

27. 实时数据库

序号	内容	页码
27.1	关系数据库优缺点	27-2
27.2	实时数据库存储模式	27-2
27.3	实时数据库结构	27-3
27.4	实时数据库选型	27-3
27.5	安装实时数据库	27-4
27.6	实时数据库连接	27-9
27.7	实时数据库Excel报表	27-19
27.8	实时数据库导出到关系数据库	27-23
27.9	实时数据库数据查询	27-28
27.10	使用脚本访问实时数据库	27-29
27.11	使用API开发包访问实时数据库	27-30
27.12	远程升级实时库加密狗	27-40
27.13	常见错误代码	

FameHistory实时数据库

稳定、实时、并发、海量、高速



27.1 关系型数据库优缺点

关系数据库, 典型存储模式:

id	时间字段	设备名	字段 1	字段 2	字段 3	字段 4
1	2012-12-21 0:0:0	"1#"	12	13	14	15	...
2	2012-12-21 1:0:0		120	130	140	150	...
3	2012-12-21 0:0:0	"2#"	22	23	24	25	...
4	2012-12-21 1:0:0		220	230	240	250	...
5	2012-12-21 0:0:0	"3#"	32	33	34	35	...
6	2012-12-21 1:0:0		320	330	340	350	...

- 关系: 处理永久稳定的数据(保证数据完整性、一致性);
- 横表存储, 适合报表分析;
- 提供索引, 实现快速查询;
- 因为索引, 导致大批量数据存储缓慢;
- 单客户连接, 不支持并发存储;
- 数据无压缩, 占用大量硬盘空间;

经验数据:
数据库连接, 800条/秒
1万点DOC历史数据库, 5000条/分钟

假设有100000个变量,
需要每秒存储1次!
可能吗? !

27.2 实时数据库存储模式

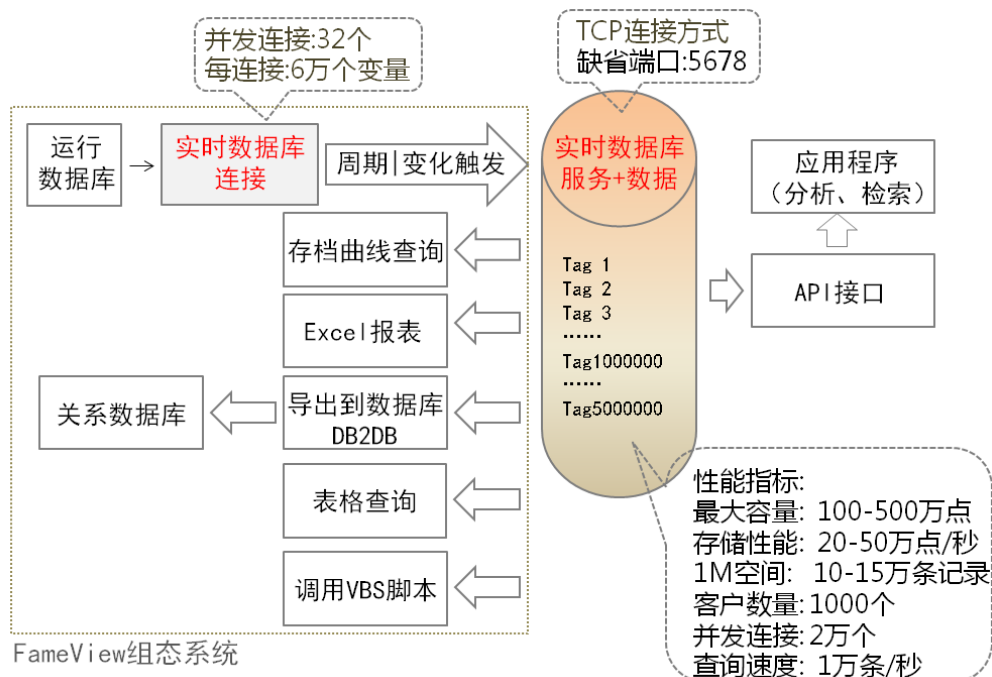
硬盘						
文件 1				文件 2	文件 m
标签 1	标签 2	...	标签 n			
101.00	201.00		1234.56			
...			
108.00	208.00		5678.12			

标签 1		标签 2		标签 3		...	标签 5000000	
ID: 000001		ID: 000002		ID: 000003			ID: 5000000	
时间	数值	时间	数值	时间	数值		时间	数值
1358611200	100	1358611200	200	1358611200	300		1358611200	1000
1358611201	101	1358611201	201	1358611201	301		1358611201	1001
1358611202	102	1358611202	202	1358611202	302		1358611202	1002
1358611203	103	1358611203	203	1358611203	303		1358611203	1003
--	--	--	--	--	--		--	--

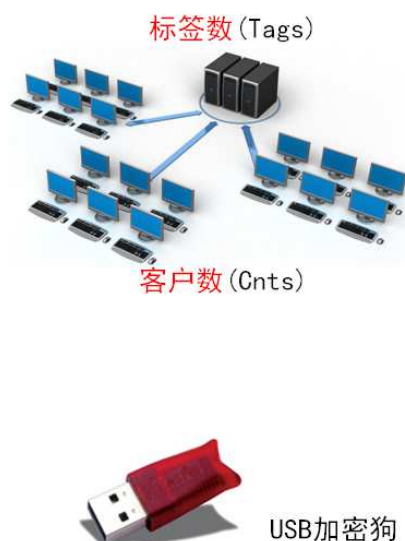
- 实时, 处理实时变化数据, 保证数据实时性、真实性;
- 竖表存储, 适合单变量快速查询(变量曲线);
- ID号索引, 能快速存储;
- 支持并发, 能实现大批量数据存储;
- 无损、有损和二级压缩, 普通硬盘能存储几年数据
- 不易报表分析查询;

单机测试数据:
普通计算机, 2G内存, Windows XP (SP3)
1个单独连接: 60000个标签/秒
4个并发连接: 240000个标签/秒

27.3 实时数据库结构(FameHistory)



27.4 实时数据库选型(FameHistory)



实时数据库服务器	
根据标签数量分类(Tags):	
第1档	100, 300, 500
第2档	1K, 3K, 4K, 5K, 6K
第3档	10K, 15K, 20K, 30K, 40K, 50K, 75K
第4档	100K, 150K, 250K, 500K
第5档	1000K, 5000K
实时数据库客户端	
根据客户端数量分类(Cnts):	
第1档	5, 10, 25, 50
第2档	100, 500, 1000
第3档	1000, 5000
第4档	10000, 20000
订货号	
FameHistory-Tags-xxxxxx-Cnts-xxxx	
标签数	客户端数
注1: 服务器缺省包含2个客户端, 超出2个客户端时, 需购买客户端;	
注2: 每客户端允许的最大连接数为32;	
例1 (3000点标签, 5客户端): FameHistory-Tags-0003000-Cnts-00005	
例2 (3000点标签, 2客户端): FameHistory-Tags-0003000-Cnts-00000	
例3 (8000点标签, 50客户端): FameHistory-Tags-0008000-Cnts-00050	

27.5 安装实时数据库 (FameHistory)

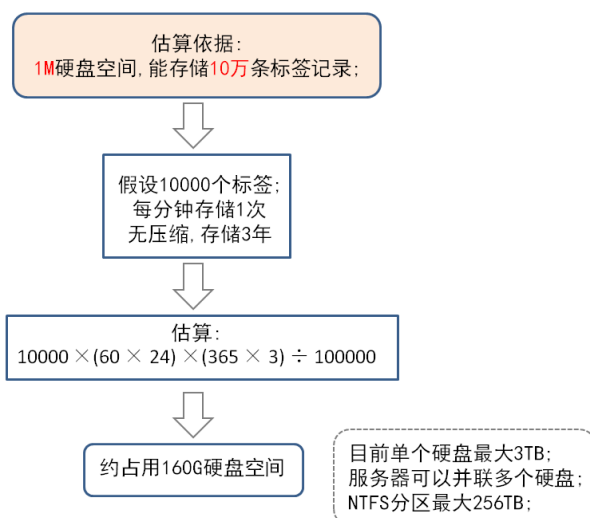
- 安装环境需求:

	标签数较少(<10000 点)	标签数较多(>=10000)
硬件平台	工作站, 7200 转硬盘 双核 CPU 内存 2G	服务器, 7200 转硬盘 至少四核 CPU 内存至少 4G, 建议 16G
网络带宽	百兆带宽	千兆带宽
软件平台	Windows XP, Windows 7 Windows Server 2008 (32/64 位)	Windows Server 2008 (64 位)
其他	可与组态同机运行	建议独立运行

- 通过 Setup.ini 文件定制安装:



- 估算硬盘空间:

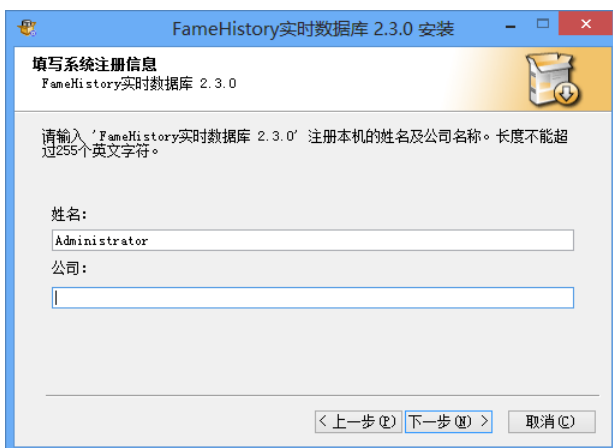


- 安装实时数据库前, 建议先临时关闭杀毒软件;
- 已购买加密狗, 安装前在 USB 口插入加密狗;

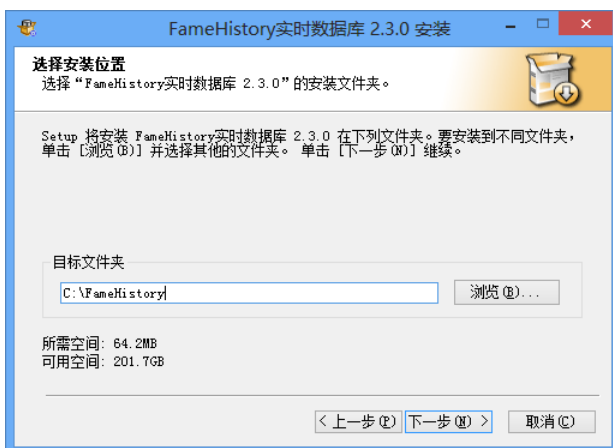
- 安装程序包括服务器和客户端, 缺省只安装服务器;
需要对服务器进行更多管理时(如更改连接口令), 则需要安装客户端(FameHistory_Client.exe);
- 执行 FameHistory_Server.exe 文件, 显示安装界面:



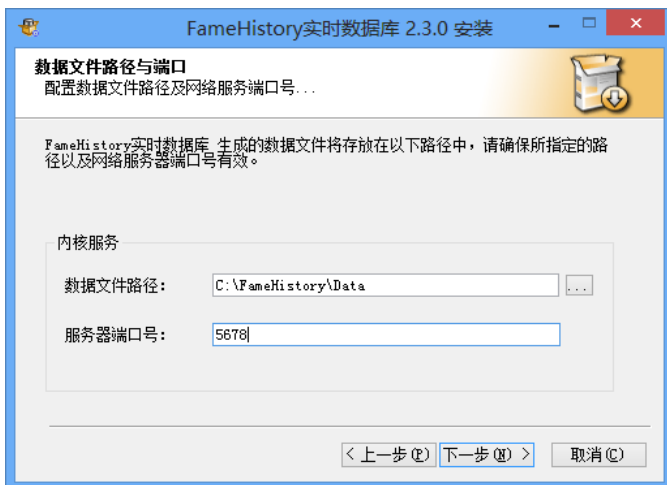
- 执行下一步, 填写注册信息:



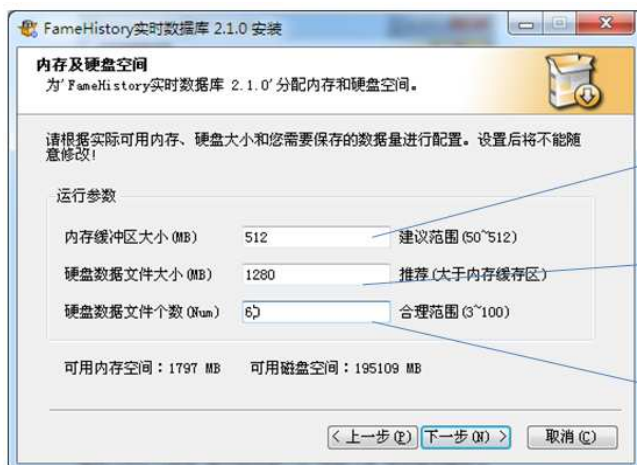
- 执行下一步, 选择运行环境安装目录, 缺省为 C:\FameHistory:



- 执行下一步, 选择数据文件存储路径, 可不与运行环境相同目录, 服务器 TCP 端口, 缺省为 5678:



- 执行下一步, 设置运行参数, 这步很重要:



32位系统: 512M
64位系统: 1024M

32位系统: 1280M
64位系统: 2560M

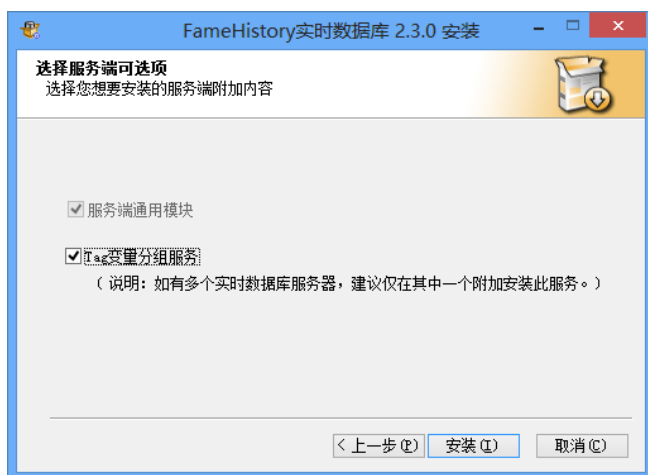
根据估算所占用硬盘空间
 $N = M \div M1$

	32 位系统(建议)	64 位系统(建议)
内存缓冲大小(N1)	512M	1024M
数据文件大小(N2)	1280M	2560M
数据文件个数(N3)	最大占用硬盘空间 $N4 = N2 * N3$; 1M 硬盘空间大概能存储 10 万条标签记录;	

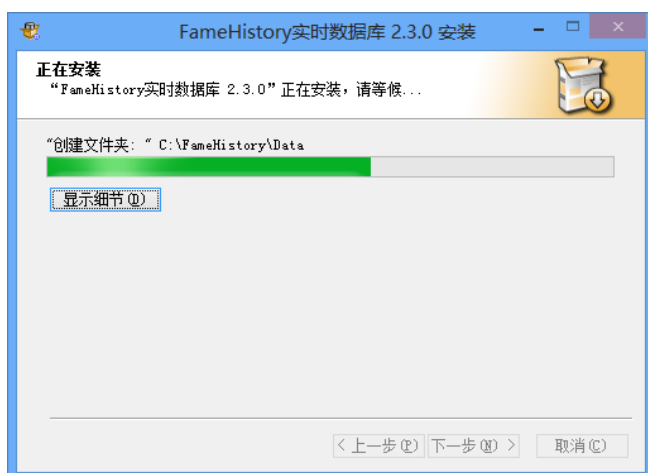
假设 10000 个标签, 每分钟存储 1 次, 存储 3 年, 则估算设置如下:

最大占用硬盘空间(N4)	157680M(约 160G)	估算: $N4 = 10000 * (60 * 24) * (365 * 3) / 100000$
数据文件大小(N2)	2560M	设置: $N2 = 2560M$
数据文件个数(N3)	60 个	计算: $N3 = N4 / N2 = 157680 / 2650$

- 执行下一步, 缺省选项即可:



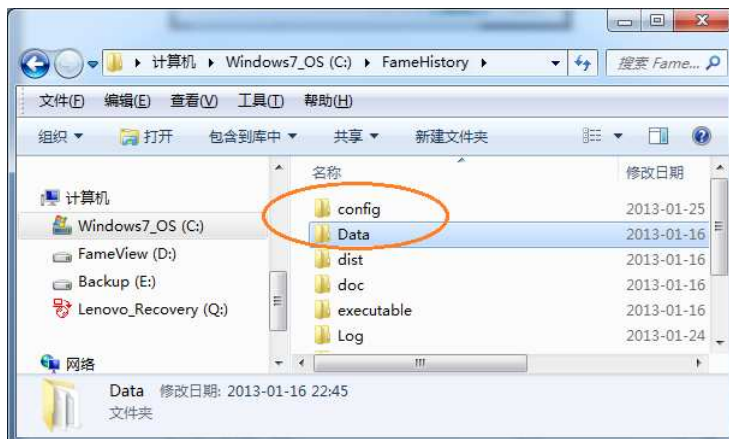
- 开始安装, 大约用时 2 分钟, 如出现安装卡死现象, 尝试关闭所有杀毒软件:



- 最后选择是否以服务方式启动, 演示版本或测试阶段可不安装为服务, 正式运行必须安装为服务:



- 安装完成后, 重新启动计算机, 首次会占用些时间优化存储空间, 有两个重要目录:



[1]. Config 目录, 存放实时数据库组态内容, 如注册标签等;

可备份此目录内容, 把实时数据库标签移植到其他计算机;

尤其要备份 Kernel.db 和 ServerCfg.xml 关键文件;

[2]. Data 目录, 存放实时数据库数据, 备份整个目录内容可移植到其他计算机;

- 如果安装为服务运行方式, 操作系统启动后, 实时数据库服务会自动启动;
- 能够手动停止或启动实时数据库后台服务

方法 1: 执行安装目录下. \ executable\hdServiceMgrTool.exe;

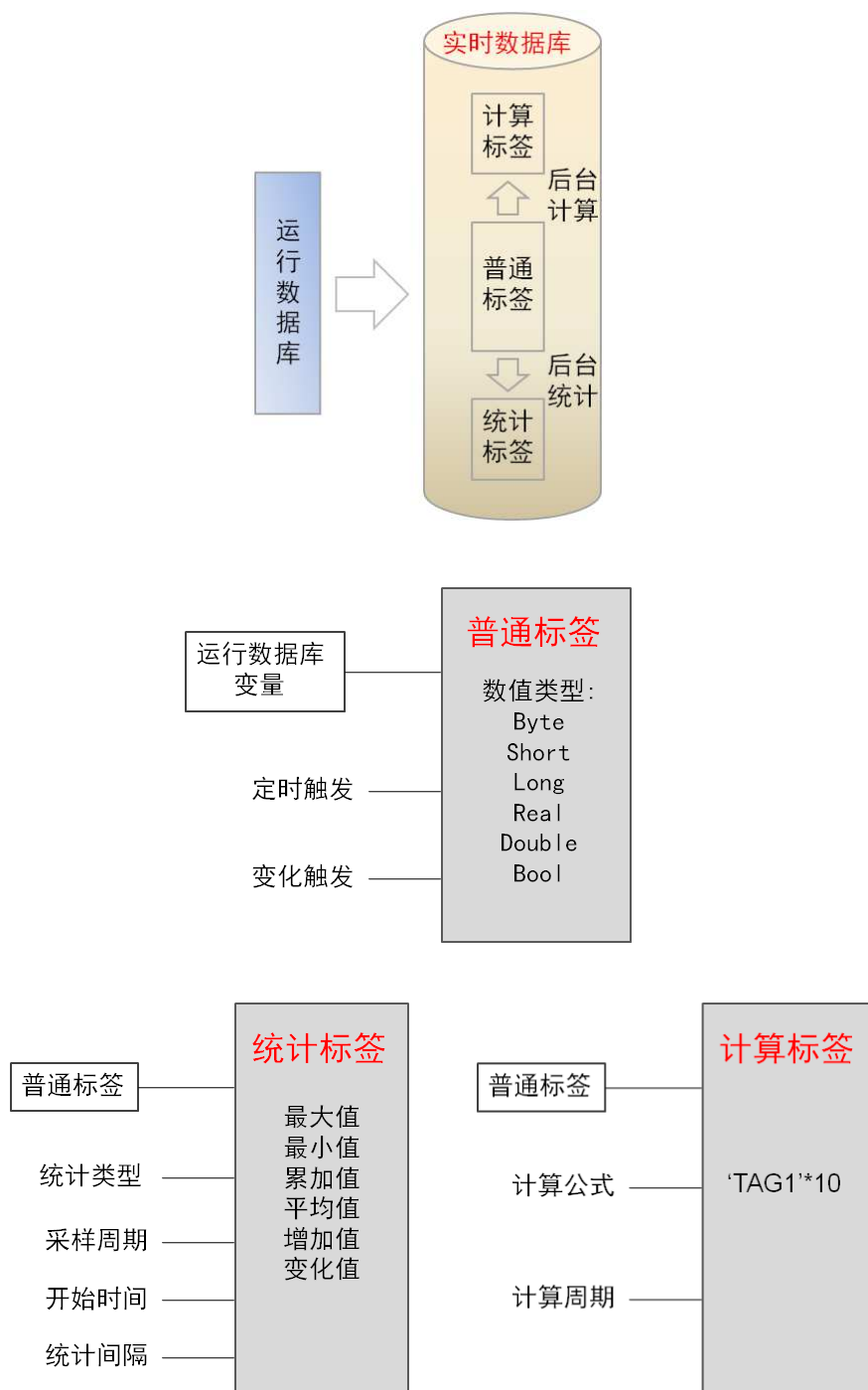
方法 2: 通过组态软件管理器, 实时数据库->客户端工具->本地系统管理工具;



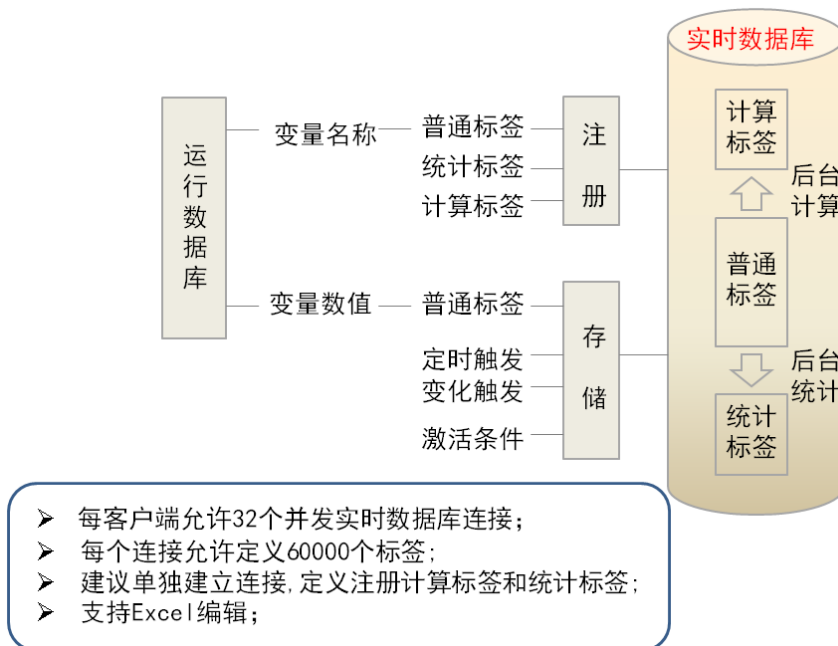
- 实时数据库通过加密狗授权后, 才能正式使用;
- 须在实时数据库运行前插入授权加密狗;
- 无加密狗时以演示方式运行, 演示版本限制: 30000 点标签, 10 客户端, 连续运行两小时;

27.6 实时数据库连接

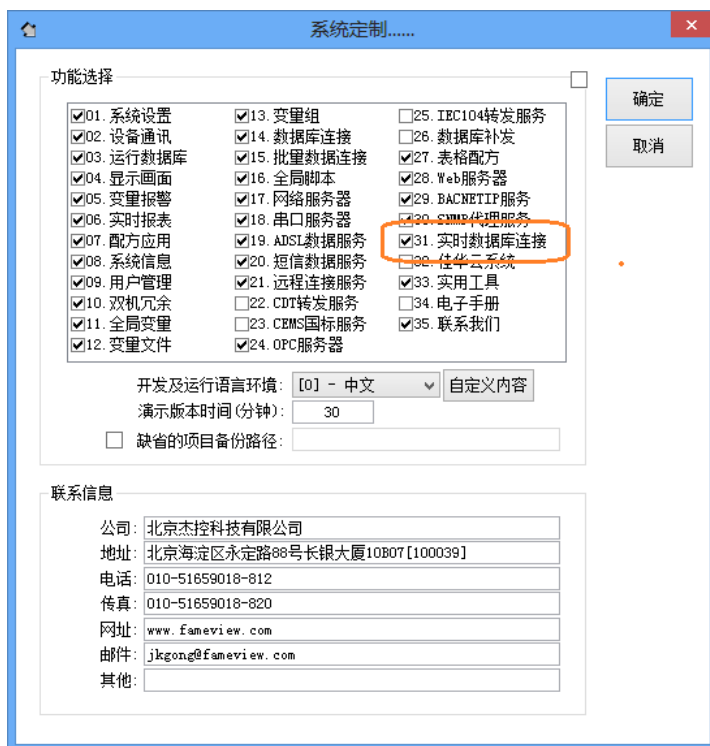
□ 实时数据库标签概念：



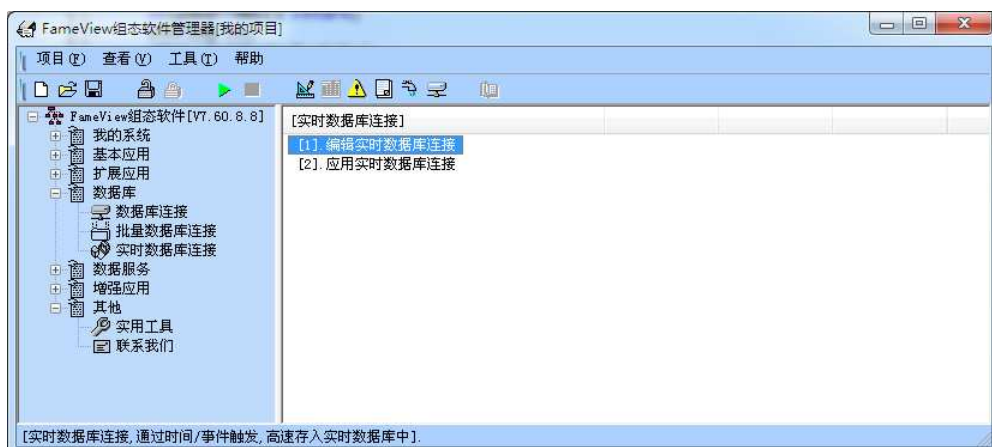
□ 实时数据库连接示意图：



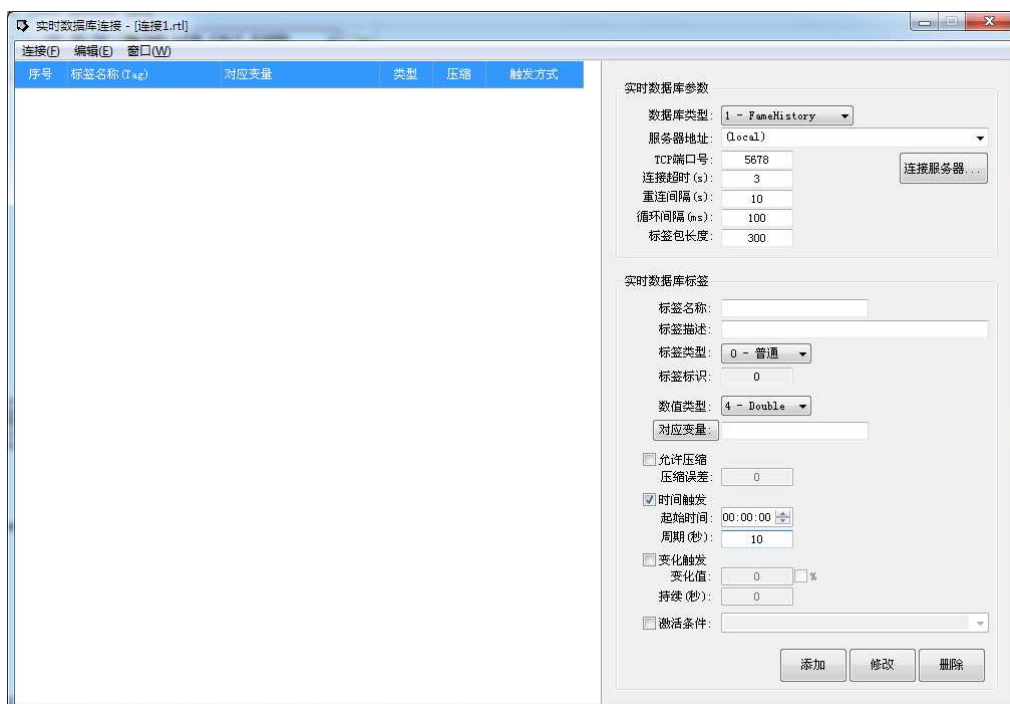
□ 定制实时数据库连接：



- 选择实时数据库连接功能：



- 编辑数据库连接文件：



- 实时数据库参数：

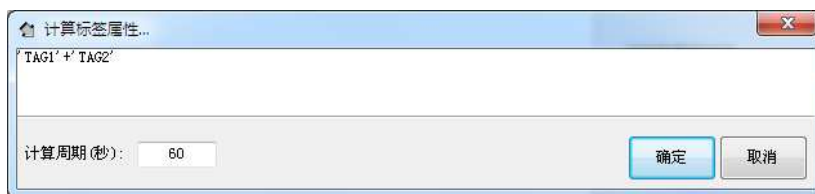
- [1]. 数据库类型, 目前仅支持 FameHistory 实时数据库;
- [2]. 服务器地址, 如果实时数据库服务器在本地, 则填写 '(local)' 即可, 如果服务器位于远程, 则需要填写远程 IP 和登录用户和口令, 如 '192.168.1.100;Uid=admin;Pwd=admin';
- [3]. TCP 端口号, 实时数据库服务指定的 TCP 端口号, 缺省为 5678;
- [4]. 连接超时, 连接实时数据库服务时的超时时间, 取值范围 1-300 秒;
- [5]. 连接间隔, 服务器连接意外断开后, 重新连接要等待的时间间隔, 取值范围 1-600 秒;
- [6]. 轮询间隔, 取值范围 10-3000 毫秒, 即每隔此时间段检查 1 次是否有数据需要存储;
- [7]. 标签包长度, 每次存储的最大标签数量, 取值范围 1-30000 个, 数值越大占用内存越大;

□ 定义实时数据库标签：

- [1]. 标签名称(TagName), 实时数据库由多个标签组成, 每隔标签对应唯一标签标识(TagID); 如要支持通过 DOC 存储变量查询, 标签名称与 DOC 变量名称一致;
- [2]. 标签描述(TagDesc), 对标签的进一步描述;
- [3]. 标签类型(TagType), 标签分为普通标签、统计标签、计算标签;
- [4]. 普通标签可以对应某个运行数据库变量;
- [5]. 统计标签的内容是对普通标签某时间段数据进行某种统计后的结果, 组态界面如下:



- [6]. 计算标签的内容是对某些普通标签计算后的结果, 组态界面如下:



- [7]. 标签标识(TagID), 标签标识不允许填写, 可在正式运行前, 通过<连接服务器>获得;
- [8]. 对应变量的, 标签所对应的运行数据库变量(AI/AO/AR/DI/DO/DR/VT);
- [9]. 数值类型, 标签的数值类型(Byte/Short/Long/Real/Double/Bool/String);
- [10]. 激活条件, 只有满足设定的激活条件才允许存储, 激活条件得表达式可以是:

$$DI, DI1=1, DI, DI1=0, AI, AI1=10, AI, AI1>10, AI, AI1=AI, AI2$$
- [11]. 时间触发, 通过<开始时间>和<触发周期>定义在每天某些时刻定期存储数据;
 如果周期为 0, 则循环存储,
- [12]. 变化触发, 对应变量的值发生了某种变化, 则进行存储;
 - (1). 变化值为 0, 当变量值发生任何变化, 则进行存储, 即:

$$Abs(x-x') > 0$$
 - (2). 变化值不为 0 且没选择<%, 当变量变化绝对值大于等于设定值时, 则进行存储, 即:

$$Abs(x-x') \geq n$$
 - (3). 变化值不为 0 且选择了<%, 当变量变化绝对值大于等于百分比设定值时, 则进行存储:

$$Abs(x-x') \geq n * x' / 100$$
 持续时间, 如果在某持续时间段内, 变量值未发生任何变化, 则到达持续时间时也进行存储, 取值范围为 0-86400 秒, 为 0 时此参数无效;
- [13]. 通过<添加>、<修改>、<删除>按钮, 编辑标签列表;

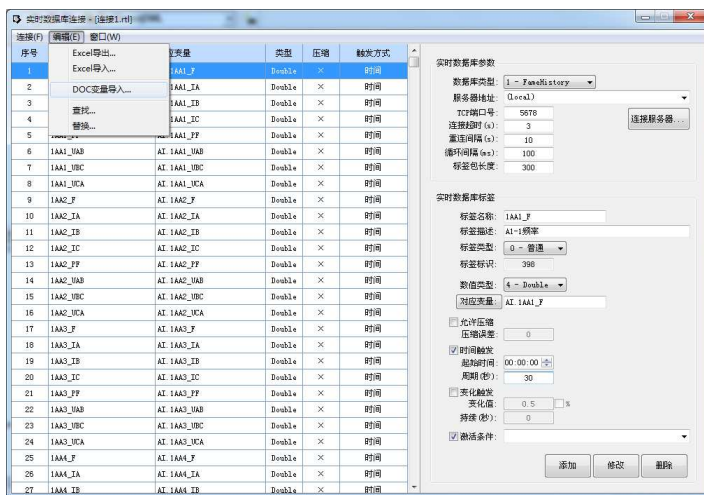
[14]. 标签列表最多支持 60000 个标签;

[15]. 使用菜单<Excel 导出>和<Excel 导入>, 可以通过 Excel 进行高效编辑标签列表;

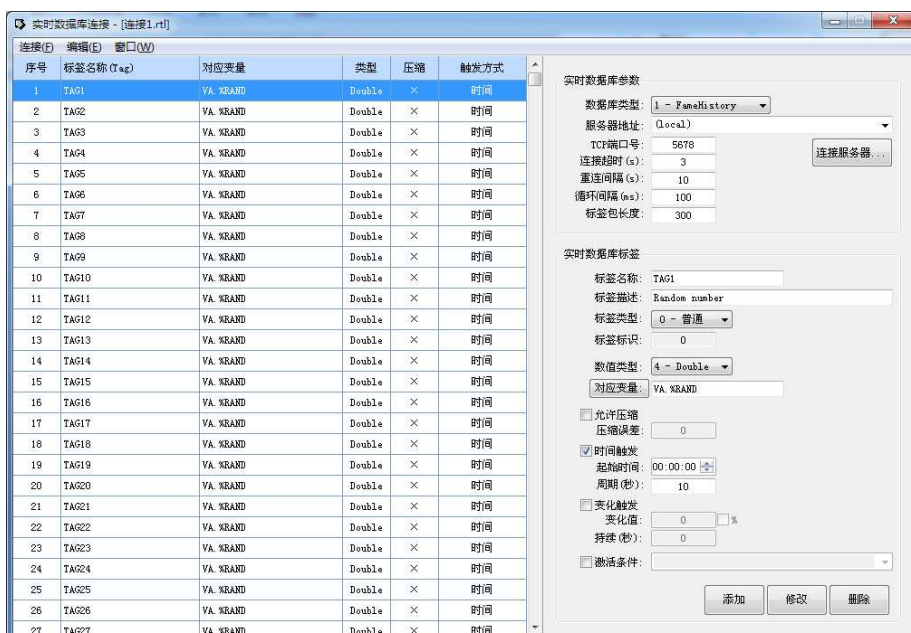


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	标签名称	标签描述	对应变量	数据类型	工程单位	激活条件	允许压缩	压缩误差	时间触发	起始时间	触发周期	变化触发	变化值	%	持续时间
2	V000001		AA.AB000001	double			0	0	1	00:00:00	10	0	0	0	0
3	V000002		AA.AB000002	double			0	0	1	00:00:00	10	0	0	0	0
4	V000003		AA.AB000003	double			0	0	1	00:00:00	10	0	0	0	0
5	V000004		AA.AB000004	double			0	0	1	00:00:00	10	0	0	0	0
6	V000005		AA.AB000005	double			0	0	1	00:00:00	10	0	0	0	0
7	V000006		AA.AB000006	double			0	0	1	00:00:00	10	0	0	0	0
8	V000007		AA.AB000007	double			0	0	1	00:00:00	10	0	0	0	0
9	V000008		AA.AB000008	double			0	0	1	00:00:00	10	0	0	0	0
10	V000009		AA.AB000009	double			0	0	1	00:00:00	10	0	0	0	0

[16]. 实时数据库类似 DOC 历史数据库, 使用菜单<DOC 变量导入>命令, 可以从 DOC 变量获取标签:



□ 编辑完成的界面如下:



- 使用编辑菜单下的替换命令, 快速批量替换表中的内容:



- 使用编辑菜单下的查找命令, 快速查找某个标签名称:



- 注册标签, 把所做的标签名注册到实时数据库服务器, 并获得标签标识(TagId):

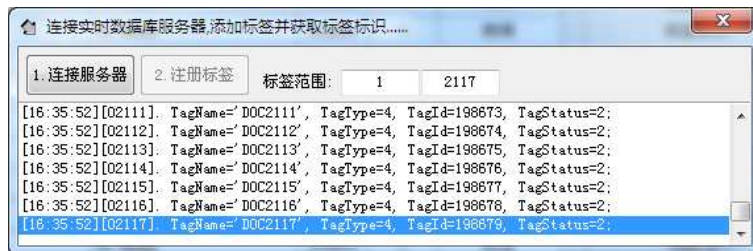
- [1]. 确信已启动并能正确连接到实时数据库服务;
- [2]. 执行<连接服务器...>按钮, 显示下面界面:



- [3]. 执行<连接服务器>按钮:



[4]. 如果显示连接服务器成功, 则可以再执行<注册标签>按钮, 从而注册并取得标签标识(TagId):



[5]. 获取标签标识的过程较慢, 每秒钟大概可获取 20 多个;

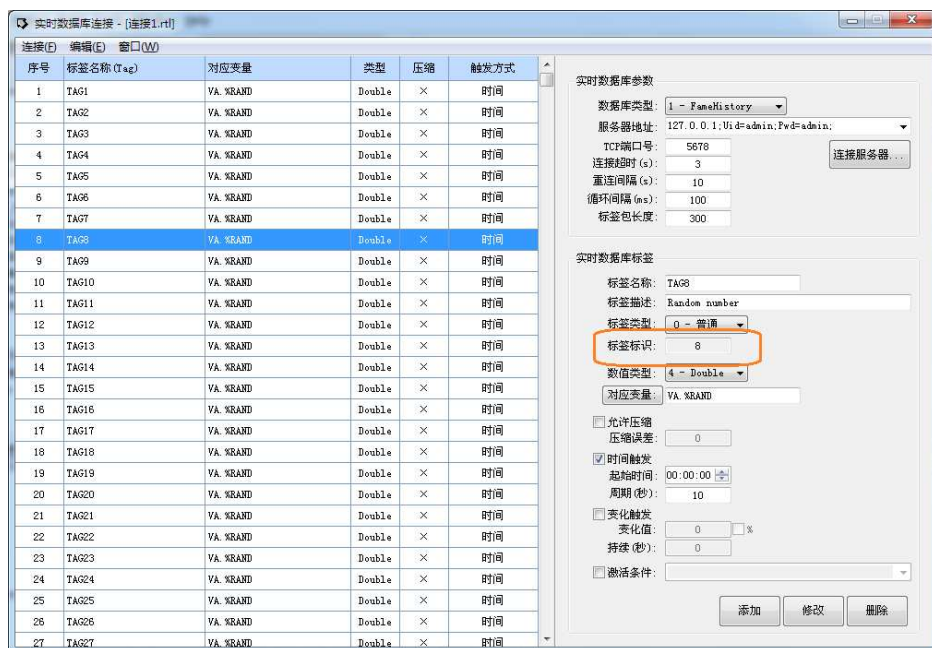
如果标签已经存在只是查询标识, 速度较快, 每秒可查询到 180 多个;

[6]. 列表中 TagName 为标签名称, TagType 为标签类型 (0-6), TagId 为获取到的标签标识 (1-100W);

[7]. TagStatus 为获取标识过程中的状态:

- 1 = 标签不存在, 添加标签而获得标识;
- 2 = 标签已存在, 且标签属性相同, 直接获得标识;
- 3 = 标签已存在, 数值类型相同, 其他属性不同, 修改标签属性并获得标识;
- 4 = 标签已存在, 数值类型不同, 删除标签重新添加并获得标识;
- 5 = 标签已存在, 标签类型不同, 删除其他类型标签重新添加并获得标识;
- ? = 获取标签失败;

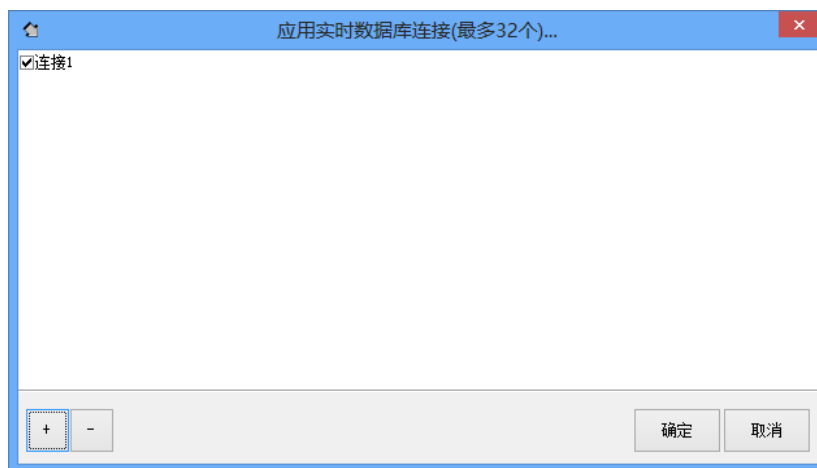
[8]. 获得标签标识后, 自动存储到连接文件, 运行时根据此标识进行存储:



[9]. 如果实时服务器发生了变化, 需要重新获取标签标识;

□ 应用实时数据库连接：

[1]. 选择实时数据库连接, 执行[应用实时数据库连接]：



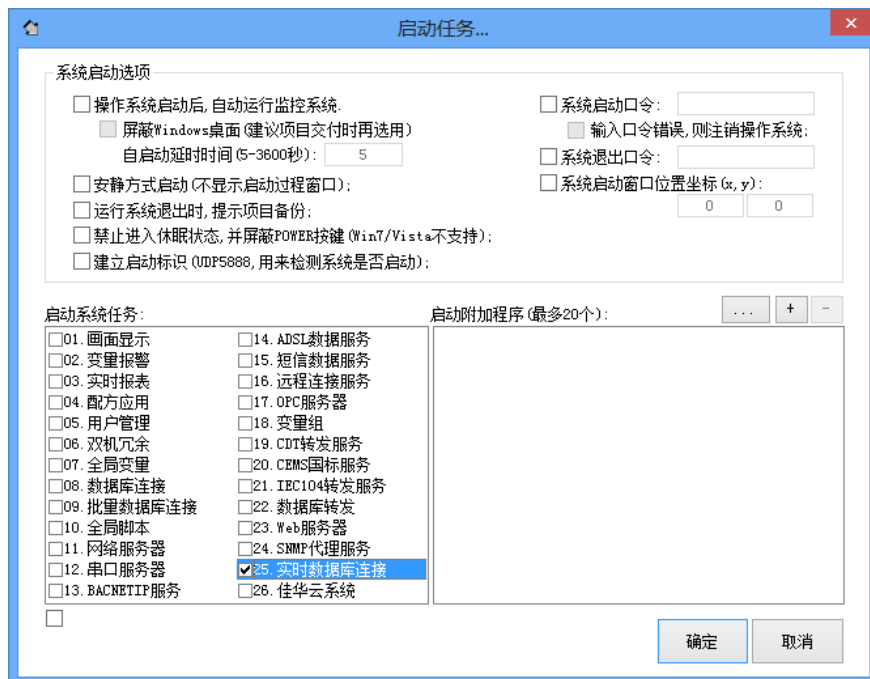
[2]. 添加连接文件到列表中：

[3]. 只有被应用的连接文件才允许工作；

[4]. 最多允许应用 32 个连接文件；

□ 启动实时数据库连接：

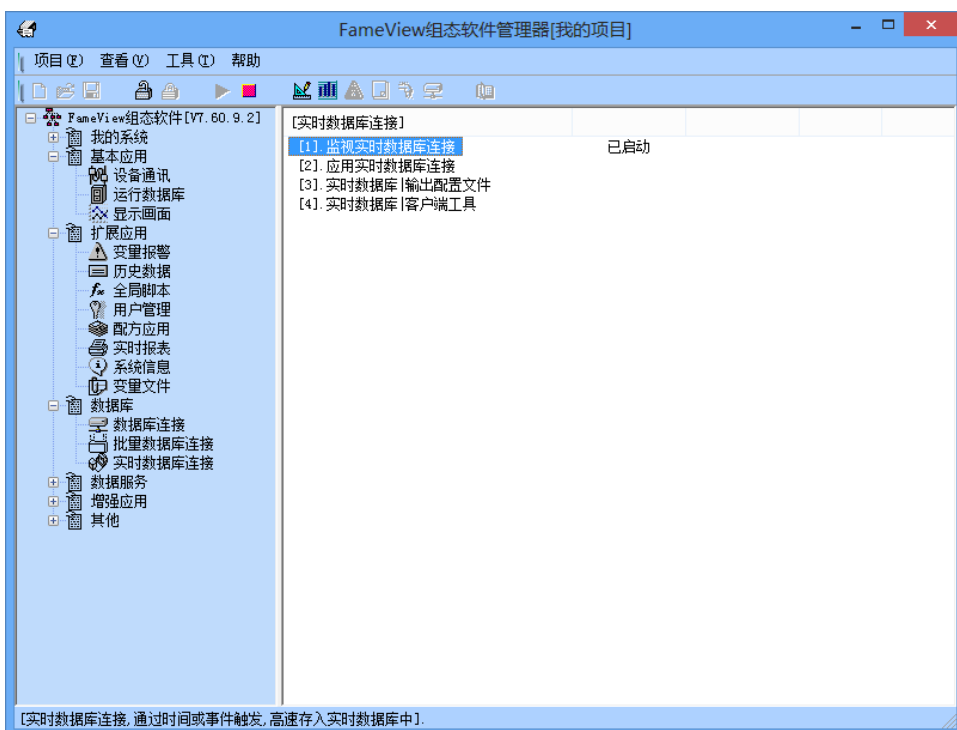
[1]. 选择“我的系统->设置”功能, 执行<2. 启动任务>：



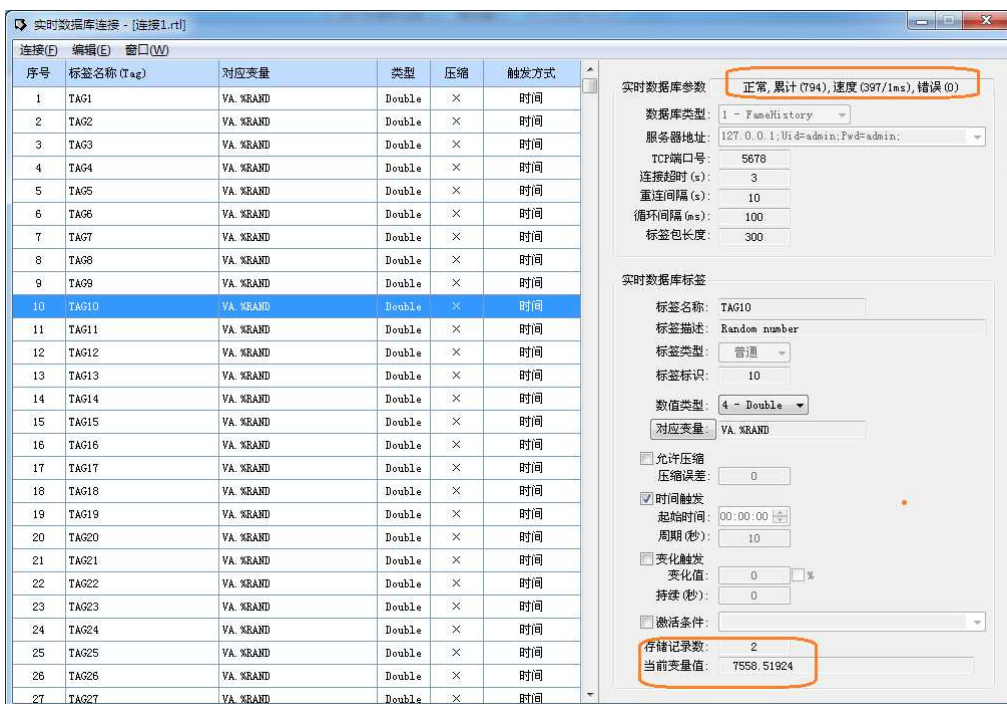
[2]. 从“系统任务”列表中选“☒实时数据库连接”；

□ 监视实时数据库连接：

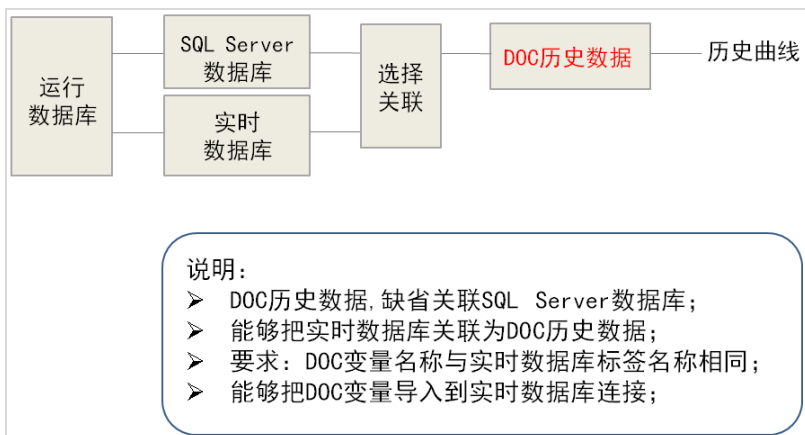
[1]. 运行状态时, 允许监视实时数据库连接：



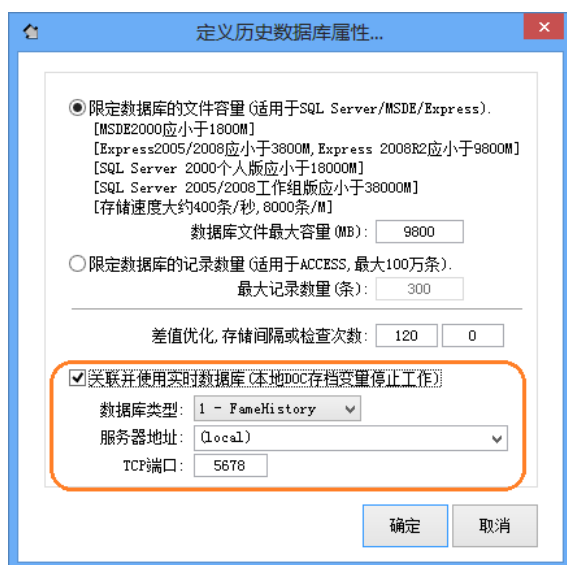
[2]. 执行“监视实时数据库连接”：



□ 通过 DOC 存档变量查询实时数据库



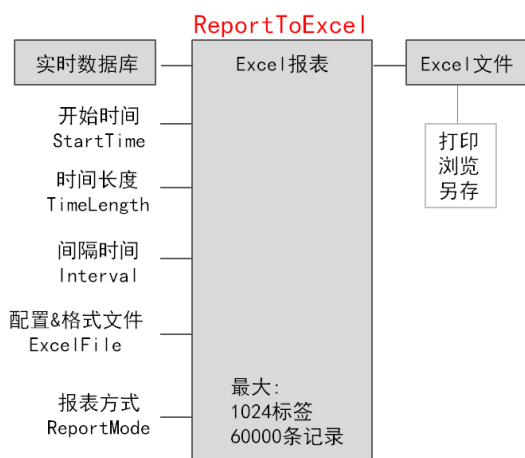
[1]. 选择<关联并使用实时数据库>：



[2]. 选择此选项后, 本地 DOC 存档变量不再存储；

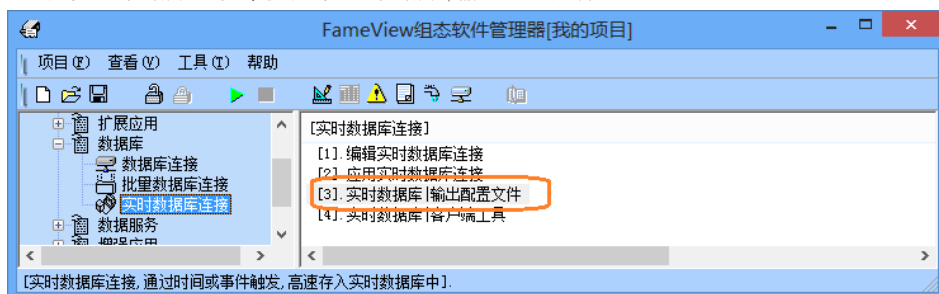
[3]. 本地和 Web 曲线查询数据, 直接来自于实时数据库；

27.7 实时数据库 Excel 报表

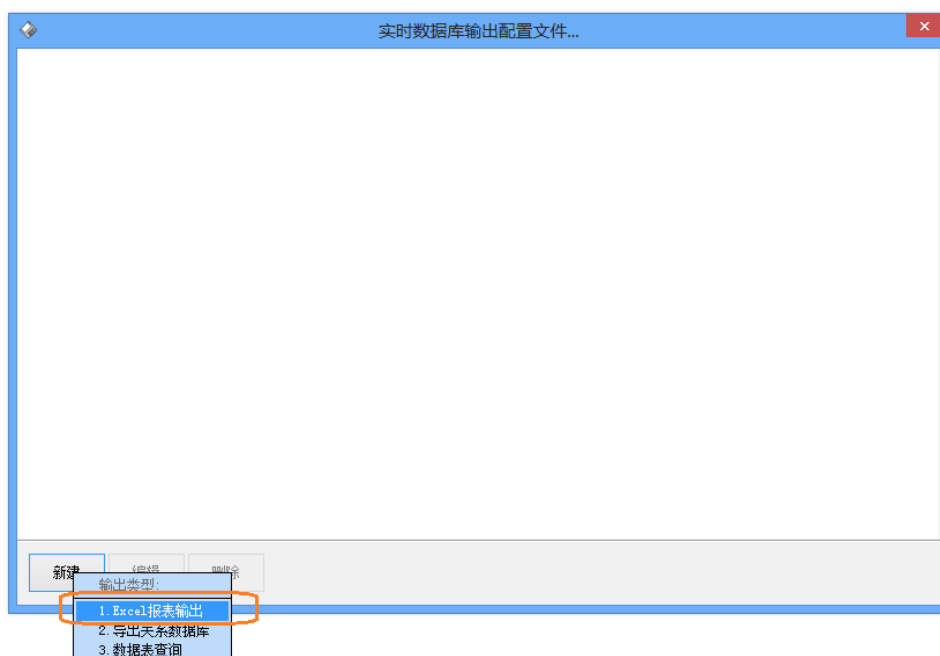


□ 建立配置文件:

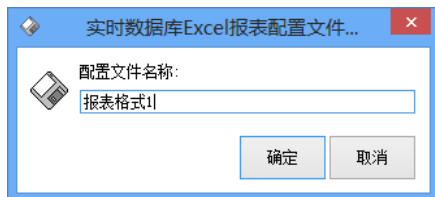
[1]. 选择<实时数据库连接>, 执行<实时数据库 | 输出配置文件>:



[2]. 执行<新建>按钮, 并选择<Excel 报表输出>:



[3]. 输入报表配置文件名称:

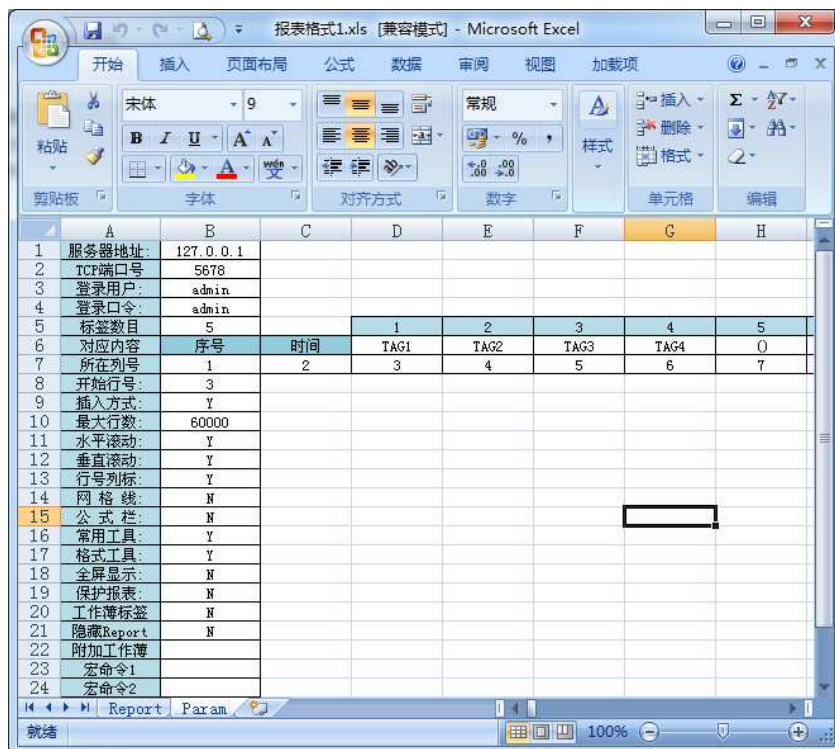


执行<确定>按钮, 组态安装目录下 RtlFile 子目录中建立 Excel 格式的配置文件;

- ☐ 打开编辑配置文件, 包含两个表 Report 和 Param:
- ☐ 通过 Report 设计报表格式, 必须设计两行:



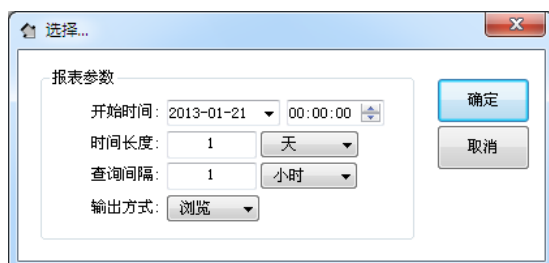
- ☐ 通过 Param 表设置数据源及报表格式参数:



- 执行脚本函数输出报表:

FameHistoryObj.ReportToExcel

"" , 1, 2, 1, 2, 1, 1, "选择..." , -1, -1, "报表格式 1. xls" , "" , "" , &H01



	A	B	C	D	E
1	报表标题				
2	序号	时间	标签1	标签2	公式1
3	1	2013-01-21 00:00:00			0
4	2	2013-01-21 01:00:00			0
5	3	2013-01-21 02:00:00			0
6	4	2013-01-21 03:00:00			0
7	5	2013-01-21 04:00:00			0
8	6	2013-01-21 05:00:00			0
9	7	2013-01-21 06:00:00			0
10	8	2013-01-21 07:00:00			0
11	9	2013-01-21 08:00:00			0
12	10	2013-01-21 09:00:00			0
13	11	2013-01-21 10:00:00			0
14	12	2013-01-21 11:00:00	58	1	59
15	13	2013-01-21 12:00:00	59	8159	8218
16	14	2013-01-21 13:00:00			0
17	15	2013-01-21 14:00:00			0
18	16	2013-01-21 15:00:00			0
19	17	2013-01-21 16:00:00			0
20	18	2013-01-21 17:00:00			0
21	19	2013-01-21 18:00:00			0
22	20	2013-01-21 19:00:00			0
23	21	2013-01-21 20:00:00			0
24	22	2013-01-21 21:00:00			0
25	23	2013-01-21 22:00:00			0
26	24	2013-01-21 23:00:00			0

- FameHistoryObj.ReportToExcel, 脚本函数说明:

查询某时间段记录, 以Excel格式打印、浏览、保存:

FameHistoryObj.ReportToExcel

startTime, timeLength, timeUnit, intervalLength, intervalUnit,
printMode, wndMode, wndTitle, wndXPos, wndYPos,
templateFile, outputPath, outputFile, Options

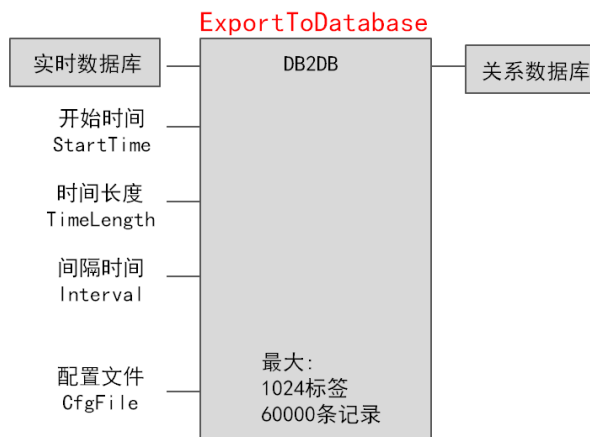
startTime	开始时间, 格式须为"YYYY-mm-dd HH:MM:SS"; startTime为空时, 自动取当日零点作为开始时间, 如"2012-12-01 00:00:00"; 可以定制开始时间, 格式为"%Y-%m-%d %H:%M:%S": - 取当日8点作为开始时间, 则startTime为"%Y-%m-%d 08:00:00" - 取当月1日零点作为开始时间, 则startTime为"%Y-%m-01 00:00:00"
timeLength	时间长度
timeUnit	时间单位, 0=分钟, 1=小时, 2=天, 3=月

intervalLength	取值间隔长度
intervalUnit	取值间隔单位, 0=秒, 1=分钟, 2=小时, 3=天
printMode	打印方式, 0=打印, 1=预览, 2=文件
dlgMode	是否显示选择窗口 dlgMode=1弹出时间输入窗口 dlgMode=0不弹出时间输入窗口, 而使用缺省的时间参数
dlgTitle	选择窗口标题内容
dlgXPos	选择窗口缺省横坐标位置, 为-1时居中显示
dlgYPos	选择窗口缺省竖坐标位置, 为-1时居中显示
templateFile	模版文件, 缺省在RtlFile子目录下 具体格式参看安装目录下fmRealdBReport.xls文件
outputPath	输出路径, 为空时, 缺省为组态目录下的Temp子目录
outputFile	输出文件, 为空时, 缺省为当前的日期时间
Options	选项内容: Bit(0)=1显示提示信息, Bit(0)=0安静方式

□ 速度测试:

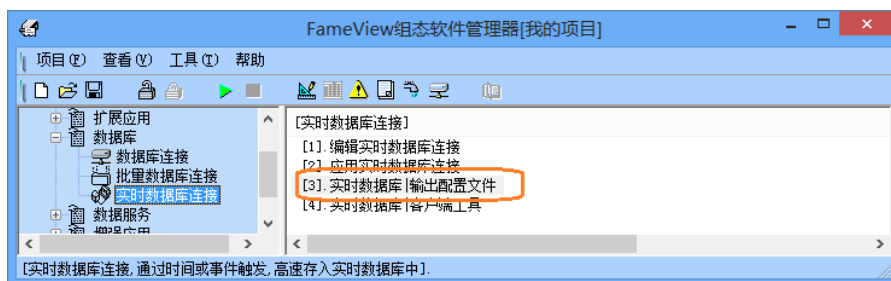
普通笔记本电脑, 2G内存	
本地实时数据库	
查询100个标签	
行数	耗时
600	5秒
1200	10秒

27.8 实时数据库导出到关系数据库

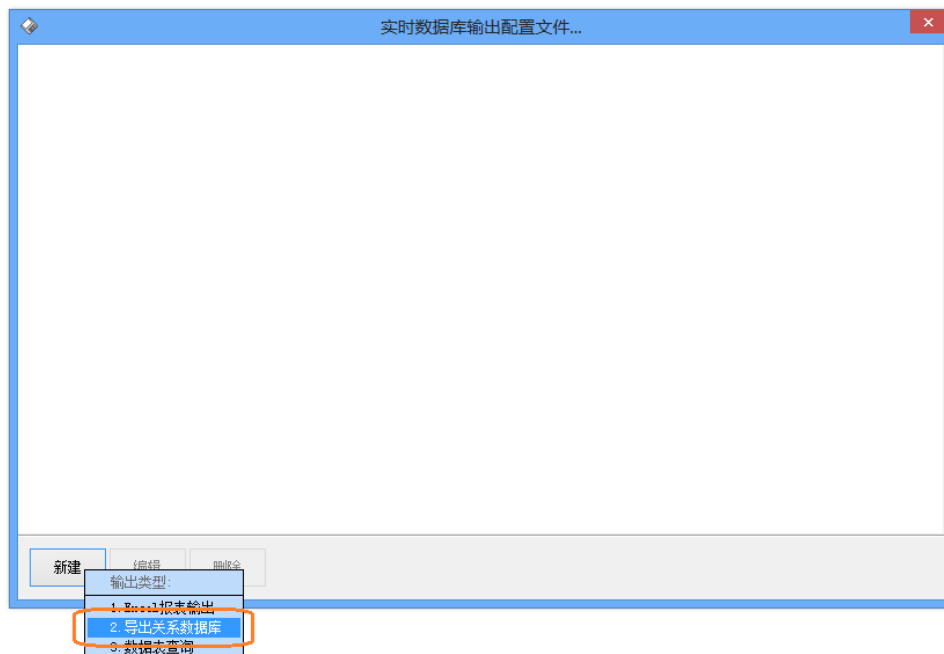


□ 建立配置文件:

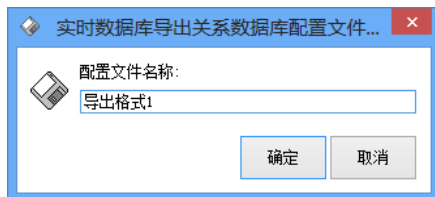
[1]. 选择<实时数据库连接>, 执行<实时数据库|输出配置文件>:



[2]. 执行<新建>按钮, 并选择<导出关系数据库>:

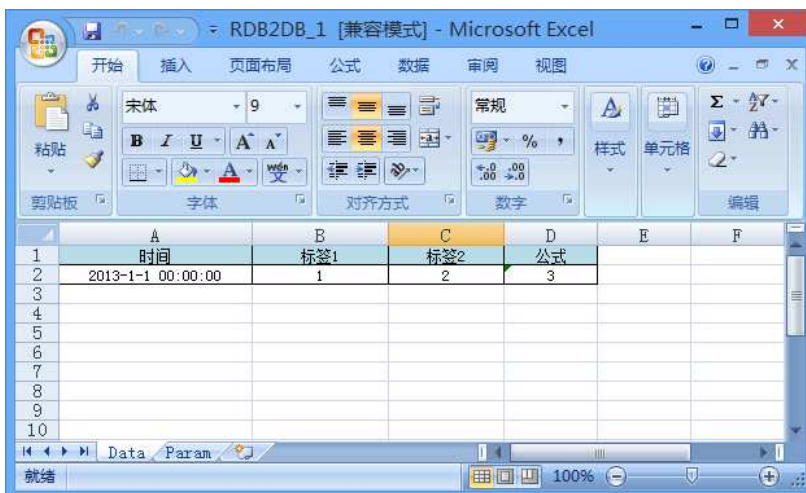


[3]. 输入配置文件名称:

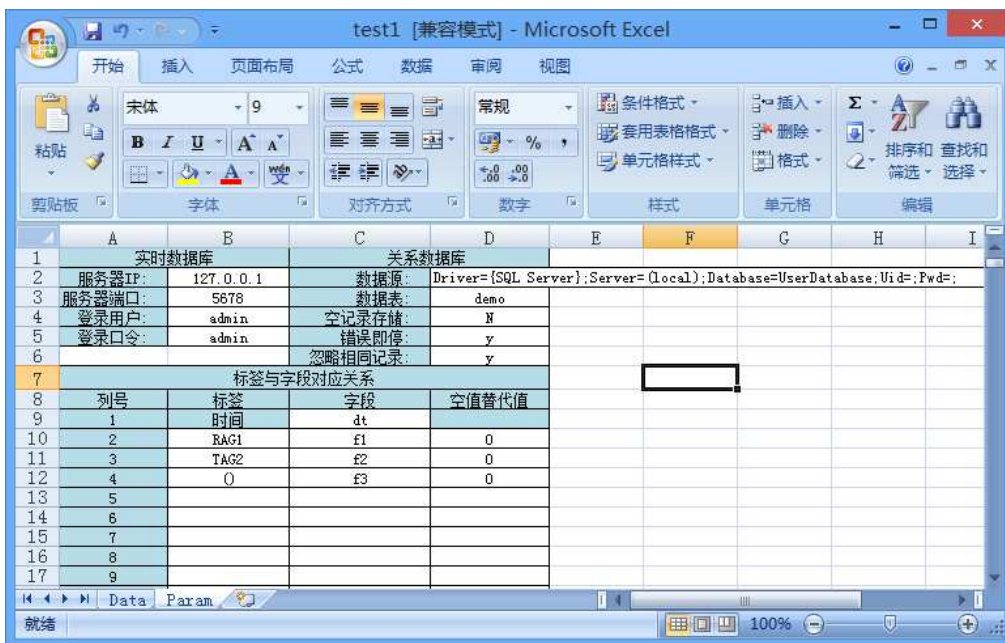


执行<确定>按钮, 组态安装目录下 RtlFile 子目录中建立 Excel 格式的配置文件;

- ☐ 打开配置文件, 包含两个表 Data 和 Param;
- ☐ 通过 Data 表设计实时数据库中多个标签的关系:



- ☐ 通过 Param 表设置数据源及标签和字段的对应关系:



标签名为"()"时, 表示为计算公式; 标签名为"[]"时, 表示为固定内容;

- 编写下面脚本函数实现报表:

`FameHistoryObj.ReportToDatabase "" , 1, 2, 1, 2, 0, "" , -1, -1, "导出格式 1.xls", &H00`

执行上面脚本, 可以把实时数据库数据导出到关系数据库:

- `FameHistoryObj.ReportToDatabase`, 脚本函数说明:

查询某时间段记录, 导出到关系数据库 (DB2DB):

`FameHistoryObj.ExportToDatabase`

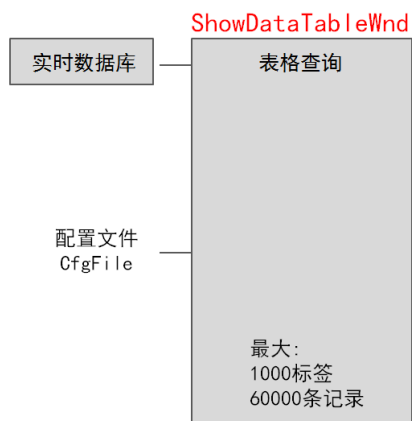
`startTime, timeLength, timeUnit, intervalLength, intervalUnit,`
`wndMode, wndTitle, wndXPos, wndYPos,`
`templateFile, options`

startTime	开始时间, 格式须为“YYYY-mm-dd HH:MM:SS” startTime为空时, 自动取当日零点作为开始时间, 如“2012-12-01 00:00:00” 可以定制开始时间, 格式为“%Y-%m-%d %H:%M:%S”: - 取当日8点作为开始时间, 则startTime为“%Y-%m-%d 08:00:00” - 取当月1日零点作为开始时间, 则startTime为“%Y-%m-01 00:00:00”
timeLength	时间长度
timeUnit	时间单位, 0=秒, 1=分钟, 2=小时, 3=天, 4=月
intervalLength	取值间隔长度
intervalUnit	取值间隔单位, 0=秒, 1=分钟, 2=小时, 3=天
dlgMode	是否显示选择窗口: dlgMode=1弹出时间输入窗口 dlgMode=0不弹出时间输入窗口, 而使用缺省的时间参数
dlgTitle	选择窗口标题内容
dlgXPos	选择窗口缺省横坐标位置, 为-1时居中显示
dlgYPos	选择窗口缺省竖坐标位置, 为-1时居中显示
templateFile	模版文件, 缺省在RtlFile子目录下, 具体格式参看安装目录下 fmRealdB2db.xls文件
options	选项内容: Bit (0)=1显示提示信息, Bit (0)=0安静方式

- 速度测试:

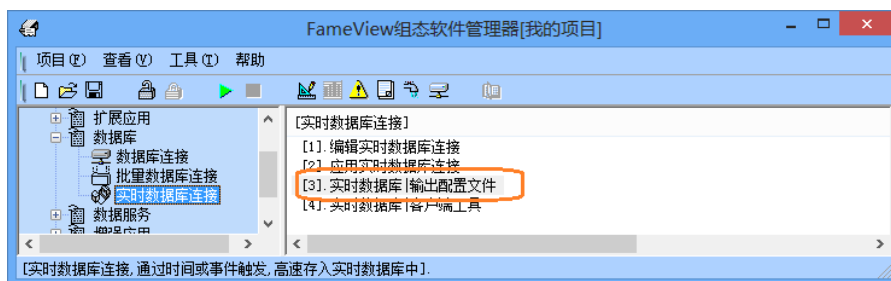
普通笔记本电脑, 2G 内存	
本地实时数据库	
100 个标签(字段), SQL Server 2000 数据库	
记录数(条)	导出时间
600	10 秒
1200	35 秒
1800	50 秒

27.9 实时数据库数据查询

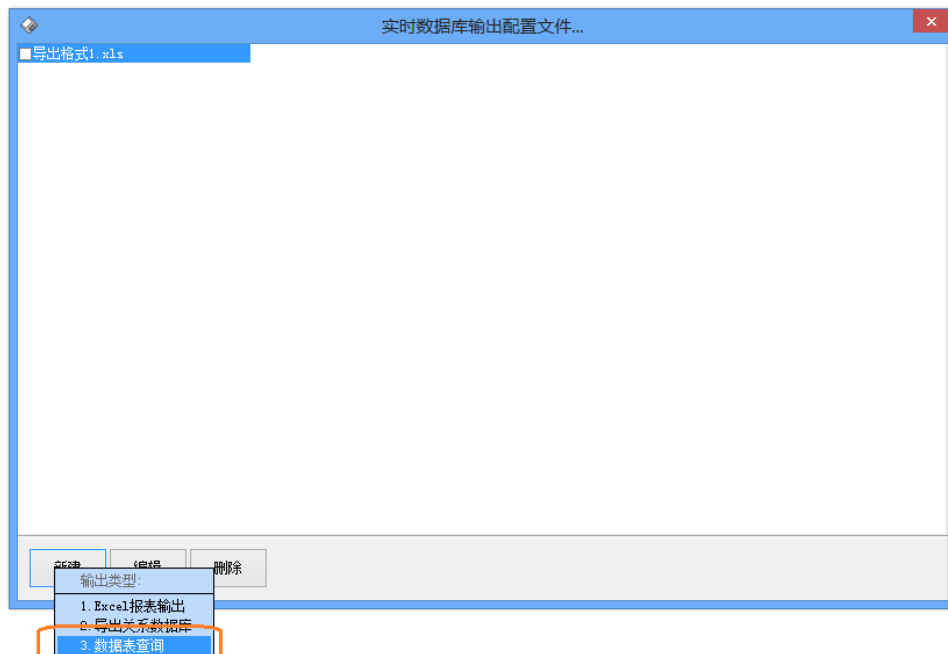


□ 建立配置文件:

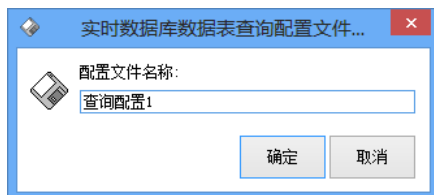
[1]. 选择<实时数据库连接>, 执行<实时数据库|输出配置文件>:



[2]. 在显示的界面中, 执行<新建>按钮, 并选择<数据表查询>:

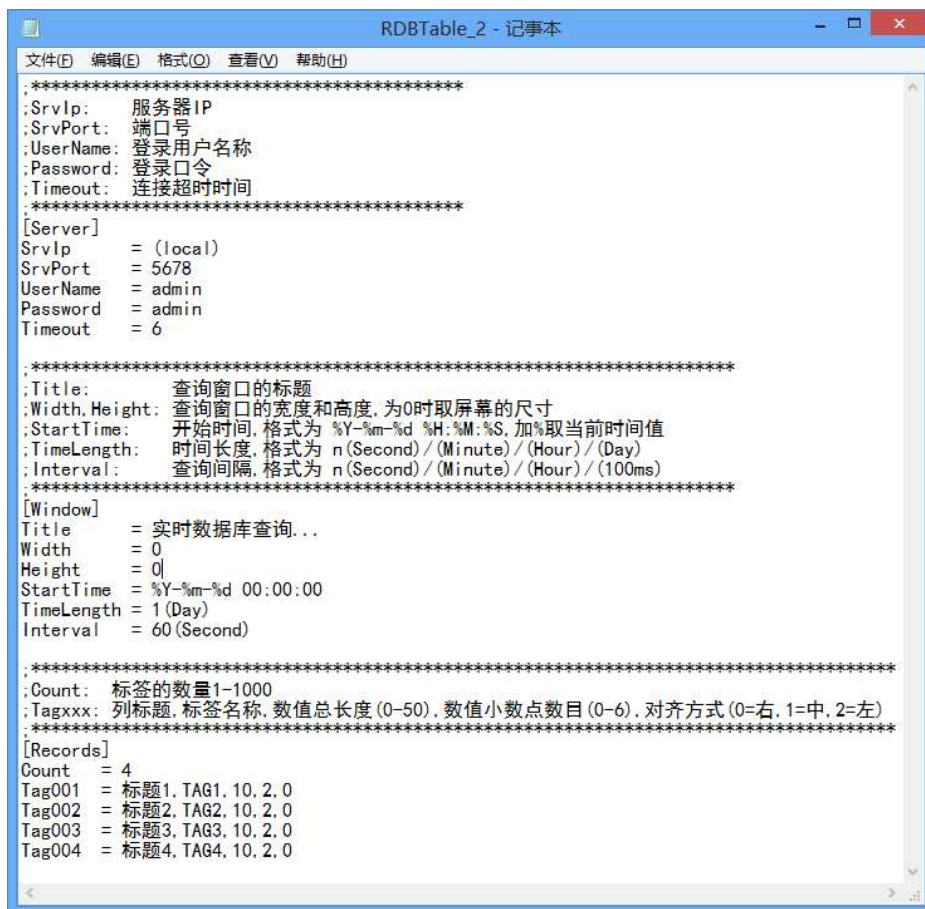


[3]. 输入配置文件名称:



执行<确定>按钮, 组态安装目录下 RtlFile 子目录中建立 ini 格式的配置文件;

□ 打开并编辑配置文件, 格式如下:



□ 编写画面脚本:

`FameHistoryObj.ShowDataTableWindow` “查询配置 1.ini”

执行脚本后, 查询界面如下:

序号	时间	标题1	标题2	标题3	标题4
1023	2013-03-06 17:02:00				
1024	2013-03-06 17:03:00	0.98	441.00	9560.00	9560.00
1025	2013-03-06 17:04:00	-0.33	922.00	9079.00	9079.00
1026	2013-03-06 17:05:00	-0.67	1401.00	8600.00	8600.00
1027	2013-03-06 17:06:00	0.97	1882.00	8119.00	8119.00
1028	2013-03-06 17:07:00	-0.25	2361.00	7640.00	7640.00
1029	2013-03-06 17:08:00	-0.71	2842.00	7159.00	7159.00
1030	2013-03-06 17:09:00	0.95	3321.00	6680.00	6680.00
1031	2013-03-06 17:10:00	-0.18	3801.00	6200.00	6200.00
1032	2013-03-06 17:11:00	-0.78	4280.00	5721.00	5721.00
1033	2013-03-06 17:12:00	0.91	4760.00	5241.00	5241.00
1034	2013-03-06 17:13:00				

通过查询按钮设置查询时间:

设置查询时间...

开始时间: 2013-03-06 00:00:00

长度: 1 天

间隔: 60 秒

确定 取消

通过上页和下页按钮查询上下时间段数据:

27.10 使用脚本访问实时数据库

```

' 建立对象
Set Obj = CreateObject("FameView.History")
' 连接FameHistory实时数据库:
' 参数分别为:服务器IP地址, 端口号, 登录用户名, 口令, 超时时间(秒)
' 成功连接返回1, 否则返回0, 可用GetLastError得到错误代码
n=Obj.Connect("127.0.0.1", 5678, "admin", "admin", 3)
If n=1 Then

    ' 查询某段时间的记录:
    ' 参数分别为: 标签名称, 开始时间, 结束时间, 秒间隔
    ' 返回值为查询到的记录数量(1-30000条), 查询失败则返回0, 可用GetLastError得到错误代码
    nCount=Obj.QueryRecords("TAG1", "2012-12-16 00:00:00", "2012-12-17 00:00:00", 60)
    If nCount>0 Then
        TxtFileObj.OpenFile "c:\temp\20121216.txt"
        For j=0 To nCount-1
            ' 得到某条记录时间和数值
            strTime=Obj.GetRecordTime(j)
            dblValue=Obj.GetRecordValue(j)
            s=strTime&"="&CStr(dblValue)&vbCrLf
            TxtFileObj.WriteFile s, 0
        Next
        TxtFileObj.CloseFile
    Else
        ErrorCode=Obj.GetLastError()
        MsgBox "查询失败:"&CStr(ErrorCode)
    End If

    ' 关闭连接
    Obj.DisConnect
Else
    ErrorCode=Obj.GetLastError()
    MsgBox "连接失败:"&CStr(ErrorCode)
End If

' 销毁对象
Set Obj=Nothing

```


27.11 使用 API 开发包访问实时数据库



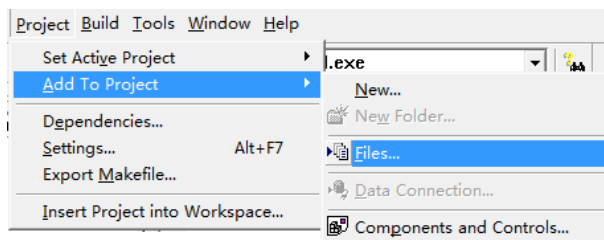
- 以 VC++ 6.0 为例子；
- API 开发包包括以下文件：

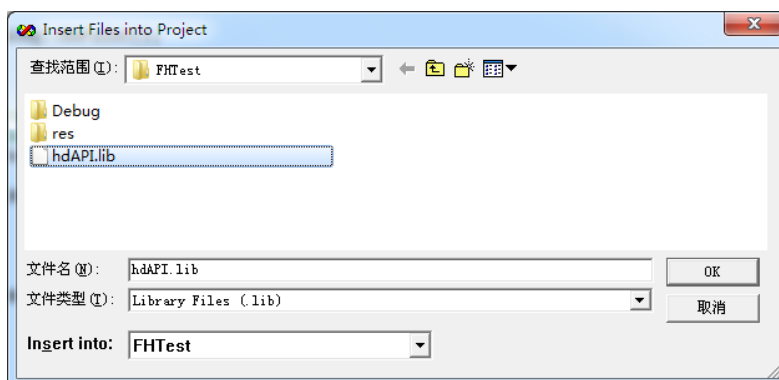
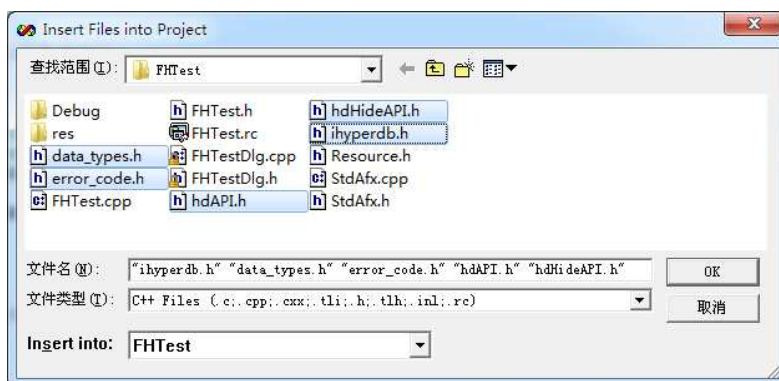
```

h data_types.h
h error_code.h
h hdAPI.h
h hdHideAPI.h
h ihyperdb.h
hdAPI.lib
hdAPI.dll
hdNetClient.dll
hdOS.dll
hdProcComm.dll

```

- 把 API 开发包文件拷贝到 VC6 的项目路径下；
- 通过 Project→Add To Project→Files 菜单, 把*.h 和*.lib 文件加入到项目中：





□ 访问数据的代码如下：

```
#include "hdApi.h"
//访问实时数据库原始记录
int CXXXXDlg::AccessRealtimeDatabase1(CTime startTime, CTime endTime)
{
    //-----
    //连接实时数据库服务器
    // nt_connect(服务器 IP, 端口号, 连接句柄, 连接超时秒数)
    HDHANDLE hServer = NULL;
    int32 nRet=nt_connect("127.0.0.1", 5678, &hServer, 3);
    if(nRet!=0) return 1;
    //-----
    //登录实时数据库服务器, 在本地访问可不登录
    // sc_login(连接句柄, 登录用户名称, 口令)
    nRet=sc_login(hServer, "admin", "admin");
    if(nRet!=0) {
        nt_disconnect(hServer);
        return 2;
    }
}
```

```

//-----
//根据变量标签名称, 获取标签标识
// pt_query_tagid(连接句柄, 标签名称, 标签 ID)
CString varName="TAG1";
uint32 tagID=0;
char tagName[128];
memset(tagName, 0, 128);
memcpy(tagName, varName, varName.GetLength());
nRet=pt_query_tagid(hServer, tagName, &tagID);
if(nRet!=0){
    nt_disconnect(hServer);
    return 3;
}

//-----
//设置查询的开始时间和结束时间
CTime m_InitTime(1970, 1, 1, 8, 0, 0); //初始时间戳
HDTime m_StartTime; //开始时间戳
HDTime m_EndTime; //结束时间戳
CTimeSpan timeSpan=startTime-m_InitTime;
m_StartTime.nSec = timeSpan.GetTotalSeconds();
m_StartTime.nMsec = 0;
timeSpan=endTime-m_InitTime;
m_EndTime.nSec = timeSpan.GetTotalSeconds();
m_EndTime.nMsec = 0;

//-----
//查询时间段内的原始记录
//ar_query_raw_records(连接句柄, 标签 ID, 开始时刻, 结束时刻,
//                      设置并返回记录数量, 记录内容)
HDRecord pRecords[30000];
int32 nRecNumQueried=30000;
nRet=ar_query_raw_records(hServer, tagID, &m_StartTime, &m_EndTime,
                          &nRecNumQueried, pRecords);
if(nRet!=0){
    nt_disconnect(hServer);
    return 4;
}
nt_disconnect(hServer);

```

```

//-----
//分析或显示数据
CString s,s1;
for(int i=0;i<nRecNumQueried;i++){

    //获取时标
    CTimeSpan timeSpan1(0,0,0,pRecords[i].nSec);
    CTime recordTime=m_InitTime+timeSpan1;
    s=recordTime.Format("%Y-%m-%d %H:%M:%S");
    s1.Format("%s.%.3i",s,pRecords[i].nMsec);

    //获取数据值
    for(;;){
        if(pRecords[i].nTagType==3){//32 位单精度浮点数
            s.Format("%.4f",pRecords[i].value.fFloat32);
            break;
        }
        if(pRecords[i].nTagType==4){//64 位双精度浮点数
            s.Format("%.6f",pRecords[i].value.fFloat64);
            break;
        }
        if(pRecords[i].nTagType==0){//8 位整数
            s.Format("%i",pRecords[i].value.nInt8);
            break;
        }
        if(pRecords[i].nTagType==1){//16 位整数
            s.Format("%i",pRecords[i].value.nInt16);
            break;
        }
        if(pRecords[i].nTagType==2){//32 位整数
            s.Format("%i",pRecords[i].value.nInt32);
            break;
        }
        break;
    }
}
return 0;
}

```

```

//访问实时数据库插值记录
int CXXXXDlg::AccessRealtimeDatabase2(CTime startTime, CTime endTime)
{
    //-----
    //连接实时数据库服务器
    // nt_connect(服务器 IP, 端口号, 连接句柄, 连接超时秒数)
    HDHANDLE hServer = NULL;
    int32 nRet=nt_connect("127.0.0.1", 5678, &hServer, 3);
    if(nRet!=0) return 1;

    //-----
    //登录实时数据库服务器, 在本地访问可不登录
    // sc_login(连接句柄, 登录用户名称, 口令)
    nRet=sc_login(hServer, "admin", "admin");
    if(nRet!=0) {
        nt_disconnect(hServer);
        return 2;
    }

    //-----
    //根据变量标签名称, 获取标签标识
    // pt_query_tagid(连接句柄, 标签名称, 标签 ID)
    CString varName="TAG1";
    uint32 tagID=0;
    char tagName[128];
    memset(tagName, 0, 128);
    memcpy(tagName, varName, varName.GetLength());
    nRet=pt_query_tagid(hServer, tagName, &tagID);
    if(nRet!=0) {
        nt_disconnect(hServer);
        return 3;
    }

    //-----
    //设置查询的开始时间和结束时间
    CTime m_InitTime(1970, 1, 1, 8, 0, 0); //初始时间戳
    HDTime m_StartTime; //开始时间戳
    HDTime m_EndTime; //结束时间戳
    CTimeSpan timeSpan=startTime-m_InitTime;

```

```

m_StartTime.nSec = timeSpan.GetTotalSeconds();
m_StartTime.nMsec = 0;
timeSpan=endTime-m_InitTime;
m_EndTime.nSec = timeSpan.GetTotalSeconds();
m_EndTime.nMsec = 0;

//-----
//根据插值间隔设置插值记录时刻
Int          m_Interval=60;//间隔 60 秒
int          m_RecordCount=(m_EndTime.nSec-m_StartTime.nSec)/m_Interval;//记录数量
HDRRecord*   pRecords=new HDRRecord[m_RecordCount];
int32*       pErrorCodes=new int32[m_RecordCount];
for(int j=0;j<m_RecordCount;j++){
    pRecords[j].nSec=nStartTime+n_Interval*j;
    pRecords[j].nMsec=999;
}

//-----
//查询时间段内的插值记录
// ar_query_interp_records_by_mode(连接句柄, 1, 标签 ID, 记录数量, 记录内容, 错误代码)
int32 nRet=ar_query_interp_records_by_mode(hServer, 1, tagId, m_RecordCount,
                                           pRecords, pErrorCodes);

if(nRet!=0 && nRet!=119611){ //查询失败
    nt_disconnect(hServer);
    return 4;
}
nt_disconnect(hServer);

//-----
//分析或显示数据
CString s, s1;
for(int i=0;i< m_RecordCount;i++){

    //获取时标
    CTimeSpan timeSpan1(0, 0, 0, pRecords[i].nSec);
    CTime recordTime=m_InitTime+timeSpan1;
    s=recordTime.Format("%Y-%m-%d %H:%M:%S");
    s1.Format("%.3i", s, pRecords[i].nMsec);

```

```
//获取数据值
for(;;) {
    if(pRecords[i].nTagType==3) { //32 位单精度浮点数
        s.Format("%.4f", pRecords[i].value.fFloat32);
        break;
    }
    if(pRecords[i].nTagType==4) { //64 位双精度浮点数
        s.Format("%.6f", pRecords[i].value.fFloat64);
        break;
    }
    if(pRecords[i].nTagType==0) { //8 位整数
        s.Format("%i", pRecords[i].value.nInt8);
        break;
    }
    if(pRecords[i].nTagType==1) { //16 位整数
        s.Format("%i", pRecords[i].value.nInt16);
        break;
    }
    if(pRecords[i].nTagType==2) { //32 位整数
        s.Format("%i", pRecords[i].value.nInt32);
        break;
    }
    break;
}
}
return 0;
}
```

```

//访问实时数据库标签的当前数值
int CXXXXDlg::GetTagCurrentValue()
{
    //-----
    //连接实时数据库服务器
    //nt_connect(服务器 IP, 端口号, 连接句柄, 连接超时秒数)
    HDHANDLE hServer = NULL;
    int32 nRet=nt_connect("127.0.0.1", 5678, &hServer, 3);
    if(nRet!=0) return 1;
    //-----
    //登录实时数据库服务器, 在本地访问可不登录
    //sc_login(连接句柄, 登录用户名称, 口令)
    nRet=sc_login(hServer, "admin", "admin");
    if(nRet!=0) {
        nt_disconnect(hServer);
        return 2;
    }
    //-----
    //根据变量标签名称, 获取标签标识
    // pt_query_tagid(连接句柄, 标签名称, 标签 ID)
    CString varName="TAG1";
    uint32 tagID=0;
    char tagName[128];
    memset(tagName, 0, 128);
    memcpy(tagName, varName, varName.GetLength());
    nRet=pt_query_tagid(hServer, tagName, &tagID);
    if(nRet!=0) {
        nt_disconnect(hServer);
        return 3;
    }
    //根据标签标识, 获取标签数值
    //sn_query_snapshot(连接句柄, 标签 ID, &标签值);
    HDRecord hdValue;
    nRet=sn_query_snapshot(hServer, tagID, &hdValue);
    if(nRet!=0) {
        nt_disconnect(hServer);
        return 4;
    }
}

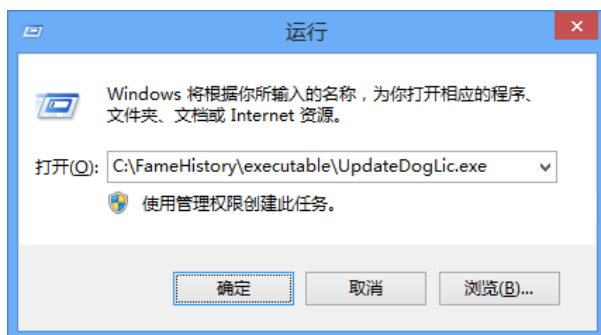
```



```
//分析标签值
for(;;){
    if(hdValue.nTagType==4){//双精度浮点数 VT_R8
        m_TagValue.Format("%. 6f", hdValue.value.fFloat64+0.0000001);
        break;
    }
    if(hdValue.nTagType==3){//单精度浮点数 VT_R4
        m_TagValue.Format("%. 4f", hdValue.value.fFloat32);
        break;
    }
    if(hdValue.nTagType==0){//8 位整数, VT_I1
        m_TagValue.Format("%i", hdValue.value.nInt8);
        break;
    }
    if(hdValue.nTagType==1){//16 位整数, VT_I2
        m_TagValue.Format("%i", hdValue.value.nInt16);
        break;
    }
    if(hdValue.nTagType==2){//32 位整数, VT_I4
        m_TagValue.Format("%i", hdValue.value.nInt32);
        break;
    }
    break;
}
return 0;
}
```

27.12 远程升级实时库加密狗

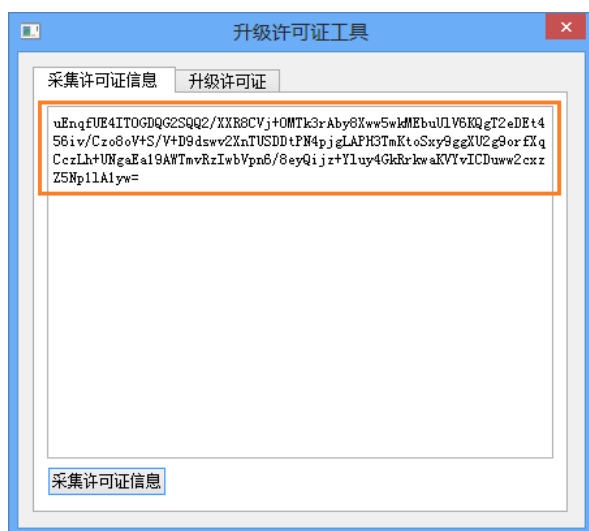
- 运行实时库安装目录下的. \executable\UpdateDogLic.exe 升级程序:



- 界面如下:

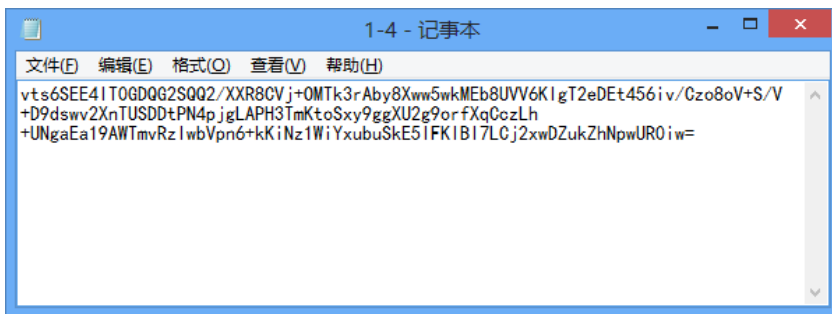


- 插入加密狗, 执行[采集许可证信息]选项下[采集许可证信息]按钮:

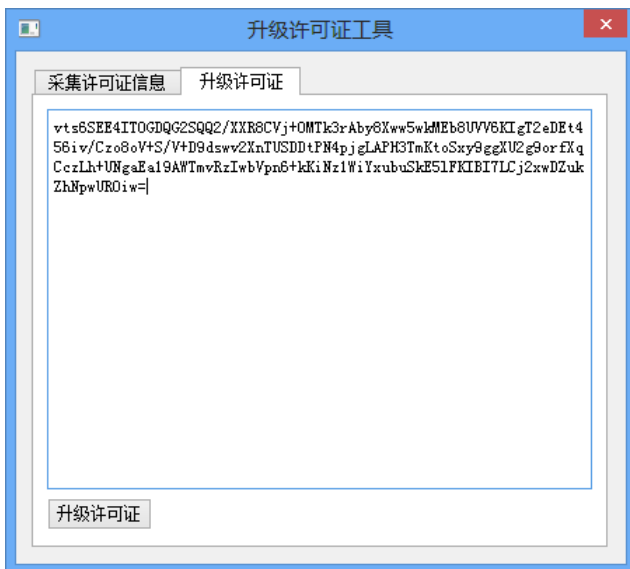


拷贝采集到的文本信息到文本文件, 联系开发商, 发送邮件给开发商;

- 开发商根据采集加密狗信息, 提供升级信息文件:



- 拷贝升级文本到 UpdateDogLic.exe 升级程序中的[升级许可证]处:



- 执行[升级许可证]按钮, 完成升级;

27.13 常见错误代码

110350	网络接收超时
119613	客户端和服务端版本不一致导致
111157	存储数据失败
119609	保存数据部分失败