白话文P2P模块源码解析

一、创建P2P server 对象

文件在 /node/node.go --start()

首先我们要知道启动P2的命令为：
geth --data1 -port 30306 console init genesis

```
func (n *Node) Start() error {
    ...

    // Initialize the p2p server. This creates the node key and
    // discovery databases.
    n.serverConfig = n.config.P2P
    n.serverConfig.PrivateKey = n.config.NodeKey()
    n.serverConfig.Name = n.config.NodeName()
    n.serverConfig.Logger = n.log
    if n.serverConfig.StaticNodes == nil {
        n.serverConfig.StaticNodes = n.config.StaticNodes()
    }
    if n.serverConfig.TrustedNodes == nil {
        n.serverConfig.TrustedNodes = n.config.TrustedNodes()
    }
    if n.serverConfig.NodeDatabase == "" {
        n.serverConfig.NodeDatabase = n.config.NodeDB()
    }
    running := &p2p.Server{Config: n.serverConfig}
    n.log.Info("Starting peer-to-peer node", "instance", n.serverConfig.Name)
    ....
}
```

首先通过ServerConfig类型设置参数，serverConfig.PrivateKey = n.config.NodeKey()为比较重要的私钥部分，接下来通过if判断，继续设置serverconfig的参数，最后边的一行代码为创建p2pServer,下面我们看一下创建service服务的过程

二、创建Service

文件在 /node/node.go

```
// Otherwise copy and specialize the P2P configuration
//拷贝并且初始化p2p的配置文件
    services := make(map[reflect.Type]Service)
    for _, constructor := range n.serviceFuncs {
        // Create a new context for the particular service
        ctx := &ServiceContext{
            config:         n.config,
            services:       make(map[reflect.Type]Service),
            EventMux:       n.eventmux,
            AccountManager: n.accman,
        }
        for kind, s := range services { // copy needed for threaded access
            ctx.services[kind] = s
        }
        // Construct and save the service
        service, err := constructor(ctx)
        if err != nil {
            return err
        }
        kind := reflect.TypeOf(service)
        if _, exists := services[kind]; exists {
            return &DuplicateServiceError{Kind: kind}
        }
        services[kind] = service
    }
```

我们可以看出，此段代码（services[kind] = service）最后将创建service存放到了service数组中，下面是启动p2pServer

三、启动P2P server

```
// Gather the protocols and start the freshly assembled P2P server
    for _, service := range services {
        running.Protocols = append(running.Protocols, service.Protocols()...)
    }
    //这里为p2p start
    if err := running.Start(); err != nil {
        return convertFileLockError(err)
    }
```

上面代码为把所有Service支持的协议放到一个数组中保存，然后调用p2p.Server的Start()方法启动代码位于p2p/server.go文件中的P2P server函数。P2P server会绑定两个端口，一个TCP另外一个为UDP协议，端口号默认都是30303。UDP端口主要用于通过Kad算法那实现结点发现，

TCP端口主要用于业务数据传输，基于RLPx加密传输协议。具体来说，Start()方法做了以下几件事情：
1：侦听UDP端口：用于结点发现
2：发起UDP请求获取结点表：内部会启动goroutine来完成
3：侦听TCP端口：用于业务数据传输，基于RLPx协议
4：发起TCP请求连接到其他结点：也是启动goroutine完成。

文件在 p2p/server.go

```
// Servers can not be re-used after stopping.
func (srv *Server) Start() (err error) {
    srv.lock.Lock()
    defer srv.lock.Unlock()
    if srv.running {
        return errors.New("server already running")
    }
    srv.running = true
    srv.log = srv.Config.Logger
    if srv.log == nil {
        srv.log = log.New()
    }
    srv.log.Info("Starting P2P networking")

// static fields
if srv.PrivateKey == nil {
    return fmt.Errorf("Server.PrivateKey must be set to a non-nil key")
}
if srv.newTransport == nil {
    srv.newTransport = newRLPX
}
if srv.Dialer == nil {
    srv.Dialer = TCPDialer{&net.Dialer{Timeout: defaultDialTimeout}}
}
srv.quit = make(chan struct{})
srv.addpeer = make(chan *conn)
srv.delpeer = make(chan peerDrop)
srv.posthandshake = make(chan *conn)
srv.addstatic = make(chan *discover.Node)
srv.removestatic = make(chan *discover.Node)
srv.peerOp = make(chan peerOpFunc)
srv.peerOpDone = make(chan struct{})

var (
    conn      *net.UDPConn
    sconn     *sharedUDPConn
    realaddr  *net.UDPAddr
```

```
	unhandled chan discover.ReadPacket
)

if !srv.NoDiscovery || srv.DiscoveryV5 {
	addr, err := net.ResolveUDPAddr("udp", srv.ListenAddr)
	if err != nil {
		return err
	}
	conn, err = net.ListenUDP("udp", addr)
	if err != nil {
		return err
	}
	realaddr = conn.LocalAddr().(*net.UDPAddr)
	if srv.NAT != nil {
		if !realaddr.IP.IsLoopback() {
			go nat.Map(srv.NAT, srv.quit, "udp", realaddr.Port, realaddr.Port, "eth
ereum discovery")
		}
		// TODO: react to external IP changes over time.
		if ext, err := srv.NAT.ExternalIP(); err == nil {
			realaddr = &net.UDPAddr{IP: ext, Port: realaddr.Port}
		}
	}
}

if !srv.NoDiscovery && srv.DiscoveryV5 {
	unhandled = make(chan discover.ReadPacket, 100)
	sconn = &sharedUDPConn{conn, unhandled}
}

// node table
if !srv.NoDiscovery {
	cfg := discover.Config{
		PrivateKey:   srv.PrivateKey,
		AnnounceAddr: realaddr,
		NodeDBPath:   srv.NodeDatabase,
		NetRestrict:  srv.NetRestrict,
		Bootnodes:    srv.BootstrapNodes,
		Unhandled:    unhandled,
	}
	ntab, err := discover.ListenUDP(conn, cfg)
	if err != nil {
		return err
	}
	srv.ntab = ntab
}

if srv.DiscoveryV5 {
    var (
        ntab *discv5.Network
        err  error
    )
    if sconn != nil {
        ntab, err = discv5.ListenUDP(srv.PrivateKey, sconn, realaddr, "", srv.NetRe
strict) //srv.NodeDatabase)
    } else {
        ntab, err = discv5.ListenUDP(srv.PrivateKey, conn, realaddr, "", srv.NetRes
trict) //srv.NodeDatabase)
    }
    if err != nil {
        return err
    }
    if err := ntab.SetFallbackNodes(srv.BootstrapNodesV5); err != nil {
        return err
    }
    srv.DiscV5 = ntab
}

dynPeers := srv.maxDialedConns()
dialer := newDialState(srv.StaticNodes, srv.BootstrapNodes, srv.ntab, dynPeers, srv
.NetRestrict)

// handshake
srv.ourHandshake = &protoHandshake{Version: baseProtocolVersion, Name: srv.Name, ID
: discover.PubkeyID(&srv.PrivateKey.PublicKey)}
for _, p := range srv.Protocols {
    srv.ourHandshake.Caps = append(srv.ourHandshake.Caps, p.cap())
}
// listen/dial
if srv.ListenAddr != "" {
    if err := srv.startListening(); err != nil {
        return err
    }
}
if srv.NoDial && srv.ListenAddr == "" {
    srv.log.Warn("P2P server will be useless, neither dialing nor listening")
}

srv.loopWG.Add(1)
go srv.run(dialer)
srv.running = true
return nil
```

}

四、启动Service

```
// Start each of the services
	started := []reflect.Type{}
	for kind, service := range services {
		// Start the next service, stopping all previous upon failure
		if err := service.Start(running); err != nil {
			for _, kind := range started {
				services[kind].Stop()
			}
			running.Stop()

			return err
		}
		// Mark the service started for potential cleanup
		started = append(started, kind)
	}
```

主要就是依次调用每个Service的Start()方法，然后把启动的Service的类型存储到started表中。