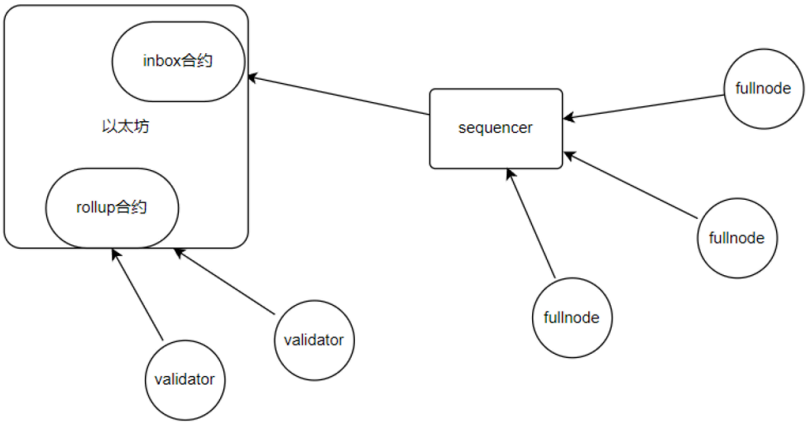
Arbitrum开发者指南

1、Arbitrum架构

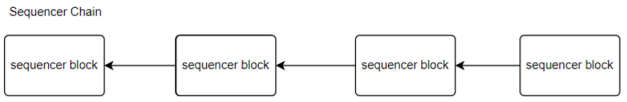
架构overview



Arbitrum 架构

1. Sequencer chain

Sequencer Chain指的是Sequencer（定序器，对二层交易进行打包排序的全节点，相当于二层网络的出块节点，不同于其他的layer1网络，Sequencer节点不需要通过共识算法来竞争出块权，而是直接具有打包区块的权力，目前Arbitrum的Sequencer由官方运行，出块权完全由官方掌握）对Rollup中交易进行打包进layer2区块并用链式结构连接起来的区块链，其数据结构和一般区块链的数据结构类似，Layer2的交易数据都通过Sequencer Chain来储存，用户发起一笔交易，通过RPC（远程过程调用）将交易发送给RPC node，RPC node再转发给Sequencer节点，Sequencer在接收到用户的交易之后立即对交易进行打包处理，实现layer2交易的初级确认。在收集到一段时间内的交易以后，Sequencer会将这段时间内的交易组成一个批次，经过压缩以后以calldata的形式将交易数据上传到Arbitrum部署在以太坊Layer1的Sequencer inbox合约中，以实现将数据可用性放在以太坊主网，而将执行迁移到layer2的设计，至此，实现了layer2的第二步确认。



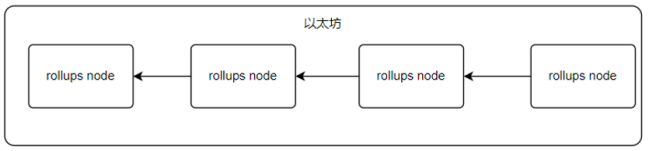
Sequence chain是抽象出来与其他区块链一样的链式数据结构。

Arbitrum Sequencer Chain

2. Rollup chain

前文我们提到，layer2的交易在Sequencer被打包以后以calldata的形式传到layer1的inbox合约中，那谁来替我们验证存到layer1的数据是否是真实的呢，Sequencer在上传的时候是否加入了虚假交易，这个时候就得说到Optimistic Rollup特殊的欺诈证明机制了。Sequncer传到inbox合约的数据都是公开透明的，可以被全网的参与者监督，这也是欺诈证明的基础。因为作恶者永远不知道有多少参与者在监督着链上的数据。这时候引入了另外一条链，即Rollup Chain，你也可以将之称为验证者链。每一个人在条件允许的情况下，都可以根据inbox合约中的calldata数据在自己的本地运行一条Sequencer链，因为inbox合约中的数据是还原layer2状态的最少数据，人们可以在本地运行所有的交易，以确保Sequencer没有作恶，但是真的要所有人来维护Arbitrum状态吗？显然是不可能的，只有类似交易所这样在Arbitrum拥有大量资产的参与者会自己在本地来维护Arbitrum状态，来确保自己的资产在Arbitrum是安全的。这就是为什么说会有很多的Sequencer Chain的原因，只不过仅有Sequencer自己运行的那条Sequencer链才是主要维持网络的链。而Rollup Chian则是网络中的验证者对于网络状态质押产生的链，其运行在以太坊主网，比方说在一段时间内，验证者会对Sequencer上传到Inbox合约中的状态进行质押，即验证者押注正确的交易，通过质押ETH发布断言，声称质押这段交易历史是正确的，如果所有的交易都是正确的，那么链的状态将会持续推进更新，可如果此时有人做恶，质押了错误的交易状态，这个时候网络中的正确验证者和作恶者产生冲突，正义验证者会押注正确的状态，此时Rollup chain出现分歧，正义验证者和作恶者通过类似二分法的方式找到存在争议的交易，查找争议交易数据的这些操作在layer2执行，当明确争议区间以后，由部署在layer1的Rollup合约重新执行争议交易，并裁决双方的断言，最后作恶者会被罚没质押的ETH，其中的一部分会奖励给正义

验证者，以奖励其对网络的保护。为了防止作恶者攻击网络，罚没的ETH会有一部分销毁，防止正义验证者和作恶者串通，零成本攻击网络。而这就是Rollup　Chain和欺诈证明。简单叙述就是网络中的众多参与者都在本地运行完整的layer2的状态，当发现有恶意交易企图篡改链的状态的时候，众多参与者会通过欺诈证明推进链的状态朝着正确的方向前进，而欺诈证明的挑战期为7天，7天以后交易状态确认，无法再进行修改。这是layer2交易的第三级确认，在以太坊主网不分叉的前提下，第三级确认即为layer2交易的最终确认状态。



Rollups Chain为验证者质押链

Arbitrum Rollup Chain

2、Arbitrum与Ethereum的一些区别

1. block.timestamp与block.number

在Layer1里面随着新区块挖出会实时更新block.timestamp与block.number，且同一区块内所有交易里block.timestamp与block.number的返回值都是一样的，返回的是当前区块高度的信息。

在layer2里面，并不是返回layer2这条链的block.number,而是返回的layer1的block.number.

因为layer1上面的区块更新速度是低于layer2的，layer2每秒钟都会更新区块，而layer1的区块是约14s更新一次，可能会导致Arbitrum上面某一段时间的区块调用block.number返回的都是同一个layer1区块的返回值，不会随着layer2区块的更新而更新，而是layer1的区块更新以后才会更新。

以下两个逻辑需要注意代码的编写：

（1）　需要防止用户单区块多次调用。

（2）　需要使用区块高度来判断相关逻辑。

2. block.coinbase

在layer1上面block.coinbase是矿工的地址，比如某区块是A矿工挖出的，那么block.coinbase的返回值就是矿工的地址。但是在layer2上面是没有

共识机制和矿工的，所以该返回值为0，不再返回矿工地址。为了保证开发者在Arbitrum部署合约时实现字节码层面的完全兼容，所以Arbitrum保留了block.coinbase，

3. block.difficulty固定在2500000000000000

block.difficulty和PoW息息相关，是矿工出块的难度值，因为在layer2不需要矿工共识出块，所以Arbitrum把这个值硬性固定为2500000000000000

4. Fullnode异同

同：

A. 都维护网络的全部状态。

B. 所有人都可以直接参与。

C. 运行全节点没有激励。

D. 可以转发用户的tx。

异：

A. Sync

Layer1的同步是全网同步，sync是往sequencer同步或者使用快照进行状态同步。

B. peers节点

layer2的fullnode没有peer节点，无P2P通信模式。只连接sequencer的Feed节点（sequencer的信息同步节点）和Layer1的RPC节点，其他的节点是不连接的。

3、Arbitrum Gas机制

1. gas计算

L1 fixed cost和L1calldata主要是放在layer1上面存档的开销，即用户的交易数据在压缩以后上传到inbox合约所需要在layer1支付的gas花销，也是二层协议的主要gas开销。

L2 computational cost和L2 storage cost是在layer2执行开销和状态变化存储的交易开销，在计算layer2交易的gas开销的时候是需要将以下四项的开销加起来计算的。

1）L1 fixed cost

2）L1 calldata cost

3）L2 computational cost

4）L2 storage cost

2. gas limit

layer1的gas limit受限于区块大小，但是layer2并不是这样，Arbitrum现在的打包方式是先到先打包，所以并没有对gas limit有太多限制，而是使用了arbGas，这里的arbGas仅仅指的是L2 computational cost，可以看到每秒钟定序器能处理的arbGas limit是120K，而arbGas pool就和layer1区块的gas limit一致，每隔一段时间就会将arbGas pool加满，而不是像layer1一样更新，如果在一段时间内arbGas pool消耗完了，为了保证layer2的稳定和防止DDoS攻击，定序器就会拒绝交易。对于单笔交易的gas限制是2.4m。

·arbGas limit per second：120K

·arbGas pool：20m

·arbGas limit per tx：2.4m

3. gas estimate

layer1有JSON-RPC，layer2也有JSON-RPC，gas estimate指的是每次通过JSON-RPC发起一笔交易的交易gas费预估。由于layer2的gas开销和layer1息息相关，是L1+L2的gas花费，而且L1的gas是随时间波动的，所以有可能A时间获得的gas estimate在B时间就无法通过，所以最好是在每次发起交易前，都调用一次RPC获得新的gas estimate。

4、Arbitrum Rollup合约解析

1. 什么是Rollups合约

A、Validator

主动验证者

通过提议新的Rollup区块来推进Rollup链的状态，由于提议Rollup区块需要质押，因此活跃的验证者需要一直提供质押，一条链只需要一个诚实的活跃验证者就可以不断推进Rolllup链的状态更新，因为它将始终会质押在正确的状态上面，当有攻击者质押错误状态的时候就会触发挑战。

防御验证者

防御型验证者并不主动参与Rollup链的状态更新，而是监视着Rollup协议的运行，如果主动验证者所提出的Rollup区块都是正确的，那么防御验证者不会参与网络。但是如果有恶意验证者提议了不正确的Rollup区块，那么防御验证者将会自行提议正确区块或者在其他已经提议的正确区块上面质押来干预。最后通过Rollups合约中的Challenge合约来裁决。在这个过程中，是存在经济博弈的，即作恶验证者需要质押资产才能提议新的Rollup区块，如果其质押了错误的状态，那么防御验证者可以通过质押正确的状态来获得作恶验证者被罚没的质押作为奖励。同时，为了防止作恶验证者和正义验证者串通进行无成本的作恶，正义验证者在接受作恶验证者的质押作为奖励时，仅能获得作恶验证者被罚没资产的一半。

瞭望塔验证者

瞭望台验证者从不质押，它只是监视Rollup协议网络，如果有恶意验证者提议了不正确的区块，瞭望塔验证者会发出警告，比如告知防御验证者去质押正确的状态来获取作恶验证者的质押作为奖励。

B、Rollups区块

由验证者打包生成的区块，主要信息为Rollups相关信息，比如挑战根，L2->L1的消息根，质押信息等。

```
contract Node {
    bytes32 stateHash;
    bytes32 challengeHash;
    bytes32 confirmData;
    uint64 prevNum;
    uint64 deadlineBlock;
    uint64 noChildConfirmedBeforeBlock;
    uint64 stakerCount;
    uint64 childStakerCount;
    uint64 firstChildBlock;
    uint64 latestChildNumber;
    uint64 createdAtBlock;
    bytes32 nodeHash;
}
```

Arbitrum Rollup区块

C、Rollups合约

Rollups合约主要负责对Arbitrum上的Rollups区块，L2->L1的Withdraw交易以及欺诈证明负责。

·Node工厂合约（Nitro升级之后为lib）

用于创建Node合约

·Node合约（Nitro升级之后为Struct）

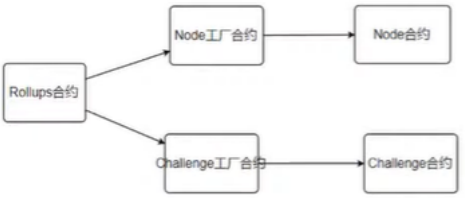
即Rollup区块，需要注意的是每个Rollups区块都有一个对应的合约。

·Challenge工厂合约

用于创建Challenge合约。

·Challenge合约

发生Challenge的合约，每个Challenge对应一个合约



Arbitrum Rollup合约

2. Rollups主要函数

·CreateNewNode

创建新的rollup区块需要调用的函数。

```
function createNewNode(
    RollupLib.Assertion calldata assertion,
    uint64 prevNodeNum,
    uint256 prevNodeInboxMaxCount,
    bytes32 expectedNodeHash
)
```

·CreateNewStake

网络加入一个新的验证者进行质押存款需要调用的函数

```
function createNewStake(
    address stakerAddress,
    uint256 depositAmount
)
```

·ConfirmNode

七天挑战期结束以后，确定一个区块内交易合法需要调用的函数。

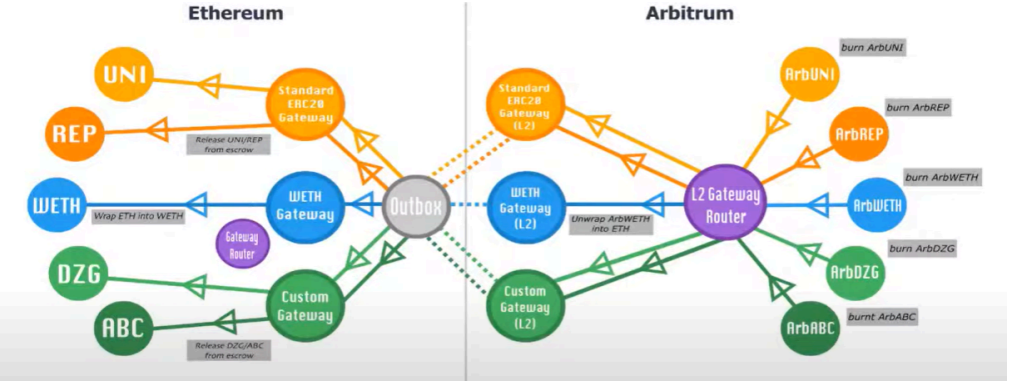
```
function confirmNode(
    uint64 nodeNum,
    bytes32 blockHash,
    bytes32 sendRoot
)
```

5、Arbitrum Inbox&Outbox合约解析

1. Outbox合约

负责处理L2到L1的消息，一笔交易从L2回撤到L1会有验证者质押来确保交易的合法性，当七天的挑战期结束以后，用户需要到Outbox合约去

claim自己回撤的资产。



Arbitrum 资产桥原理

举例说明，当L2的ArbUNI需要回撤回L1的时候，会在L2上面销毁ArbUNI，因为UNI是ERC20的代币，没有其他类似算稳的回锚功能，所以ArbUNI通过标准的ERC20的gateway关口进行转移，当到达网关以后会有专门的验证者对这笔交易进行质押，七天挑战期结束以后，资产可以传送到L1的Outbox，用户在Outbox认领，便可以得到锁仓代币的返还。

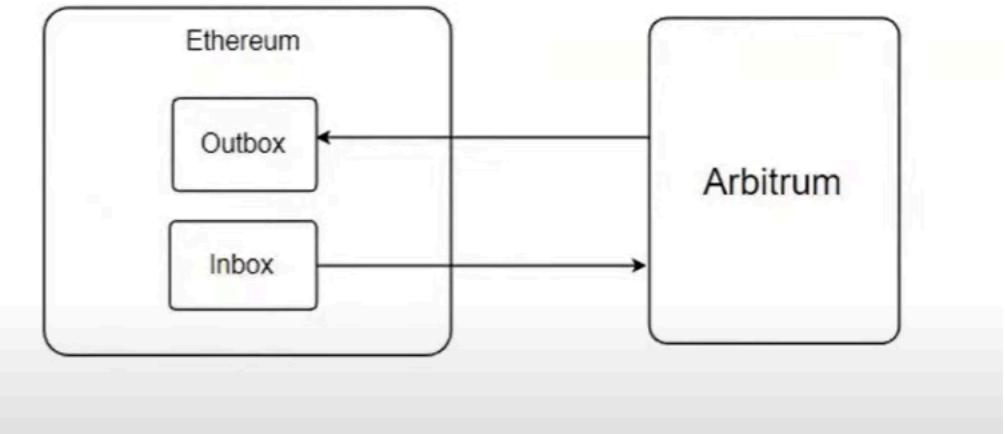
像ArbABC这样的算法稳定币是标准的ERC20的接口但不是ERC20的内部实现方式，里面有很多的合约，我们不能按照ERC20的方式去withdraw，因为不知道Custom Gateway的内部是什么样的，所以不能部署标准的合约去销毁。

简而言之，就是通用标准的代币可以直接使用标准的跨链到L2，但是有复杂机制的代币需要客户自定义部署到L2。

2. Inbox合约

A. 负责处理L1到L2的消息，L1在传递消息到L2的过程中，由ArbOS将消息发送到Sequencer进行处理，由此将消息传递到L2。

B. 负责记录Arbitrum网络中的交易存档



Arbitrum Outbox&Inbox合约

3. Arbitrum delayed inbox所支持的交易类型

1. Message type 3：L2 message

   L2 message是delayed Inbox的功能，不是跨链消息，是记录L2交易存档的type。

2. Message type 7：L2 transaction funded by L1

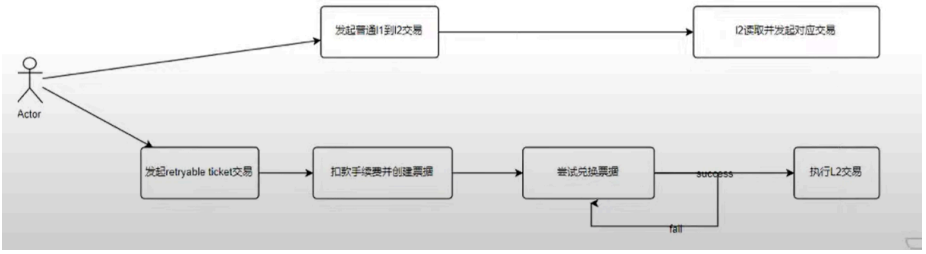
   L2 transaction funded by L1也是L2 message，不同的是其msg.value是通过L1扣款的。

3. Message type 9：Send tx to retry buffer

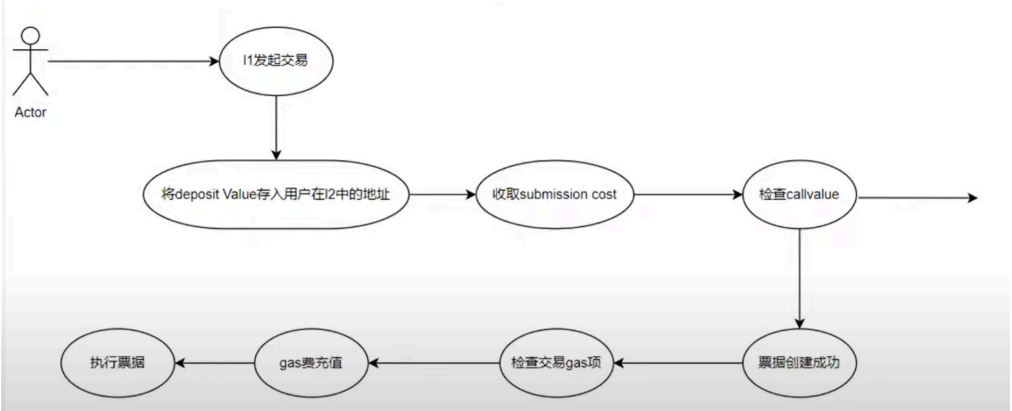
   L1到L2消息的type，通过可重试票据（Retryable）发送消息。

A. 普通跨Layer交易：实现简单粗暴，在layer1发起普通交易到layer2，layer2读取并发起对应交易，如果gas不足，则交易容易失败。

B. Retryable交易：如果执行失败可多次执行。在layer1发起交易到layer2，可能因为gas波动面临gas不足难以创建交易的情况，所以引入可重试票据（Retryable）。



Arbitrum 跨链交易类型



Retryable的生命周期

Deposit Value：用户充值进入layer2的资产

Submission cost：对于可重试票据的收费

callvalue：用户可能在Retryable自己写入一个值，callvalue会和地址余额比对，如果充足就创建成功

6、Arbitrum Precompile

Precompile合约的Address值并不是合约的哈希值而是Arbitrum官方自己编写的值，涉及到的代码是链底层的代码。以下两个例子简述两个precompile合约的作用。

ArbSys：获得L2上面区块的信息，比如网络状态，区块高度等。

ArbRetryableTx：可以获得一些可重试票据的内容。

	Address
ArbSys	0x0000000000000000000000000000000000000064
ArbRetryableTx	0x000000000000000000000000000000000000006E
ArbGasInfo	0x000000000000000000000000000000000000006C
ArbAddressTable	0x0000000000000000000000000000000000000066
ArbStatistics	0x000000000000000000000000000000000000006F
NodeInterface	0x00000000000000000000000000000000000000C8
ArbBLS	0x0000000000000000000000000000000000000067
ArbInfo	0x0000000000000000000000000000000000000065
ArbAggregator	0x000000000000000000000000000000000000006D
ArbFunctionTable	0x0000000000000000000000000000000000000068

Arbitrum Precompile合约列表

7、Nova背后的Anytrust

1.数据可用性委员会

众所周知，Arbitrum one采用的是Rollup技术，定序器会将一段时间内的交易打包为一个batch，通过压缩交易将数据以calldata的形式上传到以太坊主网的inbox合约中，此时Arbitrum one上面交易数据的可用性是通过以太坊主网来存储和保证的，这也是其安全性高度接近以太坊主网的原因。而Anytrust与rollup的区别就在于Anytrust引入了数据可用性委员会，将原本应该存储在inbox合约中的calldata数据转移到链下的数据可用性委员会存储。在Anytrust技术模式下，用户将交易发送到Sequencer以后，Sequencer会将交易数据发送到数据可用性委员会，委员会将会为批量交易签署数据可用性证书（DACerts），只有DACerts上传到主网的inbox合约，这就进一步降低了发送到主网所需的数据。为了保证数据可用性，委员会将会运行数据可用性服务器，其公开REST API，允许通过Hash获取数据批次。也就是说，当需要获取对应批次的交易数据的时候，只需要从inbox合约中获取上传的hash值，通过数据可用性服务器提供的公开API，即可查询到交易数据。相比直接将交易数据的calldata上传到

inbox合约，Anytrust的方案在一定程度上牺牲了去中心化特性，人们不得不信任数据可用性委员会保证交易并且从那里获取交易数据。但Anytrust的方案相比其他侧链和采用BFT共识的链仍然具有较高的优势，即Fallback to Rollup。

2.Fallback to rollup

Fallback to rollup是Anytrust的安全底线，即在数据可用性委员会失效的情况下，Anytrust可以回退到Rollup模式，重新将数据可用性放到layer1的inbox合约，待到数据可用性委员会恢复以后再切换回Anytrust模式。相较于一般的BFT共识，Anytrust拥有更低的信任假设，在数据可用性委员会中，仅需要两名验证者是诚实的就可以确保链的安全，签署一份数据可用性证书，需要N-1位验证者的签名，N位验证者，只要有任何一人作恶，两名诚实验证者必定有一位在N-1位签名验证者中，只要发现恶意签名，其中一名诚实验证者拒签，就无法形成签名。因此在Anytrust的信任假设中，仅需要两名诚实的验证者就可以确保签名的正确，这相较于其他BFT共识链2/3的诚实节点具有更高的安全性。这也是其独特的优势所在，同时相比其他侧链，在数据可用性委员会失效的情况下，链不会停止出块，sequencer可以将calldata发送到主网的inbox合约中实现回退到Rollup模式，确保链的安全和稳定，这也是其他侧链不具备的。