恒生行情云服务
H5SDK(C++)
开发指南

文档版本	V1.0
发布日期	2016-04-09

目录

前言

产品简介

H5SDK 是客户端接入 H5 行情云服务器的开发包，业务开发者通过使用 H5SDK 开发包可以轻松完成与 H5 行情云服务器的对接，实现行情基本信息的获取。

本文描述了 H5SDK （C++） for windows 或 linux 开发包的开发指南，相关协议请参见文档《H5 行情服务协议(修订版).xls》。

读者对象

本指南主要适用于以下人员：

h5Sdk 开发和使用人员；

手册概况

本手册主要提供 c++的开发说明，C#开发人员参考此文档进行开发。

本手册各章节内容如下表所示。

章节	内容
1 发布文件说明	H5SDK 的发布项和相关配置说明
2 开发流程	H5SDK 一般的开发流程
3 开发接口	H5SDK 的接口详细说明
4 注意事项	H5SDK 的注意事项
5 示例代码	开发的 demo 的示例代码

1 发布文件说明

1.1 发布文件及运行环境说明

H5SDK 开发包中的发布文件如下表所示。

demo			演示目录
	demo.cpp		演示代码
	makefile.gcc		linux 编译脚本
	makefile.win		windows 编译脚本
include			头文件目录
	h5sdk		h5sdk 头文件目录
		h5sdk.h	h5sdk 主头文件
		h5sdk_export.h	h5sdk 导出函数
		velocimetry.h	h5sdk 测速函数
	Include		其它头文件目录
		hscomm_message_factory_interface.h	消息工厂接口
		hscomm_message_interface.h	消息接口
		plugin_interface.h	插件接口
		pluginid.h	插件 id
		sdk_tag.h	h5sdk 常量定义
lib-linux32			32 位 linux 库文件
	libf_os.so		公共库

	libfsc_f2config.so		配置库
	libfsc_hscomm.so		通信库
	libfsc_hscommsg.so		消息库
	libh5sdk.so		h5sdk 库
	libhs_best_message.so		best 消息库
lib-win32			32 位 windows 库文件
	f_os.dll		公共库
	fsc_f2config.dll		配置库
	fsc_hscommsg.dll		消息库
	h5sdk.dll		h5sdk 库
	h5sdk.lib		h5sdk 编译用
	hs_best_message.dll		best 消息库
	zlib1.dll		zlib 压缩库
	h5sdkAdapter.dll		h5sdk c#库
	h5sdkCsharp.dll		c#支持库
H5SDK 外部开发指南.docx			开发文档
环境变量设置文档	SdkParamPath.json		配置文档

1.2 配置文件说明

1.2.1 配置文件 SdkParamPath.json 格式

```
{
  "loglevels": 7,
  "loglevels_comment":"设置日志级别 1:
ERROR ,3:ALARM,7:NOTICE,15:FILDBG,31:DEBUG,255:ALL",
  "AsyncTime": 10,
  "AsyncTime_comment":"异步请求超时时间,单位秒",
  "subPerSession": 5,
```

```
  "subPerSession_comment": "每个 session 预分配订阅通道数"
}
```

1.3 代码转换规则

目前行情服务器会对指数代码进行特殊转换，具体转换规则见 code_switch.xml。代码转换表如下所示。

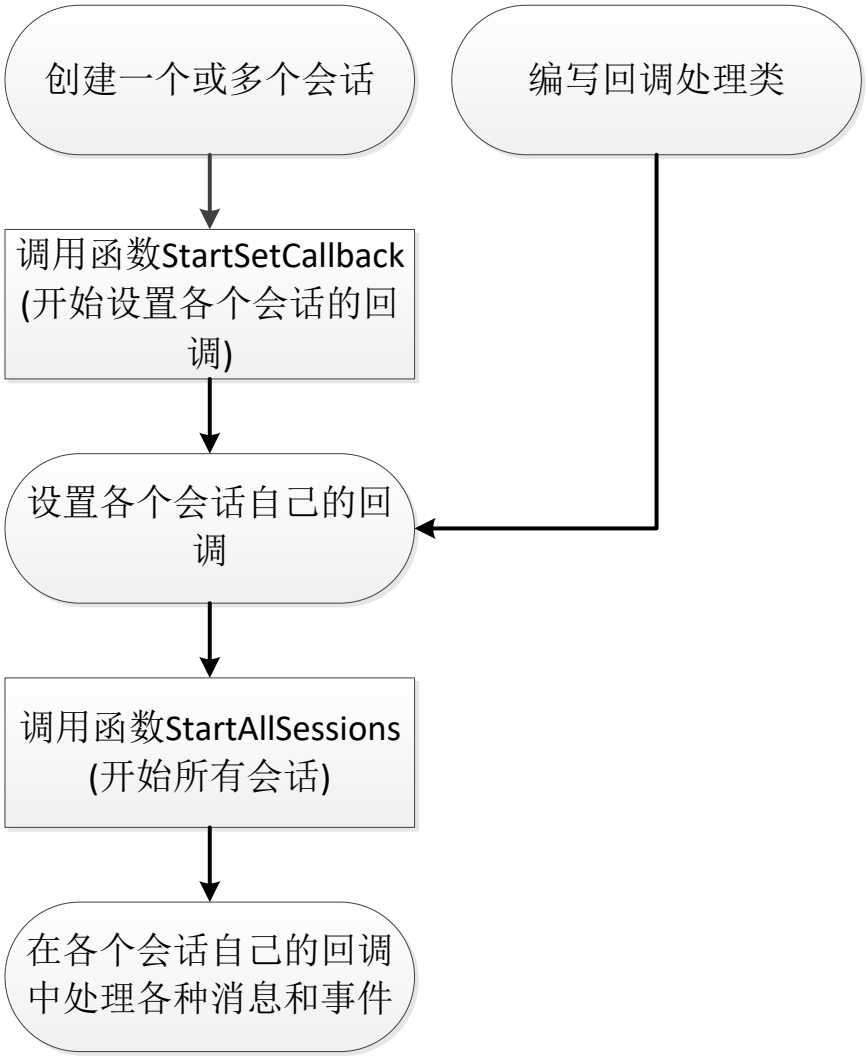
市场类型	转换前代码	转换后代码
0X101	000001	1A0001
	000002	1A0002
	000003	1A0003
	000XXX	1B0XXX
0X102	399001	2A01
	399002	2A02
	399003	2A03
	399107	2C02

当应用层使用 H5SDK 访问行情服务器时，应用层使用的代码必须与服务器上的代码一致，否则将无法访问。应用层需要根据代码转换表做代码转换工作。例如要访问 000001 指数，必须输入 1A0001 才能访问；访问 000002 指数，必须输入 1A0002。

1.4 消息定义

发布包中《 H5 行情服务协议(修订版).xls》文档详细定义了各个请求和应答消息的结构，此文档也是开发基础。

2 开发流程



经过上述步骤所示，便可将 H5SDK 启动，并使之处于工作状态之中，之后所有的任务都在会话的回调中处理。

注意

StartAllSessions 一旦调用不能再次创建会话

Sdk 相关事件均在回调类（H5SdkCallback）中触发调用。

3 开发接口

本章主要介绍开发包提供的重要接口和函数参数说明。没有特殊说明，接口都是线程安全的。

3.1H5SDK 导出函数说明

3.1.1 Sdk 版本号接口（GetSdkVersion）

函数原型

```
const char * H5SDKAPI GetSdkVersion();
```

输入参数

无

返回

返回值	含义
const char *	Sdk 版本信息

用法说明

无

3.1.2 创建会话选项(CreateSessionOptions)

函数原型

```
SessionOptions * H5SDKAPI CreateSessionOptions();
```

输入参数

无

返回

SessionOptions 指针，具体用户参考 3.2 小节，主要用于会话创建时候参数设置

用法说明

每个会话都有自己的会话选项，即 SessionOptions 也可以说是会话的属性

3.1.3 会话创建(CreateSession)

函数原型

```
Session * H5SDKAPI CreateSession(SessionOptions *sessionOptions)
```

输入参数

参数名称	参数说明
SessionOptions *sessionOptions	● 3.1.2 小节创建会话选项，创建完成后返回此参数

返回

返回值	含义
Session *	会话指针，用于登录，消息创建，是 sdk 开发的核心
NULL	失败

用法说明

3.1.4 测速接口（CreateVelocimetry）

函数原型

```
Velocimetry * H5SDKAPI CreateVelocimetry();
```

输入参数

无

返回

返回值	含义
Velocimetry *	测速对象指针
NULL	失败

用法说明

具体用法参考 5.4 小节

注意

建议测速后再创建会话选项，设置会话信息并创建会话

3.1.5 获取错误会话号（GetErrorStringByErrorNo）

函数原型

```
const char * H5SDKAPI GetErrorStringByErrorNo(int errorNo);
```

输入参数

参数名称	参数说明
int errorNo	错误号。

常见错误号：

```
const int H5SDK_DISCONNECT         = -1;
const int H5SDK_TIMEOUT           = -3;
const int H5SDK_SDK_LOGIN_ERROR    = 1010;
const int H5SDK_SERIALIZATION_ERROR = 1002;
const int H5SDK_SET_DECODER_ERROR   = 1004;
const int H5SDK_SDK_LOGIN_TIMEOUT   = 1005;
```

返回

错误信息字符串

用法说明

一般是 sdk 同步调用的错误号，获取具体错误信息

3.1.6 开始设置会话回调行情（StartSetCallback）

函数原型

```
int H5SDKAPI StartSetCallback();
```

输入参数

无

返回

返回值	含义
0	成功
1	失败

用法说明

所有会话创建完成后，才可以调用此接口设置会话的回调类，会话类设置参考 3.2 小节.

3.1.7 启动所有会话（StartAllSessions）

函数原型

```
int H5SDKAPI StartAllSessions();
```

输入参数

无

返回

返回值	含义
0	成功
1	失败

用法说明

所有会话的回调类创建完成后，才可以调用此接口，调用后会话将自动连接默认的行情服务器地址

3.1.8 停止所有会话（StopAllSessions）

函数原型

```
int H5SDKAPI StopAllSessions();
```

输入参数

无

返回

返回值	含义
0	成功
1	错误

用法说明

停止所有的会话，断开连接

注意

调用后，所有会话线程均停止工作，此函数调用完成后，相关的会话信息才可以释放。

3.1.9 设置 H5 模板路径（SetH5DataPath）

函数原型

```
void H5SDKAPI SetH5DataPath( const char *path);
```

输入参数

参数名称	参数说明
const char *path	设置 h5 模板数据存放路径，默认为工作路径下的 H5data 目录

返回

无

用法说明

如无特殊需求可以不配置。如果需要配置，请在创建会话前调用。

3.1.10 设置日志存储路径（SetH5LogPath）

函数原型

```
void H5SDKAPI SetH5LogPath( const char *path);
```

输入参数

参数名称	参数说明
const char *path	设置 h5 日志文件存放路径，默认为工作路径下的 H5Log 目录

返回

无

用法说明

如无提速需求可以不配置。如果需要配置，请在创建会话前调用。

3.1.11 设置日志级别等参数路径（SetSdkParamPath）

函数原型

```
void H5SDKAPI SetSdkParamPath( const char *path);
```

输入参数

参数名称	参数说明
const char *path	设置 SdkParamPath.json 的路径+文件名

返回

无

用法说明

如无特殊需求可以不配置。如果需要配置，请在创建会话前调用。

3.2 会话选项接口【SessionOptions】

3.2.1 设置超时时间（SetHeartbeatTimeout）

函数原型

```
virtual int H5SDKAPI SetHeartbeatTimeout(int seconds) = 0;
```

输入参数

参数名称	参数说明
int seconds	● 心跳超时时间，默认由服务器决定（60 秒） ● 心跳超时时间的最小值为 10 秒

返回

返回值	含义
0	设置成功
1	失败

用法说明

心跳超时时间就是 sdk 多久没收到来自行情服务器的报文，超过心跳超时时间则断开连接，sdk 发送心跳的间隔为一半的心跳超时时间。

3.2.2 设置指定下标的 IP 和端口（SetServerIp 和 SetServerPort）

函数原型

```
virtual int H5SDKAPI SetServerIp(const char *ip, size_t index = 0) = 0
virtual int H5SDKAPI SetServerPort(unsigned short port, size_t index = 0) = 0
```

输入参数

由于 SetServerIp 和 SetServerPort 通常一起使用用来设置行情服务器的 ip 和端口，这里一起来说明这两个函数

参数名称	参数说明
const char *ip	● 行情服务器 IP
unsigned short port	● 行情服务器指定端口
size_t index = 0	● 指定序号，从 0 开始递增，默认为 0

返回

返回值	含义
0	设置成功
1	失败

3.2.3 服务端个数获取（GetServerCount）

函数原型

```
virtual size_t H5SDKAPI GetServerCount() = 0;
```

输入参数

无

返回

返回当前会话上级行情服务器的个数，即 SetServerIp 和 SetServerPort 设置的个数。

3.2.4 获得会话对应的服务器地址（GetServerIp 和 GetServerPort）

函数原型

```
virtual const char * H5SDKAPI GetServerIp(size_t index = 0) = 0
virtual unsigned short H5SDKAPI GetServerPort(size_t index = 0) = 0
```

输入参数

参数名称	参数说明
size_t index	服务器序号，表示需要获取的第几个服务器

返回

返回值	含义
const char *	服务器 ip
unsigned short	服务器端口

用法说明

通常情况下，首先使用 GetServerCount 接口获取服务器个数，然后使用 GetServerIp 和 GetServerPort 依次遍历所有的服务器信息。

3.2.5 设置重连间隔（SetReconnintvl）

函数原型

```
virtual int H5SDKAPI SetReconnintvl(int reconnintvl) = 0;
```

输入参数

参数名称	参数说明
int reconnintvl	设置重连间隔，单位为秒，默认为 3 秒

返回

返回值	含义
0	设置成功
1	失败

3.2.6 设置重连次数（SetReconnretries）

函数原型

```
virtual int H5SDKAPI SetReconnretries(int reconnretries) = 0;
```

输入参数

参数名称	参数说明
int reconnretries	设置重连次数，默认为 5 次

返回

返回值	含义
0	设置成功
1	失败

用法说明

通常情况下，此参数保持默认即可，不需要设置，sdk 将会不断尝试连接直到连接成功

3.3 会话接口【Session】

3.3.1 设置加密秘钥和应用标识信息（SetAppInfo）

函数原型

```
virtual int H5SDKAPI SetAppInfo(const char *app_key,const char *app_secret = NULL) = 0;
```

输入参数

参数名称	参数说明
const char *app_key	在金融云开放平台(http://open.hs.net)创建一个应用后获取的 App Key
const char *app_secret	在金融云开放平台(http://open.hs.net)创建一个应用后获取的 App Secret，此参数针对 cloud 用户为必须设置，否则无法登陆

返回

返回值	含义
0	设置成功
1	失败

用法说明

登陆或者注册之前调用，和登陆/注册/修改用户信息接口非线程安全，一般为创建会话时使用使用，**目前 1.0.4.0 及以上版本的 sdk 暂时未兼容老的行情服务器**，当发现行情服务器无法登陆时请尝试创建会话时候选择默认的 app_secret 登录

3.3.2 设置回调类（SetCallback）

函数原型

```
virtual int H5SDKAPI SetCallback(H5SdkCallback *h5SdkCallback) = 0;
```

输入参数

参数名称	参数说明
H5SdkCallback *h5SdkCallback	用户的回调类，用户自定义，是会话使用的基础类，详细信息参考 3.4 小节

返回

返回值	含义
0	设置成功
H5SDK_SET_DECODER_ERROR = 1004	失败

用法说明

该接口必须在 StartSetCallback 之后，StartAllSessions 之前调用。

3.3.3 用户登录（LoginByUser 和 LoginByToken）

函数原型

```
virtual LoginAnsInfo *H5SDKAPI LoginByUser(const char *userName, const char *password, const char *imei = NULL) = 0;
virtual LoginAnsInfo *H5SDKAPI LoginByToken(const char *token, const char *auth_id, const char *imei = NULL)=0;
```

输入参数

参数名称	参数说明
const char *userName	用户名\手机号码\邮箱，需要在 open.hs.net 注册
const char *password	用户登录密码
const char *token	使用用户登录过，LoginAnsInfo 返回的 token，需要和 auth_id 一起使用
const char *auth_id	使用用户登录过，LoginAnsInfo 返回的 auth_id，需要和 token 一起使用
const char *imei	登录设备标识，（可选）

返回

LoginAnsInfo 接口,接口说明如下。

```
class LoginAnsInfo
{
public:
 virtual const char * GetUserName()=0;
 //tocken，断开连接时候可使用 tocken + authId
 virtual const char * GetToken()=0;
 virtual const char * GetAuthId()=0;

 //结果："suc"登陆成功，其他失败
 virtual const char * GetResult()=0;
 virtual const char * GetError()=0;
 //权限信息
 virtual const char * GetAuthString()=0;
 //引用计数
 virtual long  FUNCTION_CALL_MODE AddRef() =0;
 virtual int Release()=0;
};
```

用法说明

通常情况下两个接口登录结果是等价的，如果客户端已经有 token 和 auth_id 可以使用此接口登录，但是 token 和 auth_id 是有有效时间的，有效时间一般为 24 小时，超过有效时间需要使用用户名和密码登录。

LoginAnsInfo 为返回的数据接口，需要调用 `Release` 接口进行内存释放，否则会有内存泄露

3.3.4 创建消息（CreateMessage）

函数原型

```
virtual IHsCommMessage * H5SDKAPI CreateMessage(int businessId, int functionId,
int packetType) = 0
```

输入参数

参数名称	参数说明
`int businessId`	Sdk 用户填入 BIZ_H5PROTO 即可
`int functionId`	创建的消息类型，如 H5SDK_MSG_MARKET_TYPES
`int packetType`	报文类型 REQUEST 通常为请求报文

返回

返回代码类型 IHsCommMessage*接口，具体使用参考示例代码。

使用说明

消息创建和使用是 sdk 与行情服务器的基础，具体使用参考示例代码，这里不做过多解释

3.3.5 同步请求接口（SyncCall）

函数原型

```
virtual int H5SDKAPI SyncCall(IHsCommMessage *request, IHsCommMessage **response,
int milliseconds) = 0;
```

输入参数

参数名称	参数说明
`IHsCommMessage *request`	请求消息
`IHsCommMessage **response`	应答消息

int milliseconds	超时等待时间

返回

返回值	含义
0	成功
H5SDK_DISCONNECT	连接未建立或已断开
H5SDK_SERIALIZATION_ERROR	request 序列化失败，具体原因可调用 request 的 GetLastErrInfo 方法来查看
H5SDK_TIMEOUT	调用超时

用法说明

同步发送阻塞改线程，直到同步请求完成，异步接收回调不会收到异步应答。

3.3.6 异步请求接口（AsyncSend）

函数原型

```
virtual int H5SDKAPI AsyncSend(IHsCommMessage *request) = 0;
```

输入参数

参数名称	参数说明
IHsCommMessage *request	请求消息

返回

返回值	含义
0	成功
H5SDK_DISCONNECT	连接未建立或已断开
H5SDK_SERIALIZATION_ERROR	request 序列化失败，具体原因可调用 request 的 GetLastErrInfo 方法来查看

使用说明

调用此此接口后会在回调类中收到相应的应答包，用户需要在回调类中找到对应的应答包

3.3.7 设置会话使用 best 协议 (SetBestProtocol)

函数原型

```
virtual void H5SDKAPI SetBestProtocol(bool trueORfalse) = 0;
```

输入参数

参数名称	参数说明
bool trueORfalse	true 使用 best 协议，false 使用 h5 协议，默认 false

返回

无

使用说明

与网关之间的通信采用 best 协议，若连接对象为网关，则需要设置会话使用 best 协议。需要在 StartAllSessions()之前调用，默认为 false。

3.3.8 设定行情级别 (SetHqLevel)

函数原型

```
virtual void H5SDKAPI SetHqLevel(long long hqlevel) = 0;
```

输入参数

参数名称	参数说明
long long hqlevel	行情级别掩码值

返回

无

使用说明

用户可设置想获取什么等级的行情信息（level1 or level2，延时还是实时） 默认 level1 实时

如想获取 Level1 延时行情，可调用 SetHqLevel(LEVEL_1 | _DELAY);

仅针对连接网关时有效

3.4 回调类接口【H5SdkCallback】

回调类接口，主要是连接建立，断开连接，登录，接受到异步报文等时间的处理，用户需要继承 H5SdkCallback 接口实现对应事件的处理接口。

3.4.1 连接建立回调（OnConnect）

函数原型

```
virtual void OnConnect(Session *session, const char *peerIp, int peerPort, const char *localIp, int localPort) = 0;
```

输入参数

参数名称	参数说明
Session *session	会话
const char *peerIp	连接成功的服务器 ip
int peerPort	连接成功的服务器端口
const char *localIp	本机 ip
int localPort	本机端口

返回

无

用法说明

通常用户需要在此回调函数中做同步模板，也就是sdk登录（AsyncSdkLogin），具体使用见示例代码

3.4.2 Sdk 登录回调（OnSdkLogin）

函数原型

```
virtual void OnSdkLogin(Session *session) = 0;
```

输入参数

参数名称	参数说明
Session *session	会话

返回

无

用法说明

通常用户 sdk 在此接口中，通知 sdk 同步模板成功，在此接口中业务登录，或者通知外部线程可以调用业务登录接口（见 3.3.3 小节）

3.4.3 收到异步报文回调（OnReceived）

函数原型

```
virtual void OnReceived(Session *session, IHsCommMessage *response) = 0
```

输入参数

参数名称	参数说明
Session *session	会话指针
IHsCommMessage *response	异步应答消息

返回

无

用法说明

OnReceived 主要用于异步请求，主推报文的接受中使用。

注意

Response 消息需要释放否则会引起内存泄露。

3.4.4 连接断开接口（OnClose）

函数原型

```
virtual void OnClose(Session *session, int reason) = 0;
```

输入参数

参数名称	参数说明
Session *session	会话指针

int reason	断开原因，一般为连接断开值为-1

用法说明

用户可在此接口中选择重连，或通知客户手动选择连接

3.4.5 错误信息回调（OnError）

函数原型

```
virtual void OnError(Session *session, int errorNo) = 0;;
```

输入参数

参数名称	参数说明
Session *session	会话指针
Int errorNo	errorNo 目前可能为 H5SDK_SDK_LOGIN_TIMEOUT，表示sdk登录超时

返回

无

用法说明

登录超时。

3.4.6 指定次数无法连接的回调（OnCore）

函数原型

```
virtual void OnCore(Session *session) = 0;
```

输入参数

参数名称	参数说明
Session *session	会话指针

返回

无

用法说明

当重连指定次数后仍然无法连上则回调，收到此回调一般为端口不可用，需要重新连接新地址。

4 注意事项

由于 H5SDK 只有一个工作线程，因此在回调函数中不要做太多耗时的操作，以免造成接收和发送端的积压，导致与服务器的通信断开。

SDK 支持对行情快照的订阅，订阅后将会立刻返回一个最新的行情快照，但是由于返回包包含此次订阅的所有代码，如果请求的代码数目较多将导致此报文的长度过大，建议每次订阅的代码数目不超过 200 只。

行情时间戳格式为 HHMMSSsss，对应小时、分钟、秒、毫秒。如 93912310 表示 9:39:12.310；134512120 表示 13:45:12.120。

行情价格由于浮点数的区间是不连续的，无法表示所有小数，因此行情服务器返回的价格字段均是经过放大之后的值。具体放大倍数以及价格精度根据不同的市场和分类可能会有所不同，可以向行情服务器发送市场分类信息（H5SDK_MSG_MARKET_TYPES）请求获得相应市场以及具体分类的价格放大倍数和价格精度。如请求 XSHG 上海市场的市场分类信息，返回分类“上证 A 股” 放大倍数（H5SDK_TAG_PRICE_SCALE）为 1000，价格精度（H5SDK_TAG_PX_PRECISION）为 2，则表示该分类下的行情价格放大了 1000 倍，价格精度为小数点后两位。

行情快照中部分字段放大倍数：

换手率：10,000 倍

涨跌幅：10,000 倍

委比：10,000 倍

5 参见业务场景及示例代码

5.1 设置环境变量（可选）

通常用于设置 sdk 的日志，数据，配置文件路径。（示例代码）

```
////////////////////////////////////////////////////////////////////////////
 // 2. 设置环境变量
 ////////////////////////////////////////////////////////////////////////////
 //设置 SDK 日志路径
 SetH5DataPath("D:\\");
 //设置 SDK 日志路径
 SetH5LogPath("H5LOG");
 //设置日志级别配置路径
 SetSdkParamPath("SdkParamPath.json");
```

5.2 回调类创建

注意：本示例代码红色部分为必选代码，分别为 SDK 登陆和业务登陆

```
Class H5SdkCallbackImpl: public H5SdkCallback {
public:
   void OnConnect(Session *session, const char *peerIp, int peerPort, const char
*localIp, int localPort) {
       // 已经建立了 Tcp 连接
       puts(__FUNCTION__);

       PRINT(peerIp);
       PRINT(peerPort);
       PRINT(localIp);
       PRINT(localPort);
```

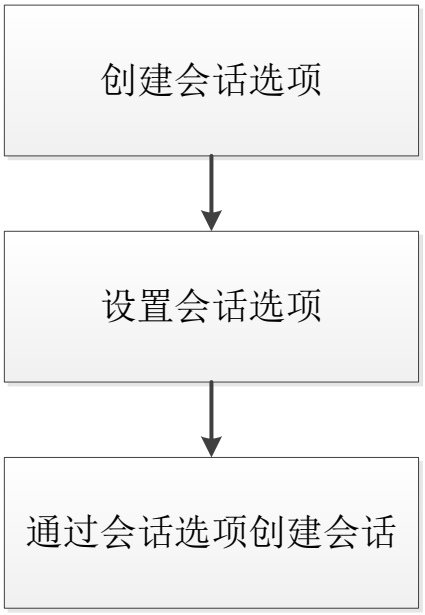
```
        // 发起异步 Sdk 登录
        // 1000 为超时时间
       //Sdk 登陆
        PRINT(session->AsyncSdkLogin(1000));
    }

    void OnSdkLogin(Session *session) {
        // 当 Sdk 登录成功后回调
        puts(__FUNCTION__);

      //需要填入对应的用户名和密码
      LoginAnsInfo *lp=session->LoginByUser("anyone","anyone");
      if (lp && !strcmp(lp->GetResult(),"suc"))
      {
          //登录成功
      }
      else
      {
          puts("异步发起业务登录失败");
      }

    }

        void OnReceived(Session *session, IHsCommMessage *response) {
        // 当会话收到异步应答时会触发该回调
        puts(__FUNCTION__);

        // 判断是否执行成功
        int headErrorNo =
response->GetHead()->GetItem(H5PROTO_TAG_ERROR_NO)->GetInt32();
        if (headErrorNo != 0)
        {
            cout << "headErrorNo = " << headErrorNo << endl;
        }

        const char *userKey =
response->GetHead()->GetItem(H5PROTO_TAG_USER_KEY)->GetString();

        PRINT(userKey);
      int fucid = response->GetFunction();
      switch (fucid)
```

```
    {
    case H5SDK_MSG_SNAPSHOT:
        //
        {
            if (response->GetPacketType() == PUSH)
            {
                //主推处理
            }
            else
            {
                //请求实时行情处理
            }
            break;
        }
    case H5SDK_MSG_MARKET_TYPES: // 市场分类信息
        {
            // 开始解析市场分类信息
            // 因为字段 H5SDK_FINANCE_MIC_GRP 为 sequence，先获取 IGroup 句柄
            IGroup *finance_mic_grp =
response->GetGroup(H5SDK_FINANCE_MIC_GRP);
            // 获取 H5SDK_FINANCE_MIC_GRP 的记录条数
            int finance_mic_grp_cnt = finance_mic_grp->GetRecordCount();
            // 解析每一条记录
            for (int i = 0; i < finance_mic_grp_cnt; i++)
            {
                // 获取记录的句柄
                IRecord *finance_mic_record = finance_mic_grp->GetRecord(i);
                // 根据各个字段的数据类型，调用不同函数获取字段值
                const char *finance_mic =
finance_mic_record->GetItem(H5SDK_TAG_FINANCE_MIC)->GetString();
                const char *finance_name =
finance_mic_record->GetItem(H5SDK_TAG_FINANCE_NAME)->GetString();
                uint32 market_date =
finance_mic_record->GetItem(H5SDK_TAG_MARKET_DATE)->GetInt32();
                uint32 init_date =
finance_mic_record->GetItem(H5SDK_TAG_INIT_DATE)->GetInt32();
                int32 timezone =
finance_mic_record->GetItem(H5SDK_TAG_TIMEZONE)->GetInt32();
                const char *timezone_code =
finance_mic_record->GetItem(H5SDK_TAG_TIMEZONE_CODE)->GetString();
                uint8 dst =
finance_mic_record->GetItem(H5SDK_TAG_DST_FLAG)->GetInt8();

                // 因为 H5SDK_TAG_TYPE_GRP 类型为 sequence，所以先必须获取 IGroup 句柄
```

```
                IGroup *type_grp =
finance_mic_record->GetGroup(H5SDK_TAG_TYPE_GRP);
                // 获取 H5SDK_TAG_TYPE_GRP 的记录条数
                int type_grp_cnt = type_grp->GetRecordCount();
                // 解析每一条记录
                for (int j = 0; j < type_grp_cnt; j++)
                {
                    // 获取记录的句柄
                    IRecord *type_record = type_grp->GetRecord(j);
                    // 根据各个字段的数据类型，调用不同函数获取字段值
                    const char *hq_type_code =
type_record->GetItem(H5SDK_TAG_HQ_TYPE_CODE)->GetString();
                    const char *hq_type_name =
type_record->GetItem(H5SDK_TAG_HQ_TYPE_NAME)->GetString();
                    uint32 px_scale =
type_record->GetItem(H5SDK_TAG_PRICE_SCALE)->GetInt32();
                    int32 px_precision =
type_record->GetItem(HSSDK_TAG_PX_PRECISION)->GetInt32();
                    // 同理可解析余下字段，注意字段类型是 sequence 的解析
                }
            }
        }
        break;

    default:
        {
            //其他业务处理;
        }
    }



      response->Release();

      // 可以在这里发起其他业务请求
  }
  void OnClose(Session *session, int reason) {
      // 当会话连接断开时回调
      puts(__FUNCTION__);
      // 为了实现断开重连，可以在 OnClose 里发起 AsyncConnect 调用
      session->AsyncConnect();
  }


  void OnError(Session *session, int errorNo) {
```

```
        // 当会话出错时回调
        puts(__FUNCTION__);
        // 打印错误号
        PRINT(errorNo);
    }

    void OnCore(Session *session) {
        // 当会话一直无法连上服务端时回调
        puts(__FUNCTION__);
        // 为了实现继续重连，可以在 OnCore 里发起 AsyncConnect 调用
        session->AsyncConnect();
    }
};
```

5.3 会话创建



会话创建的流程

5.3.1 会话选项创建

SessionOptions *session_opt = CreateSessionOptions();

//固定调用形式；返回空指针表示失败；非空指针表示成功，成功后，可设置会话选项

//session_opt ->AddRef();

//固定调用，创建成功时，必须调用，增加引用计数

5.3.2 设置会话选项

设置会话的心跳超时时间为 60 秒（该属性必须设置，心跳的时间间隔可根据实际需要填入）

session_opt->SetHeartbeatTimeout(60);

设置所连接的后台服务器的地址和端口，可以设置多个地址和端口（根据实际使用的后台部署），即调用这 2 个函数多次

session_opt->SetServerIp(“192.168.90.90”);

session_opt->SetServerPort(80);

session_opt->SetServerIp(“192.168.90.91”,1)

session_opt->SetServerPort(8080,1);

连接规则：按照设置的顺序，依次连接，直到成功连接上某一地址（此时不再连接别的地址），如果中途连接发生断开，会重复之前的连接逻辑

5.3.3 通过会话选项创建会话

Session *session = CreateSession(session_opt);

成功时，返回非空指针

session->AddRef();

固定调用，创建成功时，必须调用，增加引用计数

5.3.4 设置会话是否使用 best 协议

session->SetBestProtocol(true); //设置该会话使用 best 协议

5.3.5 StartSetCallback

固定调用，必须在所有的会话创建完成以后，且在设置任意一个会话的回调之前

5.3.6 设置会话的回调类实例

创建回调类的实例

my_callback_class *my_cb = new my_callback_class;

设置该会话的回调

session-> SetCallback(my_cb);

5.3.7 StartAllSessions

固定调用，在所有的会话的回调设置完成以后调用，该调用之后，H5SDK 便开始工作，进入事件和消息的分发状态中，以后所有事情需要在回调中处理

5.3.8 完整实例代码

```
////////////////////////////////////////////////////////////////////////////
 // 3. 会话创建
 //示例代码为创建两个会话
 ////////////////////////////////////////////////////////////////////////////
   // 创建会话选项 1
  SessionOptions *sessionOptions[2];
  Session *session[2];
#if 1
  //创建会话 1
   sessionOptions[0] = CreateSessionOptions();
    sessionOptions[0]->AddRef();

   // 设置会话选项 1
    sessionOptions[0]->SetHeartbeatTimeout(3);

 //设置两组 ip 和端口
    sessionOptions[0]->SetServerIp("121.43.71.217");
    sessionOptions[0]->SetServerPort(9999);

  sessionOptions[0]->SetServerIp("121.43.71.218",1);
  sessionOptions[0]->SetServerPort(9999,1);

   // 创建会话
 session[0] = CreateSession( sessionOptions[0]);
   session[0]->AddRef();
 //
 session[0]->SetAppInfo("app_key","app_secret");
#endif

#if 1
 //创建会话 2
 sessionOptions[1] = CreateSessionOptions();
 sessionOptions[1]->AddRef();

 // 设置会话选项 2
 sessionOptions[1]->SetHeartbeatTimeout(3);

 //设置两组 ip 和端口
 sessionOptions[1]->SetServerIp("121.43.71.217");
```

```
 sessionOptions[1]->SetServerPort(9999);

 sessionOptions[1]->SetServerIp("121.43.71.218",1);
 sessionOptions[1]->SetServerPort(9999,1);

 // 创建会话
 session[1] = CreateSession( sessionOptions[1]);
  session[1]->SetBestProtocol(true); //设置会话 2 使用 best 协议
 session[1]->AddRef();
 session[1]->SetAppInfo("app_key","app_secret");
#endif

   // 开始为所有会话设置回调
   StartSetCallback();

   // 设置回调
   H5SdkCallbackImpl h5SdkCallbackImpl;
 //两个会话可使用同一个回调类，也可以使用不同的回调类
 for (int i=0;i<2;i++)
 {
     session[i]->SetCallback(&h5SdkCallbackImpl);
 }
   // 启动所有会话
   StartAllSessions();

   // 等待回调触发
   getchar();
```

5.4 测速接口

5.4.1 测速回调函数

```
class MyVelocimetryCallback: public VelocimetryCallback
{
public:
   int H5SDKAPI OnVelocimetrySucceed(Endpoint *endpoint, int delay, int load)
   {
       cout << __FUNCTION__ << endl;
```

```
        PRINT(endpoint->userData);
        PRINT(delay);
        PRINT(load);

        return 0;
    }

    int H5SDKAPI OnVelocimetryFail(Endpoint *endpoint, VelocimetryErrorNo errorNo)
    {
        cout << __FUNCTION__ << endl;

        PRINT(endpoint->userData);
        PRINT(errorNo);

        return 0;
    }

    int H5SDKAPI OnVelocimetryComplete()
    {
        cout << __FUNCTION__ << endl;
        return 0;
    }
};
```

5.4.2 测速代码

```
// 演示测速
    Endpoint endpoint[3];

    strcpy(endpoint[0].ip, "121.43.71.217");
    endpoint[0].port = 9999;
    strcpy(endpoint[0].userData, "测试服务器");

    strcpy(endpoint[1].ip, "127.0.0.1");
    endpoint[1].port = 9999;
    strcpy(endpoint[1].userData, "端口未打开");

    strcpy(endpoint[2].ip, "192.168.94.200");
    endpoint[2].port = 9999;
    strcpy(endpoint[2].userData, "不可达服务器");

    MyVelocimetryCallback myVelocimetryCallback;
```

```
Velocimetry *velocimetry = CreateVelocimetry();

velocimetry->AddRef();
velocimetry->Start(endpoint, 3, &myVelocimetryCallback, 3000);

getchar();

velocimetry->Stop();
velocimetry->Release();

return 0;
```

5.5 构造请求报文

以请求市场分类信息为例，消息定义《H5 行情服务协议(修订版).xls》，构建消息时，必须参考消息定义格式，按照字段类型，正确设置字段值

注意：必须要有 Session 句柄

5.5.1 创建一条请求报文

IHsCommMessage *request = session->CreateMessage(BIZ_H5PROTO, H5SDK_MSG_MARKET_TYPES, REQUEST);

使用 Session 句柄的 CreateMessage 创建消息

BIZ_H5PROTO：使用附录三时，此处固定为该值

H5SDK_MSG_MARKET_TYPES：上图中，**消息宏**处的值，表明是市场分类信息

REQUEST：上图中，**类型名称**处的值，表明是请求

5.5.2 设置字段值

特别注意类型是 sequence 的设置方法，缩进格式表明多个字段隶属于该 sequence

IGroup *group = request->SetGroup(H5SDK_FINANCE_MIC_GRP);

对于 sequence 字段，使用 SetGroup 函数，获得 IGroup 句柄，之后再添加记录

添加第 1 条记录

IRecord *record = group->AddRecord();

该调用表示，在 finance_mic_grp 下添加一条记录（finance_mic_grp 是 sequence，表示它可包含一条或多条记录）

对添加的第 1 条记录设置字段值，因为该字段类型为 bytevector，所以调用 SetString 函数，传入字符串值

record->GetItem(H5SDK_TAG_FINANCE_MIC)->SetString("SS");

添加第 2 条记录

record = group->AddRecord();

对添加的第二条记录设置字段值

record->GetItem(H5SDK_TAG_FINANCE_MIC)->SetString("SZ");

5.5.3 示例代码

```
// 创建请求报文，请求"市场分类信息"
IHsCommMessage *request = session[0]->CreateMessage(BIZ_H5PROTO,
H5SDK_MSG_MARKET_TYPES, REQUEST);
// 根据 H5SDK_FINANCE_MIC_GRP 字段类型为 sequence，创建 IGroup 句柄
IGroup *group = request->SetGroup(H5SDK_FINANCE_MIC_GRP);
// 为 H5SDK_FINANCE_MIC_GRP 添加第一条记录
IRecord *record = group->AddRecord();
// 为第一条记录的字段 H5SDK_TAG_FINANCE_MIC 设置值"SS"，H5SDK_TAG_FINANCE_MIC 字段类型为 bytevector，故调用 SetString，传入字符串值
record->GetItem(H5SDK_TAG_FINANCE_MIC)->SetString("SS");
// 为 H5SDK_FINANCE_MIC_GRP 添加第二条记录
record = group->AddRecord();
// 为第二条记录的字段 H5SDK_TAG_FINANCE_MIC 设置值"SZ"
record->GetItem(H5SDK_TAG_FINANCE_MIC)->SetString("SZ");
// 发送消息同步或异步均可
session[0]->AsyncSend(request);
// 释放消息
request->Release();
```

5.6 发送请求并解析应答

5.6.1 使用同步或者异步接口发送

异步发送：session[0]->AsyncSend(request);

同步发送： session[0]->SyncCall(request,&ansMsg,3000);

5.6.2 接受报文

同步调用在出参中返回应答包，而异步接口需要在回调类函数中去使用

5.6.3 完整示例代码

```
////////////////////////////////////////////////////////////////////////////
#if 0
 //异步发送在回调函数中处理，
 session[0]->AsyncSend(request);//(见回调类接收函数)
#else
//同步发送在此函数中处理
 IHsCommMessage *ansMsg =NULL;
 int ret = session[0]->SyncCall(request,&ansMsg,3000);
 if (ret ==0 && ansMsg)
 {
     // 开始解析市场分类信息
     // 因为字段 H5SDK_FINANCE_MIC_GRP 为 sequence，先获取 IGroup 句柄
     IGroup *finance_mic_grp = ansMsg->GetGroup(H5SDK_FINANCE_MIC_GRP);
     // 获取 H5SDK_FINANCE_MIC_GRP 的记录条数
     int finance_mic_grp_cnt = finance_mic_grp->GetRecordCount();
     // 解析每一条记录
     for (int i = 0; i < finance_mic_grp_cnt; i++)
     {
         // 获取记录的句柄
         IRecord *finance_mic_record = finance_mic_grp->GetRecord(i);
         // 根据各个字段的数据类型，调用不同函数获取字段值
         const char *finance_mic =
finance_mic_record->GetItem(H5SDK_TAG_FINANCE_MIC)->GetString();
         const char *finance_name =
finance_mic_record->GetItem(H5SDK_TAG_FINANCE_NAME)->GetString();
         uint32 market_date =
finance_mic_record->GetItem(H5SDK_TAG_MARKET_DATE)->GetInt32();
         uint32 init_date =
finance_mic_record->GetItem(H5SDK_TAG_INIT_DATE)->GetInt32();
         int32 timezone =
finance_mic_record->GetItem(H5SDK_TAG_TIMEZONE)->GetInt32();
         const char *timezone_code =
finance_mic_record->GetItem(H5SDK_TAG_TIMEZONE_CODE)->GetString();
         uint8 dst = finance_mic_record->GetItem(H5SDK_TAG_DST_FLAG)->GetInt8();

         // 因为 H5SDK_TAG_TYPE_GRP 类型为 sequence，所以先必须获取 IGroup 句柄
         IGroup *type_grp = finance_mic_record->GetGroup(H5SDK_TAG_TYPE_GRP);
         // 获取 H5SDK_TAG_TYPE_GRP 的记录条数
```

```
        int type_grp_cnt = type_grp->GetRecordCount();
        // 解析每一条记录
        for (int j = 0; j < type_grp_cnt; j++)
        {
            // 获取记录的句柄
            IRecord *type_record = type_grp->GetRecord(j);
            // 根据各个字段的数据类型，调用不同函数获取字段值
            const char *hq_type_code =
type_record->GetItem(H5SDK_TAG_HQ_TYPE_CODE)->GetString();
            const char *hq_type_name =
type_record->GetItem(H5SDK_TAG_HQ_TYPE_NAME)->GetString();
            uint32 px_scale =
type_record->GetItem(H5SDK_TAG_PRICE_SCALE)->GetInt32();
            int32 px_precision =
type_record->GetItem(HSSDK_TAG_PX_PRECISION)->GetInt32();
            // 同理可解析余下字段，注意字段类型是 sequence 的解析
        }
    }
    ansMsg->Release();
 }
 else
 {
    printf("同步请求错误 ret %d",ret);
 }
```