IoT Datahub iOS SDK

by Dasudian

Contents

1 IoT Datahub iOS SDK

SDK基于MQTT协议,传输实时的消息到大数点IoT云服务器
你可以收集设备上的数据发送到云上;也可以订阅某个topic,来接收云服务器推送的 消息

如何使用SDK:
1: 创建一个客户端实例
2: 如果想接收消息,那么就订阅某个topic
3: 或者发送消息到服务器
4: 退出时,销毁该客户端

功能:
1: 当连接丢失时,SDK会尝试自动重连。如果连接失败,下一次重连将会在1s,2s,4s,8s, 16s,1s,2s,4s,8s,16s,1s,...后再次尝试,最大重连间隔为16秒
2: SDK为线程安全
3: 一个进程只能创建一个客户端, 如果需要多个客户端, 需要创建多个进程

2 更新日志

Author	Date	Version	Note
Eden Wang	10/27/2017	3.0.0	增加数据类型(JSON, TEXT, BINARY);增加设备类型;修改验证方式
Eden Wang	7/05/2017	2.2.0	更新API文档为pdf格式;修复bug
Eden Wang	4/20/2017	2.1.0	新增选项option; 订阅时可设置QoS
Eden Wang	3/21/2017	2.0.0	根据SDK的标准修改API
Jack Liu	2/28/2017	1.0.1	修复了调用publish函数过快时，阻塞UI线程的BUG
Jack Liu	2/27/2017	1.0.0	发布版本1.0.0

3 结构体说明

3.1 datahub_message_s结构体 参考

消息的结构体类型

```
#include <DataHubCommon.h>
```

成员变量

- unsigned int payload_len
- void * payload

3.1.1 详细描述

消息的结构体类型

3.1.2 结构体成员变量说明

3.1.2.1 payload_len

```
unsigned int payload_len
```

消息长度，必须大于0

3.1.2.2 payload

```
void* payload
```

发送消息的起始地址

该结构体的文档由以下文件生成:

- DataHubCommon.h

3.2 DataHubClient::datahub_options_s结构体 参考

选项

```
#import <DataHubClient.h>
```

Protected 属性

- char * server_url
  设置使用哪种协议*(ssl*, 普通的*tcp)*连接服务器, 以及设置服务器的地址和端口号.
- int debug
  开启调试选项
- int cleansession
  是否保存客户端的会话信息.
- void * context
  传递给回调函数*messageReceived()*和*connectionStatusChanged()*的参 数, 对应回调函数的第一个参数*context*

3.2.1 详细描述

选项

3.2.2 结构体成员变量说明

3.2.2.1 server_url

```
- (char*) server_url  [protected]
```

设置使用哪种协议(ssl, 普通的tcp)连接服务器, 以及设置服务器的地址和端口号.

格式为"协议://服务器地址:端口号". 协议支持普通的tcp协议和加密的ssl协议; 服务器地址和端口号由大数点提供.
默认值为DEFAULT_SERVER_URL

3.2.2.2 debug

```
- (int) debug  [protected]
```

开启调试选项

DATAHUB_TRUE表示开启调试选项
DATAHUB_FALSE表示关闭调试选项
默认值为DATAHUB_FALSE

3.2.2.3 cleansession

```
- (int) cleansession  [protected]
```

是否保存客户端的会话信息.

cleansession为DATAHUB_FALSE, 当客户端断线或下线后, 保存客户端订阅的 topic和发送给客户端的所有消息.
cleansession为DATAHUB_TRUE, 当客户端断线或下线后, 不保留客户端订阅的 topic和发送给客户端的任何消息.

3.2.2.4 context

```
- (void*) context  [protected]
```

传递给回调函数messageReceived()和connectionStatusChanged()的参数, 对应回调函数的第一个参数context

该结构体的文档由以下文件生成:

- DataHubClient.h

3.3 DataHubClient类 参考

客户端

```
#import <DataHubClient.h>
```

继承自 <NSObject> .

结构体

- struct datahub_options_s
  选项

构造函数

- (int) - datahub_create:instance_id:instance_key:client_type:client_id:options:
  该函数创建一个客户端实例,该实例可用于连接大数点*MQTT*服务器
- (int) - datahub_sendrequest:topic:msg:data_type:QoS:timeout:
  同步发送消息
- (int) - datahub_subscribe:topic:QoS:timeout:
  同步订阅某一个*topic*
- (int) - datahub_unsubscribe:topic:timeout:
  同步取消订阅某一个*topic*
- (void) - datahub_destroy:
  销毁客户端并断开连接
- (void) - datahub_callback_free:message:
  接收函数中,主题和消息占用的内存需要用户手动释放

类方法

- (instancetype) + shareInstance

Protected 类型

- typedef struct DataHubClient::datahub_options_s datahub_options
  选项

属性

- id< DataHubClientDelegate > delegate
  设置代理
- messageReceivedBlock messageReceivedBlock
  *block*变量，用于收到消息后的回调函数. 默认为空

3.3.1 详细描述

客户端

每个客户端由client id唯一确定, 所以, 需要确保每个连接服务器的设备拥有唯一的 client id

3.3.2 成员类型定义说明

3.3.2.1 datahub_options

```
- (typedef struct datahub_options_s) datahub_options  [protected]
```

选项

3.3.3 函数文档

3.3.3.1 shareInstance()

```
+ (instancetype) shareInstance
```

表示客户端实例

3.3.3.2 datahub_create:instance_id:instance_key:client_type:client_id:options:()

```
- (int) datahub_create:
            (datahub_client *) client
            instance_id:(char *) instance_id
            instance_key:(char *) instance_key
            client_type:(char *) client_type
            client_id:(char *) client_id
            options:(datahub_options *) options
```

该函数创建一个客户端实例,该实例可用于连接大数点MQTT服务器

参数

client	如果函数成功调用,则会返回一个客户端实例 注意: 不能为空
instance_id	用于连接大数点服务器的唯一标识,由大数点提供 注意: 不能为空
instance_key	用于连接大数点服务器的密码,由大数点提供 注意: 不能为空
client_type	设备类型. 如传感器"sensor", 充电桩"charging_pile", 车载电池"car_battery" 注意:可以为空
client_id	设备的id, 用于服务器唯一标记一个设备 注意： 不同的设备client id必须不同,如果两个设备有相同的id,服务器会关 掉其中一个连接 可以使用设备的mac地址,或者第三方账号系统的id(比如qq号,微信号);如果没 有自己的账号系统,则可以随机生成一个不会重复的id.或者自己指定设备的 client id,只要能保证不同客户端id不同即可. client_id不能为空
options	MQTT的选项.具体包含的选项可以查看datahub_options结构体. 如果不想设置选项, 请传递NULL. 如果想设置某些选项,先使用DATAHUB_OPTIONS_INITIALIZER初始化, 再设置 注意: 可以为空

返回

ERROR_NONE 表示成功,其他表示错误.
其他错误码请查看开发文档

3.3.3.3 datahub_sendrequest:topic:msg:data_type:QoS:timeout:()

```
- (int) datahub_sendrequest:
            (datahub_client *) client
            topic:(char *) topic
            msg:(datahub_message *) msg
            data_type:(datahub_data_type) data_type
            QoS:(int) qos
            timeout:(int) timeout
```

同步发送消息

警告

注意：程序会阻塞,建议创建一个子线程单独调用

参数

client	由函数datahub_create()成功返回的客户端实例 注意: 不能为空
topic	消息对应的topic.如果消息发送前有另一个客户端已经订阅该topic,则 另一个客户端就会收到消息. 注意: 不能为空
msg	发送的消息,使用前请使用DATAHUB_MESSAGE_INITIALIZER初始化. 注意: 消息的长度必须小于512K,否则会发生错误 注意: 不能为空
data_type	数据类型，只能为JSON或TEXT或BINARY
qos	消息的服务质量 **0**: 消息可能到达,也可能不到达 **1**: 消息一定会到达,但可能会重复,当然,前提是返回ERROR_NONE **2**: 消息一定会到达,且只到达一次,当然,前提是返回ERROR_NONE 注意: 只能为0,1,2三者中的一个,其他为非法参数
timeout	函数阻塞的最大时间. 注意: 这是函数阻塞的最大时间,不是消息的超时时间

返回

ERROR_NONE 表示成功,消息一定发送出去.
ERROR_TIMEOUT 表示阻塞等待时间的最大值已到,但是消息可能发送给服务器,也可能 未发送.如果想确保消息一定发送出去,请根据消息大小和网络状况设置较大的阻塞等 待时间.

其他错误码请查看开发文档

3.3.3.4 datahub_subscribe:topic:QoS:timeout:()

```
- (int) datahub_subscribe:
            (datahub_client *) client
            topic:(char *) topic
            QoS:(int) qos
            timeout:(int) timeout
```

同步订阅某一个topic

警告

注意：程序会阻塞

参数

client	由函数datahub_create()成功返回的客户端实例 注意: 不能为空
topic	订阅的topic 注意: 不能为空
qos	订阅消息的服务质量(发送消息的qos和订阅消息的qos共同决定服务器下发消息的qos) **0**: 消息可能到达,也可能不到达 **1**: 消息一定会到达,但可能会重复,当然,前提是返回ERROR_NONE **2**: 消息一定会到达,且只到达一次,当然,前提是返回ERROR_NONE
timeout	函数阻塞的最大时间. 注意: 这是函数阻塞的最大时间

返回

ERROR_NONE 表示成功,其他表示错误.
ERROR_TIMEOUT 表示阻塞等待时间的最大值已到,但是可能订阅成功,也可能订阅失败. 如果想确保订阅一定成功,请根据网络状况设置较大的阻塞等待时间.
其他错误码请查看开发文档

3.3.3.5 datahub_unsubscribe:topic:timeout:()

```
- (int) datahub_unsubscribe:
            (datahub_client *) client
            topic:(char *) topic
            timeout:(int) timeout
```

同步取消订阅某一个topic

参数

client	由函数datahub_create()成功返回的客户端实例 注意: 不能为空
topic	取消订阅的topic 注意: 不能为空
timeout	函数阻塞的最大时间. 注意: 这是函数阻塞的最大时间

返回

ERROR_NONE 表示成功,其他表示错误.
ERROR_TIMEOUT 表示阻塞等待时间的最大值已到,但是可能取消成功,也可能取消失败. 如果想确保取消一定成功,请根据网络状况设置较大的阻塞等待时间.
其他错误码请查看开发文档

3.3.3.6 datahub_destroy:()

```
- (void) datahub_destroy:
            (datahub_client *) client
```

销毁客户端并断开连接

参数

client	由函数datahub_create()成功返回的客户端实例 注意 不能为空

返回

无

3.3.3.7 datahub_callback_free:message:()

```
- (void) datahub_callback_free:
            (char *) topic
           message:(datahub_message *) msg
```

接收函数中,主题和消息占用的内存需要用户手动释放

参数

topic	返回的主题
msg	返回的消息

返回

无

3.3.4 属性说明

3.3.4.1 delegate

```
- (id<DataHubClientDelegate>) delegate  [read], [write], [nonatomic], [assign]
```

设置代理

3.3.4.2 messageReceivedBlock

```
- (messageReceivedBlock) messageReceivedBlock  [read], [write], [nonatomic], [strong]
```

block变量，用于收到消息后的回调函数. 默认为空

该类的文档由以下文件生成:

- DataHubClient.h

3.4 <DataHubClientDelegate >协议 参考

设置可选的代理函数

```
#import <DataHubClient.h>
```

继承自 <NSObject> .

构造函数

- (void) - messageReceived:topic:message:
  接收到消息后的回调函数
- (void) - connectionStatusChanged:isconnected:
  当客户端的状态发生改变(连接上服务器或者从服务器断开)的时候,SDK会通知用户

3.4.1 详细描述

设置可选的代理函数

3.4.2 函数文档

3.4.2.1 messageReceived:topic:message:()

```
- (void) messageReceived:
             (void *) context
             topic:(char *) topic
             message:(datahub_message *) msg   [optional]
```

接收到消息后的回调函数

参数

context	传递给选项context的内容
topic	本次消息所属的主题,需要调用datahub_callback_free()手动释放内存
msg	存放消息的结构体,需要调用datahub_callback_free()手动释放内存

3.4.2.2 connectionStatusChanged:isconnected:()

```
- (void) connectionStatusChanged:
             (void *) context
             isconnected:(int) isconnected   [optional]
```

当客户端的状态发生改变(连接上服务器或者从服务器断开)的时候,SDK会通知用户

参数

context	传递给选项context的内容
isconnected	连接状态 DATAHUB_FALSE 表示从服务器断开 DATAHUB_TRUE 表示连接上服务器

该协议的文档由以下文件生成:

- DataHubClient.h

4 文件说明

4.1 DataHubClient.h 文件参考

```
#import <Foundation/Foundation.h>
#import "DataHubCommon.h"
```

结构体

- protocol <DataHubClientDelegate >
  设置可选的代理函数
- class DataHubClient
  客户端
- struct DataHubClient::datahub_options_s
  选项

类型定义

- typedef void(^ messageReceivedBlock) (void *context, char *topic, datahub_message *msg)
  接收到消息后的回调函数

4.1.1 类型定义说明

4.1.1.1 messageReceivedBlock

```
typedef void(^ messageReceivedBlock) (void *context, char *topic, datahub_message *msg)
```

接收到消息后的回调函数

参数

context	传递给选项context的内容
topic	本次消息所属的主题,需要调用datahub_callback_free()手动释放内存
msg	存放消息的结构体,需要调用datahub_callback_free()手动释放内存

注解

如果实现了可选的函数messageReceived, 则优先使用messageReceived

4.2 DataHubCommon.h 文件参考

结构体

- struct datahub_message_s
  消息的结构体类型

宏定义

- #define DATAHUB_TRUE 1
- #define DATAHUB_FALSE 0
- #define DEFAULT_SERVER_URL "tcp://try.iotdatahub.net:1883"
- #define DEFAULT_DEBUG_OPT DATAHUB_FALSE
- #define DEFAULT_CLEANSESSION DATAHUB_FALSE
- #define DEFAULT_CONTEXT NULL
- #define DATAHUB_OPTIONS_INITIALIZER
- #define DATAHUB_MESSAGE_INITIALIZER
- #define DATAHUB_DT_INITIALIZER {0}

类型定义

- typedef void * datahub_client
  客户端的类型
- typedef struct datahub_message_s datahub_message
  消息的结构体类型
- typedef enum datahub_data_type_s datahub_data_type

枚举

- enum datahub_error_code_s {
  ERROR_NONE = 0, ERROR_ILLEGAL_PARAMETERS = -1, ERROR_DISCONNECTED = -2, ERROR_UNACCEPT_PROTOCOL_VERSION = -3,
  ERROR_IDENTIFIER_REJECTED = -4, ERROR_SERVER_UNAVAILABLE = -5, ERROR_BAD_USERNAME_OR_PASSWD = -6, ERROR_UNAUTHORIZED = -7,
  ERROR_AUTHORIZED_SERVER_UNAVAILABLE = -8, ERROR_OPERATION_FAILURE = -9, ERROR_MESSAGE_TOO_BIG = -10, ERROR_NETWORK_UNREACHABLE = -11,
  ERROR_TIMEOUT = -12, ERROR_MEMORY_ALLOCATE = -500, ERROR_MESSAGE_INVALID_JSON = -501, ERROR_INVALID_CLIENT_TYPE = -502 }
  错误码
- enum datahub_data_type_s { JSON = 0, TEXT = 1, BINARY = 2, DATA_TYPE_END }

4.2.1 宏定义说明

4.2.1.1 DATAHUB_TRUE

```
#define DATAHUB_TRUE 1
```

4.2.1.2 DATAHUB_FALSE

```
#define DATAHUB_FALSE 0
```

4.2.1.3 DEFAULT_SERVER_URL

```
#define DEFAULT_SERVER_URL "tcp://try.iotdatahub.net:1883"
```

4.2.1.4 DEFAULT_DEBUG_OPT

```
#define DEFAULT_DEBUG_OPT DATAHUB_FALSE
```

4.2.1.5 DEFAULT_CLEANSESSION

```
#define DEFAULT_CLEANSESSION DATAHUB_FALSE
```

4.2.1.6 DEFAULT_CONTEXT

```
#define DEFAULT_CONTEXT NULL
```

4.2.1.7 DATAHUB_OPTIONS_INITIALIZER

```
#define DATAHUB_OPTIONS_INITIALIZER
```

值:

```
{\
DEFAULT_SERVER_URL,\
DEFAULT_DEBUG_OPT,\
DEFAULT_CLEANSESSION,\
DEFAULT_CONTEXT,\
}
```

选项的初始化宏

4.2.1.8 DATAHUB_MESSAGE_INITIALIZER

```
#define DATAHUB_MESSAGE_INITIALIZER
```

值:

```
{\
1,\
""\
}
```

消息的初始化宏,只包含字符串结尾符'\0'

4.2.1.9 DATAHUB_DT_INITIALIZER

```
#define DATAHUB_DT_INITIALIZER {0}
```

4.2.2 类型定义说明

4.2.2.1 datahub_client

```
typedef void* datahub_client
```

客户端的类型

4.2.2.2 datahub_message

```
typedef struct datahub_message_s datahub_message
```

消息的结构体类型

4.2.2.3 datahub_data_type

```
typedef enum datahub_data_type_s datahub_data_type
```

数据类型

4.2.3 枚举类型说明

4.2.3.1 datahub_error_code_s

```
enum datahub_error_code_s
```

错误码

枚举值

ERROR_NONE	成功
ERROR_ILLEGAL_PARAMETERS	某些参数不合法
ERROR_DISCONNECTED	客户端未连接服务器
ERROR_UNACCEPT_PROTOCOL_VERSION	MQTT服务器不支持当前使用的协议版本号,请联系开发人员
ERROR_IDENTIFIER_REJECTED	client_id不可用,可能使用了不支持的字符
ERROR_SERVER_UNAVAILABLE	服务器不可用
ERROR_BAD_USERNAME_OR_PASSWD	instance_id 或者instance_key不正确,请检查或者联系客服人员
ERROR_UNAUTHORIZED	未被授权
ERROR_AUTHORIZED_SERVER_UNAVAILABLE	验证服务器不可用
ERROR_OPERATION_FAILURE	操作失败
ERROR_MESSAGE_TOO_BIG	消息过长
ERROR_NETWORK_UNREACHABLE	网络不可用
ERROR_TIMEOUT	同步超时
ERROR_MEMORY_ALLOCATE	内存申请失败

4.2.3.2 datahub_data_type_s

enum datahub_data_type_s

数据类型

枚举值

JSON	数据为JSON格式
TEXT	数据为文本/字符串
BINARY	数据为二进制
DATA_TYPE_END	

4.3 release_note.txt 文件参考

Index