IPFS之初始化流程

<center>目录</center>

一．IPFS 启动

1.1 简介

本文目的之一是帮助新了解 IPFS 的程序员，减轻阅读难度

本文的核心内容是理清 IPFS 初始化的流程以及细节，IPFS 如何监听，配置细节，参数细节，服务端与命令如何通信建立，初始化是很重要的一个部分，是一个程序主脉的入口，所谓来龙去脉自然少不了这一环，很有必要弄明白。

命令的流程首先是启动 daemon 命令，此命令启动了 ipfs 的服务节点。随后可以使用./ipfs add 等命令去添加文件

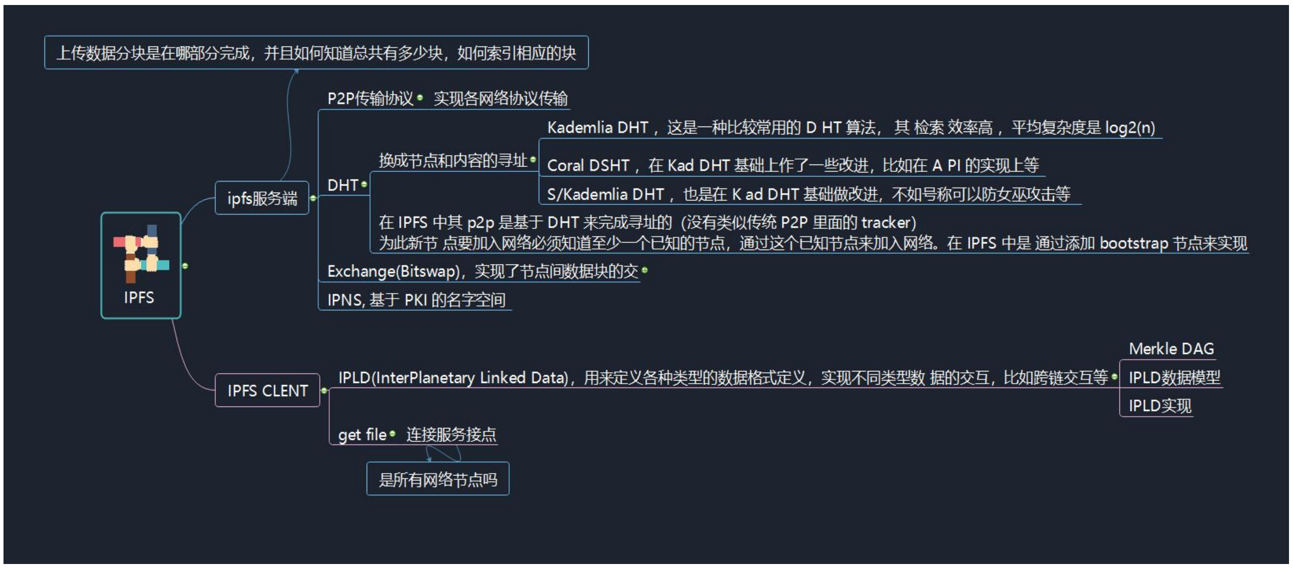
IPFS 的所有命令初始化前面都是一样，从某个点开始分化为两条路线分别执行不同功能，在文中也会详细介绍细节流程，文中回粘贴辅助许多代码进行说明

之后会以./ipfs damon 和 add file 两个命令来理清整个流程

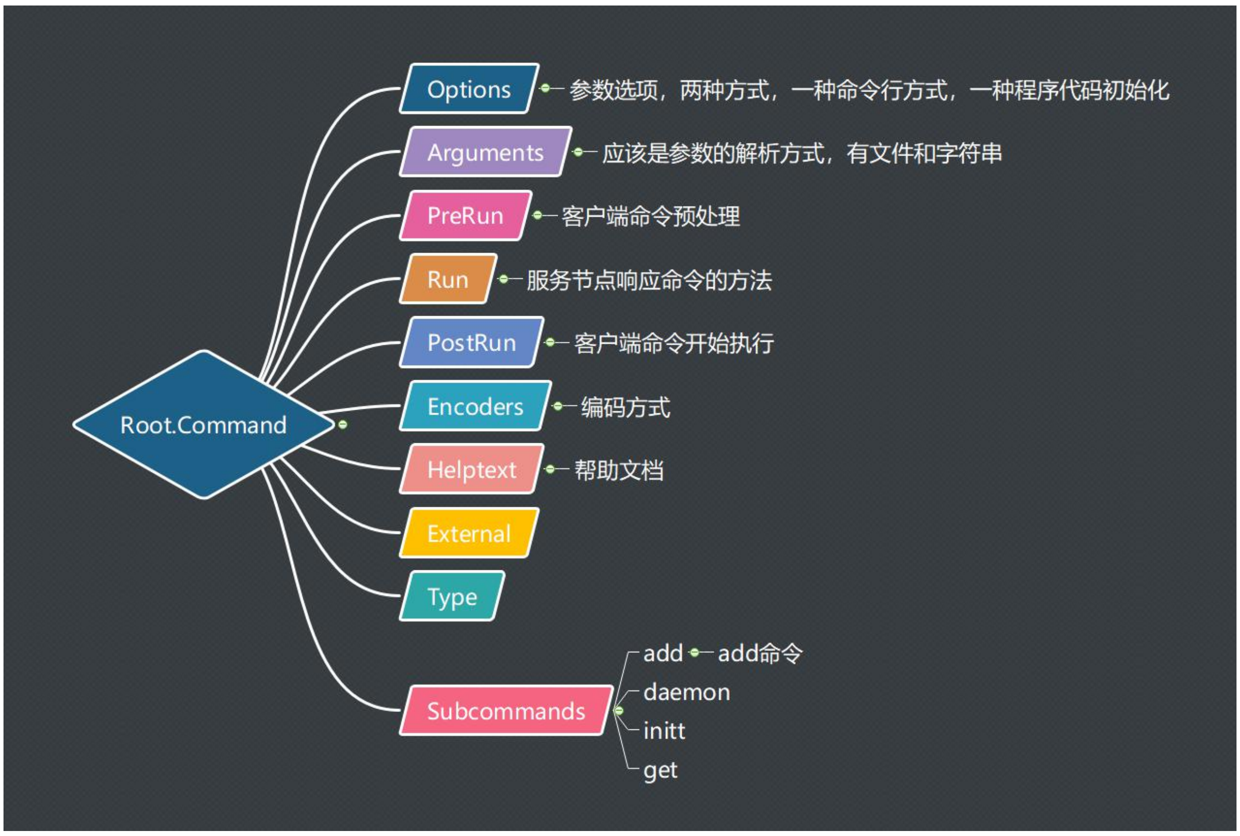
1.2 规则

1.第二章 2.x.1 是执行过程说明，如果详情阅读 2.x.2 可理解

1.3 IPFS 介绍

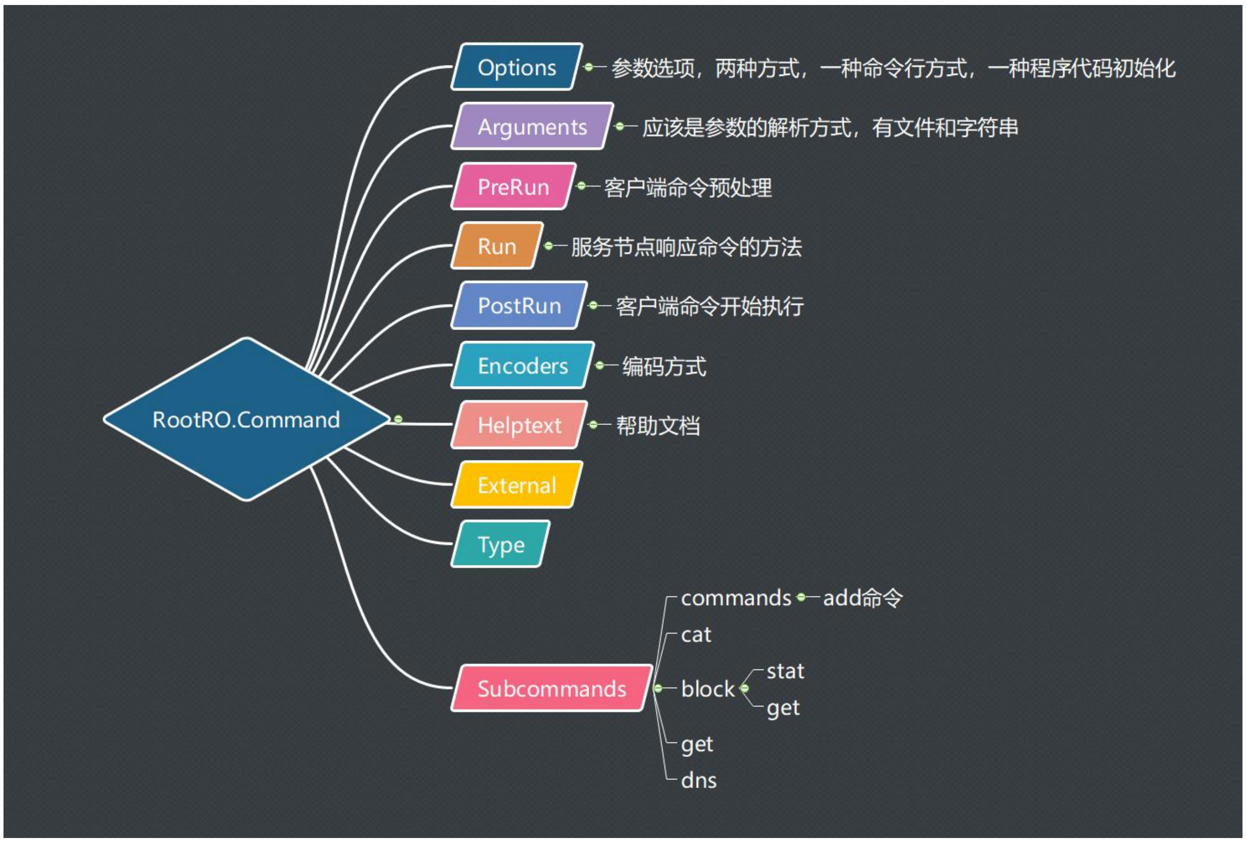

上传数据分块是在哪部分完成，并且如何知道总共有多少块，如何索引相应的块
P2P传输协议
实现各网络协议传输
Kademlia DHT，这是一种比较常用的 D HT 算法，其 检索 效率高，平均复杂度是 log2(n)
换成节点和内容的寻址
Coral DSHT，在 Kad DHT 基础上作了一些改进，比如在 A PI 的实现上等
DHT
S/Kademlia DHT，也是在 K ad DHT 基础做改进，不如号称可以防女巫攻击等
ipfs服务端
在 IPFS 中其 p2p 是基于 DHT 来完成寻址的 (没有类似传统 P2P 里面的 tracker)
为此新节 点要加入网络必须知道至少一个已知的节点，通过这个已知节点来加入网络。在 IPFS 中是 通过添加 bootstrap 节点来实现
IPFS
Exchange(Bitswap)，实现了节点间数据块的交
IPNS, 基于 PKI 的名字空间
Merkle DAG
IPFS CLENT
IPLD(InterPlanetary Linked Data)，用来定义各种类型的数据格式定义，实现不同类型数 据的交互，比如跨链交互等
IPLD数据模型
IPLD实现
get file
连接服务接点
是所有网络节点吗

1.4 IPFS Root 命令解析结构



1.5 IPFS RootRO 命令结构

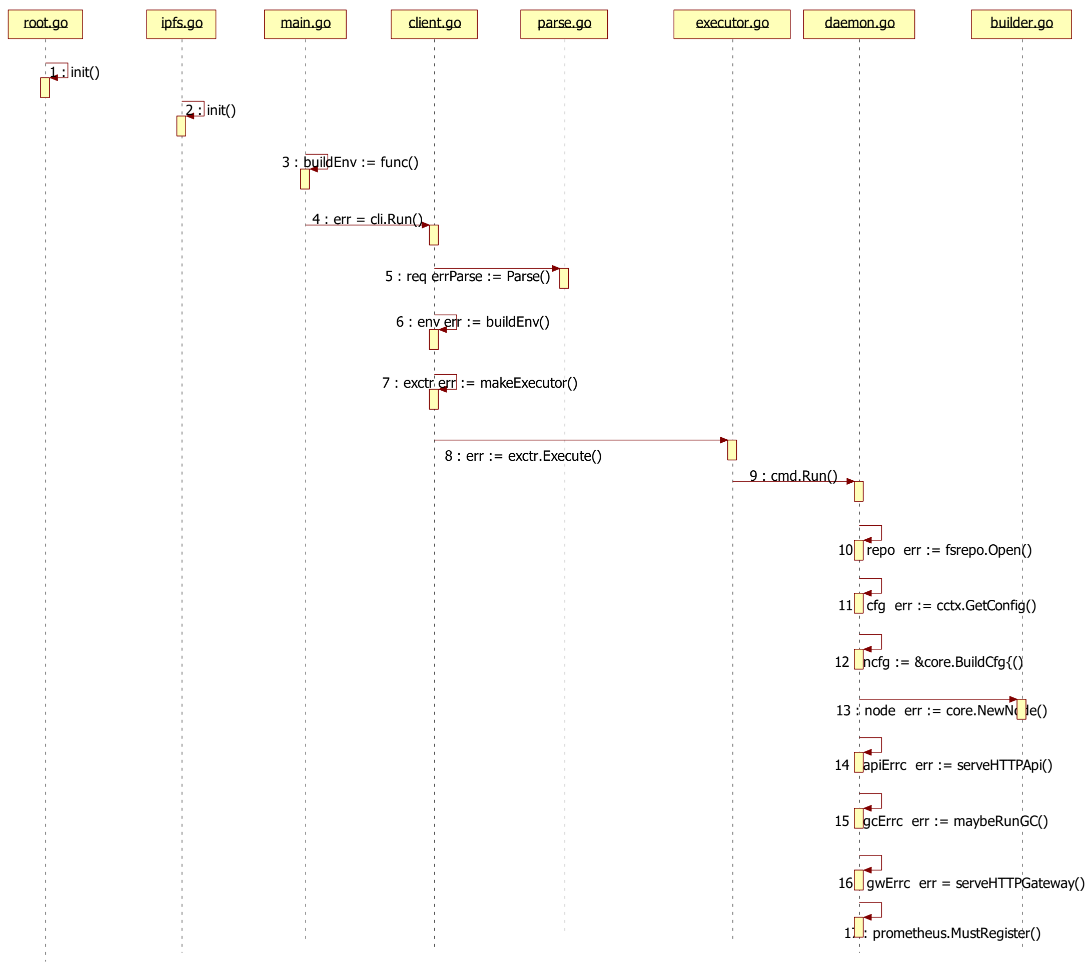
Root 和 RootRo 可以拥有相同的子命令，不同的是 RootRO 只能有只读的子命令



二． IPFS damon 流程

2.1 IPFS daemon 初始化时序图

2.1.1 demon 时序图如图 2-1



root.go
ipfs.go
main.go
client.go
parse.go
executor.go
daemon.go
builder.go
1 : init()
2 : init()
3 : buildEnv := func()
4 : err = cli.Run()
5 : req errParse := Parse()
6 : env err := buildEnv()
7 : exctr err := makeExecutor()
8 : err := exctr.Execute()
9 : cmd.Run()
10 repo err := fsrepo.Open()
11 cfg err := cctx.GetConfig()
12 ncfg := &core.BuildCfg{()
13 : node err := core.NewNode()
14 apiErrc err := serveHTTPApi()
15 gcErrc err := maybeRunGC()
16 gwErrc err = serveHTTPGateway()
17 : prometheus.MustRegister()

图 2-1

2.1.2 时序图说明

所有的命令前七部在初始化流程都是一致，从第八部开始就会产生差异，执行各自的使命

2.2 时序图第一步 root.go（init()）

2.1.1 源码初步说明

NODE2.1.1-1 执行内容

```
var RootRO = &cmds.Command{}// RootRO is the readonly version of Root
var RefsROCmd = &oldcmds.Command{}

func init() {
   fmt.Println("initinitinitinitinitinitinitinitinitinitinitinitinitinitinitinit")
   Root.ProcessHelp()  //初始化功能为，如果每个子命令的详细帮助说明则将此自命令的简单描述赋值为此子命令的说明
   *RootRO = *Root   //只读 RootRO 有一部分初始化和 Root 一样

  *RefsROCmd = *RefsCmd //与只读名关联的
   RefsROCmd.Subcommands = map[string]*oldcmds.Command{}
   rootROSubcommands["refs"] = lgc.NewCommand(RefsROCmd)//为只读 root 使用，可暂时不用古纳辛

   Root.Subcommands = rootSubcommands //关联所有 Root 的子命令  关联 NODE2.1.2-4
   RootRO.Subcommands = rootROSubcommands 关联所有 RootRo 只读的子命令 关联 NODE2.1.2-5

}
```

说明：
 最顶级的 Root 包含两种，包含了所有的子命令
 IPFS Root 所有自命令关联基本在这里完成

1. Root 包含的命令是可读写命令
2. RootRO 也是顶级命令，包含的子命令是只读的命令，如 cat get 等命令

2.1.2 相关联说明

NODE2.1.2-1

 可以说这个结构体基本包含了整个 ipfs 初始化，命令发送，响应的所有流程(oldcmds.Command 与此结构体一致)

```
type cmds.Command struct {
   Options   []cmdkit.Option 从
   Arguments []cmdkit.Argument
   PreRun    func(req Request) error  //客户端 add 命令在初始化后会执行预处理命令（daemon 不会执行）
   Run         Function（daemon 初始化完成之后会执行此命令，执行 daemon 内部的初始化操作，daemon 节点相应命令时会执行 ADD 的 Run 方法，进行处理）
   PostRun     Function//客户端 add 命令在初始化 PreRun 后会执行此命令处理命令（daemon 不会执行）
   Marshalers map[EncodingType]Marshaler
   Helptext   cmdkit.HelpText //每一个命令的帮助说明
   External bool
   Type        interface{}
   Subcommands map[string]*Command （这个很重要，比如 root 的自命令初始化种关联了所有的 add,get
等子命令，这个属性是链接所有子命令的桥梁，比如子命令 add 的 postRun 使用
Subcommands[“add”].PostRun 就直接可以操作 ）
}
```

NODE2.1.2-2

Root 相关初始化

```
var Root = &cmds.Command{
   Helptext: cmdkit.HelpText{
      Tagline:  "Global p2p merkle-dag filesystem.",
      Synopsis: "ipfs [--config=<config> | -c] [--debug=<debug> | -D] [--help=<help>] [-h=<h>]
[--local=<local> | -L] [--api=<api>] <command> ...",
      Subcommands: `
BASIC COMMANDS
  init          Initialize ipfs local configuration
  add <path>    Add a file to IPFS （省略很多）
`,
   },
   Options: []cmdkit.Option{
      cmdkit.StringOption("config", "c", "Path to the configuration file to use."),
      cmdkit.BoolOption("debug", "D", "Operate in debug mode."),
      cmdkit.BoolOption("help", "Show the full command help text."),
      cmdkit.BoolOption("h", "Show a short version of the command help text."),
      cmdkit.BoolOption("local", "L", "Run the command locally, instead of using the daemon."),
      cmdkit.StringOption(ApiOption, "Use a specific API instance (defaults to
/ip4/127.0.0.1/tcp/5001)"),

      // global options, added to every command
      cmds.OptionEncodingType,
      cmds.OptionStreamChannels,
      cmds.OptionTimeout,
   },
}
```

说明：以上主要做了 2 个部分的初始化（Helptext 帮助说明和 cmdkit.Option）

NODE2.1.2-3

RefsCmd 相关初始化

```
var RefsCmd = &cmds.Command{
   Helptext: cmdkit.HelpText{
      Tagline: "List links (references) from an object.",
      ShortDescription: `
Lists the hashes of all the links an IPFS or IPNS object(s) contains,
with the following format:
  <link base58 hash>
NOTE: List all references recursively by using the flag '-r'.
`,
   },
   Subcommands: map[string]*cmds.Command{
      "local": RefsLocalCmd,
   },
   Arguments: []cmdkit.Argument{
      cmdkit.StringArg("ipfs-path", true, true, "Path to the object(s) to list refs
from.").EnableStdin(),
   },
   Options: []cmdkit.Option{
      cmdkit.StringOption("format", "Emit edges with given format. Available tokens: <src> <dst>
<linkname>.").WithDefault("<dst>"),
      cmdkit.BoolOption("edges", "e", "Emit edge format: `<from> -> <to>`."),
      cmdkit.BoolOption("unique", "u", "Omit duplicate refs from output."),
      cmdkit.BoolOption("recursive", "r", "Recursively list links of child nodes."),
   },
   Run: func(req cmds.Request, res cmds.Response) {
      ctx := req.Context()
      n, err := req.InvocContext().GetNode()
      if err != nil {
         res.SetError(err, cmdkit.ErrNormal)
         return
      }

      unique, _, err := req.Option("unique").Bool()
      if err != nil {
         res.SetError(err, cmdkit.ErrNormal)
         return
      }

      recursive, _, err := req.Option("recursive").Bool()
      if err != nil {
         res.SetError(err, cmdkit.ErrNormal)
         return
      }

      format, _, err := req.Option("format").String()
      if err != nil {
         res.SetError(err, cmdkit.ErrNormal)
         return
      }

      edges, _, err := req.Option("edges").Bool()
      if err != nil {
         res.SetError(err, cmdkit.ErrNormal)
         return
      }
      if edges {
         if format != "<dst>" {
            res.SetError(errors.New("using format argument with edges is not allowed"),
               cmdkit.ErrClient)
            return
         }

         format = "<src> -> <dst>"
      }

      objs, err := objectsForPaths(ctx, n, req.Arguments())
      if err != nil {
         res.SetError(err, cmdkit.ErrNormal)
         return
      }

      out := make(chan interface{})
      res.SetOutput((<-chan interface{})(out))

      go func() {
         defer close(out)

         rw := RefWriter{
            out:       out,
            DAG:       n.DAG,
            Ctx:       ctx,
            Unique:    unique,
            PrintFmt:  format,
            Recursive: recursive,
         }

         for _, o := range objs {
            if _, err := rw.WriteRefs(o); err != nil {
               select {
               case out <- &RefWrapper{Err: err.Error()}:
               case <-ctx.Done():
                }
                return
            }
        }
    }()
},
Marshalers: refsMarshallerMap,
Type:       RefWrapper{},
}
```

说明 初始化内容为（Helptext，Subcommands，Arguments）

NODE2.1.2-4

此命令初始化为可读写的命令，以下的每一个子命令都包含了一个完整的已经初始化的子命令,如以下 add 命令。Root 包含了所有的 Command

```
Root.Subcommands = rootSubcommands
var rootSubcommands = map[string]*cmds.Command{
    "add":       AddCmd,
    "bitswap":   BitswapCmd,
    "block":     BlockCmd,
    "cat":       CatCmd,
    "commands":  CommandsDaemonCmd,
    "files":     FilesCmd,
    "filestore": FileStoreCmd,
    "get":       GetCmd,
    "pubsub":    PubsubCmd,
    "repo":      RepoCmd,
    "stats":     StatsCmd,
    "bootstrap": lgc.NewCommand(BootstrapCmd),
    "config":    lgc.NewCommand(ConfigCmd),
    "dag":       lgc.NewCommand(dag.DagCmd),
    "dht":       lgc.NewCommand(DhtCmd),
    "diag":      lgc.NewCommand(DiagCmd),
    "dns":       lgc.NewCommand(DNSCmd),
    "id":        lgc.NewCommand(IDCmd),
    "key":       KeyCmd,
    "log":       lgc.NewCommand(LogCmd),
    "ls":        lgc.NewCommand(LsCmd),
    "mount":     lgc.NewCommand(MountCmd),
    "name":      lgc.NewCommand(NameCmd),
    "object":    ocmd.ObjectCmd,
    "pin":       lgc.NewCommand(PinCmd),
    "ping":      lgc.NewCommand(PingCmd),
    "p2p":       lgc.NewCommand(P2PCmd),
   "refs":      lgc.NewCommand(RefsCmd),
   "resolve":   lgc.NewCommand(ResolveCmd),
   "swarm":     lgc.NewCommand(SwarmCmd),
   "tar":       lgc.NewCommand(TarCmd),
   "file":      lgc.NewCommand(unixfs.UnixFSCmd),
   "update":    lgc.NewCommand(ExternalBinary()),
   "urlstore":  urlStoreCmd,
   "version":   lgc.NewCommand(VersionCmd),
   "shutdown":  daemonShutdownCmd,
}
```

Addcmd 的命令初始化如下

```
var AddCmd = &cmds.Command{
   Helptext: cmdkit.HelpText{
      Tagline: "Add a file or directory to ipfs.",
      ShortDescription: `
Adds contents of <path> to ipfs. Use -r to add directories (recursively).
`,
      LongDescription: `
Adds contents of <path> to ipfs. Use -r to add directories.
Note that directories are added recursively, to form the ipfs
MerkleDAG.
`,
   },
   Arguments: []cmdkit.Argument{
      cmdkit.FileArg("path", true, true, "The path to a file to be added to
ipfs.").EnableRecursive().EnableStdin(),
   },
   Options: []cmdkit.Option{
      cmds.OptionRecursivePath, // a builtin option that allows recursive paths (-r, --recursive)
      cmdkit.BoolOption(quietOptionName, "q", "Write minimal output."),
      cmdkit.BoolOption(quieterOptionName, "Q", "Write only final hash."),
      cmdkit.BoolOption(silentOptionName, "Write no output."),
      cmdkit.BoolOption(progressOptionName, "p", "Stream progress data."),
      cmdkit.BoolOption(trickleOptionName, "t", "Use trickle-dag format for dag generation."),
      cmdkit.BoolOption(onlyHashOptionName, "n", "Only chunk and hash - do not write to disk."),
      cmdkit.BoolOption(wrapOptionName, "w", "Wrap files with a directory object."),
      cmdkit.BoolOption(hiddenOptionName, "H", "Include files that are hidden. Only takes effect
on recursive add."),
      cmdkit.StringOption(chunkerOptionName, "s", "Chunking algorithm, size-[bytes] or
rabin-[min]-[avg]-[max]").WithDefault("size-262144"),
      cmdkit.BoolOption(pinOptionName, "Pin this object when adding.").WithDefault(true),
      cmdkit.BoolOption(rawLeavesOptionName, "Use raw blocks for leaf nodes. (experimental)"),
      cmdkit.BoolOption(noCopyOptionName, "Add the file using filestore. Implies raw-leaves.
(experimental)"),
      cmdkit.BoolOption(fstoreCacheOptionName, "Check the filestore for pre-existing blocks.
(experimental)"),
      cmdkit.IntOption(cidVersionOptionName, "CID version. Defaults to 0 unless an option that
depends on CIDv1 is passed. (experimental)"),
      cmdkit.StringOption(hashOptionName, "Hash function to use. Implies CIDv1 if not sha2-256.
(experimental)").WithDefault("sha2-256"),
   },
   PreRun: func(req *cmds.Request, env cmds.Environment) error {

      return nil
   },
   Run: func(req *cmds.Request, res cmds.ResponseEmitter, env cmds.Environment) {
}
PostRun: cmds.PostRunMap{
}
}
```

NODE2.1.2-5

此命令初始化为只读功能的子命令，以下的每一个子命令都包含了一个完整的已经初始化的子命令,如以下 cat 命令。RootRO.Subcommands 包含了所有的只读类型自命令

```
RootRO.Subcommands = rootROSubcommands
var rootROSubcommands = map[string]*cmds.Command{
   "commands": CommandsDaemonROCmd,
   "cat":      CatCmd,
   "block": &cmds.Command{
      Subcommands: map[string]*cmds.Command{
         "stat": blockStatCmd,
         "get":  blockGetCmd,
      },
   },
   "get": GetCmd,
   "dns": lgc.NewCommand(DNSCmd),
   "ls":  lgc.NewCommand(LsCmd),
   "name": lgc.NewCommand(&oldcmds.Command{
      Subcommands: map[string]*oldcmds.Command{
         "resolve": IpnsCmd,
      },
   }),
   "object": lgc.NewCommand(&oldcmds.Command{
      Subcommands: map[string]*oldcmds.Command{
         "data":  ocmd.ObjectDataCmd,
         "links": ocmd.ObjectLinksCmd,
            "get":   ocmd.ObjectGetCmd,
            "stat":  ocmd.ObjectStatCmd,
        },
    }),
    "dag": lgc.NewCommand(&oldcmds.Command{
        Subcommands: map[string]*oldcmds.Command{
            "get":     dag.DagGetCmd,
            "resolve": dag.DagResolveCmd,
        },
    }),
    "resolve": lgc.NewCommand(ResolveCmd),
    "version": lgc.NewCommand(VersionCmd),
}
```

2.2 时序图第二步 ipfs.go（init()）

2.2.1 源码初步说明

NODE2.2.1-1 执行内容
以下内容其实就是把 daemon init commands 等子命令和 add 等自命令合并在一起作为 Root 得子命令

```
func init() {

    // setting here instead of in literal to prevent initialization loop
    // (some commands make references to Root)
    Root.Subcommands = localCommands// 此 Root.Subcommand 与几个关键的子命令加入（daamon init
commands 几个命令，详情看 NODE2.2.2-1 ）

  //（以下代码主要是把之前 add 所有的自命令和 localCommands 的自命令何在一起，所以目前的
Root.Subcommand 包含了 daemon init add 等可读写命令，关联 NODE2.1.2-4  NODE2.2.2-1）
    for k, v := range commands.Root.Subcommands {
        if _, found := Root.Subcommands[k]; !found {
            Root.Subcommands[k] = v
        }
    }
}
```

2.2.2 相关联说明

NODE2.2.2-1 执行内容()

```go
var localCommands = map[string]*cmds.Command{
   "daemon":   daemonCmd,
   "init":     initCmd,
   "commands": commandsClientCmd,
}
```

2.3 时序图第三步 main.go（mainRet()）

2.3.1 源码初步说明

NODE2.3.1-1 执行内容

```go
func mainRet() int {

//创建 ctx
  ctx := logging.ContextWithLoggable(context.Background(), loggables.Uuid("session"))
  //以上省略无用初始化
  //以下参数初始化
   // Handle `ipfs help'
   if len(os.Args) == 2 {
      if os.Args[1] == "help" {
         os.Args[1] = "-h"
      } else if os.Args[1] == "--version" {
         os.Args[1] = "version"
      }
   }
   os.Args[0] = "ipfs"
```

//先创建一个 buildEnv，为 cli.Run 生成 Cmds.Environment

```go
   buildEnv := func(ctx context.Context, req *cmds.Request) (cmds.Environment, error) {
      checkDebug(req)
      repoPath, err := getRepoPath(req)

      if err != nil {
         return nil, err
      }
      log.Debugf("config path is %s", repoPath)

      // this sets up the function that will initialize the node
      // this is so that we can construct the node lazily.
      return &oldcmds.Context{
         ConfigRoot: repoPath, //   为/root/.ipfs
```

```
            LoadConfig: loadConfig,//可以导入配置的 func 传参
            ReqLog:     &oldcmds.ReqLog{},
//一个新节点创建的方法
            ConstructNode: func() (n *core.IpfsNode, err error) {
                if req == nil {
                    return nil, errors.New("constructing node without a request")
                }
                r, err := fsrepo.Open(repoPath)
                if err != nil { // repo is owned by the node
                    return nil, err
                }
                // ok everything is good. set it on the invocation (for ownership)
                // and return it.
                n, err = core.NewNode(ctx, &core.BuildCfg{
                    Repo: r,
                })
                if err != nil {
                    return nil, err
                }

                n.SetLocal(true)
                return n, nil
            },
        }, nil
    }
```

//在之后得 run 过程执行剩余初始化，和正规程序得运行在内部处理，makeExecutor 是一个 func，详情可以参照 **NODE2.3.2-1**

```
    err = cli.Run(ctx, Root, os.Args, os.Stdin, os.Stdout, os.Stderr, buildEnv, makeExecutor)
    if err != nil {
        return 1
    }

    // everything went better than expected :)
    return 0
}
```

说明以上 main 执行 4 步

1. 创建 ctx
2. 参数简单解析
3. buildEnv func 定义
4. cli.Run（）执行后续过程（内部在之后说明）

2.3.2 相关联说明

NODE2.3.2-1

makeExecutor 说明

```
func makeExecutor(req *cmds.Request, env interface{}) (cmds.Executor, error) {
   fmt.Println("makeExecutor")
   fmt.Println("details := commandDetails(req.Path)",req.Path)
   details := commandDetails(req.Path)
   client, err := commandShouldRunOnDaemon(*details, req, env.(*oldcmds.Context))
   if err != nil {
      return nil, err
   }

   var exctr cmds.Executor
     //*********走 client.executor
   if client != nil && !req.Command.External {
      fmt.Println("client != nil && !req.Command.External")
      exctr = client.(cmds.Executor)
   } else {
        //***************走 executor.Execute()
      fmt.Println("cctx := env.(*oldcmds.Context)")
      cctx := env.(*oldcmds.Context)
      pluginpath := filepath.Join(cctx.ConfigRoot, "plugins")

      // check if repo is accessible before loading plugins
      ok, err := checkPermissions(cctx.ConfigRoot)
      if err != nil {
         return nil, err
      }
      if ok {
         if _, err := loader.LoadPlugins(pluginpath); err != nil {
            log.Error("error loading plugins: ", err)
         }
      }

      exctr = cmds.NewExecutor(req.Root)
   }

   return exctr, nil
}
```

说明

makeExecutor 最关键的工作是分离 2 类命令的工作，也就是时序图第八步分离 cmds.Executor 是一个接口，executor 和 client 都实现了其接口，如果 daemon 参数会执行 executor 的 Execute（），add 会执行 client.Execute()

2.4 时序图第四步 Run.go（cli.Run()）

2.4.1 源码初步说明

NODE2.4.1-1

```
err = cli.Run(ctx, Root, os.Args, os.Stdin, os.Stdout, os.Stderr, buildEnv, makeExecutor)
```

此方法是各类命令初始化的过程，所有的命令都要执行，从 err := exctr.Execute(req, re, env) 方法的开始，以下会标注数字，方便阅读

```
func Run(ctx context.Context, root *cmds.Command,
   cmdline []string, stdin, stdout, stderr *os.File,
   buildEnv cmds.MakeEnvironment, makeExecutor cmds.MakeExecutor) error {

   printErr := func(err error) {
      fmt.Fprintf(stderr, "Error: %s\n", err)
   }
```

//解析参数，生成 req,内容细节非常多，编号 1

```
   req, errParse := Parse(ctx, cmdline[1:], stdin, root)

   // Handle the timeout up front.
   var cancel func()
   if timeoutStr, ok := req.Options[cmds.TimeoutOpt]; ok {
      fmt.Println(" timeoutStr, ok := req.Options[cmds.TimeoutOpt];",timeoutStr)
      timeout, err := time.ParseDuration(timeoutStr.(string))
      if err != nil {
         return err
      }
      req.Context, cancel = context.WithTimeout(req.Context, timeout)
   } else {
      req.Context, cancel = context.WithCancel(req.Context)
   }
   defer cancel()

   fmt.Println("req.Path,",req.Path)
   // this is a message to tell the user how to get the help text
   printMetaHelp := func(w io.Writer) {
      cmdPath := strings.Join(req.Path, " ")
      fmt.Fprintf(w, "Use '%s %s --help' for information about this command\n", cmdline[0],
cmdPath)
    }
    printHelp := func(long bool, w io.Writer) {
        fmt.Println("printHelp")
        helpFunc := ShortHelp
        if long {
            helpFunc = LongHelp
        }

        var path []string
        if req != nil {
            path = req.Path
        }

        if err := helpFunc(cmdline[0], root, path, w); err != nil {
            // This should not happen
            panic(err)
        }
    }
```

//跟帮助相关的显示，如果不是需要执行的命令，则直接返回帮助提示 编号 2

```
    err := HandleHelp(cmdline[0], req, stdout)
    if err == nil {
        return nil
    } else if err != ErrNoHelpRequested {
        return err
    }
    // no help requested, continue.

    // ok now handle parse error (which means cli input was wrong,
    // e.g. incorrect number of args, or nonexistent subcommand)
    if errParse != nil {
        printErr(errParse)

        // this was a user error, print help
        if req != nil && req.Command != nil {
            fmt.Fprintln(stderr) // i need some space
            printHelp(false, stderr)
        }

        return err
    }
```

```
// here we handle the cases where
// - commands with no Run func are invoked directly.
// - the main command is invoked.
if req == nil || req.Command == nil || req.Command.Run == nil {
   fmt.Println("if req == nil || req.Command == nil || req.Command.Run == nil")
   printHelp(false, stdout)
   return nil
}

cmd := req.Command
```

//调用 NODE2.3.1-1 申明的方法生成 env 相关 context 编号 3

```
env, err := buildEnv(req.Context, req)
if err != nil {
   printErr(err)
   return err
}
if c, ok := env.(Closer); ok {
   defer c.Close()
}
```

//调用 NODE2.3.2-1 exctr 会根据 req 参数去分离命令，daemon 会调用 execute 的方法，add 种类命令会调用 client 的方法　编号 4

```
exctr, err := makeExecutor(req, env)
if err != nil {
   printErr(err)
   return err
}

var (
   re     cmds.ResponseEmitter
   exitCh <-chan int
)

encTypeStr, _ := req.Options[cmds.EncLong].(string)
encType := cmds.EncodingType(encTypeStr)
fmt.Println("encType := cmds.EncodingType(encTypeStr)",encTypeStr)

// use JSON if text was requested but the command doesn't have a text-encoder
if _, ok := cmd.Encoders[encType]; encType == cmds.Text && !ok {
   req.Options[cmds.EncLong] = cmds.JSON
   fmt.Println("  req.Options[cmds.EncLong] = cmds.JSON")
}

// first if condition checks the command's encoder map, second checks global encoder map (cmd
vs. cmds)
```

```
if enc, ok := cmd.Encoders[encType]; ok {
   fmt.Println("if enc, ok := cmd.Encoders[encType]; ok ")
   re, exitCh = NewResponseEmitter(stdout, stderr, enc, req)
} else if enc, ok := cmds.Encoders[encType]; ok {
   fmt.Println("else if enc, ok := cmd.Encoders[encType]; ok ")  //********************
   re, exitCh = NewResponseEmitter(stdout, stderr, enc, req)
} else {

   return fmt.Errorf("could not find matching encoder for enctype %#v", encType)
}
fmt.Println("errCh := make(chan error, 1) ")
errCh := make(chan error, 1)
go func() {
   fmt.Println("  err := exctr.Execute(req, re, env)")
```

//调用 调用此方法会进入不同命令的主线，如 daemon 会调用 Run, add 回调用 PreRun 和 PostRun 编号 5

```
   err := exctr.Execute(req, re, env)
   if err != nil {
      errCh <- err
   }
}()
```

//根据执行的结果进行响应 编号 6

```
select {
case err := <-errCh:
   fmt.Println("err := <-errCh: 1111111111111111111111111111")
   printErr(err)

   if kiterr, ok := err.(*cmdkit.Error); ok {
      err = *kiterr
   }
   if kiterr, ok := err.(cmdkit.Error); ok && kiterr.Code == cmdkit.ErrClient {
      printMetaHelp(stderr)
   }

   return err

case code := <-exitCh:
   fmt.Println("code := <-exitCh2222222222222222222 ")
   if code != 0 {
      return ExitError(code)
   }
}
```

```
    return nil
}
```
说明以上方法有重要的六步

编号 1 主要是解析命令行，生成 req 的重要部分参数，内部比较复杂未来很多参数都跟此有关系，所以非常重要

编号 2 当输入命令有问题时，会产生帮助文档，在此处进行

编号 3 执行环境变量相关参数生成方法

编号 4 会将各种命令参数的执行分成两条路，一条执行 exetute 的方法，另一条执行 client 的方法，这点分厂重要

编号 5 内部会分流，使各个命令去分别执行各自的使命

编号 6 监听命令折行的结果

2.4.2 相关联说明

NODE2.4.2-1

```
type Request struct {
    Context       context.Context //环境变量，可以获得系统相关的一些信息
    Root, Command *Command //Root 是跟命令，内部集成了所有的子命令。 Command 命令是此 Req 本身应该执行哪一条命令，若果是 daemon,对应的就是 daemon 命令的实体

//对应命令名字如 执行 ipfs/./ipfs add ipfs/ggg.jpg
    Path      []string   // Path 值为[add ]
    Arguments []string   //对应内容为 ipfs/ggg.jpg
    Options   cmdkit.OptMap  //在执行‘-’或者‘--’等参数时候有用，每个对应的子命令都有对应的参数选项 如 -p, 会解析为 process,会将 Options[process]设置为 true, 就是设置 option 选项

    Files files.File  //会把本地的文件路径对应的文件读入

    bodyArgs *arguments  //暂时未见用点
}
```

说明

对于 Request 每一个参数如何赋值，并且用途有必要了解非常清楚

Options　传参目前发现的方式有两种（非常重要对于命令参数的使用）

一种是通过 (-p) 等方式传参

第二种是在每种命令的程序代码中可以设置默认参数以下是 add command 的初始化参数，下面有 withDefault 的才会作为请求参数传递

```
Options: []cmdkit.Option{
cmdkit.BoolOption(quietOptionName, "q", "Write minimal output."),
cmdkit.BoolOption(pinOptionName, "Pin this object when adding.").WithDefault(true),
cmdkit.StringOption(hashOptionName, "Hash function to use. Implies CIDv1 if not sha2-256. (experimental)").WithDefault("sha2-256"),
},
```

2.5 时序图第五步 parse.go（Parse()）

2.5.1 源码初步说明

NODE2.5.1-1

```
req, errParse := Parse(ctx, cmdline[1:], stdin, root)
```

主要执行在 cli.Run,编号 1 中执行

```
func Parse(ctx context.Context, input []string, stdin *os.File, root *cmds.Command)
(*cmds.Request, error) {
   req := &cmds.Request{Context: ctx}
```

//编号 1　方法非常重要，解析内容为 req.Root = root(最初时序图第二部初始化), req.Command（add 子命令）　req.Path(add) req.Arguments(ggg.jpg) req.Options = opts（只针对-p，等传参进行赋值）（还未做的功能有默认 option 赋值和文件赋值）

```
   if err := parse(req, input, root); err != nil {
      return req, err
   }
```

//编号 2 对 Req.option 的默认参数赋值，此默认值一般在程序子命令的代码中进行更改

```
   if err := req.FillDefaults(); err != nil {
      return req, err
   }

   // This is an ugly hack to maintain our current CLI interface while fixing
   // other stdin usage bugs. Let this serve as a warning, be careful about the
   // choices you make, they will haunt you forever.
   if len(req.Path) == 2 && req.Path[0] == "bootstrap" {
      if (req.Path[1] == "add" && req.Options["default"] == true) ||
         (req.Path[1] == "rm" && req.Options["all"] == true) {
         stdin = nil
      }
   }
```

//编号 3 对 Req.Files 的默认参数赋值，此默认值一般在程序子命令的代码中进行更改,到此，Req 的基本参数都赋值成功

```
   if err := parseArgs(req, root, stdin); err != nil {
      return req, err
   }
```

```
    // if no encoding was specified by user, default to plaintext encoding
    // (if command doesn't support plaintext, use JSON instead)
```
//编号 4　对于 req.option 的选项编码设置，有文本和 Json 设置两种

```
    if enc := req.Options[cmds.EncLong]; enc == "" {
        if req.Command.Encoders != nil && req.Command.Encoders[cmds.Text] != nil {
            fmt.Println("********************  req.SetOption(cmds.EncLong, cmds.Text)")
            req.SetOption(cmds.EncLong, cmds.Text)
        } else {
            fmt.Println("***********************req.SetOption(cmds.EncLong, cmds.JSON)")
            req.SetOption(cmds.EncLong, cmds.JSON)
        }
    }

    return req, nil
}
```
说明解析分为四步，解析后构造一个完整的 Req

1. 编号一解析了主要的参数构成（整个 req 的参数解析基本里面完成，设计内容较多，需要理解 Req.option 的构建方式）
2. 编号 2 对 req.option 的默认参数进行设置
3. 对 Req 要传送的 files 进行设置
4. 对 Req.opention 选项的编码进行设置

2.5.2 对编号上述方法进行详细解释

NODE2.5.2-1

详细解析 NODE2.5.1-1 的编号 1

此方法构建了 Req 的绝大部分参数

```
  func parse(req *cmds.Request, cmdline []string, root *cmds.Command) (err error) {

  var (
      path = make([]string, 0, len(cmdline))
      args = make([]string, 0, len(cmdline))
      opts = cmdkit.OptMap{}
      cmd  = root
   )

    st := &parseState{cmdline: cmdline}
    fmt.Println("cmdline",cmdline)
  //以上为初始化
```
// 编号 1 此部分的代码比较重要，只有在带有 "-" ,"--"等参数的命令在此处生成的 optDefs 才会赋值给 Req 主要工作内容为分为两步

1. **把 Root 和 Add 的 option 参数数组合成一个数组　2 步.把 数组中所有的参数再拆分如下 cmdkit.BoolOption("debug", "D", "Operate in debug mode.") 类似的命令分成 2 个 map[string]cmdkit.Option 如 optionsMap["D"] = cmdkit.BoolOption("debug", "D", "Operate in debug mode.") 和 optionsMap["debug"] = cmdkit.BoolOption("debug", "D", "Operate in debug mode.") 使用此方式可以把-D 等命令变成 参数 debug = true 的映射**

```
   optDefs, err := root.GetOptions([]string{})
   if err != nil {
      return err
   }

L:
   // don't range so we can seek
   for !st.done() {
      param := st.peek()
      fmt.Println("param",param)
      switch {
      case param == "--":
         fmt.Println("case param == --")
         // use the rest as positional arguments
         args = append(args, st.cmdline[st.i+1:]...)
         break L
      case strings.HasPrefix(param, "--"):
         fmt.Println("case param == --")
         // long option
         k, v, err := st.parseLongOpt(optDefs)
         if err != nil {
            return err
         }

         if _, exists := opts[k]; exists {
            return fmt.Errorf("multiple values for option %q", k)
         }

         k = optDefs[k].Name()
         opts[k] = v

      case strings.HasPrefix(param, "-") && param != "-":
         // short options
         fmt.Println("strings.HasPrefix(param, -) && param !== -",st.cmdline)
         kvs, err := st.parseShortOpts(optDefs)

         if err != nil {
            return err
```

```
        }

        for _, kv := range kvs {
           kv.Key = optDefs[kv.Key].Names()[0]
           fmt.Println("kvs, err := st.parseShortOpts(optDefs)",kv.Key)
           if _, exists := opts[kv.Key]; exists {
              return fmt.Errorf("multiple values for option %q", kv.Key)
           }
```

//**编号 2 如果带有-p 等参数，req.option 才会赋值，为 opts[“progress”]=true**

```
           opts[kv.Key] = kv.Value
           fmt.Println("opts[kv.Key] = kv.Value",opts[kv.Key])
        }
     default:

        arg := param          // arg is a sub-command or a positional argument
```

//编号 3 root 几何了所有可读写类的子命令，在时序图一有解释，此处是把 add 子命令取出来，并作为 Req 的 command

```
        sub := cmd.Subcommands[arg]
        if sub != nil {
           cmd = sub
           path = append(path, arg)
           fmt.Println("path, append(path, arg)",path)
           optDefs, err = root.GetOptions(path)
           if err != nil {
              return err
           }

           // If we've come across an external binary call, pass all the remaining
           // arguments on to it
           if cmd.External {
              fmt.Println(" cmd.External {")
              args = append(args, st.cmdline[st.i+1:]...)
              break L
           }
        } else {
           args = append(args, arg)
           fmt.Println("else path, append(path, arg)",args)
           if len(path) == 0 {
              // found a typo or early argument
              return printSuggestions(args, root)
           }
        }
     }
```

```
        fmt.Println("!st.done() args",args)
        st.i++
    }
```

//编号 4 对 Req 的参数值进行初始化的

```
    req.Root = root //最初的 root，所有部分已经在第二步时序图处理完成
    req.Command = cmd // (root..Subcommands[“add”]),其他类推
    req.Path = path  // add
    req.Arguments = args // ipfs/ggg.jpg
    req.Options = opts  //(跟带-- - 的传参有关系，做初始化赋值)

    return nil
}
```

说明 分为 4 步，

1 第一步是初始化 Root 和 daemon add 命令的 optionc 参数合并在一起

2. 获取 Req,的命令
3. 编号 3，针对-，--等参数进行解析
4. 进行 Req 的初始化共奏

NODE2.5.2-2

详细解析 NODE2.5.1-1 的编号 2

NODE2.5.2-3

详细解析 NODE2.5.1-1 的编号 3

2.6 时序图第六步 run.go（buildEnv()）

2.6.1 源码初步说明

NODE2.6.1-1

创建环境变量，为下一步做铺垫

env err := buildEnv(req.Context, req)

```
buildEnv := func(ctx context.Context, req *cmds.Request) (cmds.Environment, error) {
    checkDebug(req)
    //编号 1 配置文件路径读取
    repoPath, err := getRepoPath(req)
    fmt.Println("repoPath",repoPath)
    if err != nil {
        return nil, err
    }
    log.Debugf("config path is %s", repoPath)

    // this sets up the function that will initialize the node
    // this is so that we can construct the node lazily.
    //编号 2 配置文件路径读取 返回环境变量
    return &oldcmds.Context{
        ConfigRoot: repoPath,
        LoadConfig: loadConfig,
        ReqLog:     &oldcmds.ReqLog{},

        ConstructNode: func() (n *core.IpfsNode, err error) {
            if req == nil {
                return nil, errors.New("constructing node without a request")
            }

        //编号 3  返回 repo.Repo
            r, err := fsrepo.Open(repoPath)
            if err != nil { // repo is owned by the node
                return nil, err
            }

            // ok everything is good. set it on the invocation (for ownership)
            // and return it.
        //编号 4  此方法基本构建了一个 ipfsNode 节点,一个 ipfsNode 几乎包含了一个节点的所有信息
            n, err = core.NewNode(ctx, &core.BuildCfg{
                Repo: r,
            })
            if err != nil {
                return nil, err
            }

            n.SetLocal(true)
            return n, nil
        },
    }, nil
}
```

说明：执行此方法有三点要注意

1. 读取配置文件的路径
2. 主要是返回一个 request.go 的环境变量

2.6.2 源码补充说明

NODE2.6.2-1

request.go 的 Context

```
type Context struct {
    Online     bool
    ConfigRoot string
    ReqLog     *ReqLog

    config     *config.Config
    LoadConfig func(path string) (*config.Config, error)

    api           coreiface.CoreAPI
    node          *core.IpfsNode
    ConstructNode func() (*core.IpfsNode, error)
}
```

NODE2.6.2-2

IpfsNodes 说明，一个 IpfsNode 关联了 Ipfs 节点的所有信息，非常庞大，每一个属性内部关联较多

```
// IpfsNode is IPFS Core module. It represents an IPFS instance.
type IpfsNode struct {

    // Self
    Identity peer.ID // the local node's identity

    Repo repo.Repo

    // Local node
    Pinning         pin.Pinner // the pinning manager
    Mounts          Mounts     // current mount state, if any.
    PrivateKey      ic.PrivKey // the local node's private Key
    PNetFingerprint []byte     // fingerprint of private network

    // Services
    Peerstore       pstore.Peerstore     // storage for other Peer instances
    Blockstore      bstore.GCBlockstore  // the block store (lower level)
    Filestore       *filestore.Filestore // the filestore blockstore
    BaseBlocks      bstore.Blockstore    // the raw blockstore, no filestore wrapping
    GCLocker        bstore.GCLocker      // the locker used to protect the blockstore during gc
    Blocks          bserv.BlockService   // the block service, get/add blocks.
    DAG             ipld.DAGService      // the merkle dag service, get/add objects.
    Resolver        *resolver.Resolver   // the path resolution system
    Reporter        metrics.Reporter
    Discovery       discovery.Service
    FilesRoot       *mfs.Root
    RecordValidator record.Validator

    // Online
    PeerHost     p2phost.Host        // the network host (server+client)
    Bootstrapper io.Closer           // the periodic bootstrapper
    Routing      routing.IpfsRouting // the routing system. recommend ipfs-dht
    Exchange     exchange.Interface  // the block exchange + strategy (bitswap)   Bitswap实现
了接口
    Namesys      namesys.NameSystem  // the name system, resolves paths to hashes
    Ping         *ping.PingService
    Reprovider   *rp.Reprovider // the value reprovider system
    IpnsRepub    *ipnsrp.Republisher

    Floodsub *floodsub.PubSub
    PSRouter *psrouter.PubsubValueStore
    DHT      *dht.IpfsDHT
    P2P      *p2p.P2P

    proc goprocess.Process
    ctx  context.Context

    mode         mode
    localModeSet bool
}
```

NODE2.6.2-3

```
r, err := fsrepo.Open(repoPath)
```

NODE2.6.2-4

```
  n, err = core.NewNode(ctx, &core.BuildCfg{
            Repo: r,
         })
```

2.7 时序图第七步 run.go（makeExecutor()）

2.7.1 源码初步说明

NODE2.7.1-1

```
exctr, err := makeExecutor(req, env)

func makeExecutor(req *cmds.Request, env interface{}) (cmds.Executor, error) {
   fmt.Println("makeExecutor")
   fmt.Println("details := commandDetails(req.Path)",req.Path)
   details := commandDetails(req.Path)
   client, err := commandShouldRunOnDaemon(*details, req, env.(*oldcmds.Context))
   if err != nil {
      return nil, err
   }

   var exctr cmds.Executor
   if client != nil && !req.Command.External {
      fmt.Println("client != nil && !req.Command.External")
      exctr = client.(cmds.Executor)
   } else {
      fmt.Println("cctx := env.(*oldcmds.Context)")
      cctx := env.(*oldcmds.Context)
      pluginpath := filepath.Join(cctx.ConfigRoot, "plugins")

      // check if repo is accessible before loading plugins
      ok, err := checkPermissions(cctx.ConfigRoot)
      if err != nil {
            return nil, err
        }
        if ok {
            if _, err := loader.LoadPlugins(pluginpath); err != nil {
                log.Error("error loading plugins: ", err)
            }
        }

        exctr = cmds.NewExecutor(req.Root)
    }

    return exctr, nil
}
```

2.7.2 源码补充说明

NODE2.7.2-1

2.8 时序图第八步 executor.go（Execute()）

2.8.1 源码初步说明

NODE2.8.1-1

```
func (x *executor) Execute(req *Request, re ResponseEmitter, env Environment) (err error) {

    fmt.Println(" (x *executor) Execute(req *Request, re ResponseEmitter, env Environment) (err
error) {")
    cmd := req.Command

    if cmd.Run == nil {
       return ErrNotCallable
    }

    err = cmd.CheckArguments(req)
    if err != nil {
       return err
    }

    // If this ResponseEmitter encodes messages (e.g. http, cli or writer - but not chan),
    // we need to update the encoding to the one specified by the command.
    if ee, ok := re.(EncodingEmitter); ok {
       encType := GetEncoding(req)
       fmt.Println("executor:encType",encType)
       // use JSON if text was requested but the command doesn't have a text-encoder
       if _, ok := cmd.Encoders[encType]; encType == Text && !ok {
          fmt.Println("encType = JSON")
          encType = JSON
       }

       if enc, ok := cmd.Encoders[encType]; ok {
          fmt.Println("executor  if enc, ok := cmd.Encoders[encType]; ok {")
          ee.SetEncoder(enc(req))
       } else if enc, ok := Encoders[encType]; ok {
          fmt.Println("executor  else if enc, ok := Encoders[encType]; ok {")
          ee.SetEncoder(enc(req))
       } else {
          log.Errorf("unknown encoding %q, using json", encType)
          fmt.Println("executor  ee.SetEncoder(Encoders[JSON](req))")
          ee.SetEncoder(Encoders[JSON](req))
       }
    }

    if cmd.PreRun != nil {
       fmt.Println("executor        if cmd.PreRun != nil {")
       err = cmd.PreRun(req, env)
       if err != nil {
          return err
       }
    }
```

```
    if cmd.PostRun != nil {
        fmt.Println("executor  if cmd.PostRun != nil {")
        if typer, ok := re.(interface {
            Type() PostRunType
        }); ok && cmd.PostRun[typer.Type()] != nil {
            re = cmd.PostRun[typer.Type()](req, re)
        }
    }

    defer func() {
        re.Close()
    }()
    defer func() {
        // catch panics in Run (esp. from re.SetError)
        if v := recover(); v != nil {
            // if they are errors
            if e, ok := v.(error); ok {
                // use them as return error
                err = re.Emit(cmdkit.Error{Message: e.Error(), Code: cmdkit.ErrNormal})
                if err != nil {
                    log.Errorf("recovered from command error %q but failed emitting it: %q", e, err)
                }
            } else {
                // otherwise keep panicking.
                panic(v)
            }
        }

    }()
    fmt.Println("  cmd.Run(req, re, env)")
    cmd.Run(req, re, env)
    return nil
}
```

2.8.2 源码补充说明

NODE2.8.2-1

2.9 时序图第六步 daemon.go（Run()）

2.9.1 源码初步说明

NODE2.9.1-1

```
func daemonFunc(req *cmds.Request, re cmds.ResponseEmitter, env cmds.Environment) {
   // Inject metrics before we do anything
   err := mprome.Inject()
   if err != nil {
      log.Errorf("Injecting prometheus handler for metrics failed with message: %s\n",
err.Error())
   }
   // let the user know we're going.
   fmt.Printf("Initializing daemon...\n")

   managefd, _ := req.Options[adjustFDLimitKwd].(bool)
   if managefd {
      if err := utilmain.ManageFdLimit(); err != nil {
         log.Errorf("setting file descriptor limit: %s", err)
      }
   }
  //编号 1 获得环境变量
   cctx := env.(*oldcmds.Context)

   go func() {
      <-req.Context.Done()
      fmt.Println("Received interrupt signal, shutting down...")
      fmt.Println("(Hit ctrl-c again to force-shutdown the daemon.)")
   }()

   // check transport encryption flag.
   unencrypted, _ := req.Options[unencryptTransportKwd].(bool)
   if unencrypted {
      log.Warningf(`Running with --%s: All connections are UNENCRYPTED.
      You will not be able to connect to regular encrypted networks.`, unencryptTransportKwd)
   }

   // first, whether user has provided the initialization flag. we may be
   // running in an uninitialized state.
   initialize, _ := req.Options[initOptionKwd].(bool)
   fmt.Println("initialize:",initialize)
   if initialize {
    cfg := cctx.ConfigRoot
    if !fsrepo.IsInitialized(cfg) {
        profiles, _ := req.Options[initProfileOptionKwd].(string)

        err := initWithDefaults(os.Stdout, cfg, profiles)
        if err != nil {
            re.SetError(err, cmdkit.ErrNormal)
            return
        }
    }
}

// acquire the repo lock _before_ constructing a node. we need to make
// sure we are permitted to access the resources (datastore, etc.)
fmt.Println("cctx.ConfigRoot",cctx.ConfigRoot)
```

//编号 2 获得 Repo 文件配置

```
repo, err := fsrepo.Open(cctx.ConfigRoot)
switch err {
default:
    fmt.Println("case default")
    re.SetError(err, cmdkit.ErrNormal)
    return
case fsrepo.ErrNeedMigration:
    fmt.Println("case fsrepo.ErrNeedMigration:")
    domigrate, found := req.Options[migrateKwd].(bool)
    fmt.Println("Found outdated fs-repo, migrations need to be run.")

    if !found {
        domigrate = YesNoPrompt("Run migrations now? [y/N]")
    }

    if !domigrate {
        fmt.Println("Not running migrations of fs-repo now.")
        fmt.Println("Please get fs-repo-migrations from https://dist.ipfs.io")
        re.SetError(fmt.Errorf("fs-repo requires migration"), cmdkit.ErrNormal)
        return
    }

    err = migrate.RunMigration(fsrepo.RepoVersion)
    if err != nil {
        fmt.Println("The migrations of fs-repo failed:")
        fmt.Printf("  %s\n", err)
        fmt.Println("If you think this is a bug, please file an issue and include this whole
log output.")
         fmt.Println("  https://github.com/ipfs/fs-repo-migrations")
         re.SetError(err, cmdkit.ErrNormal)
         return
      }

      repo, err = fsrepo.Open(cctx.ConfigRoot)
      if err != nil {
         re.SetError(err, cmdkit.ErrNormal)
         return
      }
   case nil:
      fmt.Println("case nil:")
      break
   }
```

//编号 3 获得配置文件

```
     cfg, err := cctx.GetConfig()
   if err != nil {
      re.SetError(err, cmdkit.ErrNormal)
      return
   }

   offline, _ := req.Options[offlineKwd].(bool)
   ipnsps, _ := req.Options[enableIPNSPubSubKwd].(bool)
   pubsub, _ := req.Options[enableFloodSubKwd].(bool)
   mplex, _ := req.Options[enableMultiplexKwd].(bool)
   fmt.Println("offline",offline)
   fmt.Println("ipnsps",ipnsps)
   fmt.Println("pubsub",pubsub)
   fmt.Println("mplex",mplex)
   // Start assembling node config
```

//编号 4 获得配置文件 生成 ipfs 节点需要的配置

```
   ncfg := &core.BuildCfg{
      Repo:      repo,
      Permanent: true, // It is temporary way to signify that node is permanent
      Online:    !offline,
      DisableEncryptedConnections: unencrypted,
      ExtraOpts: map[string]bool{
         "pubsub": pubsub,
         "ipnsps": ipnsps,
         "mplex":  mplex,
      },
      //TODO(Kubuxu): refactor Online vs Offline by adding Permanent vs Ephemeral
   }
```

```
    routingOption, _ := req.Options[routingOptionKwd].(string)
    fmt.Println("routingOption",routingOption)
    if routingOption == routingOptionDefaultKwd {
        cfg, err := repo.Config()
        if err != nil {
            re.SetError(err, cmdkit.ErrNormal)
            return
        }

        routingOption = cfg.Routing.Type
        if routingOption == "" {
            routingOption = routingOptionDHTKwd
        }
    }
    switch routingOption {
    case routingOptionSupernodeKwd:
        re.SetError(errors.New("supernode routing was never fully implemented and has been
removed"), cmdkit.ErrNormal)
        return
    case routingOptionDHTClientKwd:
        ncfg.Routing = core.DHTClientOption
    case routingOptionDHTKwd:
        ncfg.Routing = core.DHTOption
    case routingOptionNoneKwd:
        ncfg.Routing = core.NilRouterOption
    default:
        re.SetError(fmt.Errorf("unrecognized routing option: %s", routingOption),
cmdkit.ErrNormal)
        return
    }
```

//编号 5 生成一个新的 IPFS NODE

```
    node, err := core.NewNode(req.Context, ncfg)
    if err != nil {
        log.Error("error from node construction: ", err)
        re.SetError(err, cmdkit.ErrNormal)
        return
    }
    fmt.Println("  node.SetLocal(false)")
    node.SetLocal(false)

    if node.PNetFingerprint != nil {
        fmt.Println("Swarm is limited to private network of peers with the swarm key")
        fmt.Printf("Swarm key fingerprint: %x\n", node.PNetFingerprint)
```

```
}
fmt.Println("  printSwarmAddrs(node)")
printSwarmAddrs(node)

defer func() {
   // We wait for the node to close first, as the node has children
   // that it will wait for before closing, such as the API server.
   node.Close()

   select {
   case <-req.Context.Done():
      log.Info("Gracefully shut down daemon")
   default:
   }
}()

cctx.ConstructNode = func() (*core.IpfsNode, error) {
   return node, nil
}
```

//编号 6 HTTP 服务端端口建立和监听

```
apiErrc, err := serveHTTPApi(req, cctx)
if err != nil {
   re.SetError(err, cmdkit.ErrNormal)
   return
}

// construct fuse mountpoints - if the user provided the --mount flag
mount, _ := req.Options[mountKwd].(bool)
fmt.Println("mount",mount)
if mount && offline {
   re.SetError(errors.New("mount is not currently supported in offline mode"),
      cmdkit.ErrClient)
   return
}

if mount {
   if err := mountFuse(req, cctx); err != nil {
      re.SetError(err, cmdkit.ErrNormal)
      return
   }
}
```

//编号 7 Gc 管理

```
	gcErrc, err := maybeRunGC(req, node)
	if err != nil {
		re.SetError(err, cmdkit.ErrNormal)
		return
	}

	// construct http gateway - if it is set in the config
	var gwErrc <-chan error
	if len(cfg.Addresses.Gateway) > 0 {
		fmt.Println("len(cfg.Addresses.Gateway) > 0")
		var err error
	//编号 8 serveHTTPGateway 服务建立和监听
		gwErrc, err = serveHTTPGateway(req, cctx)
		if err != nil {
			re.SetError(err, cmdkit.ErrNormal)
			return
		}
	}

	// initialize metrics collector
	fmt.Println("  prometheus.MustRegister(&corehttp.IpfsNodeCollector{Node: node})")
	prometheus.MustRegister(&corehttp.IpfsNodeCollector{Node: node})

	fmt.Printf("Daemon is ready\n")
	// collect long-running errors and block for shutdown
	// TODO(cryptix): our fuse currently doesnt follow this pattern for graceful shutdown
	for err := range merge(apiErrc, gwErrc, gcErrc) {
		if err != nil {
			log.Error(err)
			re.SetError(err, cmdkit.ErrNormal)
		}
	}
}
```

2.9.2 源码补充说明

NODE2.9.2-1

2.10 时序图第六步 daemon.go（fsrepo.Open()）

2.10.1 源码初步说明

NODE2.10.1-1

```
cctx := env.(*oldcmds.Context)
repo, err = fsrepo.Open(cctx.ConfigRoot)

func Open(repoPath string) (repo.Repo, error) {
   fn := func() (repo.Repo, error) {
      return open(repoPath)
   }
   return onlyOne.Open(repoPath, fn)
}
```

2.10.2 源码补充说明

NODE2.10.2-1

```
func (o *OnlyOne) Open(key interface{}, open func() (Repo, error)) (Repo, error) {
   o.mu.Lock()
   defer o.mu.Unlock()
   if o.active == nil {
      o.active = make(map[interface{}]*ref)
   }

   item, found := o.active[key]
   if !found {
```

```go
		repo, err := open()
		if err != nil {
			return nil, err
		}
		item = &ref{
			parent: o,
			key:    key,
			Repo:   repo,
		}
		o.active[key] = item
	}
	item.refs++
	return item, nil
}
```

NODE2.10.2-3

```go
type ref struct {
	parent *OnlyOne
	key    interface{}
	refs   uint32
	Repo
}

type FSRepo struct {
	// has Close been called already
	closed bool
	// path is the file-system path
	path string
	// lockfile is the file system lock to prevent others from opening
	// the same fsrepo path concurrently
	lockfile io.Closer
	config   *config.Config
	ds       repo.Datastore
	keystore keystore.Keystore
	filemgr  *filestore.FileManager
}
```

NODE2.10.2-4

onlyOne repo.OnlyOne 是一个全局变量

```go
// open one.
type OnlyOne struct {
	mu     sync.Mutex
```

```
    active map[interface{}]*ref
}
```

NODE2.10.2-5

```
func open(repoPath string) (repo.Repo, error) {
    packageLock.Lock()
    defer packageLock.Unlock()

    r, err := newFSRepo(repoPath)
    fmt.Println("  r, err := newFSRepo(repoPath):",repoPath)
    if err != nil {
        return nil, err
    }

    // Check if its initialized
    if err := checkInitialized(r.path); err != nil {
        return nil, err
    }

    r.lockfile, err = lockfile.Lock(r.path, LockFile)
    if err != nil {
        return nil, err
    }
    keepLocked := false
    defer func() {
        // unlock on error, leave it locked on success
        if !keepLocked {
            r.lockfile.Close()
        }
    }()

    ver, err := mfsr.RepoPath(r.path).Version()
    if err != nil {
        if os.IsNotExist(err) {
            return nil, ErrNoVersion
        }
        return nil, err
    }

    if RepoVersion > ver {
        return nil, ErrNeedMigration
    } else if ver > RepoVersion {
        // program version too low for existing repo
        return nil, fmt.Errorf(programTooLowMessage, RepoVersion, ver)
    }

    // check repo path, then check all constituent parts.
    if err := dir.Writable(r.path); err != nil {
        return nil, err
    }

    if err := r.openConfig(); err != nil {
        return nil, err
    }

    if err := r.openDatastore(); err != nil {
        return nil, err
    }

    if err := r.openKeystore(); err != nil {
        return nil, err
    }

    if r.config.Experimental.FilestoreEnabled || r.config.Experimental.UrlstoreEnabled {
        r.filemgr = filestore.NewFileManager(r.ds, filepath.Dir(r.path))
        r.filemgr.AllowFiles = r.config.Experimental.FilestoreEnabled
        r.filemgr.AllowUrls = r.config.Experimental.UrlstoreEnabled
    }

    keepLocked = true
    return r, nil
}
```

2.11 时序图第六步 daemon.go（cctx.getConfig()）

2.11.1 源码初步说明

NODE2.11.1-1

```
func (c *Context) GetConfig() (*config.Config, error) {
    var err error
    fmt.Println("(c *Context) GetConfig() (*config.Config, error)")
    if c.config == nil {
        if c.LoadConfig == nil {
            return nil, errors.New("nil LoadConfig function")
        }
        c.config, err = c.LoadConfig(c.ConfigRoot)
        fmt.Println("c.config, err = c.LoadConfig(c.ConfigRoot)",c.ConfigRoot)
    }
    return c.config, err
}
```

2.11.2 源码补充说明

NODE2.11.2-1

```
func loadConfig(path string) (*config.Config, error) {
    return fsrepo.ConfigAt(path)
}
```

NODE2.11.2-2

```
func ConfigAt(repoPath string) (*config.Config, error) {

    // packageLock must be held to ensure that the Read is atomic.
    packageLock.Lock()
    defer packageLock.Unlock()

    configFilename, err := config.Filename(repoPath)
    fmt.Println("configFilename",configFilename)
    if err != nil {
        return nil, err
    }
    return serialize.Load(configFilename)
}
```

NODE2.11.2-3

```
// Load reads given file and returns the read config, or error.
func Load(filename string) (*config.Config, error) {
    // if nothing is there, fail. User must run 'ipfs init'
    if !util.FileExists(filename) {
        return nil, errors.New("ipfs not initialized, please run 'ipfs init'")
    }

    var cfg config.Config
    err := ReadConfigFile(filename, &cfg)
    if err != nil {
        return nil, err
    }
    return &cfg, err
}
```

NODE2.11.2-4

```
// ReadConfigFile reads the config from `filename` into `cfg`.
func ReadConfigFile(filename string, cfg interface{}) error {
    f, err := os.Open(filename)
    if err != nil {
        return err
    }
    defer f.Close()
    if err := json.NewDecoder(f).Decode(cfg); err != nil {
        return fmt.Errorf("failure to decode config: %s", err)
    }
    return nil
}
```

说明;IPFS 解析配置文件的方式是 JSON 文件格式 + 对象解析 + Json 方式

2.12 时序图第十二步 daemon.go（core.BuildCfg()）

2.12.1 源码初步说明

NODE2.12.1-1

```
ncfg := &core.BuildCfg{
    Repo:       repo,
    Permanent: true, // It is temporary way to signify that node is permanent
    Online:     !offline,
    DisableEncryptedConnections: unencrypted,
    ExtraOpts: map[string]bool{
        "pubsub": pubsub,
        "ipnsps": ipnsps,
        "mplex":  mplex,
    },
    //TODO(Kubuxu): refactor Online vs Offline by adding Permanent vs Ephemeral
}
```

2.12.2 源码补充说明

NODE2.12.2-1

```
offline, _ := req.Options[offlineKwd].(bool)  //false
ipnsps, _ := req.Options[enableIPNSPubSubKwd].(bool)  //false
pubsub, _ := req.Options[enableFloodSubKwd].(bool) //false
mplex, _ := req.Options[enableMultiplexKwd].(bool) //true
```

说明：根据 opention 程序初始化代码，很容易分析参数内容

2.13 时序图第十三步 daemon.go（core.NewNode()）

2.13.1 源码初步说明

NODE2.11.1-1

```
// NewNode constructs and returns an IpfsNode using the given cfg.
func NewNode(ctx context.Context, cfg *BuildCfg) (*IpfsNode, error) {
   if cfg == nil {
      cfg = new(BuildCfg)
   }
    //编号 1 cfg 设置默认值
   err := cfg.fillDefaults()
   if err != nil {
      return nil, err
   }
   fmt.Println("  ctx = metrics.CtxScope(ctx,)")
//编号 2  获得 IPFS 需要的环境变量
   ctx = metrics.CtxScope(ctx, "ipfs")
//编号 3   创建 Ipfs 节点
   n := &IpfsNode{
      mode:      offlineMode,
      Repo:      cfg.Repo,
      ctx:       ctx,
      Peerstore: pstore.NewPeerstore(),
   }
   fmt.Println("  n.RecordValidator = record.NamespacedValidator{")
   n.RecordValidator = record.NamespacedValidator{
      "pk":   record.PublicKeyValidator{},
      "ipns": ipns.Validator{KeyBook: n.Peerstore},
   }
```

//编号 4　IPFSNode 网络模式本地还是网络节点

```
    if cfg.Online {
        n.mode = onlineMode
    }
```

//编号 5 设置 IPFS proc

```
    n.proc = goprocessctx.WithContextAndTeardown(ctx, n.teardown)
```

//编号 6 设置 IPFSNode 很多细节

```
    if err := setupNode(ctx, n, cfg); err != nil {
        n.Close()
        return nil, err
    }
    fmt.Println("  return n, nil---------------------------")
    return n, nil
}
```

2.13.2 源码补充说明

NODE2.13.2-1

2.14 时序图第六步 daemon.go（serveHTTPApi()）

2.14.1 源码初步说明

NODE2.14.1-1

```
func serveHTTPApi(req *cmds.Request, cctx *oldcmds.Context) (<-chan error, error) {
```

//编号 1 获得配置

```
   cfg, err := cctx.GetConfig()
   if err != nil {
      return nil, fmt.Errorf("serveHTTPApi: GetConfig() failed: %s", err)
   }

   apiAddr, _ := req.Options[commands.ApiOption].(string)
   fmt.Println("apiAddr:= req.Options[commands.ApiOption].(string)",apiAddr)
   if apiAddr == "" {
      fmt.Println("apiAddr == ")
```

//编号 2 获得配置　默认为配置文件的 ip 地址和端口 服务端监听的 ip 和端口

```
      apiAddr = cfg.Addresses.API
      fmt.Println("apiAddr = cfg.Addresses.API ",apiAddr)
   }
```

//编号 3 将/ip4/192.168.8.126/tcp/5001 转为[]byte,然后生成 Multiaddr

```
      apiMaddr, err := ma.NewMultiaddr(apiAddr)
      fmt.Println("  apiMaddr, err := ma.NewMultiaddr(apiAddr)",apiMaddr)
      if err != nil {
         return nil, fmt.Errorf("serveHTTPApi: invalid API address: %q (err: %s)", apiAddr,
    err)
      }
```

//编号 4 ipfs 服务节点监听

```
      apiLis, err := manet.Listen(apiMaddr)
      if err != nil {
         return nil, fmt.Errorf("serveHTTPApi: manet.Listen(%s) failed: %s", apiMaddr, err)
      }
      // we might have listened to /tcp/0 - lets see what we are listing on
      apiMaddr = apiLis.Multiaddr()
      fmt.Printf("API server listening on %s\n", apiMaddr)
```

```go
    // by default, we don't let you load arbitrary ipfs objects through the api,
    // because this would open up the api to scripting vulnerabilities.
    // only the webui objects are allowed.
    // if you know what you're doing, go ahead and pass --unrestricted-api.
    unrestricted, _ := req.Options[unrestrictedApiAccessKwd].(bool)
    gatewayOpt := corehttp.GatewayOption(false, corehttp.WebUIPaths...)
    if unrestricted {
        gatewayOpt = corehttp.GatewayOption(true, "/ipfs", "/ipns")
    }
//编号 5 http 配置选项，
    var opts = []corehttp.ServeOption{
        corehttp.MetricsCollectionOption("api"),
        corehttp.CheckVersionOption(),
        corehttp.CommandsOption(*cctx),//关联了 daemon 节点响应的 Run
        corehttp.WebUIOption,
        gatewayOpt,
        corehttp.VersionOption(),
        defaultMux("/debug/vars"),
        defaultMux("/debug/pprof/"),
        corehttp.MetricsScrapingOption("/debug/metrics/prometheus"),
        corehttp.LogOption(),
    }

    if len(cfg.Gateway.RootRedirect) > 0 {
        opts = append(opts, corehttp.RedirectOption("", cfg.Gateway.RootRedirect))
    }
    //编号 6  创建服务响应 IPFSNODE
    node, err := cctx.ConstructNode()
    if err != nil {
        return nil, fmt.Errorf("serveHTTPApi: ConstructNode() failed: %s", err)
    }
//编号 7  设置服务监听地址
    if err := node.Repo.SetAPIAddr(apiMaddr); err != nil {
        return nil, fmt.Errorf("serveHTTPApi: SetAPIAddr() failed: %s", err)
    }

    errc := make(chan error)
    go func() {
     //编号 8  同行服务节点
      errc <- corehttp.Serve(node, manet.NetListener(apiLis), opts...)
      close(errc)
   }()
   return errc, nil
}
```

2.14.2 源码补充说明

NODE2.14.2-1

2.15 时序图第六步 daemon.go（maybeRunGC()）

2.15.1 源码初步说明

NODE2.15.1-1

```
gcErrc, err := maybeRunGC(req, node)

func maybeRunGC(req *cmds.Request, node *core.IpfsNode) (<-chan error, error) {
   enableGC, _ := req.Options[enableGCKwd].(bool)
   if !enableGC {
      return nil, nil
   }

   errc := make(chan error)
   go func() {
//编号 1 启用自动定期回收 Repo 垃圾
      errc <- corerepo.PeriodicGC(req.Context, node)
      close(errc)
   }()
   return errc, nil
}
```

2.15.2 源码补充说明

NODE2.15.2-1

2.16 时序图第十六步 daemon.go（serveHTTPGateway()）

2.16.1 源码初步说明

NODE2.16.1-1

```
// serveHTTPGateway collects options, creates listener, prints status message and starts serving
requests
func serveHTTPGateway(req *cmds.Request, cctx *oldcmds.Context) (<-chan error, error) {
   cfg, err := cctx.GetConfig()
   if err != nil {
      return nil, fmt.Errorf("serveHTTPGateway: GetConfig() failed: %s", err)
   }

   gatewayMaddr, err := ma.NewMultiaddr(cfg.Addresses.Gateway)
   if err != nil {
      return nil, fmt.Errorf("serveHTTPGateway: invalid gateway address: %q (err: %s)",
cfg.Addresses.Gateway, err)
   }

   writable, writableOptionFound := req.Options[writableKwd].(bool)
   if !writableOptionFound {
      writable = cfg.Gateway.Writable
   }

   gwLis, err := manet.Listen(gatewayMaddr)
   if err != nil {
      return nil, fmt.Errorf("serveHTTPGateway: manet.Listen(%s) failed: %s", gatewayMaddr, err)
   }
   // we might have listened to /tcp/0 - lets see what we are listing on
   gatewayMaddr = gwLis.Multiaddr()

   if writable {
      fmt.Printf("Gateway (writable) server listening on %s\n", gatewayMaddr)
   } else {
      fmt.Printf("Gateway (readonly) server listening on %s\n", gatewayMaddr)
   }

   var opts = []corehttp.ServeOption{
        corehttp.MetricsCollectionOption("gateway"),
        corehttp.CheckVersionOption(),
        corehttp.CommandsROOption(*cctx),   //ROroot
        corehttp.VersionOption(),
        corehttp.IPNSHostnameOption(),
        corehttp.GatewayOption(writable, "/ipfs", "/ipns"),
    }

    if len(cfg.Gateway.RootRedirect) > 0 {
        opts = append(opts, corehttp.RedirectOption("", cfg.Gateway.RootRedirect))
    }

    node, err := cctx.ConstructNode()
    if err != nil {
        return nil, fmt.Errorf("serveHTTPGateway: ConstructNode() failed: %s", err)
    }

    errc := make(chan error)
    go func() {
        errc <- corehttp.Serve(node, manet.NetListener(gwLis), opts...)
        close(errc)
    }()
    return errc, nil
}
```

2.16.2 源码补充说明

NODE2.16.2-1

2.17 时序图第十七步 daemon.go（serveHTTPApi()）

2.17.1 源码初步说明

NODE2.14.1-1

```go
func Serve(node *core.IpfsNode, lis net.Listener, options ...ServeOption) error {
   // make sure we close this no matter what.
   defer lis.Close()

   handler, err := makeHandler(node, lis, options...)
   if err != nil {
      return err
   }

   addr, err := manet.FromNetAddr(lis.Addr())
   if err != nil {
      return err
   }

   select {
   case <-node.Process().Closing():
      return fmt.Errorf("failed to start server, process closing")
   default:
   }

   server := &http.Server{
      Handler: handler,
   }

   var serverError error
   serverProc := node.Process().Go(func(p goprocess.Process) {
      serverError = server.Serve(lis)
   })

   // wait for server to exit.
   select {
   case <-serverProc.Closed():
   // if node being closed before server exits, close server
   case <-node.Process().Closing():
      log.Infof("server at %s terminating...", addr)

      warnProc := periodicproc.Tick(5*time.Second, func(_ goprocess.Process) {
         log.Infof("waiting for server at %s to terminate...", addr)
```

```
        })

        // This timeout shouldn't be necessary if all of our commands
        // are obeying their contexts but we should have *some* timeout.
        ctx, cancel := context.WithTimeout(context.Background(), shutdownTimeout)
        defer cancel()
        err := server.Shutdown(ctx)

        // Should have already closed but we still need to wait for it
        // to set the error.
        <-serverProc.Closed()
        serverError = err

        warnProc.Close()
    }

    log.Infof("server at %s terminated", addr)
    return serverError
}
```

2.17.2 源码补充说明

NODE2.14.2-1

三. IPFS ADD 流程

3.1 IPFS daemon 初始化时序图

3.1.1 demon 时序图如图 3-1

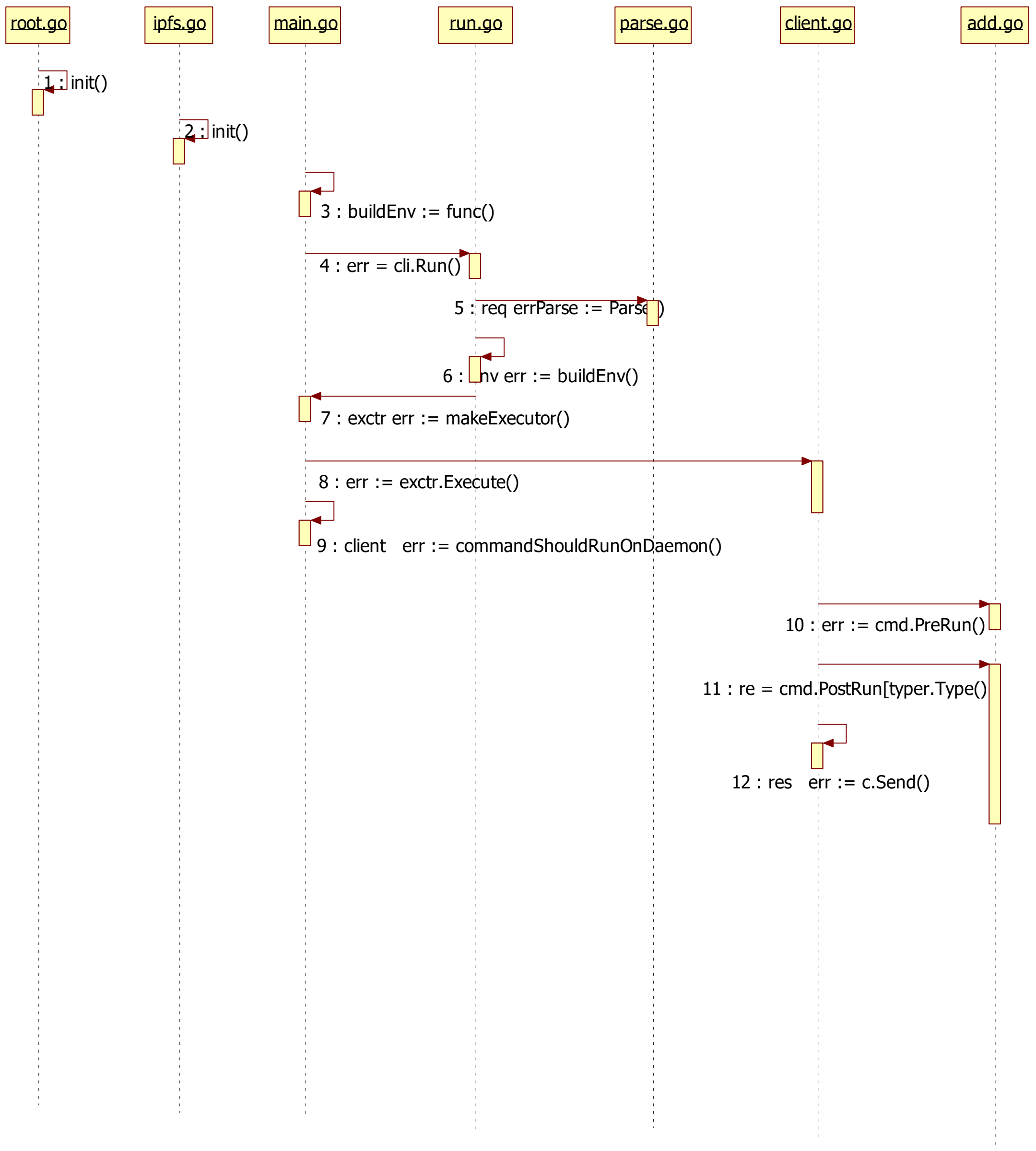

root.go
ipfs.go
main.go
run.go
parse.go
client.go
add.go
1 : init()
2 : init()
3 : buildEnv := func()
4 : err = cli.Run()
5 : req errParse := Parse()
6 : env err := buildEnv()
7 : exctr err := makeExecutor()
8 : err := exctr.Execute()
9 : client err := commandShouldRunOnDaemon()
10 : err := cmd.PreRun()
11 : re = cmd.PostRun[typer.Type()
12 : res err := c.Send()

图 3-1

3.1.2 时序图说明

所有的命令前七部在初始化流程都是一致，从第八部开始就会产生差异，执行各自的使命，daemon 已经说明了前七部，那么现在从第八步说明，第九步顺序在第八步之前，

3.2 时序图第八步 main.go（commanShouldRunOnDaemon()）

3.2.1 源码初步说明

NODE3.2.1-1

```
func commandShouldRunOnDaemon(details cmdDetails, req *cmds.Request, cctx *oldcmds.Context) (http.Client, error) {
    path := req.Path
    // root command.
    if len(path) < 1 {
        return nil, nil
    }
      if details.cannotRunOnClient && details.cannotRunOnDaemon {
        return nil, fmt.Errorf("command disabled: %s", path[0])
    }

    if details.doesNotUseRepo && details.canRunOnClient() {
        return nil, nil
    }
    apiAddrStr, _ := req.Options[corecmds.ApiOption].(string)
    //编号 1 返回一般正常
    client, err := getApiClient(cctx.ConfigRoot, apiAddrStr)

    if err == repo.ErrApiNotRunning {
        fmt.Println("if err == repo.ErrApiNotRunning {")
        if apiAddrStr != "" && req.Command != daemonCmd {
            // if user SPECIFIED an api, and this cmd is not daemon
            // we MUST use it. so error out.
            return nil, err
        }

        // ok for api not to be running
    } else if err != nil { // some other api error
        return nil, err
    }

    if client != nil {
        fmt.Println("if client != nil {")
        if details.cannotRunOnDaemon {
    //编号 2  跟传入参数有关
            fmt.Println("  if details.cannotRunOnDaemon {")
            // check if daemon locked. legacy error text, for now.
            log.Debugf("Command cannot run on daemon. Checking if daemon is locked")
            if daemonLocked, _ := fsrepo.LockedByOtherProcess(cctx.ConfigRoot); daemonLocked {
                log.Debugf("    if daemonLocked, _ := fsrepo.LockedByOtherProcess(cctx.ConfigRoot);
daemonLocked {")
                return nil, cmds.ClientError("ipfs daemon is running. please stop it to run this
command")
            }
            return nil, nil
        }
        fmt.Println("return client, nil")
          //编号 3  如果是命令参数则返回 client 实例化
        return client, nil
    }

    if details.cannotRunOnClient {
        fmt.Println("  if details.cannotRunOnClient {")
        return nil, cmds.ClientError("must run on the ipfs daemon")
    }
    fmt.Println("  return nil, nil")
    return nil, nil
}
```

3.2.2 源码补充说明

NODE3.2.2-1

```
var AddCmd = &cmds.Command{
    Helptext: cmdkit.HelpText{
        Tagline: "Add a file or directory to ipfs.",
        ShortDescription: `
Adds contents of <path> to ipfs. Use -r to add directories (recursively).
`,
        LongDescription: `
Adds contents of <path> to ipfs. Use -r to add directories.
。。。。。。。。。。。。。。。。。。。。。。。。。
`,
    },
   Arguments: []cmdkit.Argument{
      cmdkit.FileArg("path", true, true, "The path to a file to be added to
ipfs.").EnableRecursive().EnableStdin(),
   },
   Options: []cmdkit.Option{

      cmdkit.StringOption(chunkerOptionName, "s", "Chunking algorithm, size-[bytes] or
rabin-[min]-[avg]-[max]").WithDefault("size-262144"),
      cmdkit.BoolOption(pinOptionName, "Pin this object when adding.").WithDefault(true),
      cmdkit.BoolOption(rawLeavesOptionName, "Use raw blocks for leaf nodes. (experimental)"),
      cmdkit.BoolOption(noCopyOptionName, "Add the file using filestore. Implies raw-leaves.
(experimental)"),
      cmdkit.BoolOption(fstoreCacheOptionName, "Check the filestore for pre-existing blocks.
(experimental)"),
      cmdkit.IntOption(cidVersionOptionName, "CID version. Defaults to 0 unless an option that
depends on CIDv1 is passed. (experimental)"),
      cmdkit.StringOption(hashOptionName, "Hash function to use. Implies CIDv1 if not sha2-256.
(experimental)").WithDefault("sha2-256"),
   },

 PreRun: func(req *cmds.Request, env cmds.Environment) error {
      fmt.Print("PreRun: func(\n" )
          return nil
   },

 Run: func(req *cmds.Request, res cmds.ResponseEmitter, env cmds.Environment) {

       },

 PostRun: cmds.PostRunMap{
      cmds.CLI: func(req *cmds.Request, re cmds.ResponseEmitter) cmds.ResponseEmitter {
     },

 Type: coreunix.AddedObject{},

}
```

说明 这也是一种解耦方式，新增的命令关联较小，每个命令都与自身有关，不会与其他命令相关联

3.3 时序图第九步 client.go（client.Execute()）

3.3.1 源码初步说明

NODE3.3.1-1

```
func (c *client) Execute(req *cmds.Request, re cmds.ResponseEmitter, env cmds.Environment) error
{
  //编号 1   获得 req 传入的命令，如执行 add  则此处为 add 命令实例
    cmd := req.Command
    // If this ResponseEmitter encodes messages (e.g. http, cli or writer - but not chan),
    // we need to update the encoding to the one specified by the command.
    if ee, ok := re.(cmds.EncodingEmitter); ok {
        encType := cmds.GetEncoding(req)
        // note the difference: cmd.Encoders vs. cmds.Encoders
        if enc, ok := cmd.Encoders[encType]; ok {

            ee.SetEncoder(enc(req))
        } else if enc, ok := cmds.Encoders[encType]; ok {
          //编号 2   设置 req 内部参数的编码
            ee.SetEncoder(enc(req))
        } else {
            ee.SetEncoder(cmds.Encoders[cmds.JSON](req))
        }
    }
 //编号 3   如果此命令初始化有 PreRun 方法，则执行 PreRun, add 有此方法  部分命令没有此方法
    if cmd.PreRun != nil {
        println("cmd.PreRun(req, env)")
        err := cmd.PreRun(req, env)
        if err != nil {
            return err
        }
    }
//编号 4   如果此命令初始化有 PostRun 方法,则执行 PostRun, add 有此方法  部分命令没有此方法

    if cmd.PostRun != nil {
        println("cmd.PostRun != nil ")
        if typer, ok := re.(interface {
            Type() cmds.PostRunType
        }); ok && cmd.PostRun[typer.Type()] != nil {
            println("re = cmd.PostRun[typer.Type()](req, re) ")
            re = cmd.PostRun[typer.Type()](req, re)
        }
    }
    println("c.Send(req) c.Send(req)c.Send(req)c.Send(req)c.Send(req)c.Send(req)")
```

//编号 5　向 ipfs 服务节点发送命令

```
  res, err := c.Send(req)
    println("c.Send(req) after c.Send(req) after c.Send(req) after c.Send(req) after c.Send(req)
after c.Send(req) after c.Send(req) after c.Send(req) after ")
    if err != nil {
        if isConnRefused(err) {
            err = ErrAPINotRunning
        }
        return err
    }
    println(" cmds.Copy(re, res)")
    return cmds.Copy(re, res)
}
```

3.3.2 源码补充说明

NODE3.3.2-1

3.4 时序图第十步 add.go（PreRun()）

3.4.1 源码初步说明

NODE3.4.1-1

```
PreRun: func(req *cmds.Request, env cmds.Environment) error {
    fmt.Print("PreRun: func(\n" )
    quiet, _ := req.Options[quietOptionName].(bool)
    quieter, _ := req.Options[quieterOptionName].(bool)
    quiet = quiet || quieter

    silent, _ := req.Options[silentOptionName].(bool)

    if quiet || silent {
        return nil
    }
```

```
   // ipfs cli progress bar defaults to true unless quiet or silent is used
   _, found := req.Options[progressOptionName].(bool)
   if !found {
      req.Options[progressOptionName] = true
   }

   return nil
},
```
说明　参数的预处理

3.4.2 源码补充说明

NODE3.4.2-1

3.5 时序图第八步 add.go（client.PostRun()）

3.5.1 源码初步说明

NODE3.5.1-1

```
PostRun: cmds.PostRunMap{
   cmds.CLI: func(req *cmds.Request, re cmds.ResponseEmitter) cmds.ResponseEmitter {
      fmt.Print("PostRun:
cmds.PostRunMap--------------------------------------------------------------------\n" )
      reNext, res := cmds.NewChanResponsePair(req)
      outChan := make(chan interface{})

      sizeChan := make(chan int64, 1)

      sizeFile, ok := req.Files.(files.SizeFile)
      if ok {
         fmt.Println("  sizeFile, ok := req.Files.(files.SizeFile)")
         // Could be slow.
         go func() {
                size, err := sizeFile.Size()
                fmt.Println("size, err := sizeFile.Size()",size)
                if err != nil {
                    log.Warningf("error getting files size: %s", err)
                    // see comment above
                    return
                }

                sizeChan <- size
            }()
        } else {
            // we don't need to error, the progress bar just
            // won't know how big the files are
            log.Warning("cannot determine size of input file")
        }

        progressBar := func(wait chan struct{}) {
            fmt.Println("progressBar := func(wait chan struct{})-------------------------------
-------------------------------------------------- {")
            defer close(wait)

            quiet, _ := req.Options[quietOptionName].(bool)
            fmt.Println("quiet",quiet)
            quieter, _ := req.Options[quieterOptionName].(bool)
            quiet = quiet || quieter

            progress, _ := req.Options[progressOptionName].(bool)
            fmt.Println("quieter,quiet||quieter,progress",quieter,quiet,progress)
            var bar *pb.ProgressBar
            if progress {
                fmt.Println("  if progress {")
                bar = pb.New64(0).SetUnits(pb.U_BYTES)
                bar.ManualUpdate = true
                bar.ShowTimeLeft = false
                bar.ShowPercent = false
                bar.Output = os.Stderr
                bar.Start()
            }

            lastFile := ""
            lastHash := ""
            var totalProgress, prevFiles, lastBytes int64

        LOOP:
```

```
for {
   select {
   case out, ok := <-outChan:
      fmt.Println("case out, ok := <-outChan:")
      if !ok {
         fmt.Println("if quieter {")
         if quieter {
            fmt.Println("  fmt.Fprintln(os.Stdout, lastHash)")
            fmt.Fprintln(os.Stdout, lastHash)
         }

         break LOOP
      }
      output := out.(*coreunix.AddedObject)
      fmt.Println("output.Name",output.Name)
      fmt.Println("output.Hash",output.Hash)
      fmt.Println("output.Bytes",output.Bytes)
      fmt.Println("output.Size",output.Size)

      if len(output.Hash) > 0 {
         fmt.Println("  if len(output.Hash) > 0 {")
         lastHash = output.Hash
         if quieter {
            continue
         }

         if progress {
            // clear progress bar line before we print "added x" output
            fmt.Fprintf(os.Stderr, "\033[2K\r")
         }
         if quiet {
            fmt.Fprintf(os.Stdout, "%s\n", output.Hash)
         } else {
            fmt.Fprintf(os.Stdout, "added %s %s\n", output.Hash, output.Name)
         }

      } else {
         fmt.Println("if !progress {")
         if !progress {
            continue
         }

         if len(lastFile) == 0 {
            fmt.Println(" len(lastFile) == 0 ",lastFile)
```

```
                            lastFile = output.Name
                        }
                        if output.Name != lastFile || output.Bytes < lastBytes {
                            fmt.Println("output.Name != lastFile || output.Bytes < lastBytes ")
                            prevFiles += lastBytes
                            lastFile = output.Name
                        }
                        lastBytes = output.Bytes
                        delta := prevFiles + lastBytes - totalProgress
                        totalProgress = bar.Add64(delta)
                    }

                    if progress {
                        bar.Update()
                    }
                case size := <-sizeChan:
                    fmt.Println("case size := <-sizeChan:",size)
                    if progress {
                        bar.Total = size
                        bar.ShowPercent = true
                        bar.ShowBar = true
                        bar.ShowTimeLeft = true
                    }
                case <-req.Context.Done():
                    fmt.Println("case <-req.Context.Done():")
                    // don't set or print error here, that happens in the goroutine below
                    return
                }
            }
        }
        fmt.Println("go func go funcgo funcgo funcgo funcgo funcgo funcgo funcgo funcgo funcgo
funcgo func")
        go func() {
            fmt.Println("defer re.Close()")
            // defer order important! First close outChan, then wait for output to finish, then close
re
            defer re.Close()

            if e := res.Error(); e != nil {
                defer close(outChan)
                re.SetError(e.Message, e.Code)
                return
            }
            wait := make(chan struct{})
            fmt.Println("go progressBar(wait)")
            go progressBar(wait)

            defer func() { <-wait }()
            defer close(outChan)

            for {

               v, err := res.Next()
               fmt.Println("v, err := res.Next()")
               if !cmds.HandleError(err, res, re) {
                  break
               }

               select {
               case outChan <- v:
                  fmt.Println("case outChan <- v:")
               case <-req.Context.Done():
                  fmt.Println("  case <-req.Context.Done():")
                  re.SetError(req.Context.Err(), cmdkit.ErrNormal)
                  return
               }
            }
         }()

         return reNext
      },
   },
```

3.5.2 源码补充说明

NODE3.5.2-1

3.6 时序图第八步 client.go（client.send()）

3.6.1 源码初步说明

NODE3.6.1-1

```go
func (c *client) Send(req *cmds.Request) (cmds.Response, error) {

   if req.Context == nil {
      log.Warningf("no context set in request")
      req.Context = context.Background()
   }
   fmt.Println("(c *client) Send(req *cmds.Request) (cmds.Response, error) {")
   // save user-provided encoding
   previousUserProvidedEncoding, found := req.Options[cmds.EncLong].(string)

   // override with json to send to server
   req.SetOption(cmds.EncLong, cmds.JSON)
   fmt.Println("  req.SetOption(cmds.EncLong, cmds.JSON)2222222222222222222")
   // stream channel output
   req.SetOption(cmds.ChanOpt, true)
   fmt.Println("  query, err := getQuery(req)2222222222222222222")
   query, err := getQuery(req)
   if err != nil {
      return nil, err
   }

   var fileReader *files.MultiFileReader
   var reader io.Reader
   if bodyArgs := req.BodyArgs(); bodyArgs != nil {
      fmt.Println("bodyArgs",bodyArgs.Argument())
      // In the end, this wraps a file reader in a file reader.
      // However, such is life.
      fileReader = files.NewMultiFileReader(files.NewSliceFile("", "", []files.File{
         files.NewReaderFile("stdin", "", bodyArgs, nil),
      }), true)
      reader = fileReader
   } else if req.Files != nil {
      fileReader = files.NewMultiFileReader(req.Files, true)
      reader = fileReader
   }
```

```
    path := strings.Join(req.Path, "/")
    url := fmt.Sprintf(ApiUrlFormat, c.serverAddress, c.apiPrefix, path, query)

    httpReq, err := http.NewRequest("POST", url, reader)
    if err != nil {
        return nil, err
    }

    // TODO extract string consts?
    if fileReader != nil {
        httpReq.Header.Set(contentTypeHeader, "multipart/form-data;
boundary="+fileReader.Boundary())
    } else {
        httpReq.Header.Set(contentTypeHeader, applicationOctetStream)
    }
    httpReq.Header.Set(uaHeader, c.ua)

    httpReq = httpReq.WithContext(req.Context)
    httpReq.Close = true

    httpRes, err := c.httpClient.Do(httpReq)
    if err != nil {
        return nil, err
    }

    // using the overridden JSON encoding in request
    res, err := parseResponse(httpRes, req)
    if err != nil {
        return nil, err
    }

    if found && len(previousUserProvidedEncoding) > 0 {
        // reset to user provided encoding after sending request
        // NB: if user has provided an encoding but it is the empty string,
        // still leave it as JSON.
        req.SetOption(cmds.EncLong, previousUserProvidedEncoding)
    }

    return res, nil
}
```

3.6.2 源码补充说明

NODE3.2.2-1

四． IPFS Daemon 响应 ADD

3.1 IPFS daemon add 响应

3.1.1 demon 时序图如图 3-1

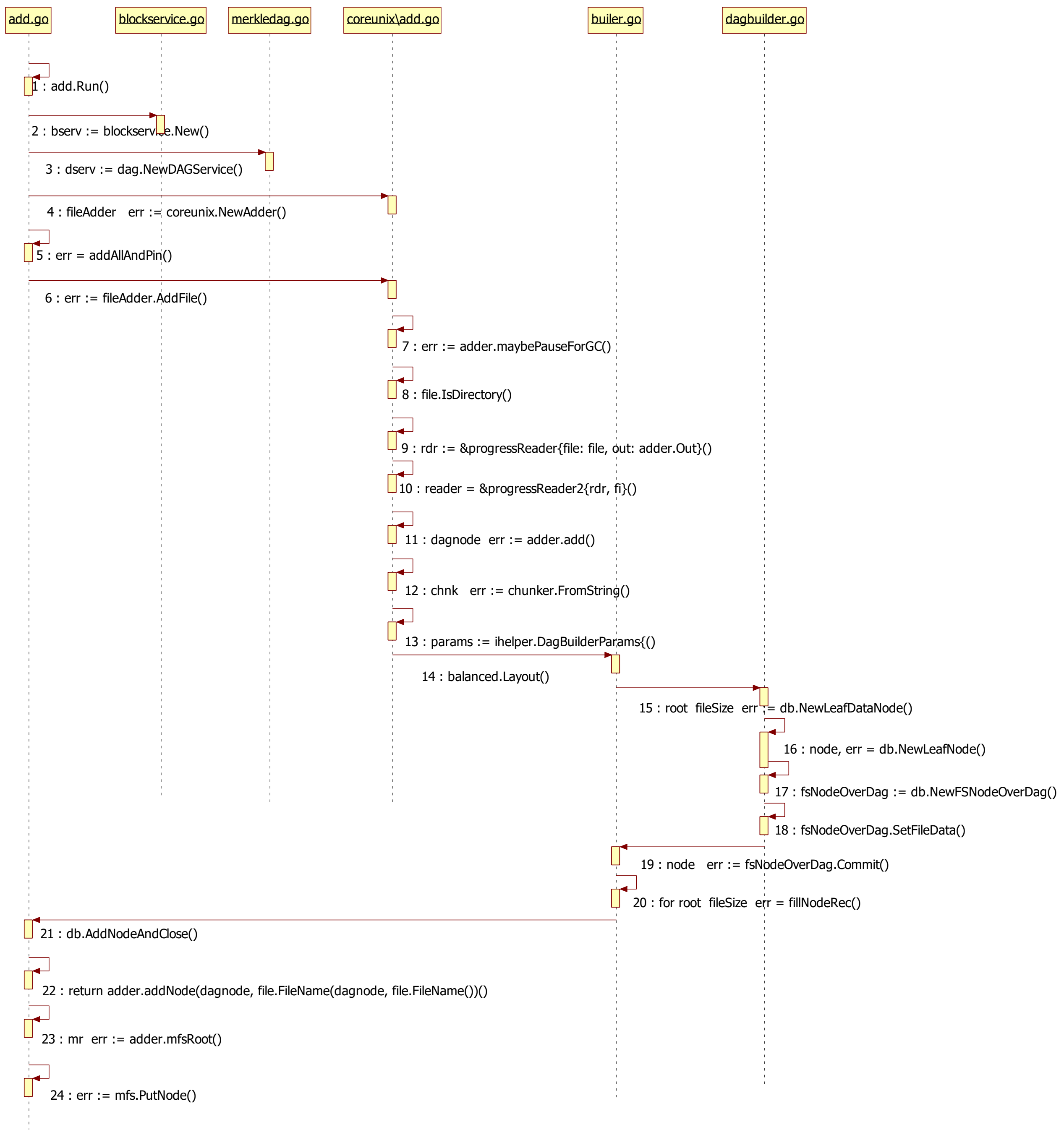

add.go
blockservice.go
merkledag.go
coreunix\add.go
builer.go
dagbuilder.go
1 : add.Run()
2 : bserv := blockservice.New()
3 : dserv := dag.NewDAGService()
4 : fileAdder err := coreunix.NewAdder()
5 : err = addAllAndPin()
6 : err := fileAdder.AddFile()
7 : err := adder.maybePauseForGC()
8 : file.IsDirectory()
9 : rdr := &progressReader{file: file, out: adder.Out}()
10 : reader = &progressReader2{rdr, fi}()
11 : dagnode err := adder.add()
12 : chnk err := chunker.FromString()
13 : params := ihelper.DagBuilderParams{()
14 : balanced.Layout()
15 : root fileSize err := db.NewLeafDataNode()
16 : node, err = db.NewLeafNode()
17 : fsNodeOverDag := db.NewFSNodeOverDag()
18 : fsNodeOverDag.SetFileData()
19 : node err := fsNodeOverDag.Commit()
20 : for root fileSize err = fillNodeRec()
21 : db.AddNodeAndClose()
22 : return adder.addNode(dagnode, file.FileName(dagnode, file.FileName())()
23 : mr err := adder.mfsRoot()
24 : err := mfs.PutNode()

图 3-1

3.1.2 时序图说明

1. ipfs 服务节点响应 add.Run 命令
2. 创建 blockservice 服务，需要传递数据，此服务主要管理数据交换和数据库存储管理，gc 回收

```
addblockstore := n.Blockstore
exch := n.Exchange
```

bserv := blockservice.New(addblockstore, exch)
3. 创建 dagService 组合了 blockservice,主要用于管理 blockservice
dserv := dag.NewDAGService(bserv)fileAdder

4. fileAdder err := coreunix.NewAdder(req.Context, n.Pinning, n.Blockstore, dserv)
创建文件地址管理服务

```
fileAdder.Out = outChan
fileAdder.Chunker = chunker
fileAdder.Progress = progress
fileAdder.Hidden = hidden
fileAdder.Trickle = trickle
fileAdder.Wrap = wrap
fileAdder.Pin = dopin
fileAdder.Silent = silent
fileAdder.RawLeaves = rawblks
fileAdder.NoCopy = nocopy
fileAdder.Prefix = &prefix
```

5. err = addAllAndPin(req.Files)
启动 func 调用 addAllAndPin(req.Files)

6. err := fileAdder.AddFile(file)
FileAdder 大总管职责，负责安排添加文件
7. err := adder.maybePauseForGC
Gc 管理

8. file.IsDirectory
如果传入的是目录，执行目录的添加细节

```
if file.IsDirectory() {
   fmt.Println("file.IsDirectory()")
   return adder.addDir(file)
}
```

9. rdr := &progressReader{file: file, out: adder.Out}

```
type progressReader struct {
   file         files.File
   out          chan interface{}
   bytes        int64
   lastProgress int64
}
```

10. reader = &progressReader2{rdr, fi}

```
type progressReader2 struct {
   *progressReader
   files.FileInfo
}
```

11. 获得 dagnode 节点

```
dagnode, err := adder.add(reader)
// Constructs a node from reader's data, and adds it. Doesn't pin.
//1 创建 splitter chunk（文件分块，超过 256K 的进行分块）
//2.构建 DagBuilderParams
//3.构建 DagBuilderHelper（DagBuilderParams + splitter chunk ）
//4.平衡二叉树方式去，文件分块，存在节点
func (adder *Adder) add(reader io.Reader) (ipld.Node, error) {

   chnk, err := chunker.FromString(reader, adder.Chunker)
   fmt.Println("Chunker: " , adder.Chunker)
   if err != nil {
      return nil, err
   }

   params := ihelper.DagBuilderParams{
      Dagserv:   adder.dagService,
      RawLeaves: adder.RawLeaves,
      Maxlinks:  ihelper.DefaultLinksPerBlock,
      NoCopy:    adder.NoCopy,
      Prefix:    adder.Prefix,
   }

   if adder.Trickle {
      fmt.Println("return trickle.Layout(params.New(chnk))")
      return trickle.Layout(params.New(chnk))
   }
   fmt.Println("return balanced.Layout(params.New(chnk))")
   return balanced.Layout(params.New(chnk))
}
```

12. 获得 Splitter 切片，用来分割一个大的文件为 256K 的块

```
chnk   err := chunker.FromString(reader, adder.Chunker)
```

13. 构建 DagBuilderHelper 实例，管理分片和数据转换，等内容

```
params := ihelper.DagBuilderParams{
   Dagserv:   adder.dagService,
   RawLeaves: adder.RawLeaves,
   Maxlinks:  ihelper.DefaultLinksPerBlock,
   NoCopy:    adder.NoCopy,
   Prefix:    adder.Prefix,
}
params.New(chnk)
// New generates a new DagBuilderHelper from the given params and a given
// chunker.Splitter as data source.
func (dbp *DagBuilderParams) New(spl chunker.Splitter) *DagBuilderHelper {
   db := &DagBuilderHelper{
      dserv:     dbp.Dagserv,
      spl:       spl,
      rawLeaves: dbp.RawLeaves,
      prefix:    dbp.Prefix,
      maxlinks:  dbp.Maxlinks,
      batch:     ipld.NewBatch(context.TODO(), dbp.Dagserv),
   }
   if fi, ok := spl.Reader().(files.FileInfo); dbp.NoCopy && ok {
      db.fullPath = fi.AbsPath()
      db.stat = fi.Stat()
   }

   if dbp.URL != "" && dbp.NoCopy {
      db.fullPath = dbp.URL
   }
   return db
}
```

14. Layout 返回节点

通过平衡二叉树 layout 返回 ipld 节点

```
balanced.Layout(params.New(chnk)

func Layout(db *h.DagBuilderHelper) (ipld.Node, error) {

   if db.Done() {
      fmt.Println("Layout, db.Done()")
      // No data, return just an empty node.
      root, err := db.NewLeafNode(nil)
      if err != nil {
            return nil, err
        }
        // This works without Filestore support (`ProcessFileStore`).
        // TODO: Why? Is there a test case missing?

        return db.AddNodeAndClose(root)
    }

    //生成一个节点
    root, fileSize, err := db.NewLeafDataNode()
    fmt.Println("Layout, db.Done()33333333")
    if err != nil {
        fmt.Println("return nil, err")
        return nil, err
    }

    // Each time a DAG of a certain `depth` is filled (because it
    // has reached its maximum capacity of `db.Maxlinks()` per node)
    // extend it by making it a sub-DAG of a bigger DAG with `depth+1`.
    for depth := 1; !db.Done(); depth++ {
        fmt.Println("depth := 1; !db.Done()")
        // Add the old `root` as a child of the `newRoot`.
        newRoot := db.NewFSNodeOverDag(ft.TFile)
        newRoot.AddChild(root, fileSize, db)

        // Fill the `newRoot` (that has the old `root` already as child)
        // and make it the current `root` for the next iteration (when
        // it will become "old").
        root, fileSize, err = fillNodeRec(db, newRoot, depth)
        if err != nil {
            return nil, err
        }
    }

    return db.AddNodeAndClose(root)
}
```

15. 先读取一个 25K 块大小的节点

```
root  fileSize  err := db.NewLeafDataNode
```

16. 获得 dagnode 节点

```
dagnode, err := adder.add(reader)
```

```go
//1.fileData, err := db.Next() 读取 splitter 的一个块，并进行处理
// 2 db.NewLeafNode(fileData)  返回一个有块数据的 ProtoNode 对象
func (db *DagBuilderHelper) NewLeafDataNode() (node ipld.Node, dataSize uint64, err error) {
   fileData, err := db.Next()
   if err != nil {
      return nil, 0, err
   }
   fmt.Println("Layout, db.Done()")
   dataSize = uint64(len(fileData))
   fmt.Println("len(fileData222",len(fileData))
   // Create a new leaf node containing the file chunk data.

   node, err = db.NewLeafNode(fileData)
   if err != nil {
      return nil, 0, err
   }

   // Convert this leaf to a `FilestoreNode` if needed.  默认不用执行
   node = db.ProcessFileStore(node, dataSize)

   return node, dataSize, nil
}
```

17. 获得一个编码之后的节点

```go
node, err = db.NewLeafNode(fileData)
/ NewLeafNode is a variation from `NewLeaf` (see its description) that
// returns an `ipld.Node` instead.
func (db *DagBuilderHelper) NewLeafNode(data []byte) (ipld.Node, error) {

   if len(data) > BlockSizeLimit {
      fmt.Println("en(data) > BlockSizeLimit",len(data))
      return nil, ErrSizeLimitExceeded
   }

   if db.rawLeaves {
      fmt.Println("len(data)",len(data))
      fmt.Println(" rawLeaves")
      // Encapsulate the data in a raw node.
      if db.prefix == nil {
         fmt.Println(" db.prefix == nil {")
         return dag.NewRawNode(data), nil
      }

      rawnode, err := dag.NewRawNodeWPrefix(data, *db.prefix)
```

```
        if err != nil {
            return nil, err
        }
        return rawnode, nil
    }
    fmt.Println("len(data)11",len(data))
    // Encapsulate the data in UnixFS node (instead of a raw node).
    fsNodeOverDag := db.NewFSNodeOverDag(ft.TFile)  //创建一个 FSNodeOverDag 对象
    fsNodeOverDag.SetFileData(data)   //给文件的 file 里面的 data 数据赋值
    node, err := fsNodeOverDag.Commit()  //将 data []byte 转成格式 protoNote 需要的 data 格式,转
换过程需要分析
    if err != nil {
        return nil, err
    }
    // TODO: Encapsulate this sequence of calls into a function that
    // just returns the final `ipld.Node` avoiding going through
    // `FSNodeOverDag`.
    // TODO: Using `TFile` for backwards-compatibility, a bug in the
    // balanced builder was causing the leaf nodes to be generated
    // with this type instead of `TRaw`, the one that should be used
    // (like the trickle builder does).
    // (See https://github.com/ipfs/go-ipfs/pull/5120.)

    return node, nil //返回的 protoNode 节点,并且节点有了一个格式化块的数据
}
```

18. 获得 dagnode 节点

fsNodeOverDag.SetFileData(data)

dag 是节点组装的

file 是对块进行编解码的

```
func (n *FSNodeOverDag) SetFileData(fileData []byte) {
    n.file.SetData(fileData)
}

type FSNodeOverDag struct {
    dag  *dag.ProtoNode
    file *ft.FSNode
}

func (db *DagBuilderHelper) NewFSNodeOverDag(fsNodeType pb.Data_DataType) *FSNodeOverDag {
    node := new(FSNodeOverDag)
    node.dag = new(dag.ProtoNode)
    node.dag.SetPrefix(db.GetPrefix())
    node.file = ft.NewFSNode(fsNodeType)

    return node
}
```

19.

node, err := fsNodeOverDag.Commit()

将文件传入的数据 data,进行编码，转换为配置需要的格式

20. fillNodeRec

将剩余未处理的块继续处理完成，编码，组装返回节点

```
for depth := 1; !db.Done(); depth++ {
    fmt.Println("depth := 1; !db.Done()")
    // Add the old `root` as a child of the `newRoot`.
    newRoot := db.NewFSNodeOverDag(ft.TFile)
    newRoot.AddChild(root, fileSize, db)

    // Fill the `newRoot` (that has the old `root` already as child)
    // and make it the current `root` for the next iteration (when
    // it will become "old").
    root, fileSize, err = fillNodeRec(db, newRoot, depth)
    if err != nil {
        return nil, err
    }
}
```

21. dbHelp 增加节点管理

db.AddNodeAndClose(root)

```
func (db *DagBuilderHelper) AddNodeAndClose(node ipld.Node) (ipld.Node, error) {

    fmt.Println("AddNodeAndClose")
    err := db.batch.Add(node)
    if err != nil {
        return nil, err
    }

    err = db.Close()
    if err != nil {
        return nil, err
    }
```

```
   return node, nil
}
```

22. 获得 dagnode 节点

```
return adder.addNode(dagnode, file.FileName())
func (adder *Adder) addNode(node ipld.Node, path string) error {
   // patch it into the root
   if path == "" {
      fmt.Println("path == ")
      path = node.Cid().String()
   }

   if pi, ok := node.(*posinfo.FilestoreNode); ok {
      fmt.Println("pi, ok := node.(*posinfo.FilestoreNode); ok  ")
      node = pi.Node
   }

   mr, err := adder.mfsRoot()
   if err != nil {
      return err
   }
   dir := gopath.Dir(path)
   if dir != "." {
      fmt.Println("dir != 。  ")
      opts := mfs.MkdirOpts{
         Mkparents: true,
         Flush:     false,
         Prefix:    adder.Prefix,
      }
      if err := mfs.Mkdir(mr, dir, opts); err != nil {
         return err
      }
   }

   if err := mfs.PutNode(mr, path, node); err != nil {
      return err
   }

   if !adder.Silent {
      return outputDagnode(adder.Out, path, node)
   }
   return nil
}
```

23. 获得 dagnode 节点

大总管联系文件管理系统

```
mr  err := adder.mfsRoot

func (adder *Adder) mfsRoot() (*mfs.Root, error) {
   if adder.mroot != nil {
      fmt.Println("adder.mroot != nil")
      return adder.mroot, nil
   }
   rnode := unixfs.EmptyDirNode()
   rnode.SetPrefix(adder.Prefix)
   fmt.Println("rnode.SetPrefix(adder.Prefix)
",adder.Prefix.Codec,adder.Prefix.MhLength,adder.Prefix.MhType,adder.Prefix.Version)
   mr, err := mfs.NewRoot(adder.ctx, adder.dagService, rnode, nil)
   if err != nil {
      return nil, err
   }
   adder.mroot = mr
   return adder.mroot, nil
}
```

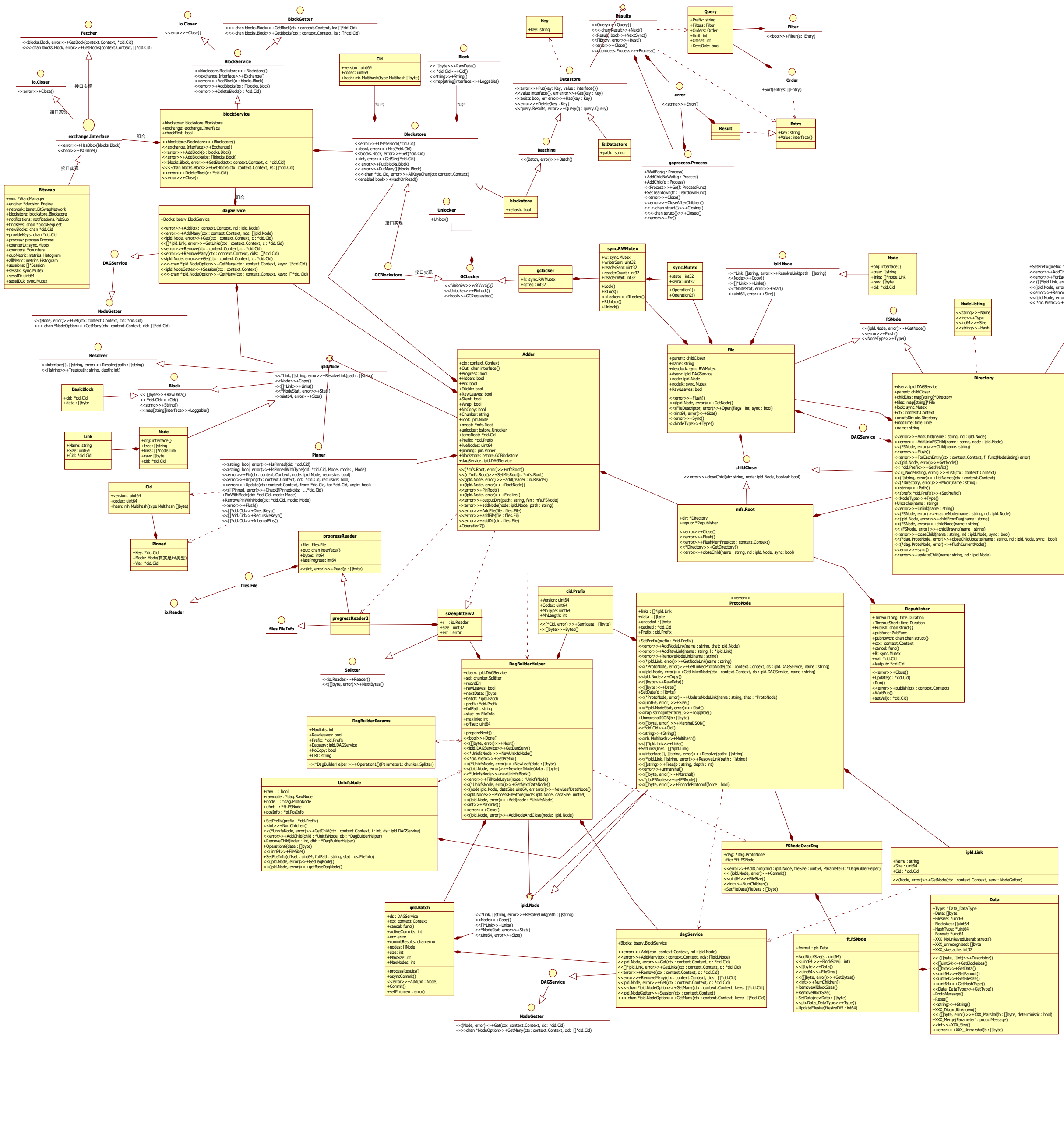
24. 添加节点文件

```
if err := mfs.PutNode(mr, path, node); err != nil {
   return err
}
```

3.1.3 ADD 命令意图

Add 功能目的是将客户端发送过来的文件按照 IPLD 的格式存储在 IPFS 节点。

3.1.4 类图说明



3.2 main.go（commanShouldRunOnDaemon()）

3.2.1 源码初步说明

NODE3.2.1-1