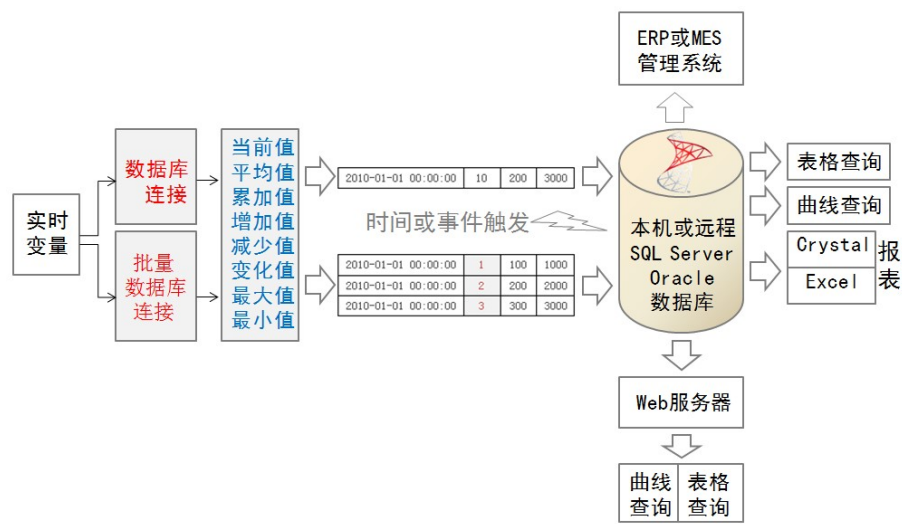


24. 批量数据连接

序号	内容	页码
24.1	编辑批量数据连接	24-2
24.2	应用批量数据连接	24-12
24.3	启动批量数据连接	24-12
24.4	编程触发批量数据连接	24-13

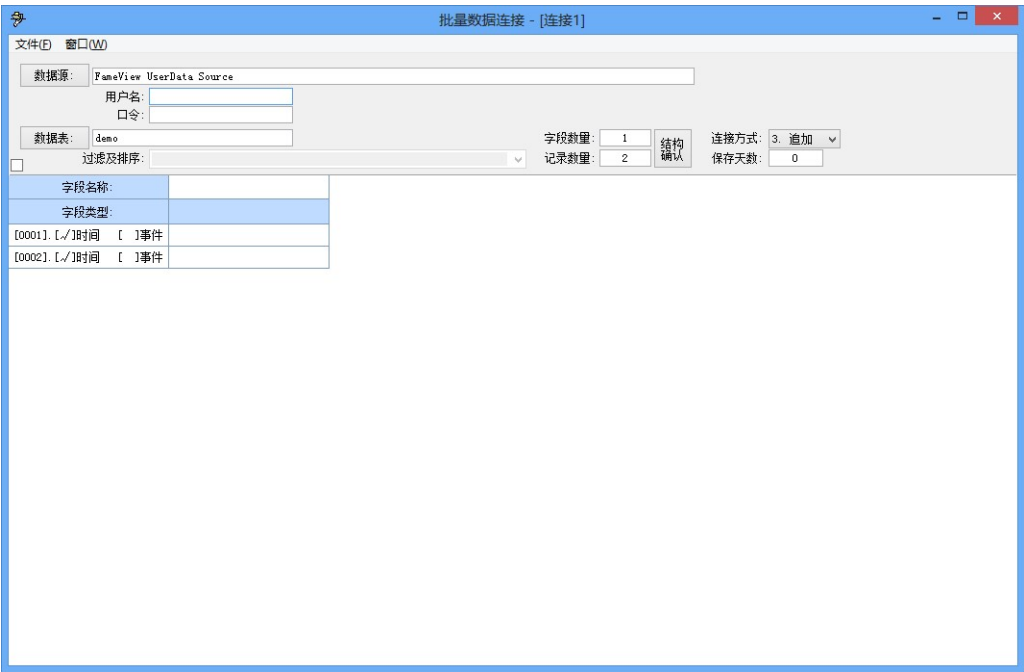


24.1 编辑批量数据连接

- 选择<批量数据库连接>功能, 执行<1. 编辑批量连接>任务, 新建连接文件:



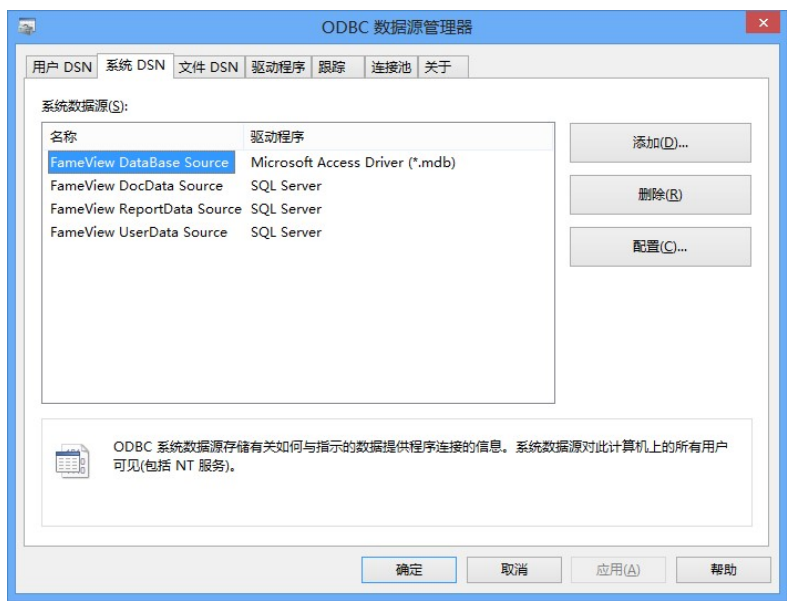
- 数据库连接文件内容:



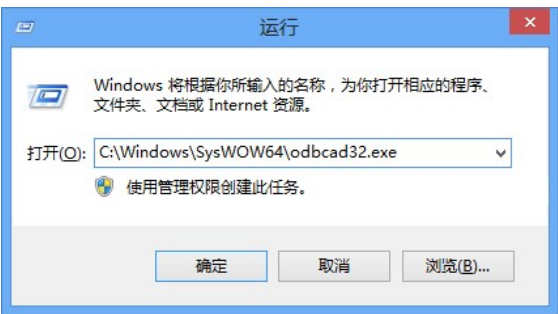
- 选择数据源
执行<数据源>按钮, 选择已定义的数据源:



32 位 Windows 系统, 进入“控制面板->管理工具”, 定义系统 DSN 数据源:



64 位 Windows 系统, 需执行命令” C:\Windows\SysWOW64\odbcad32.exe”, 定义 32 位数据源:



不基于数据库建立水晶报表, 建议执行<自定义数据源>按钮, 使用 DSN 描述型数据源:



数据库类型	数据源描述
SQL Server	Driver={SQL Server};Server=(local);Database=UserDatabase;Uid=;Pwd=;
Access	Driver={Microsoft Access Driver (*.mdb)};Dbq=c:\mydb.mdb;Uid=;Pwd=;
Oracle	Driver={Microsoft ODBC for Oracle};Server=192.168.1.100;Uid=;Pwd=;

支持备用数据源, 主数据源连接失败切换到备用数据源, 格式:Source1|Source2, 之间用竖线分隔;

- 选择连接方式:更新、读取、追加;

连接方式: 3. 追加

保存天数: 1

保存天数:追加方式且保存天数大于 0 时, 根据所关联时间字段进行定时删除;

- 执行<数据表>按钮, 选择数据表或视图:

选择数据表(表名不能含中文)...

demo

确定 取消

连接方式为更新或读取时, 允许设定数据表内容的过滤与排序:

过滤及排序: where dt>DateAdd(MINUTE,-1,GetDate())

数据表须建立标识唯一性字段;

数据表名支持动态时间格式:xxxxxx_%Y%m%d, 其中%Y(年) %m(月) %d(日);

- 数据表结构: 最大字段数 300, 最大记录数 3000, 且(字段数*记录数)<=320000:

字段数量: 4

记录数量: 3

结构确认

- 执行[结构确认]按钮, 连接表样式与数据表结构进行匹配:

批量数据连接 - [连接1]

文件(F) 窗口(W)

数据源: FaneView UserData Source

用户名:

口令:

数据表: demo

字段数量: 4

记录数量: 3

结构确认

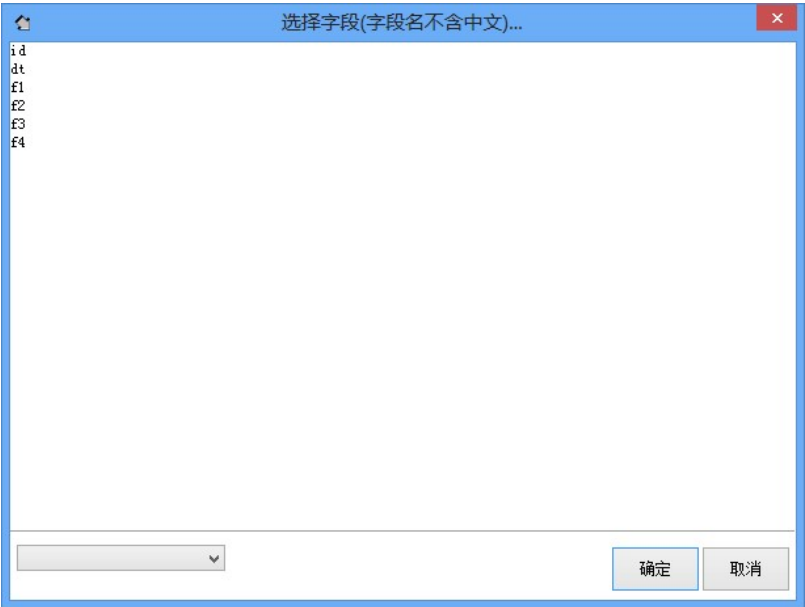
连接方式: 3. 追加

保存天数: 0

☐ 过滤及排序:

字段名称:				
字段类型:				
[0001]. [/]时间 []事件				
[0002]. [/]时间 []事件				
[0003]. [/]时间 []事件				

□ 鼠标双击表格的字段名称, 从数据表或视图中选择字段:



非法字段不能够被选择:

非法字段名称:	
1	使用关键词, 如“datetime”、“date”、“time”、“id”、“int”、“real”等
2	以数字作为开头字符
3	包含汉字或宽字符
4	包含特殊字符, 如'.', '[', ']', '(', ')', '{', '}', '+', '-', '*', '@' 等

特殊情况, 按下 Shift 键, 允许选择非法字段, 但不保证数据表正常操作;

浮点类型的字段, 允许选择存储时的小数位数 (1-5):

Real. 2 - 保留2位小数

□ 选择字段名称完成:

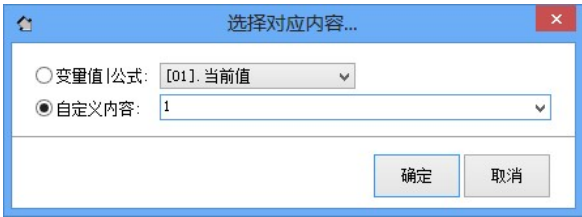
数据表:	demo	字段数量:	4	结构	连接方式:	3. 追加
	过滤及排序:		记录数量:	3	保存天数:	0
		结构	确认			
字段名称:	1. dt	2. f1	3. f2	4. f3		
字段类型:	[时间]	[Real]	[Real]	[Real]		
[0001]. [/]时间 []事件						
[0002]. [/]时间 []事件						
[0003]. [/]时间 []事件						

□ 鼠标双击 [0001]-[3000] 行字段列, 选择变量与记录字段的对应关系;

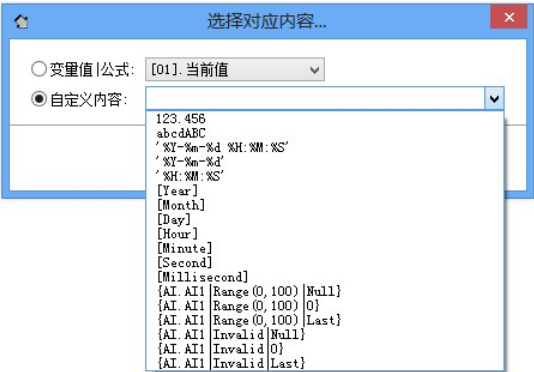
[1]. 时间字段, 使用自定义内容 '%Y-%m-%d %H:%M:%S', 获取系统当前时间:



[2]. 过滤字段, 区分不同记录, 使用自定义内容, 数值字段输入常数, 文本字段输入字符串:

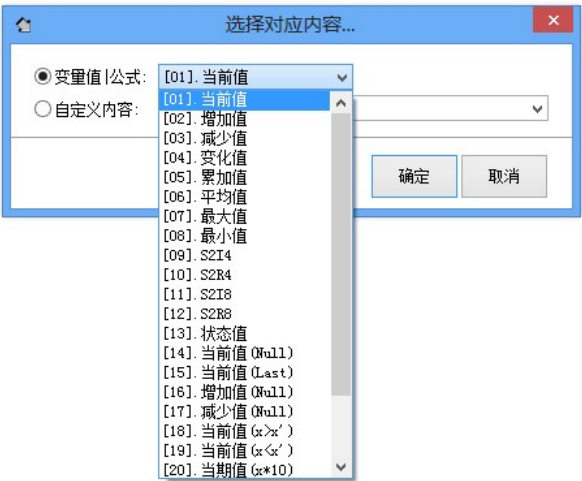


[3]. 自定义字段:



序号	表达式	描述
1	123.456	固定数值
2	abcdABC	固定文本
3	'%Y-%m-%d %H:%M:%S'	当前日期+时间
4	'%Y-%m-%d'	当前日期
5	'%H:%M:%S'	当前时间
6	[Year]	年
7	[Month]	月
8	[Day]	日
9	[Hour]	时
10	[Minute]	分
11	[Second]	秒
12	[Millisecond]	毫秒
13	[THours]	距 2000-1-1 总小时数
14	[TDays]	距 2000-1-1 总天数
15	[NULL]	空字段
16	{VA. %VA1 Range(0, 100) Null}	变量值不在某范围之内时, 取空值
17	{VA. %VA1 Range(0, 100) 0}	变量值不在某范围之内时, 取固定值
18	{VA. %VA1 Range(0, 100) Last}	变量值不在某范围之内时, 取上次有效值
19	{AI. AI01 Invalid Null}	通讯中断, 变量值无效时, 取空值
20	{AI. AI01 Invalid 0}	通讯中断, 变量值无效时, 取固定值
21	{AI. AI01 Invalid Last}	通讯中断, 变量值无效时, 取上次有效值

[4]. 变量字段, 使字段内容对应某变量, 并对变量值进行某种处理:



序号	公式	描述
1	当前值	数值不处理;
2	增加值	相临两次存储间隔的增加值;
3	减少值	相临两次存储间隔的减少值;
4	变化值	相临两次存储间隔的变化值;
5	累加值	以 1 秒为间隔, 计算相临两次存储间隔的累加值;
6	平均值	以 1 秒为间隔, 计算相临两次存储间隔的平均值;
7	最大值	以 1 秒为间隔, 计算相临两次存储间隔的最大值;
8	最小值	以 1 秒为间隔, 计算相临两次存储间隔的最小值;
9	S2I4	把文本转换为 32 位整数;
10	S2R4	把文本转换为 32 位浮点数;
11	S2I8	把文本转换为 64 位整数;
12	S2R8	把文本转换为 64 位浮点数;
13	状态值	变量只有有效状态 (0/1)
14	当前值 (Null)	变量值通讯中断无效时, 存储空值;
15	当前值 (Last)	变量值通讯中断无效时, 存储上次有效值;
16	增加值 (Null)	变量值通讯中断无效时, 存储空值;
17	减少值 (Null)	变量值通讯中断无效时, 存储空值;
18	x>x'	变量值大于上次值有效, 否则取上次值;
19	x<x'	变量值小于上次值有效, 否则取上次值;
20	x*10	变量值 x*10 运算;
21	x*100	变量值 x*100 运算;
22	x/10	变量值 x/10 运算;
23	x/100	变量值 x/100 运算;

□ 定义完成后表格如下：

批量数据连接 - [连接1]

文件(F) 窗口(W)

数据源: FaneView UserData Source

用户名:

口令:

数据表: demo

过滤及排序:

字段数量: 4

记录数量: 3

结构确认

连接方式: 3 追加

保存天数: 0

字段名称:	1. dt	2. f1	3. f2	4. f3
字段类型:	[时间]	[Real]	[Real]	[Real]
[0001]. [✓]时间 []事件	'%Y-%m-%d %H:%M:%S'	1	AI. P1_AI1[当前值]	AI. P1_AI2[当前值]
[0002]. [✓]时间 []事件	'%Y-%m-%d %H:%M:%S'	2	AI. P2_AI1[当前值]	AI. P2_AI2[当前值]
[0003]. [✓]时间 []事件	'%Y-%m-%d %H:%M:%S'	3	AI. P3_AI1[当前值]	AI. P3_AI3[当前值]

□ 双击 [0001]~[3000]行第 1 列, 设置每条记录或全部记录的存储触发条件:

触发方式[精度>100ms]...

☐ 时间触发

起始时间: 00:00:00

时间间隔: 60 秒

☐ 变量事件触发

事件变量:

事件类型: (01). X+- [任何变化]

延迟存储周期 (0-100): 0

☐ [全部修改]

确定

取消

➤ 选择时间触发

[1]. 每隔 10 秒执行 1 次;

☒ 时间触发

起始时间: 00:00:00

时间间隔: 10 秒

[2]. 每小时 0、10、20、30、40、50 分各工作 1 次:

☒ 时间触发

起始时间: 00:00:00

时间间隔: 10 分钟

[3]. 每天整点各工作 1 次;

☒ 时间触发

起始时间: 00:00:00

时间间隔: 1 小时

[4]. 启动 30 秒后工作 1 次:

☒ 时间触发

起始时间: 00:00:00

时间间隔: 30 秒单次

[5]. 追加方式, 变量值发生任何变化存储 1 次:

☒ 时间触发

起始时间: 00:00:00

时间间隔: 1 | 值变化

➤ 选择变量事件触发

触发方式[精度>100ms]...

☐ 时间触发

起始时间: 00:00:00

时间间隔: 60 秒

☒ 变量事件触发

Or And

事件变量: VD. %VD1

事件类型: (01). X+- [任何变化]

☐ [全部修改]

(01). X+- [任何变化]

(02). X++ [增加 [正跳变]]

(03). X-- [减少 [负跳变]]

(04). X->0 [跳变为0]

(05). X->1 [跳变为1]

(06). X=0 [等于0]

(07). X=1 [等于1]

(08). X>0 [大于0]

(09). X<0 [小于0]

(10). X>0 -> X=0 [大于0则置0]

(11). X<0 -> X=0 [小于0则置0]

(12). X!=0 [不等于0]

(13). X!=1 [不等于1]

延迟周期 (n) 能够过滤变量干扰:

n>0, 触发条件满足时不立刻触发执行, 等候 n*100 毫秒后, 触发条件仍满足则触发执行;

➤ 同时选择时间触发和事件触发, 须进行逻辑判断:

触发方式[精度>100ms]...

☒ 时间触发

起始时间: 00:00:00

时间间隔: 60 秒

☒ 变量事件触发

Or And

事件变量: VD. %VD1

事件类型: (01). X+- [任何变化]

延迟存储周期 (0-100): 0

☐ [全部修改]

确定 取消

➤ 修改全部记录:

触发方式[精度>100ms]...

☒ 时间触发

起始时间: 00:00:00

时间间隔: 60 秒

☐ 变量事件触发

Or And

事件变量:

事件类型: (01). X+- [任何变化]

延迟存储周期 (0-100): 0

☒ [全部修改]

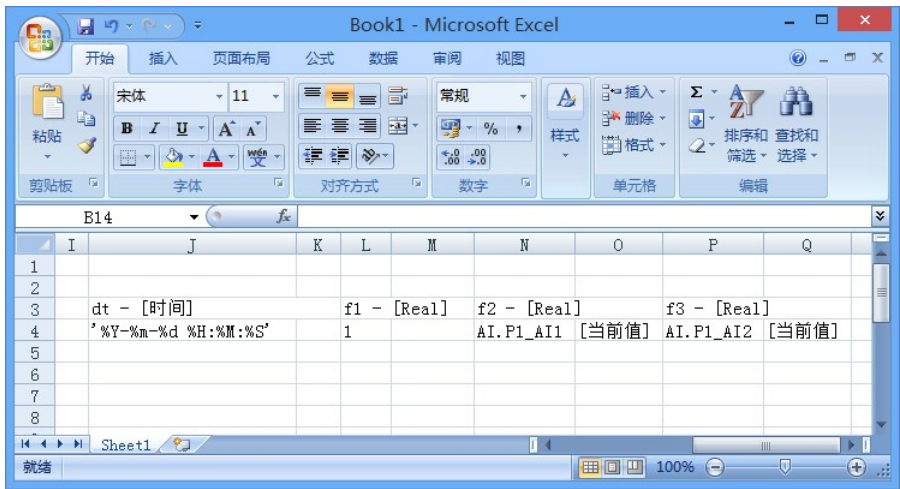
确定 取消

□ 支持 Excel 导出、导入编辑方式；

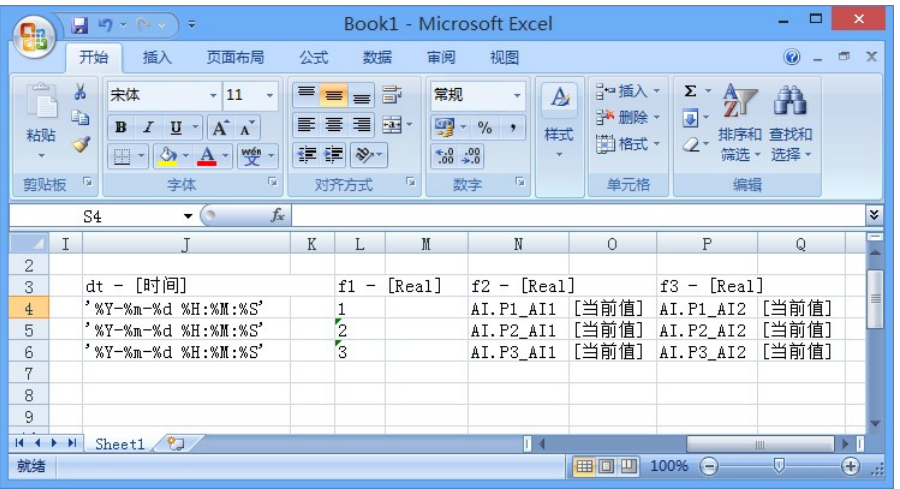
[1]. 建立基本内容



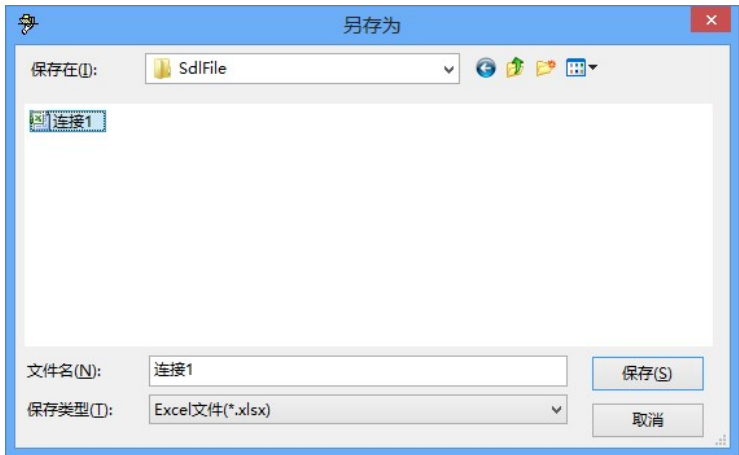
[2]. 执行“文件->Excel 导出”菜单, 导出 Excel 文件:



[3]. 通过 Excel 进行编辑并保存:



[4]. 执行“文件->Excel 导入”菜单：



[5]. 选择 Excel 文件并导入：



□ 替换内容

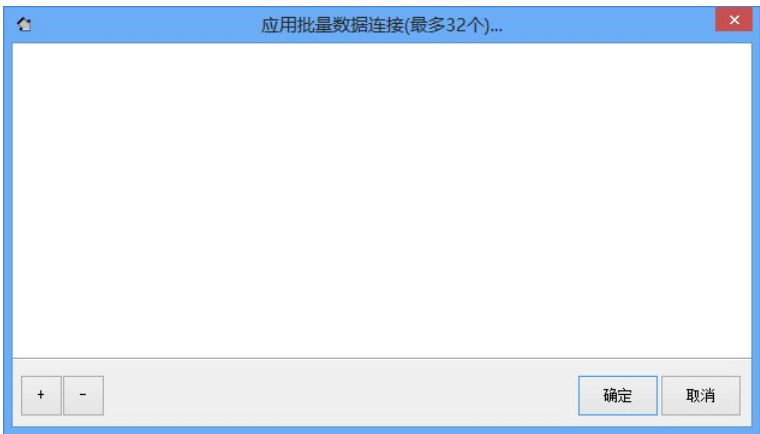
执行“文件->替换内容”，快速把某些文本改变为其他文本：



未选择“检查文本”，只替换变量名称，否则替换全部内容；

24.2 应用批量数据连接

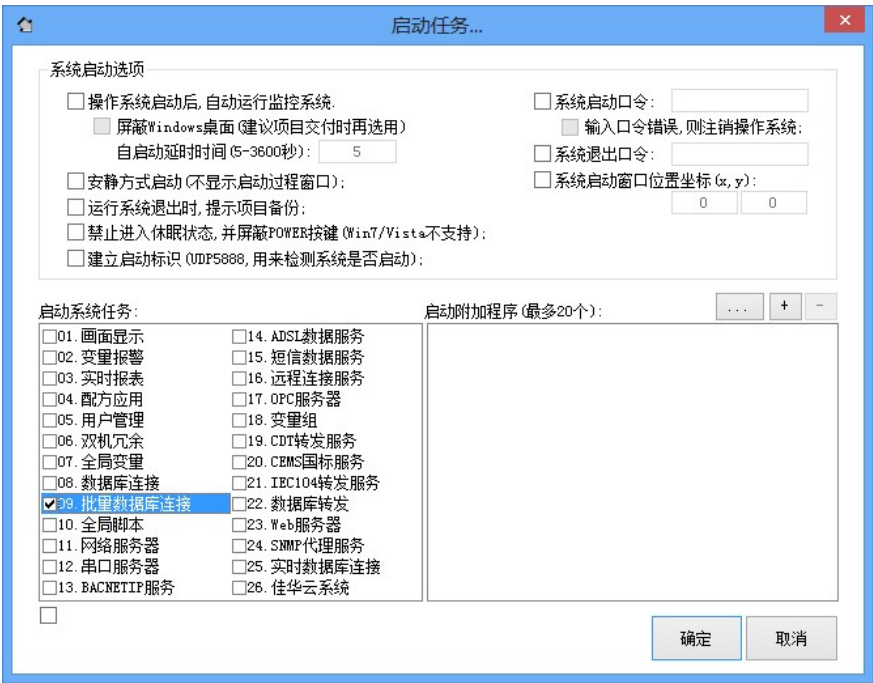
- 选择<批量数据库连接>功能, 执行<2. 应用实时连接>任务:



- 执行“+”按钮, 选择实时连接文件, 添入列表中;
- 最多选择 32 个连接文件:

24.3 启动批量数据连接

- 选择“我的系统->设置”功能, 执行<2. 启动任务>:



- 从“系统任务”列表中选“☒批量数据库连接”;

24.4 编程触发批量数据连接

- 通过脚本或编程触发批量数据库连接, 实现自定义的数据存储或刷新;
- 制作批量数据库连接文件:



- [1]. 需要通过触发内容修改的字段, 随意对应为某变量, 其它字段对应固定内容;
- [2]. 无需设定时间和事件触发;
- [3]. 应用并启动批量数据库连接文件:

- 通过脚本触发批量数据库连接:

RunSys.TriggerBatchDBLink fn,linkIndex,linkContent
 Fn, 批量数据库连接文件名称, 不包含文件后缀;
 linkIndex, 触发记录号(1-3000);
 linkContent, 触发字段内容, 字段值用' | ' 隔开, 最大长度2000个字符;
 字段顺序要与连接文件一致, 固定字段内容输入空格;
 例: RunSys.TriggerBatchDBLink "连接1", 2, " | | 123. 45|345. 12 | "

- 编写程序触发批量数据库连接, 以VC++为例:

```
int TriggerBatchDBLink(CString linkName, int linkIndex, CString linkContent)
{
    //检查输入参数
    if(linkName=="") return 1;
    if(linkIndex<0 || linkIndex>3000) return 2;
    if(linkContent.GetLength()<3 || linkContent.GetLength()>2000) return 3;
    if((linkContent.GetAt(0)!=' | ') return 4;

    //得到批量数据库连接对象
    HANDLE hSdlMutex::OpenMutex(MUTEX_ALL_ACCESS, TRUE, "FameView_SDL_EXE");
    if(hSdlMutex==NULL) 5;
    DWORD dwReturn=WaitForSingleObject(hSdlMutex, 8000);
    if(dwReturn!=WAIT_OBJECT_0 && dwReturn!=WAIT_ABANDONED) {
        CloseHandle(hSdlMutex);
        return 6;
    }
}
```

```

//设置最大长度
CString s=linkContent.Mid(1);
int totalLength=s.GetLength()+110;

int iValue=0;
CWnd* pPrevWnd=CWnd::GetDesktopWindow()->GetWindow(GW_CHILD);
while(pPrevWnd && iValue<=500) {
    if(::GetProp(pPrevWnd->GetSafeHwnd(),"FameView_SDL_EXE")) {
        BYTE* Buffer=new BYTE[totalLength];
        memset(Buffer,0,totalLength);
        COPYDATASTRUCT cds;
        cds.dwData=10001;
        cds.cbData=totalLength;
        memcpy(&Buffer[0],(linkName,(linkName.GetLength()));
        Buffer[91]=linkIndex/0x100;
        Buffer[92]=linkIndex&0xFF;
        memcpy(&Buffer[100],s,s.GetLength());
        cds.lpData=Buffer;
        ::SendMessage(pPrevWnd->m_hWnd,WM_COPYDATA,NULL,(LPARAM)&cds);
        delete[] Buffer;
        break;
    }
    if(::IsWindow(pPrevWnd->m_hWnd)==FALSE) break;
    pPrevWnd=pPrevWnd->GetWindow(GW_HWNDNEXT);
    iValue++;
}

ReleaseMutex(hSdlMutex);
CloseHandle(hSdlMutex);

return 0;
}

```