

第31讲 | 数字货币钱包服务

2018-06-04 陈浩

深入浅出区块链

[进入课程 >](#)



讲述：黄洲君

时长 08:01 大小 3.67M



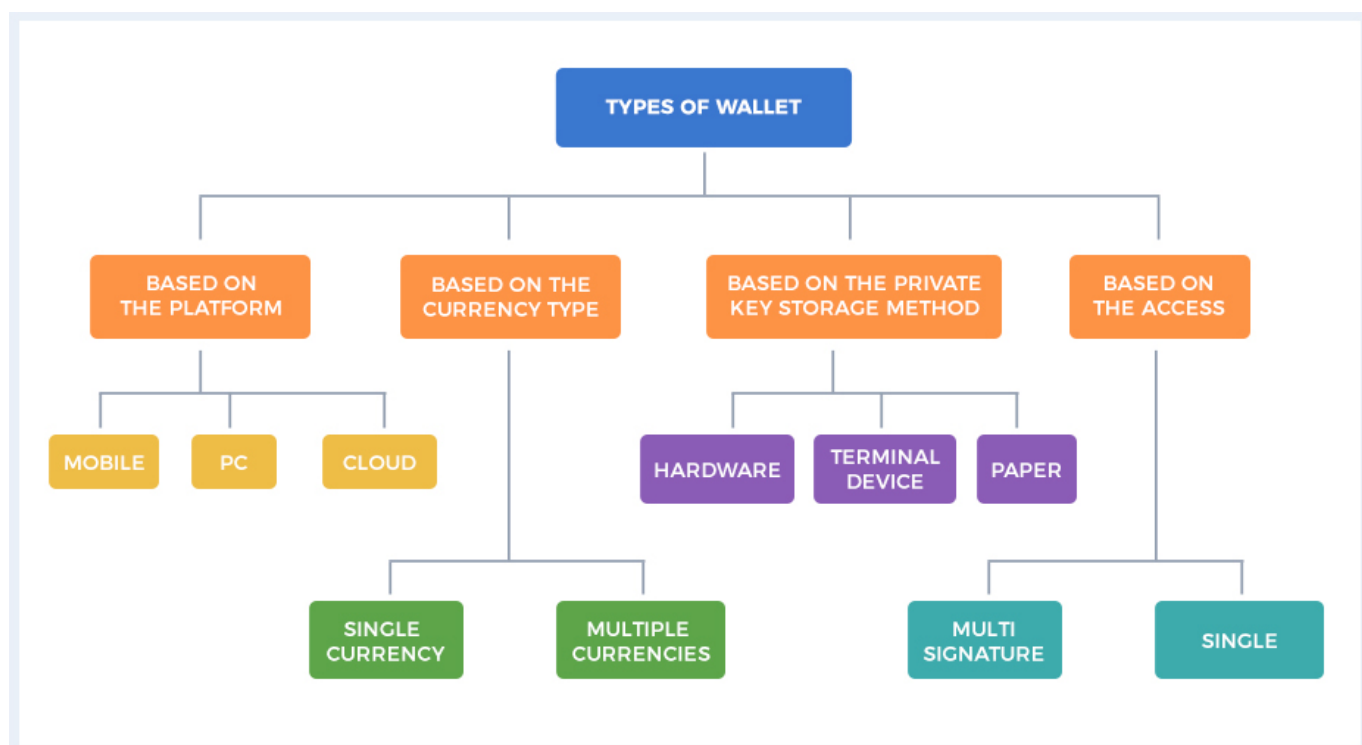
上一篇，我们谈到了“区块链即服务”的概念。实际上，区块链第一个需要解决的服务就是数字货币支付服务。如何将数字货币钱包集成到系统中，我认为这是区块链行业最为迫切的问题。

今天我们就来了解一下数字货币钱包，它面临了什么样的问题，这样的问题又需要什么技术才能解决，而数字货币平台想要定制自己的数字货币钱包服务，又应当如何集成。

数字货币钱包的分类

目前市面上的数字货币钱包有很多种，看起来似乎有些眼花缭乱，不过，我们可以将它们进行分类后，再快速了解。

下图展示了按照不同属性区分的区块链钱包。



左一是按照用户端平台划分的钱包，这种钱包实际是在服务端运行的，用户端的钱包实际上只是一个代理，所以用户不需要关心钱包的细节，使用起来十分方便，典型的例子是各种在线钱包。

左二是按照货币类型分类的钱包，主要是指这款钱包到底是否支持多币种，这里的多币种可以是基于以太坊 ERC20 Token 的同一个区块链上的多币种，也可以是集成了比特币和以太坊等不同区块链的多币种。

右二是按照私钥存储的方式来区分的钱包，实际上这里主要涉及了用户私钥是否被平台托管，如果不托管直接存储在用户端，也就是硬件、终端设备、纸质记录，这些都可以被称为 On-chain 的钱包；如果用户无法接触到私钥，私钥被托管在平台，那么这种钱包也被称为 Off-chain 的钱包。

右一是按照访问方式进行分类的钱包，例如可以多个人共同管理，同时它也是需要多重签名支持的钱包，否则就变成了个人私有的钱包。

以上的分类并不是绝对的，一个钱包可以兼具以上不同的属性，例如某个 Mobile 钱包是提供 On-chain 的，也提供多重签名、提供多币种的钱包，这种钱包通常就是业界比较流行的钱包类型。

但是，对于平台方来说，上述钱包类型可能不足以支持自己平台的需求，并发挥出最佳的功效。毕竟作为平台来说，对高可用、分叉检测、区块确认的要求是远远高于普通钱包的，这样的问题又是如何解决的呢？这就引入了一项新的技术。

扫描区块技术 Block scan

我们之前在深入区块链技术部分介绍过，构成区块链的四个核心技术是：P2P 网络协议、分布式一致性算法、加密签名算法、账户与交易模型。这四个技术对应到数字货币钱包中就是，P2P 网络、持久化存储、账户以及私钥管理、共识与交易验证四大模块。

其中，持久化存储模块是由全节点钱包自带的嵌入式数据库提供的，这里有 LevelDB、BerkeleyDB、SQLite3 等多种选择。


但是无论选择哪种嵌入式数据库，都面临了一个严峻问题，精细化的交易查询验证与性能不可兼具。换句话说，任何全节点的嵌入式数据库都无法和服务器级别的数据库相媲美。

对于平台开发来说，显然选择服务器级别的数据库是更为合适的选择。那么这里就涉及了一个问题，如何把全节点钱包中的数据转换成为数据库服务器中的数据，这就需要用到一种扫描区块技术，简称扫块。

扫块，顾名思义，就是指扫描全节点钱包中的所有区块，然后将其解析后存储到数据库服务器的过程，这些数据库可以是 MongoDB，也可以是 MySQL，取决于你的业务需要。

我们可以举元界区块链扫块的例子，元界上的区块结构与比特币接近，你可以将其类比为比特币区块链。

以下是 Python 代码，展示了基于 MySQL 的关系型表结构，目的是从元界的嵌入式数据库中扫描区块，然后存储到 MySQL 中。

 复制代码

```
1     def init_table(conn):
2     tables = []
3     tb_block = '''
4     create table if not EXISTS block (
5         number bigint primary KEY ,
6         hash char(64) not null,
7         bits bigint,
8         transaction_count INTEGER ,
```

```

9  mixhash VARCHAR (128),
10 version char(8) ,
11 merkle_tree_hash char(64),
12 previous_block_hash CHAR (64),
13 nonce varchar(128) ,
14 time_stamp bigint
15 ) DEFAULT charset=utf8;
16 '''
17
18 tb_tx = '''
19 create table if not EXISTS tx (
20     id bigint PRIMARY KEY ,
21     block_height bigint REFERENCES block(id),
22     hash char(64) not null
23 )DEFAULT charset=utf8 ;'''
24
25 tb_address = '''
26 create table if not EXISTS address(
27     id int PRIMARY KEY ,
28     address VARCHAR (64) UNIQUE
29 )DEFAULT charset=utf8;
30 '''
31
32 tb_output = '''
33 create table if not EXISTS tx_output(
34     id bigint PRIMARY key,
35     hash char(64) NOT NULL ,
36     tx_id bigint REFERENCES tx(id),
37     output_index bigint not null,
38     output_value bigint,
39     address_id bigint REFERENCES address(id),
40     script varchar(1024),
41     asset varchar(64),
42     decimal_number varchar(8)
43 )DEFAULT charset=ascii;
44 '''
45
46 tb_output_fork = '''
47 create table if not EXISTS tx_output_fork(
48     id bigint PRIMARY key,
49     hash char(64) NOT NULL ,
50     tx_id bigint,
51     output_index bigint not null,
52     output_value bigint,
53     address_id bigint,
54     script varchar(1024),
55     asset varchar(64),
56     decimal_number varchar(8)
57 )DEFAULT charset=ascii;
58 '''
59 tb_tx_fork = '''
60 create table if not EXISTS tx_fork (

```

```

61         id bigint PRIMARY KEY ,
62         block_height bigint,
63         hash char(64) not null
64     )DEFAULT charset=ascii ;'''
65
66 tb_input_fork = '''
67     create table if not EXISTS tx_input_fork(
68         id bigint PRIMARY key,
69         tx_id bigint,
70         belong_tx_id bigint,
71         tx_index bigint,
72         tx_value bigint not null,
73         script varchar(1024),
74         address_id bigint,
75         asset varchar(64),
76         decimal_number varchar(8)
77     )DEFAULT charset=ascii;
78     '''
79
80 tb_block_fork = '''
81     create table if not EXISTS block_fork (
82         number bigint primary KEY ,
83         hash char(64) not null,
84         bits bigint,
85         transaction_count INTEGER ,
86         mixhash VARCHAR (128),
87         version char(8) ,
88         merkle_tree_hash char(64),
89         previous_block_hash CHAR (64),
90         nonce varchar(128) ,
91         time_stamp bigint
92     ) DEFAULT charset=ascii;
93     '''
94
95 tb_output_asset = '''
96     create table if not EXISTS tx_output_asset(
97         id bigint PRIMARY key,
98         hash char(64) NOT NULL ,
99         tx_id bigint REFERENCES tx(id),
100        output_index bigint not null,
101        output_value bigint,
102        address_id bigint REFERENCES address(id),
103        asset_name varchar(64),
104        issuer varchar(64),
105        asset_type varchar(8),
106        description varchar(64)
107    )DEFAULT charset=utf8;
108    '''
109
110 tb_input = '''
111     create table if not EXISTS tx_input(
112         id bigint PRIMARY key,
113         tx_id bigint REFERENCES tx(id),

```

```

113     belong_tx_id bigint REFERENCES tx(id),
114     tx_index bigint REFERENCES tx_output(output_index),
115     tx_value bigint not null,
116     script varchar(1024),
117     address_id bigint REFERENCES address(id),
118     asset varchar(64),
119     decimal_number varchar(8)
120 )DEFAULT charset=ascii;
121 '''


```

我们按照元界区块链的结构，可以把表分为四大类：

第一类是区块 block；第二类是交易 Tx；第三类是交易输入输出：tb_input, tb_output；第四类是分叉处理。

下面我贴一个普通的以 JSON 格式展示的区块和交易，你可以对比一下和上述表的关系：

下面是一个区块，里面包含了一笔交易。

 复制代码

```

1
2 {
3   "header" :
4   {
5     "result" :
6     {
7       "bits" : "7097242144892",
8       "hash" : "cb36f2a1cbbf6a6300f4bf4915a5f54476ab603f2703a99e5d8d2db7ae2b37ed",
9       "merkle_tree_hash" : "3457b988bc6b61a7ad803f0742a68064c622ec618b833d99d153b92cba264d!",
10      "mixhash" : "47266114351983928450891657703600980449927404535067001902399906817438963!",
11      "nonce" : "1864926684099479906",
12      "number" : 1000000,
13      "previous_block_hash" : "049257f31f4412bf115ed44a9305012ccea888cf842c2f0b66a528f2580:",
14      "time_stamp" : 1520339120,
15      "transaction_count" : 1,
16      "version" : 1
17    }
18  },
19  "txs" :
20  {
21    "transactions" :
22    [
23      {
24        "hash" : "3457b988bc6b61a7ad803f0742a68064c622ec618b833d99d153b92cba264d53",


```

```

25     "inputs" :
26     [
27     {
28         "previous_output" :
29         {
30             "hash" : "0000000000000000000000000000000000000000000000000000000000000000",
31             "index" : 4294967295
32         },
33         "script" : "[ 0340420f ]",
34         "sequence" : 0
35     }
36 ],
37 "lock_time" : "0",
38 "outputs" :
39 [
40 {
41     "address" : "MUiW2CViWLQBg2TQDsRt1Pcj7KyrdqFPj7",
42     "attachment" :
43     {
44         "type" : "etp"
45     },
46     "index" : 0,
47     "locked_height_range" : 0,
48     "script" : "dup hash160 [ e45695c2c390625376a7225a7ebea90dbb4147cf ] equalverify (
49     "value" : 270750000
50 }
51 ],
52 "version" : "1"
53 }
54 ]
55 }
56 }

```

我们可以发现区块头部分的数据被存储到 `tb_block` 表中，然后交易哈希被存储的 `tb_tx` 表中，接着交易的输入输出被存储到 `tb_input` 和 `tb_output` 中，这三者是通过区块高度、交易哈希被链接起来的。

 复制代码

```
1  tb_block <-- 区块高度 -->  tb_tx <-- 交易哈希 --> tb_input/tb_output
```

完整的 Python 脚本可以通过这个链接查看：

整体的思路是使用 getblock 的 JSON-RPC，从第 0 个高度的区块一直扫描到最新区块，并且存储到 MySQL 中。

这里最难以处理的问题是保持 MySQL 中的区块数据与全节点数据的一致性，也就是当区块链分叉时，MySQL 需要感知到发生了分叉，接着移除被分叉的区块，并且接着同步到正确的区块上。

这个处理方法有不同的思路，上述脚本使用了移动区块数据的方法，也就是将孤儿块移动到 tb_tx_fork 下，接着同步正确的区块。实际上也可以通过标记法，即在 tb_block 中，将此块标记为孤儿块。

关系型的表结构也可以做成标准化的，区块链本身作为基础设施，历史交易已经不可篡改，如果把这些结构化的区块做成公共基础设施，并提供基于 API 的开放调用，这便就是我们常见的区块浏览器了。

上文我们介绍了扫描区块的思路和实践，实际上我们也可以使用 Presto 技术将钱包中的数据转换成类 SQL 查询，但这里服务的稳定性和性能需要经过测试才可以被平台使用。

扫描区块技术解决了所有区块链资产可视化、高并发查询的问题，所以它在一些大规模的数字货币交易所中也有应用。区块浏览器就是基于这种技术产生的一种 Web 服务，下面我们来看一看区块浏览器与扫描区块的具体关系。

区块浏览器

我在前面介绍数字货币和交易所时有提到过区块浏览器，它提供了可视化的交易查询和验证服务。

从技术上看，一个区块浏览器的主要工作就是把区块扫描到数据库服务器中，然后搭建一个 Web 访问服务，用户只需要输入交易哈希或者区块哈希，即可查询到交易是否已经被打包和确认。

目前比特币和以太坊的流行区块浏览器比较多，不局限在某一个区块浏览器，因为大家看到的区块数据是一样的，区别就是如何更好地展示，做得更好的话，还可以集成一些咨询和资产托管的功能。

从产品意义上来说，我认为区块浏览器更适合叫做资产浏览器，因为它为人们提供了资产证明的服务，而不必肉眼识别交易或者自行手动解析交易，一般来说，区块浏览器也提供基本的 API 查询服务。

区块浏览器也为人们提供了区块和交易的统计数据，帮助人们直接地了解这个区块链的活跃程度，人们也可以根据统计数据制作区块活跃度等指数帮助投资者了解这个区块链项目。

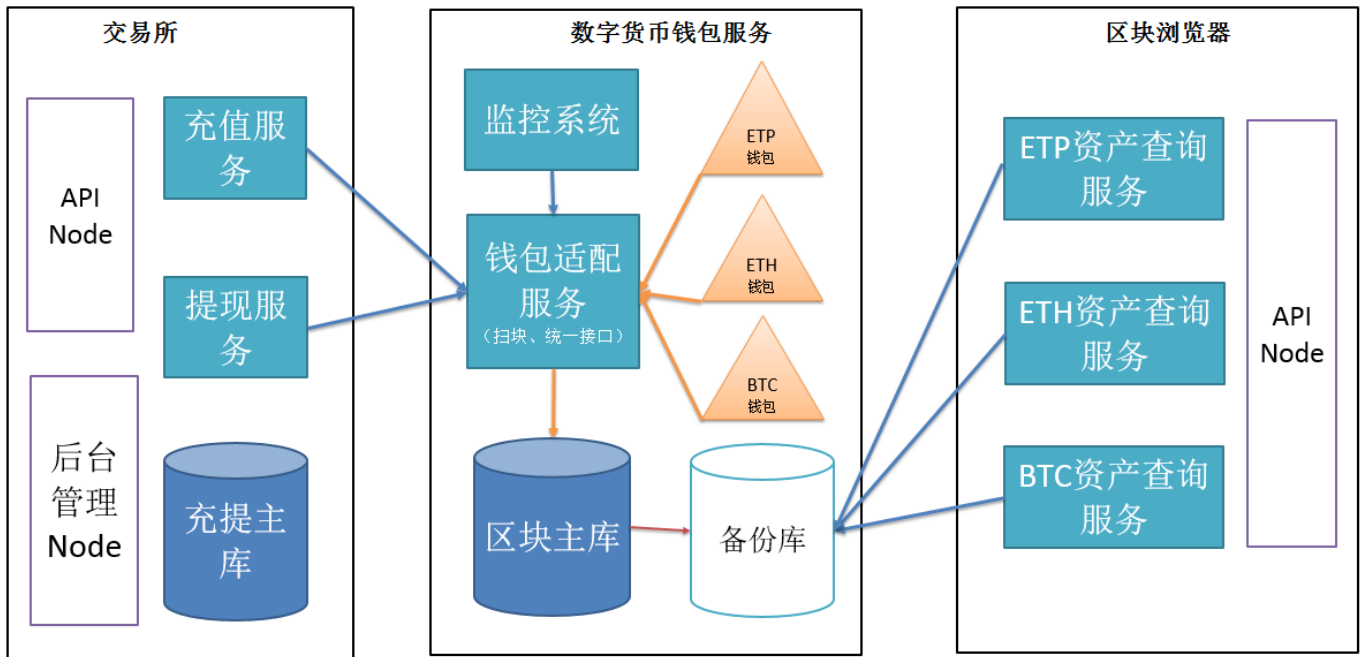
区块浏览器降低了普通人查询和验证交易的门槛，其实它也是整个区块链行业的配套基础设施，而对于平台来说，从第三方获取交易验证始终是一件不安全的事情，也面临着中心化的风险，那么平台如果想搭建自己的交易查询和验证服务，需要如何操作？

这就需要把数字货币钱包服务，集成到自己的系统里。下面我们就来聊一聊具体是如何集成的。

数字货币钱包服务

实际上，大规模的区块链应用都需要搭建一个数字货币钱包服务，数字货币钱包服务为系统中的其他模块提供了可扩展的、统一的、安全的交易查询和验证服务。

下图是我从交易平台开发归纳出来的数字货币钱包服务。



数字货币钱包服务可以为交易平台其他模块提供接口统一的 API，同时将不同的数据结构化到数据库服务中，最后可以通过传统高可用手段完成交易查询和验证。

当然，这也和交易所的规模有关，如果是一个小型交易所，扫块可能不是必需的，但是统一接口的 API 却是必须的。

我认为数字货币钱包服务应当有一套标准的钱包服务框架，支持主流数字货币，从而降低大家的使用和部署门槛，这也和我们上一篇聊到的“区块链即服务”的概念不谋而合。

可以说区块链的配套设施和技术还很原始，还有很大的发展和提升的余地。

总结

好了，今天我们先了解了一下数字货币钱包的分类，接着详细讲解扫块技术，然后又谈到了区块浏览器，最后分享了一下数字货币钱包服务的集成思路，希望可以让你对区块服务的实践有一个初步了解，你也可以根据已有的技术知识重新拆解和分析区块链技术。

那么今天的问题是，数字货币钱包服务可以应用到微服务架构中吗？



深入浅出区块链

你的区块链入门第一课



陈浩 元界 CTO

新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第30讲 | 区块链即服务BaaS

下一篇 第32讲 | 区块链与供应链（一）

精选留言 (7)

写留言



阿痕

2018-06-05

4

对区块链有写操作的应用，很难用微服务，因为是有状态的。而区块链的读服务或基于中心库查询的web服务，完全可以用微服务。



镜子

2018-06-05

2

能不能直接撸代码，边撸边讲

展开



沃野阡陌

2018-06-05

1

请问老师，从交易所提币出来到钱包自己保存，是不是就是您说的可以用区块链browser来看或验证是否已被打包成区块的过程？

作者回复: 是的，完全正确。



Ender0224

2018-06-04

1

这里最难以处理的问题是保持 MySQL 中的区块数据与全节点数据的一致性

这个是定期扫描么 还是时时刻刻有变更就扫

展开

作者回复: 取决于业务敏感性吧。一般是做分叉检测和控制入库高度，不过应该不是最优解决方案



绝露

2018-06-26

1

区块浏览器的查询功能好理解，但是验证是指什么，能解释下吗？

展开 ∨



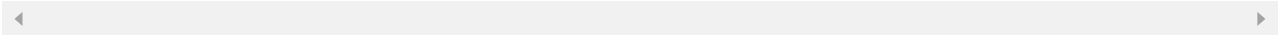
Geek_f26a6...

2018-06-18



初次读关于区块链的文章，对于区块链要解决的问题一直困惑。数字货币在公链上发行如何匹配各国的货币发行，发行规模应于经济体总量相关。就象美元与黄金挂钩，支付宝钱包里的钱与实际货币挂钩。问题若不对或对区块链误解请见谅

作者回复: 你好，同意的。但也未必是货币，也可以是其他有价凭证。



乔良qiaoli...

2018-06-09



有机会介绍一下infura.io就太好了

展开 ∨