用正确的姿势开发以太坊系列

【用 Meteor 开发以太坊应用】

功夫小猫

tanzhiguo@cn.ibm.com

计划要写四篇关于以太坊开发的文章，这是第三篇。

在去年的时候，连续写过两篇关于 Meteor 的文章，感兴趣的读者可以查看公众号历史阅读，Meteor 除了之前讲述的一些特性之外，还有一点值得追捧，那就是用 Meteor 做为以太坊应用的开发框架，这也是之前一直把 Meteor 做为研究计划的一个原因，其实不止开发以太坊应用，如果你用 Meteor 来开发超级账本的 Fabric 应用，也是非常理想的选择。

我们先看看以太坊的一位核心开发人员在官方 Wiki 中这样描述 Meteor 框架，他说「5 reasons why Meteor is a perfect fit」

1. 具备两 SPA 开发的一切特征，包括模版引擎、模型、即时编译、绑定
2. live reload，CSS 注入，以及多种预编译，LESS, Coffeescript
3. 只需要一个 html、js、css 就可以打包，随时可以发布到 swarm 上
4. 拥有像 angular 一样的数据绑定特性，以及友好的界面
5. 具备 minimongo 可以持久化存储数据

这里我们用同样完成上一篇文章中讲到的投票应用，不同的是，这次我们用 Meteor。

启动以太坊 Node 或者模拟器的方法，这里不在赘述，您可以参考之前的文章，对于一个智能合约，重要的是，我们需要得到编译后的 bytecode 以及 ABI，正如我们之前谈到的，有多种方式可以完成，这里同样不在赘述，至于部署，也是有很多办法包括通过 web3 方式，或者通过 Geth 方式，这里需要注意的是，每次部署都会生成一个不同的合约地址，那么以太坊环境中已经部署的合约要和你应用使用的地址匹配「前端通过 web3 与合约交互的地址」。

我们用 Meteor 命令行方式创建工程，

```
meteor create voting
```

然后我们通过 Meteor 安装 web3，方法，

```
meteor add ethereum:web3
```

建立这样一个文件，内容如下，然后把文件放到 client/lib 目录下，按照 Meteor 的规则，lib 下的文件会自动执行，那么这里就是自动引用上面安装的 web3，

ethereum-connector.js

```
web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));
console.log(web3.eth.accounts)
```

在同一个目录下，我们再创建一个文件，内容如下，把合约的一些变量以全局变量方式声明，这样在其他页面中就可以调用了，

ethereum-contract.js

```
contractCode = '0x6060604052341561000f57600080fd5b6040516103da3803806103da833981016040528080518
votingContract = web3.eth.contract([{"constant":true,"inputs":[{"name":"candidate","type":"bytes
defaultCandidates = ['Rama', 'Nick', 'Jose'];
votingContractAddress = '0x2f25bcc5c9985a2cd3ec65cec68b6747254b3bf2';
votingContractInstance = votingContract.at(votingContractAddress);
```

接下来我们看 js 页面以及 html 页面，创建了两个模版，代码非常简单而且清晰，这完全是 Meteor 的功劳，

```
Template.candidates.onCreated(function candidatesOnCreated() {
    var candidatesList = [];
    for (var i = 0; i < defaultCandidates.length; i++) {
        var name = defaultCandidates[i];
        var vote = votingContractInstance.totalVotesFor.call(name).toString();
        candidatesList.push({ name: name, vote: vote })
    }
    this.candidates = new ReactiveVar(candidatesList);
});

Template.address.helpers({
    contractAddress() {
        return votingContractAddress;
    },
});

Template.candidates.helpers({
    candidates() {
        return Template.instance().candidates.get();
    }
});

Template.candidates.events({
    'click button' (event, instance) {
        let name = this.name;
        let oldCandidatesList = instance.candidates.get();
        let candidatesList = [];
        votingContractInstance.voteForCandidate(name, { from: web3.eth.accounts[0] }, function() {
            var vote = votingContractInstance.totalVotesFor.call(name).toString();
            for (var i = 0; i < oldCandidatesList.length; i++) {
                if (name == oldCandidatesList[i].name) {
                    candidatesList.push({ name: name, vote: vote })
                } else {
                    var newName = oldCandidatesList[i].name;
                    var newVote = oldCandidatesList[i].vote;
                    candidatesList.push({ name: newName, vote: newVote })
                }
            }
            instance.candidates.set(candidatesList);
        });
    },
});
```

main.html

```
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="
    sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
    <title>A Simple Voting DAPP</title>
</head>

<body class="container" style="text-align: center;margin: 50px auto;width: 500px;">
    <h4>A Simple Voting DAPP</h4> {{> address}}
    <div class="table-responsive">
        {{> candidates}}
    </div>
</body>
<template name="address">
    <p>Smart contract address: <code>{{contractAddress}}</code></p>
</template>
<template name="candidates">
    <table class="table table-bordered">
        <thead>
            <tr>
                <th>Candidate</th>
                <th>Votes</th>
            </tr>
        </thead>
        <tbody>
            {{#each candidates}}
            <tr>
                <td style="text-align: left;">
                    {{name}}
                    <button class="btn btn-secondary btn-sm" style="float: right;">Vote</button>
                </td>
                <td>{{vote}} {{totalvote}}</td>
            </tr>
            {{/each}}
        </tbody>
    </table>
</template>
```

运行之后，效果与之前的一致，

A Simple Voting DAPP

Smart contract address:

0x2f25bcc5c9985a2cd3ec65cec68b6747254b3bf2

Candidate	Votes
Rama Vote	9
Nick Vote	39
Jose Vote	19

当然，你可以结合 Meteor 强大的后台处理，比如跟踪区块的生成，或者实时统计投票状态，这些对 Meteor 来说都是轻而易举的事情。

关于另外一个通用的框架 Truffle，官方有很多 tutorials，其中有一个 Pet shop 的例子，非常典型，感兴趣的朋友可以自己尝试开发，后面就不再写文章详细描述了。

如果您区块链技术感兴趣，请在公众号下回复「blockchain」，我们创建来代码仓库「区块链圣经」，对区块链技术进行知识梳理，也欢迎提交 PR 给我们！