**WebSocketServer详细设计说明书**

[1 引言 3](#_Toc489961310)

[1.1 背景 3](#_Toc489961311)

[1.2 编写目的 3](#_Toc489961312)

[1.3 定义 3](#_Toc489961313)

[2 总体设计 3](#_Toc489961314)

[2.1 需求概述 3](#_Toc489961315)

[2.2 软件结构 4](#_Toc489961316)

[3 系统详细设计 5](#_Toc489961317)

[3.1 总体类图 5](#_Toc489961318)

[3.2 变量 5](#_Toc489961319)

[3.2.1 mapSession 5](#_Toc489961320)

[3.2.2 mmapGID 5](#_Toc489961321)

[3.2.3 pLock 5](#_Toc489961322)

[3.2.4 pTagOpt 6](#_Toc489961323)

[3.2.5 pEvtOpt 6](#_Toc489961324)

[3.2.6 pAlmOpt 6](#_Toc489961325)

[3.3 数据结构 6](#_Toc489961326)

[3.3.1 KGlobalID 6](#_Toc489961327)

[3.3.2 ETagType 7](#_Toc489961328)

[3.3.3 UTagValue 7](#_Toc489961329)

[3.3.4 KDTime 8](#_Toc489961330)

[3.3.5 EEventType 9](#_Toc489961331)

[3.3.6 KTagInfo 9](#_Toc489961332)

[3.3.7 KEventInfo 10](#_Toc489961333)

[3.3.8 KAlarmInfo 11](#_Toc489961334)

[3.4 系统功能模块详细设计 12](#_Toc489961335)

[3.4.1 数据序列化/反序列化模块 12](#_Toc489961336)

[3.4.2 客户端请求信息处理子系统 23](#_Toc489961337)

[3.4.3 用户信息管理模块 26](#_Toc489961338)

[3.4.4 消息推送子系统 31](#_Toc489961339)

[3.4.5 RDBS服务器通信模块 34](#_Toc489961340)

[3.4.6 实时数据缓冲模块 43](#_Toc489961341)

[4 附录1 48](#_Toc489961342)

[4.1 WebSocketServer前后台通讯协议 48](#_Toc489961343)

# 引言

## 背景

由于开发了新的YLBuilder客户端，而服务端不支持新的YLBuilder客户端，而本项目就是为了给新YLBuilder提供服务而建立的，本服务端现阶段只需要为新YLBuilder提供所需服务，后期则会在其他业务的增加下，对新服务端进行扩展；

## 编写目的

本详细设计说明书是为了对项目进行详细设计，明确系统结构，详细地介绍系统的各个模块以及对外的接口，为进行后面的实现和测试作准备。

本详细设计说明书的预期读者为本项目小组的成员以及对该系统感兴趣，在以后想对系统进行扩展和维护的人员。

## 定义

文档中采用的专门术语的定义及缩略词简要如下：

WebSocketServer ：WebSocket服务器，就是本项目的名称

# 总体设计

## 需求概述

WebSocketServer服务端的需求有以下几点：

1、接收并解析新版本YLBuilder发送过来的WebSocket消息；

2、能够处理新版本YLBuilder发送的以下请求：

用户登录请求；

获取变量请求；

写入变量请求；

查询历史事件请求；

查询历史告警请求；

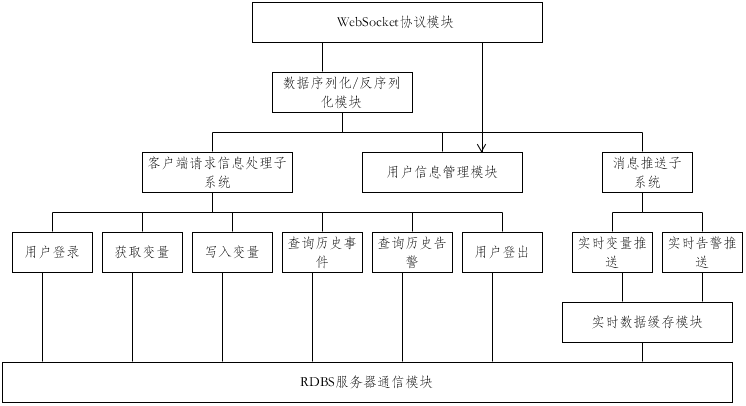
用户登出请求；

3、能够给已经连接的新版本YLBuilder用户推送以下信息：

实时变量信息；

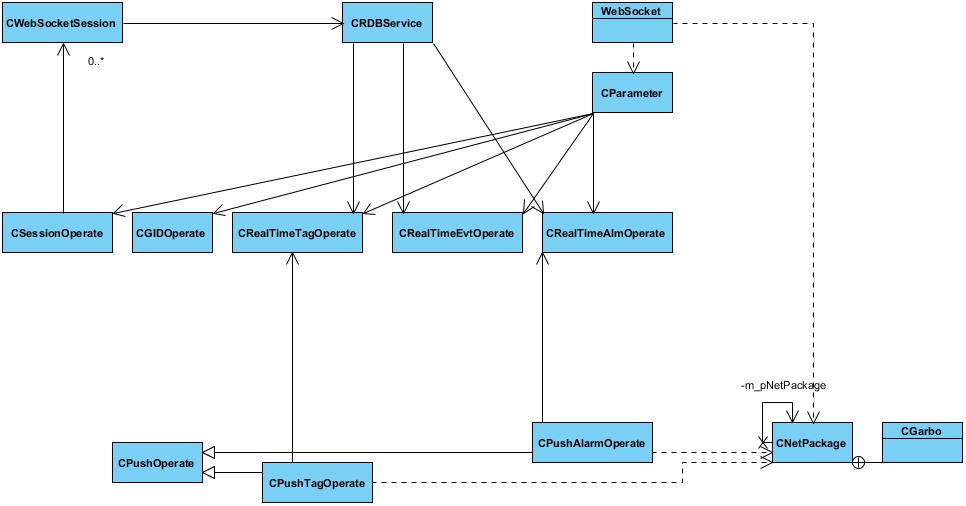
实时告警信息；

## 软件结构



# 系统详细设计

## 总体类图



## 变量

### mapSession

类型：map<int, shared\_ptr<CWebSocketSession>>；

功能：用来保存所有会话信息和每个WebSocket连接的对应关系；

### mmapGID

类型：multimap<DWORD, int>；

功能：用来保存变量ID和每个WebSocket连接的对应关系；

### pLock

类型：shared\_ptr<CLock>；

功能：用来保证mapSession和mmapGID的原子操作；

### pTagOpt

类型：shared\_ptr<CRealTimeTagOperate>；

功能：实时数据缓冲模块中用来缓存实时变量的对象；

### pEvtOpt

类型：shared\_ptr<CRealTimeEvtOperate>；

功能：实时数据缓冲模块中用来缓存实时事件的对象（暂未使用，后期扩展用）；

### pAlmOpt

类型：shared\_ptr<CRealTimeAlmOperate>；

功能：实时数据缓冲模块中用来缓存实时告警的对象；

## 数据结构

### KGlobalID

类型：struct（结构体）；

描述：全局ID（子系统ID+设备ID+变量ID）；

#### KGlobalID成员

* **ucSubsystemID**

类型：unsigned char；

描述：子系统ID

* **ucDeviceID**

类型：unsigned char；

描述：设备ID

* **usTagID**

类型：unsigned short；

描述：变量ID

### ETagType

类型：enum（枚举类型）；

描述：变量类型，因为变量存在多种类型，因此需要使用枚举类型标识变量类型；

#### ETagType成员

TYPE\_BOOL：bool型；

TYPE\_INTEGER：整型；

TYPE\_FLOAT：浮点型；

TYPE\_STRING：字符串；

### UTagValue

类型：union（共用体）；

描述：变量值，因为变量存在多种类型，因此使用公用体来表示变量；

#### UTagValue成员

* **bValue**

类型：bool型（true/false）；

描述：bool型的变量值；

* **nValue**

类型：整型（int32\_t）；

描述：整型的变量值；

* **fValue**

类型：浮点型（float）；

描述：浮点型的变量值；

* **pcValue**

类型：字符指针（char\*）；

描述：指向字符串变量的指针；

### KDTime

类型：struct（结构体）；

描述：时间（详细时间，年月日时分秒，毫秒）；

#### KDTime成员

* **nYear**

类型：int；

描述：年份；

* **nMonth**

类型：int；

描述：月份；

* **nDay**

类型：int；

描述：日期；

* **nHour**

类型：int；

描述：小时；

* **nMinute**

类型：int；

描述：分钟；

* **nSecond**

类型：int；

描述：秒；

* **nMinSec**

类型：int；

描述：毫秒；

### EEventType

类型：enum（枚举类型）；

描述：事件类型（暂时只有一种默认类型）；

#### EEventType成员

EVENT\_TYPE\_0：默认类型；

### KTagInfo

类型：struct（结构体）；

描述：一个变量所包含的信息；

#### KTagInfo成员

* **tGlobalID**

类型：struct（结构体）；

描述：同[KGlobalID](#_KGlobalID)；

* **eType**

类型：enum（枚举类型）；

描述：同[ETagType](#_ETagType)；

* **uinValue**

类型：union（共用体）；

描述：同[UTagValue](#_UTagValue)；

* **szBuffer**

类型：char型数组；

描述：用来保存uinValue中的字符串的值；

* **sQuality**

类型：short；

描述：质量（这个变量根据RDBS服务器中的代码，是个常量，并不是真实值，具体意义未知）；

* **tTimeStamp**

类型：struct（结构体）；

描述：时间戳，同[KDTime](#_KDTime)；

### KEventInfo

类型：struct（结构体）；

描述：一条事件所包含的信息；

#### KEventInfo成员

* **tTime**

类型：struct（结构体）;

描述：事件时间，同[KDTime](#_KDTime)；

* **nID**

类型：int；

描述：事件ID；

* **szAccount**

类型：char型数组；

描述：产生事件的用户名；

* **szRecord**

类型：char型数组；

描述：事件信息简单描述；

* **nAccessLevel**

类型：int；

描述：访问权限；

* **eType**

类型：enum（枚举类型）；

描述：事件类型，同[EEventType](#_EEventType)；

### KAlarmInfo

类型：struct（结构体）；

描述：一条告警所包含的信息；

#### KAlarmInfo成员

* **tTime**

类型：struct（结构体）；

描述：告警产生时间，同[KDTime](#_KDTime)；

* **nID**

类型：int；

描述：告警ID；

* **szAccount**

类型：char型数组；

描述：产生告警的用户名；

* **nCategory**

类型：int；

描述：告警类型；

* **szArea**

类型：char型数组；

描述：未知参数；

* **szDescribe**

类型：char型数组；

描述：告警信息的简单描述；

* **szTag**

类型：char型数组；

描述：未知参数；

* **szValue**

类型：char型数组；

描述：未知参数；

* **nPriority**

类型：int；

描述：报警等级；

* **szAlarmEvent**

类型：char型数组；

描述：未知参数；

* **bAck**

类型：bool（true/false）；

描述：是否应答；

## 系统功能模块详细设计

### 数据序列化/反序列化模块

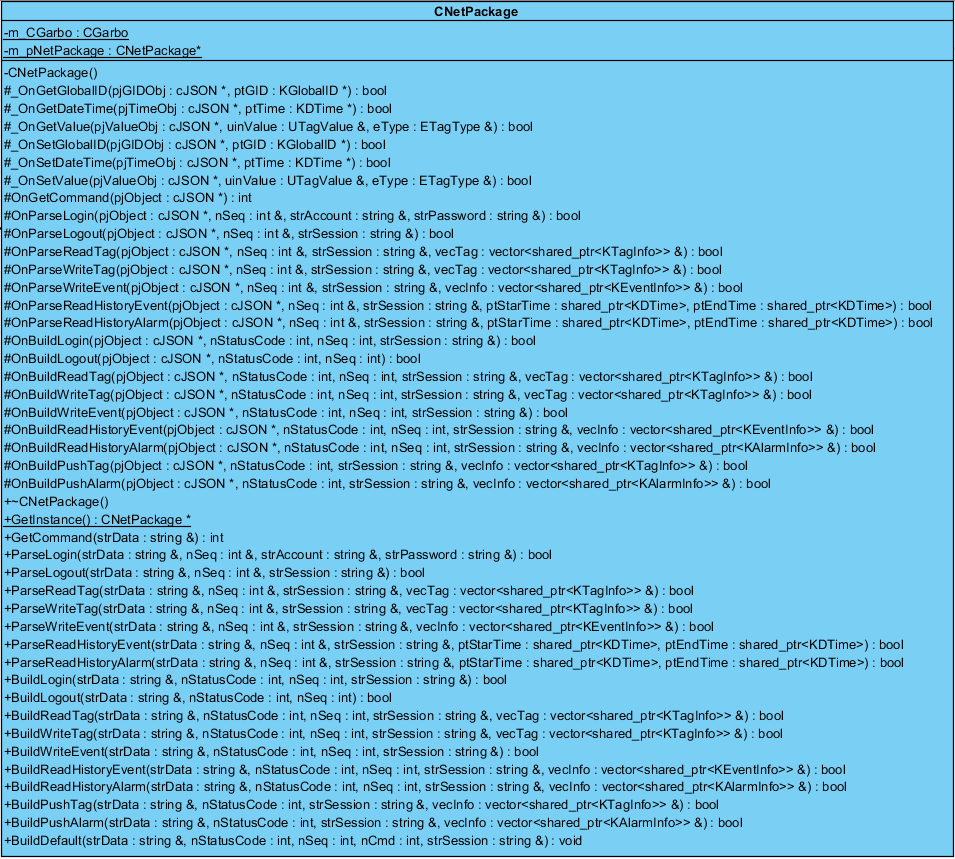
#### 功能概述

将WebSocket协议模块接收到的数据反序列化（解析）为客户端请求信息处理子系统所需要的数据格式；

将客户端请求信息处理子系统处理结果序列化（构建）为WebSocket协议模块所需要的数据格式；

注：本模块所所要反序列化的对象和序列化后的结果都是已经规定好的格式，格式详情可以参考“附录1”中的“WebSocketServer前后台通讯协”子文档；

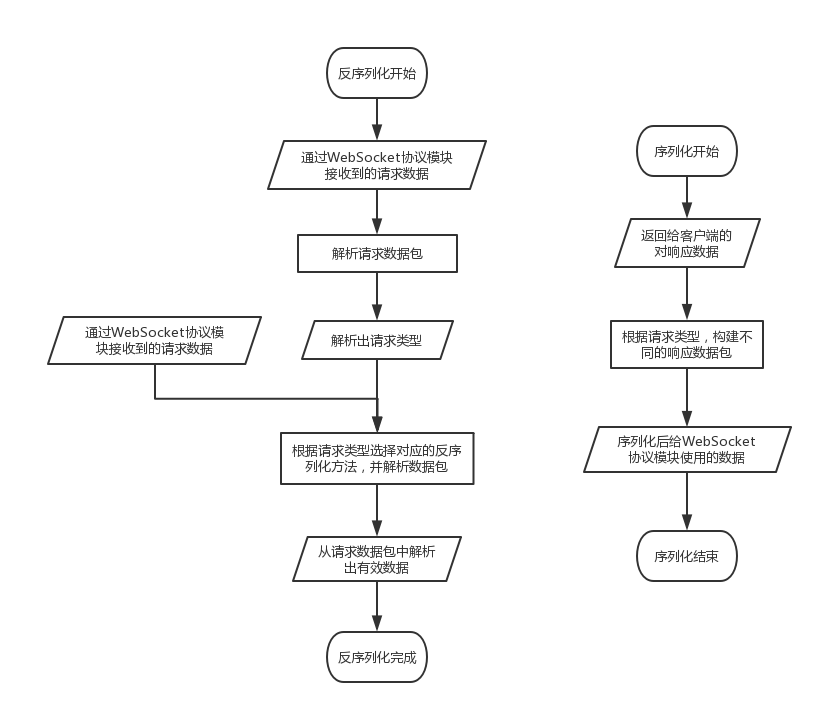
#### 详细类图



#### 成员变量

无

#### 流程图



#### 方法实现

##### 反序列化（解析）请求包并获取请求类型

* **接口**

int GetCommand(const string &strData)；

* **功能**

解析通过WebSocket协议模块接收到的请求数据，获取到请求类型，通过请求类型来判断使用哪个接口来处理请求数据；

* **输入项目**

strData：通过WebSocket协议模块接收到的请求数据；

* **输出项目**

int（返回值）：客户端请求类型；

##### 反序列化（解析）登录请求包

* **接口**

bool ParseLogin(const string &strData, int &nSeq, string &strAccount, string &strPassword)；

* **功能**

解析通过WebSocket协议模块接收到的登录请求数据，并将解析出的数据给客户端请求信息处理子系统使用；

* **输入项目**

strData：通过WebSocket协议模块接收到的登录请求数据；

* **输出项目**

nSeq：请求包序列号；

strAccount：用户名；

strPassword：用户密码（32位MD5（小写16进制））；

bool（返回值）：反序列化执行结果；

##### 反序列化（解析）登出请求包

* **接口**

bool ParseLogout(const string &strData, int &nSeq, string &strSession)；

* **功能**

解析通过WebSocket协议模块接收到的登出请求数据，并将解析出的数据给客户端请求信息处理子系统使用；

* **输入项目**

strData：通过WebSocket协议模块接收到的登出请求数据；

* **输出项目**

nSeq：请求包序列号；

strSession：会话ID；

bool（返回值）：反序列化执行结果；

##### 反序列化（解析）读变量请求包

* **接口**

bool ParseReadTag(const string &strData, int &nSeq, string &strSession, vector<shared\_ptr<KTagInfo>> &vecTag);

* **功能**

解析通过WebSocket协议模块接收到的读变量请求数据，并将解析出的数据给客户端请求信息处理子系统使用；

* **输入项目**

strData：通过WebSocket协议模块接收到的读变量请求数据;

* **输出项目**

nSeq：请求包序列号；

strSession：会话ID；

vecTag：要读取的0-N个变量的ID信息；

bool（返回值）：反序列化执行结果；

##### 反序列化（解析）写变量请求包

* **接口**

bool ParseWriteTag(const string &strData, int &nSeq, string &strSession, vector<shared\_ptr<KTagInfo>> &vecTag)；

* **功能**

解析通过WebSocket协议模块接收到的写变量请求数据，并将解析出的数据给客户端请求信息处理子系统使用；

* **输入项目**

strData：通过WebSocket协议模块接收到的写变量请求数据；

* **输出项目**

nSeq：请求包序列号；

strSession：会话ID；

vecTag：要写入的0-N个变量的信息；

bool（返回值）：反序列化执行结果；

##### 反序列化（解析）查询历史事件请求包

* **接口**

bool ParseReadHistoryEvent(const string &strData, int &nSeq, string &strSession, shared\_ptr<KDTime> ptStarTime, shared\_ptr<KDTime> ptEndTime)；

* **功能**

解析通过WebSocket协议模块接收到的查询历史事件请求数据，并将解析出的数据给客户端请求信息处理子系统使用；

* **输入项目**

strData：通过WebSocket协议模块接收到的查询历史事件请求数据；

* **输出项目**

nSeq：请求包序列号；

strSession：会话ID；

ptStarTime：要查询的数据的开始时间（查询条件）；

ptEndTime：要查询的数据的结束时间（查询条件）；

bool（返回值）：反序列化执行结果；

##### 反序列化（解析）查询历史报警请求包

* **接口**

bool ParseReadHistoryAlarm(const string &strData, int &nSeq, string &strSession, shared\_ptr<KDTime> ptStarTime, shared\_ptr<KDTime> ptEndTime)；

* **功能**

解析通过WebSocket协议模块接收到的查询历史告警请求数据，并将解析出的数据给客户端请求信息处理子系统使用；

* **输入项目**

strData：通过WebSocket协议模块接收到的查询历史告警请求数据；

* **输出项目**

nSeq：请求包序列号；

strSession：会话ID；

ptStarTime：要查询的数据的开始时间（查询条件）；

ptEndTime：要查询的数据的结束时间（查询条件）；

bool（返回值）：反序列化执行结果；

##### 序列化（构建）登录响应包

* **接口**

bool BuildLogin(string &strData, int nStatusCode, int nSeq, const string &strSession)；

* **功能**

将客户端请求信息处理子系统处理后的结果构建为登录响应包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

strSession：会话ID；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）登出响应包

* **接口**

bool BuildLogout(string &strData, int nStatusCode, int nSeq)；

* **功能**

将客户端请求信息处理子系统处理后的结果构建为登出响应包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）读变量响应包

* **接口**

bool BuildReadTag(string &strData, int nStatusCode, int nSeq, const string &strSession, const vector<shared\_ptr<KTagInfo>> &vecTag)；

* **功能**

将客户端请求信息处理子系统处理后的结果构建为读变量响应包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

strSession：会话ID；

vecTag：读取到0-N个变量信息；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）写变量响应包

* **接口**

bool BuildWriteTag(string &strData, int nStatusCode, int nSeq, const string &strSession, const vector<shared\_ptr<KTagInfo>> &vecTag)；

* **功能**

将客户端请求信息处理子系统处理后的结果构建为写变量响应包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

strSession：会话ID；

vecTag：写入的0-N个变量的ID；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）查询历史事件响应包

* **接口**

bool BuildReadHistoryEvent(string &strData, int nStatusCode, int nSeq, const string &strSession, const vector<shared\_ptr<KEventInfo>> &vecInfo)；

* **功能**

将客户端请求信息处理子系统处理后的结果构建为查询历史事件响应包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

strSession：会话ID；

vecInfo：查询到的的0-N个历史事件的信息；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）查询历史告警响应包

* **接口**

bool BuildReadHistoryAlarm(string &strData, int nStatusCode, int nSeq, const string &strSession, const vector<shared\_ptr<KAlarmInfo>> &vecInfo)；

* **功能**

将客户端请求信息处理子系统处理后的结果构建为查询历史告警响应包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

strSession：会话ID；

vecInfo：查询到的0-N个历史告警的信息；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）实时变量推送包

* **接口**

bool BuildPushTag(string &strData, int nStatusCode, const string &strSession, const vector<shared\_ptr<KTagInfo>> &vecInfo)；

* **功能**

将消息推送子系统中需要推送的实时变量构建为实时变量推送包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：消息推送子系统处理数据的结果状态；

strSession：会话ID；

vecInfo：需要推送的0-N个实时变量的信息；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）实时告警推送包

* **接口**

bool BuildPushAlarm(string &strData, int nStatusCode, const string &strSession, const vector<shared\_ptr<KAlarmInfo>> &vecInfo)；

* **功能**

将消息推送子系统中需要推送的实时告警构建为实时告警推送包，并将构建后的结果给WebSocket协议模块使用；

* **输入项目**

nStatusCode：消息推送子系统处理数据的结果状态；

strSession：会话ID；

vecInfo：需要推送的0-N个实时告警的信息；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

##### 序列化（构建）默认响应包

* **接口**

void BuildDefault(string &strData, int nStatusCode, int nSeq, int nCmd, const string &strSession)；

* **功能**

当指定响应类型的包构造失败时，则会使用本接口构造一个默认的响应包给WebSocket协议模块使用，而不至于在响应包构造失败时客户端收不到任何数据；

* **输入项目**

nStatusCode：客户端请求信息处理子系统处理后的结果状态；

nSeq：响应包序列号（与对应的请求包的序列号相同）；

nCmd：客户端请求类型；

strSession：会话ID；

* **输出项目**

strData：序列化后给WebSocket协议模块使用的数据；

bool（返回值）：序列化执行结果；

### 客户端请求信息处理子系统

#### 功能概述

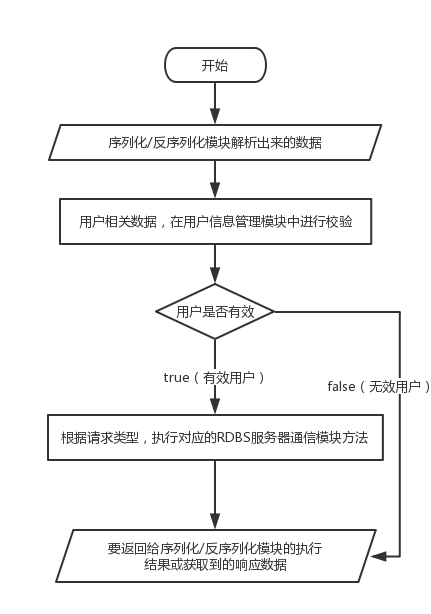
处理反序列化（解析）后的请求，根据不同的请求转发到RDBS服务通信模块中，并将返回来的处理结果转发回请求者；

注：本子系统没有具体实现的类和方法，只是通过“用户信息管理模块”检查用户信息后，将“数据序列化/反序列化模块”与“RDBS服务器通信模块”的数据后相互传输的一些过程模块；

#### 详细类图

无；

#### 流程图



#### 方法实现

##### 用户登录

* **功能**

将序列化/反序列化模块传过来信息，通过用户信息管理模块的校验后，调用RDBS服务器通信模块的登录方法进行登录，并将登陆结果返回给序列化/反序列化模块；

##### 获取变量

* **功能**

将序列化/反序列化模块传过来的信息，通过用户信息管理模块的校验后，调用RDBS服务器通信模块的获取变量方法获取变量，并将获取结果返回给序列化/反序列化模块；

##### 写入变量

* **功能**

将序列化/反序列化模块传过来的信息，通过用户信息管理模块的校验后，调用RDBS服务器通信模块的写入变量方法写入变量，并将写入结果返回给序列化/反序列化模块；

##### 查询历史事件

* **功能**

将序列化/反序列化模块传过来的信息，通过用户信息管理模块的校验后，调用RDBS服务器通信模块的查询历史事件方法查询历史事件信息，并将查询结果返回给序列化/反序列化模块；

##### 查询历史告警

* **功能**

将序列化/反序列化模块传过来的信息，通过用户信息管理模块的校验后，调用RDBS服务器通信模块的查询历史告警方法查询历史告警信息，并将查询结果返回给序列化/反序列化模块；

##### 用户登出

* **功能**

将序列化/反序列化模块传过来的信息，通过用户信息管理模块的校验后，调用RDBS服务器通信模块的用户登出方法登出用户，并将登出结果返回给序列化/反序列化模块；

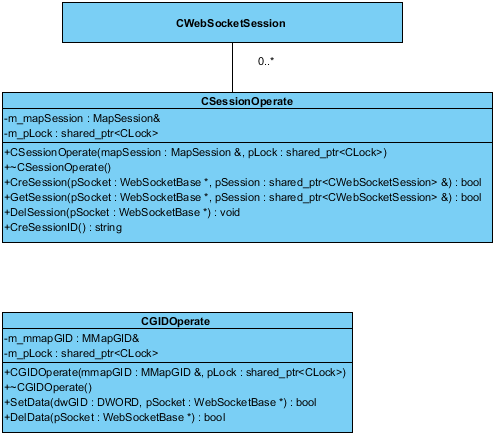
### 用户信息管理模块

#### 功能概述

为每个用户的所有连接管理一个SESSION，用以保证数据访问的安全性；

保存每个用户所拥有的变量ID的信息，可以让消息推送子系统只给用户推送用户需要的变量信息（暂时不支持单点登录）；

#### 详细类图



#### 成员变量

* **CSessionOperate类：**

m\_mapSession：通过构造函数传入的[mapSession](#_mapSession)；

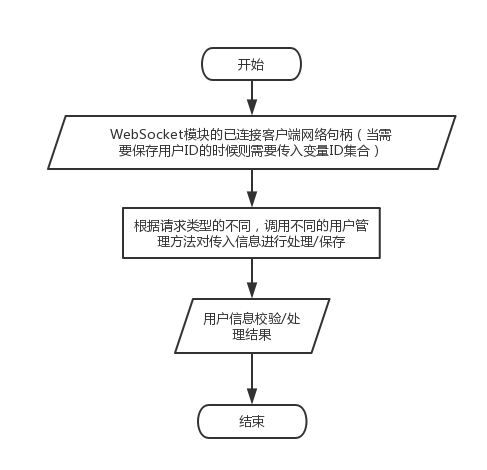
m\_pLock：通过构造函数传入的[pLock](#_pLock)；

* **CGIDOperate类：**

m\_mmapGID：通过构造函数传入的[mmapGID](#_mmapGID)；

m\_pLock：通过构造函数传入的[pLock](#_pLock)；

#### 流程图



#### 方法实现

##### 创建会话

* **接口**

bool CreSession(WebSocketBase\* pSocket, shared\_ptr<CWebSocketSession> &pSession)；

* **功能**

为WebSocket模块的新连接客户端网络句柄创建一个会话对象；

* **输入项目**

pSocket：WebSocket模块传过来的一个新的客户端网络句柄；

* **输出项目**

pSession：创建出的新会话对象；

bool（返回值）：执行是否成功；

##### 获取会话

* **接口**

bool GetSession(const WebSocketBase\* pSocket, shared\_ptr<CWebSocketSession> &pSession)；

* **功能**

通过WebSocket模块的已连接客户端网络句柄来获取对应的会话对象；

* **输入项目**

pSocket：WebSocket模块传过来的已连接客户端网络句柄；

* **输出项目**

pSession：与该连接对应的会话对象

bool（返回值）：执行是否成功；

##### 删除会话

* **接口**

void DelSession(const WebSocketBase\* pSocket);

* **功能**

通过WebSocket模块的已连接客户端网络句柄来删除对应的会话对象；

* **输入项目**

pSocket：WebSocket模块传过来的已连接客户端网络句柄；

* **输出项目**

bool（返回值）：执行是否成功；

##### 创建会话ID

* **接口**

string CreSessionID()；

* **功能**

创建一个会话ID，主要是给会话对象使用；会话ID是一个随机的用来标识一个已登录用户的字符串；

* **输入项目**

无；

* **输出项目**

string：创建出来的会话ID；

##### 将变量ID与已连接用户绑定

* **接口**

bool SetData(DWORD dwGID, const WebSocketBase\* pSocket)；

* **功能**

将变量ID绑定到WebSocket模块的已连接客户端网络句柄上，主要是在数据推送的时候可以精确的给每个已连接用户推送他们所需要的变量；

* **输入项目**

dwGID：变量ID；

pSocket：WebSocket模块的已连接客户端网络句柄；

* **输出项目**

bool（返回值）：执行是否成功；

##### 删除已经绑定的变量ID

* **接口**

bool DelData(const WebSocketBase\* pSocket)；

* **功能**

将WebSocket模块的已连接客户端网络句柄上所绑定的变量ID都删除掉；

* **输入项目**

pSocket：WebSocket模块的已连接客户端网络句柄；

* **输出项目**

bool（返回值）：执行是否成功；

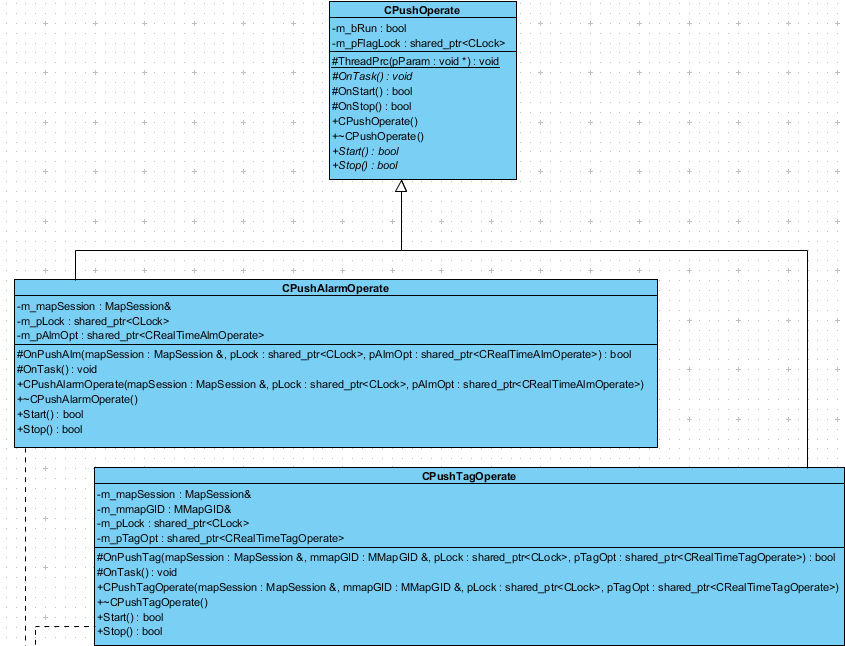
### 消息推送子系统

#### 功能概述

将实时数据缓冲模块中的实时变量和告警数据，根据用户信息管理模块的信息来将指定的变量信息传送给数据序列化/反序列化模块；

注：本功能的每个推送子模块都自己维护一个线程，用来单独推送实时数据；

#### 详细类图



#### 成员变量

* **CPushOperate类**

m\_bRun：用来控制线程是否退出运行；

m\_pFlagLock：用来确保m\_bRun的原子操作；

* **CPushTagOperate类**

m\_mapSession：通过构造函数传入的[mapSession](#_mapSession)；

m\_mmapGID：通过构造函数传入的[mmapGID](#_mmapGID)；

m\_pLock：通过构造函数传入的[pLock](#_pLock)；

m\_pTagOpt：通过构造函数传入的[pTagOpt](#_pTagOpt)；

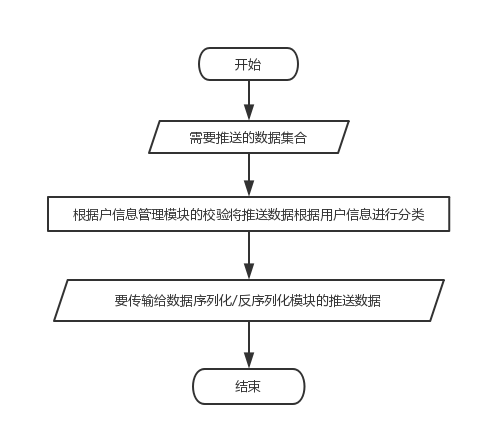
* **CPushAlarmOperate类**

m\_mapSession：通过构造函数传入的[mapSession](#_mapSession)；

m\_pLock：通过构造函数传入的[pLock](#_pLock)；

m\_pAlmOpt：通过构造函数传入的[pAlmOpt](#_pAlmOpt)；

#### 流程图



#### 方法实现

##### 实时变量信息推送

* **接口**

bool OnPushTag(const MapSession &mapSession, const MMapGID &mmapGID, const shared\_ptr<CLock> pLock, const shared\_ptr<CRealTimeTagOperate> pTagOpt)；

* **功能**

将实时数据缓存模块中的实时变量信息，根据用户信息管理模块传送到数据序列化/反序列化模块；

* **输入项目**

mapSession：用来保存每个用户会话对象的数据结构；

MMapGID：用来保存与已经连接的客户端网络句柄绑定的变量ID；

pLock：用来给mapSession和MMapGID的操作加锁；

pTagOpt：实时缓冲模块中的实时变量信息对象；

* **输出项目**

bool（返回值）：执行是否成功；

##### 实时告警信息推送

* **接口**

bool OnPushAlm(const MapSession &mapSession, const shared\_ptr<CLock> pLock, const shared\_ptr<CRealTimeAlmOperate> pAlmOpt)；

* **功能**

将实时数据缓存模块中的实时告警信息，根据用户信息管理模块传送到数据序列化/反序列化模块；

* **输入项目**

mapSession：用来保存每个用户会话对象的数据结构；

pLock：用来给mapSession的操作加锁；

pAlmOpt：实时缓冲模块中的实时告警信息对象

* **输出项目**

bool（返回值）：执行是否成功；

### RDBS服务器通信模块

#### 功能概述

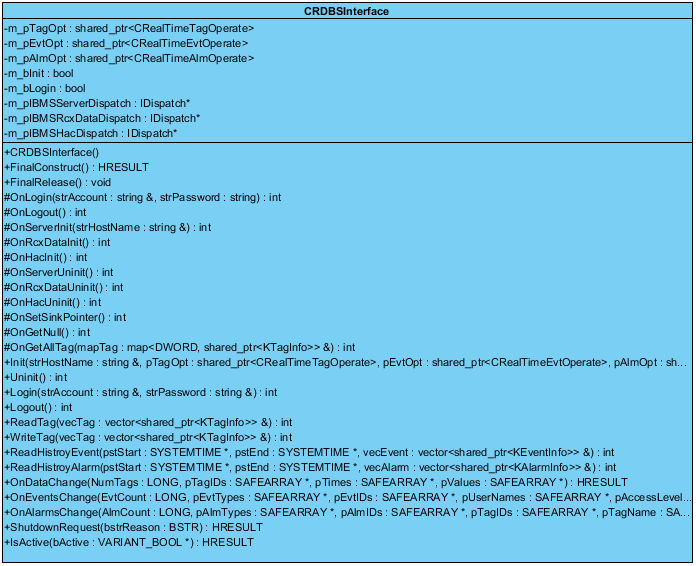
整个系统中最核心的模块，将RDBS服务器的DCOM通信协议进一步封装，处理客户端请求信息子系统的各种请求，并返回处理的结果；

本模块实现的类是CRDBSInterface，每个用户都必须对应一个实例，不同多个用户共同使用同一个实例；且当用户登出以后，必须销毁该实例，再次登陆必须使用一个新的实例；

本模块的int型返回值是错误码，具体错误码类型参考“附录1” 中的“WebSocketServer前后台通讯协”子文档

注：这个类是一个ATL类，因此通过类向导生成该类后，就会默认有两个函数“FinalConstruct”和“FinalRelease”，因此，类图中的这两个函数就不做介绍了；

#### 详细类图



#### 成员变量

* **CRDBSInterface类**

m\_pTagOpt：通过Init函数传入的[pTagOpt](#_pTagOpt)；

m\_pEvtOpt：通过Init函数传入的[pEvtOpt](#_pEvtOpt)；

m\_pAlmOpt：通过Init函数传入的[pAlmOpt](#_pAlmOpt)；

m\_pIBMSServerDispatch：BMSServer对象，内部接口通过DCOM连接到RDBS服务器后获取的；

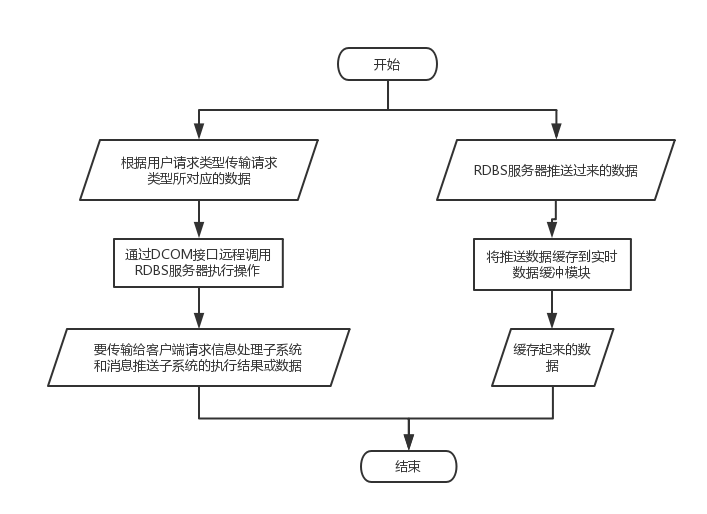
m\_pIBMSRcxDataDispatch：BMSRcxData对象，内部接口通过DCOM连接到RDBS服务器后获取的；

m\_pIBMSHacDispatch：BMSHac对象（历史数据服务），内部接口通过DCOM连接到RDBS服务器后获取的；

m\_bInit：用来在内部方法中判断对象是否被初始化，防止多次初始化；

m\_bLogin：用来在内部方法中判断对象是否已经登录过RDBS服务器，防止多次登录；

#### 流程图



#### 方法实现

##### 初始化

* **接口**

int Init(const string &strHostName, shared\_ptr<CRealTimeTagOperate> pTagOpt, shared\_ptr<CRealTimeEvtOperate> pEvtOpt, shared\_ptr<CRealTimeAlmOperate> pAlmOpt)；

* **功能**

1、传入CRDBSInterface类需要外部传入的参数；

2、与RDBS服务器建立连接，并对连接进行初始化操作；

* **输入项目**

strHostName：RDBS服务器所在的主机名（必须是主机名或者域名，不能是IP，RDBS无法处理IP）；

pTagOpt：用来保存实时变量的队列的类对象；

pEvtOpt：用来保存实时事件的队列的类对象（暂未使用，后期扩展使用）；

pAlmOpt：用来保存实时告警的队列的类对象；

* **输出项目**

int（返回值）：方法执行结果；

##### 反初始化

* **接口**

int Uninit()；

* **功能**

1、清理CRDBSInterface内部的数据；

2、断开与RDBS服务器的连接；

* **输入项目**

无；

* **输出项目**

int（返回值）：方法执行结果；

##### 登录

* **接口**

int Login(const string &strAccount, const string &strPassword)；

* **功能**

根据客户端请求信息处理子系统传过来的登录信息，通过RDBS服务器进行登录，并返回登录结果给客户端请求信息处理子系统；

* **输入项目**

strAccount：用户名；

strPassword：用户密码（32位MD5（小写16进制））；

* **输出项目**

int（返回值）：方法执行结果；

##### 登出

* **接口**

int Logout()；

* **功能**

对用户进行登出，并返回登出结果给客户端请求信息处理子系统；

* **输入项目**

无；

* **输出项目**

int（返回值）：方法执行结果；

##### 读取变量

* **接口**

int ReadTag(vector<shared\_ptr<KTagInfo>> &vecTag)；

* **功能**

根据客户端请求信息处理子系统传过来的变量ID，通过RDBS服务器获取到结果并传给客户端请求信息处理子系统；

* **输入项目**

vecTag：要读取得0-N个变量的ID；

* **输出项目**

vecTag：读取到的0-N个变量的信息；

int（返回值）：方法执行结果；

##### 写入变量

* **接口**

int WriteTag(const vector<shared\_ptr<KTagInfo>> &vecTag)；

* **功能**

根据客户端请求信息处理子系统传过来的变量信息，通过RDBS服务器获取到结果并传给客户端请求信息处理子系统；

* **输入项目**

vecTag：要写入的0-N个变量信息；

* **输出项目**

int（返回值）：方法执行结果；

##### 查询历史事件

* **接口**

int ReadHistroyEvent(const SYSTEMTIME\* pstStart, const SYSTEMTIME\* pstEnd, std::vector<shared\_ptr<KEventInfo>> &vecEvent)；

* **功能**

根据客户端请求信息处理子系统传过来的历史事件查询条件，通过RDBS服务器获取到结果并传给客户端请求信息处理子系统；

* **输入项目**

ptStarTime：要查询的数据的开始时间（查询条件）；

ptEndTime：要查询的数据的结束时间（查询条件）；

* **输出项目**

vecEvent：查询到的0-N个历史事件信息；

int（返回值）：方法执行结果；

##### 查询历史告警

* **接口**

int ReadHistroyAlarm(const SYSTEMTIME\* pstStart, const SYSTEMTIME\* pstEnd, std::vector<shared\_ptr<KAlarmInfo>> &vecAlarm)；

* **功能**

根据客户端请求信息处理子系统传过来的历史告警查询条件，通过RDBS服务器获取到结果并传给客户端请求信息处理子系统；

* **输入项目**

ptStarTime：要查询的数据的开始时间（查询条件）；

ptEndTime：要查询的数据的结束时间（查询条件）；

* **输出项目**

vecAlarm：查询到的0-N个历史告警信息；

int（返回值）：方法执行结果；

##### 实时变量推送

* **接口**

HRESULT OnDataChange(LONG NumTags, SAFEARRAY\* pTagIDs, SAFEARRAY\* pTimes, SAFEARRAY\* pValues)；

* **功能**

回调函数，接收RDBS服务器的实时变量数据；

* **输入项目**

NumTags：变量数量；

pTagIDs：变量ID集合；

pTimes：变量时间集合；

pValues：变量值的集合；

* **输出项目**

HRESULT（返回值）：执行结果；

##### 实时事件推送

* **接口**

HRESULT OnEventsChange(LONG EvtCount, SAFEARRAY\* pEvtTypes, SAFEARRAY\* pEvtIDs, SAFEARRAY\* pUserNames, SAFEARRAY\* pAccessLevels, SAFEARRAY\* pEvtTimes, SAFEARRAY\* pEvtSrc)；

* **功能**

回调函数，接收RDBS服务器的实时事件数据；

* **输入项目**

EvtCount：推送的事件信息数量；

pEvtTypes：事件类型的集合；

pEvtIDs：事件ID的集合；

pUserNames：产生事件的用户名的集合；

pAccessLevels：访问权限的集合；

pEvtTimes：事件事件的集合；

pEvtSrc：事件来源的集合；

* **输出项目**

HRESULT（返回值）：执行结果；

##### 实时告警推送

* **接口**

HRESULT OnAlarmsChange(LONG AlmCount, SAFEARRAY\* pAlmTypes, SAFEARRAY\* pAlmIDs, SAFEARRAY\* pTagIDs, SAFEARRAY\* pTagName, SAFEARRAY\* pTagValues, SAFEARRAY\* pAlmTimes, SAFEARRAY\* pAlmPriorities, SAFEARRAY\* pAck, SAFEARRAY\* pLimits, SAFEARRAY\* pStatus, SAFEARRAY\* pSources, SAFEARRAY\* pReserved)；

* **功能**

回调函数，接收RDBS服务器的实时告警数据；

* **输入项目**

AlmCount：推送来的告警信息的数量；

pAlmTypes：告警类型的集合；

pAlmIDs：告警ID的集合；

pTagIDs：全局变量ID的集合；

pTagName：变量名的集合；

pTagValues：变量值的集合；

pAlmTimes：告警时间的集合；

pAlmPriorities：告警优先级的集合；

pAck：是否应答的集合；

pLimits：报警限的集合（用来区分高低报警的阈值）；

pStatus：报警状态的集合(TRUE:报警, FALSE: 恢复)；

pSources：报警来源的集合；

pReserved：备用描述信息的集合；

* **输出项目**

HRESULT（返回值）：执行结果；

##### ATL对象销毁请求

* **接口**

HRESULT ShutdownRequest(BSTR bstrReason)；

* **功能**

回调函数，接收RDBS服务器的停止请求；

* **输入项目**

bstrReason：RDBS服务器传输过来的信息（停止的理由）；

* **输出项目**

HRESULT（返回值）：方法执行结果；

##### ATL对象状态请求

* **接口**

HRESULT IsActive(VARIANT\_BOOL\* bActive)；

* **功能**

回调函数，用来让RDBS服务器获取到WebSocket服务器的ATL对象的状态；

* **输入项目**

无；

* **输出项目**

bActive：获取到的WebSocket服务器的ATL对象是否是活动状态；

HRESULT（返回值）：方法执行结果；

### 实时数据缓冲模块

#### 功能概述

缓存通过RDBS服务器通信模块中被RDBS服务器推送过来的实时数据，并将这些信息给消息推送子系统使用；

#### 详细类图



#### 成员变量

* **CRealTimeTagOperate类**

m\_queTag：用来缓存被添加进来的实时变量信息；

m\_pLock：用来保证m\_queTag操作的原子性；

* **CRealTimeEvtOperate类（暂时未用，后期扩展使用）**

m\_queEvt：用来缓存被添加进来的实时事件信息；

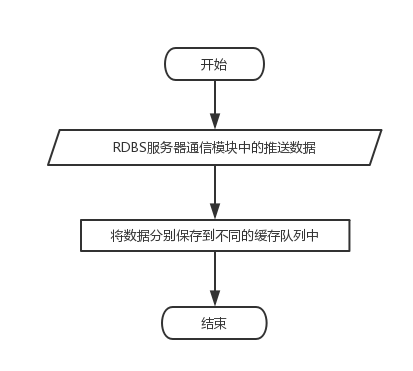
m\_pLock：用来保证m\_ queEvt操作的原子性；

* **CRealTimeAlmOperate类**

m\_queAlm：用来缓存被添加进来的实时告警信息；

m\_pLock：用来保证m\_ queAlm操作的原子性；

#### 流程图



#### 方法实现

##### 缓存实时变量信息

* **接口**

bool SetData(shared\_ptr<KTagInfo> ptTag)；

* **功能**

将RDBS服务器通信模块中被RDBS服务器推送上来的实时变量信息缓存到CRealTimeTagOperate的对象中；

* **输入项目**

ptTag：要缓存的实时变量信息；

* **输出项目**

bool（返回值）：执行是否成功；

##### 取出实时变量信息

* **接口**

shared\_ptr<KTagInfo> GetData()；

* **功能**

将CRealTimeTagOperate的对象中缓存的实时变量信息取出来，通过消息推送子系统推送出去；

* **输入项目**

无；

* **输出项目**

shared\_ptr<KTagInfo>（返回值）：取出来的实时变量信息；

##### 缓存实时事件信息

* **接口**

bool SetData(shared\_ptr<KEventInfo> ptEvt)；

* **功能**

将RDBS服务器通信模块中被RDBS服务器推送上来的实时事件信息缓存到CRealTimeEvtOperate的对象中（这个方法暂时没有使用，后面扩展使用）；

* **输入项目**

ptEvt：要缓存的实时事件信息；

* **输出项目**

bool（返回值）：执行是否成功；

##### 取出实时事件信息

* **接口**

shared\_ptr<KEventInfo> GetData()；

* **功能**

将CRealTimeEvtOperate的对象中缓存的实时事件信息取出来，通过消息推送子系统推送出去（这个方法暂时没有使用，后面扩展使用）；

* **输入项目**

无

* **输出项目**

shared\_ptr< KEventInfo >（返回值）：取出来的实时事件信息；

##### 缓存实时告警信息

* **接口**

bool SetData(shared\_ptr<KAlarmInfo> ptAlm)；

* **功能**

将RDBS服务器通信模块中被RDBS服务器推送上来的实时告警信息缓存到CRealTimeAlmOperate的对象中；

* **输入项目**

ptAlm：要缓存的实时告警信息；

* **输出项目**

bool（返回值）：执行是否成功；

##### 取出实时告警信息

* **接口**

shared\_ptr<KAlarmInfo> GetData()；

* **功能**

将CRealTimeAlmOperate的对象中缓存的实时告警信息取出来，通过消息推送子系统推送出去；

* **输入项目**

无；

* **输出项目**

shared\_ptr< KAlarmInfo >（返回值）：取出来的实时告警信息；

# 附录1

## WebSocketServer前后台通讯协议

