**ScryInfo** 协议层**SDK**接

⼝⽂档 **v0.0.5**

⽬录

[Scry协议层业务流程 1](#_TOC_250020)

[接⼝ 1](#_TOC_250019)

[Sdk 1](#_TOC_250018)

[ScryClient - Scry客户端操作 1](#_TOC_250017)

[Contract - 合约操作 2](#_TOC_250016)

[Key Manager - 密钥管理接⼝ 4](#_TOC_250015)

[Event - 合约事件 4](#_TOC_250014)

[例程 5](#_TOC_250013)

[调⽤SDK的第⼀步：初始化SDK 5](#_TOC_250012)

[第⼆步：创建普通账户，并从超级账户转⼊Eth和Token，以便交易 5](#_TOC_250011)

[下⾯可以开始交易了 6](#_TOC_250010)

[卖⽅发布数据 6](#_TOC_250009)

[买⽅收到数据发布通知，取到数据的描述数据ID，据此买⽅可以从IPFS上下载数据描述信息 7](#_TOC_250008)

[看到数据描述信息后，买⽅对该数据很感兴趣，于是设定待⽀付押⾦额度，允许协议层智能合约从其账户转移Token⽤作押⾦ 7](#_TOC_250007)

[买⽅准备购买，智能合约从买⽅账户扣除押⾦ 7](#_TOC_250006)

[买⽅得到数据证明ID，凭借这个ID，买⽅可以从IPFS上下载这些数据证明，判断数据是不是⾃⼰想要的 8](#_TOC_250005)

[买⽅决定正式购买数据 8](#_TOC_250004)

[卖⽅收到购买数据的通知，⽣成使⽤买⽅公钥加密的meta data id，发给合约 8](#_TOC_250003)

[买⽅拿到meta data id，终于可以从IPFS上下载完整数据 9](#_TOC_250002)

[买⽅看完数据，反馈数据的真实性 9](#_TOC_250001)

[交易关闭通知 10](#_TOC_250000)

# Scry协议层业务流程



Start

Buyer prepares to purchase

Buyer download the complete meta data

Buyer transfers tokens to contract for paying to seller, and download the proof data(data fragments)

.

Y

Buyer thinks the meta data is fake?

N

Buyer need to decide: buy or not?

Contract pay tokens to seller.

Return buyer his deposit.

Y

N

Contract return tokens back to buyer

Close Transaction.

End

Seller publishs data: meta data, despcription data, proofs of data.

Proofs of data are data fragments which aims to prove the

truth of data.

# 接⼝

## Sdk

Package: sdk

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | Init(  ethNodeAddr string, keyServiceAddr string, protocolAddr string, tokenAddr string, fromBlock uint64, ipfsNodeAddr string)  (error) | 在使⽤scryinfo协议层SDK之前，需要对整个SDK进⾏初始化。 |
|  | ethNodeAddr : 以太坊节点地址，由主办者提供keyServiceAddr: 密钥服务地址，由主办者提供protocolAddr : 协议智能合约的地址，由主办者提供tokenAddr : token智能合约的地址，由主办者提供f romBlock : event扫描的起始区块，⼀般默认为0 ipfsNodeAddr : IPFS节点地址，由主办者提供 |

## ScryClient - Scry客户端操作

Package: sdk/scryclient

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | CreateScryClient( password string  ) (\*ScryClient, error) | 接⼝调⽤成功后，会创建⼀个ScryClient实例(如名为scryClient)， 包括创建⽤户的公私钥账户，私钥经加密后保存。  ⽤户访问scryClient.Account.Address能够得到⽤户的账户地址。 |
|  | password: 待创建账户的密码，请牢记。 |
| 2 | SubscribeEvent( eventName string, callback  chainevents.EventCallback) | 当前ScryClient通过调⽤此接⼝，可订购链上指定事件(event)。当event到来时，会触发指定的callback函数。可以订购的event类型 有： |
|  | DataPublish TransactionCreate Buy ReadyForDownload TransactionClose Approval |
| 3 | Authenticate( password string  ) (bool, error) | 验证⽤户密码是否正确  password: 待验证密码 |
| 4 | TransferEthFrom(  from common.Address, password string, value \*big.Int  ) (error) | 从地址为from的账户向本client账户进⾏ETH转账。 |
|  | f rom: 源账户  password: 源账户密码  value: 待转账数额，单位为1 wei |
| 5 | TransferTokenFrom(  from common.Address, password string, value \*big.Int  ) (error) | 从地址为from的账户向本client账户进⾏Token转账。 |
|  | f rom: 源账户  password: 源账户密码，⽤作交易签名  value: 待转账数额，单位为1 token |
| 6 | GetEth(  owner common.Address  ) (\*big.Int, error) | 获取账户地址为owner的ETH数额，单位为wei。  owner: 账户地址 |
| 7 | GetScryToken(  owner common.Address  ) (\*big.Int, error) | 获取账户地址为owner的Token数额，单位为1 token。owner: 账户地址 |

## Contract - 合约操作

Package: sdk/scryclient/chaininterfacewrapper

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | Publish(  txParams  \*op.TransactParams, price \*big.Int, metaData []byte, proofDatas [][]byte, proofNum int,  descriptionData []byte  ) (string, error) | 发布数据到IPFS上，并将IPFS返回的资源id保存到区块链合约中。 |
|  | 该接⼝第⼀个返回值为publishId，类型为string。  txParams : 区块链交易参数。  price : 数据售价，单位为1 token metaData : 待售数据  proof Datas : 数据证明⽚段，⽤来证明所发布的数据的真实性  proof Num : 数据证明⽚段的数组⼤⼩  descriptionData : 数据描述，如title, keyword等。具体意义由应  ⽤层决定。 |
|  | 当数据成功发布到IPFS和区块链上，链会发送DataPublish事件给 所有⽤户。  DataPublish事件以Json格式定义，参见Event接⼝ |
| 2 | ApproveTransfer( txParams  \*op.TransactParams, spender common.Address, value \*big.Int  ) (error) | 准许地址spender从本接⼝调⽤⽅地址进⾏token转账， |
|  | txParams : 区块链交易参数  spender : 被允许的转账发起地址  value : 被允许的转账额度, 单位为1 token，要⼤于或等于想购买的数据的价格 |
|  | 在这个过程中，链收到请求后，会发送Approval事件给调⽤⽅。参见Event接⼝。 |
| 3 | PrepareToBuy( txParams  \*op.TransactParams, publishId string  ) (error) | 买⽅准备购买数据，希望得到待买数据的证明数据ID。  在调⽤本接⼝之前，买⽅需先调⽤ApproveTransfer接⼝，允许合 约从买⽅地址转出押⾦。 |
|  | txParams : 区块链交易参数  publishId : 同 上 |
|  | 链收到请求后，会发送TransactionCreate事件给买⽅。  TransactionCreate事件以Json格式定义，参见Event接⼝。 |
| 4 | BuyData(  txParams  \*op.TransactParams, txId \*big.Int  ) (error) | 买⽅正式购买数据。 |
|  | txParams : 区块链交易参数  txId : 本次交易ID，由TransactionCreate事件的transactionId 字段取得。 |
|  | 链收到请求后，会发送Buy事件给卖⽅。  Buy事件以Json格式定义，参见Event接⼝。 |
| 5 | SubmitMetaDataIdEncWithBuy er(  txParams  \*op.TransactParams, txId \*big.Int, encyptedMetaDataId  []byte  ) (error) | 卖⽅上传待卖数据的metaDataId到区块链上，该ID使⽤买⽅公钥加密。 |
|  | txParams : 区块链交易参数  txId : 本次交易ID，由Buy事件的transactionId字段取得  encyptedMetaDataId : 使⽤买⽅公钥加密的metaDataId |
|  | 链收到请求后，会发送ReadyForDownload事件给买⽅。  买⽅得到metaDataId并以私钥解密，得到真正的meta data ID， 就可以从IPFS上下载数据了。  ReadyForDownload事件以Json格式定义，参见Event接⼝。 |
| 6 | ConfirmDataTruth( txParams  \*op.TransactParams, txId \*big.Int, truth bool  ) (error) | 买⽅确认数据真实性。 |
|  | txParams : 区块链交易参数  txId : 本次交易ID，由Buy事件的transactionId字段取得  t ruth : 数据是否真实 |
|  | 链收到请求后，会发送TransactionClose事件给所有⽤户，并关闭交易。  若数据为真，合约会将买⽅押⾦⽀付给卖⽅。  TransactionClose事件以Json格式定义，参见Event接⼝。 |

## Key Manager - 密钥管理接⼝

Package: sdk/util/accounts

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | Encrypt(  plainText []byte, address string)  ([]byte, error) | 使⽤⽤户地址为address的公钥对明⽂plainText进⾏加密  plainText: 明⽂数据  address : ⽤户地址 |
| 2 | Decrypt(  cipherText []byte, address string, password string)  ([]byte, error) | 使⽤⽤户地址为address的私钥对密⽂cipherText进⾏解密  cipherText : 密 ⽂address : ⽤户地址password : ⽤户密码 |
| 3 | ReEncrypt(  cipherText []byte, address1 string,  address2 string, password string)  ([]byte, error) | 对已加密数据cipherText进⾏解密，然后再使⽤第⼆个⽤户地址的 公钥进⾏加密。  cipherText: 密⽂数据address1 : 解密⽤户地址address2 : 加密⽤户地址password: 解密⽤户密码 |

## Event - 合约事件

所有的Event都以JSON格式进⾏传输。

|  |  |  |
| --- | --- | --- |
| **seq** | **event name** | **example** |
| 1 | DataPublish | {  "despDataId":"QmhKnroYBFp", //数据描述ID，依据此ID可从IPFS取得待售数据的描述  "price":1000, //待售数据价格  "publishId":"155212056", //本次发布的ID，与Publish()第⼀个返回值相同"seqNo":"155212057",  "users":["0xxxx"]  } |
| 2 | Approval | {  "owner":"0x3Ab0dAA324", //授权⼈"spender":"0x3c4d26E91", //被授权⼈"value":1600 //已授权额度  } |
| 3 | TransactionCreate | {  "proof Ids":[[211,159],[170,49]], //证明数据ID "publishId":"1552121", // 数 据 发 布 ID "seqNo":"15521219",  "transactionId":4, //本次交易ID  "users":["0xxxx"]  } |

|  |  |  |
| --- | --- | --- |
| **seq** | **event name** | **example** |
| 4 | Buy | {  "metaDataIdEncSeller":"SdWeISqpSyhGA==", //使⽤卖⽅公钥加密的元数据ID  "publishId":"1552121827295982593-913139", //数据发布ID  "seqNo":"1552121924035218179-67679122",  "transactionId":4, //本次交易ID "users":["0xxxx"]  } |
| 5 | ReadyForDownload | {  "metaDataIdEncBuyer":"tql/mAZY/z0aR2g==", //使⽤买⽅公钥加密的元数据ID  "seqNo":"1552126832692923243-89400636",  "transactionId":4, //本次交易ID "users":["0xxxx"]  } |
| 6 | TransactionClose | {  "seqNo":"1552126873209063970-5512851552272651527",  "transactionId":4, //本次交易ID "users":["0xxxx"]  } |

# 例程

## 调⽤SDK的第⼀步：初始化SDK

1

2

3

4

5

6

7

8

9

10

11

12

13

14

func main() {

err := sdk.Init( "http://127.0.0.1:8545/",

"192.168.1.6:48080",

protocolContractAddr, tokenContractAddr,

0,

"/ip4/127.0.0.1/tcp/5001")

//ethereum node address

//key service host

//protocol contract address

//token contract address

//which block that scans events from

//ipfs host

if

err != nil {

fmt.Println("failed to initialize sdk, error:", err) return

}

}

## 第⼆步：创建普通账户，并从超级账户转⼊Eth和Token，以便交易

系统内置了超级账户，账户地址和密码见例⼦代码。

普通账户进⾏交易需要消耗gas和token，因此必须先从超级账户转⼊ETH和token。

1

2

3

4

5

6

7

8

9

10

//⽤户密码

clientPassword := "888888"

//超级账户地址，此为系统内置，地址是固定的

suAddress := "0xd280b60c38bc8db9d309fa5a540ffec499f0a3e8"

//超级账户密码，此为系统内置，密码是固定的

suPassword := "111111"

func CreateClientWithETHToken( token \*big.Int,

eth \*big.Int) (\*scryclient.ScryClient, error) {

|  |  |  |
| --- | --- | --- |
| 11  12 |  | client, err := scryclient.CreateScryClient(clientPassword  if err != nil { |
| 13 |  | fmt.Println("failed to create user, error:", err) |
| 14 |  | return nil, err |
| 15 |  | } |
| 16 |  |  |
| 17 |  | //转⼊ETH |
| 18 |  | \_, err = cif.TransferEth( |
| 19 |  | common.HexToAddress(suAddress), |
| 20 |  | suPassword, |
| 21 |  | common.HexToAddress(client.Account.Address), |
| 22 |  | eth) |
| 23 |  | if err != nil { |
| 24 |  | fmt.Println("failed to transfer Eth, error:", err) |
| 25 |  | return nil, err |
| 26 |  | } |
| 27 |  |  |
| 28 |  | //转⼊Token |
| 29 |  | txParam := chainoperations.TransactParams{ |
| 30 |  | From: common.HexToAddress(suAddress), |
| 31 |  | Password: suPassword, |
| 32 |  | } |
| 33 |  | err = cif.TransferTokens( |
| 34 |  | &txParam, |
| 35 |  | common.HexToAddress(client.Account.Address), |
| 36 |  | token) |
| 37 |  | if err != nil { |
| 38 |  | fmt.Println("failed to transfer token, error:", err) |
| 39 |  | return nil, err |
| 40 |  | } |
| 41 |  |  |
| 42 |  | return client, nil |
| 43 | } |  |

## 下⾯可以开始交易了

)

卖⽅发布数据

1. func SellerPublishData(supportVerify bool) {
2. seller.SubscribeEvent("DataPublish", onPublish) 3
3. //待发布的元数据
4. metaData := []byte("pussy poppy")
5. //⼀些元数据⽚段: ⽤以证明元数据的真实性

7 proofData := [][]byte{{'4', '5', '6', '3'}, {'2', '2', '1'}}

8 //元数据的描述数据

9 despData := []byte{'7', '8', '9', '3'}

10

1. txParam := chainoperations.TransactParams{
2. From: common.HexToAddress(seller.Account.Address),
3. Password: keyPassword, 14 }

15

1. cif.Publish(
2. &txParam,
3. big.NewInt(1000),
4. metaData,
5. proofData,

21 2,

22 despData, 23 )

24 }

买⽅收到数据发布通知，取到数据的描述数据ID，据此买⽅可以从IPFS上下载数据描述信息

|  |  |  |
| --- | --- | --- |
| 1  2 | func onPublish(event events.Event) bool {  //数据的发布ID | |
| 3 |  | publishId = event.Data.Get("publishId").(string) |
| 4 |  | //描述数据ID |
| 5 |  | despDataId := event.Data.Get("despDataId").(string) |
| 6 |  | //价格 |
| 7 |  | price := event.Data.Get("price").(\*big.Int) |
| 8 |  | return true |
| 9 | } |  |

看到数据描述信息后，买⽅对该数据很感兴趣，于是设定待⽀付押⾦额度，允许协议层智能合约从其账户转移Token⽤作押⾦

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | func BuyerApproveTransfer() {  //买⽅若收到onApprovalBuyerTransfer通知，表⾯approve成功  buyer.SubscribeEvent("Approval", onApprovalBuyerTransfer)  //buyerPassword: 买⽅账户密码  txParam := chainoperations.TransactParams{ | |
| 7 |  | From: common.HexToAddress(buyer.Account.Address), |
| 8 |  | Password: buyerPassword, |
| 9 |  | } |
| 10 |  |  |
| 11 |  | //protocolContractAddr是协议层合约地址, 押⾦数额要⼤于等于数据价格,否则购买操作会失败 |
| 12 |  | err := cif.ApproveTransfer(&txParam, |
| 13 |  | common.HexToAddress(protocolContractAddr), |
| 14 |  | big.NewInt(1000)) |
| 15 |  |  |
| 16 |  | if err != nil { |
| 17 |  | fmt.Println("BuyerApproveTransfer:", err) |
| 18 |  | } |
| 19 | } |  |

买⽅准备购买，智能合约从买⽅账户扣除押⾦

1

2

3

4

func PrepareToBuy(publishId string) { buyer.SubscribeEvent("TransactionCreate", onTransactionCreate)

txParam := chainoperations.TransactParams{

|  |  |  |
| --- | --- | --- |
| 5 |  | From: common.HexToAddress(buyer.Account.Address), |
| 6 |  | Password: buyPassword, |
| 7 |  | } |
| 8 |  |  |
| 9 |  | err := cif.PrepareToBuy(&txParam, publishId) |
| 10 |  | if err != nil { |
| 11 |  | fmt.Println("failed to prepareToBuy, error:", err) |
| 12 |  | } |
| 13 | } |  |

买⽅得到数据证明ID，凭借这个ID，买⽅可以从IPFS上下载这些数据证明，判断数据是不是⾃⼰想要的

|  |  |  |
| --- | --- | --- |
| 1  2 | func onTransactionCreate(event events.Event) bool { | |
| 3 |  | //本次交易ID |
| 4 |  | txId = event.Data.Get("transactionId").(\*big.Int) |
| 5 |  |  |
| 6 |  | //数据证明IDs |
| 7 |  | proofIDs := event.Data.Get("proofIds").([][32]byte) |
| 8 |  |  |
| 9 |  | return true |
| 10 | } |  |
| 11 |  |  |

买⽅决定正式购买数据

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | func Buy(txId \*big.Int) {  //卖⽅需要订购Buy事件，这样才能收到买⽅的购买消息  seller.SubscribeEvent("Buy", onPurchase)  //buyerPassword是买⽅账户密码  txParam := chainoperations.TransactParams{ | |
| 7 |  | From: common.HexToAddress(buyer.Account.Address), |
| 8 |  | Password: buyerPassword, |
| 9 |  | } |
| 10 |  |  |
| 11 |  | //txId为本次交易ID |
| 12 |  | err := cif.BuyData(&txParam, txId) |
| 13 |  |  |
| 14 |  | if err != nil { |
| 15 |  | fmt.Println("failed to buyData, error:", err) |
| 16 |  | } |
| 17 | } |  |

卖⽅收到购买数据的通知，⽣成使⽤买⽅公钥加密的meta data id，发给合约

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

func onPurchase(event events.Event) bool {

//使⽤卖⽅公钥加密的meta data id

metaDataIdEncWithSeller = event.Data.Get("metaDataIdEncSeller").([]byte)

var err error

//metaDataIdEncWithBuyer： 使⽤买⽅公钥加密的meta data id

//metaDataIdEncWithSeller: ⽤卖⽅公钥加密的meta data id

//sellerPassword: 卖家密码

metaDataIdEncWithBuyer, err = accounts.GetAMInstance().ReEncrypt( metaDataIdEncWithSeller,

seller.Account.Address, buyer.Account.Address, sellerPassword,

)

if err != nil {

fmt.Println("failed to ReEncrypt meta data id with buyer's public key") return false

|  |  |  |
| --- | --- | --- |
| 20 | } |  |
| 21 |  |
| 22 | SubmitMetaDataIdEncWithBuyer(txId) |
| 23 | return true |
| 24 | } |
| 25 |  |
| 26 | func SubmitMetaDataIdEncWithBuyer(txId \*big.Int) { |
| 27 | //买⽅需要监听ReadyForDownload事件 |
| 28 | buyer.SubscribeEvent("ReadyForDownload", onReadyForDownload) |
| 30 | txParam := chainoperations.TransactParams{ |
| 31 | From: common.HexToAddress(seller.Account.Address), |
| 32 | Password: sellerPassword, |
| 33 | } |
| 34 |  |
| 35 | err := cif.SubmitMetaDataIdEncWithBuyer( |
| 36 | &txParam, |
| 37 | txId, |
| 38 | metaDataIdEncWithBuyer) |
| 39 | if err != nil { |
| 40 | fmt.Println("failed to SubmitMetaDataIdEncWithBuyer, error:", | err) |
| 41 | } |  |
| 42 | } |  |

买⽅拿到meta data id，终于可以从IPFS上下载完整数据

29

1

2

3

4

func onReadyForDownload(event events.Event) bool { metaDataIdEncWithBuyer = event.Data.Get("metaDataIdEncBuyer").([]byte)

//买⽅解密metaDataIdEncWithBuyer，得到原始meta data id，根据这个id从IPFS上下载真正的数

据

5

6

7

8

9

10

11

metaDataId, err := accounts.GetAMInstance().Decrypt( metaDataIdEncWithBuyer,

buyer.Account.Address, buyerPassword)

if

err != nil {

fmt.Println("failed to decrypt meta data id with buyer's private key",

err)

12

13

14

15

16

return false

}

return true

}

买⽅看完数据，反馈数据的真实性

1

2

3

4

5

6

7

8

9

10

11

12

func ConfirmDataTruth(txId \*big.Int) { buyer.SubscribeEvent("TransactionClose", onClose)

txParam := chainoperations.TransactParams{

From: common.HexToAddress(buyer.Account.Address), Password: buyerPassword,

}

//txId: 本次交易ID

err

:= cif.ConfirmDataTruth( &txParam,

txId,

|  |  |  |
| --- | --- | --- |
| 13 |  | true) |
| 14 |  | if err != nil { |
| 15 |  | fmt.Println("failed to ConfirmDataTruth, error:", err) |
| 16 |  | } |
| 17 | } |  |

交易关闭通知

1. func onClose(event events.Event) bool {
2. fmt.Println("onClose:", event)
3. return true 4 }