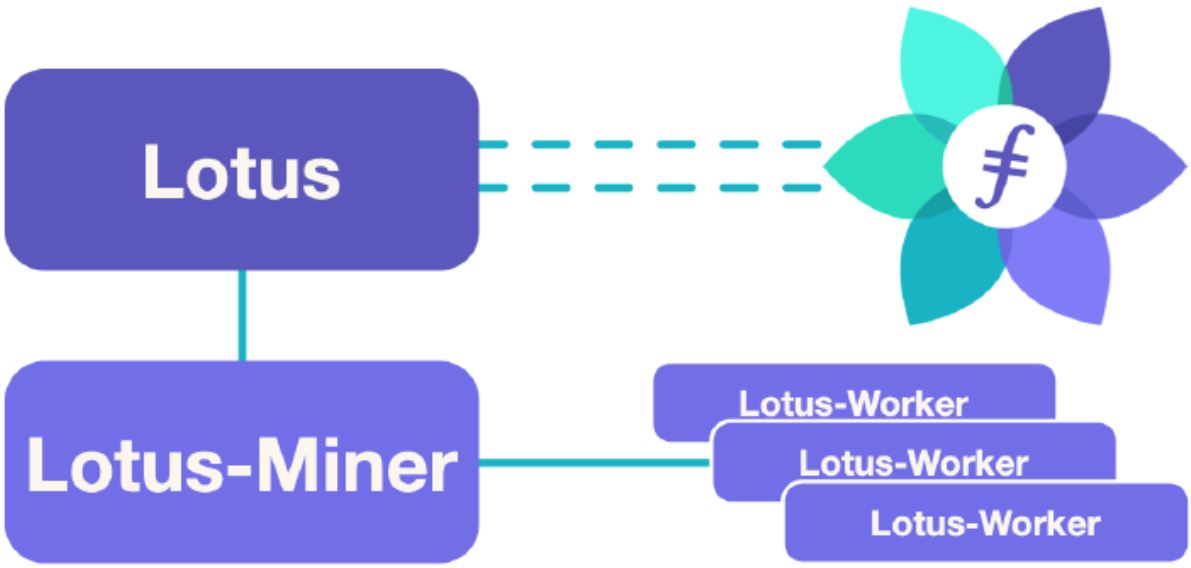
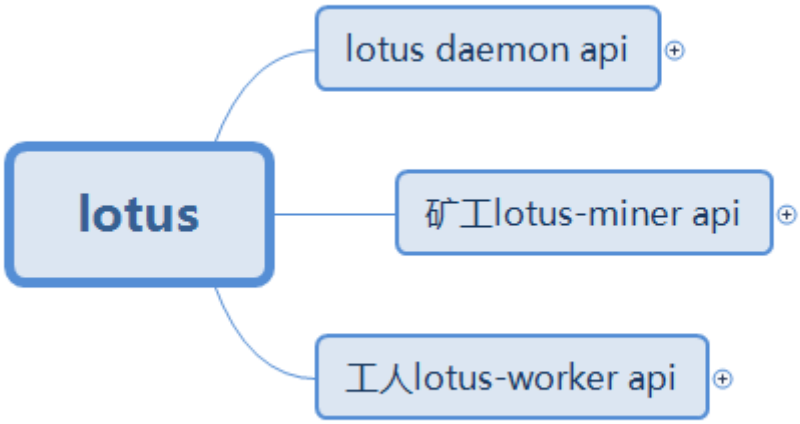
lotus架构与流程

1. lotus简介



> Filecoin是一个分布式存储网络，其也是IPFS(星际文件系统)的一种奖励机制，刺激更多的人贡献带宽与存储空间，从而提供去中心化的分布式存储服务，主要结构包括：filecoin区块链与原生加密货币（FIL）；
>
> lotus是一个与filecoin区块链进行交互的命令行程序，用户可以通过它进行文件的存储与下载，或将自己多余的空间进行出租给其他用户，以及检查计算机存储数据是否正常；
>
> lotus也是filecoin网络中的一个节点，该节点连接进入filecoin网络并与该网络中的其他节点进行通信，如上图所示。

1.1 lotus组成



`lotus`由lotus、lotus-miner和lotus-worker服务组成，其具体功能如下：

- lotus节点，负责链消息同步、钱包管理、交易等，默认启动端口1234，其实现结构大致如下图：

- lotus矿工，负责任务分发、管理、扇区存储等，默认启动端口2345，其实现接口大致如下图：

- lotus工人，封装扇区，worker分为两种，一种是做addpiece和precommit1，另外一种做precommit2以及后面的工作，具体说明看任务步骤，默认启动端口3456，其实现接口大致如下图：

`lotus`服务间的链接关系：

- lotus-miner连接lotus 节点的API（在lotus节点启动之后才能服务运行）
- lotus-worker连接lotus-miner的API（在lotus-miner服务启动之后才能进行服务运行）

1.2 filecoin网络

`filecoin`网络类别，一个主网三个分支（默认主网）：

- 主网（Mainnet），唯一的生产 Filecoin 网络，分支`master`；
- 测试网（Calibration），用于通过测试、基准测试和优化有意义的规模评估Filecoin，分支`ntwk-calibration`；
- 测试网（Hyperspace），面向开发人员的 Filecoin 测试网络；
- 测试网（Devnet），本地开发网络，旨在以最少的资源运行以进行开发测试。

1.3 lotus节点类型

- 全节点（Full node），需要 Lotus 全节点才能与 Filecoin 区块链交互；
- 完整历史节点（Full historical node），一个完整的历史节点需要重创世块进行同步；
- lite节点（Lite client node），Lite 节点是 Lotus 全节点的缩小版。这些节点需要连接到 Lotus 全节点才能运行。一旦连接到 Lotus 全节点，这些 Lotus Lite 客户端节点只能执行消息签名和处理事务。

2. 安装lotus

2.1 安装系统依赖

Ubuntu/Debian：

```
sudo apt install mesa-opencl-icd ocl-icd-opencl-dev gcc git bzr jq pkg-config
curl clang build-essential hwloc libhwloc-dev wget -y && sudo apt upgrade -y
```

Fedora:

```
sudo dnf -y install gcc make git bzr jq pkgconfig mesa-libOpenCL mesa-libOpenCL-devel opencl-headers ocl-icd ocl-icd-devel clang llvm wget hwloc hwloc-devel
```

2.2 安装go环境

```
## go安装包
wget -c https://golang.org/dl/go1.18.8.linux-amd64.tar.gz -O - | sudo tar -xz -C /usr/local
## 配置go环境变量
echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bashrc && source ~/.bashrc
go version
```

2.3 clone lotus

```
git clone https://github.com/filecoin-project/lotus.git
cd lotus/
```

2.4 env配置（仅CN）

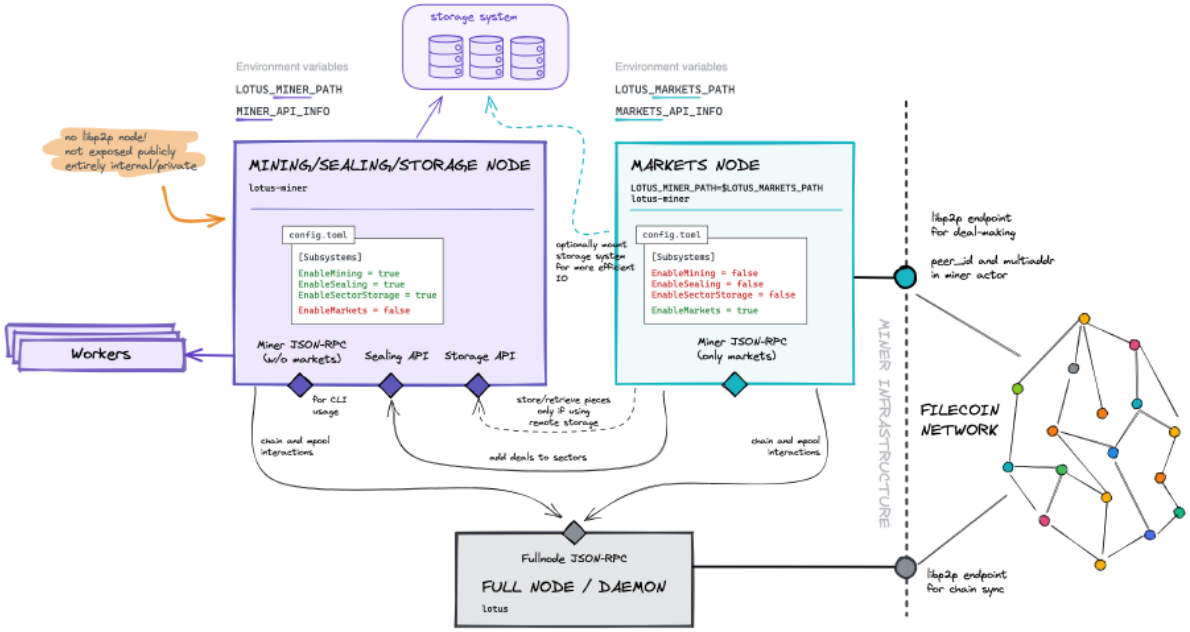
```
export IPFS_GATEWAY=https://proof-parameters.s3.cn-south-1.jdcloud-oss.com/ipfs/
export GOPROXY=https://goproxy.cn
```

2.5 构建并安装lotus

```
## 安装
make clean all
## 全局安装
sudo make install
## 查看是否安装成功
lotus --version
```

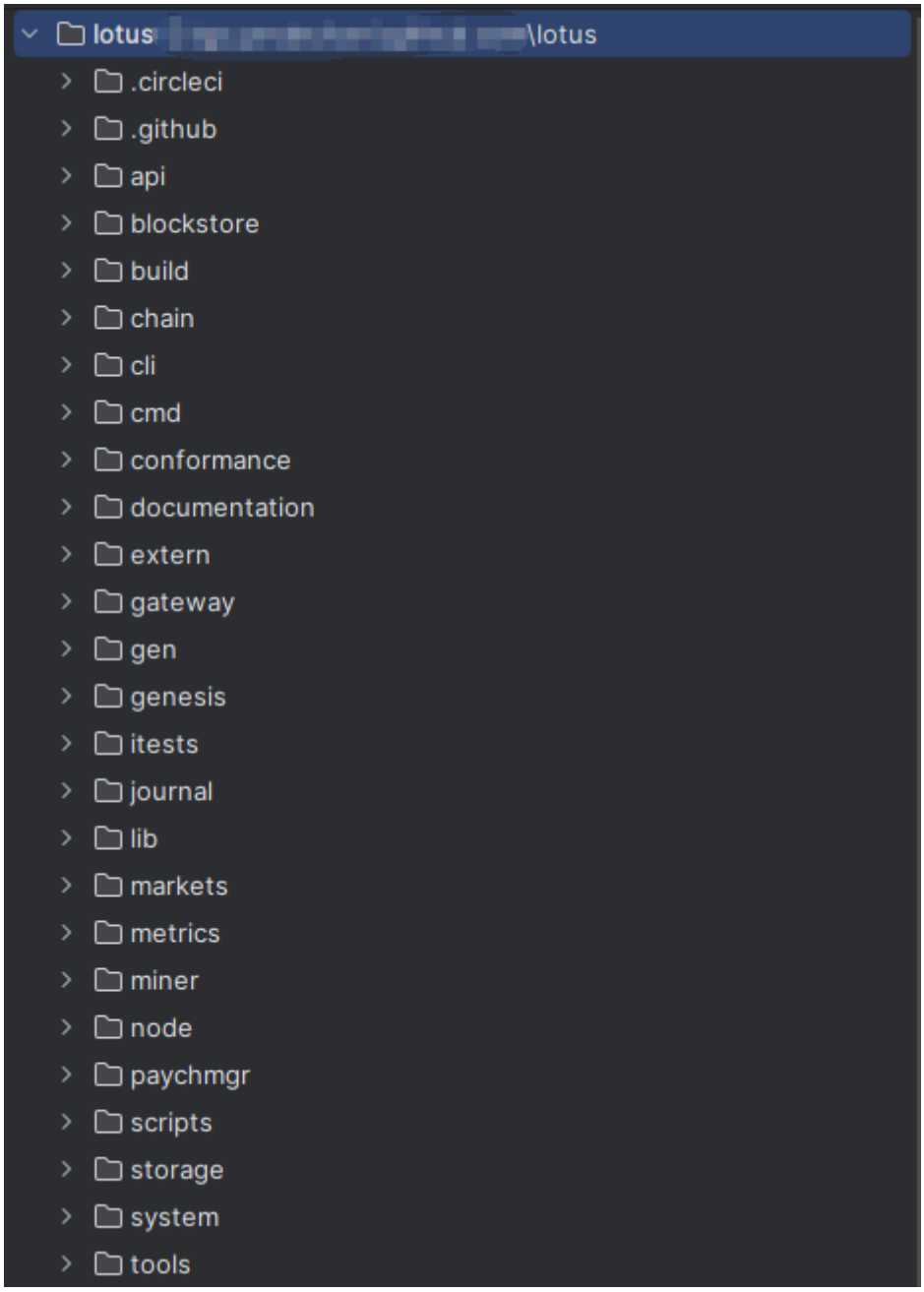
3. lotus架构

3.1 分割市场架构(1.31)



> 个人理解：full node/daemon主节点，接入Filecoin网络，进行与filecoin区块链的链消息的同步等操作；markets node 与 mining/sealing/storage node之间通过api进行交互调用，mining节点、markets节点通过rpc协议与full node/daemon主节点进行数据的传输，其中mining node在workers的协助下进行datastorage；当Filenode中有users需要存储空间，则会通过p2p的模式找到链中最符合的markets nodes，进行存储信息的询问，users满意之后，markets nodes将会调用mining的api进行存储调用；如图虚线左边的api或者rpc接口都是不暴露的，该既接入Filecoin网络的node只暴露libp2p接口。这是对mining进一步的分化，让交易与存储分开。

3.2 源码目录分析



3.2.1 api

该目下主要存放 API 相关程序，Lotus 是一个大型的区块链项目，模块之间基本都通过 API 来通信，远程 API 的调用走的是 JSONRPC 调用模式，比如 Miner 上链要调用 Lotus Chain 的 API，Worker 和 Miner 之间通信也是相互调用 API。

本目录抽象了节点定义，定义了若干go interface，如 Common（定义节点通用功能）、FullNode （定义一个全节点的行为，继承自Common）、StorageMiner（存储矿工，也从Common继承）和相关的函数。 这里有几个重要的接口：

1. **Common：** 定义了一些节点通用的功能，在 `api_common.go` 中定义。
2. **FullNode:** 定义一个全节点的所有 API，继承自 `Common`，在 `api_full.go` 中定义。
3. **StorageMiner:** 定义一个存储矿工的行为，继承自 `Common`，在 `api_storage.go` 中定义。

对应于上面几种主要接口，也提供了几个struct，分别为CommonStruct，FullNodeStruct，StorageMinerStruct作为实现，这些实现使用了代理模式，只是简单地将请求转发给各自的Internal成员，具体的功能函数需要使用者提供。

3.2.2 blockstore

区块存储工具包，包括区块的新增，删除，以及同步相关的 API，这里做了几种实现，有 leveldb 版本，内存（`mem.go`）版本以及 IPFS 版本的实现。

3.2.3 build

定义用于构建节点，创建网络相关功能，包括但不限于定义网络相关配置(如 params_main.go)，定义创世节点相关信息(genesis目录)， 以及创世节点的连接信息(bootstrap目录)，复制证明参数下载地址配置等。

3.2.4 chain

顾名思义，就是实现了 Louts 链相关的功能，主要包含了如下子模块：

- types: 定义 Louts 链中的各种数据结构。
- store: 公链存储相关，处理所有的本地链状态，包括链头、消息和状态等。
- messagesinger: 消息签名工具包。
- messagepool：消息池，定义消息打包规则。
- wallet：实现钱包相关工具。
- state: 处理 Filecoin 的状态树，内部包装了HAMT。
- actors: 账户体系，定义了各种 actor。
- vm: 智能合约虚拟机，这里实现了actor 的方法调用工具包。

3.2.5 cli

Lotus命令行工具的实现，依赖于包gopkg.in/urfave/cli.v2，里面的go文件名基本上与Lotus的子命令保持一致，比如：

- **state.go:** 对应 `lotus state` 命令
- **sync.go:** 对应 `lotus sync` 命令
- **wallet.go:** 对应 `lotus wallet` 命令
- **chain.go:**对应`lotus chain`命令
- ...

- 对应于每条子命令及子命令的子命令，都定义了一个Command对象

- 相应的Command定义在文件chain.go中：

```
var ChainCmd = &cli.Command{
    Name:  "chain",
    Usage: "Interact with filecoin blockchain",
    Subcommands: []*cli.Command{
        ChainHeadCmd,
        ChainGetBlock,
        ChainReadObjCmd,
        ChainDeleteObjCmd,
        ChainStatObjCmd,
        ChainGetMsgCmd,
        ChainSetHeadCmd,
        ChainListCmd,
        ChainGetCmd,
        ChainBisectCmd,
        ChainExportCmd,
        ChainExportRangeCmd,
        SlashConsensusFault,
```

```
        ChainGasPriceCmd,
        ChainInspectUsage,
        ChainDecodeCmd,
        ChainEncodeCmd,
        ChainDisputeSetCmd,
        ChainPruneCmd,
    },
}
```

3.2.6 cmd

内含各种不同的命令行项目，Lotus将系统分为不同的进程模块，为每个模块定义一个项目。

模块名称	模块说明
lotus	lotus 守护进程项目，负责与 lotus 公链通信，同步区块，是主要核心进程之一
lotus-miner	存储矿工的核心进程，负责时空证明，爆块，任务调度，是主要核心进程之一
lotus-seal-worker	数据密封主进程，整个复制证明工作由该进程完成，是主要核心进程之一
lotus-bench	基准测试工具项目，如果你要做性能优化，那么这个工具应该要经常使用
lotus-seed	用来生成创世区块的工具，构建网络不可或缺
lotus-shed	非常有用的一个工具包命令集合，里面又各种黑科技命令，比如终止扇区，修复消息池，多重签名工具等。

每个模块都可以单独编译，编译后的程序都可以单独启动一个进程独立运行:

```
# 编译 lotus 程序
make lotus
# 编译 miner 程序
make lotus-miner
# 编译 worker 程序
make lotus-worker
# 编译 bench 测试工具
make lotus-bench
# 编译钱包工具
make lotus-wallet
```

3.2.7 conformance

数据缓冲的迁移

3.2.8 documentation

Lotus的文档目录

3.2.9 extern

filecoin-ffi目录

FFI rust 调用封装，Lotus 的源码分两部分，公链和任务调度这块是 Golang 实现的，另外一些底层计算都是通过调用 Rust 底层库去计算的。比如 PC1，PC2，C1，C2，时空证明等都是用 Rust 实现的。Lotus 通过 filecoin-ffi 这个组件来调用 Rust 底层库

sector-storage目录

这里包含了任务调度，资源分配的的核心代码。属于整个项目比较核心的部分。

- sealtasks:

  定义了一些任务类别常量，如:

  ```
  const (
      TTAddPiece   TaskType = "seal/v0/addpiece"
      TTPreCommit1 TaskType = "seal/v0/precommit/1"
      TTPreCommit2 TaskType = "seal/v0/precommit/2"
      TTCommit1    TaskType = "seal/v0/commit/1" // NOTE: We use this to
  transfer the sector into miner-local storage for now; Don't use on workers!
      TTCommit2    TaskType = "seal/v0/commit/2"
      TTFinalize TaskType = "seal/v0/finalize"
      TTFetch TaskType = "seal/v0/fetch"
  )
  ```


- **stores:** 扇区存储相关代码，实现了扇区检索(`index.go`), 本地存储(`local.go`)，远程存储(`remote.go`)等
- **storiface:** 定义远程调用相关 API
- **worker:** worker 相关功能和属性定义
- **manager.go:** 全局管理对象，调度入口程序，它管理着 Worker，存储，以及调度等对象
- **sched.go:** 任务调度的具体实现

storage-sealing目录

这也是 Lotus 的核心模块之一，它包含如下代码:

- 扇区状态机实现(`fsm.go`)
- 扇区的各种事件定义(`event.go`)
- 扇区聚合提交上链(`precommit_batch.go`, `commit_batch.go`)
- sealing 相关的 API 实现

3.2.10 gateway

IPFFS网关

3.2.11 gen

API 代理对象的生成工具

3.2.12 genesis

定义创世节点相关信息

3.2.13 itests--

3.2.14 journal

journal 日志工具

3.2.15 lib

- 实现lotus项目各模块公用的函数
    - crypto：实现数据加密，公钥生成，签名与签名验证等
    - jsonrpc：实现了一个基于json传输数据的rpc包，包括服务端和客户端，可为其它项目提供完整的rpc功能支持
    - statestore：包装github.com/ipfs/go-datastore，实现状态追踪
    - sectorbuilder：实现扇区管理
    - bufstore:：包装github.com/ipfs/go-ipfs-blockstore，集成了Blockstore的读写功能
    - cborutil：包装github.com/ipfs/go-ipld-cbor，提供操作cbor数据的简便方法
    - auth： 实现权限认证服务HTTP接口

3.2.16 markets

- Filecoin 订单市场相关实现，包括订单的过滤，订单价格，以及存储订单和检索订单的适配器等。

3.2.17 metrics

3.2.18 miner

- 定义产出区块逻辑，与全节点通过API进行交互

3.2.19 node

定义了 lotus 节点相关的 struct 和 interface 等，各主要子目录如下：

- **config:** 定义节点相关配置结构体
- **hello:** 实现 hello 协议
- **modules:** 定义实现节点功能的各种函数，比较重要的有 `chain.go`, `storageminer.go` 等。
- **impl:** 节点各种 API 的实现，比如 `full.go` 实现全节点的相关 API，`storminer.go` 实现存储节点相关 API 等。
- **repo:** 链上数据在本地的存储仓库，与本地文件系统打交道。

3.2.20 paychmgr

各种支付凭证管理，定义了一些支付通道的 API。

3.2.21 scripts

各种运行脚本，用于布署节点和矿工等，也包括一些用于启动的配置文件

3.2.22 storage

定义存储矿工逻辑，用于实现"lotus-storage-miner"，时空证明的主程序逻辑也是在这里实现的。

3.2.23 system--

3.2.24 tools

各种打包工具封装，如使用 docker 部署 Lotus 等。

3.3 源码执行过程分析

以lotus节点服务为例

```
var log = logging.Logger("main") // name="main"的日志记录

var AdvanceBlockCmd *cli.Command // 预先初始化区块链命令

func main() {
    api.RunningNodeType = api.NodeFull // 赋予RunningNodeType类型，1为lotus，2为
miner，3为worker

    lotuslog.SetupLogLevels() // 日志初始化设置

    local := []*cli.Command{ // 初始化cli.Command结构体，该结构体是
github.com\urfave\cli\v2@v2.16.3 golang命令行库中的结构体
        DaemonCmd, // go-lotus daemon命令
        backupCmd, // 备份命令
        configCmd, // 配置命令
    }
    if AdvanceBlockCmd != nil {
        local = append(local, AdvanceBlockCmd) // 初始化AdvanceBlockCmd失败，重新添
加
    }

    jaeger := tracing.SetupJaegerTracing("lotus") // 使用Uber的分布式追踪系统
    defer func() {
        if jaeger != nil {
            _ = jaeger.ForceFlush(context.Background())// 最后关闭jaeger
        }
    }()

    for _, cmd := range local { // local是一个结构体指针，循环遍历其中的命令
        cmd := cmd
        originBefore := cmd.Before // 记录该条命令输入之前的状态，是否存在错误
        cmd.Before = func(cctx *cli.Context) error {
            if jaeger != nil {
                _ = jaeger.Shutdown(cctx.Context)
            }
            // jaeger记录cmd信息
            jaeger = tracing.SetupJaegerTracing("lotus/" + cmd.Name)
```

```
            if cctx.IsSet("color") {
                color.NoColor = !cctx.Bool("color")
            }
            // 如果执行之前就存在错误,不执行当前命令
            if originBefore != nil {
                return originBefore(cctx)
            }
            return nil
        }
    }
    // 创建一个新的span，在当前ctx下
    ctx, span := trace.StartSpan(context.Background(), "/cli")
    defer span.End()

    interactiveDef := isatty.IsTerminal(os.Stdout.Fd()) ||
isatty.IsCygwinTerminal(os.Stdout.Fd())
    // 初始化github.com\urfave\cli\v2@v2.16.3中的App结构体
    app := &cli.App{
        Name:                 "lotus",
        Usage:                "Filecoin decentralized storage network client",
        Version:              build.UserVersion(),
        EnableBashCompletion: true,
        Flags: []cli.Flag{ // Flags 用于设置参数
            &cli.StringFlag{
                Name:    "panic-reports",
                EnvVars: []string{"LOTUS_PANIC_REPORT_PATH"},
                Hidden:  true,
                Value:   "~/.lotus", // should follow --repo default
            },
            &cli.BoolFlag{
                // examined in the Before above
                Name:        "color",
                Usage:       "use color in display output",
                DefaultText: "depends on output being a TTY",
            },
            &cli.StringFlag{
                Name:    "repo",
                EnvVars: []string{"LOTUS_PATH"},
                Hidden:  true,
                Value:   "~/.lotus", // TODO: Consider XDG_DATA_HOME
            },
            &cli.BoolFlag{
                Name:  "interactive",
                Usage: "setting to false will disable interactive functionality
of commands",
                Value: interactiveDef,
            },
            &cli.BoolFlag{
                Name:  "force-send",
                Usage: "if true, will ignore pre-send checks",
            },
            cliutil.FlagVeryVerbose,
        },
        // Action/after 对应的函数就是你具体对各个参数具体的处理逻辑。
        After: func(c *cli.Context) error {
```

```
            if r := recover(); r != nil {
                // Generate report in LOTUS_PATH and re-raise panic
                build.GeneratePanicReport(c.String("panic-reports"),
c.String("repo"), c.App.Name)
                panic(r)
            }
            return nil
        },

        Commands: append(local, lcli.Commands...),
    }
    // 安装程序，初始化代码，做好准备工作
    app.Setup()
    app.Metadata["traceContext"] = ctx        // 设置类别
    app.Metadata["repoType"] = repo.FullNode // 存储类别

    lcli.RunApp(app) //程序run起来，并解析传入的参数
}
```

lotus-miner,lotus-worker服务执行过程与lotus大致相似。

> 执行总结：利用github.com\urfave\cli\v2@v2.16.3包中app结构体的初始化，再利用其结构体的Run方法，进行服务的启动；再利用Uber公式的jaeger分布式追踪系统，进行命令的追踪，当有命令以lotus开头，则连入该节点，进行该区块链的操作。

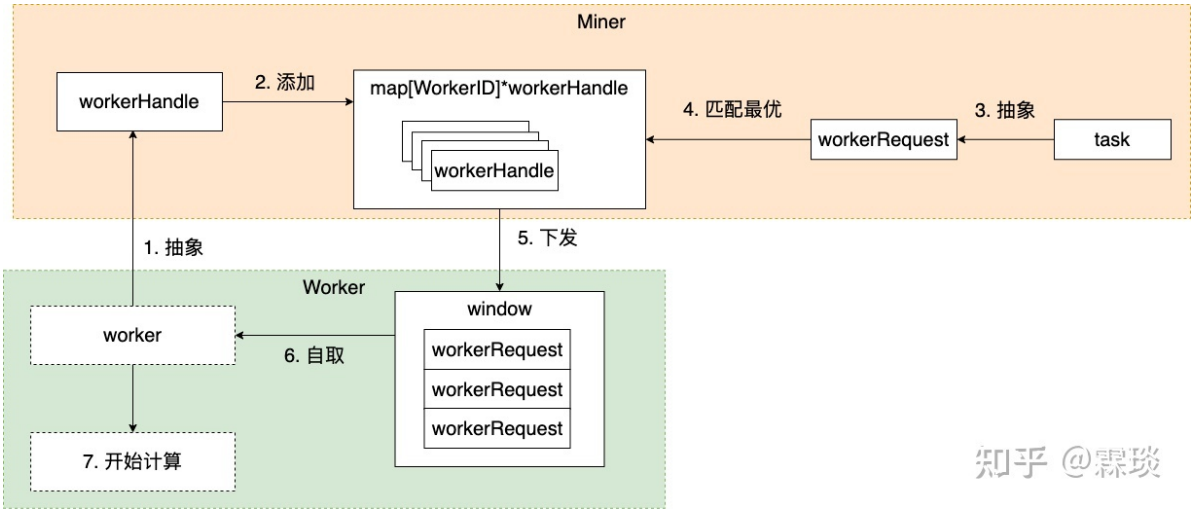
4. lotus任务调度流程与消息运转流程

4.1 lotus任务调度

4.1.1 lotus任务调度要解决的主要问题

- 问题一：调度任务复杂。lotus 的调度任务实现得比较复杂，使用了十几个数据结构，代码也分布在数十个文件中，做这么多工作，当然有它的原因，它想要解决的主要问题包括：实现**任务绑定配置化**。通过启动参数，可以灵活配置 worker 执行的任务种类，方便矿工进行灵活的集群架构调整，并实现集群机器数量的伸缩性;
- 问题二：硬件资源分配不均。不同配置的 worker 能够并行执行的任务数量也不相同，超过硬件承受能力的任务分配会导致硬件资源争用，进而导致任务执行时间直线下降。同时要均匀分配负载，不能让忙的忙死，闲的闲死;
- 问题三：需兼容各种异常情况。例如当发生 worker 或 miner 突然掉线、网络不畅、存储空间不足等情况时，要能够保证异常状态的任务不会陷入死循环中，能够在合适的时间重启或继续，并且不影响正常状态任务的执行。

4.1.2 调度核心流程



1. worker 启动后，会通过 RPC 连接 miner，miner 收到并通过 JWT 验证后，会将其抽象为一个 workerHandle 对象，主要记录 worker 的基本信息，如资源使用情况、当前任务情况等;
2. workerHandle 对象会添加到名为 worker 的 map 中，用于后续任务调度时匹配;
3. 新增的 task 会抽象为 workerRequest，并添加到 miner 的任务队列中;
4. 这一步是调度的核心。将所有待调度的 task 依次寻找最优的 workerHandle，目前的实现主要参考任务优先级、worker 空闲情况、worker 每一个任务分配时需要预留的资源;
5. 匹配好 worker 的 task 下发到对应的 worker 上，worker 同时维护多个任务 window（类似 queue），用于保存及调度所有正在运行及待运行的任务;
6. worker 定期会到 window 中取出任务进行执行;
7. 执行任务计算。

4.1.3 任务调度基本数据结构

核心数据结构包括：

- scheduler：调度器，负责在 miner 上统筹调度所有task及worker。核心调度方法是 trySched（）
- workerHandle： worker 的抽象，记录 worker 基本信息、当前资源使用、worker 的任务窗口 等
- workerRequest： task 的抽象，记录任务类型、优先级以及任务的具体执行内容等
- activeResources： 正在使用的资源
- workerResources： worker 总的资源

4.1.4 lotus任务调度总结

lotus 调度任务的整体逻辑基本还是遵循生产者-消费者模型，通过对 worker、task 的抽象，结合预配置、资源情况、优先级等因素进行调度，并通过一个有限状态机控制状态流转。

4.2 lotus消息运转

4.2.1 节点自身发送消息

各个节点都会有一个消息池`MessagePool`，消息一般会被先发送到消息池中，再经由**P2P**网络发送到相邻节点。以下方法可以向消息池中添加消息：`MessagePool.Add`，`MessagePool.Push`，`MessagePool.PushWithNonce`。 新建`MessagePool`会启动一个**go**协程，用以发送消息到**P2P**网络。

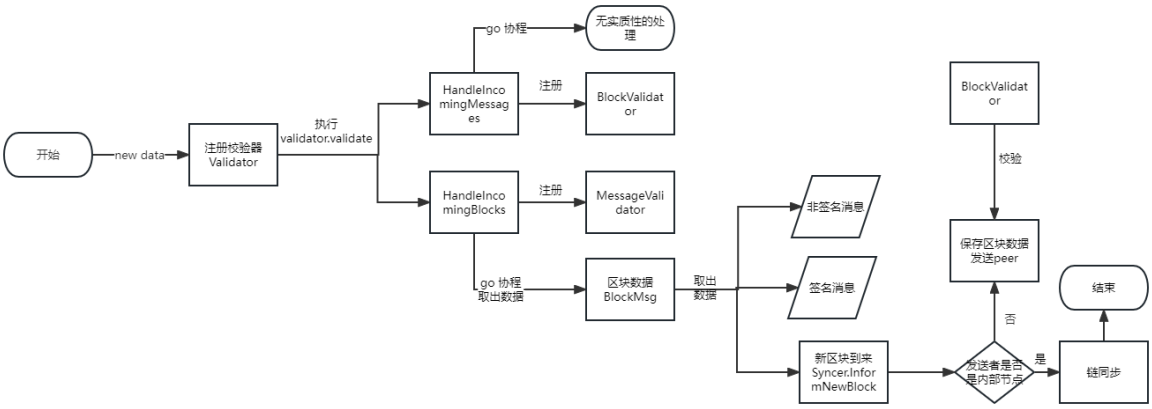
消息池会监控链头的变化(`Provider.SubscribeHeadChanges`)，并在内部动态增删消息(`MessagePool.HeadChange`)。

4.2.2 同步网络中消息

Filecoin网络中的节点都会监控该网络中的消息，目前有两种监控模式：

- 监控“到来区块”：由函数`HandleIncomingBlocks`完成
- 监控“到来消息”：由函数`HandleIncomingMessages`完成

监控消息具体流程图：



4.2.3 打包区块消息

矿工在打包(`createBlock`)区块时，会从消息池中提取消息(`SelectMessages`)进行打包，之后调用**RPC**函数`MinerCreateBlock`以创建一个新的区块，后者会调用`StateManager.TipSetState`计算状态树，并执行区块内所包含的全部消息。

4.2.4 消息执行

消息最终都是要被执行的，这样各个节点才能最终维护相同的状态。 消息执行通过虚拟机（**chain/vm.VM**）完成。 每条消息都包含接收者**To**、方法编号**Method**、执行方法所需的参数**Params**，**Filecoin**系统中的方法都是依附在各种角色**Actor**上的，**lotus**代码中定义了多种**Actor**及其所导出的方法。

虚拟机对各种**Actor**导出方法的签名有如下要求：

- 有两个输入参数
- 第一个输入参数为`specs-actors/actors/runtime.RunTime`类型
- 第二个输入参数为指针类型
- 有一个输出参数
- 输出参数需实现接口`whyrusleeping/cbor-gen.CBORMarshaler`

虚拟机执行**Actor**的方法过程中，若方法`panic`，虚拟机会尝试`recover`，并捕获`ActorError`返回，否则则将**Actor**方法的返回值序列化`[]byte`并返回。详见：`Runtime.shimCall`

4.2.5 lotus消息运转总结

节点发送消息，消息发送到本地的消息池消息池再经过P2P网络发送到相邻节点；节点同步消息则是通过其节点对网络中消息的监控，再对消息类别的判定进行此存储；当有消息到来时，矿工会自动创建一个新的区块，并将消息封装到该区块，然后进行执行;消息被执行，所有节点都将会对该结果进行存储。

5. 谈谈对lotus、Filecoin、IPFS、区块链等技术的感受

> 首先，感谢面试官给我的机会，也让我了解到了我以前几乎从未了解过的领域，这一天多的学习感觉收获还是挺大的，也知晓了自己的还有哪些不足以后需要多多改正。

> 总结一下自己学到的内容把，首先是web3.0，知晓了我们现如今web2.0存在的一些很严重的问题，比如：数据信息的垄断、安全性能得不到保障、还有大量的空间被浪费等等问题，web3.0将打破传统的前后端、数据库存储的模式，将会是前端、区块链的模式，让用户的信息去中心化的存储在区块链上；然后说说IPFS协议（星际文件系统），一开始我对它的理解就是感觉与http协议类似，只不过一个是传统服务器，一个区块链，然后慢慢对比学习，了解到了它设计思想的特点，然后又了解到它为了激励users发布了Filecoin网络，Filecoin网络内部的原生加密货币（FIL）更是其激励本质，它激励大家贡献自己的带宽与存储空间，然后给予FIL奖励等等。
>
> 然后到了这次面试官您给我任务，让我去学习lotus，一开始真的一点思路都没有，然后不断地在GitHub上，csdn，官方文档，B站上，个人博客上学习，然后总结，慢慢就理解它是干什么的与Filecoin有什么关系，然后去看它的源码，虽然有点困难，但还是慢慢的看，去弄清楚每个文件是干什么的，程序是怎么调用的等等，虽然我最后都没有跑起来这个项目，因为的电脑确实感觉不支持，我在虚拟机跑一直都停留在make cleam all，执行不下去，看了官网也晓得自己的硬件确实不够，真可惜，希望以后有机会能够跑起来试试。
>
> 最后，谢谢面试官您给我的这次机会，我希望能够进入贵公司实习，学习更多这方面的知识，谢谢。

参考学习：

lotus-csdn

ipfs-凯云实验室-bilibili

ipfs-玄月元月-bilibili

filecoin-玄月元月-bilibili

ipfs-官方文档

lotus-官网

filecoin-官网

lotus-飞行的蜗牛

IPFS星际联盟

web3.0-知乎-霖焱聊