游戏数据库 TcaplusDB

使用 TcaplusDB SDK

产品文档

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

使用 TcaplusDB SDK 变更历史

最近更新时间：2020-09-08 11:05:55

TcaplusDB SDK 3.34.0.191456

发布时间：2019-04-21

新增：

- 支持创建 pb index 表，并对索引进行批量查询。
- 支持 callback 对象自助释放优化。
- 异步模式收发包优化。
- 日志优化。

修复：

- 多线程 API 对象 init 失败问题修复。
- Get 请求支持默认 Get version。
- API 与 dir 断链反复重复链接错误。
- 修复大 dir 回包无法回调错误。

TcaplusDB SDK 3.32.0.185732

发布时间：2018-11-22

修复：

- 多线程遍历问题修复。

TcaplusDB SDK 3.32.0.174230

发布时间：2018-06-06

修复：

- RegistAllTable 错误修复。

SDK 下载

最近更新时间：2021-11-29 15:54:55

简介

为方便您接入游戏数据库 TcaplusDB，本文为您介绍 TcaplusDB SDK 的下载指引。

变更历史

下载 SDK 之前，请查阅 变更历史。

Linux C++ SDK 下载

表类型	版本	更新时间	操作系统	SDK 下载
Protobuf 表	3.36.0.192960	2020/04/21	Linux x86_64	下载
Protobuf 表	3.46.0.199033	2020/12/29	Linux x86_64	下载
TDR 表	3.46.0.199033	2020/12/29	Linux x86_64	下载

Windows C++ SDK 下载

运行库	版本	更新时间	SDK 下载
MT	3.34.0.191456	2019/04/21	下载
MD	3.34.0.191456	2019/04/21	下载

Windows 平台依赖包下载

> ⚠ 注意：
> 依赖包是分批压缩后上传的，请用户下载所有分片后再解压缩。

运行库	版本	更新时间	下载
MT	2.7.30.177450_Win64Vc14Mt	2019/04/21	下载 part001、下载 part002、下载 part003

运行库	版本	更新时间	下载
MD	2.7.30.177675_Win64Vc14Md	2019/04/21	下载 part001、下载 part002、下载 part003

Protocol Buffers 源码包下载

版本	安装包名	操作系统	下载
v2.6.1	protobuf-2.6.1.tar.gz	Linux x86-64	下载
v3.5.0	protobuf-cpp-3.5.0.tar.gz	Linux x86-64	下载

SDK for C++接口说明

最近更新时间：2021-10-13 11:24:23

简介

TcaplusDB 服务化 API 是应用访问游戏数据库 TcaplusDB 的数据访问入口，是应用存取游戏数据库 TcaplusDB 中业务数据的编程接口。当前 TcaplusDB 主要使用基于 Google Protocol Buffer（Protobuf）做通讯和数据元定义协议。

流程

您在 控制台 开通业务创建完表后，详情页会提供接入ID，访问密码，内网 IPV4 地址，以及 Protobuf 表管理页会提供已经创建的表名称和表格组 ID 等信息。使用 TcaplusDB C++ API，应用可以操作属于此集群下的多个表。

模块

您可以 Protobuf 协议来定义符合 TcaplusDB 规范的表，将表定义的元文件传到 TcaplusDB 控制台中进行创建新表或修改已经存在的表，成功后即可通过 TcaplusDB Protobuf API 进行表数据记录的读写操作，当前 TcaplusDB Protobuf API 支持的操作如下表：

操作	功能描述
GET	根据单个的字段获得单条 Value 信息
BATCHGET	根据多个的字段获得多条 Value 信息
ADD	插入一条数据，如果字段对应记录存在则报错
SET	字段对应记录存在则更新数据，否则插入数据请求
DEL	根据字段删除对应的记录
FIELDINC	指定字段的值（整数型）增加指定值
FIELDSET	更新指定字段（单个或多个）的值
FIELDGET	获取指定字段（单个或多个）的值
INDEXGET	指定索引和字段进行索引查询
TRAVERSE	指定表名进行全表遍历

TcaplusDB SDK 约定

TcaplusDB 需要定义表的主键字段等信息，扩展 tcaplusservice.optionv1.proto 存在于 TcaplusDB 系统中，您只需在自定义表时引用，创建新表或修改已经存在的表时不用上传，系统已经内置。具体内容如下：

```
extend google.protobuf.MessageOptions
{
optional string tcaplus_primary_key = 60000; //定义表的主键
repeated string tcaplus_index = 60001; //定义表的索引
optional string tcaplus_field_cipher_suite = 60002; //定义表的字段使用的加密算法
optional string tcaplus_record_cipher_suite = 60003; //定义表的字段使用的加密算法，暂时未使用
optional string tcaplus_cipher_md5 = 60004; //用于返回 cipher 字段信息摘要
optional string tcaplus_sharding_key = 60005; //Tcaplus sharding key
}

extend google.protobuf.FieldOptions
{
optional uint32 tcaplus_size = 60000; // 字段大小，暂时未使用
optional string tcaplus_desc = 60001; // 字段描述
optional bool tcaplus_crypto = 60002; // 是否加密字段，加密算法由 tcaplus_field_cipher_suite 定义
}
```

完整文件可在目录 release\x86_64\include\tcaplus_pb_api\tcaplusservice.optionv1.proto 中找到。

1. 表名要以字母或下划线开头，不能超过31个字段，不能有除数字，字母，下划线之外的特殊字段。
2. 各字段的名称要以字母或下划线命名，protobuf 已经限制。
3. 主键最多4个字段，必须是 required 类型，打包后长度不能超过1022字节。
4. 字段值打包后不能超过256KB, 同时整个记录打包不能超过256KB。
5. 除了主键字段，至少有一个 普通字段。
6. 表默认是 generic 类型。但对外不显示 generic 类型。
7. 主键字段当前只能是 protobuf 规定的标量类型（Scalar Value Type），不能包括其它复合类型，自定义类型等。

常用接口说明

- int Get(::google::protobuf::Message &msg);

  @brief 根据用户输入 req 中的 index 名称，msg 值，offset 以及 limit，通过索引获取多个记录的值填充到 res 中的 vec 结构中，并返回总记录数以及剩余记录数

```
@param [INOUT] req 用户输入的 req
@param [INOUT] res 用户输入的 res
@retval <0 失败，返回对应的错误码
@retval 0 成功
```

- int BatchGet(std::vector< ::google::protobuf::Message * > &msgs);

```
@brief 根据用户输入 msgs 中的字段值，批量获取 msg 消息的字段值，并填充到 msgs 中
@param [INOUT] msgs 用户输入的字段列表，返回指定字段填到 msgs 中
@retval <0 失败，返回对应的错误码
@retval 0 成功，至少有一个字段查询成功才会返回 0
```

- int Add(::google::protobuf::Message *msg);

```
@brief 根据用户输入 msg 中的字段，插入 msg 数据记录，如果字段存在报错退出
@param [INOUT] msg 用户输入的字段值，以及需要插入的数据记录msg
@retval <0 失败，返回对应的错误码
@retval 0 成功
```

- int Set(const ::google::protobuf::Message &msg);

```
@brief 根据用户输入 msg 中的字段，如果记录存在更新指定记录的值，否则插入指定记录
@param [INOUT] msg 用户输入的字段值，以及需要设置的数据记录 msg
@retval <0 失败，返回对应的错误码
@retval 0 成功
```

- int Del(const ::google::protobuf::Message &msg);

```
@brief 根据用户输入 msg 中的字段值，删除 msg
@param [IN] msg 用户输入的字段值，返回指定字段填到 msg 中
@retval <0 失败，返回对应的错误码
@retval 0 成功，至少有一个字段查询成功才会返回 0
```

- int FieldInc(::google::protobuf::Message &msg, const std::set &dottedpaths);

@brief 根据用户输入 msg 中的字段值和 values 增量值，和 dottedpaths 指定的字段名称，增加 msg 指定字段的值。字段为数值型变量
@param [INOUT] msg 数据记录 msg，包含用户输入的字段值，返回增量字段的结果值更新到 msg 中
@param [IN] dottedpaths 字段名称的点分嵌套字符串集
@retval <0 失败，返回对应的错误码，表示没有任何字段更新
@retval 0 成功，全部字段更新成功

- int FieldGet(const std::set<std::string> &dottedpaths, ::google::protobuf::Message *msg, std::set<std::string> *failedpaths);

@brief 根据用户输入 msg 中的字段值，和 dottedpaths 指定的字段名称，获取指定字段的值，并填充到 msg 中
@param [INOUT] msg 数据记录 msg，包含用户输入的字段值，返回指定字段填到 msg 中
@param [IN] dottedpaths 字段名称的点分嵌套字符串集
@param [OUT] failedpaths 返回查找失败的字段名称的点分嵌套字符串集
@retval <0 失败，返回对应的错误码
@retval 0 成功，至少有一个字段查询成功才会返回0

- int FieldSet(::google::protobuf::Message &msg, const std::set &dottedpaths);

@brief 根据用户输入的 msg 中的 字段值，和 dottedpaths 指定的字段名称，更新指定字段的值。服务端不存在的值会追加进去
@param [IN] msg 用户输入的字段值，返回指定字段填到 msg 中
@param [IN] dottedpaths 字段名称的点分嵌套字符串集
@retval <0 失败，返回对应的错误码，表示没有任何字段更新
@retval 0 成功，全部字段更新成功

- int Get(NS_TCAPLUS_PROTOBUF_API::IndexGetRequest& req, NS_TCAPLUS_PROTOBUF_API::IndexGetResponse *res);

@brief 根据用户输入 req 中的 index 名称，msg 值，offset 以及 limit，通过索引获取多个记录的值填充到 res 中的 vec 结构中，并返回总记录数以及剩余记录数
@param [INOUT] req 用户输入的 req
@param [INOUT] res 用户输入的 res
@retval <0 失败，返回对应的错误码
@retval 0 成功

- int Traverse(::google::protobuf::Message *msg, TcaplusTraverseCallback *cb);

```
@brief 遍历表，消息会填充到 msg 中
@param [INOUT] msg 返回指定字段填到 msg 中
@param [INOUT] cb 回调函数
@retval <0 失败，返回对应的错误码
@retval 0 遍历成功完成
```

规则与约束

Get 操作约束

- 不能查询特定版本号的记录。
- 如果待查询的字段原记录中不存在，则会返回字段默认值。

BatchGet 操作约束

- 批量查询操作必须经由 tcaproxy 进行消息路由处理，tcapsvr 进程不支持批量查询操作。
- 一个批量查询请求返回一个批量查询结果，批量查询的超时时间为10秒。
- 批量查询结果记录数同请求中记录数，查询不存在或者查询失败的记录也会在结果集中存在一条空的记录，因此需要通过 FetchRecord 进行记录获取。
- 批量查询结果集总大小不能超过256K，否则超过大小记录内容会无法返回，会返回空记录内容。
- 批量查询结果集返回的顺序与请求的记录顺序不保证一致。

FieldInc 操作约束

- message 中指定的字段的记录必须是存在。
- dottedpaths 中指定的字段必须是数值类型。
- dottedpaths 所在 set 中的元素个数总数不能起过128个。
- dottedpaths 所在 set 中的每个元素的长度不能超过1023字节。
- dottedpaths 所在 set 中的第个元素代表的字段嵌套不能超过32。

FieldGet 操作约束

- dottedpaths 所在 set 中的元素个数总数不能起过128个。
- dottedpaths 所在 set 中的每个元素的长度不能超过1023字节。
- dottedpaths 所在 set 中的第个元素代表的字段嵌套不能超过32。

FieldSet 操作约束

- dottedpaths 所在 set 中的元素个数总数不能起过128个。

- dottedpaths 所在 set 中的每个元素的长度不能超过1023字节。
- dottedpaths 所在 set 中的第个元素代表的字段嵌套不能超过32。

SDK 源文件目录结构

```
`-- release
`-- x86_64
|-- docs 文档目录
| `-- tcaplus
| `-- readme.txt 本 C++ SDK 的使用描述指引文件
|-- examples 本 C++ SDK 使用的样例目录，分同步和异步两大部分，可以直接修改使用
|-- include 本 C++ SDK 的头文件目录
| `-- tcaplus_pb_api TcaplusDB PB API 头文件夹
| |-- cipher_suite_base.h 数据加密码算法套件基类
| |-- default_aes_cipher_suite.h 数据加密码算法套件默认实现类
| |-- tcaplus_async_pb_api.h 异步模式类的头文件，使用异步模式当中的 TcaplusAsyncPbApi 类
| |-- tcaplus_coroutine_pb_api.h 协程模式类的头文件，使用协程模式当中的 TcaplusCoroutinePbApi 类
| |-- tcaplus_error_code.h 错误码头文件，所有涉及的错误码均可以在这里找到对应的定义及描述
| |-- tcaplus_protobuf_api.h API 汇总头文件，包含了其它头文件，只需引用这一个文件方便开发
| |-- tcaplus_protobuf_define.h 基础结构和宏定义头文件，包括 ClientOptions 结构和 MESSAGE_OPTI
ON_* 宏定义
| |-- tcaplusservice.optionv1.pb.h Tcaplus 表公共定义的 Protobuf 头文件
| `-- tcaplusservice.optionv1.proto Tcaplus 表公共定义的 proto 源文件, 自定义表时需包含此文件
|-- lib 本 C++ SDK 的库文件目录
| `-- libtcaplusprotobufapi.a 本 C++ SDK 的库文件, 程序最终链接库需包含
`-- version 本 C++ SDK 的版本记录文件
```

TcaplusDB 错误码

最近更新时间：2021-08-16 16:49:12

编号	错误码	描述
1	-1792	表处于只读模式，请检查 RCU，WCU 或容量是否超出用户设定的阈值
3	261	该记录不存在
4	-525	batchget 操作请求超时
5	-781	batchget，getbypartkey 或 listgetall 方法返回超过响应数据上限（1MB）
6	-1037	系统繁忙
7	-1293	记录已存在，请不要重复插入
8	-1549	访问的表字段不存在
9	-2061	表字段类型错误
10	-3085	SetFieldName 操作指定了错误的字段
11	-3341	字段值大小超过其定义类型的限制
12	-4109	list 数据类型元素下标超过范围
14	-4621	请求缺少主键字段或索引字段
15	-6157	list 表元素个数超过定义范围，请设置元素淘汰
16	-6925	result_flag 设置错误，请参考 SDK 中 result_flag 说明
17	-7949	请检查乐观锁，请求记录版本号与实际记录版本号不一致
18	-11277	操作表的方法不存在
19	-16141	Protobuf 表 GetRecord 操作失败
20	-16397	Protobuf 表非主键字段值超过限定大小（256KB）
21	-16653	Protobuf 表 FieldSetRecord 操作失败
22	-16909	Protobuf 表 FieldIncRecord 操作失败
23	-275	主键字段个数超过限制，Generic 表限制数为4，List 表限制数为3

编号	错误码	描述
24	-531	非主键字段个数超过限制，Generic 表限制数为128，List 表限制数为127
25	-787	字段名称大小超过限制（32B）
26	-1043	字段值指超过限制（256KB）
27	-1555	字段值的数据类型与其定义类型不匹配
28	-5395	请求中缺少主键
29	-9235	index 不存在
30	-12307	请求发送失败，网络过载
31	-12819	表不存在
32	-13843	后台网络异常，请求无法发送成功
33	-14099	插入的记录超过大小限制（1MB）

SDK 安装
Linux 安装手册

最近更新时间：2021-07-07 17:21:32

操作场景

本文档指导您在 Linux 系统环境下安装和使用 TcaplusDB PB 表。

运行环境

Linux 2.6、Suse 12 64 位。

前提条件

安装和使用 TcaplusDB PB 前需安装 Protobuf，TcaplusDB 当前支持 Protocol Buffers 2.6.1 版本以及3.5.0版本。
Protobuf 是 Google 推出的一种混合语言数据标准，是一种轻便的结构化数据存储格式。TcaplusDB 系统支持使用 Protobuf 格式定义文件（.proto）定义数据表。使用 TcaplusDB PB API 之前，需要在开发服务器上安装 Protobuf，推荐使用源代码进行 Protobuf 安装，安装方法如下：

1. 准备云服务器环境。
   首先需要准备安装了 CentOS6- x86_64 或 CentOS7-x86_64 版本操作系统的服务器。为了编译构建 Protobuf，需要安装以下软件，安装方法请自行查询：
   - autoconf
   - automake
   - libtool
   - curl (used to download gmock)
   - make
   - g++
   - unzip
2. 下载 Protobuf 源码安装包，请参见 SDK 下载。
3. 请根据实际需求安装指定版本的 ProtoBuf，下文以 protobuf 2.6.1 版本为例。
4. 执行以下命令，解压源码安装包，并进入源码根目录下。

```
tar -xzvf protobuf-2.6.1.tar.gz
cd ./protobuf-2.6.1
```

5. Configure 并指定安装路径前缀，推荐安装到路径/usr/local/protobuf下。

```
./configure --prefix=/usr/local/protobuf
```

6. 编译并安装。

```
make
make check
make install
```

7. 测试，使用 protoc 命令查看是否安装成功。如下所示，即表示安装成功。

```
# protoc --version
libprotoc 2.6.1
```

操作步骤

步骤1：安装 TcaplusDB SDK

首先将 TcaplusDB SDK 下载至开发服务器，然后执行命令将文件安装至指定安装目录。

```
tar -xzf <安装包路径> -C <安装目录>，
```

步骤2：校验

安装完成后，根目录结构如下表所示：

目录及文件	说明
include/tcaplus_service/	TCAPLUS 服务化 API 头文件
lib/libtcaplusserviceapi.a	TCAPLUS 服务化 API 库文件
include/tcaplus_service/protobuf/	Protobuf API 头文件
lib/libtcaplusprotobufapi.a	TCAPLUS Protobuf API 库文件
examples/tcaplus/ProtoBuf	TCAPLUS Protobuf API 应用示例

步骤3：运行 example 访问 TcaplusDB

GameSvr 游戏服务器中对应数据访问逻辑的开发，可以参考 example 中的各接口示例。

1. 解压 TcaplusDB PB API 发布包。

```
tar -xzvf TcaplusPbApi3.36.0.152096.x86_64_release_20170712.tar.gz
```

2. 配置 TcaplusDB 系统连接信息。

i. 在命令行输入如下代码进入目录：

```
cd TcaplusPbApi3.36.0.152096.x86_64_release_20170712/release/x86_64/examples/tcaplus/C++_common_for_pb2
```

ii. 在命令行输入 vi common.h 修改 common.h 头文件，根据业务情况修改如下图片内容。

DIR_URL_ARRAY：集群访问 IP 地址和端口号。

DIR_URL_COUNT：固定值，保持为1即可。

TABLE_NAME：需要访问的目标表。

APP_ID：需要访问的接入 ID。

ZONE_ID：填写需要访问的表格组 ID。

SIGNATURE：集群访问密码。

```
/**********************测试例子前需要用户手动修改的地方BEGIN**************************************
// 目标业务的tcapdir地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
        "tcp://10.191.***.99:9999"
        "tcp://10.191.***.88:9999"
};

// 目标业务的tcapdir 地址个数
static const int32_t DIR_URL_COUNT = 1;
// 目标业务的表名
static const char * TABLE_NAME = "tb_online";
// 目标业务的App ID
static const int32_t APP_ID = 3;
// 目标业务的Zone ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "**********";
/**********************测试例子前需要用户手动修改的地方END**************************************/
```

3. 修改环境配置文件。

在TcaplusPbApi3.36.0.152096.x86_64_release_20170712/release/x86_64/examples/tcaplus目录下有分别通过异步方式以及协程方式调用 API 的示例，此处以协程方式调用 Set 接口设置数据为例：

在命令行输入如下代码进入目录：

```
cd TcaplusPbApi3.18.0.152096.x86_64_release_20170712/release/x86_64/examples/tcaplus/C++_pb2_coroutine_simpletable/SingleOperation/set
```

协程方式 Set 示例的所有代码都在该目录中。修改 envcfg.env 文件，将 PROTOBUF_HOME 环境变量设置为本机 protobuf 的安装路径（--prefix指定），并将 TCAPLUS_HOME 环境变量设置为 Tcaplus PB API 包下 release/x86_64 目录的绝对路径，如下图：

```
export PROTOBUF_HOME=/usr/loacl/protobuf;
export TCAPLUS_HOME=/my_some_dir/TcaplusPbApi3.18.0.123456.x86_64_release_20161124/release/x86_64/;
```

4. 设置环境变量，在代码目录下执行如下命令：

```
source envcfg.env
bash conv.sh
```

5. 编译二进制程序。

执行make命令编译 example 二进制，编译成功生成 mytest 可执行文件。

```
total 32
-rw-r--r-- 1 1002 users  416 Jul 12 16:40 conv.sh
-rw-r--r-- 1 1002 users  232 Aug 15 16:04 envcfg.env
-rw-r--r-- 1 1002 users 1766 Jul 12 16:40 main.cpp
-rw-r--r-- 1 1002 users  678 Jul 12 16:40 Makefile
drwxr-xr-x 2 root root  4096 Aug 15 16:05 mytest
-rw-r--r-- 1 1002 users 1047 Jul 12 16:40 readme.txt
-rw-r--r-- 1 1002 users  707 Jul 12 16:40 table_test.proto
-rw-r--r-- 1 1002 users  662 Jul 12 16:40 tlogconf.xml
```

6. 执行二进制程序。

在命令行输入./mytest ，执行二进制程序。执行结果将在命令行标准输出中显示，若遇到错误，请查看代码目录下的 tcaplus_pb.log 日志文件。

Windows 安装手册

最近更新时间：2022-03-02 10:26:23

操作场景

本文为您介绍如何在 Windows(x64) 系统上操作 Tcaplus Protobuf API。

运行环境

- 操作系统：Microsoft Windows x86_64
- 编译环境：Microsoft Visual Studio 2015 (VC14.0)

操作步骤

下载软件包

1. 下载依赖包和 Tcaplus Protobuf API 软件包，参见 SDK 下载。
2. 解压缩，以下是软件包的结构。

```
Tcaplus_PbAPI_3.32.0.171987_Win64Vc14MT_Release_20180413
|-- cfg # 配置目录
|-- docs # 文档目录
| `-- tcaplus
|-- include # 依赖头文件目录
| `-- tcaplus_pb_api
|-- lib # 库目录
| |-- Debug
| `-- Release
|-- examples # 示例目录
`-- tcaplus
|-- C++_common_for_pb2 # 示例公共头文件目录
|-- C++_pb2_asyncmode_simpletable # 异步模式的pb简单表示例
|-- C++_pb2_coroutine_simpletable # 协程模式的pb简单表示例
```

准备环境

1. 请确保已经在 TcaplusDB 开通了游戏业务，并且已经获取到对应的 App 信息（例如 AppId、ZoneId、AppKey）。

2. 解压缩依赖包并安装，本文以 TSF4G_BASE-2.7.28.164975_Win64Vc14Mt_Release.zip 依赖包为例，实际以 Windows 平台依赖包下载 提供的依赖包为准。
   假设安装的根路径是 D:\Tencent\tsf4gMT，相关文件将会被安装到 D:\Tencent\tsf4gMT\win64vc14MT 路径下。
3. 编译并安装 Porotbuf-3.5.1。
   - 源码地址
   - 编译安装指南
   - 假设安装路 径为 D:\protobuf-3.5.1
4. 编译并安装 OpenSSL-1.1.0f。
   - 源代码地址
   - 编译安装指南
   - 假设安装路径为 D:\openssl-1.1.0f
5. 设置环境变量。
   - TSF4G_HOME="D:\Tencent\tsf4gMT"
   - PROTOBUF_HOME="D:\protobuf-3.5.1"
   - OPENSSL_HOME="D:\openssl-1.1.0f"

构建

1. 解压缩 Tcaplus Pb API 安装包。
2. 在 examples/tcaplus/C++_common_for_pb2/common.h 文件中设置 App 信息。
   - Tcapdir 接入点地址列表 – DIR_URL_ARRAY
   - Tcapdir 接入点地址个数 – DIR_URL_COUNT
   - 用户表名，使用之前用户需要预先使用示例目录中的 table_test.xml 文件创建表 – TABLE_NAME
   - 用户业务 ID – APP_ID
   - 用户业务区服 ID – ZONE_ID
   - 用户业务密码 – SIGNATURE

```
//examples/tcaplus/C++_common_for_pb2/common.h
/*****************用户自定义**************************/
// Tcapdir 接入点地址列表
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.xx.xx.21:9999"
};
// Tcapdir 接入点地址个数
static const int32_t DIR_URL_COUNT = 1;
// 用户表名
static const char * TABLE_NAME = "tb_online";
```

```
// 用户业务 ID
static const int32_t APP_ID = 4;
// 用户业务区服 ID
static const int32_t ZONE_ID = 1;
// 用户业务密码
static const char * SIGNATURE = "8e24269ba91fxxxa7e89b1cbb77368e";
/******************用户自定义*****************************/
```

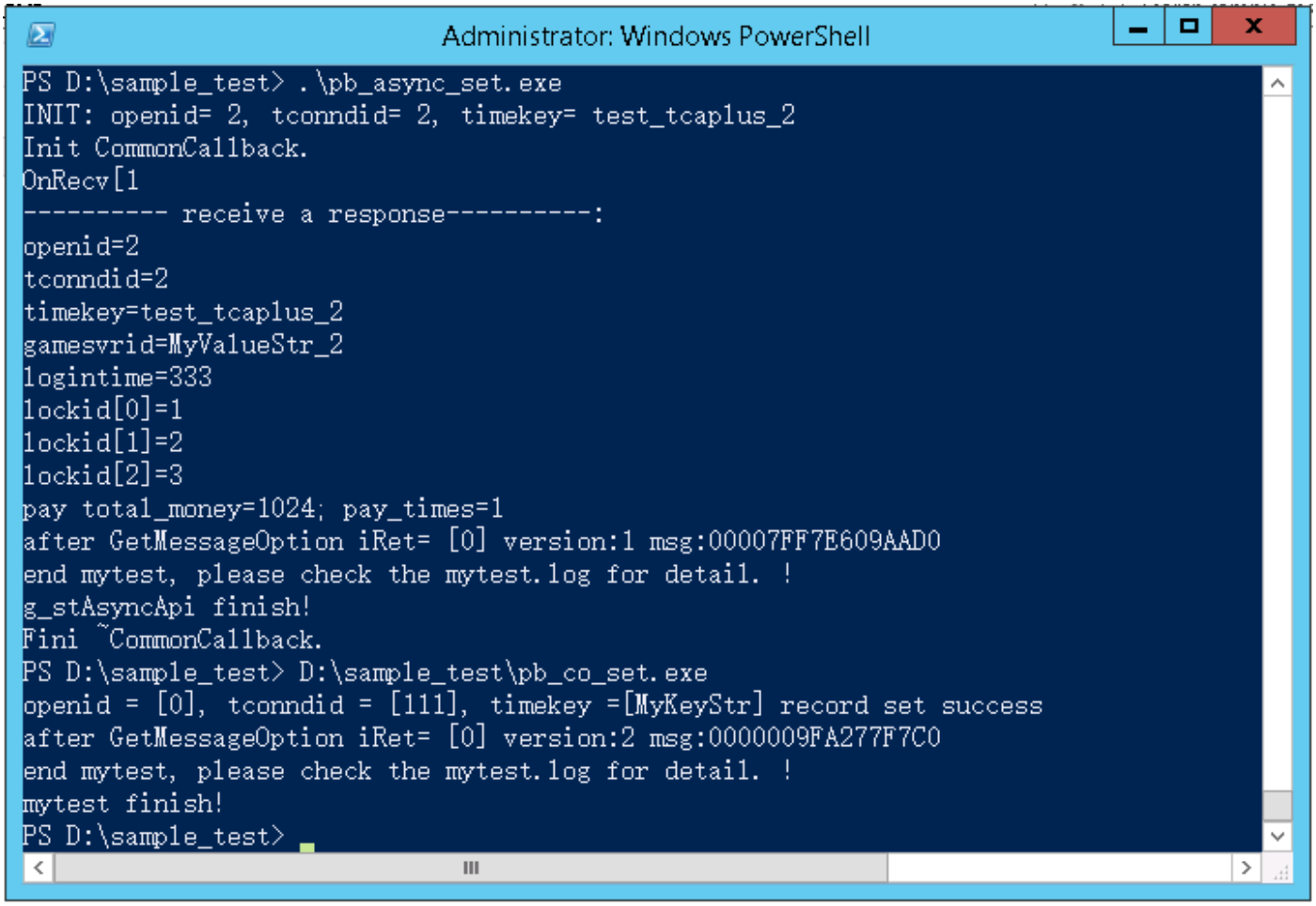
3. 以 examples\tcaplus\C++_pb2_coroutine_simpletable\SingleOperation\set 为例。

```
set/
|-- main.cpp # 示例主函数代码
|-- readme.txt
|-- table_test.proto # T 表定义 proto 文件，表需要预先创建
|-- pb_co_set.sln # 项目 VisualSudio 解决方案文件
|-- pb_co_set.vcxproj # 项目 VisualSudio 工程文件
|-- proto_generate.cmd # 编译 proto 文件脚本
`-- tlogconf.xml
```

   i. 首先，确认已经使用 table_test.proto 在目标 App 中创建表成功。
   ii. 执行 proto_generate.cmd 脚本，在当前路径下生成依赖文件。
      - table_test.pb.cc
      - table_test.pb.h
   iii. 在 Microsoft Visual Studio 2015 中打开项目文件 pb_co_set.sln。
   iv. 生成解决方案。
   v. 如果没有错误产生，在 examples\tcaplus\C++_pb2_coroutine_simpletable\SingleOperation\set/x64 路径下将会生成可执行文件 pb_co_set.exe。

测试

1. 拷贝 pb_co_set.exe、tlogconf.xml 两个文件到同一目录下。
2. 切换 administrator 身份并使用 cmd.exe 或 powershell.exe 运行可执行文件 pb_co_set.exe。
3. 检查输出。
4. 如果需要了解运行详细信息，请查看日志文件 tcaplus_pb.log。
5. 如果需要运行 *_crypto 示例，请确保 libcrypto-1_1-x64.dll 文件在系统 Path 路径下，该文件在 openssl 的编译目录下。



Tcaplus Pb API 命令列表

Tcaplus Pb API 支持多种类型操作，支持异步和协程模式，用户可以在示例中找到对应的用法。以下是 Tcaplus Pb API 命令列表：

命令	描述
SET	通过指定一条记录所有主键设置一条记录，如果记录存在执行覆盖操作，否则执行插入操作。
GET	从一个 Tcaplus Pb 表中通过指定一条记录所有主键查询一条记录，如果数据记录不存在，将会返回错误。
ADD	通过指定一条记录所有主键插入一条记录，如果记录存在返回错误。
DELETE	通过指定一条记录的所有主键删除此记录，如果数据不存在则返回错误。
BATCHGET	从一个 Tcaplus Pb 表中通过指定多组主键查询多条记录。
TRAVERSE	遍历一个 Tcaplus Pb 表，将返回多条记录。

命令	描述
FIELDGET	从一个 Tcaplus Pb 表中通过指定一条记录所有主键查询一条记录。本操作只查询和传输用户指定的字段的值，减少网络传输流量。如果数据记录不存在，将会返回错误。
FIELDSET	通过指定一条记录的所有主键修改指定字段，只传输指定字段的值。减轻网络流量。如果数据记录存在，将执行更新操作，否则将会返回错误。
FIELDINC	通过指定一条记录的所有主键对指定的字段进行自增操作，此命令字仅支持 int32，int64，uint32 和 uint64 类型字段。特性与 FIELDSET 类似。
GETBYPARTKEY	通过指定部分主键，查询符合条件的多条数据，主键集合必须在建表的时候创建了索引，否则会返回错误。

PB 表 C++SDK 操作方法
写入数据

最近更新时间：2021-08-10 14:33:45

前提

成功创建：集群，表格组，tb_online 表

tb_online 表描述文件 table_test.proto 如下：（ tcaplusservice.optionv1.proto 为依赖表 ）

```
syntax = "proto2";
package myTcaplusTable;
import "tcaplusservice.optionv1.proto";
message tb_online {
option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey";
required int32 openid = 1; //QQ Uin
required int32 tconndid = 2;
required string timekey = 3;
required string gamesvrid = 4;
optional int32 logintime = 5 [default = 1];
repeated int64 lockid = 6 [packed = true]; //repeated 类型字段使用 packed 关键字修饰
optional pay_info pay = 7;
message pay_info {
optional uint64 total_money = 1;
optional uint64 pay_times = 2;
}
}
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
```

```
// 目标集群的地址个数
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 tb_online
static const char * TABLE_NAME = "tb_online";
```

步骤2：初始化 TcaplusPB 客户端

```
//Tcaplus PB API 客户端
TcaplusAsyncPbApi g_stAsyncApi;
int32_t InitAsyncPbApi()
{
//PB API 配置
ClientOptions cfg;
cfg.app_id = APP_ID;
cfg.zones.push_back(ZONE_ID);
strcpy(cfg.signature, SIGNATURE);
for (int32_t i = 0; i < DIR_URL_COUNT; i++)
{
cfg.dirs.push_back(DIR_URL_ARRAY[i]);
}
//访问的 PB 表
cfg.tables.push_back(TABLE_NAME);
//日志配置
strncpy(cfg.log_cfg, "tlogconf.xml", sizeof(cfg.log_cfg));
//初始化连接超时时间5s
cfg.timeout = 5000;

//初始化连接
int32_t iRet = g_stAsyncApi.Init(cfg);
if (0 != iRet)
{
```

```
cout << "ERROR: g_stAsyncApi.Init failed, log cfg: " << cfg.log_cfg << ", iRet: " << iRet << "."
<< endl;
return iRet;
}
return iRet;
}
```

步骤3：定义异步回调

```
//收到响应消息计数
uint32_t g_dwTotalRevNum = 0;
class CommonCallback : public TcaplusPbCallback
{
public:
CommonCallback()
{
cout << "Init CommonCallback." << endl;
}

~CommonCallback()
{
cout << "Fini ~CommonCallback." << endl;
}
//收到响应消息的回调，msgs 为获取的记录的数组
int OnRecv(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnRecv[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}
```

```
cout << "---------- receive a response----------:" << endl;
cout << "openid=" << t->openid() << endl;
cout << "tconndid=" << t->tconndid() << endl;
cout << "timekey=" << t->timekey() << endl;
cout << "gamesvrid=" << t->gamesvrid() << endl;
cout << "logintime=" << t->logintime() << endl;
for (int32_t i = 0; i < t->lockid_size();i++)
{
cout << "lockid[" << i << "]=" << t->lockid(i) << endl;
}
tb_online_pay_info pay = t->pay();
cout << "pay total_money=" << pay.total_money() << "; pay_times=" << pay.pay_times() <<
endl;

//获取记录的版本号
std::string version;
int iRet = g_stAsyncApi.GetMessageOption(*t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION
_DATA_VERSION, &version);
cout << "after GetMessageOption iRet= [" << iRet << "] version:" << version.c_str() << " ms
g:" << t << endl;
}

return 0;
}

//收到错误响应消息的回调
int OnError(const std::vector< ::google::protobuf::Message *> &msgs, int errorcode)
{
cout << "OnError[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}
```

```
if (TcapErrCode::TXHDB_ERR_RECORD_NOT_EXIST == errorcode)
{
cout << "ERROR: openid= " << t->openid() << ", tconndid= " << t->tconndid() << ", timekey = " << t->timekey() << ", record not exists" << endl;
}
cout << "ERROR: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] failed:%d" << errorcode << endl;
}

return 0;
}

//消息超时的回调
int OnTimeout(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnTimeout[" << msgs.size() << endl;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "TIMEOUT: msgs[" << idx << "] not tb_online type!" << endl;
return -1;
}

cout << "TIMEOUT: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] timeout" << endl;
}

return 0;
}

int OnFinish(const NS_TCAPLUS_PROTOBUF_API::MsgParam &param)
{
cout << "OnFinish: " << param.m_nOperation << " req: " << param.m_vecMsgs.size()<< endl;
return 0;
```

```
}
};
```

步骤4：发送 add 请求

```
int SendAddRequest()
{
static tb_online t;
t.set_openid(2);
t.set_tconndid(2);
t.set_timekey("test_tcaplus_2");
t.set_gamesvrid("MyValueStr_2");
t.set_logintime(333);
for (int32_t i = 0; i < TOTAL_V3_ARRAY_NUM;i++)
{
t.add_lockid(i+1);
}

tb_online_pay_info* pay = new tb_online_pay_info();
pay->set_total_money(1024);
pay->set_pay_times(1);
t.set_allocated_pay(pay);

cout << "INIT: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey= " <
< t.timekey() << endl;

std::string version = "-1";
g_stAsyncApi.SetMessageOption(t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION_DATA_VER
SION, version);

static CommonCallback cb;
int32_t iRet = g_stAsyncApi.Add(&t, &cb);
if (iRet != TcapErrCode::GEN_ERR_SUC)
{
cout << "ERROR: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey= "
<< t.timekey() << ", Set Error iRet = " << iRet << endl;
return -1;
```

```
}
return 0;
}
```

示例

main.cpp 文件

```
int main(void) {

//初始化 API 客户端
int ret = InitAsyncPbApi();
if ( ret != 0)
{
printf("InitAsyncPbApi failed\n");
return -1;
}

//发送请求
ret = SendAddRequest();
if (0 != ret)
{
printf("SendAddRequest failed\n");
return -1;
}

//接收响应
do
{
// 更新，接收回包
g_stAsyncApi.UpdateNetwork();
usleep(1000 * 10);
} while (g_dwTotalRevNum != 1);
return 0;
}
```

读取数据

最近更新时间：2021-08-10 14:34:14

前提

成功创建：集群，表格组，tb_online 表

tb_online 表描述文件 table_test.proto 如下：（tcaplusservice.optionv1.proto 为依赖表）

```
syntax = "proto2";
package myTcaplusTable;
import "tcaplusservice.optionv1.proto";
message tb_online {
option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey";
required int32 openid = 1; //QQ Uin
required int32 tconndid = 2;
required string timekey = 3;
required string gamesvrid = 4;
optional int32 logintime = 5 [default = 1];
repeated int64 lockid = 6 [packed = true]; //repeated 类型字段使用 packed 关键字修饰
optional pay_info pay = 7;
message pay_info {
optional uint64 total_money = 1;
optional uint64 pay_times = 2;
}
}
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
// 目标集群的地址个数
```

```
static const int32_t DIR_URL_COUNT = 2;

// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 tb_online
static const char * TABLE_NAME = "tb_online";
```

步骤2：初始化 TcaplusPB 客户端

```
//Tcaplus PB API 客户端
TcaplusAsyncPbApi g_stAsyncApi;
int32_t InitAsyncPbApi()
{
//PB API 配置
ClientOptions cfg;
cfg.app_id = APP_ID;
cfg.zones.push_back(ZONE_ID);
strcpy(cfg.signature, SIGNATURE);
for (int32_t i = 0; i < DIR_URL_COUNT; i++)
{
cfg.dirs.push_back(DIR_URL_ARRAY[i]);
}
//访问的 PB 表
cfg.tables.push_back(TABLE_NAME);
//日志配置
strncpy(cfg.log_cfg, "tlogconf.xml", sizeof(cfg.log_cfg));
//初始化连接超时时间5s
cfg.timeout = 5000;

//初始化连接
int32_t iRet = g_stAsyncApi.Init(cfg);
if (0 != iRet)
{
```

```
cout << "ERROR: g_stAsyncApi.Init failed, log cfg: " << cfg.log_cfg << ", iRet: " << iRet << "."
<< endl;
return iRet;
}
return iRet;
}
```

步骤3：定义异步回调

```
//收到响应消息计数
uint32_t g_dwTotalRevNum = 0;
class CommonCallback : public TcaplusPbCallback
{
public:
CommonCallback()
{
cout << "Init CommonCallback." << endl;
}

~CommonCallback()
{
cout << "Fini ~CommonCallback." << endl;
}
//收到响应消息的回调，msgs 为获取的记录的数组
int OnRecv(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnRecv[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}
```

```
cout << "---------- receive a response----------:" << endl;
cout << "openid=" << t->openid() << endl;
cout << "tconndid=" << t->tconndid() << endl;
cout << "timekey=" << t->timekey() << endl;
cout << "gamesvrid=" << t->gamesvrid() << endl;
cout << "logintime=" << t->logintime() << endl;
for (int32_t i = 0; i < t->lockid_size();i++)
{
cout << "lockid[" << i << "]=" << t->lockid(i) << endl;
}
tb_online_pay_info pay = t->pay();
cout << "pay total_money=" << pay.total_money() << "; pay_times=" << pay.pay_times() <<
endl;

//获取记录的版本号
std::string version;
int iRet = g_stAsyncApi.GetMessageOption(*t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION
_DATA_VERSION, &version);
cout << "after GetMessageOption iRet= [" << iRet << "] version:" << version.c_str() << " ms
g:" << t << endl;
}

return 0;
}

//收到错误响应消息的回调
int OnError(const std::vector< ::google::protobuf::Message *> &msgs, int errorcode)
{
cout << "OnError[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}
```

```
if (TcapErrCode::TXHDB_ERR_RECORD_NOT_EXIST == errorcode)
{
cout << "ERROR: openid= " << t->openid() << ", tconndid= " << t->tconndid() << ", timekey = " << t->timekey() << ", record not exists" << endl;
}
cout << "ERROR: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] failed:%d" << errorcode << endl;
}

return 0;
}

//消息超时的回调
int OnTimeout(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnTimeout[" << msgs.size() << endl;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "TIMEOUT: msgs[" << idx << "] not tb_online type!" << endl;
return -1;
}

cout << "TIMEOUT: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] timeout" << endl;
}

return 0;
}

int OnFinish(const NS_TCAPLUS_PROTOBUF_API::MsgParam &param)
{
cout << "OnFinish: " << param.m_nOperation << " req: " << param.m_vecMsgs.size()<< endl;
return 0;
```

```
}
};
```

步骤4：发送 get 请求

```
int SendGetRequest()
{
static tb_online t;
t.set_openid(1);
t.set_tconndid(1);
t.set_timekey("test_tcaplus");

cout << "INIT: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey=" <<
t.timekey() << endl;

static CommonCallback cb;
int32_t iRet = g_stAsyncApi.Get(&t, &cb);
if (iRet != TcapErrCode::GEN_ERR_SUC)
{
cout << "ERROR: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey= "
<< t.timekey() << ", Get Error iRet = " << iRet << endl;
return -1;
}
return 0;
}
```

示例

```
int main(void) {

//初始化 API 客户端
int ret = InitAsyncPbApi();
if ( ret != 0)
{
printf("InitAsyncPbApi failed\n");
return -1;
```

```
}

//发送请求
ret = SendGetRequest();
if (0 != ret)
{
printf("SendSetRequest failed\n");
return -1;
}

//接收响应
do
{
// 更新，接收回包
g_stAsyncApi.UpdateNetwork();
usleep(1000 * 10);
} while (g_dwTotalRevNum != 1);
return 0;
}
```

更新数据

最近更新时间：2021-08-10 14:34:47

前提

成功创建：集群，表格组，tb_online 表

tb_online 表描述文件 table_test.proto 如下：（tcaplusservice.optionv1.proto 为依赖表）

```
syntax = "proto2";
package myTcaplusTable;
import "tcaplusservice.optionv1.proto";
message tb_online {
option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey";
required int32 openid = 1; //QQ Uin
required int32 tconndid = 2;
required string timekey = 3;
required string gamesvrid = 4;
optional int32 logintime = 5 [default = 1];
repeated int64 lockid = 6 [packed = true]; //repeated 类型字段使用 packed 关键字修饰
optional pay_info pay = 7;
message pay_info {
optional uint64 total_money = 1;
optional uint64 pay_times = 2;
}
}
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
// 目标集群的地址个数
```

```
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 tb_online
static const char * TABLE_NAME = "tb_online";
```

步骤2：初始化 TcaplusPB客户端

```
//Tcaplus PB API 客户端
TcaplusAsyncPbApi g_stAsyncApi;
int32_t InitAsyncPbApi()
{
//PB API 配置
ClientOptions cfg;
cfg.app_id = APP_ID;
cfg.zones.push_back(ZONE_ID);
strcpy(cfg.signature, SIGNATURE);
for (int32_t i = 0; i < DIR_URL_COUNT; i++)
{
cfg.dirs.push_back(DIR_URL_ARRAY[i]);
}
//访问的 PB 表
cfg.tables.push_back(TABLE_NAME);
//日志配置
strncpy(cfg.log_cfg, "tlogconf.xml", sizeof(cfg.log_cfg));
//初始化连接超时时间5s
cfg.timeout = 5000;

//初始化连接
int32_t iRet = g_stAsyncApi.Init(cfg);
if (0 != iRet)
{
cout << "ERROR: g_stAsyncApi.Init failed, log cfg: " << cfg.log_cfg << ", iRet: " << iRet << "."
```

```
<< endl;
return iRet;
}
return iRet;
}
```

步骤3：定义异步回调

```
//收到响应消息计数
uint32_t g_dwTotalRevNum = 0;
class CommonCallback : public TcaplusPbCallback
{
public:
CommonCallback()
{
cout << "Init CommonCallback." << endl;
}

~CommonCallback()
{
cout << "Fini ~CommonCallback." << endl;
}
//收到响应消息的回调，msgs 为获取的记录的数组
int OnRecv(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnRecv[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}

cout << "---------- receive a response----------:" << endl;
```

```
cout << "openid=" << t->openid() << endl;
cout << "tconndid=" << t->tconndid() << endl;
cout << "timekey=" << t->timekey() << endl;
cout << "gamesvrid=" << t->gamesvrid() << endl;
cout << "logintime=" << t->logintime() << endl;
for (int32_t i = 0; i < t->lockid_size();i++)
{
cout << "lockid[" << i << "]=" << t->lockid(i) << endl;
}
tb_online_pay_info pay = t->pay();
cout << "pay total_money=" << pay.total_money() << "; pay_times=" << pay.pay_times() <<
endl;

//获取记录的版本号
std::string version;
int iRet = g_stAsyncApi.GetMessageOption(*t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION
_DATA_VERSION, &version);
cout << "after GetMessageOption iRet= [" << iRet << "] version:" << version.c_str() << " ms
g:" << t << endl;
}

return 0;
}

//收到错误响应消息的回调
int OnError(const std::vector< ::google::protobuf::Message *> &msgs, int errorcode)
{
cout << "OnError[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}
```

```
if (TcapErrCode::TXHDB_ERR_RECORD_NOT_EXIST == errorcode)
{
cout << "ERROR: openid= " << t->openid() << ", tconndid= " << t->tconndid() << ", timekey = " << t->timekey() << ", record not exists" << endl;
}
cout << "ERROR: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] failed:%d" << errorcode << endl;
}

return 0;
}

//消息超时的回调
int OnTimeout(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnTimeout[" << msgs.size() << endl;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "TIMEOUT: msgs[" << idx << "] not tb_online type!" << endl;
return -1;
}

cout << "TIMEOUT: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] timeout" << endl;
}

return 0;
}

int OnFinish(const NS_TCAPLUS_PROTOBUF_API::MsgParam &param)
{
cout << "OnFinish: " << param.m_nOperation << " req: " << param.m_vecMsgs.size()<< endl;
return 0;
```

```
}
};
```

步骤4：发送 set 请求

```
int SendSetRequest()
{
//定义 PB 记录的结构体并赋值，结构体通过 proto 转换
static tb_online t;
t.set_openid(2);
t.set_tconndid(2);
t.set_timekey("test_tcaplus_2");
t.set_gamesvrid("MyValueStr_2");
t.set_logintime(333);
for (int32_t i = 0; i < 3;i++)
{
t.add_lockid(i+1);
}

tb_online_pay_info* pay = new tb_online_pay_info();
pay->set_total_money(1024);
pay->set_pay_times(1);
t.set_allocated_pay(pay);

cout << "INIT: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey= " <
< t.timekey() << endl;

//设置版本号，用于乐观锁
std::string version = "-1"; // std::string version="1"; -1表示不比较，1表示服务器数据version为1
g_stAsyncApi.SetMessageOption(t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION_DATA_VER
SION, version);

//传入定义的回调，收到响应消息后自动调用回调函数
static CommonCallback cb;
int32_t iRet = g_stAsyncApi.Set(&t, &cb);
if (iRet != TcapErrCode::GEN_ERR_SUC)
{
```

```
cout << "ERROR: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey= "
<< t.timekey() << ", Set Error iRet = " << iRet << endl;
return -1;
}
return 0;
}
```

示例

```
int main(void) {

//初始化 API 客户端
int ret = InitAsyncPbApi();
if ( ret != 0)
{
printf("InitAsyncPbApi failed\n");
return -1;
}

//发送请求
ret = SendSetRequest();
if (0 != ret)
{
printf("SendSetRequest failed\n");
return -1;
}

//接收响应
do
{
// 更新，接收回包
g_stAsyncApi.UpdateNetwork();
usleep(1000 * 10);
} while (g_dwTotalRevNum != 1);
return 0;
}
```

删除数据

最近更新时间：2021-08-10 14:35:17

前提

成功创建：集群，表格组，tb_online表

tb_online 表描述文件 table_test.proto 如下：（tcaplusservice.optionv1.proto 为依赖表）

```
syntax = "proto2";
package myTcaplusTable;
import "tcaplusservice.optionv1.proto";
message tb_online {
option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey";
required int32 openid = 1; //QQ Uin
required int32 tconndid = 2;
required string timekey = 3;
required string gamesvrid = 4;
optional int32 logintime = 5 [default = 1];
repeated int64 lockid = 6 [packed = true]; //repeated 类型字段使用 packed 关键字修饰
optional pay_info pay = 7;
message pay_info {
optional uint64 total_money = 1;
optional uint64 pay_times = 2;
}
}
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
// 目标集群的地址个数
```

```
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 tb_online
static const char * TABLE_NAME = "tb_online";
```

步骤2：初始化 TcaplusPB 客户端

```
//Tcaplus PB API 客户端
TcaplusAsyncPbApi g_stAsyncApi;
int32_t InitAsyncPbApi()
{
//PB API 配置
ClientOptions cfg;
cfg.app_id = APP_ID;
cfg.zones.push_back(ZONE_ID);
strcpy(cfg.signature, SIGNATURE);
for (int32_t i = 0; i < DIR_URL_COUNT; i++)
{
cfg.dirs.push_back(DIR_URL_ARRAY[i]);
}
//访问的 PB 表
cfg.tables.push_back(TABLE_NAME);
//日志配置
strncpy(cfg.log_cfg, "tlogconf.xml", sizeof(cfg.log_cfg));
//初始化连接超时时间5s
cfg.timeout = 5000;

//初始化连接
int32_t iRet = g_stAsyncApi.Init(cfg);
if (0 != iRet)
{
cout << "ERROR: g_stAsyncApi.Init failed, log cfg: " << cfg.log_cfg << ", iRet: " << iRet << "."
```

```
<< endl;
return iRet;
}
return iRet;
}
```

步骤3：定义异步回调

```
//收到响应消息计数
uint32_t g_dwTotalRevNum = 0;
class CommonCallback : public TcaplusPbCallback
{
public:
CommonCallback()
{
cout << "Init CommonCallback." << endl;
}

~CommonCallback()
{
cout << "Fini ~CommonCallback." << endl;
}
//收到响应消息的回调，msgs 为获取的记录的数组
int OnRecv(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnRecv[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}

cout << "---------- receive a response----------:" << endl;
```

```
cout << "openid=" << t->openid() << endl;
cout << "tconndid=" << t->tconndid() << endl;
cout << "timekey=" << t->timekey() << endl;
cout << "gamesvrid=" << t->gamesvrid() << endl;
cout << "logintime=" << t->logintime() << endl;
for (int32_t i = 0; i < t->lockid_size();i++)
{
cout << "lockid[" << i << "]=" << t->lockid(i) << endl;
}
tb_online_pay_info pay = t->pay();
cout << "pay total_money=" << pay.total_money() << "; pay_times=" << pay.pay_times() <<
endl;

//获取记录的版本号
std::string version;
int iRet = g_stAsyncApi.GetMessageOption(*t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION
_DATA_VERSION, &version);
cout << "after GetMessageOption iRet= [" << iRet << "] version:" << version.c_str() << " ms
g:" << t << endl;
}

return 0;
}

//收到错误响应消息的回调
int OnError(const std::vector< ::google::protobuf::Message *> &msgs, int errorcode)
{
cout << "OnError[" << msgs.size() << endl;
g_dwTotalRevNum++;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "ERROR: msgs[" << idx << "] not tb_online type." << endl;
return -1;
}
```

```
if (TcapErrCode::TXHDB_ERR_RECORD_NOT_EXIST == errorcode)
{
cout << "ERROR: openid= " << t->openid() << ", tconndid= " << t->tconndid() << ", timekey = " << t->timekey() << ", record not exists" << endl;
}
cout << "ERROR: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] failed:%d" << errorcode << endl;
}

return 0;
}

//消息超时的回调
int OnTimeout(const std::vector< ::google::protobuf::Message *> &msgs)
{
cout << "OnTimeout[" << msgs.size() << endl;
for (size_t idx = 0; idx < msgs.size(); idx++)
{
tb_online* t = dynamic_cast<tb_online *>(msgs[idx]);
if (NULL == t)
{
cout << "TIMEOUT: msgs[" << idx << "] not tb_online type!" << endl;
return -1;
}

cout << "TIMEOUT: openid = [" << t->openid() << "], tconndid = [" << t->tconndid() << "], timekey =[" << t->timekey() << "] timeout" << endl;
}

return 0;
}

int OnFinish(const NS_TCAPLUS_PROTOBUF_API::MsgParam &param)
{
cout << "OnFinish: " << param.m_nOperation << " req: " << param.m_vecMsgs.size()<< endl;
return 0;
```

```
}
};
```

步骤4：发送 delete 请求

```
int SendDelRequest()
{
static tb_online t;
t.set_openid(1);
t.set_tconndid(1);
t.set_timekey("test_tcaplus");

static CommonCallback cb;
std::string version = "-1";
g_stApi.SetMessageOption(t, NS_TCAPLUS_PROTOBUF_API::MESSAGE_OPTION_DATA_VERSION,
version);
int32_t iRet = g_stAsyncApi.Del(&t, &cb);
if (iRet != TcapErrCode::GEN_ERR_SUC)
{
cout << "ERROR: openid= " << t.openid() << ", tconndid= " << t.tconndid() << ", timekey= "
<< t.timekey() << ", Delete Error iRet = " << iRet << endl;
return -1;
}
return 0;
}
```

示例

```
int main(void) {

//初始化 API 客户端
int ret = InitAsyncPbApi();
if ( ret != 0)
{
printf("InitAsyncPbApi failed\n");
return -1;
```

```
}

//发送请求
ret = SendDelRequest();
if (0 != ret)
{
printf("SendDelRequest failed\n");
return -1;
}

//接收响应
do
{
// 更新，接收回包
g_stAsyncApi.UpdateNetwork();
usleep(1000 * 10);
} while (g_dwTotalRevNum != 1);
return 0;
}
```

TDR 表 C++SDK 操作方法
写入数据

最近更新时间：2021-08-10 14:36:07

前提

成功创建：集群，表格组，PLAYERONLINECNT 表

PLAYERONLINECNT 表描述文件 table_test.xml 如下：

```
<?xml version="1.0" encoding="GBK" standalone="yes" ?>
<metalib name="tcaplus_tb" tagsetversion="1" version="1">
<struct name="PLAYERONLINECNT" version="1" primarykey="TimeStamp,GameSvrID" splittablekey="TimeStamp">
<entry name="TimeStamp" type="uint32" desc="单位为分钟" />
<entry name="GameSvrID" type="string" size="64" />
<entry name="GameAppID" type="string" size="64" desc="gameapp id" />
<entry name="OnlineCntIOS" type="uint32" defaultvalue="0" desc="ios在线人数" />
<entry name="OnlineCntAndroid" type="uint32" defaultvalue="0" desc="android在线人数" />
<entry name="BinaryLen" type="smalluint" defaultvalue="1" desc="数据来源数据长度；长度为0时，忽略来源检查"/>
<entry name="binary" type="tinyint" desc="二进制" count= "1000" refer="BinaryLen" />
<entry name="binary2" type="tinyint" desc="二进制2" count= "1000" refer="BinaryLen" />
<entry name="strstr" type="string" size="64" desc="字符串"/>
<index name="index_id" column="TimeStamp"/>
</struct>
</metalib>
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
```

```
// 目标集群的地址个数
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 PLAYERONLINECNT
static const char * TABLE_NAME = "PLAYERONLINECNT";
```

步骤2：初始化 TcaplusAPI 日志句柄

日志配置文件：tlogconf.xml

```
//TCaplus service 日志类
TcaplusService::TLogger* g_pstTlogger;
LPTLOGCATEGORYINST g_pstLogHandler;
LPTLOGCTX g_pstLogCtx;
int32_t InitLog()
{
// 日志配置文件的绝对路径
const char* sLogConfFile = "tlogconf.xml";
// 日志类名
const char* sCategoryName = "mytest";
// 从配置文件初始化日志句柄
g_pstLogCtx = tlog_init_from_file(sLogConfFile);
if (NULL == g_pstLogCtx)
{
fprintf(stderr, "tlog_init_from_file failed.\n");
return -1;
}
// 获取日志类
g_pstLogHandler = tlog_get_category(g_pstLogCtx, sCategoryName);
if (NULL == g_pstLogHandler)
{
fprintf(stderr, "tlog_get_category(mytest) failed.\n");
```

```
return -2;
}
// 初始化日志句柄
g_pstTlogger = new TcaplusService::TLogger(g_pstLogHandler);
if (NULL == g_pstTlogger)
{
fprintf(stderr, "TcaplusService::TLogger failed.\n");
return -3;
}

return 0;
}
```

步骤3：初始化 TcaplusAPI 客户端

```
//TCaplus service API 的客户端主类
TcaplusService::TcaplusServer g_stTcapSvr;
//表的 meta 信息
extern unsigned char g_szMetalib_tcaplus_tb[];
LPTDRMETA g_szTableMeta = NULL;
int32_t InitServiceAPI()
{
// 初始化
int32_t iRet = g_stTcapSvr.Init(g_pstTlogger, /*module_id*/0, /*app id*/APP_ID, /*zone id*/ZONE
_ID, /*signature*/SIGNATURE);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.Init failed, iRet: %d.", iRet);
return iRet;
}

// 添加目录服务器
for (int32_t i = 0; i< DIR_URL_COUNT; i++)
{
iRet = g_stTcapSvr.AddDirServerAddress(DIR_URL_ARRAY[i]);
if (0 != iRet)
{
```

```
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.AddDirServerAddress(%s) failed, iRet: %d.", DIR_
URL_ARRAY[i], iRet);
return iRet;
}
}

// 取得表的 meta 描述
g_szTableMeta = tdr_get_meta_by_name((LPTDRMETALIB)g_szMetalib_tcaplus_tb, TABLE_NAM
E);
if(NULL == g_szTableMeta)
{
tlog_error(g_pstLogHandler, 0, 0,"tdr_get_meta_by_name(%s) failed.", TABLE_NAME);
return -1;
}

// 注册数据表（连接 dir 服务器，认证，获取表路由），10s超时
iRet = g_stTcapSvr.RegistTable(TABLE_NAME, g_szTableMeta, /*timeout_ms*/10000);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.RegistTable(%s) failed, iRet: %d.", TABLE_NAME,
iRet);
return iRet;
}

// 连接表对应的所有 tcaplus proxy 服务器
iRet = g_stTcapSvr.ConnectAll(/*timeout_ms*/10000, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.ConnectAll failed, iRet: %d.", iRet);
return iRet;
}

return 0;
}
```

步骤4：通过 API 发送 replace 请求

您也可以发送 TCAPLUS_API_INSERT_REQ 插入数据，数据存在会返回失败。
TCAPLUS_API_REPLACE_REQ 插入数据，数据存在会替换。

```
int32_t SendReplaceRequest()
{
// 请求对象类
TcaplusService::TcaplusServiceRequest* pstRequest = g_stTcapSvr.GetRequest(TABLE_NAME);
if (NULL == pstRequest)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.GetRequest(%s) failed.", TABLE_NAME);
return -1;
}

//初始化请求对象
int iRet = pstRequest->Init(TCAPLUS_API_REPLACE_REQ, NULL, 0, 0, 0, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->Init(TCAPLUS_API_REPLACE_REQ) failed, iRet: %
d.", iRet);
return iRet;
}

//为请求添加表记录
TcaplusService::TcaplusServiceRecord* pstRecord = pstRequest->AddRecord();
if (NULL == pstRecord)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->AddRecord() failed.");
return -1;
}

PLAYERONLINECNT stPLAYERONLINECNT;
memset(&stPLAYERONLINECNT, 0, sizeof(stPLAYERONLINECNT));

// 设置更新的 key 信息
stPLAYERONLINECNT.dwTimeStamp = 1;
snprintf(stPLAYERONLINECNT.szGameSvrID, sizeof(stPLAYERONLINECNT.szGameSvrID), "%s",
"mysvrid");
```

```
// 设置更新的 value 信息
snprintf(stPLAYERONLINECNT.szGameAppID, sizeof(stPLAYERONLINECNT.szGameAppID), "%s",
"myappid");
stPLAYERONLINECNT.dwOnlineCntIOS = 1;
stPLAYERONLINECNT.dwOnlineCntAndroid = 1;

// 设置基于 TDR 描述设置 record 数据
iRet = pstRecord->SetData(&stPLAYERONLINECNT, sizeof(stPLAYERONLINECNT));
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRecord->SetData() failed, iRet: %d.", iRet);
return iRet;
}

// 发送请求消息包
iRet= g_stTcapSvr.SendRequest(pstRequest);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.SendRequest failed, iRet: %d.", iRet);
return iRet;
}
return 0;
}
```

步骤5：通过 API 接收响应

```
int RecvResp(TcaplusServiceResponse*& response)
{
//此处阻塞收包，每次阻塞1ms，收5000次
unsigned int sleep_us = 1000;
unsigned int sleep_count = 5000;
do
{
usleep(sleep_us);
response = NULL;
int ret = g_stTcapSvr.RecvResponse(response);
if (ret < 0)
```

```
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse failed. ret:%d", ret);
return ret;
}

//收到一个响应包
if (1 == ret)
{
break;
}
} while ((--sleep_count) > 0);

//5s超时
if (0 == sleep_count)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse wait timeout.");
return -1;
}
return 0;
}
```

示例

请参考 main.cpp 文件

```
int main(void) {
//初始化日志
int ret = InitLog();
if ( ret != 0)
{
printf("init log failed\n");
return -1;
}

//初始化 API 客户端
ret = InitServiceAPI();
if ( ret != 0)
```

```
{
printf("init InitServiceAPI failed\n");
return -1;
}

//发送请求
ret = SendReplaceRequest();
if (0 != ret)
{
printf("SendReplaceRequest failed\n");
return -1;
}

//接收响应
TcaplusServiceResponse* response = NULL;
ret = RecvResp(response);
if (0 != ret)
{
printf("RecvResp failed\n");
return -1;
}

//获取操作的结果，0表示成功
int32_t result = response->GetResult();
if (0 != result)
{
printf("the result is %d\n", result);
return -1;
}
return 0;
}
```

读取数据

最近更新时间：2021-08-10 14:36:57

前提

成功创建：集群，表格组，PLAYERONLINECNT 表

PLAYERONLINECNT 表描述文件 table_test.xml 如下：

```
<?xml version="1.0" encoding="GBK" standalone="yes" ?>
<metalib name="tcaplus_tb" tagsetversion="1" version="1">
<struct name="PLAYERONLINECNT" version="1" primarykey="TimeStamp,GameSvrID" splittablekey="TimeStamp">
<entry name="TimeStamp" type="uint32" desc="单位为分钟" />
<entry name="GameSvrID" type="string" size="64" />
<entry name="GameAppID" type="string" size="64" desc="gameapp id" />
<entry name="OnlineCntIOS" type="uint32" defaultvalue="0" desc="ios在线人数" />
<entry name="OnlineCntAndroid" type="uint32" defaultvalue="0" desc="android在线人数" />
<entry name="BinaryLen" type="smalluint" defaultvalue="1" desc="数据来源数据长度；长度为0时，忽略来源检查"/>
<entry name="binary" type="tinyint" desc="二进制" count= "1000" refer="BinaryLen" />
<entry name="binary2" type="tinyint" desc="二进制2" count= "1000" refer="BinaryLen" />
<entry name="strstr" type="string" size="64" desc="字符串"/>
<index name="index_id" column="TimeStamp"/>
</struct>
</metalib>
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
// 目标集群的地址个数
```

```
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 PLAYERONLINECNT
static const char * TABLE_NAME = "PLAYERONLINECNT";
```

步骤2：初始化 TcaplusAPI 日志句柄

日志配置文件：tlogconf.xml

```
//TCaplus service 日志类
TcaplusService::TLogger* g_pstTlogger;
LPTLOGCATEGORYINST g_pstLogHandler;
LPTLOGCTX g_pstLogCtx;
int32_t InitLog()
{
// 日志配置文件的绝对路径
const char* sLogConfFile = "tlogconf.xml";
// 日志类名
const char* sCategoryName = "mytest";
//从配置文件初始化日志句柄
g_pstLogCtx = tlog_init_from_file(sLogConfFile);
if (NULL == g_pstLogCtx)
{
fprintf(stderr, "tlog_init_from_file failed.\n");
return -1;
}
// 获取日志类
g_pstLogHandler = tlog_get_category(g_pstLogCtx, sCategoryName);
if (NULL == g_pstLogHandler)
{
fprintf(stderr, "tlog_get_category(mytest) failed.\n");
return -2;
```

```
}
// 初始化日志句柄
g_pstTlogger = new TcaplusService::TLogger(g_pstLogHandler);
if (NULL == g_pstTlogger)
{
fprintf(stderr, "TcaplusService::TLogger failed.\n");
return -3;
}

return 0;
}
```

步骤3：初始化 TcaplusAPI 客户端

```
//TCaplus service API 的客户端主类
TcaplusService::TcaplusServer g_stTcapSvr;
//表的 meta 信息
extern unsigned char g_szMetalib_tcaplus_tb[];
LPTDRMETA g_szTableMeta = NULL;
int32_t InitServiceAPI()
{
// 初始化
int32_t iRet = g_stTcapSvr.Init(g_pstTlogger, /*module_id*/0, /*app id*/APP_ID, /*zone id*/ZONE
_ID, /*signature*/SIGNATURE);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.Init failed, iRet: %d.", iRet);
return iRet;
}

// 添加目录服务器
for (int32_t i = 0; i< DIR_URL_COUNT; i++)
{
iRet = g_stTcapSvr.AddDirServerAddress(DIR_URL_ARRAY[i]);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.AddDirServerAddress(%s) failed, iRet: %d.", DIR_
```

```
URL_ARRAY[i], iRet);
return iRet;
}
}

// 取得表的 meta 描述
g_szTableMeta = tdr_get_meta_by_name((LPTDRMETALIB)g_szMetalib_tcaplus_tb, TABLE_NAM
E);
if(NULL == g_szTableMeta)
{
tlog_error(g_pstLogHandler, 0, 0,"tdr_get_meta_by_name(%s) failed.", TABLE_NAME);
return -1;
}

// 注册数据表（连接 dir 服务器，认证，获取表路由），10s超时
iRet = g_stTcapSvr.RegistTable(TABLE_NAME, g_szTableMeta, /*timeout_ms*/10000);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.RegistTable(%s) failed, iRet: %d.", TABLE_NAME,
iRet);
return iRet;
}

// 连接表对应的所有 tcaplus proxy 服务器
iRet = g_stTcapSvr.ConnectAll(/*timeout_ms*/10000, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.ConnectAll failed, iRet: %d.", iRet);
return iRet;
}

return 0;
}
```

步骤4：通过 API 发送 get 请求

```
int32_t SendGetRequest()
{
// 请求对象类
TcaplusService::TcaplusServiceRequest* pstRequest = g_stTcapSvr.GetRequest(TABLE_NAME);
if (NULL == pstRequest)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.GetRequest(%s) failed.", TABLE_NAME);
return -1;
}

//初始化请求对象
int iRet = pstRequest->Init(TCAPLUS_API_GET_REQ, NULL, 0, 0, 0, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->Init(TCAPLUS_API_GET_REQ) failed, iRet: %d.", i
Ret);
return iRet;
}

//为请求添加表记录
TcaplusService::TcaplusServiceRecord* pstRecord = pstRequest->AddRecord();
if (NULL == pstRecord)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->AddRecord() failed.");
return -1;
}

PLAYERONLINECNT stPLAYERONLINECNT;
memset(&stPLAYERONLINECNT, 0, sizeof(stPLAYERONLINECNT));

// 设置查询的 key 信息
stPLAYERONLINECNT.dwTimeStamp = 1;
snprintf(stPLAYERONLINECNT.szGameSvrID, sizeof(stPLAYERONLINECNT.szGameSvrID), "%s",
"mysvrid");

// 设置基于 TDR 描述设置 record 数据
iRet = pstRecord->SetData(&stPLAYERONLINECNT, sizeof(stPLAYERONLINECNT));
```

```
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRecord->SetData() failed, iRet: %d.", iRet);
return iRet;
}

// 发送请求消息包
iRet= g_stTcapSvr.SendRequest(pstRequest);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.SendRequest failed, iRet: %d.", iRet);
return iRet;
}
return 0;
}
```

步骤5：通过 API 接收响应

```
int RecvResp(TcaplusServiceResponse*& response)
{
//此处阻塞收包，每次阻塞1ms，收5000次
unsigned int sleep_us = 1000;
unsigned int sleep_count = 5000;
do
{
usleep(sleep_us);
response = NULL;
int ret = g_stTcapSvr.RecvResponse(response);
if (ret < 0)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse failed. ret:%d", ret);
return ret;
}

//收到一个响应包
if (1 == ret)
{
```

```
break;
}
} while ((--sleep_count) > 0);

//5s超时
if (0 == sleep_count)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse wait timeout.");
return -1;
}
return 0;
}
```

示例

请参考 main.cpp 文件

```
int main(void) {
//初始化日志
int ret = InitLog();
if (ret != 0)
{
printf("init log failed\n");
return -1;
}
//初始化 API 客户端
ret = InitServiceAPI();
if ( ret != 0)
{
printf("init InitServiceAPI failed\n");
return -1;
}
//发送请求
ret = SendGetRequest();
if (0 != ret)
{
printf("SendGetRequest failed\n");
```

```
return -1;
}

//接收响应
TcaplusServiceResponse* response = NULL;
ret = RecvResp(response);
if (0 != ret)
{
printf("RecvResp failed\n");
return -1;
}

//获取操作的结果，0表示成功
int32_t result = response->GetResult();
if (0 != result)
{
printf("the result is %d\n", result);
return -1;
}

//获取响应中记录
//batch命令可能一个响应中有多条记录
int count = 0;
const TcaplusService::TcaplusServiceRecord* const_record = NULL;
while((count++) < response->GetRecordCount())
{
//读取记录
const_record = NULL;
printf("--- --- %3d --- ---\n", count - 1);
ret = response->FetchRecord(const_record);
if(0 != ret)
{
printf("FetchRecord() ret:%d\n", ret);
continue;
}

//将记录赋值给结构体
PLAYERONLINECNT stPLAYERONLINECNT;
```

```
memset(&stPLAYERONLINECNT, 0, sizeof(stPLAYERONLINECNT));
ret = const_record->GetData(&stPLAYERONLINECNT, sizeof(stPLAYERONLINECNT));
if(0 != ret)
{
printf("const_record->GetData() failed, ret: %d.", ret);
continue;
}

//打印结构体中的值
printf("struct dwTimeStamp %d szGameSvrID %s szGameAppID %s dwOnlineCntIOS %d dwOnli
neCntAndroid %d \n",
stPLAYERONLINECNT.dwTimeStamp,
stPLAYERONLINECNT.szGameSvrID,
stPLAYERONLINECNT.szGameAppID,
stPLAYERONLINECNT.dwOnlineCntIOS,
stPLAYERONLINECNT.dwOnlineCntAndroid);
}
return 0;
}
```

更新数据

最近更新时间：2021-08-10 14:39:29

前提

成功创建：集群，表格组，PLAYERONLINECNT 表

PLAYERONLINECNT 表描述文件 table_test.xml 如下：

```
<?xml version="1.0" encoding="GBK" standalone="yes" ?>
<metalib name="tcaplus_tb" tagsetversion="1" version="1">
<struct name="PLAYERONLINECNT" version="1" primarykey="TimeStamp,GameSvrID" splittablekey="TimeStamp">
<entry name="TimeStamp" type="uint32" desc="单位为分钟" />
<entry name="GameSvrID" type="string" size="64" />
<entry name="GameAppID" type="string" size="64" desc="gameapp id" />
<entry name="OnlineCntIOS" type="uint32" defaultvalue="0" desc="ios在线人数" />
<entry name="OnlineCntAndroid" type="uint32" defaultvalue="0" desc="android在线人数" />
<entry name="BinaryLen" type="smalluint" defaultvalue="1" desc="数据来源数据长度；长度为0时，忽略来源检查"/>
<entry name="binary" type="tinyint" desc="二进制" count= "1000" refer="BinaryLen" />
<entry name="binary2" type="tinyint" desc="二进制2" count= "1000" refer="BinaryLen" />
<entry name="strstr" type="string" size="64" desc="字符串"/>
<index name="index_id" column="TimeStamp"/>
</struct>
</metalib>
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
// 目标集群的地址个数
```

```
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 PLAYERONLINECNT
static const char * TABLE_NAME = "PLAYERONLINECNT";
```

步骤2：初始化 TcaplusAPI 日志句柄

日志配置文件：tlogconf.xml

```
//TCaplus service 日志类
TcaplusService::TLogger* g_pstTlogger;
LPTLOGCATEGORYINST g_pstLogHandler;
LPTLOGCTX g_pstLogCtx;
int32_t InitLog()
{
// 日志配置文件的绝对路径
const char* sLogConfFile = "tlogconf.xml";
// 日志类名
const char* sCategoryName = "mytest";
//从配置文件初始化日志句柄
g_pstLogCtx = tlog_init_from_file(sLogConfFile);
if (NULL == g_pstLogCtx)
{
fprintf(stderr, "tlog_init_from_file failed.\n");
return -1;
}
// 获取日志类
g_pstLogHandler = tlog_get_category(g_pstLogCtx, sCategoryName);
if (NULL == g_pstLogHandler)
{
fprintf(stderr, "tlog_get_category(mytest) failed.\n");
return -2;
```

```
}
// 初始化日志句柄
g_pstTlogger = new TcaplusService::TLogger(g_pstLogHandler);
if (NULL == g_pstTlogger)
{
fprintf(stderr, "TcaplusService::TLogger failed.\n");
return -3;
}

return 0;
}
```

步骤3：初始化 TcaplusAPI 客户端

```
//TCaplus service API 的客户端主类
TcaplusService::TcaplusServer g_stTcapSvr;
//表的meta信息
extern unsigned char g_szMetalib_tcaplus_tb[];
LPTDRMETA g_szTableMeta = NULL;
int32_t InitServiceAPI()
{
// 初始化
int32_t iRet = g_stTcapSvr.Init(g_pstTlogger, /*module_id*/0, /*app id*/APP_ID, /*zone id*/ZONE
_ID, /*signature*/SIGNATURE);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.Init failed, iRet: %d.", iRet);
return iRet;
}

// 添加目录服务器
for (int32_t i = 0; i< DIR_URL_COUNT; i++)
{
iRet = g_stTcapSvr.AddDirServerAddress(DIR_URL_ARRAY[i]);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.AddDirServerAddress(%s) failed, iRet: %d.", DIR_
```

```
URL_ARRAY[i], iRet);
return iRet;
}
}

// 取得表的 meta 描述
g_szTableMeta = tdr_get_meta_by_name((LPTDRMETALIB)g_szMetalib_tcaplus_tb, TABLE_NAM
E);
if(NULL == g_szTableMeta)
{
tlog_error(g_pstLogHandler, 0, 0,"tdr_get_meta_by_name(%s) failed.", TABLE_NAME);
return -1;
}

// 注册数据表（连接 dir 服务器，认证，获取表路由），10s超时
iRet = g_stTcapSvr.RegistTable(TABLE_NAME, g_szTableMeta, /*timeout_ms*/10000);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.RegistTable(%s) failed, iRet: %d.", TABLE_NAME,
iRet);
return iRet;
}

// 连接表对应的所有 tcaplus proxy 服务器
iRet = g_stTcapSvr.ConnectAll(/*timeout_ms*/10000, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.ConnectAll failed, iRet: %d.", iRet);
return iRet;
}

return 0;
}
```

步骤4：通过 API 发送 update 请求

```
int32_t SendUpdateRequest()
{
// 请求对象类
TcaplusService::TcaplusServiceRequest* pstRequest = g_stTcapSvr.GetRequest(TABLE_NAME);
if (NULL == pstRequest)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.GetRequest(%s) failed.", TABLE_NAME);
return -1;
}

//初始化请求对象
int iRet = pstRequest->Init(TCAPLUS_API_UPDATE_REQ, NULL, 0, 0, 0, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->Init(TCAPLUS_API_UPDATE_REQ) failed, iRet: %
d.", iRet);
return iRet;
}

//为请求添加表记录
TcaplusService::TcaplusServiceRecord* pstRecord = pstRequest->AddRecord();
if (NULL == pstRecord)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->AddRecord() failed.");
return -1;
}

PLAYERONLINECNT stPLAYERONLINECNT;
memset(&stPLAYERONLINECNT, 0, sizeof(stPLAYERONLINECNT));

// 设置更新的 key 信息
stPLAYERONLINECNT.dwTimeStamp = 1;
snprintf(stPLAYERONLINECNT.szGameSvrID, sizeof(stPLAYERONLINECNT.szGameSvrID), "%s",
"mysvrid");

// 设置更新的 value 信息
snprintf(stPLAYERONLINECNT.szGameAppID, sizeof(stPLAYERONLINECNT.szGameAppID), "%s",
```

```
"myappid2");
stPLAYERONLINECNT.dwOnlineCntIOS = 2;
stPLAYERONLINECNT.dwOnlineCntAndroid = 2;

// 设置基于 TDR 描述设置 record 数据
iRet = pstRecord->SetData(&stPLAYERONLINECNT, sizeof(stPLAYERONLINECNT));
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRecord->SetData() failed, iRet: %d.", iRet);
return iRet;
}

// 发送请求消息包
iRet= g_stTcapSvr.SendRequest(pstRequest);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.SendRequest failed, iRet: %d.", iRet);
return iRet;
}
return 0;
}
```

步骤5：通过 API 接收响应

```
int RecvResp(TcaplusServiceResponse*& response)
{
//此处阻塞收包，每次阻塞1ms，收5000次
unsigned int sleep_us = 1000;
unsigned int sleep_count = 5000;
do
{
usleep(sleep_us);
response = NULL;
int ret = g_stTcapSvr.RecvResponse(response);
if (ret < 0)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse failed. ret:%d", ret);
```

```
return ret;
}

//收到一个响应包
if (1 == ret)
{
break;
}
} while ((--sleep_count) > 0);

//5s超时
if (0 == sleep_count)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse wait timeout.");
return -1;
}
return 0;
}
```

示例

请参考 main.cpp 文件

```
int main(void) {
//初始化日志
int ret = InitLog();
if ( ret != 0)
{
printf("init log failed\n");
return -1;
}

//初始化 API 客户端
ret = InitServiceAPI();
if ( ret != 0)
{
printf("init InitServiceAPI failed\n");
```

```
return -1;
}

//发送请求
ret = SendUpdateRequest();
if (0 != ret)
{
printf("SendUpdateRequest failed\n");
return -1;
}

//接收响应
TcaplusServiceResponse* response = NULL;
ret = RecvResp(response);
if (0 != ret)
{
printf("RecvResp failed\n");
return -1;
}

//获取操作的结果，0表示成功
int32_t result = response->GetResult();
if (0 != result)
{
printf("the result is %d\n", result);
return -1;
}

return 0;
}
```

删除数据

最近更新时间：2021-08-10 14:40:06

前提

成功创建：集群，表格组，PLAYERONLINECNT 表

PLAYERONLINECNT 表描述文件 table_test.xml 如下：

```
<?xml version="1.0" encoding="GBK" standalone="yes" ?>
<metalib name="tcaplus_tb" tagsetversion="1" version="1">
<struct name="PLAYERONLINECNT" version="1" primarykey="TimeStamp,GameSvrID" splittablekey="TimeStamp">
<entry name="TimeStamp" type="uint32" desc="单位为分钟" />
<entry name="GameSvrID" type="string" size="64" />
<entry name="GameAppID" type="string" size="64" desc="gameapp id" />
<entry name="OnlineCntIOS" type="uint32" defaultvalue="0" desc="ios在线人数" />
<entry name="OnlineCntAndroid" type="uint32" defaultvalue="0" desc="android在线人数" />
<entry name="BinaryLen" type="smalluint" defaultvalue="1" desc="数据来源数据长度；长度为0时，忽略来源检查"/>
<entry name="binary" type="tinyint" desc="二进制" count= "1000" refer="BinaryLen" />
<entry name="binary2" type="tinyint" desc="二进制2" count= "1000" refer="BinaryLen" />
<entry name="strstr" type="string" size="64" desc="字符串"/>
<index name="index_id" column="TimeStamp"/>
</struct>
</metalib>
```

步骤1：定义配置参数

```
// 目标集群的访问地址
static const char DIR_URL_ARRAY[][TCAPLUS_MAX_STRING_LENGTH] =
{
"tcp://10.191.***.99:9999",
"tcp://10.191.***.88:9999"
};
// 目标集群的地址个数
```

```
static const int32_t DIR_URL_COUNT = 2;
// 目标业务的集群 ID
static const int32_t APP_ID = 3;
// 目标业务的表格组 ID
static const int32_t ZONE_ID = 1;
// 目标业务的业务密码
static const char * SIGNATURE = "*******";
// 目标业务的表名 PLAYERONLINECNT
static const char * TABLE_NAME = "PLAYERONLINECNT";
```

步骤2：初始化 TcaplusAPI 日志句柄

日志配置文件：tlogconf.xml

```
//TCaplus service 日志类
TcaplusService::TLogger* g_pstTlogger;
LPTLOGCATEGORYINST g_pstLogHandler;
LPTLOGCTX g_pstLogCtx;
int32_t InitLog()
{
// 日志配置文件的绝对路径
const char* sLogConfFile = "tlogconf.xml";
// 日志类名
const char* sCategoryName = "mytest";
//从配置文件初始化日志句柄
g_pstLogCtx = tlog_init_from_file(sLogConfFile);
if (NULL == g_pstLogCtx)
{
fprintf(stderr, "tlog_init_from_file failed.\n");
return -1;
}
// 获取日志类
g_pstLogHandler = tlog_get_category(g_pstLogCtx, sCategoryName);
if (NULL == g_pstLogHandler)
{
fprintf(stderr, "tlog_get_category(mytest) failed.\n");
return -2;
```

```
}
// 初始化日志句柄
g_pstTlogger = new TcaplusService::TLogger(g_pstLogHandler);
if (NULL == g_pstTlogger)
{
fprintf(stderr, "TcaplusService::TLogger failed.\n");
return -3;
}

return 0;
}
```

步骤3：初始化 TcaplusAPI 客户端

```
//TCaplus service API 的客户端主类
TcaplusService::TcaplusServer g_stTcapSvr;
//表的meta信息
extern unsigned char g_szMetalib_tcaplus_tb[];
LPTDRMETA g_szTableMeta = NULL;
int32_t InitServiceAPI()
{
// 初始化
int32_t iRet = g_stTcapSvr.Init(g_pstTlogger, /*module_id*/0, /*app id*/APP_ID, /*zone id*/ZONE
_ID, /*signature*/SIGNATURE);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.Init failed, iRet: %d.", iRet);
return iRet;
}

// 添加目录服务器
for (int32_t i = 0; i< DIR_URL_COUNT; i++)
{
iRet = g_stTcapSvr.AddDirServerAddress(DIR_URL_ARRAY[i]);
if (0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.AddDirServerAddress(%s) failed, iRet: %d.", DIR_
```

```
URL_ARRAY[i], iRet);
return iRet;
}
}

// 取得表的 meta 描述
g_szTableMeta = tdr_get_meta_by_name((LPTDRMETALIB)g_szMetalib_tcaplus_tb, TABLE_NAM
E);
if(NULL == g_szTableMeta)
{
tlog_error(g_pstLogHandler, 0, 0,"tdr_get_meta_by_name(%s) failed.", TABLE_NAME);
return -1;
}

// 注册数据表（连接 dir 服务器，认证，获取表路由），10s超时
iRet = g_stTcapSvr.RegistTable(TABLE_NAME, g_szTableMeta, /*timeout_ms*/10000);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.RegistTable(%s) failed, iRet: %d.", TABLE_NAME,
iRet);
return iRet;
}

// 连接表对应的所有 tcaplus proxy 服务器
iRet = g_stTcapSvr.ConnectAll(/*timeout_ms*/10000, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.ConnectAll failed, iRet: %d.", iRet);
return iRet;
}

return 0;
}
```

步骤4：通过 API 发送 delete 请求

```
int32_t SendDeleteRequest()
{
// 请求对象类
TcaplusService::TcaplusServiceRequest* pstRequest = g_stTcapSvr.GetRequest(TABLE_NAME);
if (NULL == pstRequest)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.GetRequest(%s) failed.", TABLE_NAME);
return -1;
}

//初始化请求对象
int iRet = pstRequest->Init(TCAPLUS_API_DELETE_REQ, NULL, 0, 0, 0, 0);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->Init(TCAPLUS_API_DELETE_REQ) failed, iRet: %
d.", iRet);
return iRet;
}

//为请求添加表记录
TcaplusService::TcaplusServiceRecord* pstRecord = pstRequest->AddRecord();
if (NULL == pstRecord)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRequest->AddRecord() failed.");
return -1;
}

PLAYERONLINECNT stPLAYERONLINECNT;
memset(&stPLAYERONLINECNT, 0, sizeof(stPLAYERONLINECNT));

// 设置删除的 key 信息
stPLAYERONLINECNT.dwTimeStamp = 1;
snprintf(stPLAYERONLINECNT.szGameSvrID, sizeof(stPLAYERONLINECNT.szGameSvrID), "%s",
"mysvrid");


// 设置基于 TDR 描述设置 record 数据
```

```
iRet = pstRecord->SetData(&stPLAYERONLINECNT, sizeof(stPLAYERONLINECNT));
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "pstRecord->SetData() failed, iRet: %d.", iRet);
return iRet;
}

// 发送请求消息包
iRet= g_stTcapSvr.SendRequest(pstRequest);
if(0 != iRet)
{
tlog_error(g_pstLogHandler, 0, 0, "g_stTcapSvr.SendRequest failed, iRet: %d.", iRet);
return iRet;
}
return 0;
}
```

步骤5：通过 API 接收响应

```
int RecvResp(TcaplusServiceResponse*& response)
{
//此处阻塞收包，每次阻塞1ms，收5000次
unsigned int sleep_us = 1000;
unsigned int sleep_count = 5000;
do
{
usleep(sleep_us);
response = NULL;
int ret = g_stTcapSvr.RecvResponse(response);
if (ret < 0)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse failed. ret:%d", ret);
return ret;
}

//收到一个响应包
if (1 == ret)
```

```
{
break;
}
} while ((--sleep_count) > 0);

//5s超时
if (0 == sleep_count)
{
tlog_error(g_pstLogHandler, 0, 0, "tcaplus_server.RecvResponse wait timeout.");
return -1;
}
return 0;
}
```

示例

请参考 main.cpp 文件

```
int main(void) {
//初始化日志
int ret = InitLog();
if ( ret != 0)
{
printf("init log failed\n");
return -1;
}

//初始化API客户端
ret = InitServiceAPI();
if ( ret != 0)
{
printf("init InitServiceAPI failed\n");
return -1;
}

//发送请求
ret = SendDeleteRequest();
```

```
if (0 != ret)
{
printf("SendDeleteRequest failed\n");
return -1;
}

//接收响应
TcaplusServiceResponse* response = NULL;
ret = RecvResp(response);
if (0 != ret)
{
printf("RecvResp failed\n");
return -1;
}

//获取操作的结果，0表示成功
int32_t result = response->GetResult();
if (0 != result)
{
printf("the result is %d\n", result);
return -1;
}

return 0;
}
```