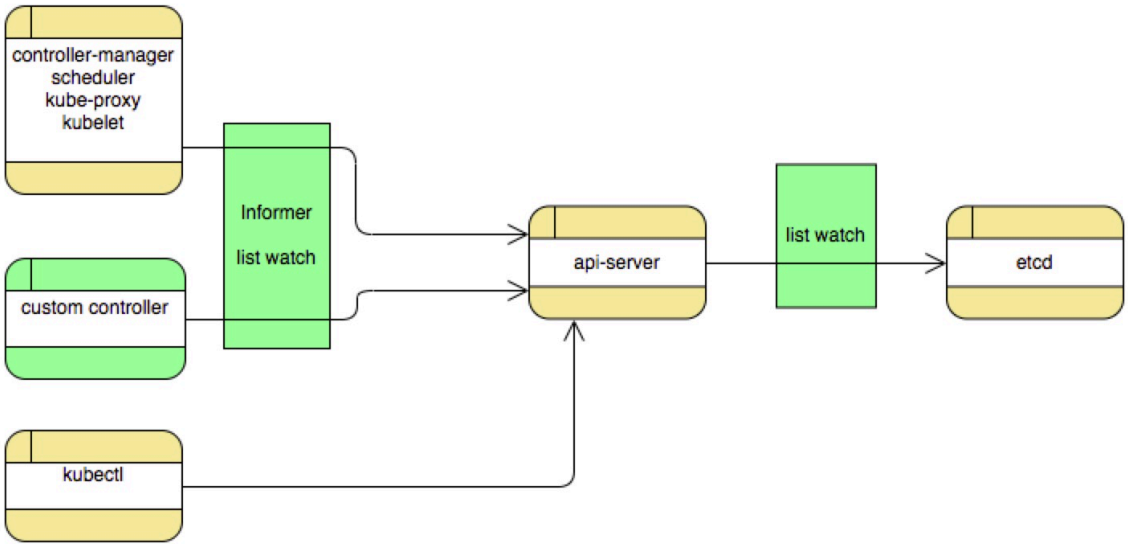
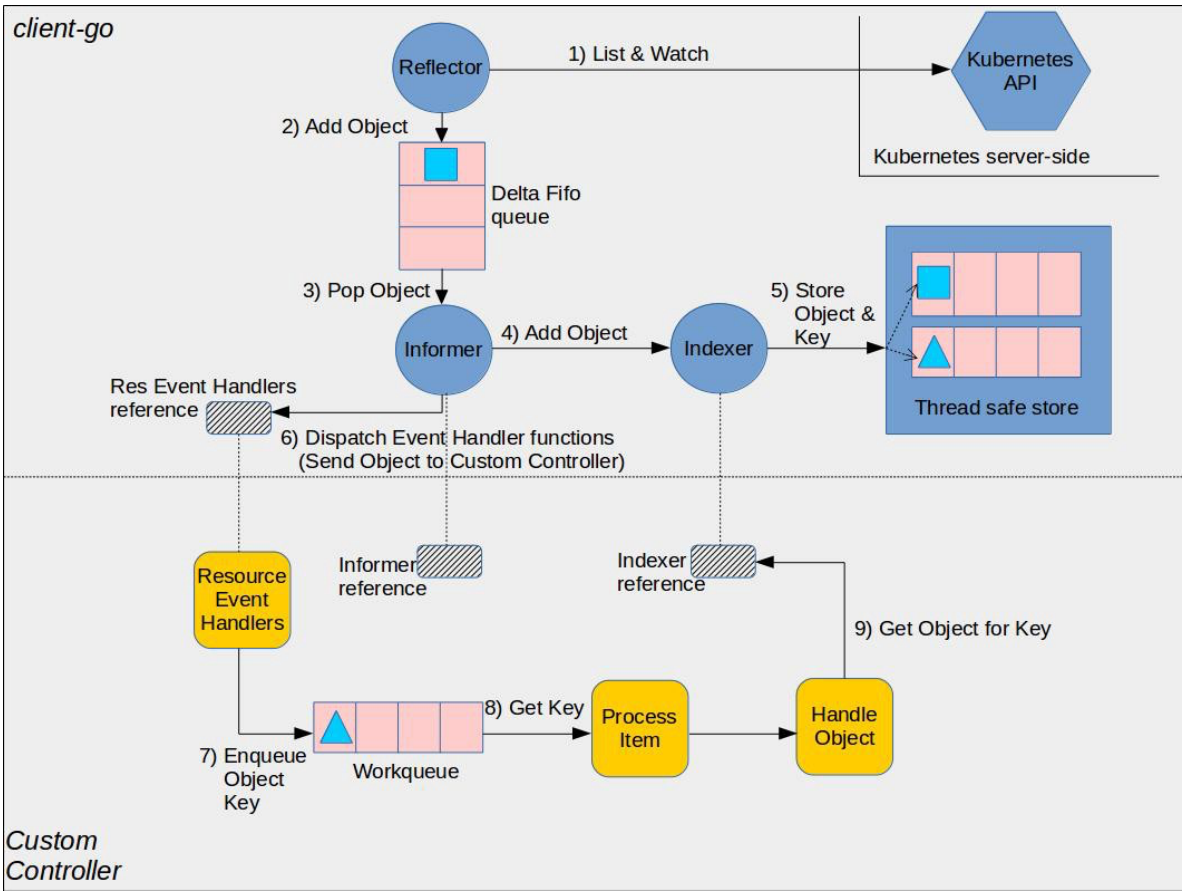
【K8s components 消息通信】



usecase-informer.go: https://github.com/chestack/k8s-codes/blob/master/listwatch/main/usecase-informer.go

https://github.com/kubernetes/sample-controller



SharedIndexInformer，注释里写的是多个listenner共用一个informer, kube-controller-manager里有很多***-controller，SharedIndexInformer如其名字，所有controllers 共用这个informer

Reflector 负责获取和监听服务端数据变化

- list
- watch

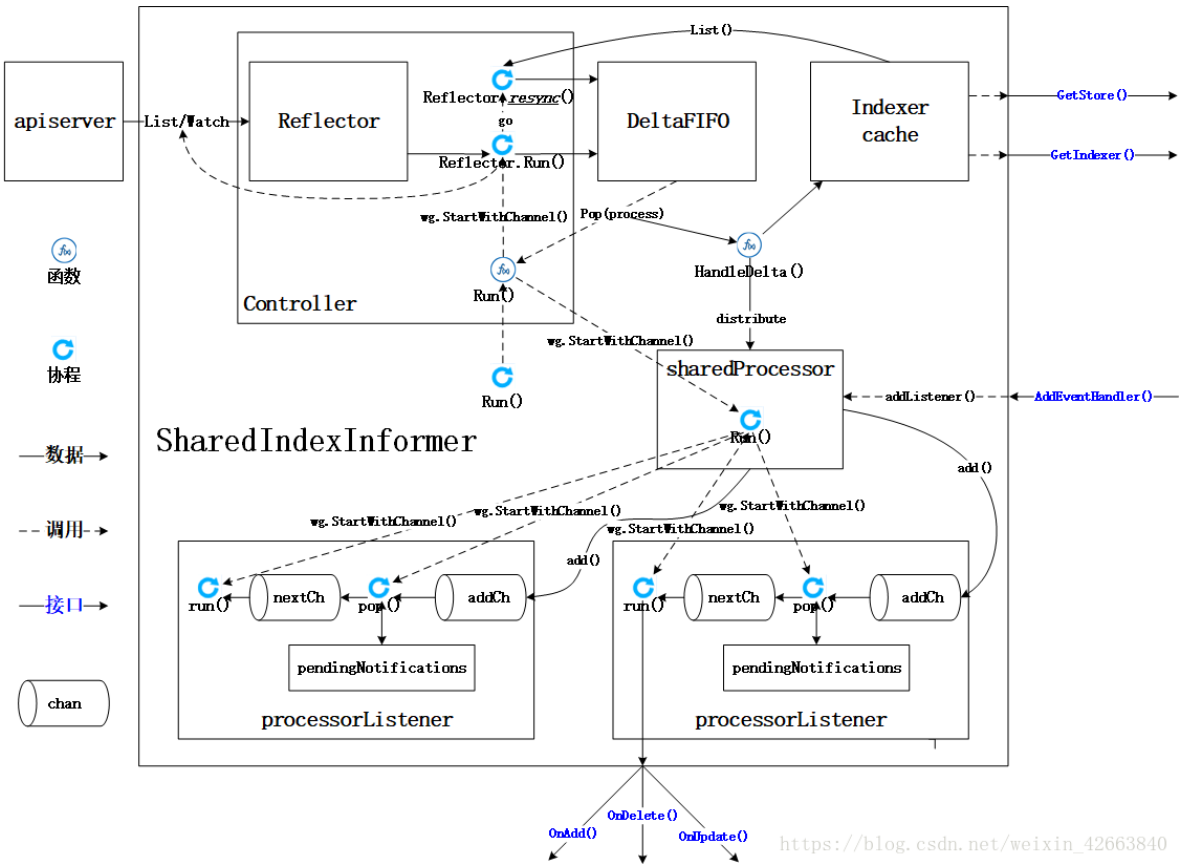
store 负责Reflector 数据的存储

- items 主要是存放key-value的数据索引
- indices 存储对象的所有索引映射
- indexer 存储通过当前索引生成的索引映射

DeltaFIFO 是一个事件队列，Reflector接收数据变化之后，将数据传给store和ResouceEventHandler

c-controller.go: https://github.com/chestack/k8s-codes/blob/master/listwatch/main/usecase-informer.go

PodStatus: Scheduled, Initialized, Running



https://blog.csdn.net/weixin_42663840/article/details/81699303

以scheduler 为例

一些细节：

1. 包管理

go get 只会去head 代码，不能解决指定版本，不能reproducible client library install
vendor目录不能解决的两个问题：1.依赖包列表；2.每个依赖包的版本信息

glide：application使用glide
godep: kubernetes, docker, coreos
go 官方的dep

2. interface

```
informer.AddEventHandler(cache.ResourceEventHandlerFuncs{
    AddFunc: func(obj interface{}) {
        key, err := cache.MetaNamespaceKeyFunc(obj)
        mObj := obj.(meta_v1.Object)
        log.Printf( format: "New Pod Added to Store: %s", mObj.GetName())
        if err == nil {
            queue.Add(key)
        }
    },
```

go 没有显式的关键字用来实现 interface，只需要实现 interface 包含的方法即可
empty interface， 空的 interface 没有方法，所以可以认为所有的类型都实现了 interface{}
Type assertions，f, ok := i.(float64)
More details: https://sanyuesha.com/2017/07/22/how-to-understand-go-interface/

3. channel

```
func (p *processorListener) pop() {
    defer utilruntime.HandleCrash()
    defer close(p.nextCh) // Tell .run() to stop

    var nextCh chan<- interface{}
    var notification interface{}
    for {
        select {
        case nextCh <- notification:
            // Notification dispatched
            var ok bool
            notification, ok = p.pendingNotifications.ReadOne()
            if !ok { // Nothing to pop
                nextCh = nil // Disable this select case
            }
        case notificationToAdd, ok := <-p.addCh:
            if !ok {
                return
            }
            if notification == nil { // No notification to pop (and pendingNotifications is empty)
                // Optimize the case - skip adding to pendingNotifications
                notification = notificationToAdd
                nextCh = p.nextCh
            } else { // There is already a notification waiting to be dispatched
                p.pendingNotifications.WriteOne(notificationToAdd)
            }
        }
    }
}
```

golang中 No-buffer的channel 读和写都是阻塞的，sharedProcessor 把notification分发给所有的listener，所以每个listener需要尽快把notification读走，否则影响其他的listener，所以每个listener需要个 goroutine pop()来负责通过addCh来读取notification
listener的另外一个 goroutine run() 来处理notification，pop()也不能阻塞，所以需要一个 pendingNotification来做消息缓存。
为啥不用buffered channel？？ 大小限制？？

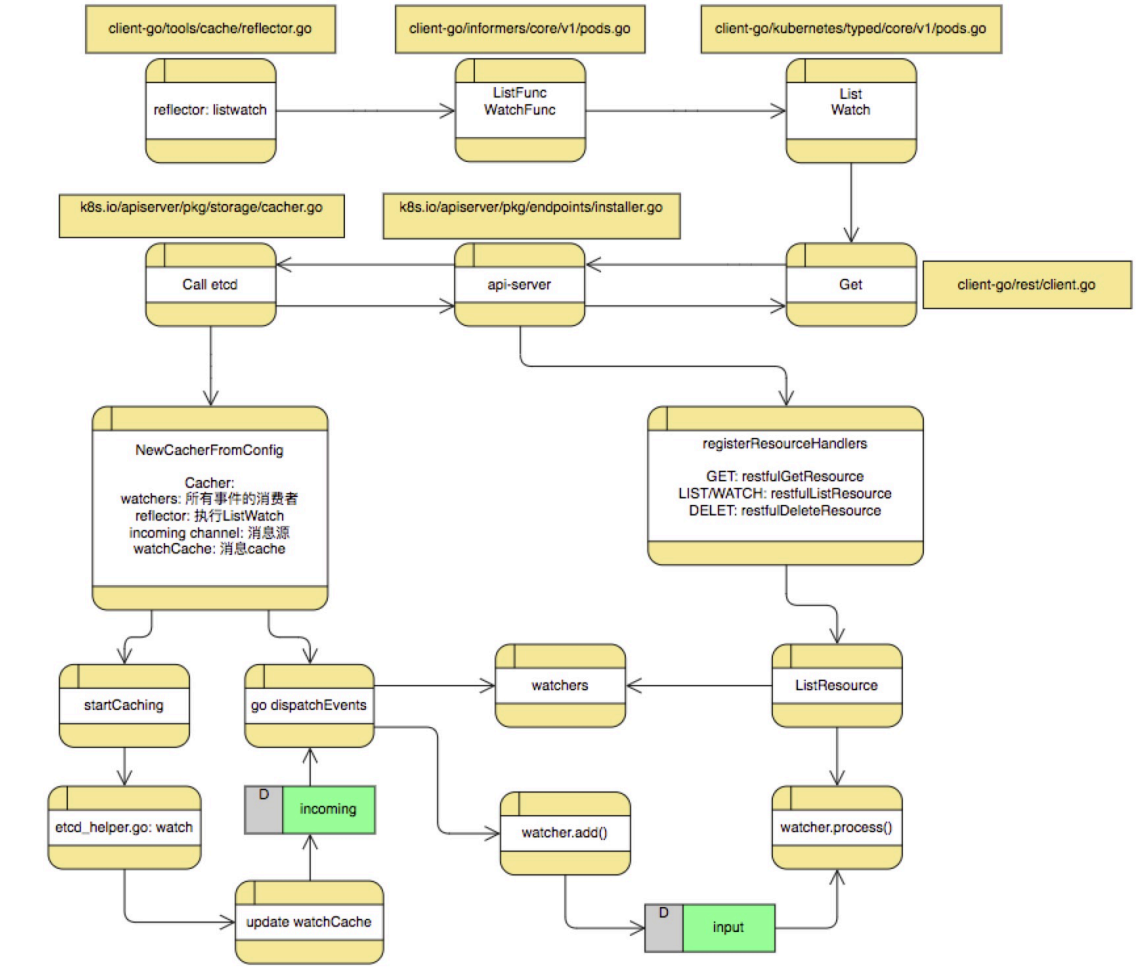
总体感觉：

RPC： consumer <—> provider
Queue：sender <—> queue <—> receiver；Queue有三好：异步，解耦，消峰

list-watch events, Rest API+长连接+重连
channel + cache(http连接): queue(中间件)

Golang 简约(语法糖少)，强悍(goroutine，channel，sync，mutex)，有很多潜在的坑(goroutine 泄露，channel 阻塞，deadlock)

apiserver to etcd



https://zhuanlan.zhihu.com/p/33335726

apiserver evnets 快速转发
consumer： 按需处理，需要store

剩余问题：
ListWatch中 resourceVersion的使用
List-Watch的底层实现(List 短连接 Watch 长连接)
Operator and CDR