技术说明

项目背景

本项目创建了一个基于以太坊区块链上的新闻发布智能合约，并构建出web前端与合约进行交互，前端可使用谷歌或火狐浏览器安装matamask后与智能合约展开通信，移动端亦可编译前端后使用DAPP钱包(如：TokenPocket钱包)，在钱包内置浏览器环境打开后进行交互。

智能合约实现了以下功能：

1、发布新闻：新闻发布后，整个数据是储存到智能合约里的，包括新闻标题及内容，发布成功后会返回给交易者一个hash，这个hash可用于溯源。

2、获取新闻：可通过新闻ID或发布者的地址筛选新闻，以及获取全部新闻列表。

3、更新新闻：可通过新闻ID更新已发布新闻的内容，成功提交后智能合约中的数据会被刷新。

4、删除新闻：可通过新闻ID删除已发布的新闻，数据会在智能合约中同步删除。

项目部署

- 部署合约
  在项目根目录执行命令：

```
1、npm install yarn truffle -g
2、yarn
3、truffle develop（将控制台底部的助记词记录一下，等会儿导入metamask）
4、migrate --reset
```

- 安装与配置Metamask
  1. 使用谷歌浏览器安装Metamask插件
  2. 点击“导入现有钱包”按钮，导入truffle develop时生成的助记词
  3. 点击“以太坊主网区域”后，点击显示隐藏测试网络，打开显示测试网络开关
  4. 切换网络为“Localhost:8545”（如果一直转圈圈就刷新浏览器重新选择网络）
  5. metamask账号成功显示99个ETH表示成功
- 运行项目前端
  编译前端，进入client目录，开启新cmd窗口，执行命令：

```
yarn
yarn dev
```

  打开浏览器访问：http://localhost:5173 ，即可浏览项目

项目结构

本项目目录结构如下：

```
├─client
│  ├─src
│  │  ├─pages
│  │  │  └─index
│  │  ├─store
│  │  │  └─modules
│  │  ├─style
│  │  └─utils
│  └─unpackage
├─contracts
└─migrations
```

其中client目录为前端项目

contracts目录为智能合约目录，里面存放了新闻智能合约的源代码

migrations目录为存放合约部署脚本的目录。

项目技术栈

本项目使用了以下技术栈：

1、truffle: 以太坊智能合约开发框架，用于编译、部署和测试智能合约。

2、web3.js: 以太坊智能合约开发框架，用于与区块链进行交互。

3、vue: 前端框架，用于构建前端页面。

4、uniapp-cli: uniapp脚手架，用于构建uniapp项目。

5、pinia: 状态管理库，用于管理项目的状态。

6、vite: 前端打包工具，用于打包项目。

7、tailwindcss: css样式库，用于构建项目的样式。

8、vant: 组件库，用于构建导航栏、按钮等项目组件。

项目实现

1、智能合约分析

新闻智信息结构体如下：

```
// 新闻信息结构体
struct News {
    // ID
    uint id;
    // 标题
    string title;
    // 内容
    string content;
    // 时间戳
    uint timestamp;
    // 发布者
    address publisher;
}
```

定义了新闻的基本要输，包括ID、标题、内容、时间戳和发布者。

发布新闻函数如下：

```
// 发布新闻
function publishNews(string memory _title, string memory _content) public {
    newsMap[latestNewsId] = News({
        id: latestNewsId,
        title: _title,
        content: _content,
        timestamp: block.timestamp,
        publisher: msg.sender
    });

    // 最新新闻ID递增
    latestNewsId++;

    // 总新闻数量递增
    newsCount++;

    // 发出NewsPublished事件
    emit NewsPublished(
        latestNewsId,
        _title,
        _content,
        block.timestamp,
        msg.sender
    );
}
```

发布新闻函数接收两个参数，分别是新闻标题和新闻内容，然后将新闻信息存入newsMap中，最后发出NewsPublished事件。

获取新闻列表函数如下：

```
// 基于演示目的直接全部返回数据，实际应用中不应包含内容部分，应该是返回ID+标题，然后详情页再通过ID获取新闻内容和其他
function getNewsList() public view returns (News[] memory) {
    News[] memory newsList = new News[](newsCount);
    for (uint i = 0; i < newsCount; i++) {
        newsList[i] = newsMap[newsCount - i - 1];
    }
    return newsList;
}
```

通过以上2个函数，我们即可实现最基础的发布新闻和获取新闻列表的功能。新闻发布的数据是储存在区块链智能合约的中的，并没有存到任何数据库上。

2、WEB前端分析

前端项目使用uniapp框架构建，使用pinia作为状态管理库，使用vite作为打包工具，使用tailwindcss作为样式库，使用vant作为组件库。

前端项目目录结构如下：

```
├─pages
│  └─index
├─store
│  └─modules
├─style
└─utils
```

pages目录为页面目录，里面存放了前端项目的页面，一共3个页面，分别是：新闻列表页，新闻详情页，发布新闻页。

store目录为状态管理目录，里面存放了前端项目的状态管理模块，里面封装了vant的全局组件样式、web3实例、合约实例、当前用户钱包数据、以及一些浏览器初始化功能及新闻合约交互等业务逻辑，其他页面只需useAppStore()后即可调用里面的数据和函数。

style目录存放了为tailwindcss引入文件。

utils目录存放了一些工具函数，如时间转换函数等。

项目入口文件为main.js，里面引入了vant的全局组件样式，store等。

```
import {
        createSSRApp
} from "vue"
import App from "./App.vue"
import store from '@/store'
import { ConfigProvider } from 'vant'
// 按需导入需要主动引用样式
import 'vant/es/toast/style'
import 'vant/es/dialog/style'
import 'vant/es/notify/style'
import 'vant/es/image-preview/style'
import '@/style/index.css'

export function createApp() {
        const app = createSSRApp(App)
                .use(store)
                .use(ConfigProvider)
        return {
                app,
        }
}
```

store钱包初始化函数如下：

```
// 钱包初始化
const dappInit = async () => {
    try {
        // 检查环境
        if (typeof window.ethereum == 'undefined') {
            throw { code: -1, message: "请安装metamask或在dapp环境中打开页面" }
        }
        // 监听钱包切换
        window.ethereum.on('accountsChanged', () => {
            window.location.reload();
        });
        // 监听网络切换
        window.ethereum.on('chainChanged', () => {
            window.location.reload()
        })
        // 检查钱包是否解锁
        window.ethereum._metamask.isUnlocked().then((res) => {
            if (!res) throw { code: -1, message: "请解锁您的钱包" }
        })
        // 检查网络
        if (window.ethereum.chainId !== 1337 && window.ethereum.networkVersion !== '1337') {
            throw { code: -1, message: "请切换到Localhost:8575网络" }
        }
        // 获取当前钱包地址
        const accounts = await ethereum.request({ method: 'eth_requestAccounts' })
        address.value = accounts[0]
        isConnect.value = true
        // 初始化web3
        dapp.web3 = new Web3(Web3.givenProvider)
        // 实例化NewsContract合约
        dapp.NewsContract = new dapp.web3.eth.Contract(NewsContract.abi, NewsContract.networks[1337].address
    } catch (error) {
        if (error.code == 4001) error.message = "用户拒绝连接钱包"
        if (error.code == -32002) error.message = "请求已经在等待处理，请耐心等待"
        return showDialog({ message: error.message })
    }
}
```

这个函数在页面加载时会自动执行，用于检查环境、检查钱包是否解锁、检查网络、获取当前钱包地址、初始化web3、实例化NewsContract合约等。

发布新闻函数如下：

```
// 发布新闻
const publishNews = async (data) => {
    try {
        const res = await dapp.NewsContract.methods.publishNews(data.title, data.content).send({from: addres
        console.log(res)
        return showDialog({ message: `发布成功，交易hash: ${res.transactionHash}` }).then(() => {
            uni.navigateTo({
                url: '/pages/index/index'
            })
        });
    } catch (error) {
        return showDialog({ message: error.message })
    }
}
```

通过调用此函数即可发布新闻。

获取新闻列表函数如下：

```
// 获取新闻列表数据
const getNewsList = async () => {
    try {
        const res = await dapp.NewsContract.methods.getNewsList().call()
        dapp.lists = res
        console.log(dapp.lists)
    } catch (error) {
        return showDialog({ message: error.message })
    }
}
```

通过调用此函数即可获取新闻列表数据，不过此函数写法不够完善，因为将内容也一同获取了，当数据量足够大时性能非常差，实际操作时应该只获取新闻ID和标题，当需要查看详情时再通过新闻ID调出新闻内容。

项目验证流程

部署项目 -> 用户打开http://localhost:5173 -> 浏览器metamask弹窗要求用户用钱包签名授权 -> 用户确认授权 -> 登录成功 -> 用户点击发布新闻按钮 -> 用户输入新闻标题和内容 -> 用户点击发布按钮 -> 浏览器metamask弹窗要求用户确认交易 -> 用户支付一定数量ETH后等待交易完成 -> 新闻数据上链成功且返回交易hash -> 浏览器调用智能合约接口输出新闻列表数据 -> 用户可使用hash在truffle中调用，进行新闻溯源

项目总结

项目整体上使用区块链技术完成了新闻发布的溯源，保证了新闻的真实性和不可篡改性。