web3.js

The web3.js library is a collection of modules which contain specific functionality for the ethereum ecosystem.

- The `web3-eth` is for the ethereum blockchain and smart contracts
- The `web3-shh` is for the whisper protocol to communicate p2p and broadcast
- The `web3-bzz` is for the swarm protocol, the decentralized file storage
- The `web3-utils` contains useful helper functions for Dapp developers.

web3 provider

http 에서 동작하는 node 에 연결하기 위해 `HttpProvider` 를 사용해 web3 객체를 생성한다.

```
// ropsten
var web3 = new Web3(new Web3.providers.HttpProvider('https://ropsten.infura.io'));
var web3 = new Web3(new Web3.providers.HttpProvider('https://api.myetherapi.com/rop
// mainnet
var web3 = new Web3(new Web3.providers.HttpProvider('https://mainnet.infura.io/'));
var web3 = new Web3(new Web3.providers.HttpProvider('https://api.myetherapi.com/eth
// private(local)
var web3 = new Web3(new Web3.providers.HttpProvider('http://127.0.0.1:8545/'));
```

web3.eth.method call

web3 api 호출은 ethereum 내 state를 조회(Query)하거나 data/value를 전송(transaction)하는 호출로 나누어 볼 수 있다.

Query

state 를 조회하는 경우에는 gas가 소모되지 않기 때문에 web3가 제공하는 Api를 호출하면 된다.

```
// synchronous
var balance = web3.eth.getBalance("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
// Asynchronous
web3.eth.getBalance(address, function(err, result) {
```

```
  balance = result
});
```

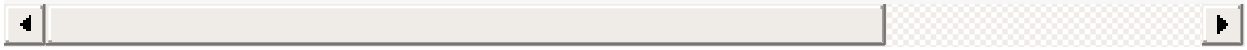
Transaction

web3를 사용해 data/value 를 전송하기 위해서 transactionObject 를 만들어야 한다.

transaction object

transactionObject 는 다음과 같이 구성된다.

```
var transactionObject = {
    nonce: transactionCount, // the count of the number of outgoing transactions, s
    gasPrice: gasPrice, //  the price to determine the amount of ether the transact
    gasLimit: gasLimit, // the maximum gas that is allowed to be spent to process t
    to: toAddress,
    from: ownerAddress,
    data: data, // could be an arbitrary message or function call to a contract or
    value: wei  // the amount of ether to send
};
```



만들어진 transactionObject 를 전송하는 방법은 `sendTransaction` 과 `sendRawTransaction` 으로 나누어 볼 수 있다.
두 방법의 차이점은 서명 방법의 차이로 sendTransaction 은 transactionObject 를 node에 보내면 node 에서 서명을 하고
sendRawTransaction 은 보낼 때 서명을 해서 보낸다. sendTransaction 을 하기 위해서는 node 에 Account 가 등록되어
있어야 하고 sendRawTransaction 하기 위해서는 privateKey 를 알고 있어야 한다.

sendTransaction

transactionObject 을 만들어서 전송한다. gasPrice, nonce, gasLimit 은 web3를 사용해 조회가 가능하다.

```
var nonce = web3.eth.getTransactionCount(fromAddress);
var gasPrice = web3.eth.gasPrice;
var value = web3.toWei(amount, 'ether');
var gasLimit = web3.eth.estimateGas({
    to: toAddress,
    from: fromAddress,
    value: value
```

```
}); // the used gas for the simulated call/transaction (,,21000)
var txObject = {
    nonce: nonce,
    gasPrice: gasPrice,
    gasLimit: gasLimit,
    to: toAddress,
    from: ownerAddress,
    value: value
};
var transactionHash = web3.eth.sendTransaction(txObject);
```

sendRawTransaction

transactionObject 를 만들 때, 파라미터에 해당하는 값들은 Hex 로 변환해서 입력한다.
`ethereumjs-tx` 를 사용해 privateKey transaction 을 서명해서 전송한다.

```
var Tx = require('ethereumjs-tx');

var rawTx = {
    nonce: web3.toHex(nonce), // the count of the number of outgoing transactions,
    gasPrice: web3.toHex(gasPrice), //  the price to determine the amount of ether
    gasLimit: web3.toHex(gasLimit), // the maximum gas that is allowed to be spent
    to: toAddress,
    from: ownerAddress,
    data: '0x00', // could be an arbitrary message or function call to a contract o
    value: web3.toHex(value)  // the amount of ether to send
};

var tx = new Tx(rawTx);
tx.sign(privateKey);

var serializedTx = tx.serialize();
var transactionHash = web3.eth.sendRawTransaction('0x' + serializedTx.toString('hex
```

contract method call

contract 의 method를 호출하는 방법도 state(데이터)를 조회(Query)하거나 데이터저장/연산수행(transaction)하는 호출로 나누어 볼 수 있다.

query

contract의 instance 생성 후, 필요한 state를 함수 호출하듯이 호출한다.

(contract 상에서 외부에서 호출이 가능하도록 구현이 필요하다.)

```
getWhiteListAddress(ruleAddress) {
    var instance = this.web3.eth.contract(this.ruleAbi).at(ruleAddress)
    return new Promise(function (resolve, reject) {
        instance.whitelist(function(err, result) {
            if(err) {
                reject(err);
            } else {
                resolve(result);
            }
        });
    });
}
```

transaction

contract method(데이터저장/연산수행) 를 호출하기 위해서 transaction 을 생성해야 한다.
transactionObject의 `data` parameter 에 method 의 signature 와 method 호출에 필요한 파라미터를 encode 한 값을 입력한다.

```
import CryptoJS from 'crypto-js';
import tx from 'ethereumjs-tx';
import coder from 'web3/lib/solidity/coder';

// signature + parameter
var contractData = contractData(functionName, types, args) {
  var fullName = functionName + '(' + types.join() + ')'
  var signature = CryptoJS.SHA3(fullName,{outputLength:256})
      .toString(CryptoJS.enc.Hex).slice(0, 8)  //  The first 32 bit
  var dataHex = signature + coder.encodeParams(types, args)
  return '0x'+dataHex;
}
var txData = contractData('setWhitelist',['address'], [whiteListAddress]);

var rawTx = {
    nonce: web3.toHex(nonce), // the count of the number of outgoing transactions,
    gasPrice: web3.toHex(gasPrice), //  the price to determine the amount of ether
    gasLimit: web3.toHex(gasLimit), // the maximum gas that is allowed to be spent
    to: toAddress,
    from: ownerAddress,
    data: txData, // could be an arbitrary message or function call to a contract o
};

var tx = new Tx(rawTx);
tx.sign(privateKey);

var serializedTx = tx.serialize();
```

```
var transactionHash = web3.eth.sendRawTransaction('0x' + serializedTx.toString('hex
```

method signature

contract method 의 signature 는 `crypto-js` or `web3` 를 사용해서 signature를 얻을 수 있다.

```
import CryptoJS from 'crypto-js';
var fullName = 'setWhitelist(address)'
var signature = CryptoJS.SHA3(fullName,{outputLength:256})
    .toString(CryptoJS.enc.Hex).slice(0, 8)
// or
// web3 instance가 있을 때
var signature = web3.sha3(fullName).replace('0x', '').slice(0, 8)
```

parameter encode

contract method 호출에 필요한 파라미터는 `coder` 나 `ethereumjs-abi` 를 사용해서 변환할 수 있다.

```
// coder 사용
import coder from 'web3/lib/solidity/coder';
var data = coder.encodeParams(types, args)
// ethereumjs-abi 사용
import abi = from 'ethereumjs-abi'
var data = abi.rawEncode([ "address" ], [whiteListAddress]).toString('hex')
```

contract event

contract 의 method 호출 후, method 에 대한 결과(return)을 얻기 위해서는 contract의 event를 사용한다.
contract 객체로부터 선언된 event 객체를 얻을 수 있고 `watch` 라는 명령을 사용해서
event 를 subcribe 할 수 있다.

```
var event = this.web3.eth.contract(this.ruleAbi).at(ruleAddress).SetWhitelist()
event.watch(function(err, result){
  // result
})
```

result parameter

event 를 subscrbie 하면 받게되는 파라미터는 다음과 같다.
(파라미터 명은 node에 따라 다를 수 있다.)

```
logIndex: Number - integer of the log index position in the block. null when its pe
transactionIndex: Number - integer of the transactions index position log was creat
transactionHash: String, 32 Bytes - hash of the transactions this log was created f
blockHash: String, 32 Bytes - hash of the block where this log was in. null when it
blockNumber: Number - the block number where this log was in. null when its pending
address: String, 32 Bytes - address from which this log originated.
data: String - contains one or more 32 Bytes non-indexed arguments of the log.
topics: Array of Strings - Array of 0 to 4 32 Bytes DATA of indexed log arguments.
type: STRING - pending when the log is pending. mined if log is already mined.
```