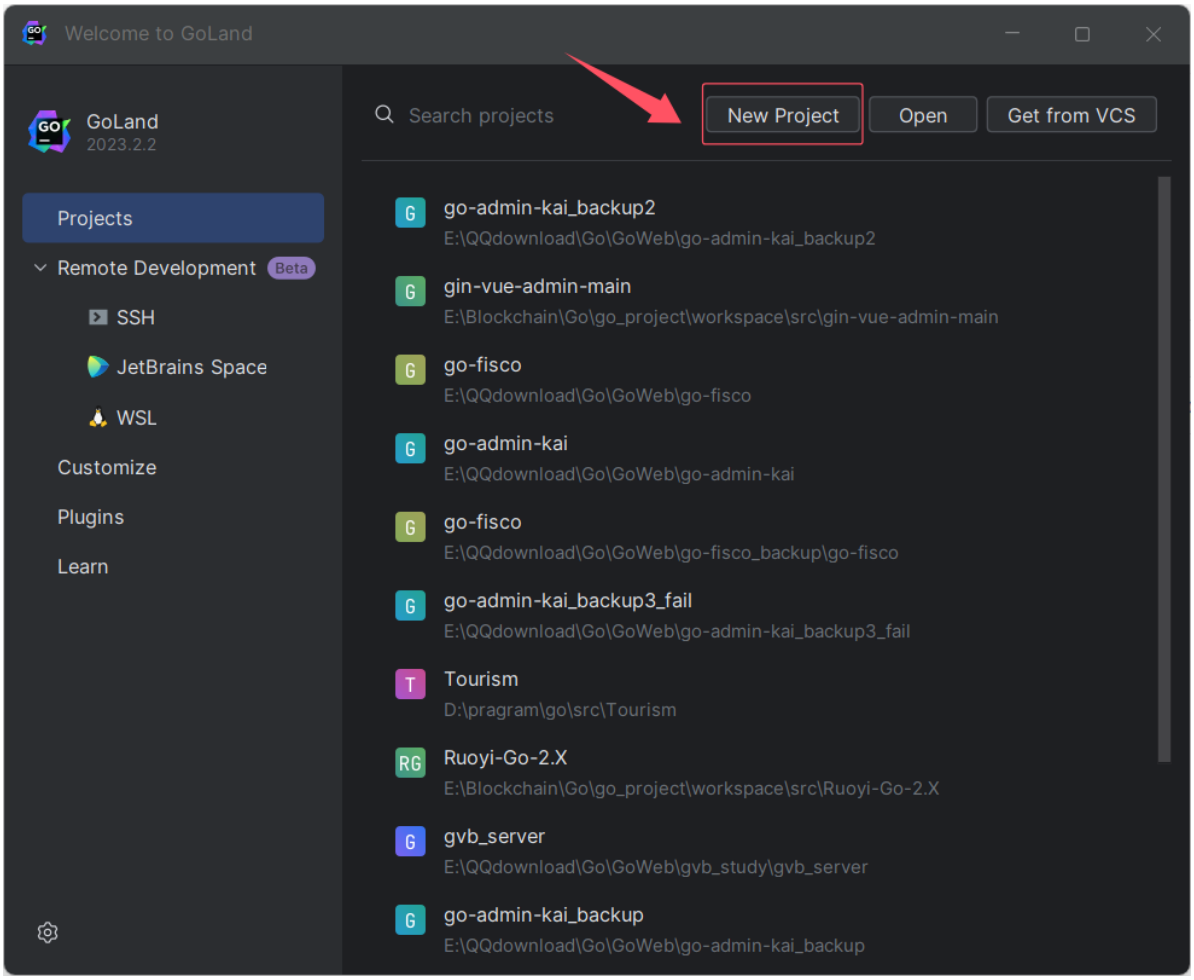
FISCOBCOS中的GoSDK操作---(引入工具包版本)

文档操作记录

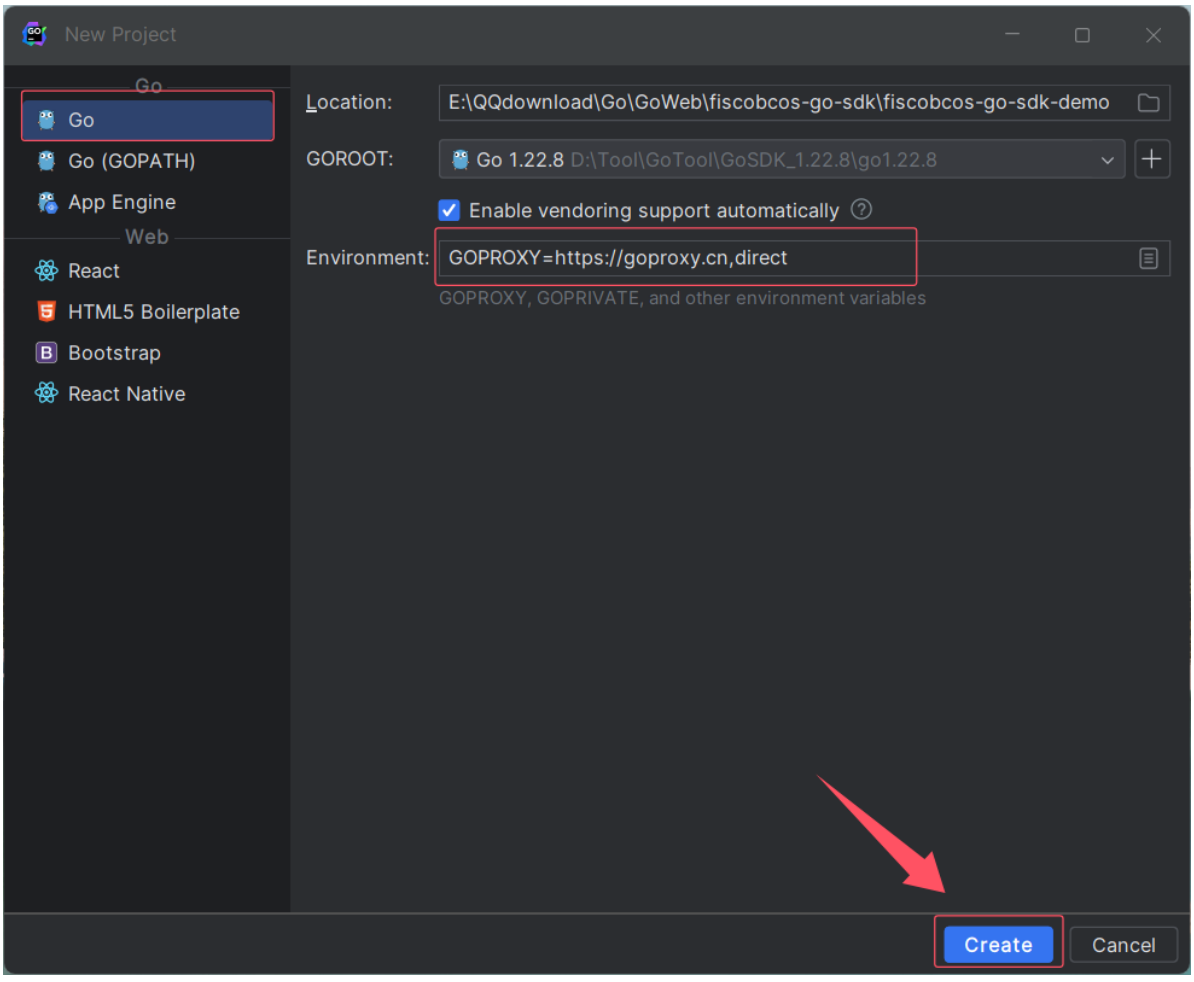
内容	作者	时间	版本号	联系方式
文档创建	陈雅凯，高俭豪，谭俊	2024-11-16	1.0	email:2040575063@qq.com

一，GoWEB项目创建：

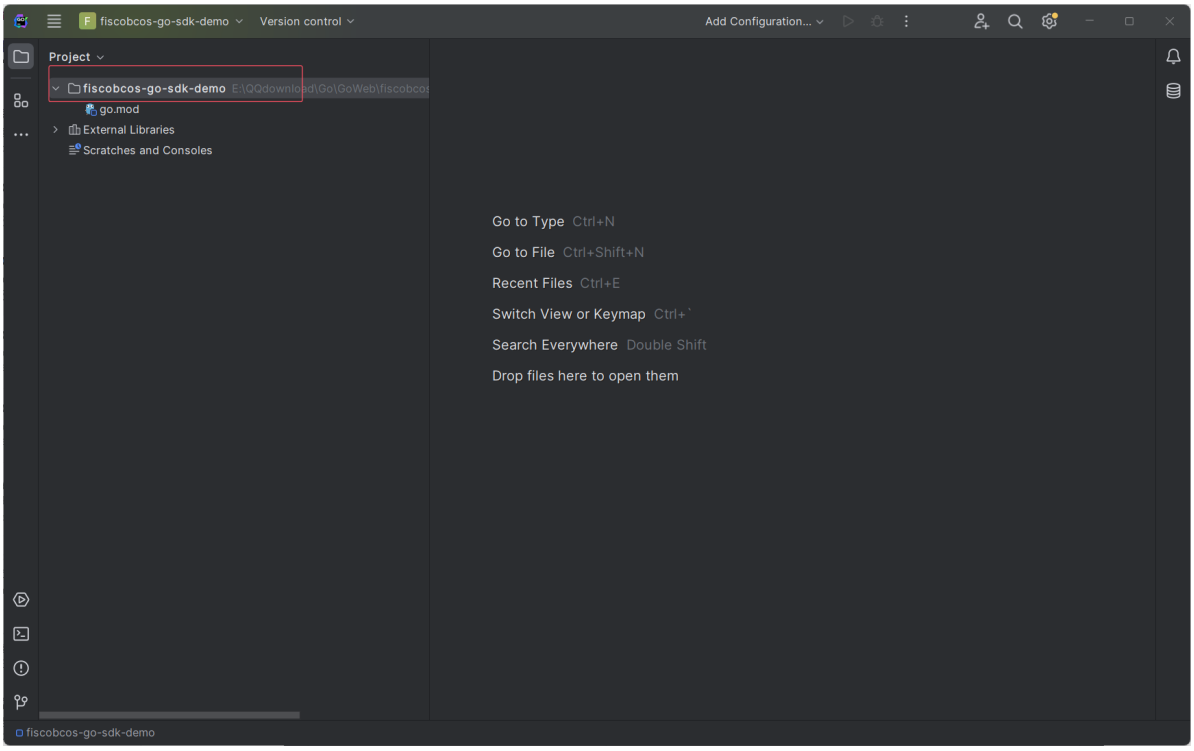
1.新建项目：


Welcome to GoLand
GoLand
2023.2.2
Search projects
New Project
Open
Get from VCS
Projects
Remote Development
Beta
SSH
JetBrains Space
WSL
Customize
Plugins
Learn
go-admin-kai_backup2
E:\QQdownload\Go\GoWeb\go-admin-kai_backup2
gin-vue-admin-main
E:\Blockchain\Go\go_project\workspace\src\gin-vue-admin-main
go-fisco
E:\QQdownload\Go\GoWeb\go-fisco
go-admin-kai
E:\QQdownload\Go\GoWeb\go-admin-kai
go-fisco
E:\QQdownload\Go\GoWeb\go-fisco_backup\go-fisco
go-admin-kai_backup3_fail
E:\QQdownload\Go\GoWeb\go-admin-kai_backup3_fail
Tourism
D:\pragram\go\src\Tourism
Ruoyi-Go-2.X
E:\Blockchain\Go\go_project\workspace\src\Ruoyi-Go-2.X
gvb_server
E:\QQdownload\Go\GoWeb\gvb_study\gvb_server
go-admin-kai_backup
E:\QQdownload\Go\GoWeb\go-admin-kai_backup

2.项目路径（随自己心意）：


New Project
Go
Go
Go (GOPATH)
App Engine
Web
React
HTML5 Boilerplate
Bootstrap
React Native
Location:
E:\QQdownload\Go\GoWeb\fiscobcos-go-sdk\fiscobcos-go-sdk-demo
GOROOT:
Go 1.22.8 D:\Tool\GoTool\GoSDK_1.22.8\go1.22.8
Enable vendoring support automatically
Environment:
GOPROXY=https://goproxy.cn,direct
GOPROXY, GOPRIVATE, and other environment variables
Create
Cancel

3.生成成功：



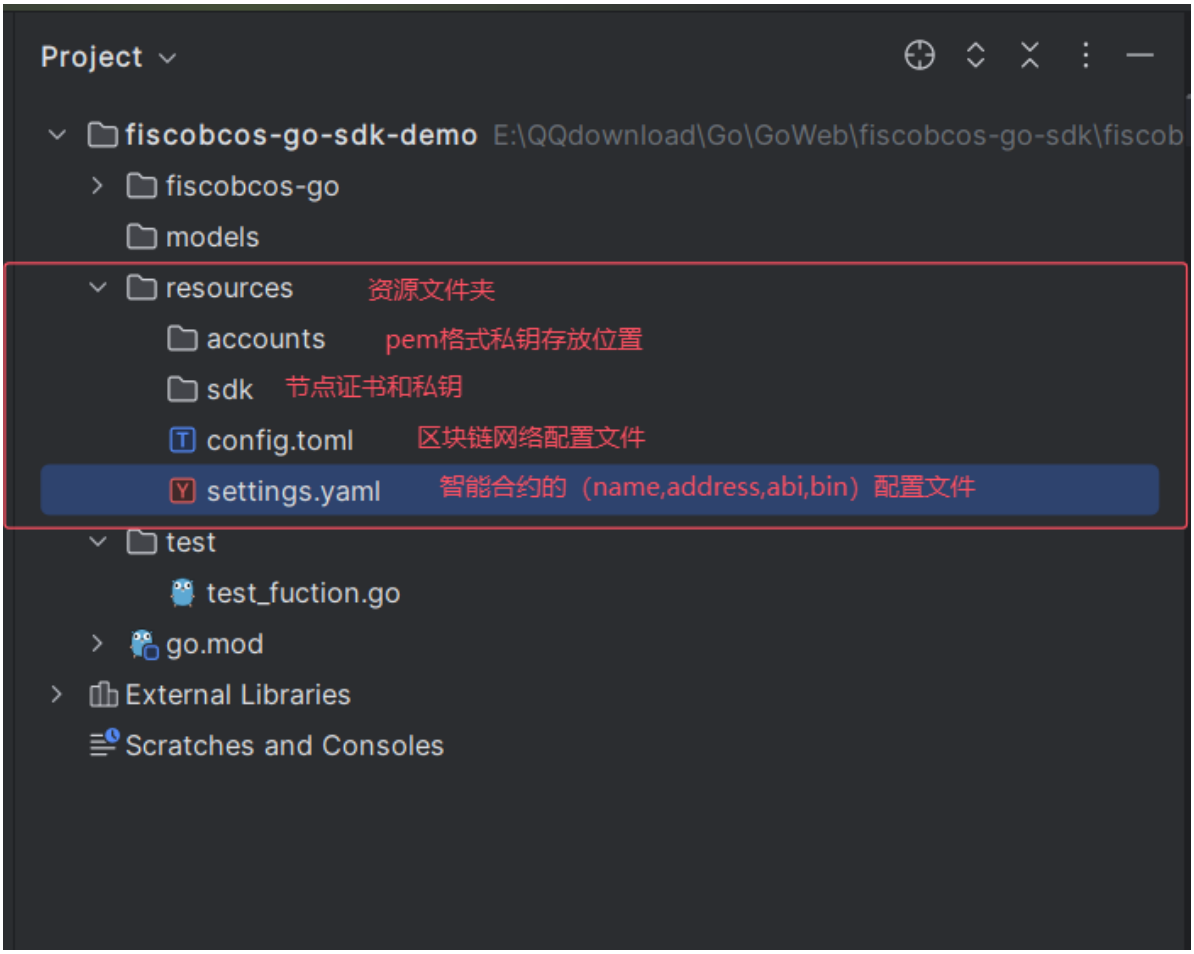
二，项目书写：

1.创建目录：

注意事项：

fiscobcos-go在打包好的文件夹中寻找。导入之后会提示项目目录爆红，修改成自己的就好了

resources文件夹配置：



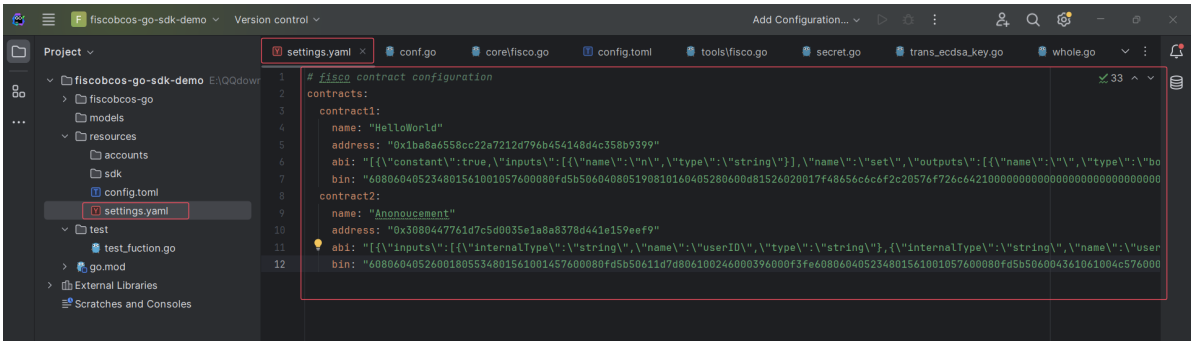
导入工具包fiscobcos-go之后(需要引入的依赖)：

```
go get -u github.com/FISCO-BCOS/go-sdk
go get -u gopkg.in/yaml.v3
```

2.合约配置文件的读取：

书写配置文件settings.yaml(书写时，输入自己合约对应的name(合约名称),address,abi,bin。bin可以填写，也可以不用填写)：

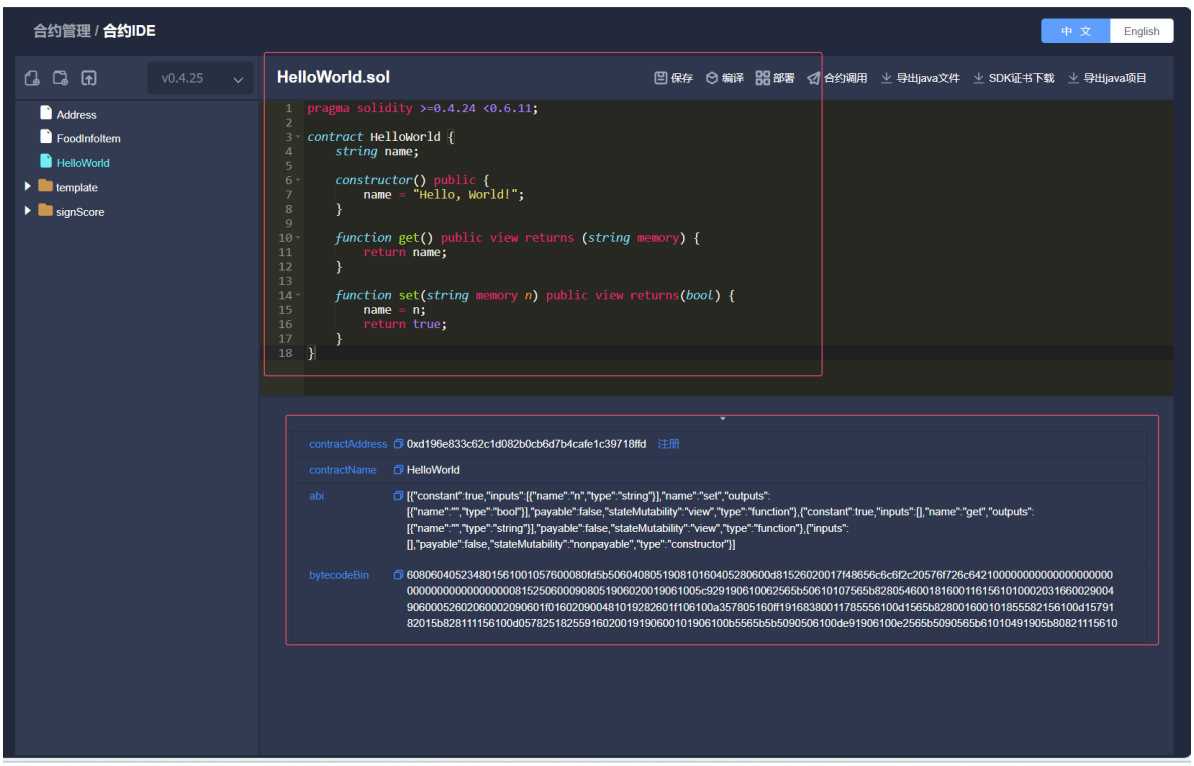
```
contracts:
 contract1: (这里的设计只是一个键对应一个结构体数据)
  name:
  address:
  abi:
  bin:
 contract2:(这里的设计只是一个键对应一个结构体数据)
  name:
  address:
  abi:
  bin:
```



注意事项：

在WeBASE-Front上编译合约获得abi和bin,部署合约获得address。

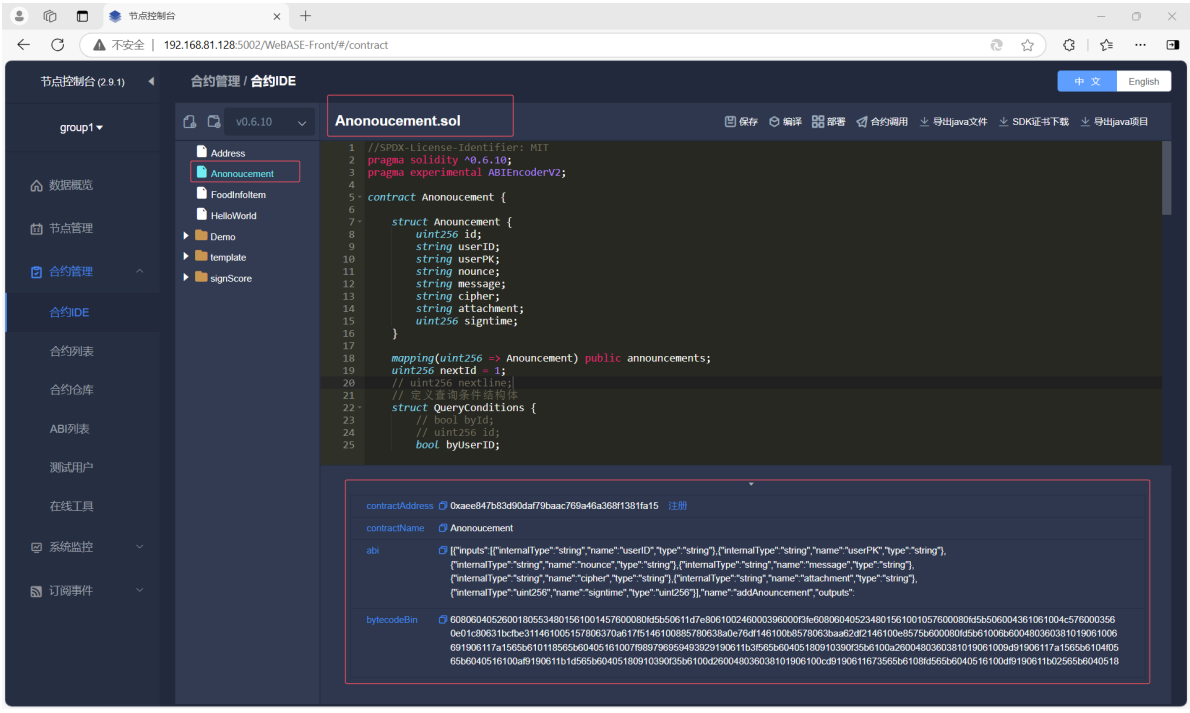
```
pragma solidity >=0.4.24 <0.6.11;

contract HelloWorld {
    string name;

    constructor() public {
        name = "Hello, World!";
    }

    function get() public view returns (string memory) {
        return name;
    }

    function set(string memory n) public view returns(bool) {
        name = n;
        return true;
    }
}
```



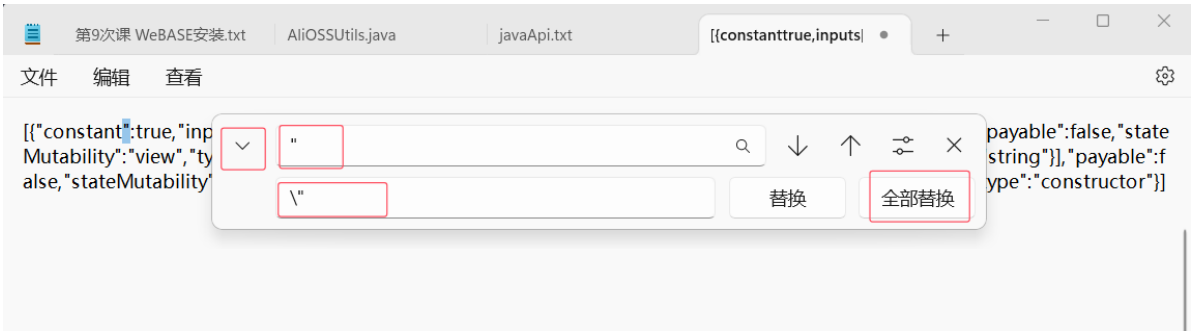
```
//SPDX-License-Identifier: MIT
pragma solidity ^0.6.10;
pragma experimental ABIEncoderV2;

contract Anonoucement {

    struct Anouncement {
        uint256 id;//宣言ID
        string userID;//用户编号
        string userPK;//用户的公钥
        string nounce;//签名的随机值
        string message;//消息
        string cipher;//消息的密文    --消息
        string attachment;//附件的摘要    --哈希
        uint256 signtime;//存证时间
    }

    mapping(uint256 => Anouncement) public announcements;
    uint256 nextId = 1;
    // uint256 nextline;
    // 定义查询条件结构体
    struct QueryConditions {
        // bool byId;
        // uint256 id;
        bool byUserID;
        string userID;
        bool byTimeRange;
        uint256 startTime;
        uint256 endTime;
        // 可以根据需要继续添加其他查询条件字段，比如按消息内容、公钥等查询
    }

    // 添加公告的函数
    // function addAnouncement(uint256 id, Anouncement memory anouncement) public
returns (bool) {
    //     if (announcements[id].id == 0) {
    //         announcements[id] = anouncement;
    //         return true;
    //     }
    //     return false;
    // }



    function addAnouncement(string memory userID,string memory userPK,string
memory nounce,
    string memory message,string memory cipher,string memory attachment,uint256
signtime) public returns (bool) {
        Anouncement memory anouncement =
Anouncement(nextId,userID,userPK,nounce,message,cipher,attachment,signtime);
        announcements[nextId] = anouncement;
        nextId++;
        return true;
    }

    // 根据公告ID获取公告信息的函数
```

```
    function getAnouncement(uint256 id) public view returns (Anouncement memory)
{
        return announcements[id];
    }


    // 根据查询条件、每页显示数量和起始位置返回公告列表的函数
    function listAnouncement(bool byUserID,string memory userID,bool
byTimeRange,uint256 startTime,uint256 endTime,uint256 pageSize, uint256
startIndex) public view returns (Anouncement[] memory) {
        uint256[] memory ids = new uint256[](120);
        // uint256[] ids;

        uint256 foundCount = 0;
        uint256 totalMatched = 0;
        uint256 nextline = 0;

        // 遍历所有公告
        for (uint256 i = 1; i < nextId; i++) {
            // Anouncement storage ann = announcements[i];
            Anouncement memory ann = announcements[i];

            bool isMatch  = true;


            // 检查是否满足查询条件
            // if (byId && ann.id!= id) {
            //     isMatch  = false;
            // }

            if (byUserID && keccak256(abi.encodePacked(ann.userID))!=
keccak256(abi.encodePacked(userID))) {
                isMatch  = false;
            }

            if (byTimeRange && (ann.signtime < startTime || ann.signtime >
endTime)) {
                isMatch  = false;
            }

            // 如果满足查询条件
            if (isMatch) {
                totalMatched++;

                // 判断是否在当前页范围内
                if (totalMatched > startIndex && foundCount < pageSize) {
                    // ids.push(i);
                    ids[nextline] = i;
                    nextline ++;
                    foundCount++;
                }
            }
        }

        // 创建一个数组来存储最终要返回的公告列表
        Anouncement[] memory result = new Anouncement[] (foundCount);

        // 将满足条件的公告添加到结果数组中
```

```
        for (uint256 j = 0; j < foundCount; j++) {
            result[j] = announcements[ids[j]];
        }

        return result;
    }
}
```



（1）这里的abi要将其中的 " 替换成 \",可以打开一个记事本,放入abi,使用快捷键ctrl+f, 全部替换。

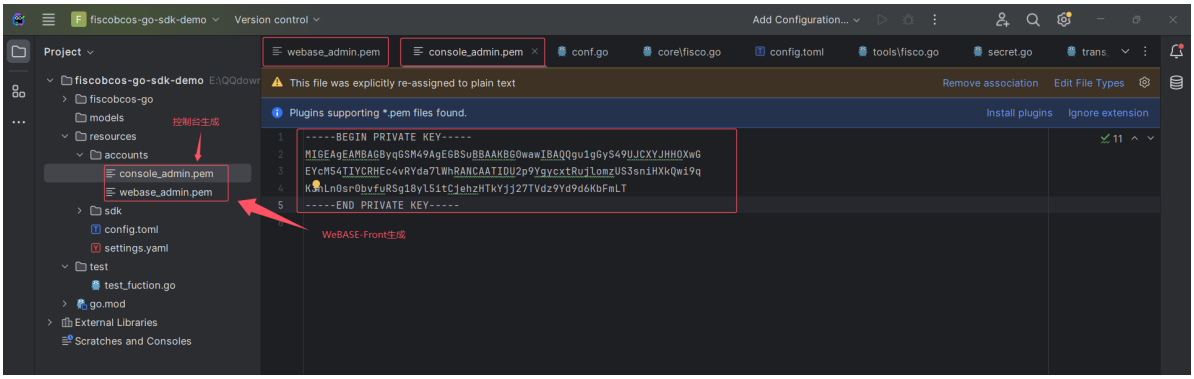


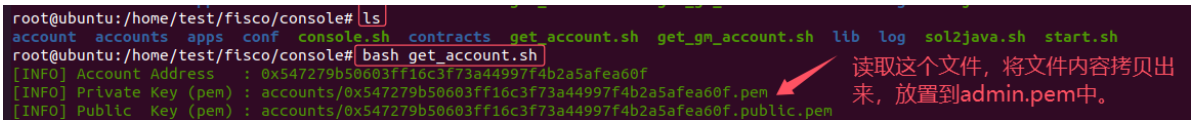
（2）也可以不用替换，直接将abi赋值给whole.Config.Fisco.Abi["contract1"]。(这里的"contract1"是一个map的键与settings配置文件中的保持一致) 换做其他的合约换成对应的 键 就好。

3.连接FISCOBCOS的网络配置文件读取：

配置fiscobcos链的连接：

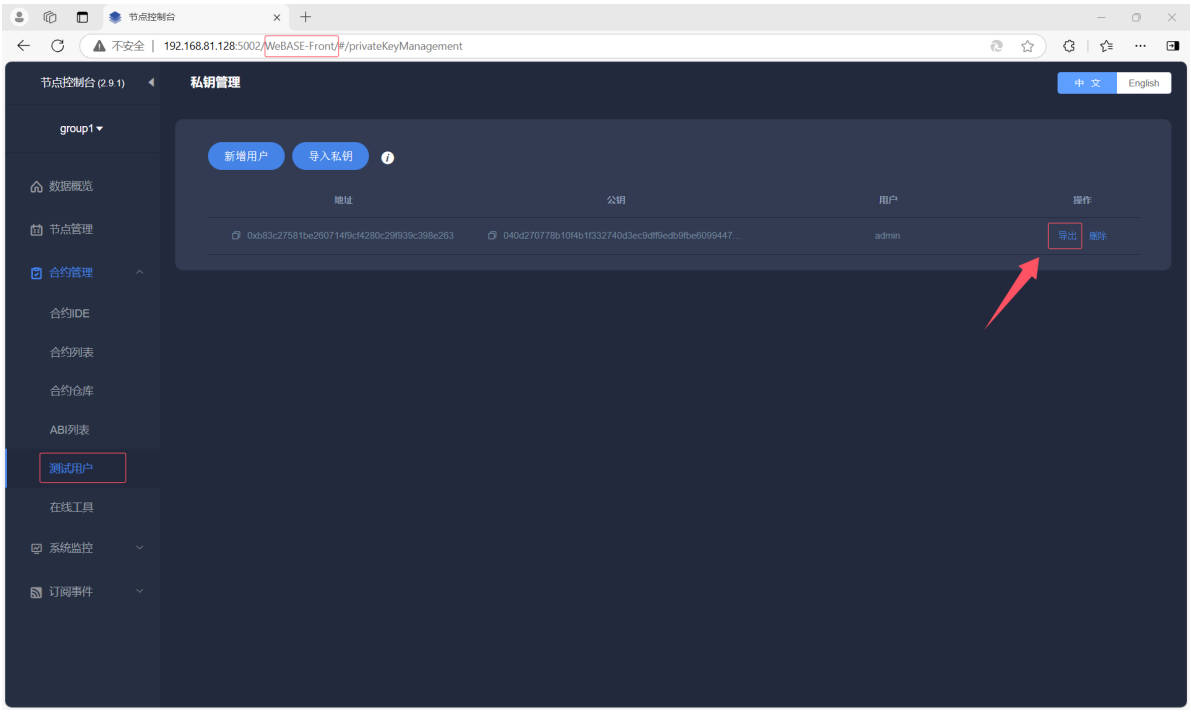
生成私钥：

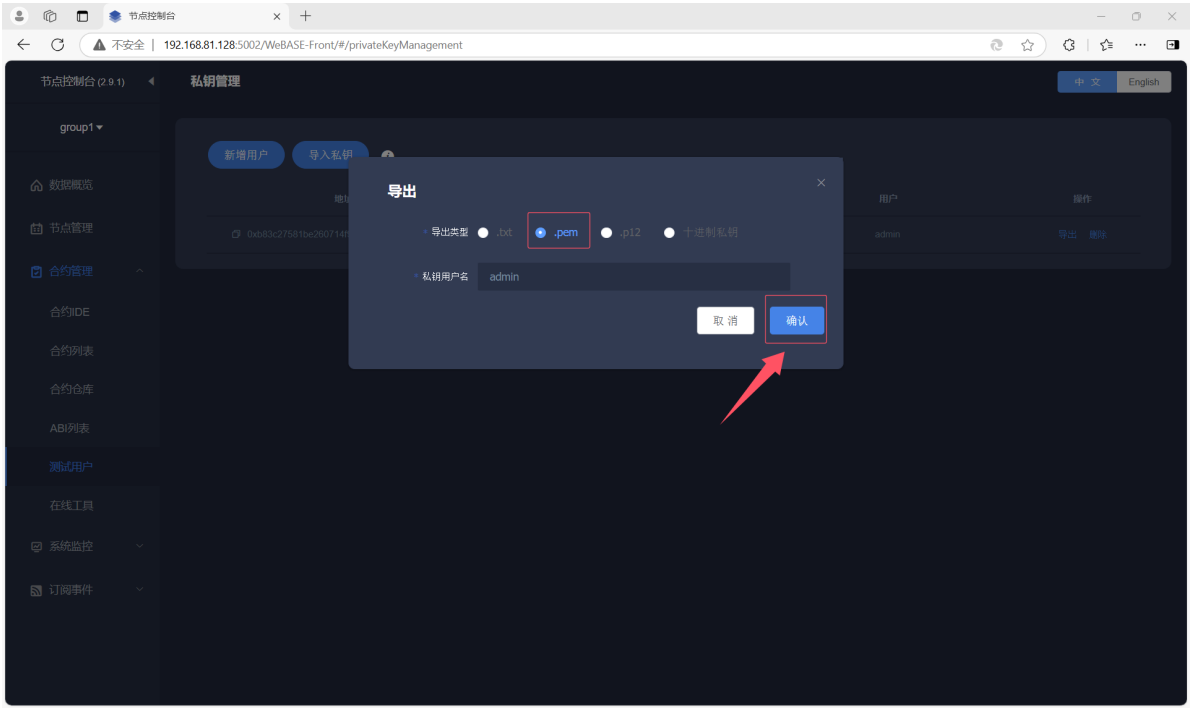




或者从WeBASE-Front上，拿取，下载到项目目录的（fisco-go-sdk-demo/fiscobcos/accounts）中：

下载完之后，记得将私钥文件改个名字（与config.toml中的配置项保持一致）。



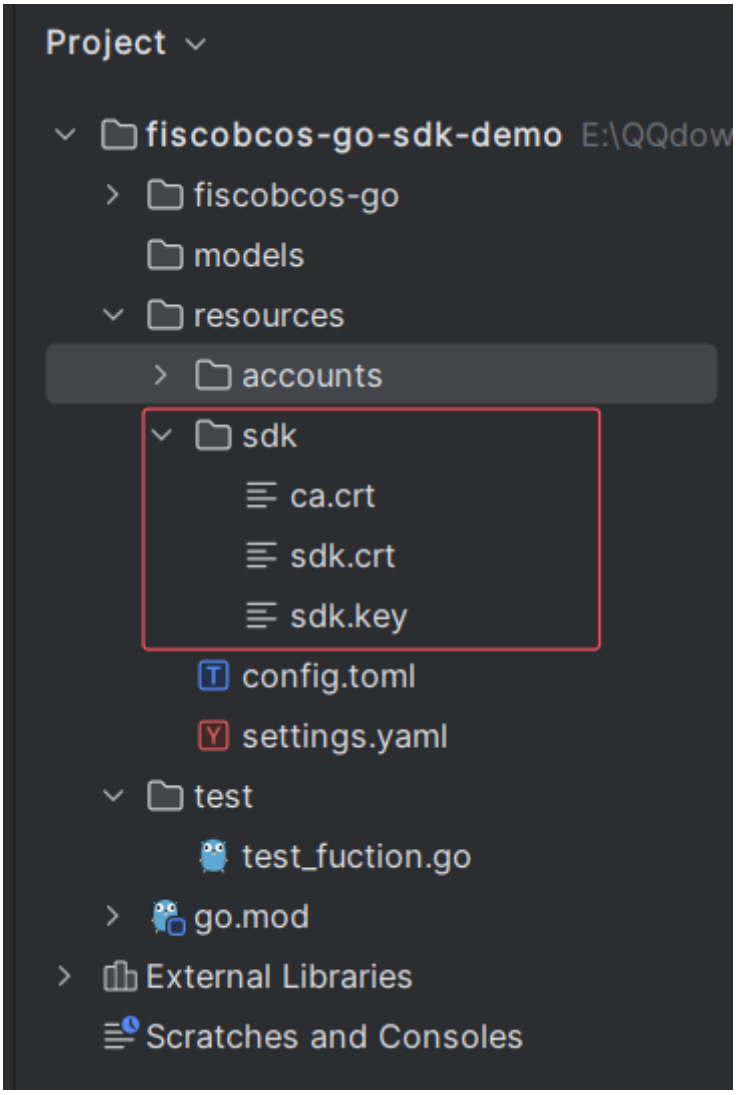


注意：

这里导出后下载到自己的项目目录fisco-go-sdk-demo/fiscobcos/accounts）中。

记得改文件名称（与config.toml中的配置项保持一致）。

拷贝节点证书文件：



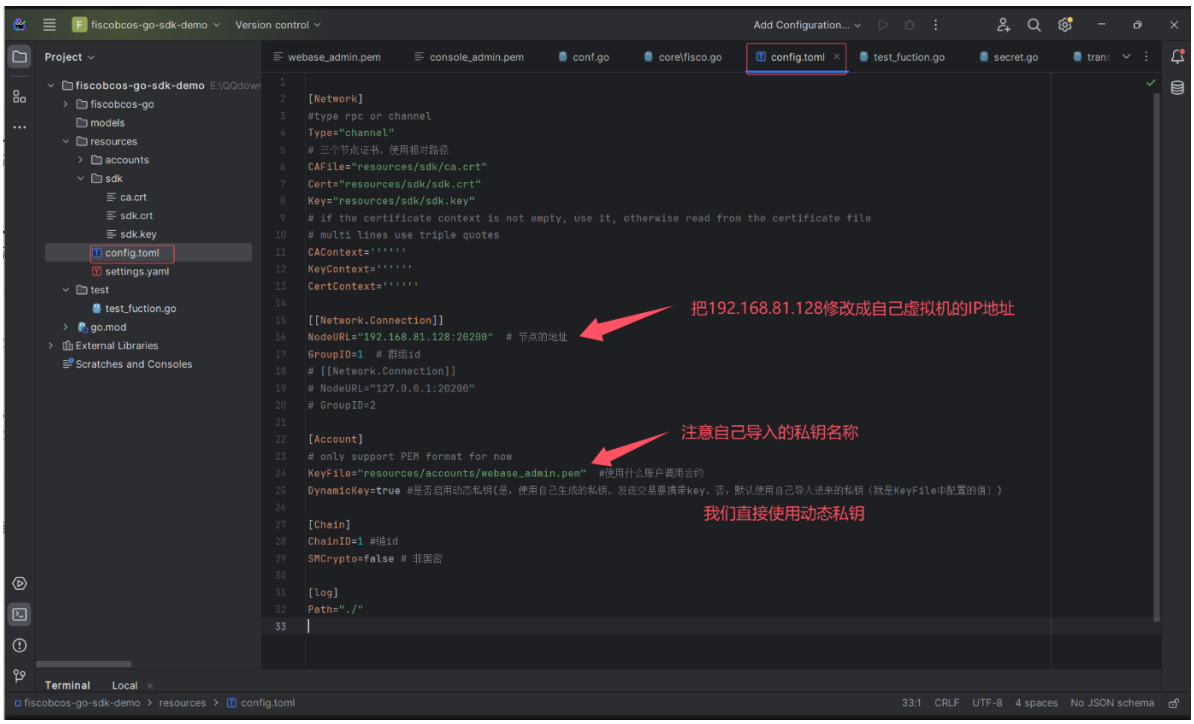
例如：

```
root@ubuntu:/home/test/fisco# ls nodes/127.0.0.1/sdk
ca.crt  sdk.crt  sdk.key
```

网络连接配置文件config.toml（有要修改的部分，认真看图）：

```
[Network]
#type rpc or channel
Type="channel"
# 三个节点证书，使用相对路径
CAFile="resources/sdk/ca.crt"
Cert="resources/sdk/sdk.crt"
Key="resources/sdk/sdk.key"
# if the certificate context is not empty, use it, otherwise read from the
certificate file
# multi lines use triple quotes
CAContext='''''''
KeyContext='''''''
CertContext='''''''

[[Network.Connection]]
NodeURL="192.168.81.128:20200"  # 节点的地址
GroupID=1  # 群组id
# [[Network.Connection]]
# NodeURL="127.0.0.1:20200"
# GroupID=2
```

```
[Account]
# only support PEM format for now
KeyFile="resources/accounts/webase_admin.pem"  #使用什么账户调用合约
DynamicKey=true #是否启用动态私钥(是：使用自己生成的私钥，发送交易要携带key。否：默认使用自己导入进来的私钥（就是KeyFile中配置的值）)

[Chain]
ChainID=1 #链id
SMCrypto=false # 非国密

[log]
Path="./"
```



注意事项：

启用动态私钥的话，只能使用携带私钥的发送交易的函数，如：（SendTransactionByKey，SendCallByKey）。

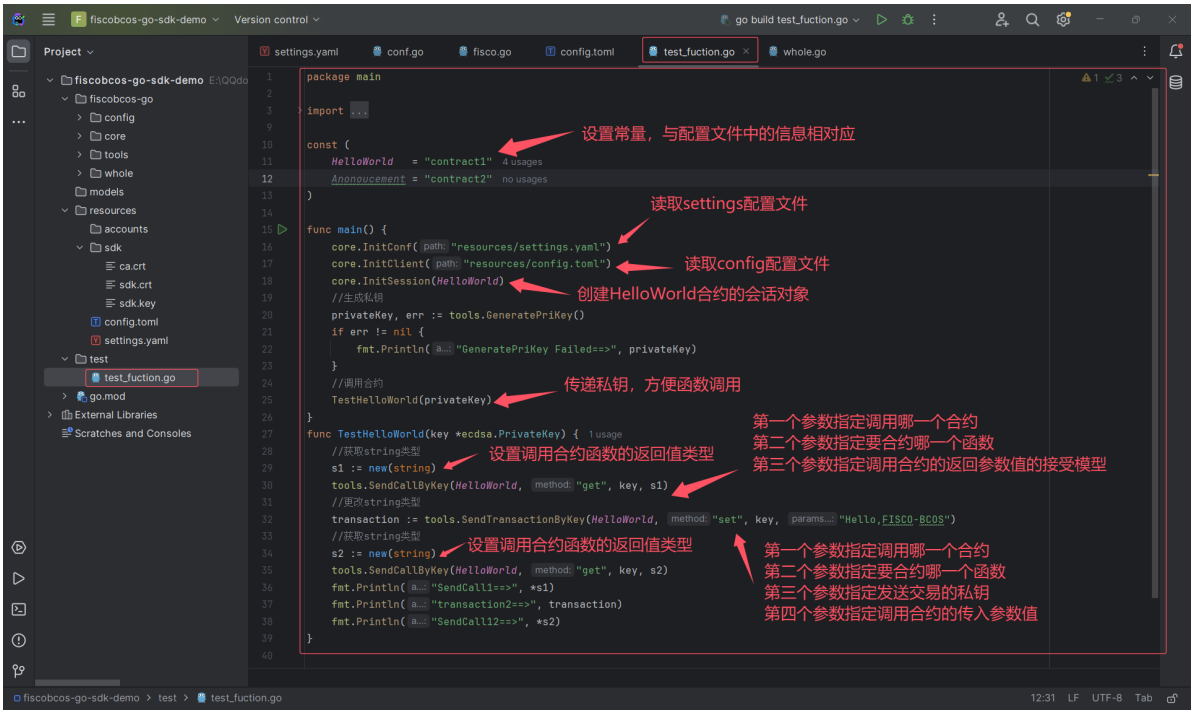
不启用动态私钥的话，只能使用不携带私钥的发送交易的函数，如：（SendTransaction，SendCall）。

4.HelloWorld的调用（通过crypto/ecdsa包下的函数生成私钥，发送交易。）

在main函数中调用WeBASE-Front上的合约：

```
package main

import (
    "crypto/ecdsa"
    "fiscobcos-go-sdk-demo/fiscobcos-go/core"
    "fiscobcos-go-sdk-demo/fiscobcos-go/tools"
    "fmt"
)
```

```go
const (
    HelloWorld = "contract1"
)

func main() {
    core.InitConf("resources/settings.yaml")
    core.InitClient("resources/config.toml")
    core.InitSession(HelloWorld)

    //生成私钥
    privateKey, err := tools.GeneratePriKey()
    if err != nil {
        fmt.Println("GeneratePriKey Failed==>", privateKey)
    }
    //调用合约
    TestHelloWorld(privateKey)
}

func TestHelloWorld(key *ecdsa.PrivateKey) {
    //获取string类型
    s1 := new(string)
    tools.SendCallByKey(HelloWorld, "get", key, s1)

    //更改string类型
    transaction := tools.SendTransactionByKey(HelloWorld, "set", key,
"Hello,FISCO-BCOS")

    //获取string类型
    s2 := new(string)
    tools.SendCallByKey(HelloWorld, "get", key, s2)
    fmt.Println("SendCall1==>", *s1)
    fmt.Println("transaction2==>", transaction)
    fmt.Println("SendCall12==>", *s2)
}
```



启动项目：

注意事项：

（1）启动程序时，如果报错了，执行go mod tidy。

（2）如果还是报错，根据报错信息拉取对应的工具包。

例如：

调用成功：

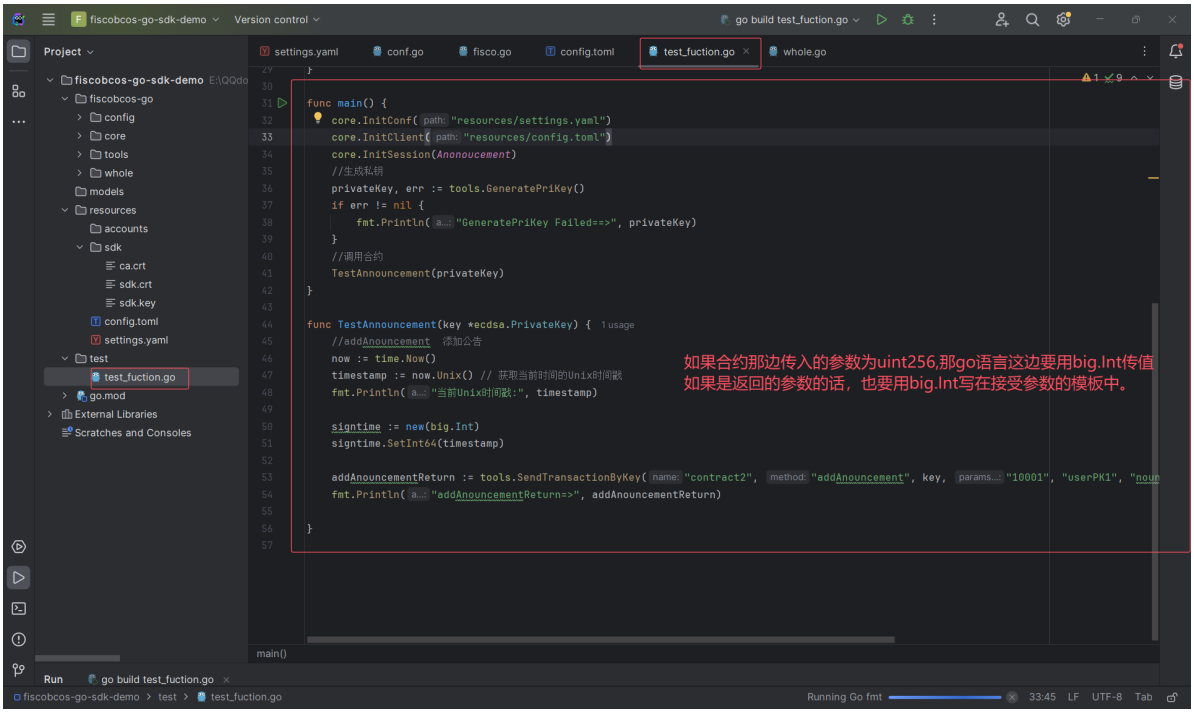
5.Anonoucement的调用（通过crypto/ecdsa包下的函数生成私钥，发送交易。）

addAnouncement（添加公告）功能测试：

```
package main

import (
    "crypto/ecdsa"
    "fiscobcos-go-sdk-demo/fiscobcos-go/core"
    "fiscobcos-go-sdk-demo/fiscobcos-go/tools"
    "fmt"
    "math/big"
    "time"
)

const (
    HelloWorld   = "contract1"
    Anonoucement = "contract2"
)

func main() {
    core.InitConf("resources/settings.yaml")
```

```
    core.InitClient("resources/config.toml")
    core.InitSession(Anonoucement)
    //生成私钥
    privateKey, err := tools.GeneratePriKey()
    if err != nil {
        fmt.Println("GeneratePriKey Failed==>", privateKey)
    }
    //调用合约
    TestAnnouncement(privateKey)
}

func TestAnnouncement(key *ecdsa.PrivateKey) {
    //addAnouncement  添加公告
    now := time.Now()
    timestamp := now.Unix() // 获取当前时间的Unix时间戳
    fmt.Println("当前Unix时间戳:", timestamp)

    signtime := new(big.Int)
    signtime.SetInt64(timestamp)

    addAnouncementReturn := tools.SendTransactionByKey(Anonoucement,
"addAnouncement", key, "10001", "userPK1", "nounce1", "message1", "cipher1",
"attachment1", signtime)
    fmt.Println("addAnouncementReturn=>", addAnouncementReturn)

}
```

下图中的TestAnnouncement中的"contract2"可替换为Anonoucement



执行结果：

announcements功能测试：

（1）创建接受数据的结构体:

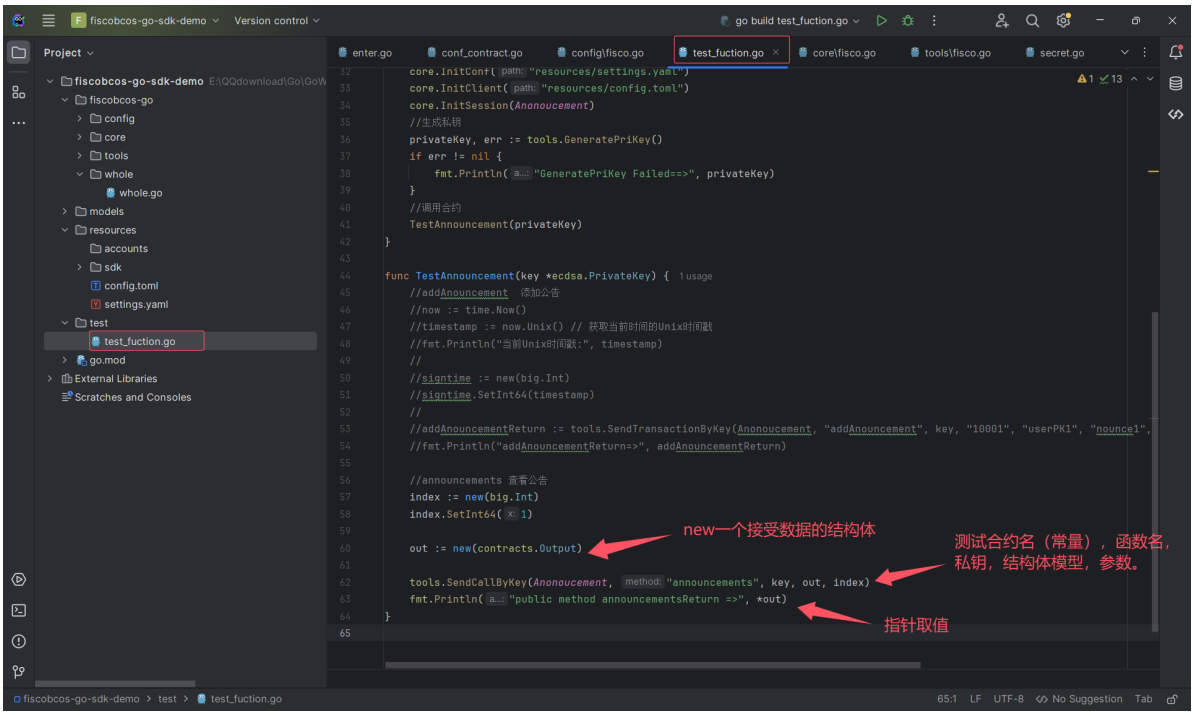
```
package contract

import "math/big"

type Output struct {
    Id         *big.Int
    UserID     string
    UserPK     string
    Nounce     string
    Message    string
    Cipher     string
    Attachment string
    Signtime   *big.Int
}
```

路径：fiscobcos-go-sdk-demo/models/contracts/anonoucement.go

(2)运行的代码

```
func TestAnnouncement(key *ecdsa.PrivateKey) {
    //addAnouncement  添加公告
    //now := time.Now()
    //timestamp := now.Unix() // 获取当前时间的Unix时间戳
    //fmt.Println("当前Unix时间戳:", timestamp)
    //
    //signtime := new(big.Int)
    //signtime.SetInt64(timestamp)
    //
    //addAnouncementReturn := tools.SendTransactionByKey(Anonoucement,
"addAnouncement", key, "10001", "userPK1", "nounce1", "message1", "cipher1",
"attachment1", signtime)
    //fmt.Println("addAnouncementReturn=>", addAnouncementReturn)

    //announcements 查看公告
    index := new(big.Int)
    index.SetInt64(1)

    out := new(contracts.Output)
```

```
    tools.SendCallByKey(Anonoucement, "announcements", key, out, index)
    fmt.Println("public method announcementsReturn =>", *out)
}
```

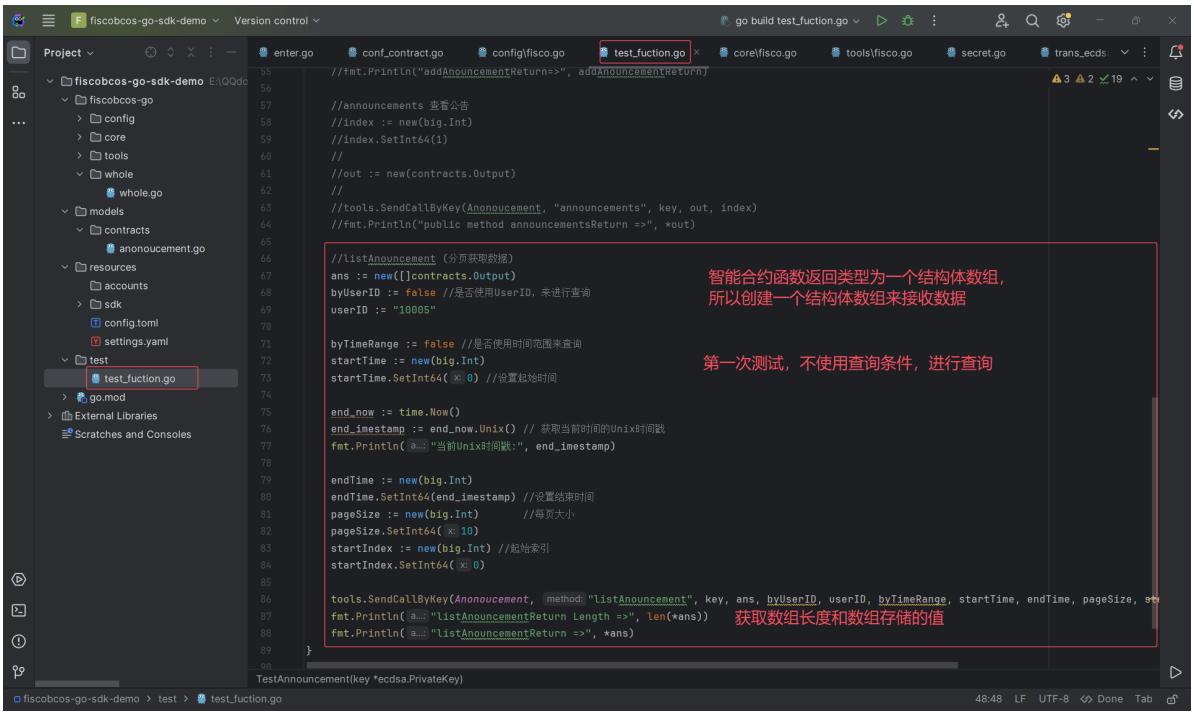


listAnouncement功能测试（在进行测试时，测试人员使用addAnouncement函数，上传了多个公告）：

（1）不用查询条件进行查询（返回所有数据）：

```
func TestAnnouncement(key *ecdsa.PrivateKey) {
    //addAnouncement  添加公告
    //now := time.Now()
    //timestamp := now.Unix() // 获取当前时间的Unix时间戳
    //fmt.Println("当前Unix时间戳:", timestamp)
    //
    //signtime := new(big.Int)
    //signtime.SetInt64(timestamp)
    //
    //addAnouncementReturn := tools.SendTransactionByKey(Anonoucement,
"addAnouncement", key, "10001", "userPK1", "nounce1", "message1", "cipher1",
"attachment1", signtime)
    //fmt.Println("addAnouncementReturn=>", addAnouncementReturn)

    //announcements 查看公告
    //index := new(big.Int)
```

```
    //index.SetInt64(1)
    //
    //out := new(contracts.Output)
    //
    //tools.SendCallByKey(Anonoucement, "announcements", key, out, index)
    //fmt.Println("public method announcementsReturn =>", *out)

    //listAnouncement (分页获取数据)
    ans := new([]contracts.Output)
    byUserID := false //是否使用UserID，来进行查询
    userID := "10005"

    byTimeRange := false //是否使用时间范围来查询
    startTime := new(big.Int)
    startTime.SetInt64(0) //设置起始时间

    end_now := time.Now()
    end_imestamp := end_now.Unix() // 获取当前时间的Unix时间戳
    fmt.Println("当前Unix时间戳:", end_imestamp)

    endTime := new(big.Int)
    endTime.SetInt64(end_imestamp) //设置结束时间
    pageSize := new(big.Int)       //每页大小
    pageSize.SetInt64(10)
    startIndex := new(big.Int) //起始索引
    startIndex.SetInt64(0)

    tools.SendCallByKey(Anonoucement, "listAnouncement", key, ans, byUserID,
userID, byTimeRange, startTime, endTime, pageSize, startIndex)
    fmt.Println("listAnouncementReturn Length =>", len(*ans))
    fmt.Println("listAnouncementReturn =>", *ans)
}
```



执行结果：

（2）使用UserID进行查询：

注意事项：将byUserID的值赋值为true

（3）使用startTime,endTime进行查询：

注意事项：

将byTimeRange的值赋值为true，将byUserID的值赋值为false

endTime（终止时间）的设置要注意

（4）使用UserID和startTime,endTime进行查询：

注意事项：

（1）将byTimeRange和byUserID的值赋值为true

（2）会根据UserID和startTime,endTime进行查询，把所有符合条件的值返回

6.通过WeBASE导出的私钥发送交易

注意事项：

代码中不展示具体的调用函数过程，但会通过私钥计算出公钥和地址，与webase上导出的文件作对比

具体函数调用过程，只需要通过将各类型的私钥，转换成*ecdsa.PrivateKey类型的私钥，就可以调用SendCallByKey和SendTransactionByKey(将私钥作为参数传入就行)。

详细内容见FISCOBCOS中的GoSDK操作---(搭建项目版本)，注意由于两个文档的目录结构不同，要注意函数在哪个包下。

三，小结：

1.发送交易的函数：

（1）在不使用动态私钥的情况：

```
SendTransaction  (作用：往链上存储信息，注意：合约函数的返回值最好设定为bool值。)
SendCall    (作用：获取链上信息,注意：合约函数的传入参数和返回值需要定义模型 )
```

（2）在使用动态私钥的情况：

```
SendTransaction (需要传入私钥) (作用：往链上存储信息，注意：被调用合约函数的返回值最好设定为bool值。)
SendCall  (需要传入私钥) (作用：获取链上信息,注意：合约函数的传入参数和返回值需要定义模型。 )
```

2.合约类型和go语言中的类型对应

合约中传入和返回参数	Go语言中使用对应类型
string	string或者*string
uint / uint256	*big.Int
uint8	*uint8
address	*common.address(用这个包：github.com/ethereum/go-ethereum/common)
bool	*bool
uint[] / uint256[]	*[]big.Int
string[]	*[]string
结构体（例如：User）	*User
结构体数组（例如：User[]）	*[]User

注意事项：

go语言中声明传入、传出的参数的模型时，可以直接：new一个类型 赋值给 变量

例如：num:= new(big.Int)