Meteorological-Data-Management-Center

By：jjyaoao

第一板块-如何开发永不停机的服务程序

开发目标

1. 生成测试数据
2. 服务程序的调度
3. 守护进程的实现
4. 两个常见小工具

生成测试数据

①：搭建程序框架

~~~c++
/*
*     project name: crtsurfdata1.cpp  用于生成全国气象站点观测的分钟数据
*     author: jjyaoao
*/
~~~

运行的参数、说明文档、运行日志

/tmp/idc/surfdata/SURF_ZH_20220514021227_4976.xml

/tmp/idc/surfdata/SURF_ZH_20220416123000_4973.xml

/tmp/idc/surfdata/SURF_ZH_20220514075211_11385.xml

②：加载站点参数

1. st_stcode 结构体，存放站点
2. 创建 st_stcode 向量实例 vstcode
3. LoadSTCode 方法来加入
4. 使用 CFile 类--自行封装--进行文件读入读出
5. m_vCmdstr--自行封装--拆分字段

③：模拟观测数据

1. st_surfdata 结构体，实现每个站点的分钟观测
2. 创造 st_surfdata 向量实例 vsurfdata
3. CrtSurfData 函数，实现分钟观测
    1. 播随机数种子
    2. 获取当前时间，作为观测时间
    3. 遍历站点容器--vstcode
    4. 随机数填充分钟观测数据的结构体
    5. 将结构体放入容器 vsurfdata

④：把站点观测数据写入文件

1. CruSurfFile 函数实现写入每分钟的观测数据

    1. 生成临时文件名--以

```
// 拼接生成数据的文件名，例如：/tmp/surfdata/SURF_ZH_20210629092200_2254.csv
char strFileName[301];
snprint(strFileName,"%s/SURF_ZH_%s_%d.%s",outpath,strddatetime,getpid(),datafmt);
```

这里 outpath 是绝对路径，strddatetime 是当前时间，getpid 是进程号，datafmt 是文件格式，进程号主要是为了保证临时文件名不重复(getpid())，这里不加也可以

2. 打开文件
3. 写入第一行标题//csv 才需要
4. 遍历存放观测数据的容器 vsurfdata，并且，对临时文件进行写入操作
5. 关闭文件

支持 csv、xml、json

服务程序调度

目标：
周期性的启动后台服务程序。

常驻内存中的服务程序异常中止，在短时间内重启。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc,char *argv[]) {
  if (argc<3) {

    printf("Using:./procctl timetvl program argv ...\n");
    printf("Example:/project/tools1/bin/procctl  5  /usr/bin/tar  zcvf
/tmp/tmp.tgz /usr/include\n\n");

    printf("本程序是服务程序的调度程序,周期性启动服务程序或 shell 脚本。\n");
    printf("timetvl 运行周期，单位：秒。被调度的程序运行结束后，在 timetvl
秒后会被 procctl 重新启动。\n");
    printf("program 被调度的程序名，必须使用全路径。\n");
    printf("argvs   被调度的程序的参数。\n");
    printf("注意，本程序不会被 kill 杀死，但可以用 kill -9 强行杀死。\n\n\n");

    return -1;//注意，这里不返回，到时候会继续往下执行
  }

// 关闭信号和 IO，本程序不希望被打扰。
  for(int i = 0; i < 64; i++){
         signal(i, SIG_IGN);
         close(i);
  }

// 生成子进程，父进程退出，让程序运行在后台，由系统 1 号进程托管。
  if(fork() != 0) exit(0);


// 启用 SIGCHLD 信号，让父进程可以 wait 子进程退出的状态。
  signal(SIGCHLD, SIG_DFL);//DFL 为默认执行信号
  char *pargv[argc];
  for(int i = 2; i < argc; i++)
         pargv[i-2] = argv[i];

  pargv[argc-2] = NULL;//这两步是将 argv[2]之后的命令存入 pargv;

  while(true){
         if(fork() == 0){//再度生成子进程?
```

```
                execv(argv[2], pargv);//以第三个参数为路径 PS:仔细看 main
函数传参就懂得了
                exit(0);//execv 成功找到路径，就不执行，不然就退出进程了
        }else{
                int status;
                wait(&status);//第二个参数就是睡眠时间
                sleep(atoi(argv[1]));//睡眠，atoi 为转化为整数
              }
  }
}
```

守护进程实现

目标：
服务程序由调度程序启动（procctl）。

如果服务程序死机（挂起），守护进程将终止它。

服务程序被终止后，调度程序（procctl）将重新启动它。

实现代码: g++ -g -o book1 book1.cpp -I/project/public /project/public/_public.cpp

```
#include "_public.h"

// 程序运行日志
CLogFile logfile;

int main(int argc, char* argv[]){
  // 程序的帮助。
  if (argc != 2)
  {
    printf("\n");
    printf("Using:./checkproc logfilename\n");

    printf("Example:/project/tools1/bin/procctl 10 /project/tools1/bin/checkproc
/tmp/log/checkproc.log\n\n");

    printf("本程序用于检查后台服务程序是否超时，如果已超时，就终止它。\n");
    printf("注意：\n");
```

```
    printf("   1）本程序由 procctl 启动，运行周期建议为 10 秒。\n");
    printf("   2）为了避免被普通用户误杀，本程序应该用 root 用户启动。\n");
    printf("   3）如果要停止本程序，只能用 killall -9 终止。\n\n\n");

    return 0;
  }

  // 忽略全部的信号和 IO，不希望程序被干扰。
  // for(int i = 1; i <= 64; i++) signal(i, SIG_IGN);
  CloseIOAndSignal(true);//该函数，缺省 false（只关信号不关 IO，用 true 就可以全关

    // 打开日志文件
    if(logfile.Open(argv[1],"a+") == false){
        printf("logfile.Open(%S) failed.\n", argv[1]);
        return -1;
    }
    // 创建/获取共享内存，键值为 SHMKEYP，大小为 MAXNUMP 个 st_procinfo 结构体的大小、我们自己实现的代码是 profo(吴哥的是 procinfo)
    // 从心跳机制抄过来的(删改了一些)，用的是用一个共享内存
    int m_shmid = 0;
    if((m_shmid = shmget((key_t)SHMKEYP, MAXNUMP*sizeof(struct st_procinfo), 0666|IPC_CREAT)) == -1){
        logfile.Write("创建/获取共享内存(%x)失败。\n", SHMKEYP);
        return false;
    }
    // 将共享内存连接到当前进程的地址空间
    struct st_procinfo *shm = (struct st_procinfo *)shmat(m_shmid, 0, 0);

    // 遍历共享内存中全部的进程心跳记录
    for(int i = 0; i < MAXNUMP; i++){
        // 如果记录的 pid == 0， 表示空记录， continue;
        if(shm[i].pid == 0) continue;

        // 如果记录的 pid != 0，表示是服务程序的心跳记录,程序稳定了就不需要写了，是用于调试
        //logfile.Write("i = %d, pid = %d, pname = %s, timeout = %d, atime = %d\n",\
        //              i, shm[i].pid, shm[i].pname, shm[i].timeout, shm[i].atime);

        // 向进程发送信号 0(不管是否超时)，判断它是否还存在，如果不存在，从共享内存中删除该记录，continue;
        int iret = kill(shm[i].pid, 0);//kill，进程不存在会返回-1,进程存在，返回 0。
        if(iret == -1){
            logfile.Write("进程 pid = %d(%S)已经不存在.\n", (shm+i) -> pid, (shm + i) -> pname);
```

```
                memset(shm+i, 0, sizeof(struct st_procinfo));//把结构体清零，细节
                continue;
            }

            time_t now = time(0);    //取当前时间
            // 如果未超时
            if(now - shm[i].atime < shm[i].timeout) continue;

            // 如果已经超时
            logfile.Write("进程 pid = %d(%S)已经超时.\n", (shm+i) -> pid, (shm + i) -> pname);

            // 发送信号 15，尝试正常终止进程
            kill(shm[i].pid, 15);// 发送信号 15，尝试正常终止进程

            // 每隔 1 秒判断一次进程是否存在，累计 5 秒，一般来说，5 秒的时间足够让进程退出
            for(int j = 0; j < 5; j++){
                sleep(1);
                iret = kill(shm[i].pid, 0); //向进程发送信号 0，判断它是否还存在
                if(iret == -1) break;       //进程已经退出
            }

            // 如果进程仍存在，发送信号 9，强行终止它
            if(iret == -1){
                logfile.Write("进程 pid = %d(%S)已经正常终止.\n", (shm+i) -> pid, (shm + i) -> pname);
            }else{
                kill(shm[i].pid, 9);  //如果进程仍然存在，就发送信号 9，强行终止它
                logfile.Write("进程 pid = %d(%s) 已经强制终止。\n", (shm+i)->pid, (shm+i)->pname);
            }

            // 从共享内存中删除已超时进程的心跳记录
            memset(shm+i, 0, sizeof(struct st_procinfo));      // 从共享内存中删除该记录
        }
        // 把共享内存从当前进程中分离
        shmdt(shm);
        return 0;
}
```

完善生成测试数据程序

目标：

1. 增加生成历史数据文件的功能，为压缩文件和清理文件模块准备历史数据文件。
2. 增加信号处理函数，处理 2 和 15 的信号
3. 解决调用 exit 函数退出时局部对象没有调用析构函数的问题
4. 把心跳信息写入共享内存,（虽然说运行很短，根本不需要使用心跳，但我们现在手里只有他，所以就拿他来玩呗)

```
/*
*     project name: crtsurfdata.cpp  用于生成全国气象站点观测的分钟数据
*     author: jjyaoao
*/

#include "_public.h"
CPActive PActive;  //进程心跳

// 全国气象站点参数结构体
struct st_stcode{
      char provname[31]; //省
      char obtid[11];//站号
      char obtname[31]; //站名
      double lat; //维度
      double lon; //经度
      double height; //海拔高度
};
// 存放全国气象站点参数的容器。
vector<struct st_stcode> vstcode;

// 把站点参数文件加载到 vstcode 容器中。
bool LoadSTCode(const char *inifile);

// 全国气象站点分钟观测数据结构
struct st_surfdata
{
  char obtid[11];          // 站点代码。
  char ddatetime[21];   // 数据时间：格式 yyyymmddhh24miss
  int   t;                    // 气温：单位，0.1 摄氏度。
  int   p;                    // 气压：0.1 百帕。
  int   u;                    // 相对湿度，0-100 之间的值。
  int   wd;                   // 风向，0-360 之间的值。
  int   wf;                   // 风速：单位 0.1m/s
  int   r;                    // 降雨量：0.1mm。
  int   vis;                  // 能见度：0.1 米。
```

```
};

// 存放全国气象站点分钟观测数据的容器
vector<struct st_surfdata> vsurfdata;

//观测数据的时间
char strddatetime[21];
// 模拟生成全国气象站点分钟观测数据，存放在 vsurfdata 容器中
void CrtsurfData();

CFile File;//各种文件操作，封装为 CFile

// 把容器 vsurfdata 中的全国气象站点分钟观测数据写入文件
bool CrtSurFile(const char *outpath, const char *datafmt);

//CLogFile logfile(10);// 指定日志文件大小为 10 兆
CLogFile logfile;  // 日志类

void EXIT(int sig); // 程序退出和信号 2、15 的处理函数

int main(int argc, char *argv[]){
    if((argc != 5) && (argc != 6)){//若传入 argc 为 6，则为指定历史时间,不指定即为当前时间
        printf("Using:./crtsurfdata inifile outpath logfile datafmt [datetime]\n");//[]意思即为你可以填，你也可以不填
        printf("Example:/project/idc1/bin/crtsurfdata /project/idc1/ini/stcode.ini /tmp/idc/surfdata /log/idc/crtsurfdata.log xml,json,csv\n\n");
        printf("Example:/project/idc1/bin/crtsurfdata /project/idc1/ini/stcode.ini /tmp/idc/surfdata /log/idc/crtsurfdata.log xml,json,csv 20220416123000\n\n");
        printf("Example:/project/tools1/bin/procctl 60 /project/idc1/bin/crtsurfdata /project/idc1/ini/stcode.ini /tmp/idc/surfdata /log/idc/crtsurfdata.log xml,json,csv\n\n");


        printf("inifile 全国气象站点参数文件名。\n");
        printf("outpath 全国气象战点数据文件存放的目录。\n");
        printf("logfile 本程序运行的日志文件名。\n");
        printf("datafmt 生成数据文件的格式，支持 xml、json 和 csv 三种格式，中间用逗号分隔。\n");
        printf("datetime 这是一个可选参数，表示生成指定时间的数据和文件\n\n\n");

        return -1;
    }

    // 关闭全部的信号和输入输出
```

```
    // 设置信号，在 shell 状态下可用"kill + 进程号" 正常终止进程
    // 但别用"kill -9 + 进程号"强制终止
    CloseIOAndSignal(true);
    signal(SIGINT, EXIT); signal(SIGTERM, EXIT);//可用数字也可以名称，为了标准这里使用名称（2, 15）

    if (logfile.Open(argv[3], "a+", false) == false) {//打开日志文件失败，程序 退出，没必要继续
        printf("logfile.Open(%s) failed.\n", argv[3]);
        return -1;
    }

    logfile.Write("crtsurfdata 开始运行. \n");

    PActive.AddPInfo(20, "crtsurfdata");//20 秒写入一次，因为太短了，程序，所以心跳的时间就不用更新了

    // 把站点参数文件加载到 vstcode 容器中
    if (LoadSTCode(argv[1]) == false) return -1;

    // 获取当前时间，当作观测时间。
    memset(strddatetime,0,sizeof(strddatetime));
    if(argc == 5)
        LocalTime(strddatetime,"yyyymmddhh24miss");
    else
        STRCPY(strddatetime, sizeof(strddatetime), argv[5]);

    // 模拟生成全国气象站点分钟观测数据，存放在 vsurfdata 容器中
    CrtsurfData();
    if(strstr(argv[4], "xml")!=0) CrtSurFile(argv[2], "xml");
    if(strstr(argv[4], "json")!=0) CrtSurFile(argv[2], "json");
    if(strstr(argv[4], "csv")!=0) CrtSurFile(argv[2], "csv");

    logfile.Write("crtsurfdata 运行结束。 \n");

        return 0;
}

// 把站点参数文件加载到 vstcode 容器中。
bool LoadSTCode(const char *inifile){
    //打开站点参数文件。
    if(File.Open(inifile, "r") == false){
        logfile.Write("File.Open(%s) failed.\n", inifile);
        return false;
```

```
    }
    char strBuffer[301];
    CCmdStr CmdStr;
    struct st_stcode stcode;
    while (true){
        //从站点参数文件中读取一行，如果读取完，跳出循环
        //通常情况我们需要初始化字符串，不然可能会有 bug，memset strBuffer 已经再
Fgets 里面做了
        if(File.Fgets(strBuffer, 300, true) == false) break;
        // 把读到的一行拆分,第三个参数为选择是否删除多余空格
        CmdStr.SplitToCmd(strBuffer, ",", true);
        //扔掉无效行
        if(CmdStr.CmdCount() != 6) continue;
        // 把站点参数的每个数据项保存到站点参数结构体中
        CmdStr.GetValue(0, stcode.provname, 30); // 省
        CmdStr.GetValue(1, stcode.obtid, 10);    // 站号
        CmdStr.GetValue(2, stcode.obtname, 30);   // 站名
        CmdStr.GetValue(3, &stcode.lat);   // 纬度
        CmdStr.GetValue(4, &stcode.lon);  // 经度
        CmdStr.GetValue(5, &stcode.height);     // 海拔高度
        // 把站点参数结构体放入站点参数容器
        vstcode.push_back(stcode);
    }
    /*
    for (int i = 0; i < vstcode.size(); i++){
        logfile.Write("provnmae=%s, obtid=%s, obtname=%s\n",\
                vstcode[i].provname, vstcode[i].obtid, vstcode[i].obtname);
    }            测试程序
    */
    // 关闭文件,在析构函数里面已经自动关闭了（makefile）
    return true;
}

// 模拟生成全国气象站点分钟观测数据，存放在 vsurfdata 容器中
void CrtsurfData(){
 // 播随机数种子。
   srand(time(0));


   struct st_surfdata stsurfdata;

   // 遍历气象站点参数的 vstcode 容器。
   for (int i=0;i<vstcode.size();i++)
   {
```

```
      memset(&stsurfdata,0,sizeof(struct st_surfdata));

      // 用随机数填充分钟观测数据的结构体。i
      // 随机函数，例如下面第一个 rand()%351,就是取 0-350 的随机函数
      strncpy(stsurfdata.obtid,vstcode[i].obtid,10); // 站点代码。
      strncpy(stsurfdata.ddatetime,strddatetime,14);   // 数据时间：格式 yyyymmddhh24miss
      stsurfdata.t=rand()%351;           // 气温：单位，0.1 摄氏度
      stsurfdata.p=rand()%265+10000; // 气压：0.1 百帕
      stsurfdata.u=rand()%100+1;        // 相对湿度，0-100 之间的值。
      stsurfdata.wd=rand()%360;          // 风向，0-360 之间的值。
      stsurfdata.wf=rand()%150;          // 风速：单位 0.1m/s
      stsurfdata.r=rand()%16;            // 降雨量：0.1mm
      stsurfdata.vis=rand()%5001+100000;   // 能见度：0.1 米

      // 把观测数据的结构体放入 vsurfdata 容器。
      vsurfdata.push_back(stsurfdata);
    }
}

// 把容器 vsurfdata 中的全国气象站点分钟观测数据写入文件。
bool CrtSurFile(const char *outpath, const char *datafmt){
      CFile File;
      // 拼接生成数据的文件名，例如：/tmp/idc/surfdata/SURF_ZH_20210629092200_2254.csv
      char strFileName[301];
      // 在文件名中加入进程编号，这是为了保证临时文件名不重复(getpid())，这里不加也可以
      sprintf(strFileName, "%s/SURF_ZH_%s_%d.%s", outpath, strddatetime, getpid(), datafmt);
      // 打开文件。
      if(File.OpenForRename(strFileName, "w") == false){//一般是没有磁盘空间或者权限不足
            logfile.Write("File.OpenForRename(%s) failed.\n", strFileName);
            return false;
      }
      // 写入第一行标题
      if(strcmp(datafmt, "csv") == 0) File.Fprintf("站点代, 数据时间, 气温, 气压, 相对湿度, 风向, 风速, 降雨量, 能见度\n");
      // 遍历存放观测数据的 vsurfdata 容器。
      for(int i = 0; i < vsurfdata.size(); i++){
            // 写入一条记录
            if(strcmp(datafmt, "csv") == 0){
                  File.Fprintf("%s, %s, %.1f, %.1f, %d, %d, %.1f, %.1f, %.1f\n",\
                                    vsurfdata[i].obtid, vsurfdata[i].ddatetime, vsurfdata[i].t / 10.0,
```

```
vsurfdata[i].p / 10.0, \
                        vsurfdata[i].u, vsurfdata[i].wd, vsurfdata[i].wf / 10.0, vsurfdata[i].r /
10.0, vsurfdata[i].vis / 10.0);
            }

            if (strcmp(datafmt, "xml") == 0)

File.Fprintf("<obtid>%s</obtid><ddatetime>%s</ddatetime><t>%.1f</t><p>%.1f</p>"\

"<u>%d</u><wd>%d</wd><wf>%.1f</wf><r>%.1f</r><vis>%.1f</vis><endl/>\n",\
              vsurfdata[i].obtid,vsurfdata[i].ddatetime,vsurfdata[i].t/10.0,vsurfdata[i].p/10.0,\

vsurfdata[i].u,vsurfdata[i].wd,vsurfdata[i].wf/10.0,vsurfdata[i].r/10.0,vsurfdata[i].vis/10.0);

            if (strcmp(datafmt, "json") == 0)
                  {

File.Fprintf("{\"obtid\":\"%s\",\"ddatetime\":\"%s\",\"t\":\"%.1f\",\"p\":\"%.1f\","\

"\"u\":\"%d\",\"wd\":\"%d\",\"wf\":\"%.1f\",\"r\":\"%.1f\",\"vis\":\"%.1f\"}",\
                    vsurfdata[i].obtid,vsurfdata[i].ddatetime,vsurfdata[i].t/10.0,vsurfdata[i].p/10.0,\

vsurfdata[i].u,vsurfdata[i].wd,vsurfdata[i].wf/10.0,vsurfdata[i].r/10.0,vsurfdata[i].vis/10.0);
                    if (i<vsurfdata.size()-1) File.Fprintf(",\n");
                    else      File.Fprintf("\n");
                  }

        if (strcmp(datafmt,"xml")==0) File.Fprintf("</data>\n");
        if (strcmp(datafmt,"json")==0) File.Fprintf("]}\n");
    }
      //sleep(10); //单元测试
      // 关闭文件
      File.CloseAndRename();

      UTime(strFileName, strddatetime); // 修改文件的时间属性

      logfile.Write("生成数据文件%s 成功，数据时间%s，记录数%d.\n", strFileName,
strddatetime, vsurfdata.size());
      return true;
}


// 程序退出和信号 2、15 的处理函数
void EXIT(int sig){
```

```
    logfile.Write("程序退出，sig = %d\n\n", sig);

    exit(0);
}
```

开发常用小工具

目标：
开发压缩文件模块

开发清理历史数据文件模块

压缩文件实现：

```
#include "_public.h"

// 程序退出和信号 2、15 的处理函数。
void EXIT(int sig);

int main(int argc,char *argv[]){
        // 程序的帮助
  if (argc != 4)
  {
    printf("\n");//pathname:扫描的目录 matchstr:需要处理这个目录下的什么文件 timeout:时间点，在此之前的会被压缩
    printf("Using:/project/tools1/bin/gzipfiles pathname matchstr timeout\n\n");

    printf("Example:/project/tools1/bin/gzipfiles /log/idc \"*.log.20*\" 0.02\n");
    printf("         /project/tools1/bin/gzipfiles /tmp/idc/surfdata \"*.xml,*.json\" 0.01\n");//下面两行表示需要由调度程序启动
    printf("         /project/tools1/bin/procctl 300 /project/tools1/bin/gzipfiles /log/idc \"*.log.20*\" 0.02\n");
    printf("         /project/tools1/bin/procctl 300 /project/tools1/bin/gzipfiles /tmp/idc/surfdata \"*.xml,*.json\" 0.01\n\n");

    printf("这是一个工具程序，用于压缩历史的数据文件或日志文件。\n");
    printf("本程序把 pathname 目录及子目录中 timeout 天之前的匹配 matchstr 文件全部压缩，timeout 可以是小数。\n");
    printf("本程序不写日志文件，也不会在控制台输出任何信息。\n");
    printf("本程序调用/usr/bin/gzip 命令压缩文件。\n\n\n");
```

```
    return -1;
  }
      // 关闭全部的信号和输入输出
      // 设置信号,在 shell 状态下可用 "kill + 进程号" 正常终止些进程。
      // 但请不要用 "kill -9 +进程号" 强行终止。
     //   CloseIOAndSignal(true); //一般来说开发阶段把这行注释掉，为了方便调试
    signal(SIGINT, EXIT);    signal(SIGTERM, EXIT);

      // 获取文件超时的时间点（人为定义）
      char strTimeOut[21];// 0-用来转化为负数
      LocalTime(strTimeOut, "yyyy-mm-dd hh24:mi:ss", 0-(int)(atof(argv[3])*24*60*60));
      //这里一定要是这种格式的时间（一共就封装了两种缺省，要改自己加源文件）

      CDir Dir;
      // 打开目录，CDir.OpenDir()
    if (Dir.OpenDir(argv[1], argv[2], 10000, true) == false){
              printf("Dir.OpenDir(%s) failed.\n", argv[1]);
              return -1;
    }
      // 遍历目录中的文件名


      char strCmd[1024];    // 存放 gzip 压缩文件的命令
      while(true){

          // 得到一个文件的信息，CDir.ReadDir()
          if(Dir.ReadDir() == false) break;
          // 与超时的时间点比较，如果更早，就需要压缩
          // matchstr 用于判断一个字符串和另外一个字符串是否匹配，为自己封装

           if((strcmp(Dir.m_ModifyTime, strTimeOut) < 0) && (MatchStr(Dir.m_FileName,
"*.gz") == false)){
          // 压缩文件，调用操作系统的 gzip 命令
          // 可以使用 execl execv 这里我们介绍新的命令 system
          // 大写的 SNPRINTF 函数和小写的 sprintf 功能是一样的，这样是封装成安全
的

        SNPRINTF(strCmd,  sizeof(strCmd),1000,  "/usr/bin/gzip  -f  %s  1>/dev/null
2>/dev/null", Dir.m_FullFileName);
              if (system(strCmd) == 0)
                    printf("gzip %s ok.\n", Dir.m_FullFileName);
              else
                    printf("gzip %s failed.\n", Dir.m_FullFileName);
```

```
                }
        }
    return 0;
}

void EXIT(int sig){
  printf("程序退出，sig=%d\n\n", sig);

  exit(0);
}
```

清理历史数据文件模块:

```
#include "_public.h"

// 程序退出和信号 2、15 的处理函数。
void EXIT(int sig);

int main(int argc,char *argv[]){
        // 程序的帮助
  if (argc != 4)
  {
    printf("\n");//pathname:扫描的目录 matchstr:需要处理这个目录下的什么文件 timeout:时间点，在此之前的会被压缩
    printf("Using:/project/tools1/bin/deletefiles /gpathname matchstr timeout\n\n");

    printf("Example:/project/tools1/bin/deletefiles /log/idc \"*.log.20*\" 0.02\n");
    printf("          /project/tools1/bin/deletefiles /tmp/idc/surfdata \"*.xml,*.json\" 0.01\n");//下面两行表示需要由调度程序启动
    printf("          /project/tools1/bin/procctl 300 /project/tools1/bin/deletefiles /log/idc \"*.log.20*\" 0.02\n");
    printf("          /project/tools1/bin/procctl 300 /project/tools1/bin/deletefiles /tmp/idc/surfdata \"*.xml,*.json\" 0.01\n\n");

    printf("这是一个工具程序，用于删除历史的数据文件或日志文件。\n");
    printf("本程序把 pathname 目录及子目录中 timeout 天之前的匹配 matchstr 文件全部删除，timeout 可以是小数。\n");
    printf("本程序不写日志文件，也不会在控制台输出任何信息。\n\n\n");

    return -1;
  }
        // 关闭全部的信号和输入输出
        // 设置信号,在 shell 状态下可用 "kill + 进程号" 正常终止些进程。
        // 但请不要用 "kill -9 +进程号" 强行终止。
         CloseIOAndSignal(true); //一般来说开发阶段把这行注释掉，为了方便调试
```

```
    signal(SIGINT, EXIT);    signal(SIGTERM, EXIT);

      // 获取文件超时的时间点（人为定义）
      char strTimeOut[21];// 0-用来转化为负数
      LocalTime(strTimeOut, "yyyy-mm-dd hh24:mi:ss", 0-(int)(atof(argv[3])*24*60*60));
      //这里一定要是这种格式的时间（一共就封装了两种缺省，要改自己加源文件）

      CDir Dir;
      // 打开目录，CDir.OpenDir()
    if (Dir.OpenDir(argv[1], argv[2], 10000, true) == false){
            printf("Dir.OpenDir(%s) failed.\n", argv[1]);
            return -1;
    }
      // 遍历目录中的文件名


      char strCmd[1024];    // 存放 gzip 压缩文件的命令
      while(true){

          // 得到一个文件的信息，CDir.ReadDir()
          if(Dir.ReadDir() == false) break;
          // 与超时的时间点比较，如果更早，就需要删除

          if(strcmp(Dir.m_ModifyTime, strTimeOut) < 0){
            if (REMOVE(Dir.m_FullFileName) == true)
                printf("REMOVE %s ok.\n", Dir.m_FullFileName);
            else
                printf("REMOVE %s failed.\n", Dir.m_FullFileName);
             }
      }
    return 0;
}

void EXIT(int sig){
  printf("程序退出，sig=%d\n\n", sig);

  exit(0);
}
```

第二板块-基于 ftp 协议的文件传输系统

开发目标

1. ftp 客户端封装
2. 文件下载功能实现
3. 文件上传功能实现

ftp 客户端封装

基本连接

在这里我采用的是，用 root 账户来连接 jjyaoao 账户，模拟远程访问的过程

https://github.com/codebrainz/ftplib

将上面的 c 代码进一步封装成 cpp 的库_ftp.h，_ftp.cpp

~~~shell
g++ -g -o ftpclient ftpclient.cpp /project/public/_ftp.cpp /project/public/_public.cpp -I/project/public -L/project/public -lftp -lm -lc
~~~

```
#include "_ftp.h"

Cftp ftp;

int main(){
    if(ftp.login("192.168.211.130:21", "jjyaoao", "gh") == false){
        printf("ftp.login(192.168.211.130:21) failed.\n");
        return -1;
    }
    printf("ftp.login(191.168.211.130:21) ok.\n");

    if(ftp.mtime("/project/public/socket/demo01.cpp") == false){
        printf("ftp.mtime(/project/public/socket/demo01.cpp) failed.\n");
        return -1;
    }
    printf("ftp.mtime(/project/public/socket/demo01.cpp) ok, mtime = %d.\n", ftp.m_mtime);
```

```
        if(ftp.size("/project/public/socket/demo01.cpp") == false){
                printf("ftp.size(/project/public/socket/demo01.cpp) failed.\n");
                return -1;
        }
        printf("ftp.size(/project/public/socket/demo01.cpp) ok, size = %d.\n", ftp.m_size);

    if(ftp.nlist("/project/public/socket", "/aaa/bbb.lst") == false){
        printf("ftp.nlist(/project/public/socket) failed.\n");
        return -1;
    }
    printf("ftp.nlist(/project/public/socket) ok.\n");

    ftp.logout();

    return 0;
}
```

文件下载功能实现

目标:
开发通用的文件下载模块，从 ftp 服务器下载文件

打开日志文件

虽然和别的之前的也一样，不过这里我忍不住提一句，由于水平上升了，才有机会在这个地方咬文嚼字。

a+，w+，r+到底是什么意思，知道是文件的读写方式，不过今天去了解了更多

- r：以只读的方式打开文本文件，文件必须存在；

- w：以只写的方式打开文本文件，文件**若存在则清空**文件内容**从**文件**头部**开始写，若不存在则根据文件名创建新文件并只写打开；

- a：以只写的方式打开文本文件，文件若存在则从**文件尾部以追加**的方式开始写，文件**原来存在的内容不会清除**（**除了**文件尾标志**EOF**），若不存在则根据文件名创建新文件并只写打开；

- r+：以可读写的方式打开文本文件，文件必须存在；

- w+：以可读写的方式打开文本文件，其他与 w 一样；

- a+：以可读写的方式打开文本文件，其他与 a 一样；

- 若打开二进制文件，可在后面加个 b 注明，其他一样，如 rb，r+b（或 rb+）。

解析 xml

使用 xml，就需要先解析他，开发框架中，有解析的函数

```
///////////////////////////////////// ////////////////////////////////////
// 解析xml格式字符串的函数族。
// xml格式的字符串的内容如下：
// <filename>/tmp/_public.h</filename><mtime>2020-01-01 12:20:35</mtime><size>18348</size>
// <filename>/tmp/_public.cpp</filename><mtime>2020-01-01 10:10:15</mtime><size>50945</size>
// xmlbuffer：待解析的xml格式字符串。
// fieldname：字段的标签名。
// value：传入变量的地址，用于存放字段内容，支持bool、int、insigned int、long、unsigned long、double和char[]类型。
// 注意，当value参数的数据类型为char []时，必须保证value数组的内存足够，否则可能发生内存溢出的问题，也可以用ilen参数限定获取字段内容的长度，ilen的缺省值为0，表示不限长度。
// 返回值：true-成功；如果fieldname参数指定的标签名不存在，返回失败。
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,char *value,const int ilen=0);
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,bool *value);
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,int  *value);
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,unsigned int *value);
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,long *value);
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,unsigned long *value);
bool GetXMLBuffer(const char *xmlbuffer,const char *fieldname,double *value);
///////////////////////////////////// ////////////////////////////////////

///////////////////////////////////// ////////////////////////////////////
```

如果第三个参数是字符串，可以用**第四个参数**指定字符串的长度，**缺省为 0**，表示**不限长度**，不限长度就一定要确保 value 数组空间足够大，否则发生内存溢出的问题

```
// 把 xml 解析到参数 starg 结构中。
bool _xmltoarg(char *strxmlbuffer)
{
  memset(&starg,0,sizeof(struct st_arg));

  GetXMLBuffer(strxmlbuffer,"host",starg.host,30);     // 远程服务器的 IP 和端口。
  if (strlen(starg.host)==0)
  { logfile.Write("host is null.\n");    return false; }

  GetXMLBuffer(strxmlbuffer,"mode",&starg.mode);     // 传输模式，1-被动模式，2-主动模式，缺省采用被动模式。
  if (starg.mode!=2)    starg.mode=1;

  GetXMLBuffer(strxmlbuffer,"username",starg.username,30);     // 远程服务器 ftp 的用户名。
  if (strlen(starg.username)==0)
  { logfile.Write("username is null.\n");    return false; }

  GetXMLBuffer(strxmlbuffer,"password",starg.password,30);     // 远程服务器 ftp 的密码。
  if (strlen(starg.password)==0)
  { logfile.Write("password is null.\n");    return false; }

  GetXMLBuffer(strxmlbuffer,"remotepath",starg.remotepath,300);     // 远程服务器存放文件的目录。
  if (strlen(starg.remotepath)==0)
  { logfile.Write("remotepath is null.\n");    return false; }
```

```
    GetXMLBuffer(strxmlbuffer,"localpath",starg.localpath,300);     // 本地文件存放的目录。
    if (strlen(starg.localpath)==0)
    { logfile.Write("localpath is null.\n");    return false; }

    GetXMLBuffer(strxmlbuffer,"matchname",starg.matchname,100);     // 待下载文件匹配的规则。
    if (strlen(starg.matchname)==0)
    { logfile.Write("matchname is null.\n");    return false; }

    return true;
}
```

更多的需求:
增量下载文件，每次只下载新增的和修改过的文件。

下载文件后，删除 ftp 服务器上的文件。

下载文件后，把 ftp 服务器上的文件移动到备份目录。

删除/备份文件

删除文件十分简单，只需要调用 ftp 的一个方法，下载文件并备份就稍微复杂一点点,需要额外多一个备份目录，并使用 strremotefilenamebak 来暂存新的目录名加文件名，然后再运用 ftprename 函数，对 strremotefilename 改名，改成这个 xxxxxbak，就 ok 了

```
            // 转存到备份目录
            if(starg.ptype == 3){
                    char strremotefilenamebak[301];
                    SNPRINTF(strremotefilenamebak, sizeof(strremotefilenamebak), 300, "%s/%s", starg.remotepathbak, vlistfile[i].
filename);
                    if(ftp.ftprename(strremotefilename, strremotefilenamebak) == false){
                            logfile.Write("ftp.ftprename(%s, %s) failed.\n", strremotefilename, strremotefilenamebak);
                            return false;
                    }
            }
```

增量下载文件

```
    if(ptype == 1){
            // 加载okfilename文件中的内容到容器vlistfile1中。
            // 比较vlistfile2和vlistfile1，得到vlistfile3和vlistfile4
            // 把容器vlistfile3中的内容写入okfilename文件，覆盖之前的旧okfilename文件
            // 把vlistfile4中的内容复制到vlistfile2中
    }
```

PS：ptype == 1 即为访问，然后什么都不做，不删除也不备份

文件上传功能的实现

上传和下载的不同：

1. 想要得到文件的目录，在下载，需要分三步走，上传，只需要 dir 一个指令即可

```
// 进入ftp服务器存放文件的目录
if(ftp.chdir(starg.remotepath) == false){
        logfile.Write("ftp.chdir(%s) failed.\n", starg.remotepath);
        return false;
}

PActive.UptATime();

// 调用ftp.nlist()方法列出服务器目录中的文件，结果存放到本地文件中。
if(ftp.nlist(".", starg.listfilename) == false){
        logfile.Write("ftp.nlist(%s) failed.\n", starg.remotepath);
        return false;
}

PActive.UptATime();

// 把ftp.nlist()方法获取到的list文件加载到容器vlistfile2中
if(LoadListFile() == false){
        logfile.Write("LoadListFile() failed.\n");
        return false;
}
```

2. listfilename 不需要了，remotepathbak 改为 localpathbak，checkmtime 不需要了（检查服务端文件的时间）原因有如下

- checkmtime 里面 while 循环中有这句，意味着程序每运行一次，都需要把服务端目录中全部的时间取回来，如果服务端的目录很多，取时间这个动作要消耗大量的资源，包括客户端等待的时间，网络带宽，还有对服务端造成的压力，如果服务端中的目录不会更新，就应该把 checkmtime 设置为 false，在文件下载的过程中，checkmtime 的取值对性能和服务端的压力有很大的影响，但在上传的过程中 checkmtime 对程序的性能不会有任何的影响（没有任何代价），所以干脆就不要了

```
if ( (starg.ptype==1) && (starg.checkmtime==true) )
{
  // 获取ftp服务端文件时间。
  if (ftp.mtime(stfileinfo.filename)==false)
  {
    logfile.Write("ftp.mtime(%s) failed.\n",stfileinfo.filename); return false;
  }

  strcpy(stfileinfo.mtime,ftp.m_mtime);
}
```

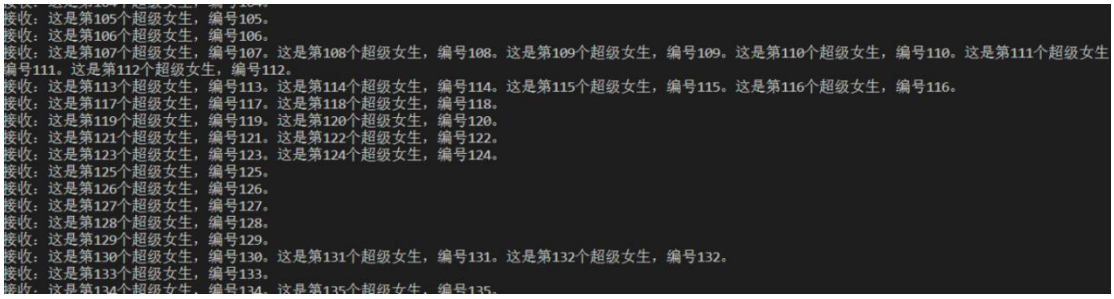
第三板块-基于 TCP 协议的文件传输系统

开发目标

1. 封装 socket 的 API
   (1) 解决 TCP 报文粘包/分包的问题
   (2) 封装 socket 的常用函数
2. 多进程的网络服务端
   (1) 搭建多进程的网络服务程序框架
   (2) TCP 短连接/长连接和心跳机制

粘包实践与解决方案

/project/public/socket 中的 demo03(客户端)和 demo04(服务端）



解决方案:
在项目开发中，采用自定义的报文格式。

报文长度＋报文内容 0010abcdefghi

采用 ASCII 码

在实际开发一般不采用，因为有一个问题，当报文的内容超过四个 9 的时候，四个字节就存不下了，用整型变量存放报文长度就不会存在这个问题

采用整型

```
char context[101];
strcpy(context,"abcdefghi");        // 待发送的报文内容。

int  ilen=strlen(context);          // 待发送报文的长度。

char TBuffer[ilen+4];               // 发送缓冲区。
memset(TBuffer,0,sizeof(TBuffer)); // 清区发送缓冲区。

memcpy(TBuffer,&ilenn,4);           // 把报文长度拷贝到缓冲区。
memcpy(TBuffer+4,context,ilen);     // 把报文内容拷贝到缓冲区。

// 把TBuffer发送出去。
```

在开发框架中，tcpwrite 和 tcpread 解决了这两个问题(粘包分包)
TcpWrite 和 TcpRead 的使用一定要成双成对的，也就是协议需要大家一起来遵守
TcpWrite()

我们看这几行代码，注意我们发送缓冲区数据是采用的封装的 Writen 函数，而不是 send（c 自带），原因就是因为 socket 有缓冲区，读和写两个缓冲区，并且大小有限，如果这时候的写缓冲区快满了，还有五百字节可以用，调用 send 函数，只能成功写入 500 字节，剩下的要等缓冲区空闲了才能再次写入。Writen 循环调用 send 函数，直到全部数据被成功的发送，返回 true，如果发送过程中 tcp 断开了或者其他原因，返回 false

```
// 向 socket 的对端发送数据。
// sockfd：可用的 socket 连接。
// buffer：待发送数据缓冲区的地址。
// ibuflen：待发送数据的字节数，如果发送的是 ascii 字符串，ibuflen 填 0 或字符串的长度，
//          如果是二进制流数据，ibuflen 为二进制数据块的大小。
// 返回值：true-成功；false-失败，如果失败，表示 socket 连接已不可用。
bool TcpWrite(const int sockfd,const char *buffer,const int ibuflen)
```

```
{
  if (sockfd==-1) return false;

  int ilen=0;  // 报文长度。

  // 如果 ibuflen==0，就认为需要发送的是字符串，报文长度为字符串的长度。
  if (ibuflen==0) ilen=strlen(buffer);
  else ilen=ibuflen;

  int ilenn=htonl(ilen);      // 把报文长度转换为网络字节序。

  char TBuffer[ilen+4];       // 发送缓冲区。
  memset(TBuffer,0,sizeof(TBuffer));  // 清区发送缓冲区。
  memcpy(TBuffer,&ilenn,4);           // 把报文长度拷贝到缓冲区。
  memcpy(TBuffer+4,buffer,ilen);      // 把报文内容拷贝到缓冲区。

  // 发送缓冲区中的数据。
  if (Writen(sockfd,TBuffer,ilen+4) == false) return false;

  return true;
}
```

TcpRead()

先不管超时时间，我们来梳理一下过程，把传进来的字节数初始化为 0(注意用了解引用*)意思是我们把这个传进来的 int 型的地址里面的内容改为了 0，接着使用 Readn()接受 sockfd 中的内容，用 ibuflen 来当做容器接受好像不一定必须转为 char，但反正 recv 底层接受的是 void*，最后，在将报文长度的 网络字节序 用 ntohl 转换为主机字节序，最后在 buffer 接受得到报文的实际内容，同样用 readn()函数，读 sockfd 里面的，读取的长度设置为 ibuflen 的大小

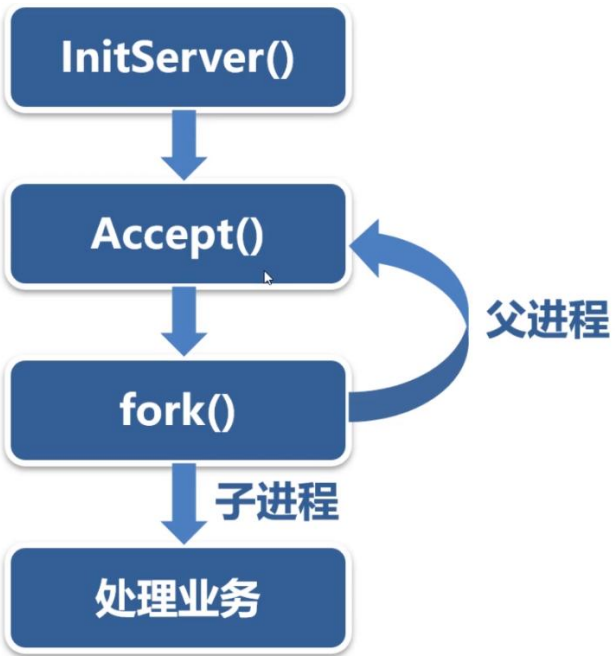
```
// 接收 socket 的对端发送过来的数据。
// sockfd：可用的 socket 连接。
// buffer：接收数据缓冲区的地址。
// ibuflen：本次成功接收数据的字节数。
// itimeout：接收等待超时的时间，单位：秒，-1-不等待；0-无限等待；>0-等待的秒数。
// 返回值：true-成功；false-失败，失败有两种情况：1）等待超时；2）socket 连接已不可用。
bool TcpRead(const int sockfd,char *buffer,int *ibuflen,const int itimeout)
{
  if (sockfd==-1) return false;

  // 如果 itimeout>0，表示需要等待 itimeout 秒，如果 itimeout 秒后还没有数据到达，返回 false。
  if (itimeout>0)
  {
```

```
    struct pollfd fds;
    fds.fd=sockfd;
    fds.events=POLLIN;
    if ( poll(&fds,1,itimeout*1000) <= 0 ) return false;
  }

  // 如果 itimeout==-1，表示不等待，立即判断 socket 的缓冲区中是否有数据，如果没有，返回 false。
  if (itimeout==-1)
  {
    struct pollfd fds;
    fds.fd=sockfd;
    fds.events=POLLIN;
    if ( poll(&fds,1,0) <= 0 ) return false;
  }

  (*ibuflen) = 0;   // 报文长度变量初始化为 0。

  // 先读取报文长度，4 个字节。
  if (Readn(sockfd,(char*)ibuflen,4) == false) return false;

  (*ibuflen)=ntohl(*ibuflen);   // 把报文长度由网络字节序转换为主机字节序。

  // 再读取报文内容。
  if (Readn(sockfd,buffer,(*ibuflen)) == false) return false;

  return true;
}
```

多进程的网络服务端

流程：

父进程先初始化服务端，然后 Accept 等待服务端的连接，新的客户端连上了之后，fork 一个子进程出来，然后父进程回到 accept 继续等待其他客户端的连接请求，让子进程与刚才连进来的客户端进行处理业务。



寄托于 fork 函数实现，我们用 while 反复迭代，达到不断接受客户端的目的（回到 Accept）

```
while (true)
{
  // 等待客户端的连接请求。
  if (TcpServer.Accept()==false)
  {
    printf("TcpServer.Accept() failed.\n"); return -1;
  }

  printf("客户端（%s）已连接。\n",TcpServer.GetIP());

  if (fork()>0) continue;  // 父进程继续回到Accept()。

  //
  char buffer[102400];

  // 与客户端通讯，接收客户端发过来的报文后，回复ok。
  while (1)
  {
    memset(buffer,0,sizeof(buffer));
    if (TcpServer.Read(buffer)==false) break; // 接收客户端的请求报文。
    printf("接收：%s\n",buffer);

    strcpy(buffer,"ok");
    if (TcpServer.Write(buffer)==false) break; // 向客户端发送响应结果。
    printf("发送：%s\n",buffer);
  }
```

子进程执行

但接下来就出现了一个问题（demo10 为服务端），现在，客户端已经全部结束了，但是服务端还有这么多进程，这是怎么回事呢？

那原因就在那个 while，当 fork 生成子进程，子进程进入 while，并且执行完自己的程序之后，就会继续进入上面那层 while，并且不断卡在连接阶段，就和父进程一起等待在哪里挂起了

所以我们应该在最下面增加一行代码，用 return 0 或者 exit(0)都可以

```
// 与客户端通讯，接收客户端发过来的报文后，回复ok。
while (1)
{
  memset(buffer,0,sizeof(buffer));
  if (TcpServer.Read(buffer)==false) break; // 接收客户端的请求报文。
  printf("接收：%s\n",buffer);

  strcpy(buffer,"ok");
  if (TcpServer.Write(buffer)==false) break; // 向客户端发送响应结果。
  printf("发送：%s\n",buffer);
}

return 0;  // exit(0);
```

第四板块-基于 Tcp 协议的文件传输系统

开发目标

文件传输的服务端板块（支持上传和下载）

文件上传的客户端板块

文件下载的客户端板块

文件上传服务端

```
// 上传文件的主函数
void RecvFileMain(){
  while(true){
    // 接受客户端的报文

    // 处理心跳报文

    // 处理上传文件的请求报文

  }
}
```

进一步的来讲是这样的，注意心跳报文，则是，如果此时接受的是<a.....则处理，最开始还没明白这是怎么处理的

```
// 上传文件的主函数
void RecvFileMain(){
  while(true){
    memset(strrecvbuffer, 0, sizeof(strrecvbuffer));
```

```
    memset(strsendbuffer, 0, sizeof(strsendbuffer));

    // 接受客户端的报文,timetvl 为扫描本地上传文件间隔，+10 代表上传的真正时间（随意，+5 也可以）
    if(TcpServer.Read(strrecvbuffer, starg.timetvl + 10) == false){
      logfile.Write("TcpServer.Read() failed.\n");
      return;
    }
    logfile.Write("strrecvbuffer = %s\n", strrecvbuffer);
    // 处理心跳报文
    if(strcmp(strrecvbuffer, "<activetest>ok</activetest>") == 0){
      strcpy(strsendbuffer, "ok");
      logfile.Write("strsendbuffer = %s\n", strsendbuffer);
      if(TcpServer.Write(strsendbuffer) == false){
        logfile.Write("TcpServer.Write() failed.\n");
        return;
      }
    }
    // 处理上传文件的请求报文
    if(strncmp(strrecvbuffer, "<filename>", 10) == 0){
      // 解析上传文件请求报文的 xml

      // 接受上传文件的内容

      // 把接受结果返回给对端
    }
  }
}
```

文件上传客户端

```
// 文件上传的主函数，执行一次文件上传的任务
bool _tcpputfiles(){
  // 调用 OpenDir()打开 starg.clientpath 目录

  while(true){
    // 遍历目录中的每个文件，调用 ReadDir()获取一个文件名

    // 把文件名、修改时间、文件大小组成报文，发送给对端

    // 把文件的内容发送给对端

    // 接受对端的确认报文
```

```
    // 删除或者转存本地的文件
  }
  return true;
}
```

上传文件内容

客户端

```
// 把文件的内容发送给对端
bool SendFile(const int sockfd, const char *filename, const int filesize){
  int  onread = 0;          // 每次调用 fread 时打算读取的字节数
  int  bytes  = 0;          // 调用一次 fread 从文件中读取的字节数
  char buffer[1000];        // 存放读取数据的 buffer
  int  totalbytes  = 0;          // 从文件中已读取的字节总数， 真的是太难了，完全没发现没有初始化
  FILE *fp = NULL;

  // 以"rb"的模式打开文件
  if( (fp = fopen(filename, "rb")) == NULL) return false;
  while(true){
    memset(buffer, 0, sizeof(buffer));

    // 计算本次应该读取的字节数，如果剩余数量超过 1000 字节，就打算读 1000 字节(模拟缓冲区，防止太大打不开)
    if(filesize - totalbytes > 1000) onread = 1000;
    else onread = filesize - totalbytes;

    // 从文件中读取数据
    bytes = fread(buffer, 1, onread, fp);

    // 把读取到的数据发送给对端
    if(bytes > 0){
      if(Writen(sockfd, buffer, bytes) == false){
        fclose(fp);
        return false;
      }
    }
    // 计算文件已读取的字节总数，如果文件已读完，跳出循环
    totalbytes = totalbytes + bytes;
```

```
        if(totalbytes == filesize)    break;
    }
    fclose(fp);
    return true;
}
```

服务端

```
// 接受上传文件的内容
bool    RecvFile(const int sockfd, const char *filename, const char *mtime, int filesize){
    // 生成临时的文件名

    // 创建临时文件

    while (true)
    {
        // 计算本次应该接受的字节数

        // 接受文件内容

        // 把接收到的内容写入文件

        // 计算已接收文件的总字节数，如果文件接受完，跳出循环
    }

    // 关闭临时文件

    // 重置文件的时间

    // 把临时文件 RENAME 为正式的文件。


    return true;
}
```

删除与转存文件

较为简单，简单看一下便是，仅仅只需要改动客户端，因为是上传功能，所以服务端没变化，客户端来决定把本地文件删除或者转存

```
// 删除或者转存本地的文件
bool AckMessage(const char *strrecvbuffer){
  char filename[301];
  char result[11];

  memset(filename, 0, sizeof(filename));
  memset(result, 0, sizeof(result));

  GetXMLBuffer(strrecvbuffer, "filename", filename, 300);
  GetXMLBuffer(strrecvbuffer, "result", result, 10);

  // 如果服务端接受文件不成功，直接返回
  if(strcmp(result, "ok") != 0) return true;

  // ptype == 1, 删除文件
  if(starg.ptype == 1){
      if(REMOVE(filename) == false) {
        logfile.Write("REMOCE(%s) failed.\n", filename);
        return false;
      }

    // ptype == 2, 移动到备份目录
    if(starg.ptype == 2){
      // 生成转存后备份目录的文件名。
      char bakfilename[301];
      STRCPY(bakfilename, sizeof(bakfilename), filename);
      UpdateStr(bakfilename, starg.clientpath, starg.clientpathbak, false);
      if(RENAME(filename, bakfilename) == false){
        logfile.Write("RENAME(%s, %s) failed.\n", filename, bakfilename);
        return false;
      }
    }
  }
  return true;
}
```

第五板块-MySQL 数据库开发

开发目标

站点数据入库

```
// 业务处理主函数。
bool _obtmindtodb(char *pathname,char *connstr,char *charset){
    sqlstatement stmt;

    CDir Dir;
    // 打开目录
    if(Dir.OpenDir(pathname, "*.xml") == false){
      logfile.Write("Dir.OpenDir(%s) failed.\n", pathname);
      return false;
    }

    CFile File;

    while (true){
        // 读取目录，得到一个数据文件名
        if(Dir.ReadDir() == false) break;

        // 连接数据库。
        if(conn.m_state == 0){
          if (conn.connecttodb(connstr, charset)!=0) {
            logfile.Write("connect database(%s) failed.\n%s\n", connstr,
conn.m_cda.message); return -1;
          }
        }
        logfile.Write("connect database(%s) ok.\n", connstr);

        if(stmt.m_state == 0){
          stmt.connect(&conn);
          stmt.prepare("insert into T_ZHOBTMIND(obtid, ddatetime, t, p, u, wd, wf, r, vis)
\
            values(:1, str_to_date(:2,
'%%Y%%m%%d%%H%%i%%s'), :3, :4, :5, :6, :7, :8, :9)");
          stmt.bindin(1, stzhobtmind.obtid, 10);// bindin2  bindin3.....
        }
        logfile.Write("filename = %s\n", Dir.m_FullFileName);

        // 打开文件
        if(File.Open(Dir.m_FullFileName, "r") == false){
          logfile.Write("File.Open(%s) failed.\n", Dir.m_FullFileName);
          return false;
```

```
        }

        char strBuffer[1001];    // 存放从文件中读取的一行。
        while(true){
            if(File.FFGETS(strBuffer, 1000, "<endl/>") == false) break;
            logfile.Write("strBuffer=%s", strBuffer);
            // 处理文件中的每一行
            memset(&stzhobtmind, 0, sizeof(struct st_zhobtmind));
            GetXMLBuffer(strBuffer, "obtid", stzhobtmind.obtid, 10);
            GetXMLBuffer(strBuffer,"ddatetime",stzhobtmind.ddatetime,14);
            char tmp[11];      // 统一单位，使得与数据库中相同
            GetXMLBuffer(strBuffer, "t", tmp, 10);          if(strlen(tmp) > 0)
snprintf(stzhobtmind.t, 10, "%d", (int)(atof(tmp)*10));
            GetXMLBuffer(strBuffer,"u",stzhobtmind.u,10);
            // 这里留两个样例，展示 tmp 的作用，.u 因为已经是整数,所以无需处理

            // 把结构体中的数据插入表中
            if(stmt.execute() != 0){
              // 1、失败的情况有哪些？是否全部的失败都要写日志？
              // 答：失败的原因主要有二：一是记录重复，二是数据内容非法。
              // 2、如果失败了怎么办？程序是否需要继续？是否 rollback？是否返回
false？
              // 答：如果失败的原因是数据内容非法，记录日志后继续；如果是记录
重复，不必记录日志，且继续。
              if(stmt.m_cda.rc != 1062){
                logfile.Write("Buffer = %s\n", strBuffer);
                logfile.Write("stmt.execute()          failed.\n%s\n%s\n",          stmt.m_sql,
stmt.m_cda.message);
              }
            }
        }

        // 删除文件、提交事务
        //File.CloseAndRemove();  因为测试，暂时先关闭，不然没数据了

        conn.commit();
    }

    return true;
}
```

优化业务一:优化日志

```
// 业务处理主函数。
bool _obtmindtodb(char *pathname,char *connstr,char *charset){
    // 打开目录
    int totalcount = 0;    // 文件的总记录数
    int insertcount = 0;   // 成功插入的记录数
    CTimer Timer;              // 计时器，记录每个文件的处理耗时

    while (true){
        // 读取目录，得到一个数据文件名

        totalcount = insertcount = 0;
        // 打开文件

        while(true){
            // 处理文件中的每一行
            totalcount++;
        }
        if(stmt.execute() != 0){
            ........
        }else insertcount++;

        // 删除文件、提交事务
    }
    logfile.Write("已处理文件%s(totalcount=%d, insertcount = %d), 耗时%.2f 秒。\n", \
                    Dir.m_FullFileName, totalcount, insertcount, Timer.Elapsed());
    return true;
}
```

优化任务二: 处理冗余程序

我们可以想象，如果我们的操作越来越多，那么程序只会越来越复杂，比如这里提到的表的字段 50 个，那我们光绑定参数，都要花 100 行，显然这是很没有必要的空间花费，我们说，还是想让程序变得优雅起来的。

```
// 全国站点分钟测试数据操作类
class CZHOBTMIND{
  public:
    connection *m_conn;        // 数据库连接
    CLogFile     *m_logfile;   // 日志

    sqlstatement m_stmt;       // 插入表操作的 sql
```

```
    char m_buffer[1024];       // 从文件中读到的一行
    struct st_zhobtmind m_zhobtmind;   // 全国站点分钟观测数据

    CZHOBTMIND();
    CZHOBTMIND(connection *conn, CLogFile *logfile);

    ~CZHOBTMIND();

    void BindConnLog(connection *conn, CLogFile *logfile); // 把 connection 和 CLogFile 的传进去
    bool SplitToBuffer(char *strBuffer);          // 把从文件读到的一行数据拆分到 m_zhobtmind 结构体中
    bool InsertTable();     // 把 m_zhobtmind 结构体中的数据插入到 T_ZHOBTMIND 表中
};

CZHOBTMIND::CZHOBTMIND(){
  m_conn=0;   m_logfile=0;
}

CZHOBTMIND::CZHOBTMIND(connection *conn, CLogFile *logfile){
  m_conn=conn;
  m_logfile=logfile;
}

CZHOBTMIND::~CZHOBTMIND(){
}

void CZHOBTMIND::BindConnLog(connection *conn, CLogFile *logfile){
  m_conn=conn;
  m_logfile=logfile;
}

// 把从文件读到的一行拆分到 m_zhobtmind 结构体中
bool CZHOBTMIND::SplitToBuffer(char *strBuffer){
  memset(&m_zhobtmind, 0, sizeof(struct st_zhobtmind));
  GetXMLBuffer(strBuffer, "obtid", m_zhobtmind.obtid, 10);
  char tmp[11];       // 统一单位，使得与数据库中相同
  GetXMLBuffer(strBuffer, "t", tmp, 10);     if(strlen(tmp) > 0) snprintf(m_zhobtmind.t, 10, "%d", (int)(atof(tmp)*10));
.....

  STRCPY(m_buffer, sizeof(m_buffer), strBuffer); // 注意这里要传过去
```

```
    return true;

}

//   把 m_zhobtmind 结构体中的数据插入到 T_ZHOBTMIND 表中
bool CZHOBTMIND::InsertTable(){
    if(m_stmt.m_state == 0){
        m_stmt.connect(m_conn);
        m_stmt.prepare("insert into T_ZHOBTMIND(obtid, ddatetime, t, p, u, wd, wf, r, vis) \
        values(:1, str_to_date(:2, '%%Y%%m%%d%%H%%i%%s'), :3, :4, :5, :6, :7, :8, :9)");
        m_stmt.bindin(1, m_zhobtmind.obtid, 10);
        ......
    }
        // 把结构体中的数据插入表中
    if(m_stmt.execute() != 0){
        // 1、失败的情况有哪些？是否全部的失败都要写日志？
        // 答：失败的原因主要有二：一是记录重复，二是数据内容非法。
        // 2、如果失败了怎么办？程序是否需要继续？是否 rollback？是否返回 false？
        // 答：如果失败的原因是数据内容非法，记录日志后继续；如果是记录重复，不必记录日志，且继续。
        if(m_stmt.m_cda.rc != 1062){
            m_logfile -> Write("Buffer = %s\n", m_buffer);
            m_logfile -> Write("m_stmt.execute() failed.\n%s\n%s\n", m_stmt.m_sql, m_stmt.m_cda.message);
        }
        return false;
    }
    return true;
}

// 业务处理主函数。
bool _obtmindtodb(char *pathname,char *connstr,char *charset){
                    // 处理文件中的每一行
                    totalcount++;

                    ZHOBTMIND.SplitToBuffer(strBuffer);

                    if(ZHOBTMIND.InsertTable() == true)    insertcount++;
}
```

这样的话，无论代码有多长（插入绑定 bindin，解析 xml），我们都只需要**两行**。

优化任务三：程序模块分离

我们仔细思考不难得到，在数据中心项目中，这种得到数据，然后上传库的程序，他们的逻辑是完全相同的，并且可以这么说，有多少种数据，就得到了多少个程序。并且，我们定义的结构体，在其他程序中也有可能用得上（存放站点代码，数据时间，温度....），我们定义的数据操作类也是同样的道理，那么，这些代码是不是可以分离出去？

我们可以这样，为这个项目，创造一个头文件和一个 cpp 文件。

- 结构体和类的声明分离到头文件中
- 类的实现代码，分离到 cpp 文件中

所以到了这里，我又去百度了头文件，cpp 文件这些标准写法，果然不断学习，就会不断有新的感悟.

参考链接：
https://blog.csdn.net/weixin_41969690/article/details/105587141

定义了 idcapp.h 和 idcapp.cpp 将其冗余部分存放

优化任务四：对 csv 格式的支持

进入业务处理主函数

1. 修改 Dir.OpenDir 里面的代码，加入支持读入*.csv
2. 加入 bool 变量 isxml true 为 xml false 为 csv
3. 读取目录，得到数据文件名后，判断他的后缀，根据后缀修改 isxml 的值
4. 读取文件的每一行时候，更改读取结束时候的判断条件
5. SplitBuffer 增加传入参数 bisxml

```
2022-05-15 18:08:41 connect database(127.0.0.1,root,123456,h2,3306) ok.
2022-05-15 18:08:42 已处理文件/idcdata/surfdata/SURF_ZH_20220515174624_14133.csv(totalcount = 839, insertcount = 839), 耗时
0.10秒。
2022-05-15 18:08:42 已处理文件/idcdata/surfdata/SURF_ZH_20220515174724_14242.csv(totalcount = 839, insertcount = 839), 耗时
0.10秒。
2022-05-15 18:08:42 已处理文件/idcdata/surfdata/SURF_ZH_20220515174824_14301.csv(totalcount = 839, insertcount = 839), 耗时
0.09秒。
```

执行 SQL 脚本文件

现在我们有一个新的需求，希望定期执行 sql 脚本，并清理历史数据，就像之前开发的，清理历史文件一样

```
[root@localhost ~]# mysql -uroot -pmysqlpwd -Dmysql < /project/idc/sql/cleardata.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
```

在 linux 下面，可以直接这样执行，注意只是报警，并没有报错，不过这里使用了输入重定

向功能，而我们的调度程序并不支持，所以只能自己写 hhhc

程序流程也十分简单，打开文件，一行一行的读取出来，执行，参数也无需绑定
具体过程是这样的：

```
    // 打开日志文件
    if(logfile.Open(argv[4], "a+") == false){
        printf("打开日志文件失败（%s）.\n", argv[4]);
        return -1;
    }

    PActive.AddPInfo(500,"obtcodetodb");    // 进程的心跳
    // 注意，在调试程序的时候，可以启用类似以下的代码，防止超时。
    // PActive.AddPInfo(5000,"obtcodetodb");

// 连接数据库，不启用事务。
  if (conn.connecttodb(argv[2], argv[3], 1)!=0){
    logfile.Write("connect database(%s) failed.\n%s\n", argv[2], conn.m_cda.message);
    return -1;
  }

  logfile.Write("connect database(%s) ok.\n", argv[2]);

  CFile File;

  // 打开 SQL 文件。
  if (File.Open(argv[1], "r")==false){
    logfile.Write("File.Open(%s) failed.\n", argv[1]);
    EXIT(-1);
  }

  char strsql[1001];      // 存放从 SQL 文件中读取的 SQL 语句。

  while (true)
  {
    memset(strsql, 0, sizeof(strsql));

    // 从 SQL 文件中读取以分号结束的一行。
    if (File.FFGETS(strsql, 1000, ";") == false) break;

    // 如果第一个字符是#，注释，不执行。
    if (strsql[0] == '#') continue;

    // 删除掉 SQL 语句最后的分号。也许有更好的方式
    char *pp = strstr(strsql, ";");
```

```
        if (pp == 0) continue;
        pp[0] = 0;

        logfile.Write("%s\n", strsql);

        int iret = conn.execute(strsql);   // 执行 SQL 语句。

        // 把 SQL 语句执行结果写日志。
        if (iret == 0) logfile.Write("exec ok(rpc=%d).\n", conn.m_cda.rpc);
        else logfile.Write("exec failed(%s).\n", conn.m_cda.message);

        PActive.UptATime();     // 进程的心跳。
    }

    logfile.WriteEx("\n");
```

```
2022-05-15 20:26:31 connect database(127.0.0.1,root,123456,h2,3306) ok.
2022-05-15 20:26:31 delete from T_ZHOBTMIND   where ddatetime<timestampadd(minute,-120,now())
2022-05-15 20:26:33 exec ok(rpc=312309).
2022-05-15 20:26:33 delete from T_ZHOBTMIND1  where ddatetime<timestampadd(minute,-120,now())
2022-05-15 20:26:33 exec failed(Table 'h2.T_ZHOBTMIND1' doesn't exist).
```

第六板块-开发数据抽取子系统

开发目标

是通用的功能模块，只需要配置脚本就可以实现对不同数据源的抽取；

支持全量和增量数据抽取的两种方式；

支持多种数据库（MySQL、Oracle、SQL Server、PostgreSQL）

搭建框架

宏结构体

```
// 程序运行参数的结构体
struct st_arg{          //注意都多了 1，因为 char 数组最后要留给\0 一个位置
        char connstr[101];          // 数据库的连接参数。
        char charset[51];           // 数据库的字符集。
        char selectsql[1024];    // 从数据源数据库抽取数据的 SQL 语句。
        char fieldstr[501];       // 抽取数据的 SQL 语句输出结果集字段名，字段名之间用逗号
```

```
分隔。
    char fieldlen[501];     // 抽取数据的 SQL 语句输出结果集字段的长度，用逗号分隔。
    char bfilename[31];     // 输出 xml 文件的前缀。
    char efilename[31];     // 输出 xml 文件的后缀。
    char outpath[301];      // 输出 xml 文件存放的目录。
    char starttime[52];     // 程序运行的时间区间
    char incfield[31];      // 递增字段名。
    char incfilename[301]; // 已抽取数据的递增字段最大值存放的文件。
    int  timeout;           // 进程心跳的超时时间。
    char pname[51];         // 进程名，建议用"dminingmysql_后缀"的方式。
} starg;

#define MAXFIELDCOUNT 100       // 结果集字段的最大数。
//#define MAXFIELDLEN     500   // 结果集字段值的最大长度。
int MAXFIELDLEN=-1;    // 结果集字段值的最大长度，存放 fieldlen 数组中元素的最大值。

char strfieldname[MAXFIELDCOUNT][31];     // 结果集字段名数组，从 starg.fieldstr 解析得到。
int  ifieldlen[MAXFIELDCOUNT];            // 结果集字段的长度数组，从 starg.fieldlen 解析得到。
int  ifieldcount;                         // strfieldname 和 ifieldlen 数组中有效字段的个数。
int  incfieldpos=-1;                      // 递增字段在结果集数组中的位置。
```

帮助文档

```
void _help(){
    printf("Using:/project/tools1/bin/dminingmysql logfilename xmlbuffer\n\n");

    printf("Sample:/project/tools1/bin/procctl 3600 /project/tools1/bin/dminingmysql /log/idc/dminingmysql_ZHOBTCODE.log \"<connstr>127.0.0.1,root,mysqlpwd,mysql,3306</connstr><charset>gbk</charset><selectsql>select obtid,cityname,provname,lat,lon,height from T_ZHOBTCODE</selectsql><fieldstr>obtid,cityname,provname,lat,lon,height</fieldstr><fieldlen>10,30,30,10,10,10</fieldlen><bfilename>ZHOBTCODE</bfilename><efilename>HYCZ</efilename><outpath>/idcdata/dmindata</outpath><timeout>30</timeout><pname>dminingmysql_ZHOBTCODE</pname>\"\n\n");
    printf(" /project/tools1/bin/procctl 30 /project/tools1/bin/dminingmysql /log/idc/dminingmysql_ZHOBTMIND.log \"<connstr>127.0.0.1,root,mysqlpwd,mysql,3306</connstr><charset>gbk</charset><selectsql>select obtid,date_format(ddatetime,'%%%%Y-%%%%m-%%%%d %%%%H:%%%%i:%%%%s'),t,p,
```

```
u,wd,wf,r,vis,keyid from t_zhobtmind where keyid>:1 and ddatetime>timestampadd(minute,-120,now())</selectsql><fieldstr>obtid,ddatetime,t,p,u,wd,wf,r,vis,keyid</fieldstr><fieldlen>10,19,8,8,8,8,8,8,8,15</fieldlen><bfilename>ZHOBTMIND</bfilename><efilename>HYCZ</efilename><outpath>/idcdata/dmindata</outpath><starttime></starttime><incfield>keyid</incfield><incfilename>/idcdata/dmining/dminingmysql_ZHOBTMIND_HYCZ.list</incfilename><timeout>30</timeout><pname>dminingmysql_ZHOBTMIND_HYCZ</pname>\"\n\n");

    printf("本程序是数据中心的公共功能模块，用于从 mysql 数据库源表抽取数据，生成 xml 文件。\n");
    printf("logfilename  本程序运行的日志文件。\n");
    printf("xmlbuffer    本程序运行的参数，用 xml 表示，具体如下：\n\n");

    printf("connstr      数据库的连接参数，格式：ip,username,password,dbname,port。\n");
    printf("charset      数据库的字符集，这个参数要与数据源数据库保持一致，否则会出现中文乱码的情况。\n");
    printf("selectsql    从数据源数据库抽取数据的 SQL 语句，注意：时间函数的百分号%需要四个，显示出来才有两个，被 prepare 之后将剩一个。\n");
    printf("fieldstr     抽取数据的 SQL 语句输出结果集字段名，中间用逗号分隔，将作为 xml 文件的字段名。\n");
    printf("fieldlen     抽取数据的 SQL 语句输出结果集字段的长度，中间用逗号分隔。fieldstr 与 fieldlen 的字段必须一一对应。\n");
    printf("bfilename    输出 xml 文件的前缀。\n");
    printf("efilename    输出 xml 文件的后缀。\n");
    printf("outpath      输出 xml 文件存放的目录。\n");
    printf("starttime    程序运行的时间区间，例如 02,13 表示：如果程序启动时，踏中 02 时和 13 时则运行，其它时间不运行。"\
           "如果 starttime 为空，那么 starttime 参数将失效，只要本程序启动就会执行数据抽取，为了减少数据源"\
           "的压力，从数据库抽取数据的时候，一般在对方数据库最闲的时候时进行。\n");
    printf("incfield     递增字段名，它必须是 fieldstr 中的字段名，并且只能是整型，一般为自增字段。"\
           "如果 incfield 为空，表示不采用增量抽取方案。");
    printf("incfilename  已抽取数据的递增字段最大值存放的文件，如果该文件丢失，将重新抽取全部的数据。\n");
    printf("timeout      本程序的超时时间，单位：秒。\n");
    printf("pname        进程名，尽可能采用易懂的、与其它进程不同的名称，方便故障排查。\n\n\n");

}
```

参数解析

这里主要体现最核心的改动，是基于宏结构体新加入的结果段而制作的，因为我们现在要做的是一个通用的模块，所以我们必须考虑兼容的情况，就会多花一些功夫

主要功能是：

- 获取字段数量
- 获取每一个字段的 name
- 核查字段数量是否一一对应
- 得到想要字段的索引

```
bool _xmltoarg(char *strxmlbuffer){
    memset(&starg,0,sizeof(struct st_arg));
    .....

    // 1、把 starg.fieldlen 解析到 ifieldlen 数组中；
    CCmdStr CmdStr;

    // 1、把 starg.fieldlen 解析到 ifieldlen 数组中；
    // ifieldlen 为 CCmdStr 自带的容器，当切断变量时，处理过的数据默认保存在内
    CmdStr.SplitToCmd(starg.fieldlen,",");

    // 判断字段数是否超出 MAXFIELDCOUNT 的限制。
    if (CmdStr.CmdCount()>MAXFIELDCOUNT){
        logfile.Write("fieldlen 的字段数太多，超出了最大限制 %d 。\n",MAXFIELDCOUNT); return false;
    }

    for (int ii=0;ii<CmdStr.CmdCount();ii++){
        CmdStr.GetValue(ii,&ifieldlen[ii]);
        // if (ifieldlen[ii]>MAXFIELDLEN) ifieldlen[ii]=MAXFIELDLEN;    // 字段的长度不能超过 MAXFIELDLEN。
        // 这里将 MAXFIELDLEN 从宏改为了变量，一旦出现溢出情况，方便扩容，不存在切断字段的风险。
        if (ifieldlen[ii]>MAXFIELDLEN) MAXFIELDLEN=ifieldlen[ii];    // 得到字段长度的最大值。
    }

    ifieldcount=CmdStr.CmdCount();

    // 2、把 starg.fieldstr 解析到 strfieldname 数组中；
    CmdStr.SplitToCmd(starg.fieldstr,",");
```

```
    // 判断字段数是否超出 MAXFIELDCOUNT 的限制。
    if (CmdStr.CmdCount()>MAXFIELDCOUNT){
        logfile.Write("fieldstr 的字段数太多，超出了最大限制%d。\n",MAXFIELDCOUNT); return false;
    }

    for (int ii=0;ii<CmdStr.CmdCount();ii++){
        CmdStr.GetValue(ii,strfieldname[ii],30);
    }

    // 判断 strfieldname 和 ifieldlen 两个数组中的字段是否一致。
    if (ifieldcount!=CmdStr.CmdCount()){
        logfile.Write("fieldstr 和 fieldlen 的元素数量不一致。\n"); return false;
    }

    // 3、获取自增字段在结果集中的位置。
    if (strlen(starg.incfield)!=0){
        for (int ii=0;ii<ifieldcount;ii++)
         // strcmp 用于判断想要检测的索引是否在当前容器中，若在则返回 0，并用 incfieldpos 记录此时的索引
        if (strcmp(starg.incfield,strfieldname[ii])==0) { incfieldpos=ii; break; }

        if (incfieldpos==-1){
        logfile.Write("递增字段名%s 不在列表%s 中。\n",starg.incfield,starg.fieldstr); return false;
        }
    }

    return true;
}
```

全量抽取功能

全量抽取就是原封不动的抽取数据，比较适合全国站点参数表，因为毕竟站点可能就几千个。

适用于数据量不大的表，例如业务餐胡鼠标，执行不带输出参数的 SQL 语句。

```
// 上传文件功能的主函数
bool _dminingmysql(){
    sqlstatement stmt(&conn);
    stmt.prepare(starg.selectsql);
    char strfieldvalue[ifieldcount][MAXFIELDLEN+1];  // 抽取数据的 SQL 执行后，存放结果集字段值的数组。
    for (int ii=1;ii<=ifieldcount;ii++){
        // 这里为了防止绑定的字段长度大于 MAXFIELDLEN，所以对宏做了处理，
        // 改为变量，利用 if 语句，一旦出现超过的，那么就赋值 max 为该值
        stmt.bindout(ii,strfieldvalue[ii-1],ifieldlen[ii-1]);
    }

    // 执行 sql 语句
    if (stmt.execute()!=0){
        logfile.Write("stmt.execute()  failed.\n%s\n%s\n",stmt.m_sql,stmt.m_cda.message);
return false;
    }

    CFile File;  // 用于操作 xml 文件。

    while (true){
        memset(strfieldvalue,0,sizeof(strfieldvalue));

        // 从 sql 的执行语句获得一条记录
        if (stmt.next()!=0) break;

        // 打开文件放在循环内，就避免生成空文件的情况
        if(File.IsOpened() == false){
            // 这个函数在下面有  // 生成 xml 文件名
            crtxmlfilename();

            // 设置写入权限
            if(File.OpenForRename(strxmlfilename, "w+") == false){
                logfile.Write("File.OpenForRename(%s) failed. \n", strxmlfilename);
                return false;
            }
            // 打开一个文件，在行首加<data>\n
            File.Fprintf("<data>\n");
        }
        // 将一句转化为 xml 格式
        for (int ii=1;ii<=ifieldcount;ii++)
            // 使得转变为 xml 格式

    File.Fprintf("<%s>%s</%s>",strfieldname[ii-1],strfieldvalue[ii-1],strfieldname[ii-1]);
```

```
            // 在这一句后面添加<end/>
            File.Fprintf("<endl/>\n");
        }

        // 行尾加</data>\n
        if(File.IsOpened() == true){
            File.Fprintf("</data>\n");


            // 关闭失败，记录原因
            if(File.CloseAndRename() == false){
                logfile.Write("File.CloseAndRename(%s) failed.\n", strxmlfilename);
                return false;
            }
            // 关闭成功，写日志
            logfile.Write("生成文件%s(%d). \n", strxmlfilename, stmt.m_cda.rpc);
        }
        return true;
}


void crtxmlfilename(){  // 生成 xml 文件名
        // xml 全路径文件名=start.outpath + starg.bfilename +  当前时间  + starg.efilename
+ .xml
        // char strxmlfilename[301]    xml 文件名
        char strLocalTime[21];
        memset(strLocalTime, 0, sizeof(strLocalTime));
        LocalTime(strLocalTime, "yyyymmddhh24miss");
        SNPRINTF(strxmlfilename,  300,  sizeof(strxmlfilename),  "%s/%s_%s_%s.xml",
starg.outpath, starg.bfilename, strLocalTime, starg.efilename);

}
```

增量抽取功能

```
bool _dminingmysql(){
    // 从 starg.incfilename 文件中获取已抽取数据的最大 id
    readincfile();

    // 如果是增量抽取，绑定输入参数（已抽取数据的最大 id）。
    // imaxincvalue 为全局变量。
    if(strlen(starg.incfield) != 0) stmt.bindin(1, &imaxincvalue);

    if (stmt.execute()!=0){
        logfile.Write("stmt.execute()  failed.\n%s\n%s\n",stmt.m_sql,stmt.m_cda.message);
return false;
    }

        // 更新自增字段的最大值。
        // 如果没有自增字段，就不需要这样判断
        //  if  (imaxincvalue<atol(strfieldvalue[incfieldpos]))
imaxincvalue=atol(strfieldvalue[incfieldpos]);
        if ( (strlen(starg.incfield)!=0) && (imaxincvalue<atol(strfieldvalue[incfieldpos])) )
       imaxincvalue=atol(strfieldvalue[incfieldpos]);

     // 把最大的自增字段的值写入 starg.incfilename 文件中。
     if (stmt.m_cda.rpc>0) writeincfile();
}

bool readincfile(){   // 从 starg.incfilename 文件中获取已抽取数据的最大 id。
    imaxincvalue=0;     // 自增字段最大值

    // 如果 starg.incfield 参数为空，表示不是增量抽取。
    if(strlen(starg.incfield)==0) return true;

    CFile File;

    // 如果打开 starg.incfilename 文件失败，表示是第一次运行程序，也不必返回失败。
    // 也可能是文件丢了，那也没办法，只能重新抽取。
    if (File.Open(starg.incfilename,"r")==false) return true;

    // 从文件中读取已抽取数据的最大 id。
    char strtemp[31];
    File.FFGETS(strtemp, 30);

    imaxincvalue = atol(strtemp);
```

```
    logfile.Write("上次已抽取数据的位置 （%s=%ld)。\n", starg.incfield, imaxincvalue);

    return true;
}
bool writeincfile(){  // 把已抽取数据的最大 id 写入 starg.incfilename 文件。
    // 如果 starg.incfield 参数为空，表示不是增量抽取。
    if (strlen(starg.incfield)==0) return true;

    CFile File;

    if (File.Open(starg.incfilename,"w+")==false)
    {
        logfile.Write("File.Open(%s) failed.\n",starg.incfilename); return false;
    }

    // 把已抽取数据的最大 id 写入文件。
    File.Fprintf("%ld",imaxincvalue);

    File.Close();

    return true;
}
```

第七板块-数据入库子系统

开发目标

是通用的功能模块，只需要配置参数就可以把 xml 文件入库；

支持对表的插入和修改两种操作；

支持多种数据库（MySQL、Oracle、SQL Server、PostgreSQL）

入库设计要求:

生成 xml 文件时，数据的标签与表字段名相同；

在参数配置文件描述了 xml 文件与表的对应关系；我们的设计是这样的，可以理解一下理论是否能做到。

读取目录中的 xml 文件，查找 xml 文件与表的对应关系；从数据字典中查找表的字段信息；

解析 xml 文件，把数据插入表，如记录已存在，以主键字段为条件，更新表中的记录。

加载文件功能

```
// 把数据入库的参数配置文件 starg.inifilename 加载到 vxmltotable 容器中。
bool loadxmltotable(){
  vxmltotable.clear();

  CFile File;
  if (File.Open(starg.inifilename,"r")==false){
    logfile.Write("File.Open(%s)  失败。\n",starg.inifilename); return false;
  }

  char strBuffer[501];

  while (true){
    if (File.FFGETS(strBuffer,500,"<endl/>")==false) break;

    memset(&stxmltotable,0,sizeof(struct st_xmltotable));

    GetXMLBuffer(strBuffer,"filename",stxmltotable.filename,100); // xml 文件的匹配规则，
用逗号分隔。
    GetXMLBuffer(strBuffer,"tname",stxmltotable.tname,30);          // 待入库的表名。
    GetXMLBuffer(strBuffer,"uptbz",&stxmltotable.uptbz);            // 更新标志：1-更新；
2-不更新。
    GetXMLBuffer(strBuffer,"execsql",stxmltotable.execsql,300);   // 处理 xml 文件之前，
执行的 SQL 语句。

    vxmltotable.push_back(stxmltotable);
  }

  logfile.Write("loadxmltotable(%s) ok.\n",starg.inifilename);

  return true;
}
```

查找文件功能

```
// 从 vxmltotable 容器中查找 xmlfilename 的入库参数，存放在 stxmltotable 结构体中。
bool findxmltotable(char *xmlfilename){
   for (int ii=0;ii<vxmltotable.size();ii++){
      if (MatchStr(xmlfilename,vxmltotable[ii].filename)==true){
         memcpy(&stxmltotable,&vxmltotable[ii],sizeof(struct st_xmltotable));
         return true;
      }
   }

   return false;
}
```

加载频率功能

我们可以思考一下 loadxmltotable 应该放在程序主流程的哪里，如果放在程序开头，意味着只加载一次，后面无法变动，如果直接放在 while 里，也意味着每次程序运行都要加载，过于频繁，因此我们可以定义一个**计数器**。

```
// 业务处理主函数。
bool _xmltodb(){
   int counter=50;   // 加载入库参数的计数器，初始化为 50 是为了在第一次进入循环的时候就加载参数。

   CDir Dir;

   while (true){
      if (counter++>30){
         counter=0;     // 重新计数。
         // 把数据入库的参数配置文件 starg.inifilename 加载到 vxmltotable 容器中。
         if (loadxmltotable()==false) return false;
      }
```

入库的参数文件会修改，但是修改的频率也不是很高。这样每循环三十次，再重新加载，就相对来说节省内存，又能及时加载，也是经典折中处理。

流程补充

```
// 业务处理主函数。
bool _xmltodb(){
  int counter=50;  // 加载入库参数的计数器，初始化为 50 是为了在第一次进入循环的时候就加载参数。

  CDir Dir;

  while (true){
    if (counter++>30){
      counter=0;    // 重新计数。
      // 把数据入库的参数配置文件 starg.inifilename 加载到 vxmltotable 容器中。
      if (loadxmltotable()==false) return false;
    }

    // 打开 starg.xmlpath 目录，为了保证先生成的数据入库，打开目录的时候，应该按文件名排序。
    if (Dir.OpenDir(starg.xmlpath,"*.XML",10000,false,true)==false){
      logfile.Write("Dir.OpenDir(%s) failed.\n",starg.xmlpath); return false;
    }

    while (true){
      // 读取目录，得到一个 xml 文件。
      if (Dir.ReadDir()==false) break;

      logfile.Write("处理文件%s...",Dir.m_FullFileName);

      // 调用处理 xml 文件的子函数。
      int iret=_xmltodb(Dir.m_FullFileName,Dir.m_FileName);

      // 处理 xml 文件成功，写日志，备份文件。
      if (iret==0){
        logfile.WriteEx("ok.\n");
        // 把 xml 文件移动到 starg.xmlpathbak 参数指定的目录中，一般不会发生错误，如果真发生了，程序将退出。
        if (xmltobakerr(Dir.m_FullFileName,starg.xmlpath,starg.xmlpathbak)==false) return false;
      }

      // 如果处理 xml 文件失败，分多种情况。
      if (iret==1){      // iret==1，找不到入库参数。暂时先一种
```

```
            logfile.WriteEx("failed，没有配置入库参数。\n");
            // 把 xml 文件移动到 starg.xmlpatherr 参数指定的目录中，一般不会发生错误，
如果真发生了，程序将退出。
            if (xmltobakerr(Dir.m_FullFileName,starg.xmlpath,starg.xmlpatherr)==false) return
false;
          }
        }

        break;// 测试时只循环一次
        sleep(starg.timetvl);
    }

    return true;
}
```

备份文件

```
// 把 xml 文件移动到备份目录或错误目录。
bool xmltobakerr(char *fullfilename,char *srcpath,char *dstpath){
    char dstfilename[301];     // 目标文件名。
    STRCPY(dstfilename,sizeof(dstfilename),fullfilename);

    UpdateStr(dstfilename,srcpath,dstpath,false);       // 小心第四个参数，一定要填 false。

    if (RENAME(fullfilename,dstfilename)==false){
        logfile.Write("RENAME(%s,%s) failed.\n",fullfilename,dstfilename); return false;
    }

    return true;
}
```

核心入库框架

```
// 处理 xml 文件的子函数，返回值：0-成功，其它的都是失败，失败的情况有很多种，暂
时不确定。
int _xmltodb(char *fullfilename,char *filename){
    // 从 vxmltotable 容器中查找 filename 的入库参数，存放在 stxmltotable 结构体中。
    if (findxmltotable(filename)==false) return 1;

    CTABCOLS TABCOLS;
```

```
  // 获取表全部的字段和主键信息，如果获取失败，应该是数据库连接已失效。
  // 在本程序运行的过程中，如果数据库出现异常，一定会在这里发现。
  if (TABCOLS.allcols(&conn,stxmltotable.tname)==false) return 4;
  if (TABCOLS.pkcols(&conn,stxmltotable.tname)==false)    return 4;

  // 如果 TABCOLS.m_allcount 为 0，说明表根本不存在，返回 2。
  if (TABCOLS.m_allcount==0) return 2; // 待入库的表不存在。

  // 拼接生成插入和更新表数据的 SQL。

  // prepare 插入和更新的 sql 语句，绑定输入变量。

  // 在处理 xml 文件之前，如果 stxmltotable.execsql 不为空，就执行它。

  // 打开 xml 文件。

/*
  while (true)
  {
    // 从 xml 文件中读取一行。

    // 解析 xml，存放在已绑定的输入变量中。

    // 执行插入和更新的 SQL。
  }
*/

  return 0;
}
```

第八板块-数据同步子系统

开发目标

开发刷新同步数据模块

全表刷新，适用于数据量不大的表，保证数据的完整性

分批刷新，适用于数据量较大的表，不能保证数据的完整性

全表刷新功能

删除本地表中全部的记录；

把 Federated 表中全部的记录插入本地表。

select * from LK_ZHOBTCODE1;

select * from LK_ZHOBTCODE1; delete from T_ZHOBTCODE2;

insert into T_ZHOBTCODE2(stid ,cityname,provname,lat,lon,altitude ,upttime,keyid)

select obtid,cityname,provname,lat,lon,height/10,upttime,keyid from LK_ZHOBTCODE1;

分批刷新功能

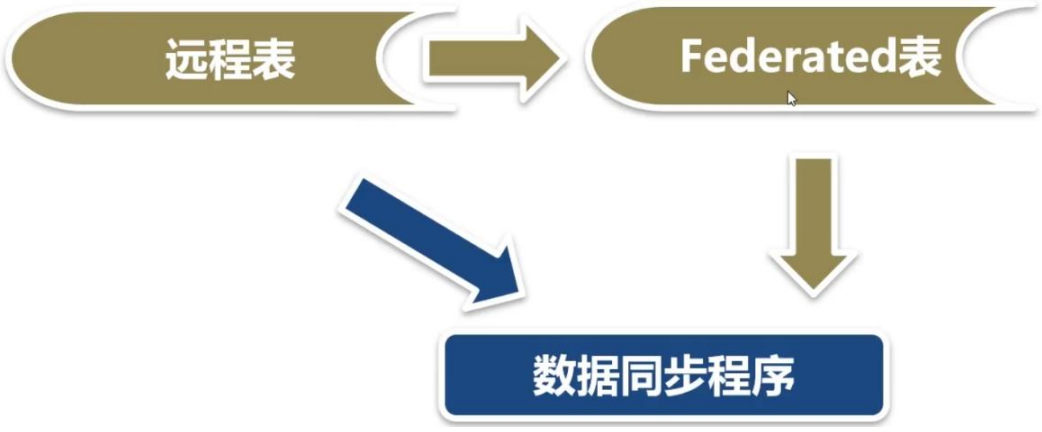
从远程表中查询需要同步的数据，把结果集分成若干批，每批的记录数在 50—256 之间；

删除本地表中指定批的记录；

把 Federated 表中指定批的记录插入本地表。

两个注意点

1. 分批操作的流程需要一个循环，在循环里面执行 2、3 步，直到全部的数据被处理完。
2. 从远程表查询需要的数据，为什么不在 federated 表，原因有两个
   1. federated 不支持普通索引，如果同步的条件不是主键，也不是唯一键，就会进行全表扫描。
   2. 就算 federated 表支持同步索引，也没有直接访问远程表来得好，因为传给 federated 需要经过一次中转，肯定没有不中转好。



不同步分批

```
  // 如果是不分批同步，表示需要同步的数据量比较少，执行一次 SQL 语句就可以搞定。
  if (starg.synctype==1){
    logfile.Write("sync %s to %s ...",starg.fedtname,starg.localtname);

    // 先删除 starg.localtname 表中满足 where 条件的记录。
    stmtdel.prepare("delete from %s %s",starg.localtname,starg.where);
    if (stmtdel.execute()!=0){
      logfile.Write("stmtdel.execute()
failed.\n%s\n%s\n",stmtdel.m_sql,stmtdel.m_cda.message); return false;
    }

    // 再把 starg.fedtname 表中满足 where 条件的记录插入到 starg.localtname 表中。
    stmtins.prepare("insert into %s(%s) select %s
from %s %s",starg.localtname,starg.localcols,starg.remotecols,starg.fedtname,starg.where);
    if (stmtins.execute()!=0){
      logfile.Write("stmtins.execute()
failed.\n%s\n%s\n",stmtins.m_sql,stmtins.m_cda.message);
      connloc.rollback();    // 如果这里失败了，可以不用回滚事务，connection 类的析构
函数会回滚。
      return false;
    }

    logfile.WriteEx(" %d rows in %.2fsec.\n",stmtins.m_cda.rpc,Timer.Elapsed());

    connloc.commit();

    return true;
  }
```

增量同步模块

开发了刷新同步之后，增量同步只需在此基础改就行。首先，增量同步肯定是分批的，我们删掉不分批的

```
// 业务处理主函数。
bool _syncincrement(bool &bcontinue){
   CTimer Timer;

   bcontinue=false;

   // 从本地表 starg.localtname 获取自增字段的最大值，存放在 maxkeyvalue 全局变量中。
   if (findmaxkey()==false) return false;

   // 从远程表查找自增字段的值大于 maxkeyvalue 的记录。
   char remkeyvalue[51];       // 从远程表查到的需要同步记录的 key 字段的值。
   sqlstatement stmtsel(&connrem);
   stmtsel.prepare("select %s from %s where %s>:1 %s order
by %s",starg.remotekeycol,starg.remotetname,starg.remotekeycol,starg.where,starg.remotekeycol
);
   stmtsel.bindin(1,&maxkeyvalue);
   stmtsel.bindout(1,remkeyvalue,50);
    // 剩下的就是同步刷新功能已经展示过的
```

第九板块-数据管理子系统

开发目标

数据清理：删除表中的符合条件数据；

数据迁移：把表中符合条件的数据迁移到其它的表（备份、归档）。

进一步的理解:

数据清理是指：数据没有价值了，需要删除。

数据迁移是指：出于性能与内存的考虑，把价值没这么大的数据移动一个位置。

数据清洗功能

根据迁移数据的条件，把源表中的唯一键字段查询出来。

以唯一键字段为条件，把源表中的记录插入目的表；

以唯一键字段为条件，删除源表中的记录。

```
// 业务处理主函数。
bool _deletetable(){
  CTimer Timer;

  char tmpvalue[51];      // 存放从表提取待删除记录的唯一键的值。

  // 从表提取待删除记录的唯一键。
  sqlstatement stmtsel(&conn1);
  stmtsel.prepare("select %s from %s %s",starg.keycol,starg.tname,starg.where);
  stmtsel.bindout(1,tmpvalue,50);

  // 拼接绑定删除 SQL 语句 where 唯一键 in (...)的字符串。
  char bindstr[2001];
  char strtemp[11];

  memset(bindstr,0,sizeof(bindstr));

  for (int ii=0;ii<MAXPARAMS;ii++){
    memset(strtemp,0,sizeof(strtemp));
    sprintf(strtemp,":%lu,",ii+1);
    strcat(bindstr,strtemp);
  }

  bindstr[strlen(bindstr)-1]=0;      // 最后一个逗号是多余的。

  char keyvalues[MAXPARAMS][51];    // 存放唯一键字段的值。

  // 准备删除数据的 SQL，一次删除 MAXPARAMS 条记录。
  sqlstatement stmtdel(&conn2);
  stmtdel.prepare("delete from %s where %s in (%s)",starg.tname,starg.keycol,bindstr);
  for (int ii=0;ii<MAXPARAMS;ii++)
    stmtdel.bindin(ii+1,keyvalues[ii],50);

  int ccount=0;
  memset(keyvalues,0,sizeof(keyvalues));

  if (stmtsel.execute()!=0){
    logfile.Write("stmtsel.execute()
failed.\n%s\n%s\n",stmtsel.m_sql,stmtsel.m_cda.message); return false;
  }

  while (true){
    memset(tmpvalue,0,sizeof(tmpvalue));
```

```
      // 获取结果集。
      if (stmtsel.next()!=0) break;

      strcpy(keyvalues[ccount],tmpvalue);
      ccount++;

      // 每 MAXPARAMS 条记录执行一次删除语句。
      if (ccount==MAXPARAMS){
        if (stmtdel.execute()!=0){
          logfile.Write("stmtdel.execute()
failed.\n%s\n%s\n",stmtdel.m_sql,stmtdel.m_cda.message); return false;
        }

        ccount=0;
        memset(keyvalues,0,sizeof(keyvalues));

        PActive.UptATime();
      }
    }

    // 如果不足 MAXPARAMS 条记录，再执行一次删除。
    if (ccount>0){
      if (stmtdel.execute()!=0){
        logfile.Write("stmtdel.execute()
failed.\n%s\n%s\n",stmtdel.m_sql,stmtdel.m_cda.message); return false;
      }
    }

    if (stmtsel.m_cda.rpc>0) logfile.Write("delete from %s %d rows
in %.02fsec.\n",starg.tname,stmtsel.m_cda.rpc,Timer.Elapsed());

    return true;
}
```

数据迁移

根据清理数据的条件，把表中的唯一键字段查询出来。

以唯一键字段为条件，删除表中的记录；

为了提高效率，每执行一次 SQL 语句删除 100 或者 200 条记录。

在数据清理的基础上多了以下步骤： 删除以前先备份。

```
// 准备插入和删除表数据的 sql，一次迁移 starg.maxcount 条记录。
  sqlstatement stmtins(&conn2);
  stmtins.prepare("insert into %s(%s) select %s from %s where %s in (%s)",starg.dsttname,TABCOLS.m_allcols,TABCOLS.m_allcols,starg.srctname,starg.keycol,bindstr);
```

第十板块-数据服务总线

开发目标

数据中心的目标是为业务系统提供数据支撑环境;

业务系统直连数据中心的应用数据库，任意访问数据;

业务系统通过数据服务总线，采用 HTTP 协议获取数据。

流程框架

```
  // 1、接收客户端的请求报文；
  // 2、解析 URL 中的参数，参数中指定了查询数据的条件；
  // 3、从 T_ZHOBTMIND1 表中查询数据，以 xml 格式返回给客户端。
```

```
// 接受请求
  CTcpServer TcpServer;

  // 服务端初始化。
  if (TcpServer.InitServer(atoi(argv[1]))==false)
  {
    printf("TcpServer.InitServer(%s) failed.\n",argv[1]); return -1;
  }

  // 等待客户端的连接请求。
  if (TcpServer.Accept()==false)
```

```
    {
      printf("TcpServer.Accept() failed.\n"); return -1;
    }

    printf("客户端（%s）已连接。\n",TcpServer.GetIP());

    char strget[102400];
    memset(strget,0,sizeof(strget));

    // 接收 http 客户端发送过来的报文。
    recv(TcpServer.m_connfd,strget,1000,0);

    printf("%s\n",strget);

    // 先把响应报文头部发送给客户端。
    char strsend[102400];
    memset(strsend,0,sizeof(strsend));
    sprintf(strsend,\
            "HTTP/1.1 200 OK\r\n"\
            "Server: demo28\r\n"\
            "Content-Type: text/html;charset=utf-8\r\n"\
            "\r\n");
            // "Content-Length: 108909\r\n\r\n");
    if (Writen(TcpServer.m_connfd,strsend,strlen(strsend))== false) return -1;

    //logfile.Write("%s",strsend);

    // 解析 GET 请求中的参数，从 T_ZHOBTMIND1 表中查询数据，返回给客户端。
    SendData(TcpServer.m_connfd,strget);
}
```

```
//
http://127.0.0.1:8080/api?username=wucz&passwd=wuczpwd&intetname=getZHOBTMIND1
&obtid=51076
// 从 GET 请求中获取参数的值：strget-GET 请求报文的内容；name-参数名；value-参数
值；len-参数值的长度。
bool getvalue(const char *strget,const char *name,char *value,const int len)
{
    value[0]=0;

    char *start,*end;
    start=end=0;

    // strstr 返回字符串 name 在 strget 中首次出现的地址。
```

```
    start=strstr((char *)strget,(char *)name);
    if (start==0) return false;

    // end 为一条参数值结束的位置。
    end=strstr(start,"&");
    // &和空格都是结束值，只不过空格是最后
    if (end==0) end=strstr(start," ");

    if (end==0) return false;

    // 每一条查询条件的长度
    int ilen=end-(start+strlen(name)+1);
    if (ilen>len) ilen=len;

    // 得到这条内容保存在 value 中
    strncpy(value,start+strlen(name)+1,ilen);

    value[ilen]=0;

    return true;
}
```

```
// 解析 GET 请求中的参数，从 T_ZHOBTMIND1 表中查询数据，返回给客户端。
bool SendData(const int sockfd,const char *strget)
{
    // 解析 URL 中的参数。
    // 权限控制：用户名和密码。
    // 接口名：访问数据的种类。
    // 查询条件：设计接口的时候决定。
    //
http://127.0.0.1:8080/api?wucz&wuczpwd&getZHOBTMIND1&51076&20211024094318&202
11024114020
    //
http://127.0.0.1:8080/api?username=wucz&passwd=wuczpwd&intetname=getZHOBTMIND1&
obtid=51076&begintime=20211024094318&endtime=20211024114020

    char username[31],passwd[31],internname[30],obtid[11],begintime[21],endtime[21];
    memset(username,0,sizeof(username));

    // 类似于解析 xml 的函数，这个是解析 get 的
    getvalue(strget,"username",username,30);       // 获取用户名。
.............

    printf("username=%s\n",username);
```

```
.............

   // 判断用户名/密码和接口名是否合法。

   // 连接数据库。
   connection conn;
   conn.connecttodb("scott/tiger@snorcl11g_132","Simplified Chinese_China.AL32UTF8");

   // 准备查询数据的 SQL。
   sqlstatement stmt(&conn);
   stmt.prepare("select  '<obtid>'||obtid||'</obtid>'||'<ddatetime>'||to_char(ddatetime,'yyyy-mm-dd
hh24:mi:ss')||'</ddatetime>'||'<t>'||t||'</t>'||'<p>'||p||'</p>'||'<u>'||u||'</u>'||'<keyid>'||keyid||'</keyid
>'||'<endl/>' from T_ZHOBTMIND1 where obtid=:1 and
ddatetime>to_date(:2,'yyyymmddhh24miss') and ddatetime<to_date(:3,'yyyymmddhh24miss')");
   char strxml[1001];   // 存放 SQL 语句的结果集。
   stmt.bindout(1,strxml,1000);
   stmt.bindin(1,obtid,10);
   stmt.bindin(2,begintime,14);
   stmt.bindin(3,endtime,14);

   stmt.execute();     // 执行查询数据的 SQL。

   Writen(sockfd,"<data>\n",strlen("<data>\n"));          // 返回 xml 的头部标签。

   while (true)
   {
      memset(strxml,0,sizeof(strxml));
      if (stmt.next()!=0) break;

      strcat(strxml,"\n");                                  // 注意加上换行符。
      Writen(sockfd,strxml,strlen(strxml));                // 返回 xml 的每一行。
   }

   Writen(sockfd,"</data>\n",strlen("</data>\n"));   // 返回 xml 的尾部标签。

   return true;
}
```