# 基于 TUXEDO 的文件传输设计与实现①

## Design and Appliation of Files Transmission Based on TUXEDO

袁宏杰　(石家庄邮电职业技术学院 邮政通信管理系 河北 石家庄 050021)

**摘　要：** 目前基于 TUXEDO 中间件的 C/S 结构的大型应用越来越广泛，在此类大型应用中离不开报表文件的传输，如何利用 TUXEDO 提供的相关技术，实现文件传输是一个很重要的技术点。结合作者使用 TUXEDO 开发的经验，设计并的实现了文件传输，并且支持文件断点续传、压缩、负载均衡功能。

**关键词：** TUXEDO 中间件 FTP 文件传输

## 1　引言

文件传输可以通过多种方式实现，像 FTP、Socket、下载工具等等，本文借助 TUXEDO 中间件产品实现此功能。TUXEDO 是一个基于 C/S 模式的中间件产品，负责网络通讯和负载均衡等主要功能。该产品支持异构的操作系统下稳定运行，意味着此文件传输可以在不同的操作系统下运行。另外可以编译成一个单独的工具独立使用，也可以做为一个文件传输模块嵌入到应用系统中去，具有较强的灵活性和实用性。下面通过一个软件结构图来说明基于 TUXEDO 文件传输实现原理。

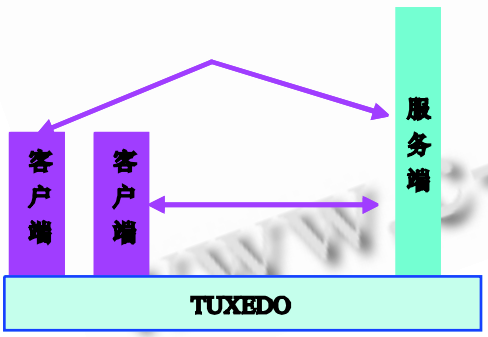


图 1　TUXEDO 文件传输原理图

客户端：表示服务的请求者，可以请求上传文件、或下载文件等。

服务端：表示服务的提供者，满足客户端的请求。

这里的客户端、服务端是一个逻辑概念，所以客户端与服务端可以在同一台计算机上，也可以分开来部署。客户端与服务端可以是异构操作系统[1]。根据客户端数量的多少，可以适当增加服务端 service 的数量来满足请求，实现负载均衡，提高文件传输的适应能力。

TUXEDO 在文件传输时起着网络通讯和信息传递的作用。TUXEDO 为通信提供了客户端和服务端调用接口，以及数据类型格式。文件传输采用 TUXEDO 同步调用方式和 FML 的数据类型来实现文件的上传和下载，这里的上传与下载文件是相对于客户端来说的。

实现原理是：首先客户端发起请求进行文件传输(上传文件还是下载文件，通过标志来体现)，发起请求也就是调用服务端的服务，服务端接收到请求后根据文件传输标志，进行处理，并将处理结果返回给调用者。

## 2　客户端设计与实现

客户端主要完成服务的调用，它可以以一个可执行文件存在，也可以通过简单修改嵌入到应用系统中。下面结合着实现进行描述。

```
if (tpinit ((TPINIT *) NULL) == -1) {return (-1);}
tux_info=(TUX_INFO*)malloc(sizeof (TUX_INFO));
memset (tux_info, 0, sizeof (TUX_INFO));
tux_info->stblock_id = 1;
```

① 收稿时间:2009-05-14

```
if (get_put == 1 || get_put == 3)
{ if (get_put == 1)
  tux_info->opt = TUX_GET;
else
  tux_info->opt = TUX_GET1;
if (tuxcall (tux_info, cltfile, serfile)) { return
-1;}
if (get_put == 1)
{memset (cTmpstr, '\0', sizeof (cTmpstr));
sprintf(cTmpstr,"mv%s/%s%s/%s.Z",(char*)g
etenv("ZHYYDOWNDIR"),cltfile,(char*)  getenv
("ZHYYDOWNDIR"), cltfile);
/*ZHYYDOWNDIR 是一个环境变量指名文件存
放的路径*/
memset (cTmpstr, '\0', sizeof (cTmpstr));
sprintf (cTmpstr, "uncompress %s/%s.Z",
(char*) getenv("ZHYYDOWNDIR"), cltfile);}}
else
{tux_info->opt = TUX_PUT;
if (tuxcall (tux_info, cltfile, serfile)) {return
-1;}
}
```

首先客户端通过调用 tpinit 初始化连接，这个过程是联系服务端，当然在执行该操作时第一步要把服务端的 IP 地址及端口号配置在客户端的 WSNADDR 环境变量中，第二步填充 get_put 标志变量，get_put=1 表示下载不带压缩的文件，get_put=3 表示下载带压缩格式的文件，get_put=2 表示上传文件。将 get_put 赋值给 tux_info 结构中的 opt 变量后执行调用 tuxcall 函数向服务端发起请求，当然要把请求的文件名，放在请求报文中。当服务端响应后客户端分下载和上传两种情况进行处理，下载处理放在服务端来说明，服务端的下载，对应着客户端的文件上传，先来看客户端上传文件：

```
sprintf (filename, "%s/%s", (char*) getenv
("ZHYYDOWNDIR"), cltfile);
fp = fopen (filename, "rb");
for (i = start_block; i < block_num + 1; i++)
{ retrynum = 0;
  if (fseek (fp, (i - 1) * block_size, 0) != 0)
{ fclose (fp); free (filebuf);
  return  CLIENT_ERROR_FSEEK; }
  sendlen = fread (filebuf, 1, block_size,
fp);
    if (!(sendlen == block_size || feof
(fp) != 0))
{fclose (fp);free (filebuf);
  return CLIENT_ERROR_FREAD;}
Fchg32 (sendbuf, TUXDATALEN, 0, (char *)
&sendlen, 0);
Fchg32 (sendbuf, TUXFILEDATA, 0,
    filebuf, (FLDLEN32) sendlen);
Fchg32 (sendbuf, TUXBLKID, 0, (char *) &i,
0);
rval =
tpcall ("TUXFTPFILE", (char *) sendbuf, 0,
  (char **) &sendbuf,&recvlen, 0);
ftime (&time_after);
  time_cost = time_after.time -
time_before.time;
while (rval < 0 && time_cost <= TUX_TIME
&&
    retrynum <= retry_num)
{userlog("Tpcallfailed,will retry several
times, tperrno=%s\n",
  tpstrerror (tperrno));
rval =
tpcall ("TUXFTPFILE", (char *) sendbuf, 0,
  (char **) &sendbuf, &recvlen, 0);
ftime (&time_after);
  time_cost = time_after.time -
time_before.time;
retrynum++;
} }fclose (fp);
```

客户端按事先设定好的文件块大小来向服务发送，并记录下该文件块发送时间，发送次数，发送时间由 time_cost 来控制，重试次数由 retrynum 来存储，当超时或着重试次数达到最大值时，发送终止。从而控制文件的传输，而不会造成死循环死机现象的发生，而且达到了合理利用资源的目的。其中 Fchg32 (sendbuf, TUXDATALEN, 0, (char *) &sendlen, 0);

类似的语句是调用 Tuxedo 客户端提供的接口，

来按变量名读取缓冲区指定数据或者是存放数据到缓冲区，从而完成与服务户端的通信。

## 3 服务端配置

文件传输的服务端，是通过 Tuxedo 的一个配置文件来实现部署的，这个文件一般称为 ubb 文件，它是二进制文件，是由 tmloadcf 命令编译一个有格式要求的文本文件形成的。首先将文件传输服务的名字，写到配置文件(文本文件)中，然后对文本文件编译，形成配置文件。下面简要看一下配置文件(文本文件)中的配置。

```
*GROUPS
GROUP1 LMID=simple GRPNO=1
*SERVERS
FTPSVR SVRGRP=GROUP1 SRVID=900
MIN=3 MAX=3
```

服务的名字是 FTPSVR SVRGRP 指定组号的变量 MIN 是最小启动几个 SERVER，MAX 最大启动几个 SERVER，可以根据这两个变量来调节服务端的处理性能[2]

当设置好配置文件以后，就可以用 tmboot 来启动服务了。

## 4 服务端设计与实现

服务端在配置文件的控制下启动，启动成功后在特定端口上监听来自客户端的请求，下面以响应客户端上传文件的实现来分析，对于客户端上传的压缩文件，服务端首先要进行解压缩，然后将数据写到对应的文件中去，下面扼要看一下其实现方法：

```
TUXFTPFILE (rqst)
TPSVCINFO *rqst;
{Fget32 (tuxbuf, TUXCHOICE, 0, (char
*)&choice, 0);
    sprintf (filename, "%s/%s", getenv
("HOME"), serfile);
    fp = fopen (filename, "rb");
    if (fp == NULL) fp = fopen (serfile, "rb");}
    strcpy (filename, serfile);
    fseek (fp, 0, SEEK_END) != 0;
    reallen = ftell (fp); fclose (fp);
    if (reallen < 2){ return 3;}
    if (block_id == reallen / block_size + 1) {
      memset (cTmpstr, '\0', sizeof (cTmpstr));
      sprintf (cTmpstr, "mv -f %s/%s %s/%s.Z",
                getenv ("HOME"), serfile,
                getenv ("HOME"), serfile);
      memset (cTmpstr, '\0', sizeof (cTmpstr));
       sprintf(cTmpstr, "uncompress %s/%s.Z",
                getenv ("HOME"), serfile); }
   ret=fwrite(filebuf, 1, len, fp); MiniDelay(10);
   tpreturn (TPSUCCESS, 0, rqst->data, 0L, 0);}
```

其中 TUXFTPFILE (rqst)为 service 名字，参数必须使用 TPSVCINFO *rqst 这样一种形式，这是 TUXEDO 服务端要求的，rqst 为数据缓冲区。

系统为了保证传输的准确可靠，增加了微型延时模块，延时数量级在毫秒。延时模块的实现如下：

```
Void  MiniDelay (interval)
unsigned short int interval;
/*需要延时的毫秒数 */
{ struct timeb start_timeb, cur_timeb;
ftime (&start_timeb);  /* 开始计时 */
do {ftime (&cur_timeb);
if((cur_timeb.time-start_timeb.time) * 1000
+
cur_timeb.millitm - start_timeb.millitm >=
interval)
break;} while (1); return;}
```

应用证明该延时模块能起到很好的调节作用，而且可以根据系统所处的网络情况进行修改，保证文件传输的稳定顺畅运行。

## 5 结论

本文借助 TUXEDO 提供的通信接口，完成了文件传输的功能，该功能在河北电子化支局系统和速递局邮政礼仪系统中得到的应用，具有很强的使用价值和推广价值，特别是基于 TUXEDO 大型的应用系统中值得参考。

### 参考文献

1 徐春金.TUXEDO 中间件开发与配置.北京:中国电力出版社, 2003.65－67.
2 经乾.TUXEDO 系统经典.北京:电子工业出版社, 2007.78－79.