蚂蚁区块链存证服务API说明书

该文档介绍如何快速使用蚂蚁区块链存证服务API接入区块链，以下章节会介绍引入SDK、初始client，常用的API调用demo以及Transactions和API client的详细信息。

引入SDK Jar

1. 在pom.xml中引入以下依赖。

```
<dependency>
    <groupId>com.alipay.antblockchain</groupId>
    <artifactId>antblockchain-gl-biz-sdk</artifactId>
    <classifier>jar-with-dependencies</classifier>
    <systemPath>${libdir}/client-sdk.jar</systemPath>
</dependency>
<dependency>
    <groupId>com.alipay.antblockchain.sdk.plus</groupId>
    <artifactId>antblockchain-sdk-plus</artifactId>
    <classifier>jar-with-dependencies</classifier>
    <scope>system</scope>
    <systemPath>${libdir}/sdk-plus.jar</systemPath>
</dependency>
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.1.0.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.3.5.Final</version>
</dependency>
```

```
<dependency>
    <groupId>javax.el</groupId>
    <artifactId>javax.el-api</artifactId>
    <version>2.2.4</version>
</dependency>
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>javax.el</artifactId>
    <version>2.2.4</version>
</dependency>
```

2. 解压从BaaS平台下载的client-sdk.zip文件，将内含的client-sdk.jar和sdk-plus.jar加入到依赖目录中，务必修改您的libdir为真实的路径。

初始化Client

1. 配置文件

   client需要配置基本信息，详细配置可见sdk.properties文件，最简包含以下配置。

   确定client要连接的主节点，以及主节点不可用时的备用节点地址。

   此外client与节点连接使用tls双向认证，需要配置x509证书。ssl_key是您的私钥证书，ssl_cert证书可以从BaaS平台的下载链接获取，ssl_key_password是您创建私钥时设置的密码。

   > 注意，在Linux系统中，文件目录采用/path/files的格式即可。Windows系统中有效的目录书写方式是D:/path/files

   ```
   # client连接的主节点地址，主节点必须且只能配置1个
   biz.sdk.primary=116.62.111.181:8080
   # 备份节点api地址，备份节点可以配置0个或多个
   # 主节点无法连接时，或切换连接备份节点，当主节点恢复时，自动切回主节点
   biz.sdk.backups=116.62.111.182:8080;24.62.34.56:8080

   ### 以下路径在Windows下格式为：D:/path/test_key_pkcs8.pem
   # client与节点连接使用tls双向认证，需要配置x509证书
   # pkcs8格式的ssl私钥文件绝对路径
   biz.sdk.ssl_key=/path/test_key_pkcs8.pem
   # x509格式的ssl证书文件绝对路径
   biz.sdk.ssl_cert=/path/test_cert.pem
   # 创建私钥时设置的ssl私钥密码
   biz.sdk.ssl_key_password=******
   # trust store文件绝对路径
   biz.sdk.trust_store=/path/test_client_trust.keystore
   # trust store密码，请咨询BaaS平台相关人员
   biz.sdk.trust_store_password=******
   ```

2. 创建client，支持两种方式初始化client

第一种，读取sdk.properties配置文件来初始化。如果从biz.sdk.ssl_key, biz.sdk.ssl_cert, biz.sdk.trust_store路径中未找到相应文件，DefaultClientPropertyConfig会默认在class路径中寻找名称为"sslkey.pem", "sslcert.pem"和"trust.keystore"的文件并通过ClassLoader载入资源

```
// 加载client配置文件
Properties p = new Properties();
p.load(new FileInputStream("/../../sdk.properties"));
ClientConfig config = new DefaultClientPropertyConfig(p, null, null, null);

// 使用指定client配置初始化client
Client client = new Client(config);
```

第二种，当私钥、证书文件和truststore来自其它位置，如网络io等，您可以直接指定资源的输入流来初始化DefaultClientPropertyConfig

```
// 加载client配置文件
Properties p = new Properties();
p.load(new FileInputStream("/../../sdk.properties"));
// 比如从网络位置获取配置项
URL sslKeyUrl= new URL("");
URL sslCertUrl= new URL("");
URL trustStoreUrl= new URL("");
ClientConfig config = new DefaultClientPropertyConfig(p,
                                    sslKeyUrl.openStream(),
                                    sslCertUrl.openStream(),
                                    trustStoreUrl.openStream());
// 使用指定client配置初始化client
Client client = new Client(config);
```

数据存证写入

业务系统根据具体场景使用不同的Transaction写数据上链，具体参考存证服务Transactions。

数据格式规范

为维护数据的可读性，区块链预先配置了不同业务场景下对应的数据格式规范，作为所有用户的存证数据格式约定。上链数据必须满足规范要求。用户可以在client-sdk.zip包内的schema.txt中查看区块链的数据格式规范。数据格式规范由多个业务数据分类(Category)组成。用户可以通过识别数据的Category在链上检索和过滤存证数据。

这里通过一个简单的schema.txt文件示例来演示如何构造、读取存证数据。文件声明了一个名为GuestInfo的Category，包含三个成员，分别是姓名、生日和邮件地址，均为字符串类型，注释中说明了成员数据须满足的约束条件。

请注意：实际开发时，请将$GuestInfo$替换为您本地$schema.txt$中实际定义的$Category$。

```
// schema.txt定义：商家顾客信息
GuestInfo {
    String userName;  // 姓名，要求非空，长度在2到20之间
    String birthday;  // 生日，要求非空
    String email;     // 邮件地址，要求非空，符合电子邮件格式
}
```

对照规范要求，在client代码中调用GuestInfoBuilder构造一条存证数据。

```
// client代码
// 构造存证数据
GuestInfoBuilder builder = new GuestInfoBuilder();
byte[] bizData = builder.buildUserName("Bob")
                        .buildBirthday("2000-01-01")
                        .buildEmail("bob@inc.com")
                        .build();
```

存证数据构造完成了，接下来构造上链的payload，这里用户需要了解一个枚举类——BizCategory。SDK根据schema.txt的约定，预先生成了这一枚举类型，帮助用户标识不同的Category。本例中的BizCategory如下所示。

```
// SDK的BizCategory声明
public class BizCategory {
    public static final int GuestInfo = 0x00010001;
    ...
}
```

接下来我们把存证数据传入payload，指定Category为BizCategory.GuestInfo，构造一个数据存证交易，并通过client发送出去，存证上链的过程就完成了。

```
// client代码
// 创建ContentOnlyNotary Transaction
TransactionDO tx = TransactionBuilder.getContentOnlyNotaryPayloadBuilder()
        .setContent(bizData)//设置需要存证的数据
        .setTimestamp(System.currentTimeMillis())//设置业务时间
        .setCategory(BizCategory.GuestInfo)//通过BizCategory设置业务分类
        .build();//build Transaction
// 发送Transaction
Response<TransactionDO> response = client.sendTransaction(tx);
```

在稍后读取数据时，用户也同样通过BizCategory.GuestInfo来识别数据对象类型，将存证数据转换回对象。

存证数据上链后，需要等待一段时间才能成块，出块后就可以查询数据了。获取块数据有两种模式，模式一是监听成块消息，处理新数据块，参见下方例4.1；模式二是通过定时任务，每隔一段时间拉取新的数据块，参见下方例4.2。

下面展示几个完整的示例。

例1. 简单的存证数据写入

```
// 加载client配置文件
Properties p = new Properties();
p.load(new FileInputStream("sdk.properties"));
ClientConfig config = new ClientPropertyConfig(p);

// 使用指定client配置初始化client
Client client = new Client(config);

// 构造存证数据
GuestInfoBuilder builder = new GuestInfoBuilder();
byte[] bizData = builder.buildUserName("Bob")
                        .buildBirthday("2000-01-01")
                        .buildEmail("bob@inc.com")
                        .build();

// 创建ContentOnlyNotary Transaction
TransactionDO tx = TransactionBuilder.getContentOnlyNotaryPayloadBuilder()
        .setContent(bizData)//设置需要存证的数据
        .setTimestamp(System.currentTimeMillis())//设置业务时间
        .setCategory(BizCategory.GuestInfo)//设置业务分类
        .build();//build Transaction

// 发送Transaction
Response<TransactionDO> response = client.sendTransaction(tx);

if(response.isSuccess()){
  // 发送成功，业务系统保留Transaction Hash与业务数据关联
  return tx.getTxHashValue();
}else{
  // 发送失败，抛异常。业务系统应该重试，rpc调用失败时区块链的状态是未知的。
  throw new RuntimeException("写入区块链失败");
}
```

例2. 存证数据加密上链

```
// 加载client配置文件
Properties p = new Properties();
p.load(new FileInputStream("client.properties"));
ClientConfig config = new ClientPropertyConfig(p);

// 使用指定client配置初始化client
Client client = new Client(config);

// 初始化链上加密key repositry
```

```
KeyRepositry keyRepositry = new KeyRepositry();

// 设置虚拟root key
KeyPath key1 = new KeyPath("/ABS_ROOT");
// 为BOC Bank类业务生成root加密key
keyRepositry.setKey(key1.forName("ALIPAY").forName("BOC"), genKey());

// 存证数据加密key path
String keyPathName = "/ABS_ROOT/ALIPAY/BOC/BILL";

// 根据key path生成加密key
byte[] key = keyRepositry.getKeyFor(keyPathName);

// 构造存证数据
GuestInfoBuilder builder = new GuestInfoBuilder();
byte[] bizData = builder.buildUserName("Bob")
                        .buildBirthday("2000-01-01")
                        .buildEmail("bob@inc.com")
                        .build();

// build transaction
TransactionDO tx = TransactionBuilder.getEncryptShareNotaryPayloadBuilder()
  .encryptContent(bizData, key, keyPathName.getBytes())//设置存证内容，加密key，key
path
  .setTimestamp(System.currentTimeMillis())//设置业务时间
  .setCategory(BizCategory.GuestInfo)//设置业务分类
  .build();

// 发送Transaction
Response<TransactionDO> response = client.sendTransaction(tx);

if(response.isSuccess()){
  // 发送成功，业务系统保留Transaction Hash与业务数据关联
  return tx.getTxHashValue();
}else{
  // 发送失败，抛异常。业务系统应该重试，rpc调用失败时区块链的状态是未知的。
  throw new RuntimeException("写入区块链失败");
}
```

例3. 存证数据读取

```
// 获取当前最新区块header
final Response<BlockHeader> blockHeader = client.getLatestBlockHeader();
if(!blockHeader.isSuccess()) {
    LOGGER.error("Get block header fail: ", blockHeader.getErrorMsg());
    return;
}
// oldHeight = 当前本地区块高度;
```

```
long beginHeight = oldHeight + 1;
long finalHeight = blockHeader.getData().getHeight();
// 发现新块则开始拉取
while(beginHeight < finalHeight) {
    long endHeight = Math.min(beginHeight + FETCH_LIMIT - 1, finalHeight);
    Response<List<Block>> response = client.getBlocks((int)beginHeight,
(int)endHeight);
    if (!response.isSuccess()) {
        LOGGER.error("Get response fail: ", response.getErrorMsg());
        return;
    }
    List<Block> blocks = response.getData();
    for (Block block : blocks) {
        // 获取区块的存证交易
        List<TransactionDO> transactions = block.getTransactions();
        for (TransactionDO transaction : transactions) {
            // 获取payload
            if
(ContentOnlyNotaryPayloadDO.class.isInstance(transaction.getPayload())) {
                ContentOnlyNotaryPayloadDO contentOnlyNotaryPayloadDO =

 ContentOnlyNotaryPayloadDO.class.cast(transaction.getPayload());
                // 构建GuestInfo对象
                ByteArrayInputStream bis = new ByteArrayInputStream
(contentOnlyNotaryPayloadDO.getContent());
                ObjectInputStream ois = new ObjectInputStream (bis);
                // 判断payload的业务数据分类
                if (contentOnlyNotaryPayloadDO.getCategory() ==
BizCategory.GuestInfo) {
                    GuestInfo renter = GuestInfo.class.cast(ois.readObject());
                    // 读取renter数据
                    System.out.println(renter.getUserName());
                }
            }
        }
        beginHeight = block.getHeader().getHeight() + 1;
    }
}
```

最后如果用户的系统要退出，建议在退出前调用shutdown，client会在shutdown中优雅的关闭线程池。

```
client.shutdown();
```

存证应用交互模式

应用使用区块链进行数据存证时，一般根据应用场景可以分为两类模式

- 数据存证模式
- 数据存取模式

数据存证模式指利用区块链不可篡改，共识时间撮的技术特性，将区块链应用为数据存证系统。该模式下，应用将关键的业务凭证信息（如合同、支付凭证等关键业务信息）写入区块链，应用保存区块链交易hash存根，将区块链视为存证系统。当有举证需求时，根据区块链的交易hash查询之前存证的原始数据以证明业务凭证信息的存在性。

数据存取模式指利用区块链不可篡改，分布式的技术特性，将区块链应用为可信分布式账本，完成基于共享的可信的数据账本的分布式业务协作。该模式下，应用将关键的业务数据写入区块链，多个参与方可根据业务数据属性在链上准实时查询检索业务数据。

以下介绍两种常用的应用交互模式。

数据存证交互模式

存证模式下，区块链系统职责是一个存证系统，共识并储存业务的存证数据，业务在需要时根据交易Hash从区块链查询原始存证数据。

应用交互流程

1. 应用远程调用区块链节点写入数据
2. 区块链节点写入交易池
3. 区块链节点返回受理成功
4. 应用保存交易Hash，与本地业务配置关联
5. 区块链异步完成数据共识上链
6. 应用根据交易Hash查询区块链交易（存在性证明）

内容存证查询

```
// 加载client配置文件
Properties p = new Properties();
p.load(new FileInputStream("sdk.properties"));
ClientConfig config = new ClientPropertyConfig(p);

// 使用指定client配置初始化client
Client client = new Client(config);

// 根据tx Hash查询
Response<TransactionDO> response
=client.getTransaction("462da7df7c10f9b12137549b83bd77b7335cfdd5bf8a060013a111748b
564e94");

if(response.isSuccess()){
  // 查询成功
  if(null == response.getData()){
    // 链上未存在该Tx
    return null;
  }
   // 根据交易类型读取具体交易内容
  if (tx.getType() == PayloadType.TX_TYPE_NOTARY_CONTENT_ONLY.code) {
```

```
    // 读取内容存证交易

    // case the payload type
    ContentOnlyNotaryPayloadDO payloadDO =
(ContentOnlyNotaryPayloadDO)tx.getPayload();

    // do something about the payload
    // ...

    // read content
    System.out.println(new String(payloadDO.getContent(), "UTF-8"));

    // read biz time
    System.out.println(payloadDO.getTimestamp());
  }
  return tx.getTxHashValue()
}else{
  // 查询异常,
}
```

Block查询

```
// 加载client配置文件
Properties p = new Properties();
p.load(new FileInputStream("sdk.properties"));
ClientConfig config = new ClientPropertyConfig(p);

// 使用指定client配置初始化client
Client client = new Client(config);

// 查询区块高度
Response<Block> response = client.getBlock(4);

if(response.isSuccess()){
  // 查询成功
  if(null == response.getData()){
    // 链上未存在该块
    return null;
  }
  // read block
  // ....
}
```

数据存取交互模式

数据存取模式下，区块链系统应用为一个分布式账本，多个参与方共同读写共享账本完成分布式协作。

数据存取系统架构

为了满足多样化的业务领域扩展需求，数据存取模式下区块链架构分为2层

- 区块链
- 业务系统

区块链完成数据共识，账本存储，及提供交易验证能力。

业务系统实时同步区块链账本，按业务需求处理区块上的交易信息，完成定制化业务逻辑、数据存储与相关索引建立。业务系统可以集群化部署，提供稳定高效的查询能力。

整体交互流程：

1. 应用远程调用区块链节点写入数据
2. 区块链节点写入交易池，返回受理成功
3. 区块链异步完成数据共识上链，并发布出块通知
4. 业务系统订阅并接收区块链节点出块通知，按需从区块链拉取区块
5. 业务系统根据业务需求处理交易数据，格式化存储，建立相关业务索引
6. 应用根据业务数据属性查询链上数据

出块事件订阅

```
// 订阅节点的出块事件，当节点生成区块时会发送推送区块通知
// 远程节点当有区块生成时会回调Consumer，在Consumer处理出块通知即可
// groupId相同的订阅方，只选择一台推送
Response<Boolean> response = client.subscribeBlockEvent("group0", new
Consumer<NewBlockEventBody>() {
    @Override
    public void accept(NewBlockEventBody body) {
        // 获取块高
        Long final_height = body.getHeight();
        // 拉取区块并处理交易数据，格式化存储
        // ...
    }
});

if(!response.isSuccess()){
    // 订阅失败
    // 处理失败流程
}
```

除事件订阅外，还可以通过定时任务的方式获取最新区块。

```
@Scheduled(cron="* 1/1 * * * *")
public void fetchBlock() {
    final Response<BlockHeader> blockHeader = client.getLatestBlockHeader();
    if(!blockHeader.isSuccess()) {
        LOGGER.error("Get block header fail: ", blockHeader.getErrorMsg());
        return;
    }
    // 获取块高
    long finalHeight = blockHeader.getData().getHeight();
    // 拉取区块并处理交易数据，格式化存储
    // ...
}
```

存证交易模型Transactions

SDK根据antblockchain-gl协议实现完整的内容存证交易Transaction模型，包括交易编码、交易验证、加密套件、交易签名。

TransactionBuilder

SDK中推荐使用TransactionBuilder来构建内容存证交易。

不同的交易模型获取不同的builder来构建

如下构建一笔存证交易

```
TransactionDO tx = TransactionBuilder.getContentOnlyNotaryPayloadBuilder()
  .setContent(bizData)
  .setCategory(category)
  .setTimestamp(System.currentTimeMillis())
  .build();
```

支持的交易类型builder

```
// 获取引用存证模型builder
TransactionBuilder.getLinkNotaryPayloadBuilder()
// 获取内容存证模型builder
TransactionBuilder.getContentOnlyNotaryPayloadBuilder()
// 获取hash存证模型builder
TransactionBuilder.getHashOnlyNotaryPayloadBuilder()
// 获取密文存证模型builder
TransactionBuilder.getEncryptNotaryPayloadBuilder()
// 获取隐私分享模型builder
TransactionBuilder.getEncryptShareNotaryPayloadBuilder()
// 获取纯密文存证模型builder
TransactionBuilder.getEncryptContentOnlyNotaryPayloadBuilder()
```

存证交易Transaction

Transaction属性如下, 其中不同类型的type有对应的Payload子类承载具体内容，类型映射见PayloadType

```
    // 交易结构版本号
    int version

    //交易类型
    int type

    //随机数，确保交易唯一性
    long nonce

    //交易内容
    Payload payload
```

Payload分类见子类,目前仅支持存证模型，已支持的Payload如下：

- ContentOnlyNotaryPayloadDO
- HashOnlyNotaryPayloadDO
- LinkNotaryPayloadDO
- EncryptContentOnlyNotaryPayloadDO
- EncryptShareNotaryPayloadDO
- EncryptNotaryPayloadDO

Transaction几个关键方法:

模型编码: marshal()对交易进行编码,unmarshal()对交易进行解码。

验证交易: verify()根据协议对交易进行有效性验证

交易哈希: getTxHash()模型不存储Hash,该方法对交易序列化后计算Hash

ContentOnlyNotaryPayloadDO

```
内容存证模型

    如果一个源文件需要存证,可以选择将源文件写上链。该模型限制了512K大小的源文件存储,在该范围内的存证文件
    可以选择写入上链。对于内容过大的存证(如图像、视频等)可以选择LinkNotaryPayloadDO使用连接模型
    的方式上链存证。

存证结构
    - content: 存证内容
        类型: byte[] 区块链不限制编码方式,存证内容由业务自行编码。
        长度: 小于等于512K
```

HashOnlyNotaryPayloadDO

哈希存证模型

　　如果一个文件需要存证,且不希望源文件上链,或者文件过大不适合写上链,可以选择将文件的Hash值上链。

存证结构
　　- hash: 源文件的hash值。根据链外的实际源文件进行摘要计算得到该hash值。
　　　　hash算法: 区块链无法获取源文件,无法验证该hash是否有效。该模型推荐使用sha-256,限制256位的摘要值。
　　　　类型: byte[]
　　　　长度: 固定32byte

LinkNotaryPayloadDO

链接存证模型

　　如果一个文件需要存证,且不希望源文件上链,或者文件过大不适合写上链,可以选择将文件的Hash值上链,此外且将文件的链接(文件固定的URI)写入链。将来检索数据时,可以根据URI去取源文件。

存证结构
　　- link: 存证内容的链接,可以写入URI,或者其他可以用于定位源文件的线索。
　　　　内容: 源文件的URI,或者其他可检索的地址
　　　　类型: byte[] （内容字符串经编码得到byte数组,区块链不校验,也不限制编码方式,由业务自行编解码)
　　　　长度: 小于等于64K

　　- hash: 源文件的hash值。根据link在链外检索到实际源文件,再对源文件进行摘要计算得到该hash值。
　　　　hash算法: 区块链无法获取link源文件,无法验证该hash是否有效。该模型推荐使用sha-256,限制256位的摘要值。
　　　　类型: byte[]
　　　　长度: 固定32byte

EncryptNotaryPayloadDO

隐私存证模型

　　如果一个文件需要存证,且不公开内容,可以选择将源文件通过对称加密算法加密后上链存证。

存证结构
　　- contentHash: 存证的明文内容的Hash值。区块链无法约束存证明文的Hash就是该Hash值。
　　　　hash算法: 区块链无法获取数据明文,无法验证该hash是否有效。该模型推荐使用sha-256,限制256位的摘要值。
　　　　类型: byte[]
　　　　长度: 固定32byte

- encryptContent: 源文件的密文。由加密key及nonce对明文加密得到该值,通过加密key及nonce解密该值,得到密文值。
  类型: byte[]
  长度: 小于等于512K

- nonce: 加密iv。通过AES进行加密时指定随机生成的iv,解密时需要使用该值。
  类型: byte[]
  长度: 小于等于16字节,使用AES GCM算法时,该值一般长度为12byte

EncryptShareNotaryPayloadDO

分享隐私存证模型

如果一个文件需要存证,且不公开内容,可以选择将源文件通过对称加密算法加密后上链存证。对明文进行加密的key通过另外一把私密的key进行key wrap后公开,需要读取该密文数据时,需要拥有私密的key,对wrap key进行解密后得到密文的key,最终得到密文。

业务可以通过该加密规则进行业务设计,例如使用KDF算法设计密钥tree,根据数据的安全级别、分享范围等给数据分配对应的私密key,通过该私密key对特定范围内的数据进行加密。向指定的对象分享该key即可分享这范围内的数据。

存证结构
- contentHash: 存证的明文内容的Hash值。区块链无法约束存证明文的Hash就是该Hash值。
  hash算法: 区块链无法获取数据明文,无法验证该hash是否有效。该模型推荐使用sha-256,限制256位的摘要值。
  类型: byte[]
  长度: 固定32byte

- encryptContent: 源文件的密文。由加密key及nonce对明文加密得到该值,通过加密key及nonce解密该值,得到密文值。
  类型: byte[]
  长度: 小于等于512K

- keyName: 密钥的kdf推导路径。密钥tree父节点根据该路径可以推导出私密key
  类型: byte[]
  长度: 最长512byte

- keyWrap: 加密key的warp key。加密key随机生成,对明文进行加密的,该key有私密key进行key warp后公开。
  使用时,通过私密key对warp key解密,得到加密key,通过该加密key对密文进行解密。
  类型: byte[]
  长度: 该模型支持antblockchain加密库,对wrap key限制40byte。

- nonce: 加密iv。通过AES进行加密时指定随机生成的iv,解密时需要使用该值。
  类型: byte[]
  长度: 小于等于16字节,使用AES GCM算法时,该值一般长度为12byte

EncryptContentOnlyNotaryPayloadDO

```
隐私存证模型

   如果一个文件需要存证,且不公开内容,可以选择将源文件通过对称加密算法加密后上链存证。

存证结构
    - encryptContent: 源文件的密文。由加密key及nonce对明文加密得到该值,通过加密key及
nonce解密该值,得到密文值。
        类型: byte[]
        长度: 小于等于512K

    - nonce: 加密iv。通过AES进行加密时指定随机生成的iv,解密时需要使用该值。
        类型: byte[]
        长度: 小于等于16字节,使用AES GCM算法时,该值一般长度为12byte
```

存证服务API Client

与节点连接有自定义的rpc协议，以下介绍Java SDK实现的api client。

Client

Client类是antblockchain-gl API通讯协议的实现，client是线程安全的，多线程可以显著提高SDK的性能，但不是越多越好，用户需要根据实际需求测试最合理的线程数。

Client采用延迟连接，client创建后并不会立即与server连接，当发生接口调用时才会尝试连接server。默认的尝试次数是3次，每次都会先尝试连接主节点，接着尝试所有的备份节点。一旦主节点连接失败则会每隔固定时间（默认10秒）进行1次主连接尝试，如果此时正处于跟备份节点连接中则会断开跟备份节点的连接，重新选择主节点连接。总之，由于联盟链网络一般都是跨机构，只有机构自己的节点（主节点）有最好的网络环境，其它备份节点（一般是别的机构的节点）的网络延迟都较大，所以client会尽可能选择主节点连接。

Client没有请求失败重发机制，比如请求发送过程中连接断开导致请求发送失败，接口返回的Reponse会携带errorMsg，是否重发请求由调用方决定。

Client中的netty只有1个io线程，与server的channel连接也只有1个。io线程只处理读写，所有的业务逻辑都在业务线程池中处理。Client使用的ssl provider是netty-tcnative-boringssl-static，会比java自带的provider性能高。

Config

创建Client需要ClientConfig，ClientConfig是一个接口，SDK提供默认的实现ClientPropertyConfig。ClientPropertyConfig通过读取properties文件获取配置。

ClientPropertyConfig	说明	必须	值
biz.sdk.primary	主节点api地址，主节点必须且只能配置1个	是	127.0.0.0:8080
biz.sdk.backups	备份节点api地址，备份节点可以配置0个或多个	否	127.0.0.0:8080;127.0.0.0:8081
biz.sdk.primary_auto_reconnect_interval	主节点失效后，自动尝试重连的间隔时间（秒为单位）	否	默认是10秒
biz.sdk.ssl_key	pkcs8格式的ssl私钥文件绝对路径	否	默认是空
biz.sdk.ssl_cert	x509格式的ssl证书文件绝对路径	否	默认是空
biz.sdk.ssl_key_password	ssl私钥密码	否	默认是空
biz.sdk.trust_store	trust store文件绝对路径	否	默认是空
biz.sdk.trust_store_password	trust store密码	否	默认是空
biz.sdk.protocol	通信协议名字，如果为null，则使用默认的Protocol名字	否	默认是1.0
biz.sdk.biz_thread_pool_size	业务线程池大小，如果为非正数，则使用默认的业务线程池大小	否	默认是CPU核数*2
biz.sdk.read_idle_time	读空闲时间（秒为单位），超过该时间SDK会向Server发送Ping消息	否	默认是60秒
biz.sdk.send_one_way_message_timeout	单向消息发送超时时间（毫秒为单位），超过该时间就认为发送失败	否	默认15000
biz.sdk.wait_response_timeout	双向消息应答超时时间（毫秒为单位），超过该时间就认为应答失败	否	默认30000
biz.sdk.socket_send_buffer_size	socket发送缓冲区大小，如果为非正数则使用java默认的缓冲区大小	否	默认0
biz.sdk.socket_recv_buffer_size	socket接收缓冲区大小，如果为非正数则使用java默认的缓冲区大小	否	默认0
biz.sdk.tcp_no_delay	是否开启Nagle算法	否	默认false
biz.sdk.explicit_flush_after_flushes	netty的flush合并策略阈值，如果为正数则开启flush合并	否	默认0

> 所有的api地址都可以有两种形式：127.0.0.1:8080或者127.0.0.1 8080

目前的测试网络中server没有开启ssl，所以client无需配置ssl参数。如果发送的请求比较大可以适当将socket发送缓冲区设置的大点。设置tcp_no_delay需要慎重，如果是单线程使用client且发送的包较小，则容易遇到40ms延迟问题。如果发送的包比较小且高频，可以尝试开启flush合并，对性能提升有帮助。