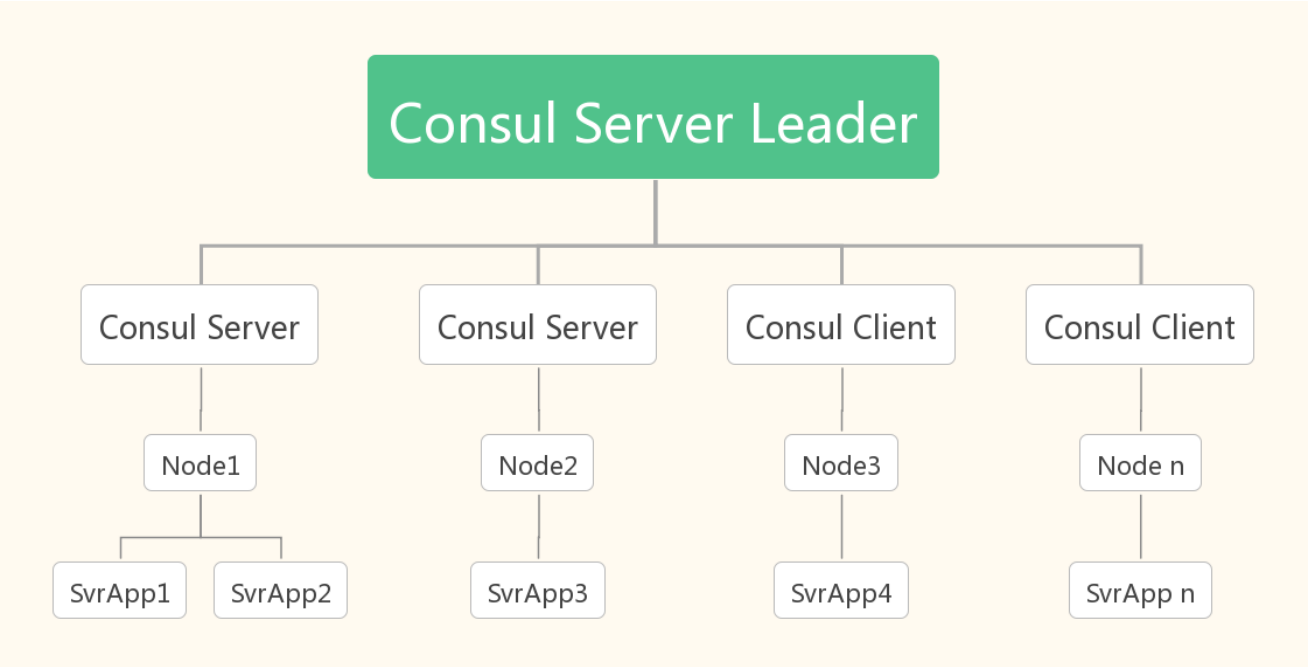
Consul应用

应用服务注册、注销，KV存储，以及监控服务和KV变化

能解决的问题

1. 服务注册与发现
2. 业务服务的配置数据进行统一管理
3. 节点监控（是否活着）
4. 服务监控（是否活着）
5. api接口注册后，可通过consul-template工具自动进行nginx路由配置
6. 服务注册后，可自动修改DNS，适用于集群业务路由

部署结构



SDK地址

1. golang: https://github.com/hashicorp/consul
2. python: https://github.com/gmr/consulate
3. PHP: https://github.com/dcarbone/php-consul-api

基本定义

字符书写规范

统一用大驼峰
例如：

1. 业务服务名：ActGuessSvr
2. 注册时服务名：ActGuessSvr_Post

3. 注册时服务ID：ActGuessSvr_Post_1234
4. 数据库配置KV：DB/Actives
5. redis配置KV：Redis/Active
6. 服务参数配置KV：Services/SampleSvr/PostCfg

服务名

1. 每个服务定义一个唯一的可识别的名称，例如 RoomSvr, GateSvr, VideoShareSvr等等。如果一 个业务服务里包含了Tcp服务和HttpApi，则需要分开注册。
2. 每个业务服务可以部署多个（多台设配同时运行、单台设备多开）
3. 由于consul健康检测通过http/tcp响应，因此可按需对应的再开一个端口（只能检测该服务是否活着）。
   注册服务名：ServiceName + [_+用途]
   类型：string

服务ID

服务名称 + [_ 用途] + [_ PID]。
变量名：ServiceId
类型：string

服务端口

业务服务中包含的Tcp业务或Http业务等。(1. 提前沟通好；2.系统自动分配)
变量名：ServicePort
类型：int

Tags

??????

健康检测响应处理

http响应处理

在业务服务下，需要注册一个http响应，以及/check的响应处理。**注册http业务时，最好加上http检测**

```
addr := "http://127.0.0.1:12308/check"
uri, err := url.Parse(addr)
if err != nil {
  return err
}

http.HandleFunc(uri.Path, func(w http.ResponseWriter, r *http.Request){
  w.Write([]byte("success"))
})
go http.ListenAndServe(uri.Host, nil)
```

tcp响应处理

在业务服务中，也可以注册tcp来响应健康监测。**注册post端口时，可不加检测，Post服务可监控到**

```
addr := "tcp://127.0.0.1:12308"
uri, err := url.Parse(addr)
if err != nil {
  return err
}
tcpAddr, err := net.ResolveTCPAddr("tcp", uri.Host)
if err != nil {
  return err
}
listen, err := net.ListenTCP("tcp", tcpAddr)
if err != nil {
  return err
}
go func(){
  for {
    if conn, err := listen.AcceptTCP(); err == nil {
      conn.Close()
    }
  }
}()
```

consul Api导入和对象获取

```
import "github.com/hashicorp/consul/api"
import "svr-frame/tools/consul"  //consul封装工具

uri := "http://127.0.0.1:8500"
//获取consul对象
agent := GetConsulInst(uri)
if agent == nil {
  return
}
```

服务注册与注销

应用服务启动时，将自己注册到consul中，以便PostSvr能及时发现该服务并连接上。
code sample:

```
//需要注册的服务信息
service := &api.AgentServiceRegistration{
  ID: ServiceId,
  Name: ServiceName,
  Address: "127.0.0.1", //当前部署的节点，需要post可见;可使用libs.GetInnerIp()方法获取局域网ip，
libs.GetExternIp()获取外网ip
  Port: ServicePort,
  Tags: []string{},
}
service.Check = &api.AgentServiceCheck{
        HTTP: fmt.Sprintf("http://%s:%d%s", service.Address, service.Port, "/check"),
        //TCP: fmt.Sprintf("%s:%d", service.Address, service.Port),
        Timeout: "10s",
        Interval: "15s",
        DeregisterCriticalServiceAfter: "30s",
}

//注册监控检测，可以http,https,script的一种，值为访问的path，能通过AgentServiceRegistration自动构建出完
整的检测地址
//attrs := make(map[string]string)
//attrs["check_http"] = "/check"  //通过访问，是否返回200来判断
//attrs["check_https"] = "/check"
//attrs["check_tcp"] = "/check"  支持
//attrs["check_script"] = "curl http://127.0.0.1:/8500/check"  //可以是python,shell脚本，返回0正常
//attrs["check_ttl"] = "ttl"
//attrs["check_timeout"] = "10s"
//attrs["check_interval"] = "15s"
//attrs["check_deregister_after"] = "30s"

//服务启动时，需要发起服务注册
if err := agent.Register(service, nil); err != nil {
  return err
}
//运行服务

//服务结束后，需要发起服务注销
if err := agent.Unregister(service.ID); err != nil {
  return err
}

//quit
```

KV保存与获取

配置信息统一保存到consul中，可以通过consul ui统一管理和更新。复杂对象可以使用json结构的字符串，例如数据库配置、redis配置。
key需要统一到对应的服务目录下，即：服务名称/key。例如db/videos，db为数据库服务配置目录。

```
key := "MyService/Foo" //服务目录
val := "test value"
//保存KV
if err := agent.PutKV(key, val); err != nil {
  return err
}

//获取KV
if val, err := agent.GetKV(key); err != nil {
  return err
} else {
  beego.Info(val)
}
```

获取服务信息

通过服务名称获取服务地址、端口等信息

```
res, err := agent.GetService(ServiceName)
if err != nil {
  return err
}
//应为服务可以部署多个地方，因此res是slice结构
for i, v := range res {
  beego.Info(i, v.Service.ID, v.Service.Service, v.Service.Address, v.Service.Port)
}
```

consul 事件处理

暂无使用场景

```
name := "user consul event name"
if err := agent.FireEvent(name); err != nil {
  return err
}
//发送事件成功
```

consul侦听

当有服务注册、注销、KV变更、事件派发时，可通过侦听来捕获。
每侦听一个对象（服务，KV，事件），会运行一个goroute

```
//consul 事件处理
func readChan(eventChan common.EventChannel) {
        for {
      select {
        case eObj := <-eventChan:
                switch d:= eObj.data.(Type){
              case *ConsulKeyChange:
                //do something
              case *ConsulServiceChange:
                //do something
              case *ConsulServicesChange:
                //do something
              case *ConsulEventChange:
                //do something
                }
      }
        }
}
//定义consul事件队列
eventChan := make(common.EventChannel, 100)
//侦听对象获取
uri := "http://127.0.0.1:8500"  //consul地址
w := NewWatcher(uri)
if w == nil {
  return
}
w.AddEventChan(eventChan)
go readEventChan(eventChan)//读取侦听消息
//do watching
//....
```

服务侦听

指定服务侦听

```
err := w.AddServiceWatch(ServiceName, "", false)
if err != nil {
  return err
}
//运行中
//...
w.stopServiceWatch(ServiceName)
```

所有服务侦听

```
err := w.AddServicesWatch(false)
if err != nil {
  return err
}
//运行中
//...
w.StopServicesWatch()
```

KV侦听

```
key := "myService/foo"
err := w.AddKeyWatch(key)
if err != nil {
  return err
}
//运行中
//...
w.StopKeyWatch(key)
```

事件侦听

```
name := "user consul event name"
err := w.AddConsulEventWatch(name)
if err != nil {
  return err
}
//运行中
//...
w.StopConsulEventWatch(name)
```