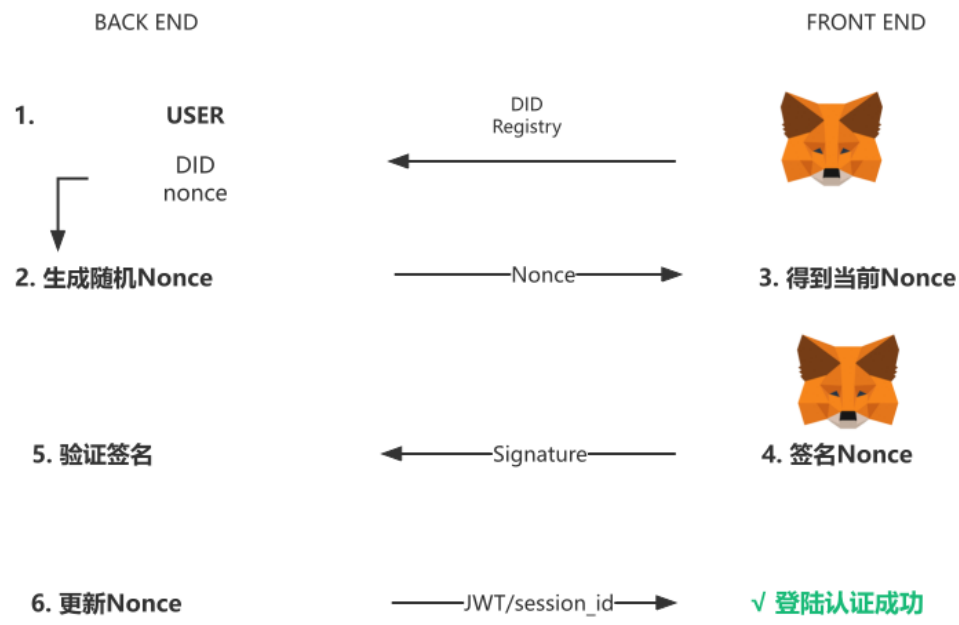
Metamask签名登陆流程

1. 流程
    1. 用户在前端网页上点击登录按钮，并且前端代码与 MetaMask 进行连接。
    2. 前端代码通过调用 `window.ethereum.request({ method: 'eth_requestAccounts' })` 来请求用户连接到 MetaMask。
    3. 后端通过Metamask的masterAccount以及DApp的信息，通过DID Contract查询到相应的用来登陆DID
    4. 前端向后端发送请求以获取与用户钱包地址关联的随机数（nonce）。
    5. 前端调用 `web3.personal.sign(nonce, web3.eth.coinbase, callback)` 来提示 MetaMask 显示签名确认弹窗，其中随机数会显示在弹窗中。
    6. 用户在 MetaMask 中确认签名，并通过回调函数将带有签名的消息（signature）传递给前端。
    7. 前端将签名消息与用户钱包地址一起发送到后端的身份验证接口。
    8. 后端根据用户钱包地址获取对应的随机数（nonce），并使用签名验证算法来验证签名的有效性。
    9. 如果签名验证成功，后端可以进行用户身份认证，并返回给前端一个身份标识符（如 JWT）以进行后续的授权访问。
    10. 为了防止重放攻击，后端会改变下次用户登录时所需的随机数（nonce）。
2. 流程图



3. js代码

```
(async () => {
    try {

        await window.ethereum.enable()
        const provider = new ethers.providers.Web3Provider(window.ethereum)
        const accounts = await provider.send("eth_requestAccounts", []);
```

```
        const defaultAccount = accounts[0]
        console.log(`default account: ${defaultAccount}`)
        //签名
        const signer = await provider.getSigner()
        const msg = web3.utils.asciiToHex("Hello Did!")
        const signature = await signer.signMessage(msg)
        console.log(signature)
        //验证
        const signStatus = verify(defaultAccount, signature, msg);
        console.log(`signStatus: ${signStatus}`)

    } catch (e) {
        console.log(e.message)
    }
  })()

  function verify(address, signature, msg) {
    let signValid = false
    console.log(`address: ${address}`)
    const decodedAddress = ethers.utils.verifyMessage(msg, signature)
    console.log(`decodedAddress: ${decodedAddress}`)
    if (address.toLowerCase() === decodedAddress.toLowerCase()) {
        signValid = true
    }
    return signValid
}
```

sdk

- [x] 封装DID Document格式
- [ ] 提供接口服务

name_service

合约

1. did->名
2. 名->did

- [x] 部署：合约地址 0x0eFCC08cD9831CE3d72c362A5b92011AAD26B23e
- [ ] 测试

其它

1. DID注册表：创建一个DID注册表智能合约，该合约将DID与名称进行映射并存储在区块链上。用户可以通过输入名称查询对应的DID，并进行验证和身份识别。
2. DNS解析：利用现有的域名系统（DNS）来实现DID和名称的映射。通过将DID作为特定域名的子域名，可以通过域名解析查询获得与该DID相关的信息。
3. 去中心化命名服务（Decentralized Naming Service）：创建一个去中心化的命名服务，类似于以太坊的ENS（Ethereum Name Service）或Handshake等。该服务将DID与易记的名称进行映射，并提供去中心化的解析和查询功能。
4. DID别名服务：创建一个DID别名服务，类似于电子邮件中的别名。用户可以将易记的名称与其DID关联，并在通信中使用该别名，而不是复杂的DID。别名服务将负责将别名映射到正确的DID上。

DID_service

- [x] 注册添加了权限，仅限owner

---

github-Metamask签名示例