数据集成

SDK 文档

腾讯云

文档目录

SDK 文档
日志采集 C++ SDK使用指南

最近更新时间：2024-02-02 15:29:11

操作场景

本文档介绍如何在客户端集成 C++ 版本的 SDK。

操作步骤

步骤一：引入 C++ SDK

需要在项目中包含 SDK 的头文件和库，进行 SDK 的使用。头文件和库提供以下两种获取方式：

1. 获取源码自行编译，请参见 SDK 编译使用 。
2. 直接使用现有的头文件和库， 点击前往下载 。

步骤二：数据上报流程

引入 SDK 后，可以通过调用 SDK 的 `send` 相关接口进行单条（批量）数据的上报，发送 demo 可参考 send_demo.cc 。整体流程包括以下三个步骤：

1. 初始化 SDK
   SDK 支持对象实例化和配置文件初始化两种方式（二选一即可）：
   - **对象实例初始化**
     首先初始化客户端配置，然后调用初始化接口：

```
// 初始化客户端配置
ClientConfig client;
// 设置client相关配置参数，其中proxy_URL_为必选参数（格式如下），其他参数详见
client_config.h文件
client.proxy_cluster_URL_="http://{Manager
url}/inlong/manager/openapi/dataproxy/getIpList";
// 初始化SDK, 返回值为零表示初始化成功，非零表示失败
int32_t result = tc_api_init(client);
```

   - **配置文件初始化**
     配置文件采用 json 格式，请参见 配置文件说明 ，通过配置文件初始化 SDK：

```
// 初始化SDK，参数为配置文件的路径名；返回值为零表示初始化成功
int32_t result = tc_api_init("/home/conf/config.json");
```

2. 调用发送接口进行数据上报
   SDK 支持单条（推荐）和批量发送，二者发送过程均为异步模式，数据上报接口是线程安全的。在进行数据上报前，可设置回调函数在数据发送失败时进行回调处理，回调函数签名如下：

```
int32_t callBackFunc(const char* inlong_group_id, const char* inlong_stream_id,
const char* msg, int32_t msg_len, const int64_t report_time, const char* client_ip);
```

   - 单条数据数据上报接口

```
// 返回值：零表示发送成功，非零表示失败，具体异常返回值详见tc_api.h中的
SDKInvalidReuslt
int32_t tc_api_send(const char* inlong_group_id, const char* inlong_stream_id,
const char* msg, int32_t msg_len, UserCallBack call_back = NULL);
```

   - 批量数据上报接口

```
int32_t tc_api_send_batch(const char* inlong_group_id, const char*
inlong_stream_id, const char** msg_list, int32_t msg_cnt, UserCallBack
call_back = NULL);
```

3. 关闭 SDK
   调用 close 接口关闭 SDK：

```
// 返回值为零表示关闭成功，后续无法再进行数据上报
// max_waitms：关闭SDK前的等待最大毫秒数，等待SDK内部数据发送完成
int32_t tc_api_close(int32_t max_waitms);
```

注意事项

1. SDK 的初始化和关闭都是进程级别的，只需初始化一次，fork 的子进程中需调用初始化接口后再进行数据上报。
2. 建议采用将 SDK 作为常驻服务来进行数据上报，避免同个进程中途频繁地初始化和关闭，重复初始化和关闭会带来更多开销。
3. SDK 发送是异步进行的，返回值为0表示数据成功存入了 SDK 内部缓冲区，等待网络发送。如果 `inlong_group_id` 本身配置有误或者网络异常，也会导致数据发送失败，所以建议用户在调用该接口时设置回调，数据多次重试发送仍失败时执行回调。

配置文件说明

配置文件格式和重要参数如下：

```
{
   "init-param": {
      "thread_num": 5, //网络收发线程的数量
      "enable_pack": true, //是否多条打包发送
      "pack_size": 409600, //数据达到pack_size大小，进行打包发送，单位字节
      "ext_pack_size": 409600, //单条数据最大长度，单位字节
      "enable_zip": true, //是否进行数据压缩
      "min_ziplen": 4096, //最小压缩长度，单位字节
      "enable_retry": true, //发送失败是否进行重试
      "retry_ms": 10000, //重试间隔时间，单位毫秒
      "retry_num": 3, //发送失败最大重试次数
      "max_active_proxy": 4, //tcp最大连接数，用于网络数据收发
      "max_buf_pool": 548576000, //单个数据缓存区大小，单位字节
       "buffer_num_per_groupId": 3, //每个groupid的数据缓存区个数
      "log_num": 10, //最大日志文件数
      "log_size": 10, //单个日志大小限制，单位MB
      "log_level": 3, //日志级别，trace(4)>debug(3)>info(2)>warn(1)>error(0)
      "log_file_type": 2, //日志输出，2->文件, 1->控制台
      "log_path": "./", //日志路径
      "proxy_cfg_preurl":
"http://127.0.0.1:8099/inlong/manager/openapi/dataproxy/getIpList", //访问manager的
url
      "need_auth": false, //是否需要认证
      "auth_id": "admin", //认证id
      "auth_key": "adminKey" //认证key
   }
}
```

日志采集 Java SDK使用指南

最近更新时间：2023-03-14 15:33:35

操作场景

本文档介绍如何在客户端集成Java版本的SDK，往 WeData中进行数据上报。

操作步骤

新建 SDK 实时同步任务

在页面创建任务。

引入 Java SDK

需要在项目中包含SDK的头文件和库，进行SDK的使用。头文件和库提供以下两种获取方式：

1. 获取源码自行编译并将SDK包部署到本地仓库，见 如何编译 。
2. 直接引用Apache仓库里的已有库，代码如下：

```
<dependency>
  <groupId>org.apache.inlong</groupId>
  <artifactId>dataproxy-sdk</artifactId>
  <version>1.4.0</version>
</dependency>
```

数据上报流程

引入 SDK 后，通过实例化一个 MessageSender 接口对象后，调用相关的同步（sendMessage()）或 异步（asyncSendMessage()）接口来完成单条或多条（批量）数据的上报任务。发送Demo可参考 TcpClientExample.java 。整体流程包括以下三个步骤：

初始化 SDK

从Demo示例代码我们可以看到，客户端初始化主要是在 `getMessageSender()` 函数中完成。

```
public DefaultMessageSender getMessageSender(String localIP, String
inLongManagerAddr, String inLongManagerPort, String netTag, String dataProxyGroup,
boolean isLocalVisit, boolean isReadProxyIPFromLocal, String configBasePath, int
msgType) {
```

```
    ProxyClientConfig dataProxyConfig = null;
    DefaultMessageSender messageSender = null;
    try {
      // 初始化客户端配置，其中“test”，“123456”是需要认证的用户名和密码，实际使用时需要
根据环境配置进行更替
      dataProxyConfig = new ProxyClientConfig(localIP, isLocalVisit,
inLongManagerAddr,        Integer.valueOf(inLongManagerPort), dataProxyGroup,
netTag, "test", "123456");
      // 设置配置信息的本地保存路径，该设置可选，缺省情况下 SDK 会在当前用户工作目录下构
造一个"/.inlong/"目录存储配置数据
      if (StringUtils.isNotEmpty(configBasePath)) {
        dataProxyConfig.setConfStoreBasePath(configBasePath);
      }
      // 设置是否允许使用本地保存的配置信息，该设置可选，缺省不启用
      dataProxyConfig.setReadProxyIPFromLocal(isReadProxyIPFromLocal);
      // 初始化MessageSender对象，异常将抛异常
      messageSender =
DefaultMessageSender.generateSenderByClusterId(dataProxyConfig);
      // 设置 SDK 与DataProxy间消息发送的消息类型，该设置可选，缺省默认为7以二进制形式进
行数据发送
      messageSender.setMsgtype(msgType);
    } catch (Exception e) {
      logger.error("getMessageSender has exception e = {}", e);
    }
    // 返回初始化结果
    return messageSender;
    }
```

调用发送接口进行数据上报

SDK 的数据发送接口时线程安全的，支持以同步或者异步模式发送单条或多条消息。Demo里采用的是单条同步消息发送，并且消息中不包含属性信息。

```
public void sendTcpMessage(DefaultMessageSender sender, String inlongGroupId,
String inlongStreamId, String messageBody, long dt) {
    SendResult result = null;
    try {
      // 以同步模式发送单条消息，不携带属性信息
      result = sender.sendMessage(messageBody.getBytes("utf8"), inlongGroupId,
inlongStreamId,
                                  0, String.valueOf(dt), 20, TimeUnit.SECONDS);
    } catch (UnsupportedEncodingException e) {
      e.printStackTrace();
    }
```

```
    logger.info("messageSender {}", result);
    }
```

您还可以根据业务需要选择不同的发送接口进行数据上报，具体接口细节可以参考MessageSender接口文件中的定义，里面有详细的接口使用及参数定义介绍，这里不做额外说明。

关闭 SDK

Demo 里没有实现关闭操作，使用时我们需要调用MessageSender接口对象的close()函数关闭数据上报服务。

注意事项

- MessageSender接口对象是基于GroupID进行初始化，因而每个MessageSender对象基于GroupID区别使用，同一个进程内允许创建多个MessageSender对象。
- SDK 封装了TCP、HTTP、UDP共三种不同的网络交互方式，并在example目录里给出了3种方式的不同示例（参考TcpClientExample.java，HttpClientExample.java，UdpClientExample.java实现），业务可以根据自身需要来初始化不同的MessageSender对象。
- SDK 中包含了复杂的网络交互，使用时需要将 SDK 作为常驻服务对象来使用，避免同个进程中途频繁地初始化和关闭MessageSender对象（重复初始化和关闭会带来很大的资源开销，并且影响数据上报的时效性）。
- SDK 不对发送失败的消息做重发处理，用户在使用 SDK 上报数据时遇到发送失败，业务要根据自身数据要求来决定是否重发消息，避免数据丢失。

错误码介绍

常见result会有以下几种值：

返回值	含义	备注
SendResult.OK	消息发送成功	-
SendResult.TIMEOUT	请求响应超时	-
SendResult.CONNECTION_BREAK	链接被断开	-
SendResult.THREAD_INTERRUPT	中断	-
SendResult.ASYNC_CALLBACK_BUFFER_FULL	SDK 待回包请求消息满	这种情况一般为前端生产数据的速度超过服务端的响应速度导致，建议发送时适当sleep避免阻塞
SendResult.NO_CO	没有可用链接	这种情况建议业务增大可用链接数

NNECTION		
SendResult.INVALID_DATA	数据无效，通过HTTP上报数据DataProxy返回失败	-
SendResult.INVALID_ATTRIBUTES	发送的数据包不合理，例如为空数据包或包含了系统预定义属性	-
SendResult.UNKOWN_ERROR	未知错误	-

ProxyClientConfig相关配置项介绍

参数设置	说明	调整建议
setAliveConnections(int aliveConnections)	设置DataProxy连接数大小；默认值：3	1. 数据量大或对时延敏感，适当增大该参数 2. 根据DataProxy集群大小，适当调整该参数，例如集群规模为30，该值可设为5～10 3. 现网经验值15～20
setTotalAsyncCallbackSize(int asyncCallbackSize)	设置异步发送时 SDK 内部缓冲队列大小；缓存队列用于暂存已发送但未收到服务端Ack的数据包。当缓冲数据达到该值，业务继续异步上报数据，会收到ASYNC_CALLBACK_BUFFER_FULL异常；默认值： 50000	1. 通常无需调整该参数 2. 数据量非常大或者DataProxy服务端负载较高情况下，可适当增大，注意不要太大导致OOM
setConnectTimeoutMillis(long connectTimeoutMillis)	设置连接超时时长，单位ms，缺省40000	根据实际环境需要设置
setRequestTimeoutMillis(long requestTimeoutMillis)	设置请求超时时长，单位ms，缺省40000	根据需要调整设置
setMaxTimeoutCnt(int maxTimeoutCnt)	设置单个DataProxy连接超时断连次数； SDK 内部会对超时未收到Ack的DataProxy连接进行计数，短时间内同一个连接超时数达到该值，会主动断开该连接，选择其他DataProxy创建新的连接进行数据发送。默认值：3	如果DataProxy集群本身规模较小，可适当调大该参数，避免短时间频繁断连

setManagerConnectionTimeout(int managerConnectionTimeout)	设置 SDK 连接Manager的超时时长，单位ms，默认10000ms	1. 网络环境不好的情况下可适当增大该值 2. 客户端解析域名时间较长情况下可适当增大该值
setManagerSocketTimeout(int managerSocketTimeout)	设置 SDK 从Manager连接读取DataProxy列表的超时时间，单位ms，默认值30000	网络环境不好的情况下可适当增大该值