web3.js 简介

2018.10

web3.js

- Web3 JavaScript app API
- web3.js 是一个JavaScript API库。要使DApp在以太坊上运行，我们可以使用web3.js库提供的web3对象
- web3.js 通过RPC调用与本地节点通信，它可以用于任何暴露了RPC层的以太坊节点
- web3 包含 eth 对象 - web3.eth（专门与以太坊区块链交互）和 shh 对象 - web3.shh（用于与 Whisper 交互）

web3 模块加载

- 首先需要将 web3 模块安装在项目中：

```
npm install web3@0.20.1
```

- 然后创建一个 web3 实例，设置一个“provider”
- 为了保证我们的 MetaMask 设置好的 provider 不被覆盖掉，在引入 web3 之前我们一般要做当前环境检查（以v0.20.1为例）：

```
if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
} else {
  web3 = new Web3(new Web3.providers
                  .HttpProvider("http://localhost:8545"));
}
```

异步回调（callback）

- web3js API 设计的最初目的，主要是为了和本地 RPC 节点共同使用，所以默认情况下发送的是同步 HTTP 请求
- 如果要发送异步请求，可以在函数的最后一个参数位置上，传入一个回调函数。回调函数是可选（optioanl）的
- 我们一般采用的回调风格是所谓的“错误优先”，例如：

```
web3.eth.getBlock(48, function(error, result){
  if(!error)
        console.log(JSON.stringify(result));
  else
        console.error(error);
});
```

回调 Promise 事件（v1.0.0）

- 为了帮助 web3 集成到不同标准的所有类型项目中，1.0.0 版本提供了多种方式来处理异步函数。大多数的 web3 对象允许将一个回调函数作为最后一个函数参数传入，同时会返回一个 promise 用于链式函数调用。
- 以太坊作为一个区块链系统，一次请求具有不同的结束阶段。为了满足这样的要求，1.0.0 版本将这类函数调用的返回值包成一个“承诺事件”（promiEvent），这是一个 promise 和 EventEmitter 的结合体。
- PromiEvent 的用法就像 promise 一样，另外还加入了.on，.once 和.off方法

```
web3.eth.sendTransaction({from: '0x123...', data: '0x432...'})
.once('transactionHash', function(hash){ ... })
.once('receipt', function(receipt){ ... })
.on('confirmation', function(confNumber, receipt){ ... })
.on('error', function(error){ ... })
.then(function(receipt){ // will be fired once the receipt is mined });
```

应用二进制接口（ABI）

- web3.js 通过以太坊智能合约的 json 接口（Application Binary Interface，ABI）创建一个 JavaScript 对象，用来在 js 代码中描述
- 函数（functions）
  - type：函数类型，默认“function”，也可能是“constructor”
  - constant, payable, stateMutability：函数的状态可变性
  - inputs, outputs: 函数输入、输出参数描述列表
- 事件（events）
  - type：类型，总是“event”
  - inputs：输入对象列表，包括 name、type、indexed

批处理请求（batch requests）

- 批处理请求允许我们将请求排序，然后一起处理它们。
- 注意：批量请求不会更快。实际上，在某些情况下，一次性地发出许多请求会更快，因为请求是异步处理的。
- 批处理请求主要用于确保请求的顺序，并串行处理。

```
var batch = web3.createBatch();
batch.add(web3.eth.getBalance.request('0x0000000000000000
        0000000000000000000000', 'latest', callback));
batch.add(web3.eth.contract(abi).at(address).balance.request(a
        ddress, callback2));

batch.execute();
```

大数处理（big numbers）

- JavaScript 中默认的数字精度较小，所以web3.js 会自动添加一个依赖库 BigNumber，专门用于大数处理
- 对于数值，我们应该习惯把它转换成 BigNumber 对象来处理

```
var balance = new
  BigNumber('131242344353464564564574574567456');

// or var balance = web3.eth.getBalance(someAddress);

balance.plus(21).toString(10);

//"131242344353464564564574574567477"
```

- BigNumber.toString(10) 对小数只保留20位浮点精度。所以推荐的做法是，我们内部总是用 wei 来表示余额（大整数），只有在需要显示给用户看的时候才转换为ether或其它单位

常用 API —— 基本信息查询

查看 web3 版本

- v0.2x.x：web3.version.api
- v1.0.0：web3.version

查看 web3 连接到的节点版本（clientVersion）

- 同步：web3.version.node
- 异步：

  web3.version.getNode((error,result)=>{console.log(result)})
- v1.0.0：web3.eth.getNodeInfo().then(console.log)

基本信息查询

获取 network id

- 同步：web3.version.network
- 异步：web3.version.getNetwork((err, res)=>{console.log(res)})
- v1.0.0：web3.eth.net.getId().then(console.log)

获取节点的以太坊协议版本

- 同步：web3.version.ethereum
- 异步：web3.version.getEthereum((err, res)=>{console.log(res)})
- v1.0.0：web3.eth.getProtocolVersion().then(console.log)

网络状态查询

是否有节点连接/监听，返回true/false

- 同步：web3.isConnect() 或者 web3.net.listening
- 异步：web3.net.getListening((err,res)=>console.log(res))
- v1.0.0：web3.eth.net.isListening().then(console.log)

查看当前连接的peer 节点

- 同步：web3.net.peerCount
- 异步：web3.net.getPeerCount((err,res)=>console.log(res))
- v1.0.0：web3.eth.net.getPeerCount().then(console.log)

Provider

查看当前设置的web3 provider

- web3.currentProvider

查看浏览器环境设置的web3 provider（v1.0.0）

- web3.givenProvider

设置 provider

- web3.setProvider(provider)
  - *web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545'))*

web3 通用工具方法

以太单位转换

- web3.fromWei　　web3.toWei

数据类型转换

- web3.toString　web3.toDecimal　web3.toBigNumber

字符编码转换

- web3.toHex　web3.toAscii　web3.toUtf8　web3.fromUtf8

地址相关

- web3.isAddress　web3.toChecksumAddress

web3.eth – 账户相关

coinbase 查询

- 同步：web3.eth.coinbase
- 异步：web3.eth.getCoinbase((err, res)=>console.log(res))
- v1.0.0：web3.eth.getCoinbase().then(console.log)

账户查询

- 同步：web3.eth.accounts
- 异步：web3.eth.getAccounts((err, res)=>console.log(res))
- v1.0.0：web3.eth.getAccounts().then(console.log)

区块相关

区块高度查询

- 同步：web3.eth. blockNumber
- 异步：web3.eth.getBlockNumber(*callback*)

gasPrice 查询

- 同步：web3.eth.gasPrice
- 异步：web3.eth.getGasPrice(*callback*)

区块相关

区块查询

- 同步：web3.eth.getBlockNumber(*hashStringOrBlockNumber* [,*returnTransactionObjects*])
- 异步：web3.eth.getBlockNumber(*hashStringOrBlockNumber*, *callback*)

块中交易数量查询

- 同步：web3.eth.getBlockTransactionCount(*hashStringOrBlockNumber*)
- 异步：web3.eth.getBlockTransactionCount(*hashStringOrBlockNumber*, *callback*)

交易相关

余额查询

- 同步：web3.eth.getBalance(*addressHexString* [, *defaultBlock*])
- 异步：web3.eth.getBalance(*addressHexString* [, *defaultBlock*]

  [, *callback*])

交易查询

- 同步：web3.eth.getTransaction(*transactionHash*)
- 异步：web3.eth.getTransaction(*transactionHash* [, *callback*])

交易执行相关

- 交易收据查询（已进块）
- 同步：web3.eth.getTransactionReceipt(*hashString*)
- 异步：web3.eth.getTransactionReceipt(*hashString* [, *callback*])

- 估计 gas 消耗量
- 同步：web3.eth.estimateGas(*callObject*)
- 异步：web3.eth.estimateGas(*callObject* [, *callback*])

发送交易

- web3.eth.sendTransaction(*transactionObject* [, *callback*])
- 交易对象：
  - from：发送地址
  - to：接收地址，如果是创建合约交易，可不填
  - value：交易金额，以wei为单位，可选
  - gas：交易消耗 gas 上限，可选
  - gasPrice：交易 gas 单价，可选
  - data：交易携带的字串数据，可选
  - nonce：整数 nonce 值，可选

消息调用

- web3.eth.call(*callObject* [, *defaultBlock*] [, *callback*])
- 参数：
  - 调用对象：与交易对象相同，只是from也是可选的
  - 默认区块：默认“latest”，可以传入指定的区块高度
  - 回调函数，如果没有则为同步调用

```
var result = web3.eth.call({ to:
  "0xc4abd0339eb8d57087278718986382264244252f",
  data:
  "0xc6888fa1000000000000000000000000000000000000000000000000
0              0000000000003" });
console.log(result);
```

日志过滤（事件监听）

- web3.eth.filter(*filterOptions* [, *callback*])

```
// filterString 可以是 'latest' or 'pending'
var filter = web3.eth.filter(filterString);
// 或者可以填入一个日志过滤 options
var filter = web3.eth.filter(options);
// 监听日志变化
filter.watch(function(error, result){ if (!error) console.log(result); });
// 还可以用传入回调函数的方法，立刻开始监听日志
web3.eth.filter(options, function(error, result){
  if (!error) console.log(result);
});
```

合约相关 —— 创建合约

- web3.eth.contract

```
var MyContract = web3.eth.contract(abiArray);

// 通过地址初始化合约实例

var contractInstance = MyContract.at(address);

// 或者部署一个新合约

var contractInstance = MyContract.new([constructorParam1]
                [, constructorParam2], {data: '0x12345...', from:
myAccount,                                  gas: 1000000});
```

调用合约函数

- 可以通过已创建的合约实例，直接调用合约函数

```
// 直接调用，自动按函数类型决定用sendTransaction 还是 call
myContractInstance.myMethod(param1 [, param2, ...] [,
  transactionObject] [, defaultBlock] [, callback]);
// 显式以消息调用形式call 该函数
myContractInstance.myMethod.call(param1 [, param2, ...] [,
  transactionObject] [, defaultBlock] [, callback]);
// 显式以发送交易形式调用该函数
myContractInstance.myMethod.sendTransaction(param1 [,
  param2, ...] [, transactionObject] [, callback]);
```

监听合约事件

- 合约的 event 类似于 filter，可以设置过滤选项来监听

```
var event = myContractInstance.MyEvent({valueA: 23}
        [, additionalFilterObject])
// 监听事件
event.watch(function(error, result){ if (!error) console.log(result); });
//还可以用传入回调函数的方法，立刻开始监听事件
var event = myContractInstance.MyEvent([{valueA: 23}]
        [, additionalFilterObject] , function(error, result){
            if (!error) console.log(result);
        }
);
```

Q&A


尚硅谷