以太坊智能合约部署（基于 geth 和 solc 方法）

1. 环境

Ubuntu 16.04

2. geth 安装

Installing from PPA：

sudo apt-get install software-properties-common

sudo add-apt-repository -y ppa:ethereum/ethereum

sudo apt-get update

sudo apt-get install ethereum

If you want to stay on the bleeding edge, install the ethereum-unstable package instead.

After installing, run geth account new to create an account on your node.

You should now be able to run geth and connect to the network.

Make sure to check the different options and commands with geth --help

You can alternatively install only the geth CLI with apt-get install geth if you don't want to install the other utilities (bootnode, evm, disasm, rlpdump, ethtest).

3. 启动环境

（1）利用 geth 启动一个以太坊（开发者）网络节点：

```
geth --datadir testNet --dev console 2>> test.log
```

命令参数说明:

–dev 启用开发者网络（模式），开发者网络会使用 POA 共识，默认预分配一个开发者账户并且会自动开启挖矿

–datadir 后面的参数是区块数据及秘钥存放目录

第一次输入命令后，它会放在当前目录下新建一个 testNet 目录来存放数据

console 进入控制台

2>> test.log 表示把控制台日志输出到 test.log 文件

（2）新开一个命令行终端，实时显示日志：

```
tail -f test.log
```

4. 账户准备

（1）部署智能合约需要一个外部账户，我们先来看看分配的开发者账户，在控制台使用以下命令查看账户：

```
> eth.accounts
```

此时我们看到，只有一个账户，即原来就有的开发者账户。

（2）使用如下命令确认该账户余额：

```
> eth.getBalance(eth.accounts[0])
```

我们会发现开发者账户中有大量的余额。如果用这个账户来部署合约

时会无法看到余额变化，为了更好的体验完整的过程，这里选择创建一个新的账户。

（3）使用如下命令创建新账户：

```
> personal.newAccount("sjtusjtu")
```

其中，sjtusjtu 是该账户的密码。此时查看账户列表：

```
> eth.accounts
["0x80b8da522ac1fb0952341e5877aefca209044dd3", "0x566ef461391272788d89d2213129d55dcd7c9477"]
```

可以看到账户数组包含两个账户，新账户在第二个（索引为 1）位置。

（4）查询我们新建账户的余额：

```
> eth.getBalance(eth.accounts[1])
```

可以发现新建账户中余额为 0，我们知道没有余额的账户是没法部署合约的，我们需要从默认账户转一些以太币给新账户。

（5）使用以下命令（请使用你自己的 eth.accounts 对应输入输出账户），转账 1 以太币：

```
> eth.sendTransaction({from:
'0x80b8da522ac1fb0952341e5877aefca209044dd3', to:
'0x566ef461391272788d89d2213129d55dcd7c9477', value:
web3.toWei(1, "ether")})
```

再次查询新账户余额：

```
> eth.getBalance(eth.accounts[1])
1000000000000000000
```

可以发现 1 以太币的转账已经到账。

5. 解锁账户

在部署合约前需要先解锁账户，使用以下命令：

```
> personal.unlockAccount(eth.accounts[1],"sjtusjtu");
```

注：解锁前需要停止挖矿：

```
> miner.stop()
```

检查挖矿是否已经停止，eth.mining 输出为 false 时表示挖矿已经停止：

```
> eth.mining
```

解锁之后一直启动挖矿（非常重要，如果不恢复挖矿无法部署智能合约）：

```
> miner.start()
```

6. 编写智能合约

（1）以 Hello World 为例，使用 Solidity 语言编写，代码如下：

```
pragma solidity ^0.4.18;
contract hello {
    string greeting;

    function hello(string _greeting) public {
        greeting = _greeting;
    }
```

```
    function say() constant public returns (string) {
        return greeting;
    }
}
```

简单解释下，我们定义了一个名为 hello 的合约，在合约初始化时保存了一个字符串（我们会传入 hello world），每次调用 say 返回字符串。

（2）使用 solc 编译代码

把这段代码拷贝到 Browser-Solidity（这是一个在线编译器：https://ethereum.github.io/browser-solidity/）。如果没有错误，点击 Details 获取部署代码。在弹出的对话框中找到 WEB3DEPLOY 部分，点拷贝，粘贴到编辑器后，进行一些修改：

第 1 行：修改字符串为 Hello World。

第 6 行：修改部署账户为新账户索引，即使用新账户来部署合约。

修改后的代码如下：

Code:

```
var _greeting = "Hello World" ;
var helloContract =
web3.eth.contract([{"constant":true,"inputs":[],"name":"say","
outputs":[{"name":"","type":"string"}],"payable":false,"stateM
utability":"view","type":"function"},{"inputs":[{"name":"_gree
ting","type":"string"}],"payable":false,"stateMutability":"non
payable","type":"constructor"}]);
var hello = helloContract.new(
   _greeting,
   {
     from: web3.eth.accounts[1],
     data:
'0x60606040523415610000f57600080fd5b6040516102b83803806102b8833
98101604052808051820191905050806000908051906020019061004192919
0610048565b50506100ed565b828054600181600116156101000203166002900
04906000526020600020906001f016020900481019282601f1061008957805
160ff19168380011785556100b7565b828001600101855582156100b757918
2015b828111156100b6578251825591602001919060010190610009b565b5b5
090506100c491906100c8565b5090565b6100ea91905b808211156100e6576
000816000905550600101610100ce565b5090565b90565b6101bc806100fc600
0396000f300606060405260043610610041576000357c01000000000000000
0000000000000000000000000000000000000000900463ffffffff1680639
54ab4b214610046575b600080fd5b341561005157600080fd5b6100596100d
4565b604051808060200182810382528381815181526020019150805190602
0019080838360005b838110156100995780820151818401526020810190506
1007e565b50505050905090810190601f1680156100c657808203805160018
36020036101000a031916815260200191505b509250505060405180910390f
35b6100dc61017c565b60008054600181600116156101000203166002900480
0601f016020809104026020016040519081016040528092919081815260200
18280546001816001161561010002031660029004801561017257806001f106
1014757610100808354040283529160200191610172565b820191906000526
020600020905b8154815290600101906020018083116101555782900360 1f1
68201915b505050505050905090565b60206040519081016040528060008152
0905600a165627a7a7230582014e5d933ceab6819d780e628120f754021768
a79be486a639b12ea90dee1437b0029',
    gas: '4700000'
  }, function (e, contract){
   console.log(e, contract);
   if (typeof contract.address !== 'undefined') {
       console.log('Contract mined! address: ' + contract.address
+ ' transactionHash: ' + contract.transactionHash);
   }
 })
```

（3）将编译结果拷贝回 geth 控制台

回车后，可以看到如下结果：

Contract mined! address:

0x42a07955d882e639696201c778d72cf2544a87c1 transactionHash:

0xd4fe1688886aee5cc7cbf5085a2c4ee28ba9ef291f22fed52ea3e68a647

eb905

合约部署成功。

7. 运行合约：

> hello.say()

"Hello World"

输出 Hello World，合约成功运行了。

运行截图如下：