如何阅读KubeEdge框架源代码？

本节内容，我们一起系统的分析了 KubeEdge 云端和边缘侧架构设计。对于有拓展需求的小伙们注意啦，在了解了 KubeEdge 架构设计的基础上，我们就能够更好的去阅读 KubeEdge 的源代码，限于篇幅，本文着重提点关键部分源代码如何阅读。

源码阅读前置条件

阅读KubeEdge源码需要具备以下条件：

- **Go语言基础：**KubeEdge是基于Golang语言开发的，因此阅读源码同学至少需要了解Golang语法。
- **了解KubeEdge架构设计：**源码就是将架构设计落地，了解架构设计，才能更好阅读源码。

阅读技巧

下载源码后，着重看cloud和edge两个文件夹，这两个分别是云端和边缘侧的应用启动入口。云和边代码结构很像，咱们下面就分析云端源码作为抛砖引玉。

进入cloud文件夹后，再进入cmd文件夹，此时有四个文件夹：

- admission
- cloudcore
- csidriver
- iptablesmanager

cloudcore是云端的核心模块，我们进一步来分析cloudcore。

入口函数：

```
...
command := app.NewCloudCoreCommand()
...
if err := command.Execute(); err != nil {
    os.Exit(1)
}
...
```

我们可以看到执行了app包下面的NewCloudCoreCommand方法，这个小伙伴们要注意啦，在KubeEdge框架当中，所有的模块的实现都是这种形式。这是是借助于 Cobra（一个 基于Go 语言 Cli 框架）。

我们重点要关注的就是这个 Command 的 Run 字段，这里会定义执行这个命令的具体内容：

```
&cobra.Command{
    ...
    Run: func(cmd *cobra.Command, args []string) {
                verflag.PrintAndExitIfRequested()
                ...
    }
    ...
}
```

对于cloudcore这个Command而言，在这个方法里面有几行代码非常关键：

读取云端cloudcore的配置文件，默认的读取路径为`/etc/kubeedge/config/cloudcore.yaml`，然后解析为对应字段的结构体。

```
opts := options.NewCloudCoreOptions()
...
config, err := opts.Config()
```

注册各个模块：

```
registerModules(config)
```

registerModules注册模块的具体实现：

```
func registerModules(c *v1alpha1.CloudCoreConfig) {
 cloudhub.Register(c.Modules.CloudHub)
 edgecontroller.Register(c.Modules.EdgeController)
 devicecontroller.Register(c.Modules.DeviceController)
 synccontroller.Register(c.Modules.SyncController)
 cloudstream.Register(c.Modules.CloudStream, c.CommonConfig)
 router.Register(c.Modules.Router)
 dynamiccontroller.Register(c.Modules.DynamicController)
}
```

这里我们可以看到，这里就是咱们在分析云端架构设计当中的各个模块，register只是把模块以名称为key放到map中。

完成注册之后，下一步就是依次启动各个模块：

```
core.StartModules()
```

StartModules启动模块的具体实现

```
func StartModules() {
 // only register channel mode, if want to use socket mode, we should also pass in common.MsgCtxTypeUS parameter
 beehiveContext.InitContext([]string{common.MsgCtxTypeChannel})

 modules := GetModules()

 for name, module := range modules {
  var m common.ModuleInfo
  switch module.contextType {
  case common.MsgCtxTypeChannel:
   m = common.ModuleInfo{
    ModuleName: name,
    ModuleType: module.contextType,
   }
  case common.MsgCtxTypeUS:
   m = common.ModuleInfo{
    ModuleName: name,
    ModuleType: module.contextType,
    // the below field ModuleSocket is only required for using socket.
    ModuleSocket: common.ModuleSocket{
     IsRemote: module.remote,
    },
   }
  default:
   klog.Exitf("unsupported context type: %s", module.contextType)
  }

  beehiveContext.AddModule(&m)
  beehiveContext.AddModuleGroup(name, module.module.Group())

  go moduleKeeper(name, module, m)
  klog.Infof("starting module %s", name)
 }
}
```

通过遍历在上一步注册的模块，着重看`beehiveContext.AddModule(&m)`，这里实际上就是给beehiveContext中的moduleContext添加了一个类型为module.moduleName的channel，并保存到channels这个map中。

然后调用`beehiveContext.AddModuleGroup(name, module.module.Group())`，这里就是把刚才创建的channel以group的名字为key创建一个map，然后把整个map放到moduleContext的typeChannels中。group的名字是硬编码在实现module接口的具体module上的，比如cloudhub的的Group返回的就是"cloudhub"

完成注册之后，调用`module.Start`启动各个模块，Start是一个接口，每个具体的module都实现了自己的Start方法。

beehive起到的作用就是完成各个模块之间的通信，初步看beehive实现了两种通信机制，一种是通过unixsocket，另一种是通过golang的channel。

总结

下面我们来对前面的源码分析做一个小小的总结，小伙伴们一定要记住，依靠别人把代码一行一行的去分析是不现实的，我们需要掌握核心的分析逻辑。通过以上的分析我们可以看出，KubeEdge的模块是借助于cobra来启动的，cobra的command:Run部分就是具体的执行逻辑。在Run实现里，我们可以看到包括先读取配置文件、注册模块、启动模块这三大步骤，通过这样做的目的就是为了让模块之间通过beehive模块完成通信。

除了cloudcore是这种逻辑之外，类似的如edgecore、iptablesmanager等都是类似的代码结构，了解了其设计原理之后，我们就可以如法炮制的去具体分析各个模块的具体内容了。