



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 .....	3
二、部署管理台 .....	3
三、通过管理台部署长安链 .....	1
四、合约编写 .....	6
五、合约部署 .....	16

## 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@chaincodearbon ~]# docker -v
Docker version 25.0.3, build 4deb41
[root@chaincodearbon ~]# 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 support
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

```
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

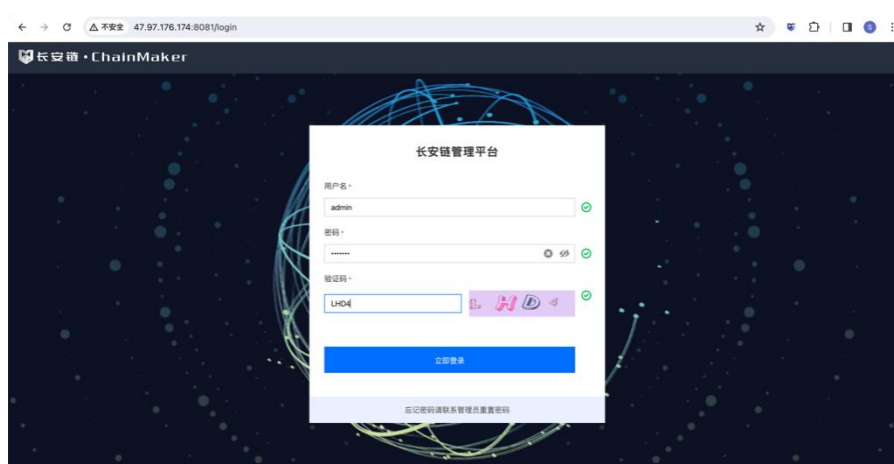


图 3.1 长安链管理平台界面

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：

申请组织证书

组织ID \*

CarbonOrganization5

✓

组织名称 \*

CarbonOrganization5

✓

密码算法 \*

国密

▼

确定

取消

图 3.2 申请组织证书

共计生成 4 个

证书账户 (PermissionedWithCert) ⓘ

组织证书 ⓘ

节点证书 ⓘ

用户证书 ⓘ

申请组织证书

导入组织证书

请输入关键词

Q

组织名称	密码算法	创建时间	操作
CarbonOrganization4	国密	2024-02-28 14:59:30	<a href="#">查看</a> <a href="#">删除</a>
CarbonOrganization3	国密	2024-02-28 14:59:17	<a href="#">查看</a> <a href="#">删除</a>
CarbonOrganization2	国密	2024-02-28 14:59:03	<a href="#">查看</a> <a href="#">删除</a>
CarbonOrganization1	国密	2024-02-28 14:58:40	<a href="#">查看</a> <a href="#">删除</a>

共 4 条

10 条 / 页

1

/ 1 页

图 3.3 组织证书账户

然后是节点证书

申请节点证书

节点名称 \*

CarbonNode4

✓

选择组织 \*

CarbonOrganization4

▼

密码算法 \*

国密

▼

节点角色 \*

共识节点

▼

✓

确定

取消

图 3.4 申请节点证书

共计生成 4 个

证书账户 (PermissionedWithCert) ①

组织证书 ①

节点证书 ①

用户证书 ①

申请节点证书

导入节点证书

节点名称

请输入关键词

Q

节点名称	所属组织	节点角色	密码算法	创建时间	操作
CarbonNode4	CarbonOrganization4	共识节点	国密	2024-02-28 15:00:52	<a href="#">查看</a> <a href="#">删除</a>
CarbonNode3	CarbonOrganization3	共识节点	国密	2024-02-28 15:00:38	<a href="#">查看</a> <a href="#">删除</a>
CarbonNode2	CarbonOrganization2	共识节点	国密	2024-02-28 15:00:25	<a href="#">查看</a> <a href="#">删除</a>
CarbonNode1	CarbonOrganization1	共识节点	国密	2024-02-28 15:00:16	<a href="#">查看</a> <a href="#">删除</a>

共 4 条

10 条 / 页

1

/ 1 页

图 3.5 节点证书账户

最后为用户证书

申请用户证书

×

用户名称 \*

CarbonUser4

✓

选择组织 \*

CarbonOrganization4

▼

密码算法 \*

国密

▼

用户角色 \*

admin

▼

确定

取消

图 3.6 用户证书

共计生成 4 个

长安链管理平台

快速引导

链账户管理

证书账户 (pwc)

公钥账户 (pk)

区块链管理

平台账号管理

日志监控

生态工具

问题反馈

证书账户 (PermissionedWithCert) ①

组织证书 ①

节点证书 ①

用户证书 ①

申请用户证书

导入用户证书

用户名称

请输入关键词

Q

用户名称	所属组织	用户角色	密码算法	链账户地址 ①	创建时间	操作
CarbonUser4	CarbonOrganization4	用户admin证书	国密	07630d0269b8430d5287...	2024-02-28 15:01:43	<a href="#">查看</a> <a href="#">删除</a>
CarbonUser3	CarbonOrganization3	用户admin证书	国密	7ccced685bb2b0c687b7...	2024-02-28 15:01:36	<a href="#">查看</a> <a href="#">删除</a>
CarbonUser2	CarbonOrganization2	用户admin证书	国密	c04148dd97a2777b6f2e9...	2024-02-28 15:01:28	<a href="#">查看</a> <a href="#">删除</a>
CarbonUser1	CarbonOrganization1	用户admin证书	国密	2c3d864b3a1ba6be25a6...	2024-02-28 15:01:21	<a href="#">查看</a> <a href="#">删除</a>

共 4 条

10 条 / 页

1

/ 1 页

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



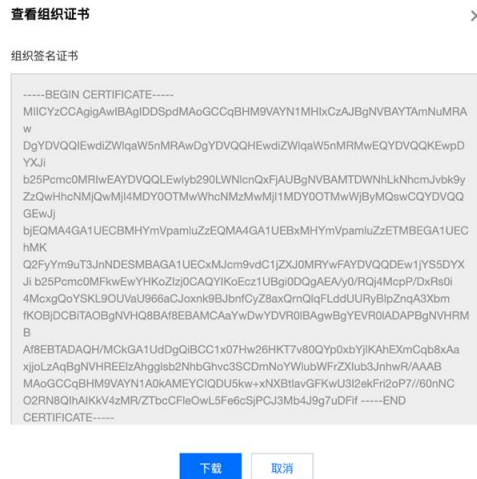


图 3.8 组织签名证书

然后进入区块链管理，在上面点击新建区块链

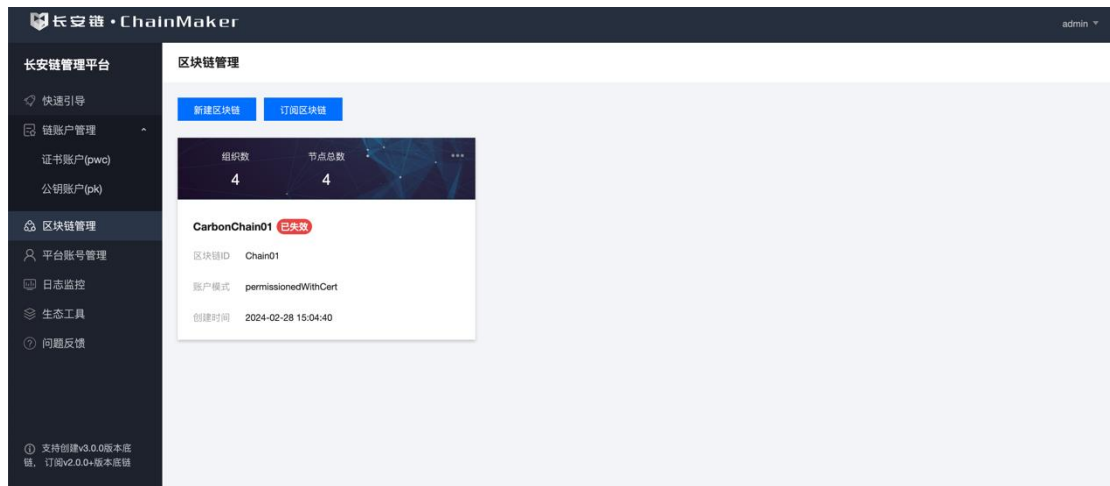


图 3.9 区块链管理

在新建区块链中，依次输入区块链 ID，区块链名称

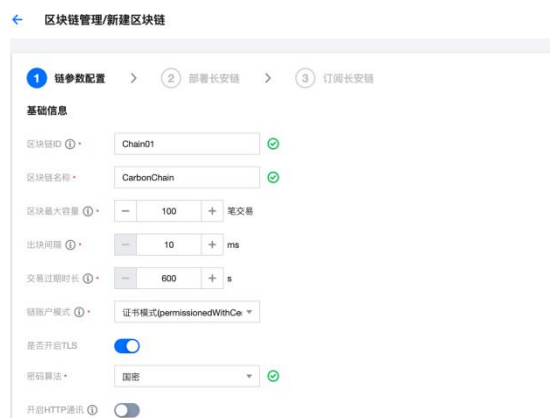


图 3.10 链参数配置 1

共识信息

共识策略 ⓘ TBFT

共识节点 ⓘ

选择节点

组织名	节点名称
CarbonOrganization1	<input checked="" type="checkbox"/> CarbonNode4
CarbonOrganization2	
CarbonOrganization3	
CarbonOrganization4	

已选择(4)

组织名	节点名称
CarbonOrganization1	CarbonNode1
CarbonOrganization2	
CarbonOrganization3	
CarbonOrganization4	

图 3.11 链参数配置 2

节点部署配置

部署方式 ⓘ ☒ 单机部署 ☐ 多机部署

CarbonNode1	47.97.176.174	12301	11301
CarbonNode2	同上	12302	11302
CarbonNode3	同上	12303	11303
CarbonNode4	同上	12304	11304

合约虚拟机配置

支持的虚拟机类型 ☒ DOCKER\_GO ☒ WASMER ☒ EVM ☒ WXVM ☒ GASM

默认支持WASMER、EVM、WXVM、GASM，支持Rust、Solidity，C++语言的合约。  
如果您想部署Go语言的合约的话，需要勾选支持Docker\_go，勾选后在部署链时除链的主程序外，还会再额外部署一个DockerVM。

报错日志采集服务 (可选项) ⓘ

监控开关 ⓘ ☒ 开启 ☐ 关闭

请先阅读《chainmaker管理平台数据收集声明协议》再确定是否开启,开启监控开关即表示您同意本协议

是否参加长安链改进计划 ⓘ ☒ 开启 ☐ 关闭

取消

下一步

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

### 部署链教程

1. 下载链配置文件压缩包
2. 单机部署的话，将压缩包移动到要部署的Linux系统机器上，解压缩包，执行release 目录下的start.sh脚本即可启动链，命令如下

```
$ cd release
$ ./start.sh
```

3. 多机部署的话，则需要分别在要部署的机器上执行上述操作，直至全部节点被成功启动，才算部署完成。注意各机器间需要保证网络通畅，才能完成节点P2P组网。
4. 成功部署区块链后，点击下一步，用管理平台订阅已部署的链。

下一步

图 3.13 部署长安链



并在下一步确认订阅

链参数配置

部署长安链

订阅长安链

1. 本处配置的节点，将用于监听链的信息，以及往链上发送交易，请配置该链上的节点，并确保稳定运行。

2. 本处配置的用户，将在后续往链上发送交易时，用于交易签名，请配置该链上的用户，以确保签名有效。

3. 若您要监听的链的证书不是通过本平台生成的，则请先导入相关组织证书、节点证书、用户证书等，再配置。

链ID

iftest01

链账户模式

证书模式(permissionedWithCe)

组织名称

wxorg1chainmakerorg

节点RPC地址

192.168.1.203:12301

用户名称

wxorguser1

是否开启TLS

上一步

确认订阅

常见问题

1. 如果发现管理平台区块高度落后于链的情况，请重新订阅。

2. 订阅链失败，与节点连接失败，请检查节点端口是否开放，则检查以下几点：

- 链节点端口是否开放。
- 所订阅的链的节点部署的ip和端口是否和生成节点证书时填写的一致。
- 链是否为容器启动，容器启动的链不能用容器启动的管理台订阅，因为网络是不通的。

3. 订阅链失败，证书错误，请检查用户证书是否正确：

- 一般导入外部链容易出现，请检查是否全部证书导入正常，是否和链使用的证书为同一套。

4. 订阅链失败，tls握手失败，请检查证书是否正确：

- 一般导入外部链容易出现，请检查TLS证书是否正确导入，是否是订阅组织下的用户。

5. 订阅链失败，chainId错误，请检查chainId是否正确：

- 一般导入外部链容易出现，请检查订阅的链的chainId和外部链是否一致。

图 3.14 订阅长安链

至此，区块链搭建完成

区块链管理

区块链概览

合约管理

上链管理

投票管理

组织信息

节点信息

区块链浏览器

区块链管理/区块链概览

关键指标

累计交易数

295

最新区块高度

294

链上节点

4个

链上组织数

4个

区块链信息

区块链ID

Chain01

区块链名称

CarbonChain01

区块链版本

v3.0.0

配置版本

0

账户模式

permissionedWithCert

共识策略

TBFT

区块链最大容量

100笔交易

交易过期时长

600s

出块间隔

10ms

连接插件钱包

连接浏览器

链权限管理

修改链配置

下载链配置

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

6

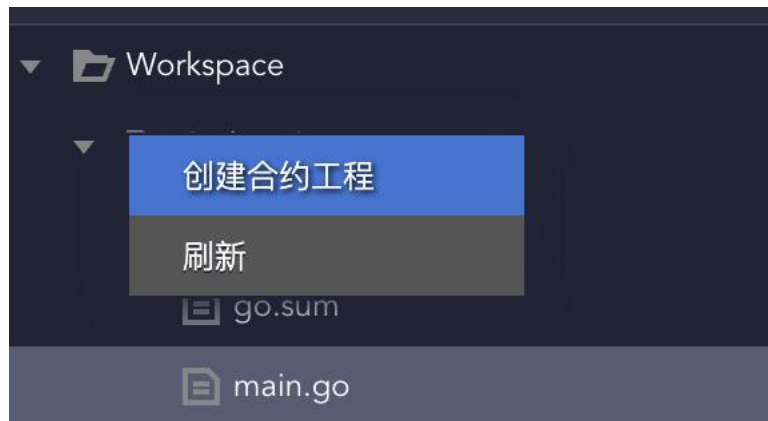


图 4.1 创建合约工程

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

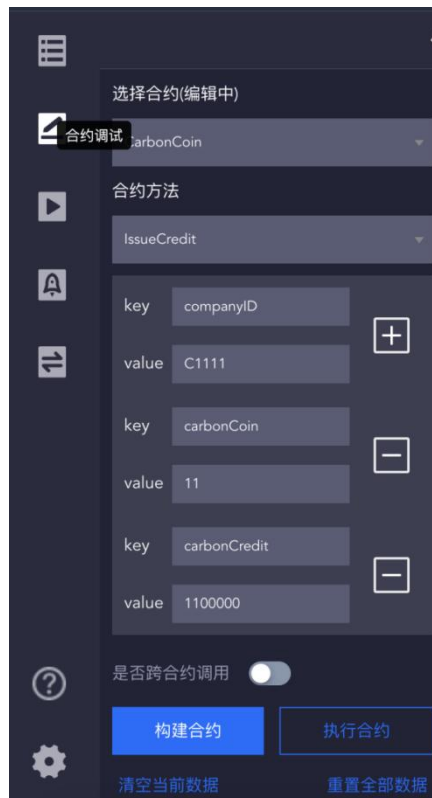


图 4.3 部署长安链

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。

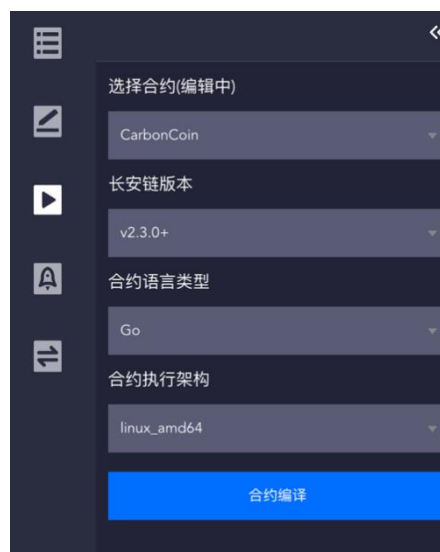


图 4.4 合约编译

## 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，

区块链管理	区块链管理/合约管理						
区块链概览	部署合约						
合约管理	请输入合约名称搜索						
上链管理	合约名称	当前版本	所属组织	创建者	更新时间	投票状态	链上状态
投票管理	CarbonModel	v1.2	CarbonOrg1	2c3d864b3a1ba...	2024-03-19 12:1...	正常	正常
组织信息	CarbonEmission	v1	CarbonOrg1	2c3d864b3a1ba...	2024-03-12 14:2...	正常	正常
节点信息	Register	v1	CarbonOrg1	2c3d864b3a1ba...	2024-03-12 11:3...	正常	正常
区块链浏览器	CarbonTrade	v1.2	CarbonOrg1	2c3d864b3a1ba...	2024-03-12 08:4...	正常	正常
	CarbonCoin	v1.3	CarbonOrg1	2c3d864b3a1ba...	2024-03-19 12:0...	正常	正常
	car	v1	CarbonOrg1	2c3d864b3a1ba...	2024-03-12 14:4...	正常	已注销
	carbon	v1	CarbonOrg1	2c3d864b3a1ba...	2024-03-05 10:4...	正常	已注销
	godemocontract	1.0.0	CarbonOrg1	2c3d864b3a1ba...	2024-03-12 14:4...	正常	已注销

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。

部署合约

合约名称

CarbonCoin

合约版本

v1

虚拟机类型

DOCKER\_GO

合约文件

CarbonCoin (2).7z

文件大小: 3566K

重新上传 删除

部署理由

部署合约需其他组织同意, 请输入部署理由 (选填)

0

额外信息(选填)

Key

Value

增加

合约调用方法(选填)

IssueCredit

调用

companyID,carbonCoin,c

增加

确定

取消

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。

长安链管理平台

快速引导

链账户管理

区块链管理

平台账号管理

日志监控

生态工具

问题反馈

区块链管理

区块链概览

合约管理

上链管理

投票管理

组织信息

节点信息

区块链浏览器

区块链管理/上链管理

发起上链

更新时间	交易ID
2024-03-07 ...	81535193e40f4c95b22d94669a8b1
2024-03-07 ...	9dc17a1b5cd3496f89548f7071f0be
2024-03-07 ...	e5a67f7f967848e6995edc6fb5f152l
2024-03-07 ...	813d82dee15d49f8be6b210d49adc
2024-03-07 ...	32de23bf75094bdf98adf3eb745e1e
2024-03-07 ...	885315688f39426686466d7b9927C

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

发起上链

选择合约 \*

CarbonCoin

✓

交互类型 ⓘ \*

调用

调用方法 ⓘ \*

IssueCredit

✓

参数 ⓘ

companyID

c

增加

carbonCoin

1

删除

carbonCredit

1

删除

确定

取消

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。