

华为云区块链引擎服务

开发指南

文档版本 01
发布日期 2023-07-14



版权所有 © 华为云计算技术有限公司 2023。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目录

1 简介	1
2 合约开发	2
2.1 概述	2
2.2 Go 合约开发	2
2.2.1 合约结构	2
2.2.2 合约相关的 API	3
2.2.3 示例 Demo	4
2.3 Wasm 合约开发 (AssemblyScript)	4
2.3.1 合约结构	4
2.3.2 合约相关的 API	5
2.3.3 示例 Demo	6
2.3.3.1 合约编译	6
2.3.3.2 Demo 工程目录	8
3 SDK 介绍	9
3.1 概述	9
3.2 Java SDK 介绍	14
3.2.1 SDK 配置	14
3.2.2 通用方法	15
3.2.3 利用合约发送交易	16
3.2.4 利用合约查询数据	18
3.2.5 其他查询	19
3.2.5.1 查询块高	19
3.2.5.2 查询区块详情	20
3.2.5.3 查询交易执行结果	21
3.2.5.4 利用交易 ID 查询交易详情	22
3.3 Go SDK 介绍	23
3.3.1 SDK 配置	23
3.3.2 通用方法	23
3.3.3 利用合约发送交易	24
3.3.4 利用合约查询数据	26
3.3.5 文件上链	27
3.3.6 文件下载	28

3.3.7 其他查询.....	28
3.3.7.1 查询区块块高.....	28
3.3.7.2 查询区块详情.....	29
3.3.7.3 查询交易执行结果.....	30
3.3.7.4 利用交易 ID 查询交易详情.....	31
3.3.7.5 查询文件历史版本.....	32
3.3.7.6 查询文件操作记录.....	32
3.4 SDK 升级.....	33
3.4.1 Java SDK 升级.....	33
3.4.2 Go SDK 升级.....	33
4 应用程序开发.....	35
4.1 概述.....	35
4.2 Java 应用程序开发.....	35
4.2.1 SDK 客户端配置.....	35
4.2.2 SDK 客户端调用.....	35
4.2.3 示例 Demo.....	36
4.3 Go 应用程序开发.....	36
4.3.1 SDK 客户端配置.....	36
4.3.2 SDK 客户端调用.....	37
4.3.3 示例 Demo.....	37

1 简介

本指导文档主要针对具备Go/Java开发经验的人员进行开发指导，合约与应用程序需客户自行开发，主要包含以下内容：

- [合约开发](#)，包含Go语言、AssemblyScript语言合约。
- [SDK介绍](#)，主要介绍Java、Golang语言SDK。
- [应用程序开发](#)，介绍Java、Golang语言客户端开发流程。

2 合约开发

2.1 概述

合约主要用于操作账本上的数据。作为运行在区块链上的、特定条件下自动执行的代码逻辑，合约是用户利用区块链实现业务逻辑的重要途径，基于区块链特点，合约的运行结果是可信的，其结果是无法被伪造和篡改的。

2.2 Go 合约开发

2.2.1 合约结构

go语言合约即一个Go文件，包含包声明、依赖包导入、智能合约的结构体定义和方法定义。创建好合约文件后就可以进行函数开发等操作。

说明

合约结构中，仅合约结构体可以更改，package名和方法签名不可更改。

合约的结构如下：

```
package usercontract

// 引入必要的包
import (
    "git.huawei.com/poisonsearch/wienerchain/contract/docker-container/contract-go/contractapi"
)

// 声明合约的结构体
type example01 struct {}

// 创建合约
func NewSmartContract() contractapi.Contract {
    return &example01{}
}

// 合约的初始化（Init）接口。将合约启动时，需要首先执行且只需要执行一次的逻辑放到此方法中。
func (e *example01) Init(stub contractapi.ContractStub) ([]byte, error) {
    // 编写时可灵活使用stub中的API
}

// 合约被调用（invoke）接口。将主要的合约执行逻辑，放到此方法内，供合约使用者调用。
```

```
func (e *example01) Invoke(stub contractapi.ContractStub) ([]byte, error) {
    // 编写时可灵活使用stub中的API
}
```

2.2.2 合约相关的 API

contractapi.ContractStub提供如下API接口，可以在合约文件中进行调用。这些API按照功能可以划分为：

表 2-1 辅助功能

接口	说明
FuncName() string	获取智能合约请求中指定的智能合约函数名称。
Parameters() [][]byte	获取请求参数。
ChainID() string	获取智能合约所在链ID。
ContractName() string	获取智能合约名称。

表 2-2 账本数据操作

接口	说明
GetKV(key string) ([]byte, error)	获取某个键对应的值。
PutKV(key string, value []byte) error	添加或更新一对键值。
PutKVCommon(key string, value interface{}) error	添加或更新一对键值。其中值为结构体，需要实现Marshal() ([]byte, error)。
DelKV(key string) error	删除一对键值。
GetIterator(startKey, endKey string) (Iterator, error)	查询指定范围内的键值，查询范围是左闭右开的。
GetKeyHistoryIterator(key string) (HistoryIterator, error)	查询某个键的所有历史值。
SaveComIndex(indexName string, attributes []string, objectKey string) error	创建一个复合键。 例如为"zhangsan":{sex="male", height=175}创建用于查询性别的复合键，可调用SaveComIndex("sex", []string{"male"}, "zhangsan")，如创建用于查询性别和身高的复合键，可调用SaveComIndex("sex/height", []string{"male", "175"}, "zhangsan")。
GetKVByComIndex(indexName string, attributes []string) (Iterator, error)	通过复合键查找满足某种查询条件的键值对。

接口	说明
DelComIndexOneRow(indexName string, attributes []string, objectKey string) error	删除某个复合键。

2.2.3 示例 Demo

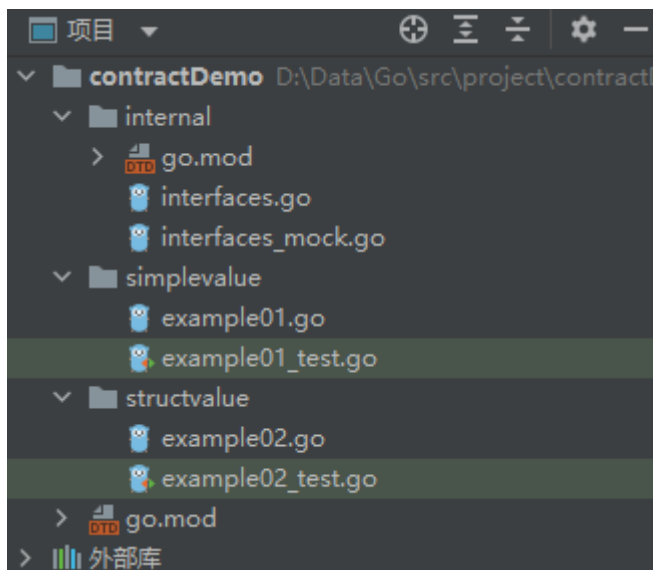
须知

合约开发需要使用go mod，因此请确保GO111MODULE为on、镜像源配置。请确保可正常访问[华为云镜像网站](#)，环境设置命令如下

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://repo.huaweicloud.com/repository/goproxy/
go env -w GONOSUMDB=*
```

合约开发和调测可参考合约示例Demo进行，单击链接获取[contractDemo](#)。

工程目录分为三部分：



1. internal目录提供一些接口以及接口的mock，用于帮助实现合约的接口使用。
2. simplevalue目录提供用于简单value的合约文件和测试文件。
3. structvalue目录提供用于结构体value的合约文件和测试文件。

2.3 Wasm 合约开发 (AssemblyScript)

2.3.1 合约结构

AssemblyScript语言合约主要包括index.ts和contract.ts两个文件，其中index.ts为开发智能合约文件(contract.ts)依赖的合约SDK，合约涉及的业务相关开发仅在contract.ts文件，智能合约文件contract.ts需要根据实际业务进行开发。

- 合约SDK(index.ts)主要结构如下:

```
// 引入智能合约文件
import { invoke, init } from "./contract";

// 合约的初始化 ( wasm_init ) 接口。包含合约文件的init()接口，合约启动时，需要首先执行且只需要执行一次的逻辑放到合约文件init()接口中。
export function wasm_init(buffer_offset: i32, size: i32):void{
    // 实际调用合约文件的init()接口
}

// 合约被调用 ( wasm_invoke ) 接口。包含合约文件的invoke()接口，供合约使用者通过SDK的wasn_invoke接口调用。
export function wasm_invoke(buffer_offset: i32, size: i32):void{
    // 实际调用合约文件的invoke()接口
}

// 合约被调用 ( wasm_prepare ) 接口，保持为空即可。
export function wasm_prepare():void{
}
}
```

- 智能合约文件(contract.ts)主要结构如下:

```
// 引入合约SDK方法
import { FuncName, smlog, Str2ArrayBuffer, Parameters, PutKV, ArrayBuffer2Str, GetKV, DelKV, MakeErrRes, MakeSuccessRes, Response, IteratorNew, IteValue, IteKey, IteNext, IteratorFree } from "./index"

// 智能合约的初始化 ( init ) 接口的实现。
export function init(txid:string):Response{
}

// 智能合约被调用 ( invoke ) 接口的实现。
export function invoke(txid:string):Response{
}
}
```

2.3.2 合约相关的 API

合约SDK(index.ts)提供如下API接口，可以在合约文件中进行调用。这些API按照功能可以划分为：

表 2-3 辅助功能

接口	说明
FuncName(txid :string) :string	获取智能合约请求中指定的智能合约函数名称。
Parameters(txid:string):Array<ArrayBuffer>	获取请求参数。

表 2-4 账本数据操作

接口	说明
GetKV(txid:string, key:string):ArrayBuffer	获取某个键对应的值。

接口	说明
PutKV(txid:string, key:string, value:ArrayBuffer):void	添加或更新一对键值。
IteNext(itor : i64):boolean	返回当前迭代器指针是否存在下一个指针。
DelKV(txid:string, key:string):i32	删除一对键值。
IteratorFree(itor : i64):i32	释放迭代器指针。
IteratorNew(txid:string, beginKey:string, endKey:string) :i64	新建范围为[beginKey, endKey]的迭代器。
IteKey(itor : i64):string	从迭代器中获取key。
IteValue(itor : i64):ArrayBuffer	返回迭代器指向的值。

2.3.3 示例 Demo

2.3.3.1 合约编译

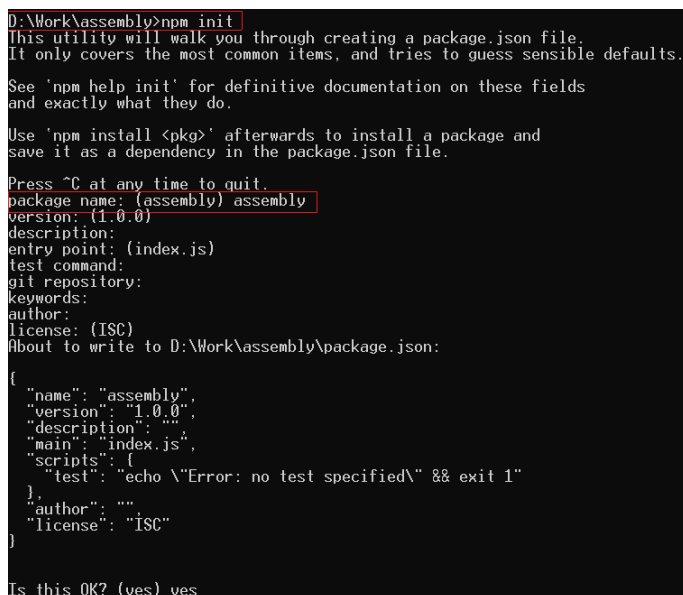
1. 下载Node.js软件并安装，安装成功后，执行如下命令查看对应版本(软件对应版本无强制要求)。

```
node -v
npm -v
```



```
C:\Users\...\.CHINA>node -v
v16.13.1
C:\Users\...\.CHINA>npm -v
8.1.2
```

2. 设置新目录assembly，在该目录下执行 npm init 命令，其中package name输入为assembly(目录名、package name建议保持一致，具体名称无强制要求，可自行定义)。



```
D:\Work\assembly>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (assembly) assembly
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to D:\Work\assembly\package.json:
{
  "name": "assembly",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
Is this OK? (yes) yes
```

3. 执行如下命令通过npm安装加载器和编译器

```
npm install --save @assemblyscript/loader
npm install --save-dev assemblyscript
```

```
D:\Work\assembly>npm install --save @assemblyscript/loader
added 1 package in 557ms
D:\Work\assembly>npm install --save-dev assemblyscript
added 6 packages in 1s
1 package is looking for funding
run 'npm fund' for details
```

 说明

若安装过程中出现“idealTree:assembly: sill idealTree buildDeps”，请确认npm使用镜像源可正常访问。

- 查看npm镜像源配置命令
npm config get registry
- 设置npm镜像源配置命令
npm config set registry 国内镜像源地址

4. 执行如下命令，利用编译器提供的脚手架设置新项目

npx asinit .

```
D:\Work\assembly>npx asinit .
Version: 0.19.20
This command will make sure that the following files exist in the project
directory 'D:\Work\assembly':
./assembly
Directory holding the AssemblyScript sources being compiled to WebAssembly.
./assembly/tsconfig.json
TypeScript configuration inheriting recommended AssemblyScript settings.
./assembly/index.ts
Example entry file being compiled to WebAssembly to get you started.
./build
Build artifact directory where compiled WebAssembly files are stored.
./build/.gitignore
Git configuration that excludes compiled binaries from source control.
./asconfig.json
Configuration file defining both a 'debug' and a 'release' target.
./package.json
Package info containing the necessary commands to compile to WebAssembly.
./index.js
Main file loading the WebAssembly module and exporting its exports.
./tests/index.js
Example test to check that your module is indeed working.
The command will try to update existing files to match the correct settings
for this instance of the compiler in 'D:\Work\assembly\node_modules\assemblyscript'.
Do you want to proceed? [Y/n] y
```

5. 参考[示例Demo](#)完成合约文件contract.ts编写与合约SDK文件index.ts引用后，执行build命令编译AssemblyScript类型的合约文件，编译成功后在build目录下生成optimized.wasm字节码文件（该字节码文件可重新命名）。

npm run asbuild

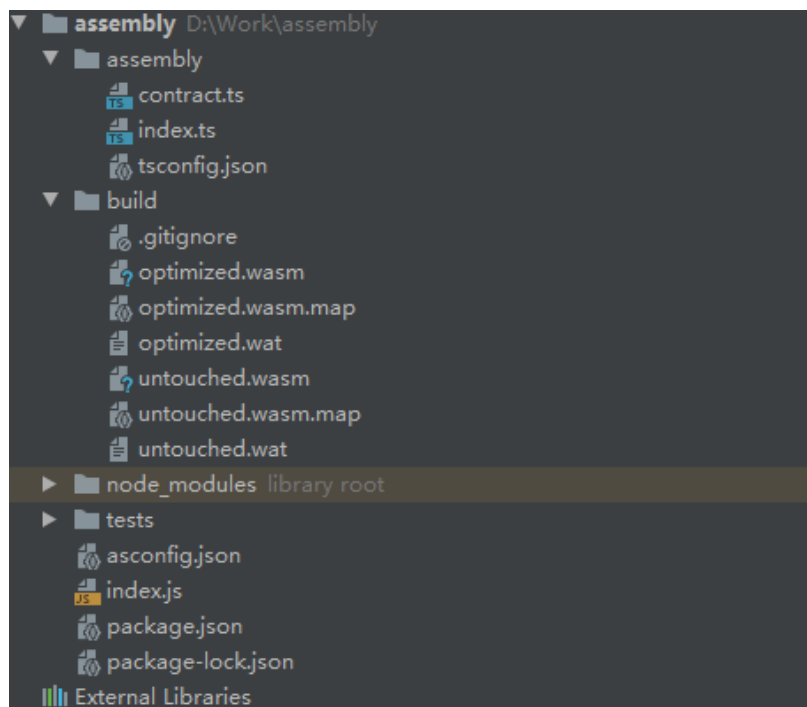
```
D:\Work\assembly>npm run asbuild
> assembly@1.0.0 asbuild
> npm run asbuild:untouched && npm run asbuild:optimized

> assembly@1.0.0 asbuild:untouched
> asc assembly/index.ts --target debug

> assembly@1.0.0 asbuild:optimized
> asc assembly/index.ts --target release
```

2.3.3.2 Demo 工程目录

合约开发和调测可参考合约示例Demo，单击链接获取AssemblyScript语言[合约工程Demo](#)。



1. build目录下optimized.wasm为合约编译后对应的wasm字节码文件(最终合约类型文件为optimized.wasm压缩成的*.zip包，[合约示例Demo](#))。
2. assembly/index.ts 为开发智能合约文件(contract.ts)依赖的合约SDK。
3. assembly/contract.ts 为智能合约文件，本Demo中合约仅进行简单展示(实际合约文件contract.ts需自行开发)。
4. 合约文件的安装请参考[合约管理](#)。

📖 说明

AssemblyScript语言类型合约不支持查询指定键的历史数据。

3 SDK 介绍

3.1 概述

SDK 说明

华为链目前提供Java、Golang两种语言SDK，区块链服务启动时会启动一系列grpc接口，监听客户端发送的消息，与客户端交互完成各种请求。在开发客户端时，如果从底层grpc接口开始，进行各种消息封装、消息发送、返回值解析等工作，不仅会导致开发量过大，并且造成重复劳动。SDK则是将区块链服务提供的各种grpc接口进行封装，同时封装各接口所需类型的消息。在开发客户端时，只需要关注自己的业务逻辑，调用相应接口封装并发送消息即可，不需要关注底层消息发送接收的具体过程。

📖 说明

1. SDK中相关方法的使用示例，可参考[应用程序开发](#)对应语言的示例Demo。
2. SDK在不同环境下支持的实例安全机制如下：
 - windows环境：ECDSA
 - linux环境：国密算法、ECDSA

SDK 逻辑结构

SDK主要提供消息封装、发送模块及相应的配套组件。配套组件主要包含异常处理、配置文件解析、节点获取等功能接口。

- 消息封装
由于消息类型较多，因此按类型进行了分类封装。消息封装相关接口均在build这个包下面，包含了ContractRawMessage、QueryRawMessage这几种消息构造的封装类。
 - ContractRawMessage：包含交易背书、落盘两阶段消息的构建。
 - QueryRawMessage：包含所有查询相关接口的消息构建，目前支持交易详情查询、链状态查询、区块查询、合约信息查询等接口。
- 消息发送
同消息封装类接口，按类型进行了分类封装。消息发送相关接口均包含在action这个包下面，包含了ContractAction、QueryAction、EventAction这几种消息发送的封装类。

- ContractAction: 对应ContractRawMessage封装的消息的发送。
- QueryAction: 对应QueryRawMessage中封装的消息的发送。
- EventAction: 主要用于监听消息的最终状态，因为参数仅包含交易ID，消息封装的方法直接内置。同时所有的消息发送接口均提供同步和异步两种接口。

📖 说明

同步接口入参均为需要发送的消息，返回值为一个ListenableFuture对象，用于监听消息发送结果。

异步接口则传入需要发送消息的同时，还需要传入一个StreamObserver对象，用于异步获取消息发送结果。

基于 SDK 开发流程

基于SDK开发客户端需要进行以下步骤。

1. 初始化
创建SDK实例对象，然后进行初始化。
2. 构造消息
由于消息类型较多，所以根据消息类型进行封装，在构造消息之前，必须先获取消息类型对象，然后再基于获取对象中对应的方法封装消息。
3. 获取节点
获取消息发送节点对象。
4. 发送消息
所有的发送接口都封装在节点类中，发送消息时，先获取节点，再调用节点的消息发送对象获取方法。不同的消息对应不同的构建接口，同理，消息发送接口也根据接口类型进行了分装，在发送消息前，必须先获取接口类型对象，然后再基于获取对象中对应的方法发送消息。
5. 结果解析
服务端返回的消息中，均包含交易最后执行结果的标志位，以此判断交易是否执行成功。若结果为不成功，则返回错误原因，用于分析定位。
6. 结果监听
对于业务交易和投票类型等需要落盘的交易，即使消息发送成功，后续落盘时还可能产生各种校验失败，导致交易无效。因此还需要监听交易是否最终落盘成功。

接口说明

由于实现各种不同的交易发送，需要多个接口互相配合，因此接口说明按照实现功能进行归类介绍，而不是逐个接口类介绍。

所有消息发送方法，都提供同步和异步两种接口，异步接口多一个StreamObserver监听对象，无返回值。

另外SDK还提供其他扩展能力，有需要的用户可以参考。如Java接口：

```
public SdkClient(String configPath, Function<byte[], byte[]> func)
public SdkClient()
public void setIdentity(String type, byte[] key, byte[] cert)
```

```
public void setTls(byte[] key, byte[] cert, byte[][] roots)
public void addWienerChainNode(String name, String host, int port)
public String getTxId(Transaction tx)
public ChainRawMessage getChainRawMessage()
public Block buildGenesisBlock(String chainId, String path)
public Block buildGenesisBlock(String chainId, String path, Function<byte[],
byte[]> func)
public Block buildGenesisBlock(String chainId, ChainConfig chainConfig)
public static GenesisConfig createGenesisConfig(String path)
public void addOrganization(String name, byte[] admin, byte[] root, byte[] tls)
public void addConsenter(String name, String org, String host, long port, byte[]
cert, byte[] tee)
public ChainConfig getChainConfig(String chainId)
public Block buildGenesisBlock(String chainId, ChainConfig chainConfig)
public void addChainNode(String name, String hostOverride, String host, int port)
public RawMessage buildJoinChainRawMessage(Block genesisBlock)
public RawMessage buildJoinChainRawMessage(ByteString genesisBlock,
Entrypoint entrypoint)
public RawMessage buildJoinChainRawMessage(ByteString genesisBlock,
ConfigInfo config, Entrypoint entrypoint)
public static Entrypoint readEntrypointFile(String path) throws ConfigException
public ListenableFuture<RawMessage> joinChain(RawMessage rawMessage)
public void joinChain(RawMessage rawMessage, StreamObserver<RawMessage>)
public RawMessage buildQuitChainRawMessage(String chainId)
public ListenableFuture<RawMessage> quitChain(RawMessage rawMessage)
public void quitChain(RawMessage rawMessage, StreamObserver<RawMessage>
responseObserver)
public RawMessage buildQueryChainRawMessage()
public ListenableFuture<RawMessage> queryAllChains(RawMessage rawMessage)
public void queryAllChains(RawMessage rawMessage,
StreamObserver<RawMessage> responseObserver)
public RawMessage buildQueryChainRawMessage(String chainId)
public ListenableFuture<RawMessage> queryChain(RawMessage rawMessage)
public void queryChain(RawMessage rawMessage, StreamObserver<RawMessage>)
public Builder.TxRawMsg buildUpdateChainRawMessage(String chainId, String
path)
```

```
public Builder.TxRawMsg buildUpdateChainRawMessage(String chainId,
ChainConfig chainConfig)

public Builder.TxRawMsg buildUpdateChainRawMessage(String chainId, String
path, Function<byte[], byte[]> func)

public TxRawMsg buildUpdateConfPolicyRawMessage(String chainId, String policy)

public TxRawMsg buildUpdateOrgRawMessage(String chainId, String path,
Function<byte[], byte[]> func)

public TxRawMsg buildUpdateOrgRawMessage(String chainId,
List<ConfigSet.OrgUpdate> orgUpdates)

public RawMessage buildQueryChainUpdateVote(String chainId)

public ListenableFuture<RawMessage> queryVote(RawMessage rawMsg)

public void queryVote(RawMessage rawMsg, StreamObserver<RawMessage>
responseObserver)

public RawMessage buildImportRawMessage(Contract contract, String path, String
sandbox, String language)

public ListenableFuture<RawMessage> contractImport(RawMessage rawMessage)

public void contractImport(RawMessage rawMessage,
StreamObserver<RawMessage> responseObserver)

public Builder.TxRawMsg buildManageRawMessage(String chain, String contract,
String option)

public RawMessage buildQueryStateRawMessage(String chain, String contract)

public ListenableFuture<RawMessage> queryState(RawMessage rawMessage)

public Builder.TxRawMsg buildVoteRawMessage(Contract contract, String
description, String policy, boolean historySupport)

public Builder.TxRawMsg buildSQLVoteRawMessage(Contract contract, String
description, String policy, String schema, boolean isHistorySupport)

public ListenableFuture<RawMessage> transaction(RawMessage rawMessage)

public void transaction(RawMessage rawMessage, StreamObserver<RawMessage>
responseObserver)

public RawMessage buildQueryLifecycleVote(String chainId, String contract)

public static ContractInvocation buildContractInvocation(String name, String
function, String[] args)

public RawMessage buildContractRawMessage(String chainId, String contract)

public ListenableFuture<RawMessage> queryContractInfo(RawMessage rawMsg)

public void queryContractInfo(RawMessage rawMsg,
StreamObserver<RawMessage> responseObserver)
```

如Go接口：


```
func NewGatewayClient(configPath string, decrypts ...func(bytes []byte) ([]byte, error)) (*GatewayClient, error)

func GenerateTimestamp() uint64

func (msg *ChainRawMessage) BuildGenesisBlock(chainID string, genesisConfigPath string, decrypts ...func(bytes []byte) (*common.Block, error))

func (msg *ChainRawMessage) BuildJoinChainRawMessage(genesisBlockBytes []byte) (*common.RawMessage, error)

func (msg *ChainRawMessage) BuildJoinMsgWithLatestConf(genesisBlockBytes []byte, latestConf *common.ConfigInfo) (*common.RawMessage, error)

func (msg *ChainRawMessage) BuildJoinMsgWithEntrypoint(genesisBlockBytes []byte, latestConf *common.ConfigInfo, entrypoint *common.Entrypoint) (*common.RawMessage, error)

func (action *ChainAction) JoinChain(rawMsg *common.RawMessage) (*common.RawMessage, error)

func (msg *ChainRawMessage) BuildQuitChainRawMessage(chainID string) (*common.RawMessage, error)

func (action *ChainAction) QuitChain(rawMsg *common.RawMessage) (*common.RawMessage, error)

func (msg *ChainRawMessage) BuildQueryAllChainRawMessage() (*common.RawMessage, error)

func (action *ChainAction) QueryAllChains(rawMsg *common.RawMessage) (*common.RawMessage, error)

func (msg *ChainRawMessage) BuildQueryChainRawMessage(chainID string) (*common.RawMessage, error)

func (action *ChainAction) QueryChain(rawMsg *common.RawMessage) (*common.RawMessage, error)

func (u *UpdateConfig) BuildUpdateConfPolicyRawMessage(chainID string, policy string) (*TxRawMsg, error)

func (u *UpdateConfig) BuildUpdateLifecycleRawMessage(chainID string, policy string) (*TxRawMsg, error)

func (u *UpdateConfig) BuildUpdateOrgRawMessageWithYaml(chainID string, path string, decrypt config.DecryptFunc) (*TxRawMsg, error)

func (u *UpdateConfig) BuildUpdateOrgRawMessage(chainID string, orgUpdates *common.ConfigSet_OrgUpdates) (*TxRawMsg, error)

func (msg *QueryRawMessage) BuildQueryChainUpdateVoteRawMessage(chainID string) (*common.RawMessage, error)

func (action *QueryAction) GetVote(rawMsg *common.RawMessage) (*common.RawMessage, error)

func (msg *LifecycleRawMessage) BuildImportRawMessage(c *Contract, path string, sandbox string, language string) (*common.RawMessage, error)
```

```

func (action *ContractAction) ContractImport(rawMsg *common.RawMessage)
(*common.RawMessage, error)

func (msg *LifecycleRawMessage) BuildManageRawMessage(chain string, contract
string, option string) (*TxRawMsg, error)

func (msg *LifecycleRawMessage) BuildVoteRawMessage(c *Contract, desc string,
policy string, historySupport bool) (*TxRawMsg, error)

func (action *ContractAction) Transaction(rawMsg *common.RawMessage)
(*common.RawMessage, error)

func (msg *QueryRawMessage) BuildQueryLifecycleVoteRawMessage(chainID
string, contract string) (*common.RawMessage, error)

func BuildTxHeader(chainID string, []string domains) *common.TxHeader

func (msg *QueryRawMessage) BuildContractRawMessage(chainID string, contract
string) (*common.RawMessage, error)

func (action *QueryAction) GetContractInfo(rawMsg *common.RawMessage)
(*common.RawMessage, error)

```

3.2 Java SDK 介绍

3.2.1 SDK 配置

📖 说明

1. Java的项目管理工具有maven和gradle两种，本指导主要以maven为例。
2. linux环境，Java SDK的编译、运行，需要手动下载并配置openssl([openssl下载链接](#))
 - 解压openssl，拷贝openssl文件夹到目录/usr/local/include/下
 - 执行库的导入命令：export LD_LIBRARY_PATH=/usr/local/include/openssl

引用SDK的步骤如下：

步骤1 打开项目中的pom.xml文件。

步骤2 粘贴如下代码引入华为镜像仓。

```

<repositories>
  <repository>
    <id>maven-proxy</id>
    <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
      <updatePolicy>always</updatePolicy>
      <checksumPolicy>fail</checksumPolicy>
    </snapshots>
  </repository>
</repositories>

```

步骤3 粘贴如下代码引用SDK。

```

<dependencies>
  <dependency>
    <groupId>com.huawei.wienerchain</groupId>

```

```
<artifactId>wienerchain-java-sdk</artifactId>
<version>2.1.0.6.41</version>
</dependency>
</dependencies>
```

步骤4 等待自动拉取依赖。

----结束

📖 说明

对于企业内部需要使用代理访问外网的情况，可以在用户目录（windows中如C:\Users\xxx\）下的.m2目录中setting.xml（用户配置）或maven安装目录下的conf目录中setting.xml（系统全局配置）里配置代理来实现。

找到setting.xml文件中的标签对，在其内配置代理信息，参考如下样例：

```
<proxies>
  <proxy>
    <id>myProxy</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>xxx</username>
    <password>xxx</password>
    <host>xxx</host>
    <port>xxx</port>
    <nonProxyHosts>*xxx*.com</nonProxyHosts>
  </proxy>
</proxies>
```

3.2.2 通用方法

相关类

com.huawei.wienerchain.SdkClient

SdkClient对象包含获取服务节点、获取各种类型的消息构造器及交易ID等方法，基于SDK开发时，必须先构造该对象。

初始化 SDK 客户端

基于标准配置文件模板初始化SDK。

- 调用方法
public SdkClient(String configPath) throws CryptoException, ConfigException, IOException
- 参数说明

参数	类型	说明
configPath	String	客户端SDK配置文件的绝对路径。

获取节点对象

根据节点名称，获取需要发送交易的节点对象。所有消息发送前，都必须调用该方法，获取发送节点对象，然后再获取对应的消息发送接口对象。

- 调用方法
public WienerChainNode getWienerChainNode(String name) throws InvalidParameterException

- 参数说明

参数	类型	说明
name	String	节点名称。

- 返回值

类型	说明
WienerChainNode	WienerChainNode对象。

获取交易 ID

交易ID作为交易的标识，是交易哈希的十六进制字符串形式。以下获得交易ID的方法是通过计算交易哈希，然后转化为十六进制字符串获得的。

- 调用方法

```
public String getTxId(Transaction tx)
```

- 参数说明

参数	类型	说明
tx	Transaction	交易实体。

- 返回值

类型	说明
String	交易ID。

3.2.3 利用合约发送交易

步骤1 合约调用信息构建。

- 接口方法

```
ContractRawMessage.class
```

```
public RawMessage buildInvokeRawMsg(String chainId, String name, String function, String[] args)
```

- 参数说明

参数	类型	说明
chainId	String	链名称。
name	String	合约名称。
function	String	调用合约中的方法名。
args	String[]	合约方法参数。

- 返回值

类型	说明
Invocation	合约调用信息。

步骤2 背书请求消息构建。

- 接口方法

ContractRawMessage.class

```
public RawMessage getRawMessageBuilder(ByteString payload) throws CryptoException
```

- 参数说明

参数	类型	说明
payload	ByteString	合约调用信息，由 invocation.toByteArray() 得到。

- 返回值

类型	说明
RawMessage	背书请求需发送的消息。

步骤3 背书请求消息发送。

- 接口方法

ContractAction.class

```
public ListenableFuture<RawMessage> invoke(RawMessage rawMessage) throws  
InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMessage	RawMessage	背书请求需发送的消息。

- 返回值

类型	说明
ListenableFuture	用于获取发送结果的future对象。

步骤4 落盘消息构建。

- 接口方法

ContractRawMessage.class

```
public TxRawMsg buildTransactionMessage(RawMessage[] rawMessages) throws  
InvalidProtocolBufferException, TransactionException, CryptoException
```

- 参数说明

参数	类型	说明
rawMessages	RawMessage	背书结果。

- 返回值

类型	说明
TxRawMsg	交易的消息数据，包含原始信息和哈希。

步骤5 落盘消息发送。

- 接口方法

ContractRawMessage.class

```
public ListenableFuture<RawMessage> transaction(RawMessage rawMessage) throws  
InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMessage	RawMessage	交易原始信息。

- 返回值

类型	说明
ByteString	落盘结果。

----结束

3.2.4 利用合约查询数据

步骤1 合约调用信息构建。

- 接口方法

ContractRawMessage.class

```
public Invocation buildInvocation(String chainId, String name, String function, String[] args)
```

- 参数说明

参数	类型	说明
chainId	String	链名称。
name	String	合约名称。
function	String	调用合约中的方法名。
args	String[]	合约方法参数。

- 返回值

类型	说明
Invocation	合约调用信息。

步骤2 查询请求消息构建。

- 接口方法

ContractRawMessage.class

```
public RawMessage getRawMessageBuilder(ByteString payload) throws CryptoException
```

- 参数说明

参数	类型	说明
payload	ByteString	合约调用信息，由 invocation.toByteString()得到。

- 返回值

类型	说明
RawMessage	查询请求需发送的消息。

步骤3 查询请求消息发送。

- 接口方法

ContractAction.class

```
public ListenableFuture<RawMessage> invoke(RawMessage rawMessage) throws InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMessage	RawMessage	查询请求需发送的消息。

- 返回值

类型	说明
ListenableFuture	用于获取查询结果的future对象。

----结束

3.2.5 其他查询

3.2.5.1 查询块高

步骤1 消息构建。

- 接口方法

QueryRawMessage.class

```
public RawMessage buildLatestChainStateRawMessage(String chainId) throws CryptoException
```

- 参数说明

参数	类型	说明
chainId	String	链名称。

- 返回值

类型	说明
RawMessage	查询块高需发送的消息。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
public ListenableFuture<RawMessage> queryLatestChainState(RawMessage rawMsg) throws  
InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMsg	RawMessage	查询块高需发送的消息。

- 返回值

类型	说明
ListenableFuture	用于获取查询结果的future对象。

----结束

3.2.5.2 查询区块详情

步骤1 消息构建。

- 接口方法

QueryRawMessage.class

```
public RawMessage buildBlockRawMessage(String chainId, long blockNum) throws CryptoException
```

- 参数说明

参数	类型	说明
chainId	String	链名称
blockNum	long	区块号。

- 返回值

类型	说明
RawMessage	查询区块详情需发送的消息。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
public ListenableFuture<RawMessage> queryBlockByNum(RawMessage rawMsg) throws  
InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMsg	RawMessage	查询区块详情需发送的消息。

- 返回值

类型	说明
ListenableFuture	用于获取查询结果的future对象。

----结束

3.2.5.3 查询交易执行结果

步骤1 消息构建。

- 接口方法

QueryRawMessage.class

```
public RawMessage buildTxRawMessage(String chainId, byte[] txHash) throws CryptoException
```

- 参数说明

参数	类型	说明
chainId	String	链名称。
txHash	byte[]	交易哈希。

- 返回值

类型	说明
RawMessage	根据交易ID查询交易执行结果需发送的消息。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
public ListenableFuture<RawMessage> queryTxResultByTxHash(RawMessage rawMsg) throws  
InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMsg	RawMessage	根据交易ID查询交易执行结果需发送的消息。

- 返回值

类型	说明
ListenableFuture	用于获取查询结果的future对象。

----结束

3.2.5.4 利用交易 ID 查询交易详情

步骤1 消息构建。

- 接口方法

QueryRawMessage.class

```
public RawMessage buildTxRawMessage(String chainId, byte[] txHash) throws CryptoException
```

- 参数说明

参数	类型	说明
chainId	String	链名称。
txHash	byte[]	交易哈希。

- 返回值

类型	说明
RawMessage	根据交易ID查询交易详情需发送的消息。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
public ListenableFuture<RawMessage> queryTxByHash(RawMessage rawMsg) throws  
InvalidParameterException
```

- 参数说明

参数	类型	说明
rawMsg	RawMessage	根据交易ID查询交易详情需发送的消息。

- 返回值

类型	说明
ListenableFuture	用于获取查询结果的future对象。

----结束

3.3 Go SDK 介绍

3.3.1 SDK 配置

📖 说明

linux环境，Go SDK的编译、运行，需要手动下载并配置openssl([openssl下载链接](#))

- 解压openssl，拷贝openssl文件夹到目录/usr/local/include/下
- 执行库的导入命令：export LD_LIBRARY_PATH=/usr/local/include/openssl

SDK版本号	下载链接
2.1.0.6.41	下载SDK

3.3.2 通用方法

相关类

- GatewayClient对象包含获取服务节点、获取各种类型的消息构造器及交易ID等方法，基于SDK开发时，必须先构造该对象。
client.GatewayClient
- BsClient对象包含富媒体文件上链、下载、操作记录查询等方法，使用区块链富媒体存储相关功能时，必须先构造该对象。
bstore.BsClient

初始化 SDK 客户端

基于标准配置文件模板初始化Gateway SDK。

- 调用方法
func NewGatewayClient(configPath string, decrypts ...func(bytes []byte) ([]byte, error)) (*GatewayClient, error)
- 参数说明

参数	类型	说明
configPath	String	必填参数，客户端SDK配置文件的绝对路径。
decrypts	func(bytes []byte) ([]byte, error)	非必填参数，指定证书密文解密算法，默认为明文，无须解密（可变参数）。

初始化 SDK 文件存储客户端

基于已初始化的SDK Gateway客户端模板初始化富媒体存储客户端。

- 调用方法
func NewBsClient(gatewayClient *client.GatewayClient, chainID string, consenterName string) (*BsClient, error)
- 参数说明

参数	类型	说明
gatewayClient	*client.GatewayClient	必填参数，已初始化过的Gateway客户端。
chainID	string	链名称。
consenterName	string	共识节点名称，如“node-0.organization”。

生成当前时间戳

根据节点名称，获取需要发送交易的节点对象。所有消息发送前，都必须调用该方法，获取发送节点对象，然后再获取对应的消息发送接口对象。

- 调用方法
func GenerateTimestamp() uint64
- 返回值

类型	说明
uint64	生成系统当前的UTC时间戳。

3.3.3 利用合约发送交易

步骤1 背书消息构建

- 接口函数
func (msg *ContractRawMessage) BuildInvokeMessage(chainID string, name string, function string, args []string) (*common.RawMessage, error)

- 参数说明

参数	类型	说明
chainID	string	链名称。
name	string	合约名称。
function	string	调用合约中的方法名。
args	[]string	合约方法参数。

- 返回值

类型	说明
*common.RawMessage	背书请求需发送的消息。
error	发送成功返回类型为nil，反之返回error。

步骤2 背书请求消息发送。

- 接口函数

```
func (action *ContractAction) Invoke(rawMsg *common.RawMessage) (*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述背书请求需发送的消息。

- 返回值

类型	说明
*common.RawMessage	背书请求需发送的消息。
error	发送成功返回类型为nil，反之返回error。

步骤3 交易消息构建。

- 接口方法

```
func (msg *ContractRawMessage) BuildTxRawMsg(rawMessages []*common.RawMessage) (*TxRawMsg, error)
```

- 参数说明

参数	类型	说明
rawMessages	[]*common.RawMessage	背书请求返回结果集合。

- 返回值

类型	说明
*TxRawMsg	包含交易hash的交易请求信息，该消息使用transaction接口发送。
error	构建成功返回类型为nil，反之返回error。

步骤4 交易消息发送。

- 接口方法

```
func (action *ContractAction) Transaction(rawMsg *common.RawMessage) (*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述生成的交易消息。

- 返回值

类型	说明
*common.RawMessage	用于获取包含发送结果的消息。
error	发送成功返回类型为nil，反之返回error。

----结束

3.3.4 利用合约查询数据

步骤1 查询请求消息构建

- 接口函数

```
func (msg *ContractRawMessage) BuildInvokeMessage(chainID string, name string, function string, args []string) (*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
chainID	string	链名称。
name	string	合约名称。
function	string	调用合约中的方法名。
args	[]string	合约方法参数。

- 返回值

类型	说明
*common.RawMessage	查询请求需发送的消息。
error	发送成功返回类型为nil，反之返回error。

步骤2 查询请求消息发送。

- 接口函数
func (action *ContractAction) Invoke(rawMsg *common.RawMessage) (*common.RawMessage, error)

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述查询请求需发送的消息。

- 返回值

类型	说明
*common.RawMessage	查询请求返回的消息。
error	发送成功返回类型为nil，反之返回error。

---结束

3.3.5 文件上链

- 接口方法
func (bc *BsClient) UploadFile(filePath, fileName string) (*UploadFileResponse, error)

- 参数说明

参数	类型	说明
filePath	string	待上链文件在本地的路径。当前支持不大于100MB的任意格式文件。
fileName	string	文件在链上的名称。

- 返回值

类型	说明
*UploadFileResponse	文件上链返回信息。
error	上链成功返回类型为nil，反之返回error。

3.3.6 文件下载

- 接口方法
func (bc *BsClient) DownloadFile(filePath, fileName string, versionId int) error
- 参数说明

参数	类型	说明
filePath	string	文件下载到本地的路径。
fileName	string	待下载文件在链上的名称。
versionId	int	待下载文件的版本号。

- 返回值

类型	说明
error	下载成功返回类型为nil，反之返回error。

3.3.7 其他查询

3.3.7.1 查询区块高

步骤1 消息构建。

- 接口方法
func (msg *QueryRawMessage) BuildLatestChainStateRawMessage(chainID string) (*common.RawMessage, error)
- 参数说明

参数	类型	说明
chainID	string	链名称。

- 返回值

类型	说明
*common.RawMessage	查询链状态需发送的消息
error	查询成功返回类型为nil，反之返回error。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
func (action *QueryAction) GetLatestChainState(rawMsg *common.RawMessage)
(*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述查询链状态消息。

- 返回值

类型	说明
*common.RawMessage	用于获取包含发送结果的消息。
error	发送成功返回类型为nil，反之返回error。

---结束

3.3.7.2 查询区块详情

步骤1 消息构建。

- 接口方法

```
func (msg *QueryRawMessage) BuildBlockRawMessage(chainID string, blockNum uint64)
(*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
chainID	string	链名称。
blockNum	uint64	区块高度。

- 返回值

类型	说明
*common.RawMessage	根据块高查询区块详情需发送的消息。
error	查询成功返回类型为nil，反之返回error。

步骤2 消息发送。

- 接口方法

```
func (action *QueryAction) GetBlockByNum(rawMsg *common.RawMessage)
(*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述根据块高查询区块详情消息。

- 返回值

类型	说明
*common.RawMessage	用于获取包含发送结果的消息。
error	发送成功返回类型为nil，反之返回error。

---结束

3.3.7.3 查询交易执行结果

步骤1 消息构建。

- 接口方法

```
func (msg *QueryRawMessage) BuildTxRawMessage(chainID string, txHash []byte)
(*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
chainID	string	链名称。
txHash	[]byte	交易Hash。

- 返回值

类型	说明
*common.RawMessage	查询交易执行结果需发送的消息。
error	构建成功返回类型为nil，反之返回error。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
func (action *QueryAction) GetTxResultByTxHash(rawMsg *common.RawMessage)
(*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述生成的查询指定交易执行结果的消息。

- 返回值

类型	说明
*common.RawMessage	用于获取包含发送结果的消息。
error	发送成功返回类型为nil，反之返回error。

----结束

3.3.7.4 利用交易 ID 查询交易详情

步骤1 消息构建。

- 接口方法

```
func (msg *QueryRawMessage) BuildTxRawMessage(chainID string, txHash []byte) (*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
chainID	string	链名称。
txHash	[]byte	交易哈希。

- 返回值

类型	说明
*common.RawMessage	根据交易ID查询交易详情需发送的消息。
error	构建成功返回类型为nil，反之返回error。

步骤2 消息发送。

- 接口方法

QueryAction.class

```
func (action *QueryAction) GetTxByHash(rawMsg *common.RawMessage) (*common.RawMessage, error)
```

- 参数说明

参数	类型	说明
rawMsg	*common.RawMessage	上述根据交易Hash查询交易详情消息。

- 返回值

类型	说明
*common.RawMessage	用于获取包含发送结果的消息。
error	发送成功返回类型为nil，反之返回error。

----结束

3.3.7.5 查询文件历史版本

- 接口方法
func (bc *BsClient) GetFileHistory(fileName string) ([]*bstore.FileHistory, error)
- 参数说明

参数	类型	说明
fileName	string	查询的链上文件名。

- 返回值

类型	说明
[]*FileHistory	文件历史版本信息列表，每条历史版本信息包含版本号、文件哈希值、首次上链时间、更新时间、上传者数据。
error	查询成功返回类型为nil，反之返回error。

3.3.7.6 查询文件操作记录

- 接口方法
func (bc *BsClient) GetFileOperation(fileName, startTime, endTime string) ([]*bstore.StorageEvent, error)
- 参数说明

参数	类型	说明
fileName	string	查询的链上文件名。
startTime	string	查询记录的起始时间（秒时间戳）。
endTime	string	查询记录的结束时间（秒时间戳）。

- 返回值

类型	说明
[]*StorageEvent	文件操作记录列表，每条操作记录包含操作者、操作类型、时间数据。
error	查询成功返回类型为nil，反之返回error。

3.4 SDK 升级

3.4.1 Java SDK 升级

为提高使用体验，推荐将SDK升级到最新版本。升级过程中需要进行修改的地方如下，具体可参考最新的Java应用程序开发的[示例Demo](#)进行修改。

- 从2.1.0.2.52升级至2.1.0.6.41
 - a. 修改pom文件

```

31 <dependencies>
32 <dependency>
33 <groupId>com.huawei.wienerchain</groupId>
34 <artifactId>wienerchain-java-sdk</artifactId>
35 <version>2.1.0.6.41</version>
36 </dependency>
37
    
```

- b. 修改合约发送、查询方法，详细参考下表。

表 3-1 更新内容

内容	旧版本 (2.1.0.2.52)	新版本(2.1.0.6.41)	SDK参考
合约调用方法	buildInvocation	buildInvokeRawMsg	利用合约发送交易
落盘消息构建方法	buildTransactionMessage	buildTxRawMsg	利用合约发送交易
合约调用信息构建方法	buildInvocation	buildInvokeRawMsg	利用合约查询数据

3.4.2 Go SDK 升级

1. 删除原有huaweichainSDK文件，下载新版本的SDK文件解压至原有的SDK文件路径。
2. SDK下载参见[SDK配置](#)。
3. 修改合约发送方法，详细参考下表。

表 3-2 更新内容

内容	旧版本(2.1.0.2.39)	新版本(2.1.0.6.41)	SDK参考
交易消息构建方法	BuildTransactionMessage	BuildTxRawMsg	利用合约发送交易

4 应用程序开发

4.1 概述

为了能在应用程序中使用区块链服务，可参考本章节完成应用程序的开发。

开发完成后，应用程序可以调用合约将业务数据发送到链上或从链上进行查询，以及查询区块链的块高、查看某个区块的详情和查看某笔交易的详情等。

4.2 Java 应用程序开发

须知

windows环境：java客户端仅支持64位操作系统与64位java sdk。

4.2.1 SDK 客户端配置

客户端配置如下：

1. SDK配置请参考[SDK配置](#)。
2. 华为链实例创建完成后，下载配置文件，具体请参考[下载配置文件](#)。

📖 说明

- 使用解压后的配置文件初始化SDK客户端，初始化方法请参考[SDK介绍](#)中的通用方法，配置示例可参考[示例Demo](#)。
- 配置文件主要包含证书文件和yaml配置，实例中每个组织对应一个yaml文件，可通过读取不同的yaml生成不同的SDK客户端对象。
- 客户端初始化需确保yaml文件中证书文件等路径为证书的实际存放路径。

4.2.2 SDK 客户端调用

SDK客户端配置后，即可调用SDK进行区块链相关的业务逻辑开发。

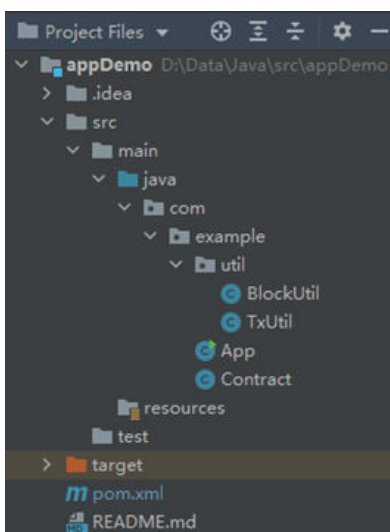
- 合约调用
使用合约的调用方法，具体请参考[利用合约发送交易](#)和[利用合约查询数据](#)。

- 其他SDK接口调用
 - 查询块高，具体请参考[查询块高](#)。
 - 查询区块详情，具体请参考[查询区块详情](#)。
 - 查询交易执行结果，具体请参考[查询交易执行结果](#)。
 - 利用交易ID查询交易，具体请参考[利用交易ID查询交易](#)。

4.2.3 示例 Demo

本节提供一个基于Java SDK的Demo，帮助开发自己的Java客户端应用程序。

可单击链接下载获取[Java应用程序Demo](#)，项目结构如下：



其中App文件即业务端调用的示例，Contract文件实现了通过合约对业务数据交互。对数据有修改的操作如插入和删除，需要调用其中的send方法。对数据的查询操作如查询某个键的历史，需要调用其中的query方法。

util文件夹提供了两个工具类，BlockUtil用于从区块上获取数据，TxUtil用于从交易上获取数据。

4.3 Go 应用程序开发

4.3.1 SDK 客户端配置

客户端配置如下：

1. SDK配置请参考[SDK配置](#)。
2. 华为链实例创建完成后，下载配置文件，具体请参考[下载配置文件](#)。

📖 说明

- 使用解压后的配置文件初始化SDK客户端，初始化方法请参考[SDK介绍](#)中的通用方法，配置示例可参考[示例Demo](#)。
- 配置文件主要包含证书文件和yaml配置，实例中每个组织对应一个yaml文件，可通过读取不同的yaml生成不同的SDK客户端对象。
- 客户端初始化需确保yaml文件中证书文件等路径为证书的实际存放路径。

4.3.2 SDK 客户端调用

SDK客户端配置后，即可调用SDK进行区块链相关的业务逻辑开发。

- 合约调用
使用合约的调用方法，具体请参考[利用合约发送交易](#)和[利用合约查询数据](#)。
- 其他SDK接口调用
 - 利用交易ID查询交易，具体请参考[利用交易ID查询交易](#)。
 - 查询区块高度，具体请参考[查询区块高度](#)。
 - 查询区块详情，具体请参考[查询区块详情](#)。
 - 查询交易执行结果，具体请参考[查询交易执行结果](#)。

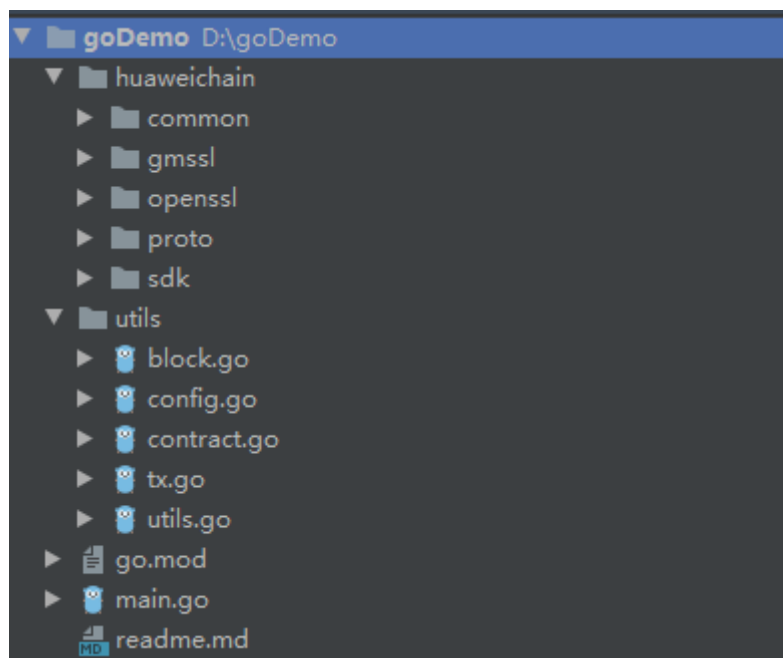
4.3.3 示例 Demo

须知

应用程序开发需要使用go mod，因此请确保GO111MODULE为on、镜像源配置。请确保可正常访问[华为云镜像网站](#)，环境设置命令如下

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://repo.huaweicloud.com/repository/goproxy/
go env -w GONOSUMDB=*
```

示例Demo基于Go SDK开发，主要用于帮助开发人员理解并开发Go客户端应用程序，[示例Demo下载链接](#)。



Demo目录主要分为：

1. main.go文件，为Go语言客户端主程序。
2. huaweichain文件夹，Go语言客户端SDK，相关介绍请参考[Go SDK介绍](#)。

3. utils文件夹，主要包含常见的SDK调用示例，详细内容可参照readme.md文件(其中utils/config.go为客户端配置文件，需要根据实际情况进行修改)。