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

T2SDKJAVA版

开发手册

|  |  |
| --- | --- |
| 文档版本 | V1.0 |
| 发布日期 | 2014-01-10 |

目录

[前言 I](#_Toc378151015)

[产品简介 I](#_Toc378151016)

[读者对象 I](#_Toc378151017)

[手册概况 I](#_Toc378151018)

[缩略语/术语 II](#_Toc378151019)

[文档版本 II](#_Toc378151020)

[1 开发包简介 1](#_Toc378151021)

[1.1 发布包说明 1](#_Toc378151022)

[1.2 版本信息获取 1](#_Toc378151023)

[1.3 获取方式 1](#_Toc378151024)

[2 开发流程 2](#_Toc378151025)

[2.1 同步开发流程 2](#_Toc378151026)

[2.2 异步开发流程 3](#_Toc378151027)

[3 开发接口 4](#_Toc378151028)

[3.1 配置文件说明 4](#_Toc378151029)

[3.1.1 performance 5](#_Toc378151030)

[3.1.2 parent 6](#_Toc378151031)

[3.1.3 limit 6](#_Toc378151032)

[3.1.4 member 7](#_Toc378151033)

[3.1.5 SSL(非必配) 7](#_Toc378151034)

[3.1.6 callBackMethods(非必配) 7](#_Toc378151035)

[3.1.7 logAdapter 8](#_Toc378151036)

[3.1.8 extCrypt(非必配) 8](#_Toc378151037)

[3.1.9 mdbsupport(非必配) 8](#_Toc378151038)

[3.1.10 mc（非必配） 8](#_Toc378151039)

[3.1.11 subscriber（非必配） 9](#_Toc378151040)

[3.1.12 publisher（非必配） 9](#_Toc378151041)

[3.1.13 mclog（非必配） 9](#_Toc378151042)

[3.2 T2SDK服务接口【IT2Services】 10](#_Toc378151043)

[3.2.1 T2SDK服务实例化（getInstance） 10](#_Toc378151044)

[3.2.2 T2SDK服务初始化（init） 10](#_Toc378151045)

[3.2.3 重载T2SDK服务初始化（init） 10](#_Toc378151046)

[3.2.4 T2SDK服务启动（start） 11](#_Toc378151047)

[3.2.5 T2SDK服务停止（stop） 11](#_Toc378151048)

[3.2.6 重载T2SDK服务停止（stop） 12](#_Toc378151049)

[3.2.7 获取上级节点接口（getClient） 12](#_Toc378151050)

[3.2.8 增加上级服务端接口（appendAParent） 13](#_Toc378151051)

[3.2.9 获取当前所有上级连接信息（getQueryAllConnsCommand） 14](#_Toc378151052)

[3.2.10 获取对应的配置参数信息（getParametersCommand） 15](#_Toc378151053)

[3.2.11 获取对应配置文件的名字（getT2sdkConfigString） 16](#_Toc378151054)

[3.2.12 设置服务对应配置文件的名字（setT2sdkConfigString） 16](#_Toc378151055)

[3.3 客户端接口【IClient】 16](#_Toc378151056)

[3.3.1 同步发送（sendReceive） 16](#_Toc378151057)

[3.3.2 异步发送（send） 18](#_Toc378151058)

[3.4 异步回调接口【ICallBackMethod】 19](#_Toc378151059)

[3.4.1 回调执行接口（execute） 19](#_Toc378151060)

[3.5 消息事件接口【IEvent】 20](#_Toc378151061)

[3.5.1 构造消息事件实例 20](#_Toc378151062)

[3.5.2 设置事件的服务号（setServiceId） 20](#_Toc378151063)

[3.5.3 设置事件的服务别名（setServiceAlias） 21](#_Toc378151064)

[3.5.4 获取事件的服务号（getServiceId） 21](#_Toc378151065)

[3.5.5 获取事件的服务别名（getServiceAlias） 22](#_Toc378151066)

[3.5.6 设置事件的类型（setEventType） 22](#_Toc378151067)

[3.5.7 获取事件的类型（getEventType） 22](#_Toc378151068)

[3.5.8 将请求事件转换为应答事件（changeToresponse） 23](#_Toc378151069)

[3.5.9 获取业务数据结果集（getEventDatas） 23](#_Toc378151070)

[3.5.10 添加业务数据结果集（putEventData） 23](#_Toc378151071)

[3.5.11 设置返回码（setReturnCode） 24](#_Toc378151072)

[3.5.12 获取返回码（getReturnCode） 24](#_Toc378151073)

[3.5.13 设置错误码（setErrorCode） 25](#_Toc378151074)

[3.5.14 获取错误码（getErrorNo） 25](#_Toc378151075)

[3.5.15 获取错误信息（getErrorInfo） 25](#_Toc378151076)

[3.5.16 获取所有属性的个数（getAttributeCount） 26](#_Toc378151077)

[3.5.17 根据名字获取属性类型（getAttributeName） 26](#_Toc378151078)

[3.5.18 根据名字判断属性是否存在（hasAttribute） 27](#_Toc378151079)

[3.5.19 添加INT类型的属性和值（setIntegerAttributeValue） 27](#_Toc378151080)

[3.5.20 获取INT类型的属性值（getIntegerAttributeValue） 27](#_Toc378151081)

[3.5.21 添加STRING类型的属性和值（setStringAttributeValue） 28](#_Toc378151082)

[3.5.22 获取STRING类型的属性值（getStringAttributeValue） 28](#_Toc378151083)

[3.5.23 添加STRING数组类型的属性和值（setStringArrayAttributeValue） 29](#_Toc378151084)

[3.5.24 获取STRING数组类型的属性值（getStringArrayAttributeValue） 29](#_Toc378151085)

[3.5.25 添加BYTE数组类型的属性和值（setByteArrayAttributeValue） 30](#_Toc378151086)

[3.5.26 获取BYTE数组类型的属性值（getByteArrayAttributeValue） 30](#_Toc378151087)

[3.5.27 删除指定名字的属性（removeAttribute） 30](#_Toc378151088)

[3.6 结果集接口【IDataset】 31](#_Toc378151089)

[3.6.1 域类型和转换 31](#_Toc378151090)

[3.6.2 结果集打包规则 32](#_Toc378151091)

[3.6.3 设置结果集名字（setDatasetName） 33](#_Toc378151092)

[3.6.4 获取结果集名字（getDatasetName） 33](#_Toc378151093)

[3.6.5 获取列个数（getColumnCount） 33](#_Toc378151094)

[3.6.6 获取指定列的类型（getColumnType） 34](#_Toc378151095)

[3.6.7 获取列名对应的列索引（findColumn） 34](#_Toc378151096)

[3.6.8 获取列索引对应的列名（getColumnName） 35](#_Toc378151097)

[3.6.9 根据列名获取INT值（getInt） 35](#_Toc378151098)

[3.6.10 根据列索引获取INT值（getInt） 36](#_Toc378151099)

[3.6.11 根据列名获取INT值并显示指定默认值（getInt） 36](#_Toc378151100)

[3.6.12 根据列索引获取INT值并显示指定默认值（getInt） 36](#_Toc378151101)

[3.6.13 根据列名获取LONG值（getLong） 37](#_Toc378151102)

[3.6.14 根据列索引获取LONG值（getLong） 37](#_Toc378151103)

[3.6.15 根据列名获取LONG值并显示指定默认值（getLong） 38](#_Toc378151104)

[3.6.16 根据列索引获取LONG值并显示指定默认值（getLong） 38](#_Toc378151105)

[3.6.17 根据列名获取DOUBLE值（getDouble） 39](#_Toc378151106)

[3.6.18 根据列索引获取DOUBLE值（getDouble） 39](#_Toc378151107)

[3.6.19 根据列名获取DOUBLE值并显示指定默认值（getDouble） 39](#_Toc378151108)

[3.6.20 根据列索引获取DOUBLE值并显示指定默认值（getDouble） 40](#_Toc378151109)

[3.6.21 根据列名获取BYTE数组值（getByteArray） 40](#_Toc378151110)

[3.6.22 根据列索引获取BYTE数组值（getByteArray） 41](#_Toc378151111)

[3.6.23 根据列名获取BYTE数组值并显示指定默认值（getByteArray） 41](#_Toc378151112)

[3.6.24 根据列索引获取BYTE数组值并显示指定默认值（getByteArray） 42](#_Toc378151113)

[3.6.25 根据列名获取STRING值（getString） 42](#_Toc378151114)

[3.6.26 根据列索引获取STRING值（getString） 43](#_Toc378151115)

[3.6.27 根据列名获取STRING值并显示指定默认值（getString） 43](#_Toc378151116)

[3.6.28 根据列索引获取STRING值并显示指定默认值（getString） 43](#_Toc378151117)

[3.6.29 根据列名获取STRING数组值（getStringArray） 44](#_Toc378151118)

[3.6.30 根据列索引获取STRING数组值（getStringArray） 44](#_Toc378151119)

[3.6.31 根据列名获取STRING数组值并显示指定默认值（getStringArray） 45](#_Toc378151120)

[3.6.32 根据列索引获取STRING数组值并显示指定默认值（getStringArray） 45](#_Toc378151121)

[3.6.33 定位到指定行（locateLine） 46](#_Toc378151122)

[3.6.34 移动到第一行前一行（beforeFirst） 46](#_Toc378151123)

[3.6.35 判断是否还有下一行（hasNext） 46](#_Toc378151124)

[3.6.36 移动到下一行（next） 47](#_Toc378151125)

[3.6.37 根据默认类型新增一列（addColumn） 47](#_Toc378151126)

[3.6.38 根据传入类型新增一列（addColumn） 47](#_Toc378151127)

[3.6.39 根据列名修改列类型（modifyColumnType） 48](#_Toc378151128)

[3.6.40 根据列索引修改列类型（modifyColumnType） 48](#_Toc378151129)

[3.6.41 根据列索引添加BYTE数组值（updateByteArray） 49](#_Toc378151130)

[3.6.42 根据列名添加BYTE数组值（updateByteArray） 49](#_Toc378151131)

[3.6.43 根据列索引添加DOUBLE值（updateDouble） 50](#_Toc378151132)

[3.6.44 根据列名添加DOUBLE值（updateDouble） 50](#_Toc378151133)

[3.6.45 根据列索引添加INT值（updateInt） 51](#_Toc378151134)

[3.6.46 根据列名添加INT值（updateInt） 51](#_Toc378151135)

[3.6.47 根据列索引添加LONG值（updateLong） 52](#_Toc378151136)

[3.6.48 根据列名添加LONG值（updateLong） 52](#_Toc378151137)

[3.6.49 根据列索引添加STRING值（updateString） 53](#_Toc378151138)

[3.6.50 根据列名添加STRING值（updateString） 53](#_Toc378151139)

[3.6.51 根据列索引添加STRING数组值（updateStringArray） 54](#_Toc378151140)

[3.6.52 根据列名添加STRING数组值（updateStringArray） 54](#_Toc378151141)

[3.6.53 新增一行数据（appendRow） 55](#_Toc378151142)

[3.6.54 清空所有行（clear） 55](#_Toc378151143)

[3.6.55 清空所有行和列（clearAll） 55](#_Toc378151144)

[3.6.56 MapReader和MapWriter 56](#_Toc378151145)

[3.7 结果集集合接口【IDatasets】 56](#_Toc378151146)

[3.7.1 添加数据集（putDataset） 56](#_Toc378151147)

[3.7.2 根据名字获取数据集（getDataset） 57](#_Toc378151148)

[3.7.3 根据索引获取数据集名字（getDatasetName） 57](#_Toc378151149)

[3.7.4 根据索引获取数据集（getDataset） 58](#_Toc378151150)

[3.7.5 获取数据集个数（getDatasetCount） 58](#_Toc378151151)

[3.7.6 清空所有的数据集（clear） 58](#_Toc378151152)

[3.8 消息中心服务【MCServers】 59](#_Toc378151153)

[3.8.1 MCServers服务初始化（MCInit） 59](#_Toc378151154)

[3.8.2 MCServers服务摧毁（Destroy） 59](#_Toc378151155)

[3.8.3 获取订阅主题信息（GetTopic） 60](#_Toc378151156)

[3.8.4 获取订阅者接口（GetSubscriber） 62](#_Toc378151157)

[3.8.5 获取发布者接口（GetPublisher） 62](#_Toc378151158)

[3.8.6 获取错误信息（GetErrorMsg） 63](#_Toc378151159)

[3.9 订阅者接口【ISubscriber】 63](#_Toc378151160)

[3.9.1 订阅主题（SubscribeTopic） 63](#_Toc378151161)

[3.9.2 取消订阅（CancelSubscribeTopic） 64](#_Toc378151162)

[3.9.3 获取已订阅的信息（GetSubcribeTopic） 65](#_Toc378151163)

[3.10 订阅参数对象【MCSubscribeParameter】 66](#_Toc378151164)

[3.10.1 设置主题名字（SetTopicName） 66](#_Toc378151165)

[3.10.2 添加附加数据（SetAppData） 66](#_Toc378151166)

[3.10.3 添加过滤字段（SetFilter） 67](#_Toc378151167)

[3.10.4 追加返回字段（AppendReturnFiled） 67](#_Toc378151168)

[3.10.5 设置补缺标志（SetFromNow） 68](#_Toc378151169)

[3.10.6 设置覆盖标志（SetReplace） 68](#_Toc378151170)

[3.10.7 设置发送间隔（SetSendInterval） 69](#_Toc378151171)

[3.10.8 获取主题名字（GetTopicName） 69](#_Toc378151172)

[3.10.9 获取附加数据（GetAppData） 70](#_Toc378151173)

[3.10.10 获取过滤字段（GetFilter） 70](#_Toc378151174)

[3.10.11 获取返回字段（GetReturnFiled） 70](#_Toc378151175)

[3.10.12 获取补缺标志（GetFromNow） 70](#_Toc378151176)

[3.10.13 获取覆盖标志（GetReplace） 71](#_Toc378151177)

[3.10.14 获取发送间隔（GetSendInterval） 71](#_Toc378151178)

[3.11 过滤条件接口【MCFilter】 71](#_Toc378151179)

[3.11.1 根据过滤字段名字获取对应值（GetFilterValue） 71](#_Toc378151180)

[3.11.2 获取过滤条件个数（GetCount） 72](#_Toc378151181)

[3.11.3 添加过滤条件（SetFilter） 72](#_Toc378151182)

[3.11.4 比较两个过滤条件是否相等（equal） 72](#_Toc378151183)

[3.12 订阅者回调接口【ISubCallback】 73](#_Toc378151184)

[3.12.1 接收到发布消息的回调（OnReceived） 73](#_Toc378151185)

[3.12.2 接收到踢人消息的回调（OnReceived） 73](#_Toc378151186)

[3.13 发布者接口【IPublisher】 74](#_Toc378151187)

[3.13.1 发布消息（PubMsg） 74](#_Toc378151188)

[4 示例代码 75](#_Toc378151189)

[4.1 配置文件t2sdk-config.xml 75](#_Toc378151190)

[4.2 同步发送代码 76](#_Toc378151191)

[4.3 异步发送代码 78](#_Toc378151192)

[4.4 订阅代码 80](#_Toc378151193)

[4.5 发布代码 82](#_Toc378151194)

[5 常见问题 84](#_Toc378151195)

[5.1 如何正确初始化JAVA T2SDK？ 84](#_Toc378151196)

[5.2 T2SDK是线程安全的么？ 84](#_Toc378151197)

[5.3 频繁调用T2Services.getInstance().getClient是否会导致对象数量膨胀，是否需要将client单例化？ 85](#_Toc378151198)

[5.4 T2Services.getInstance().stop方法在什么时候调用？ 85](#_Toc378151199)

[5.5 JAVA T2SDK是长连接吗？ 85](#_Toc378151200)

[5.6 在网速较慢的场景下(例如ping大于50ms)，如何使用JAVA T2SDK? 85](#_Toc378151201)

[5.7 JAVA T2SDK的调用会阻塞通讯线程么？ 85](#_Toc378151202)

[5.8 使用异步发送方法send，配置的回调ID是否可以定义为任意整数？ 86](#_Toc378151203)

前言

产品简介

T2SDK JAVA版主要用于对接各种基于CRES2.0平台搭建的系统，包括两个开发包：

* com.hundsun.t2sdk-ext-\*.jar：负责对接CRES2.0，完成同步/异步发包以及接收主推消息的功能，下文简称t2sdk。（\*表示jar包的版本号，例如1.1.8）
* com.hundsun.mcsdk-\*.jar：完成发布订阅的功能，下文简称mcsdk。（\*表示jar包的版本号，例如1.1.8）

读者对象

本指南主要适用于以下人员：

* JAVA开发人员，对接基于CRES2.0平台搭建的系统时使用。

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 | 介绍开发包支持的系统，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 | 介绍具体开发接口的函数说明。 |
| 4示例代码 | 介绍开发的demo的实例代码。 |
| 5 常见问题 | 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JAVA版本的开发包。下文中的服务端如无特殊说明，均指CRES2.0。 |
| 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的订阅发布不能使用该接口，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。 |
| mcsdk | 消息中心2.0开发包 | 负责与消息中心2.0进行订阅发布的开发包。 |
| T |  |  |
| t2sdk | 金融基础件2.0开发包 | 对接CRES2.0的开发包，负责同步/异步发送请求、主推消息接收等。 |
| **U** |  |  |
| UFX | 恒生统一接入系统 | 客户端的统一接入控制系统，基于CRES2.0平台搭建，可以使用T2SDK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，基于CRES2.0平台搭建，可以使用T2SDK接入。 |

文档版本

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 日期 | 修改者 | 修改说明 |
| V2.0 | 20131210 | 许欣芃 | 创建 |
|  | 20140318 | 赵真 | limit配置段中增加属性ufxPassword |
|  | 20140331 | 赵真 | 3.1.9章节完善extCrypt说明 |
|  | 20140520 | 赵真 | SSL配置段添加属性keyManageAlgorithm描述 |
|  | 20140604 | 赵真 | 1:修改章节3.1.9 extCrypt  2：修改回调方法相关内容 |

# 开发包简介

## 发布包说明

名称：com.hundsun.t2sdk-ext-\*.jar com.hundsun.mcsdk-\*.jar

开发语言：JAVA

支持操作系统：所有支持JAVA虚拟机的操作系统。

依赖项：t2sdk需要依赖外部的加密解密JAR包（commons-codec-1.4.jar）以及日志记录的JAR包（dom4j-1.6.1.jar）。mcsdk依赖t2sdk。

## 版本信息获取

版本信息的获取方法如下：

* 发布包名.jar之前的\*表示版本号，例如1.1.8。
* 通过WINRAR工具，打开JAR包，到META-INF目录下获取MANIFEST.MF文件，用工具打开这个文件，其中的Product-Version列即版本信息。

## 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，请联系相应恒生项目的接口人获取开发包。

# 开发流程

本章节主要介绍同步和异步两种开发模式的流程。

## 同步开发流程



## 异步开发流程



# 开发接口

本章主要介绍开发包提供的所有接口和函数参数说明。如无特殊说明，接口都是线程安全的。

## 配置文件说明

T2SDK的jar包是通过配置文件与上级建立连接的，配置文件的名称默认为t2sdk-config.xml，具体的配置参数说明如下：

<?xml version="1.0" encoding="UTF-8"?>

<t2sdk-configuration>

<performance heartbeatTime="4"

acquireConnWaitTime="1000" registerTime="5" reconnInterval="3"

callBackTime="1000" enmCompress="false" compressionThreshold="0" />

<parents>

<parent parentName="jres\_ar\_testplatform" safeLevel="NONE">

<limit licenseFile="c:/jcc-license.dat" encrypt="jcc" />

<members>

<member no="5" address="192.168.194.46" port="10112"

charset="utf-8" poolSize="3" />

</members>

</parent>

<parent parentName="jres\_ar\_testplatform" safeLevel="PWD">

<pwd clientId="\*\*\*" commPwd="\*\*\*"/>

<limit licenseFile="c:/jcc-license.dat" encrypt="jcc" />

<members>

<member no="5" address="192.168.194.46" port="10112"

charset="utf-8" poolSize="3" />

</members>

</parent>

<parent parentName="jres\_ar\_testplatform" safeLevel="SSL">

<ssl protocol="SSL" keyManageAlgorithm="SunX509"

trustManageAlgorithm="SunX509" keyStoreType="jks"

trustKeyStoreType="jks" keyCertFile="F:\instance\workspace\cert\client.pfx"

keyCertPwd="12345678" trustCertFile="F:\instance\workspace\cert\hundsun.pfx"

trustCertPwd="12345678" />

<limit licenseFile="c:\client\_license.dat" encrypt="HSBlowfish"/>

<members>

<member no="5" charset="utf-8" address="192.168.194.46" port="8001" poolSize="1"/>

</members>

</parent>

</parents>

<callBackMethods>

<!-- 消息中心的回调必须增加这一项 -->

<method id="-1" className="com.hundsun.mcapi.MCT2sdkCallback" />

<method id="6"

className=" com.hundsun.t2sdk.impl.test.SayHelloMethod" />

</callBackMethods>

<logAdapter className="com.hundsun.t2sdk.impl.util.CommonLogAdapter" />

<extCrypt class="com.hundsun.libsecurity.CryptDecorator" />

<mdbsupport>

<listener class="com.hundsun.t2sdk.impl.mdb.MDBTip"/>

</mdbsupport>

//消息中心订阅发布相关的配置

<mc parentName="mcserver" type="0" cachedir="D:/mc" inittimeout="3000">

<!-- 订阅者信息配置 -->

<!-- subname：订阅者名字 subcallback：订阅者的回调类，必须有，否则构造失败 timeout：订阅者的初始化超时时间 -->

<subscriber subname="xuxp" subcallback="test.mctestcallback" timeout=

"3000"/>

<!-- 发布者信息配置 -->

<!-- msgCount：本地缓存的消息数量 -->

<publisher pubname="xuxp" msgCount="200" isResetNo="false"/>

<!-- mc日志目录配置 -->

<mclog dir="D:\mclog"/>

</mc>

</t2sdk-configuration>

### performance

performance是T2SDK的基本属性，也是T2SDK定义的Parent所共有的属性，具体含义如下表所示。

| **属性** | **含义** |
| --- | --- |
| heartbeatTime | 客户端发起心跳的时间间隔。单位为秒，取值范围（2~120），小于2取2，大于120取120，默认值为5。 |
| acquireConnWaitTime | 从连接池获取可用连接的最大等待时间。单位为毫秒，取值范围(0~5000) ，默认值为100。 |
| registerTime | TCP连接到T2成功注册的最长时间，超过此时间，该连接会被close。单位为秒，取值范围（1~15），默认值为5。 |
| reconnInterval | 客户端连接断开后，再重新建立连接的时间间隔。单位为秒，取值范围(1~60) ，默认值为3。 |
| callBackTime | 同步发送等待时间。默认值为1000，单位为毫秒。 |
| localServerName | T2SDK客户端名称，不同的T2SDK客户端localServerName必须不同。  如果该参数不配置，则当作匿名客户端处理，建议一般客户以匿名方式处理。 |
| enmCompress | 是否启用业务包体压缩，包括以下选项：   * false：不采用压缩（默认值） * true：采用压缩 |
| compressionThreshold | 压缩阈值，单位为KB，0表示忽略阈值，直接压缩，默认值为0。 |

### parent

parent服务节点描述，每个服务节点可以由一个或多个member组成，每个member以连接池的形式存在。在T2SDK中以IClient的形式对外提供服务。

|  |  |
| --- | --- |
| **属性** | **含义** |
| parentName | 本节点主动去连接的T2节点名。 |
| safeLevel | 安全级别，支持以下两种级别：   * SSL：安全连接 * NONE：明文 |
| noNameConnection | 是否为无名连接， true表示无名连接，默认为false。 |

### limit

limit用于描述T2连接的许可证文件，每个服务节点都拥有自己的连接许可证文件。

|  |  |
| --- | --- |
| **属性** | **含义** |
| licenseFile | 客户端证书。 |
| encrypt | 许可证书加密算法，支持jcc（忽略大小写）和HSBlowfish两种算法。  只有指定为jcc，才采用jcc的加密算法，否则采用HSBlowfish算法。 |
| ufxPassword | 该属性非必须。当ufxPassword存在，注册时将ufxPassword打入字段21。 |

### member

服务节点成员描述，每个服务节点可由一个或多个成员组成，以实现集群。每个成员可以通过配置指定连接池的大小。

| **属性** | **含义** |
| --- | --- |
| no | 成员编号。单个成员时可不填，默认为 0；多个成员时，成员编号不能相同。 |
| address | IP地址，必填。 |
| port | 监听的端口地址（T2端口），必填。 |
| charset | 采用的编码字符集，可以为空，默认为"utf-8"。 |
| poolSize | 与该组成员节点中每个节点建立的连接个数，可不填，默认值为2，取值范围（1 ~ 64），小于1取1，大于64取64。 |

### ssl(非必配)

当parent的safeLevel为SSL时，用于描述SSL详细配置。

|  |  |
| --- | --- |
| **属性** | **含义** |
| protocol | 协议，默认为SSL。 |
| keyCertFile | 客户端证书路径。 |
| keyCertPwd | 客户端证书口令。 |
| keyManageAlgorithm | 证书算法，取值为IBMX509和SunX509，使用IBM-JDK请配置为IBMX509 |
| trustCertFile | 根证书路径，1.1.6版本后可以不配置。 |
| trustCertPwd | 根证书口令，1.1.6版本后可以不配置。 |

### pwd (非必配)

当parent的safeLevel为PWD时，用于描述PWD详细配置。

|  |  |
| --- | --- |
| **属性** | **含义** |
| clientId | 指定客户ID。 |
| commPwd | 指定通信密码。 |

### callBackMethods(非必配)

执行IClient.send(异步发送)之后的回调方法定义，通讯线程会根据应答包的sendID来执行对应的回调方法。

|  |  |
| --- | --- |
| **属性** | **含义** |
| id | 匹配T2PACK中的sendID，-1是消息中心订阅发布功能必须的配置。如果需要订阅发布功能，就必须进行如下配置：  <method id="-1" className="com.hundsun.mcapi.MCT2sdkCallback" />  该类在mcsdk中实现。 |
| className | 负责处理的class，该类实现接口ICallBackMethod。 |

注意

用户定义的回调id必须为正整数，0和负数被系统保留使用。

### logAdapter

日志适配器，该配置指向用户定义的日志输出器，与T2SDK相关的日志信息会通过export方法输出。

|  |  |
| --- | --- |
| **属性** | **含义** |
| className | 日志适配器类，继承抽象类AbstractLogAdapter。默认日志使用System.out.println输出，建议可以继承AbstractLogAdapter后合并到具体项目的日志中。 |

### extCrypt(非必配)

用户扩展的加密实例，实现类必须继承接口ICrypt。

|  |  |
| --- | --- |
| **属性** | **含义** |
| class | 用户扩展的加密实例类。 |

1：该属性用以解决恒生密码加密算法，可以通过配置具体的加密实现来使用。版本1.1.12之前首先下载加密库到工程，下载地址

<ftp://192.168.51.31/History/Patch/patch20130118(T2SDK)/> libSecurity.jar

从版本1.1.13之后t2sdk的jar包中自带了libSecurity的相关实现，因此无需添加libSecurity.jar

2：t2sdk-config.xml中添加配置指向加密库

<extCrypt class='com.hundsun.libsecurity.CryptDecorator'/>

3：通过T2Service.getCrypt获得ICrypt实例，调用HsPwdEncode(inBytes, outBytes)实现加密

4：需要注意的是，不要对空字符串进行加密。对空密码加密本身不会报错，返回'\x02'字符，

在java中，会得到字符串"\u0002"，该字符串以字符串形式入参进行T2调用(用打包器打包后

解包会有问题，二进制入参没有问题），服务端解包时可能会丢失入参数据。因此使用该函数

前要对空字符串进行判断,空字符串不要加密。

### mdbsupport(非必配)

支持CRES2.0的内存数据库。

|  |  |
| --- | --- |
| 属性 | 含义 |
| threadCount | mdb任务线程池线程个数，默认值为2。 |
| queueSize | mdb任务线程池队列长度，默认值为30。 |
| listener-class | 用户实现接口com.hundsun.t2sdk.interfaces.core.channel.IMessageHandler的类列表，当mdb类请求到达后会被执行。 |

### mc（非必配）

支持消息中心的配置相关项。

| 属性 | 含义 |
| --- | --- |
| parentName | 消息中心用到的连接名字，是上面parent节点中配置的一个上级的名字。 |
| type | 消息中心实现的功能，包括以下选项：   * 0：订阅&发布（默认值） * 1：只订阅 * 2：只发布 |
| cachedir | 本地消息中心存放缓存消息的目录。 |
| inittimeout | 消息中心初始化需要的最大时间，初始化时消息中心需要和服务端通信。 |

### subscriber（非必配）

构造订阅者的相关配置。

|  |  |
| --- | --- |
| 属性 | 含义 |
| subname | 订阅者的名字。根据该名字，上层应用可以获取对应的订阅者。 |
| subcallback | 订阅者的回调类配置，由上层自己实现，负责实现订阅回调接口。 |
| timeout | 订阅者创建的超时时间。 |

### publisher（非必配）

构造发布者的相关配置。

|  |  |
| --- | --- |
| 属性 | 含义 |
| pubname | 发布者的名字，根据该名字，上层应用可以获取对应的发布者。 |
| msgCount | 发布者在发布端缓存的消息个数。 |
| isResetNo | 发布者是否重新开始自己的发布序号。 |

### mclog（非必配）

订阅发布日志存放目录配置。

|  |  |
| --- | --- |
| 属性 | 含义 |
| dir | 订阅发布过程中，系统发生错误时都会记录日志，每天会生成一个新的文件。 |

## T2SDK服务接口【IT2Services】

T2SDK服务接口主要用于维护与上级的连接，包括获取连接、发包/收包和最后关闭所有连接等功能。

接口需要导入的包信息如下：

import com.hundsun.t2sdk.interfaces;

### T2SDK服务实例化（getInstance）

T2SDK服务接口的实例在整个应用中是一个单例，它的实例化代码如下：

import com.hundsun.t2sdk.impl.client.T2Services;

IT2Services sdkServices = T2Services.getInstance();

### T2SDK服务初始化（init）

函数原型

public void init() throws T2SDKException;

输入参数

无

有无异常

失败会抛出异常，详细信息可以在异常中获取。

用法说明

在获取到服务实例后，首先需要调用该函数进行初始化，之后才可以调用其他函数。初始化是按照配置文件进行，配置文件的默认名称是t2sdk-config.xml，也可以调用setT2sdkConfigString设置需要的配置文件。

初始化主要进行获取配置中上级连接的信息，并创建对应的IClient实例的工作。

注意

该函数在整个应用中只可以调用一次，并且不可以多线程调用。

### 重载T2SDK服务初始化（init）

函数原型

public void init(IConfiguration config) throws T2SDKException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IConfiguration config | 配置对象，配置上级连接的信息。 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

用法说明

该函数根据传入的配置对象进行初始化是另一种方式的初始化。对象的构造可以调用：

DefaultConfigurationHelper configHelp=new DefaultConfigurationHelper();

IConfiguration configuration = configHelp.loadFileToConfiguration(path);

注意

该函数在整个应用中只可以调用一次，并且不可以多线程调用。

### T2SDK服务启动（start）

函数原型

public void start();

输入参数

无

有无异常

无

用法说明

该函数用于启动服务，在服务初始化之后调用。主要功能是开启线程，然后与上级创建连接，使服务可以处于工作状态。

注意

该函数在整个应用中只可以调用一次，并且不可以多线程调用。

### T2SDK服务停止（stop）

函数原型

public void stop();

输入参数

无

有无异常

无

用法说明

应用退出之前，可以调用该函数停止服务，以断开与上级的连接，并释放socket连接等资源。服务停止后，如果需要再次运行，则需要重新执行init(),start()方法。

注意

该函数在整个应用中只可以调用一次，并且不可以多线程调用。

该函数在应用程序退出时可以不调用。

### 重载T2SDK服务停止（stop）

函数原型

public void stop(String parentName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String parentName | 上级连接名字 | 输入参数 |

有无异常

无

用法说明

该函数用于断开与指定上级的连接。服务停止之后，与该上级的连接都不可以再用。

注意

该函数在整个应用中只可以调用一次，并且不可以多线程调用。

该函数在应用程序退出时可以不调用。

### 获取上级节点接口（getClient）

该接口用于获取应用待连接的上级服务端节点。一个应用可以配置多个上级服务端，对于这些上级服务端而言，该应用就是这些服务端对应的客户端。针对配置的上级，每个名字都是唯一的，不可以重复。

函数原型

public IClient getClient(String parentName) throws T2SDKException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String parentName | 上级连接名字，配置文件中配置 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

成功返回对应上级连接的客户端接口，使用该接口可以与对应的上级服务端通信。下文有该接口的详细说明。

用法说明

在初始化以及启动服务后，应用层就可以准备业务数据并获取对应服务端的客户端接口，进行数据的收发操作。该接口可以多线程调用。

### 增加上级服务端接口（appendAParent）

函数原型

public void appendAParent(IElement appendAConfig) throws T2SDKException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IElement appendAConfig | 对应的上级连接的配置信息 | 输入参数 |

有无异常

无

用法说明

传入配置信息添加parent节点，该方法用于运行中切换或新增Parent。如果传入的配置中parentName已经存在，则覆盖原有节点。该方法一次完成配置加载和启动，无需再执行start进行启动。

参数IElement的获取通过下面代码实现：

DefaultConfigurationHelper configHelp=new DefaultConfigurationHelper();

IConfiguration configuration = configHelp.loadFileToConfiguration(path);

IElement element = configuration.getRoot().getChild("attribute-1");

### 获取当前所有上级连接信息（getQueryAllConnsCommand）

函数原型

public IDataset getQueryAllConnsCommand();

输入参数

无

有无异常

无

返回

返回当前T2SDK服务中所有的上级连接信息，返回是一个IDataset接口，关于IDataset接口的操作说明，请参见“3.6结果集接口【IDataset】”。

用法说明

IDataset的返回结果中，结果集名称是all connections，对应的字段信息描述如下表所示。

| 字段名称 | 类型 | 字段说明 |
| --- | --- | --- |
| remoteNodeName | 字符串 | 客户端名字，根据该名字可以获取对应服务端的连接。 |
| remoteNodeNo | 整数 | 客户端连接池的编号。 |
| connectionNo | 整数 | 具体的连接编号。 |
| remoteCharset | 字符串 | 连接使用的字符集。 |
| status | 整数 | 连接状态，包括以下状态：   * 0：断开 * 1：socket已创建 * 2：连接已经注册 * 3：连接当前空闲 * 4：连接忙碌 * -1：连接错误 |
| connectionType | 整数 | 连接类型：   * 0：连接未知节点 * 1：连接的是JRES节点 |
| address | 字符串 | 上级的IP地址。 |
| port | 字符串 | 上级的端口。 |
| sendPackCount | 整数 | 连接的发包个数。 |
| sendDataSize | 整数 | 连接的发包总字节数。 |
| recvPackCount | 整数 | 连接的收包个数。 |
| recvDataSize | 整数 | 连接的收包总字节数。 |
| lastActiveTime | 字符串 | 连接上一次活跃时间，时间格式为yyyy-MM-dd HH:mm:ss:SS。 |
| lastDisconnTime | 字符串 | 连接上一次断开时间，时间格式为yyyy-MM-dd HH:mm:ss:SS。 |
| lastConnectTime | 字符串 | 连接上一次重连时间，时间格式为yyyy-MM-dd HH:mm:ss:SS。 |
| reconnCount | 整数 | 连接重连次数。 |

### 获取对应的配置参数信息（getParametersCommand）

函数原型

public IDataset getParametersCommand();

输入参数

无

返回

返回当前T2SDK服务中所有配置信息。

用法说明

IDataset的返回结果中，对应的字段信息描述如下表所示。

| 字段名称 | 类型 | 字段说明 |
| --- | --- | --- |
| localServiceName | 字符串 | 作为有名连接时的客户端名字。 |
| heartbeatTime | 字符串 | 心跳间隔时间，单位为秒。 |
| callBackTime | 字符串 | 同步请求应答的超时时间，单位为毫秒。 |
| acquireConnWaitTime | 字符串 | 从连接池获取可用连接的最大等待时间，单位为毫秒。 |
| reconnInterval | 字符串 | 重连的时间间隔，单位为秒。 |
| registerTime | 字符串 | 注册的时间间隔，单位为秒。 |
| isDebugMode | 字符串 | 是否是调试模式，默认为false。 |

### 获取对应配置文件的名字（getT2sdkConfigString）

函数原型

public String getT2sdkConfigString();

输入参数

无

返回

获得t2sdk服务对应的配置文件名及路径。

### 设置服务对应配置文件的名字（setT2sdkConfigString）

函数原型

public void setT2sdkConfigString(String configString);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String configString | 配置文件名以及路径 | 输入参数 |

返回

设置t2sdk服务的配置文件名及路径(默认为t2sdk-config.xml)，如果要调用该方法重设t2sdk服务的配置名，则需要在init方法前调用。

## 客户端接口【IClient】

客户端接口的实例化是通过IT2Services的getClient接口来实现。

### 同步发送（sendReceive）

函数原型

public IEvent sendReceive(IEvent event) throws T2SDKException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IEvent event | 需要发送的消息接口 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

该函数是一个同步等待应答的发送接收函数。成功会返回服务端的应答，以IEvent接口模式返回。这里的同步等待时间是在配置文件中配置的，由performance标签里面的callBackTime参数决定。

下面有三个同名重载函数，方便不同场景下使用同步发送。

**函数原型一**

public IEvent sendReceive(IEvent event, long waitTime)

throws T2SDKException;

**输入参数**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IEvent event | 需要发送的消息接口。 | 输入参数 |
| long waitTime | 同步的超时等待时间，单位是毫秒。 | 输入参数 |

**有无异常**

失败会抛出异常，详细信息可以在异常中获取。

**返回**

与上面函数接口一样，但此函数的超时时间是应用层自己决定的。

**函数原型二**

public IEvent sendReceive(Map<String, Object> tagMap,

Map<Object, Object> dataMap) throws T2SDKException;

**输入参数**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| Map<String, Object> tagMap | 消息头部字段的map映射表，例如功能号、包类型等信息。 | 输入参数 |
| Map<Object, Object> dataMap | 业务具体参数的map映射表，例如，具体的登录功能号里面需要传入的用户名、密码等字段。 | 输入参数 |

**有无异常**

失败会抛出异常，详细信息可以在异常中获取。

**返回**

发送的内容换成MAP的格式。但是不可以传入超时时间，通过配置文件决定超时时间。

**函数原型三**

public IEvent sendReceive(Map<String, Object> tagMap,

Map<Object, Object> dataMap, long waitTime) throws T2SDKException;

**输入参数**

| 参数名称 | 参数说明 | 用法 |
| --- | --- | --- |
| Map<String, Object> tagMap | 消息头部字段的map映射表，例如，功能号、包类型等信息。 | 输入参数 |
| Map<Object, Object> dataMap | 业务具体参数的map映射表。例如，具体的登录功能号里面需要传入的用户名、密码等字段。 | 输入参数 |
| long waitTime | 同步的超时等待时间，单位是毫秒。 | 输入参数 |

**有无异常**

失败会抛出异常，详细信息可以在异常中获取。

**返回**

发送内容是MAP格式，同时上层可以控制超时时间。

### 异步发送（send）

函数原型

public void send(IEvent event) throws T2SDKException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IEvent event | 需要发送的消息接口 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

发送成功后立即返回，应答的消息通过回调返回。如果异步发送的请求由应答返回，则需要在请求中增加EventTagdef.TAG\_SENDERID，该标识会在应答中返回，就会根据配置里面的类回调，用户定义的回调id必须为正整数，0和负数被系统保留使用。

//请求中增加发送者编号

event.setIntegerAttributeValue(EventTagdef.TAG\_SENDERID, 1);

client.send(event);

对应的配置文件的配置项：

<callBackMethods>

<method id="1" className="test.T2sdkCallback" />

</callBackMethods>

下面是同一个函数的重载，是采用MAP形式传递要发送的数据。建议使用上面的IEvent接口发送函数，因为IEvent接口比较灵活，可以添加任何数据。

**函数原型**

public void send(Map<String, Object> tagMap, Map<Object, Object> dataMap) throws T2SDKException;

**输入参数**

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| Map<String, Object> tagMap | 消息头部字段的map映射表，例如功能号、包类型等信息。 | 输入参数 |
| Map<Object, Object> dataMap | 业务具体参数的map映射表。例如具体的登录功能号里面需要传入的用户名、密码等字段。 | 输入参数 |

**有无异常**

失败会抛出异常，详细信息可以在异常中获取。

**返回**

发送成功后立刻返回，应答的消息通过回调返回。

## 异步回调接口【ICallBackMethod】

该接口是IClient接口中异步发送的回调接口，异步发送在函数调用后会立即返回，等服务端应答返回时，就会通过该回调接口返回给上层应用。因此该接口是上层应用自己实现的。

### 回调执行接口（execute）

函数原型

public void execute(IEvent event,ClientSocket clientSocket);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IEvent event | 服务端返回的报文，以IEvent方式返回。 | 回调 |
| ClientSocket clientSocket | 报文来自的套接字，通过该套接字可以直接返回应答。ClientSocket拥有三个方法，其中send方法用做返回报文信息，getEnvironment和getClientEnvironment方法获得T2SDK的公有属性信息和连接属性信息 | 回调 |

用法说明

第一个回调返回的是具体的消息，通过IEvent接口解析；第二个回调返回的是报文来自哪个连接，通过ClientSocket的send直接发送相应的应答。类似代码如下：

public class T2sdkCallback implements ICallBackMethod {

public void execute(IEvent arg0, ClientSocket arg1) {

printEvent(arg0);

//构造需要回复的IEvent event

**arg1.send(event);**

}

}

## 消息事件接口【IEvent】

该接口负责封装与后台服务通信的消息。后台的消息格式类似于fix的消息，格式大致如下：

3=0|5=100|1=100|8=…………|11=1

服务端和客户端已经约定每个数字的特定含义。具体的数字都有常量定义，在com.hundsun.t2sdk.interfaces.share.event. EventTagdef有详细的说明。

### 构造消息事件实例

一个全新的消息构造如下所示：

IEvent event = **ContextUtil.getServiceContext().getEventFactory()**

**.getEventByAlias**("339602", EventType.ET\_REQUEST);

消息通过工厂获取，获取时需要传入服务的名称以及事件的类型。事件分为四种类型：一般请求ET\_REQUEST，一般应答ET\_ANSWER，内部请求 ET\_INTERNAL\_REQUEST，内部应答ET\_INTERNAL\_ANSWER，这些类型的常量定义在com.hundsun.t2sdk.interfaces.share.event.EventType中，根据业务接口需要传入需要的业务类型。获取到消息之后就可以根据具体服务需要的参数，调用下面的接口进行消息封装。

### 设置事件的服务号（setServiceId）

函数原型

void setServiceId(String serviceId);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String serviceId | 服务号 | 输入参数 |

返回

无

用法说明

根据上层应用的需要，设置相应的后台服务的服务号，服务号是后台服务端定义的。

### 设置事件的服务别名（setServiceAlias）

函数原型

void setServiceAlias(String alias);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String alias | 服务别名 | 输入参数 |

返回

无

用法说明

服务别名是对应服务号的，是后台服务的另一个称呼。

### 获取事件的服务号（getServiceId）

函数原型

String getServiceId()

输入参数

无

返回

返回消息中的服务号。

### 获取事件的服务别名（getServiceAlias）

函数原型

String getServiceAlias()

输入参数

无

返回

返回消息中的服务别名。

### 设置事件的类型（setEventType）

函数原型

void setEventType(int type)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int type | 消息事件的类型 | 输入参数 |

返回

无

用法说明

事件分为四种类型：

* 一般请求ET\_REQUEST
* 一般应答ET\_ANSWER
* 内部请求 ET\_INTERNAL\_REQUEST
* 内部应答ET\_INTERNAL\_ANSWER

这些类型的常量定义在com.hundsun.t2sdk.interfaces.share.event.EventType中。根据业务接口需要传入需要的业务类型。消息类型一般在构造时指定，无需单独调用该接口来指定类型。

### 获取事件的类型（getEventType）

函数原型

int getEventType();

输入参数

无

返回

返回对应的事件类型。

### 将请求事件转换为应答事件（changeToresponse）

函数原型

int changeToresponse()

输入参数

无

返回

转换成功则返回0，转换失败则返回非0。如果已经是应答类事件则返回非0。

用法说明

将一般请求事件转换为一般应答事件，将内部请求事件转换为内部应答事件。该函数一般在开发服务端程序时调用，收到了一个请求事件并经过处理之后，转成应答事件返回。

### 获取业务数据结果集（getEventDatas）

函数原型

IDatasets getEventDatas();

输入参数

无

返回

获取消息事件的业务数据集的集合，里面包含多个数据结果集。

用法说明

该接口一般用于接收服务端返回的应答事件，调用该接口获取服务端的业务应答信息，具体的信息通过IDatasets接口可以获取，IDatasets接口详细说明请参考“3.7结果集集合接口【IDatasets】”。

### 添加业务数据结果集（putEventData）

函数原型

void putEventData(IDataset dataset);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IDataset dataset | 业务数据需要传入的结果集数据 | 输入参数 |

返回

无

用法说明

上层应用根据后台服务的要求，组装需要的业务参数数据库集，然后通过该接口设置到消息中，通过消息传递给后台服务处理。如果服务需要添加多个结果集，则需要重复调用该接口。

### 设置返回码（setReturnCode）

函数原型

void setReturnCode(int returnCode);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int returnCode | 返回码 | 输入 |

返回

无

用法说明

返回码用于设置当前消息的处理成功与否，如果上层开发的应用是服务处理端，则通过该函数告知客户端处理的结果，0表示成功，其他值需要设置相应的错误码和错误信息来告知对方具体的错误原因。

### 获取返回码（getReturnCode）

函数原型

int getReturnCode();

输入参数

无

返回

相应的返回码

用法说明

返回码用于标识业务的处理成功与否，一般返回0表示成功，其他的返回码就需要根据错误码和错误信息来确定具体的错误信息。一般用于上层应用判断返回的应答是成功还是失败。

### 设置错误码（setErrorCode）

函数原型

void setErrorCode(String errorNo, String errorInfo);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String errorNo | 错误码 | 输入 |
| String errorInfo | 错误信息 | 输入 |

返回

无

用法说明

如果作为服务处理端，则通过该函数告知服务发起方相应的处理错误信息。

### 获取错误码（getErrorNo）

函数原型

String getErrorNo()

输入参数

无

返回

相应的错误码。

用法说明

如果返回码表示失败，则通过该函数获取相应的错误码，从而获取具体的错误信息。

### 获取错误信息（getErrorInfo）

函数原型

String getErrorInfo()

输入参数

无

返回

相应的错误信息

用法说明

如果返回码表示失败，则通过该函数获取相应的错误信息。

### 获取所有属性的个数（getAttributeCount）

函数原型

int getAttributeCount();

输入参数

无

返回

消息事件中属性的个数。

用法说明

获取之前设置的所有属性的个数。

### 根据名字获取属性类型（getAttributeName）

函数原型

char getAttributeType(String name);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String name | 属性名字 | 输入参数 |

返回

根据属性名返回具体的属性类型。

用法说明

属性类型主要有整型（’I’）、字符串型（’S’）、二进制数据型（’R’）、字符串数组型（’A’）。

### 根据名字判断属性是否存在（hasAttribute）

函数原型

boolean hasAttribute(String attrName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |

返回

如果存在，返回true，否则返回false。

用法说明

根据属性名字判断消息中是否已存在该字段。

### 添加INT类型的属性和值（setIntegerAttributeValue）

函数原型

void setIntegerAttributeValue(String attrName, long value);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |
| long value | 属性值 | 输入参数 |

返回

无

用法说明

根据属性名字设置整数类型的属性。如果属性名字已经存在，则覆盖前一个属性，类型采用新的数据。

### 获取INT类型的属性值（getIntegerAttributeValue）

函数原型

long getIntegerAttributeValue(String attrName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |

返回

根据属性名返回INT型的属性值，如果属性字段不存在就返回0.

用法说明

如果原来字段的类型不是整型，则按照整数类型进行强制转换，转换失败返回0。

### 添加STRING类型的属性和值（setStringAttributeValue）

函数原型

void setStringAttributeValue(String attrName, String value);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |
| String value | 属性值 | 输入参数 |

返回

无

用法说明

根据属性名设置STRING类型的属性。如果属性名已经存在则覆盖前一个属性，类型采用新的数据。

### 获取STRING类型的属性值（getStringAttributeValue）

函数原型

String getStringAttributeValue(String attrName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |

返回

根据属性名返回STRING类型的属性值，如果属性字段不存在就返回NULL。

用法说明

如果原来字段的类型不是STRING，则按照STRING类型进行强制转换，转换失败返回NULL。

### 添加STRING数组类型的属性和值（setStringArrayAttributeValue）

函数原型

void setStringArrayAttributeValue(String attrName, String[] value);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |
| String[] value | 属性值 | 输入参数 |

返回

无

用法说明

根据属性名设置String[]类型的属性。如果属性名已经存在则覆盖前一个属性，类型采用新的数据。

### 获取STRING数组类型的属性值（getStringArrayAttributeValue）

函数原型

String[] getStringArrayAttributeValue(String attrName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |

返回

根据属性名返回对应STRING[]型的属性值，如果属性字段不存在就返回NULL。

用法说明

如果原来字段的类型不是STRING[]类型，就会按照STRING[]类型进行强制转换，转换失败返回NULL。

### 添加BYTE数组类型的属性和值（setByteArrayAttributeValue）

函数原型

void setByteArrayAttributeValue(String attrName, byte[] value);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |
| byte[] value | 属性值 | 输入参数 |

返回

无

用法说明

根据属性名设置byte[]类型的属性。如果属性名已经存在则覆盖前一个属性，类型采用新的数据。

### 获取BYTE数组类型的属性值（getByteArrayAttributeValue）

函数原型

byte[] getByteArrayAttributeValue(String attrName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String attrName | 属性名字 | 输入参数 |

返回

对应属性名字的byte[]值，如果属性字段不存在则返回NULL.

用法说明

如果原来字段的类型不是byte[]类型，就会按照byte[]类型进行强制转换，转换失败返回NULL。

### 删除指定名字的属性（removeAttribute）

函数原型

void removeAttribute(String name);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String name | 属性名字 | 输入参数 |

返回

无

用法说明

删除消息中不需要的属性字段。

## 结果集接口【IDataset】

IDataset是进行数据传输的基础数据结构，是一个二维表结构，表头为域名（或称字段名、字段域名）。IDataset可以有多行，每行的每个字段值就是相应域的值。可以认为IDataset是由一个或者多个KEY完全一样的map组成，每一个map对应IDataset的一行。IDataset可以通过名字和索引两种方式访问行信息；名字就是列名，索引就是列索引，从1开始。

表头除了定义字段域的名字之外，还定义了该列数据的类型。





注意

IDataSet的行和列都是从1开始。

### 域类型和转换

域类型用于定义某一列数据的数据类型，但并未规定该列中所有的数据必须是这个类型。例如，IDataset某一列的域类型为整型，但是该列中有一个值为字符串"123"，则认为该值也是有效的，因为在获取该值时，IDataset会做尽力转换，将字符串"123"转换为整数123返回。IDataset支持的数据类型包括如下：



说明

表中的obj，表示IDataset中的值；上表说明了如果当前值的类型是“兼容类型”，则要采用“Type”所指定的类型来获取，返回的值为“转换后的值”。上表中，未出现的转换映射被视为“无法转换”。

### 结果集打包规则

首先需要构造一个全新的结果集，构造代码如下：

IDataset dataset = DatasetService.getDefaultInstance().getDataset();

有了结果集对象之后，接着添加列和行的数据。添加行列数据有顺序要求，必须先添加列数据再添加行数据，行与行之间需要通过appendRow显式说明，如下代码所示：

//添加列

dataset.addColumn("identity\_type");

dataset.addColumn("op\_branch\_no");

dataset.addColumn("op\_entrust\_way");

dataset.addColumn("op\_station");

//添加第一行

dataset.appendRow();

dataset.updateString("identity\_type", "");

dataset.updateString("op\_branch\_no", "");

dataset.updateString("op\_entrust\_way", "7");

//添加第二行

dataset.appendRow();

dataset.updateString("identity\_type", "");

dataset.updateString("op\_branch\_no", "");

dataset.updateString("op\_entrust\_way", "7");

### 设置结果集名字（setDatasetName）

函数原型

void setDatasetName(String name);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String name | 结果集名字 | 输入参数 |

返回

无

用法说明

设置当前结果集的名字，服务端可以根据设置的名字获取对应的结果集。结果集有默认名字PLATFORM\_DEFAULT\_DATASET\_NAME。如果一个IDAtasets里面有多个同名的结果集，则根据名字获取结果集就会出现问题。

### 获取结果集名字（getDatasetName）

函数原型

String getDatasetName();

输入参数

无

返回

当前结果集已经设定的名字。

### 获取列个数（getColumnCount）

函数原型

int getColumnCount();

输入参数

无

返回

返回当前结果集中列的个数。

### 获取指定列的类型（getColumnType）

函数原型

char getColumnType(int column);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列的索引 | 输入参数 |

返回

返回对应索引的列类型。

用法说明

索引是从1开始，因此第一列应该传入索引值1，不是0。如果列索引无效，就返回DatasetColumnType.DS\_UNKNOWN。具体的类型如下表所示。

|  |  |  |
| --- | --- | --- |
| 列类型 | 常量定义 | 字符 |
| 未知类型 | DatasetColumnType.DS\_UNKNOWN | ‘N’ |
| 整型 | DatasetColumnType. DS\_INT | 'I' |
| 长整形 | DatasetColumnType. DS\_LONG | 'L' |
| 二进制 | DatasetColumnType. DS\_BYTE\_ARRAY | 'R' |
| 长浮点型 | DatasetColumnType. DS\_DOUBLE | 'D' |
| 字符串类型 | DatasetColumnType. DS\_STRING | 'S' |
| 字符串数组 | DatasetColumnType. DS\_STRING\_ARRAY | 'A' |

### 获取列名对应的列索引（findColumn）

函数原型

int findColumn(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

返回列名的索引。如果不存在，就返回0。

### 获取列索引对应的列名（getColumnName）

函数原型

String getColumnName(int column);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int column | 列的索引 | 输入参数 |

返回

返回对应列索引的列名。索引无效就返回NULL。

### 根据列名获取INT值（getInt）

函数原型

int getInt(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

根据列名返回对应的INT类型的值。

用法说明

指针不在有效行中或者columnName列名不存在，就会获取失败，返回默认值0。如果需要判断该返回值是默认值还是真实值，可以先调用findColumn确定列是否存在。如果列类型不是INT的，会进行转换，转换规则参见“3.6.1域类型和转换”中所示。

### 根据列索引获取INT值（getInt）

函数原型

int getInt(int columnIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |

返回

根据列索引返回对应列数据，按照INT类型返回。

用法说明

如果列索引无效，就返回默认值0。类型不匹配的话，就做相应的转换。

### 根据列名获取INT值并显示指定默认值（getInt）

函数原型

int getInt(String columnName, int def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| int def | 默认值 | 输入参数 |

返回

根据列名返回对应的INT类型的值。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回的默认值是按照传入的默认值参数返回。类型不匹配的话，就做相应的转换。

### 根据列索引获取INT值并显示指定默认值（getInt）

函数原型

int getInt(int columnIndex, int def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| int def | 默认值 | 输入参数 |

返回

根据列索引返回对应的INT类型的值。

用法说明

如果列索引不合法，返回的默认值是按照传入的默认值参数返回。类型不匹配的话，就做相应的转换。

### 根据列名获取LONG值（getLong）

函数原型

long getLong(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

根据列名返回对应的LONG类型的值。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回默认值0。类型不匹配的话，就做相应的转换。

### 根据列索引获取LONG值（getLong）

函数原型

long getLong(int columnIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |

返回

根据列索引返回对应列数据，按照LONG类型返回。

用法说明

如果列索引无效，就返回默认值0。如果类型不匹配，就做相应的转换。

### 根据列名获取LONG值并显示指定默认值（getLong）

函数原型

long getLong(String columnName, long def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| long def | 默认值 | 输入参数 |

返回

根据列名返回对应的值，用LONG类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列索引获取LONG值并显示指定默认值（getLong）

函数原型

int getInt(int columnIndex, int def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| long def | 默认值 | 输入参数 |

返回

根据列索引返回对应的值，用LONG类型返回。

用法说明

如果列索引不合法，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列名获取DOUBLE值（getDouble）

函数原型

double getDouble(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

根据列名返回对应的值，用DOUBLE类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回默认值0。如果类型不匹配，就做相应的转换。

### 根据列索引获取DOUBLE值（getDouble）

函数原型

double getDouble(int columnIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |

返回

根据列索引返回对应列数据，按照DOUBLE类型返回。

用法说明

如果列索引无效，就返回默认值0。如果类型不匹配，就做相应的转换。

### 根据列名获取DOUBLE值并显示指定默认值（getDouble）

函数原型

double getDouble(String columnName, double def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| double def | 默认值 | 输入参数 |

返回

根据列名返回对应的值，用DOUBLE类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列索引获取DOUBLE值并显示指定默认值（getDouble）

函数原型

double getDouble(int columnIndex, double def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| double def | 默认值 | 输入参数 |

返回

根据列索引返回对应的值，用DOUBLE类型返回。

用法说明

如果列索引不合法，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列名获取BYTE数组值（getByteArray）

函数原型

byte[] getByteArray(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

根据列名返回对应的值，用BYTE数组类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回默认值（new byte[0]）。如果类型不匹配，就做相应的转换。

### 根据列索引获取BYTE数组值（getByteArray）

函数原型

byte[] getByteArray(int columnIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |

返回

根据列索引返回对应列数据，按照BYTE数组类型返回。

用法说明

如果列索引无效，就返回默认值（new byte[0]）。如果类型不匹配，就做相应的转换。

### 根据列名获取BYTE数组值并显示指定默认值（getByteArray）

函数原型

byte[] getByteArray(String columnName, byte[] def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| byte[] def | 默认值 | 输入参数 |

返回

根据列名返回对应的值，用BYTE数组类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列索引获取BYTE数组值并显示指定默认值（getByteArray）

函数原型

byte[] getByteArray(int columnIndex, byte[] def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| double def | 默认值 | 输入参数 |

返回

根据列索引返回对应的值，返回值类型为BYTE数组类型。

用法说明

如果列索引不合法，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列名获取STRING值（getString）

函数原型

String getString(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

根据列名返回对应的值，返回值类型为STRING类型。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回默认值(“”)。如果类型不匹配，就做相应的转换。

### 根据列索引获取STRING值（getString）

函数原型

String getString(int columnIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |

返回

根据列索引返回对应列的数据，返回值类型为STRING类型。

用法说明

如果列索引无效，就返回默认值(“”)。如果类型不匹配，就做相应的转换。

### 根据列名获取STRING值并显示指定默认值（getString）

函数原型

String getString(String columnName, String def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| String def | 默认值 | 输入参数 |

返回

根据列名返回对应的值，返回值类型为STRING类型。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列索引获取STRING值并显示指定默认值（getString）

函数原型

String getString(int columnIndex, String def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| String def | 默认值 | 输入参数 |

返回

根据列索引返回对应的值，返回值类型为STRING类型。

用法说明

如果列索引不合法，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列名获取STRING数组值（getStringArray）

函数原型

String[] getStringArray(String columnName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |

返回

根据列名返回对应的值，按照STRING数组类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回默认值（new String[0]）。类型不匹配的话，就做相应的转换。

### 根据列索引获取STRING数组值（getStringArray）

函数原型

String[] getStringArray(int columnIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |

返回

根据列索引返回对应列数据，按照STRING数组类型返回。

用法说明

如果列索引无效，就返回默认值(new String[0])。如果类型不匹配，就做相应的转换。

### 根据列名获取STRING数组值并显示指定默认值（getStringArray）

函数原型

String[] getStringArray(String columnName, String[] def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| String[] def | 默认值 | 输入参数 |

返回

根据列名返回对应的值，按照STRING数组类型返回。

用法说明

指针不在有效行上或者columnName列名不存在，就会获取失败，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 根据列索引获取STRING数组值并显示指定默认值（getStringArray）

函数原型

String[] getStringArray(int columnIndex, String[] def);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| String def | 默认值 | 输入参数 |

返回

根据列索引返回对应的值，按照STRING数组类型返回。

用法说明

如果列索引不合法，返回的默认值是按照传入的默认值参数返回。如果类型不匹配，就做相应的转换。

### 定位到指定行（locateLine）

函数原型

void locateLine(int lineIndex);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int lineIndex | 行索引 | 输入参数 |

返回

无

用法说明

把结果集当前指针指向指定的行，行索引从1开始，如果行索引不正确就会抛出异常DatasetRuntimeException。

### 移动到第一行前一行（beforeFirst）

函数原型

void beforeFirst();

输入参数

无

返回

无

用法说明

把结果集当前指针移动到第一行前一行，一般用于开始遍历之前，把当前行指到第一行的前一行。

### 判断是否还有下一行（hasNext）

函数原型

boolean hasNext();

输入参数

无

返回

如果有下一行数据，就返回true，如果没有就返回false。

用法说明

一般用于遍历过程中，判断是否有下一行，然后执行next函数，获取下一行。

### 移动到下一行（next）

函数原型

void next();

输入参数

无

返回

无

用法说明

将指针从当前位置下移一行。DataSet 指针最初位于第一行之前，第一次调用next 方法使第一行成为当前行，第二次调用使第二行成为当前行，以此类推。调用此方法不能保证每次都成功，为避免失败，建议先调用hasNext()。

### 根据默认类型新增一列（addColumn）

函数原型

void addColumn(String colName);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String colName | 列名 | 输入参数 |

返回

无

用法说明

根据列名添加一个新的列，该列采用默认类型STRING类型。如果已经存在该列名，则覆盖。

### 根据传入类型新增一列（addColumn）

函数原型

void addColumn(String colName, int type);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String colName | 列名 | 输入参数 |
| int type | 列类型 | 输入参数 |

返回

无

用法说明

根据传入的列类型和名字，添加一个新的列。如果已经存在该列名，则覆盖。

### 根据列名修改列类型（modifyColumnType）

函数原型

void modifyColumnType(String colName, int type) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String colName | 列名 | 输入参数 |
| int type | 列类型 | 输入参数 |

返回

无

用法说明

修改指定名字对应的列的类型，只是修改列的类型，列本身已有的行数据不会更改。如果传入的列名不存在或者传入的类型无效，就会抛出异常。

### 根据列索引修改列类型（modifyColumnType）

函数原型

void modifyColumnType(int colIndex, int type) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int colIndex | 列索引 | 输入参数 |
| int type | 列类型 | 输入参数 |

返回

无

用法说明

修改指定索引列的类型，只是修改列的类型，列本身已有的行数据不会更改。如果传入的列名不存在、列类型不匹配或者传入的类型无效，就会抛出异常。

### 根据列索引添加BYTE数组值（updateByteArray）

函数原型

public void updateByteArray(int columnIndex, byte[] v) throws DatasetRuntimeException;

输入参数

| 参数名称 | 参数说明 | 用法 |
| --- | --- | --- |
| int columnIndex | 列索引 | 输入参数 |
| byte[] v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应索引列的数据，根据传入的数据填入。索引从1开始，如果列索引超出范围、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列名添加BYTE数组值（updateByteArray）

函数原型

public void updateByteArray(String columnName, byte[] v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| byte[] v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应名字列的数据，根据传入的数据填入。如果列名不存在、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列索引添加DOUBLE值（updateDouble）

函数原型

public void updateDouble(int columnIndex, double v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| double v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应索引列的数据，根据传入的数据填入。索引从1开始，如果列索引超出范围、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列名添加DOUBLE值（updateDouble）

函数原型

public void updateDouble(String columnName, double v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| double v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应名字列的数据，根据传入的数据填入。如果列名不存在，列类型不匹配，或者指针不在有效行上，就抛出异常。

### 根据列索引添加INT值（updateInt）

函数原型

public void updateInt(int columnIndex, int v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| int v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应索引列的数据，根据传入的数据填入。索引从1开始，如果列索引超出范围、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列名添加INT值（updateInt）

函数原型

public void updateInt(String columnName, int v) throws DatasetRuntimeException;

输入参数

| 参数名称 | 参数说明 | 用法 |
| --- | --- | --- |
| String columnName | 列名 | 输入参数 |
| int v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应名字列的数据，根据传入的数据填入。如果列名不存在、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列索引添加LONG值（updateLong）

函数原型

public void updateLong(int columnIndex, long v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| long v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应索引列的数据，根据传入的数据填入。索引从1开始，如果列索引超出范围、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列名添加LONG值（updateLong）

函数原型

public void updateLong(String columnName, long v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| long v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应名字列的数据，根据传入的数据填入。如果列名不存在、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列索引添加STRING值（updateString）

函数原型

public void updateString(int columnIndex, String v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| String v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应索引列的数据，根据传入的数据填入。索引从1开始，如果列索引超出范围、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列名添加STRING值（updateString）

函数原型

public void updateString(String columnName, String v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String columnName | 列名 | 输入参数 |
| String v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应名字列的数据，根据传入的数据填入。如果列名不存在、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列索引添加STRING数组值（updateStringArray）

函数原型

public void updateStringArray(int columnIndex, String[] v) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int columnIndex | 列索引 | 输入参数 |
| String[] v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应索引列的数据，根据传入的数据填入。索引从1开始，如果列索引超出范围、列类型不匹配或者指针不在有效行上，就抛出异常。

### 根据列名添加STRING数组值（updateStringArray）

函数原型

public void updateStringArray(String columnName, String[] v) throws DatasetRuntimeException;

输入参数

| 参数名称 | 参数说明 | 用法 |
| --- | --- | --- |
| String columnName | 列名 | 输入参数 |
| String[] v | 数据值 | 输入参数 |

返回

无

用法说明

在当前行上增加对应名字列的数据，根据传入的数据填入。如果列名不存在、列类型不匹配或者指针不在有效行上，就抛出异常。

### 新增一行数据（appendRow）

函数原型

public boolean appendRow();

输入参数

无

返回

添加行成功返回true，否则返回false。

用法说明

添加一行数据之前，必须调用该接口，调用之后才可以使用update方法把数据添加到这一行，一行添加完成后，添加下一行之前需要再次调用该方法。第一行数据添加之前也需要调用该函数，即在addColumn所有列添加成功之后，需要先调用该函数，才可以添加数据到第一行。

### 清空所有行（clear）

函数原型

void clear();

输入参数

无

返回

无

用法说明

清除所有的行数据，列信息会保留，可以再次新增行数据。

### 清空所有行和列（clearAll）

函数原型

void clearAll();

输入参数

无

返回

无

用法说明

清除所有的列信息和行数据，清除之后，可以重新添加列和行数据。

### MapReader和MapWriter

为简化多/单行数据集操作，提供了工具类MapWriter和MapReader，可以像java.util.Map一样读或写Idataset。

MapWriter mw = new MapWriter();

mw.put("1", 11);

mw.put("2", "22");

mw.put("3", 33.3);

mw.put("4", new byte[] { '4', '4', '4', '4' });

mw.put("5", new String[] { "5", "5", "5", "5" });

mw.put("6", new Character('A'));

IDataset ds = mw.getDataset(); // 获取相应的IDataset实例，并且保持字段存放次序

MapReader mr = new MapReader(ds);

System.out.println(mr.getString("1"));

System.out.println(mr.getString("2"));

System.out.println(mr.getString("3"));

System.out.println(mr.getString("4"));

System.out.println(mr.getStringArray("5"));

System.out.println(mr.getString("6"));

System.out.println(mr.getInt("3"));

System.out.println(mr.getDouble("7", 333)); // 如果获取字段"7"失败，则采用默认值333

说明

上面的两个工具类不保证线程安全。对于MapReader，如果出现传入的ds为空或者要获取的字段不存在等获取失败的情况，则返回默认值。

## 结果集集合接口【IDatasets】

### 添加数据集（putDataset）

函数原型

void putDataset(IDataset dataset) throws DatasetRuntimeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IDataset dataset | 数据结果集 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

无

用法说明

在数据集集合中添加一个新的结果，如果数据集的名字重复，后者将会覆盖前者。最多只能有一个数据集没有指定名字，尝试去存放两个无名的数据集，会抛出运行时异常。

### 根据名字获取数据集（getDataset）

函数原型

IDataset getDataset(String name);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String name | 数据结果集名字 | 输入参数 |

返回

对应名字的结果集存在，就返回对应结果集，如果不存在返回NULL。

用法说明

根据名字获取对应的结果集。

### 根据索引获取数据集名字（getDatasetName）

函数原型

String getDatasetName(int index);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int index | 结果集对应的下标 | 输入参数 |

返回

对应结果集的名字，没有则返回NULL。

用法说明

结果集下标从0开始，上限是结果集个数。

### 根据索引获取数据集（getDataset）

函数原型

IDataset getDataset(int index);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int index | 结果集对应的下标 | 输入参数 |

返回

根据下标获取对应的结果集，没有则返回NULL。

用法说明

结果集下标从0开始，上限是结果集个数。

### 获取数据集个数（getDatasetCount）

函数原型

int getDatasetCount();

输入参数

无

返回

获取结果集集合中的结果集个数。

### 清空所有的数据集（clear）

函数原型

void clear();

输入参数

无

返回

无

用法说明

清空当前集合中的所有结果集。

## 消息中心服务【MCServers】

通过MCServers类的静态方法，可以完成订阅发布服务初始化、订阅发布服务停止摧毁、获取主题、获取订阅者和获取发布者等功能。

### MCServers服务初始化（MCInit）

函数原型

public static void MCInit() throws T2SDKException,MCException

输入参数

无

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

无

用法说明

初始化主要是根据配置文件里的相关参数获取订阅发布服务需要使用的连接，然后根据订阅者名字，发布者名字和回调参数等初始化订阅者和发布者。初始化时会从服务端获取主题，如果主题获取失败，就会返回错误。

### MCServers服务摧毁（Destroy）

函数原型

public static void Destroy()

输入参数

无

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

无

用法说明

摧毁订阅发布服务，负责摧毁之前建立的订阅者和发布者，并释放资源。在上层应用退出时调用。

### 获取订阅主题信息（GetTopic）

函数原型

public static IDataset GetTopic(boolean byForce,int iTimeOut) throws T2SDKException,MCException

输入参数

| 参数名称 | 参数说明 | 用法 |
| --- | --- | --- |
| boolean byForce | 是否强制从服务端刷新主题。如果不从服务端获取，则直接读取本地内存，初始化过程中，主题已经缓存在本地内存中。 | 输入参数 |
| int iTimeOut | 超时时间，从服务端获取主题时的超时时间，如果从本地获取主题，该参数无意义。 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

返回主题属性组装而成的结果集接口。结果集字段说明如下：

| 字段名称 | 类型 | 字段说明 |
| --- | --- | --- |
| TopicName | 字符串 | 主题名字。 |
| TopicNo | 整型 | 主题编号。 |
| ReliableLevel | 整型 | 可靠性等级,：   * 0：尽力而为 * 1：尽力有序 * 2：内存级 * 3：文件级 * 4：系统级 |
| IssuePriority | 整型 | 发布优先级，数字越大优先级越高。默认是6，取值范围（6-13）。 |
| MsgLifetime | 整型 | 主题消息的有效期，以分钟为单位，默认为0。只对文件级和系统级的主题有效。 |
| Stutas | 整型 | 主题状态：   * 0：启用 * 1：禁用 |
| TickStrategy | 整型 | 重名踢人策略：   * 0：不需要踢人 * 1：踢之前登录的 * 2：踢后登录的 |
| BusinessVerify | 整型 | 是否需要业务校验：   * 0：不需要 * 1：需要 |
| Local | 整型 | 是否是局部主题：   * 1：局部主题 * 0：全局主题 |
| FilterField1 | 字符串 | 第1个过滤字段名字 |
| FilterField2 | 字符串 | 第2个过滤字段名字 |
| FilterField3 | 字符串 | 第3个过滤字段名字 |
| FilterField4 | 字符串 | 第4个过滤字段名字 |
| FilterField5 | 字符串 | 第5个过滤字段名字 |
| FilterField6 | 字符串 | 第6个过滤字段名字 |
| FilterField7 | 字符串 | 第7个过滤字段名字 |
| FilterField8 | 字符串 | 第8个过滤字段名字 |
| FilterField9 | 字符串 | 第9个过滤字段名字 |
| FilterField10 | 字符串 | 第10个过滤字段名字 |
| FilterField11 | 字符串 | 第11个过滤字段名字 |
| FilterField12 | 字符串 | 第12个过滤字段名字 |
| FilterField13 | 字符串 | 第13个过滤字段名字 |
| FilterField14 | 字符串 | 第14个过滤字段名字 |
| FilterField15 | 字符串 | 第15个过滤字段名字 |
| FilterField16 | 字符串 | 第16个过滤字段名字 |
| SubscribeStr | 字符串 | 订阅者操作列表 |
| PublishStr | 字符串 | 发布者操作列表 |

用法说明

上层应用在查询主题信息时，可以调用该接口。

### 获取订阅者接口（GetSubscriber）

函数原型

public static ISubscriber GetSubscriber() throws MCSubscribeException

输入参数

无

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

失败返回NULL，成功返回对应的订阅者接口。

用法说明

构造一个订阅者接口，可以订阅、取消订阅、接收发布消息、根据初始化时获取的配置项来创建订阅者，是一个单例，如果已经存在就直接返回。

### 获取发布者接口（GetPublisher）

函数原型

public static IPublisher GetPublisher() throws MCPublishException

输入参数

无

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

失败返回NULL，成功返回对应的发布者接口。

用法说明

构造一个发布者接口，可以发布消息，根据初始化时获取的配置项创建发布者，是一个单例，如果已经存在就直接返回。

### 获取错误信息（GetErrorMsg）

函数原型

public static String GetErrorMsg(int errorNo)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int errorNo | 具体的错误码 | 输入参数 |

有无异常

无

返回

返回对应的错误码的详细错误信息。

用法说明

订阅发布接口函数调用返回错误码时，可以调用该函数获取具体的错误信息。

## 订阅者接口【ISubscriber】

订阅者主要负责订阅和取消订阅，以及接收推送回来的消息，订阅接口的实例来自于MCServers的GetSubscriber函数。订阅者也需要在配置文件里面配置，具体配置内容如下：

**<subscriber subname="xuxp" subcallback="test.mctestcallback" timeout="3000"/>**

订阅者是一个单例，上层需要订阅时，就需要通过MCServers的GetSubscriber函数，传入上面配置中的”xuxp”名字，获取对应的订阅接口。

### 订阅主题（SubscribeTopic）

函数原型

public int SubscribeTopic(MCSubscribeParameter subParam,int iTimeout) throws MCSubscribeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| MCSubscribeParameter subParam | 订阅参数对象，包含主题和过滤条件等信息，具体的操作可以参考下面的订阅参数说明。 | 输入参数 |
| int iTimeout | 订阅是同步的，该参数表示超时时间。 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

返回大于0表示订阅成功；-2表示重复订阅，也可以当成订阅成功处理；其他返回值可以用上面的消息中心服务GetErrorMsg接口获取详细错误信息。

返回值也是订阅标识，为取消订阅的标识输入。

用法说明

使用该函数时需要构造自己需要的订阅参数对象，订阅参数的构造代码如下：

MCSubscribeParameter subParam = new MCSubscribeParameter();

//调用接口添加订阅属性

subParam.SetTopicName("secu.deal\_return");

………..

### 取消订阅（CancelSubscribeTopic）

函数原型

public int CancelSubscribeTopic(int subIndex) throws MCSubscribeException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| int subIndex | 订阅标识，来自于上面订阅成功的返回值 | 输入参数 |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

返回0表示取消订阅成功，其他返回值表示取消订阅失败，具体信息可以使用消息中心服务GetErrorMsg接口获取。

用法说明

取消订阅的参数来自于订阅成功的返回值，因此需要订阅成功时保存标识与订阅参数的对应关系，在程序需要取消订阅时，可以根据订阅属性对应到该标识。也可以通过下面的GetSubcribeTopic接口获取订阅的标识。

### 获取已订阅的信息（GetSubcribeTopic）

函数原型

public void GetSubcribeTopic(IDataset outDataSet);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| IDataset outDataSet | 外面传入空的IDataset结果集接口，内部会把订阅信息添加到里面 | 输出参数 |

返回

无

用法说明

所有的订阅信息会打包在结果集outDataSet里，具体的结果集返回字段如下表所示。

| 字段名称 | 类型 | 字段说明 |
| --- | --- | --- |
| SubcribeIndex | 整型 | 订阅标识，可以用于取消订阅 |
| IsBornTopic | 整型 | 先天订阅标志：   * 0：不是先天订阅 * 1：先天订阅 |
| TopicName | 字符串 | 主题名称 |
| TopicNo | 整型 | 主题编号 |
| FilterRaw | 字符串 | 过滤条件的字符串表达形式 |
| Appdata | 二进制 | 订阅的附加二进制数据 |
| SendInterval | 整型 | 发送间隔 |
| ReturnFileds | 字符串 | 返回字段字符串表达式 |
| isReplace | 整型 | 是否是替换订阅：   * 1：是 * 0：否 |
| isFromNow | 整型 | 是否补缺：   * 1：需要补缺 * 0：不需要补缺 |
| Stutas | 整型 | 订阅所处的状态   * 0：补缺 * 1：实时发布状态 * 2：补缺结束状态 |
| QueueCount | 整型 | 队列长度，表示补缺完成之前，实时发布过来的消息的积压个数。 |

## 订阅参数对象【MCSubscribeParameter】

该对象作为订阅接口SubscribeTopic的参数传入，主要用于设置和获取相关参数，包括主题名字、附加数据、过滤字段、返回字段、覆盖标志、补缺标志和发送间隔等。

### 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 | 输入参数 |

返回

无

用法说明

订阅的主题名字。名字必须是服务端配置的主题，可以通过MCServers 服务的GetTopic()函数来获取服务端配置的主题。

### 添加附加数据（SetAppData）

函数原型

public void SetAppData(byte[] data)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| byte[] data | 附加数据 | 输入参数 |

返回

无

用法说明

附加数据是指订阅时由订阅者送入，然后订阅服务端在推送消息时，每个消息发布都会原样返回的数据。一般用于框架程序定位到订阅具体属于某个窗口时。

### 添加过滤字段（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 | 过滤字段名字 | 输入参数 |
| String filterValue | 过滤字段值 | 输入参数 |

返回

无

用法说明

过滤字段是在主题里定义的，过滤字段名字需要根据主题里面的属性来设置。如果设置的过滤字段不在主题定义里面，服务端会将其当成自定义过滤字段处理，自定义过滤字段最多允许两个。自定义过滤字段和主题已定义的过滤字段区别在于，后台服务端处理自定义过滤字段的效率比较低，一般不建议使用自定义过滤。如果需要订阅某个主题的所有消息，则不设置过滤字段。

### 追加返回字段（AppendReturnFiled）

函数原型

public void AppendReturnFiled(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 | 输入参数 |

返回

无

用法说明

订阅者可以要求服务端针对某个主题的消息，只返回规定的字段，主要目的是减少带宽占用，排除不用的字段信息。该接口主要用于追加返回字段，可以多次调用，如果设置的字段不在发布包里，则该字段无效。多个返回字段之间用‘\2’字符隔开。

### 设置补缺标志（SetFromNow）

函数原型

public void SetFromNow(boolean bFromNow)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| boolean bFromNow | 是否补缺，true表示补缺，false表示不补缺，默认不补缺 | 输入参数 |

返回

无

用法说明

补缺是指订阅端不在线期间，如果文件级及系统级的主题有消息发布出来，等订阅端再订阅时，可以找服务端补缺这个期间的报文。

### 设置覆盖标志（SetReplace）

函数原型

public void SetReplace(boolean bReplace)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| boolean bReplace | 是否覆盖，true表示覆盖，false表示不覆盖，默认不覆盖 | 输入参数 |

返回

无

用法说明

覆盖是指上层应用发起很多个订阅，如果想取消之前所有的订阅，可以在发起新的订阅时，指定为覆盖方式的订阅，就可以取消之前所有的订阅，只留下当前的订阅。

### 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，单位是秒。默认是实时推送，没有间隔。 | 输入参数 |

返回

无

用法说明

发送间隔是指订阅端希望服务端向自己推送报文的间隔，不要实时的推送。

### 获取主题名字（GetTopicName）

函数原型

public String GetTopicName()

输入参数

无

返回

返回订阅参数里面的主题名字。

### 获取附加数据（GetAppData）

函数原型

public byte[] GetAppData()

输入参数

无

返回

返回订阅参数里面的附加数据，以二进制的数组方式返回。

### 获取过滤字段（GetFilter）

函数原型

public MCFilter GetFilter()

输入参数

无

返回

返回订阅参数里面的过滤字段，以过滤对象的方式返回，具体过滤对象的接口下面介绍。

### 获取返回字段（GetReturnFiled）

函数原型

public String GetReturnFiled()

输入参数

无

返回

返回订阅参数里所有的返回字段，用字符’\2’间隔。

### 获取补缺标志（GetFromNow）

函数原型

public boolean GetFromNow()

输入参数

无

返回

返回订阅参数里的补缺标志。

### 获取覆盖标志（GetReplace）

函数原型

public boolean GetReplace()

输入参数

无

返回

返回订阅参数里的覆盖标志。

### 获取发送间隔（GetSendInterval）

函数原型

public int GetSendInterval()

输入参数

无

返回

返回订阅参数里面的发送间隔。

## 过滤条件接口【MCFilter】

过滤条件是在存放订阅时，根据主题配置的过滤字段传入过滤字段名字和值，便于服务端根据需要推送客户端关心的数据，是一个字段名字和值一一对应的数据结构。

### 根据过滤字段名字获取对应值（GetFilterValue）

函数原型

public String GetFilterValue(String fileterName)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String fileterName | 过滤字段的名字 | 输入参数 |

返回

返回过滤条件中对应过滤字段名字的值，没有对应的过滤字段值就返回NULL。

### 获取过滤条件个数（GetCount）

函数原型

public int GetCount()

输入参数

无

返回

返回过滤条件中过滤字段的个数。

### 添加过滤条件（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 | 过滤字段的名字 | 输入参数 |
| String filterValue | 过滤字段的值 | 输入参数 |

返回

设置过滤条件信息，如果过滤字段已经存在，则覆盖原来过滤字段对应的值。

### 比较两个过滤条件是否相等（equal）

函数原型

public boolean equal(MCFilter dstFilter)

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| MCFilter dstFilter | 被比较的过滤字段 | 输入参数 |

返回

相等返回true，失败则返回false。参数传入的过滤条件与当前过滤条件的字段名和字段值一一比较，不相等就返回false。

## 订阅者回调接口【ISubCallback】

该回调接口由上层程序实现，并在配置文件中配置，配置内容如下：

**<subscriber subname="xuxp" subcallback="test.mctestcallback" timeout="3000"/>**

只要是该订阅者订阅的消息，在发布的时候都会回调上层的test.mctestcallback类。

### 接收到发布消息的回调（OnReceived）

函数原型

public void OnReceived(String topicName,IEvent event);

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String topicName | 主题名字。 | 回调 |
| IEvent event | 发布过来的完整消息，从中获取具体的发布内容。 | 回调 |

用法说明

发布消息的所有字段都存在event中，上层应用通过event结构获取各个字段的内容，从而获取需要的数据，具体的接口介绍可以参考IEvent的接口说明。

### 接收到踢人消息的回调（OnReceived）

函数原型

public void OnRecvTickMsg(MCSubscribeParameter param,String tickMsgInfo);

输入参数

| 参数名称 | 参数说明 | 用法 |
| --- | --- | --- |
| MCSubscribeParameter param | 被踢掉的订阅对象 | 回调 |
| String tickMsgInfo | 踢人的详细信息 | 回调 |

用法说明

该接口回调时表明自己的订阅被其他同名订阅踢掉，具体哪个订阅被踢，以及被踢的相关信息在接口参数中返回。

## 发布者接口【IPublisher】

发布者主要负责发布消息到消息中心，发布者对象是一个单例，可通过MCServers的GetPublisher函数获取，对应的发布者配置信息如下所示：

**<publisher pubname="xuxp" msgCount="200" isResetNo="false"/>**

与订阅一样，上层应用需要通过传入名字”xuxp”获取发布者，进行发布消息操作。

### 发布消息（PubMsg）

函数原型

public int PubMsg(String topicName,IDataset pubDataSet,int iTimeout) throws MCPublishException;

输入参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数说明 | 用法 |
| String topicName | 主题名字 | 输入参数 |
| IDataset pubDataSet | 具体发布的消息内容，以结果集的形式发布 | 输入参数 |
| Int iTimeout | 超时时间 | - |

有无异常

失败会抛出异常，详细信息可以在异常中获取。

返回

返回0表示成功，其他返回值表示失败，详细的错误信息可以用消息中心服务GetErrorMsg接口获取。

用法说明

pubDataSet是一个结果集的形式，该结果集要求必须包含发布的主题规定的过滤字段，否则就会发布失败。

# 示例代码

本章主要是代码的示例，分成下面四个开发模式。本文未展示完整代码，完整代码请参考发布包中的例子。

## 配置文件t2sdk-config.xml

<?xml version="1.0" encoding="UTF-8"?>

<t2sdk-configuration>

<performance heartbeatTime="400"

acquireConnWaitTime="3000" registerTime="5" reconnInterval="3"

localServerName="mctest" callBackTime="5000" enmCompress="true" />

<parents>

<parent parentName="mcserver" safeLevel="NONE">

<limit licenseFile="c:\Hundsun\HsAdmin\Bin\FBaseAdmin\Config\client\_license.dat" encrypt="HSBlowfish" />

<members>

<member no="5" address="192.168.94.43" port="8002"

charset="gbk" poolSize="1" />

</members>

</parent>

</parents>

<callBackMethods>

<method id="3" className="com.jres.test.CallBack" />

<!-- 消息中心的回调必须增加这一项 -->

<method id="-1" className="com.hundsun.mcapi.MCT2sdkCallback" />

</callBackMethods>

<logAdapter className="com.hundsun.t2sdk.impl.util.CommonLogAdapter" />

<mc parentName="mcserver" type="0" cachedir="./mc" inittimeout="3000">

<subscriber subname="xuxp" subcallback="test.mctestcallback" timeout="3000"/>

<publisher pubname="xuxp" msgCount="200" isResetNo="false"/>

<mclog dir="D:\mclog"/>

</mc>

</t2sdk-configuration>

## 同步发送代码

public class JresTest {

public static void main(String[] args) {

// TODO Auto-generated method stub

IClient client = null;

T2Services server = T2Services.getInstance();

// 获取T2SDK配置文件中配置的连接

try {

server.init();

server.start();

// myserver是配置在配置文件中的服务

client = server.getClient("myserver");

// 获取event

IEvent event = ContextUtil.getServiceContext().getEventFactory()

.getEventByAlias("88", EventType.ET\_REQUEST);

// 往event中添加dataset

IDataset dataset = DatasetService.getDefaultInstance().getDataset();

dataset.addColumn("branch\_no");

dataset.addColumn("operator\_no");

dataset.addColumn("fund\_account");

dataset.appendRow();

dataset.updateString("branch\_no", "1");

dataset.updateString("operator\_no", "1");

dataset.updateString("fund\_account", "2");

event.putEventData(dataset);

// 同步发送如下

IEvent rsp = client.sendReceive(event, 10000);

//先判断返回值

if(rsp.getReturnCode() != EventReturnCode.I\_OK){ //返回错误

System.out.println(rsp.getErrorNo() +" : " + rsp.getErrorInfo());

}else{

StringBuilder sb = new StringBuilder();

//获得结果集

IDatasets result = rsp.getEventDatas();

//获得结果集总数

int datasetCount = result.getDatasetCount();

//遍历所有的结果集

for (int i = 0; i < datasetCount; i++) {

sb.append("\n" + "dataset - " + result.getDatasetName(i)

+ "\n");

// 开始读取单个结果集的信息

IDataset ds = result.getDataset(i);

int columnCount = ds.getColumnCount();

// 遍历单个结果集列信息

for (int j = 1; j <= columnCount; j++) {

sb.append(ds.getColumnName(j));

sb.append("|");

sb.append(ds.getColumnType(j));

sb.append("\n");

}

// 遍历单个结果集记录，遍历前首先将指针置到开始

ds.beforeFirst();

while (ds.hasNext()) {

sb.append("\n");

ds.next();

for (int j = 1; j <= columnCount; j++) {

sb.append(ds.getString(j));

}

}

}

System.out.println(sb);

}

// 同步发送到此结束

// 最后

server.stop();

} catch (T2SDKException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

## 异步发送代码

异步方式首先需要实现回调接口，回调接口实现如下：

public class CallBack implements ICallBackMethod{

@Override

public void execute(IEvent arg0, ClientSocket arg1) {

// TODO Auto-generated method stub

//DatasetService.printDataset(arg0.getEventDatas().getDataset(0));

//先判断返回值

if(arg0.getReturnCode() != EventReturnCode.I\_OK){ //返回错误

System.out.println(arg0.getErrorNo() +" : " + arg0.getErrorInfo());

}else{

StringBuilder sb = new StringBuilder();

//获得结果集

IDatasets result = arg0.getEventDatas();

//获得结果集总数

int datasetCount = result.getDatasetCount();

//遍历所有的结果集

for (int i = 0; i < datasetCount; i++) {

sb.append("\n" + "dataset - " + result.getDatasetName(i)

+ "\n");

// 开始读取单个结果集的信息

IDataset ds = result.getDataset(i);

int columnCount = ds.getColumnCount();

// 遍历单个结果集列信息

for (int j = 1; j <= columnCount; j++) {

sb.append(ds.getColumnName(j));

sb.append("|");

sb.append(ds.getColumnType(j));

sb.append("\n");

}

// 遍历单个结果集记录，遍历前首先将指针置到开始

ds.beforeFirst();

while (ds.hasNext()) {

sb.append("\n");

ds.next();

for (int j = 1; j <= columnCount; j++) {

sb.append(ds.getString(j));

}

}

}

System.out.println(sb);

}

}

}

异步发送代码如下所示：

public class JresTest {

public static void main(String[] args) {

// TODO Auto-generated method stub

IClient client = null;

T2Services server = T2Services.getInstance();

// 获取T2SDK配置文件中配置的连接

try {

server.init();

server.start();

// myserver是配置在配置文件中的服务

client = server.getClient("myserver");

// 获取event

IEvent event = ContextUtil.getServiceContext().getEventFactory()

.getEventByAlias("88", EventType.ET\_REQUEST);

// 往event中添加dataset

IDataset dataset = DatasetService.getDefaultInstance().getDataset();

dataset.addColumn("branch\_no");

dataset.addColumn("operator\_no");

dataset.addColumn("fund\_account");

dataset.appendRow();

dataset.updateString("branch\_no", "1");

dataset.updateString("operator\_no", "1");

dataset.updateString("fund\_account", "2");

event.putEventData(dataset);

// 异步发送如下

// 注意，在配置中配置的回调函数<method id="3" className="com.jres.test.CallBack"

// />的id号必须填入event,否则回调不回应答

event.setAttributeValue(EventTagdef.TAG\_SENDERID, 3);

client.send(event);

// 这样异步发送出去后，应答会在com.jres.test.CallBack回调中返回

// 异步发送到此结束

// 最后

server.stop();

} catch (T2SDKException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

## 订阅代码

下面是消息中心2.0的订阅代码，后台必须配置消息中心2.0才可以。

public class TestSubscribe {

public static void main(String[] args) throws MCSubscribeException {

T2Services server = T2Services.getInstance();

// TODO Auto-generated method stub

//启动T2SDK，只需要在服务器启动时调用一次就可以了

try {

server.init();

} catch (T2SDKException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

server.start();

//初始化MC服务

try {

MCServers.MCInit();

} catch (T2SDKException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (MCException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

//获取MC服务中订阅接口

ISubscriber subscriber = MCServers.GetSubscriber();

MCSubscribeParameter subParam = new MCSubscribeParameter();

subParam.SetTopicName("secu.rzrq\_contract\_flow");

byte[] data = {1,3,5,8,9,10,2,3,0};

subParam.SetAppData(data);

subParam.SetFromNow(true);

subParam.SetReplace(true);

subParam.SetFilter("branch\_no", "B");

subParam.SetFilter("fund\_account", "600625");

int iRet = subscriber.SubscribeTopic(subParam, 3000);

//返回iRet>0表示成功

System.out.println("RESULT:"+iRet);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("输入任意字符取消订阅");

try {

System.in.read();

} catch (IOException e) {

// TODO Auto-generated catch block

//e.printStackTrace();

}

iRet = subscriber.CancelSubscribeTopic(iRet);

System.out.println("CancelSubscribeTopic:"+subParam.GetTopicName()+" 返回码："+iRet);

System.out.println("输入任意字符退出");

try {

System.in.read();

} catch (IOException e) {

// TODO Auto-generated catch block

//e.printStackTrace();

}

//释放资源

MCServers.Destroy();

server.stop();

System.out.println("退出！");

}

}

相应的回调接口实现代码如下：

public class mctestcallback implements ISubCallback{

public static int g\_iCount = 0;

@Override

public void OnReceived(String topicName, IEvent event) {

// TODO Auto-generated method stub

g\_iCount++;

//if(g\_iCount%100 == 0)

System.out.println("Count:"+g\_iCount);

System.out.println(topicName);

DatasetService.printDataset(event.getEventDatas().getDataset(0));

}

@Override

public void OnRecvTickMsg(MCSubscribeParameter param,String tickMsgInfo) {

// TODO Auto-generated method stub

}

}

## 发布代码

public class TestPublish {

public static void main(String[] args) {

T2Services server = T2Services.getInstance();

// TODO Auto-generated method stub

try {

// 启动T2SDK，只需要在服务器启动时调用一次就可以了

server.init();

server.start();

// 初始化MC服务

MCServers.MCInit();

// 获取数据集

IDataset dataset = DatasetService.getDefaultInstance().getDataset();

// 填充数据集字段

dataset.addColumn("branch\_no");

// dataset.addColumn("operator\_no");

dataset.addColumn("fund\_account");

dataset.appendRow();

dataset.updateString("branch\_no", "B");

// dataset.updateString("operator\_no", "1");

dataset.updateString("fund\_account", "600625");

// 获取MC服务中发布接口

IPublisher publisher = MCServers.GetPublisher();

// 发布消息

while (true) {

int iRet = publisher.PubMsg("secu.rzrq\_contract\_flow", dataset,

5000);

System.out.println("发布返回结果：" + iRet);

System.out.println("是否退出，[y]退出，[其他字符]继续");

char ch = '0';

try {

ch = (char) System.in.read();

} catch (IOException e) {

// TODO Auto-generated catch block

// e.printStackTrace();

}

if (ch == 'y') {

break;

}

}

// 释放资源

MCServers.Destroy();

server.stop();

} catch (T2SDKException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (MCException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (MCPublishException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

# 常见问题

## 如何正确初始化JAVA T2SDK？

使用T2Services.getInstance().init(); T2Services.getInstance().start(); 这两个方法初始化T2SDK时，只需在在整个JAVA进程启动时执行一次即可。

如果在webServer中使用T2SDK（如tomcat），建议在ServletContextListener的contextInitialized方法中执行；如果写的是JAVA程序，建议在JAVA代码的静态代码段中执行。

典型的错误用法，在每次业务代码调用同步发送(sendReceive)之前来做T2SDK的init和start，这样会存在两个问题。首先init和start使得连接被重置，而导致之前的应答不能返回。此外T2SDK采用的是长连接模式，如果每次都做init和start会增加连接建立的开销，抛弃了T2SDK的优点。

## T2SDK是线程安全的么？

T2SDK的同步发送和异步发送方法

T2Services.getInstance().getClient(clientName).sendReceive 和send是线程安全的工具类方法。即在不同的JAVA 线程中调用sendReceive或者send本身是不会发生问题的，但是调用者要确保发送相关的参数不会被跨线程调用。 譬如下面这种场景： Thread 1和Thread 2是执行同一代码的不同线程。

Thread 1

**static IEvent event;**

……

T2Services.getInstance().getClient(clientName).sendReceive(event);

Thread 2

**static IEvent event;**

……

T2Services.getInstance().getClient(clientName).sendReceive(event);

这里错误，是因为将同步发送的参数指向了一个static的成员，那么当Thread 1和Thread 2对event操作时，会发生线程安全问题，而跨线程调用sendReceive本身是线程安全的。

## 频繁调用T2Services.getInstance().getClient是否会导致对象数量膨胀，是否需要将client单例化？

重复调用T2Services.getInstance().getClient不会导致client个数的增加，getClient每次调用时，只要name参数相同，返回的都是同一个client实例，因此不需要业务代码来保存client实例。

## T2Services.getInstance().stop方法在什么时候调用？

T2Services.getInstance().stop方法，通常在JAVA程序退出时被调用，它的作用是释放JAVA T2SDK的线程及socket资源。如果程序是在tomcat等webServer中执行的话，在关闭webServer时，资源会被自动释放，因此也可以不调用stop方法。

## JAVA T2SDK是长连接吗？

是socket长连接，JAVA T2SDK连接池会维护socket连接的可用。

## 在网速较慢的场景下(例如ping大于50ms)，如何使用JAVA T2SDK?

在网速较慢且不稳定的情况下，JAVA T2SDK初始化连接和注册需要较长的时间。调用者可以通过设置performance-> connectionWaitTimes来增加注册的等待时间，该参数设置为整型值，等待时间= connectionWaitTimes\*1.5秒，connectionWaitTimes默认为1。

## JAVA T2SDK的调用会阻塞通讯线程么？

当服务吞吐量上不去的时候，用户常常会有这样的疑问，JAVA T2SDK在同步应答时间较长或回调方法时间较长的情况下，会不会一直占用socket，导致性能下降呢？答案是“不会”，在异步调用的情况下，通讯线程会将报文交给专门的处理线程池并立即返回，通讯线程不会阻塞。而在同步调用情况下，当请求被写入socket后，socket使用权即被释放，在请求发出而应答未到达这段时间中，连接是可以被复用的。

## 使用异步发送方法send，配置的回调ID是否可以定义为任意整数？

不行，用户定义的回调ID必须使用正整数，0和负数被系统保留使用。