##### kubernetes clinet-go

<https://blog.csdn.net/boling_cavalry/article/details/113753087>

kubernetes API References Docs：

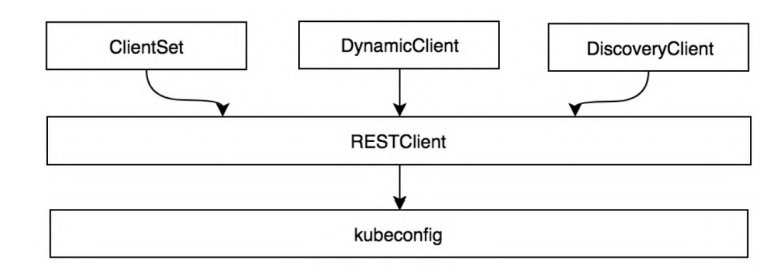
<https://v1-19.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/#-strong-api-overview-strong->

<https://cn.wbsnail.com/p/dive-into-kubernetes-informer>

client-go 是 k8s 的 go 客户端。通过client-go可以与k8s集群交互。

client-go是k8s的控制平面其他组件（controller-manager、scheduler等）与apiservier通信的组件。

client-go 中封装了三种客户端、 REST Client、clientSet、dynamicClient以及DiscoveryClient



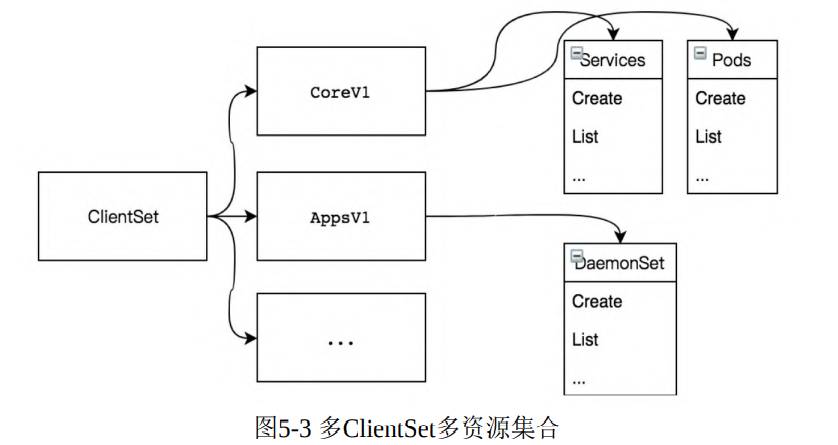
其中，RESTCLient是其他几种客户端的基础；

在使用RESTCLient的时候，需要先定义result的结构、然后设置请求的Group、Version等；这部分在clientSet中进行了封装；

DynamicClient最大的不同是实现了对CRD资源的控制。因为在CRD资源有自定义的字段等。

clientSet 通过封装RESTClient的不同的G、V对应的资源，可以使用一个clientSet获得多种资源的Client；

clientset, err := kubernetes.NewForConfig(config)  
podClient := clientset.CoreV1().Pods(apiv1.NamespaceDefault)  
deploymentClient := clientset.AppsV1().Deployments("kube-system")  
...



clientSet通过封装可以获得多个Group、Version的资源

#### Informor机制

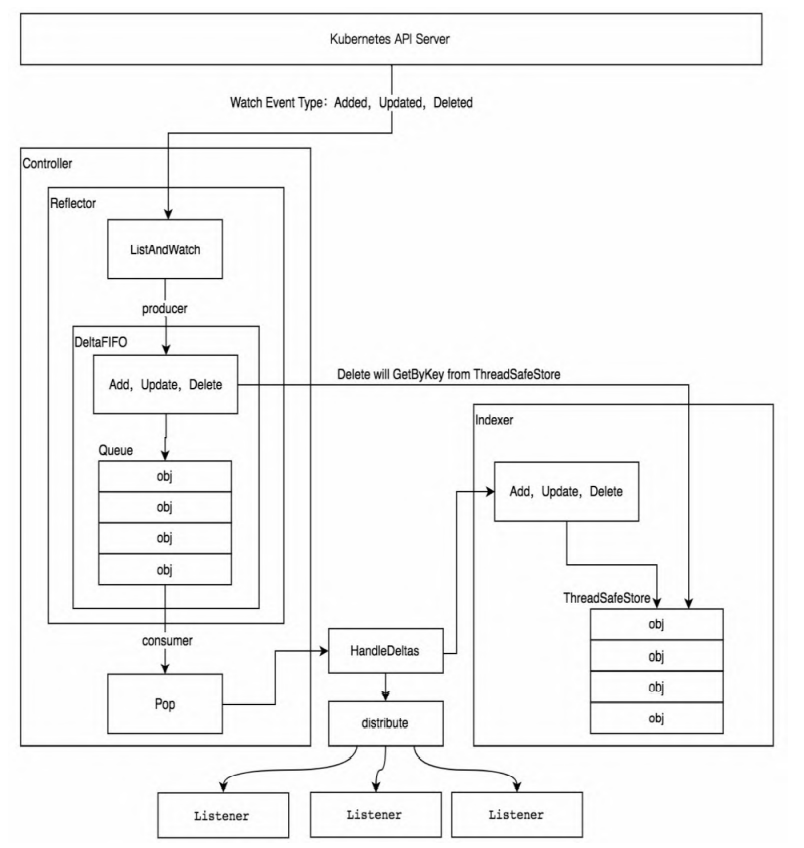


image-20211112105110331

informor 运行原理图；informor是在client-go中负责监控k8s的api-server中是否有事件发生

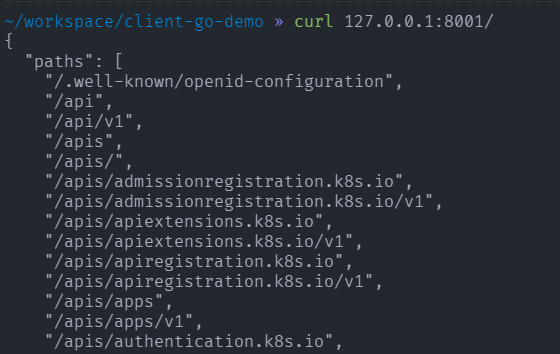
如果有事件发生，informor就会根据list、watch机制把发生的事件对象缓存起来，交给自定的的 listener 使用

其中Informor包含多个组件：

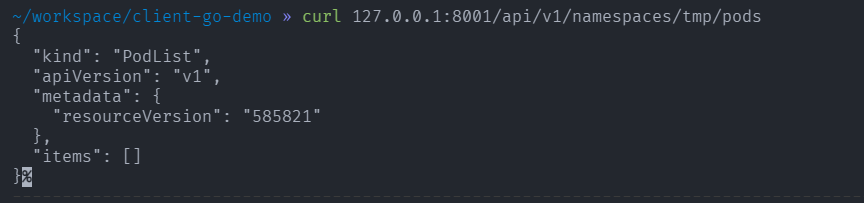
* Reflector：k8s资源发生变化时、触发响应的事件ADD、DELETE、UPDATE...
* DeltaFIFO：分为FIFO以及Delta，分别保存事件类型以及响应的响应函数如ADD、UPDATE等
* Indexer：是client-go在本地的一个缓存、减轻对api-server的访问压力

##### 0、api-server

1、通过 kubectl proxy 命令把 k8s 的 api-server 代理到集群外，然后通过curl 127.0.0.1:8001 即可得到在api-server中的资源类型的存放key值，即资源是通过这种方式存在etcd中的。



2、直接请求api-server，得到相应的数据。如下是tmp命名空间中的pods数据，此时命名空间没有pod；



##### 1、RESTClient、ClientSet、DynamilClient、DiscoveryClient

这些都是对最基本的http请求的封装，可以分别请求某一个Group、Version的资源

但是通过这种方式、只能实现最基本的数据请求，

##### 2、listerwatcher机制

listerwatcher通过注册对某个资源的关注，以响应式的事件的方式得到资源变更。首先调用list函数，得到某种资源、然后调用watch监听资源的变化。

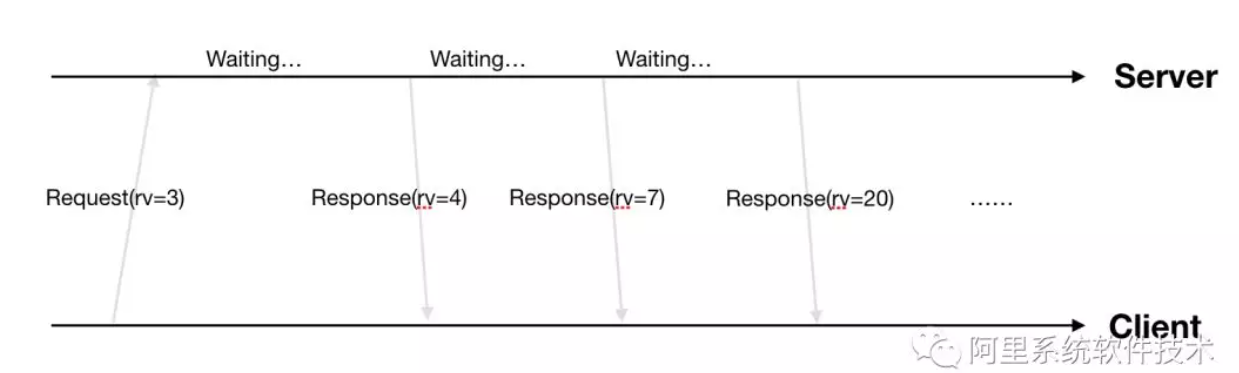
和api-server通信的最底层是listerwatcher机制，通过新建一个newListWatchFromClient 来新建一个对于某种资源的listwatch机制。

##### 3、reflector

reflector把数据存储listwatcher监听到的事件缓存在队列中。然后我们需要从队列中取出资源的变化，然后根据队列中取出来的资源变更对cache缓存进行同步的更新。

ResourceVersion: kubernetes 为每个REST数据加上了一个ResourceVersion字段，该字段由ETCD来保证全局的单调递增，（当ETCD写入数据时，全局的REsourceVersion就会加一，）这样就保证了 不同时刻的数据的ResourceVersion是不同的，最新的数据具有的ResourceVersion更大。这样当客户端发起Watch请求的时候，只需要带上本地缓存数据的最大RV字段，ETCD就会根据RV的值，把具有更大RV的数据按照顺序推给客户端，这样就保证了数据在ETCD以及缓存中的一致性。

因为resurceversion字段全局加一，可能对应到具体的某一种资源的RV不会逐步的加一



##### 4、Informer

到reflector为止，都需要判断事件的类型，根据事件维护客户端的缓存状态，Informer就是封装了这些逻辑，只需要关注于用户逻辑。

\_, controller := cache.NewInformer(lw, &corev1.ConfigMap{}, 0, cache.ResourceEventHandlerFuncs{  
 AddFunc: func(obj interface{}) {  
 configMap, ok := obj.(\*corev1.ConfigMap)  
 if !ok {  
 return  
 }  
 fmt.Printf("created: %s\n", configMap.Name)  
 },  
 UpdateFunc: func(old, new interface{}) {  
 configMap, ok := old.(\*corev1.ConfigMap)  
 if !ok {  
 return  
 }  
 fmt.Printf("updated: %s\n", configMap.Name)  
 },  
 DeleteFunc: func(obj interface{}) {  
 configMap, ok := obj.(\*corev1.ConfigMap)  
 if !ok {  
 return  
 }  
 fmt.Printf("deleted: %s\n", configMap.Name)  
 },  
 })

在NewInformer函数中传入参数，参数包含了AddFunc、UpdateFunc、DeleteFunc等，分别在发生对应的事件的时候相应。有了informer我们只需要关系对事件的处理程序(handler)。

##### 5、sharedInformer

在informer的示例中，对于每一种资源，需要一个informer，而每个informer其实都是需要和api-server通信来获得最新的资源变更数据的。因此加重了api-server的负担。sharedInformer就是针对于同一种资源的informer进行的复用机制。多个处理程序handler可以共享同一个informer。

通过sharedInforemer机制注册的处理函数handler可以在事件发生的时候广播得到这个事件。