43Kubernetes 系列（四十）Helm — 流程控制

Kubernetes 系列（四十）Helm — 流程控制

Helm 流程控制使用 if/else, with 以及 range

Helm 的模板语言提供一下的流程控制结构：

• if/else — 创建条件区块

• with — 指定一个新的作用区域

• range — 提供类似"for each" 的循环结构

在本文中，我们将讨论 helm 流程控制的一些案例。

If/else

我们可以在 helm chart 使用 if/else,就像其他一些编程语言：

```
{{ if PIPELINE }}
  # Do something
{{ else if OTHER PIPELINE }}
  # Do something else
{{ else }}
  # Default case
{{ end }}
```

在下面的几种情况中，PIPELINE 的值将作为 False:

- 一个 boolean 值 false
- 一个数字 0
- 一个空字符串
- 一个 nil 值(空或者 null)
- 一个空的集合（map，slice,tuple,dict,array)

现在，让我们看看在 helm 中如何使用它。

假设，我们的 values.yaml 文件内容如下:

```
# values.yaml
configMap:
  data:
    mode: dark
    env: test
```

现在我们创建一个 configmap.yaml 文件，我们在里面使用 if /else :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  {{ if .Values.configMap.data.mode}}          # boolean check
  mode: dark
  {{ else }}
  mode: light
  {{ end }}
  {{ if eq .Values.configMap.data.env "prod"  }}    # string check
  env: prod
  {{ else if eq .Values.configMap.data.env "dev" }}
  env: dev
  {{ else }}
  env: test
  {{ end }}
```

通过 **helm template** 命令生成清单内容:

```
>> helm template ~/webserver---
# Source: webserver/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: release-name-configmap
data:

  mode: dark

  env: test
```

控制空白内容

在上面的演示中，我们可以看到一些黑色的行/空间。要删除前导空格，可以在if/else语句前使用破折号 `{{-` 。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  {{- if .Values.configMap.data.darkMode }}          # boolean check
  mode: dark
  {{- else }}
  mode: light
  {{- end }}
  {{- if eq .Values.configMap.data.env "prod"  }}    # string check
  env: prod
  {{- else if eq .Values.configMap.data.env "dev" }}
  env: dev
  {{- else }}
  env: test
  {{- end }}
```

```
>> helm template ~/webserver---
# Source: webserver/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: release-name-configmap
data:
  mode: dark
  env: test
```

在前面的演示中，我们使用了" eq "函数和if语句。我们可以根据需要使用其他逻辑和流量控制功能。

让我们试着和if语句一起使用 and 函数;为此，我们将向值中添加更多的条目。Yaml文件:

```yaml
# values.yaml

configMap:
  data:
    darkMode: true
    os: mac
    env: test
```

我们想要实现的是，如果** darkMode: true ** **和 os: mac 定义在values.yaml文件中，然后是我们的**configmap.yaml

我们必须修改**configmap。Yaml **模板文件相应实现上述目标:

```yaml
# configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  {{- if and (.Values.configMap.data.darkMode) ( eq .Values.configMap.data.os "mac") }}
  mode: dark
  {{- else }}
  mode: light
  {{- end }}
  {{- if eq .Values.configMap.data.env "prod"  }}
  env: prod
  {{- else if eq .Values.configMap.data.env "dev" }}
  env: dev
  {{- else }}
  env: test
  {{- end }}
```

在上面的演示中，我们使用了 and 函数根据需求进行决策。

现在，生成模板来验证 configmap.yaml 文件是否正常运行:

```
>> helm template ~/webserver/

---
# Source: webserver/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: release-name-configmap
data:
  mode: dark
  env: test
```

```
  env: test
```

使用“with”修改作用域

我们将要讨论的下一个控制结构是with action。这控制变量作用域。在前面，我们已经看到** .Values **告诉模板在图表中的

```
{{ with PIPELINE }}

  {{- toYaml . | nindent 2 }}
{{ end }}
```

在 `with` 范围内 `.` 并不指向根对象。在with对象中 `.` 用于访问当前对象的作用域。

让我们看一些例子来更好地理解。

Example 1: 假设我们有values.yaml文件，包含以下条目:

```
# values.yaml

configMap:
  data:
    env: test
    platfrom:
     - java
     - python
     - golang
```

我们将创建一个configmap.yaml模板，其中 `with` 将用于变量作用域:

```
#configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  env: {{ .Values.configMap.data.env }}
  {{- with .Values.configMap.data.platfrom }}
  platfrom: {{- toYaml . | nindent 2 | upper  }}
  {{- end }}
```

现在，生成目标文件:

```
>> helm template ~/webserver---
# Source: webserver/templates/configmap.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: release-name-configmap
data:
  env: test
  platfrom:
  - JAVA
  - PYTHON
  - GOLANG
```

Example 2: 让我们为这些值添加一些额外的条目:

```
# values.yaml

configMap:
  data:
    env: test
    platfrom:
     - java
     - python
     - golang
    conf:
      os: linux
      database: mongo
```

然后我们会修改configmap.yaml模板文件。这样我们就可以检索添加到值中的附加数据:

```
# configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  env: {{ .Values.configMap.data.env }}
  {{- with .Values.configMap.data.platfrom }}
  platfrom: {{- toYaml . | nindent 2 | upper  }}
  {{- end }}
  {{- with .Values.configMap.data.conf }}
  operating-system: {{ .os }}
  database-name: {{ .database }}
  {{- end }}
```

最后，生成目标文件:

```
>> helm template ~/webserver
# Source: webserver/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: release-name-configmap
data:
  env: test
  platfrom:
  - JAVA
  - PYTHON
  - GOLANG
  operating-system: linux
  database-name: mongo
```

正如我们前面所讨论的，在with块中引用一个特定的对象。但是在某些情况下，我们可能需要访问根对象或其他不属于当前作用

```
{{- with .Values.configMap.data.conf }}
  operating-system: {{ .os }}
  database-name: {{ .database }}
  k8s-namespace: {{ .Release.Namespace }}
{{- end }}
```

然后上面的代码将抛出如下的错误:

```
Error: template: webserver/templates/configmap.yaml:14:28: executing "webserver/templates/configmap.y
```

因为，我们引用的是驻留在当前作用域之外的对象。

为了解决这个问题，我们可以在**Release**对象前面使用**$符号。因为根作用域也由**$符号表示。

```
{{- with .Values.configMap.data.conf }}
  operating-system: {{ .os }}
  database-name: {{ .database }}
  k8s-namespace: {{ $.Release.Namespace }}
{{- end }}
```

现在，上面的代码可以正常工作了，因为我们在Release对象前面添加了一个$符号。

Range

`range` **类似于 for/foreach**循环，就像其他编程语言一样。在Helm的模板语言中，迭代集合的方法是使用** `range` ** 操作

假设我们有个values.yaml文件，包含以下条目:

```
# values.yaml
configMap:
  data:
    env: test
```

```
  platfrom:
    - java
    - python
    - golang
```

在values.yaml文件中我们有一个"platform"定义列表，让我们创建一个configmap.yaml模板文件使用** range ** 操作符检

```
# configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  env: {{ .Values.configMap.data.env }}
  platfrom: |
  {{- range .Values.configMap.data.platfrom }}
    - {{ . | title | quote }}
  {{- end }}
```

最后，生成目标文件:

```
>> helm template ~/webserver---
# Source: webserver/templates/test.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: release-name-configmap
data:
  env: test
  platfrom: |
    - "Java"
    - "Python"
    - "Golang"
```

欢迎关注我的公众号"**云原生拓展**"，原创技术文章第一时间推送。