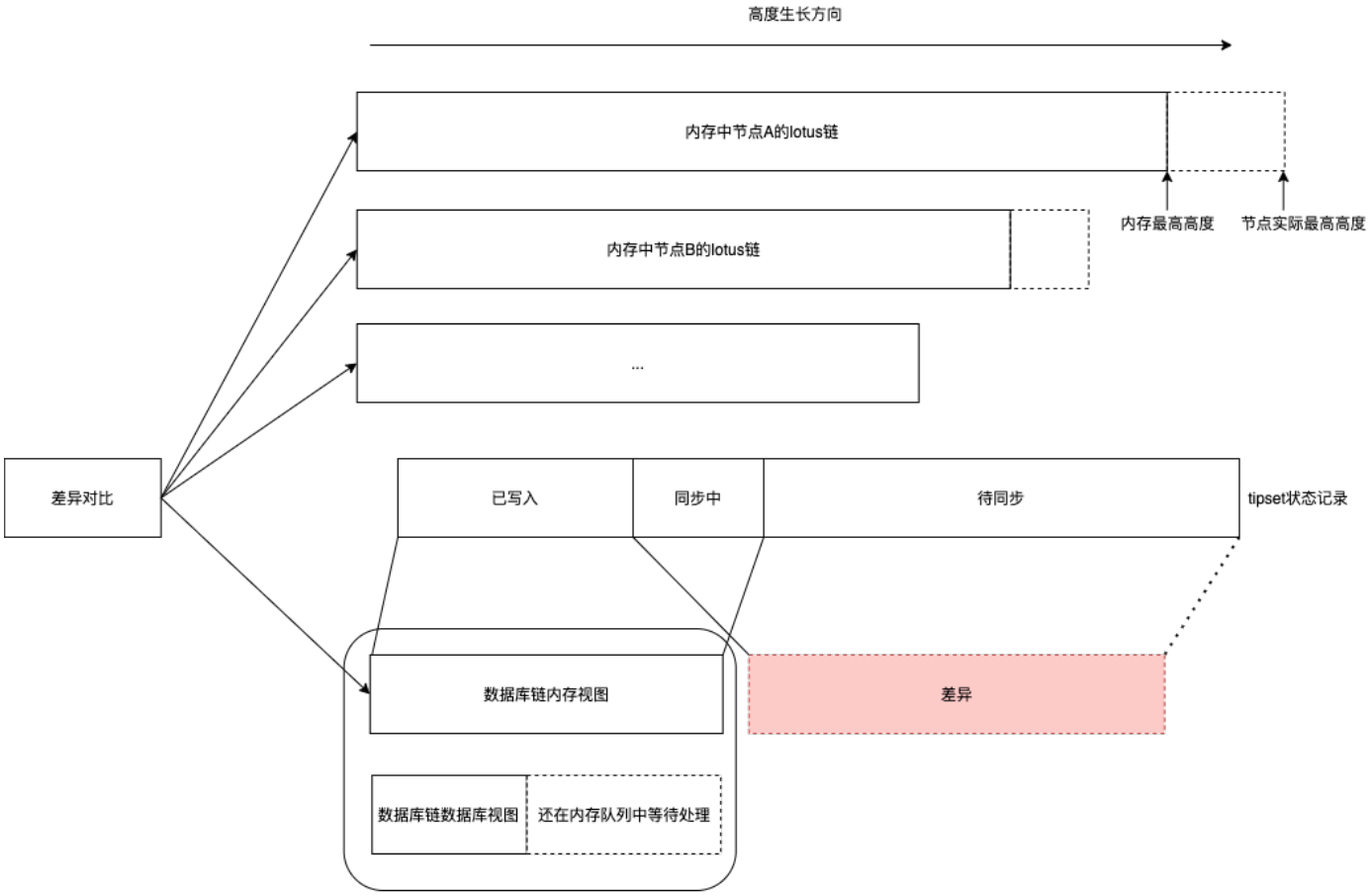
一、逻辑模型

基本思想是先收集区块和消息，获取tipset相关的信息，tipset逐步趋于收敛，当tipset最终变为不可逆时，记录哪些信息是没有在规范链上的。



数据同步逻辑模型

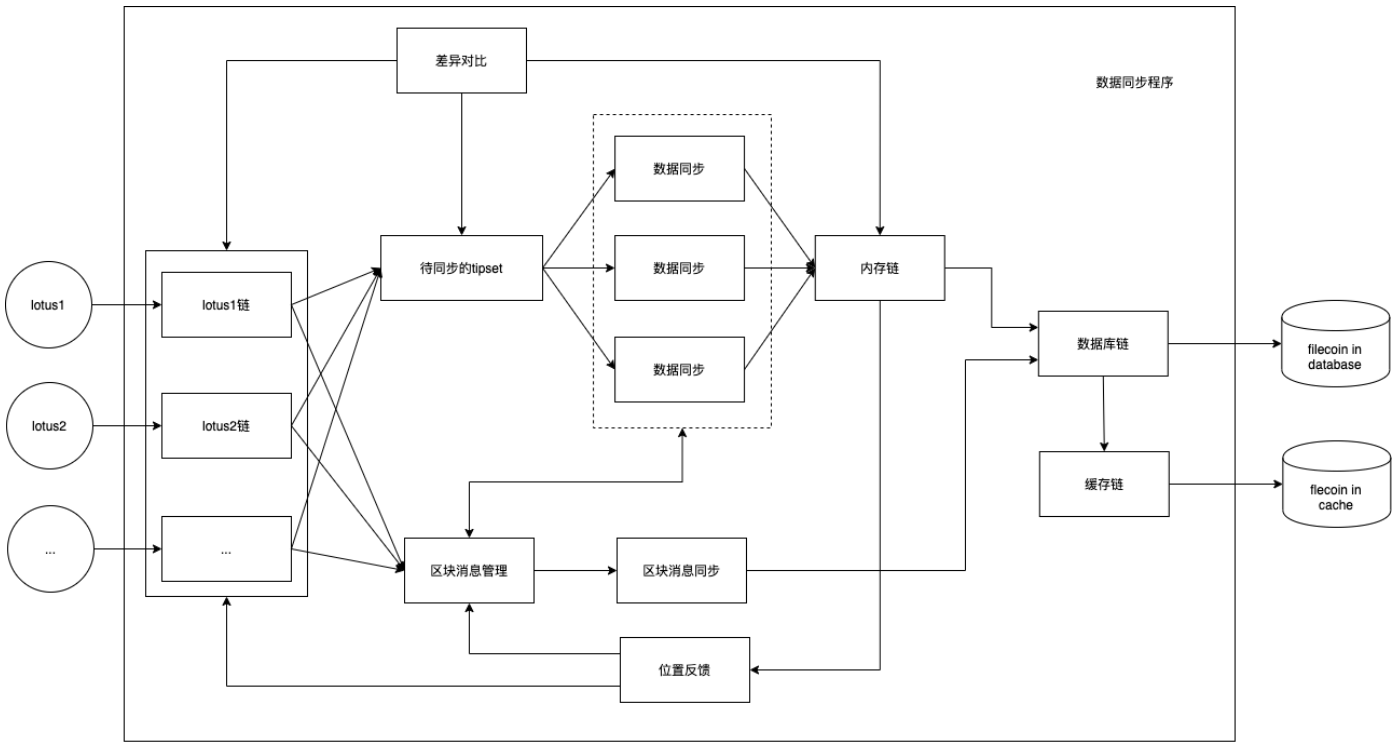
使用了多级缓存来实现一致性以及性能。lotus链比数据库链在内存中是超前的。记录了一系列数据库信息

缓存1:datasource里的lotus链会记录比数据库链更多的tipset，提前获取差异的原始tipset。

缓存2:tipset状态管理，可以对差异中未同步的数据提前同步。

缓存3:写入数据库时，先写入内存视图，确保对外状态的及时变更，

各模块异步处理。



数据同步组件模型

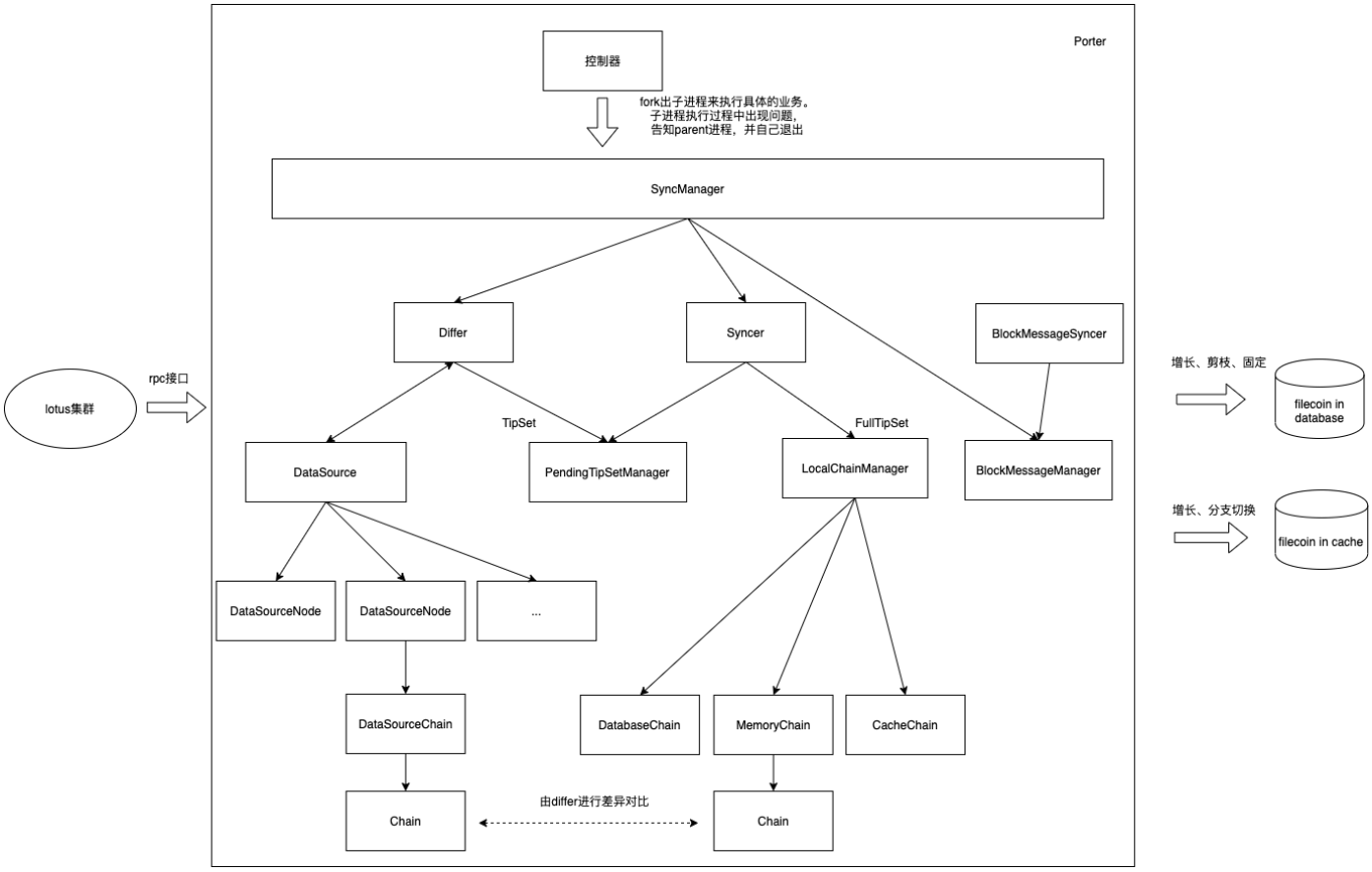
可以连接多个lotus，本地会因此构造多个组件来维护从lotus获取到的信息。每个lotus节点用一个同步协程来处理。

本地记录的数据源链是完整的链，待同步的tipset记录了链的一些待同步片段，本地链记录的是完整的链，但是本地链会比数据源链落后。

缓存里记录了一些索引信息，这些信息直接从数据库查性能会很差。

注：数据模型参考《FIleInfinity数据库链》

二、模块结构



上图描述了porter的功能是从lotus集群把数据同步到数据库以及缓存中。

控制器通过创建数据库表来获取锁。datasource监听模块监听lotus的ChainHead变化，并在本地维护tipset链式结构。Differ负责对比DataSource里的链与LocalChainManager里链的差异，写入到PendingTipSetManager中。Syncer根据PendingTipSetManager中对需要同步的tipset执行具体的数据同步以及数据写入。LocalChainManager保证了和数据库中一样的链信息，并处理数据写入的动作。

从宏观上看，就是对比datasouce和LocalChainManager中的链的差异（用tipset来表示链结构，有点像是以太坊中的block header），找出需要同步的tipset，同步tipset的所有消息、区块、变化的actor（像以太坊中的block body），再把整个结果写入到数据库。在datasource和LocalChainManager这一级别只维护了tipset作为链结构，既满足链结构关系，又降低内存消耗。fulltipset是隶属于tipset之下的具体数据。

采用自适应模型，确保同步和写库的速度是平衡的，当同步的速度大于写入速度时，也会让同步速度降低。

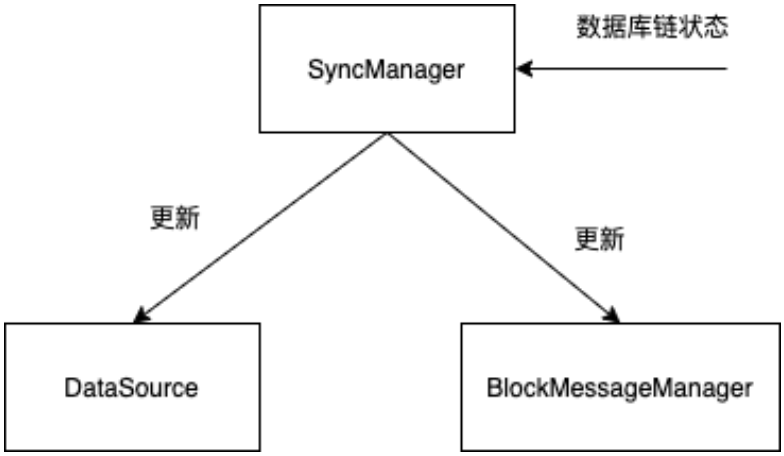
porter程序通过rpc接口与lotus集群交互，舍弃了kafka的推送方式，kafka推送方式是采用了中间件解耦生产和消费数据，让生产者和消费者异步进行，但是毕竟lotus可能会推送失败，而且即使是推送失败lotus还会继续运行。在失败的情况下，需要主动拉取数据来弥补数据缺失。这意味着一个问题用了两个方法来解决。

利用rpc与lotus交互，可以使得通过接口与lotus交互，尽量不去改动lotus，使得任意测试环境都方便测试。即使要改lotus，也是在lotus上新增接口，依旧通过rpc进行交互。如果修改了lotus，那么可以进一步增加协议获取信息来确保这是经过修改的lotus。

从lotus获取到的数据，可以遵照区块链的生长规则，在不同的链上并行生长。

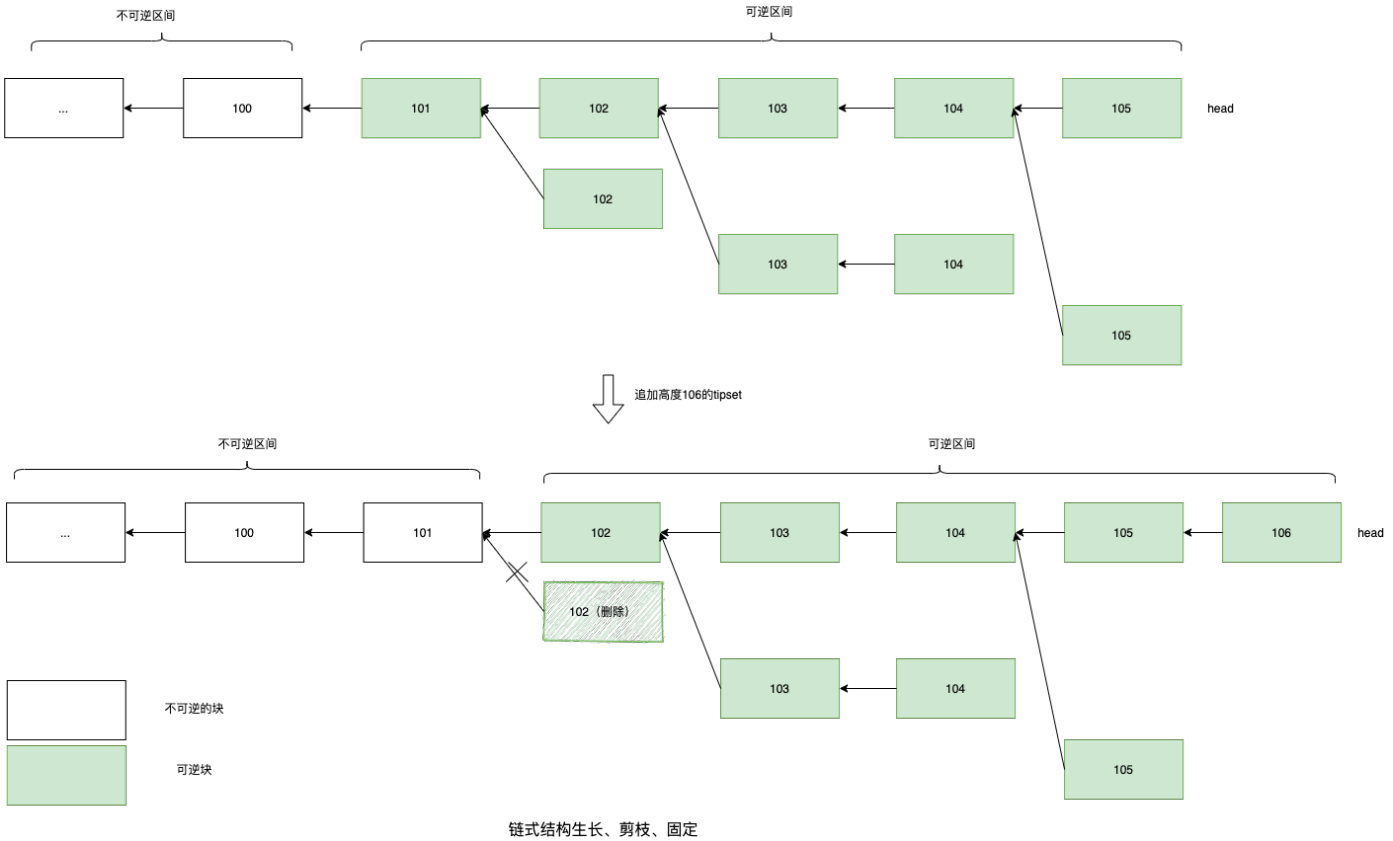
2.1 SyncManager

启动各个模块，监听数据库链事件，并把数据库链的状态同步到对应模块。



2.2 Chain

DataSource记录的是lotus链结构在porter中的表示，LocalChainManager管理的是数据库链结构在porter的表示。从本质上来说，这两者都是链式结构的，因此，可以用同一种数据结构。



链式结构分为不可逆区间与可逆区间，在这里，不可逆区间用最后一个不可逆tipset来表示，可逆区间可以认为包含了一系列的branch，headchain是branch中的最重链，可以简单的用head的TipSetKey来表示它所在的分支。

其中，tipset只能基于已存在parent进行生长，并且它的parent必须是在可逆区间内的。

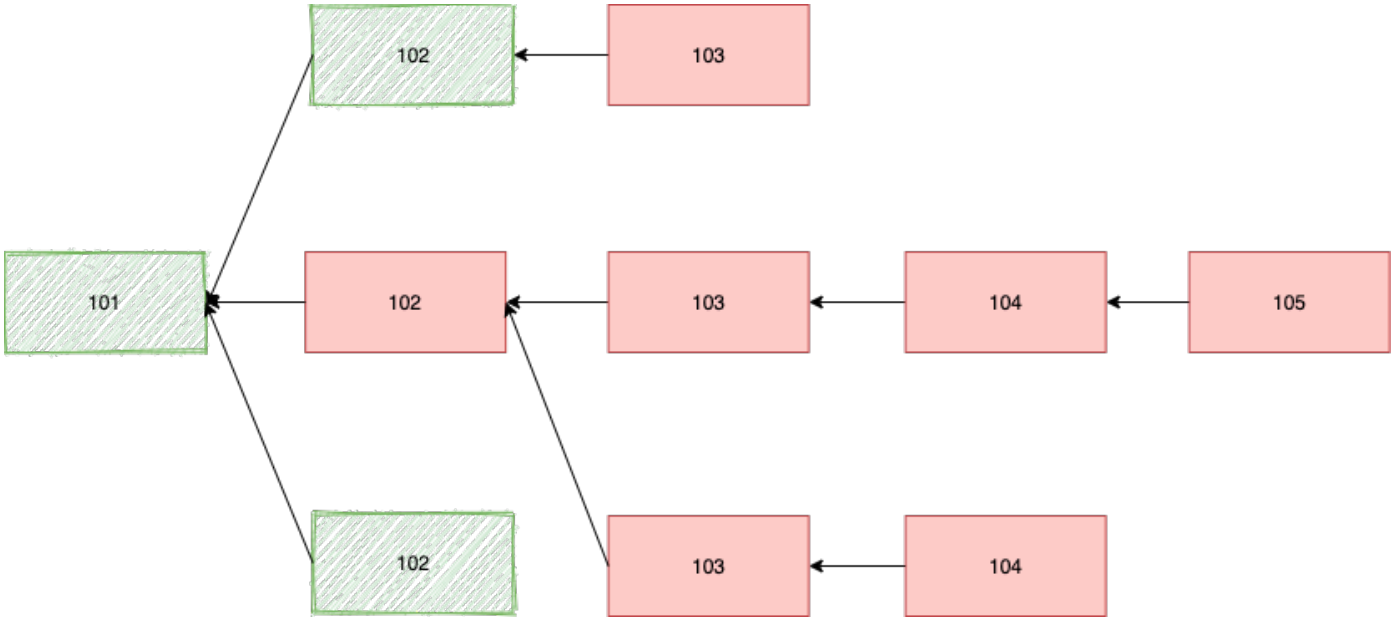
追加tipset后，可能会导致headchain变长，超过了一定长度，这时候就需要对链进行剪枝、并把超出的链进行固定。

为了简化块的增长，要求只能在可逆区间的块上生长。目前提供了三个接口，生长、根据headchain数量修剪、根据不可逆高度修剪。

内部也只记录了tipset，如果要扩展功能（比如需要存储额外的数据），需要在外部模块上进行扩展。

2.3 PendingTipSetManager

对于任意一个未写入到数据库里的tipset，由该模块进行管理。内部记录了tipset的不同状态，每种状态下有对应的数据结构来表示对应信息。



上图中，绿色是已同步的，红色是未同步的，已同步完毕的tipset会放入到一个同步完毕的集合中，所以ptsm里记的是链的片段。同一个高度记录多个tipset。按照从低到高的方式获取一个待同步的tipset，如果某个tipset的parent还在未同步/同步集合中，就忽略这个tipset。

为了避免写入的链被重复处理，同步完毕的tipset需要缓存一定的时间。最终确保同一个tipset不会被重复同步、写入。

2.4 Differ

由这一层计算差异，输入是dbchain和dschain。一次对比出所有足够多的差异，比如同一个tipset可以往多个节点同步，放入到PendingTipSetManager中。

首先获取dbChain，根据不可逆区间向datasource进行剪枝与固定，然后复制一份Chain，然后进行对比。

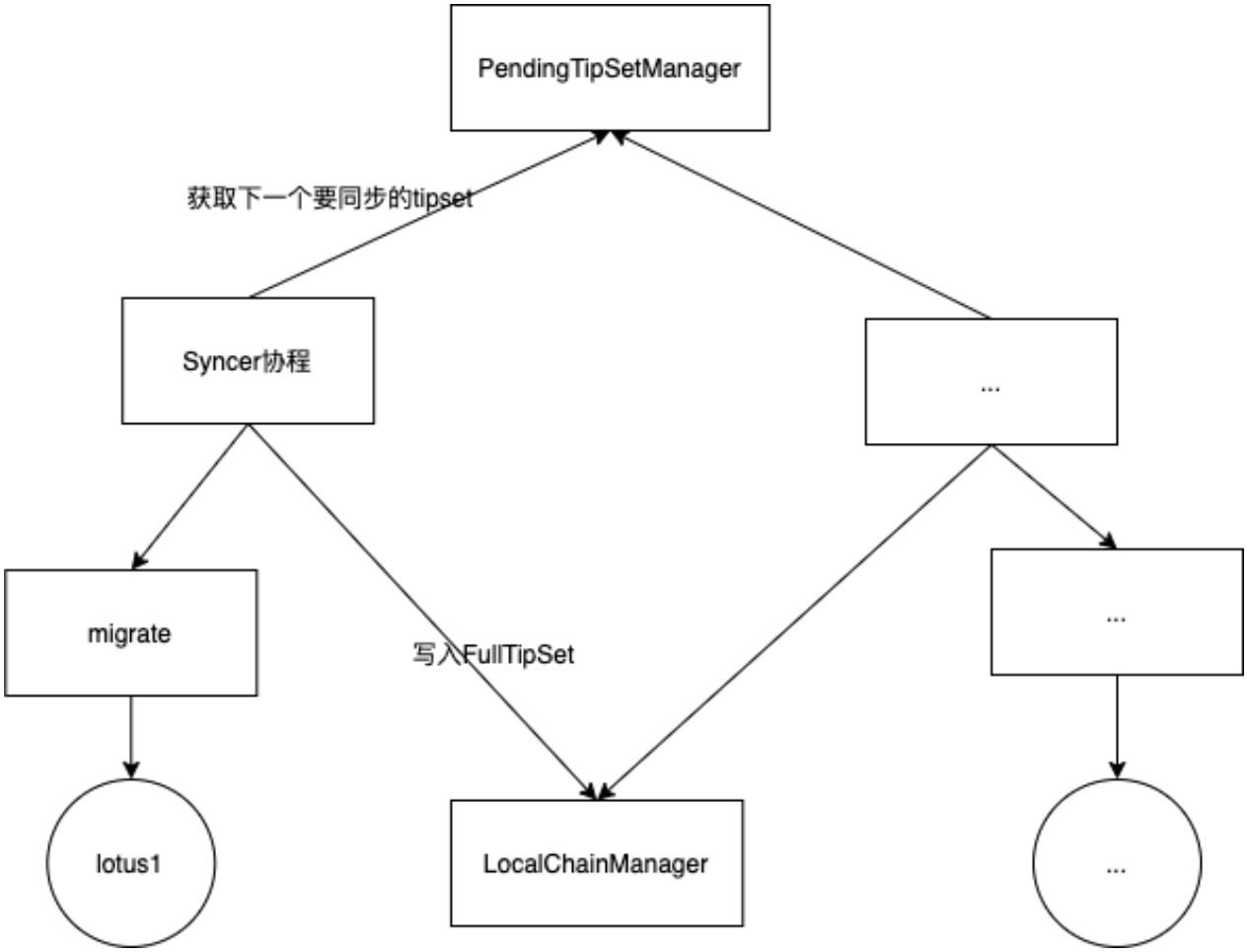
```
diffSet := make(map[types.TipSetKey]*types.TipSet)
```

```
    for _, branchHeadTs := range dsChainBranchHeadTsSet {

      curTs := branchHeadTs

      for {
        prevTs, exist := dsChainBranchTsSet[curTs.Parents()]
        if !exist {
          break
        }

        if dbChain.ContainInBranch(curTs.Key()) {
          break
        }

        diffSet[curTs.Key()] = curTs

        curTs = prevTs
      }
    }

    d.ptsm.appendDifferences(diffSet, node)
```

思想：每个dsChain的branch的每个tipset都尝试向dbChain查找，如果找不到，放入到difference中，如果找到了，说明该分支历史数据都是有的，忽略这个分支。

该算法对差异数据没有任何多余的对比动作。

2.5 Syncer



因为当向一个节点同步时，该节点负载非常高，因此，单节点上不同的tipset要串行同步。可以向不同的节点并行同步tipset。

因此，实现了每个节点一个协程，从而实现单节点串行同步。不同的协程竞争同一个tipset的同步任务。从而尽可能让不同的节点满载，并提高同步速度，又能实现节点间负载均衡。一个tipset里的所有变更的actor之间没有任何关联，是可以并发处理的。具有前后依赖关系的tipset是串行同步的。

数据同步模块对上层暴露的是纯粹的数据同步逻辑，实际上，它需要把最初的tipset转化为加工后的fulltipset，中间会涉及到原始数据同步，变更actor检查，消息处理，变更actor处理。

1.获取tipset的原始变更结果RawFullTipSet

（1）message

（2）receipt

消息执行的收据以及内部消息?

通过StateCompute的方式回放该Message的gas消耗。

（3）变更的actor

对比两个tipset的状态树来获得差异。

2.处理，形成FullTipSet

处理是放在processor里执行的

（1）处理所有变更的actor，获取这些actor的actorInfo，包括balance和state接口，根据具体的actor类型进行处理，调用具体actor的state的load方式，处理其中的state。考虑到actor的处理主要耗时是在网络查询上，获取到actor详细信息后的后续计算很少，也就是会有较多的io等待，因此并行处理的协程数量可以比cpu核数多。为了让每个协程负载均衡以及避免空闲，当协程返回处理结果后，给该协程派发任务。

（2）message的关系处理

（3）gas费计算

具体处理细节如下：

chainpower处理

变更的是power actor。是从变更actor里的head读取详细状态，tipset级的。

chainreward处理

变更的是reward actor。根据rewardAddress获取到奖励信息，是tipset级的。为什么chainreward和chainpower这两种方式不一样。

minerpower

变更的是miner，minerActor呢？通过powerActor获得到所有矿工的算力，再从中获取到所有变更的矿工的算力。

multi_sig处理

变更的不是用于管理的multi_sig actor，似乎是一个普通账户。根据actor.head获取详细状态，再获取该地址的前一个状态。

支付通道处理

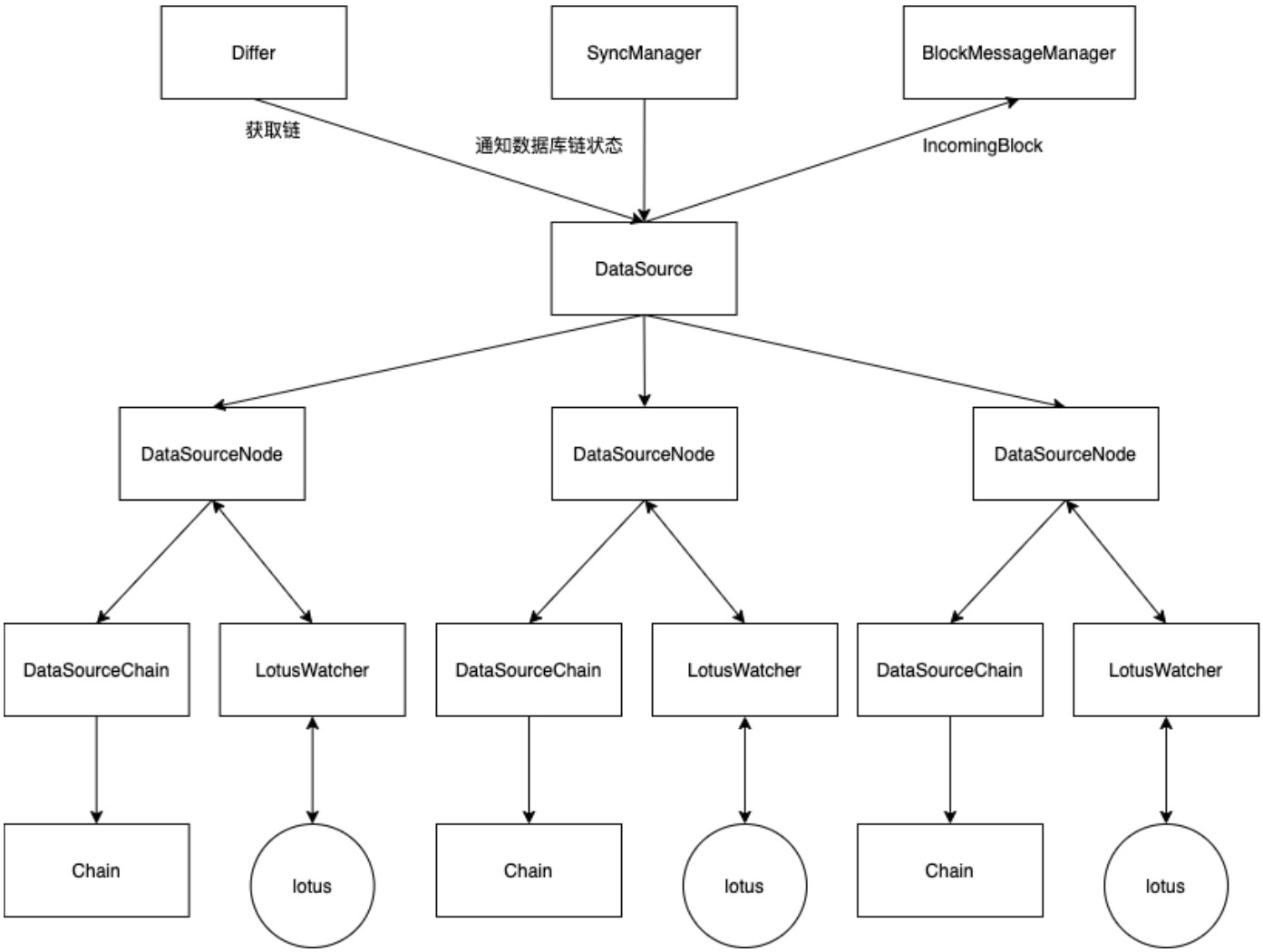
每个创建的支付通道是一个actor。会根据actor.head获取该通道的详细信息。

marketdeal处理

变更的是用于管理的actor。通过和之前的对比，获得差量。

注：为了减少io等待的导致的cpu浪费，可以同时处理多个tipset。

2.6 DataSource



datasource的功能非常纯粹，负责监听多个lotus节点，并在本地维护一些tipset的链结构。DsChain中选取dbChainHeadHeight+2*HeadChainLimit+10个高度作为可逆区间，总之一定要比LocalChainManager的要大。一方面能更强的确立X区间最长链的不可变，另一方面，提前预支可以避免用的时候再去同步tipset。对外提供了3个接口，获取lotus链，lotus节点head变更通知，数据库链状态通知。

2.6.1 DataSourceChain

内部包含对外可见的链以及不可见的heads，可见的链也是用Chain实现的。因为tipset的一些信息（权重、状态根、基本手续费）是放在下一个tipset上的，但是写入数据库的时候希望一个tipset记录了这个tipset的所有信息。所以实际上fulltipset包含的是tipsetA，以及指向这个tipset的下一个tipsetB，但是数据库里存的是tipsetA以及部分从tipsetB获取的本该属于tipsetA的信息。

正因为如此，在公共的migrate中需要用到下一个高度的tipset，所以datasouce暴露给差异对比的高度会比当前分支的高度小1。

2.6.2 DataSourceNode

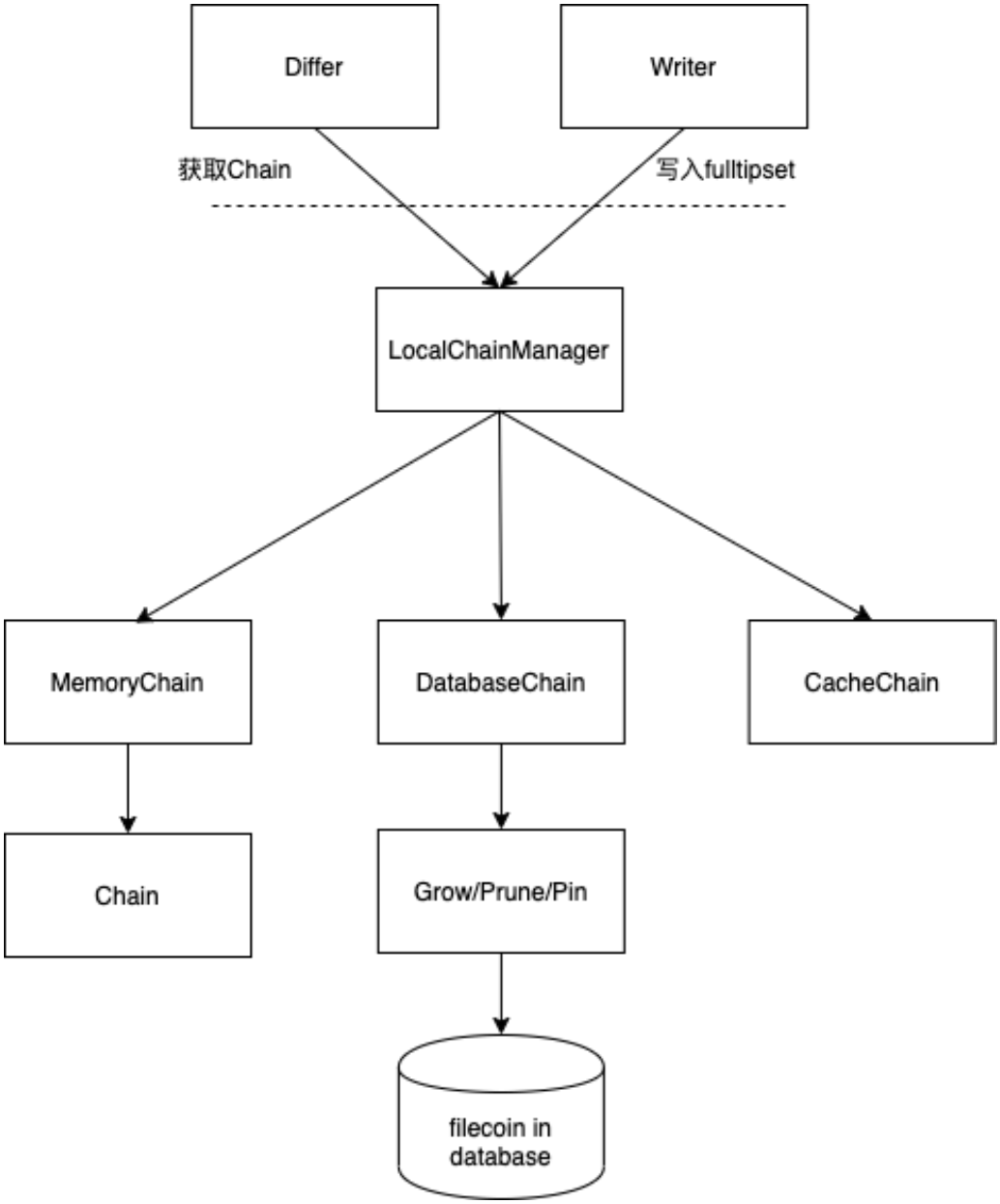
主要用于维护链的变化，它负责监听链上节点，并把链上节点的tipset放入到本地的DataSourceChain。

本地记录的远程tipset信息和远端真实的的tipset高度可能还有很差异。如果高度差于大于一定高度，就用顺序同步的方法。如果期间发现新获取到的tipset无法和前一个tipset连接，说明出现了headchain切换，用新获取的tipset作为head进行反向同步。否则第一步顺序同步时高度小于一定差异了，用远程给出的head进行反向获取tipset。不同的branch是并行获取的的，一旦发现前一个tipset已经有了，追加本地同步的数据到一个数组里，这样可以确保不多个head不会同步相同的数据，并且写入时也可以确保前面的数据都有。

获得远程的branchHead列表，如果branchHead比当前的要高很多，从低到高同步，否则从高到低同步。

为了减少需要同步的tipset，当收到lotus的HeadChange时，可以延迟一定时间，比如10秒，然后再向lotus查询当前最新的head，这样需要同步的tipset数量就只有原来的1/5左右，可以降低向lotus同步数据的次数、数据库写入频率、客户端变更通知频率。并且还需要监听收到的区块。

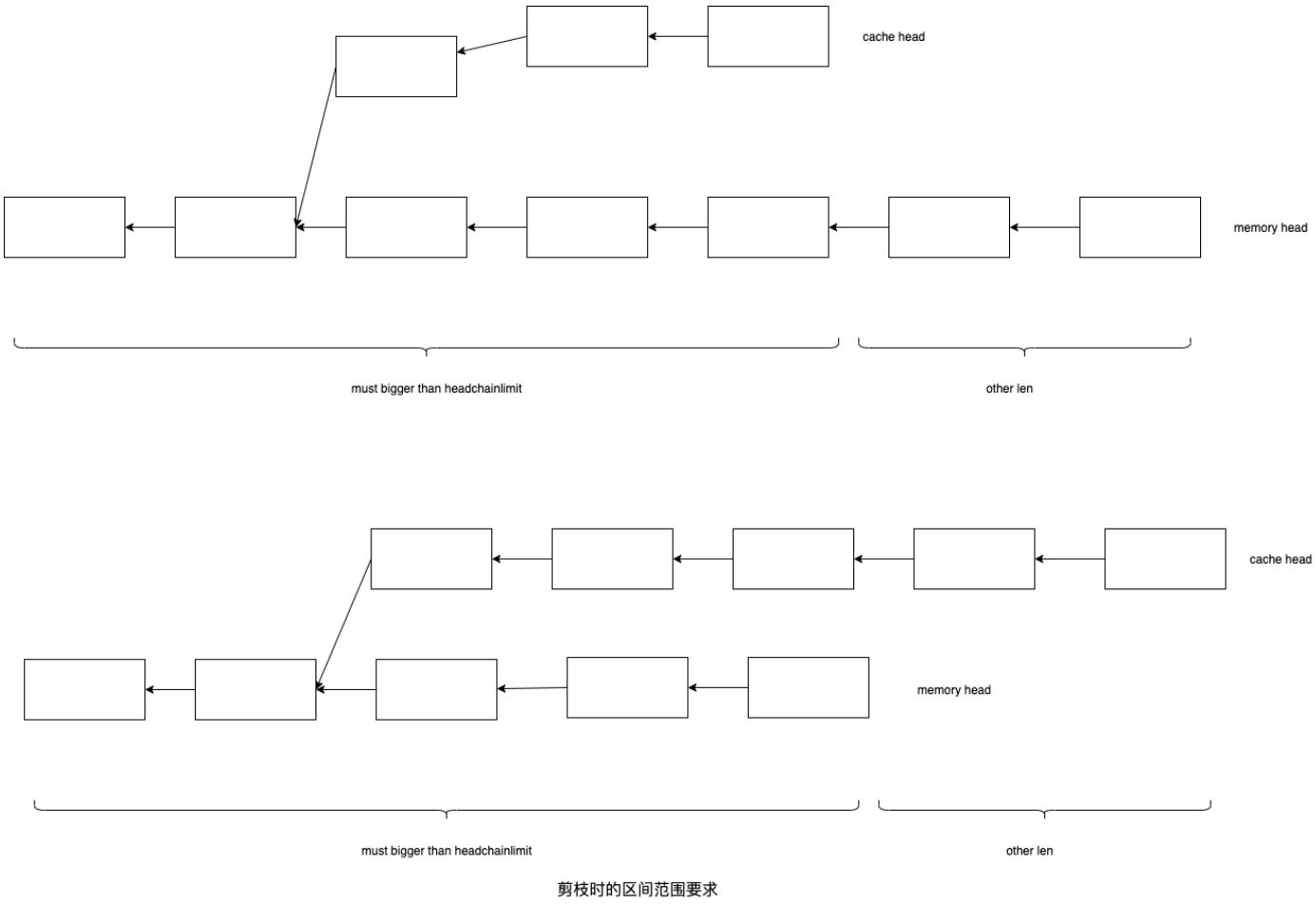
2.7 LocalChainManager



本地依赖三个模块，MemoryChain、DatabaseChain、CacheChain，MemoryChain就是内存中的链，记录可逆区间的所有tipset和分支，DataBaseChain负责对fulltipset写入，CacheChain实现把数据库链写入到缓存。外部提交fulltip时，首先会更新MemoryChain中的链结构，并产生写数据库操作事件，然后异步的写入到数据库中。CacheChain定时把HeadChain上的数据写入或者是进行分支切换，然后把缓存链的head告知LocalChainManager，LocalChainManager根据数量的限制控制MemoryChain，从而获得一个剪枝和固定列表。剪枝/固定完毕后，异步通知DataSource当前内存链的不可逆高度。

模块对外展现的就好像对内存中的链操作一样，不会感知到写数据库的过程。

为了让数据库对外都是一致性的，剪枝对于一个分支从高到低删除的，多分支间不同高度无关联。



剪枝时的区间范围要求

剪枝时，需要保证短的那条链的最新高度数量区间要大于headchainlimit。

2.7.1 MemoryChain

内部包含了通用的Chain。

2.7.2 DatabaseChain

参考《lotus数据库链》

数据库操作过程中有任何处理失败，都要告知主控逻辑并退出程序。

1.生长

从数据关系上看，一个tipset的所有内容对外要么暴露，要么不暴露，最合适的方法是事务提交。但是db的事务大小有限，fulltipset是有可能超出事务大小导致失败。因此要要拆分成多次提交，并在外部的查询请求上做一些特殊处理，防止不完整的tipset对外暴露。tipset之间是串行处理的，保证链的增长过程。系统的瓶颈并不在写入tipset写集上，而是如何获取tipset写集，所以串行写入并不会产生瓶颈。

(1)增加记录到tipset表，增加记录到branch表（要确保parent在branch表中存在），该tipset的状态置为写入中。过程是事务的。

(2)写业务数据到各表，这个过程可以拆分成多个协程并发。

(3)更新branch中的tipset状态为可用，根据实际情况看是否需要更新headchain。过程是事务的。

2.剪枝

在计算下一个快照点时，发现距离下一个快照点的路径上还有一些分支，就需要删除这些分支，这个动作叫做剪枝。剪枝的过程是从高度高往低剪枝的。

（1）tipset必须在branch表里的，设置状态为删除中。过程是事务的。

（2）从各个表中删除这个tipset的数据，如果不考虑键依赖，都是可以并发的，否则有些操作是可并发的，有些是需要顺序的。

（3）从branch、tipset表里删除该记录。过程是事务的。

3.固定

当一个tipset可以变成不可逆的tipset就可以固定该tipset。

（1）branch表里按高度排序是第一个，且tipsetkey相同。从不可逆表里删除parent。从branch和head chain里移除自身。添加到不可逆表。过程是事务的。

4.从数据库中恢复

如果写入fulltipset/删除tipset的过程中被打断了，需要进行清理。

(1)创建PidFile表，失败退出。

(2)查询branch表，获取不是可用状态的tipset，然后清除。

(3)根据branch、irreversible_chain、head_chain的信息构建出chain。

(4)获取配置

以上1和23的过程是可以异步的，但是为了简单期间，三者整体都会串行。假设不可逆的是不会分叉的，否则需要人工介入处理。

以上操作确保了，提交fulltipset的任何过程出现故障时，都可以在新的进程上被正确回滚，使得这个提交tipset写集操作看起来是事务的。脏数据永远只会在最高高度短暂存在，而且不会暴露给外部。

对于外部查询，总是基于head_chain向数据库中查询具体的业务数据，确保了数据总是正确的。

2.7.3 CacheChain

控制数据写入缓存，并反馈给LocalChainManager当前缓存链的高度，从而在LocalChainManager上通过数量限制实现正确的剪枝和固定。由于因为缓存而产生的剪枝所依赖的headchainlimit是和正常的headchainlimit是相同的，极端情况下，缓存没有持久化，但是剪枝后缓存里的早期的head已经不存在了，就会导致缓存后续就无法更新。为了解决这个情况，可以有额外的一个缓存，只根据不可逆高度前的数据生成缓存。这样即使发生异常，用这个备用的去替换用于正常写入的cache，就能恢复正常。

在设计上，把数据分为kv和list两种结构，通过统一的方法获取kv和list，如果是回滚，对于kv就是删除，list就是RPop，如果是增长，kv就是set，list是RPush。

2.8 BlockMessage

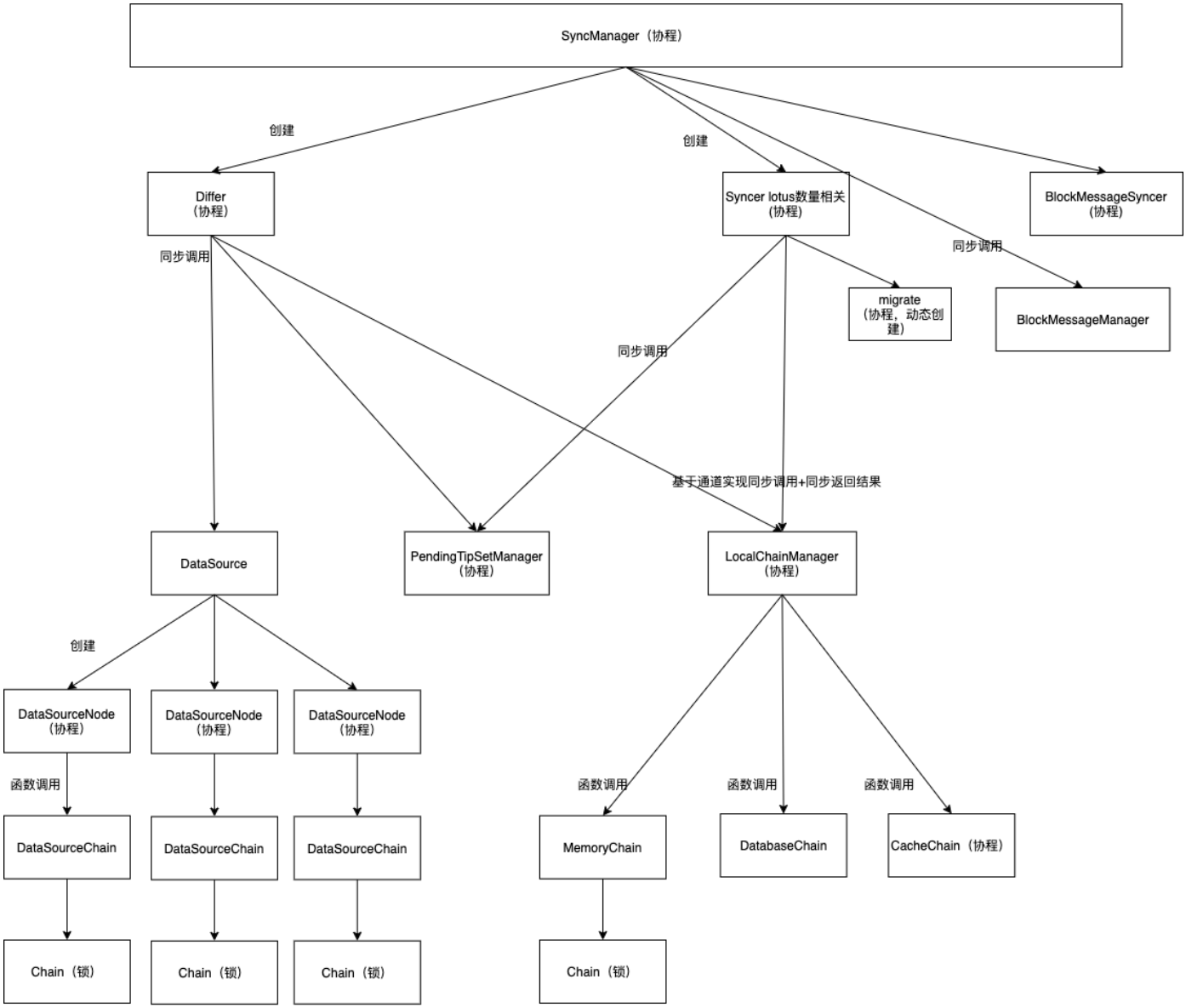
2.8.1 BlockMessageManager

记录待同步的Block对应的消息。会根据内存链的状态而删除旧数据。

2.8.2 BlockMessageSyncer

定时获取待同步的Block，同步它的消息并写入。

三、并发模型



并发模型

模块只提供同步接口（无论内部是同步还是异步实现的），如果想要异步返回机制，就需要新建协程等待返回结果。

其中，datasource的DataSourceNode内部都是协程异步处理。

有些接口是通过协程+通道实现的。比如DataSource，LocalChainManager。

migrate内部有很多协程进行并发处理。

四、代码依赖

数据同步模块功能独立，从lotus到内存fulltipset，写入fulltipset到数据库，这两块逻辑是infinity公共代码。此外，它和lotus的代码是独立的，把lotus代码看作是一个模块，只依赖lotus的rpc接口和数据结构。

五、非形式化验证

为了容错引入了数据库锁。

对于链的增长，参考了共识算法中的安全性和活性的思想。

1.安全性

参考提交/回滚tipset写集

中间任意阶段故障，都不会影响数据的正确性。由lotus链的正确性加上数据的安全性约束，从而保证了db链的正确性。

2.活性

以数据库锁来确保只有一个worker在工作。Differ能定时检查数据库链和lotus链的差异。

1.lotus链的正确性

2.db链的正确性（各种故障加上数据正确处理）

六、单元测试与集成测试

每个模块都是可以独立测试的。通过mock各种输入信息来模拟业务。