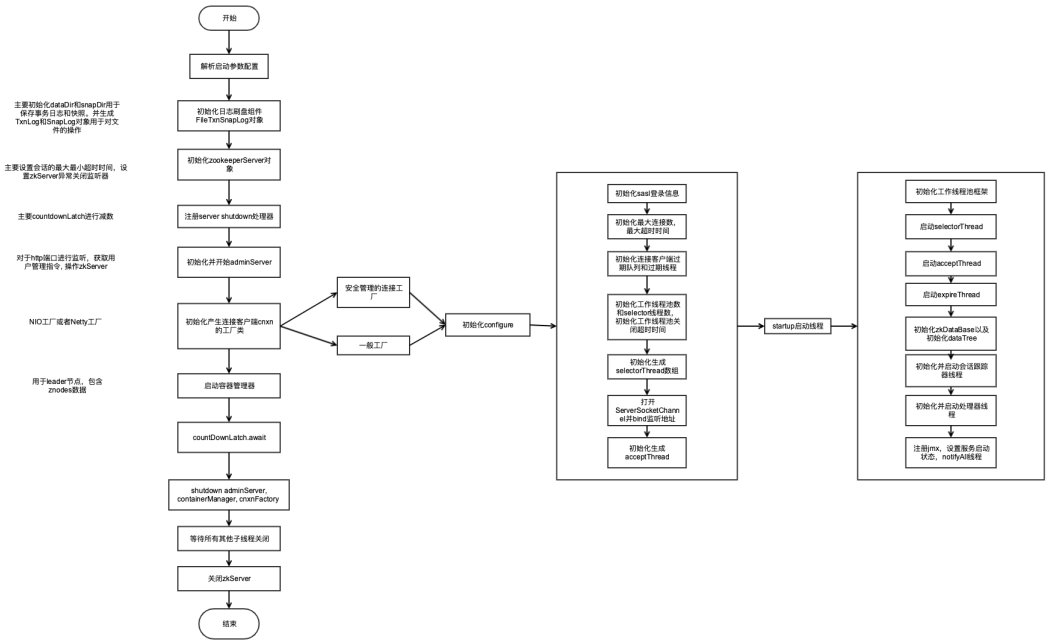
ZookeeperServer

一、启动层 ZookeeperServerMain

main()方法主要负责初始化并启动下列组件。

1. Config组件： ServerConfig - 解析保存服务端的配置类
2. DataLog组件： FileTxnSnapLog - 快照及事务日志刷盘组件
3. 数据组件： ZKDatabase - zk数据组件
4. Server组件： ZookeeperServer - 服务端组件与客户端通信，请求处理，快照与事务日志，数据树，协议解析等
5. 管理Server组件： AdminServer - server管理组件
6. 容器管理组件：containerManager - 负责管理zkDatabase

详细启动流程如下图所示：



二、数据层

2.1 ZKDatabase

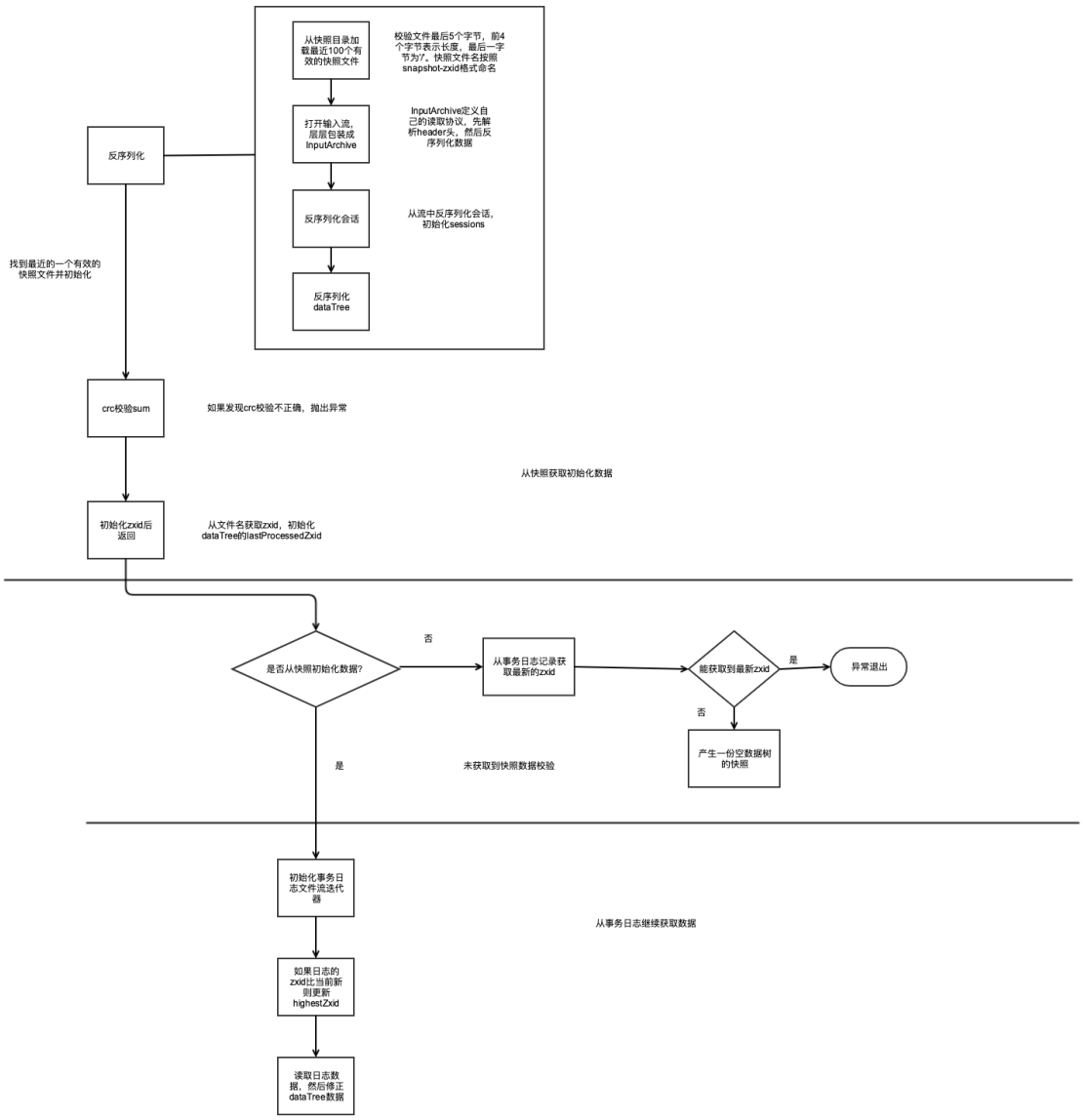
zkDb主要用来管理 dataTree的事务操作，客户端会话管理，快照管理， dataTree序列化到文件及从文件中恢复等操作。zkDb是对dataTree的装饰和增强，保证dataTree的一致性。

2.1.1 启动zkDb

1. 初始化dataTree对象，初始化sessionWithTimeout容器，初始化快照大小因子
2. 如果zkDb已经被初始化了，则从zkDb获取最大的Zxid，如果没有初始化则加载数据
3. 清除zkDb中无效的Session, 总sessionWithTimeout容器中取出会话，加入到deadSession中，然后一一从dataTree(ephemeral容器)中删除会话对应的节点路径，然后遍历路径从dataTree中删除对应zxid的节点

2.1.2 loadData

如果是新的leader执行，则zkDb可能已经初始化则直接记录DataTree的lastProcessedZxid。server第一次启动则通过FileTxnSnapLog组件从文件读取数据，并初始化dataTree对象。初始化过程如下流程图。在初始化数据树完成之后，清除过期session，然后重新保存一份快照。



2.2 DataTree的数据结构

2.2.1 数据容器：

提供了2个平行的数据结构， hashmap用来匹配path和路径节点， tree用来存储datanodes

nodes ： ConcurrentHashMap<String, DataNode> 存储一个路径的节点数据

ephemerals ： ConcurrentHashMap<Long, HashSet<String>> 用于存储一个会话创建的节点

containers: Set<String> 保存所有container节点的path的

ttls: Set<String> 保存所有ttl节点的path

dataWatches，childWatches ： watcher容器保存了客户端注册的所有watcher，如果对应节点有更改将回调所有监听器。

PathTrie用于存储追踪quota

三个核心路径：

/zookeeper ， 管理zookeeperServer描述其状态

/zookeeper/quotas　　quota管理节点

/zookeeper/config　　配置管理节点

2.2.2 DataNode核心状态字段

1. czxid　　：　　czxid为创建节点的事务id
2. mzxid　　：　mzxid为修改节点的事务id
3. pzxid　　：　记录子节点更改的事务id
4. ctime　　：　创建时间
5. mtime　：修改时间
6. version ：数据的版本号
7. cversion ：　记录子节点数据更改的版本号.
8. aversion ：　acl版本
9. ephemeralOwner　　：　　拥有该节点的sessionId

zxid:

zxid为事务操作的id，针对每次更改请求都会由zkServer递增该值(类似于数据库的自增键值)。在zkServer启动的时候会初始化hzxid:AtomicLong用户生成zxid。 在更新请求递交到firstProcessor后，会请求递增该值获取一个新的zxid。对于读数据请求，会使用当前的zxid。

cxid:

cxid为客户端记录的id如果客户端，在服务端返回的时候需要设置cxid。客户端接收到响应后，会从中获取xid与pengdingQueue的第一个比较，如果cxid不对，则认为是一个失序的请求。直接返回连接丢失异常。

version:

1. firstProcessor记录数据的版本号，该版本号在校验更改节点数据请求(setData，setACL，deleteNode，createNode)的时候递增，并记录在outstandingChanges中。如果更改记录不存在则从dataTree中获取初始信息进行记录，如果客户端请求的版本号与服务端更改记录的版本号不同则异常。 在createNode和 setData setACL和 check的请求中会将新版本号传入后dataTree中

2. DataTree中记录数据更改的版本号。 createNode设置父节点cversion为传入进来的版本参数; setData，setACL设置当前节点version为传入的版本参数

3. 版本校验， 在客户端的请求中，只有 deleteNode, setData, setACL，check4类请求需要校验版本号，如果更改的版本号不等于当前更改记录的版本号则异常。

2.2.3 数据操作：

创建节点：
初始化dataNode的状态，取出父节点更新父节点的状态，然后创建新节点初始化状态后加入nodes中。根据创建节点的类型，将其添加到对应的路径容器中container, ttl, ephermal。如果是quota的节点则进行特殊处理，然后触发 watcher容器中注册的所有监听事件。

第一次创建node的 czxid, mzxid, pzxid都初始化为zxid。 ctime, mtime初始化为请求时间，version，aversion初始化为0。

如果发现父路径不存在，或者父路径存在该子节点。则直接异常返回。

删除节点 ： 与创建节点对应，删除对应容器中的数据。设置父节点的pzxid为当前zxid.

更新数据： 从nodes获取dataNode更新其data及mtime mzxid。同时更新数据的版本号version。
如果是quota相关则特殊处理，最后触发监听器.

序列化相关

关闭会话: 从ephemerals中移除对应数据，然后从nodes中删除对应路径的dataNode

备注：在从nodes中获取dataNode时候都会返回对象的一个拷贝。

在修改dataNode的时候都会加synchronized关键字，每次修改都有事务id和版本。

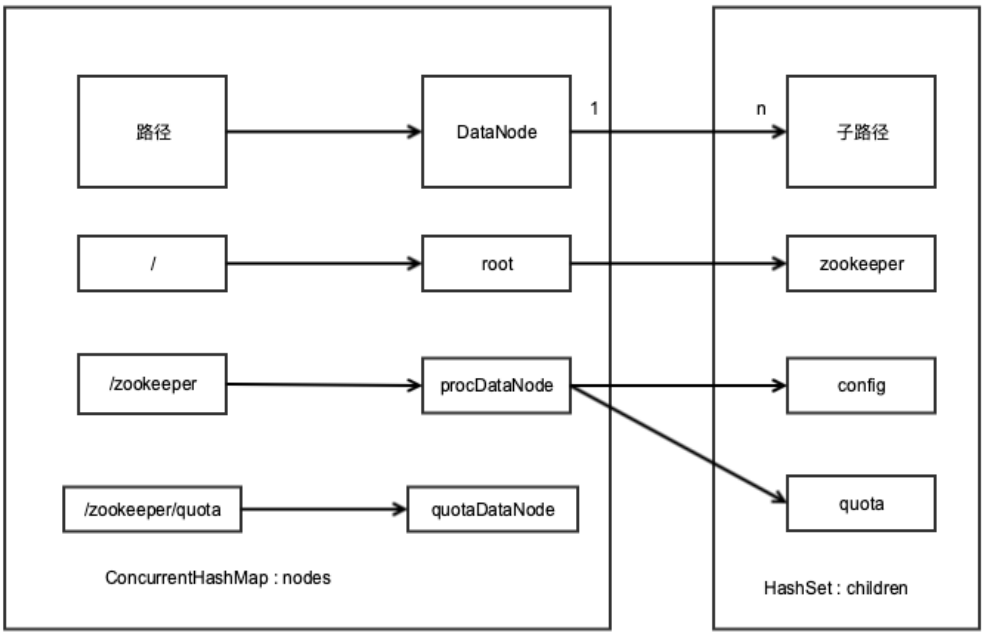
操作聚合：

DataTree提供一个聚合操作的方法processTxn(TxnHead head, Record txn)该方法解析客户端请求的参数信息，根据Head中的操作类型，调用对应的操作方法，并记录事务zxid，然后将结果写入ProcessTxnResult中。

初始化空dataTree：

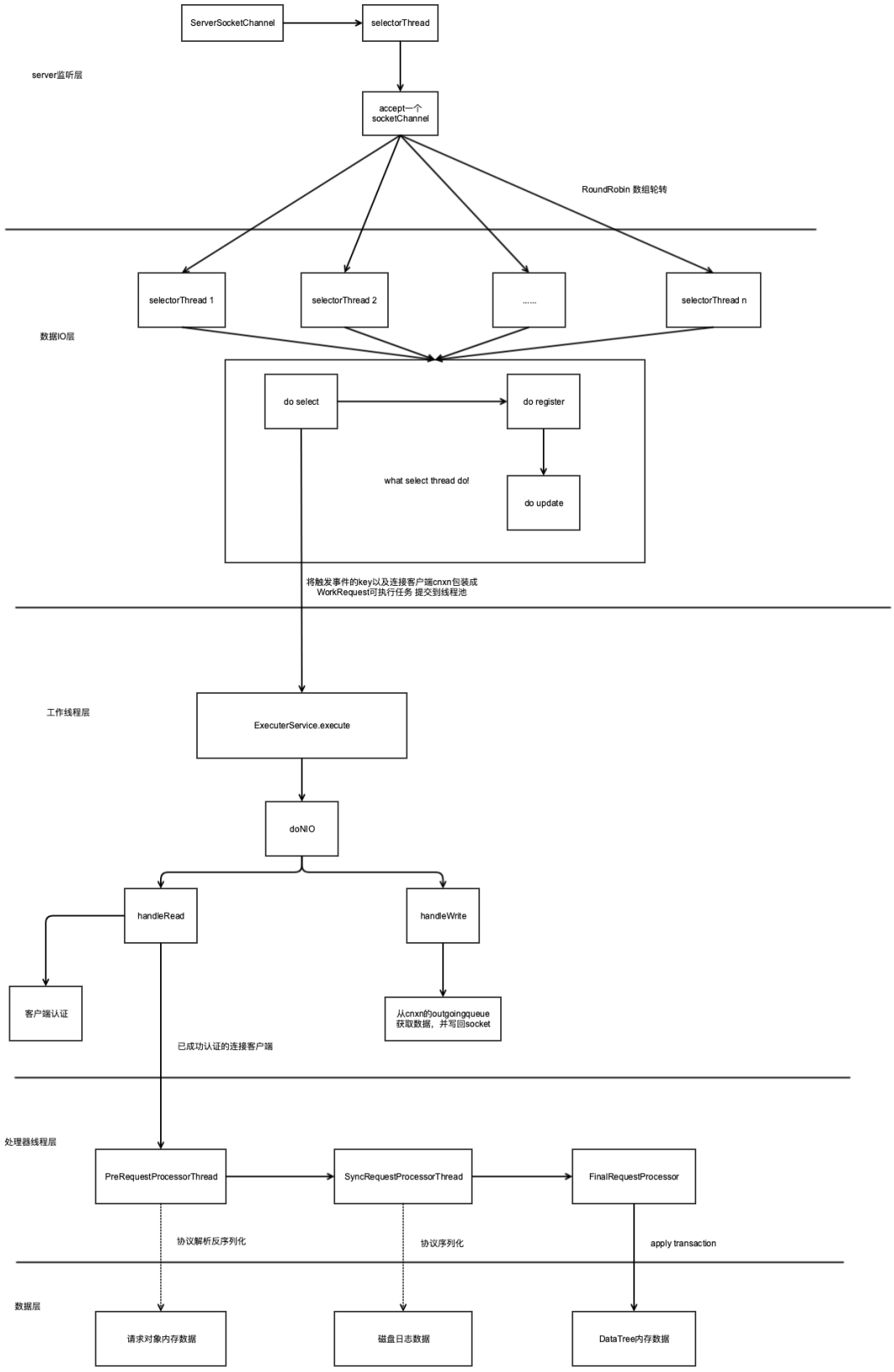
在dataTree的构造函数中，会初始化nodes容器并初始化rootNode，放入跟路径 ’/’ 中 nodes.put("", root)，并为root节点添加zookeeper节点

在/zookeeper节点下面初始化config节点



三、通信层架构及多线程交互工作原理

3.0 线程架构总览



3.1 RequestProcessorThread

请求的处理器链，
从1->2->3的顺序处理。在处理用户请求的时候，如果不是auth，或者sasl请求。工作线程submitRequest时将请求提交给处理器线程处理。在提交时已经将cnxn, sessionId,xid,type,bytebuffer,authinfo等信息封装成request对象，由处理器负责对request对象进行处理。

1.PrepRequestProcessor

2.SyncRequestProcessor

3. FinalRequestProcessor

这些处理器都是一个线程组件，每个处理器都拥有请求队列，该线程不断从队列中获取请求进行相应处理，在处理完成后将请求提交到下一个处理器的队里中。由于各处理器都是单线程，所以每个请求最终按照顺序经所有处理器处理完毕。

3.1.1 firstProcessor

初始化处理器，为更改请求开启事务，并统计系统中的事务数量。

1. 生成事务头, 记录sessionId, cxid, zxid, 当前时间及处理类型.

2. 对于更改类型的请求，进行校验并生成更改记录对象。主要记录新的version, zxid，路径，节点状态，acl，子节点数量等信息。

3.
进行反序列化，根据请求头的不同类型，生成各类型的Record子对象，并调用该对象将请求体request中bytebuffer按照协议反序列化。

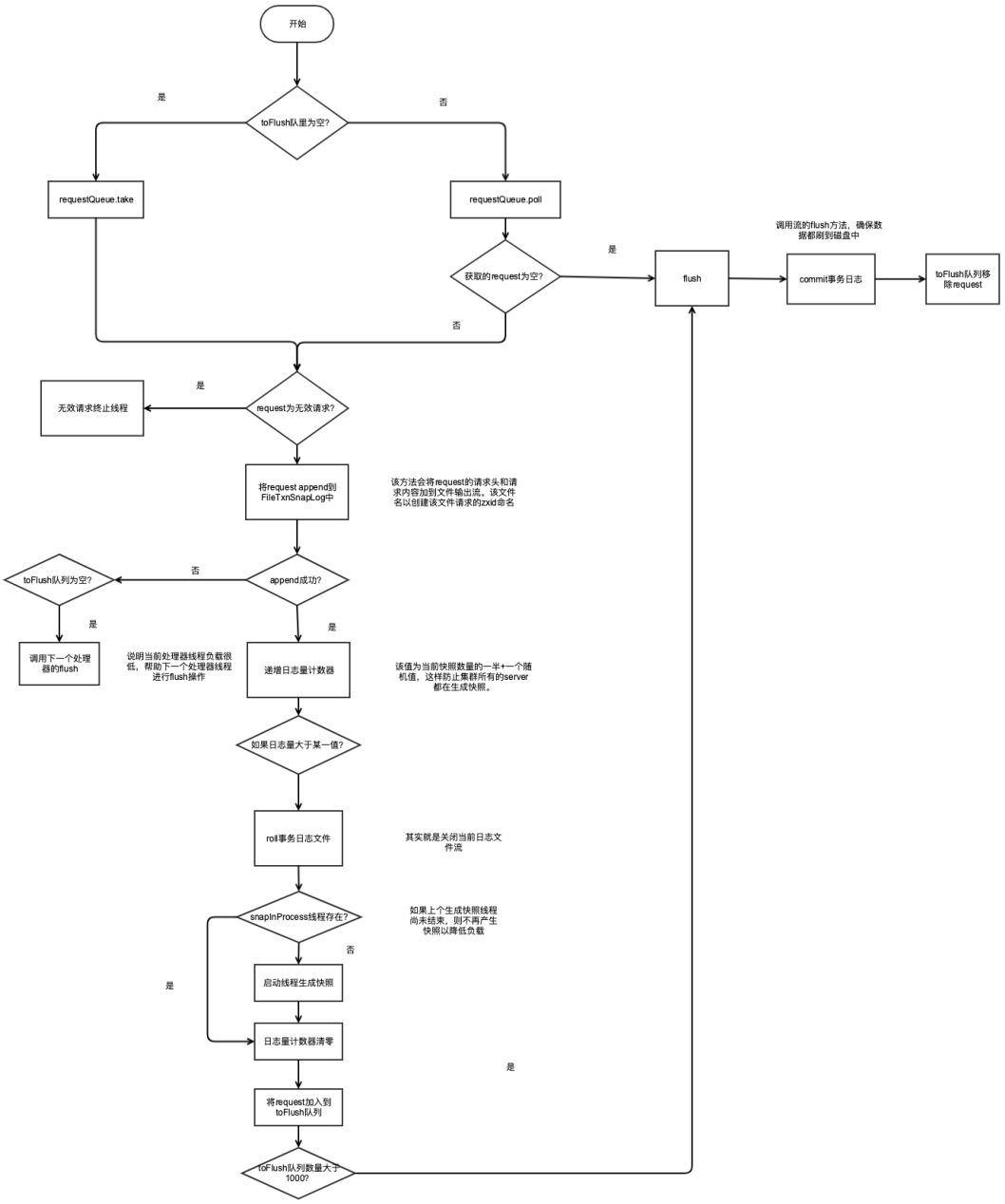
4. 根据不同的请求类型，对解析的数据进行校验处理后，将记录存储在request对象的txn字段中.

5. 统一递增zxid 服务端事务id(描述服务端最新的事务请求)

3.1.2 syncProcessor

同步处理器用来初始事务日志同步到磁盘，只有当事务的日志被同步到磁盘后，才会将请求提交给下一个处理器处理。

该处理器中包含2个队列。1：需要即将被刷盘的日志数据队列toFlush，2 由上个处理器提交过来的request队列。具体交互过程如下:



3.1.3 finalProcessor

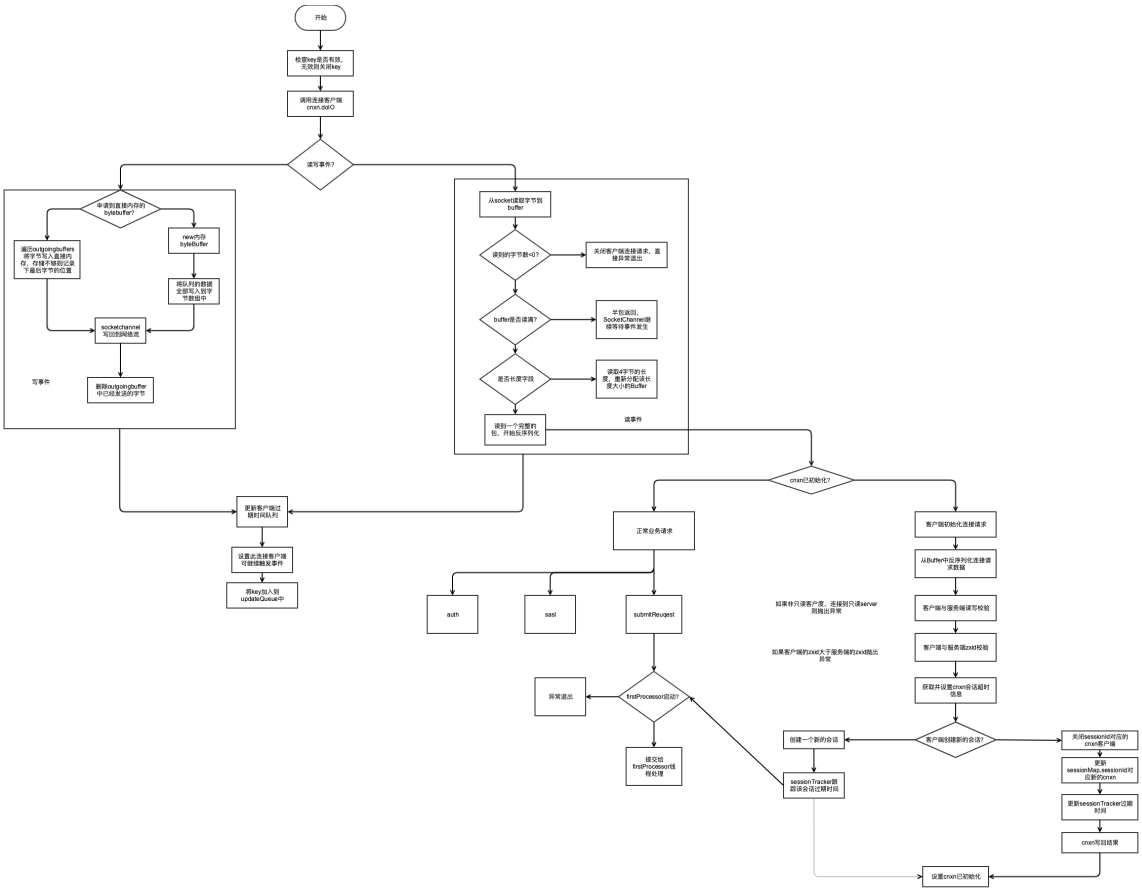
finalProcessor并不是一个线程结构，该处理器负责将客户端的更改事务请求写入到内存的dataTree中，并处理客户端的查询请求。主要根据不同的请求头，对内存数据进行相应的处理，然后通过request中的cnxn连接客户端给client响应数据。主要处理流程如下：

1. 对于更新类型的事务请求，通过zkDatabase的操作，将数据更新到dataTree中，然后删除zk.outstandingChanges队列在PreRequestProcessor中添加的changRecord。
2. 如果是选举类型的请求，在zkDatabase中提交proposal
3. 关闭会话请求，则关闭客户端连接。
4. 对于读取数据的请求，获取数据并生成响应体对象。
5. 生成响应头对象
6. 通过连接客户端cnxn对象将响应消息写回。

3.2 WorkService工作线程组

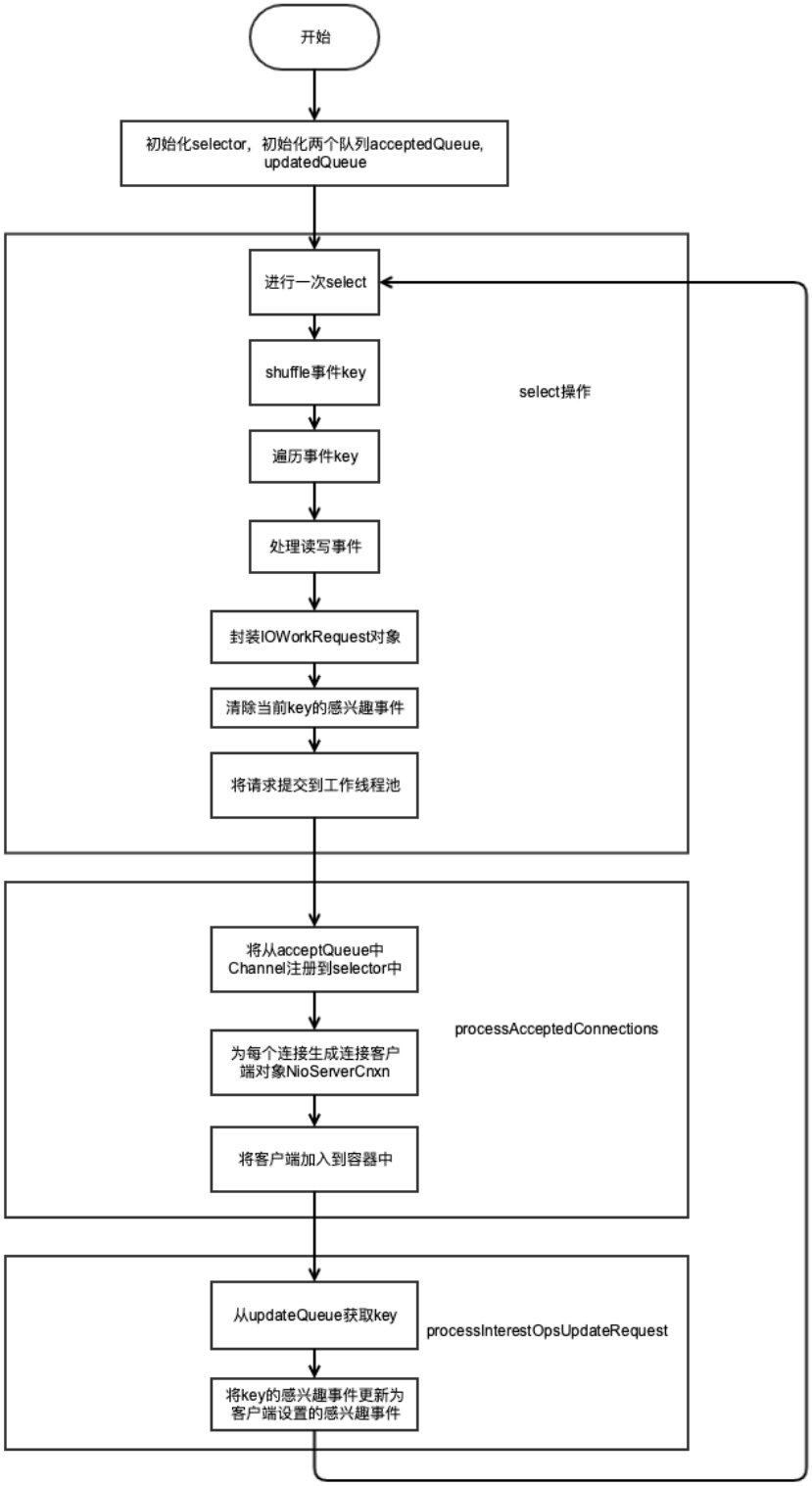
WorkServcie是实际的业务工作线程，负责处理用户请求数据，反序列化等操作。在客户端第一次建立连接的时候，需要创建并初始化会话到数据树中，并进行认证等操作，在这些完成之前都必须避免触发读事件。

注意：在每次接受到业务请求数据的时候，接受数据请求的buffer都是重新创建的所以，除初始化连接需要关闭读事件，其他都不需要。



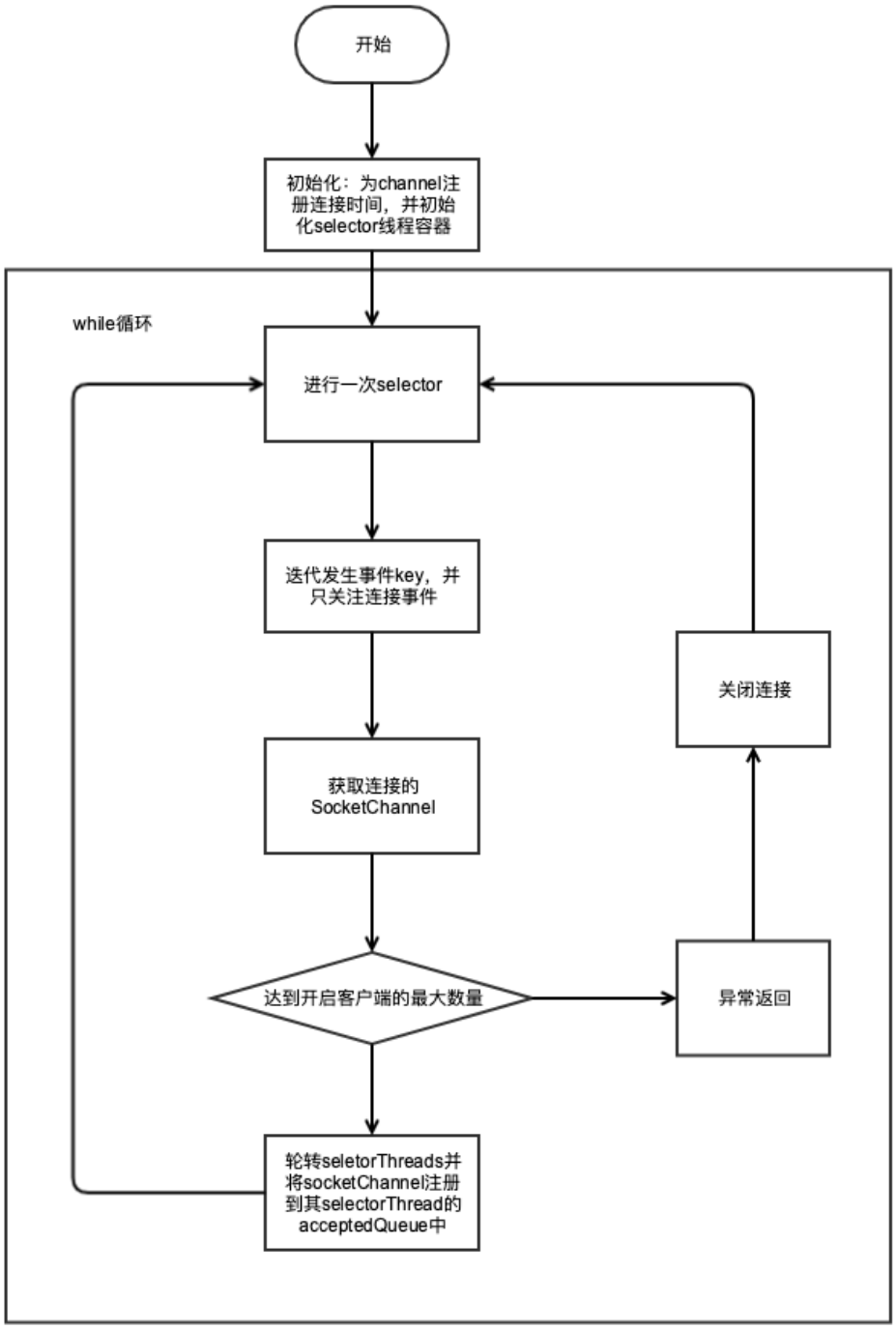
3.3 SelectorThreads

该线程对象中保存了，acceptQueue: LinkedBlockingQueue<SocketChannel>和 updateQueue: LinkedBlockingQueue<SelectionKey>两个队列。 在selectThread中将从acceptQueue获取连接注册到对应的selector.



3.4 AcceptThreads

顾名思义，这个线程负责监听服务端口，维护了acceptSocket:ServerSocketChannel对象
并维护了selectorThreads容器，当获取一个客户端连接后，加入到selectThread的队列中。



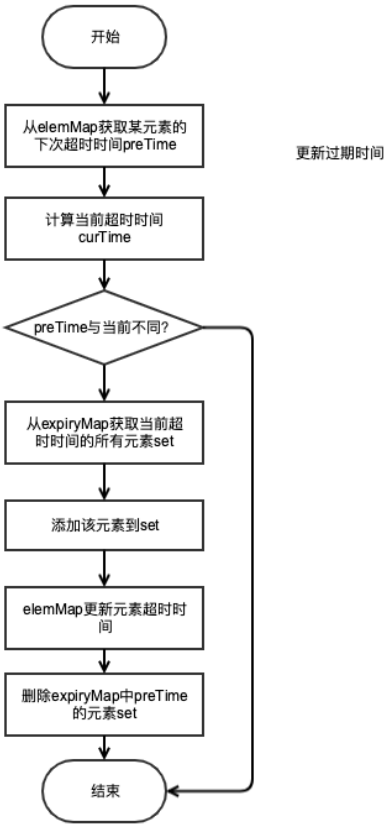
3.5 ExpiryQueue, ConnectionExpiryThread, SessionTracker

3.5.1 expiryQueue

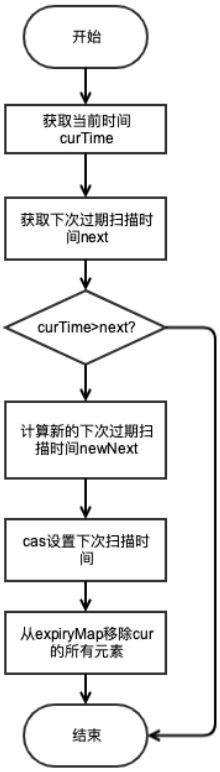
该线程记录可客户端连接的时间，并每过一定时间间隔，对内部容器进行一次遍历，然后将过期的客户端移除。

队列主要封装了3个数据结构：1. elemMap: ConcurrentHashMap<E, Long>, 2. expiryMap: ConcurrentHashMap<Long, Set<E>>, 3. nextExpirationTime: AtomicLong, 4. expirationInterval: int

1. 中主要通过元素获取其过期时间。2. 主要通过过期时间获取所有的元素，3. 下一次过期时间。4. 过期检查时间间隔



更新过期时间



poll过期元素

3.5.2 connectionExpiryThread

该线程主要维护连接客户端 NIOcnxn的过期时间，该线程中等待时间间隔后从expiryQueue.poll出所有过期cnxn然后调用器close方法关闭回收所有资源。

该队列中的元素，在服务监听到一个客户端连接，创建了NIOServerCnxn后被加入。并在每次接受或者处理完该cnxn的读写请求后更新其下次过期时间。

3.5.3 sessiontrackerThread

用户服务端会话过期时间的跟踪，在客户端第一次连接，服务端cnxn实例被创建，并初始化的时候，会生成一个全局sessionId，让后加入到sessionTracker中进行跟踪

由于创建会话需要在zkDb中创建会话记录数据，所以在会话过期的时候，需要删除zkDb中记录的会话，及其对应的节点数据。

与sessionId相关的数据: 1. sessionMap<Long, NIOServerCnxn>，该容器在NIOFactory中维护，在重连的时候可以只替换sessionId对应的cnxn。 2 sessionId<Long, SessionImpl>，该容器在sessionTracker中维护，用于记录session相关的信息(超时时间，是否关闭，id等)。

四、类架构设计

Class Diagram