



# 学会 Substrate 区块链应用开发

## 4.2 - Polkadot-JS API

---

**Jimmy Chu**

jimmy.chu@parity.io

获取帮助: <https://substrate.io>

# 内容

---

- 介绍
- 创建接口 及 调试
- 链上查询 及 大数值 bn.js
- 查看常量 及 远程过程调用
- 交易 (Extrinsics)
- 撷取用户帐号
- 订阅事件
- 自定义类型

# 介紹

---

- 官方的与基于 Substrate 框架开发的区块链 API
- 可用来查询链上状态, 对交易作签名, 提交交易(具签名及不具签名的), 订阅链上事件等。
- 用了 [JS Promise](#) 的写法。因此也需要熟悉用 js 里 async / await 的语法
- API 里与链互动的函数是根据链上的元数据 (meta-data) 动态生成。

例子: 你要有一个 `templateModule pallet` 的 `doSomething()` 函数在 Substrate 内, 才在 JS api 里有 `api.tx.templateModule.doSomething()` 这函数。

# 开始使用

---

官方文档: <https://polkadot.js.org/api/>

开始使用:

# 如果只是与 Substrate 节点交互

```
yarn add @polkadot/api
```

```
yarn add @polkadot/keyring
```

# 如果是在浏览器内互前端交互, 最好还加

```
yarn add @polkadot/extension-dapp
```

```
yarn add @polkadot/ui-keyring
```

```
yarn add @polkadot/ui-settings
```

# 材料

---

git clone 这两个项目 (如果还没做)

- [Substrate node template](#)

- Node template 內的 templateModule 及 doSomething()

# 创建接口

---

然后第一件事就是创建一个 api 对象并连到远端接口

```
import { ApiPromise, WsProvider } from '@polkadot/api';

// 指定远端接口
const wsProvider = new WsProvider('ws://127.0.0.1:9944');

// 创建接口
const api = await ApiPromise.create({ provider: wsProvider });

// 简单测试
console.log(api.genesisHash.toHex());
```

# 调试 Substrate API

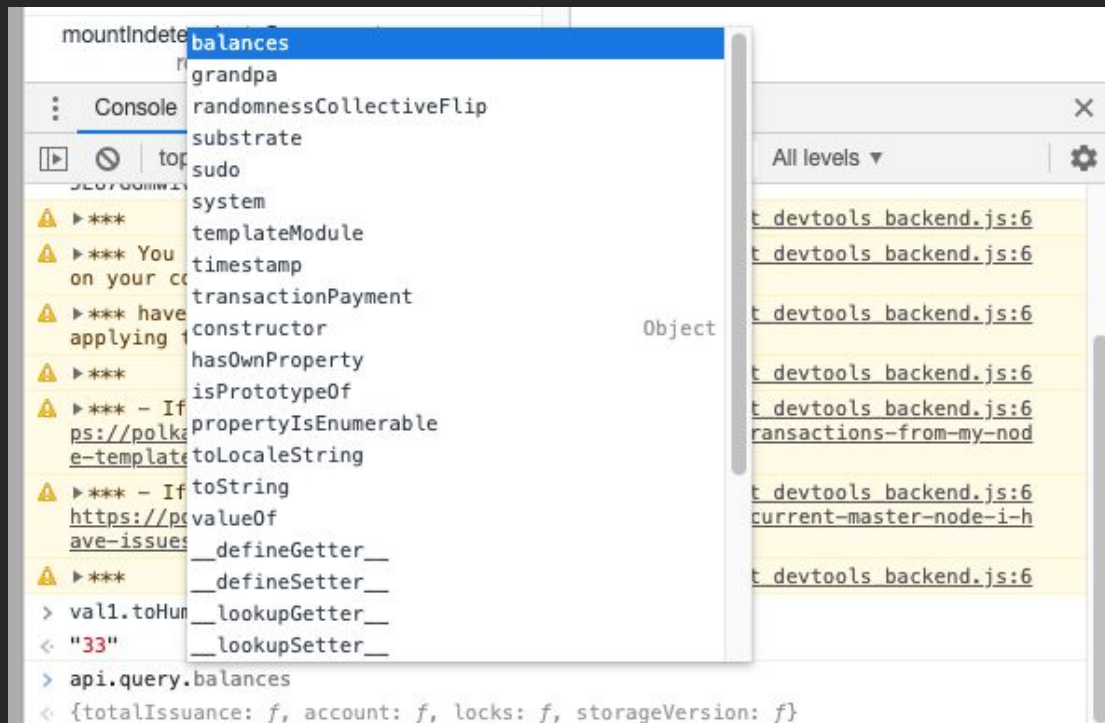
---

在 polkadot-JS App 内 Javascript 一栏, 在代码里加

`debugger;`

跑的时候, 就可以在 console 内看到有什么函数可以调用

# 调试 Substrate API





# 链上查询

---

```
api.query.<pallet>.<storage>
```

比如:

```
const val = await api.query.templateModule.something();
```

这是对应着连接的 Substrate 节点, 有 templateModule, 并在 decl\_storage!{...} 内定义了 something 的 getter 函数

# 链上查询

---

```
const val = await api.query.templateModule.something();
```

这里返回的是一个 对象, 接着可以用:

- `val.isSome / val.isNone` - 看是不是有值
- `val.toJSON()` - 查看其值
- `val.toString()` - 变成字符串
- `val.toHuman()` - 显示用户友好的值

# 大数值 bn.js

- 很多区块链的数值都是大于 JS 本身 Number 的值  $2^{53} - 1$ , Polkadot-JS API 用了 [bn.js](#) 来包装它。比如:

```
const acct = await api.query.system
  .account("5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY");
const freeBalance = acct.data.free;
debugger;
```

freeBalance 是一个 bn 对象, 可以用 `freeBalance.toString(10)` 看它十进制的值, 以字符串显示。

# 查看常量

---

以 `api.consts.<pallet>.<constant>` 来获取。比如：

```
const minPeriod = api.consts.timestamp.minimumPeriod.toString();
```

因为常量在连接到节点时已取得，不是动态查询，所以不是异步操作，不需要用 `await`。

# 远程过程调用 (RPC)

---

`api.rpc.<callable>.<function>` 比如:

```
const hash = await api.rpc.chain.getBlockHash();  
console.log(hash.toString());
```

这里 `hash` 这个对象可以用 `toString()` 来查看最新出块的哈希值。

你也可传入一个数 block number, 看到对应区块数的哈希值。

# 交易 (Extrinsics) - 具签名

---

要作交易, 主要是用到 `api.tx.\*` 的

```
const hash = await api.tx.templateModule.doSomething(22)
  .signAndSend(alice)
```

- 用 alice 這個用戶, 发出了一个具签名的交易
- 对应用有 templateModule pallet 并在 decl\_module!{...} 有定义 doSomething() 的函数。

# 交易 (Extrinsics) - 不具签名

---

```
(async () => {  
  const hash = await api.tx.templateModule.someFunction(22)  
    .sign()  
})();
```

- 注意 Substrate 节点默认不处理不具签名的交易
- 在 Substrate 节点得定义不具签名交易的处理函数, 这交易才能顺利处理

# 撷取用户帐号

---

- 不建议用户在应用内创建钱包帐号。创建钱包帐号可以透过 [polkadot-JS extension](#) 来进行, 支援 Chrome 及 Firefox 浏览器。就像 Metamask
- 当 Substrate 节点以开发模式 (`--dev`)运行时, 它是会自动生成 `Alice`, `Bob`, `Charles`, `Dave`, `Eve` 的帐号。



# 撷取用户帐号

---

所以我们用以下方法要拿取这两类不同 (dev 及 extension )的帐号

```
import { web3Accounts, web3Enable } from '@polkadot/extension-dapp';
import keyring from '@polkadot/ui-keyring';

// 允许 app 可读取 polkadot-js extension 管理的帐号
await web3Enable('My DAPP');

// 遍历 polkadot-js extension 管理的帐号
let allAccounts = await web3Accounts();
allAccounts = allAccounts.map(({ address, meta }) =>
  ({ address, meta: { ...meta, name: `${meta.name} (${meta.source})` } }));

// 载入包括开发节点的帐号
keyring.loadAll({ isDevelopment: true }, allAccounts);
```

# 撷取用户帐号

---

跟着以 `keyring.getPairs()` 就能遍历你能取得的帐号。然后 `alice` 就是

```
const alice = keyring.getPairs()[0];
```

# 订阅事件

---

在 `api.query.*`, `api.rpc.*` 可在最后一个参数放入一个订阅函数:

```
const unsub = await api.query.system.account(alice, acct => {  
  // 每当这帐号资讯变更, 订阅函数会被呼叫一次  
});  
  
...  
unsub();
```

返回一个取消订阅事件的函数

# 订阅事件

---

在 api.tx.\*

// 假设 alice 已经存储着一个用户帐号

```
const unsub = await api.tx.templateModule.doSomething(22).signAndSend(alice, result
=> {
  const { status, events } = result;
  // ...
}).catch(err => {
  // ...
});
```

# 订阅事件

---

在 ``sendAndSend()`` 或 ``send()`` 后再输入一个函数参数, 这就是一个订阅函数, 可知道交易的最新状况。里面有:

- ``status`` 可知道是否导入了区块里 ``isInBlock``, 或已最后确认 ``isFinalized``.
- ``events`` 可知道这交易触发的事件。
- ``.catch()`` 里面的就是错误处理。
- 返回一个取消订阅的函数。

# 增加自定义类型

---

- 如在 Substrate 开发自己的模块增加了自定义类型 (新的 `type` 或 `struct`), 那在开发前端也需要告知 api 这类型。
- 方法是当你在一开始创建 api 接口时, 再传入一个 types 的参数

```
const api = await ApiPromise.create({  
  provider: wsProvider,  
  types: {  
    Balance: 'u64'  
  }  
});
```

# Questions?

---

官网文档: [substrate.dev](https://substrate.dev)

知乎专栏: [parity.link/zhihu](https://parity.link/zhihu)

[jimmy.chu@parity.io](mailto:jimmy.chu@parity.io)