EOS 块数据结构：块内通信和跨链

EOS 编译、测试指南（三）

扫一扫关注欧链小秘书

关注时请注明“技术社区”

EOS 团队于 2017 年 7 月 28 日推出了单机测试版，欧链团队也在第一时间对代码进行了编译和测试，陆续发布了两版《EOS 编译、测试系列指南》，详细介绍如何让 EOS 在自己的本地跑起来以及智能合约编写、代币的发行。一周以来，EOS 团队陆续在 Git 上更新了自己的代码，欧链团队也在持续跟进 EOS 代码，编译和测试，以下将分析 EOS 中重要的数据结构 block，并解释 EOS 的两大优势：高效账户间通信和跨链机制。

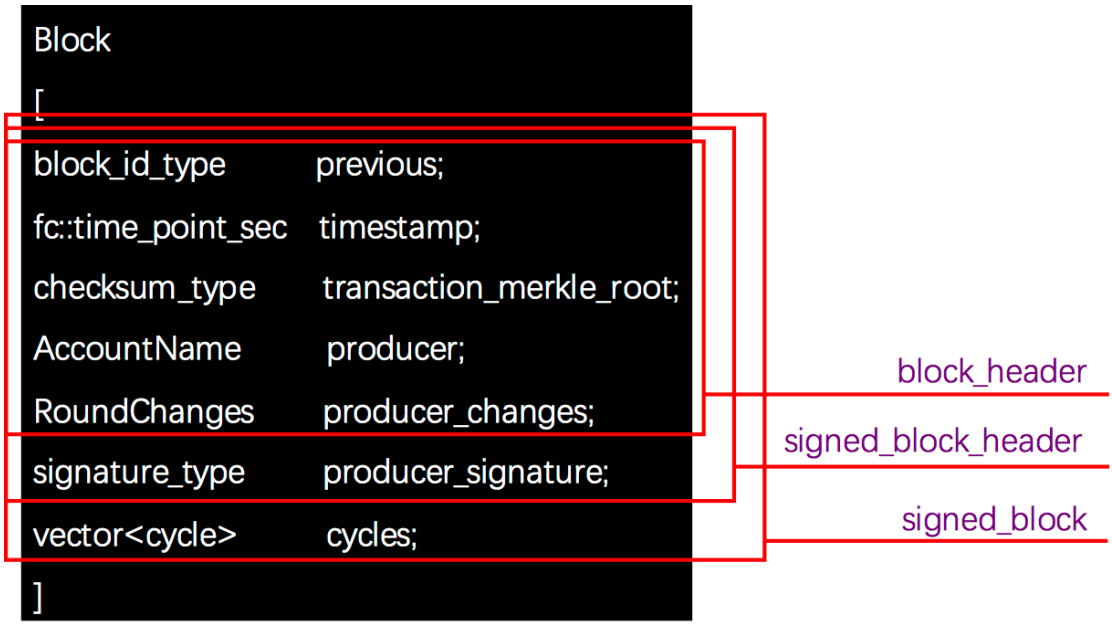
该指南主要依据 EOS 的官方文档，以及欧链团队的实践经验编写，开发者有任何技术问题可以关注欧链小秘书、在欧链科技社区里向 OracleChain 团队提问或者发送邮件到 contact@oraclechain.io。

EOS 文档：https://eosio.github.io/eos/

EOS 代码：https://github.com/EOSIO/eos

EOS 白皮书：https://github.com/EOSIO/Documentation

一、块主要结构



block 是区块链中最重要的数据结构，EOS 在 block 结构中设计了 7 个参数，下面我们将一一进行分析解释。

previous

previous 为指向前一个区块的 hash 值，EOS 采用的 hash 算法是 SHA。

timestamp

timestamp 为时间戳，该区块的生成时间。

transaction_merkle_root

transaction_merkle_root 为该区块内所有交易的 merkle 根，用于快速验证交易的完整性。

producer

producer 为该区块的生成者。EOS 采用 DPOS 共识机制，预计每 3 秒生成一个区块。任何时刻，只有一个生成者被授权生成区块。

producer_changes

EOS 架构中区块的产生是以 21 个区块为一个周期,在每个出块周期开始前，21 个区块生成者会被投票选出。producer_change 就记录了未来的 21 个出块者。

producer_signature

producer_signature 是该区块的生成者对该区块的签名。EOS 采用的是 ECDSA 签名算法，使用的椭圆曲线参数是 secp256k1。

cycles

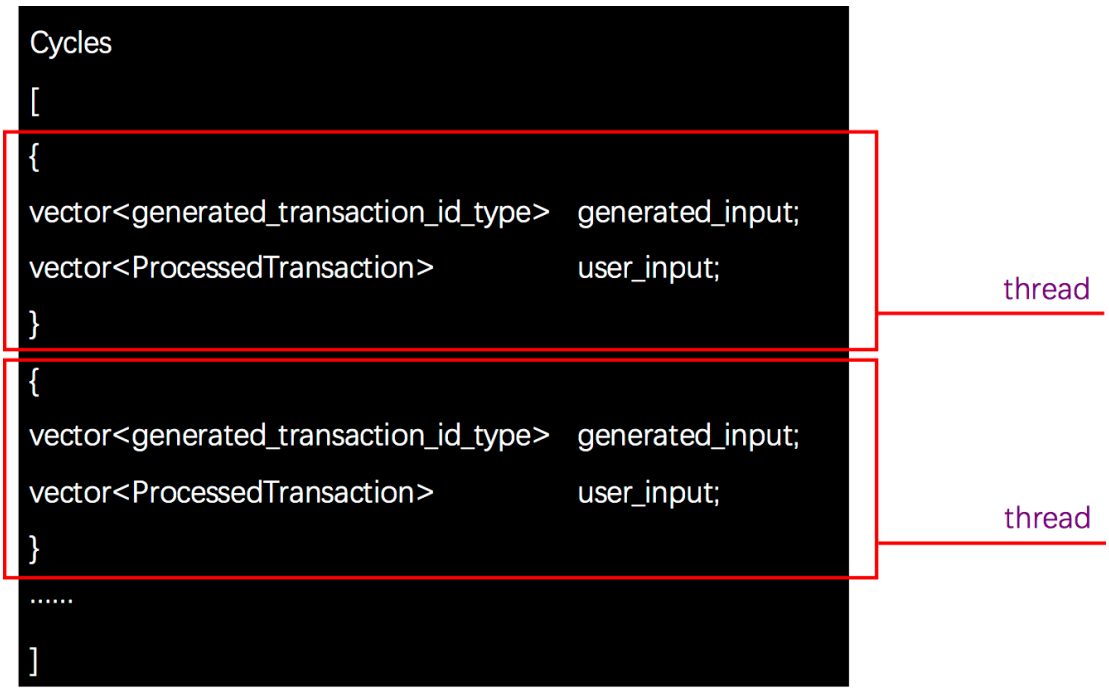
EOS 设计的目标之一是使得两个账户（合约）能够在单个区块内来回交换消息（交易），而不必在每个消息之间等待 3 秒。为了实现这一点，EOS 将块内的消息分成了 cycle 来顺序处理。cycles 就是 cycle 的一个顺序向量，那么 A 发给 B 的消息在 cycles[1]中处理，B 返回的消息就可以在后续的 cycles[4]中进行处理。区块生成器将不断把 cycle 添加到区块中，直到最长的区块时间间隔达到，或者没有新的可传送交易生成。

二、Cycles 主要结构

前面提到了 EOS 为了提升消息交互的效率，在 block 中设计了 cycle 结构来实现了一条“小链”。而在 cycle 内部，EOS 设计了 threads 来实现消息、交易的快速并行处理。

```
Block
  Cycles (sequential)
    Threads (parallel)
      Transactions (sequential)
        Messages (sequential)
          Receiver and Notified Accounts (parallel)
```

相比较 cryptonomex 中的设计，EOS 的这种设计将更好的支持高并发管理，利于提升 EOS 的整体效率。



generated_input

generated_input 包含了一组 GeneratedTransaction 的 id，用于顺序查找和处理 GeneratedTransaction。我们将在后面对 transaction 的类别进行详细的解释和分析。这些 transaction 将被顺序处理。

user_input

user_input 包含了一组 ProcessedTransaction，这些是已经被处理过的交易。

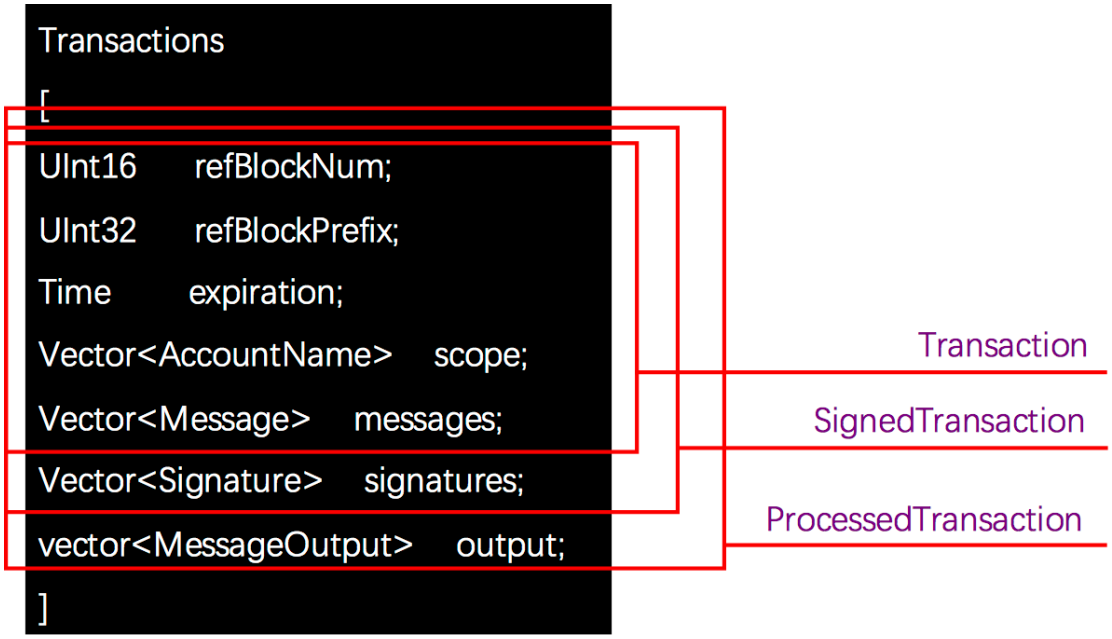
thread

不同的 thread 将会根据主机的实际情况被并行处理。在同一个 cycle 内，当两笔交易 A 和 B 同时涉及到同一个账户时，这两笔交易将会被放在同一个 thread 下进行处理。

三、Transaction 结构

EOS 中主要使用三种 transaction 结构。其中 SignedTransaction 是由用户发出的交易请求，ProcessedTransaction 是由区块生成者处理完用户发出的交易请求后生成的交易实体，GeneratedTransaction 是由区块链生成的交易请求，特别的是由

智能合约生成的交易请求。



Transaction

Transaction 作为所有 transaction 的父类，包含了 5 个参数。refBlockNum 用于指定之前的一个块，其中将包含一个由该交易签名者申明的已知状态申明，作为该交易执行的上下文。refBlockNum 只使用 16 个 bit，所以只能回溯到两天前的 65,536 个块。refBlockPrefix 就是被指定的块的 hash 的前 4 个字节（32 个比特）。expiration 给出了交易的内存过期时间，这样将保证节点的内存不用保存过多的交易。

scope 内将记载此次交易涉及到的账户信息。

messages 内将记载此次交易的实质内容，它可以包含多个消息。在设计上，只有所有的消息都被接受了，整个交易才会被接受，任何一个消息被拒绝都会导致此笔交易失败。这些消息将会在同一个线程里面顺序执行，因此将多个消息放到一笔交易里面在性能上并不理想（比如同时给两个人转账）。而且 EOS 会根据交易涉及到的用户数量数来收取手续费，因此涉及到多个账户的交易的时候，最

好采用原子交易来进行（每笔交易只包含一个接收账户）。

SignedTransaction

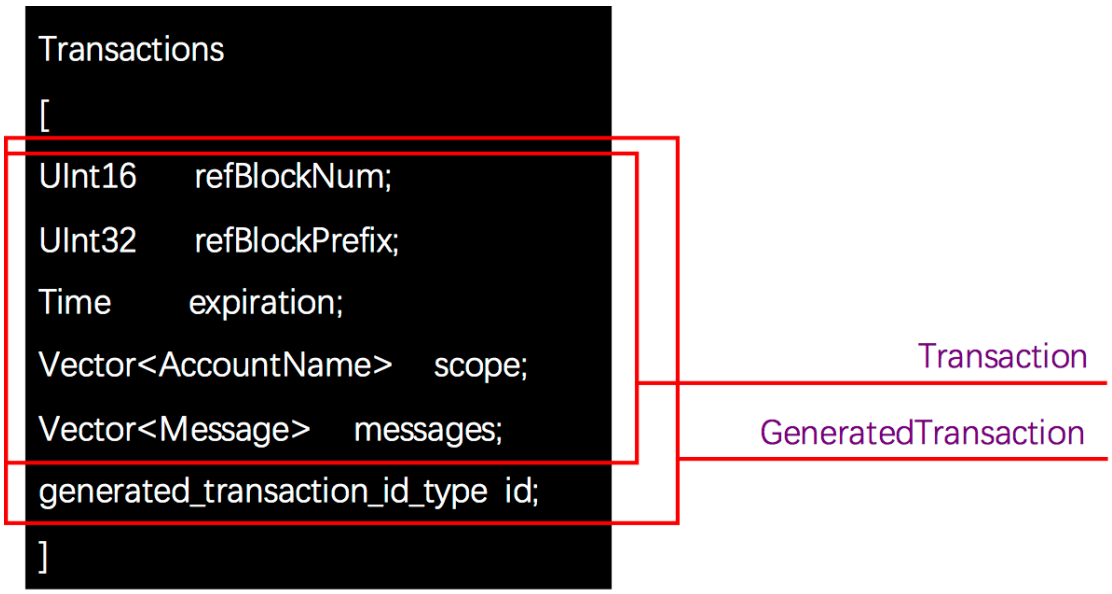
SignedTransaction 比 Transaction 多了一个参数 signatures，将用于保存用户对该交易涉及的所有签名，每一个 message 至少对应一个 signature。

ProcessedTransaction

ProcessedTransaction 比 SignedTransaction 又多了一个参数 output，用于记录对用户发出的 SignedTransaction 进行处理后的输出。特别的，这个输出包含一条或多条交易信息 GeneratedTransaction。

GeneratedTransaction

GeneratedTransaction 是由区块链生成的交易，比如智能合约。它比 Transaction 多了一个参数 id，用于方便查找到该交易。



值得注意的是，因为这个交易是由区块链（智能合约）产生的，它可以用于 EOS 上不同应用间的通信。但又是因为它由区块链（智能合约）产生，因此它并不含有签名，所以它的验证必须要连同触发产生该交易的智能合约的交易一同被验证。

例如，当多家大宗交易所、普通用户向 OraleChain（智能合约）提交了目前

的猪肉价格（以 SignedTransaction 形式）后，OraleChain 根据自己的 PoRD 机制计算得到一个可信猪肉价格，然后连同所有搜集到的价格生成一个 GeneratedTransaction，作为最后一个提交价格的交易的处理结果 ProcessedTransaction 的输出信息，发送给另一个智能合约 AgriculturalInsurance，AgriculturalInsurance 就可以根据这个可信猪肉价格进行保险赔偿等智能操作。

正是这样的巧妙设计，实现了 EOS 上不同的智能合约间的通信，也就是我们常说的跨链，这将极大的丰富 EOS 的生态圈，为开发者开发更丰富的应用提供了可能。

四、Message 结构

```
Message
[
AccountName code;
FuncName type;
Vector<AccountPermission>  authorization;
Bytes data;
]
```

Message 用于记录交易的主要内容，包含了 4 个参数。

code

code 指向了执行这个消息的智能合约。比如“eos”是指 EOS 自身自带的智能合约，在前面系列指南里进行 eos 转账的操作，出发的就是这个合约。

type

type 指向需要执行的智能合约里的方法，比如“eos”里面的转账操作“transfer”。

authorization

authorization 记录了处理该消息所需要的权限。EOS 里面目前设计了三种权限：owner、active 和 recovery。owner 权限可以做所有事情，active 权限可以做出了更改所有者之外的所有操作。recovery 可用于 EOS 用户在密钥被盗时恢复其账户控制。

data

data 里面则存储了这个消息的其他参数。

五、MessageOutput 结构

```
MessageOutput
[
vector<NotifyOutput> notify;
vector<ProcessedTransaction>  sync_transactions;
vector<GeneratedTransaction>  async_transactions;
]

NotifyOutput
[
AccountName name;
MessageOutput output;
]
```

当智能合约处理一笔 SignedTransaction 时，有可能会生成一笔新的交易 GeneratedTransaction，它将被存储 MessageOutput 结构体中。

MessageOutput 包含了被通知对象集合 notify，在完成通知以后被应用了的 ProcessedTransaction 集合 sync_transactions，以及由智能合约生成的不带签名的交易 GeneratedTransaction 集合 async_transactions。

通知对象集合结构 NotifyOutput 里面包含了被通知的对象 name 和一个输出消息 MessageOutput。这和前面形成了一个嵌套关系，因为被通知的对象（合约）

还有可能生成新的交易，如此递归下去。

由于 EOS 代码中对 MessageOutput 和 NotifyOutput 的处理还没有完全完成，我们会在后面的系列文章中做更详细的解释和分析。

六、总结

EOS 在块内设计了 cycle 结构，实现了块内的通信，这样账户（合约）之间的通信将变得更加快捷。

在设计上，EOS 是模块化的，每个人不应该运行所有的东西，因此不能假定另一个账户（合约）的状态在同一台机器上是可访问的，这就意味着如果允许一个合约同步调用另一个合约,但如果这个合约不在内存中,系统将会崩溃。因此，EOS 要求所有账户间的通信都必须通过区块链上的交易进行传递。

为了实现跨链机制，EOS 特别设计了不带签名即可完成验证的交易机制，这种验证是通过状态信任链来完成的:第一个带签名的交易触发合约产生了第二个不带签名的交易，该交易又顺序产生了第三个不带签名的交易，依次下去，最后所有的交易的验证都依赖于第一个交易的签名，和每一次合约执行后的状态。

从对 EOS 块数据的分析，我们可以看到 EOS 在设计上就已经考虑了高速、融汇、跨链等区块链未来的发展方向，我们有理由相信 EOS 代表了区块链的一种新的未来。

OracleChain 团队
2017 年 8 月 25 日