             // 值
    int  id_offset;            // 偏移
}   id_pair_t;                 // ID键值对
//
// 存储服务器读写磁盘文件缓冲区
//
#define STORAGE_RCVWR_SIZE (512 * 1024) // 接收写入缓冲区字节数
#define STORAGE_RDSND_SIZE (512 * 1024) // 读取发送缓冲区字节数
```

~/TNV/src/01_common/02_proto.h

```
// 公共模块
// 定义与报文规约有关的宏和数据类型
//
#pragma once

#include "01_types.h"
//
// |包体长度|命令|状态|  包体  |
// |    8   |  1 |  1 |包体长度|
//
#define BODYLEN_SIZE 8 // 包体长度字节数
#define COMMAND_SIZE 1 // 命令字节数
#define STATUS_SIZE  1 // 状态字节数
#define HEADLEN (BODYLEN_SIZE + COMMAND_SIZE + STATUS_SIZE) // 包头长度
//
// |包体长度|命令|状态|错误号|错误描述|
// |    8   |  1 |  1 |   2  | <=1024 |
//
#define ERROR_NUMB_SIZE    2 // 错误号字节数
#define ERROR_DESC_SIZE 1024 // 错误描述最大字节数(含结尾空字符)
//
// |包体长度|命令|状态|应用ID|用户ID|文件ID|
// |    8   |  1 |  1 |  16  |  256 |  128 |
//
#define APPID_SIZE   16 // 应用ID最大字节数(含结尾空字符)
#define USERID_SIZE 256 // 用户ID最大字节数(含结尾空字符)
#define FILEID_SIZE 128 // 文件ID最大字节数(含结尾空字符)
//
// 存储服务器加入包和心跳包
//
typedef struct storage_join_body {
    char sjb_version[STORAGE_VERSION_MAX+1];     // 版本
    char sjb_groupname[STORAGE_GROUPNAME_MAX+1]; // 组名
```

```
    char sjb_hostname[STORAGE_HOSTNAME_MAX+1];    // 主机名
    char sjb_port[sizeof(in_port_t)];             // 端口号
    char sjb_stime[sizeof(time_t)];               // 启动时间
    char sjb_jtime[sizeof(time_t)];               // 加入时间
}   storage_join_body_t; // 存储服务器加入包体

typedef struct storage_beat_body {
    char sbb_groupname[STORAGE_GROUPNAME_MAX+1]; // 组名
    char sbb_hostname[STORAGE_HOSTNAME_MAX+1];    // 主机名
}   storage_beat_body_t; // 存储服务器心跳包体
//
// 命令
//
#define CMD_TRACKER_JOIN   10 // 存储服务器向跟踪服务器发送加入包
#define CMD_TRACKER_BEAT   11 // 存储服务器向跟踪服务器发送心跳包
#define CMD_TRACKER_SADDRS 12 // 客户机从跟踪服务器获取存储服务器地址列表
#define CMD_TRACKER_GROUPS 13 // 客户机从跟踪服务器获取组列表

#define CMD_ID_GET 40 // 存储服务器从ID服务器获取ID

#define CMD_STORAGE_UPLOAD   70 // 客户机向存储服务器上传文件
#define CMD_STORAGE_FILESIZE 71 // 客户机向存储服务器询问文件大小
#define CMD_STORAGE_DOWNLOAD 72 // 客户机从存储服务器下载文件
#define CMD_STORAGE_DELETE   73 // 客户机删除存储服务器上的文件

#define CMD_TRACKER_REPLY 100 // 跟踪服务器应答
#define CMD_ID_REPLY      101 // ID服务器应答
#define CMD_STORAGE_REPLY 102 // 存储服务器应答
```

~/TNV/src/01_common/03_util.h

```
// 公共模块
// 声明几个实用函数
//
#pragma once

#include <string>
#include <vector>

// long long类型整数主机序转网络序
void llton(long long ll, char* n);
// long long类型整数网络序转主机序
long long ntoll(char const* n);

// long类型整数主机序转网络序
void lton(long l, char* n);
// long类型整数网络序转主机序
long ntol(char const* n);

// short类型整数主机序转网络序
void ston(short s, char* n);
// short类型整数网络序转主机序
short ntos(char const* n);

// 字符串是否合法，即是否只包含26个英文字母大小写和0到9十个阿拉伯数字
```

```cpp
int valid(char const* str);

// 以分号为分隔符将一个字符串拆分为多个子字符串
int split(char const* str, std::vector<std::string>& substrs);
```

~/TNV/src/01_common/04_util.cpp

```cpp
// 公共模块
// 定义几个实用函数
//
#include <string.h>
#include "01_types.h"
#include "03_util.h"

// long long类型整数主机序转网络序
void llton(long long ll, char* n) {
    for (size_t i = 0; i < sizeof(ll); ++i)
        n[i] = ll >> (sizeof(ll) - i - 1) * 8;
}

// long long类型整数网络序转主机序
long long ntoll(char const* n) {
    long long ll = 0;
    for (size_t i = 0; i < sizeof(ll); ++i)
        ll |= (long long)(unsigned char)n[i] << (sizeof(ll) - i - 1) * 8;
    return ll;
}

// long类型整数主机序转网络序
void lton(long l, char* n) {
    for (size_t i = 0; i < sizeof(l); ++i)
        n[i] = l >> (sizeof(l) - i - 1) * 8;
}

// long类型整数网络序转主机序
long ntol(char const* n) {
    long l = 0;
    for (size_t i = 0; i < sizeof(l); ++i)
        l |= (long)(unsigned char)n[i] << (sizeof(l) - i - 1) * 8;
    return l;
}

// short类型整数主机序转网络序
void ston(short s, char* n) {
    for (size_t i = 0; i < sizeof(s); ++i)
        n[i] = s >> (sizeof(s) - i - 1) * 8;
}

// short类型整数网络序转主机序
short ntos(char const* n) {
    short s = 0;
    for (size_t i = 0; i < sizeof(s); ++i)
        s |= (short)(unsigned char)n[i] << (sizeof(s) - i - 1) * 8;
    return s;
}
```

```
49
50  // 字符串是否合法，即是否只包含26个英文字母大小写和0到9十个阿拉伯数字
51  int valid(char const* str) {
52      if (!str)
53          return ERROR;
54
55      size_t len = strlen(str);
56      if (!len)
57          return ERROR;
58
59      for (size_t i = 0; i < len; ++i)
60          if (!(('a' <= str[i] && str[i] <= 'z') ||
61                ('A' <= str[i] && str[i] <= 'Z') ||
62                ('0' <= str[i] && str[i] <= '9')))
63              return ERROR;
64
65      return OK;
66  }
67
68  // 以分号为分隔符将一个字符串拆分为多个子字符串
69  int split(char const* str, std::vector<std::string>& substrs) {
70      if (!str)
71          return ERROR;
72
73      size_t len = strlen(str);
74      if (!len)
75          return ERROR;
76
77      char* buf = new char[len+1];
78      strcpy(buf, str);
79
80      char const* sep = ";";
81      for (char* substr = strtok(buf, sep); substr;
82          substr = strtok(NULL, sep))
83          substrs.push_back(substr);
84
85      delete[] buf;
86
87      return OK;
88  }
```