

5 客户机

~/TNV/src/05_client/01_conn.h

```
1  // 客户机
2  // 声明连接类
3  //
4  #pragma once
5
6  #include <string>
7  #include <lib_acl.hpp>
8  //
9  // 连接类
10 //
11 class conn_c: public acl::connect_client {
12 public:
13     // 构造函数
14     conn_c(char const* destaddr, int ctimeout = 30,
15           int rtimeout = 60);
16     // 析构函数
17     ~conn_c(void);
18
19     // 从跟踪服务器获取存储服务器地址列表
20     int saddrs(char const* appid, char const* userid,
21               char const* fileid, std::string& saddrs);
22     // 从跟踪服务器获取组列表
23     int groups(std::string& groups);
24
25     // 向存储服务器上传文件
26     int upload(char const* appid, char const* userid,
27               char const* fileid, char const* filepath);
28     // 向存储服务器上传文件
29     int upload(char const* appid, char const* userid,
30               char const* fileid, char const* filedata, long long filesize);
31     // 向存储服务器询问文件大小
32     int filesize(char const* appid, char const* userid,
33                  char const* fileid, long long* filesize);
34     // 从存储服务器下载文件
35     int download(char const* appid, char const* userid,
36                  char const* fileid, long long offset, long long size,
37                  char** filedata, long long* filesize);
38     // 删除存储服务器上的文件
39     int del(char const* appid, char const* userid, char const* fileid);
40
41     // 获取错误号
42     short errnumb(void) const;
43     // 获取错误描述
44     char const* errdesc(void) const;
45
46 protected:
47     // 打开连接
48     bool open(void);
49     // 关闭连接
```

```

50     void close(void);
51
52 private:
53     // 构造请求
54     int makerequ(char command, char const* appid,
55                 char const* userid, char const* fileid, char* requ);
56     // 接收包体
57     int recvbody(char** body, long long* bodylen);
58     // 接收包头
59     int recvhead(long long* bodylen);
60
61     char*          m_destaddr; // 目的地址
62     int            m_ctimeout; // 连接超时
63     int            m_rtimeout; // 读写超时
64     acl::socket_stream* m_conn; // 连接对象
65     short          m_errnumb; // 错误号
66     acl::string     m_errdesc; // 错误描述
67 };

```

~/TNV/src/05_client/02_conn.cpp

```

1  // 客户机
2  // 实现连接类
3  //
4  #include <sys/sendfile.h>
5  #include <lib_acl.h>
6  #include "02_proto.h"
7  #include "03_util.h"
8  #include "01_conn.h"
9
10 // 构造函数
11 conn_c::conn_c(char const* destaddr, int ctimeout /* = 30 */,
12               int rtimeout /* = 60 */): m_ctimeout(ctimeout),
13               m_rtimeout(rtimeout), m_conn(NULL) {
14     // 检查目的地址
15     acl_assert(destaddr && *destaddr);
16     // 复制目的地址
17     m_destaddr = acl_mystrdup(destaddr);
18 }
19
20 // 析构函数
21 conn_c::~~conn_c(void) {
22     // 关闭连接
23     close();
24     // 释放目的地址
25     acl_myfree(m_destaddr);
26 }
27
28 // 从跟踪服务器获取存储服务器地址列表
29 int conn_c::saddrs(char const* appid, char const* userid,
30                   char const* fileid, std::string& saddrs) {
31     // |包体长度|命令|状态|应用ID|用户ID|文件ID|
32     // | 8 | 1 | 1 | 16 | 256 | 128 |
33     // 构造请求
34     long long bodylen = APPID_SIZE + USERID_SIZE + FILEID_SIZE;

```

```

35     long long requlen = HEADLEN + bodylen;
36     char requ[requlen];
37     if (makerequ(CMD_TRACKER_SADDRS,
38         appid, userid, fileid, requ) != OK)
39         return ERROR;
40     llton(bodylen, requ);
41
42     if (!open())
43         return SOCKET_ERROR;
44
45     // 发送请求
46     if (m_conn->write(requ, requlen) < 0) {
47         logger_error("write fail: %s, requlen: %lld, to: %s",
48             acl::last_serror(), requlen, m_conn->get_peer());
49         m_errnumb = -1;
50         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
51             acl::last_serror(), requlen, m_conn->get_peer());
52         close();
53         return SOCKET_ERROR;
54     }
55
56     char* body = NULL; // 包体指针
57
58     // 接收包体
59     int result = recvbody(&body, &bodylen);
60
61     // 解析包体
62     if (result == OK)
63         // |包体长度|命令|状态|组名|存储服务器地址列表|
64         // | 8 | 1 | 1 |16+1| 包体长度-(16+1) |
65         saddrs = body + STORAGE_GROUPNAME_MAX + 1;
66     else if (result == STATUS_ERROR) {
67         // |包体长度|命令|状态|错误号|错误描述|
68         // | 8 | 1 | 1 | 2 | <=1024 |
69         m_errnumb = ntos(body);
70         m_errdesc = bodylen > ERROR_NUMB_SIZE ?
71             body + ERROR_NUMB_SIZE : "";
72     }
73
74     // 释放包体
75     if (body) {
76         free(body);
77         body = NULL;
78     }
79
80     return result;
81 }
82
83 // 从跟踪服务器获取组列表
84 int conn_c::groups(std::string& groups) {
85     // |包体长度|命令|状态|
86     // | 8 | 1 | 1 |
87     // 构造请求
88     long long bodylen = 0;
89     long long requlen = HEADLEN + bodylen;
90     char requ[requlen] = {};

```

```

91     htonl(bodylen, requ);
92     requ[BODYLEN_SIZE] = CMD_TRACKER_GROUPS;
93     requ[BODYLEN_SIZE+COMMAND_SIZE] = 0;
94
95     if (!open())
96         return SOCKET_ERROR;
97
98     // 发送请求
99     if (m_conn->write(requ, requlen) < 0) {
100         logger_error("write fail: %s, requlen: %lld, to: %s",
101             acl::last_serror(), requlen, m_conn->get_peer());
102         m_errnumb = -1;
103         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
104             acl::last_serror(), requlen, m_conn->get_peer());
105         close();
106         return SOCKET_ERROR;
107     }
108
109     char* body = NULL; // 包体指针
110
111     // 接收包体
112     int result = recvbody(&body, &bodylen);
113
114     // 解析包体
115     if (result == OK)
116         // |包体长度|命令|状态|组列表|
117         // | 8 | 1 | 1 |包体长度|
118         groups = body;
119     else if (result == STATUS_ERROR) {
120         // |包体长度|命令|状态|错误号|错误描述|
121         // | 8 | 1 | 1 | 2 | <=1024 |
122         m_errnumb = ntos(body);
123         m_errdesc = bodylen > ERROR_NUMB_SIZE ?
124             body + ERROR_NUMB_SIZE : "";
125     }
126
127     // 释放包体
128     if (body) {
129         free(body);
130         body = NULL;
131     }
132
133     return result;
134 }
135
136 // 向存储服务上传文件
137 int conn_c::upload(char const* appid, char const* userid,
138     char const* fileid, char const* filepath) {
139     // |包体长度|命令|状态|应用ID|用户ID|文件ID|文件大小|文件内容|
140     // | 8 | 1 | 1 | 16 | 256 | 128 | 8 |文件大小|
141     // 构造请求
142     long long bodylen = APPID_SIZE + USERID_SIZE + FILEID_SIZE +
143         BODYLEN_SIZE;
144     long long requlen = HEADLEN + bodylen;
145     char requ[requlen];
146     if (makerequ(CMD_STORAGE_UPLOAD,

```

```

147     appid, userid, fileid, requ) != OK)
148         return ERROR;
149     acl::ifstream ifs;
150     if (!ifs.open_read(filepath)) {
151         logger_error("open file fail, filepath: %s", filepath);
152         return ERROR;
153     }
154     long long filesize = ifs.fsize();
155     llton(filesize, requ + HEADLEN +
156         APPID_SIZE + USERID_SIZE + FILEID_SIZE);
157     bodylen += filesize;
158     llton(bodylen, requ);
159
160     if (!open())
161         return SOCKET_ERROR;
162
163     // 发送请求
164     if (m_conn->write(requ, requlen) < 0) {
165         logger_error("write fail: %s, requlen: %lld, to: %s",
166             acl::last_serror(), requlen, m_conn->get_peer());
167         m_errnumb = -1;
168         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
169             acl::last_serror(), requlen, m_conn->get_peer());
170         close();
171         return SOCKET_ERROR;
172     }
173
174     // 发送文件
175     long long remain = filesize; // 未发送字节数
176     off_t offset = 0; // 文件读写位置
177     while (remain) { // 还有未发送数据
178         // 发送数据
179         long long bytes = std::min(remain,
180             (long long)STORAGE_RCVWR_SIZE);
181         long long count = sendfile(m_conn->sock_handle(),
182             ifs.file_handle(), &offset, bytes);
183         if (count < 0) {
184             logger_error(
185                 "send file fail, filesize: %lld, remain: %lld",
186                 filesize, remain);
187             m_errnumb = -1;
188             m_errdesc.format(
189                 "send file fail, filesize: %lld, remain: %lld",
190                 filesize, remain);
191             close();
192             return SOCKET_ERROR;
193         }
194         // 未发递减
195         remain -= count;
196     }
197     ifs.close();
198
199     char* body = NULL; // 包体指针
200
201     // 接收包体
202     int result = recvbody(&body, &bodylen);

```

```

203
204 // 解析包体
205 if (result == STATUS_ERROR) {
206     // |包体长度|命令|状态|错误号|错误描述|
207     // | 8 | 1 | 1 | 2 | <=1024 |
208     m_errnumb = ntos(body);
209     m_errdesc = bodylen > ERROR_NUMB_SIZE ?
210         body + ERROR_NUMB_SIZE : "";
211 }
212
213 // 释放包体
214 if (body) {
215     free(body);
216     body = NULL;
217 }
218
219 return result;
220 }
221
222 // 向存储服务上传文件
223 int conn_c::upload(char const* appid, char const* userid,
224     char const* fileid, char const* filedata, long long filesize) {
225     // |包体长度|命令|状态|应用ID|用户ID|文件ID|文件大小|文件内容|
226     // | 8 | 1 | 1 | 16 | 256 | 128 | 8 | 文件大小|
227     // 构造请求
228     long long bodylen = APPID_SIZE + USERID_SIZE + FILEID_SIZE +
229         BODYLEN_SIZE;
230     long long requlen = HEADLEN + bodylen;
231     char requ[requlen];
232     if (makerequ(CMD_STORAGE_UPLOAD,
233         appid, userid, fileid, requ) != OK)
234         return ERROR;
235     llton(filesize, requ + HEADLEN +
236         APPID_SIZE + USERID_SIZE + FILEID_SIZE);
237     bodylen += filesize;
238     llton(bodylen, requ);
239
240     if (!open())
241         return SOCKET_ERROR;
242
243     // 发送请求
244     if (m_conn->write(requ, requlen) < 0) {
245         logger_error("write fail: %s, requlen: %lld, to: %s",
246             acl::last_serror(), requlen, m_conn->get_peer());
247         m_errnumb = -1;
248         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
249             acl::last_serror(), requlen, m_conn->get_peer());
250         close();
251         return SOCKET_ERROR;
252     }
253
254     // 发送文件
255     if (m_conn->write(filedata, filesize) < 0) {
256         logger_error("write fail: %s, filesize: %lld, to: %s",
257             acl::last_serror(), filesize, m_conn->get_peer());
258         m_errnumb = -1;

```

```

259         m_errdesc.format("write fail: %s, filesize: %lld, to: %s",
260             acl::last_serror(), filesize, m_conn->get_peer());
261         close();
262         return SOCKET_ERROR;
263     }
264
265     char* body = NULL; // 包体指针
266
267     // 接收包体
268     int result = recvbody(&body, &bodylen);
269
270     // 解析包体
271     if (result == STATUS_ERROR) {
272         // |包体长度|命令|状态|错误号|错误描述|
273         // | 8 | 1 | 1 | 2 | <=1024 |
274         m_errnumb = ntos(body);
275         m_errdesc = bodylen > ERROR_NUMB_SIZE ?
276             body + ERROR_NUMB_SIZE : "";
277     }
278
279     // 释放包体
280     if (body) {
281         free(body);
282         body = NULL;
283     }
284
285     return result;
286 }
287
288 // 向存储服务询问文件大小
289 int conn_c::filesize(char const* appid, char const* userid,
290     char const* fileid, long long* filesize) {
291     // |包体长度|命令|状态|应用ID|用户ID|文件ID|
292     // | 8 | 1 | 1 | 16 | 256 | 128 |
293     // 构造请求
294     long long bodylen = APPID_SIZE + USERID_SIZE + FILEID_SIZE;
295     long long requlen = HEADLEN + bodylen;
296     char requ[requlen];
297     if (makerequ(CMD_STORAGE_FILESIZE,
298         appid, userid, fileid, requ) != OK)
299         return ERROR;
300     llton(bodylen, requ);
301
302     if (!open())
303         return SOCKET_ERROR;
304
305     // 发送请求
306     if (m_conn->write(requ, requlen) < 0) {
307         logger_error("write fail: %s, requlen: %lld, to: %s",
308             acl::last_serror(), requlen, m_conn->get_peer());
309         m_errnumb = -1;
310         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
311             acl::last_serror(), requlen, m_conn->get_peer());
312         close();
313         return SOCKET_ERROR;
314     }

```

```

315
316     char* body = NULL; // 包体指针
317
318     // 接收包体
319     int result = recvbody(&body, &bodylen);
320
321     // 解析包体
322     if (result == OK)
323         // |包体长度|命令|状态|文件大小|
324         // | 8 | 1 | 1 | 8 |
325         *filesize = ntoll(body);
326     else if (result == STATUS_ERROR) {
327         // |包体长度|命令|状态|错误号|错误描述|
328         // | 8 | 1 | 1 | 2 | <=1024 |
329         m_errnumb = ntos(body);
330         m_errdesc = bodylen > ERROR_NUMB_SIZE ?
331             body + ERROR_NUMB_SIZE : "";
332     }
333
334     // 释放包体
335     if (body) {
336         free(body);
337         body = NULL;
338     }
339
340     return result;
341 }
342
343 // 从存储服务器下载文件
344 int conn_c::download(char const* appid, char const* userid,
345     char const* fileid, long long offset, long long size,
346     char** filedata, long long* filesize) {
347     // |包体长度|命令|状态|应用ID|用户ID|文件ID|偏移|大小|
348     // | 8 | 1 | 1 | 16 | 256 | 128 | 8 | 8 |
349     // 构造请求
350     long long bodylen = APPID_SIZE + USERID_SIZE + FILEID_SIZE +
351         BODYLEN_SIZE + BODYLEN_SIZE;
352     long long requlen = HEADLEN + bodylen;
353     char requ[requlen];
354     if (makerequ(CMD_STORAGE_DOWNLOAD,
355         appid, userid, fileid, requ) != OK)
356         return ERROR;
357     llton(bodylen, requ);
358     llton(offset, requ + HEADLEN +
359         APPID_SIZE + USERID_SIZE + FILEID_SIZE);
360     llton(size, requ + HEADLEN +
361         APPID_SIZE + USERID_SIZE + FILEID_SIZE + BODYLEN_SIZE);
362
363     if (!open())
364         return SOCKET_ERROR;
365
366     // 发送请求
367     if (m_conn->write(requ, requlen) < 0) {
368         logger_error("write fail: %s, requlen: %lld, to: %s",
369             acl::last_serror(), requlen, m_conn->get_peer());
370         m_errnumb = -1;

```

```

371         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
372             acl::last_error(), requlen, m_conn->get_peer());
373         close();
374         return SOCKET_ERROR;
375     }
376
377     char* body = NULL; // 包体指针
378
379     // 接收包体
380     int result = recvbody(&body, &bodylen);
381
382     // 解析包体
383     if (result == OK) {
384         // |包体长度|命令|状态|文件内容|
385         // | 8 | 1 | 1 | 内容大小|
386         *filedata = body;
387         *filesize = bodylen;
388         return result;
389     }
390     if (result == STATUS_ERROR) {
391         // |包体长度|命令|状态|错误号|错误描述|
392         // | 8 | 1 | 1 | 2 | <=1024 |
393         m_errnumb = ntos(body);
394         m_errdesc = bodylen > ERROR_NUMB_SIZE ?
395             body + ERROR_NUMB_SIZE : "";
396     }
397
398     // 释放包体
399     if (body) {
400         free(body);
401         body = NULL;
402     }
403
404     return result;
405 }
406
407 // 删除存储服务器上的文件
408 int conn_c::del(char const* appid, char const* userid,
409     char const* fileid) {
410     // |包体长度|命令|状态|应用ID|用户ID|文件ID|
411     // | 8 | 1 | 1 | 16 | 256 | 128 |
412     // 构造请求
413     long long bodylen = APPID_SIZE + USERID_SIZE + FILEID_SIZE;
414     long long requlen = HEADLEN + bodylen;
415     char requ[requlen];
416     if (makerequ(CMD_STORAGE_DELETE,
417         appid, userid, fileid, requ) != OK)
418         return ERROR;
419     hton(bodylen, requ);
420
421     if (!open())
422         return SOCKET_ERROR;
423
424     // 发送请求
425     if (m_conn->write(requ, requlen) < 0) {
426         logger_error("write fail: %s, requlen: %lld, to: %s",

```

```

427         acl::last_error(), requlen, m_conn->get_peer());
428         m_errnumb = -1;
429         m_errdesc.format("write fail: %s, requlen: %lld, to: %s",
430             acl::last_error(), requlen, m_conn->get_peer());
431         close();
432         return SOCKET_ERROR;
433     }
434
435     char* body = NULL; // 包体指针
436
437     // 接收包体
438     int result = recvbody(&body, &bodylen);
439
440     // 解析包体
441     if (result == STATUS_ERROR) {
442         // |包体长度|命令|状态|错误号|错误描述|
443         // | 8 | 1 | 1 | 2 | <=1024 |
444         m_errnumb = ntos(body);
445         m_errdesc = bodylen > ERROR_NUMB_SIZE ?
446             body + ERROR_NUMB_SIZE : "";
447     }
448
449     // 释放包体
450     if (body) {
451         free(body);
452         body = NULL;
453     }
454
455     return result;
456 }
457
458 // 获取错误号
459 short conn_c::errnumb(void) const {
460     return m_errnumb;
461 }
462
463 // 获取错误描述
464 char const* conn_c::errdesc(void) const {
465     return m_errdesc.c_str();
466 }
467
468 // 打开连接
469 bool conn_c::open(void) {
470     if (m_conn)
471         return true;
472
473     // 创建连接对象
474     m_conn = new acl::socket_stream;
475
476     // 连接目的主机
477     if (!m_conn->open(m_destaddr, m_ctimeout, m_rtimeout)) {
478         logger_error("open %s fail: %s",
479             m_destaddr, acl_last_error());
480         delete m_conn;
481         m_conn = NULL;
482         m_errnumb = -1;

```

```

483         m_errdesc.format("open %s fail: %s",
484             m_destaddr, acl_last_error());
485         return false;
486     }
487
488     return true;
489 }
490
491 // 关闭连接
492 void conn_c::close(void) {
493     if (m_conn) {
494         delete m_conn;
495         m_conn = NULL;
496     }
497 }
498
499 // 构造请求
500 int conn_c::makerequ(char command, char const* appid,
501     char const* userid, char const* fileid, char* requ) {
502     // |包体长度|命令|状态|应用ID|用户ID|文件ID|
503     // | 8 | 1 | 1 | 16 | 256 | 128 |
504     requ[BODYLEN_SIZE] = command; // 命令
505     requ[BODYLEN_SIZE+COMMAND_SIZE] = 0; // 状态
506
507     // 应用ID
508     if (strlen(appid) >= APPID_SIZE) {
509         logger_error("appid too big, %lu >= %d",
510             strlen(appid), APPID_SIZE);
511         m_errnumb = -1;
512         m_errdesc.format("appid too big, %lu >= %d",
513             strlen(appid), APPID_SIZE);
514         return ERROR;
515     }
516     strcpy(requ + HEADLEN, appid);
517
518     // 用户ID
519     if (strlen(userid) >= USERID_SIZE) {
520         logger_error("userid too big, %lu >= %d",
521             strlen(userid), USERID_SIZE);
522         m_errnumb = -1;
523         m_errdesc.format("userid too big, %lu >= %d",
524             strlen(userid), USERID_SIZE);
525         return ERROR;
526     }
527     strcpy(requ + HEADLEN + APPID_SIZE, userid);
528
529     // 文件ID
530     if (strlen(fileid) >= FILEID_SIZE) {
531         logger_error("fileid too big, %lu >= %d",
532             strlen(fileid), FILEID_SIZE);
533         m_errnumb = -1;
534         m_errdesc.format("fileid too big, %lu >= %d",
535             strlen(fileid), FILEID_SIZE);
536         return ERROR;
537     }
538     strcpy(requ + HEADLEN + APPID_SIZE + USERID_SIZE, fileid);

```

```

539         return OK;
540     }
541 }
542
543 // 接收包体
544 int conn_c::recvbody(char** body, long long* bodylen) {
545     // 接收包头
546     int result = recvhead(bodylen);
547
548     // 既非本地错误亦非套接字通信错误且包体非空
549     if (result != ERROR && result != SOCKET_ERROR && *bodylen) {
550         // 分配包体
551         if (!(*body = (char*)malloc(*bodylen))) {
552             logger_error("call malloc fail: %s, bodylen: %lld",
553                 strerror(errno), *bodylen);
554             m_errnumb = -1;
555             m_errdesc.format("call malloc fail: %s, bodylen: %lld",
556                 strerror(errno), *bodylen);
557             return ERROR;
558         }
559
560         // 接收包体
561         if (m_conn->read(*body, *bodylen) < 0) {
562             logger_error("read fail: %s, from: %s",
563                 acl::last_serror(), m_conn->get_peer());
564             m_errnumb = -1;
565             m_errdesc.format("read fail: %s, from: %s",
566                 acl::last_serror(), m_conn->get_peer());
567             free(*body);
568             *body = NULL;
569             close();
570             return SOCKET_ERROR;
571         }
572     }
573
574     return result;
575 }
576
577 // 接收包头
578 int conn_c::recvhead(long long* bodylen) {
579     if (!open())
580         return SOCKET_ERROR;
581
582     char head[HEADLEN]; // 包头缓冲区
583
584     // 接收包头
585     if (m_conn->read(head, HEADLEN) < 0) {
586         logger_error("read fail: %s, from: %s",
587             acl::last_serror(), m_conn->get_peer());
588         m_errnumb = -1;
589         m_errdesc.format("read fail: %s, from: %s",
590             acl::last_serror(), m_conn->get_peer());
591         close();
592         return SOCKET_ERROR;
593     }
594

```

```

595 // |包体长度|命令|状态|
596 // | 8 | 1 | 1 |
597 // 解析包头
598 if ((*bodylen = ntohl(head)) < 0) { // 包体长度
599     logger_error("invalid body length: %lld < 0, from: %s",
600         *bodylen, m_conn->get_peer());
601     m_errnumb = -1;
602     m_errrdesc.format("invalid body length: %lld < 0, from: %s",
603         *bodylen, m_conn->get_peer());
604     return ERROR;
605 }
606 int command = head[BODYLEN_SIZE]; // 命令
607 int status = head[BODYLEN_SIZE+COMMAND_SIZE]; // 状态
608 if (status) {
609     logger_error("response status %d != 0, from: %s",
610         status, m_conn->get_peer());
611     return STATUS_ERROR;
612 }
613 logger("bodylen: %lld, command: %d, status: %d",
614     *bodylen, command, status);
615
616 return OK;
617 }

```

~/TNV/src/05_client/03_pool.h

```

1 // 客户机
2 // 声明连接池类
3 //
4 #pragma once
5
6 #include <lib_acl.hpp>
7 //
8 // 连接池类
9 //
10 class pool_c: public acl::connect_pool {
11 public:
12     // 构造函数
13     pool_c(char const* destaddr, int count, size_t index = 0);
14
15     // 设置超时
16     void timeouts(int ctimeout = 30, int rtimeout = 60,
17         int itimeout = 90);
18     // 获取连接
19     acl::connect_client* peek(void);
20
21 protected:
22     // 创建连接
23     acl::connect_client* create_connect(void);
24
25 private:
26     int m_ctimeout; // 连接超时
27     int m_rtimeout; // 读写超时
28     int m_itimeout; // 空闲超时

```

~/TNV/src/05_client/04_pool.cpp

```

1 // 客户机
2 // 实现连接池类
3 //
4 #include "01_conn.h"
5 #include "03_pool.h"
6
7 // 构造函数
8 pool_c::pool_c(char const* destaddr, int count, size_t index /* = 0 */):
9     connect_pool(destaddr, count, index),
10     m_ctimeout(30), m_rtimeout(60), m_itypeout(90) {
11 }
12
13 // 设置超时
14 void pool_c::timeouts(int ctimeout /* = 30 */, int rtimeout /* = 60 */,
15     int itimeout /* = 90 */) {
16     m_ctimeout = ctimeout;
17     m_rtimeout = rtimeout;
18     m_itypeout = itimeout;
19 }
20
21 // 获取连接
22 acl::connect_client* pool_c::peek(void) {
23     connect_pool::check_idle(m_itypeout);
24     return connect_pool::peek();
25 }
26
27 // 创建连接
28 acl::connect_client* pool_c::create_connect(void) {
29     return new conn_c(addr_, m_ctimeout, m_rtimeout);
30 }

```

~/TNV/src/05_client/05_mgr.h

```

1 // 客户机
2 // 声明连接池管理器类
3 //
4 #pragma once
5
6 #include <lib_acl.hpp>
7 //
8 // 连接池管理器类
9 //
10 class mgr_c: public acl::connect_manager {
11 protected:
12     // 创建连接池
13     acl::connect_pool* create_pool(
14         char const* destaddr, size_t count, size_t index);
15 };

```

~/TNV/src/05_client/06_mngr.cpp

```
1 // 客户机
2 // 实现连接池管理器类
3 //
4 #include "03_pool.h"
5 #include "05_mngr.h"
6
7 // 创建连接池
8 acl::connect_pool* mngr_c::create_pool(
9     char const* destaddr, size_t count, size_t index) {
10     return new pool_c(destaddr, count, index);
11 }
```

~/TNV/src/05_client/07_client.h

```
1 // 客户机
2 // 声明客户机类
3 //
4 #pragma once
5
6 #include <vector>
7 #include <string>
8 #include <lib_acl.hpp>
9 //
10 // 客户机类
11 //
12 class client_c {
13 public:
14     // 初始化
15     static int init(char const* taddr,
16         int tcount = 16, int scount = 8);
17     // 终结化
18     static void deinit(void);
19
20     // 从跟踪服务器获取存储服务器地址列表
21     int saddrs(char const* appid, char const* userid,
22         char const* fileid, std::string& saddrs);
23     // 从跟踪服务器获取组列表
24     int groups(std::string& groups);
25
26     // 向存储服务器上传文件
27     int upload(char const* appid, char const* userid,
28         char const* fileid, char const* filepath);
29     // 向存储服务器上传文件
30     int upload(char const* appid, char const* userid,
31         char const* fileid, char const* filedata, long long filesize);
32     // 向存储服务器询问文件大小
33     int filesize(char const* appid, char const* userid,
34         char const* fileid, long long* filesize);
35     // 从存储服务器下载文件
36     int download(char const* appid, char const* userid,
37         char const* fileid, long long offset, long long size,
38         char** filedata, long long* filesize);
```

```

39 // 删除存储服务器上的文件
40 int del(char const* appid, char const* userid, char const* fileid);
41
42 private:
43     static acl::connect_manager* s_mngr; // 连接池管理器
44     static std::vector<std::string> s_taddrs; // 跟踪服务器地址表
45     static int s_scount; // 存储服务器连接数上限
46 };

```

~/TNV/src/05_client/08_client.cpp

```

1 // 客户机
2 // 实现客户机类
3 //
4 #include <lib_acl.h>
5 #include "01_types.h"
6 #include "03_util.h"
7 #include "01_conn.h"
8 #include "03_pool.h"
9 #include "05_mngr.h"
10 #include "07_client.h"
11
12 #define MAX_SOCKERRS 10 // 套接字通信错误最大次数
13
14 acl::connect_manager* client_c::s_mngr = NULL;
15 std::vector<std::string> client_c::s_taddrs;
16 int client_c::s_scount = 8;
17
18 // 初始化
19 int client_c::init(char const* taddrs,
20     int tcount /* = 16 */, int scount /* = 8 */) {
21     if (s_mngr)
22         return OK;
23
24     // 跟踪服务器地址表
25     if (!taddrs || !*taddrs) {
26         logger_error("tracker addresses is null");
27         return ERROR;
28     }
29     split(taddrs, s_taddrs);
30     if (s_taddrs.empty()) {
31         logger_error("tracker addresses is empty");
32         return ERROR;
33     }
34
35     // 跟踪服务器连接数上限
36     if (tcount <= 0) {
37         logger_error("invalid tracker connection pool count %d <= 0",
38             tcount);
39         return ERROR;
40     }
41
42     // 存储服务器连接数上限
43     if (scount <= 0) {
44         logger_error("invalid storage connection pool count %d <= 0",

```

```

45         scount);
46         return ERROR;
47     }
48     s_scount = scount;
49
50     // 创建连接池管理器
51     if (!(s_mngr = new mngr_c)) {
52         logger_error("create connection pool manager fail: %s",
53             acl_last_serror());
54         return ERROR;
55     }
56
57     // 初始化跟踪服务器连接池
58     s_mngr->init(NULL, taddrs, tcount);
59
60     return OK;
61 }
62
63 // 终结化
64 void client_c::deinit(void) {
65     if (s_mngr) {
66         delete s_mngr;
67         s_mngr = NULL;
68     }
69
70     s_taddrs.clear();
71 }
72
73 // 从跟踪服务器获取存储服务器地址列表
74 int client_c::saddrs(char const* appid, char const* userid,
75     char const* fileid, std::string& saddrs) {
76     if (s_taddrs.empty()) {
77         logger_error("tracker addresses is empty");
78         return ERROR;
79     }
80
81     int result = ERROR;
82
83     // 生成有限随机数
84     srand(time(NULL));
85     int ntaddrs = s_taddrs.size();
86     int nrand = rand() % ntaddrs;
87
88     for (int i = 0; i < ntaddrs; ++i) {
89         // 随机抽取跟踪服务器地址
90         char const* taddr = s_taddrs[nrand].c_str();
91         nrand = (nrand + 1) % ntaddrs;
92
93         // 获取跟踪服务器连接池
94         pool_c* tpool = (pool_c*)s_mngr->get(taddr);
95         if (!tpool) {
96             logger_warn("tracker connection pool is null, taddr: %s",
97                 taddr);
98             continue;
99         }
100

```

```

101     for (int sockerrs = 0; sockerrs < MAX_SOCKERRS; ++sockerrs) {
102         // 获取跟踪服务器连接
103         conn_c* tconn = (conn_c*)tpool->peek();
104         if (!tconn) {
105             logger_warn("tracker connection is null, taddr: %s",
106                 taddr);
107             break;
108         }
109
110         // 从跟踪服务器获取存储服务器地址列表
111         result = tconn->saddrs(appid, userid, fileid, saddrs);
112
113         if (result == SOCKET_ERROR) {
114             logger_warn("get storage addresses fail: %s",
115                 tconn->errdesc());
116             tpool->put(tconn, false);
117         }
118         else {
119             if (result == OK)
120                 tpool->put(tconn, true);
121             else {
122                 logger_error("get storage addresses fail: %s",
123                     tconn->errdesc());
124                 tpool->put(tconn, false);
125             }
126             return result;
127         }
128     }
129 }
130
131 return result;
132 }
133
134 // 从跟踪服务器获取组列表
135 int client_c::groups(std::string& groups) {
136     if (s_taddrs.empty()) {
137         logger_error("tracker addresses is empty");
138         return ERROR;
139     }
140
141     int result = ERROR;
142
143     // 生成有限随机数
144     srand(time(NULL));
145     int ntaddrs = s_taddrs.size();
146     int nrand = rand() % ntaddrs;
147
148     for (int i = 0; i < ntaddrs; ++i) {
149         // 随机抽取跟踪服务器地址
150         char const* taddr = s_taddrs[nrand].c_str();
151         nrand = (nrand + 1) % ntaddrs;
152
153         // 获取跟踪服务器连接池
154         pool_c* tpool = (pool_c*)s_mgr->get(taddr);
155         if (!tpool) {
156             logger_warn("tracker connection pool is null, taddr: %s",

```

```

157         taddr);
158         continue;
159     }
160
161     for (int sockerrs = 0; sockerrs < MAX_SOCKETS; ++sockerrs) {
162         // 获取跟踪服务器连接
163         conn_c* tconn = (conn_c*)tpool->peek();
164         if (!tconn) {
165             logger_warn("tracker connection is null, taddr: %s",
166                 taddr);
167             break;
168         }
169
170         // 从跟踪服务器获取组列表
171         result = tconn->groups(groups);
172
173         if (result == SOCKET_ERROR) {
174             logger_warn("get groups fail: %s", tconn->errdesc());
175             tpool->put(tconn, false);
176         }
177         else {
178             if (result == OK)
179                 tpool->put(tconn, true);
180             else {
181                 logger_error("get groups fail: %s",
182                     tconn->errdesc());
183                 tpool->put(tconn, false);
184             }
185             return result;
186         }
187     }
188 }
189
190 return result;
191 }
192
193 // 向存储服务器上传文件
194 int client_c::upload(char const* appid, char const* userid,
195     char const* fileid, char const* filepath) {
196     // 检查参数
197     if (!appid || !*appid) {
198         logger_error("appid is null");
199         return ERROR;
200     }
201     if (!userid || !*userid) {
202         logger_error("userid is null");
203         return ERROR;
204     }
205     if (!fileid || !*fileid) {
206         logger_error("fileid is null");
207         return ERROR;
208     }
209     if (!filepath || !*filepath) {
210         logger_error("filepath is null");
211         return ERROR;
212     }

```

```

213
214 // 从跟踪服务器获取存储服务器地址列表
215 int result;
216 std::string ssaddrs;
217 if ((result = saddr(sappid, userid, fileid, ssaddrs)) != OK)
218     return result;
219 std::vector<std::string> vsaddrs;
220 split(ssaddrs.c_str(), vsaddrs);
221 if (vsaddrs.empty()) {
222     logger_error("storage addresses is empty");
223     return ERROR;
224 }
225
226 result = ERROR;
227
228 for (std::vector<std::string>::const_iterator saddr =
229     vsaddrs.begin(); saddr != vsaddrs.end(); ++saddr) {
230     // 获取存储服务器连接池
231     pool_c* spool = (pool_c*)s_mgr->get(saddr->c_str());
232     if (!spool) {
233         s_mgr->set(saddr->c_str(), s_scount);
234         if (!(spool = (pool_c*)s_mgr->get(saddr->c_str()))) {
235             logger_warn(
236                 "storage connection pool is null, saddr: %s",
237                 saddr->c_str());
238             continue;
239         }
240     }
241
242     for (int sockerrs = 0; sockerrs < MAX_SOCKETERRS; ++sockerrs) {
243         // 获取存储服务器连接
244         conn_c* sconn = (conn_c*)spool->peek();
245         if (!sconn) {
246             logger_warn("storage connection is null, saddr: %s",
247                 saddr->c_str());
248             break;
249         }
250
251         // 向存储服务器上文件
252         result = sconn->upload(sappid, userid, fileid, filepath);
253
254         if (result == SOCKET_ERROR) {
255             logger_warn("upload file fail: %s", sconn->errdesc());
256             spool->put(sconn, false);
257         }
258         else {
259             if (result == OK)
260                 spool->put(sconn, true);
261             else {
262                 logger_error("upload file fail: %s",
263                     sconn->errdesc());
264                 spool->put(sconn, false);
265             }
266             return result;
267         }
268     }

```

```

269     }
270
271     return result;
272 }
273
274 // 向存储服务上传文件
275 int client_c::upload(char const* appid, char const* userid,
276     char const* fileid, char const* filedata, long long filesize) {
277     // 检查参数
278     if (!appid || !*appid) {
279         logger_error("appid is null");
280         return ERROR;
281     }
282     if (!userid || !*userid) {
283         logger_error("userid is null");
284         return ERROR;
285     }
286     if (!fileid || !*fileid) {
287         logger_error("fileid is null");
288         return ERROR;
289     }
290     if (!filedata || !*filedata) {
291         logger_error("filedata is null");
292         return ERROR;
293     }
294
295     // 从跟踪服务器获取存储服务地址列表
296     int result;
297     std::string ssaddrs;
298     if ((result = saddrss(appid, userid, fileid, ssaddrs)) != OK)
299         return result;
300     std::vector<std::string> vsaddrs;
301     split(ssaddrs.c_str(), vsaddrs);
302     if (vsaddrs.empty()) {
303         logger_error("storage addresses is empty");
304         return ERROR;
305     }
306
307     result = ERROR;
308
309     for (std::vector<std::string>::const_iterator saddr =
310         vsaddrs.begin(); saddr != vsaddrs.end(); ++saddr) {
311         // 获取存储服务连接池
312         pool_c* spool = (pool_c*)s_mgr->get(saddr->c_str());
313         if (!spool) {
314             s_mgr->set(saddr->c_str(), s_scount);
315             if (!(spool = (pool_c*)s_mgr->get(saddr->c_str()))) {
316                 logger_warn(
317                     "storage connection pool is null, saddr: %s",
318                     saddr->c_str());
319                 continue;
320             }
321         }
322
323         for (int sockerrs = 0; sockerrs < MAX_SOCKETS; ++sockerrs) {
324             // 获取存储服务连接

```

```

325         conn_c* sconn = (conn_c*)spool->peek();
326         if (!sconn) {
327             logger_warn("storage connection is null, saddr: %s",
328                 saddr->c_str());
329             break;
330         }
331
332         // 向存储服务上传文件
333         result = sconn->upload(
334             appid, userid, fileid, filedata, filesize);
335
336         if (result == SOCKET_ERROR) {
337             logger_warn("upload file fail: %s", sconn->errdesc());
338             spool->put(sconn, false);
339         }
340         else {
341             if (result == OK)
342                 spool->put(sconn, true);
343             else {
344                 logger_error("upload file fail: %s",
345                     sconn->errdesc());
346                 spool->put(sconn, false);
347             }
348             return result;
349         }
350     }
351 }
352
353 return result;
354 }
355
356 // 向存储服务询问文件大小
357 int client_c::filesize(char const* appid, char const* userid,
358     char const* fileid, long long* filesize) {
359     // 检查参数
360     if (!appid || !*appid) {
361         logger_error("appid is null");
362         return ERROR;
363     }
364     if (!userid || !*userid) {
365         logger_error("userid is null");
366         return ERROR;
367     }
368     if (!fileid || !*fileid) {
369         logger_error("fileid is null");
370         return ERROR;
371     }
372
373     // 从跟踪服务器获取存储服务地址列表
374     int result;
375     std::string ssaddrs;
376     if ((result = saddrs(appid, userid, fileid, ssaddrs)) != OK)
377         return ERROR;
378     std::vector<std::string> vsaddrs;
379     split(ssaddrs.c_str(), vsaddrs);
380     if (vsaddrs.empty()) {

```

```

381     logger_error("storage addresses is empty");
382     return ERROR;
383 }
384
385 result = ERROR;
386
387 for (std::vector<std::string>::const_iterator saddr =
388      vsaddrs.begin(); saddr != vsaddrs.end(); ++saddr) {
389     // 获取存储服务器连接池
390     pool_c* spool = (pool_c*)s_mgr->get(saddr->c_str());
391     if (!spool) {
392         s_mgr->set(saddr->c_str(), s_scount);
393         if (!(spool = (pool_c*)s_mgr->get(saddr->c_str()))) {
394             logger_warn(
395                 "storage connection pool is null, saddr: %s",
396                 saddr->c_str());
397             continue;
398         }
399     }
400
401     for (int sockerrs = 0; sockerrs < MAX_SOCKERRS; ++sockerrs) {
402         // 获取存储服务器连接
403         conn_c* sconn = (conn_c*)spool->peek();
404         if (!sconn) {
405             logger_warn("storage connection is null, saddr: %s",
406                         saddr->c_str());
407             break;
408         }
409
410         // 向存储服务器询问文件大小
411         result = sconn->filesize(appid, userid, fileid, filesize);
412
413         if (result == SOCKET_ERROR) {
414             logger_warn("get filesize fail: %s", sconn->errdesc());
415             spool->put(sconn, false);
416         }
417         else {
418             if (result == OK)
419                 spool->put(sconn, true);
420             else {
421                 logger_error("get filesize fail: %s",
422                             sconn->errdesc());
423                 spool->put(sconn, false);
424             }
425             return result;
426         }
427     }
428 }
429
430 return result;
431 }
432
433 // 从存储服务器下载文件
434 int client_c::download(char const* appid, char const* userid,
435                       char const* fileid, long long offset, long long size,
436                       char** filedata, long long* filesize) {

```

```

437 // 检查参数
438 if (!appid || !*appid) {
439     logger_error("appid is null");
440     return ERROR;
441 }
442 if (!userid || !*userid) {
443     logger_error("userid is null");
444     return ERROR;
445 }
446 if (!fileid || !*fileid) {
447     logger_error("fileid is null");
448     return ERROR;
449 }
450
451 // 从跟踪服务器获取存储服务器地址列表
452 int result;
453 std::string ssaddrs;
454 if ((result = saddrs(appid, userid, fileid, ssaddrs)) != OK)
455     return ERROR;
456 std::vector<std::string> vsaddrs;
457 split(ssaddrs.c_str(), vsaddrs);
458 if (vsaddrs.empty()) {
459     logger_error("storage addresses is empty");
460     return ERROR;
461 }
462
463 result = ERROR;
464
465 for (std::vector<std::string>::const_iterator saddr =
466     vsaddrs.begin(); saddr != vsaddrs.end(); ++saddr) {
467     // 获取存储服务器连接池
468     pool_c* spool = (pool_c*)s_mgr->get(saddr->c_str());
469     if (!spool) {
470         s_mgr->set(saddr->c_str(), s_scount);
471         if (!(spool = (pool_c*)s_mgr->get(saddr->c_str()))) {
472             logger_warn(
473                 "storage connection pool is null, saddr: %s",
474                 saddr->c_str());
475             continue;
476         }
477     }
478
479     for (int sockerrs = 0; sockerrs < MAX_SOCKETERRS; ++sockerrs) {
480         // 获取存储服务器连接
481         conn_c* sconn = (conn_c*)spool->peek();
482         if (!sconn) {
483             logger_warn("storage connection is null, saddr: %s",
484                 saddr->c_str());
485             break;
486         }
487
488         // 从存储服务器下载文件
489         result = sconn->download(
490             appid, userid, fileid, offset, size, filedata, filesize);
491
492         if (result == SOCKET_ERROR) {

```

```

493         logger_warn("download file fail: %s", sconn->errdesc());
494         spool->put(sconn, false);
495     }
496     else {
497         if (result == OK)
498             spool->put(sconn, true);
499         else {
500             logger_error("download file fail: %s",
501                 sconn->errdesc());
502             spool->put(sconn, false);
503         }
504         return result;
505     }
506 }
507 }
508
509 return result;
510 }
511
512 // 删除存储服务器上的文件
513 int client_c::del(char const* appid, char const* userid,
514     char const* fileid) {
515     // 检查参数
516     if (!appid || !*appid) {
517         logger_error("appid is null");
518         return ERROR;
519     }
520     if (!userid || !*userid) {
521         logger_error("userid is null");
522         return ERROR;
523     }
524     if (!fileid || !*fileid) {
525         logger_error("fileid is null");
526         return ERROR;
527     }
528
529     // 从跟踪服务器获取存储服务器地址列表
530     int result;
531     std::string ssaddrs;
532     if ((result = saddrs(appid, userid, fileid, ssaddrs)) != OK)
533         return ERROR;
534     std::vector<std::string> vsaddrs;
535     split(ssaddrs.c_str(), vsaddrs);
536     if (vsaddrs.empty()) {
537         logger_error("storage addresses is empty");
538         return ERROR;
539     }
540
541     result = ERROR;
542
543     for (std::vector<std::string>::const_iterator saddr =
544         vsaddrs.begin(); saddr != vsaddrs.end(); ++saddr) {
545         // 获取存储服务器连接池
546         pool_c* spool = (pool_c*)s_mgr->get(saddr->c_str());
547         if (!spool) {
548             s_mgr->set(saddr->c_str(), s_scount);

```

```

549         if (!(spool = (pool_c*)s_mgr->get(saddr->c_str())) {
550             logger_warn(
551                 "storage connection pool is null, saddr: %s",
552                 saddr->c_str());
553             continue;
554         }
555     }
556
557     for (int sockerrs = 0; sockerrs < MAX_SOCKERRS; ++sockerrs) {
558         // 获取存储服务器连接
559         conn_c* sconn = (conn_c*)spool->peek();
560         if (!sconn) {
561             logger_warn("storage connection is null, saddr: %s",
562                 saddr->c_str());
563             break;
564         }
565
566         // 删除存储服务器上的文件
567         result = sconn->del(appid, userid, fileid);
568
569         if (result == SOCKET_ERROR) {
570             logger_warn("delete file fail: %s", sconn->errdesc());
571             spool->put(sconn, false);
572         }
573         else {
574             if (result == OK)
575                 spool->put(sconn, true);
576             else {
577                 logger_error("delete file fail: %s",
578                     sconn->errdesc());
579                 spool->put(sconn, false);
580             }
581             return result;
582         }
583     }
584 }
585
586 return result;
587 }

```

~/TNV/src/05_client/09_main.cpp

```

1  // 客户机
2  // 定义主函数
3  //
4  #include <unistd.h>
5  #include <lib_acl.h>
6  #include <lib_acl.hpp>
7  #include "01_types.h"
8  #include "07_client.h"
9
10 // 打印命令行用法
11 void usage(char const* cmd) {
12     fprintf(stderr, "Groups : %s <taddr> groups\n", cmd);
13     fprintf(stderr, "Upload : %s <taddr> upload "

```

```

14     "<appid> <userid> <filepath>\n", cmd);
15     fprintf(stderr, "Filesize: %s <taddr> filesize "
16     "<appid> <userid> <fileid>\n", cmd);
17     fprintf(stderr, "Download: %s <taddr> download "
18     "<appid> <userid> <fileid> <offset> <size>\n", cmd);
19     fprintf(stderr, "Delete : %s <taddr> delete "
20     "<appid> <userid> <fileid>\n", cmd);
21 }
22
23 // 根据用户ID生成文件ID
24 std::string genfileid(char const* userid) {
25     srand(time(NULL));
26
27     struct timeval now;
28     gettimeofday(&now, NULL);
29
30     acl::string str;
31     str.format("%s%d%lx%d",
32         userid, getpid(), acl_pthread_self(), rand());
33
34     acl::md5 md5;
35     md5.update(str.c_str(), str.size());
36     md5.finish();
37
38     char buf[33] = {};
39     strncpy(buf, md5.get_string(), 32);
40     memmove(buf, buf + 8, 16);
41     memset(buf + 16, 0, 16);
42
43     static int count = 0;
44     if (count >= 8000)
45         count = 0;
46
47     acl::string fileid;
48     fileid.format("%08lx%06lx%s%04d%02d", now.tv_sec,
49         now.tv_usec, buf, ++count, rand() % 100);
50
51     return fileid.c_str();
52 }
53
54 int main(int argc, char* argv[]) {
55     char const* cmd = argv[0];
56
57     if (argc < 3) {
58         usage(cmd);
59         return -1;
60     }
61
62     char const* taddr = argv[1];
63     char const* subcmd = argv[2];
64
65     // 初始化ACL库
66     acl::acl_cpp_init();
67
68     // 日志打印到标准输出
69     acl::log::stdout_open(true);

```

```

70
71 // 初始化客户机
72 if (client_c::init(taddrs) != OK)
73     return -1;
74
75 client_c client; // 客户机对象
76
77 // 从跟踪服务器获取组列表
78 if (!strcmp(subcmd, "groups")) {
79     std::string groups;
80     if (client.groups(groups) != OK) {
81         client_c::deinit();
82         return -1;
83     }
84     printf("%s\n", groups.c_str());
85 }
86 else
87 // 向存储服务器上传文件
88 if (!strcmp(subcmd, "upload")) {
89     if (argc < 6) {
90         client_c::deinit();
91         usage(cmd);
92         return -1;
93     }
94     char const* appid    = argv[3];
95     char const* userid   = argv[4];
96     std::string fileid   = genfileid(userid);
97     char const* filepath = argv[5];
98     if (client.upload(appid, userid,
99         fileid.c_str(), filepath) != OK) {
100         client_c::deinit();
101         return -1;
102     }
103     printf("Upload success: %s\n", fileid.c_str());
104 }
105 else
106 // 向存储服务器询问文件大小
107 if (!strcmp(subcmd, "filesize")) {
108     if (argc < 6) {
109         client_c::deinit();
110         usage(cmd);
111         return -1;
112     }
113     char const* appid    = argv[3];
114     char const* userid   = argv[4];
115     char const* fileid   = argv[5];
116     long long  filesize = 0;
117     if (client.filesize(appid, userid, fileid, &filesize) != OK) {
118         client_c::deinit();
119         return -1;
120     }
121     printf("Get filesize success: %lld\n", filesize);
122 }
123 else
124 // 从存储服务器下载文件
125 if (!strcmp(subcmd, "download")) {

```

```

126         if (argc < 8) {
127             client_c::deinit();
128             usage(cmd);
129             return -1;
130         }
131         char const* appid    = argv[3];
132         char const* userid   = argv[4];
133         char const* fileid   = argv[5];
134         long long  offset    = atoll(argv[6]);
135         long long  size      = atoll(argv[7]);
136         char*      filedata  = NULL;
137         long long  filesize  = 0;
138         if (client.download(appid, userid,
139             fileid, offset, size, &filedata, &filesize) != OK) {
140             client_c::deinit();
141             return -1;
142         }
143         printf("Download success: %lld\n", filesize);
144         free(filedata);
145     }
146     else
147         // 删除存储服务器上的文件
148         if (!strcmp(subcmd, "delete")) {
149             if (argc < 6) {
150                 client_c::deinit();
151                 usage(cmd);
152                 return -1;
153             }
154             char const* appid  = argv[3];
155             char const* userid = argv[4];
156             char const* fileid = argv[5];
157             if (client.del(appid, userid, fileid) != OK) {
158                 client_c::deinit();
159                 return -1;
160             }
161             printf("Delete success: %s\n", fileid);
162         }
163     else {
164         client_c::deinit();
165         usage(cmd);
166         return -1;
167     }
168
169     // 终结化客户机
170     client_c::deinit();
171
172     return 0;
173 }

```

~/TNV/src/05_client/Makefile

```

1  PROJ    = ../../bin/client
2  LPRJ    = ../../lib/libclient.a
3  OBJS    = $(patsubst %.cpp, %.o, $(wildcard ../01_common/*.cpp *.cpp))
4  LOBJ    = $(filter-out %main.o, $(OBJS))

```

```
5 CC      = g++
6 LINK     = g++
7 AR        = ar rv
8 RM        = rm -rf
9 CFLAGS    = -c -Wall \
10           -I/usr/include/acl-lib/acl -I/usr/include/acl-lib/acl_cpp \
11           -I../01_common
12 LIBS      = -pthread -lacl_all
13
14 all: $(PROJ) $(LPRJ)
15
16 $(PROJ): $(OBJS)
17     $(LINK) $^ $(LIBS) -o $@
18
19 $(LPRJ): $(LOBJ)
20     $(AR) $@ $^
21
22 .cpp.o:
23     $(CC) $(CFLAGS) $^ -o $@
24
25 clean:
26     $(RM) $(PROJ) $(LPRJ) $(OBJS)
```