导读

以太坊是什么？

以太坊是一个全新开放的区块链平台，它允许任何人在平台中建立和使用通过区块链技术运行的去中心化应用。就像比特币一样，以太坊不受任何人控制，也不归任何人所有——它是一个开放源代码项目，由全球范围内的很多人共同创建。和比特币协议有所不同的是，以太坊的设计十分灵活，极具适应性。在以太坊平台上创立新的应用十分简便，随着 Homestead 的发布，任何人都可以安全地使用该平台上的应用。

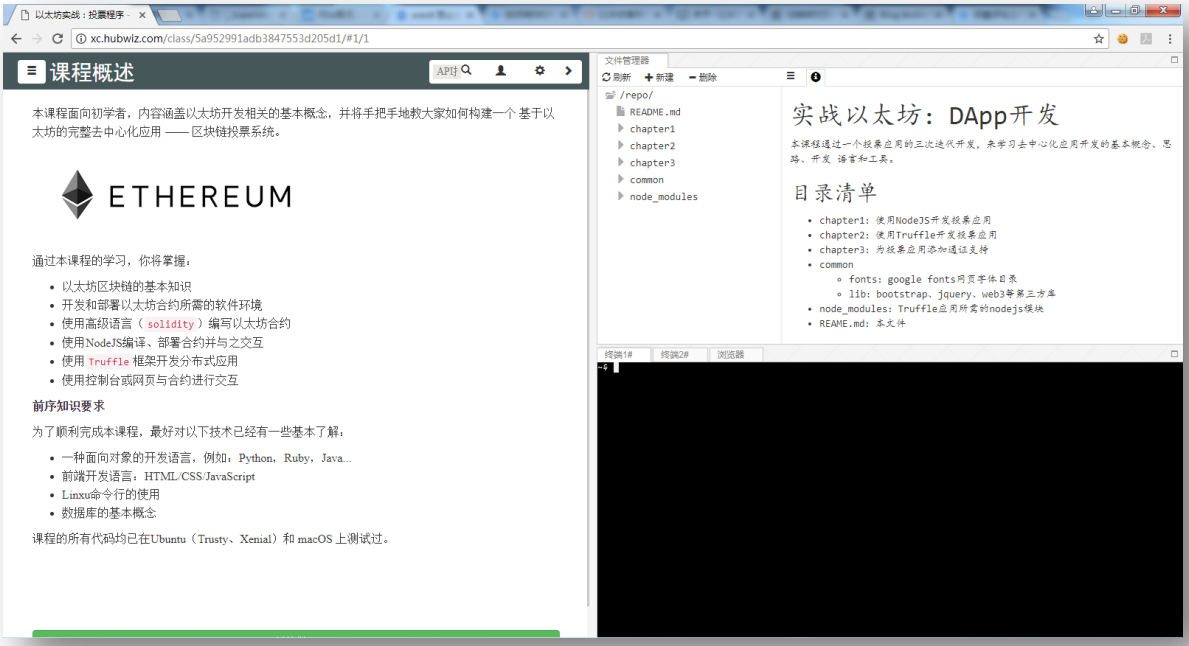
本电子书原文最早发布于区块链技术博客（http:// tryblockchain.org），由汇智网（http://www.hubwiz.com）编目整理，是目前网上流传的最完整的 web3.jsAPI 文档中文版。

但由于以太坊本身（以及周边生态）的发展非常快，一些实践性内容已经落后于现状。因此编者建议本电子书的读者，在阅读时应注意吸收核心的理念思想，而不要过分关注书中的实践操作环节。

为了弥补这一遗憾，汇智网推出了在线交互式以太坊 DApp 实战开发课程，以去中心化投票应用（Voting DApp）为课程项目，通过三次迭代开发过程的详细讲解与在线实践，并且将区块链的理念与去中心化思想贯穿于课程实践过程中，为希望快速入门区块链开发的开发者提供了一个高效的学习与价值提升途径。读者可以通过以下链接访问《以太坊 DApp 开发实战入门》在线教程：

http://xc.hubwiz.com/course/5a952991adb3847553d205d1?affid=w3

教程预置了开发环境。进入教程后，可以在每一个知识点立刻进行同步实践，而不必在开发环境的搭建上浪费时间：



汇智网 Hubwiz.com

2018.2

web3

`web3` 对象提供了所有方法。

示例:

```
//初始化过程
var Web3 = require('web3');

if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
} else {
```

```
  // set the provider you want from Web3.providers
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
}
```

web3.version.api

web3.version.api

返回值：

`String` - 以太坊 js 的 api 版本

示例：

```
//省略初始化过程
var version = web3.version.api;
console.log(version);

$ node test.js
0.18.2
```

web3.version.node

同步方式：

web3.verison.node

异步方式：

web3.version.getNode(callback(error, result){ ... })

返回值：

`String` - 客户端或节点的版本信息

示例：

```
//省略初始化过程
var version = web3.version.node;
console.log(version);

$ node test.js
EthereumJS TestRPC/v3.0.3/ethereum-js
```

web3.version.network

同步方式：

web3.version.network

异步方式：

web3.version.getNetwork(callback(error, result){ ... })

返回值:

`String` - 网络协议版本

示例：

```
//省略初始化过程
var version = web3.version.network;
console.log(version);

$ node test.js
1488546587563
```

web3.version.ethereum

同步方式：

web3.version.ethereum

异步方式：

web3.version.getEthereum(callback(error, result){ ... })

返回值：

`String` - 以太坊的协议版本

示例：

```
//省略初始化过程
var version = web3.version.ethereum;
console.log(version);

$ node test.js
60
```

注意：`EthereumJS testRPC` 客户端不支持这个命令，报错 `Error: Error: RPC method eth_protocolVersion not supported.`

web3.version.whisper

同步方式：

web3.version.whisper

异步方式：

web3.version.getWhisper(callback(error, result){ ... })

返回值：

`String` - `whisper` 协议的版本

示例：

```
//省略初始化过程
var version = web3.version.whisper;
console.log(version);

$ node test.js
20
```

注意：`EthereumJS testRPC` 客户端不支持这个命令，报错 `Error: Error: RPC method shh_version not supported.`

web3.isConnected

web3.isConnected

可以用来检查到节点的连接是否存在（connection to node exist）。

参数：

无

返回值：

Boolean

示例：

```
//省略初始化过程
var connected = web3.isConnected();
if(!connected){
  console.log("node not connected!");
}else{
  console.log("node connected");
}
```

web3.setProvider

web3.setProvider

设置 `Provider`

参数：

无

返回值：

undefined

示例：

```
web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545'));
```

web3.currentProvider

web3.currentProvider

如果已经设置了 `Provider`，则返回当前的 `Provider`。这个方法可以用来检查在

使用 `mist` 浏览器等情况下已经设置过 `Provider`，避免重复设置的情况。

返回值：

Object - null 或 已经设置的 `Provider`。

示例：

```
if(!web3.currentProvider)
    web3.setProvider(new
web3.providers.HttpProvider("http://localhost:8545"));
```

web3.reset

web3.reset

用来重置 `web3` 的状态。重置除了 `manager` 以外的其它所有东西。卸载 `filter`，停止状态轮询。
参数：

1. Boolean - 如果设置为 `true`，将会卸载所有的 `filter`，但会保留 `web3.eth.isSyncing()` 的状态轮询。

返回值：

undefined

示例：

```
//省略初始化过程
console.log("reseting ... ");
web3.reset();
console.log("is connected:" + web3.isConnected());

$ node test.js
reseting ...
is connected:true
```

web3.sha3

web3.sha3(string, options)

参数：

1. `String` - 传入的需要使用 `Keccak-256 SHA3` 算法进行哈希运算的字符串。
2. `Object` - 可选项设置。如果要解析的是 `hex` 格式的十六进制字符串。需要设置 `encoding` 为 `hex`。因为 JS 中会默认忽略 `0x`。

返回值：

`String` - 使用 `Keccak-256 SHA3` 算法哈希过的结果。

示例：

```
//省略初始化过程
var hash = web3.sha3("Some string to be hashed");
console.log(hash);
var hashOfHash = web3.sha3(hash, {encoding: 'hex'});
console.log(hashOfHash);
```

web3.toHex

web3.toHex

将任何值转为 `HEX`。

参数：

1. `String|Number|Object|Array|BigNumber` - 需要转化为 `HEX` 的值。如果是一个对象或数组类型，将会先用 `JSON.stringify`[1]进行转换成字符串。如果传入的是 `BigNumber`[2]，则将得到对应的 `Number` 的 `HEX`。

示例：

```
//初始化基本对象
var Web3 = require('web3');
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
var BigNumber = require('bignumber.js');

var str = "abcABC";
var obj = {abc: 'ABC'};
var bignumber = new BigNumber('12345678901234567890');

var hstr = web3.toHex(str);
var hobj = web3.toHex(obj);
var hbg = web3.toHex(bignumber);

console.log("Hex of Sring:" + hstr);
console.log("Hex of Object:" + hobj);
console.log("Hex of BigNumber:" + hbg);

$ node test.js
Hex of Sring:0x616263414243
Hex of Object:0x7b22616263223a22414243227d
```

```
Hex of BigNumber:0xab54a98ceb1f0ad2
```

web3.toAscii

web3.toAscii(hexString)

将 `HEX` 字符串转为 `ASCII`[3]字符串

参数：

1. `String` - 十六进制字符串。

返回值：

`String` - 给定十六进制字符串对应的 `ASCII` 码值。

示例：

```
var str =
web3.toAscii("0x657468657265756d000000000000000000000000000000000000000000000000");
console.log(str); // "ethereum"
```

web3.fromAscii

web3.fromAscii

将任何的 `ASCII` 码字符串转为 `HEX` 字符串。

参数：

1. `String` - `ASCII` 码字符串
2. `Number` - 返回的字符串字节大小，不够长会自动填充。

返回值：

`String` - 转换后的 `HEX` 字符串。

示例：

```
var str = web3.fromAscii('ethereum');
console.log(str); // "0x657468657265756d"

var str2 = web3.fromAscii('ethereum', 32);
```

```
console.log(str2); //
"0x657468657265756d0000000000000000000000000000000000000000000000000"

$ node test.js
0x657468657265756d
0x657468657265756d
```

备注： 填充 `padding` 功能好像不可用[4]。

web3.toDecimal

web3.toDecimal

将一个十六进制转为一个十进制的数字

参数：

1. `String` - 十六进制字符串

返回：

`Number` - 传入字符串所代表的十六进制值。

示例：

```
var number = web3.toDecimal('0x15');
console.log(number); // 21
```

web3.fromDecimal

web3.fromDecimal

将一个数字，或者字符串形式的数字转为一个十六进制串。

参数：

1. `Number|String` - 数字

返回值：

`String` - 给定数字对应的十六进制表示。

示例：

```
var value = web3.fromDecimal('21');
```

```
console.log(value); // "0x15"
```

web3.fromWei

web3.fromWei(number, 单位)

以太坊货币单位之间的转换。将以 `wei` 为单位的数量，转为下述的单位，可取值如下：

- kwei/ada
- mwei/babbage
- gwei/shannon
- szabo
- finney
- ether
- kether/grand/einstein
- mether
- gether
- tether

参数：

1. `Number|String|BigNumber` - 数字或 `BigNumber`。
2. `String` - 单位字符串

返回值：

`String|BigNumber` - 根据传入参数的不同，分别是字符串形式的字符串，或者是 `BigNumber`。

示例：

```
var value = web3.fromWei('21000000000000', 'finney');
console.log(value); // "0.021"
```

web3.toWei

web3.toWei(number, 单位)

按对应货币转为以 `wei` 为单位。可选择的单位如下：

- kwei/ada
- mwei/babbage

- gwei/shannon
- szabo
- finney
- ether
- kether/grand/einstein
- mether
- gether
- tether

参数：

1. `Number|String|BigNumber` - 数字或 `BigNumber`
2. `String` - 字符串单位

返回值：

`String|BigNumber` - 根据传入参数的不同，分别是字符串形式的字符串，或者是 `BigNumber`。

示例：

```
var value = web3.toWei('1', 'ether');
console.log(value); // "1000000000000000000"
```

web3.toBigNumber

web3.toBigNumber(数字或十六进制字符串)

将给定的数字或十六进制字符串转为 `BigNumber`[5]。

参数：

1. `Number|String` - 数字或十六进制格式的数字

返回值：

`BigNumber` - `BigNumber` 的实例

示例：

```
var value = web3.toBigNumber('200000000000000000000001');
console.log(value); // instanceOf BigNumber
console.log(value.toNumber()); // 2.000000000000002e+23
console.log(value.toString(10)); // '200000000000000000000001'
```

web3.net

web3.net.listening

同步方式：

web3.net.listening

异步方式：

web3.net.getListener(callback(error, result){ ... })

此属性是只读的，表示当前连接的节点，是否正在 `listen` 网络连接与否。`listen` 可以理解为接收。
返回值：

`Boolean` - `true` 表示连接上的节点正在 `listen` 网络请求，否则返回 `false`。

示例：

```
var listening = web3.net.listening;
console.log("client listening: " + listening);

$ node test.js
client listening: true
```

备注： 如果关闭我们要连接的测试节点，会报错 `Error: Invalid JSON RPC response: undefined`。所以这个方法返回的是我们连上节点的 `listen` 状态。

web3.net.peerCount

同步方式：

web3.net.peerCount

异步方式：

web3.net.getPeerCount(callback(error, result){ ... })

属性是只读的，返回连接节点已连上的其它以太坊节点的数量。

返回值：

`Number` - 连接节点连上的其它以太坊节点的数量

示例：

```
var peerCount = web3.net.peerCount;
console.log("Peer count: " + peerCount);

$ node test.js
Peer count: 0
```

web3.eth

包含以太坊区块链相关的方法

示例：

```
var eth = web3.eth;
```

web3.eth.defaultAccount

web3.eth.defaultAccount

默认的地址在使用下述方法时使用，你也可以选择通过指定 `from` 属性，来覆盖这个默认设置。

- web3.eth.sendTransaction()
- web3.eth.call()

默认值为 `undefined`，20 字节大小，任何你有私匙的你自己的地址。

返回值：

`String` - 20 字节的当前设置的默认地址。

示例：

```
console.log("Current default: " + web3.eth.defaultAccount);
web3.eth.defaultAccount = '0x8888f1f195afa192cfee860698584c030f4c9db1';
console.log("Current default: " + web3.eth.defaultAccount);

$ node test.js
Current default: undefined
Current default: 0x8888f1f195afa192cfee860698584c030f4c9db1
```

web3.eth.defaultBlock

web3.eth.defaultBlock

使用下述方法时，会使用默认块设置，你也可以通过传入 `defaultBlock` 来覆盖默认配置。

- web3.eth.getBalance()
- web3.eth.getCode()
- web3.eth.getTransactionCount()
- web3.eth.getStorageAt()
- web3.eth.call()
- contract.myMethod.call()
- contract.myMethod.estimateGas()

可选的块参数，可能下述值中的一个：

- `Number` - 区块号
- `String` - `earliest`，创世块。
- `String` - `latest`，最近刚出的最新块，当前的区块头。
- `String` - `pending`，当前正在 `mine` 的区块，包含正在打包的交易。

默认值是 `latest`

返回值：

`Number|String` - 默认要查状态的区块号。

示例：

```
console.log("defaultBlock: " + web3.eth.defaultBlock);
web3.eth.defaultBlock = 231;
console.log("defaultBlock: " + web3.eth.defaultBlock);

$ node test.js
defaultBlock: latest
defaultBlock: 231
```

web3.eth.syncing

同步方式：

web3.eth.syncing

异步方式：

web3.eth.getSyncing(callback(error, result){ ... })

这个属性是只读的。如果正在同步，返回同步对象。否则返回 `false`。

返回值：

`Object|Boolean` - 如果正在同步，返回含下面属性的同步对象。否则返回 `false`。

返回值：

- `startingBlock`: `Number` - 同步开始区块号
- `currentBlock`: `Number` - 节点当前正在同步的区块号
- `highestBlock`: `Number` - 预估要同步到的区块

```
var sync = web3.eth.syncing;
console.log(sync);

$ node test.js
false
//正在 sync 的情况
$ node test.js
{
   startingBlock: 300,
   currentBlock: 312,
   highestBlock: 512
}
```

web3.eth.isSyncing

web3.eth.isSyncing(callback)

提供同步开始，更新，停止的回调函数方法。

返回值：

`Object` - 一个 `syncing` 对象，有下述方法：

- `syncing.addCallback()`：增加另一个回调函数，在节点开始或停止调用时进行调用。
- `syncing.stopWatching()`：停止同步回调。

回调返回值：

- `Boolean` - 同步开始时，此值为 `true`，同步停止时此回调值为 `false`。
- `Object` - 当正在同步时，会返回同步对象。
  - `startingBlock`：`Number` - 同步开始区块号
  - `currentBlock`：`Number` - 节点当前正在同步的区块号
  - `highestBlock`：`Number` - 预估要同步到的区块

示例：

```
//初始化基本对象
var Web3 = require('web3');
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
var BigNumber = require('bignumber.js');


web3.eth.isSyncing(function(error, sync){
    if(!error) {
        // stop all app activity
        if(sync === true) {
          // we use `true`, so it stops all filters, but not the web3.eth.syncing polling
          web3.reset(true);

        // show sync info
        } else if(sync) {
          console.log(sync.currentBlock);

        // re-gain app operation
        } else {
            // run your app init function...
        }
    }
});
```

web3.eth.coinbase

同步方式：

web3.eth.coinbase

异步方式：

web3.eth.getCoinbase(callback(error, result){ ... })

只读属性，节点配置的，如果挖矿成功奖励的地址。

返回值：

`String` - 节点的挖矿奖励地址。

示例：

```
var coinbase = web3.eth.coinbase;
console.log(coinbase); // "0x407d73d8a49eeb85d32cf465507dd71d507100c1"
```

web3.eth.mining

同步方式：

web3.eth.mining

异步方式：

web3.eth.getMining(callback(error, result){ ... })

属性只读，表示该节点是否配置挖矿。

返回值：

`Boolean` - `true` 表示配置挖矿，否则表示没有。

```
var mining = web3.eth.mining;
console.log(mining); // true or false
```

web3.eth.hashrate

同步方式：

web3.eth.hashrate

异步方式：

web3.eth.getHashrate(callback(error, result){ ... })

属性只读，表示的是当前的每秒的哈希难度。

返回值：

`Number` - 每秒的哈希数

示例：

```
var hashrate = web3.eth.hashrate;
console.log(hashrate);
```

web3.eth.gasPrice

同步方式：

web3.eth.gasPrice

异步方式：

web3.eth.getGasPrice(callback(error, result){ ... })

属性是只读的，返回当前的 gas 价格。这个值由最近几个块的 gas 价格的中值[6]决定。

返回值：

`BigNumber` - 当前的 gas 价格的 `BigNumber` 实例，以 `wei` 为单位。

```
var gasPrice = web3.eth.gasPrice;
console.log(gasPrice.toString(10)); // "10000000000000"
```

web3.eth.accounts

同步方式：

web3.eth.accounts

异步方式：

web3.eth.getAccounts(callback(error, result){ ... })

只读属性，返回当前节点持有的帐户列表。

返回值：

`Array` - 节点持有的帐户列表。

示例：

```
var accounts = web3.eth.accounts;
console.log(accounts);
```

web3.eth.blockNumber

同步方式：

web3.eth.blockNumber

异步方式：

web3.eth.getBlockNumber(callback(error, result){ ... })

属性只读，返回当前区块号。

```
var number = web3.eth.blockNumber;
console.log(number); // 2744
```

web3.eth.register

web3.eth.register(addressHexString [, callback])

（暂未实现）将给定地址注册到 `web3.eth.accounts`。这将允许无私匙的帐户，如合约被关联到有私匙的帐户，如合约钱包。
参数：

- `String` - 要注册的地址。
- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

待确定

示例：

```
web3.eth.register("0x407d73d8a49eeb85d32cf465507dd71d507100ca")
```

web3.eth.unRegister

异步方式

web3.eth.unRegister(addressHexString [, callback])

（暂未实现）取消注册给定地址

参数：

- `String` - 要取消注册的地址
- `Function` - （可选） 回调函数，用于支持异步的方式执行 [7]。

返回值：

待确定

示例：

```
web3.eth.unRegister("0x407d73d8a49eeb85d32cf465507dd71d507100ca")
```

web3.eth.getBalance

web3.eth.getBalance(addressHexString [, defaultBlock] [, callback])

获得在指定区块时给定地址的余额。

参数：

- `String` - 要查询余额的地址。
- `Number|String` -（可选）如果不设置此值使用 `web3.eth.defaultBlock` 设定的块，否则使用指定的块。
- `Funciton` - （可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 一个包含给定地址的当前余额的 `BigNumber` 实例，单位为 `wei`。

示例：

```
var balance =
web3.eth.getBalance("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
```

```
console.log(balance); // instanceof BigNumber
console.log(balance.toString(10)); // '1000000000000'
console.log(balance.toNumber()); // 1000000000000
```

web3.eth.getStorageAt

web3.eth.getStorageAt(addressHexString, position [, defaultBlock] [, callback])

获得某个地址指定位置的存储的状态值。

合约由控制执行的 EVM 字节码和用来保存状态的 `Storage` 两部分组成。`Storage` 在区块链上是以均为 32 字节的键，值对的形式进行存储 [8]。
参数：

- `String` - 要获得存储的地址。
- `Number` - 要获得的存储的序号
- `Number|String` -（可选）如果未传递参数，默认使用 `web3.eth.defaultBlock` 定义的块，否则使用指定区块。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 给定位置的存储值

示例：

```
var state =
web3.eth.getStorageAt("0x407d73d8a49eeb85d32cf465507dd71d507100c1", 0);
console.log(state); // "0x03"
```

web3.eth.getCode

web3.eth.getCode(addressHexString [, defaultBlock] [, callback])

获取指定地址的代码

参数：

- `String` - 要获得代码的地址。

- `Number|String` -（可选）如果未传递参数，默认使用 `web3.eth.defaultBlock` 定义的块，否则使用指定区块。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 给定地址合约编译后的字节代码。

示例：

```
var code = web3.eth.getCode("0xd5677cf67b5aa051bb40496e68ad359eb97cfbf8");
console.log(code); //
"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000
f25b600060078202905091905056"
```

web3.eth.getBlock

web3.eth.getBlock(blockHashOrBlockNumber [, returnTransactionObjects] [, callback])

返回块号或区块哈希值所对应的区块

参数：

- `Number|String` -（可选）如果未传递参数，默认使用 `web3.eth.defaultBlock` 定义的块，否则使用指定区块。
- `Boolean` -（可选）默认值为 `false`。`true` 会将区块包含的所有交易作为对象返回。否则只返回交易的哈希。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值 - 区块对象：

- `Number` - 区块号。当这个区块处于 `pending` 将会返回 `null`。
- `hash` - 字符串，区块的哈希串。当这个区块处于 `pending` 将会返回 `null`。
- `parentHash` - 字符串，32 字节的父区块的哈希值。

- `nonce` - 字符串，8 字节。POW 生成的哈希。当这个区块处于 `pending` 将会返回 `null`。
- `sha3Uncles` - 字符串，32 字节。叔区块的哈希值。
- `logsBloom` - 字符串，区块日志的布隆过滤器[9]。当这个区块处于 `pending` 将会返回 `null`。
- `transactionsRoot` - 字符串，32 字节，区块的交易前缀树的根。
- `stateRoot` - 字符串，32 字节。区块的最终状态前缀树的根。
- `miner` - 字符串，20 字节。这个区块获得奖励的矿工。
- `difficulty` - `BigNumber` 类型。当前块的难度，整数。
- `totalDifficulty` - `BigNumber` 类型。区块链到当前块的总难度，整数。
- `extraData` - 字符串。当前块的 `extra data` 字段。
- `size` - `Number`。当前这个块的字节大小。
- gasLimit - `Number`，当前区块允许使用的最大 `gas`。
- `gasUsed` - 当前区块累计使用的总的 `gas`。
- `timestamp` - `Number`。区块打包时的 `unix` 时间戳。
- `transactions` - 数组。交易对象。或者是 32 字节的交易哈希。
- `uncles` - 数组。叔哈希的数组。

示例：

```
var info = web3.eth.getBlock(3150);
console.log(info);
/*
{
  "number": 3,
  "hash":
"0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "parentHash":
"0x2302e1c0b972d00932deb5dab9eb2982f570597d9d42504c05d9c2147eaf9c88",
```

```
  "nonce": "0xfb6e1a62d119228b",
  "sha3Uncles":
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  "logsBloom":
"0x000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000",
  "transactionsRoot":
"0x3a1b03875115b79539e5bd33fb00d8f7b7cd61929d5a3c574f507b8acf415bee",
  "stateRoot":
"0xf1133199d44695dfa8fd1bcfe424d82854b5cebef75bddd7e40ea94cda515bcb",
  "miner": "0x8888f1f195afa192cfee860698584c030f4c9db1",
  "difficulty": BigNumber,
  "totalDifficulty": BigNumber,
  "size": 616,
  "extraData": "0x",
  "gasLimit": 3141592,
  "gasUsed": 21662,
  "timestamp": 1429287689,
  "transactions": [
    "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b"
  ],
  "uncles": []
}
*/
```

web3.eth.getBlockTransactionCount

web3.eth.getBlockTransactionCount(hashStringOrBlockNumber [, callback])

返回指定区块的交易数量。

参数：

- `Number|String` - （可选）如果未传递参数，默认使用 `web3.eth.defaultBlock` 定义的块，否则使用指定区块。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`Nubmer` - 给定区块的交易数量。

示例：

```
var number =
web3.eth.getBlockTransactionCount("0x407d73d8a49eeb85d32cf465507dd71d507100
c1");
console.log(number); // 1
```

web3.eth.getUncle

web3.eth.getUncle(blockHashStringOrNumber, uncleNumber [, returnTransactionObjects] [, callback])

通过指定叔位置，返回指定叔块。

参数：

- `Number|String` -（可选）如果未传递参数，默认使用 `web3.eth.defaultBlock` 定义的块，否则使用指定区块。
- `Number` - 叔的序号。
- `Boolean` -（可选）默认值为 `false`。`true` 会将区块包含的所有交易作为对象返回。否则只返回交易的哈希。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`Object` - 返回的叔块。返回值参考 `web3.eth.getBlock()`。

备注： 叔块没有自己的交易数据。

示例：

```
var uncle = web3.eth.getUncle(500, 0);
console.log(uncle); // see web3.eth.getBlock
```

web3.eth.getTransaction

web3.eth.getTransaction(transactionHash [, callback])

返回匹配指定交易哈希值的交易。

参数：

- `String` - 交易的哈希值。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`Object` - 一个交易对象

- `hash`: `String` - 32 字节，交易的哈希值。
- `nonce`: `Number` - 交易的发起者在之前进行过的交易数量。
- `blockHash`: `String` - 32 字节。交易所在区块的哈希值。当这个区块处于 `pending` 将会返回 `null`。
- `blockNumber`: `Number` - 交易所在区块的块号。当这个区块处于 `pending` 将会返回 `null`。
- `transactionIndex`: `Number` - 整数。交易在区块中的序号。当这个区块处于 `pending` 将会返回 `null`。
- `from`: `String` - 20 字节，交易发起者的地址。
- `to`: `String` - 20 字节，交易接收者的地址。当这个区块处于 `pending` 将会返回 `null`。
- `value`: `BigNumber` - 交易附带的货币量，单位为 `Wei`。
- `gasPrice`: `BigNumber` - 交易发起者配置的 `gas` 价格，单位是 `wei`。
- `gas`: `Number` - 交易发起者提供的 `gas`。.
- `input`: `String` - 交易附带的数据。

示例：

```
var blockNumber = 668;
var indexOfTransaction = 0

var transaction = web3.eth.getTransaction(blockNumber, indexOfTransaction);
console.log(transaction);
/*
{
  "hash":
"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
  "nonce": 2,
  "blockHash":
"0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "transactionIndex": 0,
  "from": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
  "to": "0x6295ee1b4f6dd65047762f924ecd367c17eabf8f",
  "value": BigNumber,
  "gas": 314159,
  "gasPrice": BigNumber,
  "input": "0x57cb2fc4"
}
*/
```

web3.eth.getTransactionFromBlock

```
getTransactionFromBlock(hashStringOrNumber, indexNumber [, callback])
```

返回指定区块的指定序号的交易。

参数：

- `String` - 区块号或哈希。或者是 `earliest`， `latest` 或 `pending`。查看 `web3.eth.defaultBlock` 了解可选值。
- `Number` - 交易的序号。
- `Function` - 回调函数，用于支持异步的方式执行[7]。

返回值：

`Object` - 交易对象，详见 `web3.eth.getTransaction`

示例：

```
var transaction = web3.eth.getTransactionFromBlock('0x4534534534', 2);
console.log(transaction); // see web3.eth.getTransaction
```

web3.eth.getTransactionReceipt

web3.eth.getTransactionReceipt(hashString [, callback])

通过一个交易哈希，返回一个交易的收据。

备注：处于 `pending` 状态的交易，收据是不可用的。

参数：

- `String` - 交易的哈希
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`Object` - 交易的收据对象，如果找不到返回 `null`

- `blockHash`: `String` - 32 字节，这个交易所在区块的哈希。
- `blockNumber`: `Number` - 交易所在区块的块号。
- `transactionHash`: `String` - 32 字节，交易的哈希值。
- `transactionIndex`: `Number` - 交易在区块里面的序号，整数。
- `from`: `String` - 20 字节，交易发送者的地址。
- `to`: `String` - 20 字节，交易接收者的地址。如果是一个合约创建的交易，返回 `null`。
- `cumulativeGasUsed`: `Number` - 当前交易执行后累计花费的 `gas` 总值 [10]。
- `gasUsed`: `Number` - 执行当前这个交易单独花费的 `gas`。
- `contractAddress`: `String` - 20 字节，创建的合约地址。如果是一个合约创建交易，返回合约地址，其它情况返回 `null`。
- `logs`: `Array` - 这个交易产生的日志对象数组。

示例：

```
var receipt =
web3.eth.getTransactionReceipt('0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f
3dd2d66f4c9c6c445836d8b');
console.log(receipt);
{
  "transactionHash":
"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
  "transactionIndex": 0,
  "blockHash":
"0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "contractAddress": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
  "cumulativeGasUsed": 314159,
  "gasUsed": 30234,
  "logs": [{
        // logs as returned by getFilterLogs, etc.
    }, ...]
}
```

web3.eth.getTransactionCount

web3.eth.getTransactionCount(addressHexString [, defaultBlock] [, callback])

返回指定地址发起的交易数。

参数：

- `String` - 要获得交易数的地址。
- `Number|String` -（可选）如果未传递参数，默认使用 `web3.eth.defaultBlock` 定义的块，否则使用指定区块。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`Number` - 指定地址发送的交易数量。

示例：

```
var number =
web3.eth.getTransactionCount("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
```

```
console.log(number); // 1
```

web3.eth.sendTransaction

web3.eth.sendTransaction(transactionObject [, callback])

发送一个交易到网络。

参数：

- `Object` - 要发送的交易对象。
  - `from`: `String` - 指定的发送者的地址。如果不指定，使用 `web3.eth.defaultAccount`。
  - `to`: `String` - （可选）交易消息的目标地址，如果是合约创建，则不填.
  - `value`: `Number|String|BigNumber` - （可选）交易携带的货币量，以 `wei` 为单位。如果合约创建交易，则为初始的基金。
  - `gas`: `Number|String|BigNumber` - （可选）默认是自动，交易可使用的 `gas`，未使用的 `gas` 会退回。
  - `gasPrice`: `Number|String|BigNumber` - （可选）默认是自动确定，交易的 `gas` 价格，默认是网络 `gas` 价格的平均值 。
  - `data`: `String` - （可选）或者包含相关数据的字节字符串，如果是合约创建，则是初始化要用到的代码。
  - `nonce`: `Number` - （可选）整数，使用此值，可以允许你覆盖你自己的相同 `nonce` 的，正在 `pending` 中的交易[11]。
- `Function` - 回调函数，用于支持异步的方式执行[7]。

返回值：

`String` - 32 字节的交易哈希串。用 16 进制表示。

如果交易是一个合约创建，请使用 `web3.eth.getTransactionReceipt()` 在交易完成后获取合约的地址。
示例：

```
// compiled solidity source code using
https://chriseth.github.io/cpp-ethereum/
var code =
"603d80600c6000396000f3007c01000000000000000000000000000000000000000000000000000
0000000006000350463c6888fa18114602d57005b6007600435028060005260206000f3";

web3.eth.sendTransaction({data: code}, function(err, address) {
  if (!err)
    console.log(address); // "0x7f9fade1c0d57a7af66ab4ead7c2eb7b11a91385"
});
```

web3.eth.sendRawTransaction

web3.eth.sendRawTransaction(signedTransactionData [, callback])

发送一个已经签名的交易。比如可以用下述签名的例子：
https://github.com/SilentCicero/ethereumjs-accounts

参数：

- `String` - 16 进制格式的签名交易数据。
- `Function` - 回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 32 字节的 16 进制格式的交易哈希串。

如果交易是一个合约创建，请使用 `web3.eth.getTransactionReceipt()` 在交易完成后获取合约的地址。
示例：

```
var Tx = require('ethereumjs-tx');
var privateKey = new
Buffer('e331b6d69882b4cb4ea581d88e0b604039a3de5967688d3dcffdd2270c0fd109',
'hex')

var rawTx = {
  nonce: '0x00',
```

```
  gasPrice: '0x09184e72a000',
  gasLimit: '0x2710',
  to: '0x0000000000000000000000000000000000000000',
  value: '0x00',
  data:
'0x7f7465737432000000000000000000000000000000000000000000000000000000600057
'
}

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

var serializedTx = tx.serialize();

//console.log(serializedTx.toString('hex'));
//0xf889808609184e72a00082271094000000000000000000000000000000000000080a
47f7465737432000000000000000000000000000000000000000000000000006000571c
a08a8bbf888cfa37bbf0bb965423625641fc956967b81d12e23709cead01446075a01ce999b
56a8a88504be365442ea61239198e23d1fce7d00fcfc5cd3b44b7215f

web3.eth.sendRawTransaction(serializedTx.toString('hex'), function(err, hash)
{
  if (!err)
    console.log(hash); //
"0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385"
});
```

web3.eth.sign

web3.eth.sign(address, dataToSign, [, callback])

使用指定帐户签名要发送的数据，帐户需要处于 `unlocked` 状态。

参数：

- `String` - 签名使用的地址
- `String` - 要签名的数据
- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 签名后的数据。

返回的值对应的是 `ECDSA（Elliptic Curve Digital Signature Algorithm）`[12]签名后的字符串。

```
r = signature[0:64]
s = signature[64:128]
v = signature[128:130]
```

需要注意的是，如果你使用 `ecrecover`，这里的 `v` 值是 `00` 或 `01`，所以如果你想使用他们，你需要把这里的 `v` 值转成整数，再加上 `27`。最终你要用的值将是 `27` 或 `28`[13]。

示例：

```
var result = web3.eth.sign("0x135a7de83802408321b74c322f8558db1679ac20",
    "0x9dd2c369a187b4e6b9c402f030e50743e619301ea62aa4c0737d4ef7e10a3d49");
// second argument is web3.sha3("xyz")
console.log(result); //
"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aae
c7977f05e923889b33c0d0dd27d7226b6e6f56ce737465c5cfd04be400"
```

备注：如果你使用以太坊的客户端进行签名时，它们会在你要签名的数据前增加前缀`\x19Ethereum Signed Message:\n`[14]，感谢读者@刘兵同学的反馈。

eth_sign

The sign method calculates an Ethereum specific signature with: sign(keccak256("\x19Ethereum Signed Message:\n" + len(message) + message))).

By adding a prefix to the message makes the calculated signature recognisable as an Ethereum specific signature. This prevents misuse where a malicious DApp can sign arbitrary data (e.g. transaction) and use the signature to impersonate the victim.

web3.eth.call

web3.eth.call(callObject [, defaultBlock] [, callback])

在节点的 VM 中，直接执行消息调用交易。但不会将数据合并区块链中（这样的调用不会修改状态）。

参数：

- `Object` - 返回一个交易对象，同 `web3.eth.sendTransaction`。与 `sendTransaction` 的区别在于，`from` 属性是可选的。

- `Number|String` - （可选）如果不设置此值使用 `web3.eth.defaultBlock` 设定的块，否则使用指定的块。
- `Function` - （可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 函数调用返回的值。

示例：

```
var Web3 = require('web3');

if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
} else {
  // set the provider you want from Web3.providers
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
}

var from = web3.eth.accounts[0];
//部署合约的发布地址
/*合约内容如下
pragma solidity ^0.4.0;

contract Calc{
  function add(uint a, uint b) returns (uint){
    return a + b;
  }
}
*/
var to = "0xa4b813d788218df688d167102e5daff9b524a8bc";

//要发送的数据
//格式说明见:
http://me.tryblockchain.org/Solidity-call-callcode-delegatecall.html
var data =
"0x771602f7000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000000002";

var result = web3.eth.call({
  from : from,
  to : to,
  data : data
});
```

```
//返回结果 32 字长的结果 3
console.log(result);
```

web3.eth.estimateGas

web3.eth.estimateGas(callObject [, callback])

在节点的 VM 节点中执行一个消息调用，或交易。但是不会合入区块链中。返回使用的 `gas` 量。

参数：

同 `web3.eth.sendTransaction`，所有的属性都是可选的。

返回值：

`Number` - 模拟的 `call/transcation` 花费的 `gas`。

示例：

```
var result = web3.eth.estimateGas({
    to: "0xc4abd0339eb8d57087278718986382264244252f",
    data:
"0xc6888fa1000000000000000000000000000000000000000000000000000000000000003
"
});
console.log(result); //
"0x0000000000000000000000000000000000000000000000000000000000000015"
```

web3.eth.filter

参数：

- `String|Object` - 字符串的可取值[latest，pending]。`latest` 表示监听最新的区块变化，`pending` 表示监听正在 `pending` 的区块。如果需要按条件对象过滤，如下：
    - `fromBlock`: `Number|string` - 起始区块号(如果使用字符串 `latest`，意思是最新的，正在打包的区块），默认值是 `latest`。

- `toBlock`: `Number|string` - 终止区块号(如果使用字符串 `latest`，意思是最新的，正在打包的区块），默认值是 `latest`。
- `address`: `String` - 单个或多个地址。获取指定帐户的日志。
- `topics`: `String[]` - 在日志对象中必须出现的字符串数组。顺序非常重要，如果你想忽略主题，使用 `null`。如，[null,'0x00...']，你还可以为每个主题传递一个单独的可选项数组，如 [null,['option1','option1']]。

返回值：

`Object` - 有下述方法的过滤对象。

- `filter.get(callback)`: 返回满足过滤条件的日志。
- `filter.watch(callback)`: 监听满足条件的状态变化，满足条件时调用回调[7]。
- `filter.stopWatching()`: 停止监听，清除节点中的过滤。你应该总是在监听完成后，执行这个操作。

监听回调返回值：

- `String` - 当使用 `latest` 参数时。返回最新的一个区块哈希值。
- `String` - 当使用 `pending` 参数时。返回最新的 `pending` 中的交易哈希值。
- `Object` - 当使用手工过滤选项时，将返回下述的日志对象。
  - `logIndex`: `Number` - 日志在区块中的序号。如果是 `pending` 的日志，则为 `null`。
  - `transactionIndex`: `Number` - 产生日志的交易在区块中的序号。如果是 `pending` 的日志，则为 `null`。
  - `transactionHash`: `String`，32 字节 - 产生日志的交易哈希值。

- `blockHash`: `String`,32 字节 - 日志所在块的哈希。如果是 `pending` 的日志，则为 `null`。
- `blockNumber`: `Number` - 日志所在块的块号。如果是 `pending` 的日志，则为 `null`。
- `address`: `String`，32 字节 - 日志产生的合约地址。
- `data`: `string` - 包含日志一个或多个 32 字节的非索引的参数。
- `topics`: `String[]` - 一到四个 32 字节的索引的日志参数数组。(在 Solidity 中，第一个主题是整个事件的签名（如，`Deposit(address,bytes32,uint256)`），但如果使用匿名的方式定义事件的情况除外）

事件监听器的返回结果，见后`合约对象的事件`。

示例：

```
var filter = web3.eth.filter('pending');

filter.watch(function (error, log) {
  console.log(log); //
{"address":"0x0000000000000000000000000000000000000000",
"data":"0x0000000000000000000000000000000000000000000000000000000000000000"
, ...}
});

// get all past logs again.
var myResults = filter.get(function(error, logs){ ... });

...

// stops and uninstalls the filter
filter.stopWatching();
```

web3.eth.contract

web3.eth.contract(abiArray)

创建一个 Solidity 的合约对象，用来在某个地址上初始化合约。

参数：

- `Array` - 一到多个描述合约的函数，事件的 ABI 对象。

返回值：

`Object` - 一个合约对象。

示例：

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

// instantiate by address
var contractInstance = MyContract.at([address]);

// deploy new contract
var contractInstance = MyContract.new([contructorParam1] [, contructorParam2],
{data: '0x12345...', from: myAccount, gas: 1000000});

// Get the data to deploy the contract manually
var contractData = MyContract.new.getData([contructorParam1] [,
contructorParam2], {data: '0x12345...'});
// contractData = '0x1234564321345600000000023434234'
```

你可以或者使用一个在某个地址上已经存在的合约，或者使用编译后的字节码部署一个全新的的合约。

```
// Instantiate from an existing address:
var myContractInstance = MyContract.at(myContractAddress);


// Or deploy a new contract:

// Deploy the contract asyncronous from Solidity file:
...
const fs = require("fs");
const solc = require('solc')

let source = fs.readFileSync('nameContract.sol', 'utf8');
let compiledContract = solc.compile(source, 1);
let abi = compiledContract.contracts['nameContract'].interface;
let bytecode = compiledContract.contracts['nameContract'].bytecode;
let gasEstimate = web3.eth.estimateGas({data: bytecode});
let MyContract = web3.eth.contract(JSON.parse(abi));

var myContractReturned = MyContract.new(param1, param2, {
```

```
   from:mySenderAddress,
   data:bytecode,
   gas:gasEstimate}, function(err, myContract){
    if(!err) {
       // NOTE: The callback will fire twice!
       // Once the contract has the transactionHash property set and once its
deployed on an address.

       // e.g. check tx hash on the first call (transaction send)
       if(!myContract.address) {
           console.log(myContract.transactionHash) // The hash of the
transaction, which deploys the contract

       // check address on the second call (contract deployed)
       } else {
           console.log(myContract.address) // the contract address
       }

       // Note that the returned "myContractReturned" === "myContract",
       // so the returned "myContractReturned" object will also get the address
set.
    }
  });

// Deploy contract syncronous: The address will be added as soon as the contract
is mined.
// Additionally you can watch the transaction by using the "transactionHash"
property
var myContractInstance = MyContract.new(param1, param2, {data: myContractCode,
gas: 300000, from: mySenderAddress});
myContractInstance.transactionHash // The hash of the transaction, which
created the contract
myContractInstance.address // undefined at start, but will be auto-filled later
```

示例：

```
// contract abi
var abi = [{
     name: 'myConstantMethod',
     type: 'function',
     constant: true,
     inputs: [{ name: 'a', type: 'string' }],
     outputs: [{name: 'd', type: 'string' }]
}, {
     name: 'myStateChangingMethod',
     type: 'function',
```

```
    constant: false,
    inputs: [{ name: 'a', type: 'string' }, { name: 'b', type: 'int' }],
    outputs: []
}, {
    name: 'myEvent',
    type: 'event',
    inputs: [{name: 'a', type: 'int', indexed: true},{name: 'b', type: 'bool',
indexed: false}]
}];

// creation of contract object
var MyContract = web3.eth.contract(abi);

// initiate contract for an address
var myContractInstance =
MyContract.at('0xc4abd0339eb8d57087278718986382264244252f');

// call constant function
var result = myContractInstance.myConstantMethod('myParam');
console.log(result) // '0x25434534534'

// send a transaction to a function
myContractInstance.myStateChangingMethod('someParam1', 23, {value: 200, gas:
2000});

// short hand style
web3.eth.contract(abi).at(address).myAwesomeMethod(...);

// create filter
var filter = myContractInstance.myEvent({a: 5}, function (error, result) {
  if (!error)
    console.log(result);
    /*
    {
        address: '0x8718986382264244252fc4abd0339eb8d5708727',
        topics: "0x1234567890123456789012345678901234567890123456789012",
"0x0000000000000000000000000000000000000000000000000000000000000005",
        data:
"0x0000000000000000000000000000000000000000000000000000000000000001",
        ...
    }
    */
});
```

合约对象的方法

```
// Automatically determines the use of call or sendTransaction based on the method type
myContractInstance.myMethod(param1 [, param2, ...] [, transactionObject] [, defaultBlock] [, callback]);

// Explicitly calling this method
myContractInstance.myMethod.call(param1 [, param2, ...] [, transactionObject] [, defaultBlock] [, callback]);

// Explicitly sending a transaction to this method
myContractInstance.myMethod.sendTransaction(param1 [, param2, ...] [, transactionObject] [, callback]);

// Get the call data, so you can call the contract through some other means
var myCallData = myContractInstance.myMethod.getData(param1 [, param2, ...]);
// myCallData = '0x45ff3ff6000000000004545345345345..'
```

合约对象内封装了使用合约的相关方法。可以通过传入参数，和交易对象来使用方法。

参数：

- `String|Number` - （可选）零或多个函数参数。如果传入一个字符串，需要使用十六进制编码，如，`0xdedbeef`。
- `Object` - （可选）最后一个参数（如果传了 `callback`，则是倒数第二个参数），可以是一个交易对象。查看 `web3.eth.sendTransaction` 的第一个参数说明来了解更多。注意，这里不需要填 `data` 和 `to` 属性。
- `Number|String` -（可选）如果不设置此值使用 `web3.eth.defaultBlock` 设定的块，否则使用指定的块。
- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 如果发起的是一个 `call`，对应的是返回结果。如果是 `transaction`，则要么是一个创建的合约地址，或者是一个 `transaction` 的哈希值。查看 `web3.eth.sendTransaction` 了解更多。

示例：

```
// creation of contract object
var MyContract = web3.eth.contract(abi);

// initiate contract for an address
var myContractInstance =
MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

var result = myContractInstance.myConstantMethod('myParam');
console.log(result) // '0x25434534534'

myContractInstance.myStateChangingMethod('someParam1', 23, {value: 200, gas:
2000}, function(err, result){ ... });
```

合约对象的事件

你可以像 `web3.eth.filter` 这样使用事件，他们有相同的方法，但需要传递不同的对象来创建事件过滤器。
参数：

- `Object` - 你想返回的索引值（过滤哪些日志）。如，`{'valueA': 1, 'valueB': [myFirstAddress, mySecondAddress]}`。默认情况下，所以有过滤项被设置为 `null`。意味着默认匹配的是合约所有的日志。
- `Object` - 附加的过滤选项。参见 `web3.eth.filter` 的第一个参数。默认情况下，这个对象会设置 `address` 为当前合约地址，同时第一个主题为事件的签名。
- `Function` -（可选）传入一个回调函数，将立即开始监听，这样就不用主动调用 `myEvent.watch(function(){})`[7]。

回调返回值：

`Object` - 事件对象，如下：

- `address`: `String，32 字节` - 日志产生的合约地址。
- `args`: `Object` - 事件的参数。
- `blockHash`: `String，32 字节` - 日志所在块的哈希。如果是 `pending` 的日志，则为 `null`。
- `blockNumber`: `Number` - 日志所在块的块号。如果是 `pending` 的日志，则为 `null`。
- `logIndex`: `Number` - 日志在区块中的序号。如果是 `pending` 的日志，则为 `null`。
- `event`: `String` - 事件名称。
- `removed`: `bool` - 标识产生事件的这个交易是否被移除（因为孤块），或从未生效（被拒绝的交易）。
- `transactionIndex`: `Number` - 产生日志的交易在区块中的序号。如果是 `pending` 的日志，则为 `null`。
- `transactionHash`: `String，32 字节` - 产生日志的交易哈希值。

示例：

```
var MyContract = web3.eth.contract(abi);
var myContractInstance =
MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

// watch for an event with {some: 'args'}
var myEvent = myContractInstance.MyEvent({some: 'args'}, {fromBlock: 0, toBlock:
'latest'});
myEvent.watch(function(error, result){
   ...
});

// would get all past logs again.
var myResults = myEvent.get(function(error, logs){ ... });
```

```
...

// would stop and uninstall the filter
myEvent.stopWatching();
```

合约 allEvents

```
var events = myContractInstance.allEvents([additionalFilterObject]);

// watch for changes
events.watch(function(error, event){
  if (!error)
    console.log(event);
});

// Or pass a callback to start watching immediately
var events = myContractInstance.allEvents([additionalFilterObject,]
function(error, log){
  if (!error)
    console.log(log);
});
```

调用合约创建的所有事件的回调。

参数：

- `Object` - 附加的过滤选项。参见 `web3.eth.filter` 的第一个参数。默认情况下，这个对象会设置 `address` 为当前合约地址，同时第一个主题为事件的签名。
- `Function` -（可选）传入一个回调函数，将立即开始监听，这样就不用主动调用 `myEvent.watch(function(){})`[7]。

回调返回值：

`Object` - 详见 合约对象的事件 了解更多。

示例：

```
var MyContract = web3.eth.contract(abi);
var myContractInstance =
MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

// watch for an event with {some: 'args'}
```

```
var events = myContractInstance.allEvents({fromBlock: 0, toBlock: 'latest'});
events.watch(function(error, result){
   ...
});

// would get all past logs again.
events.get(function(error, logs){ ... });

...

// would stop and uninstall the filter
myEvent.stopWatching();
```

web3.eth.getCompilers

web3.eth.getCompilers([callback])

返回可用的编译器。

参数值：

- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`Array` - 返回一个字符串数组，可用的编译器。

web3.eth.compile.solidity

web3.eth.compile.solidity(sourceString [, callback])

编译 Solidity 源代码。

参数：

- `String` - Solidity 源代码。
- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`Object` - 合约和编译信息。

示例：

```
var source = "" +
    "contract test {\n" +
    "   function multiply(uint a) returns(uint d) {\n" +
    "       return a * 7;\n" +
    "   }\n" +
    "}\n";
var compiled = web3.eth.compile.solidity(source);
console.log(compiled);
// {
  "test": {
    "code":
"0x605280600c6000396000f3006000357c010000000000000000000000000000000000000000000000000000000090048063c6888fa114602e57005b6037600436041565b806000526020
06000f35b60006007820290506004d565b91905056",
    "info": {
      "source": "contract test {\n\tfunction multiply(uint a) returns(uint d)
{\n\t\treturn a * 7;\n\t}\n}\n",
      "language": "Solidity",
      "languageVersion": "0",
      "compilerVersion": "0.8.2",
      "abiDefinition": [
        {
          "constant": false,
          "inputs": [
            {
              "name": "a",
              "type": "uint256"
            }
          ],
          "name": "multiply",
          "outputs": [
            {
              "name": "d",
              "type": "uint256"
            }
          ],
          "type": "function"
        }
      ],
      "userDoc": {
        "methods": {}
      },
      "developerDoc": {
```

```
        "methods": {}
      }
    }
  }
}
```

web3.eth.compile.lll

web3. eth.compile.lll(sourceString [, callback])

编译 LLL 源代码。

参数：

- `String` - LLL 源代码。
- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 十六进制格式编译后的 LLL 编码。

示例：

```
var source = "...";

var code = web3.eth.compile.lll(source);
console.log(code); //
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b6021600435602b565b8060005260206000f35b600081600702905091905056"
```

web3.eth.compile.serpent

web3.eth.compile.serpent(sourceString [, callback])

编译 serpent 源代码。

参数：

- `String` - serpent 源代码。
- `Function` -（可选）回调函数，用于支持异步的方式执行 [7]。

返回值：

`String` - 十六进制格式的编译后的 serpent 编码。

web3.eth.namereg

web3.eth.namereg

返回一个全球注意的对象。

使用方式：

查看这里的例子：
https://github.com/ethereum/web3.js/blob/master/example/namereg.html

web3.db

web3.db.putString

web3.db.putString(db, key, value)

这个方法应当在我们打算以一个本地数据库的级别存储一个字符串时使用。

参数：

- `String` - 存储使用的数据库。
- `String` - 存储的键。
- `String` - 存储的值。

返回值：

`Boolean` - `true` 表示成功，否则返回 `false`。

示例：

```
web3.db.putString('testDB', 'key', 'myString') // true
```

web3.db.getString

web3.db.getString(db, key)

从本地的 leveldb 数据库中返回一个字符串。

参数：

- `String` - 存储使用的数据库。
- `String` - 存储的键。

返回值：

`String` - 存储的值。

示例：

```
var value = web3.db.getString('testDB', 'key');
console.log(value); // "myString"
```

web3.db.putHex

web3.db.putHex(db, key, value)

在本地的 leveldb 中存储二进制数据。

参数：

- `String` - 存储使用的数据库。
- `String` - 存储的键。
- `String` - 十六进制格式的二进制。

返回值：

`Boolean` - 成功返回 `true`，失败返回 `false`。

示例：

```
web3.db.putHex('testDB', 'key', '0x4f554b443'); // true
```

web3.db.getHex

web3.db.getHex(db, key)

返回本地的 leveldb 中的二进制数据。

参数：

- `String` - 存储使用的数据库。
- `String` - 存储的键。

返回值：

`String` - 存储的十六进制值。

示例：

```
var value = web3.db.getHex('testDB', 'key');
console.log(value); // "0x4f554b443"
```

---

1. https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify ↩
2. Big Number 文档链接： https://github.com/MikeMcl/bignumber.js ↩
3. ASCII 码表 http://baike.baidu.com/link?url=fz4Ytl_tjpUhsPbUb4fa3hyXHKJDMcRB5M3K1p0VStnminbvLX4-UmwPCovk1pZUOIemosGv2hRT-r0flGMtEGk3ON8sQctG4-KU67G3fBiOJX6r1CHoKTi-K6BlEEa6egHulQju1p1n1ce1axyeBK ↩
4. 工作正在进行中？ https://github.com/ethereum/web3.js/pull/375 ↩
5. Web3 Javascript Đapp API ↩
6. http://baike.baidu.com/link?url=9QLxfeXVf7pRpSTSugt2I9ylZA9_vh3sbqs8S8J-dVaZJg9AeADF4P0HlwyOjHOsENJKQS8z7cb0YFSDVMmsYf-xgtODmMXdQNovqRqE2B7 ↩
7. 参见 Web3.js API 基本中的`使用 callback`的章节。 ↩
8. 关于`getStroageAt`的说明来

   源： http://ethereum.stackexchange.com/questions/5865/how-does-web3-eth-getstorageat-work ↩
9. https://zh.wikipedia.org/wiki/布隆过滤器 ↩
10. http://ethereum.stackexchange.com/questions/3346/what-is-and-how-to-calculate-cumulative-gas-use[^cumulative]: http://ethereum.stackexchange.com/questions/3346/what-is-and-how-to-calculate-cumulative-gas-used ↩

11. http://zeltsinger.com/2016/11/07/neat-ethereum-tricks-the-transaction-nonce/ ↩

12. https://zh.wikipedia.org/wiki/椭圆曲线密码学 ↩

13. 这里整理了一个文章，使用 web3.js 加密，再使用 Solidity 的 ecrecover 校验签名的完整过程。 http://me.tryblockchain.org/web3js-sign-ecrecover-decode.html ↩

14. https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sign ↩