声明式 API 是如何实现的

声明式 API

和声明式 API 对立的就是命令式 API，可以简单地认为声明式 API 是对命令式 API 的一层封装

释义
- 命令式 API —— 命令机器如何去做事情（How）
- 声明式 API —— 告诉机器我想要什么（What）

对于 MySQL 开发人员来说，需要明确告诉 OS 这一份数据要如何写入磁盘中，所以是命令式 API。
对于 MySQL 使用人员来说，只需要写一条 SQL 告诉 MySQL 我想要什么，MySQL 会自己去判断如何取数，比如该怎么指定执行计划、使用哪些索引等，对用户是屏蔽的，所以是声明式 API

在 Kubernetes 中，声明式 API 使得用户可以不关心集群当前的实际状态，只需要告知 Kubernetes 我们想要什么（比如 2 个 Pod），Kubernetes 会自行地进行一系列的操作，来满足用户的要求

调谐（Reconcile）

调谐可以认为是控制循环+满足期望状态的一个简称，当控制器发现某一个资源并不满足期望条件时，将会主动地去进行调整，不管这个调整是创建、修改还是删除

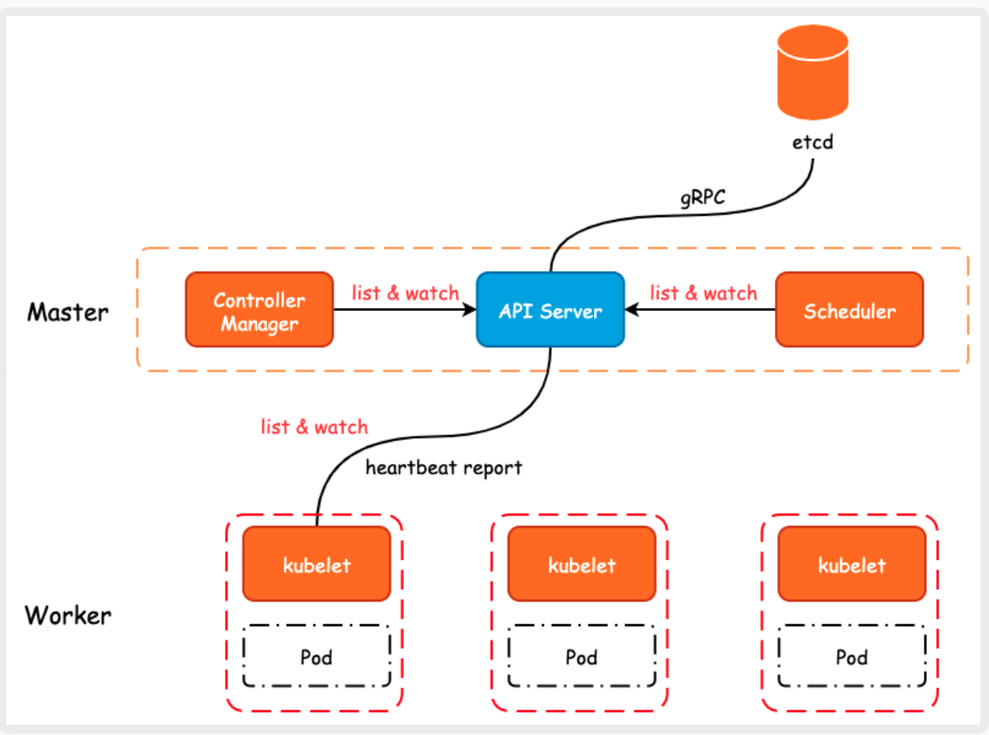
```
while (true) {
   Actual State = 获取对象 E 的实际状态
   Expected State = 获取对象 E 的期望状态

   if (Actual State == Expected State)
      do nothing
   else
      调整对象 E 至期望状态
}
```

左侧的伪代码其实就是 Controller 的工作模式，和事件模型不一样的是，Controller 不会等待事件通知才做相应的调谐，而是更积极主动地去发现"不同"，从而做出调整

实际上，Kubernetes 中不仅仅只有 PULL 模型，即主动地去获取对象期望状态，同时还存在事件通知机制，也就是 PUSH 模型。这两者的综合使用使得 Kubernetes 能够更快地感知到集群状态的变化，从而即时地对资源进行调整

Kubernetes 声明式 API 的具体实现其实非常类似于微服务中的数据同步。我们可以通过主动推送（PUSH）来向外广播增量数据，同时外部系统也可以主动地来拉取（PULL）数据以确保数据的完整性，避免因为 PUSH 过来的数据出现丢失或者是处理该条数据时出错导致的数据不一致问题

PUSH 保证数据变更的实时性，PULL 则是一种补偿机制，确保数据的完整性

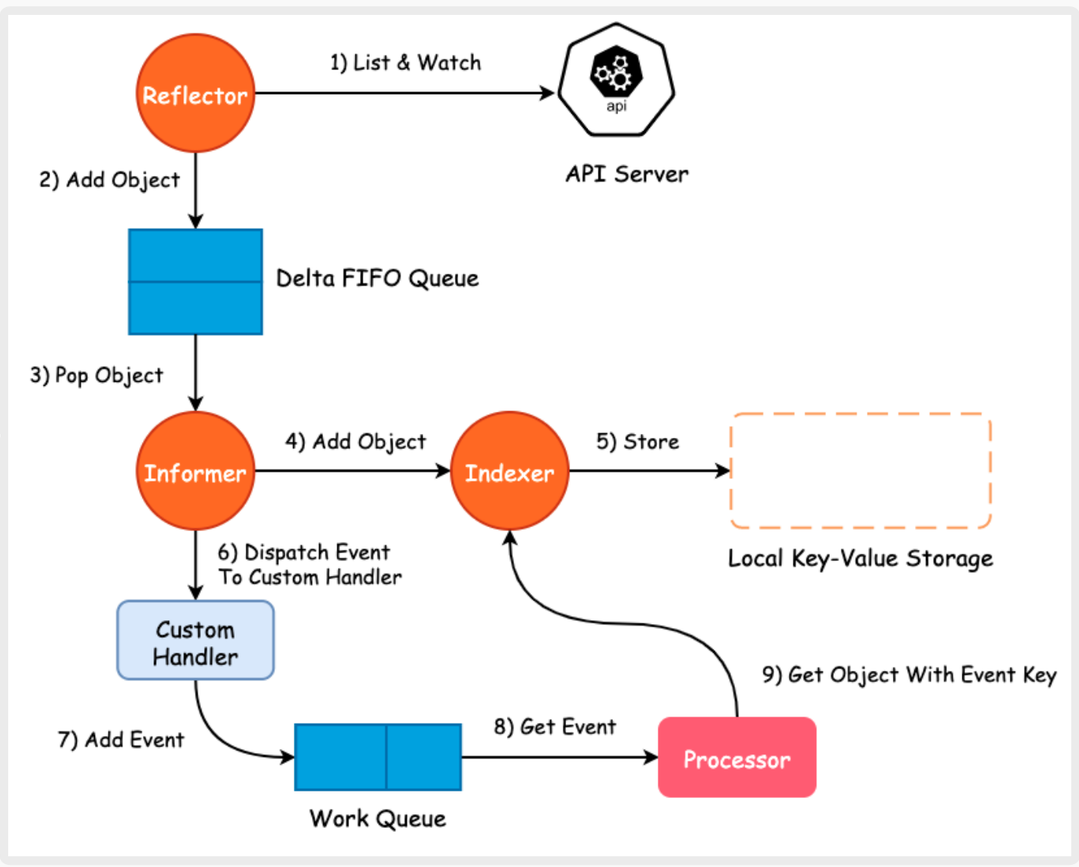


左图为 k8s 简单组成结构图，可以看到，API Server 占有绝对重要的地位，k8s 中所有的组件都要与之进行通信—因为 API Server 是唯一能够和 etcd 通信的组件

图中所绘制的 list & watch 其实就是我们上面讨论的 PULL & PUSH 的具体实现

Informer

etcd 能够作为 k8s 的存储部件不仅仅是因为它具有强一致性的特性，同时还因为 etcd 提供了 Watch API，也就是监听数据的改变。在有了 etcd Watch API 以后，那么 API Server 就可以和 etcd 建立长连接，来监听对象的改变



Informer 工作流程

1. 首先，Reflector 会和 API Server 建立长连接，并使用 ListAndWatch 方法获取并监听某一个资源的变化
   - List 方法将会获取某个资源的所有实例，仅在初始化 Informer 时调用
   - Watch 方法则监听资源对象的创建、更新以及删除事件，获取到的事件称之为一个增量（Delta），该增量会被放进 Delta FIFO Queue，防止 Customer Handler 拖垮 Informer
2. 接着，Informer 会不断的从 Delta FIFO Queue 中 pop 增量事件，并根据事件的类型来决定是插入、更新或者是删除本地缓存，也就是 Local Key-Value Sotrage
3. Informer 的另外一个职责就是根据事件类型来触发事先注册好的 Event Handler。在回调函数中通常只会做一些简单的过滤处理，然后将该事件丢到 Work Queue 这个工作队列中
4. **接下来就是 Controller 的表演时间了，也就是上图中的 Processor。Processor 根据 Work Queue 中的 key 从 Indexer 中获取实际的对象，这个对象就是控制循环中的"期望状态"。然后再根据相关的 API 获取对象在集群中的实际情况，来进行相应的处理，这就是 Controller 的工作模式**

综上，Informer 的本质其实就是一个本地缓存，并且根据 API Server 提供的 Watch API 来更新缓存，以减轻 API Server 和 etcd 的查询压力

小结

声明式 API 的本质就是让 Kubernetes 来完成"命令式 API"的实现，我们只需要提供对某一个对象的期望状态给 Kubernetes，那么如何让集群对象满足我们的期望，完全由 Kubernetes 控制，这其实就是 Controller 所做的工作

```
while (true) {
   Actual State = 获取对象 E 的实际状态
   Expected State = 获取对象 E 的期望状态

   if (Actual State == Expected State)
      do nothing
   else
      调整对象 E 至期望状态
}
```

Controller 的工作原理其实就是左侧的伪代码，最为复杂的地方其实就在于当期望状态和实际状态不一致时，Kubernetes 需要选择一条合适的"路径"，使得实际状态满足期望状态

"获取对象 E 的期望状态"是一个典型的读多写少的场景，而 Kubernetes 又不希望引入额外的外部组件，所以使用 Informer 机制来实现缓存，并且缓存的更新由 API Server 进行主动推送，减少 etcd 存储的压力

最后，我们以创建一个 ReplicSet 为例，来完整地描述组件间的通信和 Informer 机制的运行