以太坊工作原理（干货）

<https://blog.csdn.net/Metal1/article/details/80138647>

一．以太坊之P2P网络（以太坊使用Kad协议）

1.单个节点如何连入以太坊网络的？

如果单个节点运行以太坊程序，它是如何接入网络的呢，这是一个先有蛋还是先有鸡的问题。

所以需要有一些固定节点供大家接入，主网的节点是写在变量MainnetBootnodes中（跟比特币类似，比特币也配置了一些dns服务器用来获取其他比特币节点的ip地址），以太坊运行起来的时候先连接配置的这些节点，然后再从连接成功的节点继续获取更多的节点，最终建立自己的节点列表

2.K桶的概念

单个节点会保存若干个其他节点的信息

每个节点的ID是自己随机生成的512位的NodeId（每个节点的id都是这么来的，因为位数多，所以随机生成的基本不会重复），然后跟其他节点的NodeId异或的值（最高位为1的位数）算作节点间的距离，

然后不同距离建立一个list，称为一个K桶（K桶内节点是一个队列），所以最终根据距离的远近建立了若干个K桶

每个K桶有一定的数量限制（待读以太代码确认），当新加入一个节点并且所在K桶已经满的情况下，会先查询下最早加入的队头的节点是否还活跃，如果还活跃移到队尾同时丢弃新加入的节点（因为一个节点如果在线时间越长就意味着继续在线的概率更高），如果不活跃删除节点然后添加新的节点到队尾

3,。节点的存储

以太坊的节点是存储在leveldb里的，存储的节点信息的是节点的NodeId，IP地址和UDP port num等

详情与源码请参考：<https://blog.csdn.net/XinLianOrg/article/details/78667917>

二．关于 Kademlia协议（简称Kad）

 内容包含节点，距离，路由表，协议消息，内容查询，资源查询与存储，新节点加入。

  Kademlia协议（以下简称Kad)，是一种分布式哈希表（DHT，Distributed Hash Table）技术，不过和其他DHT实现技术比较，如Chord、CAN、Pastry等，Kad通过独特的以异或算法（XOR）为距离度量基础，建立了一种全新的DHT拓扑结构，相比于其他算法，大大提高了路由查询速度。

    在Kademlia网络中，所有信息均以的哈希表条目形式加 以存储，这些条目被分散地存储在各个节点上，从而以全网方式构成一张巨大的分布式哈希表。我们可以形象地把这张哈希大表看成是一本字典：只要知道了信息索 引的key，我们便可以通过Kademlia协议来查询其所对应的value信息，而不管这个value信息究竟是存储在哪一个节点之上。在eMule、 BitTorrent等P2P文件交换系统中，Kademlia主要充当了文件信息检索协议这一关键角色，但Kad网络的应用并不仅限于文件交换。

节点

    每个用户都有一个ID号, ID长度根据不同网络而定，例如以太坊中是512bit，eMule是128bit。在系统设计上的好处是——对分布式系统所依赖的物理网络的解耦。 ID是在你第一次使用Kad时随机生成的. 出现两个相同ID的概率实在太小了, 这几乎不可能发生. 我们可以认为, 在Kad网络里, 没有两个用户具有相同的ID号。

     节点ID不仅可以用来做身份标识，还可以用来进行值定位(值通常是文件的散列或者关键词)，节点ID与文件散列直接对应，它所表示的那个节点存储着哪儿能够获取文件和资源的相关信息

    很多 DHT 的设计会让“node ID”采用跟“data key”同构的哈希值。这么搞的好处是：1、当散列值空间足够大的时候，随机碰撞忽略不计，因此也就确保了 node ID 的唯一性。2、可以简化系统设计——比如简化路由算法，采用这种风格来设计路由机制，好处是：key 本身已经提供了足够多的路由信息。

距离

    在Kad网络中，两个节点之间距离并不是依靠物理距离、路由器跳数来衡量的，事实上，Kad网络将任意两个节点之间的距离d定义为其二者ID值的逐比特二进制和数，即异或。假定两个节点的ID分别为a与b，则有：d=a XOR b。在Kad中，每一个节点都可以根据这一距离概念来判断其他节点距离自己的“远近”，当d值大时，节点间距离较远，而当d值小时，则两个节点相距很近。 这里的“远近”和“距离”都只是一种逻辑上的度量描述而已；

    举个例子：

    01010000与01010010距离（即是2个ID的异或值）为00000010（换算为十进制即为2）；

    01000000与00000001距离为01000001（换算为十进制即为26+1，即65）；如此类推。

    异或计算距离的特点，1、节点和它本身之间的异或距离是0。2、异或距离是对称的：即从A到B的异或距离与从B到A的异或距离是等同的。3、异或距离符合三角形不等式：给定三个顶点A B C，假如AC之间的异或距离最大,那么AC之间的异或距离必小于或等于AB异或距离和BC异或距离之和.4、对于给定的一个距离，距离A只存在有唯一的一个节点B，也即单向性，在查找路径上也是单向的，这个和地理距离不同。

    由于以上的这些属性，在实际的节点距离的度量过程中计算量将大大降低。对于越低bit位，XOR可能得到的结果越小，对于越高位的bit位，XOR可能得到的值就越大，并且是呈现2的指数方式增长的，所以，从数学上来说，一个DHT网络中的所有节点，通过这种方式(XOR距离)进行寻址，每次前进一个bit，最大只需要log2N次即可到达目标节点(log2逼近的思路，即bit 2可以表示世界上任何数字)。

路由表

    Kad的路由表是通过一些称之为K桶的表格构造起来的。Kad路由表由多个列表组成，每个列表对应节点ID的一位，一个列表中包含多个条目。列表条目中的这些数据通常是由其他节点的IP地址，端口和节点ID组成。

    每个列表对应于与节点相距"特定范围距离"的一些节点，节点的第n个列表中所找到的节点的第n位与该节点的第n位肯定不同，而前n-1位相同。这就意味着很容易使用网络中远离该节点的一半节点来填充第一个列表(第一位不同的节点最多有一半)，而用网络中四分之一的节点来填充第二个列表(比第一个列表中的那些节点离该节点更近一位)，依次类推。如果ID有128个二进制位，则网络中的每个节点按照不同的异或距离把其他所有的节点分成了128类，ID的每一位对应于其中的一类。

K桶的更新机制非常高效的实现了一种把最近看到的节点更新的策略，除非在线节点一直未从K桶中移出过。也就是说在线时间长的节点具有较高的可能性继续保留在K桶列表中。采用这种机制是基于对Gnutella网络上大量用户行为习惯的研究结果，既节点的失效概率和在线时长成反比关系，用户在线时间越长，他在下一时段继续在线的可能性就越高。

    所以，通过把在线时间长的节点留在K桶里，Kad就明显增加K桶中的节点在下一时间段仍然在线的概率，这对应Kad网络的稳定性和减少网络维护成本（不需要频繁构建节点的路由表）带来很大好处。这种机制的另一个好处是能在一定程度上防御DOS攻击，因为只有当老节点失效后，Kad才会更新K桶的信息，这就避免了通过新节点的加入来泛洪路由信息。

为了防止K桶老化，所有在一定时间之内无更新操作的K桶，都会分别从自己的K桶中随机选择一些节点执行RPC\_PING操作。上述这些K桶机制使Kad缓和了流量瓶颈（所有节点不会同时进行大量的更新操作），同时也能对节点的失效进行迅速响应。

协议消息

Kademlia协议包括四种远程RPC操作：PING、STORE、FIND\_NODE、FIND\_VALUE。

1、PING操作，探测一个节点，用以判断其是否仍然在线。

2、STORE操作，通知一个节点存储一个<key,value>对，以便以后查询需要。

3、FIND\_NODE，本操作的接受者返回它所知道的更接近目标ID的K个节点的信息。这些节点的信息可以是从一个单独的K桶获得，也可以从多个K桶获得（如果最接近目标ID的K桶未满）。不管是哪种情况，接受者都将返回K个节点的信息给操作发起者。但如果接受者所有K桶的节点信息加起来也没有K个，则它会返回全部节点的信息给发起者。

4、FIND\_VALUE操作，和FIND\_NODE操作类似，不同的是它只需要返回一个节点的信息。如果本操作的接受者收到同一个key的STORE操作，则会直接返回存储的value值。

算法的三个参数：keyspace，k和α:

    Keyspace:即ID有多少位；决定每个节点的k桶大小；

    K: 每个一层k-bucket里装k个node的信息，即<node ID, IP Adress, port>；每次查找node时，返回k个node的信息； 对于某个特定的data，离其key最近的k个节点被会要求存储这个data。

α:每次向其他node请求查找某个node时，会向α个node发出请求。

节点查询

    在节点查询的时候，从K桶中找出离所查询的键值最近的K个节点，然后向这K个节点发起FIND\_NODE消息请求。消息接收者收到这些请求消息后将在他们的K桶中进行查询，找到某个条目和目标节点XOR为0，即已经寻址成功，则直接返回；如果没找到XOR结果为0的条目，则选取那个XOR值最小的条目对应的K桶中的K个条目返回给查询者。

     消息的请求者在收到响应后将使用它所收到的响应结果来更新它的结果列表，返回的结果也应该插入到刚才发起请求的那个K桶里，这个结果列表总是保持K个响应FIND\_NODE消息请求的最优节点； 然后向这K个最优节点发起查询，因为刚开始的查询很可能K桶里存的不全是目标节点，而是潜在地离目标节点较近的节点；

     不断地迭代执行上述查询过程。因为每一个节点比其他节点对它周边的节点有更好的感知能力，因此响应结果将是一次一次离被搜索键值越来越近的某节点。如果本次响应结果中的节点没有比前次响应结果中的节点离被搜索键值更近了，这个查询迭代也就终止了；

     当这个迭代终止的时候，响应结果集中的K个最优节点就是整个网络中离被搜索键值最近的K个节点，从以上过程看，这显然是局部的，而非整个网络，因为这本质和最优解搜索算法一样，可能陷入局部最优解而无法获得全局最优解；节点信息中可以增加一个往返时间，或者叫做RTT的参数，这个参数可以被用来定义一个针对每个被查询节点的超时设置，即当向某个节点发起的查询超时的时候，另一个查询才会发起，当然，针对某个节点的查询在同一时刻从来不超过α个。

资源查询与存储

    当某个节点得到了新加入的数据（K/V），它会先计算自己与新数据的 key 之间的“距离”；然后再计算它所知道的其它节点与这个 key 的距离。如果计算下来，自己与 key 的距离最小，那么这个数据就保持在自己这里。否则的话，把这个数据转发给距离最小的节点。收到数据的另一个节点，也采用上述过程进行处理（递归处理）。

    当某个节点接收到查询数据的请求（key），它会先计算自己与 key 之间的“距离”；然后再计算它所知道的其它节点与这个 key 的距离。如果计算下来，自己与 key 的距离最小，那么就在自己这里找有没有 key 对应的 value。有的话就返回 value，没有的话就报错。否则的话，把这个数据转发给距离最小的节点。收到数据的另一个节点，也采用上述过程进行处理（递归处理）。

    考虑到节点未必都在线的情况，资源的值被存在多个节点上(节点中的K个)，并且，为了提供冗余，还有可能在更多的节点上储存值；储存值的节点将定期搜索网络中与储存值所对应的键接近的K个节点并且把值复制到这些节点上，这些节点可作为那些下线的节点的补充 。

    对于那些普遍流行的内容，可能有更多的请求需求，通过让那些访问值的节点把值存储在附近的一些节点上来减少存储值的那些节点的负载，这种新的存储技术就是缓存技术，通过这种技术，依赖于请求的数量，资源的值被存储在离键越来越远的那些节点上(资源热度越高，缓存cache就越广泛)，这使得那些流行的搜索可以更快地找到资源的储存者 。

    由于返回值的节点的NODE\_ID远离值所对应的关键字，网络中的"热点"区域存在的可能性也降低了。依据与键的距离，缓存的那些节点在一段时间以后将会删除所存储的缓存值。DHT的某些实现(如Kad)即不提供冗余(复制)节点也不提供缓存，这主要是为了能够快速减少系统中的陈旧信息。在这种网络中，提供文件的那些节点将会周期性地更新网络上的信息(通过NODE\_LOOKUP消息和STORE消息)。当存有某个文件的所有节点都下线了，关于该文件的相关的值(源和关键字)的更新也就停止了，该文件的相关信息也就从网络上完全消失了。

新节点加入

1、想要加入网络的节点首先要经历一个引导过程。在引导过程中，节点需要知道其他已加入该网络的某个节点的IP地址和端口号。随机生成一个散列值作为自己的 ID；

2. 新节点向它的唯一邻居(引导节点)发起NODE\_LOOKUP操作请求来定位自己，这种"自我定位"将使得Kademlia的其他节点(收到请求的节点)能够使用新加入节点的ID填充他们的K桶。同时也能够使用那些查询过程的中间节点来填充新加入节点的K桶。这一自查询过程使得新加入节点自引导节点所在的那个K桶开始，由远及近，对沿途的所有节点逐步得到刷新，整条链路上的邻居都认识了这个新邻居

3. 最初的时候，节点仅有一个K桶(覆盖所有的ID范围)，当有新节点需要插入该K桶时，如果K桶已满，K桶就开始分裂，分裂发生在节点的K桶的覆盖范围包含了该节点本身的ID的时候。对于节点内距离节点最近的那个K桶，Kademlia可以放松限制(即可以到达K时不发生分裂)，因为桶内的所有节点离该节点距离最近，这些节点个数很可能超过K个，而且节点希望知道所有的这些最近的节点。因此，在路由树中，该节点附近很可能出现高度不平衡的二叉子树。假如K是20，新的节点可能有21个以上。这点保证使得该节点能够感知网络中附近区域的所有节点。

中心化的P2P网络中，会有一个server用来搜集peer信息，这样在数据交互过程中，每个peer一般情况下都是先通过这个server拿到一定数量的peer列表，然后挨个去建立链接，最后进行数据交互。但是众所周知，区块链是一个去中心化的系统，这种server的存在将会彻底破坏区块链可信任的基础，以太坊通过Kademlia算法（分布式存储及路由的算法）保证经过最多n步后找到需要的数据。详情：<https://www.jianshu.com/p/f2c31e632f1d>（通俗易懂）

P2P源码分析

分三部分说明：

1.类型或类说明，这部分单独拎出来介绍主要是因为很庞杂，如果没有一个快速检索的地方，后面介绍代码的时候需要来回查看，降低了效率

2.重点环节说明，至少会将主干流程完整的按照代码走一遍，可能涉及很多代码，如果只是说函数名走完流程的话，其实是丧失了很多细节，而且也无法体会作者的创新独到之处

3.整体结构分析及总结，这一点与我们分析各个部件来完成整个项目的了解如出一辙，对不同类、不同环节了解清楚之后，会自然而然对整个架构设计有了进一步的了解，由点及面，然后再从宏观角度去审视，可以感受到不一样的风景

一．基础类型描述参考资料：<https://blog.csdn.net/fusan2004/article/details/80903365>

该部分类型定义只包括成员变量的定义，不涉及复杂的逻辑，只要在节点和网络方面。注意Node，NodeEntry等关键词。

二．节点发现流程参考代码：<https://blog.csdn.net/fusan2004/article/details/80906455>

三．整体结构分析，UDP底层通信：

<https://blog.csdn.net/fusan2004/article/details/81024742>

上一篇讲的是节点发现是通过UDP的方式来进行的，三中介绍udp通信的详细细节。

以上是以kadmlia协议基于C++的源码。

以太坊详细CPP解析：

第一部分包含以太坊架构，CPP代码的结构简要包含构成的模块：

<https://blog.csdn.net/Metal1/article/details/80116186>

第二部分包含SHA-3哈希加密，RLP编码的概念。区块和区块头的定义、数据结构，代码中详细展示h256类型定义：

<https://blog.csdn.net/Metal1/article/details/80203233>

第三部分实则是一些算法，包含难度计算、难度校验

<https://blog.csdn.net/Metal1/article/details/80206028>

该部分是基于C++的以太坊开发。