区块链开发ethers

1.web3.js

https://web3js.readthedocs.io/en/v1.5.2/

以太坊Web3.js库，Web3.js库是一个javascript库，用于与以太坊区块链进行交互。

web3.js 库是一系列模块的集合，服务于以太坊生态系统的各个功能，如：

- web3-eth 用来与以太坊区块链及合约的交互；
- web3-shh Whisper 协议相关，进行p2p通信和广播；
- web3-bzz swarm 协议（去中心化文件存储）相关；
- web3-utils 包含一些对 DApp 开发者有用的方法。

2.ethers.js

我们将使用ethers.js代替web3.js

https://learnblockchain.cn/docs/ethers.js/index.html

ethers.js库旨在为以太坊区块链及其生态系统提供一个小而完整的 JavaScript API 库 它最初是与 ethers.io 一起使用，现在已经扩展为更通用的库。

包含功能

- 将私钥保存在客户端，**安全** 可信赖
- 可支持导入和导出的 **JSON钱包文件** （Geth，Parity和crowdsale）
- 导入和导出 BIP39 **助记词** （12个词备份短语）和 HD（分层确定性）钱包（英语，意大利语，日语，韩语，简体中文，繁体中文; 更多即将推出）
- 从任何合同ABI创建JavaScript 元类对象，包括 **ABIv2** 和 **可读的** ABI
- 支持通过 JSON-RPC， INFURA， Etherscan 或 MetaMask 连接到以太坊节点。
- **ENS名称** 是“一等公民”；它们可以在任何可以使用以太坊地址的地方使用
- 库 **非常小** （压缩~88kb;未压缩284kb）
- 功能 **完整** ，满足所有的以太坊相关需求
- 文档全面： 中文文档 及 documentation
- 大量维护和添加的 **测试用例**
- 完全支持 **TypeScript** 准备好，有定义文件和完整的TypeScript源文件
- 宽松的 **MIT 协议许可** （包括所有依赖）；完全开源可以随意使用

签名方法：

https://learnblockchain.cn/docs/ethers.js/index.html

2.2.1钱包类 Wallet 和 签名器 Signer

Wallet：钱包

Wallet 类管理着一个公私钥对用于在以太坊网络上密码签名交易以及所有权证明。

Wallet 实现了 Signer API，因此可以在任何需要签名器的地方使用 Wallet，它包含了 签名器所有的属性。

签名器（Signer） 接口

Signer API 是一个抽象类，当需要 签名器（Signer） 时就可以扩展实现它（不过是本库还是其他的库）。

Wallet 就是 签名器（Signer） 的一个继承实，以及 JsonRpcSigner 和 Ledger Hardware Wallet Signer 。

提供者 Provider

是一个连接以太坊网络的抽象，用与查询以太坊网络状态或者发送更改状态的交易。

EtherscanProvider 和 **InfuraProvider** 提供连接公开的第三方节点服务提供商，无需自己运行任何以太坊节点。

代码：结合下方web3modal一起展示

3.web3modal

简单说就是连接钱包的。

Web3Modal可帮助开发人员通过简单的配置，为其应用添加钱包支持。通过 Web3Modal 库，支持 Metamask、Dapper、WalletConnect 及其他 Web3 浏览器等，还可以轻松配置库以支持 Fortmatic 和 Portis。

代码：

```
// 连接钱包，获取签名
const web3Modal = new Web3Modal({
        network: "mainnet",
        cacheProvider: true,
});
const connection = await web3Modal.connect();
const provider = new ethers.providers.Web3Provider(connection);
const signer = provider.getSigner();

```

4.开发工具

node@v16.13.1

vue/cli @v4.5.15
vue@2.0
web3@ 1.5.2
ethers@5.5.4
Web3Modal@1.9.5
ipfs-http-client@56.0.1
axios@0.25.0

5.准备工作

该步骤在web3.js初级调用智能合约文档中有详细步骤

5.1准备MetaMask交易账号

Metamask用来管理账户和将当前用户连接到区块链。 MetaMask使用户能够以几种不同的方式管理他们的账户和密钥，同时将密钥与网站环境隔离。
一旦用户连接了MetaMask钱包，作为开发者，你就可以与全局可用的以太坊 API（window.ethereum）进行交互，该API可以识别与web3兼容浏览器的用户（比如MetaMask用户），每当你请求交易签名时，MetaMask都会以尽可能可理解的方式提示用户。

5.2创建vue项目，下载相应开发工具

前端：Vue是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

6.检查钱包

6.1检查是否安装钱包与连接网络

http://cw.hubwiz.com/card/c/metamask-api/1/2/1/

```
 // 引入web3
import Web3 from "web3";

//写在 created()里面
//判断用户是否安装MetaMask钱包插件
if (typeof window.ethereum === "undefined") {
//没安装MetaMask
  console.log("请安装MetaMask钱包")
} else {
```

```
  //如果用户安装了MetaMask，你可以要求他们授权应用登录并获取其账号
  window.ethereum.enable().catch(function (reason) {
   //如果用户拒绝了登录请求
   if (reason === "User rejected provider access") {
      // 用户拒绝登录后执行语句;
      console.log("请授权")
      } else {
      // 本不该执行到这里，但是真到这里了，说明发生了意外
      console.log("There was a problem signing you in");
      }
 }).then(async (accounts) => {
      // 判断是否连接以太或者自定义网络
      //84350是私网的链ID
      if (window.ethereum.networkVersion !== "84350") {
          console.log("请连接xx网络", accounts);
      } else {
      // 获取账号和余额
      const web3 = new Web3(window.web3.currentProvider);
      const [acc] = await web3.eth.getAccounts(function () {});
      web3.eth.getBalance(acc, (err, wei) => {
      //这是MetaMask钱包目前第一个的账户
       let account = acc;
       //这里获取的余额是ETH的，不能获取智能合约上添加的，如需要要从智能合约中调用函数获取
       //余额单位从wei转换为ether
       let balance = web3.utils.fromWei(wei, "ether");
       console.log(account,balance)
      });
    }
  });
}
```

7.页面展示与调用合约

7.1CreateItem页面

页面内容:上传文件到IPFS保存好后上上传至NFT合约

7.1.1上传文件到ipfs

IPFS，它是一种去中心化的文件分发和存储协议。它与现有互联网技术最大的区别就是它对信息的访问不再是以地址的方式进行访问，而是以内容进行访问。
本文介绍的使用的是vue中的elemnet ui插件进行上传的

7.1.2将ipfs上的文件在nft合约上铸造

铸币过程，将自己上传的文件在nft上铸造出来，然后我们通过智能合约就能查看自己铸造的，然后进行选择性的上传到market商城上进行售卖。

7.1.3代码展示

页面代码：

```
<template>
  <div class="createItem">
    <el-form ref="form" :model="form" label-width="9rem">
      <el-form-item label="Asset Name">
        <el-input v-model="form.name" placeholder="Asset Name"></el-input>
      </el-form-item>
      <el-form-item label="Asset Description">
        <el-input
          type="textarea"
          v-model="form.desc"
          placeholder="Asset Description"
        ></el-input>
      </el-form-item>
    </el-form>
    <div class="createItem-Btn">
      <el-upload
        class="img-upload"
        action="tmp"
        :auto-upload="false"
        :limit="1"
        :on-change="handleChange"
        :on-exceed="handleExceed"
      >
        <el-button slot="trigger" size="medium" type="primary"
          >Select Picture</el-button
        >
        <el-button
          style="margin-left: 10px"
```

```
          size="medium"
          type="success"
          @click="onSubmit"
          >Create Digital Asset</el-button
        >
      </el-upload>
    </div>
  </div>
</template>
```

js代码

```
<script>
// 引入相关文件
import { ethers } from "ethers"; //引入ethers.js
import Web3Modal from "web3modal"; //引入Web3Modal
// NFT合约
import NFT from "../../abi/NFT01_ABI.json"; // 引入abi
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
import { create } from "ipfs-http-client";
export default {
  name: "CreateItem",
  // 模板引入
  components: {},
  // 数据
  data() {
    return {
      fileList: [], //图片
      form: {
        name: "", //名字
        desc: "", //描述
        price: "", //价格
      },
    };
  },
  // 方法
  methods: {
    //文件上传状态改变时的钩子
    handleChange(files, fileList) {
      this.fileList = fileList;
    },
```

```
    //文件超出个数限制时的钩子
    handleExceed(files, fileList) {
      this.$message.warning(
        `当前限制选择 1 个文件，本次选择了 ${files.length} 个文件，共选择了 ${
          files.length + fileList.length
        } 个文件`
      );
    },
    //点击上传按钮
    onSubmit() {
      // || !this.form.price
      if (this.fileList.length < 1 || !this.form.name || !this.form.desc) {
        //所有内容必须不为空
        this.$message.warning("请选择需要上传的文件");
      } else {
        const reader = new window.FileReader();
        // console.log(this.fileList[0].raw);
        reader.readAsArrayBuffer(this.fileList[0].raw);
        reader.onloadend = () => {
          let buffer = Buffer(reader.result);
          this.uploadToIPFS(buffer);
        };
      }
    },
    async uploadToIPFS(buffer) {
      //连接ipfs
      const ipfs = create("http://192.168.11.117:5001");
      //将图片上传到ipfs
      let result = await ipfs.add(buffer);
      // 获取到图片的地址
      this.src = `https://ipfs.io/ipfs/${result.path}`;
      let fileUrl = this.src;
      // 将图片地址和图片描述内容转为字符串
      const data = JSON.stringify({
        name: this.form.name,
        description: this.form.desc,
        image: fileUrl,
      });
      try {
        //将描述内容与图片地址再次上传
```

```
        let added = await ipfs.add(data);
        const url = `https://ipfs.io/ipfs/${added.path}`;
        this.$message.warning("上传至ipfs成功");
        // 上传成功后获取参数，上传到nft合约上面。
        this.createSale(url);
      } catch (error) {
        this.$alert("失败", "失败", {
          dangerouslyUseHTMLString: true,
        });
        console.log("Error uploading file: ", error);
      }
    },
    async createSale(url) {
      // console.log(url);
      // 连接钱包，获取签名
      const web3Modal = new Web3Modal();
      const connection = await web3Modal.connect();
      const provider = new ethers.providers.Web3Provider(connection);
      const signer = provider.getSigner();
      /* next, create the item */
      // 连接NFT合约，进行铸币
      let contract = new ethers.Contract(nftaddress, NFT.abi, signer);
      let transaction = await contract.mintOneToken(url);
      // 数据刷新
       await transaction.wait();
      if (transaction) {
        this.$message.warning("上传至NFT成功");
        this.$router.push({
          path: "/index/CreatorDashboard",
        });
      }else{
        this.$alert("上传失败", "失败", {
          dangerouslyUseHTMLString: true,
        });
      }
    },
  },
  // 创建后
  created() {},
  // 挂载后
```

```
  mounted() {},
  // 更新后
  updated() {},
};
</script>
```

7.2CreatorDashboard页面

查看我NFT合约上拥有的所有商品。上传到NFT合约（铸造的）和我购买的。

查看我上传到Market合约（Market市场）的所有商品

查看我上传到Market合约（Market市场）已经出售的

代码:

```
<template>
  <div class="create">
    <!-- 我NFT上拥有的并且未上传到Market商城的列表展示 -->
    <h1>Unsold goods</h1>
    <div class="sell-showBox">
      <div
        class="sell-showBox-one"
        v-for="(item, index) in unsoldGoods"
        :key="index"
      >
        <img :src="item.image" alt="" />
        <h2>
          {{ item.name }}
        </h2>
        <p>{{ item.description }}</p>
        <div class="sell-showBox-one-bottom">
          <h3>
            price:
            <input v-model="item.price" placeholder="price" class="elinput" />
            ETH
          </h3>
          <button @click="uploadMarket(item, price)">upload market</button>
        </div>
      </div>
    </div>
```

```
    <!-- 我上传至Market商城已经出售的列表展示 -->
    <h1>The goods I sell</h1>
    <div class="sell-showBox">
      <div
        class="sell-showBox-one"
        v-for="(item, index) in sellCommodity"
        :key="index"
      >
        <img :src="item.image" alt="" />
        <h2>
          {{ item.name }}
        </h2>
        <p>{{ item.description }}</p>
        <div class="sell-showBox-one-bottom">
          <h3>{{ item.price }} ETH</h3>
          <!-- <button @click="buyNft(item)">Buy</button> -->
        </div>
      </div>
    </div>
    <!-- 我上传至Market商城所有的列表展示 -->
    <h1>My products on the shelves</h1>
    <div class="sell-showBox">
      <div
        class="sell-showBox-one"
        v-for="(item, index) in uploadGoods"
        :key="index"
      >
        <img :src="item.image" alt="" />
        <h2>
          {{ item.name }}
        </h2>
        <p>{{ item.description }}</p>
        <div class="sell-showBox-one-bottom">
          <h3>{{ item.price }} ETH</h3>
          <!-- <button @click="buyNft(item)">Buy</button> -->
        </div>
      </div>
    </div>
  </div>
</template>
```

```
<script>
// 引入相关文件
import axios from "axios";
import { ethers } from "ethers"; //引入ethers.js
import Web3Modal from "web3modal"; //引入web3modal
// NFT合约
import NFT from "../../abi/NFT01_ABI.json"; // 引入abi
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
// Market合约
import Market from "../../abi/Mkt_ABI.json"; // 引入abi
const nftmarketaddress = "0xFC2Ea5A1F3Bed1B545A6be182BF52C20B5e45921";
export default {
  name: "Home",
  // 模板引入
  components: {},
  // 数据
  data() {
    return {
      unsoldGoods: [], //未上架商品
      sellCommodity: [], //出售商品
      uploadGoods: [], //上架商品
      price: "", //价格
    };
  },
  // 方法
  methods: {
    async loadNFTs() {
      //连接账户
      const web3Modal = new Web3Modal({
        network: "mainnet",
        cacheProvider: true,
      });
      const connection = await web3Modal.connect();
      const provider = new ethers.providers.Web3Provider(connection);
      const signer = provider.getSigner();
      //获取传到nft上的所有数据
      const nftContract = new ethers.Contract(nftaddress, NFT.abi, signer);
      const nftdata = await nftContract.fetchMyNFTs();
```

```
      //通过axios获取到ipfs上的数据遍历数据，处理数据
      const nftitems = await Promise.all(
        nftdata.map(async (i) => {
          const meta = await axios.get(i.uri);
          let item = {
            itemId: i.itemId.toNumber(),
            owner: i.owner,
            image: meta.data.image,
            name: meta.data.name,
            description: meta.data.description,
          };
          return item;
        })
      );
      this.unsoldGoods = nftitems;
      //获取传到market上的所有数据
      const marketContract = new ethers.Contract(
        nftmarketaddress,
        Market.abi,
        signer
      );
      //通过nft中的合约的tokenURI方法，获取到ifps上的数据遍历赋值
      const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
      const tokendata = await marketContract.fetchItemsCreated();
      const tokenitems = await Promise.all(
        tokendata.map(async (i) => {
          const tokenUri = await tokenContract.tokenURI(i.tokenId);
          const meta = await axios.get(tokenUri);
          let price = ethers.utils.formatUnits(i.price.toString(), "ether");
          let item = {
            price,
            tokenId: i.tokenId.toNumber(),
            seller: i.seller,
            owner: i.owner,
            sold: i.sold,
            image: meta.data.image,
            name: meta.data.name,
            description: meta.data.description,
          };
          return item;
```

```
        })
      );
      /* create a filtered array of items that have been sold */
      // 通过sold筛选，true表示已经出售
      const soldItems = tokenitems.filter((i) => i.sold);
      // console.log(soldItems);
      // console.log(tokenitems);
      //我上传到ntf上所有的
      this.uploadGoods = tokenitems;
      //我已经出售的
      this.sellCommodity = soldItems;
    },
    //上传到market
    async uploadMarket(item) {
      if (item.price > 0) {
        // 建立连接
        const web3Modal = new Web3Modal({
          network: "mainnet",
          cacheProvider: true,
        });
        const connection = await web3Modal.connect();
        const provider = new ethers.providers.Web3Provider(connection);
        const signer = provider.getSigner();
        // /* then list the item for sale on the marketplace */
        // 拿到相关数据后传入到market合约（市场）上面
        let contract = new ethers.Contract(
          nftmarketaddress,
          Market.abi,
          signer
        );
        // 转换价格单位
        const price = ethers.utils.parseUnits(item.price, "ether");
        let listingPrice = await contract.getListingPrice();
        listingPrice = listingPrice.toString();
        //数据上传market合约上
        let transaction = await contract.createMarketItem(
          nftaddress,
          item.itemId,
          price,
          { value: listingPrice }
```

```
        );
        // console.log(transaction);
        // 数据刷新
        await transaction.wait();
        if (transaction) {
          this.$message.warning("上传market市场成功");
          this.$router.push({
            path: "/index/SellDigitalAsset",
          });
        }
      } else {
        console.log("请输入正确价格");
      }
    },
  },
  // 创建后
  created() {
    this.loadNFTs();
  },
  // 挂载后
  mounted() {},
  // 更新后
  updated() {},
};
</script>
```

7.3SellDigitalAsset页面

查看所有上架到Market商城（合约），可以进行购买

```
<template>
  <div class="sell">
    <!-- 商城商品列表展示 -->
    <h1>Sell Digital Asset</h1>
    <div class="sell-showBox">
      <div
        class="sell-showBox-one"
        v-for="(item, index) in commodity"
        :key="index"
      >
```

```
        <img :src="item.image" alt="" />
        <h2>
          {{ item.name }}
        </h2>
        <p>{{ item.description }}</p>
        <div class="sell-showBox-one-bottom">
          <h3>{{ item.price }} ETH</h3>
          <button @click="buyNft(item)">Buy</button>
        </div>
      </div>
    </div>
  </div>
</template>
```

```
<script>
// 引入相关文件
import axios from "axios";
import { ethers } from "ethers"; //引入ethers.js
import Web3Modal from "web3modal"; //引入web3modal
// console.log(ethers);
// 交易合约
import NFT from "../../abi/NFT01_ABI.json"; // 引入abi
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
// 市场合约
import Market from "../../abi/Mkt_ABI.json"; // 引入abi
const nftmarketaddress = "0xFC2Ea5A1F3Bed1B545A6be182BF52C20B5e45921";


// console.log("tokenContract:", tokenContract);
// console.log("marketContract:", marketContract);
export default {
  name: "sell",
  // 模板引入
  components: {},
  // 数据
  data() {
    return {
      commodity: [], //上架商品信息
```

```
    };
  },
  // 方法
  methods: {
    //调用合约查询数据
    async callContract() {
      // 网关地址
      // const url = "http://192.168.11.120:8545";
      //创立连接
      // const provider = new ethers.providers.JsonRpcProvider(url);
      const provider = new ethers.providers.JsonRpcProvider(
        "http://192.168.11.120:8545"
      );
      // 构建合约与abi
      const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
      const marketContract = new ethers.Contract(
        nftmarketaddress,
        Market.abi,
        provider
      );
      // 获取传到market商城上的所有数据
      const data = await marketContract.fetchMarketItems();
      // 通过nft中的合约的tokenURI方法，获取到ifps上的数据遍历赋值
      const items = await Promise.all(
        data.map(async (i) => {
          //获取tokenid循环查询所有
          const tokenUri = await tokenContract.tokenURI(i.tokenId);
          // console.log("tokenUri",tokenUri);
          //注意，不是axios跨域问题，是地址拼接问题
          let meat1 = tokenUri.replace(/"/gi, "");
          const meta = await axios.get(meat1);
          // 价格单位转换
          let price = ethers.utils.formatUnits(i.price.toString(), "ether");
          let item = {
            price,
            itemId: i.itemId.toNumber(),
            seller: i.seller,
            owner: i.owner,
            image: meta.data.image,
            name: meta.data.name,
```

```
            description: meta.data.description,
          };
          return item;
        })
      );
      // console.log(items);
      this.commodity = items;
    },
    // 去购买
    async buyNft(item) {
      // console.log(item);
      const web3Modal = new Web3Modal();
      const connection = await web3Modal.connect();
      const provider = new ethers.providers.Web3Provider(connection);
      const signer = provider.getSigner();
       //价格转换
      const price = ethers.utils.parseUnits(item.price.toString(), "ether");
      // 使用nft合约进行购买
      const contract = new ethers.Contract(
        nftmarketaddress,
        Market.abi,
        signer
      );
      // console.log(nftaddress, item.itemId, price);
      const transaction = await contract.createMarketSale(
        nftaddress,
        item.itemId,
        {
          value: price,
        }
      );
      // 数据刷新
      await transaction.wait();
      this.callContract();
      if (transaction) {
        this.$message.warning("购买成功");
        this.$router.push({
          path: "/index/MyDigitalAssets",
        });
      } else {
```

```
        this.$alert("购买失败", "失败", {
          dangerouslyUseHTMLString: true,
        });
      }
    },
  },
  // 创建后
  created() {
    this.callContract();
  },
  // 挂载后
  mounted() {},
  // 更新后
  updated() {},
};
</script>
```

7.4MyDigitalAssets页面

查看本账户在Market上购买的所有产品

代码

```
<template>
  <div class="my">
    <!-- 我拥有的商品列表展示 -->
    <h1>My Digital Assets</h1>
    <div class="sell-showBox">
      <div
        class="sell-showBox-one"
        v-for="(item, index) in commodity"
        :key="index"
      >
        <img :src="item.image" alt="" />
        <h2>
          {{ item.name }}
        </h2>
        <p>{{ item.description }}</p>
        <div class="sell-showBox-one-bottom">
          <h3>{{ item.price }} ETH</h3>
          <!-- <button @click="buyNft(item)">Buy</button> -->
```

```
        </div>
      </div>
    </div>
  </div>
</template>
```

```
<script>
// 引入相关文件
// 引入相关文件
import axios from "axios";
import { ethers } from "ethers"; //引入ethers.js
import Web3Modal from "web3modal"; //引入web3modal
// NFT合约
import NFT from "../../abi/NFT01_ABI.json"; // 引入abi
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
// Market合约
import Market from "../../abi/Mkt_ABI.json"; // 引入abi
const nftmarketaddress = "0xFC2Ea5A1F3Bed1B545A6be182BF52C20B5e45921";

export default {
  name: "my",
  // 模板引入
  components: {},
  // 数据
  data() {
    return {
      commodity: [], //所有商品
    };
  },
  // 方法
  methods: {
    async loadNFTs() {
      const web3Modal = new Web3Modal({
        network: "mainnet",
        cacheProvider: true,
      });
      const connection = await web3Modal.connect();
      const provider = new ethers.providers.Web3Provider(connection);
      const signer = provider.getSigner();
```

```
      //通过market合约，查看我购买的所有商品
      const marketContract = new ethers.Contract(
        nftmarketaddress,
        Market.abi,
        signer
      );
      const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
      const data = await marketContract.fetchMyNFTs();
      const items = await Promise.all(
        data.map(async (i) => {
          const tokenUri = await tokenContract.tokenURI(i.tokenId);
          let meat1 = tokenUri.replace(/"/gi, "");
          const meta = await axios.get(meat1);
          let price = ethers.utils.formatUnits(i.price.toString(), "ether");
          let item = {
            price,
            tokenId: i.tokenId.toNumber(),
            seller: i.seller,
            owner: i.owner,
            image: meta.data.image,
            name: meta.data.name,
            description: meta.data.description,
          };
          return item;
        })
      );
      // console.log(items);
      this.commodity = items;
    },
  },
  // 创建后
  created() {
    this.loadNFTs();
  },
  // 挂载后
  mounted() {},
  // 更新后
  updated() {},
};
</script>
```

技术支持：

十二生肖网络科技有限公司

技术部：总监：吕磊

前端：杨加骏，陆俊佑