碳链记·GreenTrace Chain

基于区块链的碳核算和碳交易系统部署教程

开发团队：你的外包我来包

目录

一、环境部署

首先确认服务器的环境，这里我们需要 docker 版本为 20.10.7 或以上 docker-compose 版本为 1.29.2 或以上

可以通过如下链接下载

https://docs.docker.com/engine/install/

https://docs.docker.com/compose/install/

下载后确认版本，使用 docker -v 和 docker-compose -v 检查版本。

```
[root@chaincodecarbon ~]# docker -v
Docker version 25.0.3, build 4debf41
[root@chaincodecarbon ~]# docker-compose -v
/usr/local/lib/python3.6/site-packages/paramiko/transport.py:32: CryptographyDeprecationWarning: Python 3.6 is no longer supported
by the Python core team. Therefore, support for it is deprecated in cryptography. The next release of cryptography will remove supp
ort for Python 3.6.
  from cryptography.hazmat.backends import default_backend
docker-compose version 1.29.2, build unknown
```

图 1.1 版本查看

二、部署管理台

使用

```
git clone -b v3.0.0 --depth=1
https://git.chainmaker.org.cn/chainmaker/management-backend.git
```

下载管理台代码，下载后端代码后，进入 management-backend 目录，执行以下命令

```
cd management-backend
docker-compose up
```

如有端口占用可以修改 docker-compose.yml 文件中的端口

```yaml
version: "3.9"

services:
  cm_db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: Baec&chainmaker
      MYSQL_USER: chainmaker
      MYSQL_PASSWORD: Baec&chainmaker
      MYSQL_DATABASE: chainmaker_dev
    command: ['mysqld', '--character-set-server=utf8mb4', '--collation-server=utf8mb4_unicode_ci', '--max_allowed_packet=200M']

  cm_mgmt_server:
    depends_on:
      - cm_db
    image: chainmakerofficial/management-backend:v2.3.2
    ports:
      - "9999:9999"
    restart: always

  cm_mgmt_web:
    depends_on:
      - cm_mgmt_server
    image: chainmakerofficial/management-web:v2.3.2
    ports:
      - "8081:80"
    restart: always

volumes:
  db_data: {}
```

图 2.1 docker-compose 文件

三、通过管理台部署长安链

输入 http://47.97.176.174:8081/login 进入管理平台，用户名为 admin，密码为 a123456

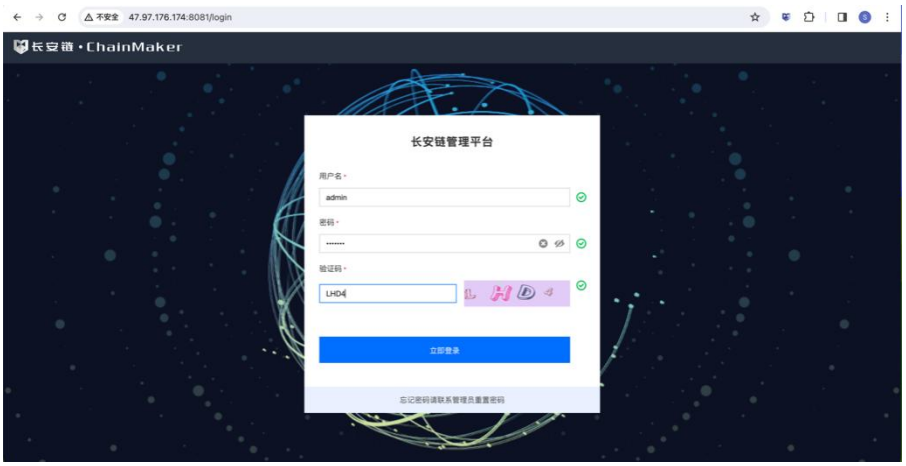

图 3.1 长安链管理平台界面

登录后进入链账户管理，在组织证书，节点证书，用户证书中分别申请 4 个证书，

首先是组织证书：

图 3.2 申请组织证书

共计生成 4 个

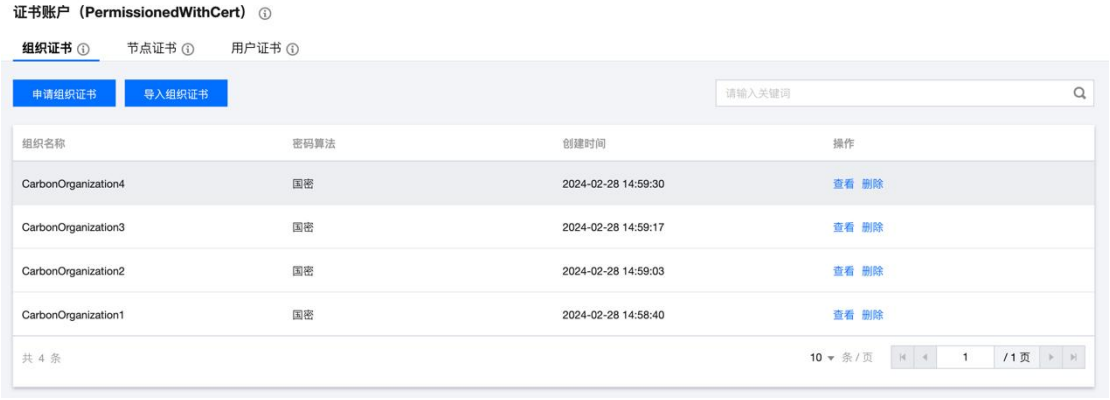

图 3.3 组织证书账户

然后是节点证书

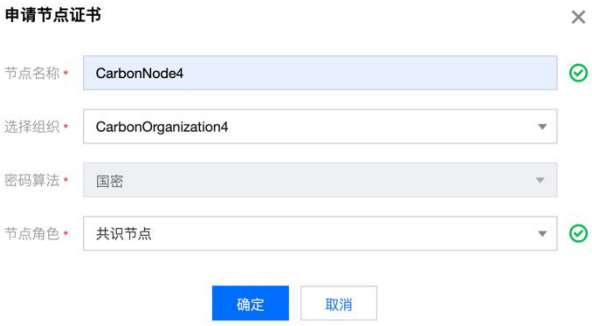

图 3.4 申请节点证书

共计生成 4 个

图 3.5 节点证书账户

最后是用户证书

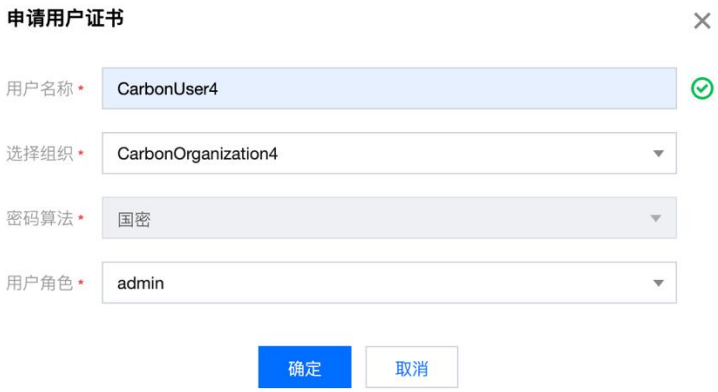

图 3.6 用户证书

共计生成 4 个

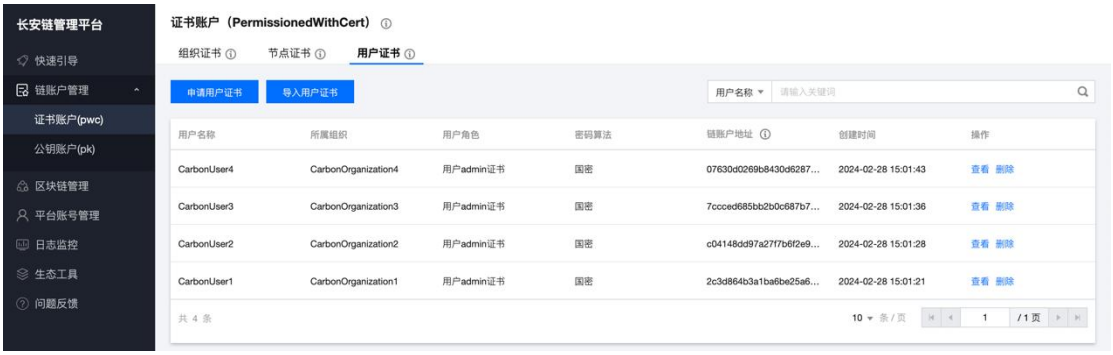

图 3.7 用户证书账户

生成后的证书都生成了签名证书和私钥，都可进行下载

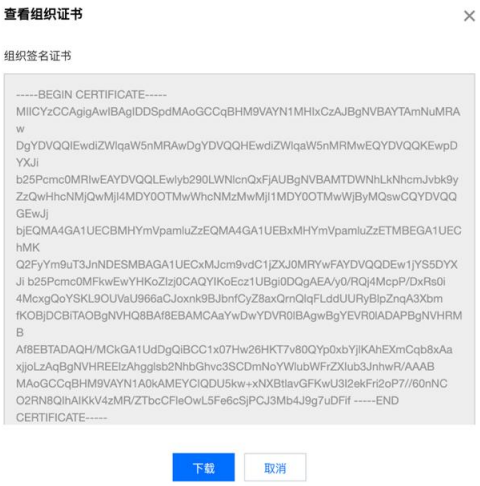

图 3.8 组织签名证书

然后进入区块链管理，在上面点击新建区块链

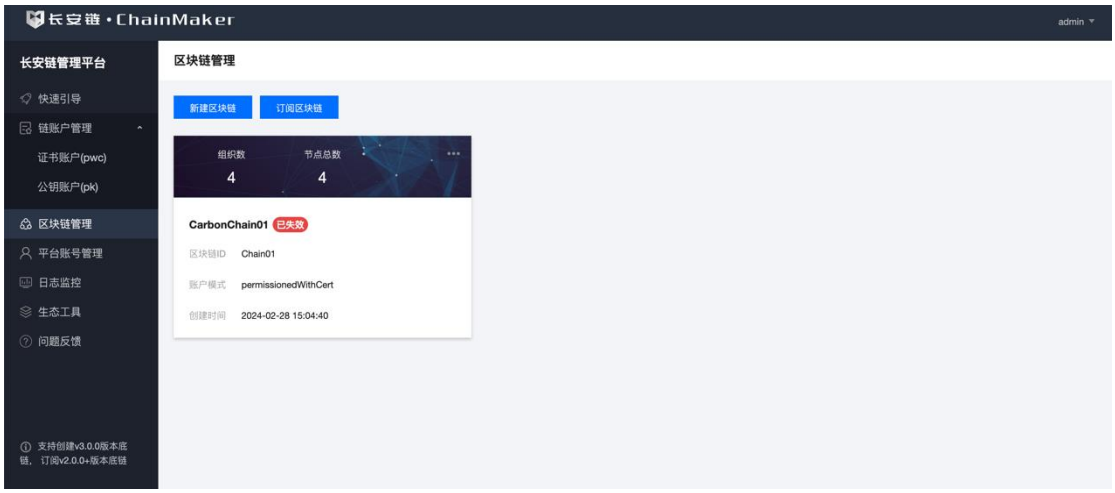

图 3.9 区块链管理

在新建区块链中，依次输入区块链 ID，区块链名称

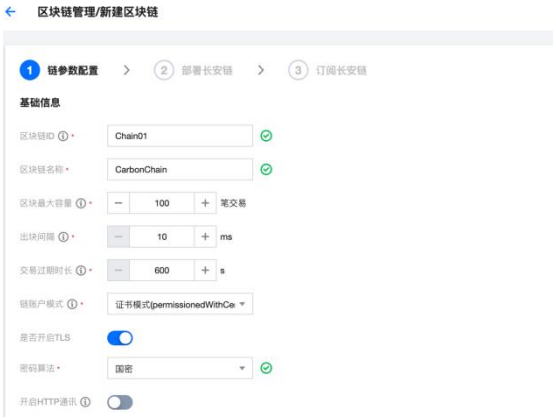

图 3.10 链参数配置 1

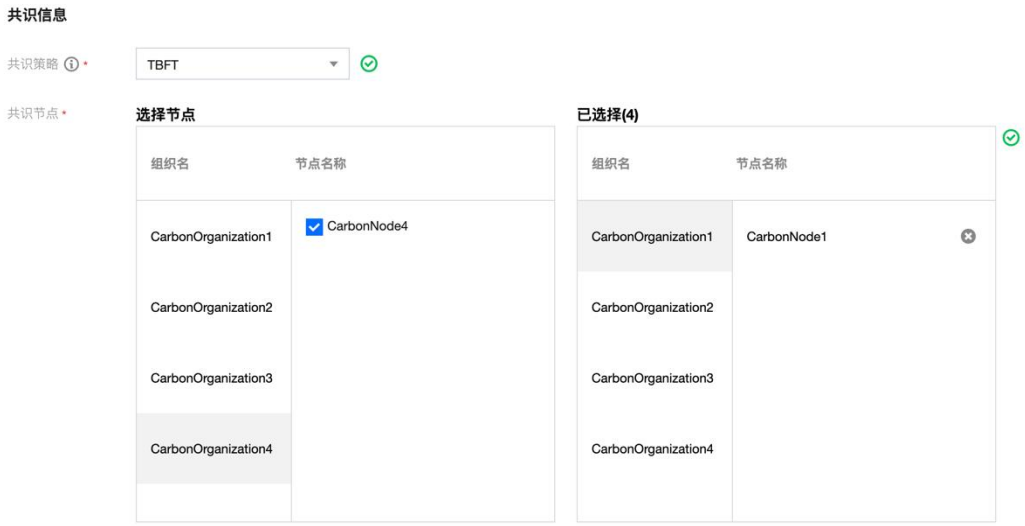

图 3.11 链参数配置 2

图 3.12 链参数配置 3

下一步之后下载链配置文件，解压进入到 release 文件夹，使用./start.sh 启动区块链

图 3.13 部署长安链

并在下一步确认订阅

图 3.14 订阅长安链

至此，区块链搭建完成

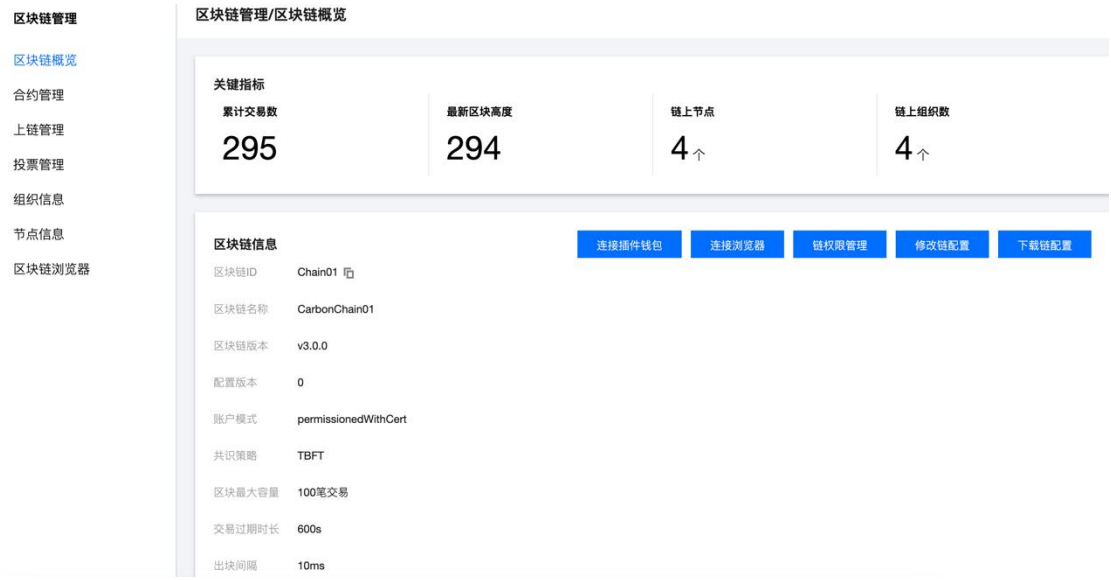

图 3.15 区块链概览

四、合约编写

打开 https://ide230.chainmaker.org.cn/，右键 Workspace 创建合约工程，合约工程名自定，以 CarbonCoin 为例

图 4.1 创建合约工程

具体合约代码如下

```
package main


import (
	"chainmaker.org/chainmaker/contract-sdk-go/v2/pb/protogo"
	"chainmaker.org/chainmaker/contract-sdk-go/v2/sandbox"
	"chainmaker.org/chainmaker/contract-sdk-go/v2/sdk"
	"encoding/json"
	"strconv"
)


// CarbonCoinContract 碳币合约实现
type CarbonCoinContract struct{}


// CreditInfo 用于存储和转移的碳币和碳额度信息
type CreditInfo struct {
	CarbonCoin    int `json:"carbonCoin"`
	CarbonCredit int `json:"carbonCredit"`
}


// InitContract 初始化合约
func (c *CarbonCoinContract) InitContract() protogo.Response {
```

```
	return sdk.Success(nil)
}

// UpgradeContract 升级合约
func (c *CarbonCoinContract) UpgradeContract() protogo.Response {
	return sdk.Success(nil)
}

// InvokeContract 调用合约方法
func (c *CarbonCoinContract) InvokeContract(method string) (result protogo.Response) {

	switch method {
	case "IssueCredit":
		return c.IssueCredit()
	case "QueryCredit":
		return c.QueryCredit()
	case "TransferCredit":
		return c.TransferCredit()
	default:
		return sdk.Error("invalid method")
	}
}

// IssueCredit 分配碳币和碳额度
func (c *CarbonCoinContract) IssueCredit() protogo.Response {
	params := sdk.Instance.GetArgs()
	companyID := string(params["companyID"])
	carbonCoin, err := strconv.Atoi(string(params["carbonCoin"]))
	if err != nil {
		return sdk.Error(err.Error())
```

```
    }
    carbonCredit, err := strconv.Atoi(string(params["carbonCredit"]))
    if err != nil {
        return sdk.Error(err.Error())
    }

    creditInfo := CreditInfo{CarbonCoin: carbonCoin, CarbonCredit: carbonCredit}
    creditInfoBytes, err := json.Marshal(creditInfo)
    if err != nil {
        return sdk.Error(err.Error())
    }

    err = sdk.Instance.PutStateByte(companyID, "creditInfo", creditInfoBytes)
    if err != nil {
        return sdk.Error(err.Error())
    }

    return sdk.Success(nil)
}

// QueryCredit 查询公司的碳币和碳额度
func (c *CarbonCoinContract) QueryCredit() protogo.Response {
    params := sdk.Instance.GetArgs()
    companyID := string(params["companyID"])

    creditInfoBytes, err := sdk.Instance.GetStateByte(companyID, "creditInfo")
    if err != nil {
        return sdk.Error(err.Error())
    }
    if creditInfoBytes == nil {
```

```
        return sdk.Error("credit info not found")
    }

    return sdk.Success(creditInfoBytes)
}

// TransferCredit 两公司之间转移碳币和碳额度
func (c *CarbonCoinContract) TransferCredit() protogo.Response {
    params := sdk.Instance.GetArgs()
    companyAID := string(params["companyAID"])
    companyBID := string(params["companyBID"])
    carbonCoinToTransfer, err := strconv.Atoi(string(params["carbonCoin"]))
    if err != nil {
        return sdk.Error("invalid carbonCoin: " + err.Error())
    }
    carbonCreditToTransfer, err := strconv.Atoi(string(params["carbonCredit"]))
    if err != nil {
        return sdk.Error("invalid carbonCredit: " + err.Error())
    }

    // 查询公司 A 的碳币和碳额度信息
    creditInfoABytes, err := sdk.Instance.GetStateByte(companyAID, "creditInfo")
    if err != nil {
        return sdk.Error("failed to get credit info for company A: " + err.Error())
    }
    if creditInfoABytes == nil {
        return sdk.Error("credit info for company A not found")
    }
    var creditInfoA CreditInfo
    err = json.Unmarshal(creditInfoABytes, &creditInfoA)
```

```
    if err != nil {
        return sdk.Error("failed to unmarshal credit info for company A: " + err.Error())
    }

    // 查询公司 B 的碳币和碳额度信息
    creditInfoBBytes, err := sdk.Instance.GetStateByte(companyBID, "creditInfo")
    if err != nil {
        return sdk.Error("failed to get credit info for company B: " + err.Error())
    }
    if creditInfoBBytes == nil {
        return sdk.Error("credit info for company B not found")
    }
    var creditInfoB CreditInfo
    err = json.Unmarshal(creditInfoBBytes, &creditInfoB)
    if err != nil {
        return sdk.Error("failed to unmarshal credit info for company B: " + err.Error())
    }

    // 执行转移逻辑
    if creditInfoA.CarbonCoin < carbonCoinToTransfer || creditInfoA.CarbonCredit <
carbonCreditToTransfer {
        return sdk.Error("company A does not have enough carbonCoin or carbonCredit
to transfer")
    }
    creditInfoA.CarbonCoin -= carbonCoinToTransfer
    creditInfoA.CarbonCredit -= carbonCreditToTransfer
    creditInfoB.CarbonCoin += carbonCoinToTransfer
    creditInfoB.CarbonCredit += carbonCreditToTransfer

    // 更新公司 A 的碳币和碳额度信息
```

```
    creditInfoABytes, err = json.Marshal(creditInfoA)
    if err != nil {
        return sdk.Error("failed to marshal updated credit info for company A: " +
err.Error())
    }
    err = sdk.Instance.PutStateByte(companyAID, "creditInfo", creditInfoABytes)
    if err != nil {
        return sdk.Error("failed to update credit info for company A: " + err.Error())
    }

    // 更新公司 B 的碳币和碳额度信息
    creditInfoBBytes, err = json.Marshal(creditInfoB)
    if err != nil {
        return sdk.Error("failed to marshal updated credit info for company B: " +
err.Error())
    }
    err = sdk.Instance.PutStateByte(companyBID, "creditInfo", creditInfoBBytes)
    if err != nil {
        return sdk.Error("failed to update credit info for company B: " + err.Error())
    }

    return sdk.Success(nil)
}

func main() {
    err := sandbox.Start(new(CarbonCoinContract))
    if err != nil {
        sdk.Instance.Errorf(err.Error())
    }
}
```

接着点击右边保存

图 4.2 点击保存

然后分析这里面的代码，

看如下代码得知，我们有 IssueCredit，QueryCredit，TransferCredit 方法：

```
// InvokeContract 调用合约方法
func (c *CarbonCoinContract) InvokeContract(method string) (result protogo.Response) {


    switch method {
    case "IssueCredit":
        return c.IssueCredit()
    case "QueryCredit":
        return c.QueryCredit()
    case "TransferCredit":
        return c.TransferCredit()
    default:
        return sdk.Error("invalid method")
    }
}
```

以 IssueCredit 方法举例，这里看到具体的方法：

```
func (c *CarbonCoinContract) IssueCredit() protogo.Response {
    params := sdk.Instance.GetArgs()
    companyID := string(params["companyID"])
    carbonCoin, err := strconv.Atoi(string(params["carbonCoin"]))
    if err != nil {
        return sdk.Error(err.Error())
    }
```

```
    carbonCredit, err := strconv.Atoi(string(params["carbonCredit"]))
    if err != nil {
        return sdk.Error(err.Error())
    }

    creditInfo := CreditInfo{CarbonCoin: carbonCoin, CarbonCredit: carbonCredit}
    creditInfoBytes, err := json.Marshal(creditInfo)
    if err != nil {
        return sdk.Error(err.Error())
    }

    err = sdk.Instance.PutStateByte(companyID, "creditInfo", creditInfoBytes)
    if err != nil {
        return sdk.Error(err.Error())
    }

    return sdk.Success(nil)
}
```

这里可以看到 companyID, carbonCoin，carbonCredit，这个是我们在调试中所用到的 key，value 就是具体传的值

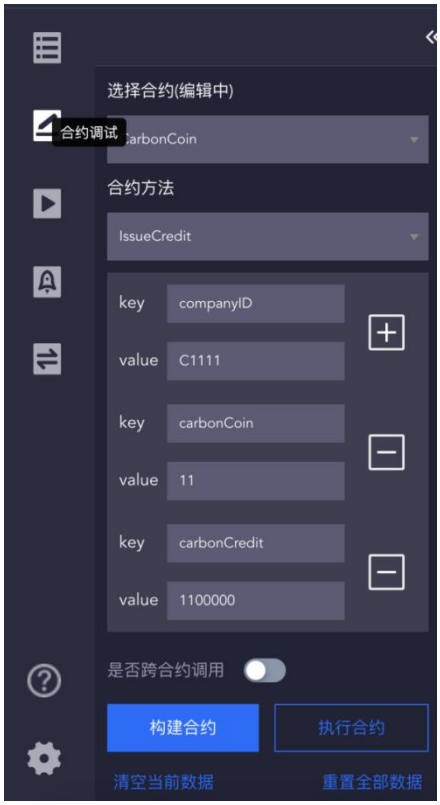

图 4.3 部署长安链

这里选择合约，然后选择合约方法，输入对应信息，然后构建合约，执行合约，在模拟区块链上进行调试。

合约没问题之后点击第三个按钮合约编译，编译完会有一个.7z 的下载包下载下来。

图 4.4 合约编译

五、合约部署

进入区块链管理平台，进来之后找到区块链管理，合约管理，这里部署合约，

图 5.1 合约管理界面

具体部署合约如下，合约名称和一开始在 IDE 里新建的合约名称一致，合约版本自定义，虚拟机类型 DOCKER_GO，合约文件则将刚刚下载的.7z 后缀文件上传。

这里尤其注意合约调用方法，这里需要把合约里的方法都写上去，这里我以 IssueCredit 举例，写了这么一个方法，然后它的 key 是 companyID,carbonCoin,carbonCredit，需把这些都要写进去，将下面的图一一对应起来。

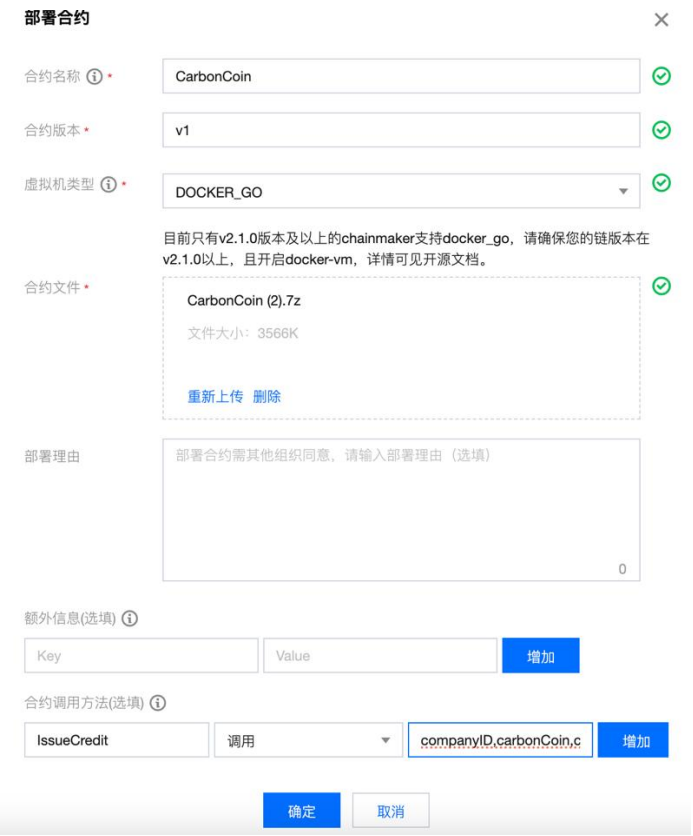

图 5.2 部署合约界面

部署后点击确定，然后如果要升级合约，具体操作和原来大致相同，部署完成后找到上链管理，发起上链。

图 5.3 上链管理界面

选择合约，交互类型，调用方法，对应添加每个参数，运行一下可以检查每个功能能否正确部署

图 5.4 发起上链界面

能正确上链的基本都是交易成功，如果不成功可能是 value 输入有误。

更新时间	交易ID	发起组织	发起用户	合约	上链状态	交易状态	操作
2024-03-07 ...	81535193e40f4c95b22d94669a8b1d8ea6650a58...	CarbonOrga...	CarbonUser1	CarbonCoin	已上链	交易成功	查看
2024-03-07 ...	9dc17a1b5cd3496f89548f7071f0baabea2711c8d...	CarbonOrga...	CarbonUser1	CarbonCoin	已上链	交易成功	查看
2024-03-07 ...	e5a67f7f967848e6995edc6fb5f1528ae4013496ed...	CarbonOrga...	CarbonUser1	CarbonCoin	已上链	交易成功	查看

图 5.5 上链列表

然后点击合约可以查看具体信息

合约执行信息

所属合约 CarbonCoin

消耗GAS量 2572

合约调用参数

invoke_contract

#	Key	Value
0	companyAID	tc1
1	companyBID	tc2
2	carbonCoin	1
3	carbonCredit	1000
4	method	TransferCredit

图 5.6 上链信息

至此示例合约部署完成。