1. RPC packet types 四类操作

|  |
| --- |
| const (      pingPacket = iota + 1 // zero is 'reserved'      pongPacket      findnodePacket      neighborsPacket  ) |

Ping与pong消息是一对

|  |
| --- |
| Type ping struct {  Version uint  From, To rpcEndpoint //ping操作需要同时包含源端与目的端地址  Expiration uint64 //此消息的存活时间  } |

|  |
| --- |
| Type pong struct {  // This field should mirror the UDP envelope address  // of the ping packet, which provides a way to discover the  // the external address (after NAT).  To rpcEndpoint  ReplyTok []byte // ping消息的哈希值  Expiration uint64 // Absolute timestamp at which the packet becomes invalid.  } |

findnode是对接近给定目标的节点的查询，与neighbors消息是一对

|  |
| --- |
| Type findnode struct {  Target NodeID // doesn't need to be an actual public key  Expiration uint64  } |

reply to findnode

|  |
| --- |
| Type neighbors struct {  Nodes []rpcNode  Expiration uint64  } |

1. 两种类型的节点结构体

|  |
| --- |
| Type rpcNode struct {  IP net.IP // len 4 for IPv4 or 16 for IPv6  UDP uint16 // for discovery protocoll'l'l'l'l'l'l'l'l'l'l'l'l'l'l'l'l'l'l  TCP uint16 // for RLPx protocol  ID NodeID  }  Type rpcEndpoint struct {          IP  net.IP // len 4 for IPv4 or 16 for IPv6          UDP uint16 // for discovery protocol          TCP uint16 // for RLPx protocol      } |

1. 函数

|  |
| --- |
| func makeEndpoint(addr \*net.UDPAddr, tcpPort uint16) rpcEndpoint  根据UDP地址和TCP端口，返回一个rpcEndpoint类型的结构体 |

|  |
| --- |
| func nodeFromRPC(rn rpcNode) (n \*Node, valid bool)  判断传入的节点是否为有效节点(检查IP地址是否有效)，若有效则将rpcNode类型转换为Node类型 |

|  |
| --- |
| func nodeToRPC(n \*Node) rpcNode  将Node类型转化为rpcNode类型 |

|  |
| --- |
| func ListenUDP(priv \*ecdsa.PrivateKey, laddr string, natm nat.Interface, nodeDBPath string) (\*Table, error)  根据传入的本地UDP地址创建一个新的Kad路由表(尚未包含其他远端节点信息)，包括本节点的UDP通信socket、私钥、nat映射信息、数据库路径(底层调用newUDP函数) |

|  |
| --- |
| func ParseUDP(priv \*ecdsa.PrivateKey, laddr string) (\*Node, error)  根据给定的监听地址和私钥，创建并返回一个新的Node结构体 |

|  |
| --- |
| func newUDP(priv \*ecdsa.PrivateKey, c conn, natm nat.Interface, nodeDBPath string) (\*Table, \*udp) 涉及了nat包  根据形参的信息，新建udp类型结构体和Kad路由表  并调用nat包完成NAT上的端口映射(因此本节点的IP:PORT将是NAT上的映射地址)  go udp.loop() 🡪 作用：本协程负责处理所有挂起事件(检测挂起事件是否发生)  go udp.readLoop() 🡪作用：本协程负责读取来自其他节点的消息 |

1. Udp结构体和相关的结构体 (重点)

|  |
| --- |
| type udp struct {  conn conn //接收udp数据包的IP:PORT对应的socket  priv \*ecdsa.PrivateKey //私钥  ourEndpoint rpcEndpoint //目的端点地址  addpending chan \*pending //等待处理(挂起事件)管道  gotreply chan reply //响应管道  closing chan struct{}  nat nat.Interface  \*Table  } |

|  |
| --- |
| type pending struct {      from  NodeID      ptype byte      deadline time.Time      callback func(resp interface{}) (done bool)      errc chan<- error //only write channel  } |

|  |
| --- |
| type reply struct {      from  NodeID      ptype byte      data  interface{}      matched chan<- bool  } |

1. Udp结构体的方法

|  |
| --- |
| func (t \*udp) close()  关闭字段closing管道，关闭socket conn字段 |

所有挂起事件的回调函数都在loop方法的case r := <-t.gotreply下根据事件是否真实发生进行返回

|  |
| --- |
| func (t \*udp) ping(toid NodeID, toaddr \*net.UDPAddr) error  向指定的远端节点发送ping消息，同时等待对方回复   1. 调用t.send向远端节点发送消息 2. 调用t.pending在本地挂起一个回调(pong包的事件)，等待来自远端节点的pong回复。若收到pong回复，回调函数返回true，因此ping方法会返回true 🡪见loop方法的case r := <-t.gotreply |

|  |
| --- |
| func (t \*udp) waitping(from NodeID) error  等待指定的远端节点的ping消息。  底层调用t.pending在本地挂起一个回调，等待来自该远端节点的ping消息。若收到ping消息，此方法会返回true |

|  |
| --- |
| func (t \*udp) findnode(toid NodeID, toaddr \*net.UDPAddr, target NodeID) ([]\*Node, error)  findnode方法发送findnode请求给指定的节点，然后等待此节点将它的k个邻居发送完成   1. 调用t.send向指定的远端节点发送findnode消息 2. 调用t.pending在本地挂起一个回调，等待来自远端节点的neighbors回复。 3. 若收到neighbors回复，从回调函数中取出该远端节点的若干个邻居节点。选取不超过bucketSize个邻居节点作为本方法的返回值 |

|  |
| --- |
| func (t \*udp) pending(id NodeID, ptype byte, callback func(interface{}) bool) <-chan error {  pending方法为每一个等待处理的事件(四种RPC操作的一种，而且是来自于指定远端节点的操作)添加一个回调函数。该函数返回值是一个管道，且是一个只读管道  实现方式：将形参提供的信息组合成pending结构体输入到udp的addpending管道中  只有当closing管道中有数据(socket被关闭)时返回的只读管道有数据，否则读取返回的只读管道时会被阻塞(ping / waitping / findnode方法都读取了此管道，都会被阻塞？) |

|  |
| --- |
| func (t \*udp) handleReply(from NodeID, ptype byte, req packet) bool  此方法在6中各类具体的handle方法中被调用，也就是当节点收到其他节点消息时会被调用  根据形参构建reply结构体，然后将该reply结构体输入到udp结构体的gotreply管道中。  然后等待：若match管道先有数据则返回该管道内容(loop方法中会向match管道填充数据) |

|  |
| --- |
| func (t \*udp) loop() 重点需理解   1. 每隔1h 管道refresh.C会产生信号，此时会新开协程调用udp结构体的refresh()方法。对整个Kad路由表进行更新 2. 每当udp的closing管道中产生信号(udp结构体对于的socket被关闭)，遍历整个pengding切片所有剩余挂起事件。向其errc管道写入错误消息。再清空pending后退出此方法 3. 取出udp结构体挂起等待处理的事件，设置处理截止时长并加入到本地挂起队列 4. 收到远端节点的回复消息。在本地挂起队列中查找于此reply对应的挂起事件，若能成功找到则将此事件从挂起队列中删除 5. 发生超时事件。遍历所有超时事件，将其从挂起队列中删除；同时刷新计时器 |

|  |
| --- |
| func (t \*udp) send(toaddr \*net.UDPAddr, ptype byte, req interface{})  将消息req加密编码打包后发送给指定的远端节点  加密编码通过函数encodePacket()实现 |

|  |
| --- |
| func (t \*udp) readLoop()  有一条单独的协程负责运行此方法。负责不断从UDP通信的socket上读取数据(编码加密后的packet)  收到packet之后，由udp结构体的handlePacket方法负责对packet进行解码 |

|  |
| --- |
| func (t \*udp) handlePacket(from \*net.UDPAddr, buf []byte)  负责对接收到的udp packet数据包进行解码 (由函数decodePacket()实现)  解码之后，将解码后的数据包交由packet的handle方法处理(decodePacket()在进行解码时，可以获取到packet的不同类型：pingPacket、pongPacket、findnodePacket、neighborsPacket，对于四种RPC操作。Handle方法(接口)针对这四种类型采取不同的处理措施) |

1. 对handle方法的实现(四类RPC操作各自实现)
   1. 收到对方Ping消息

|  |
| --- |
| func (req \*ping) handle(t \*udp, from \*net.UDPAddr, fromID NodeID, mac []byte)   1. 调用udp的send方法，向指定的远端节点发送pong回复 2. 调用udp的handleReply方法（类型为ping）在gotreply管道中输入信号(应该是没有完成任何挂起事件)。接着，若本节点与对方节点尚未建立bond则会返回false，此时用一条新的协程与对方节点建立bond |

* 1. 收到对方Pong消息

|  |
| --- |
| func (req \*pong) handle(t \*udp, from \*net.UDPAddr, fromID NodeID, mac []byte) error  调用udp的handleReply方法（类型为pong）在gotreply管道中输入信号。正常情况下正好可以完成对应pending挂起队列中的挂起的pongpacket事件；否则报错(errUnsolicitedReply) |

* 1. 收到对方findnode消息

|  |
| --- |
| func (req \*findnode) handle(t \*udp, from \*net.UDPAddr, fromID NodeID, mac []byte)   1. 首先需要检查此消息的源节点是否在本地数据库中，如果不在说明没有建立bond，那么将不会对此数据包进行处理(是为了防止DDos攻击) 2. 从req中提取目标节点ID并求哈希 3. 调用udp的closest方法找出本节点的Kad路由表中距离目标节点最近的bucketSize个节点 4. 调用udp的send方法将这bucketSize个最近节点回复给源节点(neighbors消息) |

* 1. 收到对方neighbors消息

|  |
| --- |
| func (req \*neighbors) handle(t \*udp, from \*net.UDPAddr, fromID NodeID, mac []byte) error  调用udp的handleReply方法（类型为neighbors）在gotreply管道中输入信号。正常情况下正好可以完成对应pending挂起队列中的挂起的neighborspacket事件；否则报错(errUnsolicitedReply) |