这里说的控制器，准确来说应该是 ***control loop***，中文可以称为控制循环。这个术语在机器人学和自动化控制领域，是一个用来管理系统状态的一个死循环。下面以恒温器来举个简单的例子。

当我们设置了房间温度之后，恒温器就会记录下这个温度当成期望状态（***desired state***），而房间的实际温度则是当前状态（***current state***）。恒温器会不断比较目标温度和当前温度，并根据之间的温度高低来觉得是制冷还是加热。

1. 概述

我们前面介绍的 ReplicaController、Deployment 等都是 Kubernetes 中的控制器，控制器的工作方式就是控制循环。控制循环的工作原理比较简单，就是不断比较资源的状态是不是期望状态，如果不是期望状态，则执行一些动作，否则什么都不做。

```
while:
  currentState = getResourceCurrentState()
  desiredState = getResourceDesiredState()
  if currentState == desiredState:
    noop
  else:
    reconcileLoop()
```

举个具体的例子，下面是一个简单的 Deployment 的 yaml 定义。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

这个 Deployment 的期望状态就是带有标签 app=nginx 的 Pod 个数为 2 个，那么控制循环将不断的判断集群中是不是有 2 个带有标签 app=nginx 的 Pod。如果低于 2 个，就启动新的 Pod；如果多于 2 个，就将多余的 Pod 杀掉，这个就是 reconcile。

2. 控制器模式

在 Kubernetes 中有很多控制器，比如 Deployment、DaemonSet。每种控制器都会负责至少一种 Kubernetes 资源类型。这些资源对象的 spec 中有一个字段域用来表示期望状态。控制器的工作就是不断将该资源的当前状态调整不断逼近期望状态。

那么控制器是如何去调整的呢？一般有两种方式：通过 API Server 控制或者直接控制。

通过 API Server 控制

我们以 Kubernetes 的原生控制器 Job Controller 为例。原生控制器通过和集群的 API Server 交互来管理状态。

Job 作为 Kubernetes 中的一种资源类型，通过启动一个或者多个 Pod 去执行指定 task。当 Job 控制器拿到新任务时，它会保证一组 Node 节点上的 kubelet 可以运行正确数量的 Pod 来完成工作。Job 控制器并不会直接运行 Pod 或者 Container，而是通过 API Server 去创建或者移除 Pod。当我们创建新的 Job，期望状态就是 Job 变成 Completed 状态。Job 控制器需要做的事情就是不断逼近期望状态：创建 Pod 去执行 Job 的 task。

控制器也会跟新资源对象的状态，比如 Job 的 task 都执行完了之后，Job 控制器更新 Job 对象，标记为 Finished。

直接控制

和 Job 控制器对比，有些控制器需要对集群外的资源进行处理。比如，我们使用一个控制循环去确保我们的集群有足够的 Node 数，那么控制器需要集群外的系统去为我们的集群创建新的 Node。

和外部状态交互的控制器通过 API Server 获取到期望状态，然后直接和外部系统交互来完成这个调节的过程。

3. 状态

上面有提到控制循环里面的两种状态：当前状态和期望状态，那么这两种状态是如何获取的呢？

当前状态

Kubernetes 也可以认为是一种 server-agent 架构，server 可以是 API Server 等，agent 是运行在每个 Node 上的 kubelet。当前状态正式由 kubelet 来上报的。kubelet 通过心跳汇报其所在的节点上面运行的容器状态和节点状态。

除此之外，控制器还可以主动收集它感兴趣的信息。

期望状态

Kubernetes 中的所有 API 对象都会提交给 API Server，然后保存到 ETCD 中。期望状态来源于用户提交的 YAML 文件，也是存储在 ETCD 中，但是控制器一般不会直接去和 ETCD 交互，而是通过 API Server 来中转。ETCD 提供了一种 watch 机制来实现对资源的监控，这个映射到 Kubernetes 中叫 Informer。

下面举个例子，看一下 Deployment 的控制器模式实现。

1. Deployment 控制器从 API Server 获取到所有带有特定标签的 Pod，并统计数目，这个就是实际状态；
2. Deployment 对象中的 Replica 字段的值是期望状态；
3. Deployment 控制器比较这两个状态，然后根据比较结果来决定是创建新的 Pod，还是删除已有的 Pod。
4. 设计

作为设计的一个原则，Kubernetes 使用了很多控制器，每个控制器管理集群状态的一个特定方面。最常见控制器使用一种类型的资源作为它的期望状态，然后控制另外一种类型的资源来帮助集群接近期望状态。比如，Job 控制器会同时监听 Job 对象和 Pod 对象：监听 Job 对象来发现新的 workload，监听 Pod 对象来运行 task 并且等 workload 完成。（这个例子中 Job 是由其他系统创建的，Job 控制器只负责创建 Pod）。

使用简单、分开的多个控制器优于一个集成所有功能于一体的单体控制环。因为控制器会失败，当然 Kubernetes 针对这种情况做了 failover 设计。

> **注意：** 可以有多个控制器来创建或者更新相同类型的对象。针对这种情况，Kubernetes 控制器确保他们只关心和它们控制资源相关联的资源。 例如，你可以有 Deployments 和 Jobs；它们都可以创建 Pod。Job 控制器不删除 Deployment 创建的 Pod，因为有信息 (标签让控制器可以区分这些 Pod。

5. 运行控制器的方式

Kubernetes 自带有一组内置的控制器，运行在 kube-controller-manager 内，代码在 `kubernetes/pkg/controller/`。这些内置的控制器提供了重要的核心功能。

```
$ cd kubernetes/pkg/controller/
$ ls -d */
deployment/             job/                    podautoscaler/
cloud/                  disruption/             namespace/
replicaset/             serviceaccount/         volume/
cronjob/                garbagecollector/       nodelifecycle/
replication/            statefulset/            daemon/
...
```

Deployment 控制器和 Job 控制器是 Kubernetes 内置控制器的典型例子。Kubernetes 运行一个弹性的控制平面，所以如果任意内置控制器失败了，控制平面的另外一部分会接替它的工作。

你会发现控制平面外面运行的控制器，扩展了 Kubernetes 的能力。或者，如果你愿意，你也可以写一个新控制器。你可以以一组 Pod 来运行你的控制器，或者运行在 Kubernetes 外面。什么是最合适的控制器，这将取决于特定控制器的功能。

6. 总结

本篇文章讨论 Kubernetes 的控制器模式的设计，并以 Deployment 和 Job 控制器为例来分析了 Kubernetes 如何通过 ***control loop*** 来实现对整个集群的资源的状态的监听和控制。

实际上，Kubernetes 中的其他控制器也都和 Deployment 控制器非常相似，***control loop*** 的最后执行结果要么创建新的 Pod，要么删除已存在的老的 Pod，或者更新等。

control loop 可以认为是 Kubernetes 项目进行容器编排的核心原理，而且这也是 Kubernetes 引以为豪的声明式编程范式的基础。

}

一手微信itit11223344