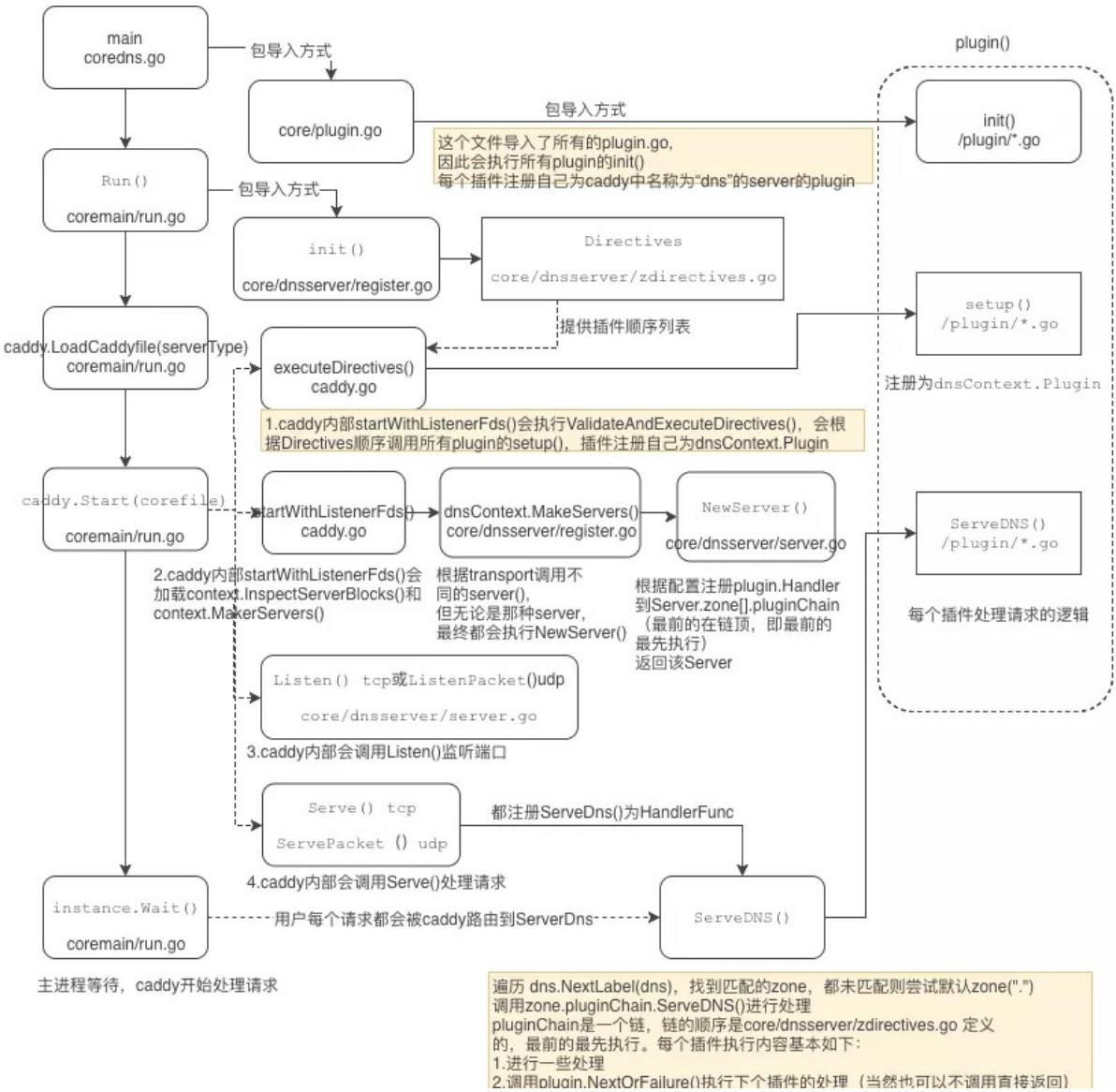
CoreDns源码解析



1. 首先main方法在coredns.go内，因为这个文件导入了core/plugin.go,而core/plugin.go内导入了所有插件，所以会执行所有插件的init方法,每个插件的init方法功能都一样，就是把自己注册为caddy中名为"dns" 的server的plugin

coredns.go

```
import (
 "github.com/coredns/coredns/coremain"

 // Plug in CoreDNS
 _ "github.com/coredns/coredns/core/plugin"
)

func main() {
 coremain.Run()
}
```

```
package plugin

import (
 // Include all plugins.
 _ "github.com/caddyserver/caddy/onevent"
 _ "github.com/coredns/coredns/plugin/any"
 _ "github.com/coredns/coredns/plugin/auto"
 _ "github.com/coredns/coredns/plugin/autopath"
 _ "github.com/coredns/coredns/plugin/bind"
 ....
```

```
func init() {
 caddy.RegisterPlugin("any", caddy.Plugin{
  ServerType: "dns",
  Action:     setup,
 })
}
```

2. 然后main 方法内就一行，运行coremain/run.go内的Run方法,因为这个文件导入了core/dnsserver/register.go,因此会执行该文件的init方法，该方法通过caddy.RegisterServerType 注册了"dns"类型的sever的context,这里有两个参数，一个是Directives，是个string列表，包含了所有的插件名称，还有一个newContext，是初始化context的

run.go

```
import (
    ...
 "github.com/coredns/coredns/core/dnsserver"
    ...
)
```

register.go

```
func init() {
 flag.StringVar(&Port, serverType+".port", DefaultPort, "Default port")

 caddy.RegisterServerType(serverType, caddy.ServerType{
  Directives: func() []string { return Directives },
  DefaultInput: func() caddy.Input {
   return caddy.CaddyfileInput{
    Filepath:       "Corefile",
    Contents:       []byte(".:" + Port + " {\nwhoami\n}\n"),
    ServerTypeName: serverType,
   }
  },
  NewContext: newContext,
 })
}
```

3. coremain/run.go内的Run方法内开始执行caddy.LoadCaddyfile 加载配置文件，然后执行caddy.start方法，这个方法内会进行一系列初始化，主要会处理1，2注册的server

```
func Run() {
 caddy.TrapSignals()

 // Reset flag.CommandLine to get rid of unwanted flags for instance from glog (used in kubernetes).
 // And read the ones we want to keep.
 flag.VisitAll(func(f *flag.Flag) {
  if _, ok := flagsBlacklist[f.Name]; ok {
   return
  }
  flagsToKeep = append(flagsToKeep, f)
 })

 flag.CommandLine = flag.NewFlagSet(os.Args[0], flag.ExitOnError)
 for _, f := range flagsToKeep {
  flag.Var(f.Value, f.Name, f.Usage)
 }

 flag.Parse()

 if len(flag.Args()) > 0 {
  mustLogFatal(fmt.Errorf("extra command line arguments: %s", flag.Args()))
 }

 log.SetOutput(os.Stdout)
 log.SetFlags(0) // Set to 0 because we're doing our own time, with timezone

 if version {
  showVersion()
  os.Exit(0)
 }
 if plugins {
  fmt.Println(caddy.DescribePlugins())
  os.Exit(0)
 }

 // Get Corefile input
 corefile, err := caddy.LoadCaddyfile(serverType)
 if err != nil {
  mustLogFatal(err)
 }

 // Start your engines
 instance, err := caddy.Start(corefile)
 if err != nil {
  mustLogFatal(err)
 }

 logVersion()
 if !dnsserver.Quiet {
  showVersion()
 }

 // Twiddle your thumbs
 instance.Wait()
}
```

注：配置文件见coredns部署篇

- 根据第二步中的newContext初始化dnsContext
- 其次会执行caddy内部的ValidateAndExecuteDirectives方法，该方法的主要是根据第二步注册的Directives和配置文件 依次调用配置文件启用的每个插件的setup()，把启用的插件注册为dnsContext.Plugin

```
func setup(c *caddy.Controller) error {
 a := Any{}

 dnsserver.GetConfig(c).AddPlugin(func(next plugin.Handler) plugin.Handler {
  a.Next = next
  return a
 })

 return nil
}
```

- 然后执行Context.InspectServerBlock()和Context.MakerServers(),即coremain/dnsserver/register.go内的dnsContext的两个方法，起作用分别是加载配置文件,注册pluginChain并返回该Server

```
// InspectServerBlocks make sure that everything checks out before
// executing directives and otherwise prepares the directives to
// be parsed and executed.
func (h *dnsContext) InspectServerBlocks(sourceFile string, serverBlocks []caddyfile.ServerBlock) ([]caddyfile.ServerBlock, error) {
 // Normalize and check all the zone names and check for duplicates
 for ib, s := range serverBlocks {
  for ik, k := range s.Keys {
   za, err := normalizeZone(k)
   if err != nil {
    return nil, err
   }
   s.Keys[ik] = za.String()
   // Save the config to our master list, and key it for lookups.
   cfg := &Config{
       //DNS区域(ZONE)：DNS域名空间中连续的树，将域名空间按照需要划分为若干较小的管理单位。
    Zone:        za.Zone,
    ListenHosts: []string{""},
    Port:        za.Port,
    Transport:   za.Transport,
   }
   keyConfig := keyForConfig(ib, ik)
   if za.IPNet == nil {
    h.saveConfig(keyConfig, cfg)
    continue
   }

   ones, bits := za.IPNet.Mask.Size()
   if (bits-ones)%8 != 0 { // only do this for non-octet boundaries
    cfg.FilterFunc = func(s string) bool {
     // TODO(miek): strings.ToLower! Slow and allocates new string.
     addr := dnsutil.ExtractAddressFromReverse(strings.ToLower(s))
     if addr == "" {
      return true
     }
     return za.IPNet.Contains(net.ParseIP(addr))
    }
   }
   h.saveConfig(keyConfig, cfg)
  }
 }
 return serverBlocks, nil
}
```

```
// MakeServers uses the newly-created siteConfigs to create and return a list of server instances.
func (h *dnsContext) MakeServers() ([]caddy.Server, error) {

 // Now that all Keys and Directives are parsed and initialized
 // lets verify that there is no overlap on the zones and addresses to listen for
 errValid := h.validateZonesAndListeningAddresses()
 if errValid != nil {
  return nil, errValid
 }

 // we must map (group) each config to a bind address
 groups, err := groupConfigsByListenAddr(h.configs)
 if err != nil {
  return nil, err
 }
 // then we create a server for each group
 var servers []caddy.Server
 for addr, group := range groups {
  // switch on addr
  switch tr, _ := parse.Transport(addr); tr {
  case transport.DNS:
   s, err := NewServer(addr, group)
   if err != nil {
    return nil, err
   }
   servers = append(servers, s)

  case transport.TLS:
   s, err := NewServerTLS(addr, group)
   if err != nil {
    return nil, err
   }
   servers = append(servers, s)

  case transport.GRPC:
   s, err := NewServergRPC(addr, group)
   if err != nil {
    return nil, err
   }
   servers = append(servers, s)

  case transport.HTTPS:
   s, err := NewServerHTTPS(addr, group)
   if err != nil {
    return nil, err
   }
   servers = append(servers, s)
  }

 }

 return servers, nil
}
```

- 调用Server的Lisen或ListenPacket方法，即core/dnsserver/server.go内的Server两个方法，启动TCP或者UDP监听

```
// Listen implements caddy.TCPServer interface.
func (s *Server) Listen() (net.Listener, error) {
 l, err := listen("tcp", s.Addr[len(transport.DNS+"://"):])
 if err != nil {
  return nil, err
 }
 return l, nil
}

// ListenPacket implements caddy.UDPServer interface.
func (s *Server) ListenPacket() (net.PacketConn, error) {
 p, err := listenPacket("udp", s.Addr[len(transport.DNS+"://"):])
 if err != nil {
  return nil, err
 }

 return p, nil
}
```

- 调core/dnsserver/server.go内Server或ServePacket方法注册处理TCP或者UDP请求的Handle,这两个方法都会注册ServeDNS方法为Handle

```
// ServePacket starts the server with an existing packetconn. It blocks until the server stops.
// This implements caddy.UDPServer interface.
func (s *Server) ServePacket(p net.PacketConn) error {
 s.m.Lock()
 s.server[udp] = &dns.Server{PacketConn: p, Net: "udp", Handler: dns.HandlerFunc(func(w dns.ResponseWriter, r *dns.Msg) {
  ctx := context.WithValue(context.Background(), Key{}, s)
  s.ServeDNS(ctx, w, r)
 })}
 s.m.Unlock()

 return s.server[udp].ActivateAndServe()
}
```

4. 执行caddy.Wait()主进程进入等待，caddy开始处理请求，这里每个请求都会发送给core/dnsserver/server.go的ServeDNS处理，这里是dns的总逻辑

- 遍历 dns.NextLabel(dns)，找到匹配的zone，都未匹配则尝试默认zone(".")
- 调用zone.pluginChain.ServeDNS()进行处理，pluginChain是一个链，链的顺序是core/dnsserver/zdirectives.go 定义的，最前的最先执行。

```
func (s *Server) ServeDNS(ctx context.Context, w dns.ResponseWriter, r *dns.Msg) {
  .....
  // Wildcard match, if we have found nothing try the root zone as a last resort.
 if h, ok := s.zones["."]; ok && h.pluginChain != nil {
  rcode, _ := h.pluginChain.ServeDNS(ctx, w, r)
  if !plugin.ClientWrite(rcode) {
   errorFunc(s.Addr, w, r, rcode)
  }
  return
 }
}
```

顺序:

```
var Directives = []string{
 "metadata",
 "cancel",
 "tls",
 "reload",
 "nsid",
 "root",
 "bind",
 "debug",
 "trace",
 "ready",
 "health",
 "pprof",
 "prometheus",
 "errors",
 "log",
 "dnstap",
 "any",
 "chaos",
 "loadbalance",
 "cache",
 "rewrite",
 "dnssec",
 "autopath",
 "template",
 "hosts",
 "route53",
 "federation",
 "k8s_external",
 "kubernetes",
 "file",
 "auto",
 "secondary",
 "etcd",
 "loop",
 "forward",
 "grpc",
 "erratic",
 "whoami",
 "on",
}
```

5. 每个插件的ServeDNS()会进行如下处理

- 进行一些插件自定义处理
- 调用plugin.NextOrFailure()执行下个插件的处理（当然也可以不调用直接返回）
  进行一些插件自定义处理

```
// ServeDNS implements the plugin.Handler interface.
func (rr RoundRobin) ServeDNS(ctx context.Context, w dns.ResponseWriter, r *dns.Msg) (int, error) {
 wrr := &RoundRobinResponseWriter{w}
 return plugin.NextOrFailure(rr.Name(), rr.Next, ctx, wrr, r)
}
```