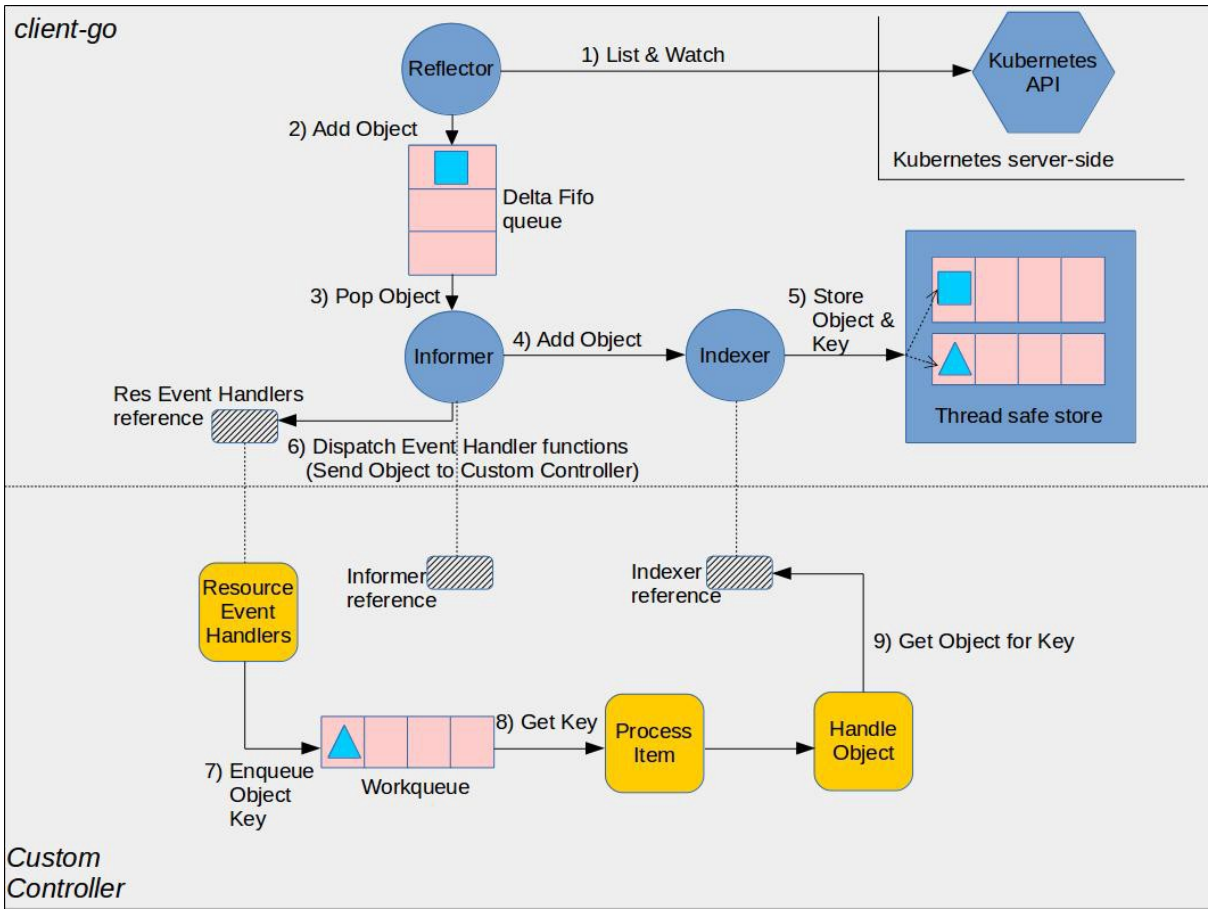
Informer 架构说明

本文主要介绍 Informer 的架构设计

上图是整个 client-go 的完整架构图，或者说是我们要去实现一个自定义的控制器的一个整体流程，其中黄色图标是开发者需要自行开发的部分，而其它的部分是 client-go 已经提供的，直接使用即可。 由于 client-go 实现非常复杂，我们这里先对上图中最核心的部分 Informer 进行说明。在 Informer 的架构中包含如下几个核心的组件：

Informers 是 client-go 中非常重要得概念，接下来我们来仔细分析下 Informers 的实现原理，下图是 client-go 的官方实现架构图：



Reflector（反射器）

Reflector 用于监控（Watch）指定的 Kubernetes 资源，当监控的资源发生变化时，触发相应的变更事件，例如 Add 事件、Update 事件、Delete 事件，并将其资源对象存放到本地缓存 DeltaFIFO 中。

DeltaFIFO

DeltaFIFO 是一个生产者-消费者的队列，生产者是 Reflector，消费者是 Pop 函数，FIFO 是一个先进先出的队列，而 Delta 是一个资源对象存储，它可以保存资源对象的操作类型，例如 Add 操作类型、Update 操作类型、Delete 操作类型、Sync 操作类型等。

Indexer

Indexer 是 client-go 用来存储资源对象并自带索引功能的本地存储，Reflector 从 DeltaFIFO 中将消费出来的资源对象存储至 Indexer。Indexer 与 Etcd 集群中的数据保持完全一致。这样我们就可以很方便地从本地存储中读取相应的资源对象数据，而无须每次从远程 APIServer 中读取，以减轻服务器的压力。

这里理论知识太多，直接去查看源码显得有一定困难，我们可以用一个实际的示例来进行说明，比如现在我们删除一个 Pod，一个 Informers 的执行流程是怎样的：

1. 首先初始化 Informer，Reflector 通过 List 接口获取所有的 Pod 对象
2. Reflector 拿到所有 Pod 后，将全部 Pod 放到 Store（本地缓存）中
3. 如果有人调用 Lister 的 List/Get 方法获取 Pod，那么 Lister 直接从 Store 中去拿数据
4. Informer 初始化完成后，Reflector 开始 Watch Pod 相关的事件
5. 此时如果我们删除 Pod1，那么 Reflector 会监听到这个事件，然后将这个事件发送到 DeltaFIFO 中
6. DeltaFIFO 首先先将这个事件存储在一个队列中，然后去操作 Store 中的数据，删除其中的 Pod1
7. DeltaFIFO 然后 Pop 这个事件到事件处理器（资源事件处理器）中进行处理
8. LocalStore 会周期性地把所有的 Pod 信息重新放回 DeltaFIFO 中去