web3js实践

首先，安装nodejs，我已经安装了，所以不演示了。

创建一个目录，名为web3js-tutorial

执行npm install web3，安装web3js

文本

描述已自动生成

文本

描述已自动生成

web3是一个compiled dependency，compiled dependency的部分代码是用C或C++写的，速度更快。

web3通过连接以太坊节点访问智能合约。web3自身并不知道如何连接以太坊节点，在初始化web3的时候，需要传入一个provider对象，真正和以太坊节点通信的是provider对象，在Dapp的代码中，不需要直接调用provider，只需要调用web3的方法，但在幕后，web3会把方法的执行交给provider，由provider发送API调用给以太坊节点。

图示

描述已自动生成

之所以设计provider的目的是灵活性，因为不需要限制用户连接到区块链的方式，用户可以通过不同的钱包连接到区块链，通常钱包都会暴露一个provider对象作为一个JavaScript全局对象，所以Dapp会测试是否能找到一个provider全局对象，如果找到了，就会用这个provider对象来初始化web3，这样的话，就可以用用户的钱包连接到以太坊节点。

图示

描述已自动生成

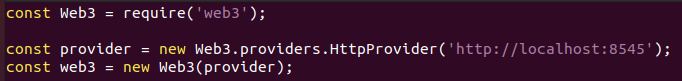
可以在后端nodejs应用中import web3，也可以在前端应用中import web3.

接下来创建一个文件index.js，写下以下两行代码

文本

描述已自动生成

实际上就是引入Web3，并初始化一个web3实例，可以看到，这里并没有显式用到provider，实际上，在幕后，是用到了provider的，provider的类型是HttpProvider，上面的代码等价于下面的代码



上面的代码是直接通过http url连接到以太坊节点。在大部分Dapp中，是直接通过钱包暴露的provider连接到以太坊节点，写法如下

文本

描述已自动生成

我们还是先继续通过http url连接。接下来通过web3实例创建一个contract实例，contract实例可以直接和链上的合约进行交互，这需要两个信息：合约地址，合约abi。abi是一个json文件，描述了合约所有可以从外部被调用的方法。

在创建contract实例之前，需要先通过truffle开发一个智能合约。新创建一个目录web3js-tutorial-contract-instance

执行npx truffle init，初始化truffle项目

文本

描述已自动生成

在contracts目录里创建一个合约MyContract.sol，代码如下

文本

描述已自动生成

执行npx truffle compile，编译合约

文本

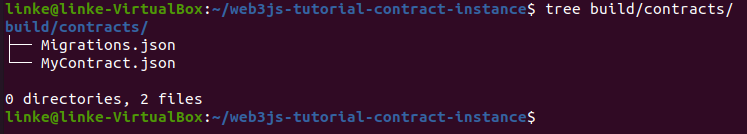
描述已自动生成

在web3js中初始化一个合约的代码如下

文本

描述已自动生成

需要abi和合约地址两个参数。abi文件在truffle compile后可以在build/contracts中找到



MyContract.json里包含如下内容

* contractName
* abi
* metadata
* bytecode
* deployedBytecode
* immutableReferences
* generatedSources
* deployedGeneratedSources
* compiler
* networks（在合约部署之前，这部分的值是空的，什么都没有）
* 其他

可以看到，这个json文件包含abi和编译后的字节码。值得注意的是，abi中不包含private方法。

此时合约编译成功了，但还没有部署，接下来就用npx truffle develop命令启动一个内置的本地ganache网络，develop代表Open a console with a local development blockchain

文本

描述已自动生成

可以看到，启动了一个本地以太坊节点，端口是9545，同时打开了一个truffle的命令行接口。

接下来创建一个MyContract的部署文件

文本

描述已自动生成

内容为

文本

描述已自动生成

接下来执行migrate --reset，把contracts目录中的合约部署到当前的这个本地以太坊节点中。

文本

描述已自动生成

文本

描述已自动生成

--reset表示从头开始运行所有的迁移脚本，而不是从最后完成的开始。

部署完成之后查看build/contracts/MyContract.json的networks部分，内容如下

文本

描述已自动生成

从上图可以看到，合约部署后的地址，所以所在的网络id。

接下来继续丰富web3js的代码，引入abi和合约地址，初始化合约

文本

描述已自动生成

第二行引入build/contracts/MyContract.json很重要，无论MyContract.json里的合约地址怎么变，index.js的代码都不用变，因为是从MyContract.json读取合约地址。

在web3js中初始化了合约之后，接下来就是调用合约了。调用合约有两种方式，一种是call，一种是sendTransaction

图示

描述已自动生成

首先演示call，用call调用MyContract的getData方法，是一个只读方法。在web3js代码中加上两行代码

文本

描述已自动生成

然后执行node index.js

文本

描述已自动生成

getData返回0，正常，默认值。