PDX UTOPIA
应用开发手册

版本 1.1.6

目　　录

1 概述

自主创新、专利保护的 PDX Utopia 区块链协议栈，成功解决传统区块链性能较差、不能支持私密应用、不能多链跨链的问题。PDX Utopia 从算法和架构上支持大规模、高性能，以及全链共识的私密智能合约。PDX Utopia 区块链的一些颠覆性创新包括：

1) 世界上第一个异步共识算法（2019/2/16 在 Github 公布），在安全公平和多数共识的基础上做到 O(n) 复杂度，从算法上支持大规模、低延迟、高并发、高吞吐。
2) 专利保护的、大规模 PDX 账本算法，解决账本随时间推移造成的可用性和效率问题。
3) 专利保护的、并行+异步区块链架构，架构上支持低延迟、高并发、高吞吐。
4) 专利保护的智能合约架构，支持实现具有全链共识的私密合约应用。
5) 专利保护的交易依赖机制，支持实现多方可信的“工作流”。

为了最大程度的降低/消除客户迁移的成本，PDX Utopia 区块链兼容超级账本 Chaincode 合约规范，以太坊的 Solidity 合约规范和 eWASM 合约规范，并兼容以太坊钱包、ERC20，方便资产交易等生态对接。PDX Utopia 区块链支持高性能的 GOLANG 动态库合约，并内生支持性能无损的跨链积分共享，方便形成多链共生的生态。

一个 PDX Utopia 区块链节点，可以加入 PDX Unity 区块链 IaaS 平台，实现“一键式”建链、“一键式”智能合约部署，甚至形成多链跨链的融合生态。

2 开发依赖

2.1 客户端：Java 语言

```
<dependency>
   <groupId>ltd.pdx.driver</groupId>
   <artifactId>utopia-driver</artifactId>
   <version>1.3.9.2</version>
</dependency>
<repository>
   <id>releases</id>
   <name>Releases</name>
   <url>https://repo.pdx.ltd/nexus/content/repositories/releases/</url>
</repository>
```

2.2 客户端：GO 语言

```
require (
    github.com/ethereum/go-ethereum v1.10.14
    github.com/golang/protobuf v1.5.2
    github.com/hyperledger/fabric-chaincode-go v0.0.0-20210718160520-38d29fabecb9
    github.com/hyperledger/fabric-protos-go v0.0.0-20211118165945-23d738fc3553
    golang.org/x/crypto v0.0.0-20211215153901-e495a2d5b3d3
    golang.org/x/net v0.0.0-20211216030914-fe4d6282115f
    google.golang.org/grpc v1.43.0
)
```

2.3 Chaincode 智能合约：Java 语言

```
<dependency>
        <groupId>org.hyperledger.fabric-chaincode-java</groupId>
        <artifactId>fabric-chaincode-shim</artifactId>
        <version>1.4.8.2</version>
</dependency>
<repository>
    <id>releases</id>
    <name>Releases</name>
    <url>https://repo.pdx.ltd/nexus/content/repositories/releases/</url>
</repository>
```

2.4 Chaincode 智能合约：GO 语言

```
require (
    github.com/hyperledger/fabric-chaincode-go
    github.com/hyperledger/fabric-protos-go
)
```

2.5 DynNative 智能合约：GO 语言

```
require (
    github.com/PDXbaap/utopia_spi v1.0.0
)
```

3 Solidity 合约

使用您喜爱的 Solidity 工具，例如 http://remix.ethereum.org/开发 solidity d-App 并将其部署到 PDX Utopia 区块链实例。

PDX Utopia 支持的 Solidity 版本号： 0.4.0 - 0.8.11。

3.1 开发

请参考官方文档 https://solidity-cn.readthedocs.io/zh/develop/。

合约示例：

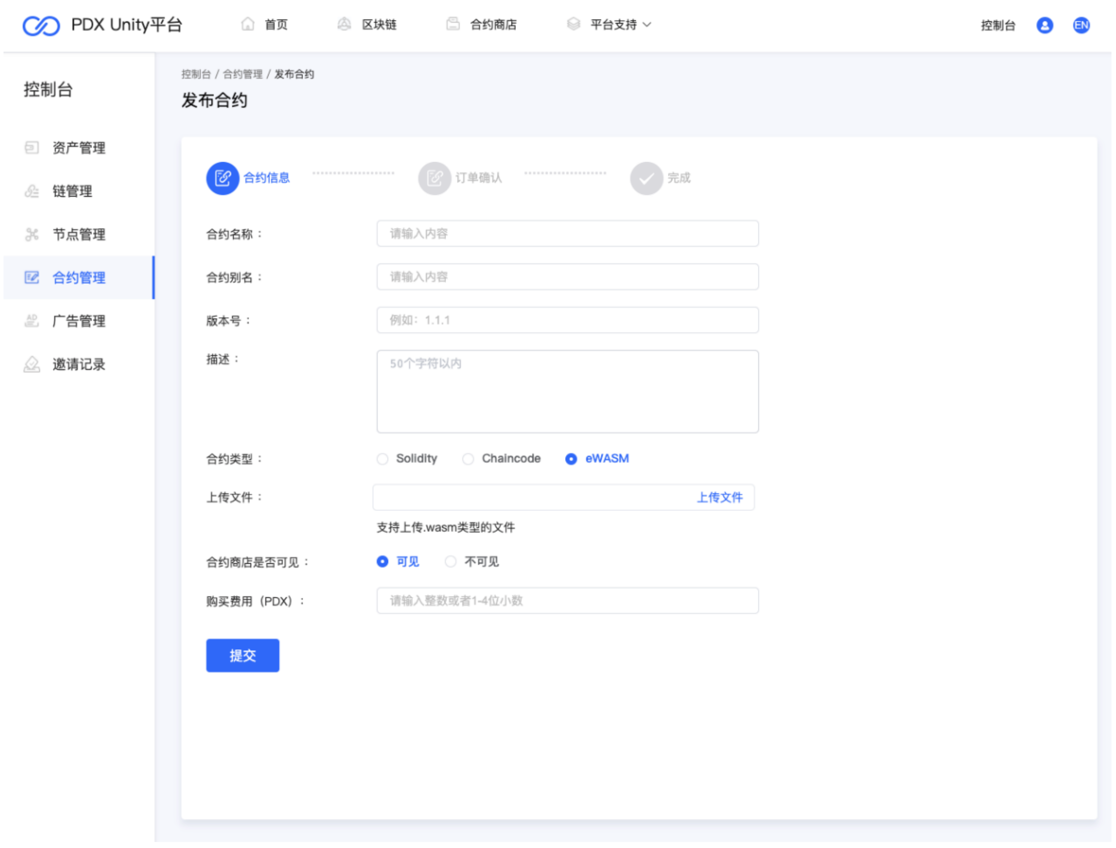
```
pragma solidity ^0.5.0;
contract Solidity_Sample{
    mapping(bytes => bytes)storageContent;
    address _ower;
    event Put(address indexed sender,bytes key,bytes value);
    event Destroy(address indexed sender);
    constructor() public    {
      _ower = msg.sender;
     }
     modifier only_ower(){
        require(_ower == msg.sender,"You are not the owner of this");
        _;
     }
     function put(bytes memory key,bytes memory value) public    {
        storageContent[key] = value;
        emit Put(msg.sender,key,value);
     }
     function get(bytes memory key) public view returns(bytes memory value){
         value = storageContent[key];
         return value;
     }
     function destroy() public only_ower{
          selfdestruct(msg.sender);
          emit Destroy(msg.sender);
     }
}
```

3.2 部署

3.2.1 通过 PDX Unity 部署

3.2.1.1 发布合约

如果您的 PDX Utopia 区块链实例是 PDX Unity 可信数字平台的一部分，您可以通过 PDX Unity 发布 Solidity 智能合约、eWASM 智能合约和 Chaincode 合约。操作界面如下图：

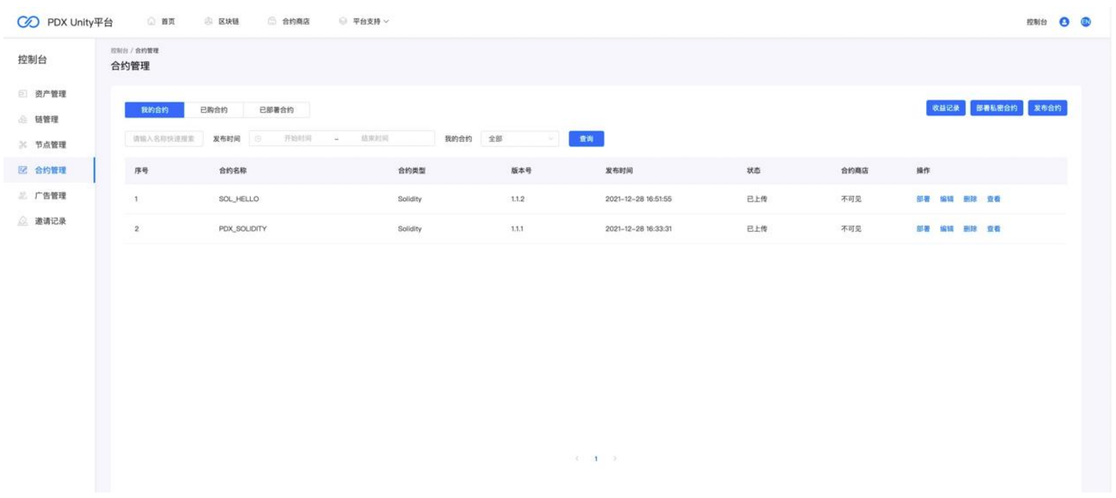


合约名称：	请输入内容
合约别名：	请输入内容
版本号：	例如：1.1.1
描述：	50个字符以内
合约类型：	Solidity Chaincode [✓ eWASM]
上传文件：	上传文件
	支持上传.wasm类型的文件
合约商店是否可见：	[✓ 可见] 不可见
购买费用（PDX）：	请输入整数或者1-4位小数

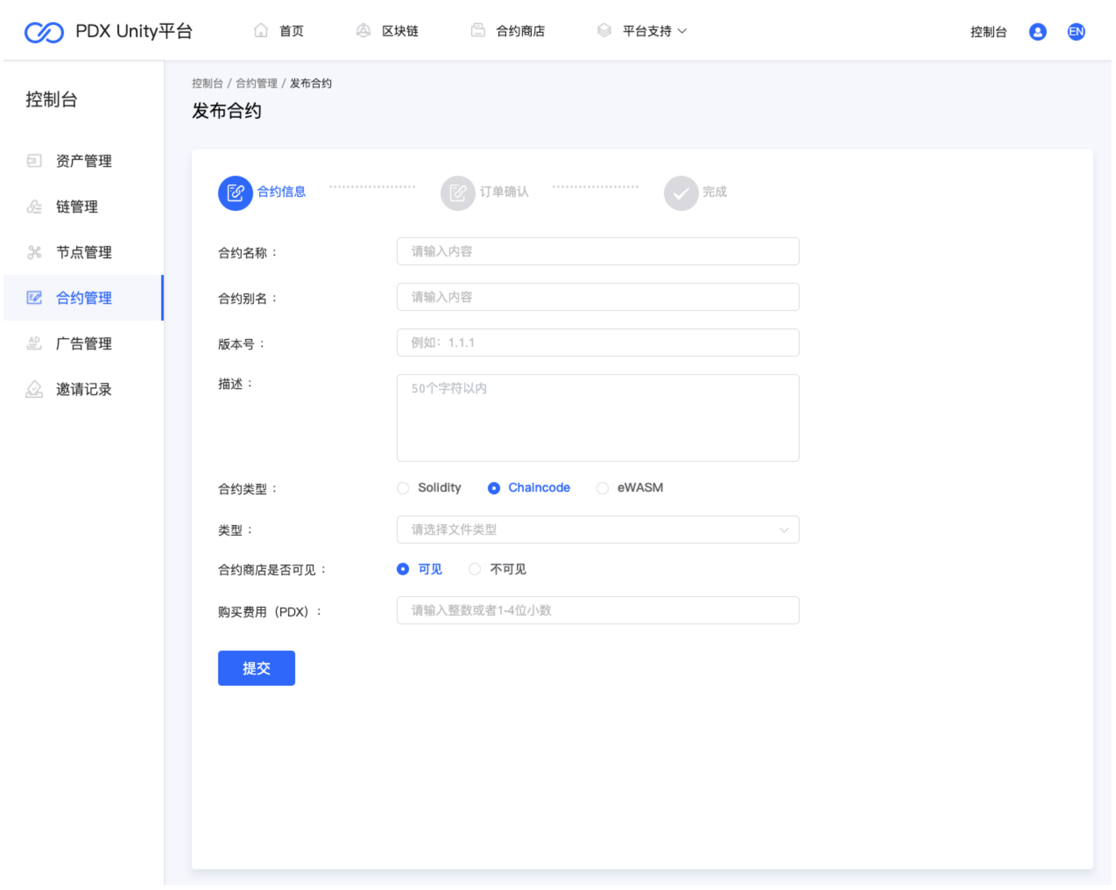
3.2.1.2 部署合约

点击“部署”，可以进行合约的部署。选择合约文件中要部署的合约、合约部署参数（如有）、所属链，进行部署。

注意：如果是公有链，部署合约需要用户在该条链上有该链的积分，否则会提示“余额不足”。如果是联盟链且配置了相应权限，则需要有对应的授权，请参考《PDX UTOPIA 联盟链配置手册》。



序号	合约名称	合约类型	版本号	发布时间	状态	合约商店	操作
1	SOL_HELLO	Solidity	1.1.2	2021-12-28 16:51:55	已上传	不可见	部署 编辑 删除 查看
2	PDX_SOLIDITY	Solidity	1.1.1	2021-12-28 16:33:31	已上传	不可见	部署 编辑 删除 查看

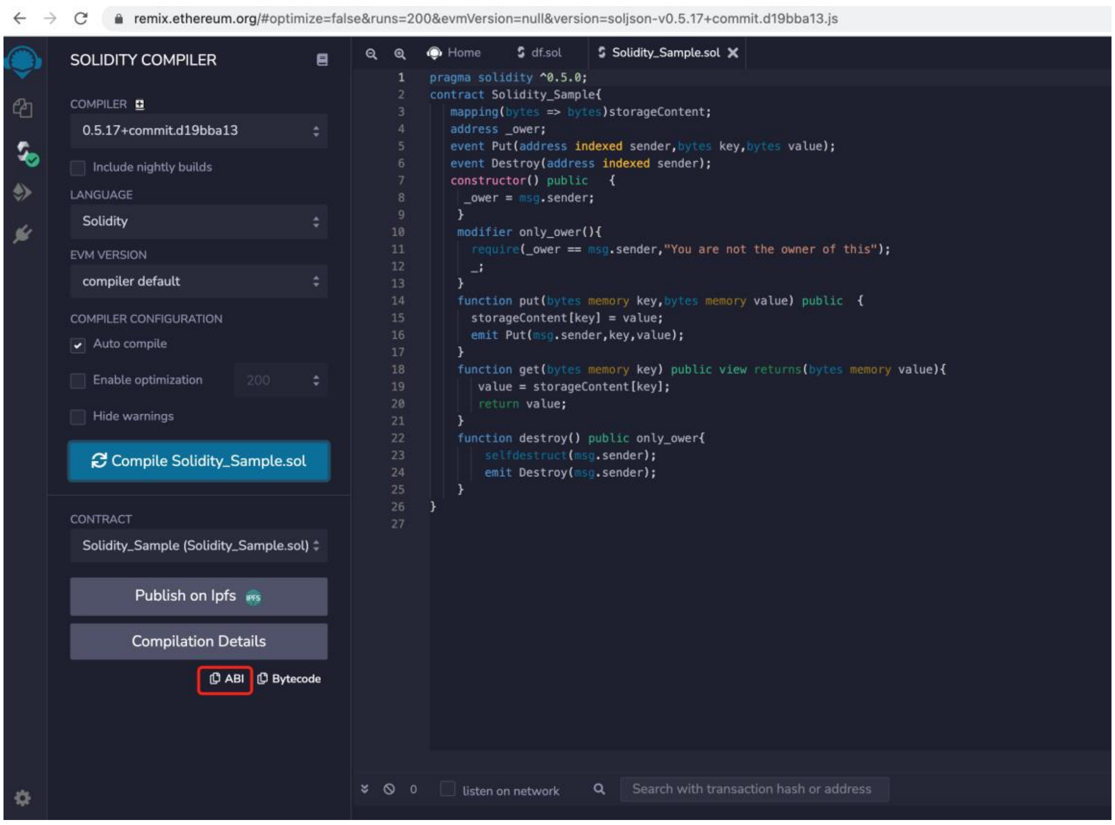


合约名称：	请输入内容
合约别名：	请输入内容
版本号：	例如：1.1.1
描述：	50个字符以内
合约类型：	Solidity [✓ Chaincode] eWASM
类型：	请选择文件类型
合约商店是否可见：	[✓ 可见] 不可见
购买费用（PDX）：	请输入整数或者1-4位小数

3.2.2 编程方式部署

3.2.2.1 合约编译

Solidity 智能合约，可以通过 http://remix.ethereum.org/ 编程并获得 ABI（Application Binary Interface）。



```
pragma solidity ^0.5.0;
contract Solidity_Sample{
   mapping(bytes => bytes)storageContent;
   address _ower;
   event Put(address indexed sender,bytes key,bytes value);
   event Destroy(address indexed sender);
   constructor() public   {
     _ower = msg.sender;
    }
    modifier only_ower(){
      require(_ower == msg.sender,"You are not the owner of this");
      _;
    }
    function put(bytes memory key,bytes memory value) public  {
      storageContent[key] = value;
      emit Put(msg.sender,key,value);
    }
    function get(bytes memory key) public view returns(bytes memory value){
       value = storageContent[key];
       return value;
    }
    function destroy() public only_ower{
        selfdestruct(msg.sender);
        emit Destroy(msg.sender);
    }
}
```

3.2.2.2 Java 语言

```
/**
 * deploySol
 *
 * @throws BlockchainDriverException
 */
public void deploySol() throws BlockchainDriverException {
    // Solidity 合约的 bytecode，例如 0xAB…CD
    byte[] payload = Hex.decode("bytecode-of-solidity-contract");
    String txId = driver.deploy(payload);
    System.out.println("txId:" + txId);
}
```

3.2.2.3 GO 语言

```
import (
    "client_Sample/privKeys"
    "client_Sample/tool"
    "context"
    "encoding/hex"
    "fmt"
```

```
        "github.com/ethereum/go-ethereum/core/types"
        "github.com/ethereum/go-ethereum/crypto"
        "math/big"
)
func main() {
// 区块链节点的 JSON RPC 地址（不包括 path）
        var host = "http://1.13.251.80:8545"
        client, err := tool.ToolConnect(host)
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        priKey, err := crypto.HexToECDSA(privKeys.PrivKeys[0])
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        from := crypto.PubkeyToAddress(priKey.PublicKey)
        fmt.Println("from:", from.String())
        nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        // Solidity 合约的 bytecode，例如 0xAB···CD
        code, err := hex.DecodeString("bytecode-of-solidity-contract")

        //如果 genesis.json 文件配置 blocksize,则不需要预估 gas
        var (
                gas         uint64 = 0
                gasPrice    big.Int = new(big.Int)
        )
        /*
                //如果 genesis.json 文件没有配置 blocksize，则需要预估 gas
                msg := ethereum.CallMsg{ From: from, To:    nil, Data: code, }
                gas, err = client.EthClient.EstimateGas(context.Background(), msg)
                if err != nil {
                        fmt.Println("预估的 gas err", err)
                        return
                }
                gasPrice            = new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
        */

        amount := big.NewInt(0)
```

```
    tx := types.NewContractCreation(nonce, amount, gas, gasPrice, code)
    // 区块链 id，这里为 777
    signer := types.NewEIP155Signer(big.NewInt(777))
    signedTx, _ := types.SignTx(tx, signer, priKey)
    txHash, err := client.SendRawTransaction(context.TODO(), signedTx)
    if err != nil {
        fmt.Printf(err.Error())
        return
    }
    if err != nil {
        fmt.Println("tx err", err)
        return
    }
    to := crypto.CreateAddress(from, tx.Nonce())
    fmt.Println("txHash", txHash.Hex(), "to", to.Hex())
}
```

3.3 调用

安全起见，Utopia 区块链节点为每个账户维护一个从 0 开始的 nonce 值。区块链节点每成功接收一个交易，发起方账户的 nonce 加 1。 Utopia 账户向一个节点发起交易时，需要在交易结构体中设定其当前的 nonce。如果账户不知道其在 Utopia 区块链节点的当前 nonce，可通过如下方法查询：

Java 示例：

```
public String getNonce() throws BlockchainDriverException {
        // 账户地址
        String address = “0xeda3cceff74dcdb14a04a995d51e7fa06e807a1a”;
        // 第四个参数：是”latest”或 pending，”latest”最新状态，”
pending”待执行交易的状态
        String nonceStr = driver.rpcCall(1, "eth_getTransactionCount", address,
"latest");
        return DriverUtil.parseResult(nonceStr).get("result").substring(2);
    }
```

GO 示例：

```
func getNonce(ec *rpc.Client, from common.Address) (uint64, error) {
    var nonce hexutil.Uint64
    err := ec.CallContext(context.Background(), &nonce, "eth_getTransactionCount",
from, nil)
    return uint64(nonce), err
}
```

3.3.1 Java 语言

3.3.1.1 查询

```
    public void query() throws Exception {
        // ABI（Application Binary Interface）：应用程序二进制接口，描述了应用程序和 Utopia 协议栈之间的接口，获取方式请参考 3.2.2.1
        String abiStr = "{\"constant\": true,\"inputs\": [{\"internalType\": \"bytes\",\"name\": \"key\",\"type\": \"bytes\"}],\"name\": \"get\",\"outputs\": [{\"internalType\": \"bytes\",\"name\": \"value\",\"type\": \"bytes\"}],\"payable\": false,\"stateMutability\": \"view\",\"type\": \"function\"}";
        String contractAddr = "0x8C56F7029629fd965D77A625557Ff9e6EF4b3110";
        CallTransaction.Function function =
CallTransaction.Function.fromJsonInterface(abiStr);
        //调用合约方法所需参数：“key”
        byte[] callData = function.encode("key");
        Map<String, Object> params = new HashMap<>();
        params.put("to", contractAddr);
        params.put("data", "0x" + Hex.toHexString(callData));
        // 第四个参数：是”latest”或 pending，”latest”最新状态，”pending”待执行交易的状态
        String result = driver.rpcCall(1, "eth_call", params, "latest");
        System.out.println(result);
    }
```

3.3.1.2 交易

```
    public void set() throws BlockchainDriverException {
        String abiStr = "{\"constant\": false,\"inputs\": [{\"internalType\":
\"bytes\",\"name\": \"key\",\"type\": \"bytes\"}, {\"internalType\":
\"bytes\",\"name\": \"value\",\"type\": \"bytes\"}],\"name\": \"put\",\"outputs\":
[],\"payable\": false,\"stateMutability\": \"nonpayable\",\"type\": \"function\"}";
        String contract = "0x8C56F7029629fd965D77A625557Ff9e6EF4b3110";
        CallTransaction.Function function =
CallTransaction.Function.fromJsonInterface(abiStr);
        // "key"、" value"调用合约方法所需参数。
        byte[] callData = function.encode("key", "value");
        Long nonce = null;
        try {
            // 16 进制的 nonce 字符串转 long 类型
            nonce = Long.parseLong(getNonce(), 16);
        } catch (BlockchainDriverException e) {
            e.printStackTrace();
        }

        // 如果 genesis.json 文件配置了 blocksize，则不需要预估 gas, gasprice、
gaslimit 可设置为 0

        // 16 进制的 gasprice 字符串转 long 类型
        Long gasprice = Long.parseLong("05a817c800", 16);
        // 16 进制的 gaslimit 字符串转 long 类型
        Long gaslimit = Long.parseLong("47e7c4", 16);
        // 16 进制的 value 字符串转 long 类型
        Long value = Long.parseLong("0de0b6b3a7640000", 16);

        String txId = driver.exec(contract, callData, nonce, gasprice, gaslimit, value);
        System.out.println(txId);
    }
```

3.3.2 GO 语言

3.3.2.1 查询

```
package main
import (
    "context"
    "fmt"
    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/accounts/abi"
```

```
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/ethclient"
	"log"
	"strings"
)
var contractAbi =
`[{"constant":true,"inputs":[{"internalType":"bytes","name":"key","type":"bytes"}],"na
me":"get","outputs":[{"internalType":"bytes","name":"value","type":"bytes"}],"payable"
:false,"stateMutability":"view","type":"function"}]`
func main() {
	// 区块链节点的 JSON RPC 地址（不包括 path）
	var host = "http://101.35.8.134:8545"
	// 合约地址
	var to =
common.HexToAddress("0xe28D7B5Da87cCA2545429B56Adc2DF6FBE3F4513")
	abi, err := abi.JSON(strings.NewReader(contractAbi))
	if err != nil {
		log.Fatalln("JSON fail", err)
	}
	//get 是智能合约的方法名称, []byte{97}是对应的参数
	abiBuf, err := abi.Pack("get",[]byte{97})
	if err != nil {
		log.Fatalln("Pack fail", err)
	}
	callMsg := ethereum.CallMsg{
		To:     &to,
		Data: abiBuf,
	}
	client, err := ethclient.Dial(host)
	result, err := client.CallContract(context.TODO(), callMsg, nil)
	if err != nil {
		log.Fatalln("CallContract fail", err)
	}
	//get 是智能合约的方法名称
	r, err := abi.Unpack("get", result)
	if err != nil {
		log.Fatalln("Unpack", err)
	}
	fmt.Println(r)
}
```

3.3.2.2 交易

```
package main
import (
        "client_Sample/privKeys"
        "client_Sample/tool"
        "context"
        "fmt"
        "github.com/ethereum/go-ethereum/accounts/abi"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/core/types"
        "github.com/ethereum/go-ethereum/crypto"
                "log"
                "math/big"
                "strings"
        )
        func main() {
                // 区块链节点的 JSON RPC 地址（不包括 path）
                var host = "http://101.35.8.134:8545"
                client, err := tool.ToolConnect(host)
                if err != nil {
                        fmt.Printf(err.Error())
                        return
                }
                priKey, err := crypto.HexToECDSA(privKeys.PrivKeys[0])
                if err != nil {
                        fmt.Printf(err.Error())
                        return
                }
                from := crypto.PubkeyToAddress(priKey.PublicKey)
                fmt.Println("from:", from.String())
                nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
                if err != nil {
                        fmt.Printf("nonce err: %s", err.Error())
                        return
                }
                // 合约地址
                to :=
common.HexToAddress("0xDa3Ce11D916fFBa4a1289cEf66A7f142eC5A0f74")
                data := creatAbi([]byte{97}, []byte{98})
                //如果 genesis.json 文件配置 blocksize,则不需要预估 gas
                var (
                        gas        uint64 = 0
                        gasPrice            = big.NewInt(0)
                )
                /*
                        //如果 genesis.json 文件没有配置 blocksize，则需要预估 gas
                        msg := ethereum.CallMsg{
                                        From: from, To:     &to, Data: data,}
                        gas, err = client.EthClient.EstimateGas(context.Background(), msg)
                        if err != nil {
                                fmt.Println("预估的 gas err", err)
                                return
                        }
                        fmt.Println("预估的 gas", "gas", gas)
                        gasPrice            = new(big.Int).Mul(big.NewInt(1e9),
big.NewInt(4000))
                */
                amount := big.NewInt(0)
                tx := types.NewTransaction(nonce, to, amount, gas, gasPrice, data)
                // 区块链 id，这里为：777
                signer := types.NewEIP155Signer(big.NewInt(777))
                signedTx, err := types.SignTx(tx, signer, priKey)
                if err != nil {
                        fmt.Println("types.SignTx", err)
                        return
                }
                hash, err := client.SendRawTransaction(context.TODO(), signedTx)
                fmt.Println("交易 hash", hash)
                if err != nil {
                        fmt.Println("send raw transaction err:", err.Error())
                        return
                }
        }

        func creatAbi(key, value []byte) []byte {
                myContractAbi :=
`[{\"constant\":false,\"inputs\":[{\"internalType\":\"bytes\",\"name\":\"key\",\"typ
e\":\"bytes\"},{\"internalType\":\"bytes\",\"name\":\"value\",\"type\":\"bytes\"}],
\"name\":\"put\",\"outputs\":[],\"payable\":false,\"stateMutability\":\"nonpayable\"
,\"type\":\"function\"}]`
                abi, err := abi.JSON(strings.NewReader(myContractAbi))
                if err != nil {
                        log.Fatalln("JSON err", err)
                }
                //put 是智能合约的方法名称, key, value 是对应的参数
                abiBuf, err := abi.Pack("put", key, value)
                if err != nil {
                        log.Fatalln("Pack err", err)
                }
                return abiBuf
            }
```

4 eWASM 合约

4.1 开发

请参考 https://github.com/PDXbaap/ewasm-rust-demo/blob/master/README.md。

4.2 部署

4.2.1 通过 PDX Unity 部署

见 3.2.1。

4.2.2 编程方式部署

4.2.2.1 Java 语言

示例代码：

```
   public void deployEwasm() throws Exception {
            // @发送方客户端
            byte[] payload = Files.readAllBytes(new File("/path-to-
your/Sample.wasm").toPath());
            String txId = driver.deploy(payload);
            System.out.println(txId);
    }
```

4.2.2.2 GO 语言

示例代码：

```
package main
import (
    "client_Sample/privKeys"
    "client_Sample/tool"
    "context"
    "fmt"
    "github.com/ethereum/go-ethereum/core/types"
    "github.com/ethereum/go-ethereum/crypto"
```

```
	"io/ioutil"
	"math/big"
)
func main() {
// 区块链节点的 JSON RPC 地址（不包括 path）
	var host = "http://1.13.251.80:8545"
	client, err := tool.ToolConnect(host)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	priKey, err := crypto.HexToECDSA(privKeys.PrivKeys[0])
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	from := crypto.PubkeyToAddress(priKey.PublicKey)
	fmt.Println("from:", from.String())
	nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	// @发送方客户端
	path := "/path-to-your/Sample.wasm"
	code, err := ioutil.ReadFile(path)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	//如果 genesis.json 文件配置 blocksize,则不需要预估 gas
	var (
		gas        uint64 = 0
		gasPrice          = new(big.Int)
	)
	/*
		//如果 genesis.json 文件没有配置 blocksize，则需要预估 gas
		msg := ethereum.CallMsg{
				From: from, To:     nil,    Data: code,}
		gas, err = client.EthClient.EstimateGas(context.Background(), msg)
		if err != nil {
			fmt.Println("预估的 gas err", err)
			return
		}
```

```
        fmt.Println("预估的 gas", "gas", gas)
        gasPrice          = new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
    */
    amount := big.NewInt(0)
    tx := types.NewContractCreation(nonce, amount, gas, gasPrice, code)
    // 区块链 id 为: 777
    signer := types.NewEIP155Signer(big.NewInt(777))
    signedTx, _ := types.SignTx(tx, signer, priKey)
    txHash, err := client.SendRawTransaction(context.TODO(), signedTx)
    if err != nil {
        fmt.Printf(err.Error())
        return
    }
    if err != nil {
        fmt.Println("tx err", err)
        return
    }
    fmt.Println("txHash", txHash.Hex())
    fmt.Println("合约地址", crypto.CreateAddress(from, tx.Nonce()).Hex())
    return
}
```

4.3 调用

4.3.1 Java 语言

4.3.1.1 查询

示例代码:

```
    /**
     * 读取数据
     * @throws Exception
     */
    public void runReadMethod() throws Exception {
    // "get:key" get 调用的智能合约方法, key 为智能合约的参数
        String data = "0x" + Hex.toHexString("get:key".getBytes());
        Map<String, String> params = new HashMap<>();
        //合约地址
        params.put("to", "0x1614ef0bfe4c3cf88c8a643c02c4dc8019d87bef");
        //交易数据
        params.put("data", data);
         // 第四个参数: 是" latest" 或 pending, " latest" 最新状态, "
```

```
pending”待执行交易的状态
            String result = driver.rpcCall(1, "eth_call", params, "latest");
            System.out.println(result);
    }
```

4.3.1.2 交易

示例代码：

```
    /**
     * 写入数据
     * @throws Exception
     */
    public void runWriteMethod() throws Exception {
            long nonce = getNonce();
            String contractAddress = "0x1614ef0bfe4c3cf88c8a643c02c4dc8019d87bef";
            // “put: key,value "” put 调用的智能合约方法，key、value 为智能合约
的参数
            byte[] payload = "put:key,value".getBytes();
            String txId = driver.exec(contractAddress, payload, nonce,
Constants.BAAP_DEFAULT_GAS_PRICE,
Constants.BAAP_DEFAULT_GAS_LIMIT, 0);
            System.out.println(txId);
    }
```

4.3.2 GO 语言

4.3.2.1 查询

示例代码：

```
package main
import (
        "client_Sample/tool"
        "context"
        "fmt"
        "github.com/ethereum/go-ethereum"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "log"
)
func main() {
        // 区块链节点的 JSON RPC 地址（不包括 path）
```

```
	var host = "http://127.0.0.1:8547"
	client, err := tool.ToolConnect(host)
	if err != nil {
		log.Fatal("ethclient Dial fail", err)
	}
	// 合约地址
	to :=
common.HexToAddress("0x5d85F01d4B0Eedd70a07a7472F7350e25c24BE08")
	// key 为智能合约 get 方法的参数
	key := "pdx"
	callMsg := ethereum.CallMsg{
		To:	&to,
		Data: []byte("get:" + key),
	}
	fmt.Println(hexutil.Bytes(callMsg.Data))
	bytes, err := client.EthClient.CallContract(context.Background(), callMsg, nil)
	if err != nil {
		log.Println(err)
	}
	fmt.Printf("key=%v,value=%v", key, bytes)
}
```

4.3.2.2 交易

示例代码：

```
package main
import (
	"client_Sample/privKeys"
	"client_Sample/tool"
	"context"
	"fmt"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/core/types"
	"github.com/ethereum/go-ethereum/crypto"
	"log"
	"math/big"
)
func main() {
	//ewasm 合约部署完成后需要等待哨兵合约检查完成后才能进行调用,大概
需要 10 个 normal 区块的时间进行确认
	// 区块链节点的 JSON RPC 地址（不包括 path）
	var host = "http://127.0.0.1:8547"
	client, err := tool.ToolConnect(host)
	if err != nil {
		log.Fatal("ethclient Dial fail", err)
	}
	// 合约地址
	to :=
common.HexToAddress("0x5d85F01d4B0Eedd70a07a7472F7350e25c24BE08")
	priKey, err := crypto.HexToECDSA(privKeys.PrivKeys[0])
	if err != nil {
		log.Fatal("err", err)
	}
	from := crypto.PubkeyToAddress(priKey.PublicKey)
	fmt.Println("from", from.String())
	nonce, err := client.EthClient.PendingNonceAt(context.Background(), from)
	if err != nil {
		log.Fatal("err", err)
	}
	//如果 genesis.json 文件配置 blocksize,则不需要预估 gas
	var (
		gas        uint64 = 0
		gasPrice          = new(big.Int)
	)
	/*
		//如果 genesis.json 文件没有配置 blocksize, 则需要预估 gas
		msg := ethereum.CallMsg{
					From: from,
					To:   &to,
					Data: []byte("put:pdx,222"),
		}
		gas, err = client.EthClient.EstimateGas(context.Background(), msg)
		if err != nil {
			fmt.Println("预估的 gas err", err)
			return
		}
		fmt.Println("预估的 gas", "gas", gas)
		gasPrice          = new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
	*/

	amount := big.NewInt(0)
	//put 为智能合约的方法,key=pdx value=222
	data := []byte("put:pdx,222")
	tx := types.NewTransaction(nonce, to, amount, gas, gasPrice, data)
	// 区块链 id 为: 777
	signer := types.NewEIP155Signer(big.NewInt(777))
	signedTx, err := types.SignTx(tx, signer, priKey)
```

```
        if err != nil {
                fmt.Println("types.SignTx", err)
                return
        }
        hash, err := client.SendRawTransaction(context.TODO(), signedTx)
        fmt.Println("交易 hash", hash)
        if err != nil {
                fmt.Println("send raw transaction err:", err.Error())
                return
        }
        fmt.Println("txHash", hash.String())
}
```

5 Chaincode 合约

5.1 开发

5.1.1 Java 语言

示例代码：

```
package ltd.pdx.utopia.driver.example;
import org.hyperledger.fabric.shim.ChaincodeBase;
import org.hyperledger.fabric.shim.ChaincodeStub;
import org.hyperledger.fabric.shim.ResponseUtils;
import org.hyperledger.fabric.shim.ledger.KeyModification;
import org.hyperledger.fabric.shim.ledger.KeyValue;
import org.hyperledger.fabric.shim.ledger.QueryResultsIterator;
import java.nio.charset.StandardCharsets;
import java.util.List;
public class Chaincode_Java_Sample extends ChaincodeBase {
        public static void main(String[] args) {
            new Chaincode_Java_Sample().start(args);
        }
        @Override
        public Response init(ChaincodeStub chaincodeStub) {
                return ResponseUtils.newSuccessResponse();
        }
        @Override
        public Response invoke(ChaincodeStub chaincodeStub) {
                String response = "";
                try {
                        final String function = chaincodeStub.getFunction();
                        final List<String> params = chaincodeStub.getParameters();
                        switch (function) {
                                case "put":
                                        chaincodeStub.putStringState(params.get(0), params.get(1));
                                        break;
                                case "get":
                                        String result = chaincodeStub.getStringState(params.get(0));
                                        response = result;
                                        break;
                                case "getHis":
                                        StringBuilder resultHis = new StringBuilder();
                                        QueryResultsIterator<KeyModification> historyForKey =
chaincodeStub.getHistoryForKey(params.get(0));
                                        historyForKey.forEach(e -> {
                                                resultHis.append(String.format("key : %s /value : %s
/isDelete : %s", params.get(0), e.getStringValue(), e.isDeleted()));
                                        });
                                        historyForKey.close();
                                        response = resultHis.toString();
                                        break;
                                case "getRange":
                                        StringBuilder resultRange = new StringBuilder();
                                        QueryResultsIterator<KeyValue> stateByRange =
chaincodeStub.getStateByRange(params.get(0), params.get(1));
                                        stateByRange.forEach(e -> {
                                                resultRange.append(String.format("key : %s /value : %s",
e.getKey(), e.getStringValue()));
                                        });
                                        stateByRange.close();
                                        response = resultRange.toString();
                                        break;
                                case "del":
                                        chaincodeStub.delState(params.get(0));
                                        break;
                                default:
                                        break;
                        }
                } catch (Exception exception) {
                        exception.printStackTrace();
                        return ResponseUtils.newErrorResponse(exception);
                }
                return
ResponseUtils.newSuccessResponse(response.getBytes(StandardCharsets.UTF_8));
```

```
        }
}
```

5.1.2 GO 语言

按照 Hyperledger go Chaincode API 开发 Chaincode 合约。

示例代码：

```
package main
import (
        "fmt"
        "github.com/hyperledger/fabric-chaincode-go/shim"
        pb "github.com/hyperledger/fabric-protos-go/peer"
        "strconv"
        "strings"
)
// SimpleChaincode example simple Chaincode implementation
type SimpleChaincode struct {
}
var A, B string
var Aval, Bval, X int
// Init callback representing the invocation of a chaincode
// This chaincode will manage two accounts A and B and will transfer X units from A
to B upon put
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
        var err error
        _, args := stub.GetFunctionAndParameters()
        if len(args) != 4 {
                return shim.Error("Incorrect number of arguments. Expecting 4")
        }
        // Initialize the chaincode
        A = args[0]
        Aval, err = strconv.Atoi(args[1])
        if err != nil {
                return shim.Error("Expecting integer value for asset holding")
        }
        B = args[2]
        Bval, err = strconv.Atoi(args[3])
        if err != nil {
                return shim.Error("Expecting integer value for asset holding")
        }
        fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)
        /************
                // Write the state to the ledger
        err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
        if err != nil {
            return nil, err
        }
        stub.PutState(B, []byte(strconv.Itoa(Bval)))
        err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
        if err != nil {
            return nil, err
        }
    ************/
    return shim.Success(nil)
}
func (t *SimpleChaincode) put(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        fmt.Printf("Error put [%s:%s] to state: %s", args[0], args[1], err)
        return shim.Error(fmt.Sprintf("Error put [%s:%s] to state: %s", args[0],
args[1], err))
    }
    fmt.Printf("put state success!!!!!!!!!!!")
    return shim.Success(nil)
}
func (t *SimpleChaincode) get(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    res, err := stub.GetState(args[0])
    if err != nil {
        fmt.Printf("Error get [%s] from state: %s", args[0], err)
        return shim.Error(fmt.Sprintf("Error get [%s] from state: %s", args[0], err))
    }
    fmt.Printf("get state success!!!!!!!!!!!")
    return shim.Success(res)
}
func (t *SimpleChaincode) del(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    err := stub.DelState(args[0])
    if err != nil {
        fmt.Printf("Error del [%s] from state: %s", args[0], err)
        return shim.Error(fmt.Sprintf("Error del [%s] from state: %s", args[0], err))
    }
    fmt.Printf("del state success!!!!!!!!!!!")
    return shim.Success(nil)
}
func (t *SimpleChaincode) his(stub shim.ChaincodeStubInterface, args []string)
```

```
pb.Response {
	hisIterator, err := stub.GetHistoryForKey(args[0])
	if err != nil {
		fmt.Printf("Error del [%s] from state: %s", args[0], err)
		return shim.Error(fmt.Sprintf("Error del [%s] from state: %s", args[0], err))
	}
	defer hisIterator.Close()
	var myHis []string
	for hisIterator.HasNext() {
		keyModify, err := hisIterator.Next()
		if err != nil {
			fmt.Printf("Error his [%s] from state: %s", args[0], err)
			return shim.Error("history iterator next:"+err.Error())
		}
		myHis = append(myHis, fmt.Sprintf("key : %s, value : %s, isDelete : %s \n",
args[0], keyModify.Value, strconv.FormatBool(keyModify.IsDelete)))
	}
	fmt.Printf("his state success!!!!!!!!!!!")
	return shim.Success([]byte(strings.Join(myHis,"")))
}
func (t *SimpleChaincode) rangeData(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
	stateIterator, err := stub.GetStateByRange(args[0], args[1])
	if err != nil {
		fmt.Printf("Error range [%s:%s] from state: %s", args[0], args[1], err)
		return shim.Error(fmt.Sprintf("Error del [%s] from state: %s", args[0], err))
	}
	defer stateIterator.Close()
	var myRange []string
	for stateIterator.HasNext() {
		queryResult, err := stateIterator.Next()
		if err != nil {
			fmt.Printf("Error range [%s] from state: %s", args[0], err)
			shim.Error("range iterator next:"+err.Error())
		}
		myRange = append(myRange, fmt.Sprintf("key : %s, value : %s \n",
queryResult.Key, queryResult.Value))
	}
	fmt.Printf("range state success!!!!!!!!!!!")
	return shim.Success([]byte(strings.Join(myRange,"")))
}
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
	function, args := stub.GetFunctionAndParameters()
	switch function {
        case "put":
                return t.put(stub, args)
        case "get":
                return t.get(stub, args)
        case "del":
                return t.del(stub, args)
        case "getHis":
                return t.his(stub, args)
        case "getRange":
                return t.rangeData(stub, args)
        }
        return shim.Error("Invalid function name. Expecting \"put or get or del or his or
range\"")
}
```

5.2 部署

5.2.1 通过 PDX Unity 部署

见 3.2.1。

5.2.2 直连协议栈部署

5.2.2.1 Java 语言

5.2.2.1.1 编译

编译 Chaincode_Java_Sample.java 为 Chaincode_Java_Sample.jar

5.2.2.1.2 启动

```
// -a 协议栈的 IP 和 GRPC 端口 -i {合约拥有者的地址}:{合约名称}
Java -jar Chaincode_Java_Sample.jar   -a   127.0.0.1:6000   -i
58dfe602278d3f82ebce7355624279b8a5d4c14a:Chaincode_Java_Sample
```

5.2.2.2 GO 语言

5.2.2.2.1 编译

编译合约： go build Chaincode_Go_Sample.go

5.2.2.2.2 启动

```
// -a  协议栈的 IP 和 GRPC 端口  -i {合约拥有者的地址}:{合约名称}
./Chaincode_Go_Sample    -a    127.0.0.1:6000    -i
58dfe602278d3f82ebce7355624279b8a5d4c14a:Chaincode_Go_Sample
```

5.2.3 编程方式部署

如果 PDX Utopia 是与 PDX BaaP 一起部署在每个区块链节点上，则可以通过编程方式部署 Chaincode 合约。

5.2.3.1 Java 语言

示例代码：

```
/**
         *  部署 Chaincode 合约
         *
         * @throws IOException
         * @throws BlockchainDriverException
         */
        public void deployCC() throws IOException, BlockchainDriverException {
                //智能合约的 Java 文件  @  发送方客户端
                File ccFile = new File("/path-to-your/chaincode_java_sample.java");
                String txId = driver.deploy(ChaincodeType.JAVA, name, version, ccFile);
                System.out.println(txId);
        }
```

5.2.3.2 GO 语言

示例代码：

```
package main
import (
        "client_Sample/chaincode/protos"
        "client_Sample/tool"
        "context"
        "encoding/json"
        "fmt"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/core/types"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/golang/protobuf/proto"
        "golang.org/x/crypto/sha3"
        "io/ioutil"
        "math/big"
```

```
)
const (
	//合约的拥有者、合约名称、合约版本号
	owner = "8000d109DAef5C81799bC01D4d82B0589dEEDb33"
	name    = "sample"
)
func main() {
	// 区块链节点的 JSON RPC 地址（不包括 path）
	host := "http://127.0.0.1:8545"
	client, err := tool.ToolConnect(host)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	priKey, err :=
crypto.HexToECDSA("d29ce71545474451d8292838d4a0680a8444e6e4c14da018b4a08
345fb2bbb84")
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	from := crypto.PubkeyToAddress(priKey.PublicKey)
	fmt.Println("from:", from.String())
	//预编译合约的地址，必须是 :baap-deploy
	to := iKeccak256ToAddress(":baap-deploy")
	nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	deployInfo := struct {
		FileName       string `json:"fileName"`
		ChaincodeId string `json:"chaincodeId"`
		Pbk                 string `json:"pbk"`
	}{
		"MyCc.java",
		owner + ":" + name + ":",
		string(crypto.CompressPubkey(&priKey.PublicKey)),
	}
	deployInfoBuf, err := json.Marshal(deployInfo)
	if err != nil {
		fmt.Printf("marshal deployInfo err: %v", err)
		return
	}
```

```
//  @发送方客户端
myccBuf, err := ioutil.ReadFile("/path-to-your/MyCc.java")
if err != nil {
        fmt.Printf("read java file err:%v", err)
        return
}
invocation := &protos.Invocation{
        Fcn:    "deploy",
        Args: [][]byte{deployInfoBuf},
        Meta: map[string][]byte{
                "baap-tx-type": []byte("exec"),//Baap 要求
                "baap-cc-code": myccBuf,
        },
}
dep := &protos.Deployment{
        Owner:      owner,
        Name:        name,
        Payload: invocation,
}
payload, err := proto.Marshal(dep)
if err != nil {
        fmt.Printf("proto marshal invocation error:%v", err)
        return
}
ptx := &protos.Transaction{
        Type:        2, //1invoke 2deploy
        Payload: payload,
}
data, err := proto.Marshal(ptx)
if err != nil {
        fmt.Printf("!!!!!!!!proto marshal error:%v", err)
        return
}
//如果 genesis.json 文件配置 blocksize,则不需要预估 gas
var (
        gas           uint64 = 0
        gasPrice               = new(big.Int)
)

/*
    //如果 genesis.json 文件没有配置 blocksize, 则需要预估 gas

        msg := ethereum.CallMsg{
                        From: from,
                        To:     &to,
                        Data: data,
                }
                gas, err = client.EthClient.EstimateGas(context.Background(), msg)
                if err != nil {
                        fmt.Println("预估的 gas err", err)
                        return
                }
                fmt.Println("预估的 gas", "gas", gas)
                gasPrice            = new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
        */

        amount := big.NewInt(0)

        fmt.Println("nounce:", nonce)
        tx := types.NewTransaction(nonce, to, amount, gas, gasPrice, data)
        // 区块链 id 为: 777
        signer := types.NewEIP155Signer(big.NewInt(777))
        signedTx, _ := types.SignTx(tx, signer, priKey)
        txHash, err := client.SendRawTransaction(context.TODO(), signedTx)
        if err != nil {
                fmt.Printf("send raw tx:%s", err.Error())
                return
        }
        fmt.Printf("Transaction hash: %s\n", txHash.String())
}

func iKeccak256ToAddress(ccName string) common.Address {
        hash := sha3.NewLegacyKeccak256()
        crypto.Keccak256()
        var buf []byte
        hash.Write([]byte(ccName))
        buf = hash.Sum(buf)
        fmt.Println("keccak256ToAddress:", common.BytesToAddress(buf).String())
        addr := common.BytesToAddress(crypto.Keccak256([]byte(ccName))[12:])
        fmt.Println("keccak256ToAddress:", addr.String())
        return common.BytesToAddress(buf)
}
```

5.3 调用

5.3.1 Java 语言

5.3.1.1 查询

示例代码：

```
public void query() throws BlockchainDriverException {
        // 方法名： get，查询键 pdx
        byte[] result = driver.state(ccAddress, "get", "pdx");
        System.out.println(new String(result));
}
```

5.3.1.2 交易

示例代码：

```
/**
         * 写入 key/value
         *
         * @throws BlockchainDriverException
         */
        public void exec() throws BlockchainDriverException {
                List<byte[]> args = new ArrayList<>();
                // 状态键： key ,值： value
                args.add("key".getBytes());
                args.add("value".getBytes());
                Invocation tx = Invocation.builder()
                                .fcn("put")
                                .args(args)
                                .build();
                String txId = driver.exec(ccAddress, tx);
                System.out.println(txId);
        }
```

5.3.2 GO 语言

5.3.2.1 查询

示例代码：

```
package main
import (
        "client_Sample/chaincode/protos"
        "client_Sample/tool"
        "context"
        "fmt"
```

```
	"github.com/ethereum/go-ethereum/common"
	"github.com/golang/protobuf/proto"
	"time"
)
func main() {
// 区块链节点的 JSON RPC 地址（不包括 path）
	var host = "http://81.69.236.242:8545"
	client, err := tool.ToolConnect(host)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	// 合约地址
	to :=
common.HexToAddress("0x14632e85D8Cb91D943BD63e08E15DA8411DF2382")
	invocation := &protos.Invocation{
		Fcn:   "get", //getHis,get,getRange,del,
		Args: [][]byte{[]byte("a")}, //智能合约的参数
		Meta: map[string][]byte{"to": []byte(to.String())},
	}
	payload, err := proto.Marshal(invocation)
	if err != nil {
		fmt.Printf("proto marshal invocation error:%v", err)
		return
	}
	ptx := &protos.Transaction{
		Type:     1, //1invoke 2deploy
		Payload: payload,
	}
	data, err := proto.Marshal(ptx)
	if err != nil {
		fmt.Printf("!!!!!!!!proto marshal error:%v", err)
		return
	}
	c, _ := context.WithTimeout(context.Background(), 800*time.Millisecond)
	var result string
	result, err = client.BaapQuery(c, data)
	if err != nil {
		fmt.Println("query tx error", "err", err)
		return
	}
	fmt.Println("baap query", "resp", result)
	if err != nil {
		fmt.Printf("err:%v", err)
```

```
            return
        }
}
```

5.3.2.2 交易

示例代码：

```
package main
import (
        "client_Sample/chaincode/protos"
        "client_Sample/privKeys"
        "client_Sample/tool"
        "context"
        "fmt"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/core/types"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/golang/protobuf/proto"
        "math/big"
)
func main() {
// 区块链节点的 JSON RPC 地址（不包括 path）
        var host = "http://101.34.220.60:8545"
        client, err := tool.ToolConnect(host)
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        priKey, err := crypto.HexToECDSA(privKeys.PrivKeys[0])
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        from := crypto.PubkeyToAddress(priKey.PublicKey)
        fmt.Println("from:", from.String())
        // 合约地址
        to :=
common.HexToAddress("0x14632e85D8Cb91D943BD63e08E15DA8411DF2382")
        fmt.Printf("to:%s\n", to.String())
        nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
        if err != nil {
                fmt.Printf(err.Error())
                return
```

```
}
var a []byte
for i := 0; i < 10; i++ {
        a = append(a, []byte("a")...)
}
invocation := &protos.Invocation{
        // 智能合约方法
        Fcn:    "put",
        // 智能合约方法参数
        Args: [][]byte{[]byte("a"), a},
        // Baap 要求
        Meta: map[string][]byte{"baap-tx-type": []byte("exec")},
}
payload, err := proto.Marshal(invocation)
if err != nil {
        fmt.Printf("proto marshal invocation error:%v", err)
        return
}
ptx := &protos.Transaction{
        Type:        1, //1invoke 2deploy
        Payload: payload,
}
data, err := proto.Marshal(ptx)
if err != nil {
        fmt.Printf("!!!!!!!!proto marshal error:%v", err)
        return
}
fmt.Println("nounce:", nonce)
//如果 genesis.json 文件配置 blocksize,则不需要预估 gas
var (
        gas          uint64 = 0
        gasPrice              = new(big.Int)
)
/*
        //如果 genesis.json 文件没有配置 blocksize，则需要预估 gas

        msg := ethereum.CallMsg{
                        From: from,
                        To:       &to,
                        Data: data,
        }
        gas, err = client.EthClient.EstimateGas(context.Background(), msg)
        if err != nil {
                fmt.Println("预估的 gas err", err)
                return
            }
            fmt.Println("预估的 gas", "gas", gas)
            gasPrice            = new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
        */

        amount := big.NewInt(0)
        tx := types.NewTransaction(nonce, to, amount, gas, gasPrice, data)
        // 区块链 id 为: 777
        signer := types.NewEIP155Signer(big.NewInt(777))
        signedTx, _ := types.SignTx(tx, signer, priKey)
        txHash, err := client.SendRawTransaction(context.TODO(), signedTx)
        if err != nil {
            fmt.Printf(err.Error())
            return
        }
        fmt.Printf("Transaction hash: %s\n", txHash.String())
}
```

6 DynNative 合约

PDX Utopia 支持部署 GOLANG 动态库合约。值得注意的是，部署加载后的动态库合约在 Utopia 进程之内，其质量将直接影响整个 Utopia 进程的稳定性。

6.1 开发

6.1.1 接口与方法

```
package contract // under utopia
        import "container/list"

            // 动态库合约需要实现以下方法

        type DynNativeContract interface {
            Run(stub interface{}) ([]byte,error)
        }
        // PDX UTOPIA 协议栈提供如下方法（通过 StubInterface）:
        type StubInterface interface {
            // GetArgs returns the arguments intended for the so Run as an array of byte
arrays
            GetArgs() [][]byte
            // GetStringArgs returns the arguments intended for the so Run as a string
array.
            GetStringArgs() []string
            // GetFunctionAndParameters returns the first argument as the function
name
            // and the rest of arguments as parameters in a string array.
            GetFunctionAndParameters() (string,[][]byte)
            // GetHistoryForKey returns a history of key values across time.
            // For each historic key update,the historic value and associated block num.
            GetHistoryForKey(key string,start,end uint64) (*list.Element,error)
            // GetState returns the value of the specified `key` from the
            // ledger.Note that GetState doesn`t read data from the writest,which
            // has not been committed to the state.
            GetState(key []byte) ([]byte,error)
            // PutState puts the specified `key` and `value` into the state.simple keys
            // must not be an empty string and must not start with a null character
            PutState(key []byte,value []byte) error
            // DelState records the specified `key` to be deleted in the state.
            DelState(key []byte) error
}
```

6.1.2 开发

示例代码:

```
    package main
    import (
        "errors"
        "github.com/PDXbaap/utopia_spi/contract"
    )
    var DynNative dynNativeContract
    type dynNativeContract struct{}

    func (s *dynNativeContract) Run(stub interface{}) ([]byte, error) {
        v, ok := stub.(contract.StubInterface)
        if !ok {
            return nil, nil
        }
        function, inputs := v.GetFunctionAndParameters()
        if function == "get" {
            return s.get(v, inputs)
        } else if function == "put" {
            return s.put(v, inputs)
```

```
		}
		return []byte{}, nil
	}

	func (s *dynNativeContract) get(stub contract.StubInterface, args [][]byte) ([]byte,
error) {
		if len(args) != 1 {
			return []byte{}, errors.New("查询输入有误")
		}
		key := args[0]
		v, err := stub.GetState(key)
		if err != nil {
			return nil, err
		}
		return v, nil
	}

	func (s *dynNativeContract) put(stub contract.StubInterface, args [][]byte) ([]byte,
error) {
		if len(args) != 2 {
			return nil, errors.New("输入有误")
		}
		key := args[0]
		v := args[1]
		err := stub.PutState(key, v)
		if err != nil {
			return nil, err
		}
		return v, nil
	}
	func main() {
}
```

6.2 编译

```
go build -buildmode=plugin -o DynNative.1.0.so ./ DynNative.go
```

6.3 部署

6.3.1 Java 语言

示例代码：

```
        /*
        * 部署 DynNative 合约
        */
public void deploy() throws Exception {
            // 动态库本地 path @发送方客户端
            String soPath = "/path-to-your/DynNative_sample.1.11.so";
            // 读取动态库文件二进制数据
            byte[] soData = Files.readAllBytes(new File(soPath).toPath());
            /*
            初始化交易 payload
            soName                    动态库名称 只有部署合约时传入
            lookUpClassName          动态库对外提供的类名称 部署、调用时都需
要传入
            args                      调用动态库合约参数 只有调用合约时传入
            soData                    动态库文件二进制数据 只有部署合约时传入
             */
            DynNativeCallParam dynNativeCallParam = new
DynNativeCallParam("sample", "DynNative", null, soData);
            // 获取 rlp 编码
            byte[] payload = dynNativeCallParam.getRlpEncoded();
            // DEPLOY_ADDRESS
0x6067b1C683c96EDEb4031cA8D75e2902D0dfB9dD 部署动态库合约的预编译合
约地址
            String txHash = driver.exec(DEPLOY_ADDRESS, payload);
            System.out.println("txHash: " + txHash);
      }
```

6.3.2 GO 语言

示例代码：

```
package main
import (
      "client_Sample/tool"
      "context"
      "fmt"
      "github.com/ethereum/go-ethereum/common"
      "github.com/ethereum/go-ethereum/core/types"
      "github.com/ethereum/go-ethereum/crypto"
      "github.com/ethereum/go-ethereum/rlp"
      "golang.org/x/crypto/sha3"
      "io/ioutil"
      "math/big"
```

```
	"os"
	"strings"
)
func main() {
	// 区块链节点的 JSON RPC 地址（不包括 path）
	var host = "http://110.42.191.221:8545"
	client, err := tool.ToolConnect(host)
	if err != nil {
		fmt.Println("err", err.Error())
		return
	}
	priKey, err :=
crypto.HexToECDSA("a2f1a32e5234f64a6624210b871c22909034f24a52166369c26196
81390433aa")
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	from := crypto.PubkeyToAddress(priKey.PublicKey)
	fmt.Println("from:", from.String())
	//预编译合约的地址，必须是 callso
	to := iKeccak256ToAddress("callso")
	fmt.Printf("to:%s\n", to.String())
	nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
	if err != nil {
		fmt.Printf(err.Error())
		return
	}
	// 动态库本地 path @发送方客户端
	path := "/path-to-your/DynNative_sample.1.11.so"
	soData := readSoFile(path)

	type callSoInfo struct {
		SoName          string
		LookUpClassName string
		Args            [][]byte
		Data            []byte
	}
	soInfo := &callSoInfo{
		SoName:          "DynNative",
		LookUpClassName: "DynNativeContract",
		Data:            soData,
	}
	data, _ := rlp.EncodeToBytes(soInfo)
	//如果 genesis.json 文件配置 blocksize,则不需要预估 gas
	var (
		gas		uint64 = 0
		gasPrice		= new(big.Int)
	)

	/*
		//如果 genesis.json 文件没有配置 blocksize，则需要预估 gas

		msg := ethereum.CallMsg{
				From: from,
				To:	&to,
				Data: data,
		}
		gas, err = client.EthClient.EstimateGas(context.Background(), msg)
		if err != nil {
			fmt.Println("预估的 gas err", err)
			return
		}
		fmt.Println("预估的 gas", "gas", gas)
		gasPrice		= new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
	*/

	amount := big.NewInt(0)
	tx := types.NewTransaction(nonce, to, amount, gas, gasPrice, data)
	// 区块链 id 为： 777
	signer := types.NewEIP155Signer(big.NewInt(777))
	signedTx, _ := types.SignTx(tx, signer, priKey)
	txHash, err := client.SendRawTransaction(context.TODO(), signedTx)
	if err != nil {
		fmt.Printf("send raw tx:%s", err.Error())
		return
	}
	fmt.Printf("Transaction hash: %s\n", txHash.String())
	soname := soInfo.SoName
	ownerS := from.String()
	soName := strings.ToLower(ownerS[2:]) + ":" + soname
	address := common.BytesToAddress(crypto.Keccak256([]byte(soName))[12:])
	fmt.Println("address", address)
}

func iKeccak256ToAddress(ccName string) common.Address {
	hash := sha3.NewLegacyKeccak256()
```

```
        var buf []byte
        hash.Write([]byte(ccName))
        buf = hash.Sum(buf)
        fmt.Println("keccak256ToAddress:", common.BytesToAddress(buf).String())
        return common.BytesToAddress(buf)
}

func readSoFile(path string) []byte {
        file, err := os.Open(path)
        if err != nil {
                println("err:", err.Error())
        }
        data, err := ioutil.ReadAll(file)
        if err != nil {
                println("err:", err.Error())
        }
        return data
}
```

6.4 调用

6.4.1 Java 语言

6.4.1.1 查询

示例代码：

```
  public void query() throws BlockchainDriverException {
                byte[][] args = new byte[][]{"get".getBytes(StandardCharsets.UTF_8),
"key".getBytes(StandardCharsets.UTF_8)};
                String contractAddress = EncryptUtil.keccak256ToAddress(owner +
Constants.BAAP_CC_NAME_SEPARATOR + "sample");
                DynNativeCallParam dynNativeCallParam = new DynNativeCallParam(null,
"DynNative", args, null);
                byte[] payload = dynNativeCallParam.getRlpEncoded();
                String data = "0x" + Hex.toHexString(payload);
                Map<String, String> params = new HashMap<>();
                params.put("to", "0x" + contractAddress);
                params.put("data", data);
                // 第四个参数：是”latest”或 pending，”latest”最新状态，”
pending”待执行交易的状态
                String result = driver.rpcCall(1, "eth_call", params, "latest");
```

```
            System.out.println("result: " + result);
  }
```

6.4.1.2 交易

示例代码：

```
        public void exec() throws Exception {
      // args 第一项为合约方法名称，其他是合约方法参数
      byte[][] args = new byte[][]{"put".getBytes(StandardCharsets.UTF_8),
"key".getBytes(StandardCharsets.UTF_8), "value".getBytes(StandardCharsets.UTF_8)};
            //合约地址规则：“合约所有者:合约名称”SHA3 结果的后 20 字节
            String contractAddress = EncryptUtil.keccak256ToAddress(owner +
Constants.BAAP_CC_NAME_SEPARATOR + "sample");
            DynNativeCallParam dynNativeCallParam = new DynNativeCallParam(null,
" DynNative ", args, null);
            byte[] payload = dynNativeCallParam.getRlpEncoded();
            String txHash = driver.exec(contractAddress, payload);
            System.out.println("txHash: " + txHash);
        }
```

6.4.2 GO 语言

6.4.2.1 查询

示例代码：

```
package main
import (
        "client_Sample/tool"
        "context"
        "fmt"
        "github.com/ethereum/go-ethereum"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/rlp"
)
func main() {
         // 区块链节点的 JSON RPC 地址（不包括 path）
        var host = "http://192.168.3.47:8545"
        client, err := tool.ToolConnect(host)
        if err != nil {
                fmt.Println("err", err.Error())
```

```
		return
	}
	// 合约地址
	to :=
common.HexToAddress("0x4bbd27AaC056178dc3eea9d14E1c8Eb10Fa3732f")
	fmt.Printf("to:%s\n", to.String())
	type callSoInfo struct {
		SoName          string
		LookUpClassName string
		Args            [][]byte
		Data            []byte
	}
	soInfo := &callSoInfo{
		LookUpClassName: "DynNativeContract",
		Args:            [][]byte{[]byte("get"), []byte("name")},
	}
	data, err := rlp.EncodeToBytes(soInfo)
	if err != nil {
		fmt.Println(err)
		return
	}
	callMsg := ethereum.CallMsg{
		To:   &to,
		Data: data,
	}
	result, err := client.EthClient.CallContract(context.TODO(), callMsg, nil)
	if err != nil {
		fmt.Println("err", err)
		return
	}
	fmt.Println("result", result)
}
```

6.4.2.2 交易

示例代码：

```
package main
import (
	"client_Sample/tool"
	"context"
	"encoding/binary"
	"fmt"
	"github.com/ethereum/go-ethereum/common"
```

```
        "github.com/ethereum/go-ethereum/core/types"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/rlp"
        "math/big"
)
func main() {
          // 区块链节点的 JSON RPC  地址（不包括 path）
        var host = "http://110.42.191.221:8545"
        client, err := tool.ToolConnect(host)
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        priKey, err :=
crypto.HexToECDSA("a2f1a32e5234f64a6624210b871c22909034f24a52166369c26196
81390433aa")
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        from := crypto.PubkeyToAddress(priKey.PublicKey)
        // 合约地址
        to :=
common.HexToAddress("0x178e1910226c15f65073AB1c2f78DA726B7d36A8")
        fmt.Println("from:", from.String())
        fmt.Printf("to:%s\n", to.String())
        nonce, err := client.EthClient.NonceAt(context.TODO(), from, nil)
        if err != nil {
                fmt.Printf(err.Error())
                return
        }
        type callSoInfo struct {
                SoName               string
                LookUpClassName string
                Args                 [][]byte
                Data                 []byte
        }
        soInfo := &callSoInfo{
                LookUpClassName: "DynNativeContract",
                Args:                [][]byte{[]byte("put"), []byte("name"),
Uint64ToByte(nonce)},
        }
        data, err := rlp.EncodeToBytes(soInfo)
        if err != nil {
            fmt.Printf(err.Error())
            return
        }
        //如果 genesis.json 文件配置 blocksize,则不需要预估 gas
        var (
            gas         uint64 = 0
            gasPrice            = new(big.Int)
        )
        /*
            //如果 genesis.json 文件没有配置 blocksize，则需要预估 gas

            msg := ethereum.CallMsg{
                From: from,
                To:     &to,
                Data: data,
            }
            gas, err = client.EthClient.EstimateGas(context.Background(), msg)
            if err != nil {
                fmt.Println("预估的 gas err", err)
                return
            }
            fmt.Println("预估的 gas", "gas", gas)
            gasPrice            = new(big.Int).Mul(big.NewInt(1e9), big.NewInt(4000))
        */

        amount := big.NewInt(0)
        fmt.Println("nounce:", nonce)
        tx := types.NewTransaction(nonce, to, amount, gas, gasPrice, data)
        // 区块链 id，这里为： 777
        signer := types.NewEIP155Signer(big.NewInt(777))
        signedTx, _ := types.SignTx(tx, signer, priKey)
        txHash, err := client.SendRawTransaction(context.TODO(), signedTx)
        if err != nil {
            fmt.Printf("send raw tx:%s", err.Error())
            return
        }
        fmt.Printf("Transaction hash: %s\n", txHash.String())
}
func Uint64ToByte(n uint64) []byte {
        b := make([]byte, 8)
        binary.BigEndian.PutUint64(b, n)
        return b
}
```

7 区块链事件侦听

PDX UTOPIA 区块链支持以太坊的事件侦听机制。应用程序可以订阅 PDX UTOPIA 区块链协议本身和其上的智能合约生成的区块链事件。

7.1 Java 语言

示例代码：

```
public static void main(String[] args) throws Exception {
		UtopiaChaincodeDriver driver = new UtopiaChaincodeDriver();
		Properties props = new Properties();
		props.setProperty(Constants.BAAP_SENDER_PRIVKEY,
"65035d9621f7be3bb6dc1f5a646e6ee2ef6bddf3f1ce57782d409c23857401a6");
		// 区块链节点的 JSON RPC 地址（不包括 path）
		props.setProperty(Constants.BAAP_BLOCKCHAIN_RPC,
"http://127.0.0.1:8545");
		// 区块链节点的 websocket 地址
		props.setProperty(Constants.BAAP_BLOCKCHAIN_WS_RPC,
"ws://192.168.5.17:8546");
		props.setProperty(Constants.BAAP_ENGINE_ID,
Constants.BAAP_ENGINE_ID_DEFAULT);
		driver.init(props, new Listener() {
			@Override
			public void event(String clientAssignedId, String result) throws
IOException {
				System.out.println(clientAssignedId + "--->" + result);
			}
		});
		// 订阅新日志事件
		subcribeLogs(driver);
		// 订阅待定交易事件
		driver.subscribe("1",
SubscribeType.NEWPENDINGTRANSACTIONS.getName());
		// 订阅新区块头事件
		// driver.subscribe("1", SubscribeType.NEWHEADS.getName());
		// 订阅节点同步事件
		// driver.subscribe("1",SubscribeType.SYNCING.getName());
		// driver.unsubscribe("1");
	}

	// 订阅新日志事件
	static public void subcribeLogs(UtopiaChaincodeDriver driver) throws Exception
{
        Object[] params = new Object[2];
        params[0] = (SubscribeType.LOGS.getName());
        Map<String, Object> map = new HashMap<String, Object>();
        // 订阅合约地址数组,topics 只能是同一个事件的，多个事件不能写
在同一个 topics 里
        map.put("address",
Arrays.asList("0xD6FAC09B4f87485C774912b4B913fBe98D3aa271"));
        // 订阅 topic 数组   第一个元素为为合约事件签名（签名的参数顺
序不可变），
        // 由于 solidity 可以在事件参数上增加 indexed 属性，最多可以对三
个参数增加这样的属性。所以 topics 最多为 4 个
        // 即第一个默认主题事件签名和三个 indexed 修饰的参数的值，且
另外三个参数只能是此事件下的 indexed 参数作为 topic
        // topics 数组的顺序要和事件 indexed 参数顺序一致
        // 如果数组，包括字符串，字节数据做为索引参数，实际主题是对
应值的 Keccak-256 哈希值。
        List paramsArr = Arrays.asList(new TypeReference<Address>() {
        }, new TypeReference<DynamicBytes>() {
        }, new TypeReference<DynamicBytes>() {
        });
        Event event = new Event("Put", paramsArr);
        String functionEventSig = EventEncoder.encode(event);
        List<String> topics = Arrays.asList();
        map.put("topics", new ArrayList<String>() {{
            add(functionEventSig);
        }});
        params[1] = map;
        driver.subscribe("1", params);
    }
```

7.2 GO 语言

示例代码：

```
package main
import (
    "context"
    "fmt"
    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/core/types"
    "github.com/ethereum/go-ethereum/ethclient"
    "log"
```

```
)
func main() {
// 块链节点的 websocket 地址
	client, err := ethclient.Dial("ws://127.0.0.1:8546")
	if err != nil {
		log.Fatal("1", err)
	}
	contractAddress :=
common.HexToAddress("0xDa3Ce11D916fFBa4a1289cEf66A7f142eC5A0f74")
	query := ethereum.FilterQuery{
		Addresses: []common.Address{contractAddress},
	}
	logs := make(chan types.Log)
   // 订阅新日志事件
sub, err := client.SubscribeFilterLogs(context.Background(), query, logs)
	if err != nil {
		log.Fatal(err)
	}
	for {
		select {
		case err := <-sub.Err():
			log.Fatal(err)
		case vLog := <-logs:
			fmt.Println(vLog) // pointer to event log
		}
	}
}
package main
import (
	"context"
	"fmt"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/ethclient/gethclient"
	"github.com/ethereum/go-ethereum/rpc"
	"log"
)
func main() {
	dial, err := rpc.Dial("ws://127.0.0.1:8546")
	client := gethclient.New(dial)
	event := make(chan common.Hash)
// 订阅待定交易事件
tx, err := client.SubscribePendingTransactions(context.Background(), event)
	if err != nil {
		log.Fatal(err)
```

```
        }
        for {
                select {
                case err := <-tx.Err():
                        log.Fatal(err)
                case vLog := <-event:
                        fmt.Println(vLog) // pointer to event log
                }
        }
}
package main
import (
        "context"
        "fmt"
        "github.com/ethereum/go-ethereum/core/types"
        "github.com/ethereum/go-ethereum/ethclient"
        "log"
)
func main() {
        client, err := ethclient.Dial("ws://127.0.0.1:8546")
        if err != nil {
                log.Fatal("1", err)
        }
        event := make(chan *types.Header)
// 订阅新区块头事件
        header, err := client.SubscribeNewHead(context.Background(), event)
        if err != nil {
                log.Fatal(err)
        }
        for {
                select {
                case err := <-header.Err():
                        log.Fatal(err)
                case vLog := <-event:
                        fmt.Println(vLog) // pointer to event log
                }
        }
}
```