# 导读

本电子书由汇智网（http://www.hubwiz.com）创作，适用于乌班图（Ubuntu）平台下以太坊 DApp 开发环境的搭建。

汇智网推出了在线交互式以太坊 DApp 实战开发课程，以去中心化投票应用（Voting DApp）为课程项目，通过三次迭代开发过程的详细讲解与在线实践，并且将区块链的理念与去中心化思想贯穿于课程实践过程中，为希望快速入门区块链开发的开发者提供了一个高效的学习与价值提升途径。读者可以通过以下链接访问《以太坊 DApp 开发实战入门》在线教程：

http://xc.hubwiz.com/course/5a952991adb3847553d205d1?affid=ubuntu7878

教程预置了开发环境。进入教程后，可以在每一个知识点立刻进行同步实践，而不必在开发环境的搭建上浪费时间：

# 一、安装前的准备

## 1.1 查看当前 CPU 架构

在终端中执行以下命令，确定是 32 位架构还是 64 位架构：

```
~$ uname –p
x86_64
```

如果你看到输出 x86_64，那么就是 64 位系统，否则是 32 位。

## 1.2 下载工具

确保你安装了下载工具 wget：

```
~$ wget –V
GNU Wget 1.17.1 built on linux-gnu
```

如果还没有安装 wget，使用 apt-get 来安装

```
~$ sudo apt-get install wget
```

# 二、安装 DApp 开发环境

## 2.1 安装 Node.js

首先根据你的 ubuntu 是 32 位还是 64 位，分别下载不同的预编译版本，我们使用官方长期支持的 8.10.0LTS 版本：

64 位：

```
~$ wget https://nodejs.org/dist/v8.10.0/node-v8.10.0-linux-x64.tar.gz
```

32 位：

```
~$ wget https://nodejs.org/dist/v8.10.0/node-v8.10.0-linux-x86.tar.gz
```

然后解压到当前目录，以 64 位为例：

```
~$ tar zxvf node-v8.10.0-linux-x64.tar.gz
```

然后接下来修改.bashrc 来设置相关的环境变量：

```
~$ echo "export NODE_HOME=$HOME/node-v8.10.0-linux-x64" >> .bashrc
~$ echo "export NODE_PATH=$NODE_HOME/lib/node_modules" >> .bashrc
~$ echo "export PATH=$NODE_HOME/bin:$PATH" >> .bashrc
```

最后重新载入.bashrc（或者重新登陆）来使 node 生效：

```
~$ source .bashrc
```

现在，你可以使用 node 了：

```
~$ node –v
v8.10.0
```

## 2.2 安装 Geth

在终端执行以下命令：

```
~$ wget https://gethstore.blob.core.windows.net/builds/geth-linux-amd64-1.8.3-329ac18e.tar.gz
~$ mv get-linux-amd64-1.8.3-329ac18e geth
~$ echo export PATH=$HOME/geth:$PATH >> .bashrc
~$ soure .bashrc
```

安装完毕后，执行命令验证安装成功：

```
~$ geth version
Geth
Version: 1.8.3-stable
```

## 2.3 安装 solidity 编译器

```
~$ npm install -g solc
```

安装完毕后，执行命令验证安装成功

```
~$ solcjs -version
0.40.2+commit.3155dd80.Emscripten.clang
```

## 2.4 安装 web3

```
~$ npm install -g web3@0.20.2
```

安装验证：

```
~$ node -p 'require("web3")'
{[Function: Web3]
  providers:{…}}
```

## 2.5 安装 truffle 框架

执行以下命令安装 truffle 开发框架：

```
~$ npm install -g truffle
```

验证安装：

```
~$ truffle version
Truffle v4.1.3 (core 4.1.3)
```

## 2.6 安装 webpack

执行以下命令安装 webpack：

```
~$ npm install -g webpack@3.11.0
```

验证安装

```
~$ webpack –v
3.11.0
```

# 三、运行私链节点

## 3.1 创世块配置

创建一个节点目录 node1，并在其中创建私链的创世块配置文件：

```
~$ mkdir node1
~$ cd node1
~/node1$ touch private.json
```

然后编辑内容如下：

```
{
    "config": {
        "chainId": 7878,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "200",
    "gasLimit": "2100000",
    "alloc": {
        "7df9a875a174b3bc565e6424a0050ebc1b2d1d82": { "balance": "300000" },
        "f41c74c9ae680c1aa78f42e5647a62f353b7bdde": { "balance": "400000" }
    }
}
```

config.chainId 用来声明以太坊网络编号，选择一个大于 10 的数字即可。

difficulty 用来声明挖矿难度，越小的值难度越低，也就能更快速地出块。

## 3.2 初始化私链节点

执行 geth 的 init 命令初始化私链节点：

```
~/node1$ geth --datadir ./data init private.json
```

这会在当前目录下创建 data 目录，用来保存区块数据及账户信息：

```
~/node1$ ls
```

```
data private.json
```

可以上述命令写到一个脚本 init.sh 里，这样避免每次都输入那么多记不住的东西：

```
~/node1$ touch init.sh
~node1$ chmod +x init.sh
```

编辑内容如下：

```
#!/bin/bash
geth --datadir ./data init private.json
```

在部署下一个节点时，就可以直接执行这个脚本进行初始化了。例如，在另一台机器上：

```
~/node1$ ./init.sh
```

## 3.3 启动私链节点

从指定的私链数据目录启动并设定一个不同的网络编号来启动节点：

```
~/node1$ geth --rpc --datadir ./data --networkid 7878 console
```

同样，你可以用一个脚本 console.sh 来简化启动节点时的输入：

```
~/node1$ touch console.sh
~/node1$ chmod +x console.sh
```

编辑内容如下：

```
#!/bin/bash
geth --rpc \
     --rpcaddr 0.0.0.0 \
     --rpccorsdomain "*" \
     --datadir ./data \
     --networkid 7878 \
     console
```

rpcaddr 参数用来声明节点 RPC API 的监听地址，设为 0.0.0.0 就可以从其他机器访问 API 了；rpccorsdomain 参数是为了解决 web3 从浏览器中跨域调用的安全限制问题。

以后启动节点，只要直接执行这个脚本即可：

```
~/node1$ ./console.sh
```

## 3.4 账户管理

### 3.4.1 查看账户列表

在 geth 控制台，使用 eth 对象的 accounts 属性查看目前的账户列表：

```
> eth.accounts
[]
```

因为我们还没有创建账户，所以这个列表还是空的。

### 3.4.2 创建新账户

在 geth 控制台，使用 personal 对象的 newAccount()方法创建一个新账户，参数为你自己选择的密码：

```
> personal.newAccount('78787878')
0xd8bcf1324d566cbec5d3b67e6e14485b06a41d49
```

输出就是新创建的账户地址（公钥），你的输出不会和上面的示例相同。geth 会保存到数据目录下的 keystore 文件中。密码要自己记住，以后还需要用到。

### 3.4.3 查询账户余额

在 geth 控制台，使用 personal 对象的 getBalance()方法获取指定账户的余额，参数为账户地址：

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

或者直接输入账户地址：

```
> eth.getBalance('0xd8bcf1324d566cbec5d3b67e6e14485b06a41d49')
0
```

新创建的账户，余额果然为 0。

## 3.4.4 挖矿

没钱的账户什么也干不了，需要挖矿来挣点钱。

在 geth 控制台执行 miner 对象的 start()方法来启动挖矿：

```
> miner.start(1)
```

等几分钟以后，检查账户余额：

```
> eth.getBalance(eth.accounts[0])
2.695e+21
```

钱不少了，2695ETH 了，目前市值将近 500 万人民币了，哈。

执行 miner 对象的 stop()方法停止挖矿：

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

## 3.4.5 解锁账户

在部署合约时需要一个解锁的账户。在 geth 控制台使用 personal 对象的 unlockAccount()

方法来解锁指定的账户，参数为账户地址和账户密码（在创建账户时指定的那个密码）：

```
> eth.unlockAccount(eth.accounts[0],'78787878')
true
```

# 四、构建示例项目

## 4.1 新建 DApp 项目

执行以下命令创建项目目录并进入该目录：

```
~$ mkdir demo
~$ cd demo
```

然后用 webpack 模版初始化项目骨架结构：

```
~/demo$ truffle unbox webpack
Downloading…
Unpacking…
Setting up…
Unbox successful. Sweet!
```

## 4.2 安装项目依赖的 NPM 包

执行以下命令安装 nmp 包：

```
~/demo$ npm install
```

## 4.3 修改 truffle 配置

truffle.js 中，修改 port 为 8545，因为 geth 默认在 8545 端口监听：

```
module.exports = {
  networks:{
    development: {
      …
      port: 8545
      …
    }
  }
}
```

## 4.4 启动节点

在 另一个终端，执行以下命令启动节点软件，以便部署合约并执行交易：

```
~$ cd node1
~/node1$ ./console.sh
>
```

**注意：**为了在节点上部署合约，别忘了启动 geth 后先解锁账户：

```
> personal.unlockAcount(eth.accounts[0],'78787878')
true
```

## 4.5 编译合约

执行以下命令编译项目合约：

```
~/demo$ truffle compile
```

## 4.6 部署合约

执行以下命令来部署合约：

```
~/demo$ truffle migrate
```

如果你之前忘了在 geth 控制台解锁账户，会看到如下错误，参考前面说明进行解锁即可：

```
…
Error: authentication needed: password or unlock
```

如果已经正确地解锁了账户，你会看到部署过程停止在如下状态：

```
Replacing Migrations…
… 0x3088762a5bc9…
```

这是因为 truffle 在等待部署交易提交，但是我们在私链中还没有启动挖矿。

现在切换回 geth 终端窗口，查看交易池的状态：

```
> txpool.status
{
  pending:1,
  queued:0
}
```

果然有一个挂起的交易！启动挖矿就是了：

```
> miner.start(1)
```

稍等小会儿，再查看交易池的状态：

```
> txpool.status
{
  pending:0,
  queued:0
}
```

交易已经成功提交了。我们可以停止挖矿了，因为它太占 CPU 了：

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

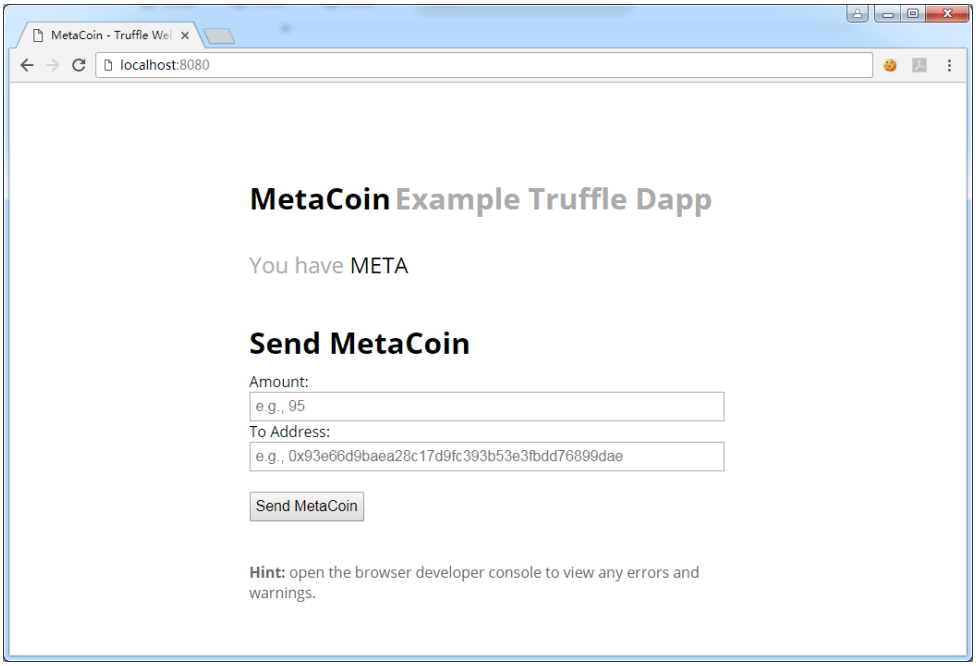
现在切换回 truffle 那个终端，部署过程也正确地执行完了。

## 4.7 启动 DApp

执行以下命令来启动 DApp：

```
~/demo$ npm run dev
```

在浏览器里访问 http://localhost:8080 即可




如果你希望从别的机器也可以访问你的 DApp 应用，修改一下 package.json：

```
{
```

```
  scripts:{
    "dev": "webpack-dev-server --host 0.0.0.0"
  }
}
```