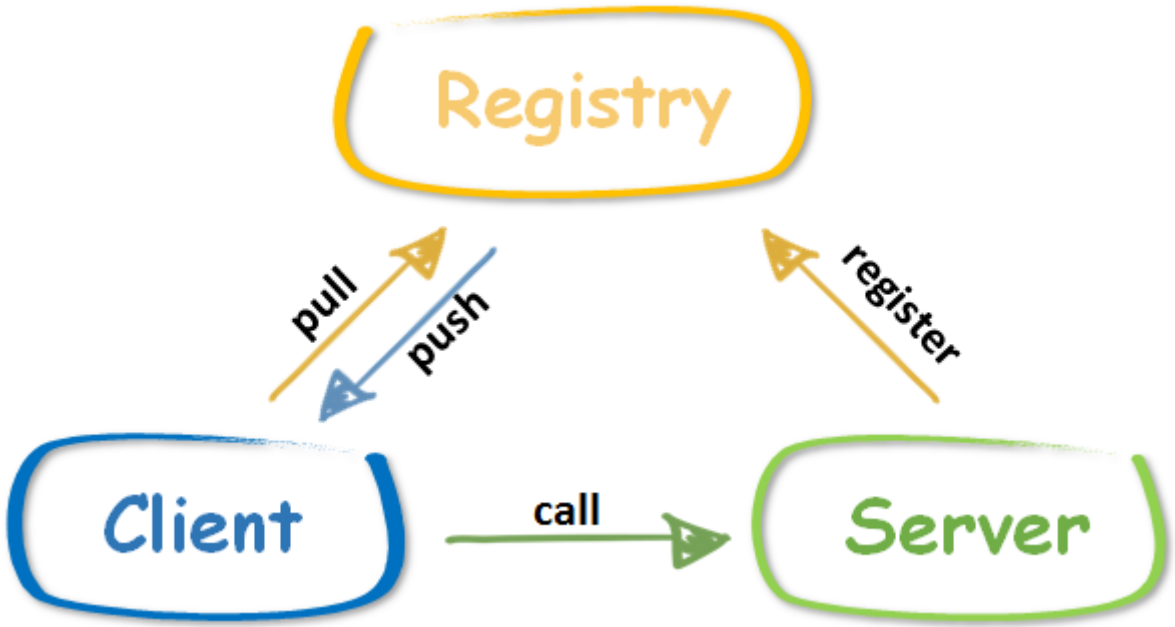
注册组件开发

联系QQ：2816010068，加入会员群

目录

- 服务注册&发现原理
- 注册中心选型
- 注册组件设计
- Etcd注册插件开发

服务注册&发现原理



注册中心选型

Feature	Consul	zookeeper	etcd	euerka
服务健康检查	服务状态，内存，硬盘等	长连接，keepalive	连接心跳	可配支持
多数据中心	支持	—	—	—
kv存储服务	支持	支持	支持	—
一致性	raft	paxos	raft	—
cap	cp	cp	cp	ap
使用接口(多语言能力)	支持http和dns	客户端	http/grpc	http（sidecar）
watch支持	全量/支持long polling	支持	支持 long polling	支持 long polling/大部分增量
自身监控	metrics	—	metrics	metrics
安全	acl /https	acl	https	—
spring cloud集成	已支持	已支持	已支持	已支持

选项模式介绍

- 问题描述

```
type Options struct {
    StrOption1 string
    StrOption2 string
    StrOption3 string
    IntOption1 int
    IntOption2 int
    IntOption3 int
}
```

选项模式介绍

- 解决方案一

```go
func InitOptions1(strOption1 string, strOption2 string, strOption3 string,
    IntOption1 int, IntOption2 int, IntOption3 int) {
    options := &Options{}
    options.StrOption1 = strOption1
    options.StrOption2 = strOption2
    options.StrOption3 = strOption3
    options.IntOption1 = IntOption1
    options.IntOption2 = IntOption2
    options.IntOption3 = IntOption3

    fmt.Printf("init options1:%#v\n", options)
    return
}
```

选项模式介绍

- 解决方案二

```go
func InitOptions2(opts ...interface{}) {
    options := &Options{}
    for index, opt := range opts {
        switch index {
        case 0:
            str, ok := opt.(string)
            if !ok {
                return
            }
            options.StrOption1 = str
        case 1:
            str, ok := opt.(string)
            if !ok {
                return
            }
            options.StrOption2 = str
        case 2:
            str, ok := opt.(string)
            if !ok {
                return
            }
            options.StrOption3 = str
        case 3:
            val, ok := opt.(int)
            if !ok {
                return
            }
            options.IntOption1 = val
        case 4:
            val, ok := opt.(int)
```

选项模式介绍

- 解决方案三（选项模式）

```go
type Option func(opts *Options)

func InitOption3(opts ...Option) {
    options := &Options{}
    for _, opt := range opts {
        opt(options)
    }

    fmt.Printf("init options3:%#v\n", options)
}

func WithStringOption1(str string) Option {
    return func(opts *Options) {
        opts.StrOption1 = str
    }
}

func WithStringOption2(str string) Option {
    return func(opts *Options) {
        opts.StrOption2 = str
    }
}
```

注册组件设计

- 目标
  - 支持多注册中心
  - 支持可扩展
  - 基于接口的设计思路
  - 基于插件的设计思路

注册组件设计

- 接口定义

```
3  import (
4      "context"
5  )
6
7  type Register interface {
8      Init(opts ...Option)
9      Register(ctx context.Context, service *Service)
10     Unregister(ctx context.Context, service *Service)
11 }
12
```

注册组件设计

- 数据结构定义

```go
package register

// 服务抽象
type Service struct {
    Name  string
    Nodes []*Node
}

// 服务节点的抽象
type Node struct {
    Id   string
    IP   string
    Port int
}
```

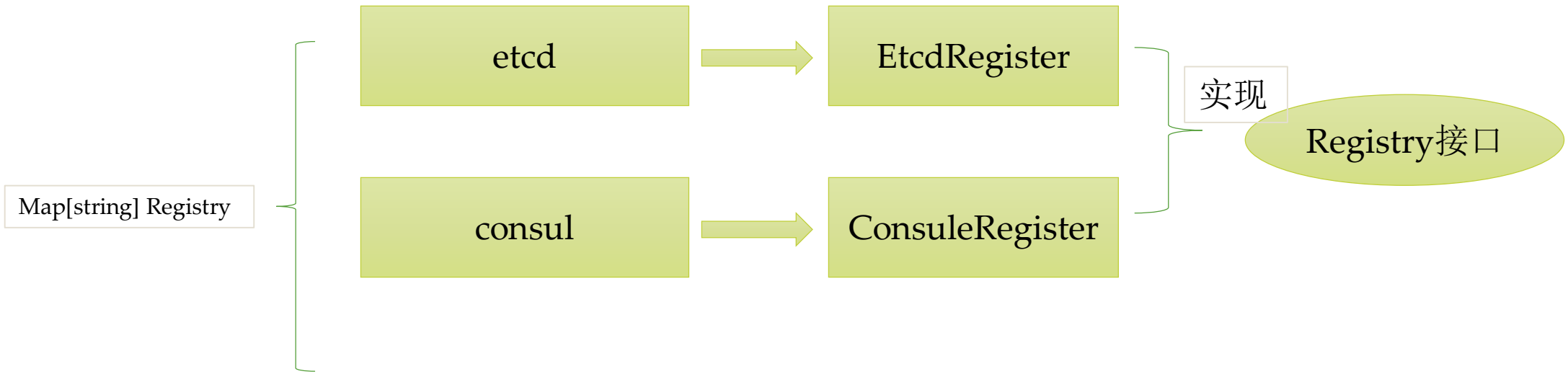
基于插件的设计思路

- 插件抽象
  - 实现了Registry接口，就是一个注册插件
    - 基于etcd的注册插件
    - 基于consul的注册插件
    - 基于zookeeper的注册插件

- 插件管理
  - 提供RegisterPlugin函数，用来注册插件
  - 基于名字对插件进行管理
  - 插件初始化，通过名字对插件进行初始化

基于插件的设计思路

- 插件管理

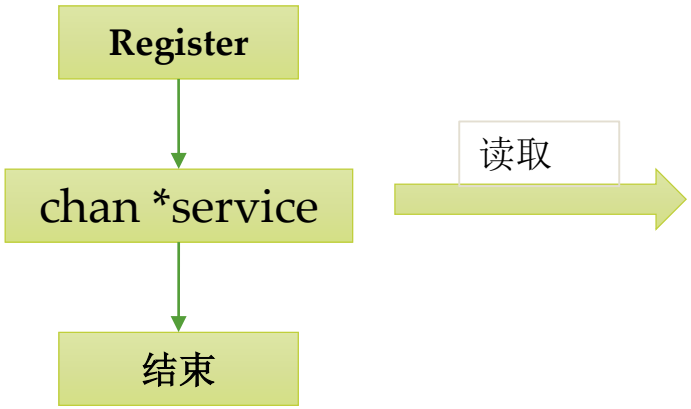


基于插件的设计思路

- 插件查找
  - 通过名字查找到对应的插件，并返回
  - 调用方通过返回的插件实现操作服务注册和发现

etcd注册插件开发

- 实现Register接口
  - 服务注册



读取