92Kubernetes 系列（八十六）K8s Operator — 管理状态

在这个专门针对 Kubernetes Operator 的系列中，我们今天将看到一些非常有用的内容，特别是对于调试或了解自定义资源的状态。在这个专门针对 Kubernetes Operator 的系列中，我们今天将看到一些非常有用的内容，特别是对于调试或了解自定义资源的状态。

状态的目标是什么？

与任何其他 Kubernetes 资源一样，资源状态为我们提供了有关其当前状态的一些信息。它可以用很多东西来表示，从布尔值来表示是否一切正常，或者每个生成的子资源的名称。

状态结构定义

状态必须在填充之前定义其结构。该结构在文件 `api/../xxx_types.go` 中定义。

例如：

```
type MyProxyStatus struct {
}
```

一开始完全是空的，您可以在那里添加您想要/需要的内容！

在我们的示例中，我们将完成之前所做的工作（一个资源 MyProxy，它部署了一个带有 2 个 pod 的 nginx 部署），并且我们将列出 nginx 部署生成的所有 pod 名称。

```
type MyProxyStatus struct {
   PodNames []string `json:"pod_names"`
}
```

> 当我们更新 api 文件夹中的文件时，不要忘记执行 `make manifests & make generate` ！

填充状态

现在我们已经有了资源的结构，我们可以转到控制器，更准确地说是在协调函数中。

对于我们的示例，要做的第一件事是尝试获取与我们的部署相关的 pod 列表。

```
podList := &corev1.PodList{}
listOpts := []client.ListOption{
   client.InNamespace("test_ns"),
   client.MatchingLabels(map[string]string{
      "test_label": myProxy.Spec.Name,
   }),
}
if err = r.List(ctx, podList, listOpts...); err != nil {
   log.Error(err, "Failed to list pods", "Example.Namespace", "test_ns")
   return ctrl.Result{}, err
}
podNames := getPodNames(podList.Items)
```

如您所见，为了检索项目列表，我们使用过滤器。在我们的例子中，我们使用其中两个：

- 应部署 pod 的命名空间的名称
- pod 必须具有的标签 根据您是否正在寻找精确的内容，您可以更新此过滤器列表以找到您要查找的内容。

然后，我们可以像更新其规范一样更新 myProxy 实例。

```
if !reflect.DeepEqual(podNames, myProxy.Status.PodNames) {
    myProxy.Status.PodNames = podNames
    err := r.Status().Update(ctx, myProxy)
    if err != nil {
        log.Error(err, "Failed to update MyProxy status")
        return ctrl.Result{}, err
    }
}
```

就是这样！您现在可以管理自定义资源的状态！

在本系列的下一篇文章中，我们将了解如何在有条件的情况下完成此状态部分！

我希望它能对你有所帮助，如果你有任何问题（没有愚蠢的问题）或某些点你不清楚，请不要犹豫，在评论中添加你的问题。