|  |  |
| --- | --- |
| 渐变_logo_2中 |  |
|  |  |
|  |  |
|  |  |
|  |  |

金融基础件2.0接入

T2SDK C#版开发手册

|  |  |
| --- | --- |
| 文档版本 | V2.0 |
| 发布日期 | 2013-06-27 |

**修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 日期 | 修改者 | 修改说明 |
| V2.0 | 20130627 | 许欣芃 | 创建 |

目 录

目录

[目 录 3](#_Toc378603214)

[前 言 8](#_Toc378603215)

[产品简介 8](#_Toc378603216)

[读者对象 8](#_Toc378603217)

[手册概况 8](#_Toc378603218)

[缩略语/术语 8](#_Toc378603219)

[1. 开发包简介 10](#_Toc378603220)

[2. 开发流程 11](#_Toc378603221)

[2.1 同步开发流程 11](#_Toc378603222)

[2.2 异步开发流程 11](#_Toc378603223)

[3. 开发接口 12](#_Toc378603224)

[3.1 配置类【CT2ConfigInterface】 12](#_Toc378603225)

[3.1.1 读取配置文件（Load） 12](#_Toc378603226)

[3.1.2 保存配置文件（Save） 15](#_Toc378603227)

[3.1.3 获取字符型配置项值（GetString） 15](#_Toc378603228)

[3.1.4 获取整型配置项值（GetInt） 16](#_Toc378603229)

[3.1.5 设置字符型配置项值（SetString） 16](#_Toc378603230)

[3.1.6 设置整型配置项值（SetInt） 16](#_Toc378603231)

[3.2 连接回调接口【CT2CallbackInterface】 17](#_Toc378603232)

[3.2.1 TCP连接成功回调（OnConnect） 17](#_Toc378603233)

[3.2.2 安全连接成功回调（OnSafeConnect） 17](#_Toc378603234)

[3.2.3 注册成功回调（OnRegister） 17](#_Toc378603235)

[3.2.4 连接断开回调（OnClose） 18](#_Toc378603236)

[3.2.5 数据发送回调（OnSent） 18](#_Toc378603237)

[3.2.6 收到消息回调（OnReceivedBiz） 19](#_Toc378603238)

[3.2.7 收到消息扩展回调（OnReceivedBizEx） 19](#_Toc378603239)

[3.2.8 收到业务消息回调（OnReceivedBizMsg） 20](#_Toc378603240)

[3.3 业务消息类【CT2BizMessage】 21](#_Toc378603241)

[3.3.1 设置功能号（SetFunction） 21](#_Toc378603242)

[3.3.2 获取功能号（GetFunction） 21](#_Toc378603243)

[3.3.3 设置包类型（SetPacketType） 22](#_Toc378603244)

[3.3.4 获取包类型（GetPacketType） 22](#_Toc378603245)

[3.3.5 设置营业部号（SetBranchNo） 22](#_Toc378603246)

[3.3.6 获取营业部号（GetBranchNo） 22](#_Toc378603247)

[3.3.7 设置系统号（SetSystemNo） 23](#_Toc378603248)

[3.3.8 获取系统号（GetSystemNo） 23](#_Toc378603249)

[3.3.9 设置子系统号（SetSubSystemNo） 23](#_Toc378603250)

[3.3.10 获取子系统号（GetSubSystemNo） 24](#_Toc378603251)

[3.3.11 设置发送者编号（SetSenderId） 24](#_Toc378603252)

[3.3.12 获取发送者编号（GetSenderId） 24](#_Toc378603253)

[3.3.13 设置包序号（SetPacketId） 24](#_Toc378603254)

[3.3.14 获取包序号（GetPacketId） 25](#_Toc378603255)

[3.3.15 设置目的地路由（SetTargetInfo） 25](#_Toc378603256)

[3.3.16 获取目的地路由（GetTargetInfo） 26](#_Toc378603257)

[3.3.17 设置发送者路由（SetSendInfo） 26](#_Toc378603258)

[3.3.18 获取发送者路由（GetSendInfo） 26](#_Toc378603259)

[3.3.19 设置错误号（SetErrorNo） 26](#_Toc378603260)

[3.3.20 获取错误号（GetErrorNo） 27](#_Toc378603261)

[3.3.21 设置错误信息（SetErrorInfo） 27](#_Toc378603262)

[3.3.22 获取错误信息（GetErrorInfo） 27](#_Toc378603263)

[3.3.23 设置返回码（SetReturnCode） 28](#_Toc378603264)

[3.3.24 获取返回码（GetReturnCode） 28](#_Toc378603265)

[3.3.25 设置业务内容（SetContent） 28](#_Toc378603266)

[3.3.26 获取业务内容（GetContent） 29](#_Toc378603267)

[3.3.27 设置订阅类型（SetIssueType） 29](#_Toc378603268)

[3.3.28 获取订阅类型（GetIssueType） 30](#_Toc378603269)

[3.3.29 设置订阅序号（SetSequeceNo） 30](#_Toc378603270)

[3.3.30 获取订阅序号（GetSequeceNo） 30](#_Toc378603271)

[3.3.31 设置订阅关键字段（SetKeyInfo） 31](#_Toc378603272)

[3.3.32 获取订阅关键字段（GetKeyInfo） 31](#_Toc378603273)

[3.3.33 设置附加数据（SetAppData） 32](#_Toc378603274)

[3.3.34 获取附加数据（GetAppData） 32](#_Toc378603275)

[3.3.35 请求转换成应答（ChangeReq2AnsMessage） 32](#_Toc378603276)

[3.3.36 消息转换成二进制流（GetBuff） 33](#_Toc378603277)

[3.3.37 二进制流转换成消息（SetBuff） 33](#_Toc378603278)

[3.4 打包器类【CT2Packer】 33](#_Toc378603279)

[3.4.1 设置缓存区空间（SetBuffer） 33](#_Toc378603280)

[3.4.2 开始打包（BeginPack） 34](#_Toc378603281)

[3.4.3 构造新结果集（NewDataset） 34](#_Toc378603282)

[3.4.4 添加新字段（AddField） 34](#_Toc378603283)

[3.4.5 添加string型数据（AddStr） 35](#_Toc378603284)

[3.4.6 添加int型数据（AddInt） 35](#_Toc378603285)

[3.4.7 添加double型数据（AddDouble） 35](#_Toc378603286)

[3.4.8 添加char型数据（AddChar） 36](#_Toc378603287)

[3.4.9 添加二进制数据（AddRaw） 36](#_Toc378603288)

[3.4.10 结束打包（EndPack） 36](#_Toc378603289)

[3.4.11 获取打包结果指针（GetPackBuf） 37](#_Toc378603290)

[3.4.12 获取打包结果长度（GetPackLen） 37](#_Toc378603291)

[3.4.13 获取打包缓存长度（GetPackBufSize） 37](#_Toc378603292)

[3.4.14 获取打包版本（GetVersion） 37](#_Toc378603293)

[3.4.15 设置返回码（SetReturnCode） 38](#_Toc378603294)

[3.4.16 获取解包接口（UnPack） 38](#_Toc378603295)

[3.4.17 释放打包内存（FreeMem） 38](#_Toc378603296)

[3.4.18 清空最后一个结果集数据（ClearValue） 39](#_Toc378603297)

[3.4.19 开始打包扩展接口（BeginPackEx） 39](#_Toc378603298)

[3.5 解包器类【CT2UnPacker】 39](#_Toc378603299)

[3.5.1 获取解包器版本（GetVersion） 39](#_Toc378603300)

[3.5.2 二进制转换为解包器（Open） 40](#_Toc378603301)

[3.5.3 获取结果集个数（GetDatasetCount） 40](#_Toc378603302)

[3.5.4 根据下标设置当前结果集（SetCurrentDatasetByIndex） 40](#_Toc378603303)

[3.5.5 根据结果集名字设置当前结果集（SetCurrentDataset）、 40](#_Toc378603304)

[3.5.6 获取解包器缓存指针（GetPackBuf） 41](#_Toc378603305)

[3.5.7 获取解包器缓存长度（GetPackLen） 41](#_Toc378603306)

[3.5.8 获取当前结果集行数（GetRowCount） 41](#_Toc378603307)

[3.5.9 指向结果集第一行记录（First） 42](#_Toc378603308)

[3.5.10 指向结果集最后一行记录（Last） 42](#_Toc378603309)

[3.5.11 跳转到结果集任意行（Go） 42](#_Toc378603310)

[3.5.12 获取列个数（GetColCount） 42](#_Toc378603311)

[3.5.13 根据下标获取列名字（GetColName） 43](#_Toc378603312)

[3.5.14 获取列类型（GetColType） 43](#_Toc378603313)

[3.5.15 获取列精度（GetColScale） 43](#_Toc378603314)

[3.5.16 获取列最大长度（GetColWidth） 44](#_Toc378603315)

[3.5.17 根据列名字找到列下标（FindColIndex） 44](#_Toc378603316)

[3.5.18 根据列下标获取string类型值（GetStrByIndex） 44](#_Toc378603317)

[3.5.19 根据列名字获取string类型值（GetStr） 44](#_Toc378603318)

[3.5.20 根据列下标获取char类型值（GetCharByIndex） 45](#_Toc378603319)

[3.5.21 根据列名字获取char类型值（GetChar） 45](#_Toc378603320)

[3.5.22 根据列下标获取double类型值（GetDoubleByIndex） 45](#_Toc378603321)

[3.5.23 根据列名字获取double类型值（GetDouble） 45](#_Toc378603322)

[3.5.24 根据列下标获取int类型值（GetIntByIndex） 46](#_Toc378603323)

[3.5.25 根据列名字获取int类型值（GetInt） 46](#_Toc378603324)

[3.5.26 根据列下标获取二进制类型值（GetRawByIndex） 46](#_Toc378603325)

[3.5.27 根据列名字获取二进制类型值（GetRaw） 47](#_Toc378603326)

[3.5.28 最后一次取得值是否为空（WasNull） 47](#_Toc378603327)

[3.5.29 结果集下一条记录（Next） 47](#_Toc378603328)

[3.5.30 结果集是否到结尾（IsEOF） 47](#_Toc378603329)

[3.5.31 结果集是否为空（IsEmpty） 48](#_Toc378603330)

[3.5.32 结果集释放（Destroy） 48](#_Toc378603331)

[3.6 过滤器类【CT2FilterInterface】 49](#_Toc378603332)

[3.6.1 根据下标获取条件名字（GetFilterNameByIndex） 49](#_Toc378603333)

[3.6.2 根据下标获取条件值（GetFilterValueByIndex） 49](#_Toc378603334)

[3.6.3 根据条件名字好获取条件值（GetFilterValue） 49](#_Toc378603335)

[3.6.4 获取过滤条件个数（GetCount） 50](#_Toc378603336)

[3.6.5 设置过滤条件名字和值（SetFilter） 50](#_Toc378603337)

[3.7 订阅参数类【CT2SubscribeParamInterface】 50](#_Toc378603338)

[3.7.1 设置主题名字（SetTopicName） 50](#_Toc378603339)

[3.7.2 设置附加数据（SetAppData） 51](#_Toc378603340)

[3.7.3 设置过滤条件（SetFilter） 51](#_Toc378603341)

[3.7.4 设置返回字段（SetReturnFiled） 51](#_Toc378603342)

[3.7.5 设置补缺标志（SetFromNow） 52](#_Toc378603343)

[3.7.6 设置覆盖标志（SetReplace） 52](#_Toc378603344)

[3.7.7 设置发送间隔（SetSendInterval） 52](#_Toc378603345)

[3.7.8 获取主题名字（GetTopicName） 53](#_Toc378603346)

[3.7.9 获取附加数据（GetAppData） 53](#_Toc378603347)

[3.7.10 根据下标获取过滤字段名字（GetFilterNameByIndex） 53](#_Toc378603348)

[3.7.11 根据下标获取过滤字段值（GetFilterValueByIndex） 53](#_Toc378603349)

[3.7.12 根据过滤名字获取值（GetFilterValue） 54](#_Toc378603350)

[3.7.13 获取过滤条件个数（GetFilterCount） 54](#_Toc378603351)

[3.7.14 获取返回字段（GetReturnFiled） 54](#_Toc378603352)

[3.7.15 获取补缺标志（GetFromNow） 55](#_Toc378603353)

[3.7.16 获取覆盖标志（GetReplace） 55](#_Toc378603354)

[3.7.17 获取发送间隔（GetSendInterval） 55](#_Toc378603355)

[3.8 订阅回调接口【CT2SubCallbackInterface】 55](#_Toc378603356)

[3.8.1 收到发布消息回调（OnReceived） 56](#_Toc378603357)

[3.8.2 收到剔除订阅回调（OnRecvTickMsg） 56](#_Toc378603358)

[3.9 订阅类【CT2SubscribeInterface】 57](#_Toc378603359)

[3.9.1 订阅主题 57](#_Toc378603360)

[3.9.2 取消订阅 58](#_Toc378603361)

[3.9.3 取消订阅扩展接口 58](#_Toc378603362)

[3.9.4 获取已经订阅的信息 58](#_Toc378603363)

[3.10 发布类【CT2PublishInterface】 59](#_Toc378603364)

[3.10.1 业务发包格式发送 59](#_Toc378603365)

[3.10.2 二进制数据发送 60](#_Toc378603366)

[3.11 连接类【CT2Connection】 61](#_Toc378603367)

[3.11.1 初始化连接对象（Create） 61](#_Toc378603368)

[3.11.2 开始连接/注册（Connect） 61](#_Toc378603369)

[3.11.3 断开连接（Close） 62](#_Toc378603370)

[3.11.4 取连接服务器地址（GetServerAddress） 62](#_Toc378603371)

[3.11.5 取连接状态（GetStatus） 62](#_Toc378603372)

[3.11.6 取服务器负载（GetServerLoad） 63](#_Toc378603373)

[3.11.7 取错误码对应的错误信息（GetErrorMsg） 63](#_Toc378603374)

[3.11.8 取连接错误号（GetConnectError） 64](#_Toc378603375)

[3.11.9 发送业务数据（SendBiz） 64](#_Toc378603376)

[3.11.10 接收业务数据（RecvBiz） 64](#_Toc378603377)

[3.11.11 发送业务数据扩展接口（SendBizEx） 65](#_Toc378603378)

[3.11.12 接收业务数据扩展接口（RecvBizEx） 66](#_Toc378603379)

[3.11.13 连接初始化的扩展接口（CreateEx） 67](#_Toc378603380)

[3.11.14 创建订阅者（NewSubscriber） 68](#_Toc378603381)

[3.11.15 获取发布者（NewPublisher） 68](#_Toc378603382)

[3.11.16 获取主题信息（GetTopic） 69](#_Toc378603383)

[3.11.17 获取消息中心的最后错误（GetMCLastError） 70](#_Toc378603384)

[3.11.18 连接用IBizMessage接口初始化（Create2BizMsg） 70](#_Toc378603385)

[3.11.19 连接发送IBizMessage业务消息（SendBizMsg） 70](#_Toc378603386)

[3.11.20 连接接收IBizMessage业务消息（RecvBizMsg） 71](#_Toc378603387)

[4. 注意事项 71](#_Toc378603388)

[4.1 相关限制 71](#_Toc378603389)

[4.2 调用限制 72](#_Toc378603390)

[4.3 回调线程 72](#_Toc378603391)

[4.4 编程建议 72](#_Toc378603392)

[5. 示例代码 73](#_Toc378603393)

[5.1 建立连接 73](#_Toc378603394)

[5.2 断开连接 73](#_Toc378603395)

[5.3 同步发送接收 74](#_Toc378603396)

[5.4 异步发送接收 76](#_Toc378603397)

[5.5 订阅 79](#_Toc378603398)

[5.6 发布 84](#_Toc378603399)

[6. 附录 86](#_Toc378603400)

[6.1 T2sdk错误码 86](#_Toc378603401)

[6.2 INI配置文件说明 88](#_Toc378603402)

前 言

产品简介

C#版本的开发包库名称为ext\_t2sdkEx，这个库是在C++版本的t2sdk基础上，进行了托管代码封装，所以ext\_t2sdkEx库必须依赖于t2sdk库。

读者对象

本指南主要适用于以下人员：

* 开发人员，仅限于C#开发，对接各种在CRES2.0平台基础上搭建的系统，譬如：UFX，UF2.0等等。

手册概况

本手册各章节内容如下表所示。

|  |  |
| --- | --- |
| 章节 | 内容 |
| 1开发包简介 | 介绍开发包的支持系统，获取方式 |
| 2开发流程 | 介绍开发的步骤，异步同步的开发模式 |
| 3 开发接口 | 具体的开发接口的函数说明 |
| 4示例代码 | 开发的demo的实例代码 |
| 5 使用注意事项 | 介绍接口调用时禁止的方式 |

缩略语/术语

下面列出了本手册中出现的缩略语和术语。

|  |  |  |
| --- | --- | --- |
| **C** |  |  |
| CRES2.0 | 金融基础件2.0 | 恒生的新一代金融基础件平台，本文档介绍的开发包就是接入此平台的C++版本的开发包。下面说的服务端不特殊说明都是指CRES2.0. |
| **U** |  |  |
| UFX | 恒生统一接入系统 | 客户端的统一接入控制系统，是基于CRES2.0平台之上搭建，用T2SDK可以接入。 |
| UF2.0 | 恒生经济运营平台 | 恒生的证券经纪运营平台系统，是基于CRES2.0平台之上搭建，用T2SDK可以接入。 |
| M |  |  |
| MC2.0 | 消息中心2.0 | 本文档提到的订阅发布接口都是针对消息中心2.0,1.0的订阅发布不能用这个接口，需要自己开发代码 |
| MC1.0 | 消息中心1.0 | 消息中心1.0需要通过业务消息中增加字段来订阅，没有现成的接口调用。 |

# 开发包简介

名称：ext\_t2sdkEx.dll

开发语言：C#

支持操作系统：windows

**版本信息获取：**

通过右键查看属性，在详细信息的界面上面会显示库的具体版本信息。

**获取方式：**

* 恒生内部的员工，以项目组的形式向研发中心客服申请使用。
* 恒生外部开发商，向对应的项目的恒生接口人获取开发包。

# 开发流程

本章节主要介绍两种开发的模式，同步、异步。

* 1. 同步开发流程



* 1. 异步开发流程



# 开发接口

本章主要介绍开发包提供的所有类，以及类成员函数参数说明。

* 1. 配置类【CT2ConfigInterface】

这个类所属的命名空间是：**hundsun.t2sdk**。这个类的对象实例，通过new方法来构造。

### 读取配置文件（Load）

**函数原型：**

public int Load(string szFileName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szFileName | 连接属性的配置文件 | 输入参数 |

**返回：**

加载成功返回0，其他值表示失败

**用法说明：**

这里加载的文件是ini格式的文件，下面是文件的简单实例：

[t2sdk]

;许可证文件

license\_file=

;服务器地址

servers=192.168.194.56:6002;192.168.194.56:8001

;接受缓存初始化大小

init\_recv\_buf\_size=512

;发送缓存初始化大小

init\_send\_buf\_size=512

;发送队列

send\_queue\_size=1000

[safe]

;安全级别

safe\_level=none

;客户号

client\_id=123456

;客户密码

comm\_pwd=888888

;ssl证书

cert\_file=xxx.pem

;ssl证书密码

cert\_pwd=xxxxxxxx

每个标签以及属性都有自己的含义，下面是一些通用标签和含义的解释：

|  |  |  |
| --- | --- | --- |
| 标签名 | 属性名 | 含义 |
| [t2sdk] |  | 基本参数 |
|  | servers | 客户端要连接的服务端的IP和端口，可以配置一个或多个，中间以;分割，第一次连接时，会从这串服务端地址中，随即选择一个地址来尝试建立连接； 再次连接时，会取当前地址的后一个地址来尝试建立连接，以此类推 |
|  | lang | 语言取值及含义如下：  2052：中文（缺省）  1033：英文 |
|  | errormsg | 用户指定错误信息文件，错误号从-70到0，缺省为””。 |
|  | init\_recv\_buf\_size | 接收缓存的初始大小，单位字节，实际接收到服务端的数据时，可能会扩大（如果需要） |
|  | init\_send\_buf\_size | 每块发送缓存的初始大小，单位字节，该大小也会根据需要动态扩大 |
|  | send\_queue\_size | 发送队列的大小，该大小不会动态变化，若该配置项很小，且连接发包很频繁，则可能因为发送队列满而造成发送失败 |
|  | license\_file | 此配置项配置许可证文件的路径，如果没有配置，则默认为当前目录下的license.dat。该许可证需向恒生电子股份有限公司申请。 |
|  | heartbeat\_time | 客户端给服务端发送心跳的间隔时间，单位为秒，最小值为5秒，不配或者配<=0，表示不开启客户端心跳 |
|  | if\_error\_log | 是否记录错误日志的参数，1表示记录，0表示不记录，默认不记录。 |
|  | logdir | 记录日志的目录，默认在程序运行的当前目录，只有开启记录日志，才会记录 |
|  | if\_sendRecv\_log | 记录发包和收包的日志，用于客户端查找问题，1表示记录，0表示不记录，默认不记录 |
|  | login\_name | 有名客户端的名字，默认是匿名 |
|  | license\_pwd | 许可证的加密密码，用于UFX接入时，默认没有密码 |
|  | event\_count | 指定事件个数，缺省为10。 |
|  | connect\_time | 指定连接超时，缺省为5。 |
|  | support\_multi | 指定是否支持多线程同步调用，0:不支持（缺省）非0:支持。 |
|  | enable\_nagle | 指定是否禁用Nagle算法，1:禁用，非1:开启（缺省）。 |
|  | addipmac | 指定是否在注册包中加IP和MAC，1:是（缺省），非1:否。 |
|  | machine\_code | 指定机器码。 |
|  | auto\_reconnect | 指定是否自动重连，1:是，非1:否（缺省）。 |
| [safe] |  | 连接相关 |
|  | safe\_level | 连接的安全模式，可以选择明文（none），通信密码（pwd），SSL（ssl）,注意大小写敏感 |
|  | client\_id | 当连接的安全模式为pwd时，client\_id配置项才生效,客户号，一个客户号对应一个密码 |
|  | comm\_pwd | 在pwd模式下，当client\_id为空时，comm\_pwd为默认的密钥; 注意当client\_id为空时，comm\_pwd必须和服务端的配置一致方可正常使用 |
|  | ca\_file | 指定客户端的证书（根证书，校验服务端用） |
|  | ca\_pwd | 若证书为\*.pfx，则需要配ca\_pwd，若为\*.pem，则可配可不配 |
|  | cert\_file | cert\_file配置被服务端校验的证书路径和密码 |
|  | cert\_pwd | 不管格式如何，cert\_pwd必配 |
|  | check\_server\_cert | 是否校验服务端，可不配，若不配，默认校验服务端，0表示不校验。1表示校验 |
|  | version | 指定SSL版本，缺省为””。 |
|  | verify\_depth | 指定校验深度，缺省为0。 |
| [proxy] |  | 代理参数 |
|  | proxy\_type | 可配置采用何种代理配置http/socks4/socks5，空表示不采用代理 |
|  | ip | 若proxy\_type不为空，下面配置生效; 代理服务器的IP地址 |
|  | port | 代理服务器的端口 |
|  | user\_name | 登陆代理服务器的用户名 |
|  | password | 登陆代理服务器的密码 |

### 保存配置文件（Save）

**函数原型：**

public int Save(string szFileName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szFileName | 需要保存属性的配置文件 | 输入参数 |

**返回：**

保存成功返回0，其他值表示失败

**用法说明：**

这个函数是把属性内存落地成文件，文件格式如上面加载的文件。

### 获取字符型配置项值（GetString）

**函数原型：**

public string GetString(string szSection, string szEntry, string szDefault)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szSection | 节名 | 输入参数 |
| string szEntry | 变量名 | 输入参数 |
| string szDefault | 默认值 | 输入参数 |

**返回：**

获取ini配置里面相应的节点名和变量名的属性值，用字符串形式返回。

**用法说明：**

[t2sdk]

license\_file=

上面t2sdk就是节点名，license\_file就是属性名，如果配置中没有节点名和属性名，用第三个参数作为返回值。

### 获取整型配置项值（GetInt）

**函数原型：**

public int GetInt(string szSection, string szEntry, int iDefault)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szSection | 节名 | 输入参数 |
| string szEntry | 变量名 | 输入参数 |
| int iDefault | 默认值 | 输入参数 |

**返回：**

获取ini配置里面相应的节点名和变量名的属性值，用整型形式返回。

**用法说明：**

节点名和属性名定义如上面所示，同样的，如果不存在这个属性，就返回第三个参数。

### 设置字符型配置项值（SetString）

**函数原型：**

public int SetString(string szSection, string szEntry, string szValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szSection | 节名 | 输入参数 |
| string szEntry | 变量名 | 输入参数 |
| string szValue | 需要设置的值 | 输入参数 |

**返回：**

设置字符串型属性值到对应的节点和变量中。返回0表示设置成功，其他表示失败。

**用法说明：**

通过代码设置属性到配置项中，可以不需要读取配置文件。

### 设置整型配置项值（SetInt）

**函数原型：**

public int SetInt(string szSection, string szEntry, int iValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szSection | 节名 | 输入参数 |
| string szEntry | 变量名 | 输入参数 |
| int iValue | 需要设置的值 | 输入参数 |

**返回：**

设置整型属性值到对应的节点和变量中。返回0表示设置成功，其他表示失败。

**用法说明：**

下面代码展示如果通过代码设置连接属性，不需要读取配置文件，当然读取文件之后，也是可以再设置属性，属性设置是覆盖的：

CT2ConfigInterface config = new CT2Configinterface();

config.SetString("t2sdk", "servers", "192.168.94.30:9999");

config.SetString("t2sdk", "license\_file", "license.dat");

config.SetInt("t2sdk", "send\_queue\_size", 100);

config.SetString("safe", "safe\_level", "none");

* 1. 连接回调接口【CT2CallbackInterface】

如果是异步的模式，需要完成的回调接口定义。上层业务代码需要定义类继承这个接口，然后实现相应的功能。这个接口所属的命名空间是：**hundsun.t2sdk**。

### TCP连接成功回调（OnConnect）

这个函数在TCP连接成功之后就会回调。

**函数原型：**

public abstract void OnConnect(hundsun.t2sdk.CT2Connection lpConnection)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |

**用法说明：**

在SOCKET套接字建立成功之后，回调这个函数，这个回调里面一般也就是简单的展示连接信息，无法进行任何操作。

### 安全连接成功回调（OnSafeConnect）

这个函数在安全连接成功之后就会回调，安全连接就是指，密码安全模式下，密码交换成功之后；或者SSL连接建立成功之后。

**函数原型：**

public abstract void OnSafeConnect(hundsun.t2sdk.CT2Connection lpConnection)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |

**用法说明：**

在安全连接建立成功之后，回调这个函数，这个回调里面一般也就是简单的展示连接信息，无法进行任何操作。

### 注册成功回调（OnRegister）

这个函数在成功收到服务端注册应答之后就会回调，注册动作时发生在安全连接之后。

**函数原型：**

public abstract void OnRegister(hundsun.t2sdk.CT2Connection lpConnection)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |

**用法说明：**

在注册成功之后，回调这个函数，这个回调里面可以通知其他线程已经可以开始干活了，连接已经建立，可以发包收包做业务操作了。

### 连接断开回调（OnClose）

这个函数在连接断开之后，会进入这个回调，连接断开可能是服务器断开，或者客户端检测到连接断开，等等。

**函数原型：**

public abstract void OnClose(hundsun.t2sdk.CT2Connection lpConnection)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |

**用法说明：**

连接断开回调里面，可以自己重新连接，但是不可以在这个回调函数里面直接调用连接的connect函数，需要通过事件通知，其他线程进行重连，当然也可以不进行重连，开发包底层会在下一次发包之前判断连接状态，然后就会进行重连。

### 数据发送回调（OnSent）

这个函数在连接上有数据发送成功之后，就会回调这个函数。

**函数原型：**

public abstract void OnSent(hundsun.t2sdk.CT2Connection lpConnection, int hSend, void\* lpData, int nLength, int nQueuingData)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接指针 | 回调 |
| int hSend | 发送的句柄，连接的发送函数成功之后返回的句柄 | 回调 |
| void\* lpData | 发送数据 | 回调 |
| int nLength | 发送数据的长度 | 回调 |
| int nQueuingData | 剩余发送队列长度 | 回调 |

**用法说明：**

这个回调是在连接成功发送数据之后回调，主要是为了告诉上层应用需要发送的数据已经成功发送。是通过参数hsend来匹配是哪部分数据发送成功。这个回调里面一般不建议做业务操作，最多就展示。

### 收到消息回调（OnReceivedBiz）

这个函数在连接上有数据接收成功之后，就会回调这个函数。下面介绍的两个也是同样的回调，只是返回的内容多少有所不同，回调接口只会调用其中一个，具体调用哪一个回调，是由连接的Creat函数决定的。这个回调是调用连接接口的Create函数。

**函数原型：**

public abstract void OnReceivedBiz(hundsun.t2sdk.CT2Connection lpConnection, int hSend, string lppStr, hundsun.t2sdk.CT2UnPacker lppUnPacker, int nResult)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |
| int hSend | 发送的句柄，这表示接收到的这个数据是针对哪个请求的应答，通过这个句柄进行匹配 | 回调 |
| string lppStr | 返回错误的，错误信息 | 回调 |
| CT2UnPacker lppUnPacker | 正确应答的业务解包器对象 | 回调 |
| int nResult | 返回值，决定上一个参数的表达形式 | 回调 |

**用法说明：**

* 如果nResult等于0，表示业务数据接收成功，并且业务操作成功，lppUnPacker有对应的应答数据，lppStr为null。
* 如果nResult等于1，表示业务数据接收成功，但业务操作失败了，lppUnPacker有对应的应答数据，lppStr为null。
* 如果nResult等于2，表示收到非业务错误信息，lppStr为具体错误信息，但是lppUnPacker为null。
* 如果nResult等于3，表示业务包解包失败。lppStr为null，lppUnPacker为null。

这个回调主要就是把异步推送过来的消息交给上层处理，可能是之前发送的请求的应答， 也可以是服务端主推的消息，根据hsedn来匹配。在这个函数中，建议不要做太多的业务操作，否则会阻塞回调的线程，建议直接拷贝数据，然后入队列，其他线程从队列中获取数据处理。

### 收到消息扩展回调（OnReceivedBizEx）

这个函数和上面的函数一样，只是增加了一些返回的内容。触发这个回调，是调用连接接口的CreateEx函数。

**函数原型：**

public abstract void OnReceivedBizEx(hundsun.t2sdk.CT2Connection lpConnection, int hSend, hundsun.t2sdk.CT2RespondData lpRetData, string lppStr, hundsun.t2sdk.CT2UnPacker lppUnPacker, int nResult)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |
| int hSend | 发送的句柄，这表示接收到的这个数据是针对哪个请求的应答，通过这个句柄进行匹配 | 回调 |
| CT2RespondData lpRetData | 回调的应答对象，是应答消息中部分内容。 | 回调 |
| string lppStr | 返回错误的，错误信息 | 回调 |
| CT2UnPacker lppUnPacker | 正确应答的业务解包器对象 | 回调 |
| int nResult | 返回值，决定上一个参数的表达形式 | 回调 |

**用法说明：**

nResult的说明跟前面的接口一样，不再重复，这里介绍一下额外返回的业务体对象：

public class CT2RespondData

{

public string errorInfo; //错误信息

public int errorNo; //错误号

public int functionID; //返回的功能号

public int issueType; //MC1.0的订阅类型

public int keyInfoLen; //MC1.0的关键字段长度

public void\* lpKeyInfo;//MC1.0的消息关键字段，unpack格式的数据

public int returnCode; //返回码

public readonly CT2RouteInfo sendInfo;// 应答里面发送者信息

public CT2RespondData();

}

通过这个对象返回一些之前的OnReceivedBiz没有返回的信息，便于业务层的操作，尤其是需要消息中心1.0的订阅和发布的功能的时候，就必须使用这个回调接口了。

### 收到业务消息回调（OnReceivedBizMsg）

这个函数和上面的函数一样，返回了一个完整的业务消息，所有的应答内容都会在这个消息中包含。触发这个回调，是调用连接接口的Create2BizMsg函数。建议使用这个回调接口，因为包含所有需要的消息，比上面两个接口返回的字段都全。

**函数原型：**

public abstract void OnReceivedBizMsg(hundsun.t2sdk.CT2Connection lpConnection, int hSend, hundsun.t2sdk.CT2BizMessage lpMsg)

**回调参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Connection lpConnection | 当前回调产生的连接对象 | 回调 |
| int hSend | 发送的句柄，这表示接收到的这个数据是针对哪个请求的应答，通过这个句柄进行匹配 | 回调 |
| CT2BizMessage lpMsg | 返回的业务消息对象，通过对象函数可以访问需要关心的内容。 | 回调 |

**用法说明：**

这里返回消息接口的定义，下面有详细的介绍。如果程序是一个服务端，通过异步接收请求，然后处理完之后转成应答返回，可以调用消息接口的请求转应答操作。建议，所有的开发，以后都采用这个回调接口，方便业务的操作。

* 1. 业务消息类【CT2BizMessage】

业务消息接口，主要是返回消息中心各种需要关心的字段，当然发送消息也可以通过设置相关字段，发送请求。这个类所属的命名空间是：**hundsun.t2sdk**。

### 设置功能号（SetFunction）

**函数原型：**

public void SetFunction(int nFUnctionNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nFUnctionNo | 功能号 | 输入参数 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个功能号，所有消息都必须有的字段，后台根据功能号选择相应的业务逻辑处理。

### 获取功能号（GetFunction）

**函数原型：**

public int GetFunction()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的功能号

**用法说明：**

一般用于从应答消息中获取对应的功能号。

### 设置包类型（SetPacketType）

**函数原型：**

public void SetPacketType(int nPacketType)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nPacketType | 消息类型，0表示请求，1表示应答。没有其他值。 | 输入 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个消息类型，所有消息都必须有的字段，客户端一般是发请求，如果作为服务处理端，那就是接收请求，返回应答。

### 获取包类型（GetPacketType）

**函数原型：**

public int GetPacketType()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的消息类型，0表示请求，1表示应答。

**用法说明：**

一般用于客户端收到消息之后，根据消息类型，来选择不同的处理逻辑。

### 设置营业部号（SetBranchNo）

**函数原型：**

public void SetBranchNo(int nBranchNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nBranchNo | 营业部号 | 输入 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个营业部号，根据业务接口需要，来决定是否需要设置营业部号。

### 获取营业部号（GetBranchNo）

**函数原型：**

public int GetBranchNo()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的营业部号。

**用法说明：**

### 设置系统号（SetSystemNo）

**函数原型：**

public void SetSystemNo(int nSystemNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nSystemNo | 系统号 | 输入 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个系统号，根据业务接口需要，来决定是否需要设置系统号。系统号是指消息应该属于哪个系统处理，譬如：融资融券系统、集中交易系统等等。后台都会对系统编号，前台就根据后台规定传入规定的编号。这样才能保证业务能被后台系统处理。

### 获取系统号（GetSystemNo）

**函数原型：**

public int GetSystemNo()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的系统号。

**用法说明：**

### 设置子系统号（SetSubSystemNo）

**函数原型：**

public void SetSubSystemNo(int nSubSystemNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nSubSystemNo | 子系统号 | 输入 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个子系统号，根据业务接口需要，来决定是否需要设置子系统号。子系统号是系统号下面的详细分类，譬如：集中交易系统下面会分为，账户管理、交易、资金、历史、查询等等各个子系统。这些子系统也是由后台统一划分，前台根据接口说明传入对应的编号。

### 获取子系统号（GetSubSystemNo）

**函数原型：**

public int GetSubSystemNo()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的子系统号。

**用法说明：**

### 设置发送者编号（SetSenderId）

**函数原型：**

public void SetSenderId(int nSenderId)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nSenderId | 发送者编号 | 输入 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个发送者编号，发送者编号用于上层业务在异步模式下，匹配请求和应答，这个字段在请求中有上层业务系统自己送入，后台应答会原样返回。 也可以用在选择哪个线程处理，获取其他的应用中都可以。

### 获取发送者编号（GetSenderId）

**函数原型：**

public int GetSenderId()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的发送者编号。

**用法说明：**

根据这个编号可以来匹配请求，也可以选择相应线程来处理。

### 设置包序号（SetPacketId）

**函数原型：**

public void SetPacketId(int nPacketId)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nPacketId | 包序号 | 输入 |

**返回：**

没有返回值

**用法说明：**

为消息设置一个包序号，每个消息都有一个自己序号，上层应用可以自己设置这个包序号，然后应答根据包序号来匹配请求。如果上层应用没有设置包序号，开发包底层业务设置这个序号，底层设置的序号是连续递增的。一般如果没有特殊需要，这个包序号是不需要上层业务自己添加的。

### 获取包序号（GetPacketId）

**函数原型：**

public int GetPacketId()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的包序号。

**用法说明：**

### 设置目的地路由（SetTargetInfo）

**函数原型：**

public void SetTargetInfo(hundsun.t2sdk.CT2RouteInfo targetInfo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2RouteInfo targetInfo | 目的地路由信息 | 输入 |

**返回：**

没有返回值

**用法说明：**

如果知道这条消息应该去那个中间件节点处理，可以调用这个函数设置中间件的路由信息，路由信息结构体中具体的字段定义如下：

public class CT2RouteInfo

{

public int connectID; //连接号

public int memberNO; //成员编号

public string nbrName; //中间件节点的邻居名字

public string ospfName; //路由目标节点中间件名字

public string pluginID; //中间件插件名

public string svrName; //中间件的进程名字

public CT2RouteInfo();

}

### 获取目的地路由（GetTargetInfo）

**函数原型：**

public void GetTargetInfo(hundsun.t2sdk.CT2RouteInfo targetInfo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2RouteInfo targetInfo | 消息中的目标路由信息 | 输出 |

**返回：**

返回消息中对应的目标路由。

**用法说明：**

一般用于客户端在获取应答的目标路由时使用。

### 设置发送者路由（SetSendInfo）

**函数原型：**

public void SetSendInfo(hundsun.t2sdk.CT2RouteInfo sendInfo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2RouteInfo sendInfo | 发送者路由信息 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数传入参数和设置路由信息的参数是一致，主要用于客户端在收到请求之后，发送应答时可以设置自己的路由信息作为发送者信息，便于服务端知道应答是由哪个客户端发起的。

### 获取发送者路由（GetSendInfo）

**函数原型：**

public void GetSendInfo(hundsun.t2sdk.CT2RouteInfo sendInfo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2RouteInfo sendInfo | 消息中的发送者路由信息 | 输出 |

**返回：**

返回消息中对应的发送者路由。

**用法说明：**

一般用于客户端在获取应答的发送者路由时使用。

### 设置错误号（SetErrorNo）

**函数原型：**

public void SetErrorNo(int nErrorNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nErrorNo | 具体的错误号 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于客户端处理了请求之后，返回相应的错误号信息，便于请求发起方知道具体的错误原因，错误号：0表示成功，1，-1这两个错误号被中间件占用，上层的业务如果需要设置错误号，不可以使用1，-1。

### 获取错误号（GetErrorNo）

**函数原型：**

public int GetErrorNo()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的错误号信息。

**用法说明：**

一般用于客户端在收到应答之后，根据错误号来判断之前的请求是否正确处理。

### 设置错误信息（SetErrorInfo）

**函数原型：**

public void SetErrorInfo(string strErrorInfo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string strErrorInfo | 具体的错误信息 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于客户端处理了请求之后，根据错误号设置相应的错误信息，便于对方收到错误号的时候，还可以显示相应的错误信息。

### 获取错误信息（GetErrorInfo）

**函数原型：**

public string GetErrorInfo()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的错误信息。

**用法说明：**

一般用于客户端在收到应答之后，获取错误号对应的错误信息。

### 设置返回码（SetReturnCode）

**函数原型：**

public void SetReturnCode(int nReturnCode)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nReturnCode | 具体的返回码 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于客户端处理了请求之后，返回相应的返回码，一般与错误码一致。

### 获取返回码（GetReturnCode）

**函数原型：**

public int GetReturnCode()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的返回码信息。

**用法说明：**

一般用于客户端在收到应答之后，获取对应返回码，做不同的处理。

### 设置业务内容（SetContent）

**函数原型：**

public void SetContent(void\* lpContent, int iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void\* lpContent | 具体的业务二进制内容 | 输入 |
| int iLen | 上面二进制内容的长度 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于设置业务二进制的内容，一般业务请求或者应答都是打包解包的结构。下面是该函数的使用例子：

CT2Packer packer = new CT2Packer(2);

sbyte strType = Convert.ToSByte('S');

sbyte intType = Convert.ToSByte('I');

packer.BeginPack();

packer.AddField("plugin\_id", strType, 255, 4);

packer.AddField("function\_id", intType, 255, 4);

packer.AddStr("com.hundsun.fbase.f2core");

packer.AddInt(100);

packer.EndPack();

unsafe

{

BizMessage.SetContent(packer.GetPackBuf(), packer.GetPackLen());

}

### 获取业务内容（GetContent）

**函数原型：**

public void\* GetContent(int\* iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int\* iLen | 获取业务内容的长度 | 输出 |

**返回：**

返回消息中对应的业务内容的指针，长度由输出参数返回。

**用法说明：**

一般用于客户端在收到应答之后，从应答中获取对应的业务内容，用解包器解包。如下所示：

unsafe

{

int iLen = 0;

void\* lpdata = BizMessage.GetContent(&iLen);

CT2UnPacker unpacker = new CT2UnPacker(lpdata, (uint)iLen);

}

### 设置订阅类型（SetIssueType）

**函数原型：**

public void SetIssueType(int nIssueType)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nIssueType | 订阅类型 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于向消息中心1.0发起订阅，设置订阅类型，原来消息中心1.0的订阅类型是用数字表示的，具体什么类型对应哪个数字，可以参考消息中心1.0的配置，如下所示：

<issuetype note="允许发布类型">

<type **value="0"** superior="0" note="文件更新">

<key name="update\_type"/>

</type>

<type **value="1"** superior="0" credibility="0" note="系统公告">

<key name="branch\_no"/>

<key name="operator\_no"/>

</type>

。。。。。。。。。。。。。。。。。。。。。。。。。

### 获取订阅类型（GetIssueType）

**函数原型：**

public int GetIssueType()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的订阅类型。

**用法说明：**

一般用于客户端在收到订阅应答或者发布消息的时候，获取对应消息的订阅类型。

### 设置订阅序号（SetSequeceNo）

**函数原型：**

public void SetSequeceNo(int nSequeceNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nSequeceNo | 订阅序号 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于向消息中心1.0发起订阅，设置序号，不同的序号表示不同的客户端，一般没有特殊要求，都不需要设置。

### 获取订阅序号（GetSequeceNo）

**函数原型：**

public int GetSequeceNo()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回消息中对应的订阅序号。

**用法说明：**

一般用于客户端在收到订阅应答或者发布消息的时候，获取对应消息的订阅序号。

### 设置订阅关键字段（SetKeyInfo）

**函数原型：**

public void SetKeyInfo(void\* lpKeyData, int iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void\* lpKeyData | 关键字段的二进制内容 | 输入 |
| int iLen | 二进制对应的长度 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于向消息中心1.0发起订阅，设置相应的关键字段，服务端根据这个关键字段，向客户端推消息中和关键字段相符的消息。关键字段的一般用打包器的方式打包，一个订阅类型中包含哪些关键字段也是由消息中心1.0配置文件决定，订阅相应的类型的时候，必须指定所有关键字段的取值。如下所示：

CT2Packer packer = new CT2Packer(2);

sbyte strType = Convert.ToSByte('S');

packer.BeginPack();

packer.AddField("branch\_no", strType, 255, 4);

packer.AddField("fund\_account", strType, 255, 4);

packer.AddStr("8888");

packer.AddStr("20130107");

packer.EndPack();

unsafe

{

BizMessage. SetKeyInfo (packer.GetPackBuf(), packer.GetPackLen());

}

### 获取订阅关键字段（GetKeyInfo）

**函数原型：**

public void\* GetKeyInfo(int\* iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int\* iLen | 返回二进制的长度 | 输出 |

**返回：**

返回消息中对应的关键字段的二进制内容。

**用法说明：**

一般用于客户端在收到订阅应答或者发布消息的时候，获取对应消息的关键字段，一般用解包器进行解包，获取相应的关键字段的取值。

### 设置附加数据（SetAppData）

**函数原型：**

public void SetAppData(void\* lpAppdata, int nAppLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void\* lpAppdata | 附加数据的二进制内容 | 输入 |
| int nAppLen | 二进制对应的长度 | 输入 |

**返回：**

没有返回值

**用法说明：**

这个函数用于向消息中心1.0发起订阅，设置相应的附加数据，附加数据在消息推送的时候会原样返回给客户端，一般附加数据存放的是客户端的窗口句柄，用于区分不同窗口的订阅。附加数据的格式没有规定形式，可以任意形式存放。

### 获取附加数据（GetAppData）

**函数原型：**

public void\* GetAppData(int\* nAppLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int\* nAppLen | 返回二进制的长度 | 输出 |

**返回：**

返回消息中对应的附加数据的二进制内容。

**用法说明：**

一般用于客户端在收到发布消息的时候，获取对应消息的附加数据字段，用来匹配相应的窗口，调用不同的窗口处理发布消息。

### 请求转换成应答（ChangeReq2AnsMessage）

**函数原型：**

public int ChangeReq2AnsMessage()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

转换成功返回0，失败返回其他值。

**用法说明：**

这个函数是从请求转成应答，用于客户端收到请求之后，转换成应答，返回给请求方。

### 消息转换成二进制流（GetBuff）

**函数原型：**

public void\* GetBuff(int\* nBuffLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int\* nBuffLen | 返回二进制的长度 | 输出 |

**返回：**

返回二进制指针。

**用法说明：**

这个函数用于把消息转换成二进制数据流，用于网络上的传输。

### 二进制流转换成消息（SetBuff）

**函数原型：**

public int SetBuff(void\* lpBuff, int nBuffLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void\* lpBuff | 二进制数据指针 | 输入 |
| int nBuffLen | 数据长度 | 输入 |

**返回：**

成功返回0，失败返回其他值。

**用法说明：**

这个函数把二进制数据流转换成消息对象，便于消息对象的接口调用获取需要的内容。

* 1. 打包器类【CT2Packer】

打包器版本一共分为两种，V1.0和V2.0。V1.0是针对原来老版本的打包器，兼容原有的协议。V2.0是后来新使用的。客户端传入何种版本的打包器，是由后台服务决定。

打包器由多个结果集组成，一个结果集里面有自己的返回值，名字，列，行等数据，每个结果集里面会有多条记录。V1.0的打包器不支持多结果，只支持单结果集。

这个类所属的命名空间是：**hundsun.t2sdk**。

### 设置缓存区空间（SetBuffer）

**函数原型：**

public void SetBuffer(void\* pBuf, int iBufSize, int iDataLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void \* pBuf | 缓冲区地址 | 输入 |
| int iBufSize | 缓冲区空间 | 输入 |
| int iDataLen=0 | 已有数据长度，新增数据加在已有数据之后（只对V1.0格式的包有效） | 输入 |

**返回：**

没有返回值

**用法说明：**

本函数主要用设置打包器的空间，由外部传入一个指定的空间。一般情况下，都是内容分配空间。如果传入外部空间，那么这个空间由外部维护，最后不可以调用FreeMem来释放空间。第三个参数只针对V1.0的打包器有效。

### 开始打包（BeginPack）

**函数原型：**

public void BeginPack()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

没有返回值

**用法说明：**

开始打包，在打包之前一定要先调用这个函数，表明下面是开始打字段和值。同时这个函数会清空打包缓存，之前的打包数据会全部清空。

### 构造新结果集（NewDataset）

**函数原型：**

public int NewDataset(string szDatasetName, int iReturnCode)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szDatasetName | 结果的名字 | 输入 |
| int iReturnCode = 0 | 结果集的返回值 | 输入 |

**返回：**

成功返回0，失败返回其他值。

**用法说明：**

在打单结果集的包时，可以不调用本方法,均取默认值，打多结果集的时候，需要先调用这个函数。这个函数只有V2.0打包器支持。

### 添加新字段（AddField）

**函数原型：**

public int AddField(string szFieldName, sbyte cFieldType, int iFieldWidth, int iFieldScale)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szFieldName | 字段名字 | 输入 |
| sbyte cFieldType | 字段类型 | 输入 |
| int iFieldWidth | 字段长度 | 输入 |
| int iFieldScale | 字段精度 | 输入 |

**返回：**

负数表示失败，否则为目前包的长度

**用法说明：**

此函数用于在结果集里面增加一个列，列类型支持五种：I整数，F浮点数，C字符，S字符串，R任意二进制数据；字段长度是指当前字段最大长度，在添加值的时候，超出最大长度，就会截断；字段精度只用于类型为F的时候。

### 添加string型数据（AddStr）

**函数原型：**

public int AddStr(string szValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szValue | 字符串数据 | 输入 |

**返回：**

负数表示失败，否则为目前包的长度

**用法说明：**

此函数用于添加字符串数据，针对的是字符串类型的列。

### 添加int型数据（AddInt）

**函数原型：**

public int AddInt(int iValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int iValue | 整型数据 | 输入 |

**返回：**

负数表示失败，否则为目前包的长度

**用法说明：**

此函数用于添加整型数据，针对的是整数类型的列。

### 添加double型数据（AddDouble）

**函数原型：**

public int AddDouble(double fValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| double fValue | 浮点数数据 | 输入 |

**返回：**

负数表示失败，否则为目前包的长度

**用法说明：**

此函数用于添加浮点数数据，针对的是浮点数类型的列。

### 添加char型数据（AddChar）

**函数原型：**

public int AddChar(sbyte cValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| sbyte cValue | 字符数据 | 输入 |

**返回：**

负数表示失败，否则为目前包的长度

**用法说明：**

此函数用于添加字符数据，针对的是字符类型的列。

### 添加二进制数据（AddRaw）

**函数原型：**

public int AddRaw(void\* lpBuff, int iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void \* lpBuff | 二进制数据 | 输入 |
| int iLen | 数据长度 | 输入 |

**返回：**

负数表示失败，否则为目前包的长度

**用法说明：**

此函数用于添加二进制数据，针对的是二进制类型的列。只有V2.0版本打包器才支持二进制列。

### 结束打包（EndPack）

**函数原型：**

public void EndPack()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

没有返回值

**用法说明：**

此函数用于最后结束打包，打包完之后必须调用此函数。

### 获取打包结果指针（GetPackBuf）

**函数原型：**

public void\* GetPackBuf()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

二进制数据首地址

**用法说明：**

此函数是返回打包器的二进制的数据流首地址。把打包器结构转换为二进制数据。

### 获取打包结果长度（GetPackLen）

**函数原型：**

public int GetPackLen()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

有效数据的长度

**用法说明：**

此函数返回打包器转换为二进制之后的有效数据长度。配合上面的GetPackBuf使用。

### 获取打包缓存长度（GetPackBufSize）

**函数原型：**

public int GetPackBufSize()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

二进制数据的缓存区长度

**用法说明：**

此函数返回打包器转换为二进制之后的缓存区长度。配合上面的GetPackBuf使用。

### 获取打包版本（GetVersion）

**函数原型：**

public int GetVersion()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回版本信息，1表示版本v1.0，其他值表示为v2.0

**用法说明：**

获取打包器版本信息。

### 设置返回码（SetReturnCode）

**函数原型：**

public void SetReturnCode(uint dwRetCode)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| uint dwRetCode | 返回码 | 输入 |

**返回：**

没有返回值

**用法说明：**

此函数用于为当前结果集设置返回码，返回码取缺省值0，则不设置，如果设置，则必须在EndPack()之前调用。

### 获取解包接口（UnPack）

**函数原型：**

public hundsun.t2sdk.CT2UnPacker UnPack()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

当前打包器的解包接口，失败返回NULL指针

**用法说明：**

必须在EndPack()之后才能调用,在打包器释放时相应的解包器实例也释放。

### 释放打包内存（FreeMem）

**函数原型：**

public void FreeMem(void\* lpBuf)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void\* lpBuf | 缓存指针 | 输入 |

**返回：**

没有返回值。

**用法说明：**

该函数用于释放内部的空间，需要显示调用这个函数。如下所示：

pack.FreeMem(pack->GetPackBuf());

### 清空最后一个结果集数据（ClearValue）

**函数原型：**

public void ClearValue()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

没有返回值。

**用法说明：**

该函数用于删除当前打包器的最后一个结果集，清除最后一个结果集的数据，这样再次添加数据而不必添加列信息，以便提高性能。

### 开始打包扩展接口（BeginPackEx）

**函数原型：**

public void BeginPackEx(string szName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szName | 结果集名字 | 输入 |

**返回：**

没有返回值。

**用法说明：**

该函数用于为第一个默认结果增加名字，是BeginPack的扩展函数。

* 1. 解包器类【CT2UnPacker】

此接口是继承自上面结果集接口，实现所有结果集接口之外，支持多结果集解析。这个类所属的命名空间是：**hundsun.t2sdk**。

### 获取解包器版本（GetVersion）

**函数原型：**

public int GetVersion()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回解包器版本，1表示是V1.0版本，33表示V2.0版本。

**用法说明：**

### 二进制转换为解包器（Open）

**函数原型：**

public int Open(void\* lpBuffer, uint iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void \* lpBuffer | 二进制数据指针 | 输入 |
| unsigned int iLen | 长度 | 输入 |

**返回：**

0表示解析成功，其他表示失败。

**用法说明：**

这个函数用客户端收到二进制数据之后，用解包器来解包，转换成解包器结构之后，便于获取行列数据。

### 获取结果集个数（GetDatasetCount）

**函数原型：**

public int GetDatasetCount()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回结果集个数。

**用法说明：**

V2.0的版本支持多结果集，如果是v1.0解包器，用于返回1。通过这个函数可以知道当前解包器有几个结果集，用于遍历所有结果集，获取相关数据。

### 根据下标设置当前结果集（SetCurrentDatasetByIndex）

**函数原型：**

public int SetCurrentDatasetByIndex(int nIndex)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nIndex | 结果集序号 | 输入 |

**返回：**

0表示成功，否则失败。设置的序号不可以超过上面函数GetDatasetCount值。

**用法说明：**

这个函数是把解包器设置到指定序号的结果集，然后就可以获取当前结果集的数据。结合上面的GetDatasetCount就可以遍历所有结果集数据。

### 根据结果集名字设置当前结果集（SetCurrentDataset）、

**函数原型：**

public int SetCurrentDataset(string szDatasetName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szDatasetName | 结果集名字 | 输入 |

**返回：**

0表示成功，否则失败。如果对应的名字的结果集不存在，返回失败。

**用法说明：**

这个函数是把解包器设置到指定名字的结果集，如果结果集有名字，就可以用这个函数。

### 获取解包器缓存指针（GetPackBuf）

**函数原型：**

public void\* GetPackBuf()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回解包器内部的缓存指针。

**用法说明：**

这个缓存一般就是Open函数传入的指针。

### 获取解包器缓存长度（GetPackLen）

**函数原型：**

public uint GetPackLen()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回解包器内部的缓存长度。

**用法说明：**

这个缓存一般就是Open函数传入的长度。

### 获取当前结果集行数（GetRowCount）

**函数原型：**

public uint GetRowCount()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回当前结果集记录行数。

**用法说明：**

根据这个函数返回的行数，可以用下面Go函数直接指定行访问数据。

### 指向结果集第一行记录（First）

**函数原型：**

public void First()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

无返回值。

**用法说明：**

指向当前结果集的首条记录。用于一个结果集记录的遍历开始。

### 指向结果集最后一行记录（Last）

**函数原型：**

public void Last()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

无返回值。

**用法说明：**

指向当前结果集的最后一条记录。

### 跳转到结果集任意行（Go）

**函数原型：**

public void Go(int nRow)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nRow | 要跳转到的记录行 | 输入 |

**返回：**

无返回值。

**用法说明：**

取结果集的第n条记录，取值范围[1, GetRowCount()]，方便任意访问结果集中的具体某行。

### 获取列个数（GetColCount）

**函数原型：**

public int GetColCount()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回结果集中列的个数。

**用法说明：**

用于获取结果集中列的个数，可以根据列来遍历结果集数据。

### 根据下标获取列名字（GetColName）

**函数原型：**

public string GetColName(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回列的名字。下标越界返回NULL。

**用法说明：**

根据传入的列下标获取对应的列名字，可以结合GetColCount函数来遍历所有的列名字。

### 获取列类型（GetColType）

**函数原型：**

public sbyte GetColType(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回列的类型。下标越界默认返回S。

**用法说明：**

根据传入的列下标获取对应的列类型，列类型返回值就五种：I整数，F浮点数，C字符，S字符串，R任意二进制数据。

### 获取列精度（GetColScale）

**函数原型：**

public int GetColScale(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回列的精度。下标越界默认返回0。

**用法说明：**

double类型的列才有精度，其他类型的列精度设置了也没有用。

### 获取列最大长度（GetColWidth）

**函数原型：**

public int GetColWidth(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回列的最大长度。下标越界默认返回1。

**用法说明：**

### 根据列名字找到列下标（FindColIndex）

**函数原型：**

public int FindColIndex(string columnName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string columnName | 列名字 | 输入 |

**返回：**

返回列的下标。不存在返回-1

**用法说明：**

根据列的名字获取对应的下标，然后可以根据下标获取对应的值。也可以用这个函数判断是否存在对应的列。

### 根据列下标获取string类型值（GetStrByIndex）

**函数原型：**

public string GetStrByIndex(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回当前行记录中对应的列下标的值，以string形式返回。下标越界返回NULL。

**用法说明：**

请确保当前列类型为’S’，才用这个函数获取，否则数据可能不准确。

### 根据列名字获取string类型值（GetStr）

**函数原型：**

public string GetStr(string columnName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string columnName | 列名 | 输入 |

**返回：**

返回当前行记录中对应列名的列值，以string形式返回。名字不存在返回NULL。

**用法说明：**

请确保当前列类型为’S’，才用这个函数获取，否则数据可能不准确。

### 根据列下标获取char类型值（GetCharByIndex）

**函数原型：**

public sbyte GetCharByIndex(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回当前行记录中对应列下标的值，以char形式返回。下标越界返回字符’\0’。

**用法说明：**

请确保当前列类型为’C’，才用这个函数获取，否则数据可能不准确。

### 根据列名字获取char类型值（GetChar）

**函数原型：**

public sbyte GetChar(string columnName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string columnName | 列名 | 输入 |

**返回：**

返回当前行记录中对应列名的列值，以char形式返回。名字不存在返回’\0’。

**用法说明：**

请确保当前列类型为’C’，才用这个函数获取，否则数据可能不准确。

### 根据列下标获取double类型值（GetDoubleByIndex）

**函数原型：**

public double GetDoubleByIndex(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回当前行记录中对应列下标的值，以double形式返回。下标越界返回0。

**用法说明：**

请确保当前列类型为’F’，才用这个函数获取，否则数据可能不准确。

### 根据列名字获取double类型值（GetDouble）

**函数原型：**

public double GetDouble(string columnName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string columnName | 列名 | 输入 |

**返回：**

返回当前行记录中对应列名的列值，以double形式返回。名字不存在返回0。

**用法说明：**

请确保当前列类型为’F’，才用这个函数获取，否则数据可能不准确。

### 根据列下标获取int类型值（GetIntByIndex）

**函数原型：**

public int GetIntByIndex(int column)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |

**返回：**

返回当前行记录中对应列下标的值，以int形式返回。下标越界返回0。

**用法说明：**

请确保当前列类型为’I’，才用这个函数获取，否则数据可能不准确。

### 根据列名字获取int类型值（GetInt）

**函数原型：**

public int GetInt(string columnName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string columnName | 列名 | 输入 |

**返回：**

返回当前行记录中对应列名的列值，以int形式返回。名字不存在返回0。

**用法说明：**

请确保当前列类型为’I’，才用这个函数获取，否则数据可能不准确。

### 根据列下标获取二进制类型值（GetRawByIndex）

**函数原型：**

public void\* GetRawByIndex(int column, int\* lpRawLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列下标 | 输入 |
| int \* lpRawLen | 二进制数据长度 | 输出 |

**返回：**

返回当前行记录中对应列下标的值，以二进制形式返回。下标越界返回null。

**用法说明：**

请确保当前列类型为’R’，才用这个函数获取，其他类型的列调用这个函数获取都是null。

### 根据列名字获取二进制类型值（GetRaw）

**函数原型：**

public void\* GetRaw(string columnName, int\* lpRawLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string columnName | 列名 | 输入 |
| int \* lpRawLen | 二进制数据长度 | 输出 |

**返回：**

返回当前行记录中对应列名的列值，以二进制形式返回。名字不存在返回null。

**用法说明：**

请确保当前列类型为’R’，才用这个函数获取，其他类型的列调用这个函数获取都是NULL。

### 最后一次取得值是否为空（WasNull）

**函数原型：**

public int WasNull()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

0表示是空，1表示不是空。

**用法说明：**

用来判断最后一次取得列值是不是空。可以结合前面的函数，一起判断当前列获取默认值是原来的值， 还是空值。

### 结果集下一条记录（Next）

**函数原型：**

public void Next()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

没有返回值

**用法说明：**

用于遍历使用，当前记录所有列的值都获取完了，就可以移到一条记录。

### 结果集是否到结尾（IsEOF）

**函数原型：**

public int IsEOF()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

0表示没有到结果集结尾；1表示到了结果集结尾。

**用法说明：**

在遍历的时候，判断当前结果集的所有记录是不是已经遍历完了，结合Next函数一起使用。遍历代码如下：

while(!unpack.IsEOF())

{

……..//读列值

unpack.Next();

}

### 结果集是否为空（IsEmpty）

**函数原型：**

public int IsEmpty()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

0表示结果集记录数不为0；1表示结果集记录数为0。

**用法说明：**

在遍历之前可以先做记录数个数判断，为空就不需要遍历了。

### 结果集释放（Destroy）

**函数原型：**

public void\* Destroy()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

无返回值

**用法说明：**

结果集释放函数，一般都是调用Release接口释放，不使用这个函数。这个函数功能和Release接口一样。

* 1. 过滤器类【CT2FilterInterface】

过滤条件接口用于向消息中心2.0订阅的时候，传入的过滤信息。用于上层程序根据自己需要做相关订阅。这个类所属的命名空间是：**hundsun. mcapi**。

### 根据下标获取条件名字（GetFilterNameByIndex）

**函数原型：**

public string GetFilterNameByIndex(int index)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int index | 下标 | 输入 |

**返回：**

返回对应下标的过滤条件名字，没有就返回null。

**用法说明：**

根据下标获取过滤条件名字，下标的取值范围在0，GetCount之间。

### 根据下标获取条件值（GetFilterValueByIndex）

**函数原型：**

public string GetFilterValueByIndex(int index)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int index | 下标 | 输入 |

**返回：**

返回对应下标的过滤条件值，没有就返回NULL。

**用法说明：**

根据下标获取过滤条件值，下标的取值范围在0，GetCount之间。

### 根据条件名字好获取条件值（GetFilterValue）

**函数原型：**

public string GetFilterValue(string fileName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string fileName | 过滤条件名字 | 输入 |

**返回：**

根据过滤条件名字返回对应的值，不存在就返回NULL。

**用法说明：**

如果知道过滤条件的名字就可以根据这个函数获取值。

### 获取过滤条件个数（GetCount）

**函数原型：**

public int GetCount()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回过滤条件的个数。

**用法说明：**

这个函数可以获取过滤条件的个数，根据个数，按照上面下标获取方式可以遍历所有的过滤条件名字和值。

### 设置过滤条件名字和值（SetFilter）

**函数原型：**

public void SetFilter(string filterName, string filterValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string filterName | 过滤条件名字 | 输入 |
| string filterValue | 过滤条件值 | 输入 |

**返回：**

无

**用法说明：**

这个函数是往过滤条件结构中添加一个过滤条件名字和值对应关系。用于订阅的时候设置过滤条件的取值。

* 1. 订阅参数类【CT2SubscribeParamInterface】

订阅参数接口，用于向消息中心2.0订阅的时候，传入所有和订阅相关的参数。这个类所属的命名空间是：**hundsun.mcapi**。

### 设置主题名字（SetTopicName）

**函数原型：**

public void SetTopicName(string szName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string szName | 主题名字 | 输入 |

**返回：**

无

**用法说明：**

订阅必须存在主题，如果订阅参数没有主题，订阅就会失败。主体必须是后台消息中心必须存在的主题，否则会报主题不存在的错误。

### 设置附加数据（SetAppData）

**函数原型：**

public void SetAppData(void\* lpData, int iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| void\* lpData | 二进制数据首地址 | 输入 |
| int iLen | 数据长度 | 输入 |

**返回：**

无

**用法说明：**

附加数据是订阅的时候传入，发布消息的时候，服务端会原样返回，一般用于传递窗口句柄，便于发布消息回来的时候找到对应的窗口处理。

### 设置过滤条件（SetFilter）

**函数原型：**

public void SetFilter(string filterName, string filterValue)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string filterName | 过滤条件名字 | 输入 |
| string filterValue | 过滤条件值 | 输入 |

**返回：**

无

**用法说明：**

设置订阅的过滤条件，和前面的过滤条件接口是一样，只不过在订阅参数里面集成了，方便上层调用。

### 设置返回字段（SetReturnFiled）

**函数原型：**

public void SetReturnFiled(string filedName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string filedName | 返回字段的名字 | 输入 |

**返回：**

无

**用法说明：**

订阅的时候，可以先设置好需要返回的字段，譬如，只关心发布消息中的“branchno,fundaccut”两个字段，那么就可以在订阅时候，调用下面的代码：

SubscribeParam.SetReturnFiled(“branchno”);

SubscribeParam.SetReturnFiled(“fundaccut”);

这样订阅之后，收到推送过来的消息就只有这两个字段的值。

### 设置补缺标志（SetFromNow）

**函数原型：**

public void SetFromNow(bool bFromNow)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| bool bFromNow | False表示不补缺，默认值  True表示需要补缺 | 输入 |

**返回：**

无

**用法说明：**

补缺，就是是否需要不在线期间这个主题发布的消息，当然前提是这个主题的可靠级别是文件级才行。主要用于某些重要的通知消息，如果订阅端没有上线，就不能实时接收到，等下次客户端订阅的时候，可以通过补缺来接收之前的消息，防止消息丢失。

### 设置覆盖标志（SetReplace）

**函数原型：**

public void SetReplace(bool bReplace)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| bool bReplace | False表示不覆盖，默认值  True表示需要覆盖 | 输入 |

**返回：**

无

**用法说明：**

覆盖，就是覆盖之前所有的订阅，譬如，一个客户端之前做了很多订阅，从某一个时刻开始，只要当前订阅，之前订阅都不需要了，那么就可以在当前订阅发起的时候设置这个参数，就无需取消之前所有的订阅，方便操作。

### 设置发送间隔（SetSendInterval）

**函数原型：**

public void SetSendInterval(int nSendInterval)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nSendInterval | 发送间隔，单位秒 | 输入 |

**返回：**

无

**用法说明：**

发送间隔，就是服务端推送消息给客户端的频率。譬如：客户端订阅了行情，但是又不想要每一笔行情，只要5秒能收到一笔就可以，这样就可以设置这个参数，那么客户端就会每5秒接收到一笔行情。

### 获取主题名字（GetTopicName）

**函数原型：**

public string GetTopicName()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回订阅参数中的主题名字。

**用法说明：**

### 获取附加数据（GetAppData）

**函数原型：**

public void\* GetAppData(int\* iLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int \*iLen | 附加数据长度 | 输出 |

**返回：**

返回订阅参数中的附加数据指针，没有返回NULL。

**用法说明：**

### 根据下标获取过滤字段名字（GetFilterNameByIndex）

**函数原型：**

public string GetFilterNameByIndex(int index)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int index | 过滤条件下标 | 输入 |

**返回：**

返回订阅参数中对应下标的过滤字段名字。

**用法说明：**

这个函数类似于前面的过滤条件的接口。

### 根据下标获取过滤字段值（GetFilterValueByIndex）

**函数原型：**

public string GetFilterValueByIndex(int index)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int index | 过滤条件下标 | 输入 |

**返回：**

返回订阅参数中对应下标的过滤字段值。

**用法说明：**

这个函数类似于前面的过滤条件的接口。

### 根据过滤名字获取值（GetFilterValue）

**函数原型：**

public string GetFilterValue(string fileName)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string fileName | 过滤条件名字 | 输入 |

**返回：**

返回对应过滤条件名字的值。

**用法说明：**

这个函数类似于前面的过滤条件的接口。

### 获取过滤条件个数（GetFilterCount）

**函数原型：**

public int GetFilterCount()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回对应过滤条件个数。

**用法说明：**

这个函数类似于前面的过滤条件的接口。

### 获取返回字段（GetReturnFiled）

**函数原型：**

public string GetReturnFiled()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回订阅参数的返回字段。没有返回NULL。

**用法说明：**

多个返回字段，用’\2’符号隔开，譬如” branchno\2fundaccut”

### 获取补缺标志（GetFromNow）

**函数原型：**

public void SetFromNow(bool bFromNow)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回补缺标志。

**用法说明：**

### 获取覆盖标志（GetReplace）

**函数原型：**

public void SetReplace(bool bReplace)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回覆盖标志。

**用法说明：**

### 获取发送间隔（GetSendInterval）

**函数原型：**

public void SetSendInterval(int nSendInterval)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回发布间隔。

**用法说明：**

* 1. 订阅回调接口【CT2SubCallbackInterface】

订阅回调的接口，一般是上层实现回调接口的函数，在接收到发布的消息之后，会回调接口给上层应用，这是上层应用程序唯一获取发布消息的接口。这个类所属的命名空间是：**hundsun. mcapi**。

### 收到发布消息回调（OnReceived）

**函数原型：**

virtual void FUNCTION\_CALL\_MODE OnReceived(CSubscribeInterface \*lpSub,int subscribeIndex, const void \*lpData, int nLength,LPSUBSCRIBE\_RECVDATA lpRecvData) = 0;

public abstract void OnReceived(hundsun.mcapi.CT2SubscribeInterface lpSub, int subscribeIndex, void\* lpData, int nLength, hundsun.mcapi.tagSubscribeRecvData lpRecvData)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2SubscribeInterface lpSub | 当前回调订阅接口类对象 | 回调 |
| int subscribeIndex | 当前回调对应订阅标识 | 回调 |
| void \*lpData | 接收到发布内容指针 | 回调 |
| int nLength | 发布内容的长度 | 回调 |
| tagSubscribeRecvData lpRecvData | 订阅相关信息类对象 | 回调 |

**返回：**

无

**用法说明：**

回调的结构体包含内容如下：

public class tagSubscribeRecvData

{

public int iAppDataLen; /\*\*< 附加数据的长度 \*/

public int iFilterDataLen; /\*\*< 过滤字段的数据长度 \*/

public void\* lpAppData; /\*\*< 附加数据的数据头指针 \*/

public void\* lpFilterData; /\*\*< 过滤字段的数据头指针，用解包器解包 \*/

public string szTopicName; /\*\*< 主题名字 \*/

public tagSubscribeRecvData();

}

这个回调会在收到发布消息之后回调进去，上层应用在这个函数里面做业务处理。回调参数里面，还支持当前消息属于哪个订阅，以及过滤条件是什么，附加数据是什么，是什么主题等信息，都返回给了上层函数实现者。

### 收到剔除订阅回调（OnRecvTickMsg）

**函数原型：**

public abstract void OnRecvTickMsg(hundsun.mcapi.CT2SubscribeInterface lpSub, int subscribeIndex, string TickMsgInfo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2SubscribeInterface lpSub | 当前回调的订阅对象 | 回调 |
| int subscribeIndex | 当前回调的对应订阅标识 | 回调 |
| string TickMsgInfo | 剔除订阅的原因 | 回调 |

**返回：**

无

**用法说明：**

此回调会在订阅被踢出的时候调用，什么时候订阅会被服务端踢出，在主题设置了踢出前面订阅者的踢人策略的时候，前面已经订阅的订阅者就会收到这个回调。

* 1. 订阅类【CT2SubscribeInterface】

这个接口主要负责订阅，取消订阅，接收发布消息回调给上层应用，接口实例化有连接接口里面的函数实现。这个类所属的命名空间是：**hundsun. mcapi**。

### 订阅主题

**函数原型：**

public int SubscribeTopicEx(hundsun.mcapi.CT2SubscribeParamInterface lpSubscribeParamInter, uint uiTimeout, out hundsun.t2sdk.CT2UnPacker lpBizUnPack, hundsun.t2sdk.CT2Packer lpBizPack)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2SubscribeParamInterface lpSubscribeParamInter | 订阅传入的参数对象 | 输入 |
| unsigned int uiTimeout | 订阅超时时间，单位毫秒 | 输入 |
| CT2UnPacker lpBizUnPack | 业务校验失败返回解包器类对象 | 输出 |
| CT2Packer lpBizPack | 业务校验需要传入信息 | 输入 |

**返回：**

返回订阅标识，大于0表示有效的订阅标识，否则其他表示错误。

**用法说明：**

订阅有时候需要业务校验，这层校验是服务端上进行了，除了校验订阅的主题和过滤条件之外，还可以进行业务层校验，需要业务人员在服务端上进行开发。业务校验需要的额外的参数就需要用lpBizPack这个参数传递进去；如果业务校验出错，具体的错误信息就会在lppBizUnPack返回，这里lppBizUnPack用法如下所示：

CT2UnPacker unpack = null;

iRet = lpSub.SubscribeTopicEx(lpSubscribeParam, 5000, out unpack, null);

//最后根据返回值，如果是失败的就判断 lpBizUnPack 是不是null.

if (unpack != null)

{

//最后错误信息获取完之后,释放

unpack.Dispose();

}

### 取消订阅

**函数原型：**

public int CancelSubscribeTopic(int subscribeIndex)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int subscribeIndex | 订阅标识 | 输入 |

**返回：**

返回0表示取消订阅成功，返回其他值，根据错误号获取错误信息.

**用法说明：**

这里输入的订阅标识就是SubscribeTopic返回的值。

### 取消订阅扩展接口

**函数原型：**

public int CancelSubscribeTopicEx(string topicName, hundsun.mcapi.CT2FilterInterface lpFilterInterface)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string topicName | 主题名字 | 输入 |
| CT2FilterInterface lpFilterInterface | 过滤条件 | 输入 |

**返回：**

返回0表示取消订阅成功，返回其他值，根据错误号获取错误信息.

**用法说明：**

这里输入的是主题名字和过滤条件，根据这个属性可以唯一确定一个订阅标识，如果没有记录之前订阅成功的标识的话，可以用这个接口取消订阅。最好是采用前面的CancelSubscribeTopic函数，效率会高。

### 获取已经订阅的信息

**函数原型：**

public void GetSubcribeTopic(hundsun.t2sdk.CT2Packer lpPack)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2Packer lpPack | 所有订阅信息打包器，外面申请，函数内部添加字段和值 | 输入 |

**返回：**

无返回值.

**用法说明：**

函数内部返回的打包器字段如下所示：

- SubcribeIndex 订阅标识，也就是SubscribeTopic函数返回值

- IsBornTopic 先天订阅标识，0表示不是先天订阅

- TopicName 主题名字

- TopicNo 主题编号

- FilterRaw 过滤字段信息

- Appdata 附加数据

- SendInterval 发送间隔

- ReturnFileds 返回字段

- isReplace 覆盖标识

- isFromNow 补缺标识

根据上面的字段获取相关的订阅信息。

* 1. 发布类【CT2PublishInterface】

这个接口主要用于发布消息到消息中心2.0，然后推送给关心的客户端。发布的消息分两种，一种是采用打包器的行列方式的数据，另一种就是随意的二进制。这个类所属的命名空间是：**hundsun. mcapi**。

### 业务发包格式发送

**函数原型：**

public int PubMsgByPackerEx(string topicName, hundsun.t2sdk.CT2UnPacker lpUnPacker, int iTimeOut, out hundsun.t2sdk.CT2UnPacker lpBizPack, bool bAddTimeStamp)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string topicName | 主题名字 | 输入 |
| CT2UnPacker lpUnPacker | 发布的业务包 | 输入 |
| int iTimeOut | 超时时间 | 输入 |
| CT2UnPacker lpBizPack | 业务校验失败返回 | 输出 |
| bool bAddTimeStamp | 是否添加时间戳 | 输入 |

**返回：**

返回0表示成功，返回其他值，根据错误号获取错误信息.

**用法说明：**

参数中解包器，是发布内容先用pack打包，然后调用unpack接口传入，代码如下：

CT2Packer lpOnePack = new CT2Packer(2);

lpOnePack.BeginPack();

lpOnePack.AddField("branchno");

lpOnePack.AddStr("1000");

lpOnePack.EndPack();

CT2UnPacker lpUnPack = lpOnePack.UnPack();

CT2UnPacker unpack = null;

int iRet = lpPublish.PubMsgByPackerEx(topicName,lpUnPack,5000, out unpack,false);

注意的是，打的pack包里面必须要有这个主题的所有过滤字段信息，否则就会发布出错。

业务校验和订阅类似，不过不需要单独成传业务参数，只要接收错误信息即可，错误信息的处理和上面订阅接口一样。如下所示：

CT2UnPacker unpack = null;

int iRet = lpPublish.PubMsgByPackerEx(topicName,lpUnPack,5000, out unpack,false);

//最后根据返回值，如果是失败的就判断 lpBizUnPack 是不是NULL.

if(unpack!=null)

{

//最后错误信息获取完之后,释放

unpack.Dispose();

}

### 二进制数据发送

**函数原型：**

public int PubMsgEx(string topicName, hundsun.mcapi.CT2FilterInterface lpFilterInterface, byte[] lpData, int iTimeOut, out hundsun.t2sdk.CT2UnPacker lpBizPack, bool bAddTimeStamp)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string topicName | 主题名字 | 输入 |
| CT2FilterInterface lpFilterInterface | 过滤信息 | 输入 |
| byte[] lpData | 二进制数据首地址 | 输入 |
| int iTimeOut | 二进制数据长度 | 输入 |
| CT2UnPacker lpBizPack | 业务校验失败返回 | 输出 |
| bool bAddTimeStamp | 是否添加时间戳 | 输入 |

**返回：**

返回0表示成功，返回其他值，根据错误号获取错误信息.

**用法说明：**

这个函数唯一不同就是，传入的数据不是pack结构的数据，是随意的二进制数据，指针和长度传入即可。因为不是pack的形式数据，所以需要单独指定过滤字段，当然过滤字段包含当前发布主题的所有过滤字段，表明当前消息属于什么过滤信息，这样才可以推送给关心这个消息的订阅端。

* 1. 连接类【CT2Connection】

这个接口是主要的操作接口，上层通过这个接口收发数据，建立连接，进行发布订阅等等操作。这个类所属的命名空间是：**hundsun.t2sdk**。

### 初始化连接对象（Create）

**函数原型：**

public int Create(hundsun.t2sdk.CT2CallbackInterface lpCallback)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2CallbackInterface lpCallback | 回调接口实现类对象 | 输入 |

**返回：**

返回0表示成功，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

如果应用不需要任何回调方法，则可向该方法传递NULL，而不必自定义回调类和对象。如果有回调实例，这个初始化的数据接收函数只会调用OnReceivedBiz函数。

### 开始连接/注册（Connect）

**函数原型：**

public int Connect(uint uiTimeout)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| uint uiTimeout | 连接同步等待的超时时间 | 输入 |

**返回：**

返回0表示成功，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

此函数用于向后台进行建TCP连接和注册的过程，如果是密码级别的连接，还包括了密钥的交互，SSL级别的连接就包括了SSL的连接。不同级别连接，建立的时间都不一样，上层根据实际传入不同的连接超时时间。如果超时时间传入0，那么连接就会是异步的模式，上层就需要调用GetStatus函数判断连接是否已经注册成功，否则发包都会包“连接未注册”的错误。

### 断开连接（Close）

**函数原型：**

public int Close()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回0表示成功，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

此函数用于主动断开和服务端的连接。

### 取连接服务器地址（GetServerAddress）

**函数原型：**

public string GetServerAddress(int\* lpPort)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int \*lpPort | 服务端的端口 | 输出 |

**返回：**

返回服务器IP地址，参数返回是端口。

**用法说明：**

这个函数返回当前连接对应的服务端的IP地址和端口，在服务端地址有多个情况下，可以确定当前连接连的是哪个服务器。

### 取连接状态（GetStatus）

**函数原型：**

public int GetStatus()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回服务器的连接状态。

**用法说明：**

连接的状态有以下几种：

///连接状态，组合而成

enum ConnectionStatus

{

Disconnected = 0x0000, /\*\*< 未连接 \*/

Connecting = 0x0001, /\*\*< socket正在连接 \*/

Connected = 0x0002, /\*\*< socket已连接 \*/

SafeConnecting = 0x0004, /\*\*< 正在建立安全连接 \*/

SafeConnected = 0x0008, /\*\*< 已建立安全连接 \*/

Registering = 0x0010, /\*\*< 正注册 \*/

Registered = 0x0020, /\*\*< 已注册 \*/

Rejected = 0x0040 /\*\*< 被拒绝，将被关闭 \*/

};

连接状态是组合，如果需要判断一个连接的状态是否已经注册，代码如下所示：

if (lpConnection.GetStatus() & Registered)

{

//连接已经注册

}

### 取服务器负载（GetServerLoad）

**函数原型：**

public int GetServerLoad()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回服务器负载（非负数），越大表示越繁忙，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

使用者可以同时创建多个连接实例同时连接不同的服务器，根据完成连接的时间以及负载决定使用哪个服务器，负载指的是后台业务处理队列的积压情况。

### 取错误码对应的错误信息（GetErrorMsg）

**函数原型：**

public string GetErrorMsg(int nErrorCode)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int nErrorCode | 错误码 | 输入 |

**返回：**

返回错误信息

**用法说明：**

根据错误码，返回错误，错误信息返回支持中文和英文两种格式。可以连接配置参数中设置语言。

### 取连接错误号（GetConnectError）

**函数原型：**

public int GetConnectError()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回连接建立和注册过程出现的错误

**用法说明：**

当连接无法与服务端完成注册时，既可通过Connect的返回值获取错误号，也可通过调用本方法来获取错误号，然后用该错误号调用GetErrorMsg可获取可读的错误信息。

### 发送业务数据（SendBiz）

**函数原型：**

public int SendBiz(int iFunID, hundsun.t2sdk.CT2Packer lpPacker, int nAsy, int iSystemNo, int nCompressID)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int iFunID | 功能号 | 输入 |
| CT2Packer lpPacker | 业务请求数据 | 输入 |
| int nAsy | 是否异步发送 | 输入 |
| int iSystemNo | 系统号 | 输入 |
| int nCompressID | 是否压缩 | 输入 |

**返回：**

返回发送句柄（正数），否则表示失败，通过调用GetErrorMsg可以获取详细错误信息。

**用法说明：**

当异步发送标志为0，就表示同步发送，同步发送的包，通过调用RecvBiz来接收；如果异步发送标志位1，就表示异步发包，当收到应答包后，自动触发回调函数OnReceivedBiz。这个函数需要配合Create这个初始化函数一起使用。

### 接收业务数据（RecvBiz）

**函数原型：**

public int RecvBiz(int hSend, out string lppStr, out hundsun.t2sdk.CT2UnPacker lppUnPacker, uint uiTimeout, uint uiFlag)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int hSend | 发送句柄，就是SendBiz返回值 | 输入 |
| string lppStr | 返回的错误信息 | 输出 |
| CT2UnPacker lppUnPacker | 返回的数据 | 输出 |
| unsigned uiTimeout | 接收数据的超时时间 | 输入 |
| unsigned uiFlag | 接收选项，接收超时之后是否删除对应发送句柄的数据。 | 输入 |

**返回：**

小于0表示RecvBiz操作本身失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

* + - 如果nResult等于0，表示业务数据接收成功，并且业务操作成功，lppUnPacker有对应的应答数据，lppStr为null。
    - 如果nResult等于1，表示业务数据接收成功，但业务操作失败了，lppUnPacker有对应的应答数据，lppStr为null。
    - 如果nResult等于2，表示收到非业务错误信息，lppStr为具体错误信息，但是lppUnPacker为null。
    - 如果nResult等于3，表示业务包解包失败。lppStr为null，lppUnPacker为null。

接收选项参数说明如下：

* + - * 0表示接收超时后仍可继续调用RecvBiz来接收，
      * 1表示当接收超时后，把hSend相关数据删除

最后注意：外部所指向的解包器的内存由SDK内部管理，外部切勿释放！这个函数也是配合SendBiz，Create使用。

### 发送业务数据扩展接口（SendBizEx）

**函数原型：**

public int SendBizEx(int iFunID, hundsun.t2sdk.CT2Packer lpPacker, string svrName, int nAsy, int iSystemNo, int nCompressID, int branchNo, hundsun.t2sdk.CT2RequestData lpRequest)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int iFunID | 功能号 | 输入 |
| CT2Packer lpPacker | 业务请求数据 | 输入 |
| string svrName | 指定中间件的节点 | 输入 |
| int nAsy | 是否异步发送 | 输入 |
| int iSystemNo | 系统号 | 输入 |
| int nCompressID | 是否压缩 | 输入 |
| int branchNo | 营业部号 | 输入 |
| CT2RequestData lpRequest | 其他相关的请求参数类对象 | 输入 |

**返回：**

返回发送句柄（正数），否则表示失败，通过调用GetErrorMsg可以获取详细错误信息。

**用法说明：**

当异步发送标志为0，就表示同步发送，同步发送的包，通过调用RecvBizEx来接收；当异步发送标志为1，就表示异步发送，异步发送的包，当收到应答包后，自动触发回调函数OnReceivedBizEx。这个函数需要配合CreateEx这个初始化函数一起使用。

参数类CT2RequestData定义如下所示：

public class CT2RequestData

{

public int fileHeadLen; // 消息中心1.0附加数据长度

public int issueType; //消息中心1.0发布类型

public int iSubSystemNo; //参数中增加子系统号传入

public int keyInfoLen; //消息中心1.0关键字段长度

public void\* lpFileHead; // 消息中心1.0附加数据

public void\* lpKeyInfo; //消息中心1.0关键字段，pack数据

public int packetType; //包类型

public readonly CT2RouteInfo routeInfo; //请求里面增加路由信息

public int sequeceNo; //消息中心1.0的序号

public CT2RequestData();

}

### 接收业务数据扩展接口（RecvBizEx）

**函数原型：**

virtual int FUNCTION\_CALL\_MODE RecvBizEx(int hSend, void \*\*lppUnpackerOrStr, LPRET\_DATA\* lpRetData, unsigned uiTimeout = 1000, unsigned uiFlag = 0) = 0;

public int RecvBizEx(int hSend, out string lppStr, out hundsun.t2sdk.CT2UnPacker lppUnPacker, out hundsun.t2sdk.CT2RespondData lpRetData, uint uiTimeout, uint uiFlag)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int hSend | 发送句柄，就是SendBiz返回值 | 输入 |
| string lppStr | 返回错误信息 |  |
| CT2UnPacker lppUnPacker | 返回的数据 | 输出 |
| CT2RespondData lpRetData | 返回的其他数据 | 输出 |
| unsigned uiTimeout | 接收数据的超时时间 | 输入 |
| unsigned uiFlag | 接收选项，接收超时之后是否删除对应发送句柄的数据。 | 输入 |

**返回：**

小于0表示RecvBizEx操作本身失败，通过调用GetErrorMsg可以获取详细错误信息。

**用法说明：**

* + - * 如果nResult等于0，表示业务数据接收成功，并且业务操作成功，lppUnPacker有对应的应答数据，lppStr为null。
      * 如果nResult等于1，表示业务数据接收成功，但业务操作失败了，lppUnPacker有对应的应答数据，lppStr为null。
      * 如果nResult等于2，表示收到非业务错误信息，lppStr为具体错误信息，但是lppUnPacker为null。
      * 如果nResult等于3，表示业务包解包失败。lppStr为null，lppUnPacker为null。

接收选项参数说明如下：

* + - * 0表示接收超时后仍可继续调用RecvBizEx来接收，
      * 1表示当接收超时后，把hSend相关数据删除

最后注意：

* 外部所指向的解包器的内存由SDK内部管理，外部切勿释放！
* 外部所指向的CT2RespondData 的内存由SDK内部管理，外部切勿释放！
* 这个函数也是配合SendBizEx，CreateEx使用。

其他需要返回的字段的结构体定义如下：

public class CT2RespondData

{

public string errorInfo; //错误信息

public int errorNo; //错误号

public int functionID; //功能号

public int issueType; //消息中心1.0发布类型

public int keyInfoLen; //消息中心1.0关键字段长度

public void\* lpKeyInfo; //消息中心1.0关键字段，Pack数据

public int returnCode; //返回码

public readonly CT2RouteInfo sendInfo; //应答里面增加发送者信息

public CT2RespondData();

}

### 连接初始化的扩展接口（CreateEx）

**函数原型：**

public int CreateEx(hundsun.t2sdk.CT2CallbackInterface lpCallback)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2CallbackInterface lpCallback | 回调接口实现类对象 | 输入 |

**返回：**

返回0表示成功，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

如果应用不需要任何回调方法，则可向该方法传递NULL，而不必自定义回调类和对象。如果有回调实例，这个初始化的数据接收函数只会调用OnReceivedBizEx函数。

### 创建订阅者（NewSubscriber）

**函数原型：**

public hundsun.mcapi.CT2SubscribeInterface NewSubscriber(hundsun.mcapi.CT2SubCallbackInterface lpCallback, string SubScribeName, int iTimeOut, int iInitRecvQLen, int iStepRecvQLen)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2SubCallbackInterface lpCallback | 订阅者接收消息的回调类对象 | 输入 |
| string SubScribeName | 订阅者名字 | 输入 |
| int iTimeOut | 超时时间 | 输入 |
| int iInitRecvQLen | 初始化接收队列长度 | 输入 |
| int iStepRecvQLen | 接收队列扩大步长 | 输入 |

**返回：**

返回订阅接口实例，失败返回NULL，错误信息通过GetMCLastError获取。

**用法说明：**

连接属性里面没有这个配置，初始化就会失败。一个连接对应多个订阅，每个订阅者的都有自己的名字，名字不可以一样，是通过参数传进来的。

订阅端会有数据需要落地的，落地的目录可以配置，对应的ini配置形式如下：

[t2sdk]

logdir=D:\\mc

如果没有配置，就会在程序运行的当前目录下。

### 获取发布者（NewPublisher）

**函数原型：**

public hundsun.mcapi.CT2PublishInterface NewPublisher(string PublishName,int msgCount, int iTimeOut, bool bResetNo)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| string PublishName | 业务名 | 输入 |
| int msgCount | 本地缓存消息个数 | 输入 |
| int iTimeOut | 初始化发布者超时时间 | 输入 |
| bool bResetNo | 是否重置序号 | 输入 |

**返回：**

返回发送接口实例，一个T2SKD内不能有相同业务名，失败返回NULL，错误信息通过GetMCLastError获取。

**用法说明：**

本地缓存的消息个数是针对文件级，或者系统级的主题而言，这两种主题发布，会在本地落地，落地的目录可以配置，对应的ini配置形式如下：

[t2sdk]

logdir=D:\\mc

如果没有配置，就会在程序运行的当前目录下，建立一个目录，目录名字为mc开头，后面跟着PublishName配置的名字。

### 获取主题信息（GetTopic）

**函数原型：**

public hundsun.t2sdk.CT2UnPacker GetTopic(bool byForce, int iTimeOut)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| bool byForce | 是否强制从服务端获取主题 | 输入 |
| int iTimeOut | 获取主题的超时时间 | 输入 |

**返回：**

返回消息中心的主题信息，用解包器的形式返回。

**用法说明：**

解包器外面需要调用release接口进行释放。解包器的返回字段有：

- TopicName

- TopicNo

- ReliableLevel

- IssuePriority

- MsgLifetime

- Stutas

- TickStrategy

- BusinessVerify

- Local

- FilterField1

- FilterField2

- FilterField3

- FilterField4

- FilterField5

- FilterField6

- FilterField7

- FilterField8

- FilterField9

- FilterField10

- FilterField11

- FilterField12

- FilterField13

- FilterField14

- FilterField15

- FilterField16

- SubscribeStr

- PublishStr

### 获取消息中心的最后错误（GetMCLastError）

**函数原型：**

public string GetMCLastError()

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| 无 |  |  |

**返回：**

返回创建订阅者和获取发布者过程中的最后的错误信息

**用法说明：**

### 连接用IBizMessage接口初始化（Create2BizMsg）

**函数原型：**

public int Create2BizMsg(hundsun.t2sdk.CT2CallbackInterface lpCallback)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2CallbackInterface lpCallback | 回调接口实现类对象 | 输入 |

**返回：**

返回0表示成功，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

如果应用不需要任何回调方法，则可向该方法传递NULL，而不必自定义回调类和对象。如果有回调实例，这个初始化的数据接收函数只会调用OnReceivedBizMsg函数。

### 连接发送IBizMessage业务消息（SendBizMsg）

**函数原型：**

public int SendBizMsg(hundsun.t2sdk.CT2BizMessage lpMsg, int nAsy)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| CT2BizMessage lpMsg | 业务请求消息对象 | 输入 |
| int nAsy | 异步发送标志 | 输入 |

**返回：**

返回发送句柄（正数），否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

当异步发送标志为0，就表示同步发送，同步发送的包，通过调用IBizMessage来接收；如果异步发送标志位1，就表示异步发包，当收到应答包后，自动触发回调函数OnReceivedBizMsg。这个函数需要配合Create2BizMsg这个初始化函数一起使用。

### 连接接收IBizMessage业务消息（RecvBizMsg）

**函数原型：**

public int RecvBizMsg(int hSend, out hundsun.t2sdk.CT2BizMessage lpMsg, int uiTimeout, int uiFlag)

**输入参数：**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int hSend | 发送句柄，就是SendBiz返回值 | 输入 |
| CT2BizMessage lpMsg | 返回的消息对象 | 输出 |
| unsigned uiTimeout | 接收数据的超时时间 | 输入 |
| unsigned uiFlag | 接收选项，接收超时之后是否删除对应发送句柄的数据。 | 输入 |

**返回：**

返回0表示成功，否则表示失败，通过调用GetErrorMsg可以获取详细错误信息

**用法说明：**

接收选项参数说明如下：

* 0表示接收超时后仍可继续调用RecvBizMsg来接收，
* 1表示当接收超时后，把hSend相关数据删除

最后注意：外部所指向的消息对象的内存由SDK内部管理，外部切勿释放！这个函数也是配合SendBizMsg，Create2BizMsg使用。

# 注意事项

* 1. 相关限制
* 我们发布的ext\_t2sdkEx.dll是在2.0的框架下编译集成的，目的是为了让3.0，3.5,4.0框架都可以使用。如果上层程序是在4.0框架上开发，需要在app.config里面增加“useLegacyV2RuntimeActivationPolicy="true"”，如下所示：

<?xml version="1.0"?>

<configuration>

<startup **useLegacyV2RuntimeActivationPolicy="true"**>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>

</startup>

</configuration>

* 开发者在进行同步SendBiz，RecvBiz时，一般的调用流程为一次SendBiz后，马上RecvBiz。然而，如果开发者先进行N次SendBiz操作，再进行N次RecvBiz，此时，T2\_SDK可能会出现RecvBiz操作很慢，而实际上要接收的数据早已到达的情况。为了避免这种情况的发送，开发者可在配置文件中配置event\_count，使其数量等于N。即使event\_count配置的数量达不到N，通过扩大event\_count的数量也可以降低出现上述糟糕情况的概率。但请注意，event\_count的数量如果配置太大，将占用大量的系统资源。
* 使用ext\_t2sdkEx.dll发送的包的最大长度为16M-1字节。
* Windows下，单进程使用ext\_t2sdkEx.dll最多可以和服务端建立1024个连接，可通过多进程线性扩展。
* 在使用ext\_t2sdkEx.dll时，对同一个连接，请勿多次Create。
  1. 调用限制

1. OnClose回调中不能调用Connect、release、new CT2Connection
2. OnSent回调中不能调用Connect、Close、release、new CT2Connection
3. OnReceivedBiz回调中不能调用Connect、release、new CT2Connection
   1. 回调线程

ext\_t2sdkEx.dll本身有两个线程：接收线程，发送线程，加上调用线程，总共3种线程，CT2CallbackInterface的回调函数，需要注意线程安全问题。

|  |  |
| --- | --- |
| 函数 | 线程 |
| OnConnect | 调用线程，发送线程 |
| OnSafeConnect | 调用线程，发送线程，接收线程 |
| OnRegister | 接收线程 |
| OnClose | 调用线程，发送线程，接收线程 |
| OnSent | 发送线程 |
| OnReceivedBiz | 接收线程 |

* 1. 编程建议
* 回调函数中，外部调用者切勿做大量耗时的操作，否则会引起数据发送/接收阻塞。
* 回调函数中，建议不要做类似同步的操作，否则可能会产生死锁。比如OnReceivedBiz回调中有段代码是m\_lstInfo.AddString(info)，其中m\_lstInfo是一个MFC文本编辑框，每次做一次AddString操作，线程会同步通知（SendMessageA）主线程刷新页面。注意，这是一个同步操作，假如当时主线程正在处理业务，而且下一步刚好是NewConnection之类的限制操作，那么就会产生死锁。

# 示例代码

本章主要是代码的示例，分成下面四个开发模式。展示代码不是很完整，完整代码请参考我们发布包的例子。

* 1. 建立连接

config = new CT2Configinterface();

config.Load("t2sdk.ini");

conn = new CT2Connection(config);

callback = new callbacktest(this);

conn.Create2BizMsg(callback);

int iret = conn.Connect(5000);

textBox1.Text = "连接" + conn.GetErrorMsg(iret) + "\r\n" + textBox1.Text;

if (iret == 0)

{

button3.Enabled = false;

button1.Enabled = true;

button2.Enabled = true;

button4.Enabled = true;

button5.Enabled = true;

button6.Enabled = true;

button7.Enabled = true;

}

* 1. 断开连接

config.Dispose();

conn.Dispose();

config = null;

conn = null;

textBox1.Text = "";

button3.Enabled = true;

button1.Enabled = false;

button2.Enabled = false;

button4.Enabled = false;

button5.Enabled = false;

button6.Enabled = false;

button7.Enabled = false;

* 1. 同步发送接收

CT2BizMessage BizMessage = new CT2BizMessage();//构造消息

BizMessage.SetFunction(8);//设置功能号

BizMessage.SetPacketType(0);//设置消息类型为请求

//打包请求报文

CT2Packer packer = new CT2Packer(2);

sbyte strType = Convert.ToSByte('S');

sbyte intType = Convert.ToSByte('I');

packer.BeginPack();

//插件编号

packer.AddField("plugin\_id", strType, 255, 4);

//管理功能号

packer.AddField("function\_id", intType, 255, 4);

packer.AddStr("com.hundsun.fbase.f2core");

packer.AddInt(100);

packer.EndPack();

unsafe

{

BizMessage.SetContent(packer.GetPackBuf(), packer.GetPackLen());

}

int iRet = conn.SendBizMsg(BizMessage, 0);

if (iRet < 0)

{

DisplayText(conn.GetErrorMsg(iRet));

}

else

{

CT2BizMessage AnsBizMessage = null;

iRet = conn.RecvBizMsg(iRet,out AnsBizMessage, 5000, 0);

if (iRet < 0)

{

DisplayText(conn.GetErrorMsg(iRet));

}

else

{

int iRetCode = BizMessage.GetErrorNo();//获取返回码

int iErrorCode = BizMessage.GetErrorNo();//获取错误码

if (iRetCode != 0)

{

DisplayText("同步接收出错：" + BizMessage.GetErrorNo().ToString() + BizMessage.GetErrorInfo());

}

else

{

CT2UnPacker unpacker = null;

unsafe

{

int iLen = 0;

void\* lpdata = BizMessage.GetContent(&iLen);

unpacker = new CT2UnPacker(lpdata, (uint)iLen);

}

//返回业务错误

if(iErrorCode!=0)

{

DisplayText("同步接收业务出错：\n");

PrintUnPack(unpacker);

}

//正常业务返回

else

{

DisplayText("同步接收业务成功：\n");

PrintUnPack(unpacker);

}

if (unpacker != null)

{

unpacker.Dispose();

}

}

}

}

packer.Dispose();

BizMessage.Dispose();

* 1. 异步发送接收

//异步回调类实现

public unsafe class callbacktest : CT2CallbackInterface

{

public callbacktest(Form1 form1) { m\_lpOwner = form1; }

private Form1 m\_lpOwner;

public override void OnConnect(CT2Connection lpConnection)

{

System.Console.WriteLine("OnConnect");

}

public override void OnSafeConnect(CT2Connection lpConnection)

{

System.Console.WriteLine("OnSafeConnect");

}

public override void OnRegister(CT2Connection lpConnection)

{

System.Console.WriteLine("OnRegister");

}

public override void OnClose(CT2Connection lpConnection)

{

System.Console.WriteLine("OnClose");

}

public override void OnReceivedBiz(CT2Connection lpConnection, int hSend, String lppStr, CT2UnPacker lppUnPacker, int nResult)

{

}

public override void OnReceivedBizEx(CT2Connection lpConnection, int hSend, CT2RespondData lpRetData, String lppStr, CT2UnPacker lppUnPacker, int nResult)

{

}

public override void OnSent(CT2Connection lpConnection, int hSend, void\* lpData, int nLength, int nQueuingData)

{

}

public override void OnReceivedBizMsg(CT2Connection lpConnection, int hSend, CT2BizMessage lpMsg)

{

int iRetCode = lpMsg.GetErrorNo();//获取返回码

int iErrorCode = lpMsg.GetErrorNo();//获取错误码

if (iRetCode != 0)

{

m\_lpOwner.DisplayText("异步接收出错：" + lpMsg.GetErrorNo().ToString() + lpMsg.GetErrorInfo());

}

else

{

CT2UnPacker unpacker = null;

unsafe

{

int iLen = 0;

void\* lpdata = lpMsg.GetContent(&iLen);

unpacker = new CT2UnPacker(lpdata, (uint)iLen);

}

//返回业务错误

if (iErrorCode != 0)

{

m\_lpOwner.DisplayText("异步接收业务出错：\n");

m\_lpOwner.PrintUnPack(unpacker);

}

//正常业务返回

else

{

m\_lpOwner.DisplayText("异步接收业务成功：\n");

m\_lpOwner.PrintUnPack(unpacker);

}

if (unpacker != null)

{

unpacker.Dispose();

}

}

}

};

//异步请求发送代码

CT2BizMessage BizMessage = new CT2BizMessage();//构造消息

BizMessage.SetFunction(8);//设置功能号

BizMessage.SetPacketType(0);//设置消息类型为请求

//打包请求报文

CT2Packer packer = new CT2Packer(2);

sbyte strType = Convert.ToSByte('S');

sbyte intType = Convert.ToSByte('I');

packer.BeginPack();

//插件编号

packer.AddField("plugin\_id", strType, 255, 4);

//管理功能号

packer.AddField("function\_id", intType, 255, 4);

packer.AddStr("com.hundsun.fbase.f2core");

packer.AddInt(100);

packer.EndPack();

unsafe

{

BizMessage.SetContent(packer.GetPackBuf(), packer.GetPackLen());

}

int iRet = conn.SendBizMsg(BizMessage, 1);

if (iRet < 0)

{

DisplayText(conn.GetErrorMsg(iRet));

}

else

{

DisplayText("异步请求发送成功。\n");

}

packer.Dispose();

* 1. 订阅

这里订阅是针对服务端为消息中心2.0的代码。

public unsafe class CSubCallback : CT2SubCallbackInterface

{

private Form1 m\_lpOwner;

public CSubCallback(Form1 form1)

{

m\_lpOwner = form1;

}

public override void OnReceived(CT2SubscribeInterface lpSub, int subscribeIndex, void\* lpData, int nLength, tagSubscribeRecvData lpRecvData)

{

m\_lpOwner.DisplayText("订阅收到数据\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

String strInfo = String.Format("附加数据长度： {0}\n", lpRecvData.iAppDataLen);

m\_lpOwner.DisplayText(strInfo);

if (lpRecvData.iAppDataLen > 0)

{

unsafe

{

strInfo = String.Format("附加数据： {0}\n", Marshal.PtrToStringAuto(new IntPtr(lpRecvData.lpAppData)));

m\_lpOwner.DisplayText(strInfo);

}

}

m\_lpOwner.DisplayText("过滤字段部分：\n");

if (lpRecvData.iFilterDataLen > 0)

{

CT2UnPacker lpUnpack = new CT2UnPacker(lpRecvData.lpFilterData, (uint)lpRecvData.iFilterDataLen);

m\_lpOwner.PrintUnPack(lpUnpack);

lpUnpack.Dispose();

}

CT2UnPacker lpUnPack1 = new CT2UnPacker((void\*)lpData, (uint)nLength);

if (lpUnPack1 != null)

{

m\_lpOwner.PrintUnPack(lpUnPack1);

lpUnPack1.Dispose();

}

m\_lpOwner.DisplayText("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

public override void OnRecvTickMsg(CT2SubscribeInterface lpSub, int subscribeIndex, string TickMsgInfo)

{

}

//定也代码

//建立订阅相关项

if (subscriberCallback == null)

{

subscriberCallback = new CSubCallback(this);

}

if (lpSub == null)

{

lpSub = conn.NewSubscriber(subscriberCallback, "test", 5000, 2000, 100);

}

if (lpSub == null)

{

DisplayText(conn.GetMCLastError());

return;

}

//订阅参数获取

CT2SubscribeParamInterface lpSubscribeParam = new CT2SubscribeParamInterface();

string topicName = config.GetString("subcribe", "topic\_name", "");//主题名字

lpSubscribeParam.SetTopicName(topicName);

string isFromNow = config.GetString("subcribe", "is\_rebulid", "");//是否补缺

if (isFromNow.Equals("true"))

{

lpSubscribeParam.SetFromNow(true);

}

else

{

lpSubscribeParam.SetFromNow(false);

}

string isReplace = config.GetString("subcribe", "is\_replace", "");//是否覆盖

if (isReplace.Equals("true"))

{

lpSubscribeParam.SetReplace(true);

}

else

{

lpSubscribeParam.SetReplace(false);

}

string lpApp = "xuxinpeng";

unsafe

{

lpSubscribeParam.SetAppData(Marshal.StringToBSTR(lpApp).ToPointer(), 9);//添加附加数据

}

//添加过滤字段

int nCount = config.GetInt("subcribe", "filter\_count", 0);

for (int i = 1; i <= nCount; i++)

{

String lName = String.Format("filter\_name{0}", i);

string filterName = config.GetString("subcribe", lName, "");

String lValue = String.Format("filter\_value{0}", i);

string filterValue = config.GetString("subcribe", lValue, "");

lpSubscribeParam.SetFilter(filterName, filterValue);

}

//添加发送频率

lpSubscribeParam.SetSendInterval(config.GetInt("subcribe", "send\_interval", 0));

//添加返回字段

nCount = config.GetInt("subcribe", "return\_count", 0);

for (int k = 1; k <= nCount; k++)

{

String lName = String.Format("return\_filed{0}", k);

string filedName = config.GetString("subcribe", lName, "");

lpSubscribeParam.SetReturnFiled(filedName);

}

DisplayText("开始订阅\n");

int iRet = -1;

CT2UnPacker unpack = null;

iRet = lpSub.SubscribeTopicEx(lpSubscribeParam, 5000, out unpack, null);

if (unpack != null)

{

unpack.Dispose();

}

lpSubscribeParam.Dispose();

if (iRet > 0)

{

subIndex = iRet;

String strInfo = String.Format("SubscribeTopic info:{0} 成功\n", iRet);

DisplayText(strInfo);

}

else

{

String strInfo = String.Format("SubscribeTopic info:[{0}] {1}\n", iRet, conn.GetErrorMsg(iRet));

DisplayText(strInfo);

return;

}};BizMessage.Dispose();

* 1. 发布

这里的发布代码也是针对服务器为消息中心2.0的代码。

//取发布的信息

string topicName = config.GetString("publish", "topic\_name", "");//获取发布的主题名

//获取过滤字段名字和值

int nCount = config.GetInt("publish", "filter\_count", 0);

string[] strFilterName = new String[6];

string[] strFilterValue = new String[6];

for (int i = 1; i <= nCount; i++)

{

String str = String.Format("filter\_name{0}", i);

strFilterName[i - 1] = config.GetString("publish", str, "");

str = String.Format("filter\_value{0}", i);

strFilterValue[i - 1] = config.GetString("publish", str, "");

}

DisplayText("构造发布者\n");

if (lpPub == null)

{

lpPub = conn.NewPublisher("test", 200, 5000, false);

}

if (lpPub == null)

{

String strInfo = String.Format("NewPublish Error: {0}\n", conn.GetMCLastError());

DisplayText(strInfo);

return;

}

DisplayText("开始发布\n");

//构造发布的业务包内容

CT2Packer lpOnePack = new CT2Packer(2);

lpOnePack.BeginPack();

for (int j = 0; j < nCount; j++)

{

lpOnePack.AddField(strFilterName[j], (sbyte)'S', 255, 4);

}

for (int k = 0; k < nCount; k++)

{

lpOnePack.AddStr(strFilterValue[k]);

}

lpOnePack.EndPack();

CT2UnPacker lpUnPack = lpOnePack.UnPack();

//业务包构造完毕

//调用业务的发送接口进行发布

int iRet = -1;

unsafe

{

iRet = lpPub.PubMsgByPacker(topicName, lpUnPack, 5000, null, false);

//打印错误信息

String strErr = String.Format("Next {0},{1},MsgID:{2}\n", iRet, conn.GetErrorMsg(iRet), lpPub.GetMsgNoByTopicName(topicName));

DisplayText(strErr);

lpOnePack.Dispose();

}

lpPub.Dispose();

# 附录

* 1. T2sdk错误码

-1套接字的值超过FD\_SETSIZE  
-2不存在指定编号的服务器端口  
-3包ID无效  
-4内存申请失败  
-5接收超时  
-6文件打开失败  
-7配置对象行数超过限制  
-8配置对象未指定服务器地址  
-9代理连接失败  
-10连接失败（非阻塞错误）  
-11同步连接超时或发生错误  
-12连接还未注册  
-13参数无效  
-14待发送数据长度异常  
-15发送队列满  
-16连接重连失败  
-17接收超时  
-18超出许可证的发送包数限制  
-19超出许可证的发送字节数限制  
-20注册超时  
-21SSL握手失败，可能客户端和服务端SSL版本不一致  
-22客户端被服务端校验的证书不存在，是否配置有误？  
-23该许可证允许接入客户端数已满  
-24不允许有名注册  
-25许可证不适用于T2  
-26许可证无效  
-27非注册包  
-28客户端通信密码错  
-29交换数据密码包中业务数据无效  
-30交换数据密码包中无业务数据  
-31非交换数据密码包，可能安全模式不匹配  
-32服务端在指定端口上没有进程在监听  
-33客户端被服务端校验的的证书密码错误（本地证书保护密码错），是否配置有误？  
-34客户端被服务端校验的证书过期，需要向服务端重新申请  
-35客户端校验服务端的证书不存在，证书名字是否配错？  
-36客户端校验服务端的证书密码错误  
-37服务端不支持SSL或者证书错误  
-38没有一个有效的证书（被服务端校验的证书）  
-39没有一个有效的根证书  
-40没有一个可用事件用来发送报文  
-41可用事件已经存在。

-42 mc客户端目录创建失败，请确定[logdir]配置参数目录存在

-43 没有配置mc标签下面的client\_name项

-44 mc客户端类实例创建失败

-45 业务名不合法

-101 没有业务名，初始化失败

-102 全局实例申请内存失败

-103 全局订阅者申请内存失败,创建会话超时

-104 全局发布者申请内存失败

-105 解包器解包失败

-106 esb消息解析失败

-107 订阅没有主题

-108 服务端没有这个主题

-109 订阅过滤条件非法

-110 申请打包器失败

-111 发布时没有指定发布主题

-112 发布指定的主题不存在

-113 发布信息初始化失败

-114 申请消息失败

-115 发布过滤条件非法

-116 入发送队列失败

-117 写文件失败

-118 发布内容为空

-119 目录创建失败,

-120 内存不足

-121 发布操作列表校验失败

-122 订阅操作列表校验失败

-123 业务权限校验失败

-124 没有令牌

-125 令牌校验失败

-126 服务端暂时不可用

-127 服务端不存在此主题

-128 服务端主题被禁用

-129 取消订阅，没有对应的订阅项

-130 取消订阅，必须有主题和过滤条件

-131 收到订阅应答，但是没有对应的订阅请求

-132 取消订阅，先天订阅者不可以取消

-133 订阅没有回调接口

-134 申请不到会话

-135 连接状态非法

-136 会话号超出上限

-137 没有对应的会话信息

-138 SYN|ACK超时

-139 心跳超时

-140 重传次数超出限制

-141 序号错误

-142 t2sdk 连接断开

-143 此主题有踢人策略，有相同订阅已经存在

-144 会话管理类申请内存失败

-145 同名的订阅者已经存在

-146 没有订阅者名字

-147 转发错误

-148 没有配置mc标签下面的client\_name项

-149 没有发布者名字

-150 该业务名发布者已存在

-151 该业务名订阅者已存在

* 1. INI配置文件说明

| 配置项 | | 含义 |
| --- | --- | --- |
| t2sdk | lang | 语言取值及含义如下：  2052：中文（缺省）  1033：英文 |
| errormsg | 用户指定错误信息文件，错误号从-70到0，缺省为””。 |
| servers | 指定连接服务IP和端口号，格式如127.0.0.1:9008。 |
| if\_error\_log | 底层出错是否记录日志，1:记录，非1:不记录（缺省）。 |
| logdir | 记录日志目录，缺省值为当前目录。 |
| if\_sendRecv\_log | 是否记录发送和接收报文，1:记录，非1:不记录（缺省）。 |
| init\_recv\_buf\_size | 指定接收缓存大小，缺省为1024字节。 |
| send\_queue\_size | 指定发送队列大小，缺省为100。 |
| init\_send\_buf\_size | 指定发送缓存大小，缺省为1024字节。 |
| event\_count | 指定事件个数，缺省为10。 |
| heartbeat\_time | 指定心跳时间，缺省为30。 |
| connect\_time | 指定连接超时，缺省为5。 |
| support\_multi | 指定是否支持多线程同步调用，0:不支持（缺省）非0:支持。 |
| license\_file | 指定许可证文件，缺省为license.dat。 |
| license\_no | 指定许可证，缺省为””。 |
| login\_name | 指定有名连接名。 |
| enable\_nagle | 指定是否禁用Nagle算法，1:禁用，非1:开启（缺省）。 |
| license\_pwd | 指定许可证文件保护密码。 |
| addipmac | 指定是否在注册包中加IP和MAC，1:是（缺省），非1:否。 |
| machine\_code | 指定机器码。 |
|
| auto\_reconnect | 指定是否自动重连，1:是，非1:否（缺省）。 |
| proxy |  |  |
| proxy\_type | 指定代理类型，缺省为””。 |
| ip | 指定代理IP，缺省为””。 |
| port | 指定代理端口，缺省为””。 |
| user\_name | 指定代理用户名，缺省为NULL。 |
| password | 指定代理用户密码，缺省为NULL。 |
| safe | safe\_level | 安全级别，取值及含义如下：   * none：T2协议明文（缺省）， * pwd：T2协议通信密码， * ssl：T2协议SSL通道， * t1或hscomsvr：T1协议。 |
| client\_id | 指定客户ID，缺省为””。 |
| comm\_pwd | 指定通信密码，缺省为””。 |
| ca\_file | 指定CA文件，缺省为NULL。 |
| ca\_pwd | 指定CA密码，缺省为NULL。 |
| cert\_file | 指定Cert文件，缺省为NULL。 |
| cert\_pwd | 指定Cert密码，缺省为NULL。 |
| check\_server\_cert | 指定是否验证服务器证书，0:否，非0:是（缺省）。 |
| version | 指定SSL版本，缺省为””。 |
| verify\_depth | 指定校验深度，缺省为0。 |