S-SDK 开发手册

V4.0.0.0

易通星云（北京）科技发展有限公司

官网地址：www.kaifakuai.com

官网 QQ 群：445880047

微信公众号：开发快

文档修改记录

版本号	发布日期	描述	作者
V1.6.0	2016-01-24	创建文档	huangminghai
V1.6.1	2016-04-19	修复因混淆问题导致的 bug	huangminghai
V1.6.2	2016-04-29	修复在断网情况下手动重连不成功的问题	huangminghai
V1.6.3	2016-05-06	调整 S-SDK 默认平台标识	huangminghai
V1.6.4	2016-05-27	1、当传送非法格式的文件 json 过来时，SDK 会异常离线的 bug。 2、修复当接收到非法的文件名时，SDK 会异常离线的 bug;	huangminghai
V1.7.0	2016-07-07	1、 新增新版负载均衡模块 2、 publishToGroup 方法。 3、 按照需求约定统一部分 API。 4、 取消自动重连机制。	huangminghai
V3.1.7.1	2016-07-20	1、 点对点消息 qos 级别无效	huangminghai
V3.3.0.0	2016-10-19	原有文件 FileTo、上传、下载、删除接口添加带校验功能。修复部分 bug.	huangminghai
V3.3.0.1	2016-10-28	UID 传递非 34 位长度后，用户依然能够验证通过的隐含问题	huangminghai
V3.3.1.0	2016-12-09	文件传输新增解析服务器返回的验证信息及重新定义文件错误码。	Huangminghai
V4.0.0.0	2017-02-20	全新改版，异步接口,增加 disconver 发现功能。	Huangminghai

1. 简介

为第三方 Java-SDK 应用提供相关 API 调用。通过 SDK 中的相关功能接口实现即时消息、文件传输、用户/群管理功能

这是整合了消息接口，用户/群管理接口和文件接口在一起的 SDK，隐藏了单个 SDK 的一些逻辑细节，简化了 SDK 接口个数和使用的复杂度。

1.1缩略词

AppKey	应用标识码。当开发者需要为一款智能产品开发应用（包括终端与设备）时，申请生成，应用开发时需要填入
SecretKey	应用安全识别码，在调用一些管理接口时要填入
EtFactory	SDK 中的创建工厂
IContext	SDK 中应用程序全局上下文参数存储对象
UID / ClientId	平台标识码，在整个云平台唯一。应用开发时需要填入
ISDK	SSDK 标准 API 接口
QoS	消息质量控制。值域[0,1,2]。SDK 中，默认为 1。用户可根据业务场景设置。数值越小，消息发送越快。
ICallback	回调接口，用于通知用户本次操作是否成功
FileCallBack	文件操作回调接口
IFileReceiveListener	用户文件传输处理接口
IUserStatusListener	用户状态接口 处理关于用户的所有状态定义接口

1.2通用说明

默认消息内容字符处理编码均采用 UTF-8 编码。
HTTP 协议采用 POST 方式请求。
SDK 最低要求 JDK1.7，推荐 JDK1.7+

请注意

1、请不要在任何回调函数中做耗时操作，因为那将影响 SDK 处理消息的速度。按照设计思想，SDK 只提供最底层的消息通信及访问服务，对于消息内容的处理，应由用户应用程序进行存储或线程池+队列 处理。

2. 接口协议

2.1 接口功能描述

本 SDK 具备的功能调用：

消息发送，主题订阅及取消，手动重连，带验证功能文件传输，用户、群管理功能，用于服务器端通信开发。

3. 接口说明

3.1. 环境依赖

3.1.1. Jar 包依赖

SDK 依赖 SLF4J 库，所以，如果您的项目不是 maven 工程，则可以在 slf4J 官网(http://www.slf4j.org)下载最新的库即可。如果是 maven，则在您的 pom.xml 中，加入如下依赖：

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.13</version>
</dependency>
```

示例代码 1

如您想看到 SDK 日志输出，可根据您的项目采用哪种日志实现，自行配置即可。

以 Log4j 为日志实现库为例：

```
在您的 pom.xml 文件中引入以下依赖.
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.13</version>
</dependency>

<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.13</version>
        <scope>provided</scope>
</dependency>

<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version> 1.2.17</version>
</dependency>
```

示例代码 2

在您的项目日志配置文件(xml 或 properites)中，粘贴如下日志记录配置：

```
<!-- 省略其他日志配置 -->
<logger name="com.beidouapp.et" additivity="false">
        <level value="debug" />
        <appender-ref ref="console" />
</logger>
```

示例代码 3

3.2. 初始化环境

EtFactory 工厂，用于创建客户端相关服务。主要用于获得 IContext 应用上下文接口、IWeb 注册接口及 ISDK 业务接口对象。

用户首先需要实例化 EtFactory 工厂，通过该工厂，获得全局参数上下文接口对象 IContext。

IContext 接口提供各种参数配置。用户只需设置 AppKey、SecretKey、UserName，服务器域名或 IP 及端口号，其他默认即可正常运行。

用户通过 EtFactory 工厂调用 createSDK (IContext ctx)方法，传递 IContext 上下文和所需的模块类型即可获取服务管理器 ISDK 对象。

示例代码：

```
// 创建 ET 工厂，通过工厂，获取全局上下文 IContext 和 EtManager 服务管理器.
EtFactory factory = new EtFactory ();
```

示例代码 4

3.3.配置参数说明

3.3.1. 参数说明

（1） IContext 全局上下文

IContext 全局上下文接口，用于初始化配置各种参数服务。例如 AppKey、SecureKey、UserName、消息质量等级 QoS、keepAlive 参数等。
在同一个 Factory 中多次获取 IContext 为同一对象。
样例代码：

```
private static final String APP_KEY      = "iLink 云平台应用标识码";
private static final String SECURE_KEY = "iLink 应用安全识别码";
private static final String UID    ="iLink 平台标识码（系统唯一，相当于账号）";
public static final String DOMAIN = "iLink  平台负载均衡服务器域名";
public static final int PORT = iLink  平台负载均衡服务器端口;

// 获取并配置上下文.
IContext etContext = factory.createContext ().setAppKey (APP_KEY).setSecretKey
(SECURE_KEY).setUserName(UID)  .setServerDomain(DOMAIN).setServerPort(POR
T);
```

示例代码 5

IContext 中更多参数，参见如下表格.

属性	数据类型	必填	描述内容	备注
AppKey	String	**是**	应用标识码，当开发者需要为一款智能产品开发应用（包括终端与设备）时，申请生成，应用开发时需要填入	
SecretKey	String	**是**	应用安全识别码，在调用一些管理接口时要填入	
UID	String	**是**	平台标识码（系统唯一，相当于账号），应用开发时需要填入，长度 34 位	UserName

ServerDomain	String	**是**	服务器域名或 IP	
ServerPort	Integer	**是**	服务器端口	
KeepAlive	Short	否	心跳时间	默认 60 秒
CleanSession	Boolean	否	是否保存离线消息. true：保存离线消息(默认)，false：删除离线消息.	默认 true
DefaultQos	Integer	否	QoS 消息质量控制级别. 值域[0,1,2]	默认为 1

（2） ISDK API 接口

为客户端提供各种服务功能。例如发布订阅、点对点消息、群/用户管理和文件传输功能。推荐一个应用程序只存在一个 ISDK。

注意：

1、用户应该通过 ISDK 进行连接服务器和释放资源。 EtManager 中的 connect()， destroy()方法对所有资源进行总体控制。

2、用户调用 destroy()方法后,如需再次使用，请重新实例化。

样例代码：

```
ISDK sdk = factory.create (etContext);
```

示例代码 6

（3） IReceiveListener 消息监听器.

此监听器为接口类型。当 IM 接受到服务器推送过来的数据时，会通过调用 onMessage(String topic, byte[] payload)通知客户端。

如果与服务器断开，则会触发 connectionLost (int reasonCode, Throwable cause)方法执行,用户可通过 cause 中抛出的异常码确认原因，具体参见异常编码表.

提示：

1、IReceiveListener 必须设置，且不应该在此方法中处理耗时操作，否则导致 SDK 消息吞吐量下降、处理超时等问题发生。

2、用户在使用 ISDK 接口发送消息给服务端时,会通过用户传入的 ICallback 通知用户，即 onSuccess()或 onFailer()方法。其意图只是告诉客户端是否将消息成功发送给服务器，并不包含服务器端的业务处理过程及结果。

3、用户真实的请求结果，是服务端接受到消息并处理后，主动推送到初始化 ISDK 中设置的 receiveListener.onMessage()方法中，通知客户端。

样例代码：

```
// 初始化代码略。
//… …
ISDK sdk = factory. createSDK (etContext);
// 设置消息服务监听器，接收服务器推送过来的消息
sdk. setCallback (new IReceiveListener ()
{
  @Override
  public void onMessage (String topic, byte[] payload)
  {
    // TODO:接收服务器端推送的消息，进行消息处理. SDK 不做消息内容存储和转义，如果消息量巨大或存在耗时操作，推荐应用程序进行缓存和多线程处理。
  }

  @Override
  public void connectionLost (int reasonCode, Throwable cause)
  {
   // 与服务器异常断开连接时触发. 通过获取 cause 中的异常码，根据您的业务需求进行相应处理.
  }
});
```

示例代码 7

（4） 发现服务器

发现可以被连接的服务器设备。执行后，会异步返回一个服务器信息对象。用户使用此对象进行连接。

```
sdk.discoverServers(5000, new ICallback<EtServerInfo>() {
                @Override
                public void onSuccess(EtServerInfo etServerInfo) {
                    System.err.println(String.format("onSuccess return EtServerInfo = %s", etServerInfo.toString()));
                }

                @Override
                public void onFailure(Throwable value) {

                }
            });
```

示例代码 8

3.4. 连接

3.4.1. 方法签名

```
/**
 * 连接服务器.
 */
public void connect(EtServerInfo serverInfo, short keepAlive, boolean isCleanSession, int timeout, ICallback<Void> callback);
```

示例代码 9

3.4.2. 输入参数说明

参数名称	必填	类型	描述
serverInfo	是	EerverInfo	服务器信息对象。用户调用发现函数获得。
keepAlive	是	short	心跳时间(秒)。值域范围[15, 300]
isCleanSession	是	boolean	是否清除用户数据。True：清除，false：不清除。
timeout	是	int	首次连接超时时间(秒)。值域范围[5, 60]
callback	是	ICallback	消息发送是否成功的回调(必填)。

3.4.3. 代码示例

```
// 连接 iLink 服务器
sdk.connect(serverInfo, (short) 15, true, 8, new ICallback<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            System.out.println("成功");
        }

        @Override
        public void onFailure(Throwable ex) {
            System.err.println(ex);
        }
    });
```

示例代码 10

3.5. 断开与服务器的连接

3.5.1. 方法签名

```
/**
 * 断开与服务器的连接.<br/>
 *
 * @param callback 消息发送是否成功的回调(必填).
 */
public void disconnect(ICallback<Void> callback);
```

示例代码 11

断开与服务器的连接。成功后，当前连接状态将会变成 DISCONNECTED 状态。断开服务器链接后，请不要再调用其他接口，这样会导致部分接口在回调函数中报 CONNACK_OFFLINE 的 EtRuntimeException 或其他类型异常.

断开连接后，可以通过调用 reconnect () 进行重新链接。

注意：如果调用 reconnect ( )方法，只是通知 SDK 进行重新连接，但因为网络等其他因数的原因，可能会连接失败。

3.5.2. 输入参数说明

参数名称	必填	类型	描述
callback	是	ICallback	消息发送是否成功的回调(必填)。

3.5.3. 代码示例

```
private void onDisconnect() {
        im.disconnect(new ICallback<Void>() {
            @Override
            public void onSuccess(Void value) {
                System.out.println("disconnect success");
            }

            @Override
            public void onFailure(Throwable value) {
                System.err.println("disconnect failure");
                value.printStackTrace();
            }
        });
```

示例代码 12

3.6. 获取指定用户状态

3.6.1. 方法签名

```
/**
 * 获取指定用户状态.
 *
 * @param userId     用户 ID.
 * @param callback  消息发送是否成功的回调(必填).
 */
public void getUserState(String userId,   ICallback<EtMsg> callback);
```

示例代码 13

此功能仅将请求发送给服务端，服务端接收成功或失败，会通过 callback 通知用户。
此接口需要用户在 ISDK 接口中设置 IUserStatusListener 监听器。

3.6.2. 输入参数说明

参数名称	必填	类型	描述
userId	是	**String**	用户 ID，必填，否则抛异常
callback	是	**ICallback**	消息发送是否成功的回调(必填)。必填，否则抛 参数

			为 Null 异常

3.6.3. 代码示例

```
//需先设置用户状态监听器.
sdk.setUserStatusListener(new IUserStatusListener() {
                public void concernOnlineStatus(String userId, String statusCode) {
                    // userId          ：关注的用户 ID，
                    // statusCode    ：关注的用户 ID 的在线状态. 0:离线, 1:在线
                    System.out.println(String.format("userId=%s,  status=%s",  userId,
statusCode));
                }
            });

sdk. getUserState ("34 位用户 UID",    new ICallback< Void>() {
                    public void onSuccess(Voidvalue) {
                        System.out.println("peerState 34 位用户 UID onsuccess");
                    }

                    public void onFailure(Throwable value) {
                        value.printStackTrace(System.err);
                    }
```

示例代码 14

3.7. 发布消息

3.7.1. 方法签名

```
/**
 * 发布消息.
 * 其中 callback 为空时，抛 EtRuntimeException.
 *
 * @param topic      消息主题.
 * @param payload    消息内容字节数组形式.
 * @param qos        消息服务质量等级枚举.
 * @param callback   发布消息是否成功的结果回调(必填).
 */
public void publish(String topic, byte[] payload, QosLevelEnum qos, ICallback<Void> callback);
```

示例代码 15

消息长度提示
一个中文汉字=3 个字节(UTF-8)，否则消息发送失败。

3.7.2. 输入参数说明

参数名称	必填	类型	描述
topic	是	**String**	用户需要发布的主题名称。必填，否则抛 PARAM_NULL_OR_EMPTY 异常。
content	是	**String/ byte[]**	发布的主题内容。此内容支持 String 及 byte[]两种类型，必填
callback	否	**ICallback**	同上。必填。
qos	否	**QosLevelEnum**	消息质量控制等级。

3.7.3. 代码示例

```
sdk.publish(" 主 题 ", " 内 容 ".getBytes(), QosLevelEnum.QoS_1, new ICallback<Void>() {
        @Override
        public void onSuccess(Void v) {
                System.out.println(String.format("publish(%s, %d)成功. threadName=%s", TOPIC, QosLevelEnum.QoS_1.getCode(),
        }

        @Override
        public void onFailure(Throwable ex) {
                ex.printStackTrace();
        }
```

示例代码 16

3.8. 订阅消息

3.8.1. 方法签名

```
    /**
     * 订阅消息.
     *
     * @param topic       消息主题.
     * @param qos         消息质量.
     * @param callback  消息发送是否成功的回调(必填).
     */
    public void subscribe(String topic, QosLevelEnum qos, ICallback<EtAck> callback);
```

示例代码 17

3.8.2. 输入参数说明

参数名称	必填	类型	描述
topic	是	**String**	topic：用户需要订阅的主题名称。必填。

qos	是	**QosLevelEnum**	消息质量控制等级。
callback	是	**ICallback**	同上。必填。

3.8.3. 代码示例

```
sdk.subscribe(TOPIC, QosLevelEnum.QoS_1, new ICallback<EtAck>() {
                @Override
                public void onSuccess(EtAck etAck) {
                    System.out.println(String.format("subscribe(%s,  %d) 成 功 .
threadName=%s", TOPIC, QosLevelEnum.QoS_1.getCode(),));
                }

                @Override
                public void onFailure(Throwable ex) {
                }
            });
```

示例代码 18

3.9. 取消消息订阅

3.9.1. 方法签名

```
    /**
     * 取消订阅.
     *
     * @param topic       消息主题.
     * @param callback  消息发送是否成功的回调(必填).
     */
    public void unsubscribe(String topic, ICallback<Void> callback);
```

示例代码 19

取消已订阅过的 topic。不需要关系 topic 是否有过订阅。

3.9.2. 输入参数说明

参数名称	必填	类型	描述
topic	是	**String**	topic：用户需要订阅的主题名称。必填。
callback	是	**ICallback**	同上。必填。

3.9.3. 代码示例

```
sdk.unsubscribe(TOPIC, new ICallback<Void>() {
                @Override
                public void onSuccess(Void v) {
                    System.out.println(String.format("unsubscribe(%s)  成  功  .
threadName=%s", TOPIC, Thread.currentThread().getName()));
                    cdl4.countDown();
                }

                @Override
                public void onFailure(Throwable ex) {
                    cdl4.countDown();
                }
            });
```

示例代码 20

3.10. 点对点发送消息

3.10.1. 方法签名

```
    /**
     *  点对点发送消息.<br/>
     *  可能会抛出 EtRuntimeException 异常.
     *
     * @param userId     用户 Id.
     * @param payload    消息内容.
     * @param callback  消息发送是否成功的回调(必填).
     */
    public void chatTo(String userId, byte[] payload, ICallback<Void> callback);
```

示例代码 21

用户可以进行点对点的消息发送。

3.10.2. 输入参数说明

参数名称	必填	类型	描述

userId	是	**String**	接收用户 ID
content	是	**String/ byte[]**	同上。必填
callback	是	**ICallback**	同上。必填

3.10.3. 代码示例

```
sdk.chatTo(34 位 UID, msg.getBytes(), new ICallback<Void>() {
                @Override
                public void onSuccess(Void v) {
                    System.out.println(String.format("chatTo(%s,  %s)  成  功  .
threadName=%s", 34 位 UID, msg, Thread.currentThread().getName()));
                }

                @Override
                public void onFailure(Throwable ex) {
                    ex.printStackTrace();
                }
            });
```

示例代码 22

3.11. 手动重连

3.11.1. 方法签名

```
/**
 * 手动重连.
 */
public void reconnect();
```

示例代码 23

提供断开连接后，再次重连服务器功能。此处断开连接表示物理网络断开，比如用户切断 WIFI 网络，拔掉网线，3G 网络无信号等链路层断开。在断开连接后，请不要在尝试 IM 中其他消息发送方法。

重连提示

1、请注意此方法仅执行重连操作，如果网络未恢复的情况下执行，则依然会提示连接失败。

2、如果断网情况下，用户反复循环，建议每次至少休眠 10 秒以上。

3.11.2. 输入参数说明

无.

3.11.3. 代码示例

以下示例代码是通过手动重连方法来实现断网自动重连功能的简单实现。

```
//覆盖 IReceiveListener 中的 connectionLost 方法
@Override
        public void connectionLost(int reasonCode, Throwable cause) {
            if (EtExceptionCode.SYS_KICK == reasonCode) {
                return;
            }
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            sdk.reConnect();
        }
```

示例代码 24

3.12. 获取当前连接状态

3.12.1. 方法签名

```
/**
 * 获取当前连接状态<br/>
 * 0:已连接; 1:正在连接; 2:正在断开; 3:已断开.
 *
 * @return
 */
public ConnectStateEnum getConnectionStatus();
```

示例代码 25

获取当前用户连接状态。同步方法，返回值为 ConnectStateEnum 枚举，值域范围[0：已连接(CONNECTED)、1：正在连接(CONNECTING)、2：正在断开(DISCONNECTING)、3：已断开(DISCONNECTED)]。

3.12.2. 输入参数说明

无.

3.12.3. 代码示例

略

3.13. 销毁连接

3.13.1. 方法签名

```
/**
 * 销毁连接.
 */
public void destroy();
```

示例代码 26

同步方法。结束 SDK 生命周期时，调用 destroy()进行资源释放。

3.13.2. 输入参数说明

无.

3.13.3. 代码示例

```
public void destroy() {
        im.destroy();
}
```

示例代码 27

3.14. 向服务器主动获取离线消息

3.14.1. 方法签名

```
/**
 * 向服务器主动获取离线消息.<br/>
 * 此接口调用后，服务器会将离线消息推送到客户端。如果用户离线后，需要再次调用此接口获取。用户在线时，多次调用此方法时，服务器忽略.
 * @param callback 消息发送是否成功的回调(必填).
 */
public void requestOfflineMessage(ICallback<Void> callback);
```

示例代码 28

当用户登录成功后，可根据自己的业务调用此方法，调用后，服务器会将当前用户收到的离线消息推送过来。

3.14.2. 输入参数说明

参数名称	必填	类型	描述
callback	是	**ICallback**	同上。必填

3.14.3. 代码示例

```
//例如用户登录成功，立即获取离线消息。
sdk.requestOfflineMessage(new ICallback<Void>() {
                @Override
                public void onSuccess(Void v) {

System.out.println(String.format("~~~~~~~~~requestOfflineMessage() 成 功 . threadName=%s", Thread.currentThread().getName()));
                }

                @Override
                public void onFailure(Throwable ex) {
                }
        });
```

示例代码 29

3.15. 用户状态订阅/取消订阅

3.15.1. 方法签名

```
/**
 * 用户状态订阅.
 *
 * @param userId     需要关注状态的发布者 ID.
 * @param callback  消息发送是否成功的回调.
 */
public void subUserState(String userId, ICallback<Void> callback);
/**
 * 取消用户状态订阅.
 *
 * @param userId     需要取消关注状态的发布者 ID.
 * @param callback  消息发送是否成功的回调.
 */
public void unSubUserState(String userId, ICallback<Void> callback);
```

示例代码 30

用户状态订阅、取消用户状态订阅接口，能够通知服务器，当所关注的 userId 上、下线时，服务器将其关注者的状态推送给自己。此接口的使用，需要在 ISDK 中设置 setUserStatusListener (IUserStatusListener userStatusListener)方法。

3.15.2. 输入参数说明

参数名称	必填	类型	描述
userId	是	**String**	接收用户 ID
callback	是	**ICallback**	同上。选填

3.15.3. 代码示例

```
//初始化时设置.
sdk.setUserStatusListener(new IUserStatusListener() {
                @Override
                public void concernOnlineStatus(String userId, String statusCode) {
                        System.out.println(String.format("userId=%s,          status=%s",
userId, statusCode));
                }
        });

//业务调用中.
//订阅状态.
sdk. subUserState ("用户 ID", new ICallback<Void>() {
                @Override
                public void onSuccess(Void value) {
                        System.out.println("stateSubscribe onsuccess");
                }

                @Override
                public void onFailure(Throwable value) {
                        value.printStackTrace(System.err);
                }
        });

//取消指定用户的状态.
sdk. unsubUserState ("用户 ID ", new ICallback< Void >() {
                @Override
                public void onSuccess(Void value) {
                        System.out.println("stateUnsubscribe onsuccess");
                }

                @Override
                public void onFailure(Throwable value) {
                        value.printStackTrace(System.err);
                }
        });
```

示例代码 31

3.16. 主动发送文件

文件传输模块主要是负责与 iLink 云平台进行 文件传输服务。用户通过它可以进行文件发送，上传，下载功能。

案例参见附页代码.

3.16.1. 方法签名

```
/**
 * 主动发送文件.
 *
 * @param receiverId      接收文件的用户 ID.
 * @param fileFullPath    文件全路径名.
 * @param desc            文件描述信息.
 * @param IFileCallBack   文件操作回调接口.
 */
public void fileTo(String receiverId, String fileFullPath, String desc, IFileCallBack IFileCallBack);
```

示例代码 32

3.16.2. 输入参数说明

参数名称	必填	类型	描述
receiverId	是	**String**	接收文件的用户 ID.
fileFullName	是	**String**	文件全路径名.
desc	是	**String**	文件描述信息.
callback	是	**FileCallBack**	文件监听器.

3.16.3. 代码示例

```
//设置 File 接收监听器.
sdk.setFileReceiveListener(new IFileReceiveListener() {
                @Override
                public void onReceived(String senderId, String documentInfoJson) {
                    System.out.println("~~~~~~~~~File.onReceived(senderId="  +
senderId + ", documentInfoJson=" + documentInfoJson + ")");
                }
            });


//发送文件
sdk.fileTo(uid, path, desc, new IFileCallBack() {
                @Override
                public void onProcess(String fileFullPath, long currentIndex, long total)
{

System.out.println(String.format("fileTo.onProcess(currentIndex=  %d,  total=  %d,
fileFullPath= %s)", currentIndex, total, fileFullPath));
                }

                @Override
                public void onSuccess(String documentInfoJson, String fileFullPath) {
                    System.out.println("fileTo.onSuccess(documentInfoJson="  +
documentInfoJson + ", String fileFullPath=" + fileFullPath + ")");
                    cdl.countDown();
                }

                @Override
                public void onFailure(String fileFullPath, Throwable throwable) {
                    System.err.println(throwable);
                }
            });
```

示例代码 33

3.17. 下载文件

3.17.1. 方法签名

```
/**
 * 下载文件.
 *
 * @param documentInfoJson  文件信息 Json 格式字符串.
 * @param saveFilePath      文件保存路径(需用户指定).
 * @param IFileCallBack     文件操作回调接口.
 */
public void downloadFile(String documentInfoJson, String saveFilePath, IFileCallBack fileCallBack);
```

示例代码 34

3.17.2. 输入参数说明

参数名称	必填	类型	描述
documentInfo	是	**DocumentInfo**	**文件信息.**
fileName	是	**String**	**文件保存路径(需用户指定)**
callback	是	**FileCallBack**	文件监听器.

3.17.3. 代码示例

```
sdk.downloadFile(documentInfoJson, saveFilePath, new IFileCallBack() {
                @Override
                public void onProcess(String fileFullPath, long currentIndex, long total)
{

System.out.println(String.format("downloadFile.onProcess(currentIndex=              %d,
total= %d, fileFullPath= %s)", currentIndex, total, fileFullPath));
                }

                @Override
                public void onSuccess(String documentInfoJson, String fileFullPath) {
                    System.out.println("downloadFile.onSuccess(documentInfoJson="
+ documentInfoJson + ", String fileFullPath=" + fileFullPath + ")");
                }

                @Override
                public void onFailure(String fileFullPath, Throwable throwable) {
                    System.err.println(throwable);
                }
            });
```

3.18. 上传文件

3.18.1. 方法签名

```
    /**
     * 异步上传文件.
     *
     * @param fileFullPath   上传文件的全路径.
     * @param fileCallBack  文件操作回调接口.
     */
    public void uploadFile(String fileFullPath, IFileCallBack fileCallBack);
```

示例代码 35

3.18.2. 输入参数说明

参数名称	必填	类型	描述
fileFullPath	是	**String**	上传文件的全路径
fileCallBack	是	**IFileCallBack**	文件监听器.

3.18.3. 代码示例

```
sdk.fileTo(uid, path, desc, new IFileCallBack() {
                @Override
                public void onProcess(String fileFullPath, long currentIndex, long total)
{

System.out.println(String.format("fileTo.onProcess(currentIndex= %d, total= %d,
fileFullPath= %s)", currentIndex, total, fileFullPath));
                }

                @Override
                public void onSuccess(String documentInfoJson, String fileFullPath) {
                    System.out.println("fileTo.onSuccess(documentInfoJson="  +
documentInfoJson + ", String fileFullPath=" + fileFullPath + ")");
                }

                @Override
                public void onFailure(String fileFullPath, Throwable throwable) {
                    System.err.println(throwable);
                }
            });
```

示例代码 36

3.19. 添加好友

用户/群管理模块主要是负责与 iLink 云平台进行 群用户管理。用户通过它可以进行创建、删除、新增群，添加用户、好友，获取用户列表等功能。

3.19.1. 方法签名

```
/**
 * 添加好友.配置是否通知好友.<br/>
 *
 * @param friendId 好友用户 id. 必填.
 * @param notify 通知好友. 必填. 0:加好友时不通知好友; 非 0:加好友时通知好友.
 * @param cb 执行结果回调. 返回 Json 格式数据。 必填
 */
public void addBuddy(String friendId, int notify, ICallback<String> cb);
```

示例代码 37

3.19.2. 输入参数说明

参数名称	必填	类型	描述
friendId	是	String	好友用户 id
notify	是	int	通知好友. 必填. 0:加好友时不通知好友; 非 0:加好友时通知好友
cb		ICallback	结果回调。Json 格式。

3.19.3. 代码示例

```
sdk.addBuddy(testSSDK2, 0, new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("addBuddy not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("addBuddy onSuccess. " + s);
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("addBuddy onFailure. " + ex);
                }
            });
```

示例代码 38

3.20. 获取好友列表

3.20.1. 方法签名

```
/**
     * 获取好友列表.
     *
     * @param cb
     *                操作用户 id. 必填.
     * @param cb 执行结果回调. 返回 Json 格式数据。 必填
     */
    public void getBuddies(ICallback<String> cb);
```

示例代码 39

3.20.2. 输入参数说明

参数名称	必填	类型	描述
cb	是	ICallback	结果回调。Json 格式

3.20.3. 代码示例

```
sdk.getBuddies(new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("getBuddies not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("getBuddies onSuccess. " + s);
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("addUser onFailure. " + ex);
                }
            });
```

示例代码 40

3.21. 删除好友

3.21.1. 方法签名

```
/**
        *  删除好友.配置是否通知好友.<br/>
        * @param friendId.
        *                  好友用户 id. 必填.
        * @param notify
        *                  通知好友. 必填. 0:加好友时不通知好友; 非 0:加好友时通知
好友.
        * @param cb  执行结果回调. 返回 Json 格式数据。  必填
        */
       public void removeBuddy(String friendId, int notify, ICallback<String> cb);
```

示例代码 41

3.21.2. 输入参数说明

参数名称	必填	类型	描述
friendId	是	String	好友用户 id.
notify	是	int	通知好友. 必填. 0:加好友时不通知好友; 非 0:加好友时通知好友
cb	是	ICallback	结果回调。Json 格式

3.21.3. 代码示例

```
sdk.removeBuddy(testSSDK2, 1, new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("removeBuddy not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("removeBuddy onSuccess. " + s);
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("removeBuddy onFailure. " + ex);
                }
            });
```

示例代码 42

3.22. 创建群

3.22.1. 方法签名

```
    /**
     * 创建群.
     *
     * @param groupName  组群名称. 必填.
     * @param userIdList 拉进群的用户 id. 选填.
     * @param cb 执行结果回调. 返回 Json 格式数据。 必填
     */
    public void createGroup(String groupName, List<String> userIdList,
```

示例代码 43

3.22.2. 输入参数说明

参数名称	必填	类型	描述
groupName	是	String	组群名称
userIdList	否	List<String>	拉进群的用户 id
cb	是	ICallback	结果回调。Json 格式

3.22.3. 代码示例

```
sdk.createGroup(groupName, userIdList, new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("createGroup not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("createGroup onSuccess. " + s);
                        JSONObject jj = jo.getJSONObject("groupinfo");
                        groupId = jj.getString("topic");
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("createGroup onFailure. " + ex);
                }
            });
```

示例代码 44

3.23. 获取群列表

3.23.1. 方法签名

```
/**
 * 获取群列表.
 *
 * @param userId
 *                  创建者用户 id. 必填.
 * @param cb 执行结果回调. 返回 Json 格式数据。 必填
 */
public void getGroups(ICallback<String> cb);
```

示例代码 45

3.23.2. 输入参数说明

参数名称	必填	类型	描述
userId	是	String	创建者用户 id.
cb		ICallback	结果回调。Json 格式。

3.23.3. 代码示例

```
sdk.getGroups(new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("getGroups not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("getGroups onSuccess. " + s);
                        //Response result: {"ret":0,"message":"getgrouplist success","grouplist":[{"topic":"982e2236-a67c-979870","groupname":"testGroupFor4.0"},
                        //{"topic":"86aeac7e-af84-623282","groupname":"testGroupFor4.0"}]}
                        JSONArray jj = jo.getJSONArray("grouplist");
                        for (int i = 0; i < jj.size(); i++) {
                            String ss = jj.getJSONObject(i).getString("topic");
                            System.out.println("groupid = " + ss);
                            sdk.destroyGroup(ss, new ICallback<String>() {
                                @Override
                                public void onSuccess(String s) {
                                    JSONObject jo = JSON.parseObject(s);
                                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                                    if (0 != returnCode) {
                                        System.err.println("destroyGroup not onSuccess. " + returnInfo);
                                    } else {
                                        System.out.println("destroyGroup onSuccess. " + s);
                                    }
                                }

                                @Override
                                public void onFailure(Throwable ex) {
                                    System.err.println("destroyGroup onFailure. " + ex);

                                }
                            });
                        }
                    }
```

3.24. 添加群成员

3.24.1. 方法签名

```
/**
 * 添加群成员.
 *
 * @param groupId
 *                群 Id，在系统里群 Id 唯一. 必填.
 * @param userList
 *                进群的用户 id 列表. 必填.
 * @param cb 执行结果回调. 返回 Json 格式数据。 必填
 */
public void addGroupMembers(String groupId, List<String> userList,
ICallback<String> cb);
```

示例代码 46

3.24.2. 输入参数说明

参数名称	必填	类型	描述
groupId	是	String	群 Id，在系统里群 Id 唯一.
userList	是	List<String>	进群的用户 id 列表. 必填
cb		ICallback	结果回调。Json 格式。

3.24.3. 代码示例

```
sdk.addGroupMembers(groupId, userIdList, new ICallback<String>() {
        @Override
        public void onSuccess(String s) {
            JSONObject jo = JSON.parseObject(s);
            int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
            String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
            if (0 != returnCode) {
                System.err.println("addGroupMembers not onSuccess. " + returnInfo);
            } else {
                System.out.println("addGroupMembers onSuccess. " + s);
            }
        }

        @Override
        public void onFailure(Throwable ex) {
            System.err.println("addGroupMembers onFailure. " + ex);
        }
    });
```

示例代码 47

3.25. 删除群成员

3.25.1. 方法签名

```
/**
 * 删除群成员.
 *
 * @param userId
 *                建群用户，创建此群的用户 ID. 必填.
 * @param groupId
 *                群 Id，在系统里群名称唯一. 必填.
 * @param userList
 *                该群中要删除的用户 id 列表. 必填.
 * @param cb 执行结果回调. 返回 Json 格式数据。 必填
 */
public void removeGroupMembers(String groupId, List<String> userList, ICallback<String> cb);
```

示例代码 48

3.25.2. 输入参数说明

参数名称	必填	类型	描述
groupId	是	String	群 Id，在系统里群名称唯一
userList	是	List<String>	该群中要删除的用户 id 列表.
cb		ICallback	结果回调。Json 格式。

3.25.3. 代码示例

```
sdk.removeGroupMembers(groupId, userIdList, new ICallback<String>() {
        @Override
        public void onSuccess(String s) {
            JSONObject jo = JSON.parseObject(s);
            int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
            String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
            if (0 != returnCode) {
                System.err.println("addGroupMembers not onSuccess. " + returnInfo);
            } else {
                System.out.println("addGroupMembers onSuccess. " + s);
            }
        }

        @Override
        public void onFailure(Throwable ex) {
            System.err.println("addGroupMembers onFailure. " + ex);
        }
    });
```

示例代码 49

3.26. 获取群成员列表

3.26.1. 方法签名

```
/**
 * 获取群成员列表.
 *
 * @param groupId
 *                群 Id. 必填.
 * @param userId
 *                当前调用者用户 id. 必填.
 * @param cb  执行结果回调. 返回 Json 格式数据。 必填
 */
public void getGroupMembers(String groupId, ICallback<String> cb);
```

示例代码 50

3.26.2. 输入参数说明

参数名称	必填	类型	描述
groupId	是	String	群 Id
cb		ICallback	结果回调。Json 格式。

3.26.3. 代码示例

```
sdk.getGroupMembers(groupId, new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("getGroupMembers not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("getGroupMembers onSuccess. " + s);
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("createGroup onFailure. " + ex);
                }
            });
```

示例代码 51

3.27. 用户主动退出群.

3.27.1. 方法签名

```
/**
 * 用户主动退出群.<br/>
 *
 * @param groupId
 *                群 Id.
 * @param userId
 *                主动退出群的用户 Id.
 * @param cb  执行结果回调. 返回 Json 格式数据。 必填
 */
public void exitGroup(String groupId, ICallback<String> cb);
```

示例代码 52

3.27.2. 输入参数说明

参数名称	必填	类型	描述
groupId	是	String	群 Id.
cb		ICallback	结果回调。Json 格式。

3.27.3. 代码示例

```
sdk.exitGroup(groupId, new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("exitGroup not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("exitGroup onSuccess. " + s);
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("exitGroup onFailure. " + ex);
                }
            });
```

3.28. 注销群

3.28.1. 方法签名

```
/**
 * 注销群.
 *
 * @param groupId
 *                群 Id. 必填.
 * @param userId
 *                创建群的用户 id. 必填.
 * @param cb 执行结果回调. 返回 Json 格式数据。 必填
 */
public void destroyGroup(String groupId, ICallback<String> cb);
```

示例代码 53

3.28.2. 输入参数说明

参数名称	必填	类型	描述
groupId	是	String	群 Id
cb		ICallback	结果回调。Json 格式。

3.28.3. 代码示例

```
sdk.destroyGroup(groupId, new ICallback<String>() {
                @Override
                public void onSuccess(String s) {
                    JSONObject jo = JSON.parseObject(s);
                    int returnCode = jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                    String returnInfo = (String) jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                    if (0 != returnCode) {
                        System.err.println("destroyGroup not onSuccess. " + returnInfo);
                    } else {
                        System.out.println("destroyGroup onSuccess. " + s);
                        groupId = null;
                    }
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println("destroyGroup onFailure. " + ex);
                }
            });
```

示例代码 54

3.29. 注册用户

3.29.1. 方法签名

```
/**
        * 注册用户.<br/>
        *
        * @param appKey
        * @param secretKey
        * @param balanceHost 服务器地址
        * @param balancePort 服务器端口号
        * @param username    注册的用户名
        * @param nickname    昵称
        * @param cb 执行结果回调. 返回 Json 格式数据。 必填
        */    public void addUser(String appKey, String secretKey, String balanceHost,
int balancePort, String userName, String nickName, ICallback<String> cb);
```

示例代码 55

3.29.2. 输入参数说明

参数名称	必填	类型	描述
appKey	是	String	用户授权码
secretKey	是	String	用户授权秘钥
balanceHost	是	String	服务器地址
balancePort	是	int	服务器端口
userName	是	String	注册的用户名
nickName	是	String	昵称
cb	是	ICallback	结果回调。Json 格式。

3.29.3. 代码示例

注意，此接口需从 EtFactory 获取，不需要进行连接操作即可.

```
EtFactory f = new EtFactory();
        IWeb iw = f.createWeb();
        final CountDownLatch cdl1 = new CountDownLatch(1);
        iw.addUser(APP_KEY, SECRET_KEY, DOMAIN, 8085, "testSSDK0",
"test_ssdk0", new ICallback<String>() {
            @Override
            public void onSuccess(String json) {
                JSONObject jo = JSON.parseObject(json);
                int returnCode =
jo.getInteger(EtConstants.MSG_RESPONSE_CODE);
                String returnInfo = (String)
jo.get(EtConstants.MSG_RESPONSE_CONTENT);
                if (0 != returnCode) {
                    System.err.println("addUser not onSuccess. " + returnInfo);
                } else {
                    System.out.println("addUser onSuccess. " + json);
                }
                cdl1.countDown();
            }

            @Override
            public void onFailure(Throwable ex) {
                System.err.println("addUser onFailure. " + ex);
                cdl1.countDown();
            }
        });
        try {
            cdl1.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

示例代码 56

3.30. 发布群消息

3.30.1. 方法签名

```
/**
 * 发布群消息.<br/>
 * 消息内容通过 byte[]数组传入.
 *
 * @param groupId
 *                群 ID.
 * @param content
 *                消息内容的字节数组.
 * @param callback
 *                发布消息是否成功的结果回调(必填).
 */
public void publishToGroup(String groupId, byte[] payload, ICallback<Void> callback);
```

示例代码 57

3.30.2. 输入参数说明

参数名称	必填	类型	描述
groupId	是	List<UserInfo>	群 ID
content	是	byte[]	消息内容的字节数组.
callback	是	ICallback<EtMsg>	发布消息是否成功的结果回调(必填).

3.30.3. 代码示例

```
sdk.publishToGroup(groupId, msg.getBytes(), new ICallback<Void>() {
                @Override
                public void onSuccess(Void v) {
                    System.out.println(String.format("publishToGroup(%s, %s) 成功.
threadName=%s", groupId, msg, Thread.currentThread().getName()));
                    cdl.countDown();
                }

                @Override
                public void onFailure(Throwable ex) {
                    ex.printStackTrace();
                    cdl.countDown();
                }
            });
```

示例代码 58

4. 附页

4.1. Java SDK 异常编码表

编码	值	描述
SUCCESS	0	成功.
FAIL	-1	失败.
BUFFER_OVERFLOW	-2	数据过长，内存溢出
PARAM_ILLEGAL	-3	参数非法
ALLOC_FAILED	-4	资源不足.
CONFIG_NONSUPPORT	-5	不支持的设置
ET_ERR_ACCOUNT_INVALID	-7	账号信息无效
ET_ERR_LB_SOCKET_FAILED	-8	创建 LB Socket 失败
ET_ERR_LB_DNS_FAILED	-9	LB DNS 解析失败
ET_ERR_LB_CONN_FAILED	-10	LB 连接失败
LB_GET_SERVER_FAILED	-11	从 LB 获取服务器地址失败
LB_SERVER_DNS_FAILED	-12	解析各功能服务器地址失败
LB_RESP_ERROR	-13	LB 返回数据存在错误或长度不正确
LB_PROTOCOL_INVALID	-14	LB 协议不支持
LB_UID_INVALID	-15	LB UID 非法

LB_SERVICE_INVALID	-16	LB 不存在该服务
LB_SERVICE_UNKNOWN_EXCEPTION	-17	LB 未知异常
LB_SERVICE_CONTENT_EXCEPTION	-18	LB 解析内容异常
SOCKET_TIMEOUT_EXCEPTION	-4098	Socket timeouts
CONNECTION_FAILED	-4099	连接 iLink 服务器失败
SYSTEM_UNKNOWN_EXCEPTION	200	系统未知异常
SYSTEM_INIT_EXCEPTION	201	系统初始化异常
CHARACTER_SET_INCORRECT	202	字符集不匹配
OBJECT_NOT_INSTANTIATED	203	对象未实例化.
INIT_APPLICATION_SERVER_INFO	204	获取应用服务器地址信息异常
IO_NET_EXCEPTION	205	网络或 IO 异常
NO_ROUTE_TO_HOST	206	网络路由异常
SERVER_PEER_DISCONNECTED	207	服务器主动断开
NETWORK_IS_UNREACHABLE	208	网络不可用(no further information)
PARAM_NULL	210	参数为 Null
PARAM_NULL_OR_EMPTY	211	参数为 Null 或 空
PARAM_STATE_ILLEGAL	213	参数状态异常
CONNECT_LOAD_BALANCE	214	连接负载均衡服务器异常
ANALY_STATUS_CODES	215	解析状态码异常
CONTENT_IS_TOO_LONG	216	内容过长
ONLINE	217	用户在线失败
OFFLINE	218	用户下线失败
PORT_WHITOUT_RANGE	219	端口范围异常(1025~65535)
STATE_ILLEGAL	220	非法状态异常
JSON_ILLEGAL	221	无效的 Json 格式异常
GET_SYSTEM_TIME_ILLEGAL	222	获取服务器系统时间异常
KEY_VALUE_NOT_EXIST	300	key 对应的值未找到
CLIENT_EXCEPTION	5001	客户端异常
STATE_ILLEGAL	220	非法状态异常
IM_OFFLINE	10000	IM 未连接
CONNACK_CONNECTION_ REFUSED_UNACCEPTED_ PROTOCOL_VERSION	10001	协议版本不正确
CONNACK_CONNECTION_ REFUSED_IDENTIFIER_REJECTED	10002	标识符异常
CONNACK_CONNECTION_REFUSED _SERVER_UNAVAILABLE	10003	服务器不可用
CONNACK_CONNECTION_REFUSED _BAD_USERNAME_OR_PASSWORD	10004	非法的用户名密码
CONNACK_CONNECTION_REFUSED _NOT_AUTHORIZED	10005	授权被拒绝

CONNACK_CONNECTION_NETWORK_IS_UNREACHABLE	10006	网络不可用
CONNECT_SSL_HANDSHAKE	10007	SSL 握手异常
PLATFORM_TYPE	10008	平台标识异常
PING_TIMEOUT	10009	ping 超时断开连接
INVALID_MESSAGE_ID	10010	无效的消息 ID
INVALID_MQTT_COMMAND	10011	无效的 MQTT 指令
PUBLISH_FAIL	10100	发布失败
PUBLISH_TO_GROUP_FAIL	10101	发布群消息失败
SUBSCRIBE_FAIL	10200	订阅失败
UNSUBSCRIBE_FAIL	10300	取消订阅失败
PEERSTATE_FAIL	10400	获取好友状态失败
CHAT_TO_FAIL	10500	获取点对点发送消息失败
DISCONNECT	10600	主动断开连接
SYS_KICK	10601	被踢下线
WEB_UNKNOWN_EXCEPTION	10700	web 未知异常
WEB_INIT_CONFIG	10701	web 初始化配置异常
WEB_NOT_CONFIGURED	10702	web 功能未配置
WEB_SERVER_CONNECT	10703	web 服务器连接异常.
WEB_URL_INFO	10704	web 服务器地址获取
WEB_ADD_BUDDY	10712	添加好友
WEB_REMOVE_BUDDY	10713	删除好友
WEB_BUDDIES	10714	获取好友列表
WEB_CREATE_GRP	10715	创建群
WEB_GRPS	10716	获取群列表
WEB_DISMISS_GRP	10717	注销群
WEB_ADD_GRP_MEMBER	10718	添加群成员
WEB_REMOVE_GRP_MEMBER	10719	删除群成员
WEB_GRP_MEMBERS	10720	获取群成员列表
WEB_REGIST_USER	10721	添加用户
WEB_PUBLISH	10722	Web 发布消息
WEB_GRP_QUIT	10723	用户主动退群
WEB_ADD_BUDDY_EX	10724	添加好友扩展(是否通知对方)
WEB_REMOVE_BUDDY_EX	10725	删除好友扩展(是否通知对方)
FILE_UNKNOWN_EXCEPTION	8192	File 未知异常
FILE_FAILED_PARSE_EXCEPTION	8193	File 解析文件消息失败
FILE_MESSAGE_MISSING_FIELD_WITH_TYPE_EXCEPTION	8194	File 文件消息缺字段 (缺 type)
FILE_MESSAGE_MISSING_FIELD_WITH_FILE	8195	File 文件消息缺字段 (缺 fileId)

ID_EXCEPTION		
FILE_MESSAGE_MISSING_FIELD_WITH_FILENAME_EXCEPTION	8196	File 文件消息缺字段 (缺 fileName)
FILE_MESSAGE_MISSING_FIELD_WITH_SIZE_EXCEPTION	8197	File 文件消息缺字段 (缺 size)
FILE_MESSAGE_MISSING_FIELD_WITH_IP_EXCEPTION	8198	File 文件消息缺字段 (缺 ip)
FILE_MESSAGE_MISSING_FIELD_WITH_PORT_EXCEPTION	8199	File 文件消息缺字段 (缺 port)
FILE_ILLEGAL_MESSAGE_EXCEPTION	8448	File 不合法的文件消息类型
FILE_INIT_CONFIG	8704	File 初始化文件客户端失败
FILE_CONNECT_SERVER_FAILED_EXCEPTION	8705	File 连接文件服务器失败
FILE_ILLEGAL_FILE_ID_EXCEPTION	8706	File 文件 id 不合法
FILE_REMOVE_EXCEPTION	8707	File 删除异常
FILE_STATUS_IS_ABNORMAL_EXCEPTION	8708	File 文件状态异常
FILE_RECEIVE_IS_FAILED_EXCEPTION	8709	File 接收文件内容失败
FILE_DOWNLOAD_EXCEPTION	8710	File 下载异常
FILE_UPLOAD_EXCEPTION	8711	File 上传异常
FILE_BASE_EXCEPTION	8781	File 错误码开始位，与服务器返回值累加得出异常
FILE_SYSTEM_ERROR	8960	File 服务系统发生错误时的编码
FILE_NOFOUND	8961	File 服务没有找到该文件
FILE_AUTH_FAIL	8971	HTTP 服务访问失败
FILE_AUTH_FAIL_TOKEN_INVALID	8972	Token 校验错误
FILE_AUTH_FAIL_FILE_SIZE_LIMIT	8973	单个文件上传或下载大小超出字节限制
FILE_AUTH_FAIL_TOTAL_SIZE_LIMIT	8974	总文件上传或下载大小超出业务系统限制
FILE_AUTH_FAIL_UNKNOW_ERROR	8975	服务器未知错误
FILE_AUTH_FAIL_REPEATED_REQUEST	8976	重复请求错误
FILE_AUTH_FAIL_API_INVALID	8977	API 无效

4.2. 文件传输案例

发送者：

```
private static final String APP_KEY      = "应用标识码，当开发者需要为一款智能产品开发应用（包括终端与设备）时，申请生成，应用开发时需要填入";
private static final String SECURE_KEY = "应用安全识别码，在调用一些管理接口时要填入";
private static final String USERNAME   ="平台标识码（系统唯一，相当于账号），应用开发时需要填入";
public static final int TIMEOUT = 5000;
private static EtServerInfo serverInfo = null;
String documentJson = "";

@Test
    public void testCreate() throws Exception {
        final CountDownLatch cdl = new CountDownLatch(1);
        EtFactory f = new EtFactory();
        IContext ctx = f.createContext();

ctx.setAppKey(APP_KEY).setSecretKey(SECRET_KEY).setUserName(testSSDK3).setServerDomain(DOMAIN).setServerPort(8085);
        ISDK sdk = f.createSDK(ctx);
        sdk.setCallback(new IReceiveListener() {
            @Override
            public void connectionLost(int reasonCode, Throwable cause) {
                System.err.println("~~~~~~~~~~~~~~reasonCode=" + reasonCode);
                cause.printStackTrace();
            }

            @Override
            public void onMessage(String topic, byte[] payload) {
                System.out.println("++++++++++++++++++topic =" + topic + ". payload = " + new String(payload));
            }
        });
        sdk.discoverServers(TIMEOUT, new ICallback<EtServerInfo>() {
            @Override
            public void onSuccess(EtServerInfo etServerInfo) {
                System.err.println(String.format("onSuccess return EtServerInfo = %s", etServerInfo.toString()));
                serverInfo = etServerInfo;
                cdl.countDown();
            }
            @Override
            public void onFailure(Throwable value) {
```

示例代码 59

```
                    System.err.println(String.format("discoverServers() is failed, occur
= %s", value.getLocalizedMessage()));
                    //cdl.countDown();
                }
            });
            cdl.await();

            final CountDownLatch cdl2 = new CountDownLatch(1);
            sdk.connect(serverInfo, DEFAULT_KEEP_ALIVE, true, 6000, new
ICallback<Void>() {

                @Override
                public void onSuccess(Void aVoid) {
                    System.out.println("~~~~~~~~~connect()成功");
                    cdl2.countDown();
                }

                @Override
                public void onFailure(Throwable ex) {
                    System.err.println(ex);
                }
            });
            cdl2.await();
            final CountDownLatch cdl = new CountDownLatch(1);
            sdk.fileTo(uid, path, desc, new IFileCallBack() {
                @Override
                public void onProcess(String fileFullPath, long currentIndex, long total)
{

System.out.println(String.format("~~~~~~~~~fileTo.onProcess(currentIndex=      %d,
total= %d, fileFullPath= %s)", currentIndex, total, fileFullPath));
                }

                @Override
                public void onSuccess(String documentInfoJson, String fileFullPath) {
```

示例代码 60

```
System.out.println("~~~~~~~~~~fileTo.onSuccess(documentInfoJson="                    +
documentInfoJson + ", String fileFullPath=" + fileFullPath + ")");
                        cdl.countDown();
                    }

                    @Override
                    public void onFailure(String fileFullPath, Throwable throwable) {
                        System.err.println(throwable);
                    }
                });
                try {
                    cdl.await();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                sdk.disconnect(new ICallback<Void>() {
                    @Override
                    public void onSuccess(Void aVoid) {
                        System.out.println("~~~~~~~~~~disconnect()成功");
                    }

                    @Override
                    public void onFailure(Throwable ex) {
                        System.err.println("**********disconnect()  失  败  .  "  +
ex.getLocalizedMessage());
                    }
                });
                sdk.destroy();
            }
```

示例代码 61

接收者：

```
private static final String APP_KEY      = "应用标识码，当开发者需要为一款智能产品开发应用（包括终端与设备）时，申请生成，应用开发时需要填入";
private static final String SECURE_KEY = "应用安全识别码，在调用一些管理接口时要填入";
private static final String USERNAME    ="平台标识码（系统唯一，相当于账号），应用开发时需要填入

    public static final int TIMEOUT = 5000;
    private static EtServerInfo serverInfo = null;
    String documentJson = "";

    @Test
    public void testCreate() throws Exception {
        final CountDownLatch cdl = new CountDownLatch(1);
        EtFactory f = new EtFactory();
        IContext ctx = f.createContext();

ctx.setAppKey(APP_KEY).setSecretKey(SECRET_KEY).setUserName(testSSDK2).setServerDomain(DOMAIN).setServerPort(8085);
        ISDK sdk = f.createSDK(ctx);
        sdk.setCallback(new IReceiveListener() {
            @Override
            public void connectionLost(int reasonCode, Throwable cause) {
                System.err.println("~~~~~~~~~~~~~~reasonCode=" + reasonCode);
                cause.printStackTrace();
            }

            @Override
            public void onMessage(String topic, byte[] payload) {
                System.out.println("++++++++++++++++++topic =" + topic + ".payload = " + new String(payload));
            }
        });
```

示例代码 62

```
            sdk.setFileReceiveListener(new IFileReceiveListener() {
                @Override
                public void onReceived(String senderId, String documentInfoJson) {
                    System.out.println("~~~~~~~~~File.onReceived(senderId="        +
senderId + ", documentInfoJson=" + documentInfoJson + ")");

                }
            });
            sdk.discoverServers(TIMEOUT, new ICallback<EtServerInfo>() {
                @Override
                public void onSuccess(EtServerInfo etServerInfo) {
                    System.err.println(String.format("onSuccess    return    EtServerInfo
= %s", etServerInfo.toString()));
                    serverInfo = etServerInfo;
                    cdl.countDown();
                }

                @Override
                public void onFailure(Throwable value) {
                    System.err.println(String.format("discoverServers() is failed, occur
= %s", value.getLocalizedMessage()));
                    //cdl.countDown();
                }
            });
            cdl.await();
```

示例代码 63

```
        final CountDownLatch cdl2 = new CountDownLatch(1);
        sdk.connect(serverInfo, DEFAULT_KEEP_ALIVE, true, 6000, new
ICallback<Void>() {

            @Override
            public void onSuccess(Void aVoid) {
                System.out.println("~~~~~~~~~~connect()成功");
                cdl2.countDown();
            }

            @Override
            public void onFailure(Throwable ex) {
                System.err.println(ex);
            }
        });
        cdl2.await();
        sdk.requestOfflineMessage(new ICallback<Void>() {
            @Override
            public void onSuccess(Void etValue) {
                System.out.println("~~~~~~~~~~requestOfflineMessage()成功");
            }

            @Override
            public void onFailure(Throwable ex) {
                System.err.println(ex);
            }
        });
    }
```

示例代码 64

重要申明

注意

本手册描述的产品及其附件的某些特性和功能，取决于当地网络的设计和性能，以及您安装

的软件。某些特性和功能可能由于当地网络运营商或网络服务供应商不支持，或者由于当地网络的设置，或者您安装的软件不支持而无法实现。因此，本手册中的描述可能与您购买的产品或其附件并非完全一一对应。
易通星云（北京）科技发展有限公司保留随时修改本手册中任何信息的权利，无需进行任何提前通知且不承担任何责任。

无担保声明

本手册中的内容均“如是”提供，除非适用法要求，易通星云（北京）科技发展有限公司对本手册中的所有内容不提供任何明示或暗示的保证，包括但不限于适销性或者适用于某一特定目的的保证。
在法律允许的范围内，易通星云（北京）科技发展有限公司在任何情况下，都不对因使用本手册相关内容而产生的任何特殊的、附带的、间接的、继发性的损害进行赔偿，也不对任何利润、数据、商誉或预期节约的损失进行赔偿。

联系方式

易通星云（北京）科技发展有限公司
公司地址：四川省成都市天府大道北段 1480 号德商国际 B703
邮编：610000
官网地址：www.kaifakuai.com
支持邮箱：support@beidouapp.com
服务电话：028-87582803
官网 QQ 群：445880047
微信公众号：开发快