

华东师范大学数据科学与工程学院实验报告

课程名称: 区块链与分享型数据库	年级: 2018	上机实践成绩:
指导教师: 张召	姓名: 池欣宁	学号: 10185501409
上机实践名称: 以太坊-2		上机实践日期: 2021.4.29
上机实践编号:	组号:	上机实践时间: 10:00-11:30

一、实验目的

了解以太坊 DApp 开发, 使用 Truffle 框架搭建基于以太坊的宠物商店 Dapp, 进一步熟悉基于以太坊的编程。

二、实验任务

- (1) 创建 Truffle 项目
- (2) 编写智能合约
- (3) 编译和部署智能合约到区块链
- (4) 如何通过 Web3 和智能合约交互
- (5) 学习 Ganache 和 MetaMask 的使用
- (6) 编写 Student 智能合约代码并通过测试

三、使用环境

Vmware 虚拟机;
Macos Catalina;
Ubuntu;
Ganache;
MetaMask;
Python3, Web3;

四、实验过程

第一部分: 基于以太坊的 DApp 开发

Step1. 创建项目

项目结构:

contracts/: 智能合约文件夹(Adoption.sol)

migrations/: 用来处理部署 (迁移) 智能合约, 迁移是一个额外特别的合约用来保存合约的变化。

test/: 智能合约测试用例文件夹。

truffle.js: 配置文件

Step2. Truffle 框架下的项目创建与合约编译

编写智能合约文件 Adoption.sol, 实验指导书上已给出, 在这里不重复。

2.1 编译 truffle compile

2.2 编写部署脚本

在 **migrations/** 文件夹下已经有一个 **1_initial_migration.js** 部署脚本, 用来部署 Migrations.sol

合约。它用来确保不会部署相同的合约。

2.3 开启区块链

部署需要区块链，使用 Ganache 可快速开启一个私链来进行开发测试，默认在 7545 端口上运行一个开发链。

2.4 部署 truffle migrate

部署以后，就可以在 Ganache 账户页面可见消耗的以太币，区块页面可见产生了四个区块【如下图所示】。

ACCOUNTS

BLOCKS

TRANSACTIONS

CONTRACTS

EVENTS

LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK

4

GAS PRICE

20000000000

GAS LIMIT

6721975

HARDFORK

MUIRGLACIER

NETWORK ID

5777

RPC SERVER

HTTP://127.0.0.1:7545

MINING STATUS

AUTOMINING

WORKSPACE

QUICKSTART

SAVE

SV

BLOCK

4

MINED ON

2021-04-21 19:36:25

GAS USED

27338

1 T

BLOCK

3

MINED ON

2021-04-21 19:36:24

GAS USED

203827

1 T

BLOCK

2

MINED ON

2021-04-21 19:36:24

GAS USED

42338

1 T

BLOCK

1

MINED ON

2021-04-21 19:36:24

GAS USED

191943

1 T

BLOCK

0

MINED ON

2021-04-21 19:30:54

GAS USED

0

NO T

Step3:测试

根据 Step1.创建项目 中所陈述的项目结构可以知道，在 test/目录下新建 TestAdoption.sol 即可自行编写测试合约。

truffle 框架会帮我们自行生成 Assert.sol 和 DeployedAddress.sol

使用 truffle test 命令进行测试。

```

File | dase@ubuntu: ~/dase/pet-shop | 2_deploy_contract.js
Truffle v5.2.6 (core: 5.2.6)
Node v10.19.0
dase@ubuntu:~/dase/pet-shop$ truffle test
Using network 'development'.

Compiling your contracts...
=====
> Compiling ./test/TestAdoption.sol
> Artifacts written to /tmp/test--6534-qR3vIK77sQRf
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang

TestAdoption
  ✓ testUserCanAdoptPet (109ms)
  ✓ testGetAdopterAddressByPetId (97ms)
  ✓ testGetAdopterAddressByPetIdInArray (231ms)

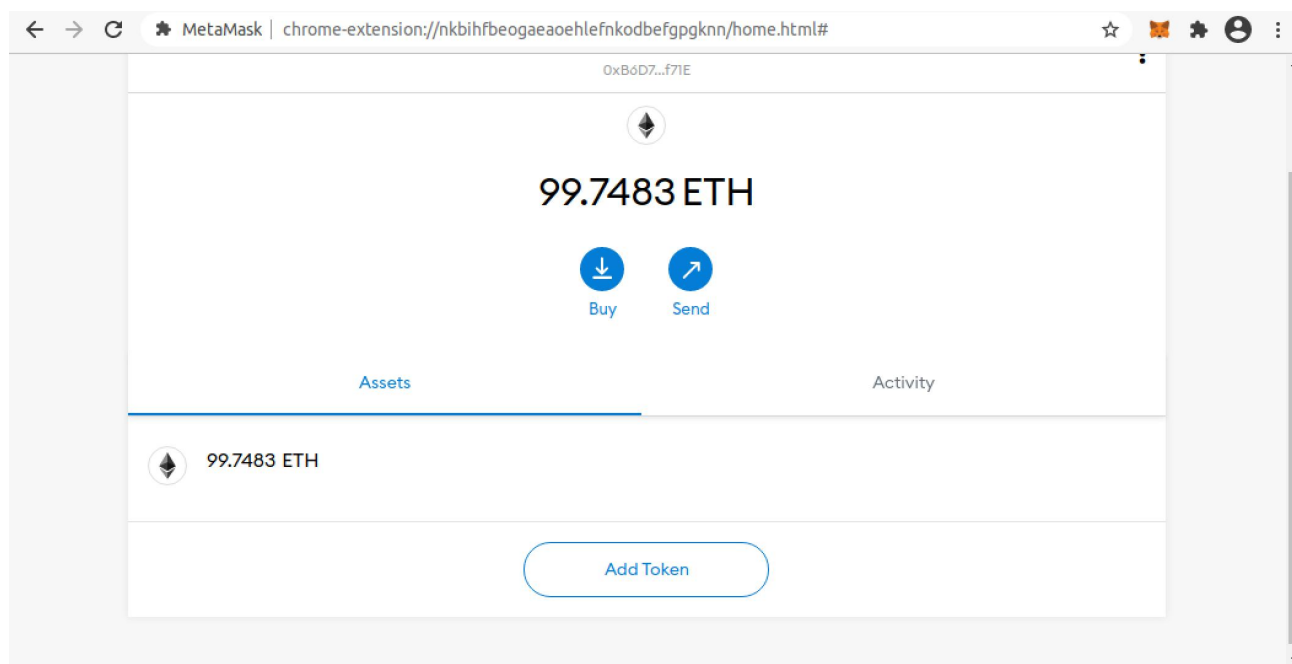
3 passing (12s)

dase@ubuntu:~/dase/pet-shop$
JS: truffle-confinis
> OUTLINE
Ln 29, Col 2 Spaces: 4 UTF-8 LF Solidity
  
```

Step4:在浏览器中运行 MetaMask

MetaMask 是一款在谷歌浏览器 Chrome 上使用的插件类型的以太坊钱包，该钱包不需要下载，只需要在谷歌浏览器添加对应的扩展程序即可，非常轻量级，使用起来也非常方便。

经过 MetaMask 一系列 GetStarted 部署以后，与本次实验的测试开发链进行链接。可以从下图看出 Assets 中的余额为 99.7483ETH，与我们从 Ganache 的 UI 上看到的结果相同。



Step5:启动 Web 应用服务 PetShop，在浏览器中与 dapp 交互

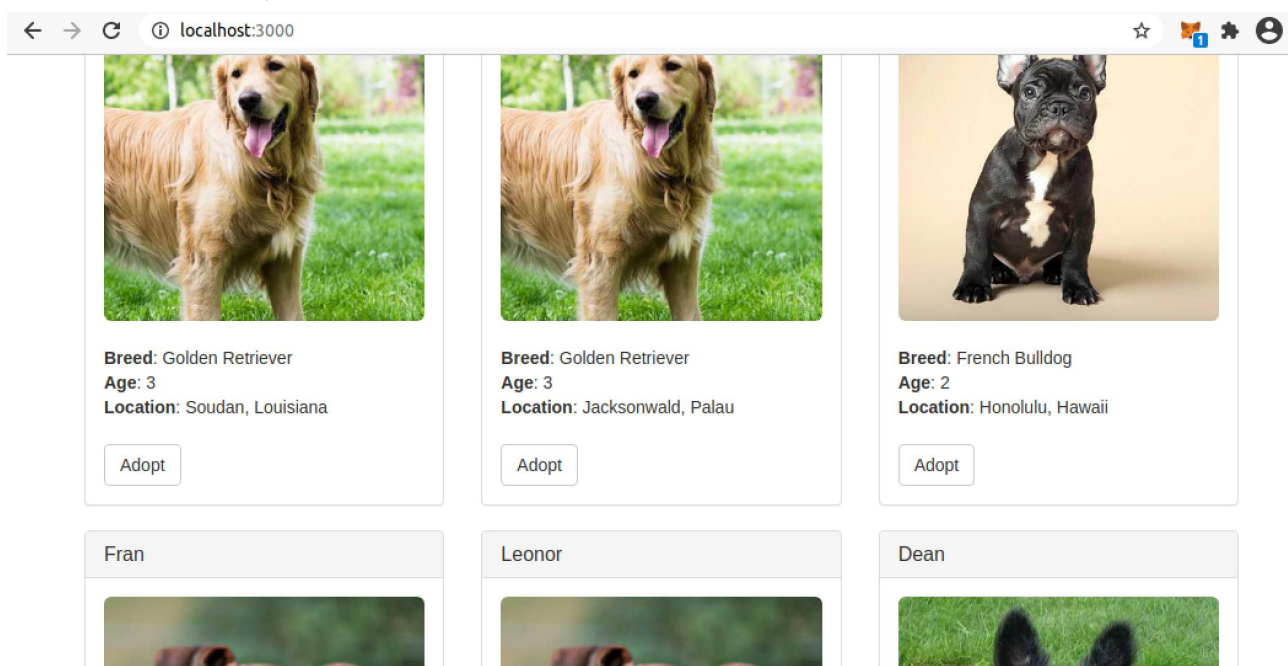
使用 `npm run dev` 启动 truffle 框架为我们打包好的 web 应用。

如果 `npm run dev` 无法运行，可以先清楚 npm 缓存【我在实验中就遇到了这个问题:D】

```
npm cache clean -force
rm -rf node_modules
rm -rf package-lock.json
npm install
npm run dev
```

```
dase@ubuntu: ~/dase/pet-shop
Local: http://localhost:3000
External: http://192.168.244.142:3000
-----
UI: http://localhost:3001
UI External: http://localhost:3001
-----
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Watching files...
[Browsersync] Couldn't open browser (if you are using BrowserSync in a headless
environment, you might want to set the open option to false)
21.04.21 20:04:18 200 GET /index.html
21.04.21 20:04:18 200 GET /css/bootstrap.min.css
21.04.21 20:04:18 200 GET /js/bootstrap.min.js
21.04.21 20:04:18 200 GET /js/web3.min.js
21.04.21 20:04:18 200 GET /js/app.js
21.04.21 20:04:18 200 GET /js/truffle-contract.js
21.04.21 20:04:19 200 GET /pets.json
21.04.21 20:04:19 200 GET /images/scottish-terrier.jpeg
21.04.21 20:04:19 200 GET /images/french-bulldog.jpeg
21.04.21 20:04:19 200 GET /images/boxer.jpeg
21.04.21 20:04:19 200 GET /images/golden-retriever.jpeg
21.04.21 20:04:19 404 GET /favicon.ico
```

上面流程成功以后，就可以看到我们的宠物商店惹！



第二部分：Student 智能合约的编写与测试

首先是测试结果：

【PS：因为我的本机环境拼命报 studentcontract Not Found 的错误，本来打算最终只能借用同学的代码环境，复制我的完整的 StudentContract.sol 的代码进行最终的 test 测试，但是莫名其妙重新启动 ganache 和 vmware 以后后就可以在本机运行了:D！

但是在实验中，不能完全的依赖 test 的报错信息，在任何编程场景都是如此，比如 test 的报错并不一定是输出信息提示的错误那样，错误的来源千奇百怪，需要通过及时调整，打印结

果等来更好的排查报错信息。

Step1.阅读 student_test.py 代码明确需求

要求：编写 StudentContract.sol 并通过最终测试

这次实验失去了助教学长手把手的教学，在一开始有些许的猝不及防，但是代码总得自己看，所以从 test 代码查看编写需求，也符合上学期数据库也是从 test 代码入手的测试驱动开发的要求不是:D

阅读了 student_test.py 代码以后，发现实验的需求还是写的比较清楚的，并且其中需要传参的 API 接口都很明确的写在了 student_test.py 里。

```
tx_hash = student_contract.functions.insert(*student).transact()
select_count = student_contract.functions.select_count().call()
student = student_contract.functions.select_id(id).call()
all_id = student_contract.functions.select_all_id().call()
```

大致上就是实现四个函数，分别是 insert,select_count,select_id,select_all_id;

Step2.整体 sol 代码框架的确定

根据实验课上所学的例子，可以很好的确定 sol 代码框架：

```
Contract StudentContract{
    Struct Student{}//类似于 C++的结构体
    Constructor(){}
    Function insert(){}
    Function select_count(){}
    Function select_all_id(){}
    Function select_id(){}
}
```


Step3.代码编写

可以看出，需要定义一个 `students` 数组，其次 `insert` 就是需要向这个数组新增元素，新增元素则需要 `students.push` 函数。

同时需要定义事件：`event Insert(uint id)`；和类似于与 C++ 成员变量的一个东西，即 `count` 记数来记录到底插入了多少。

```
function insert(uint _id, string memory _name, string memory: _sex,uint _age,string memory
_dept) public
{
    require(msg.sender==root);
    students.push(Student({id:_id,name:_name,sex:_sex,age:_age,dept:_dept}));
    count=count+1;//插入时对全局变量的一个记数
    emit Insert(_id);
}
```

返回记数的函数非常简单，只要返回全局数组即可。

```
function select_count()public view returns (uint _count){
    _count=count;
    return _count;
}
```

返回学生所有 `id` 的逻辑，我用了一种非常简单且粗暴的思路，这种代码的复杂度比较高，需要遍历 `students` 数组 $O(n)$ 。

在实验中，我只需要遍历 `students` 数组并且将每一个 `student` 成员变量的 `id` 存入 `uint` 类型的 `student ids` 数组即可。

```
function select_all_id() public returns (uint[] memory) {
    for(uint i=0;i<count;i++){
        student_ids.push(students[i].id);
    }
    return student_ids;
}
```

最后是返回 `students` 某 `id` 的 `student` 信息，在与同学交流讨论的过程中，我还是直接选择了遍历这一简单粗暴的方法，跟选择 `map` 的方法相比，具有 $O(n)$ 的复杂度，当然映射的话是 $O(1)$ 的复杂度。

```
function select_id(uint _id) public view returns (Student memory) {
    for(uint i=0;i<count;i++){
        if(students[i].id==_id){
            return students[i];
        }
    }
}
```

五、总结

1. 测试代码 debug 的时候不能完全依赖报错结果:

比如我一开始在进行 select_all_id 和 select_id 的测试时, 我发现我无论怎么检查我的 select_id 函数的逻辑都会报错, 同时, 通过在 student_test.py 文件中新增打印的两行, 发现可以打印 test_select_id 函数 (也就是最后一个测试函数的执行前一条和后一条, 联想到 py 作为脚本语言的按行执行的特点, 我就以为我这个测试函数其实也是执行的了, 但是非常奇怪的一点就是, 我不知道为什么报错信息, 也就是打分函数中, 并不告诉我我在哪里 fail 了从而

```

49
50     function select_id(uint _id) public view returns (Student memory) {
51         for(uint i=0;i<num;i++){
52             if(students[i].id == _id){
53                 return students[i];
54             }
55         }
56     }
57 }

```

```

PROBLEMS 15 OUTPUT TERMINAL DEBUG CONSOLE

insert (9, 'Vkpffr', 'female', 20, ' ') success!
    ==> Get 5 point, current point: 115
emit event success! event args: AttributeDict({'id': 9})
    ==> Get 5 point, current point: 120
actual_num(=10) == num(=10) success!
    ==> Get 10 point, current point: 130
actual_num(=10) == select_count(=10) success!
    ==> Get 10 point, current point: 140
select_all_id(=[]) success!
    ==> Get 20 point, current point: 160
*****888s
only admin success!
    ==> Get 10 point, current point: 170
point: 170
dase@ubuntu:~/student$

```

没有得到分数。

后来再检查了一遍 sol 代码, 发现是全球变量 num 和 count 重复定义和使用的错误, 并且进行了订正(因为得到 170 分的我天真地以为我只有最后一个函数 select_id 出错而不是拥有什么其他错误), 但还是很奇怪, select_id 函数的测试还是不能通过, 有同样逻辑错误但还未修改的 select_all_id 函数的测试却意外的通过了。

最后当我把 select_id 函数的逻辑修改正确以后, 我就发现我的测试通过了, 而导致我错误的原因是从 select_id 就有逻辑的部分错误, test_student.py 中 select_all_id 的测试虽然表面上通过了, 导致却没有打印 select_id 的信息。

2.solidity 编程中的 uint 和 int.

在编程中遇到的二者的区别很容易弄混淆, 就索性全部设置为 uint 了, 毕竟这里 id 或者 count 信息都是自然数。在 solidity 中的整数类型有 int 和 uint, int 类型可以存储负数, uint 类型只能够存储非负数。

其实 uint 和 int 的区别在大二上学期 CSAPP 就有所接触, 但无奈理论不能很好的联系编程实际, 且在计算机底层的语言中, 二者的区别体现在 overflow 的问题。

3.以太坊编程学习

这次关于以太坊编程的学习过程: 本来以为以太坊编程是一个比较困难的部分, 但是接触以后, 发现计算机编程里的很多逻辑都是相通的, 所以在看到 insert 函数的测试通过后, 本来以为很难的这次实践也变得更好上手了起来, 发现只要注意 solidity 的一些编程特性, 结合测试代码(测试驱动开发)的调试, 就可以完成所有的 test_case。

此次试验, 受益匪浅, 收获颇丰 ~

最后附上 StudentContract.sol 的完整代码:

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.16 <0.9.0; // 编译指令, 指明 solidity 版本, 上面一行是自由软件许可证

```
contract StudentContract {
    event Insert(uint id);
    struct Student {
        uint id;
        string name;
        string sex;
        uint age;
        string dept;
    }
    uint[] student_ids;//保存 student_ids 的数组
    address public root;
    Student[] students;
    uint count=0;
    constructor(){
        //合约
        root=msg.sender;
        count=0;
    }

    function insert(uint _id, string memory _name, string memory _sex,uint _age,string memory _dept) public
    {
        require(msg.sender==root);
        students.push(Student({id:_id,name:_name,sex:_sex,age:_age,dept:_dept}));
        count=count+1;//插入时对全局变量的一个记数
        emit Insert(_id);
    }

    function select_count()public view returns (uint _count){
        _count=count;
        return _count;
    }

    function select_all_id() public returns (uint[] memory) {
        for(uint i=0;i<count;i++){
            student_ids.push(students[i].id);
        }
        return student_ids;
    }

    function select_id(uint _id) public view returns (Student memory) {
        for(uint i=0;i<count;i++){
            if(students[i].id==_id){
                return students[i];
            }
        }
    }
}
```