TrueChain main network-p2p module node discovery mechanism

The Beta version of the initial chain main website was officially launched at 08:00 on September 28, 2018, Singapore time. During this period, many blockchain enthusiasts wrote blogs based on their personal understanding of the initial chain source code. Everyone learns and draws on. Today, I will talk about the node discovery mechanism that I understand based on the initial source code.

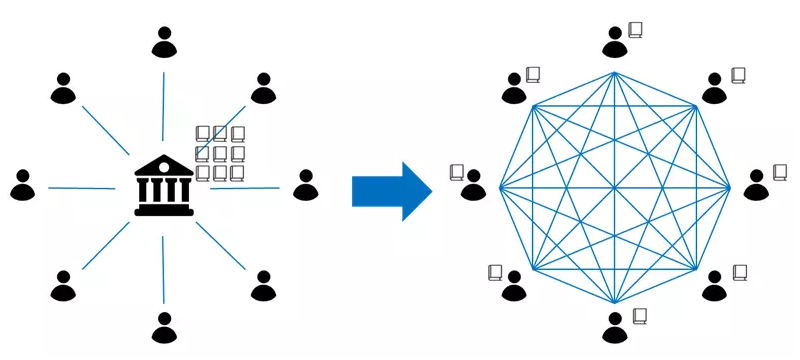
The initial chain is based on the kademlia protocol to implement its automatic node discovery mechanism, which completes the construction and refresh of the entire network topology relationship.

1. So we have to understand what is kademlia first?

Kademlia was proposed by PetarP.Manmounkov and David Mazieres of New York University in 2002 as a distributed hash table (DHT) technology. Kademlia built a new model based on the unique XOR algorithm. DHT topology, greatly improved routing query speed compared to other algorithms。

Now let's use an example to understand what Kademlia's distributed storage and routing algorithms are.

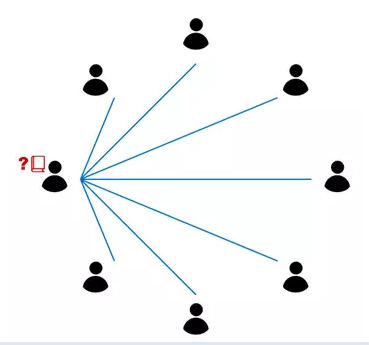
Imagining a school which consists of 1,000 students. Now the school suddenly decided to close the library (do not set up a centralized server), and distribute all the books in the library to each student (all the files are scattered and stored in each on the node). That is, all the students together form a distributed library.



there are several problems:

Which books are assigned to each student? That is, how to allocate storage content to each node, how to add/delete content?

When you need to find a book, such as "C++", how do you know which classmate has "C++"? (A question for 1000 people, "Do you have "C++"?", obviously an uneconomical practice), how to contact this classmate. That is, if a node wants to obtain a specific file, how to find the node/address/path of the stored file.



Let's see how the Kademlia algorithm cleverly solves these problems?

Elements of the node:

First let's see what properties each classmate (node) has?

Student ID (Node ID, binary, 160 bit).

Mobile number (node's IP address and port).

Each classmate will have below information:

Books distributed from the library (allocated to the content that needs to be stored), each book has a book name and book content (the content is stored in the form of <key, value>, which can be understood as the file name and file content);

An address book containing a small number of other students' student numbers and mobile phone numbers. The address book is layered by student number (a routing table called "k-bucket", layered by Node ID, and records a limited number of other nodes. ID and IP address and port).

According to the above analogy, you can look at this table:

|  |  |
| --- | --- |
| Protocol concept | Analogy concept |
| Node ID | Student's student number |
| IP Address ,port | Student’s phone number |
| Key | The hash value of the book’s name |
| Value | Book |
| Routing table ,K-bucket | An address book maintained by each student, and some students <student number, phone number>are recorded in the address book. |

About why not every student has a full address book (each node maintains full routing information)?

First, the entry and exit of nodes in a distributed system is quite frequent. Whenever there is a change, the entire network broadcasts the address book update, and the traffic volume will be large;

Second, once any classmate is kidnapped by the bad guys (the node is hacked), the bad guys immediately have the mobile number of everyone, which is not safe.

File storage and lookup

in the library, the books that are neatly placed by the index number, in what way is it distributed to the students? The general principles are as follows:

1) Books can be distributed more evenly among the students, and there will be no cases where some of the students have a lot of books, and most of them do not even have a book;

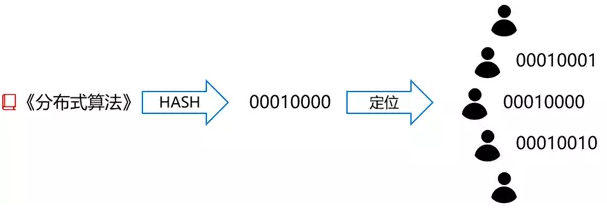
2) When a student wants to find a particular book, he can find the book in a relatively simple way.

Kademlia made the following arrangement:

Assuming that the book name for the book "C++" is 00010000, then the book will be belong to the student with the student number 00010000. (This requires that the value range of the hash algorithm is consistent with the value range of the node ID. Kademlia's Node ID is 160-bit binary. The example here is abbreviated for the Node ID.)

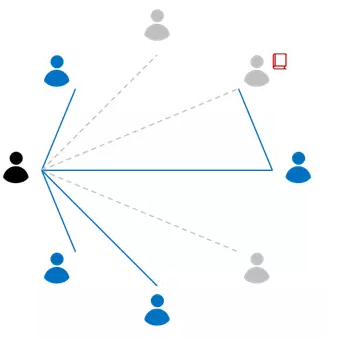
But it is also necessary to consider that may be some student is absent. If 00010000 didn't come to school today (the node didn't go online or completely not access to the network), How to get “C++ “ book? The algorithm requires that the book not only exist in one student's hand, but is required to be stored in the hands of the k students with the student number closest to 00010000, that is, 00010001, 00010010, 00010011... and other students will have this book.

Similarly, when you need to find the book "C++", hash the title and get 00010000. This is the call number, and you know which class(s) you should find. The remaining question is to find the mobile phone number of this (several) classmate.



XOR distance of nodes

Since you only have a part of your classmate's address book, you probably don't have a phone number (IP address) of 00010000. How do you contact that classmates?



A workable idea is to find a student with the contact information of the target classmate in your address book. As mentioned earlier, the address book of each student's hand is layered by distance. The design of the algorithm is that if a classmate is closer to you, the probability that the mobile phone number of his or her is stored in the address book of your hand is greater. The core idea of the algorithm can be: When you know the distance between the target student Z and you, you can find a class B that you think is the closest to classmate Z in your address book, and ask class B to go further. Go find the phone number of classmate Z.

The distance mentioned above is the XOR distance between the student IDs (Node IDs). XOR is an operation for yes/no or binary.

The algorithm of XOR is: 0⊕0=0,1⊕0=1,0⊕1=1,1⊕1=0 (the same as 0, the difference is 1)

Given two examples:

1) The distance between 01010000 and 01010010 (that is, the XOR value of 2 IDs) is 00000010 (converted to decimal is 2);

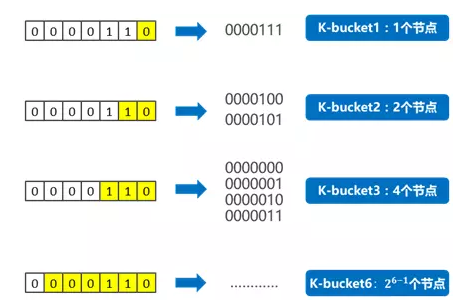
2) 01000000 and 00000001 distance is 01000001 (converted to decimal is 26+1, which is 65); And so on.

How does the address book layer by distance? The following example will tell you that layering by XOR distance is basically understood as layering by number of bits. Imagine the following scenario:

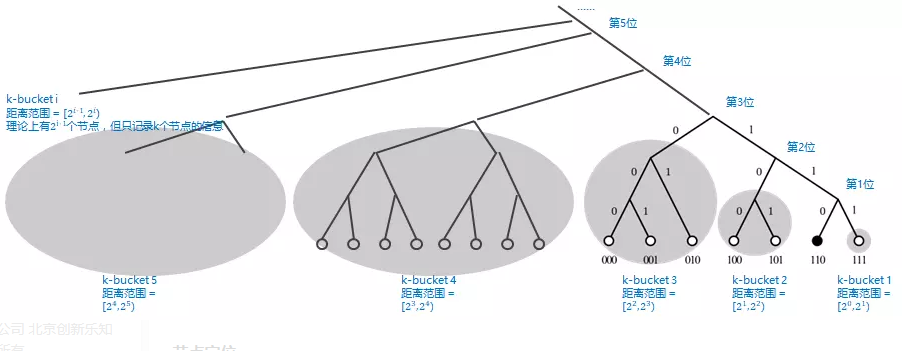
Based on 0000110, if the ID of a node is the same as all the previous digits, only the last one is different. There is only one such node - 0000111, and the XOR value of the base node is 0000001, that is, the distance is 1; for 0000110, such a node is classified as "k-bucket 1";

If the ID of a node, all the previous digits are the same, starting from the second digit of the last digit, there are only two such nodes: 0000101, 0000100, and the XOR values with the base node are 0000011 and 0000010, that is, the distance range is 3 and 2. For 0000110, such a node is classified as "k-bucket 2";

If the ID of a node, all the previous digits are the same, starting from the nth digit of the reciprocal, such a node has only 2 (i-1), and the distance from the base node is [2(i-1), 2i); For 0000110, such a node is classified as "k-bucket i";



Another way of understanding the above description: If the nodes of the entire network are sorted into a binary tree arranged by the node ID, and each leaf at the end of the tree is a node, the following figure is more intuitively displayed, the node’s relationship between the distances.



Go back to our analogy. Each classmate only maintains a part of the address book. This address book is layered according to distance (it can be understood as layering according to the student number and its own student number from the first few), i.e. k-bucket1, k-bucket 2, K-bucket 3... Although the number of students actually present in each k-bucket is gradually increasing, each classmate only records the mobile phone number of k students in each of its own k-buckets (address and port of k nodes) , where k is an adjustable constant parameter).

Since the student number (node ID) has 160 bits, each student's address book is divided into 160 layers (the node has a total of 160 k-buckets). The entire network can accommodate up to 2^160 students (nodes), but each classmate (node) maintains only up to 160 \* k line contacts (addresses and ports of other nodes).

Node positioning

Let us now elaborate on a complete book-entry process.

A classmate (study number 00000110) wants to find "C++", A first needs to calculate the hash value of the book name, hash ("C++") = 00010000. Then A knows that ta needs to find the class 00010000 (named Z classmate) or the student with the student number and Z.

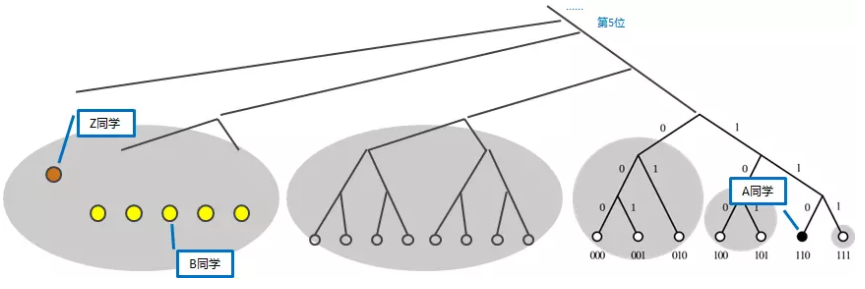
Z's student number 00010000 and his own XOR distance is 00010110, the distance range is [24, 25), so this Z classmate may be in k-bucket 5 (or, Z classmate's student number and A classmate's student number from The 5th place is different, so Z students may be in k-bucket 5).Then A students look at their k-bucket 5 and have Z classmates:

If Z’s classmate has it, then contact Z’s classmate directly;

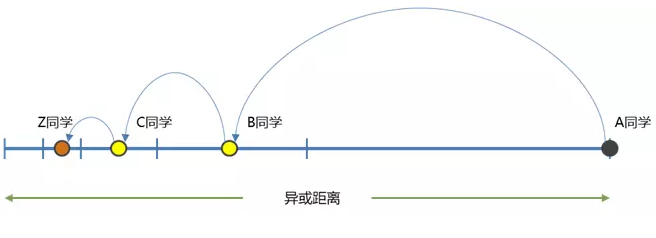
If not, find a B classmate in k-bucket 5 (note any B classmate, its 5th student number is definitely the same as Z, that is, its distance from Z classmate will be less than 24, which is equivalent to Z, A The distance between the two is shortened by more than half). Ask the B classmate to find the Z classmate in the same way of finding in their own address book:

If B knows Z classmates, then tell Z's mobile phone number (IP Address) to A;

If B does not know Z, then B can find a C student who is closer to Z in his address book (the distance between Z and C is less than 23), and recommend C to A; A classmate C asked C students to do the next step.



Kademlia's query mechanism is a bit like arbitrarily folding a piece of paper to narrow the search range, ensuring that for any n students, at most only log 2 (n) times, you can find the contact information of the target students (ie For any network with [2(n−1), 2n) nodes, you only need n steps to find the target node).



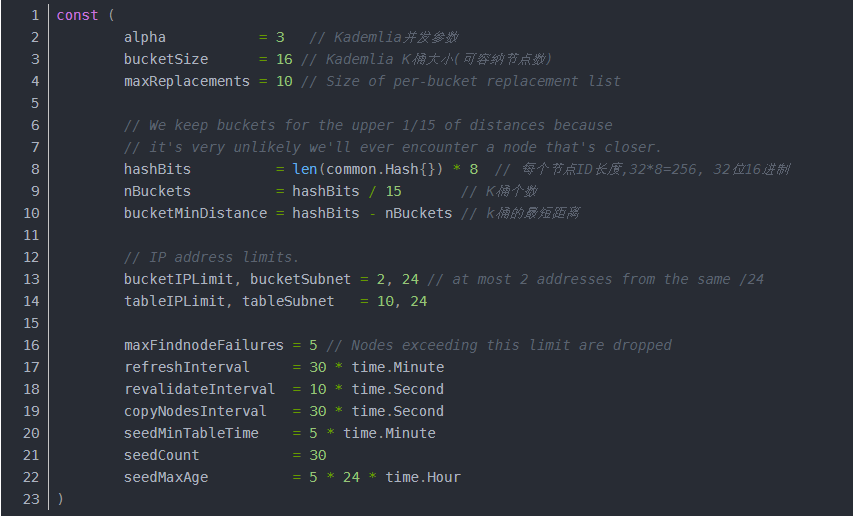
The above is the basic principle of Kademlia algorithm

**Kademlia-like protocol**

The kademlia network (kad for short) and the standard kad network are partially different.

The following is a comparison of the initial chain source, explaining several concepts in the kad network:

The source code is located at p2p/discover/table. Go



Alpha, which is an optimization parameter in the system, controls the maximum number of nodes taken from the K bucket each time, and the initial chain value is 3.

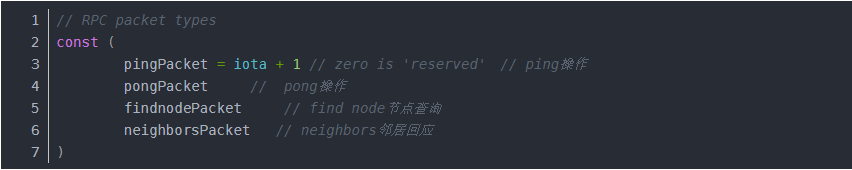
Bucket Size, K bucket size, initial chain take 16

Hash Bits, node length 256 bits

nBuckets, the number of K buckets, currently taking 17

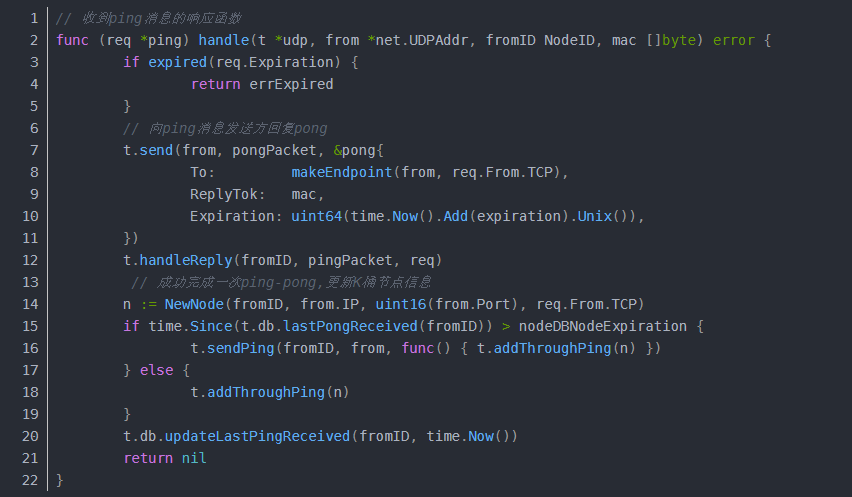
The inter-node communication in the initial chain Kad network is based on UDP. The Kad network defines a total of four message types:

The source code is located at p2p/discover/udp.go



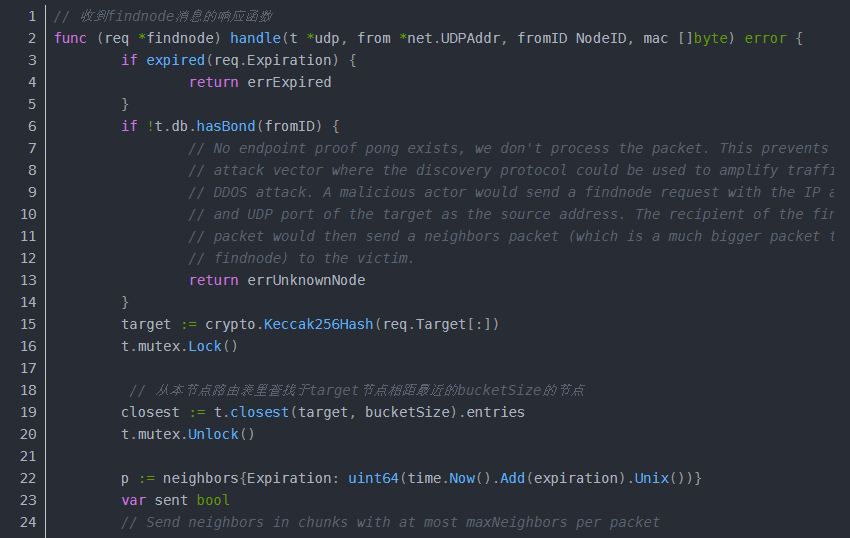
Ping and pong are a pair of operations used to detect node activity. The node replies to the pong response immediately after receiving the ping message:

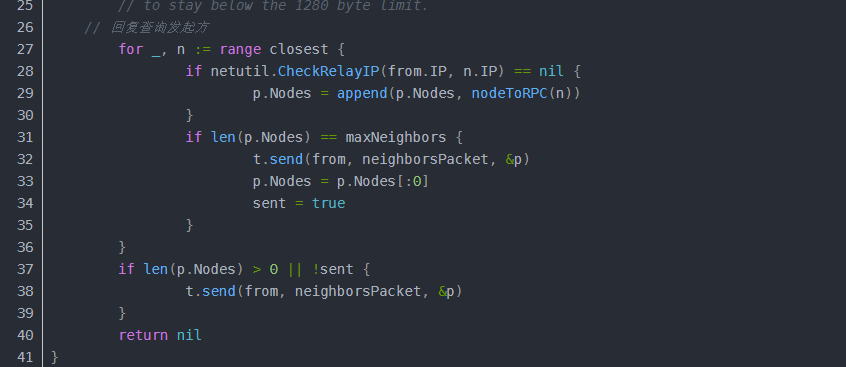
The source code is located at p2p/discover/udp.go



Findnode and neighbors are a pair of operations. Findnode is used to find the node closest to a node. After finding it, it will reply to the initiator with a neighbor’s type message.

The source code is located at p2p/discover/udp.go





Initial chain node discovery mechanism

See how the initial chain node discovery mechanism flows.

First, start UDP "port listening" when the node starts:

server. Start()==>discover.ListenUDP ==> newUDP()

newUDP() forks out three processes, all of which are infinite loops:

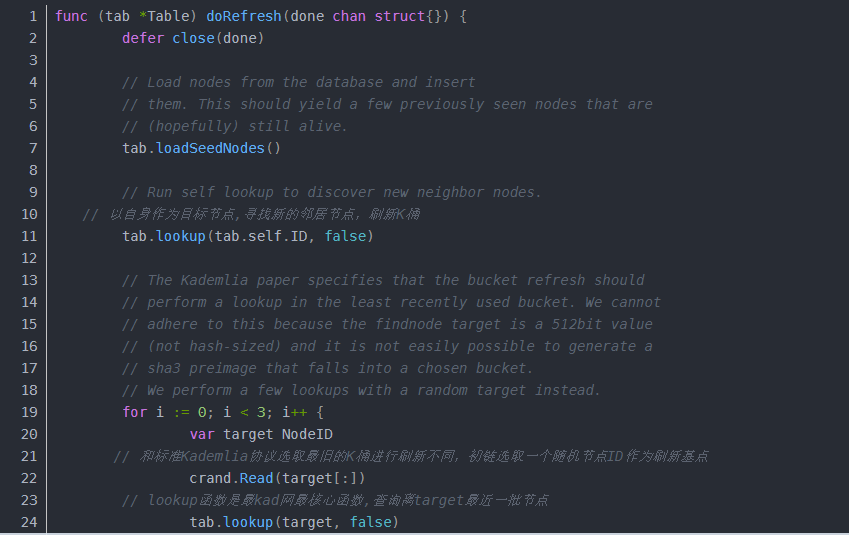
Func (tab \*Table) doRefresh()

Func (t \*udp) loop()

Func (t \*udp) readLoop(unhandled chan ReadPacket)

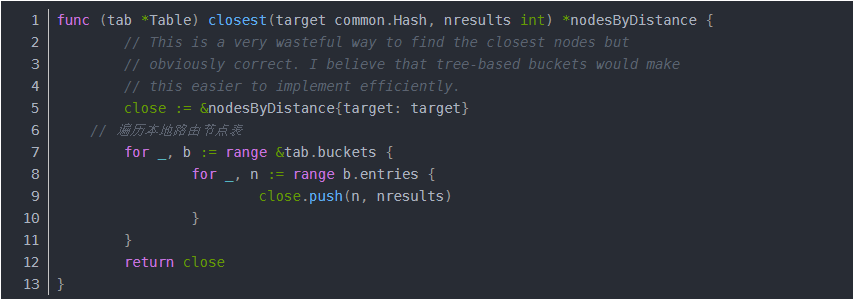
doRefresh()

The process refreshes the K bucket every hour or on demand. The core logic implementation is located in the doRefresh function. The source code is located at p2p/discover/table.go:

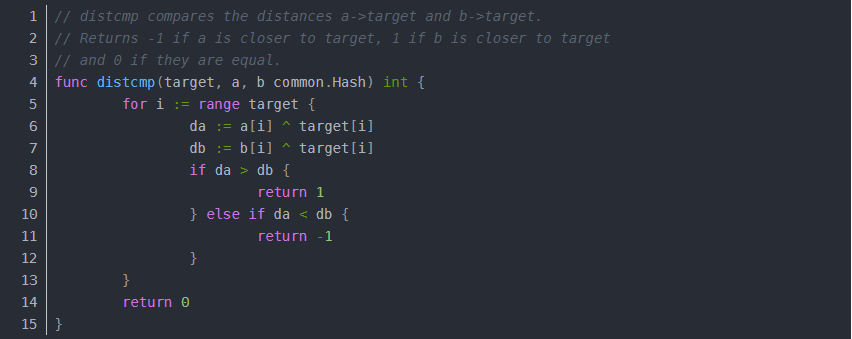


The tab.lookup function is critical, but its logic is actually very simple.

First query the nearest batch of nodes from the target, the distance calculation is the realization of the XOR (XOR) distance calculation of the kad network, the source code is located at p2p/discover/table.go



Close.push finally calls dispcmp for XOR calculation. The source code is located at p2p/discover/node.go



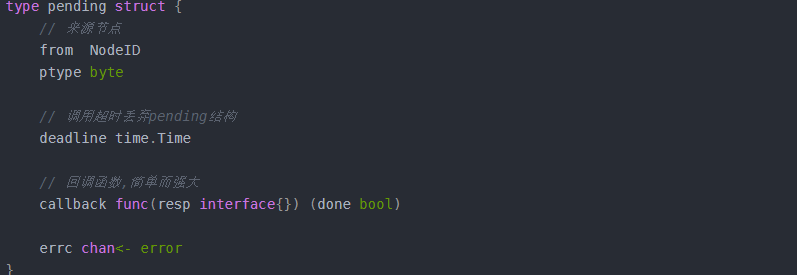
Then iterate all the nodes found in the previous step, initiate a findnode operation to these nodes to query the list of nodes closest to the target node, perform ping-pong test on the obtained nodes, and save the nodes that pass the test.

After this process, the node's K bucket can update different network nodes to the local K bucket more evenly.

Loop() and readLoop()

The two loop processes are put together to say that they are mainly an engineering implementation that serializes the asynchronous call code through the channel. The business is mainly responsible for handling the sending and receiving of four message types: ping, pong, findnode, and neighbors.

The only thing worth mentioning is the pending structure:



Overview, neighbor node discovery process:

1) The system starts the random generation of the local node NodeId for the first time, which is the LocalId, which is fixed after generation.

This node is generated the first time it is started and will not change after a subsequent restart. Each node will have a unique flag NodeId. A and B have their own NODEid

2) The system reads the public node information, and after the ping-pang handshake is completed, writes it to the K bucket.

Read the public node, which means that everyone knows that each node has the same public node information. Set to C. That is to say, A does not know B, B does not know A, but both A and B know C. This is C point, that is, know A point and B point.

3) Enter the brush bucket cycle

a) Randomly generate the target node Id, recorded as TargetId, and record the number of discovery and refresh time from 1. Each node generates a target node id, which means that each node initiates a brush bucket operation at the same time.

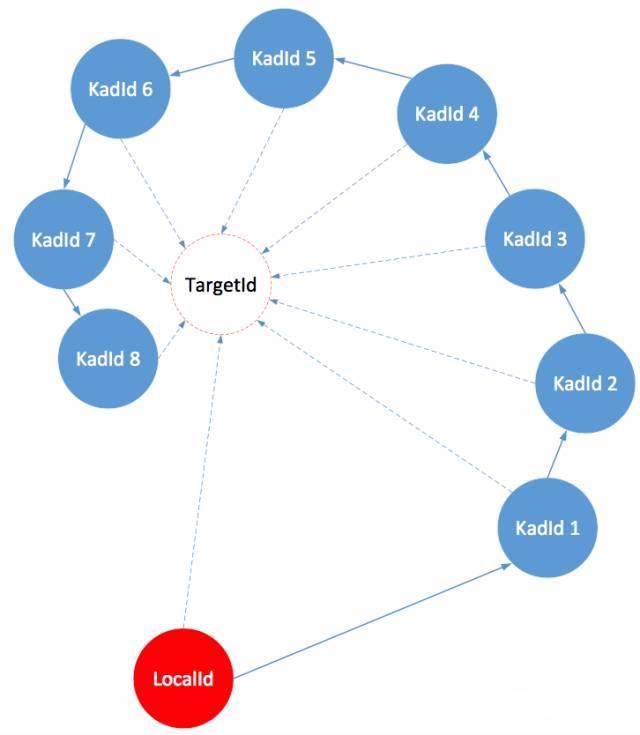
b) Calculate the distance between TargetId and LocalId, and record it as Dlt to calculate the distance between the local node and the target node.

c) The node NodeId in the k bucket is recorded as KadId, and the distance between KadId and TargetId is calculated, which is recorded as the public node information in the k-pass in each node of the Dkt.

d) Find the node in the K bucket with Dlt greater than Dkt, denoted as the K bucket node, send the FindNODE command to the k bucket, and the FindNODE command includes the TargetId. Because the K bucket node already knows the local node that sends the FindNode command, it is now necessary to record the local node. information. That is to say, A sends a message to point C, and C does not need to record the information of point A.

e) After receiving the FindNODE command, the k-bucket node uniformly executes the b-d process, and the node found from the K-pass is sent back to the local node using the Neighboras command. Point C sends the information of point B to point A.

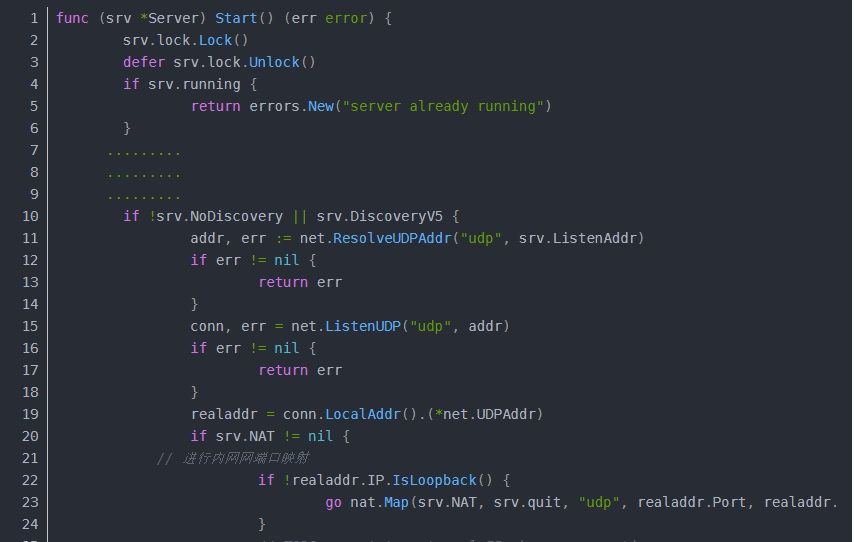
f) After receiving the Neighbour, the local node writes the received node to the K bucket, and the A point records the B information, so that the A point knows the B point. In the same process, point B knows point A through point C.

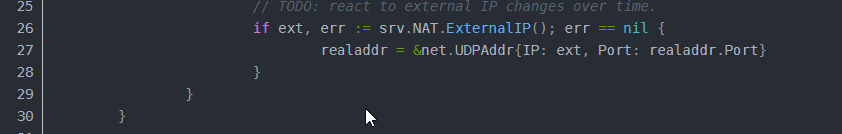


Internal network penetration

The initial chain is based on p2p communication. All operations may involve intranet penetration. At present, the most common method for intranet penetration is udp hole, which is why kad network uses udp as the basic communication protocol.

The specific code was not found, but the trace can be seen in the node initialization function. nat.Map() method port mapping: the source code is located at p2p/server.go.





First, the initial chain tcp/udp shares a port, and then uses the uPnp protocol cluster for internal and external network port mapping to complete the link and pass through the intranet.

The package is located in the nat module, and the implementation also uses the three-party library goupnp.

The above is the part of the p2p module node discovery mechanism of the initial version of the primary network Beta.