**Aelf区块链系统搭建步骤、方法及智能合约开发介绍**

一、概要：

本文介绍基于Linux ubuntu 操作系统搭建AElf Boilerplate开发环境、运行智能合约DAPP的方法和步骤，重点是介绍和官方文档不一样的地方，也就是重点介绍搭建环境和运行Demo时所入的坑（问题），希望对要在Aelf上搭建智能合约开发提供快速顺利搭建的环境，为开起基于Aelf智能合约愉快之旅的第一步。

二、环境

1、整体运行环境：本文档是基于Linux  Ubuntu 18.0.4 + Windows 10 搭建和运行，其中Linux  操作系统跑节点和服务，windows跑客户端和JS，通过ssh协议终端远程访问Linux 上的Aelf服务，这是一种最通用的操作方式和BS架构。

2、 参考文档，来自Aelf官方文档：https://docs.aelf.io/main/main/setup

3、开发环境：

1）、准备安装linux Ubuntu 操作系统，版本18.0.4 以上，server版；

2)、开发工具：

A：安装git:安装参考开发文档：<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

在linux命令下输入以下命令：

sudo apt install git-all

  sudo apt-get install dh-autoreconf libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev

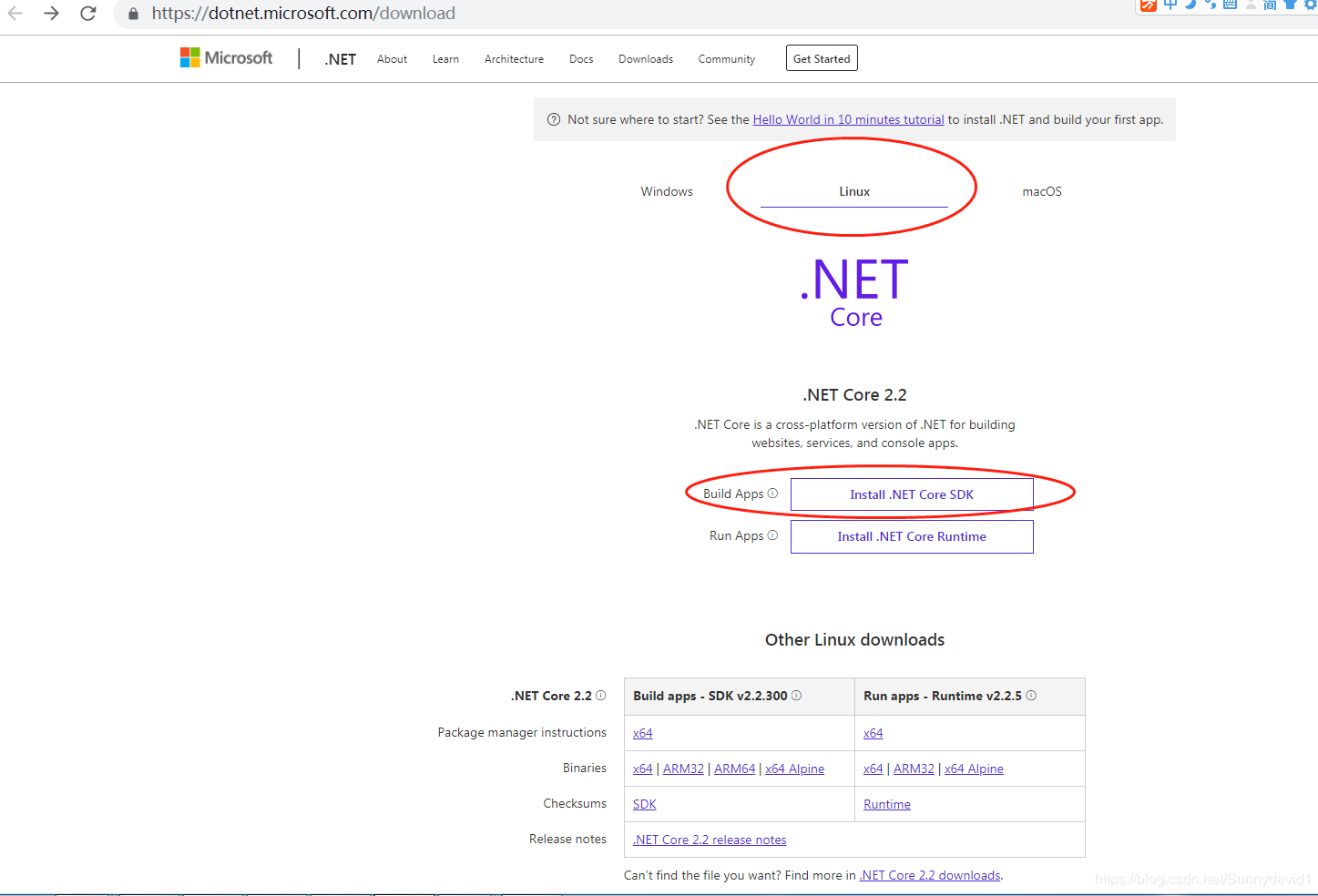
  sudo apt-get install asciidoc xmlto docbook2x

  sudo apt-get install install-info

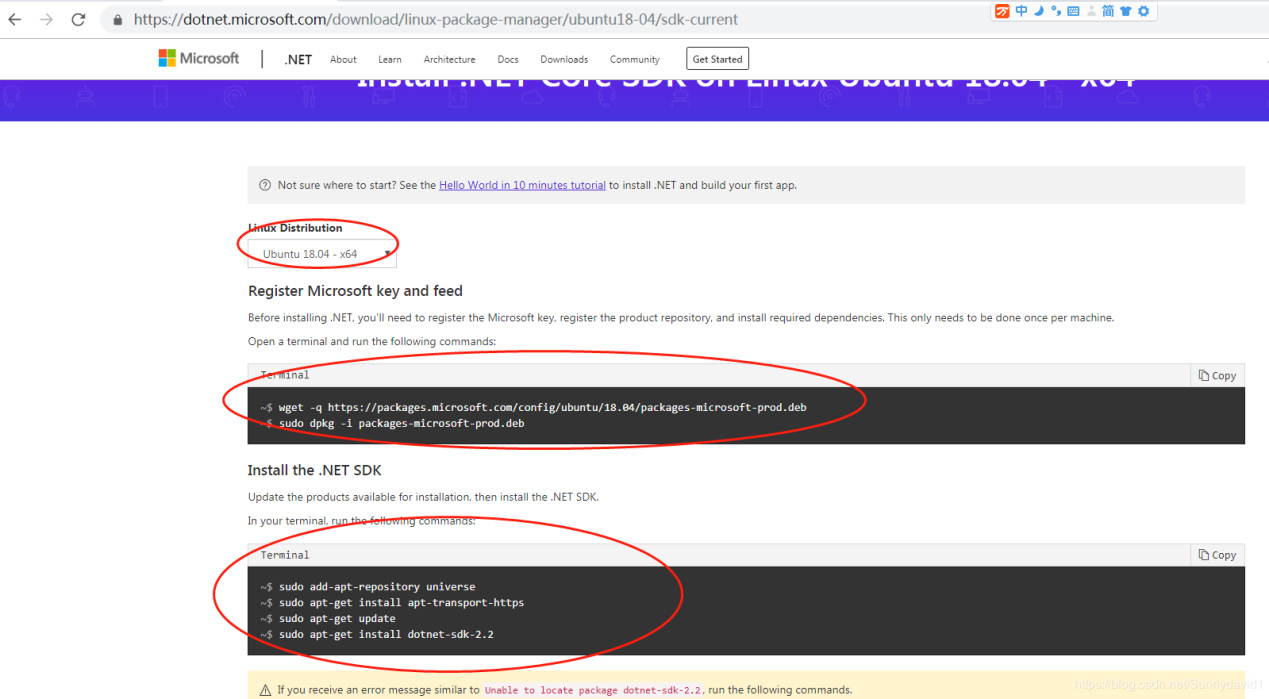
  sudo apt-get install getopt

B: dotnet core sdk 2.2 官方下载地址：https://dotnet.microsoft.com/download

linux 版本下载以下：



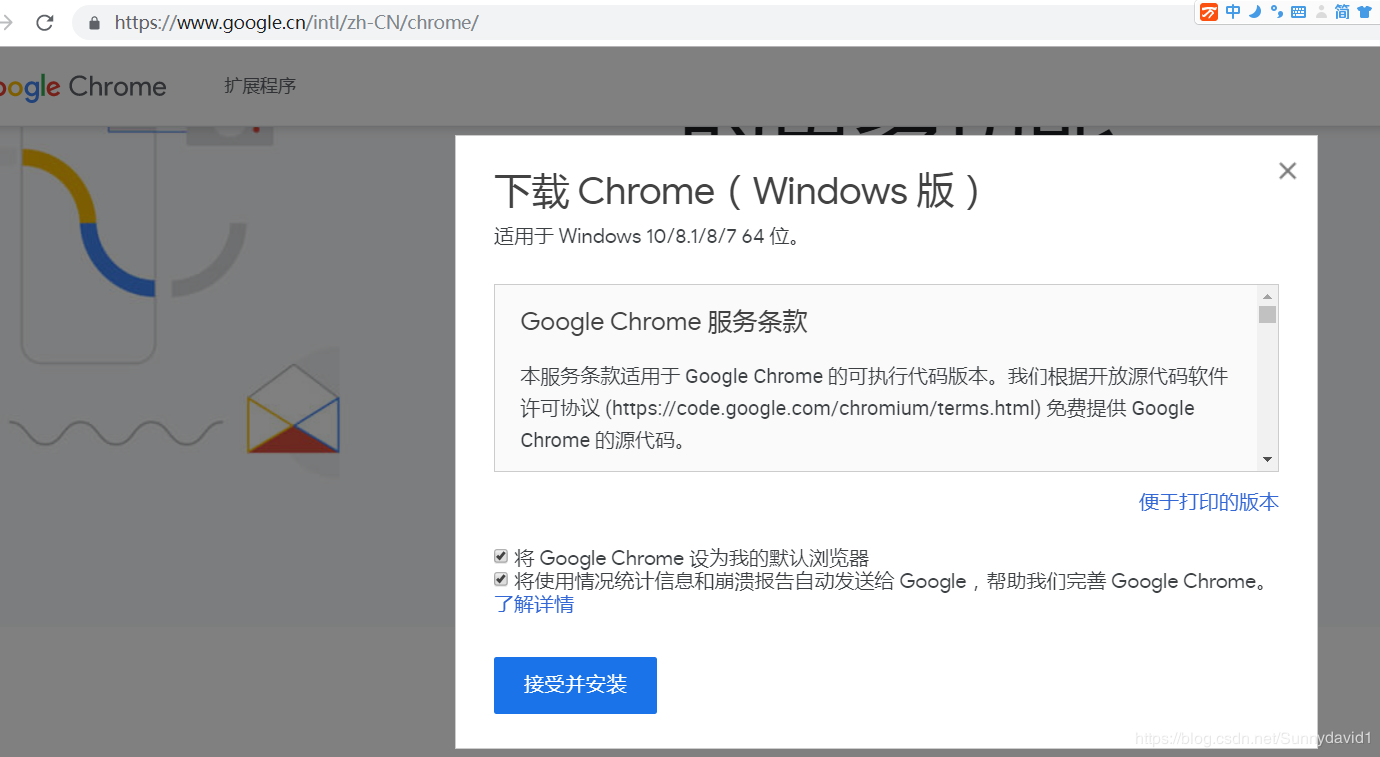
点击 Install Net Core SDK,如下：按以下红框选择和输入命令安装：



C：在windows 系统安装chrome浏览器。

注意：chrom的版本位要和系统版本位一致，不然后面跑demo 3时，即时安装钱包插件，通信也有问题，也就是说如果你的windos系统是64位，那么安装chrom也要是64位，同理，如果你的windos系统是32位，那么安装chrom也要是32位。

官方下载地址：https://www.google.cn/intl/zh-CN/chrome/



D:安装IDE  Visual Studio Code (vscode)，官方下载地址：https://code.visualstudio.com/docs/setup/linux

在linux下安装命令：

sudo apt-get install apt-transport-https

      sudo apt-get update

      sudo apt-get install code # or code-insiders

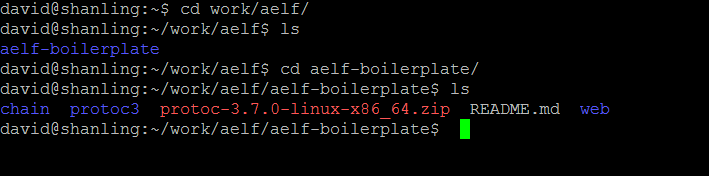
到此，开发环境 已搭建完毕，可以小熄一下，喝杯茶，接下来开始智能合约Deom 体验之旅~~！

三、体验Demo

1、git clone aelf-boilerplate, 命令如下：

git clone https://github.com/AElfProject/aelf-boilerplate

这个命令将在目录下创建AElfProject/aelf-boilerplate文件夹，下载代码，大概需要几分种时间，如我的截图：



2、运行Demo

1)、安装导入工程和编译运行链节点,

注意：以下命令模式下全程需要把用户加入到超级用户组，运行命令时要在前面加入sudo ，不然会报很多错误，大部分都是权限问题，如果报错，记得在命令行前加入sudo,以下操作我都会在官方文档命令前面带上sudo。

A:安装：

    在linux 终端运行以下命令

sudo bash chain/scripts/install.sh

如果输入sudo sh chain/scripts/install.sh  会报以下错误：

  chain/scripts/install.sh: 14: [: linux: unexpected operator

  chain/scripts/install.sh: 16: [: linux: unexpected operator

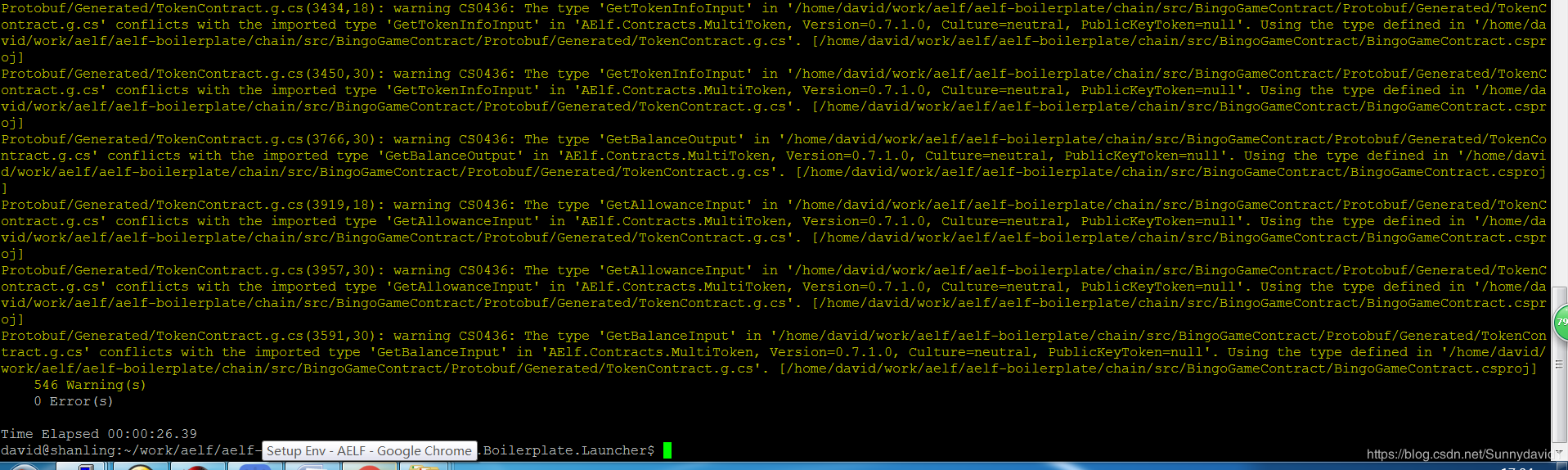
B:Build and Run

cd aelf-boilerplate

  cd chain/src/AElf.Boilerplate.Launcher/

  sudo dotnet build

编译预计几分钟，会出现很多黄色字体，忽略，只要不爆红都OK，以下表示编译成功：



启动运行智能合约链：

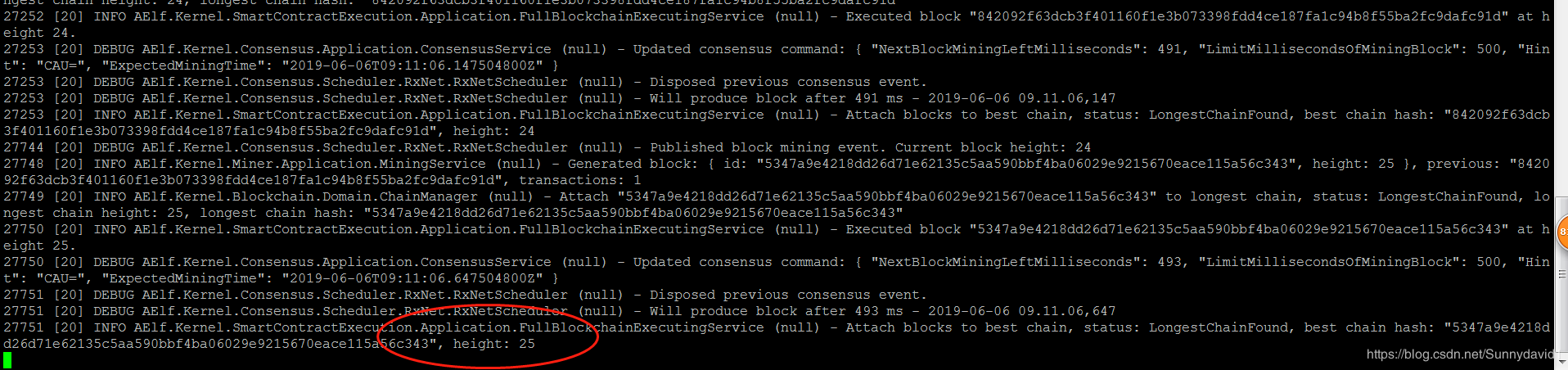
注意：这条链在整个Demo运行中，都得在run,不能关闭和退出。

在命令行中输入以下命令：

  sudo dotnet run bin/Debug/netcoreapp2.2/AElf.Boilerplate.Launcher

出现以下提示表示成功，些时会500ms左右更新一个区块高度，高度不断叠加。

如果你想退出，按Ctrl+C 停止或退出链，要启动，再按以上命令即可。



 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~·

到此分割线，智能主链已运行起来，开始Demo之旅。！！

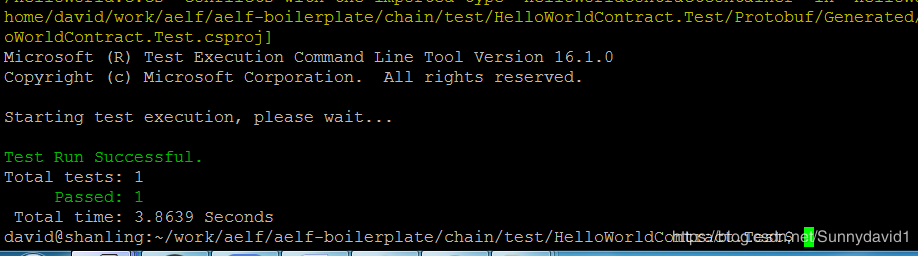
2、.Demo1: HelloWorld 智能合约 Demo

  打开另一个终端，输入以下命令：

cd chain/test/HelloWorldContract.Test/

sudo dotnet test

出现以下表示成功：



3、Demo2：运行 JS SDK Demo

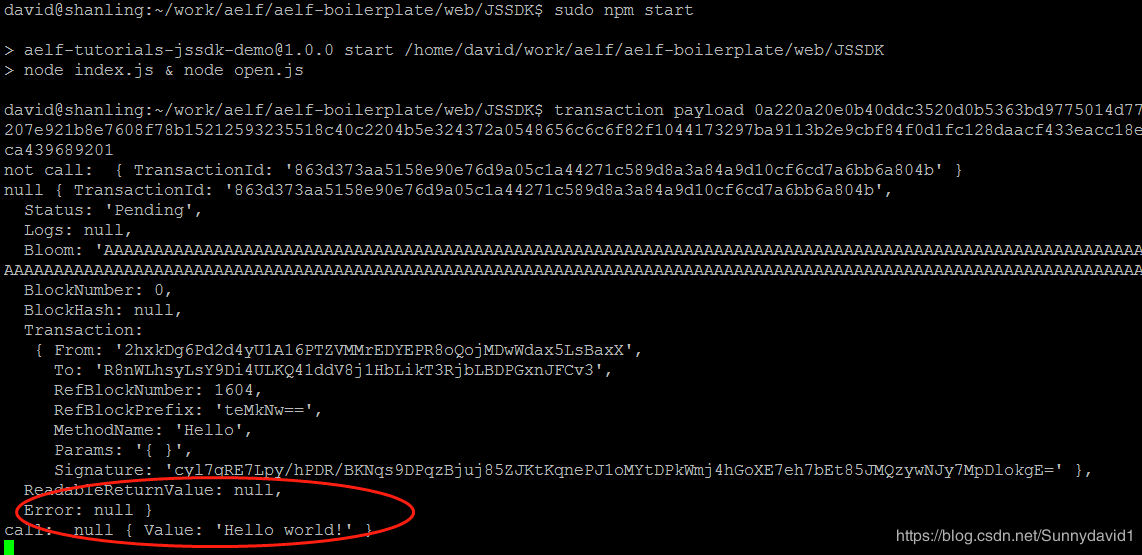
在终端中输入以下命令：

  cd ../../../web/JSSDK/

  sudo npm install

  sudo npm start

在终端中出现以下界面，成功：

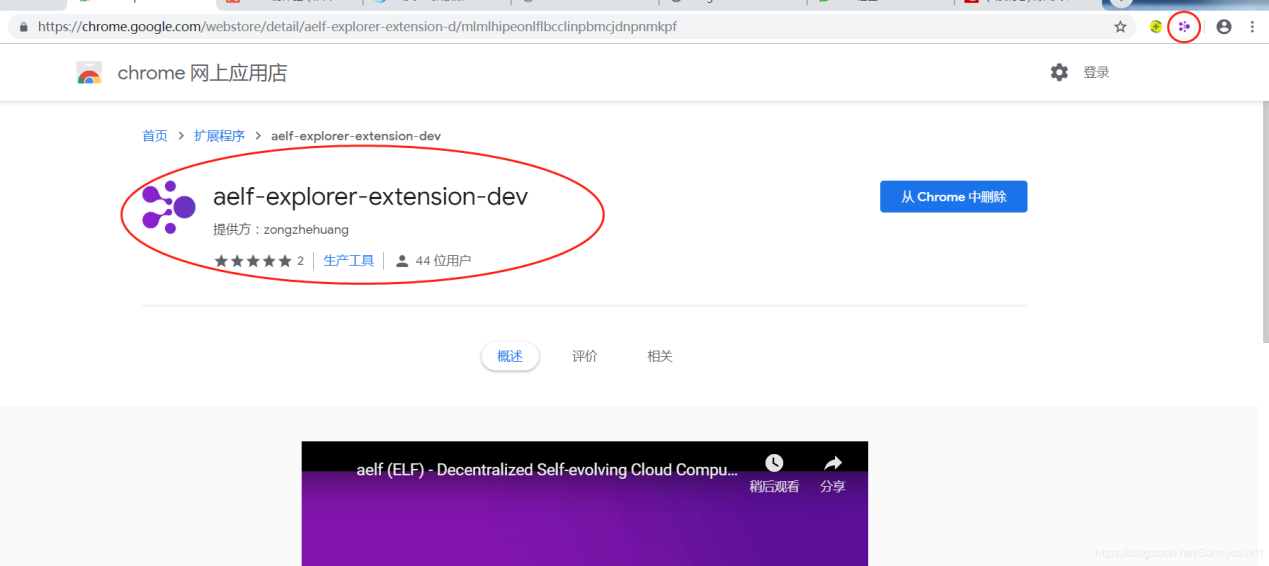


4：Demo3:运行 AElf 浏览器插件——NightELF Demo

A)、在Windos端安装浏览器插件，注意：此步需要科学上网，要梯子，通俗就叫翻墙，自己准备。

用chrome浏览器打开以下地址：https://chrome.google.com/webstore/detail/aelf-explorer-extension-d/mlmlhipeonlflbcclinpbmcjdnpnmkpf

在搜索框中输入aelf-explorer-extension-dev 点击安装即可。



B)、打开Aelf Nigth  钱包插件，创建钱包，钥匙，Open the extension, create your wallet, create your keypair.，打开chrome浏览器，点

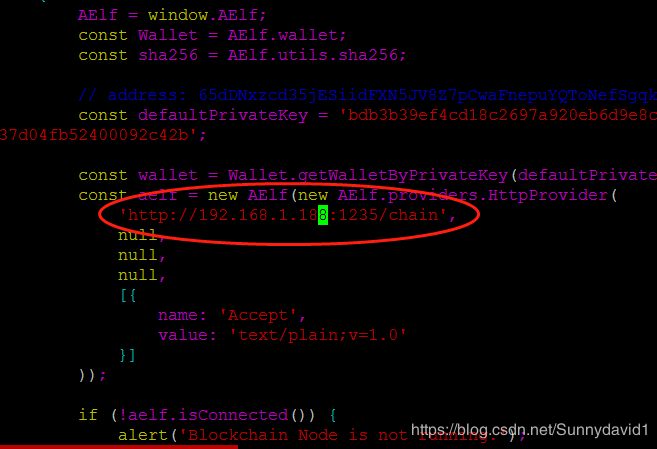


以上图红圈，出现Night Aelf,  配置方法参考以下文档：

NightELF 的使用教程：https://zmh3788.gitbook.io/aelf-web-extensions/

C)、修改相关代码和脚本，具体如下：

1、修改index.html ,搜索文件中的127.0.0.1 ，把此IP改为服务器的Ip地址，保存退出。我的服务器IP地址为192.168.1.188



2、修改index.js，搜索文件中的127.0.0.1 ，把此IP改为服务器的Ip地址，保存退出。



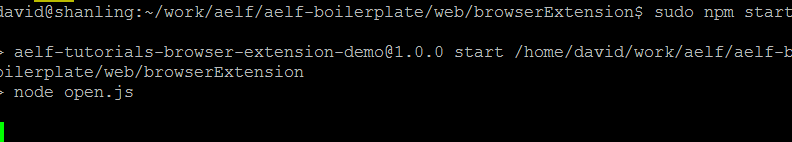
3、在linux 终端输入以下命令：

cd ../browserExtension/

sudo npm install

sudo npm start

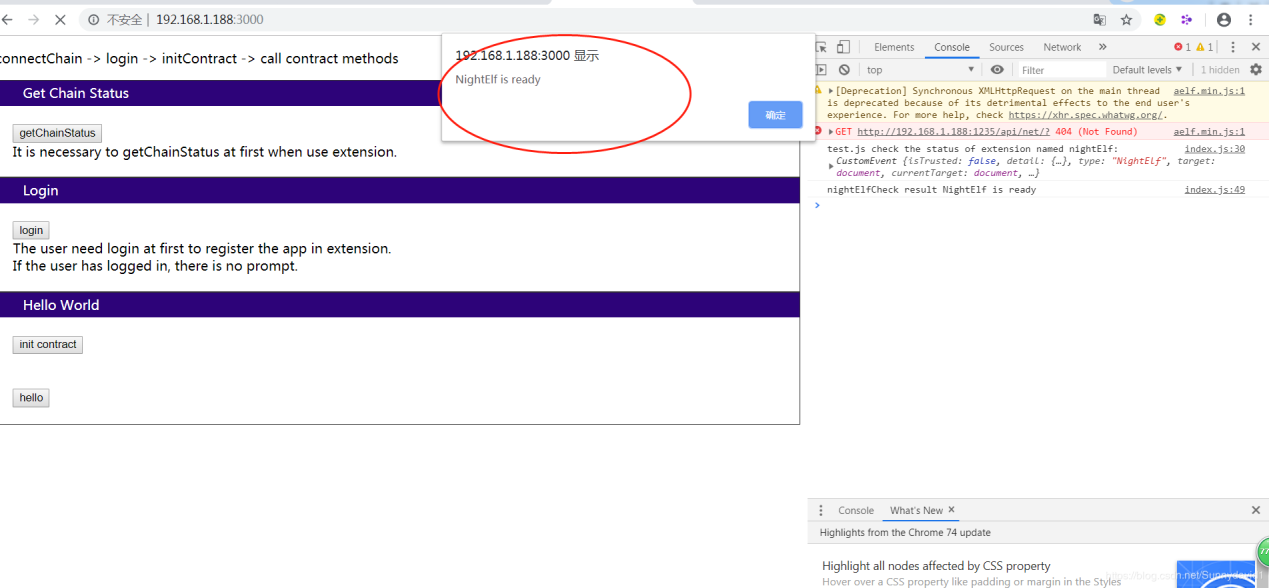
在终端中出现以下窗口：



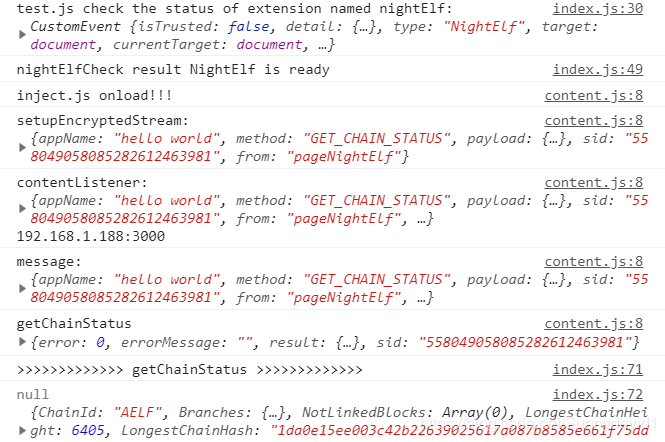
D: 在windows浏览器中输入以下：

  http://IP:3000,  此IP表示linux 服务器的IP地址，我测试的linuxIP地址为192.168.1.188 因此输入： http://192.168.1.188:3000

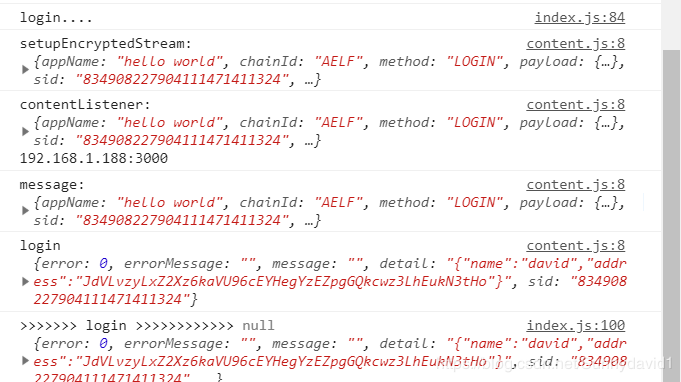
按F12（进入开发者模式），具体如下：



点击getChainStatus出现：



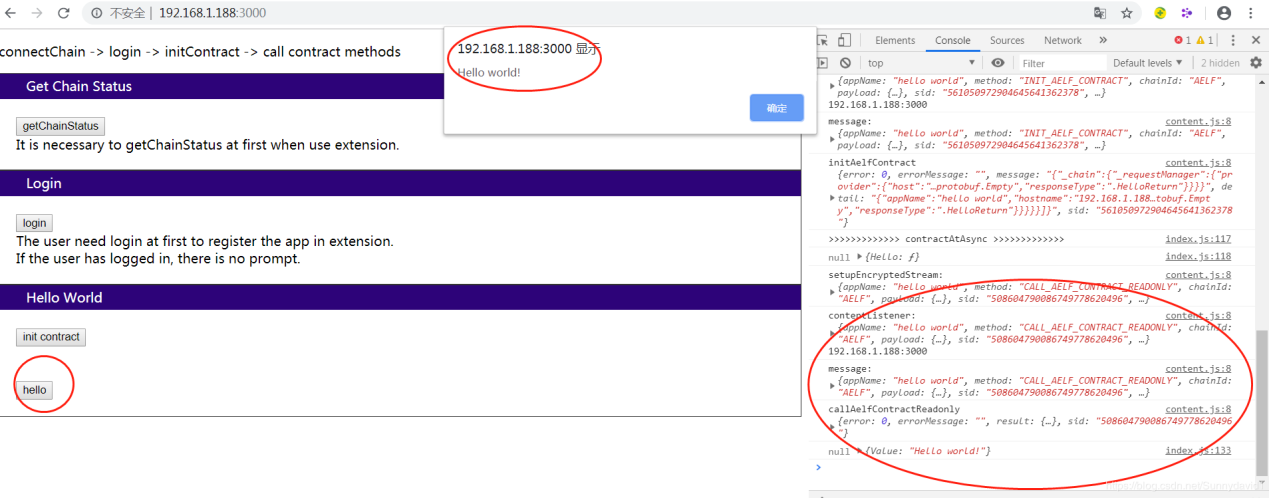
点击login出现：



点击demo init contract出现：



点击hello出现：



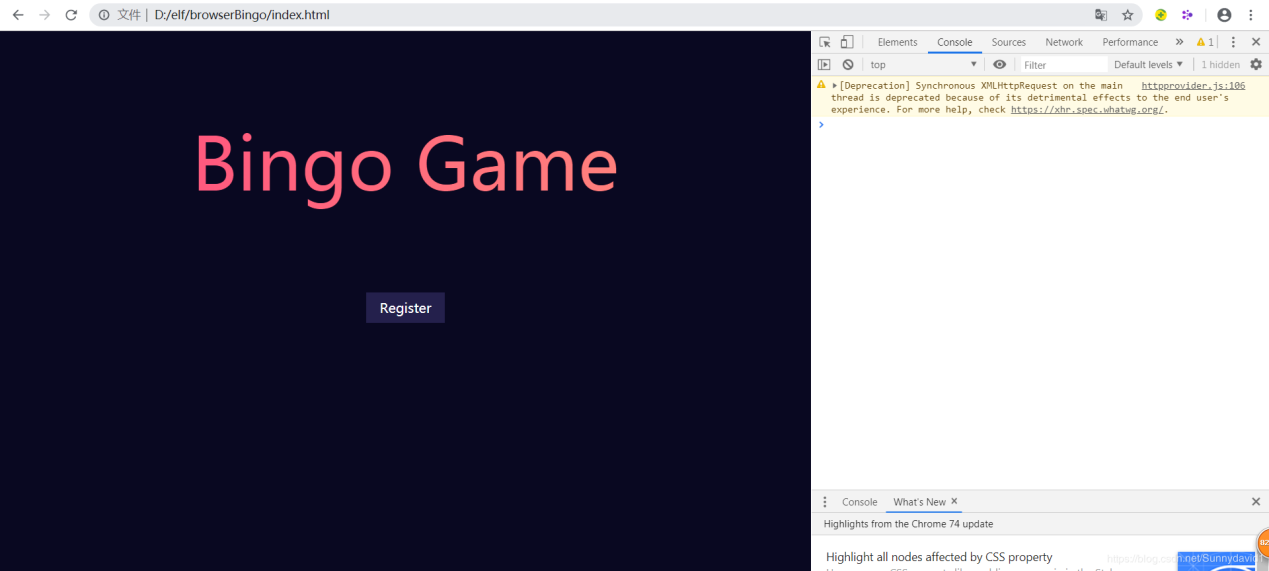
5:Demo4,运行 DAPP Demo——BingoGame:

在linux终端输入以下命令：

cd ../browserBingo/

 修改dist/bingo.bundle.js，搜索文件中的127.0.0.1 ，把此IP改为服务器的Ip地址，保存退出。

 把browseBingo 文件夹复制到windows本地端，因BingoGame是一个客户端程序，所以直接手动打开index.html即可，界面如下：



点击register 初始化

然后点击下注金额，点击play 按照提示进行即可。

那么到了这里，这个4个demo就已经体验完成了。

谢谢大家，有问题可以留言。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

以下是编写智能合约的教程，来自Aelf官方，可参考学习修改：

智能合约编写

aelf 通过类GRPC的protobuf服务描述文件定义智能合约，实现了一个性能等价于GRPC Server的智能合约运行环境。

## Hello World 智能合约讲解

简化目录结构

```bash

.

├── AElf.Boilerplate.sln

├── protobuf

│   ├── hello\_world.proto

├── src

│   ├── AElf.Boilerplate.Launcher

│   │   ├── bin

│   │   │   └── Debug

│   │   │       └── netcoreapp2.2

│   │   │           ├── AElf.Boilerplate.Launcher.dll

│   └── HelloWorldContract

│   │   ├── bin

│   │   │   └── Debug

│   │   │       └── netcoreapp2.2

│   │   │           ├── HelloWorldContract.dll

│       ├── HelloWorldContract.cs

│       ├── HelloWorldContract.csproj

```

### 1.在protobuf/hello\_world.proto中

    通过service 定义了对应的rpc方法，通过message来定义了数据格式。更多的内容可以参考protobuf的文档。

### 2.在src/HelloWorldContract中

    在HelloWorldContract.csproj引用了对应的protobuf文件。

    在HelloWorldContract.cs中实现了具体的合约逻辑。

### 3.src/HelloWorldContract/bin/Debug/netcoreapp2.2/HelloWorldContract.dll

    这个文件是执行 dotnet build后生成的文件。

    在boilerplate这个模板中，dotnet run时会自动发布到我们启动的链上。

    如果需要将自己的合约发布到其它基于AELF系统运行的区块链网络上，只需要发布对应dll即可。

### 4.代码讲解

```protobuf

// protobuf的定义

syntax = "proto3";

import "aelf\_options.proto";

import "google/protobuf/empty.proto";

option csharp\_namespace = "HelloWorldContract";

service HelloWorldContract {

    option (aelf.csharp\_state) = "HelloWorldContractState";

    // 定义了合约的方法，和该方法的返回

    rpc Hello (google.protobuf.Empty) returns (HelloReturn) { }

}

// 定义了返回数据的数据格式

message HelloReturn {

    string Value = 1;

}

```

```C#

// C#中的逻辑实现

using Google.Protobuf.WellKnownTypes;

namespace HelloWorldContract

{

    public partial class HelloWorldContract : HelloWorldContractContainer.HelloWorldContractBase

    {

        // 对应proto文件中的rpc Hello

        // 返回为 HelloReturn

        public override HelloReturn Hello(Empty input)

        {

            return new HelloReturn {Value = "Hello world!"};

        }

}

```

## 新增合约方法教程

下面我们新增一个Add方法，输入两个整型参数 a和b, 输出 a + b

### 1.在hello\_world.proto文件中增加方法的定义和数据类型的定义

```proto

syntax = "proto3";

import "aelf\_options.proto";

import "google/protobuf/empty.proto";

option csharp\_namespace = "HelloWorldContract";

service HelloWorldContract {

    option (aelf.csharp\_state) = "HelloWorldContractState";

    rpc Hello (google.protobuf.Empty) returns (HelloReturn) { }

    // 新增方法Add, 入参为AddInput, 返回为 AddOutput

    rpc Add (AddInput) returns (AddOutput) { }

}

message HelloReturn {

    string Value = 1;

}

// AddInput的格式定义，根据protobuf的协议规定，Key的首字母须大写

message AddInput {

    sint64 A = 1;

    sint64 B = 2;

}

// AddOutpu的定义

message AddOutput {

    sint64 Value = 1;

}

```

### 2.在HelloWorldContract.cs新增逻辑

```C#

using Google.Protobuf.WellKnownTypes;

namespace HelloWorldContract

{

    public partial class HelloWorldContract : HelloWorldContractContainer.HelloWorldContractBase

    {

        public override HelloReturn Hello(Empty input)

        {

            return new HelloReturn {Value = "Hello world!"};

        }

        // 新增Add方法

        public override AddOutput Add(AddInput input) {

            return new AddOutput {Value = input.A + input.B};

        }

    }

}

```

### 3.重新编译并启动链

在src/AElf.Boilerplate.Launcher 下执行 dotnet build 完成编译

dotnet run bin/Debug/netcoreapp2.2/AElf.Boilerplate.Launcher

新的合约会自动发布到链上。

### 4.通过JS文件使用新的合约

```bash

cd web/JSSDK

# 执行node, 进入nodejs的命令行交互界面

node

```

把下面一段js输入

```js

const AElf = require('aelf-sdk');

const Wallet = AElf.wallet;

const sha256 = AElf.utils.sha256;

// address: 65dDNxzcd35jESiidFXN5JV8Z7pCwaFnepuYQToNefSgqk9

const defaultPrivateKey = 'bdb3b39ef4cd18c2697a920eb6d9e8c3cf1a930570beb37d04fb52400092c42b';

const wallet = Wallet.getWalletByPrivateKey(defaultPrivateKey);

const aelf = new AElf(new AElf.providers.HttpProvider(

    'http://127.0.0.1:1235/chain',

    null,

    null,

    null,

    [{

        name: 'Accept',

        value: 'text/plain;v=1.0'

    }]

));

const helloWorldContractName = 'HelloWorldContract';

const {

    GenesisContractAddress

} = aelf.chain.getChainStatus();

const zeroC = aelf.chain.contractAt(GenesisContractAddress, wallet);

const helloWorldContractAddress = zeroC.GetContractAddressByName.call(sha256(helloWorldContractName));

const helloWorldC = aelf.chain.contractAt(helloWorldContractAddress, wallet);

// 刚新增的方法

helloWorldC.Add.call({A: 1, B:3}, (error, result) => {console.log(error, result);})

```

稍等一下看到输出结果

```js

null { Value: '4' }

```

看不到结果

1. 确认编译成功。

2. 确认链在运行。

3. 确认JS代码没输入错。

多学多看多搜索，愿大家在工程师的道路上越走越远。