实验六：简单 Dapp 的开发

一、实验概述

DApp（Decentralized Application）去中心化应用，自 P2P 网络出现以来就已经存在，是一种运行在计算机 P2P 网络而不是单个计算机上的应用程序。DApp 以一种不受任何单个实体控制的方式存在于互联网中。在区块链技术产生之前，BitTorrent，Popcorn Time，BitMessage 等都是运行在 P2P 网络上的 DApp，随着区块链技术的产生和发展，DApp 有了全新的载体和更广阔的发展前景。DApp 应具备代码开源、激励机制、非中心化共识和无单点故障四个要素，而最为基本的 DApp 结构即为前端+智能合约形式。

本实验以以太坊为基础，首先用 Solidity 语言编写实现会议报名登记功能的智能合约，加深编写智能合约的能力；之后学习以太坊私有链的搭建、以及合约在私有链上的部署，从而脱离 Remix，学习更为强大的 Truffle 开发组件；进而学习 web3.js 库，实现前端对智能合约的调用，构建完整的 DApp；最后可对该 DApp 加入个性化机制，例如加入 Token 机制等，作为实验选做项。该实验实现了一个简单的 DApp，但包含了 DApp 开发的必备流程，为将来在以太坊上进行应用开发打下了基础。

实验内容概述如下：

A. 编写实现会议报名功能的智能合约（发起会议，注册，报名会议，委托报名，事件触发）

B. 利用 Truffle 套件将合约部署到以太坊私有链（私有链搭建，合约部署，合约测试）

C. 利用 web3.js 实现前端对合约的调用（账户绑定、合约 ABI、RPC 调用）

D. 拓展实验 1：为 DApp 加入 ETH 抵押机制；

拓展实验 2：实现 n-m 门限委托报名机制。

二、预备知识

1. **Solidity 与智能合约：**请参照实验五

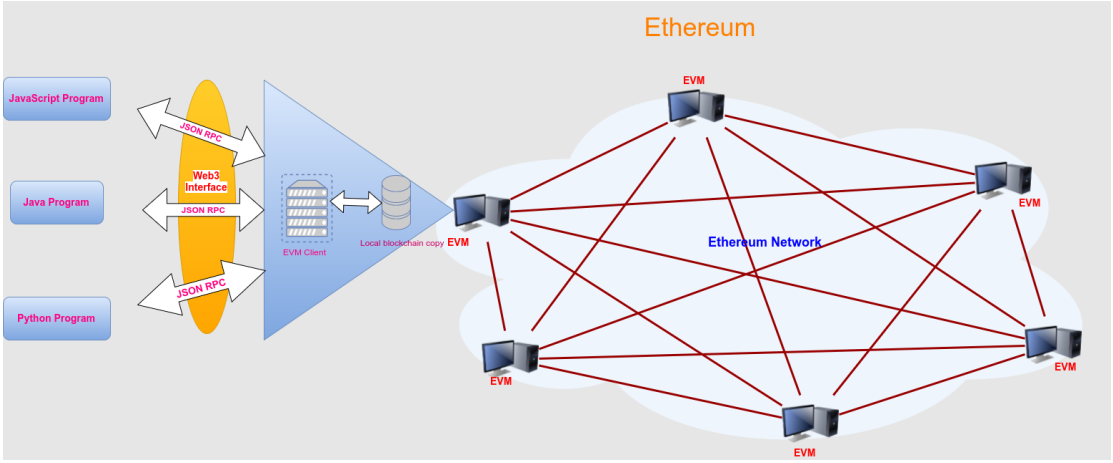
2. **Truffle 组件：**Truffle 针对基于以太坊的 Solidity 语言的一套开发框架。本身基于 Javascript。其对客户端做了深度集成，开发，测试，部署都可以通过一行命令完成。该组件分为三个工具，分别是 Truffle，Ganache 和 Drizzle，本实验需要用到前两者，Truffle 可以自动构建智能合约项目，并简化了智能合约从编写到上线的全部流程；Ganache 则是用于创建以太坊私有链的客户端，从而方便对合约进行链上测试。

Truffle 官网：https://www.trufflesuite.com

官方文档：https://www.trufflesuite.com/docs

中文文档：https://learnblockchain.cn/docs/truffle/index.html

3. **Web3.js：**web3.js 是一个 JavaScript 库，可以使用 HTTP 或 IPC 连接本地或远程以太坊节点进行交互。通过 Web3.js 库与以太坊节点建立连接后，它可以实现检索用户帐户，发送交易，与智能合约交互等功能。



在 DApp 开发中，Web3.js 通常被用于前端与智能合约的交互。它主要包含以下几类 API:

eth：Etherum 区块链相关方法;

net：节点的网络状态;

personal：帐户功能和发送;

db：获取/放置本地 LevelDB;

shh：使用 Whisper 的 P2P 消息传递;

4. **API:** Application Programming Interface，即应用程序接口，规定了运行在一个端系统上的软件请求因特网基础设施向运行在另一个端系统上的特定目的地软件交付数据的方式。API 的使用使得软件系统的职责得到合理划分，有助于提高系统的可维护性和可扩展性。

5. **MetaMask**：MetaMask 是一个开源的以太坊钱包，以浏览器插件的形式运行，用户能够方便地在浏览器中通过该插件连接到以太坊网络中，控制自己的账号进行交易。

三、实验准备

实验系统：任意;

使用软件：最新版本 Chrome 浏览器、nodejs、truffle 以及 ganache;

四、实验内容

实验 1 会议报名登记系统的基本功能与实现。

本节将介绍该实验需要完成的报名系统所具有的基本功能，并指导完成合约部分的编写。

提示：编写合约前可先熟悉下实验 3 提供的前端接口，使合约的数据结构及功能更好地与前端匹配。

a) 系统功能要求：

合约参与方包含一个管理员以及其余参与者，管理员可以发起不止一个会议，并指定会议信息以及总人数。参与者首先需要进行注册，将个人基本信息与以太坊地址相关联，并存储在合约上，之后可进行报名，或委托他人为自己报名。当会议报名人满时，该会议将不再可报名。

当合约内某些数据发生变化时，应能够触发事件（event）使前端重新获取并渲染数据，例如当某个会议报名人满时，应触发相应事件使前端及时更新可报名会议列表。

b) 合约文件名称：Enrollment.sol

合约的成员定义：（仅供参考，只要能满足功能均可）

```
//管理员
address public administrator;
//参与者，必包含姓名，已报名会议列表
struct Participant{}
//会议信息，必须包含会议名、人数上限、是否报满（可否报名）
struct Conference{}
//参与者地址到参与者信息的映射
mapping (address => Participant) participants;
//会议信息列表
```

Conference[] public conferences;

//受托方地址与委托方信息的映射(受托方不一定需要进行注册，且受托

//方受托人数可能不止一人)

mapping (address => Participant[]) trustees;

c) 合约的函数名及实现：（仅供参考，只要能满足功能均可）

//合约构造函数。

constructor(){}

//参与方函数。

//注册报名者信息

function signUp() public {}

//将报名权限委托给其他以太坊地址

function delegate() public {}

//为自己报名

function enroll() public {}

//为委托者报名

function enrollFor() public {}

//管理者函数

//发起新会议。

function newConference() public {}

//销毁合约

function destruct() private {}

//查询函数

//查询可报名会议列表

function queryConfList() public {}

//查询我报名的会议列表

function queryMyConf() public {}

d) 合约事件

//新的会议发布(提示前端更新 ConferenceList 中的可报名会议)

event NewConference()

//会议报名已满(提示前端将 ConferenceList 中已报满会议移除)

event ConferenceExpire();

//用户报名会议(提示前端更新 MyConference 中的会议)

event MyNewConference();

练习 1：1）应在合约的哪个函数指定管理员身份？如何指定?

2）在发起新会议时,如何确定发起者是否为管理员？简述 require()、assert()、revert()的区别。

3）简述合约中用 memory 和 storage 声明变量的区别。

实验 2 学习用 Truffle 组件部署和测试合约。

本节我们学习如何用 truffle 组件对刚才编写的合约进行测试和部署到本地私链上。

2.1 安装 Truffle 和 Ganache

安装 Truffle 需要用到包管理工具 npm，以及结合后续实验需要，首先应安装 nodejs。https://nodejs.org/en/ 在官网下载安装 nodejs，该操作会同时安装好包管理工具 npm。完成后可在终端输入 npm -v 查看 npm 版本。

接下来进行 truffle 的安装：打开系统的终端，输入 npm install truffle -g 即可完成安装,之后可输入 truffle -v 查看版本信息。

下一步安装 Ganache，Ganache 有图形界面和命令行两种版本，命令行支持数据持久化。为了操作方便，本处仅下载图形界面客户端即可，https://www.trufflesuite.com/ganache 下载安装完成后，其界面如图。

2.2 新建 truffle 项目并导入合约

进入任意项目文件夹，在终端输入 truffle init lab8，truffle 会为你初始化一个以太坊项目：

```
Starting init...
================

> Copying project files to lab8

Init successful, sweet!
```

该项目文件结构如下所示：

```
.
├── contracts
│   └── Migrations.sol
├── migrations
│   └── 1_initial_migration.js
├── test
│   └── .gitkeep
└── truffle-config.js
```

其中 contracts 内的 Migrations.sol 用来管理和升级智能合约; migrations 内的 1_initial_migration.js 用来部署 Migrations.sol。这两项不需要配置，由 truffle 自动生成；我们需要在 contracts 中加入我们写的合约，并在 migrations 中写相应的部署脚本。

首先，将我们写好的合约 Enrollment.sol 放入 contracts 文件夹，之后编写 migrations 中的部署脚本，部署脚本文件名为 2_deploy_contracts.js，具体写法如下：

```
const ConvertLib = artifacts.require("ConvertLib");
const Enrollment = artifacts.require("Enrollment");

module.exports = function(deployer) {
  deployer.deploy(ConvertLib);
  deployer.link(ConvertLib, Enrollment);
  deployer.deploy(Enrollment);
};
```

该脚本会先部署库合约 ConverLib，之后将库和报名合约进行链接，然后部署报名合约。

最后配置 truffle-config.js，用于之后配置 Ganache。配置方式如下：

```
networks: {
  // Useful for testing. The `development` name is special - truffle
  // if it's defined here and no other network is specified at the co
  // You should run a client (like ganache-cli, geth or parity) in a
  // tab if you use this network and you must also set the `host`, `p
  // options below to some value.
  //
  development: {
    host: "127.0.0.1",     // Localhost (default: none)
    port: 7545,            // Standard Ethereum port (default: none)
    network_id: "*",       // Any network (default: none)
  },
```

host 和 port 表示之后对 Ganache 进行 RPC 连接的地址和端口。（network 可分为开发环境、测试环境等，本处不作区分，均统一在 Ganache 私链上进行连接）

2.3 为合约编写测试文件

由于智能合约的部署需要花费 gas，并且一经部署将难以修改，因此在部署之前应确保对合约做过单元测试，尽可能减少出错。合约的单元测试文件有两种写法，分别是用 JavaScript 和 Solidity 来实现，通常来说，使用 JavaScript 编写的测试文件更为复杂，但能实现更多功能；用 Solidity 则相对较为简单，为便于学习，本节将指导用 Solidity 编写合约的测试文件。

```
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/Enrollment.sol";
```

首先导入以上合约，除了我们要测试 Enrollment 合约，还有 truffle 提供的 Assert 和 DeployedAddresses 合约。该测试合约的命名须为 TestEnrollment，T 须为大写,如下：

```
contract TestEnrollment {
   function testXXX() public {}
   …
}
```

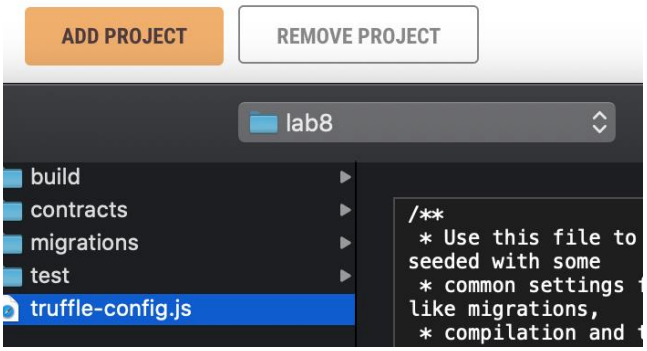
之后需要通过 DeployedAddresses 来获取被部署测试的合约地址：

```
DeployedAddresses.<contract name>();
```

获取到合约地址后，即可编写测试函数，在测试函数内通过合约地址调用合约函数，并通过 Assert 进行测试结果的判断。详细单元测试写法可参考此教程：https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-solidity。

2.4 用 Ganache 搭建私链

打开 Ganache 客户端，选择 NEW WORKSPACE，在 ADD PROJECT 处导入上一步配置的 truffle-config.js 文件，再点击右上角的 save workspace 即可。



此时将会在本地运行一个以太坊私链，并创建 10 个账户供使用，在客户端上方的标签清楚的展示了账户、区块、交易、合约、事件以及日志页，可实时监测该私链的运行情况。

到这里我们已经成功运行了以太坊私链并生成供测试的账户了，下面将部署合约至该链上进行测试。

2.5 对合约进行测试和部署

在终端进入 truffle 项目 lab8 的目录，输入：

truffle test

truffle 会自动编译并在之前用 Ganache 搭建的私链上调用测试合约，输出如下（测试结果根据测试文件写法而不同）：

```
Compiling your contracts...
===========================
> Compiling ./contracts/ConvertLib.sol
> Compiling ./contracts/Enrollment.sol
> Compiling ./contracts/Migrations.sol
> Compiling ./test/TestEnrollment.sol
> Artifacts written to /var/folders/2q/r33l2yrs4mjbqkmx_
08-fYyAluuF2u5f
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang


  TestEnrollment
    ✓ testAdministratorIdentification (182ms)
    ✓ testEnroll (108ms)
    ✓ testDelegate (223ms)


  3 passing (9s)
```

在合约测试完成后，即可进行合约的部署了，部署之前，建议使用 Ganache 重新搭建一条私链，以清除之前进行测试留下的区块和交易记录，便于观察其部署过程。部署合约，只需要在终端输入：

truffle migrate

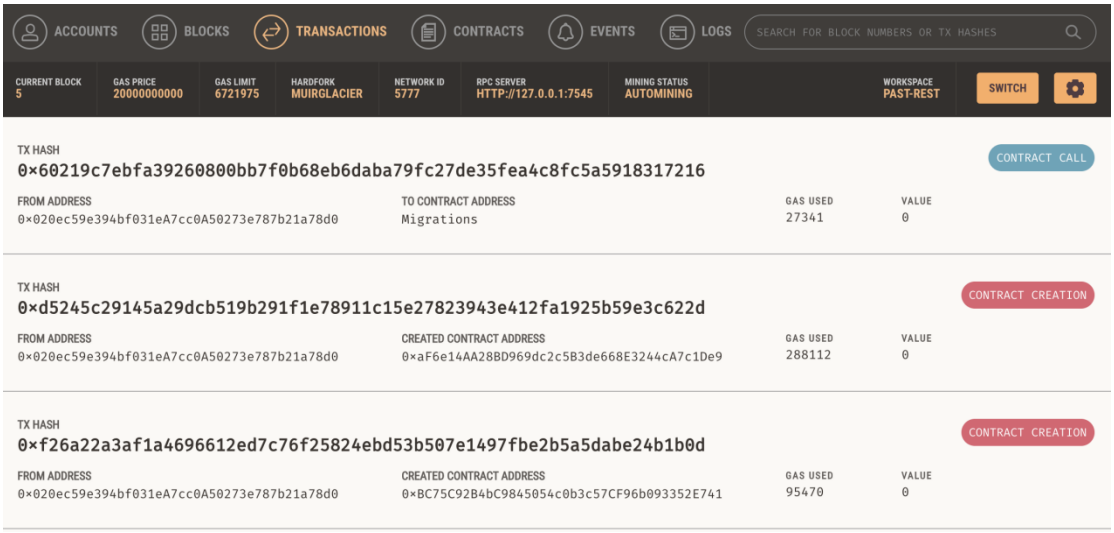
便会自动进行合约的编译和部署。部署完成后，会输出各合约的部署情况以及总共的 gas 花费，Ganache 中的第一个账户即为该合约的部署者：

```
Summary
=======
> Total deployments:   3
> Final cost:          0.01095514 ETH
```

至此，该合约已被测试并部署于以太坊私链上了。在项目文件夹的 build 目录中，会生成一系列 json 格式文件，这些文件将在下一节中起作用。

练习 2：观察合约的部署过程：

请观察部署完成后 Ganache 的 Blocks、Transactions 以及 Logs 记录，完整叙述合约的部署流程以及合约调用。

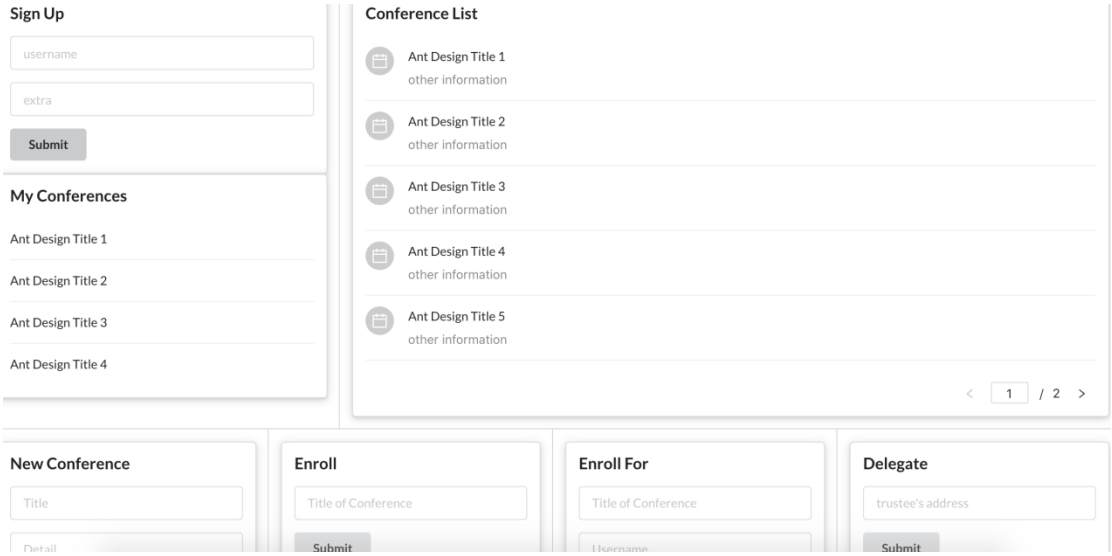


实验 3 利用 Web3.js 实现合约与前端的结合。

前文提到，完整 DApp 的基本组成是智能合约和前端，目前智能合约已成功运行以太坊，接下来将通过 Web3.js 实现二者之间的调用和订阅。

3.1 前端界面接口

本实验已提供了一简单前端界面供使用，为附件中的 lab7-frontend，同学们只需实现 1）通过前端对以太坊节点进行 RPC 调用，执行合约中的函数。2）将合约返回的数据，以及订阅的合约事件提醒及时展示在前端界面。前端界面如下：

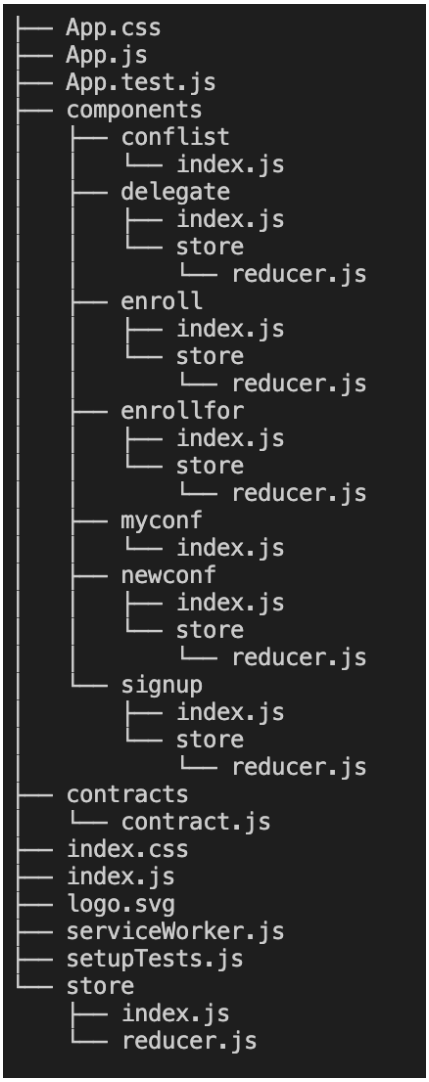


其共有 7 大组件，分别对应注册（将个人信息与地址绑定，存储在合约中）、会议列表（展示目前可报名会议）、我的会议（展示我已报名的会议）、新的会议（合约管理员发起会议）、报名（为自己报名可报名会议）、为委托人报名、委托（将报名权限委托给他人）。

其中注册、新的会议、报名、为委托人报名以及委托这五个组件均为表单，不起数据展示作用，仅在 submit 后提示合约执行成功还是失败；剩下两个组件起数据展示作用，会订阅合约中的事件，当收到事件通知后，会根据通知调用合约内的查询函数（view 类型函数，需自行实现），更新显示数据。

本前端项目采用的 React 框架，该框架的技术特点和数据流处理不在此赘述，

仅需要在某些特定位置编写相关代码。该项目的文件结构如下：



文件夹 components 下包含了上文提到的 7 个组件，需要同学添加代码的部分也仅限于各组件一级目录下的 index.js 文件。

3.2 在前端项目文件中配置合约信息

为了使前端能够通过 Web3 调用合约，前端就必须获取合约的接口信息。智能合约会在编译完成后提供 ABI，即应用二进制接口能提供上述信息。上一节提到的合约编译后，在/build 文件夹下会生成 json 文件，该文件就包含了 ABI 信息，具体格式可参考下图：

```
"abi": [
  {
    "inputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "constant": true,
    "inputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "name": "participants",
    "outputs": [
      {
```

在前端项目的 src/contracts/contract.js 文件中的标注位置粘贴即可。之后，可在 Ganache 的 contracts 中找到部署的 Enrollment 合约，复制合约地址，同样粘贴到 contract.js 的标注位置。之后便可生成 web3 实例，并注入合约信息。

```
// window.web3.currentProvider 为当前浏览器的 web3 Provider
const web3 = new Web3(window.web3.currentProvider);

// 允许连接到 metamask
window.ethereum.enable();

// 导出合约实例
export default new web3.eth.Contract(abi, address);
```

3.3 通过 Web3.js 实现前端与合约交互

本实验用到的 Web3.js 方法主要有如下几种：

web3.eth.Contract: 该对象包含很多方法对合约进行操作的方法，如 methods(为合约方法创建交易)；call(调用合约方法)；send(为调用合约方法发送交易)。还包括 events 方法，可以订阅对应的合约事件，从而让前端及时响应合约数据的变化，进行页面的更新。

web3.eth.accounts:可以获取当前的账户信息等。

关于这些方法的具体使用，可参考如下链接中的中文教程：

http://cw.hubwiz.com/card/c/web3.js-1.0/

下面是对表单类组件（通过 submit 提交数据到合约）的交互写法：

```
const mapDispatchToProps = (dispatch) => {
  return {
    submit(username) {
      contract.methods.signUp() //输入参数
      .send({from:window.web3.eth.accounts[0]},function(err,res){consol
e.log(res)})  //function 中的 res 为方法返回值
      .then((res)=>console.log(res)); //该 res 为交易执行完后的具体交易信
息，如 TxHash 等
      dispatch({
        type: 'submit_signup'
      })
    },
```

请打开 src/components/表单类组件名/index.js，找到下方 mapDispatchToProps 中的 submit 函数，并在 dispatch()之前补充 web3js 代码。如上述代码，contract.methods 后可接合约方法名，创建对应交易，signUp 方法中，参数分别为地址，用户名及额外信息，其中 web3.eth.accounts[0]可查询当前账户列表的第一个以太坊账号地址，而用户名和额外信息则来自于表单中的输入。

由于 signUp 会将信息保存在合约中，所以需要花费 Gas，因此要用 send()方

法指定支付人，本处与注册者保持一致，均为 web3.eth.accounts[0]。

send()执行完后，合约会返回执行结果，本例程仅简单在控制台进行输出（console.log(res)）。

这样就完成了前端的注册组件与合约的交互了。

提示：上述代码的 send 方法有两个返回对象 res，一个是作为 send 方法内的回调函数输入，一个作为整个 send 方法执行完后的输入，前者返回的是合约方法返回值，后者返回的是交易单信息，应进行区分，例如下一步中想从合约中获取展示数据，那么该数据就是第一个返回值。

练习：请参考上述注册组件代码中交互代码的写法，完成另外四个表单类组件对合约的调用。

对于另外两个数据展示类组件（可报名会议列表、本人已报名会议），要求首先能调用合约中的查询类方法，获取会议数据，在前端陈列；之后，当管理员发布新会议、或自己进行报名后，前端应监听到相应的事件，并重新调用查询方法，让前端及时更新。

前者类似于表单类组件，通过 contract.methods 进行调用即可。

问题：这里的调用，应采用 call()方法还是 send()方法?

而后者则需要在组件中设置对订阅的监听。对事件的监听写法如下：

```
  componentDidMount(){
  //先执行一遍查询操作
    contract.events.updateNewConf({
      filter: {}, // Using an array means OR: e.g. 20 or 23
      fromBlock: window.web3.eth.getBlockNumber()
  }, function(error, event){  }) //重新执行查询操作
  .on('data', function(event){
      console.log(event.returnValues); // same results as the optional
callback above
  })
```

```
  .on('changed', function(event){
      // remove event from local database
  })
  .on('error', console.error);
  }
```

ComponentDidMount()是一种生命周期函数，表示该函数内的语句会在该组件加载完成后开始执行。首先应执行一遍查询操作，使页面一加载好就有数据被展示出来；之后调用 contract.events.updateNewConf，表示订阅 updateNewConf 事件，参数中的 fromBlock 设置为从当前区块高度开始订阅，否则会订阅到历史上所有对该事件的触发；当合约中执行了 emit updateNewConf()时会被本代码捕捉到，将事件内容传递到 function(error, event){ }中，而我们则需要在该函数中重新执行查询操作，完成页面的更新。

查询操作需要将查询结果保存在该组件内的 const data 数组内，之后在 List 组件中，将该 data 作为数据源，通过 renderItem 遍历渲染 data 中的数据，如下方代码所示：

```
      <List
        itemLayout="horizontal"
        dataSource={data}
        pagination={{ pageSize: 5, simple: 'true' }}
        renderItem={item => (
          <List.Item>
            <List.Item.Meta
              avatar={<Avatar icon={<CalendarOutlined />} />}
              title={item.title}
              description={item.detail}
            />
          </List.Item>
        )}
      />
```

拓展实验 4 参考该代码，完成 My Conference 和 Conference List 两个组件的订阅和更新功能。

五、参考文献

[1] https://web3js.readthedocs.io/en/v1.5.2/，访问时间：2021.10.19

[2] https://github.com/MetaMask/metamask-extension/releases，访问时间：2021.10.19

[3] https://trufflesuite.com/docs/ganache/，访问时间：2021.10.19

[4] https://trufflesuite.com/docs/truffle/，访问时间：2021.10.19

[5] https://segmentfault.com/a/1190000016608230，访问时间：2021.10.19