矩阵元区块链

智能合约开发指南

文档版本号：	1.0.0	文档编号：	
文档密级：		归属部门/项目：	
产品名：	Juzix_BlockChain	子系统名：	
编写人：		编写日期：	

修订记录：

版本号	修订人	修订日期	修订内容
V1.0.0	区块链项目组	2017.7.5	矩阵元区块链智能合约开发指南

所有权声明

目录

1. 适用范围 4
2. 术语解释 4
3. Quick Start 4
3.1. 合约功能描述 4
3.2. 合约规划与列表说明 4
3.2.1. 合约存储规划 4
3.2.2. 合约接口规划 5
3.2.3. 合约权限规划 5
3.2.4. 合约文件规划 5
3.3. 合约代码样例 5
3.4. 合约编译发布 11
3.4.1. 前提要求 11
3.4.2. 合约编译 12
3.4.3. 合约发布 **错误！未定义书签。**
3.4.4. 合约测试 12
4. 系统合约接口说明 13
4.1. Library 库接口说明 13
4.1.1. LibString.sol 13
4.1.2. LibInt.sol 17
4.1.3. LibJson.sol 19
4.2. 系统合约接口说明 22
4.2.1. OwnerNamed.sol 22
4.2.2. RegisterManager.sol 24
4.2.3. UserManager.sol 25
4.2.4. RoleManager.sol 30
4.2.5. DepartmentManager.sol 33
4.2.6. ActionManager.sol 37
4.3. 系统角色与权限说明 43
5.合约业务开发流程介绍 43
5.1 完成合约基础数据的处理 44
5.1.1 string 型数据的截取： 44
5.1.2 int 型数据到 string 型数据的转换： 44
5.2 实现注册合约 45
5.3 定义业务数据结构 46
5.3.1 组织合约，组织的数据结构： 46
5.3.2 角色合约，角色的数据结构： 47
5.3.3 用户合约，用户的数据结构： 47
5.3.4 权限合约，权限的数据结构： 48
5.4 根据接口文档实现业务合约的业务逻辑 48
5.5 模块化管理 49

1. 适用范围

本规范描述了矩阵元区块链系统智能合约的开发约束与规范，用以指导 DAPP 开发者按照本规范开发基于矩阵元区块链运行的应用。

2. 术语解释

术语	术语解释
DAPP	去中心化应用
Truffle	智能合约开发 IDE

3. Quick Start

本章节描述一个简单的智能合约开发样例，用以描述基于矩阵元区块链的智能合约开发标准与规范。给 DAPP 应用的开发提供参考。

3.1. 合约功能描述

此合约开发用例用来管理学生数据，并给不同用户分配不同的数据处理权限，实现数据的访问控制。

3.2. 合约规划与列表说明

3.2.1. 合约存储规划

学生数据在合约的存储属性为：

id	学生 Id（对应于钱包地址）
name	学生名字
classId	学生所属班级 id
status	标记学生数据是否开除： 0 未开除 1 已开除

3.2.2. 合约接口规划

```
//新增学生
func addStudent(string _studentJson) public returns{};

//开除学生
func deleteStudentById0(string id) public{};

//根据 Id 查询用户
func findStudentById(string id) constant public returns{};
```

3.2.3. 合约权限规划

角色列表	角色具有的权限
校长	deleteStudentById, addStudent, findStudentById
班主任	addStudent, findStudentById
学生	findStudentById

3.2.4. 合约文件规划

- LibStudent.sol: 学生的数据结构合约；
- StudentManager.sol: 实现对学生数据的各种处理；
- Module.sol: 实现对合约、角色、权限的模块化处理。

3.3. 合约代码样例

- LibStudent.sol:

```
pragma solidity ^0.4.2;

import "LibInt.sol";
import "LibString.sol";
import  "LibJson.sol";

library LibStudent {
   using LibJson for *;
```

```
using LibString for *;
using LibInt for *;
using LibStudent for *; //引入库的语法
struct Student{
    address     id;                 //学生 Id
    string      name;               //学生名称
    string      classId;            //学生班级 Id
    uint        status;             //学生状态 0 未开除 1 开除
} //student 对象结构体

 //将 student 对象转换成 json 字符串
function toJson(Student storage _self) internal returns(string _strjson){
    _strjson = "{";
    string memory strAddr = "0x";
    strAddr = strAddr.concat(_self.id.addrToAsciiString());
    _strjson = _strjson.concat(strAddr.toKeyValue("id"), ",")
    _strjson = _strjson.concat(_self.name.toKeyValue("name"), ",");
    _strjson = _strjson.concat(_self.classId.toKeyValue("classId"), ",");
    _strjson = _strjson.concat(uint(_self.status).toKeyValue("status"), " ");
    _strjson = _strjson.concat("}");
 }

 //从 student 的 json 字符串中提取相应值
function fromJson(Student storage _self, string _stuJson) internal returns(bool) {
     _self.clear();

    string memory strAddr = _stuJson.getStringValueByKey("id");
    strAddr = _stuJson.getStringValueByKey("id");
    _self.id= strAddr.toAddress();
    _self.name = _stuJson.jsonRead("name");
    _self.classId= _stuJson.jsonRead("classId");
    _self.status = _stuJson.jsonRead("status").toUint();
    return true;
 }

 //将 student 对象置空
function clear(Student storage _self) internal {
    _self.id= address(0);
    _self.name = "";
    _self.classId= "";
    _self.status = 1;
}
```

```
    //重置 student 对象
    function reset(Student storage _self) internal {
      clear(_self);
      _self.clear();
    }
}
```

StudentManager.sol:

```
pragma solidity ^0.4.2;

import "LibStudent.sol";
import "LibString.sol";
import "LibInt.sol";

contract StudentManager {

   using LibStudent for *;
   using LibString for *;
   using LibInt for *;  //引入所需库合约

   event Notify(uint _errno, string _info); //定义事件处理

    //此 mapping 对象将地址映射到对象本身
   mapping(address=>LibStudent.Student) studentMap;
   address[]                            addrList;
   address[]                            tempList;

   LibStudent.Student internal          student;

   string[]                             tmpArray;

   enum StudentError {
      NO_ERROR,
      BAD_PARAMETER,
      NAME_EMPTY,
      CLASS_NOT_EXISTS,
      STUDENT_NOT_EXISTS
   }//错误码定义

   uint errno = 0;

   function StudentManager() {
   }//构造函数
```

```
    //新增学生
    function addStudent(string _studentJson) public returns(uint) {
        log("insert", "StudentManager");

        if (student.fromJson(_studentJson) == false) {
            log("insert bad json", "StudentManager");
            errno = 9000 + uint(StudentError.BAD_PARAMETER);
            Notify(errno, "insert bad json");
            return errno;
        }

        if (student.name.equals("")) {
            log("student name is invalid", "StudentManager");
            errno = 9000 + uint(StudentError.NAME_EMPTY);
            Notify(errno, "student name is invalid");
           return errno;
        }

        if (student.classId.equals("")) {
            log("student classId is invalid", "StudentManager");
            errno = 9000 + uint(StudentError.CLASS_NOT_EXISTS);
            Notify(errno, "student classId is invalid");
           return errno;
        }

        student.status = uint(0);

        studentMap[student.id] = student;
        addrList.push(student.id);

        errno = uint(StudentError.NO_ERROR);

        log("add a student succcess", "StudentManager");
        Notify(errno, "insert a student succcess");
        student.reset();
        return errno;
    }
    //根据学生 Id 查找学生
    function findStudentById(address _id) constant public returns(string _ret) {
        _ret = "{\"ret\":0,\"data\":{";

        if (studentMap[_id].status != 1) {
```

```
            _ret = _ret.concat(studentMap[_id].toJson());
        } else {
            _ret = _ret.concat(""));
        }
        _ret = _ret.concat("}}");
    }
    //根据学生 Id 删除学生
    function deleteStudentById(address _id) public {

        if (studentMap[_id].status == 1) {
            log("student not exists: ", _id);
            errno = 9000 + uint(StudentError.STUDENT_NOT_EXISTS);
            Notify(errno, "student not exists");
            return;
        }
        delete tempList;
        for (uint i = 0; i < addrList.length; i++)  {
            if(_id == studentMap[addrList[i]].id) {
                continue;
            }
            else {
                tempList.push(addrList[i]);
            }
        }

        delete addrList;
        for (uint j = 0; j < tempList.length; ++j) {
            addrList.push(tempList[j]);
        }

        studentMap[_id].status = 1;
        Notify(errno, "fire student success");
    }
}
```

ModuleManager.sol:

```
pragma solidity ^0.4.2;

import "LibModule.sol";
import "LibContract.sol";
import “BaseModule.sol”;

contract ModuleManager is BaseModule{
```

```
    //构造 module 对象
    LibModule.Module     tmpModule;

    //构造 contract 对象
    LibContract.Contract tmpContract;

    //构造函数，发布入链即自动执行
    func ModuleManager(){

        tmpModule = LibModule.Module({moduleId:"systemModule001",moduleName:"系统模块
",moduleVersion:"0.0.1.0",deleted:false,moduleEnable:1,moduleDescription:"系统模块管理
",moduleCreateTime:nowTime,moduleUpdateTime:nowTime,moduleCreator:msg.sender,contractIdLi
st:tmpArr01,roleIds:tmpArr01});
       addModule(tmpModule);          //添加此模块
       initContractData();            //添加合约
       initActionData();              //添加权限
       initRoleData();                //添加角色

   }

    //添加合约数据
   func initContractData() private returns(uint){
       address userAddr = rm.getContractAddress("UserManager","0.0.1.0");
       tmpContract.moduleId = sysModuleId;
       tmpContract.cctId = innerContractMapping["UserManager"];
       tmpContract.cctName = "用户管理";
       tmpContract.cctVersion = "0.0.1.0";
       tmpContract.deleted = false;
       tmpContract.enable = 1;
       tmpContract.description = "用户管理合约";
       tmpContract.createTime = nowTime;
       tmpContract.updateTime = nowTime;
       tmpContract.creator = msg.sender;
       tmpContract.cctAddr = userAddr;
       addContract(tmpContract.toJson());
       return 1;
   }

    //添加权限数据
   func initActionData() private returns(uint){
       string memory actionStr = "";
```

```
        //权限数据对象
        jsonStr="{\"id\":\"action1000\",\"moduleId\":\"module001\",\"name\":\"addStudent\",
        \"resKey\":\"StudentManager\",\"opKey\":\"addStudent(string)\"}";
        addAction(jsonStr);

    jsonStr="{\"id\":\"action1001\",\"moduleId\":\"module001\",\"name\":\"findStudentById
\",\"resKey\":\"StudentManager\",\"opKey\":\"findStudentById(string)\"}";
        addAction(jsonStr);

    jsonStr="{\"id\":\"action1002\",\"moduleId\":\"module001\",\"name\":\"deleteStudentBy
Id\",\"resKey\":\"StudentManager\",\"opKey\":\"deleteStudentById(string)\"}";
        addAction(jsonStr);
        return 1;
    }

    //添加角色数据
    func initRoleData() private returns(uint){
        string memory roleJsonStr="";

        //角色数据对象
        roleJsonStr="{\"id\":\"role1000\",\"name\":\"校长
\",\"status\":1,\"moduleId\":\"systemModule001\",\"actionIdList\":[\"action1000\",\"actio
n1001\",\"action1002\"]}";
        addRole(roleJsonStr);
        roleJsonStr="{\"id\":\"role1001\",\"name\":\"班主任
\",\"status\":1,\"moduleId\":\"systemModule001\",\"actionIdList\":[\"action1000\",\"actio
n1001\"]}";
        addRole(roleJsonStr);
        roleJsonStr="{\"id\":\"role1002\",\"name\":\"学生
\",\"status\":1,\"moduleId\":\"systemModule001\",\"actionIdList\":[\"action1001\",]}";
        addRole(roleJsonStr);
        return 1;
    }
}
```

3.4. 合约编译发布

3.4.1. 前提要求

- 本地矩阵元区块链平台已经启动运行。
  矩阵元区块链平台搭建请参考《矩阵元区块链环境安装部署》文档；

- 本地已搭建好 Truffle 环境。
  Truffle 编译发布环境搭建请参考《矩阵元区块链 Truffle 环境搭建》文档。

3.4.2. 合约编译与发布

1) Truffle 环境初始化
   - 新建合约目录 DApp
   - 在新建的合约目录 DApp 下执行 truffle init 初始化目录结构，得到如下的目录树

   - 修改 truffle.js 文件中 RPC 地址为平台地址
   - 删除示例合约 ConvertLib.sol，MetaCoin.sol，Migrations.sol
2) 上传应用智能合约
   - 将编写的上述智能合约上传到 truffle init 生成目录结构下的 contract 中
3) 合约编译
   - 在新建的 Dapp 目录下执行命令 truffle compile 完成合约的编译
4) 合约发布
   - 修改部署配置文件：
     在 migrations 目录下修改 migrations/1_initial_migration.js
     注释掉 deployer.deploy(Migrations);

     在 migrations 目录下修改 migrations/2_deploy_contracts.js
     添加自己要发布的合约，例：

```
module.exports = function(deployer) {
    deployer.deploy(LibInt);
    deployer.deploy(LibString);
    deployer.deploy(LibRole);
    deployer.deploy(RoleManager);
};
```

   - 在新建的 Dapp 目录下执行命令 truffle migrate 完成合约的发布。

3.4.3. 合约测试

- 进入 truffle 控台： truffle console
- 实例化待测合约对象，如 StudentManager.sol

```
> var stu = StudentManager.deployed();
undefined
```

或者

```
> var stu = StudentManager.at("0x002b4f09741a896e757f276d8f5f0c24bca870bf");
undefined
```

注：该处此地址为合约发布时输出的对应地址

- 调用合约函数：

```
> stu.addStudent('{"id": "0x006d8f5f0c24bca870bf2b4f09741a896e757f27", "name": "juzix",
    "classId": "SuperClass","status": 0}');
"0x00aa1676c77bb7861ef115652715671880aa1677"
```

```
> stu.findStudentById("0x006d8f5f0c24bca870bf2b4f09741a896e757f27");
'{"id": "0x006d8f5f0c24bca870bf2b4f09741a896e757f27", "name": "juzix", "classId":
"SuperClass","status": 0}'
```

```
> stu.deleteStudentById("0x006d8f5f0c24bca870bf2b4f09741a896e757f27");
"0x5652715671880aa00aa1676c77bb7861ef11ba21"
```

4. 系统合约接口说明

4. 1. Library 库接口说明

4.1.1. LibString.sol

接口原型	function memcpy(uint dest, uint src, uint len)		
函数描述	字符串拷贝		
	参数名称	参数类型	取值说明
输入参数	dest	uint	复制对象
	src	uint	被复制对象
	len	uint	复制长度
返回值	无		uint 型数据复制
接口原型	function compare(string _self, string _str) internal returns (int8 _ret)		
函数描述	字符串比较（区分大小写）		
	参数名称	参数类型	取值说明
输入参数	_self	string	比较的第一个对象
	_str	string	比较的第二个对象
返回值	_ret	Int8	-1 前比后小 0 相等 1 前比后大
接口原型	function compareNoCase(string _self, string _str) internal returns (int8 _ret)		
函数描述	比较两字符串大小（不区分大小写）		
	参数名称	参数类型	取值说明
输入参数	_self	string	比较的第一个对象

	_str	string	比较的第二个对象
返回值	_ret	Int8	-1 前比后小 0 相等 1 前比后大
接口原型	function equals(string _self, string _str) internal returns (bool _ret)		
函数描述	判断两字符串是否相等（区分大小写）		
	参数名称	参数类型	取值说明
输入参数	_self	string	第一个对象
	_str	string	第二个对象
返回值	_ret	bool	True 相等 false 不等
接口原型	function equalsNoCase(string _self, string _str) internal returns (bool _ret)		
函数描述	判断两字符串是否相等（不区分大小写）		
	参数名称	参数类型	取值说明
输入参数	_self	string	第一个对象
	_str	string	第二个对象
返回值	_ret	bool	True 相等 false 不等
接口原型	function substr(string _self, uint _start, uint _len) internal returns (string _ret)		
函数描述	字符串截取		
	参数名称	参数类型	取值说明
输入参数	_self	string	取值对象
	_start	uint	开始长度
	_len	uint	结束长度
返回值	_ret	string	截取后的子字符串
接口原型	function concat(string _self, string _str) internal returns (string _ret)		
函数描述	两字符串拼接		
	参数名称	参数类型	取值说明
输入参数	_self	string	第一个对象
	_str	string	第二个对象
返回值	_ret	string	拼接后的字符串
接口原型	function concat(string _self, string _str1, string _str2) internal returns (string _ret)		
函数描述	三字符串拼接		
	参数名称	参数类型	取值说明
输入参数	_self	string	第一个对象
	_str1	string	第二个对象
	_str2	string	第三个对象
返回值	_ret	string	拼接后的字符串
接口原型	function concat(string _self, string _str1, string _str2, string _str3) internal returns (string _ret)		
函数描述	四字符串拼接		
	参数名称	参数类型	取值说明
输入参数	_self	uint	第一个对象
	_str1	uint	第二个对象
	_str2	uint	第三个对象
	_str3	uint	第四个对象
返回值	_ret	string	拼接后的字符串

接口原型	function trim(string _self) internal returns (string _ret)		
函数描述	首尾删除空白		
	参数名称	参数类型	取值说明
输入参数	_self	string	被操作字符串
返回值	_ret	string	返回的字符串
接口原型	function trim(string _self, string _chars) internal returns (string _ret)		
函数描述	删除具体字符		
	参数名称	参数类型	取值说明
输入参数	_self	string	被操作字符串
	_char	string	被删除的字符
返回值	_ret	string	返回的字符串
接口原型	function indexOf(string _self, string _str) internal returns (int _ret)		
函数描述	在源字符串取具体字符串的索引号		
	参数名称	参数类型	取值说明
输入参数	_self	string	源字符串
	_str	string	待取字符串
返回值	_ret	int	返回的索引号
接口原型	function indexOf(string _self, string _str, uint pos) internal returns (int _ret)		
函数描述	在源字符串取具体字符串的索引号		
	参数名称	参数类型	取值说明
输入参数	_self	string	源字符串
	_str	string	待取字符串
	Pos	uint	待取字符串的指定起始数
返回值	_ret	int	返回的索引号
接口原型	function toInt(string _self) internal returns (int _ret)		
函数描述	字符串型数据转 int 型		
	参数名称	参数类型	取值说明
输入参数	_self	string	源字符串
返回值	_ret	int	返回的 int 型数据
接口原型	function fromHexChar(bytes1 _i) internal returns (int8 _ret)		
函数描述	将 16 进制字节转 int8		
	参数名称	参数类型	取值说明
输入参数	_i	bytes1	待转字节
返回值	_ret	int	返回的 int 数据
接口原型	function toHex(string _self) internal returns (bytes _ret)		
函数描述	字符串转字节		
	参数名称	参数类型	取值说明
输入参数	_self	string	源字符串
返回值	_ret	bytes	返回的字节
接口原型	function toAddress(string _self) internal returns (address _ret)		
函数描述	字符串转地址		
	参数名称	参数类型	取值说明

输入参数	_self	string	源字符串
返回值	_ret	address	返回的地址
接口原型	function toKeyValue(string _self, string _key) internal returns (string _ret)		
函数描述	输出 key：value 型		
	参数名称	参数类型	取值说明
输入参数	_self	string	String 型 value
	_key	string	key
返回值	_ret	string	返回 key+value 如：”key”:”hello”
接口原型	function toKeyValue(string[] storage _self, string _key) internal returns (string _ret)		
函数描述	输出 key:value 型		
	参数名称	参数类型	取值说明
输入参数	_self	string[]	String[]型数据对象
	_key	string	Key
返回值	_ret	string	返回 key+value 如： “key”:[“a”,”b”]
接口原型	function getStringValueByKey(string _self, string _key) internal returns (string _ret)		
函数描述	获得对应的 string 型 value		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象
	_key	string	Key
返回值	_ret	string	返回对应的 value
接口原型	function getIntValueByKey(string _self, string _key) internal returns (int _ret)		
函数描述	获得对应的 int 型 value		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象
	_key	string	Key
返回值	_ret	int	返回对应的 value
接口原型	function getArrayValueByKey(string _self, string _key) internal returns (string _ret)		
函数描述	获得对应的 array 型 value		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象
	_key	string	Key
返回值	_ret	string	返回对应的 value
接口原型	function keyExists(string _self, string _key) internal returns (bool _ret)		
函数描述	判断 key 是否存在		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象
	_key	string	Key
返回值	_ret	bool	False 不存在 true 存在
接口原型	function storageToUint(string _self) internal returns (uint _ret)		
函数描述	String 型数据转 uint		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象

返回值	_ret	uint	返回 uint 型数据
接口原型	function inArray(string _self, string[] storage _array) internal returns (bool _ret)		
函数描述	判断 array 型数据(区分大小写)是否在源数据对象是否存在		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象
	_array	string[]	指定的 array 对象
返回值	_ret	bool	false 不存在 true 存在
接口原型	function inArrayNoCase(string _self, string[] storage _array) internal returns (bool _ret)		
函数描述	判断 array 型数据(不区分大小写)是否在源数据对象是否存在		
	参数名称	参数类型	取值说明
输入参数	_self	string	源数据对象
	_array	string[]	指定的 array 对象
返回值	_ret	bool	false 不存在 true 存在
接口原型	function addrToAsciiString(address x) internal returns (string)		
函数描述	地址型数据转字符串		
	参数名称	参数类型	取值说明
输入参数	X	address	地址型数据
返回值	_ret	string	返回的字符串
接口原型	function toChar(byte b) internal returns (byte c)		
函数描述	字节转字符		
	参数名称	参数类型	取值说明
输入参数	b	byte	源字节
返回值	c	byte	返回的字符
接口原型	function toUpper(string _self) internal returns (string _ret)		
函数描述	字符串变大写		
	参数名称	参数类型	取值说明
输入参数	_self	string	源字符串
返回值	_ret	string	返回的字符串
接口原型	function toLower(string _self) internal returns (string _ret)		
函数描述	字符串小写		
	参数名称	参数类型	取值说明
输入参数	_self	string	源字符串
返回值	_ret	string	返回的字符串

4.1.2. LibInt.sol

接口原型	function toString(uint _self, uint width) internal returns (string _ret)		
函数描述	Uint 型转字符串		
	参数名称	参数类型	取值说明
输入参数	_self	uint	Uint 数据对象

	width	uint	待转换长度
返回值	_ret	string	返回的字符串
接口原型	function toString(uint _self) internal returns (string _ret)		
函数描述	Uint 型转字符串		
	参数名称	参数类型	取值说明
输入参数	_self	uint	Uint 数据对象
返回值	_ret	string	返回的字符串
接口原型	function toHexString(uint _self) internal returns (string _ret)		
函数描述	Uint 转 16 进制字节码		
	参数名称	参数类型	取值说明
输入参数	_self	uint	Uint 数据对象
返回值	_ret	string	返回的 16 进制字节码
接口原型	function toHexString64(uint _self) internal returns (string _ret)		
函数描述	Uint 转”0x”开头的 16 进制字节码		
	参数名称	参数类型	取值说明
输入参数	_self	uint	Uint 数据对象
返回值	_ret	string	返回的以”0x”开头的 16 进制字节码
接口原型	function toString(int _self) internal returns (string _ret)		
函数描述	Int 转字符串		
	参数名称	参数类型	取值说明
	_self	int	int 数据对象
返回值	_ret	string	返回的字符串
接口原型	function toAddrString(uint _self) internal returns (string _ret)		
函数描述	uint 型转地址型		
	参数名称	参数类型	取值说明
输入参数	_self	uint	Uint 数据对象
返回值	_ret	string	返回地址型字符串
接口原型	function toKeyValue(uint _self, string _key) internal returns (string _ret)		
函数描述	返回 key:value 型（value 为 uint）		
	参数名称	参数类型	取值说明
输入参数	_self	uint	Uint 型 value
	_key	string	Key
返回值	_ret	string	返回 key+value 如 “key”:1
接口原型	function toKeyValue(int _self, string _key) internal returns (string _ret)		
函数描述	返回 key:value 型（value 为 int）		
	参数名称	参数类型	取值说明
输入参数	_self	int	int 型 value
	_key	string	key
返回值	_ret	string	返回 key+value 如 ”key”:1
接口原型	function toKeyValue(address _self, string _key) internal returns (string _ret)		
函数描述	返回 key:value 型（value 为 address）		
	参数名称	参数类型	取值说明

输入参数	_self	address	address 型 value
	_key	string	key
返回值	_ret	string	返回 key+value 如 “key”:”0x00ad201d3f88e4991728215”
接口原型	function recoveryToString(uint _self) internal returns (string _ret)		
函数描述	uint 转字符串		
	参数名称	参数类型	取值说明
输入参数	_self	uint	uin 数据对象
返回值	_ret	string	Uint 型转 string 型

4.1.3. LibJson.sol

接口原型	function isJson(string _json) internal constant returns(bool _ret)		
函数描述	判断 string 对象是否 json		
	参数名称	参数类型	取值说明
输入参数	_json	string	json 对象
返回值	_ret	bool	是否 json
接口原型	jsonRead(string _json, string _keyPath) internal constant returns(string _ret)		
函数描述	获得 key 对应的 value		
	参数名称	参数类型	取值说明
输入参数	_json	string	Json 对象
	_keyPath	string	所读取的 key
返回值	_ret	string	Key 对应的 value
接口原型	function jsonKeyExists(string _json, string _keyPath) internal constant returns(bool _ret)		
函数描述	判断 key 是否存在		
	参数名称	参数类型	取值说明
输入参数	_json	string	Json 对象
	_keyPath	string	所读取的 key
返回值	_ret	bool	Key 是否存在

4.1.4. LibPaillier.sol

接口原型	function pai_add(string d1, string d2) internal constant returns (string result)		
函数描述	加法同态计算		
	参数名称	参数类型	取值说明
输入参数	_d1	string	第一个对象
	_d2	string	第二个对象
返回值	result	string	同态相加返回结果

4.1.5. LibContract.sol

合约对象的数据结构：

```
struct Contract{
    string          moduleId;           //合约所在模块 id
    string          contractId;         //合约 id
    string          contractName;       //合约名称
    string          contractVersion;    //合约版本
    bool            deleted;            //合约是否删除
    uint            enable;             //合约是否启用
    string          description;        //合约描述
    uint            createTime;         //合约创建时间
    uint            updateTime;         //合约更新时间
    address         creator;            //合约创建者
    address         contractAddr;       //合约地址
}
```

接口原型	function toJson(Contract storage _self) internal constant returns (string _json)		
函数描述	将 contract 对象转换成 json 字符串		
	参数名称	参数类型	取值说明
输入参数	_self	Contract	合约对象
返回值	_json	string	返回的 json 字符串
接口原型	function fromJson(Contract storage _self, string _json) internal constant returns(bool succ)		
函数描述	从 contract 的 json 字符串中提取相应字段的值		
	参数名称	参数类型	取值说明
输入参数	_self	Contract	合约对象
	_json	string	合约的 json 字符串
返回值	succ	bool	是否取值成功

4.1.6.LibModule.sol

模块对象的数据结构：

```
struct Module{
    string      moduleId;               //模块 id
    string      moduleName;             //模块名称
    string      moduleVersion;          //模块版本
    bool        deleted;                //模块是否删除
    uint        moduleEnable;           //模块开关：0 false 1 true
    string      moduleDescription;      //模块描述
    uint        moduleCreateTime;       //模块创建时间
    uint        moduleUpdateTime;       //模块更新时间
```

```
    address       moduleCreator;            //模块创建者
    string[]      contractIdList;           //模块下的合约列表
    string[]      roleIds;                  //模块下的角色列表
}
```

接口原型	function toJson(Module storage _self) constant internal returns(string _json)		
函数描述	将 Module 对象转换成 json 字符串		
	参数名称	参数类型	取值说明
输入参数	_self	Module	模块对象
返回值	_json	string	返回的 json 字符串
接口原型	function fromJson(Module storage _self, string _json) constant internal returns(bool succ)		
函数描述	从 Module 的 json 字符串中提取相应字段的值		
	参数名称	参数类型	取值说明
输入参数	_self	Module	模块对象
	_json	string	模块的 json 字符串
返回值	succ	bool	是否取值成功

4.1.6.LibNIZK.sol

零知识证明参数的数据结构：

```
struct NizkParam{
    string        cipher1;              //第一个加数/减数（密文）
    string        cipher2;              //第二个加数/减数（密文）
    string        pais;                 //证明的加密密文
    string        balapubcipher;        //账户余额的密文
    string        traapubcipher;        //转出账户需要转出的金额密文
    address       trabpubcipher;        //转入账户需要转入的金额密文
    address       apukkey;              //转出账户的公钥
    address       bpukkey;              //转入账户的公钥
    address       nizkpp;               //零知识证明的全局结构
}
```

接口原型	function nizk_setup() internal constant returns (string)		
函数描述	全局一次调用生成链对应的零知识证明结构		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	无	string	返回的零知识证明结构
接口原型	function nizk_apubcipheradd(LibNizkParam.NizkParam param) internal returns (string)		
函数描述	零知识证明下的同态加法		
	参数名称	参数类型	取值说明
输入参数	param	NizkParam	零知识证明入参结构
返回值	无	string	返回相加后的密文

接口原型	function nizk_apubciphersub(LibNizkParam.NizkParam param) internal returns (string)		
函数描述	零知识证明下的同态减法		
	参数名称	参数类型	取值说明
输入参数	param	NizkParam	零知识证明入参结构
	无	string	返回相减后的密文
接口原型	function nizk_verifyproof(string pais, string balapubcipher, string traapubcipher, string trabpubcipher, string apukkey, string bpukkey, string nizkpp) internal returns (uint)		
函数描述	验证零知识证明		
	参数名称	参数类型	取值说明
输入参数	pais	string	证明的加密密文
	balapubcipher	string	账户余额的密文
	traapubcipher	string	转出账户需要转入的金额密文
	trabpubcipher	string	转入账户需要转入的金额密文
	apukkey	string	转出账户的公钥
	bpukkey	string	转入账户的公钥
	nizkpp	string	零知识证明的全局结构
返回值	无	uint	验证结果,验证通过为 1

4.2. 系统合约接口说明

4.2.1. OwnerNamed.sol

此合主要提供一些日志输出函数，所有业务合约均继承此合约。

接口原型	function register(string _name, string _version) public		
函数描述	合约注册		
	参数名称	参数类型	取值说明
输入参数	__name	string	合约名称
	_version	string	合约版本
返回值	无		合约注册
接口原型	function kill() public		
函数描述	注销合约		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	无		注销合约
接口原型	function getOwner() constant public returns (string _ret)		
函数描述	获得合约地址		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_ret	string	返回合约地址

接口原型	function getSender() constant public returns (string _ret)		
函数描述	获得合约地址		
	参数名称	参数类型	取值说明
输入参数	_无		
返回值	_ret	String	返回合约地址
接口原型	function getErrno() constant returns (uint)		
函数描述	获得错误码		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_ret	uint	返回错误码
接口原型	function log(string _str) constant public returns(uint _ret)		
函数描述	日志输出 1 个字符串语句		
	参数名称	参数类型	取值说明
输入参数	_str	string	日志输出 1 个字符串语句
返回值	_ret	uint	返回 0 如：log(“Okay”)
接口原型	function log(string _str, string _str2) constant public returns(uint _ret)		
函数描述	日志输出 2 个字符串语句		
	参数名称	参数类型	取值说明
输入参数	_str	string	第 1 个字符串语句
	_str2	string	第 2 个字符串语句
返回值	_ret	uint	返回 0 如: log(“okay”,OwnerNamel”)
接口原型	function log(string _str, string _str2, string _str3) constant public returns(uint _ret)		
函数描述	日志输出 3 个字符串语句		
	参数名称	参数类型	取值说明
输入参数	_str	string	第 1 个字符串语句
	_str2	string	第 2 个字符串语句
	_str3	string	第 3 个字符串语句
返回值	_ret	uint	返回 0，输出成功
接口原型	function log(string _str, uint _ui) constant public returns(uint _ret)		
函数描述	日志输出 1 个字符串语句 +1 个 uint 型数据		
	参数名称	参数类型	取值说明
输入参数	_str	string	第 1 个字符串语句
	_ui	uint	输出的 uint 型数据
返回值	_ret	uint	返回 0，输出成功
接口原型	function log(string _str, int _i) constant public returns(uint _ret)		
函数描述	日志输出 1 个字符串语句 +1 个 int 型数据		
	参数名称	参数类型	取值说明
输入参数	_str	string	第 1 个字符串语句
	_i	int	输出的 int 型数据
返回值	_ret	uint	返回 0，输出成功
接口原型	function log(string _str, address _addr) constant public returns(uint _ret)		
函数描述	日志输出 1 个字符串语句 +1 个地址型数据		

	参数名称	参数类型	取值说明
输入参数	_str	string	第 1 个字符串语句
	_addr	address	输出的地址型数据
返回值	_ret	uint	返回 0，输出成功
接口原型	function writedb(string _name, string _key, string _value) public constant returns(uint _ret)		
函数描述	将数据写入底层		
	参数名称	参数类型	取值说明
输入参数	_name	string	写入数据名称
	_key	string	数据 key
	_value	string	数据 value
返回值	_ret	uint	返回 0，写入成功

4.2.2. RegisterManager.sol

此合约提供合约注册接口，所有业务合约均会在此合约注册。

接口原型	function register(string _name, string _version)		
函数描述	合约注册		
	参数名称	参数类型	取值说明
输入参数	_name	string	合约名称
	_version	string	合约版本
返回值	无		合约注册
接口原型	function unRegister()		
函数描述	合约注销		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	无		合约注销
接口原型	function getContractAddress(string _name,string _version) constant returns (address _address)		
函数描述	获取合约地址		
	参数名称	参数类型	取值说明
输入参数	_name	string	合约名称
	_version	string	合约版本
返回值	_address	address	获取合约地址
接口原型	function IfContractRegist(address _contractAddr) constant returns(bool)		
函数描述	判断合约是否注册		
	参数名称	参数类型	取值说明
输入参数	_contractAddr	address	合约地址
返回值	_ret	bool	判断合约是否注册
接口原型	function findResNameByAddress(address _addr) constant public returns(uint _contractName)		
函数描述	返回合约名称		
	参数名称	参数类型	取值说明
输入参数	_addr	address	合约地址

返回值	_contractName	uint	返回合约名称
接口原型	function getRegisteredContract(uint _pageNum, uint _pageSize) constant public returns(string _json)		
函数描述	返回已注册过的合约		
	参数名称	参数类型	取值说明
输入参数	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_json	string	返回已注册过的合约
接口原型	function transferContract(string _fromName, string _fromVersion, string _toName, string _toVersion, string _signString) public returns (uint _errno)		
函数描述	转移合约		
	参数名称	参数类型	取值说明
输入参数	_fromName	string	被转移合约名称
	_fromVersion	string	被转移合约版本
	_toName	string	转移目标合约名称
	_toVersion	string	转移目标合约版本
	_signString	string	签名消息
返回值	_errno	uint	返回错误码

4.2.3. UserManager.sol

接口原型	function getUserState(address _userAddr) constant public returns (uint _state)		
函数描述	查找用户状态		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_state	uint	返回用户状态
接口原型	function getAccountState(string _account) constant public returns (uint _state)		
函数描述	查找账户状态		
	参数名称	参数类型	取值说明
输入参数	_account	string	用户账户
返回值	_state	uint	返回账户状态
接口原型	function pageByAccountStatus(uint _accountStatus, uint _pageNo, uint _pageSize) public constant returns(string _strjson)		
函数描述	根据账户状态分页查询用户数据		
	参数名称	参数类型	取值说明
输入参数	_accountStatus	uint	账户状态
	_pageNo	uint	分页号
	_pageSize	uint	分页大小
返回值	_strJson	string	返回用户信息
接口原型	function findByAddress(address _userAddr) constant public returns(string _ret)		
函数描述	根据用户地址查询用户		

	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_ret	String	返回用户信息
接口原型	function findByLoginName(string _name) constant public returns(string _strjson)		
函数描述	根据登录名查询用户		
	参数名称	参数类型	取值说明
输入参数	_name	string	用户登录名
返回值	_strjson	string	返回用户信息
接口原型	function findByAccount(string _account) constant public returns(string _strjson)		
函数描述	根据账户查询用户		
	参数名称	参数类型	取值说明
输入参数	_account	string	用户账户名
返回值	_strjson	string	返回用户信息
接口原型	function findByMobile(string _mobile) constant public returns(string _strjson)		
函数描述	根据手机查找用户		
	参数名称	参数类型	取值说明
输入参数	_mobile	string	用户手机号
返回值	_strjson	string	返回用户信息
接口原型	function findByEmail(string _email) constant public returns(string _strjson)		
函数描述	根据邮箱查找用户		
	参数名称	参数类型	取值说明
输入参数	_email	string	用户邮箱
返回值	_strjson	string	返回用户信息
接口原型	function log(string _str, uint _ui) constant public returns(uint _ret)		
函数描述	日志输出类型字符串 +1 uint 型数据		
	参数名称	参数类型	取值说明
输入参数	_str	string	输出的字符串
	_ui	uint	输出的 uint 型数据
返回值	_ret	uint	返回 0， 输出成功
接口原型	function findByDepartmentId(string _departmentId) constant public returns(string _strjson)		
函数描述	根据组织 Id 查找用户		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
返回值	_strjson	string	返回用户数据
接口原型	function findByDepartmentIdTree(string _departmentId, uint _pageNum, uint _pageSize) constant public returns(string _strjson)		
函数描述	根据组织树查找用户		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_strjson	string	根据组织树查找用户

接口原型	function findByDepartmentIdTreeAndContion(uint _status,string _name,string _departmentId, uint _pageNum, uint _pageSize) constant public returns(string _strjson)		
函数描述	根据组织树以及查找条件查找用户		
	参数名称	参数类型	取值说明
输入参数	_status	uint	用户状态
	_name	string	用户名称
	_departmentId	string	组织 Id
	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_strjson	string	根据组织树以及查找条件查找用户
接口原型	function findByRoleId(string _roleId) constant public returns(string _strjson)		
函数描述	根据角色 Id 查找用户		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id
返回值	_strjson	string	根据角色 Id 查找用户
接口原型	function getUserDepartmentId(address _userAddr) constant returns(uint _departId)		
函数描述	根据用户地址查找组织		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_departId	uint	根据用户地址查找组织
接口原型	function checkUserRole(address _userAddr, string _roleId) constant public returns(uint _ret)		
函数描述	判断用户是否具备此角色		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_roleId	string	角色 Id
返回值	_ret	uint	判断用户是否具备此角色
接口原型	function checkUserAction(address _userAddr, string _actionId) constant public returns (uint _ret)		
函数描述	判断用户是否具备此权限		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_actionId	string	权限 Id
返回值	_ret	uint	判断用户是否具备此权限
接口原型	function checkUserPrivilege(address _userAddr, address _contractAddr, string _funcSha3) constant public returns (uint _ret)		
函数描述	判断此用户是否可调用此函数		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_contractAddr	address	合约地址
	_funcSha3	string	此合约的函数
返回值	_ret	uint	判断此用户是否可调用此函数
接口原型	function userExists(address _userAddr) constant public returns(uint _ret)		

函数描述	判断此用户是否存在		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_ret	uint	判断此用户是否存在
接口原型	function insert(string _userJson) public returns(uint)		
函数描述	新增用户		
	参数名称	参数类型	取值说明
输入参数	_userJson	string	用户数据对象
返回值	_ret	uint	新增用户
接口原型	function update(string _userJson) public returns(bool _ret)		
函数描述	更新用户信息		
	参数名称	参数类型	取值说明
输入参数	_userJson	string	用户数据对象
返回值	_ret	bool	更新用户信息
接口原型	function updateUserStatus(address _userAddr, uint _status) public returns(bool _ret)		
函数描述	更新用户状态		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_status	uint	用户状态
返回值	_ret	bool	更新用户状态
接口原型	function updateAccountStatus(address _userAddr, uint _status) public returns(bool _ret)		
函数描述	更新账户状态		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_status	uint	账户状态
返回值	_ret	bool	更新账户状态
接口原型	function updatePasswordStatus(address _userAddr, uint _status) public returns(bool _ret)		
函数描述	更新口令状态		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_status	uint	口令状态
返回值	_ret	bool	更新口令状态
接口原型	function addUserRole(address _userAddr, string _roleId) returns(uint)		
函数描述	用户添加角色		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_roleId	string	角色 Id
返回值	_ret	uint	用户添加角色
接口原型	function deleteByAddress(address _userAddr) public		
函数描述	删除用户		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址

返回值	无		删除用户
接口原型	function login(string _account) public returns(string _json)		
函数描述	用户登录		
	参数名称	参数类型	取值说明
输入参数	_account	string	账户名
返回值	_json	string	用户登录
接口原型	function listAll() constant public returns(string _userListJson)		
函数描述	查找所有用户列表		
	参数名称	参数类型	取值说明
输入参数	_无		
返回值	_userListjson	string	查找所有用户列表
接口原型	function getUserCount() constant public returns(uint _count)		
函数描述	获取所有用户数量		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_count	uint	获取所有用户数量
接口原型	function getUserRoleId(address _userAddr, uint _index) constant returns (uint _ret)		
函数描述	获取用户角色		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_index	uint	角色索引号
返回值	_ret	uint	获取用户角色
接口原型	function getRevision() constant public returns(uint _revision)		
函数描述	获取版本号		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_revision	uint	获取版本号
接口原型	function getUserCountByDepartmentId(string _departmentId) constant public returns(uint _count)		
函数描述	查找组织下的用户数目		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
返回值	_count	uint	查找组织下的用户数目
接口原型	function getUserCountByActionId(string _actionId) constant public returns(uint _count)		
函数描述	获取拥有指定权限的用户数量		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
返回值	_count	uint	获取拥有指定权限的用户数量
接口原型	function roleUsed(string _roleId) constant public returns (uint _used)		
函数描述	检查 role 是否被使用		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id

返回值	_used	uint	检查 role 是否被使用
接口原型	function getUserCountMappingByRoleIds(string _roleIds) constant public returns(string _json)		
函数描述	根据角色 ID 获取用户数量		
	参数名称	参数类型	取值说明
输入参数	_roleids	string	角色列表
返回值	_json	string	根据角色 ID 获取用户数量
接口原型	function __getDepartmentAdmin(string departmentId) constant internal returns(address admin)		
函数描述	获取组织管理员		
	参数名称	参数类型	取值说明
输入参数	departmentId	string	组织 Id
返回值	Admin	address	获取组织管理员
接口原型	function __checkWritePermission(address _addr, string _departmentId) constant internal returns (uint _ret)		
函数描述	判断该地址用户在该机构是否具有操作权限		
	参数名称	参数类型	取值说明
输入参数	_addr	string	用户地址
	_departmentId	string	组织 Id
返回值	_ret	uint	0 有 1 无
接口原型	function __getDepartmentIdTree(string _departmentId, string[] storage _departmentIdTree)		
函数描述	获取组织 Id 树		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
	_departmentIdTree	string[]	组织树
返回值	无		获取组织 Id 树
接口原型	function __getRoleIdListByActionId(string _actionId, string[] storage roleIdList)		
函数描述	根据权限 Id 获得相关的角色列表		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
	roleIdList	String[]	角色列表
返回值	无		根据权限 Id 获得相关的角色列表

4.2.4. RoleManager.sol

接口原型	function insert(string _json) public returns(uint)		
函数描述	新增角色		
	参数名称	参数类型	取值说明
输入参数	_json	string	角色数据对象
返回值	_ret	uint	新增角色
接口原型	function update(string _json)		
函数描述	更新角色信息		

	参数名称	参数类型	取值说明
输入参数	_json	string	角色数据对象
返回值	无		更新角色信息
接口原型	function listAll() constant public returns (string _json)		
函数描述	查找所有角色		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_json	string	查找所有角色
接口原型	function findById(string _id) constant public returns(string _json)		
函数描述	根据角色 Id 查找相应角色		
	参数名称	参数类型	取值说明
输入参数	_id	string	角色 Id
返回值	_json	string	根据角色 Id 查找相应角色
接口原型	function findByName(string _name) constant public returns(string _json)		
函数描述	根据角色名称查找相应角色		
	参数名称	参数类型	取值说明
输入参数	_name	string	角色名称
返回值	_json	string	根据角色名称查找相应角色
接口原型	function checkRoleAction(string _roleId, string _actionId) constant public returns (uint _ret)		
函数描述	判断角色是否具有相应权限		
	参数名称	参数类型	取值说明
输入参数	_roleid	string	角色 Id
	_actionId	string	权限 Id
返回值	_ret	uint	判断角色是否具有相应权限
接口原型	function roleExists(string _roleId) constant public returns (uint _ret)		
函数描述	判断角色是否存在		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id
返回值	_ret	uint	判断角色是否存在
接口原型	function actionUsed(string _actionId) constant public returns (uint _used)		
函数描述	判断权限是否存在		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
返回值	_used	uint	判断权限是否存在
接口原型	function pageByName(string _name, uint _pageNum, uint _pageSize) constant public returns (string _json)		
函数描述	根据角色名称分页查询角色信息		
	参数名称	参数类型	取值说明
输入参数	_name	string	角色名称
	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_json	string	返回权限数据

接口原型	function pageByNameAndModuleId(string _moduleId,string _name, uint _pageNum, uint _pageSize) constant public returns (string _json)		
函数描述	根据角色名称和模块 Id 分页查询角色信息		
	参数名称	参数类型	取值说明
输入参数	_moduleId	string	模块 Id
	_name	string	角色名称
	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_json	string	返回权限数据
接口原型	function checkRoleActionWithKey(string _roleId, address _resKey, string _opKey) constant public returns (uint _ret)		
函数描述	判断角色是否具有相应权限		
	参数名称	参数类型	取值说明
输入参数	_roleid	string	角色 Id
	_resKey	string	资源符
	_opKey	string	操作符
返回值	_ret	uint	判断角色是否具有相应权限
接口原型	function deleteById(string _roleId)		
函数描述	删除角色		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id
返回值	无		删除角色
接口原型	function getRoleIdByActionIdAndIndex(string _actionId, uint _index) constant public returns (uint _roleId)		
函数描述	根据权限 Id 和索引号查找角色 Id		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
	_index	uint	索引号
返回值	_roleId	uint	根据权限 Id 和索引号查找角色 Id
接口原型	function __roleUsed(string _roleId) constant internal returns (uint _ret)		
函数描述	判断角色是否使用		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id
返回值	_ret	uint	判断角色是否使用
接口原型	function __actionExists(string _actionId) constant internal returns (uint _ret)		
函数描述	判断权限是否存在		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
返回值	_ret	uint	判断权限是否存在
接口原型	function __userExists(address _userAddr) constant internal returns (uint _ret)		
函数描述	判断用户是否存在		
	参数名称	参数类型	取值说明

输入参数	_userAddr	address	用户地址
返回值	_ret	uint	判断用户是否存在
接口原型	function __getUserDepartmentId(address _userAddr) constant internal returns (string _ret)		
函数描述	根据用户地址查找用户组织 Id		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_ret	string	查找用户组织 Id
接口原型	function __getUserRoleIdList(address _userAddr, string[] storage roleIdList)		
函数描述	获取用户角色列表		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
	_roleIdList	string[]	角色列表
返回值	无		获取用户角色列表
接口原型	function getActionIdListByRoleIdList(string[] storage roleIdList, string[] storage actionIdList)		
函数描述	根据角色列表活动权限列表		
	参数名称	参数类型	取值说明
输入参数	roleIdList	string[]	角色列表
	actionIdList	string[]	权限列表
返回值	无		根据角色列表活动权限列表
接口原型	function getRoleModuleId(string _roleId) constant public returns (uint _ret)		
函数描述	根据角色 Id 活动模块 Id		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id
返回值	_ret	uint	根据角色 Id 活动模块 Id

4.2.5. DepartmentManager.sol

接口原型	function insert(string _json) public returns(uint)		
函数描述	新增组织		
	参数名称	参数类型	取值说明
输入参数	_json	string	组织数据对象
返回值	_ret	uint	新增组织
接口原型	function update(string _json) public returns(uint)		
函数描述	更新组织信息		
	参数名称	参数类型	取值说明
输入参数	_json	string	组织数据对象
返回值	_ret	uint	更新组织信息
接口原型	function setDepartmentStatus(string _departmentId, uint _status) public returns(uint)		
函数描述	设置组织状态		
	参数名称	参数类型	取值说明

输入参数	_departmentId	string	组织 Id
	_status	uint	组织状态
返回值	_ret	uint	设置组织状态
接口原型	function setAdmin(string _departmentId, address _adminAddr) public returns(uint)		
函数描述	设置组织管理员		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
	_adminAddr	address	管理员地址
返回值	_ret	uint	设置组织管理员
接口原型	function eraseAdminByAddress(address _userAddr)		
函数描述	删除组织管理员		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	无		删除组织管理员
接口原型	function departmentExistsByCN(string _commonName) constant public returns (uint _ret)		
函数描述	根据组织 CN 判断组织是否存在		
	参数名称	参数类型	取值说明
输入参数	_commonName	string	组织 CN
返回值	_ret	uint	根据组织 CN 判断组织是否存在
接口原型	function departmentExists(string _departmentId) constant public returns (uint _exists)		
函数描述	根据组织 Id 判断组织是否存在		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
返回值	_exists	uint	根据组织 Id 判断组织是否存在
接口原型	function departmentEmpty(string _departmentId) constant public returns (bool _empty)		
函数描述	判断组织是否空		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
返回值	_empty	bool	判断组织是否空
接口原型	function deleteById(string _departmentId)		
函数描述	根据组织 Id 删除组织		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
返回值	无		根据组织 Id 删除组织
接口原型	function listAll() constant public returns (string _json)		
函数描述	查找所有组织		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_json	string	查找所有组织
接口原型	function findById(string _id) constant public returns(string _json)		
函数描述	根据组织 Id 查询组织信息		
	参数名称	参数类型	取值说明

输入参数	_id	string	组织 Id
返回值	_json	string	根据组织 Id 查询组织信息
接口原型	function findByName(string _name) constant public returns(string _json)		
函数描述	根据组织名称查询组织信息		
	参数名称	参数类型	取值说明
输入参数	_name	string	组织名称
返回值	_json	string	根据组织名称查询组织信息
接口原型	function findByParentId(string _parentId) constant public returns(string _json)		
函数描述	根据父组织 Id 查询组织信息		
	参数名称	参数类型	取值说明
输入参数	_parentId	string	父组织 Id
返回值	_json	string	根据父组织 Id 查询组织信息
接口原型	function getAdmin(string _departmentId) constant public returns(uint _admin)		
函数描述	查询组织管理员		
	参数名称	参数类型	取值说明
输入参数	departmentId	string	组织 Id
返回值	_admin	uint	查询组织管理员
接口原型	function getRevision() constant public returns(uint _revision)		
函数描述	查询版本号		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_revision	uint	查询版本号
接口原型	function pageByName(string _name, uint _pageNum, uint _pageSize) constant public returns(string _json)		
函数描述	根据组织名称分页查询组织信息		
	参数名称	参数类型	取值说明
输入参数	_name	string	组织名称
	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_json	string	返回组织数据
接口原型	function pageByNameAndStatus(string _parentId,uint _status, string _name, uint _pageNum, uint _pageSize) constant public returns(string _json)		
函数描述	根据组织名称和状态分页查询组织信息		
	参数名称	参数类型	取值说明
输入参数	_parentId	string	父组织 Id
	_status	uint	组织状态
	_name	string	组织名称
	_pageNum	uint	分页号
	_pageSize	uint	分页大小
返回值	_json	string	返回组织数据
接口原型	function getChildIdByIndex(string _departmentId, uint _index) constant public returns (uint _childDepartmentId)		

函数描述	根据组织 Id 和索引号查找子组织 Id		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 Id
	_index	uint	索引号
返回值	_childDepartmentId	uint	根据组织 Id 和索引号查找子组织 Id
接口原型	function checkWritePermission(address _addr, string _departmentId) constant public returns (uint _ret)		
函数描述	判断用户是否有操作权限		
	参数名称	参数类型	取值说明
输入参数	_addr	address	用户地址
	_departmentId	String	组织 Id
返回值	_ret	uint	判断用户是否有操作权限
接口原型	function getIndexById(string _id) constant private returns (uint)		
函数描述	获得此组织的索引号		
	参数名称	参数类型	取值说明
输入参数	_id	string	组织 Id
返回值	_ret	uint	获得此组织的索引号
接口原型	function getIndexByCommonName(string _commonName) constant private returns (uint)		
函数描述	根据组织 CN 获得索引号		
	参数名称	参数类型	取值说明
输入参数	_commonName	string	组织 CN
返回值	_ret	uint	根据组织 CN 获得索引号
接口原型	function __roleExists(string _roleId) constant internal returns (uint _ret)		
函数描述	判断角色是否存在		
	参数名称	参数类型	取值说明
输入参数	_roleId	string	角色 Id
返回值	_ret	uint	判断角色是否存在
接口原型	function __userExists(address _userAddr) constant internal returns (uint _ret)		
函数描述	判断用户是否存在		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_ret	uint	判断用户是否存在
接口原型	function __getUserDepartmentId(address _userAddr) constant internal returns (string _ret)		
函数描述	根据用户地址获得用户组织 Id		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_ret	string	获得用户组织 Id
接口原型	function __getUserCountByDepartmentId(string _departmentId) constant internal returns(uint _count)		
函数描述	根据组织 id 获得组织下的用户数量		
	参数名称	参数类型	取值说明

输入参数	_departmentId	string	组织 Id
返回值	_count	uint	获得组织下的用户数量

4.2.6. ActionManager.sol

接口原型	function getActionListByModuleId(string _moduleId) constant public returns(string _json)		
函数描述	根据模块 Id 获得模块下的权限列表		
	参数名称	参数类型	取值说明
输入参数	_moduleId	string	模块 Id
返回值	_json	string	获得模块下的权限列表
接口原型	function getActionListByContractId(string _contractId) constant public returns(string _json)		
函数描述	根据合约 id 获得合约下的权限列表		
	参数名称	参数类型	取值说明
输入参数	_contractId	string	合约 Id
返回值	_json	string	获得合约下的权限列表
接口原型	function actionExists(string _actionId) constant public returns(uint _ret)		
函数描述	判断权限是否存在		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
返回值	_ret	uint	判断权限是否存在
接口原型	function findByKey(string _resKey, string _opKey) constant public returns(string _actionJson)		
函数描述	返回权限信息		
	参数名称	参数类型	取值说明
输入参数	_resKey	string	资源符（合约名称）
	_opkey	string	操作符（函数名）
返回值	_actionJson	string	返回权限信息
接口原型	function findById(string _actionId) constant public returns(string _actionJson)		
函数描述	根据权限 Id 查找权限信息		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 id
返回值	_actionJson	string	根据权限 Id 查找权限信息
接口原型	function listAll() constant public returns(string _actionListJson)		
函数描述	查找所有权限列表		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_actionListJson	string	查找所有权限列表
接口原型	function listContractActions(string _contractName) constant public returns(string _actionListJson)		
函数描述	查询合约下的所有权限		
	参数名称	参数类型	取值说明
输入参数	_contractName	string	合约名称

返回值	_actionListJson	string	查询合约下的所有权限
接口原型	function checkActionWithKey(string _actionId, address _contractAddr, string _opSha3Key) constant public returns(uint _ret)		
函数描述	判断函数是否是此合约下的权限		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 Id
	_contractAddr	address	合约地址
	_opSha3Key	string	合约函数
返回值	_ret	uint	判断函数是否是此合约下的权限
接口原型	function insert(string _actionJson) public returns(bool _ret)		
函数描述	新增权限		
	参数名称	参数类型	取值说明
输入参数	_actionJson	string	权限数据对象
返回值	_ret	bool	新增权限
接口原型	function deleteById(string _actionId)		
函数描述	根据权限 Id 删除权限信息		
	参数名称	参数类型	取值说明
输入参数	_actionId	string	权限 id
返回值	无		根据权限 Id 删除权限信息
接口原型	function getCount() constant returns(uint _count)		
函数描述	查询权限数量		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_count	uint	查询权限数量

4.2.7. NodeInfoManager.sol

接口原型	function insert(string _json) public returns(uint)		
函数描述	新增节点		
	参数名称	参数类型	取值说明
输入参数	_json	string	节点 json 数据对象
返回值	_ret	uint	返回相应错误码
接口原型	function update(string _json) public returns(uint)		
函数描述	根据合约 id 获得合约下的权限列表		
	参数名称	参数类型	取值说明
输入参数	_json	string	节点 json 数据对象
返回值	_ret	uint	返回相应错误码
接口原型	function getEnodeList() constant public returns(string _json)		
函数描述	查找节点 IP 信息		
	参数名称	参数类型	取值说明
输入参数	无		

返回值	_json	string	返回节点 IP 的 json 字符串
接口原型	function ActivateEnode(string _pubkey) public		
函数描述	激活节点		
	参数名称	参数类型	取值说明
输入参数	_pubkey	string	节点公钥
返回值	无		
接口原型	function isInWhiteList(string _commonName, string _ip) constant public returns (string _json)		
函数描述	判断节点是否在白名单		
	参数名称	参数类型	取值说明
输入参数	_commonName	string	节点证书 CN
	_ip	string	节点 IP
返回值	_json	string	是：”true” 否： “false”
接口原型	function getNodeAdmin(string _nodeId) constant public returns(uint _admin)		
函数描述	查找节点管理员		
	参数名称	参数类型	取值说明
输入参数	_nodeId	string	节点 ID
返回值	_admin	uint	节点管理员地址
接口原型	function setAdmin(string _nodeId, address _adminAddr) public returns(uint)		
函数描述	设置节点管理员		
	参数名称	参数类型	取值说明
输入参数	_nodeId	string	节点 id
	_adminAddr	string	管理员地址
返回值	_ret	uint	返回相应错误码
接口原型	function eraseAdminByAdd(address _userAddr) public returns(uint)		
函数描述	删除管理员		
	参数名称	参数类型	取值说明
输入参数	_userAddr	address	用户地址
返回值	_ret	uint	返回相应错误码
接口原型	function nodeInfoExists(string _nodeId) constant public returns (uint _exists)		
函数描述	节点是否存在		
	参数名称	参数类型	取值说明
输入参数	_nodeId	string	节点 id
返回值	_exists	uint	0 存在 1 不存在
接口原型	function deleteById(string _nodeId)		
函数描述	根据节点 Id 删除节点信息		
	参数名称	参数类型	取值说明
输入参数	_nodeId	string	节点 id
返回值	无		
接口原型	function getRevision() constant public returns (uint _ret)		
函数描述	查找版本号		
	参数名称	参数类型	取值说明
输入参数	无		

返回值	_ret	uint	返回版本号
接口原型	function IPUsed(string _ip) constant public returns (uint _used)		
函数描述	检查 IP 是否已使用		
	参数名称	参数类型	取值说明
输入参数	_ip	String	ip
返回值	_userd	uint	1 已使用 0 未使用
接口原型	function listAll() constant public returns (string _json)		
函数描述	查找所有节点信息		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_json	string	返回所有节点信息 json 字符串
接口原型	function findById(string _id) constant public returns(string _json)		
函数描述	根据节点 id 查询节点信息		
	参数名称	参数类型	取值说明
输入参数	_id	string	节点 id
返回值	_json	string	返回节点信息
接口原型	function findByName(string _name) constant public returns(string _json)		
函数描述	根据节点名称查询节点信息		
	参数名称	参数类型	取值说明
输入参数	_name	string	节点名称
返回值	_json	string	返回节点信息
接口原型	function findByDepartmentId(string _departmentId) constant public returns(string _json)		
函数描述	根据节点组织 id 查询节点信息		
	参数名称	参数类型	取值说明
输入参数	_departmentId	string	组织 id
返回值	_json	string	返回节点信息
接口原型	function findByNodeAdmin(address _nodeAdmin) constant public returns(string _json)		
函数描述	根据节点管理员查询节点信息		
	参数名称	参数类型	取值说明
输入参数	_nodeAdmin	address	节点管理员地址
返回值	_json	string	返回节点信息
接口原型	function findByPubkey(string _pubkey) constant public returns(string _json)		
函数描述	根据节点公钥查询节点信息		
	参数名称	参数类型	取值说明
输入参数	_pubkey	string	节点公钥
返回值	_json	string	返回节点信息
接口原型	function checkWritePermission(address _addr, string _nodeInfoId) constant public returns (uint _ret)		
函数描述	判断用户是否由写入权限		
	参数名称	参数类型	取值说明
输入参数	_addr	address	用户地址
	_nodeInfoId	string	节点 Id

返回值	_ret	uint	0 有权限 1 无权限

4.2.8.NodeApplyManager.sol

接口原型	function insert(string _json) public returns(uint)		
函数描述	新增节点申请		
	参数名称	参数类型	取值说明
输入参数	_json	string	节点申请信息
返回值	_ret	uint	返回相应错误码
接口原型	function update(string _json) public returns(uint)		
函数描述	更新节点申请信息		
	参数名称	参数类型	取值说明
输入参数	_json	string	节点申请信息
返回值	_ret	uint	返回相应错误码
接口原型	function nodeApplyExists(string _nodeApplyId) constant public returns(uint _exists)		
函数描述	判断节点是否已申请		
	参数名称	参数类型	取值说明
输入参数	_nodeApplyId	string	节点申请 Id
返回值	_exists	uint	0 已申请 1 未申请
接口原型	function auditing(string _json) public returns(uint)		
函数描述	节点申请审核		
	参数名称	参数类型	取值说明
	_json	string	节点申请信息
返回值	_ret	uint	返回相应错误码
接口原型	function pageByNameAndStatus(uint _status, string _deptName, uint _pageNum, uint _pageSize) constant public returns(string _json)		
函数描述	根据名称和状态分页查询节点申请信息		
	参数名称	参数类型	取值说明
输入参数	_status	uint	节点申请状态
	_deptName	string	组织名称
	_pageNum	uint	页码
	_pageSize	uint	分页大小
返回值	_json	string	返回节点申请信息
接口原型	function listAll() constant public returns(string _json)		
函数描述	查找所有节点申请列表		
	参数名称	参数类型	取值说明
输入参数	无		
返回值	_json	string	返回所有节点申请列表
接口原型	function findByState(uint _state) constant public returns(string _strjson)		
函数描述	根据节点申请状态查找节点申请信息		
	参数名称	参数类型	取值说明

输入参数	_state	uint	申请状态
返回值	_strjson	string	返回节点申请信息
接口原型	function findByApplyId(string _nodeApplyId) constant public returns (string _json)		
函数描述	根据节点申请 id 查找申请信息		
	参数名称	参数类型	取值说明
输入参数	_nodeApplyId	string	申请 Id
返回值	_json	string	返回节点申请信息
接口原型	function deleteById(string _nodeApplyId) public returns(bool)		
函数描述	删除节点申请信息		
	参数名称	参数类型	取值说明
输入参数	_nodeApplyId	string	申请 id
返回值	_ret	bool	true 成功 false 失败

4.2.8. BaseModule.sol

接口原型	function addModule(string _json) internal returns(uint _ret)		
函数描述	新增模块		
	参数名称	参数类型	取值说明
输入参数	_json	string	模块对象
返回值	_ret	uint	返回的错误码
接口原型	function addContract(string _json) internal returns(uint _ret)		
函数描述	新增合约		
	参数名称	参数类型	取值说明
输入参数	_json	string	合约对象
返回值	_ret	uint	返回的错误码
接口原型	function addMenu(string _json) internal returns(uint _ret)		
函数描述	新增菜单		
	参数名称	参数类型	取值说明
输入参数	_json	string	菜单对象
返回值	_ret	uint	返回的错误码
接口原型	function addAction(string _json) internal returns(uint _ret)		
函数描述	新增权限		
	参数名称	参数类型	取值说明
输入参数	_json	string	权限对象
返回值	_ret	uint	返回的错误码
接口原型	function addRole(string _json) internal returns(uint _ret)		
函数描述	新增角色		
	参数名称	参数类型	取值说明
输入参数	_json	string	角色对象
返回值	_ret	uint	返回的错误码

4.3. 系统角色与权限说明

系统内置四种角色以及每个角色对应的权限，分别为：节点管理员、链管理员、系统管理员、权限管理员，每个角色以及其对应的权限说明如下：

角色名称	**节点管理员**	
角色权限	权限描述	四大合约权限以及节点信息管理权限
	所有权限	UserManager, DepartmentManager, ActionManager, RoleManager, NodeInfoManager
角色名称	**链管理员**	
角色权限	权限描述	四大合约权限以及节点申请权限
	所有权限	UserManager, DepartmentManager, ActionManager, RoleManager, NodeApplyManager
角色名称	**系统管理员**	
角色权限	权限描述	四大合约权限
	所有权限	UserManager, DepartmentManager, ActionManager, RoleManager
角色名称	**权限管理员**	
角色权限	权限描述	四大合约权限控制
	所有权限	UserManager, DepartmentManager, ActionManager, RoleManager

5.合约业务开发流程介绍

使用智能合约开发业务，需按照以下 5 个步骤进行：

1）完成合约基础数据的处理；

2）实现注册合约；

3）定义业务数据的结构，并实现对业务数据的处理；

4）根据接口文档实现业务合约的业务逻辑；

5）模块化管理。

下面将详细介绍这 5 个步骤。

5.1 完成合约基础数据的处理

合约基础数据为 int, string 类型，因此首先需要实现对这些基础数据的各种处理，如：

5.1.1 string 型数据截取：

```
function substr(string _self, uint _start, uint _len) internal returns (string _ret) {
       if (_len > bytes(_self).length-_start) {
           _len = bytes(_self).length-_start;
       }
       if (_len <= 0) {
           _ret = "";
           return;
       }
       _ret = new string(_len);
       uint selfptr;
       uint retptr;
       assembly {
           selfptr := add(_self, 0x20)
           retptr := add(_ret, 0x20)
       }
       memcpy(retptr, selfptr+_start, _len);
}
```

5.1.2 int 型数据转 string 型：

```
function toString(int _self) internal returns (string _ret) {
       if (_self == 0) {
           return "0";
       }
       uint ui = uint(_self);
       bool positive = true;
       uint8 len = 0;
       if (_self < 0) {
           ui = uint(-_self);
           positive = false;
           len++;
```

```
        }

        uint tmp = ui;
        while (tmp > 0) {
            tmp /= 10;
            len++;
        }

        _ret = new string(len);
        if (!positive) {
            bytes(_ret)[0] = '-';
        }

        uint8 i = len-1;
        while (ui > 0) {
            bytes(_ret)[i--] = byte(ui%10+0x30);
            ui /= 10;
        }
    }
```

总的来说这一步，就是完成基础数据的处理来满足所有需求。

5.2 实现注册合约

所有的业务合约都需在注册合约中注册，注册合约需实现合约注册、注销、获取合约地址等接口，从而所有注册过的发布入链合约，均能通过注册合约找到其在链上的地址，注册合约接口如：

注册信息由 Register 结构体构成：

```
struct Register {
        string name;                    //合约名称
        string version;                 //合约版本号
        address addr;                   //合约地址
        address origin;                 //coinbase 地址
}
```

注册接口示例代码：

```
function register(string _name, string _version) {
        address sendAddr = msg.sender;
        Register register = registerMap[sendAddr];
        register.name = _name;
        register.version = _version;
```

```
        register.addr = sendAddr;
        register.origin= tx.origin;
        string memory key = _name.concat(_version);
        keyMap[key] = sendAddr;
        contractAddrList.push(sendAddr);
 }
```

如发布组织合约 DepartmentManager，需在其构造函数内实现：

```
function DepartmentManager() {
        register("DepartmentManager", "0.0.1.0");
}
```

这里 register 函数是通过先在待发布合约中实例化注册合约，然后再调用注册合约的 register 函数。

5.3 定义业务数据结构

要完成所有业务合约，首先得先定义业务的数据结构，对应于合约的就是业务合约相应的 Lib 库。现有权限模型包含了组织、角色、用户、权限四大合约，以及管理节点信息的合约和处理节点审核信息的合约。

5.3.1 组织合约，组织的数据结构：

```
struct Department {
        string id;                      //组织 Id
        string name;                    //组织名称
        uint departmentLevel;           //组织层级：0 链本身 1 链下的一级组织 2 一级组织下的二级组织
        string parentId;                //父级组织 Id
        string description;             //组织描述
        uint creTime;                   //组织创建时间
        uint updTime;                   //组织更新时间
        string commonName;              //组织证书 CN
        string stateName;               //组织省份
        string countryName;             //组织国家
        address admin;                  //组织管理员
        address creator;                //组织创建者
        bool deleted;                   //组织是否删除：false 未删除 true 删除
        string orgaShortName;           //组织简称
        uint status;                    //组织状态：0 禁用 1 激活
        string groupPubkey;             //群公钥
    }
```

5.3.2 角色合约，角色的数据结构：

```
struct Role {
       string id;                    //角色 Id
       string name;                  //角色名称
       uint status;                  //角色状态
       string moduleId;              //角色所属模块 Id
       string contractId;            //角色所属合约 Id
       string description;           //角色描述
       uint creTime;                 //角色创建时间
       uint updTime;                 //角色更新时间
       bool deleted;                 //角色是否删除：false 未删除 true 删除
       string[] actionIdList;        //角色的权限列表
       address creator;              //角色创建者
   }
```

5.3.3 用户合约，用户的数据结构：

```
struct User {
        address    userAddr;          //用户地址
        string     name;              //用户名称
        string     account;           //用户账户
        string     email;             //用户邮箱
        string     mobile;            //用户手机
        string     departmentId;      //用户所属组织 ID
        uint       accountStatus;     //用户账户状态：0 解锁 1 锁定
        uint       passwordStatus;    //用户口令状态：  //待定？？？？
        uint       deleteStatus;      //用户删除状态：0 删除 1 未删除 【delete】
        string     uuid;              //用户的 uuid
        string     publicKey;         //用户公钥
        string     cipherGroupKey;    //用自己私钥加密群私钥后的密文
        uint       createTime;        //用户创建时间
        uint       updateTime;        //用户更新时间
        uint       loginTime;         //用户登录时间
        string[]   roleIdList;        //用户角色列表
        UserState  state;             //标记用户数据是否有效
        uint       status;            //是否禁用用户：0 禁用 1 激活
}
```

5.3.4 权限合约，权限的数据结构：

```
struct Action {
       string     id;                 //权限 ID
       string     name;               //权限名称
       string     contractId;         //权限所属合约 Id
       string     moduleId;           //权限所属模块 Id
       uint       level;              //权限层级：1 1 级菜单 2 2 级菜单 3 合约的函数接口
       uint       Type;               //权限类型：0 按钮 1 菜单 2 合约函数
       uint       enable;             //是否启用：0 未启用 1 启用
       string     parentId;           //权限父 Id
       string     url;                //权限资源定位器
       string     description;        //权限描述
       string     resKey;             //权限所属合约的合约名称，若权限是指菜单，则为菜单名称
       string     opKey;              //权限所指的合约函数，若权限是指菜单，则用阿拉伯数字
       string     version;            //版本号
       uint       createTime;         //权限创建时间
       uint       updateTime;         //权限更新时间
       ActionState state;             //权限状态：0 无效 1 有效
       address    creator;            //权限创建者
}
```

业务数据对象均以 Json 格式存储，因此在业务合约对应的 Lib 库合约中实现对业务数据的 Json 解析。

5.4 根据接口文档实现业务逻辑

业务合约首先定义好接口，其次业务合约需定义好每个合约的错误码，以供开发人员进行调试，满足业务各种异常情况处理，举用户合约的例子来说，用户合约的错误码定义为：

```
enum UserError {
       NO_ERROR,                      //0 无误
       BAD_PARAMETER,                 //1 参数不对
       NAME_EMPTY,                    //2 用户名字为空
       DEPT_NOT_EXISTS,               //3 组织 Id 不存在
       ROLE_ID_INVALID,               //4 角色 ID 无效
       USER_NOT_EXISTS,               //5 用户数据不存在
       ROLE_ID_ALREADY_EXISTS,        //6 角色已存在
       ADDRESS_ALREADY_EXISTS,        //7 地址已存在
       ACCOUNT_ALREADY_EXISTS,        //8 账号已存在
       ACCOUNT_CANNOT_UPDATE,         //9 账户不可被更新
       USER_LOGIN_FAILED,             //10 用户登录失败
```

```
    DEPT_CANNOT_UPDATE,             //11 用户组织信息不可更新
    NO_PERMISSION                   //12 无权限
}
```

通常每个业务合约固定一个错误码偏移量，如用户合约参数不对，返回错误码 error = 15000 + UserError.BAD_PARAMETER. 用户合约需实现用户的增加，删除，查询，用户信息更新等功能，新增用户接口传参为 string _userJson, 即用户数据，如

```
{
    "userAddr": "0x336d8f5f0c24bca870bf2b4f09741a896e757f27",
    "name": "name0001",
    "account": "applyUser0001",
    "email": "juzhen@juzix.io",
    "mobile": "13400001111",
    "departmentId": "department0001",
    "passwordStatus": 0,
    "accountStatus": 0,
    "deleteStatus": 0,
    "uuid": "uuid0001",
    "publicKey": "publicKey0001",
    "cipherGroupKey": "cipherGroupKey0001",
    "updateTime": 0,
    "createTime": 0,
    "loginTime": 0,
    "roleIdList": ["role100002"],
    "state": 0,
    "status": 0
}
```

将其转换成 string 型作为传参，新增用户成功的话，返回 0；反之，新增失败则根据返回的错误码，可以找到对应的缘由。

5.5 模块化管理

模块化管理，有助于进行合约管理，方便地实现合约的可插拔性。
合约的数据结构为：

```
struct Contract{
    string                  moduleId;           //所属模块 ID
    string                  contractId;         //合约 ID
    string                  contractName;       //合约名称
    string                  contractVersion;    //合约版本号
    bool                    deleted;            //合约是否删除：false 删除 true 未删除
    uint                    enable;             //合约是否启用：0 未启用 1 启用
    string                  description;        //合约描述
```

```
    uint                    createTime;         //合约创建时间
    uint                    updateTime;         //合约更新时间
    address                 creator;            //合约创建者
    address                 contractAddr;       //合约地址
}
```

每个业务合约均带有一个模块 ID，表示归属于此模块，目前所有业务合约均挂在系统模块下。模块下同时挂载了所有的权限，这里权限指的是所有合约的逻辑处理；也挂载了所有菜单，这里菜单指的是权限的功能集合；同时模块下也挂载了开发时设定的角色，即内置角色，每个角色分配好其所具有的权限，给每个用户分配权限时，只需给其分配相应的角色，用户就可获得此角色下的所有权限。

现在我们所有业务合约均归属于系统模块 SystemModule，因此在系统模块合约 SystemModuleManager 中，将所有业务合约、菜单、权限、角色加入到系统模块中，在此合约的构造函数中实现：

```
function SystemModuleManager(){
    register("SystemModuleManager","0.0.1.0");


    initContractData();     //添加合约数据
    initMenuData();         //添加菜单数据
    initActionData();       //添加权限数据
    initRoleData();         //添加角色数据
}
```

这样就将所有需求功能内置，在做实现应用时直接调取相应接口就可实现所需功能。