开发 VMware vRealize Orchestrator 的 Web 服务客户端

vRealize Orchestrator 7.6

vmware®

您可以从 VMware 网站下载最新的技术文档：
https://docs.vmware.com/cn/。
VMware 网站还提供了最近的产品更新。
如果您对本文档有任何意见或建议，请将反馈信息发送至：
docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

北京办公室
北京市
朝阳区新源南路 8 号
启皓北京东塔 8 层 801
www.vmware.com/cn

上海办公室
上海市
淮海中路 333 号
瑞安大厦 804-809 室
www.vmware.com/cn

广州办公室
广州市
天河路 385 号
太古汇一座 3502 室
www.vmware.com/cn

目录

开发 VMware vRealize Orchestrator 的 Web 服务客户端 5

1 开发 Web 服务客户端 6

2 使用 vRealize Orchestrator REST API 7

针对 Orchestrator 和第三方系统进行身份验证 8

使用通过 Orchestrator REST API 进行的 vCenter Single Sign-On 身份验证 8

访问 Orchestrator REST API 的参考文档 13

使用 Java REST SDK 13

工作流操作 14

查找工作流并检索其定义 14

运行工作流 17

根据工作流展示验证工作流输入参数后再运行工作流 19

在工作流运行时与其交互 23

检索工作流的交互 29

访问工作流架构 29

使用任务 30

创建任务 30

修改任务 31

查看任务的状态 31

在 Orchestrator 清单中查找对象 32

按类型和 ID 查找对象 32

按关系查找对象 33

应用筛选器 34

导入和导出 Orchestrator 对象 35

导入工作流 35

导出工作流 35

导入操作 36

导出操作 36

导入软件包 37

导出软件包 38

导入资源 38

导出资源 39

导入配置元素 39

导出配置元素 39

删除 Orchestrator 对象 40

删除工作流 40

删除操作 40

删除软件包 41
删除资源 41
删除配置元素 41
在 Orchestrator 对象上设置权限 42
REST API 权限 42
检索工作流的权限 43
删除工作流的权限 43
设置工作流权限 43
检索操作的权限 44
删除操作的权限 44
设置操作权限 44
检索软件包的权限 45
删除软件包的权限 45
设置软件包权限 46
检索资源的权限 46
删除资源的权限 46
设置资源权限 47
检索配置元素的权限 47
删除配置元素的权限 48
设置配置元素的权限 48
执行插件操作 48
检索关于插件的信息 49
导入插件 49
导出插件 49
启用或禁用插件 50
执行服务器配置操作 50
检索有关 Orchestrator 服务器配置的信息 50
导入 Orchestrator 服务器配置 50
导出 Orchestrator 服务器配置 51
执行标记操作 51
标记对象 52
取消标记对象 52
列出对象标记 53
按类型列出已标记的对象 53
列出标记所有者 53
按用户列出标记 53
按用户列出按标记名称筛选的标记 54
按用户移除标记 54

开发 VMware vRealize Orchestrator 的 Web 服务客户端

《开发 VMware vRealize Orchestrator 的 Web 服务客户端》提供了有关为 VMware® vRealize Orchestrator 开发 Web 服务客户端的信息。

Orchestrator 提供了 Web 服务 API，因此您可以开发应用程序以通过 Web 访问和使用工作流。Orchestrator 提供了表述性状态转移 (REST) API，您可借此在工作流中执行各种操作。

目标读者

本文档提供的信息主要面向想要通过 RESTful Web 服务在网络中访问 Orchestrator 进程的 Web 应用程序开发人员。

开发 Web 服务客户端

1

VMware vRealize Orchestrator 提供了 Web 服务 API，可用于开发相关应用程序以通过 Web 访问工作流。Orchestrator Web 服务 API 的主要目的是允许您在基于 Web 的自定义应用程序中集成 Orchestrator 工作流。

Orchestrator 提供基于表述性状态转移 (REST) API 的 Web 服务 API。Orchestrator REST API 公开了 Orchestrator 清单中的对象以及可通过预定义 URL 访问的作为资源安装的插件的清单。这些 URL 中的 HTTP 请求会通过工作流触发操作。Orchestrator REST API 通过一组 RESTful Web 服务将清单对象公开为资源，您可用来检索工作流定义、运行工作流、检查正在运行的工作流的状态、取消工作流运行、处理等待中的用户交互、检索工作流展示等。

使用 vRealize Orchestrator REST API 2

Orchestrator REST API 提供的功能可让您通过 HTTP 与 Orchestrator 服务器直接通信，并可在工作流中执行各种工作流相关操作。

Orchestrator REST API 公开了 Orchestrator 服务器清单中的对象，以及作为资源安装在预定义 URL 中的插件。您可以在这些 URL 处进行 HTTP 调用，以在 Orchestrator 中触发操作。这样，您就可以在工作流中执行各种任务：

- 运行工作流、调度工作流、检索工作流运行、响应用户交互，以及取消工作流运行。
- 检索有关工作流的详细信息（例如其输入和输出参数及其展示）。
- 检索有关工作流运行的详细信息（例如其状态、生成的日志、开始日期和结束日期）。
- 浏览 Orchestrator 清单和安装的插件。
- 导入并导出工作流、操作和软件包。

注 请勿使用 vRealize Orchestrator REST API 安装插件。

通过使用 Orchestrator REST API，您可以在可使用任意编程语言构建的自定义应用程序中集成 Orchestrator 工作流。

Orchestrator REST API 还提供了电子标记支持及响应数据缓存机制。

本章讨论了以下主题：

- 针对 Orchestrator 和第三方系统进行身份验证
- 访问 Orchestrator REST API 的参考文档
- 使用 Java REST SDK
- 工作流操作
- 使用任务
- 在 Orchestrator 清单中查找对象
- 导入和导出 Orchestrator 对象
- 删除 Orchestrator 对象
- 在 Orchestrator 对象上设置权限

- 执行插件操作
- 执行服务器配置操作
- 执行标记操作

针对 Orchestrator 和第三方系统进行身份验证

您必须对通过 Orchestrator REST API 发起的 HTTP 请求内的 Orchestrator 进行身份验证。如果您使用 Orchestrator REST API 访问第三方系统上的资源（如 vCenter Server 或 vRealize Automation），则必须也对该系统进行身份验证。

例如，若要访问 Orchestrator 清单中的所有工作流，您必须对 Orchestrator 进行身份验证。然而，若要对 vCenter Server 运行工作流，您必须对 Orchestrator 和 vCenter Server 进行身份验证。

Orchestrator REST API 的身份验证方案会有所不同，具体取决于您是否将使用 vRealize Automation 或 vSphere 的 Orchestrator 配置为身份验证提供程序。如果 Orchestrator 使用 vCenter Single Sign-On，根据具体配置，您可以使用 vCenter Single Sign-On 服务器颁发的密钥持有人令牌进行身份验证。如果 Orchestrator 配置为使用 vRealize Automation，您可以通过 OAuth 持有者访问令牌进行身份验证。

如果在 Orchestrator REST API 的顶层 URL 发起 HTTP 请求，则无需对 Orchestrator 进行身份验证。Orchestrator REST API 的顶层 URL 为 https://*orchestrator_host*:*port*/vco/api/。

注 外部 Orchestrator 的默认端口号为 8281。嵌入到 vRealize Automation 中的 Orchestrator 实例的默认端口号为 443。

REST API 顶层 URL 的 GET 请求会返回可通过 API 访问的所有资源的 URL。若要在这些 URL 发起 HTTP 请求，您必须对 Orchestrator 进行身份验证。

使用通过 Orchestrator REST API 进行的 vCenter Single Sign-On 身份验证

如果 Orchestrator 配置为通过 vSphere 身份验证模式使用 vCenter Single Sign-On 服务器，您需要主体密钥持有人令牌才可通过 Orchestrator REST API 访问 Orchestrator 中的系统对象。若要通过 Orchestrator 服务器访问使用 vCenter Single Sign-On 服务器的 vCenter Server 或第三方系统，您需要 Orchestrator 的委派密钥持有人令牌和您的主体令牌。

如果 Orchestrator 配置为使用 vCenter Single Sign-On 服务器，您必须使用有效的凭据进行身份验证，且将由 Orchestrator 管理密钥持有人令牌。

访问 Orchestrator 中的系统对象

您可以在清单的 URL 以及 REST API 的目录服务处访问 Orchestrator 中的系统对象。

- https://*orchestrator_host*:*port*/vco/api/inventory/System/
- https://*orchestrator_host*:*port*/vco/api/catalog/System/

访问 Orchestrator 中的系统对象时，您需要将发起的 HTTP 请求中 `Authorization` 标头内的主体密钥持有人令牌传递到清单或目录服务。

例如，若要检索类型 `Workflow` 的所有系统对象，您需要在 https://*orchestrator_host*:*port*/vco/api/catalog/System/Workflow/ 发起 `GET` 请求。若要对 Orchestrator 进行身份验证，您需要传递请求中 `Authorization` 标头内的主体密钥持有人令牌。

访问第三方系统中的对象

若要通过 Orchestrator REST API 在注册到 vCenter Single Sign-On 服务器的第三方系统中执行操作，您必须对 Orchestrator 和第三方系统进行身份验证。您需要在通过 Orchestrator REST API 发起的 HTTP 调用中包含两个标头。

- `Authorization`。您必须传递此标头中的主体密钥持有人令牌。
- `VCOAuthorization`。您必须传递此标头中的 Orchestrator 的委派密钥持有人令牌。您必须从 vCenter Single Sign-On 服务器获取 Orchestrator 的委派令牌。Orchestrator 使用委派令牌代表您对第三方系统进行身份验证。

例如，若要通过 Orchestrator REST API 运行使用虚拟机的工作流，您要访问同时位于 Orchestrator 和 vCenter Server 中的资源。若要对 Orchestrator 和 vCenter Server 进行身份验证，您必须传递所发起请求中的 `Authorization` 标头内的主体密钥持有人令牌，以及 `VCOAuthorization` 标头内的委派令牌。这样，您可以使用主体令牌对 Orchestrator 进行身份验证，而 Orchestrator 使用委派令牌代表您对 vCenter Server 进行身份验证。

vCenter Single Sign-On 服务器将 Orchestrator 视为解决方案，并且会使用唯一用户名向 vCenter Single Sign-On 服务器注册每个解决方案。通过将 Orchestrator 的解决方案用户名和主体密钥持有人令牌传递到 vCenter Single Sign-On 服务器，请求 Orchestrator 的委派令牌。vCenter Single Sign-On 服务器颁发的令牌是供 Orchestrator 代表您对第三方系统进行身份验证时使用的委派密钥持有人令牌。

示例：以 vCenter Single Sign-On 模式获取会话

以下示例代码以 vCenter Single Sign-On 模式获取会话。

```
URI uri = URI.create("https://orchestrator-server:8281/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);

//provide the address of the vCenter Single Sign-On server
URI ssoUri = URI.create("https://sso-server:7444/ims/STSService?wsdl");

//set the tokens to be valid for an hour
long lifeTimeSeconds = 60 * 60;

//create a factory for vCenter Single Sign-On tokens
SsoAuthenticator sso = new SsoAuthenticator(URI ssoUri, URI adminUri, VcoSessionFactory
vcoSessionFactory, long lifeTimeSeconds);

//provide vCenter Single Sign-On credentials
```

```
SsoAuthentication authentication = sso.createSsoAuthentication("username", "password");

VcoSession session = sessionFactory.newSession(authentication);
//use session here
```

获取 Orchestrator 解决方案用户名

vCenter Single Sign-On 服务器将 Orchestrator 视为解决方案，并且会使用唯一用户名向 vCenter Single Sign-On 服务器注册每个解决方案。若要从 vCenter Single Sign-On 服务器请求 Orchestrator 的委派密钥持有人令牌，您需要 Orchestrator 的解决方案用户名。

前提条件

验证您拥有 vCenter Single Sign-On 颁发的有效主体密钥持有人令牌。

步骤

1 在 Orchestrator 解决方案用户名的 URL 发起 `GET` 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/users/
```

2 在请求的 `Authorization` 标头中提供主体密钥持有人令牌。

响应中的 `<user solution-user="OrchestratorSolutionUserName"/>` 元素包含了 Orchestrator 的解决方案用户名。以下是 Orchestrator 解决方案用户名的示例。

```
<user xmlns="http://www.vmware.com/vco" admin-rights="true" solution-
user="vCO-15d98795afa5b0d6f47ee3aeab3">
```

后续步骤

使用 Orchestrator 解决方案用户名以及主体密钥持有人令牌，从 vCenter Single Sign-On 服务器请求委派密钥持有人令牌。

使用带有已配置的 vRealize Automation 身份验证的 VRealize Orchestrator REST API SDK

您可以在多租户或单租户环境中使用带有已配置的 vRA 身份验证的 REST API SDK。

要获取以下代码所需的身份验证令牌 (Oauth2.0)，请参见知识库文章《使用 Oauth2.0 身份验证的 vRO REST API 授权 (2148518)》。

注 以 VRealize Automation 身份验证模式获取会话

以下示例代码在单租户和多租户环境中以 vRealize Automation 身份验证模式获取会话。

- 如果未启用多租户功能：

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);
String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTRiM2UtYmYyYi04ZmRmNzY3N"
+
"GZiZWEiLCJwcm4iOiJhZG1pbmlzdHJhdG9yQFZTUEhFUkUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxvY2FsIiwidXNlcl9
pZCI6Ij"
+
"MiLCJhdXRoX3RpbWUiOjE1MDIyMDIxMTAsImlzcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLmN"
+
"vbS9TQUFTL3QvdnNwaGVyZS5sb2NhbC9hdXRoIiwiYXVkIjoiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJ1LmVuZy52bXdhc
mUuY29tL"
+
"1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvb2F1dGh0b2tlbiIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpvYXNpczpuYW1lczp
"
+
"0YzpTQU1MOjIuMDphYzpjbGFzc2VzOlBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNTAyMjAyMTEwLFwia
WRcIjoxM"
+
"n1dIiwic2NwIjoidXNlciIsImlkcCI6IjAiLCJlbWwiOiJhZG1pbmlzdHJhdG9yQHNmLTI5LTEwLTI5LnNvZi1tYnUuZW5nLnZ
"
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlNEx6bURnIiwiZGlkIjoiIiwid2lkIjoiIiwiZXhwIjoxNTAyMjMwOTEwL
CJpYXQiOj"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybl90eXBlIjoiVVNFUiJ9
."
+ "G9gEQPtmEH5jYab-
IlTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLFkCLwa50OQCQ-qgKutZl21R"
+ "m740qBKLhmBB0NQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

VcoSession session = sessionFactory.newSession(auth);
//Use the session here
```

- 如果启用了多租户功能：
  - 对于常规租户用户：

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);
String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTRiM2UtYmYyYi04ZmRmNzY3N"
+
"GZiZWEiLCJwcm4iOiJhZG1pbmlzdHJhdG9yQFZTUEhFUkUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxvY2FsIiwidXN
lcl9pZCI6Ij"
+
"MiLCJhdXRoX3RpbWUiOjE1MDIyMDIxMTAsImlzcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJl
LmN"
```

```
+
"vbS9TQUFTL3QvdnNwaGVyZS5sb2NhbC9hdXRoIiwiYXVkIjoiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJ1LmVuZy52b
XdhcmUuY29tL"
+
"1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvb2F1dGh0b2tlbiIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpvYXNpczpuYW1
lczp"
+
"0YzpTQU1MOjIuMDphYzpjbGFzc2VzOlBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNTAyMjAyMTEwL
FwiaWRcIjoxM"
+
"n1dIiwic2NwIjoidXNlciIsImlkcCI6IjAiLCJlbWwiOiJhZG1pbmlzdHJhdG9yQHNmLTI5LTEwLTI5LnNvZi1tYnUuZW5
nLnZ"
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlNEx6bURnIiwiZGlkIjoiIiwid2lkIjoiIiwiZXhwIjoxNTAyMjMwO
TEwLCJpYXQiOj"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybl90eXBlIjoiVVNF
UiJ9."
+ "G9gEQPtmEH5jYab-
IlTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLFkCLwa50OQCQ-qgKutZl21R"
+ "m740qBKLhmBB0NQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

VcoSession session = sessionFactory.newSession(auth);
//The operations will be executed in the scope of the tenant authenticated with the token above.

//Use the session below
```

- 对于解决方案用户：

  解决方案用户可以在其自身租户范围内和常规租户范围内工作。这些用户可以更改所执行操作的范围。

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);

exapmle

String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTRiM2UtYmYyYi04ZmRmNzY3N"
+
"GZiZWEiLCJwcm4iOiJhZG1pbmlzdHJhdG9yQFZTUEhFUkUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxvY2FsIiwidXN
lcl9pZCI6Ij"
+
"MiLCJhdXRoX3RpbWUiOjE1MDIyMDIxMTAsImlzcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJl
LmN"
+
"vbS9TQUFTL3QvdnNwaGVyZS5sb2NhbC9hdXRoIiwiYXVkIjoiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJ1LmVuZy52b
XdhcmUuY29tL"
+
"1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvb2F1dGh0b2tlbiIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpvYXNpczpuYW1
lczp"
+
"0YzpTQU1MOjIuMDphYzpjbGFzc2VzOlBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNTAyMjAyMTEwL
```

```
FwiaWRcIjoxM"
+
"n1dIiwic2NwIjoidXNlciIsImlkcCI6IjAiLCJlbWwiOiJhZG1pbmlzdHJhdG9yQHNmLTI5LTEwLTI5LnNvZi1tYnUuZW5nLnZ"
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlNEx6bURnIiwiZGlkIjoiIiwid2lkIjoiIiwiZXhwIjoxNTAyMjMwOTEwLCJpYXQiOj"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybl90eXBlIjoiVVNFUiJ9."
+ "G9gEQPtmEH5jYab-
IlTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLFkCLwa50OQCQ-qgKutZl21R"
+ "m740qBKLhmBB0NQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);


// By default each tenant works in its tenant scope. However, solution users can overrde the
tenant in which they performa a given operation:
// Here, users of SDK should provide a value that is meanningful to their context.
String overrideWithTenant = "nonSolutionUserTenant";


VcoSession session = sessionFactory.newSession(auth, overrideWithTenant);

//Use session below
```

访问 Orchestrator REST API 的参考文档

Orchestrator REST API 的参考文档包含了有关 API 的 RESTful Web 服务、适用于 API 的数据模型、API 的有效响应代码、代码实例等信息。

Orchestrator REST API 参考文档随同 Orchestrator 一起安装。参考文档位于：https://*orchestrator_host*:*port*/vco/api/docs/。

如需获取 Swagger 官方规范，请访问 https://swagger.io/specification/。

使用 Java REST SDK

您可以使用 Java SDK 库在 Java 应用程序中的 Orchestrator REST API 上调用操作，并直接使用对象。

Orchestrator REST SDK 的每项 RESTful Web 服务都具有封装 Java 类以及使用该服务时可运行的操作对应的方法。

Java REST SDK 随 Orchestrator 一同安装。Java REST SDK 项目可在以下位置找到。

注 仅当部署 Orchestrator Appliance 后，您才能访问项目。

- https://*orchestrator_host*:*port*/vco-repo/com/vmware/o11n/o11n-rest-client/
- https://*orchestrator_host*:*port*/vco-repo/com/vmware/o11n/o11n-rest-client-examples/

- https://*orchestrator_host*:*port*/vco-repo/com/vmware/o11n/o11n-rest-client-services/
- https://*orchestrator_host*:*port*/vco-repo/com/vmware/o11n/o11n-rest-client-stubs/

示例：运行工作流并等待其完成

以下示例代码运行了工作流并等待其完成。

```
//start a new session to Orchestrator by using specified credentials
VcoSession session = DefaultVcoSessionFactory.newLdapSession(new URI("https://orchestrator-server:
8281/vco/api/"), "username", "password");

//create the services
WorkflowService workflowService = new WorkflowService(session);
ExecutionService executionService = new ExecutionService(session);

//find a workflow by ID
Workflow workflow = workflowService.getWorkflow("1231235");

//create an ExecutionContext from the user's input
ExecutionContext context = new ExecutionContextBuilder().addParam("name", "Jerry").addParam("age",
18).build();

//run the workflow
WorkflowExecution execution = executionService.execute(workflow, context);

//wait for the workflow to reach the user interaction state, checking every 500 milliseconds
execution = executionService.awaitState(execution, 500, 10, WorkflowExecutionState.CANCELED,
WorkflowExecutionState.FAILED, WorkflowExecutionState.COMPLETED);

String nameParamValue = new ParameterExtractor().fromTheOutputOf(execution).extractString("name");
System.out.println("workflow was executed with 'name' input set to" + nameParamValue);
```

工作流操作

您可以使用 Orchestrator REST API 提供的 Web 服务执行各种工作流操作。

查找工作流并检索其定义

为了能够使用工作流执行任意类型的操作，您必须在 Orchestrator 清单中找到该工作流并检索其定义。定义列出了工作流输入和输出参数，并包含可用工作流运行的链接、工作流展示以及其他对象。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 查找工作流的清单条目。

- 如果您拥有工作流的全名或名称中的关键字，请应用筛选器以在工作流服务的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows?conditions=name={workflowFullName}

GET https://{orchestrator_host}:{port}/vco/api/workflows?conditions=name~{keyWord}
```

- 在作为工作流清单条目入口点的 URL 发起 GET 请求，通过目录或清单服务搜索工作流：

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/Workflow/

GET https://{orchestrator_host}:{port}/vco/api/inventory/System/Workflows/
```

2 在工作流清单条目的 URL 发起 GET 请求以检索工作流清单条目：

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/Workflow/{workflowID}/
```

3 在定义的 URL 发起 GET 请求以检索工作流定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

示例：搜索“发送问候”工作流

您可以查找“发送问候”工作流并检索其定义：

1 若要查找“发送问候”工作流，请应用筛选器以在工作流服务的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/workflows?conditions=name~Hello
```

您会收到一组工作流列表，工作流名称中会包含问候一词：

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<inventory-items xmlns="http://www.vmware.com/vco" total="2">
   <link rel="down"
      href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080E6808080013086668236014a0614d1
6e1/">
       <attributes>
          <attribute name="id"
value="CF80808080808080808080808080808080E6808080013086668236014a0614d16e1"/>
          <attribute name="canExecute" value="true" />
          <attribute name="description" value="" />
          <attribute name="name" value="Interactive Hello World" />
          <attribute name="type" value="Workflow"/>
          <attribute name="canEdit" value="true"/>
      </attributes>
    </link>
    <link rel="down"
      href="https://localhost:
```

```
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080DA8080800130866682360l4a0614d1
6e1/">
      <attributes>
          <attribute name="id"
value="CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1"/>
          <attribute name="canExecute" value="true" />
          <attribute name="description" value="" />
          <attribute name="name" value="Send Hello" />
          <attribute name="type" value="Workflow"/>
          <attribute name="canEdit" value="true"/>
      </attributes>
   </link>
</inventory-items>
```

2 在“发送问候”工作流的清单条目的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080DA8080800130866682360l4a0614d1
6e1/
```

您会在响应正文中收到“发送问候”工作流的清单条目：

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<inventory-item xmlns="http://www.vmware.com/vco"
   href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF80808080808080808080808080808080DA8080800130866682360l4a0614d1
6e1/">
   <relations>
      <link rel="down"
       href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/" />
      </relations>
   <attributes>
      <attribute name="id"
value="CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1"/>
      <attribute name="canExecute" value="true" />
      <attribute name="description" value="" />
      <attribute name="name" value="Send Hello" />
      <attribute name="type" value="Workflow"/>
      <attribute name="canEdit" value="true"/>
   </attributes>
</inventory-item>
```

3 若要检索工作流定义，请在其 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/
```

您会在响应正文中收到“发送问候”工作流的定义：

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
   href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/">
   <relations>
      <link rel="up"
         href="https://localhost:8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
        <link rel="add"
            href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/" />
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/" />
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/presentati
on/" />
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/tasks/" />
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/icon/" />
   </relations>
   <input-parameters>
      <parameter name="name" type="string" />
   </input-parameters>
   <output-parameters>
      <parameter name="message" type="string" />
   </output-parameters>
   <name>Send Hello</name>
      <description></description>
</workflow>
```

运行工作流

您可以为特定工作流创建新执行对象以通过 Orchestrator REST API 运行工作流。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在定义的 URL 发起 GET 请求以检索要运行的工作流的定义：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

您会在请求的响应正文中收到工作流定义。在工作流定义中，您可以查看工作流的输入参数、工作流描述和其他信息。

2 在持有工作流执行对象的 URL 发起 POST 请求：

```
POST https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/
```

3 在请求正文中，为 execution-context 元素中的工作流输入参数提供值。

如果在请求正文中提供空的 execution-context，则工作流会使用其输入参数的默认值（如有）运行。

如果 POST 请求成功，您会收到状态代码 202，并且响应正文为空，同时还会收到在 Location 标头中新创建执行对象的链接。

示例：运行“发送问候”工作流

您可以检索“发送问候”工作流的定义并进行运行。

1 在持有“发送问候”工作流定义的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080808080DA8080800130866682360l4a0614d16e1/
```

您会在请求的响应正文中收到工作流定义：

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
   href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080808080DA8080800130866682360l4a0614d16e1/">
   <relations>
      <link rel="up"
        href="https://localhost:8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
      <link rel="add"
        href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/" />
      <link rel="down"
        href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/" />
      <link rel="down"
        href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080808080DA8080800130866682360l4a0614d16e1/presentati
on/" />
      <link rel="down"
           href="https://localhost:
```

```
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/tasks/" />
      <link rel="down"
        href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/icon/" />
   </relations>
   <input-parameters>
      <parameter name="name" type="string" />
   </input-parameters>
   <output-parameters>
      <parameter name="message" type="string" />
   </output-parameters>
   <name>Send Hello</name>
      <description></description>
</workflow>
```

2 在持有工作流执行对象的 URL 发起 POST 请求：

```
POST https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/
```

在请求正文中，为 execution-context 元素中的输入参数传递值：

```
<execution-context  xmlns="http://www.vmware.com/vco">
   <parameters>
      <parameter name="name" type="string">
         <string>John Smith</string>
       </parameter>
    </parameters>
</execution-context>
```

根据工作流展示验证工作流输入参数后再运行工作流

工作流的展示可以定义传递到工作流输入参数的值的限制，例如预定义值列表或特定值范围。若要确保工作流成功运行，您必须根据工作流展示的定义对传递到工作流输入参数的值进行验证。

在自定义应用程序中集成工作流时，您可能需要创建一个向导，在其中输入工作流运行时的输入参数值。通过使用工作流展示服务，您可以实例化工作流的展示并对应向导的不同屏幕分批传递输入参数值。您可以根据工作流展示中定义的限制对传递到输入参数的值进行验证。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在包含工作流定义的 URL 发起 GET 请求来检索想要运行的工作流的定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

您会在请求的响应正文中收到工作流定义。在工作流定义中，您可以查看工作流的输入参数、工作流描述和其他信息。

2 在工作流展示的 URL 发起 GET 请求以检索其定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/presentation/
```

3 在请求的响应正文中，检查工作流展示是否定义了可传递到输入参数的值的任何限制。

例如，某个输入参数可能有一个预定义值列表来选择相应值。

4 在展示实例的 URL 发起 POST 请求来实例化工作流展示：

```
POST https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/presentation/instances/
```

5 在请求正文中提供 execution-context 元素来实例化展示。

您可以传递空 execution-context 或传递包含仅适用于部分输入参数值的 execution-context。

6 若要分批将值传递到输入参数，根据需要在持有展示实例的 URL 发起多个 POST 或 PUT 请求：

```
PUT https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/presentation/instances/{executionID}/
```

7 查看所发起 POST 或 PUT 请求的响应正文。

如果传递到输入参数的值有效，您会在 execution 标记中找到 valid="true" 属性。如果展示有效，您可以获取响应的 out-parameters 元素中列出的值，并在运行工作流时将其作为值传递到输入参数。

8 如果输入参数的值有效，请在持有工作流执行的 URL 发起 POST 请求来运行工作流：

```
POST https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/
```

9 为 execution-context 元素中的工作流输入参数提供有效值。

示例：验证其输入参数来运行发送问候工作流

您可以根据“发送问候”工作流的展示定义验证其输入参数，从而运行此工作流。

1 在持有“发送问候”工作流定义的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/
```

您会在请求的响应正文中收到工作流定义：

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
   href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/">
   <relations>
      <link rel="up"
          href="https://localhost:8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
      <link rel="add"
          href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/" />
      <link rel="down"
          href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/" />
      <link rel="down"
          href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/presentati
on/" />
      <link rel="down"
             href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/tasks/" />
      <link rel="down"
          href="https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/icon/" />
   </relations>
   <input-parameters>
      <parameter name="name" type="string" />
   </input-parameters>
   <output-parameters>
      <parameter name="message" type="string" />
   </output-parameters>
   <name>Send Hello</name>
      <description></description>
</workflow>
```

2 在持有工作流展示定义的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/presentati
on/
```

3 在持有工作流展示执行实例的 URL 发起 POST 请求：

```
POST https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/presentati
on/instances/
```

提供空的 execution-context 以便仅实例化展示而不提供任何输入参数值：

```
<execution-context xmlns="http://www.vmware.com/vco"/>
```

响应正文包含附加到每个字段的错误消息，表示输入参数的值无效。

```
.......
<fields>
  <field type="string" hidden="false" id="name">
    <display-name>name</display-name>
    <description>name</description>
    <messages>
       <message severity="ERROR" code="VCO-CNS0002">
          <Summary>
                The minimum number of characters allowed for this field is 3.0
          </Summary>
       </message>
    </messages>
    <constraints>
       <number-range max="15.0" min="3.0" />
    </constraints>
.......
```

4 在持有特定展示实例的 URL 发起 POST 请求：

```
POST https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/presentati
on/instances/888080808080808080808080808080803F8080800132145338690643f66a027ec/
```

在请求正文中，提供输入参数的值：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 <execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

在请求的响应正文中，您可以检查输入参数的值是否有效：

```
<execution started-by="vcoadmin" .... valid="true".....>
```

5 如果展示有效，则在持有工作流执行的 URL 发起 POST 请求来运行工作流：

```
POST https://localhost:
8281/vco/api/workflows/CF80808080808080808080808080808080DA8080800130866682360l4a0614d16e1/executions/
```

在请求正文中，将值传递到工作流的输入参数。使用返回为工作流展示输出参数的相同值，或直接使用对工作流展示发起的最后一个 POST 请求的请求正文。

在工作流运行时与其交互

Orchestrator REST API 能让您在工作流运行期间执行各种操作。您可以获取正在运行的工作流状态、响应等待的用户交互和取消工作流运行。

获取工作流运行对象和查看工作流状态

您可以获取有关工作流运行的信息，例如开始和结束日期、运行状态和输入参数的值。您还可以获得针对工作流运行生成的日志。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在工作流的 URL 发起 GET 请求以检索要查看其状态的工作流的定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

您会在请求的响应正文中收到工作流定义。工作流定义包含工作流执行实例的链接。

2 在工作流可用执行实例的 URL 发起 GET 请求以检索其执行实例：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/
```

相应请求的响应正文列出了工作流的可用执行实例，您可以在其中查看每个工作流运行的开始和结束日期及其状态和发起者。

3 （可选）若要获取有关工作流特定运行的更多详细信息，请在该运行的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/{executionID}/
```

在相应请求的响应正文中，您会收到特定工作流运行的 XML 表现形式。您可以查看为该运行传递的输入参数值、发起运行的用户、开始和结束日期以及运行的状态。

4 （可选）若要检索针对工作流运行生成的日志，请在持有该日志的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/logs/
```

5 （可选）若要检索有关运行状态的更多信息，请在持有工作流状态的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/state/
```

示例：获取“发送问候”工作流的运行和查看特定运行的状态

如果运行了“发送问候”工作流，您可以获取可用执行对象并查看其详细信息。

1 在持有定义的 URL 发起 GET 请求以获取“发送问候”工作流的定义：

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/
```

2 在持有工作流执行对象的 URL 发起 GET 请求以获取可用工作流运行：

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/
```

3 在相应请求的响应正文中，选择工作流运行并发起 GET 请求以进行检索：

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/execution
s/88808080808080808080808080808080803A8080800132145338690643f66a027ec/
```

响应正文包含具有指定 ID 的工作流运行的 XML 表现形式，其中您可以查看该运行的详细信息：

```
.......
<input-parameters>
   <parameter name="name" type="string">
      <string>John Smith</string>
   </parameter>
</input-parameters>
<output-parameters>
   <parameter name="message" type="string">
      <string>Hello, John Smith!</string>
   </parameter>
</output-parameters>
<start-date>2012-01-31T14:28:40.223+03:00</start-date>
<end-date>2012-01-31T14:28:40.410+03:00</end-date>
<started-by>vcoadmin</started-by>
<name>Send Hello</name>
......
```

响应等待中的用户交互

您可以使用 Orchestrator REST API 响应某个工作流运行内等待中的用户交互。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在持有可用用户交互对象的 URL 发起 GET 请求，或仅筛选等待中的用户交互，以检索所有用户交互对象的列表：

URL	描述
https://*orchestrator_host*:*port*/vco/api/catalog/System/UserInteraction	持有 Orchestrator 中的可用用户交互对象
https://*orchestrator_host*:*port*/vco/api/catalog/System/UserInteraction?status=0	仅筛选等待中的用户交互对象。

您会收到可用用户交互对象的列表。等待中的用户交互的属性包含名称 `state` 和值 `waiting`。

2 在持有要响应的等待中用户交互清单条目的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/UserInteraction/{userInteractionID}/
```

清单条目会包含用户交互实例的链接。用户交互实例与特定工作流运行关联。

3 在特定工作流执行的用户交互实例的 URL 发起 POST 请求：

```
POST https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/
```

4 在请求正文中，为 `execution-context` 元素中的用户交互输入参数提供值。

成功响应用户交互后，REST API 会返回 204 状态。

示例：响应交互式“你好世界”工作流的用户交互

您可以运行交互式“你好世界”示例工作流并响应其用户交互。

1 在目录服务的用户交互对象端点处发起 GET 请求，搜索工作流内等待中的用户交互：

```
GET https://localhost:8281/vco/api/catalog/System/UserInteraction?status=0
```

2 找到交互式“你好世界”工作流的用户交互清单对象，并在其 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/8880808080808080808080808080805A8080800132145338690643f66a027ec/
```

3 在当前正在运行的工作流执行的用户交互对象的 URL 处发起 POST 请求：

```
POST https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080E68080800130866682360l4a0614d16e1/executions/88808080808080808080808080808057808080013214533869064 3f66a027ec/interaction/
```

在请求正文中提供输入参数的值：

```
<execution-context xmlns="http://www.vmware.com/vco">
   <parameters>
     <parameter name="name" type="string">
       <string>John Smith</string>
     </parameter>
   </parameters>
</execution-context>
```

验证输入参数后响应用户交互

用户交互的展示可能定义了可传递到工作流输入参数的值的限制。响应用户交互时，您可以根据用户交互的展示中定义的限制对传递到输入参数的值进行验证。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在持有可用用户交互对象的 URL 发起 GET 请求，或仅筛选等待中的用户交互，以检索所有用户交互对象的列表：

URL	描述
https://*orchestrator_host*:*port*/vco/api/catalog/System/UserInteraction	持有 Orchestrator 中的可用用户交互对象。
https://*orchestrator_host*:*port*/vco/api/catalog/System/UserInteraction?status=0	仅筛选等待中的用户交互对象。

您会收到可用用户交互对象的列表。等待中的用户交互的属性包含名称 `state` 和值 `waiting`。

2 在持有想要响应的等待中用户交互清单条目的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/System/UserInteraction/{userInteractionID}/
```

响应正文包含用户交互实例的链接。用户交互实例与特定工作流运行关联。

3 在用户交互实例的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/
```

在响应正文中，您会发现用户交互展示的下行链接。

4 在用户交互展示的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/presentation/
```

您会在响应正文中收到用户交互展示的定义。

5 在展示定义中，查看您可以传递到输入参数的值的限制。

6 在展示实例驻留的 URL 发起 POST 请求来运行用户交互展示：

```
POST https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/presentation/instances/
```

7 在请求正文中，为 `execution-context` 元素中的输入参数提供值。

在响应正文中，您会收到用户交互展示的实例。如果传递到输入参数的值有效，您会在 `execution` 元素中找到 `valid="true"` 属性。在 `output-parameters` 元素中，您会找到可用于响应用户交互的输入参数的有效值。

8 在用户交互实例驻留的 URL 发起 POST 请求来响应用户交互：

```
POST https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/interaction/
```

9 在请求正文中，传递包含输入参数值的 `execution-context` 上下文。

您可以将此同一请求正文用于在用户交互展示的 URL 发起的 POST 请求中。

如果上一请求成功，您会收到状态代码 204 和空响应正文。

示例：验证输入参数来响应交互式“你好世界”工作流的用户交互

您可以根据用户交互展示中定义的限制对输入参数的值进行验证，从而对交互式“你好世界”工作流的用户交互作出响应。

1 在目录服务的用户交互对象端点处发起 GET 请求，搜索工作流内等待中的用户交互：

```
GET https://localhost:8281/vco/api/catalog/System/UserInteraction?status=0
```

2 找到交互式“你好世界”工作流的用户交互清单对象，并在其 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080808080805A8080800132145338690643f66a027ec/
```

3 在用户交互实例的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080808080805A8080800132145338690643f66a027ec/interaction/
```

4 在用户交互展示的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080808080805A8080800132145338690643f66a027ec/interaction/presentation/
```

展示将输入参数定义为必需参数，并包含您可以传递的字符串的长度限制。

5 在持有用户交互展示的实例的 URL 发起 POST 请求：

```
POST https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080808080805A8080800132145338690643f66a027ec/interaction/presentation/instances/
```

在请求正文中提供输入参数的值：

```
<execution-context xmlns="http://www.vmware.com/vco">
   <parameters>
     <parameter name="name" type="string">
       <string>John Smith</string>
     </parameter>
   </parameters>
</execution-context>
```

响应正文的 execution 元素包含 valid="true" 属性，表示根据用户交互展示中的限制，输入参数值有效。有效值列出在 output-parameters 元素中：

```
.............
<output-parameters>
  <parameter name="name" type="string">
     <string>John Smith</string>
   </parameter>
</output-parameters>
.............
```

6 通过传递与步骤 5 中 POST 请求相同的请求正文以在用户交互实例的 URL 发起 POST 请求。

```
POST https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080808080805A8080800132145338690643f66a027ec/interaction/
```

取消工作流运行

您可以使用 Orchestrator REST API 取消运行工作流。

步骤

1 在工作流定义的 URL 发起 GET 请求以检索工作流定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

工作流定义包含工作流可用执行对象的链接。

2 向持有工作流可用执行对象的 URL 发起 GET 请求以获取可用工作流运行：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/executions/
```

3 在可用工作流执行列表中，选择要取消的工作流并在其 URL 发起 DELETE 请求：

```
DELETE https://{orchestrator_host}:
{port}/vco/api/workflows/{workflowID}/executions/{executionID}/state
```

检索工作流的交互

您可以使用 Orchestrator REST API 检索工作流所有用户交互的列表。

步骤

1 在工作流定义的 URL 发起 GET 请求以检索工作流定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

2 在工作流交互的 URL 发起 GET 请求以获取工作流交互列表：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/interactions/
```

如果 GET 请求成功，您会收到状态代码 200 以及工作流所有可用用户交互的列表。

访问工作流架构

您可以使用 Orchestrator REST API 访问某个工作流的架构映像。

步骤

1 在工作流定义的 URL 发起 GET 请求以检索工作流定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

2 向工作流架构的 URL 发起 GET 请求以获取工作流架构映像：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/schema/
```

如果 GET 请求成功，您会收到状态代码 200，以及表示工作流架构的映像的二进制数据。响应内容类型设置为正确的介质类型，例如 `Content-Type:image/png`。

使用任务

使用 Orchestrator REST API 的任务服务，您可以执行与管理 Orchestrator 中的任务相关的任意操作。您可以创建用于调度工作流的任务、修改已存在任务的属性，以及删除任务等。

Orchestrator 支持的已调度任务最大数量为 50。

创建任务

您可以使用 Orchestrator REST API 为调度工作流创建任务。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在工作流的 URL 发起 GET 请求以检索要为其创建任务的工作流的定义：

```
GET https://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

在工作流定义中，您可以查看工作流的名称和 ID，以及其输入参数。

2 若要创建工作流的新任务，请在任务服务的 URL 发起 POST 请求：

```
POST https://{orchestrator_host}:{port}/vco/api/tasks/
```

3 在请求正文中，为 task 元素中的新任务提供参数。

如果请求成功，API 会返回响应的状态代码 202 和空的响应正文。

示例：创建“发送问候”工作流的任务

您可以创建用于调度“发送问候”工作流的任务，从而自特定日期开始，在每小时的第十五分钟进行运行。

1 在“发送问候”工作流的 URL 发起 GET 请求以检索其定义：

```
GET https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080808080DA8080800130866682360l4a0614d16e1/
```

2 在请求正文中提供新任务的参数，从而在任务服务的 URL 发起 POST 请求：

```
POST https://localhost:8281/vco/api/tasks/
```

```
<task xmlns="http://www.vmware.com/vco">
    <name>Send Hello Task</name>
    <recurrence-cycle>every-hours</recurrence-cycle>
    <recurrence-start-date>2012-01-31T11:00:00+00:00</recurrence-start-date>
    <recurrence-end-date>2012-02-05T11:00:00+00:00</recurrence-end-date>
```

```
    <recurrence-pattern>15:15</recurrence-pattern>
    <input-parameters>
        <parameter name="name" type="string">
            <string>John Smith</string>
        </parameter>
    </input-parameters>
    <workflow href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/">
        <name>Send Hello</name>
    </workflow>
    <start-mode>normal</start-mode>
</task>
```

修改任务

您可以使用 Orchestrator REST API 更改现有任务的属性。

您仅可以将新的调度属性添加到任务或更改已存在属性的值。如果想要替换任务的调度属性，必须删除任务后重建。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在想要修改的任务的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/tasks/{task ID}/
```

2 在请求的响应正文中查看任务属性。

3 若要修改任务，请在请求正文中提供 `task-data` 元素的任务新属性，从而在任务的 URL 发起 POST 请求。

如果 POST 请求成功，API 会返回状态代码 200，并在响应正文中返回更新后的任务。

示例：更新“发送问候”示例任务

您可以更新任务的开始和结束日期。您可以修改 创建任务 中采用的示例任务。您必须在请求正文中提供新的开始和结束日期，从而在任务的 URL 发起 POST 请求：

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<task-data xmlns="http://www.vmware.com/vco">
   <recurrence-start-date>2012-02-01T14:00:00+02:00</recurrence-start-date>
   <recurrence-end-date>2012-02-05T14:00:00+02:00</recurrence-end-date>
</task-data>
```

查看任务的状态

您可以查看当前可用任务的状态或某个任务的所有执行实例的状态。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

- 若要查看所有当前可用任务的状态，请在任务服务的 URL 发起 GET 请求：

  ```
  GET https://{orchestrator_host}:{port}/vco/api/tasks/
  ```

  响应正文包含 Orchestrator 中当前可用任务的定义。每个任务的状态会显示在 `attribute` 元素中，其名称为 `state`。并且元素的值可以分别为 `finished`、`pending`、`running` 等。

- 若要查看某个任务的所有执行的状态，请在任务执行驻留的 URL 发起 GET 请求：

  ```
  GET https://{orchestrator_host}:{port}/vco/api/tasks/{taskID}/executions/
  ```

  您会在响应正文中收到任务可用执行的列表。每个执行的状态会显示在任务执行对象的 `state` 元素中。

在 Orchestrator 清单中查找对象

您可以使用目录或清单服务在 Orchestrator 清单中查找任意对象。您可以在从中发起 HTTP 请求的 URL 的结尾处应用筛选器参数，仅访问特定对象子集。

您可以使用目录服务在 Orchestrator 清单中查找特定类型的对象，或按其类型和 ID 检索特定对象。例如，您可以检索所有 `workflow` 或 `action` 类型的对象，或检索特定工作流或操作。

清单服务允许您按父项-子项关系浏览 Orchestrator 清单。使用清单服务，您可以访问位于 Orchestrator 清单中指定位置的对象。例如，您可以浏览到数据中心管理的所有工作流在 Orchestrator 清单中的位置（即 `Library/vCenter/Datacenter`），对所有工作流进行检索。

Orchestrator REST API 中的每项服务都支持您发起 HTTP 请求时在 URL 结尾添加的筛选参数。使用筛选参数，您可以缩小在指定 URL 请求的响应正文中所收到结果的范围。

按类型和 ID 查找对象

您可以使用 REST API 的目录服务在 Orchestrator 中按类型和 ID 查找对象。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在目录服务的 URL 发起 GET 请求：

  ```
  GET https://{orchestrator_host}:{port}/vco/api/catalog/
  ```

请求的响应正文包含插件（该插件在 Orchestrator 中公开了清单）目录入口点以及 Orchestrator 中系统对象的下行链路：

- `https://{orchestrator_host}:{port}/vco/api/catalog/{plug-in namespace}/`
- `https://{orchestrator_host}:{port}/vco/api/catalog/System/`

2 若要访问插件公开的对象或 Orchestrator 中的系统对象，请在插件目录入口点的 URL 或 Orchestrator 系统对象驻留的 URL 发起 GET 请求。

请求的响应正文包含公开的对象类型的链接。

3 在想要访问的对象类型的 URL 发起 GET 请求。

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/
```

4 在想要查找的特定对象的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectID}/
```

示例：查找“发送问候”工作流

您可以使用“目录服务”查找示例“发送问候”工作流。

1 在目录服务的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/catalog/
```

2 在 Orchestrator 中所有系统对象所在的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/catalog/System/
```

3 在所有工作流驻留的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/catalog/Workflow/
```

4 在“发送问候”工作流的 URL 发起 GET 请求：

```
GET https://localhost:
8281/vco/api/catalog/Workflow/CF8080808080808080808080808080DA8080800130866682360l4a0614d16e1/
```

按关系查找对象

您可以使用 Orchestrator REST 的清单服务按层次结构浏览 Orchestrator 和插件清单。

前提条件

确认您已导入 Orchestrator 中的示例工作流软件包。该软件包随附在 Orchestrator 示例应用程序 ZIP 文件中，您可从 Orchestrator 文档页面中下载该文件。

步骤

1 在清单服务的 URL 发起 GET 请求：

```
GET https://{orchestrator_host}:{port}/vco/api/inventory/
```

响应正文包含已安装插件的注册清单的下行链路，以及 System 下 Orchestrator 中系统对象的下行链路。

2 在想要访问的清单的下行链路中，发起 GET 请求。

3 在清单中条目的上行和下行链路发起 GET 请求，直到到达要查找的对象为止。

示例：查找“发送问候”工作流

您可以浏览 Orchestrator 清单来查找“发送问候”工作流。

1 在清单服务的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/inventory/
```

2 在 Orchestrator 中系统对象驻留的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/inventory/System/
```

3 在 Orchestrator 中所有工作流驻留的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/inventory/System/Workflows/
```

4 在示例工作流类别的 URL 发起 GET 请求：

```
GET https://localhost:8281/vco/api/inventory/System/Workflows/Samples/
```

5 使用“你好世界”工作流类别的下行链路（可在其中找到“发送问候”工作流）。

应用筛选器

Orchestrator REST API 的服务支持额外的 URL 参数，可让您缩小向 API 发出的 HTTP 请求所返回对象的范围。

可通过 REST API 访问的资源的每条 URL 都支持不同查询参数。若要了解哪些查询参数适用于 URL，请参见 vRealize Orchestrator REST API 参考文档。

步骤

- 若要缩小特定 URL 请求的结果范围，请在 URL 结尾应用筛选器：

  URL? *filter_1*& *filter_2*&*filter_3*&....&*filter_N*。每个筛选器都包含相关 URL 有效的查询参数。有关每条 URL 的有效查询参数的信息，请参见 Orchestrator REST API 参考文档。

示例：筛选工作流

如果要查找在名称中包含特定字词的工作流（例如数据存储），可以在向目录服务发起的请求中应用以下筛选器：

```
GET https://localhost:8281/vco/api/catalog/System/Workflow?conditions=name~datastore
```

若要将返回的工作流数量限制为特定数量（例如 5），请在请求中应用额外筛选器：

```
GET https://localhost:8281/vco/api/catalog/System/Workflow?conditions=name~datastore&maxResult=5
```

导入和导出 Orchestrator 对象

您可以使用 Orchestrator REST API 提供的 Web 服务导入和导出工作流、操作、软件包、资源和配置元素。

导入工作流

您可以使用 Orchestrator REST API 导入某个工作流。

根据 REST 客户端应用程序库的具体情况，您可以使用定义了工作流属性的自定义代码。

注　无法使用 REST API 导入在基于 HTML5 的 vRealize Orchestrator Client 中创建的工作流的输入表单展示。要导入工作流展示，请使用 vRealize Orchestrator Client 的软件包功能。

前提条件

工作流二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1　在 REST 客户端应用程序中，添加请求标头为想要导入的工作流定义属性。

2　在工作流对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/workflows/
```

如果 POST 请求成功，您会收到状态代码 202。

导出工作流

您可以使用 Orchestrator REST API 导出工作流并将其下载为文件。

注　无法使用 REST API 导出在基于 HTML5 的 vRealize Orchestrator Client 中创建的工作流的输入表单展示。要导出工作流展示，请使用 vRealize Orchestrator Client 的软件包功能。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**accept**
- 值：**application/zip**

2 在想要导出的工作流的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

如果 GET 请求成功，您会收到状态代码 200。工作流二进制内容会以附件形式呈现，默认文件名为 *workflow_name*.workflow。您可以使用 REST 客户端应用程序保存文件。

导入操作

您可以使用 Orchestrator REST API 导入某个操作。

根据 REST 客户端应用程序库的具体情况，您可以使用定义了操作属性的自定义代码。

前提条件

操作二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要导入的操作定义属性。

2 在操作对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/actions/
```

如果 POST 请求成功，您会收到状态代码 202。

导出操作

您可以使用 Orchestrator REST API 导出操作并将其下载为文件。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**accept**
- 值：**application/zip**

2 在想要导出的操作的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/
```

如果 GET 请求成功，您会收到状态代码 200。操作二进制内容会以附件形式呈现，默认文件名为 *action_name*.action。您可以使用 REST 客户端应用程序保存文件。

导入软件包

您可以使用 Orchestrator REST API 导入某个软件包。

根据 REST 客户端应用程序库的具体情况，您可以使用定义了软件包属性的自定义代码。

默认情况下，如果导入了重名的 Orchestrator 软件包，不会覆盖现有软件包。您可以在请求中使用参数来指定是否覆盖现有软件包。

默认情况下，Orchestrator 软件包导入时会包含配置元素的属性值。您可以在请求中使用参数来导入不含属性值的软件包。

默认情况下会导入 Orchestrator 软件包中包含的标记，但如果 Orchestrator 服务器上已存在相同标记，则会保留现有标记的值。您可以在请求中使用参数来指定是否保留现有标记值。

前提条件

软件包二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要导入的软件包定义属性。

2 在软件包对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/packages/
```

3 （可选）若要导入软件包并覆盖同名的现有软件包，请在 POST 请求中使用 overwrite 参数：

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?overwrite=true
```

4 （可选）若要在导入软件包时不导入软件包配置元素的属性值，请在 POST 请求中使用 importConfigurationAttributeValues 参数：

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?importConfigurationAttributeValues=false
```

5 （可选）若要在导入软件包时不导入其中的标记，请在 POST 请求中使用 tagImportMode 参数：

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?tagImportMode=DoNotImport
```

6 （可选）若要在导入软件包时一并导入其中的标记并覆盖现有标记值，请在 POST 请求中使用 tagImportMode 参数：

```
POST http://{orchestrator_host}:{port}/vco/api/packages/?
tagImportMode=ImportAndOverwriteExistingValue
```

如果 POST 请求成功，您会收到状态代码 202。

导出软件包

您可以使用 Orchestrator REST API 导出软件包并将其下载为文件。

默认情况下，Orchestrator 软件包导出时会包含配置元素的属性值和全局标记。您可以在请求中使用参数以导出不含属性值或全局标记的软件包。您还可以为下载的软件包文件指定自定义名称。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**accept**
- 值：**application/zip**

2 在想要导出的软件包的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/
```

3 （可选）若要为导出的软件包设置自定义名称，请在 GET 请求中使用 packageName 参数：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?packageName={custom_name}
```

4 （可选）若要在导出软件包时不导出软件包配置元素的属性值，请在 GET 请求中使用 exportConfigurationAttributeValues 参数：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?
exportConfigurationAttributeValues=false
```

5 （可选）若要在导出软件包时不导出全局标记，请在 GET 请求中使用 exportGlobalTags 参数：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?exportGlobalTags=false
```

如果 GET 请求成功，您会收到状态代码 200。软件包二进制内容会以附件形式呈现，默认文件名为 *package_name*.package。您可以使用 REST 客户端应用程序保存文件。

导入资源

您可以使用 Orchestrator REST API 导入某个资源。

根据 REST 客户端应用程序库的具体情况，您可以使用定义了资源属性的自定义代码。

前提条件

资源二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要导入的资源定义属性。

2 在资源对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/resources/
```

如果 POST 请求成功，您会收到状态代码 202。

导出资源

您可以使用 Orchestrator REST API 导出某个资源。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**accept**
- 值：**application/octet-stream**

2 在想要导出的资源的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了资源的内容。

导入配置元素

您可以使用 Orchestrator REST API 导入某个配置元素。

根据 REST 客户端应用程序库的具体情况，您可以使用定义了配置元素属性的自定义代码。

前提条件

配置元素二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要导入的配置元素定义属性。

2 在配置元素对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/configurations/
```

如果 POST 请求成功，您会收到状态代码 202。

导出配置元素

您可以使用 Orchestrator REST API 导出某个配置元素。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**accept**
- 值：**application/vcoobject+xml**

2 在想要导出的配置元素的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了配置元素内容。

删除 Orchestrator 对象

您可以使用 Orchestrator REST API 提供的 Web 服务删除工作流、操作、软件包、资源和配置元素。

删除工作流

您可以使用 Orchestrator REST API 删除某个工作流。

步骤

1 提出 GET 请求，并从返回的工作流列表中检索该工作流的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

2 在工作流的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/
```

如果 DELETE 请求成功，您会收到状态代码 200，并且响应正文为空。

删除操作

您可以使用 Orchestrator REST API 删除某个操作。

步骤

1 提出 GET 请求，并从返回的操作列表中检索该操作的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

2 在操作的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/
```

如果 DELETE 请求成功，您会收到状态代码 200，并且响应正文为空。

删除软件包

您可以使用 Orchestrator REST API 删除某个软件包。

删除软件包时，软件包中的元素不会被删除。如果想要删除软件包的内容，必须提供选项参数。

步骤

1 提出 GET 请求，并从返回的软件包列表中检索该软件包的名称：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

2 在软件包 URL 发起 DELETE 请求，并且如果想要从软件包删除元素，请在请求的结尾处提供选项参数：

```
DELETE http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/?option={parameter}
```

参数	描述
deletePackage	仅删除软件包，保留其内容。
deletePackageWithContent	软件包及其所有内容都被删除。如果其他软件包与删除的软件包共享元素，则共享元素会从其他软件包中删除。
deletePackageKeepingShared	软件包和未共享内容会被删除。与其他软件包共享的元素不会被删除。

如果未提供选项参数，则会使用默认 `deletePackage` 参数。

如果 DELETE 请求成功，您会收到状态代码 200，并且响应正文为空。

删除资源

您可以使用 Orchestrator REST API 删除某个资源。

步骤

1 提出 GET 请求，并从返回的资源列表中检索该资源的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

2 在资源的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/
```

如果 DELETE 请求成功，您会收到状态代码 200，并且响应正文为空。

删除配置元素

您可以使用 Orchestrator REST API 删除某个配置元素。

步骤

1 提出 GET 请求，并从返回的配置元素列表中检索该配置元素的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

2 在配置元素的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/
```

如果 DELETE 请求成功，您会收到状态代码 200，并且响应正文为空。

在 Orchestrator 对象上设置权限

您可以使用 REST API 为 Orchestrator 对象设置自定义权限。若要设置权限，您必须在对象权限的 URL 发起 POST 请求，并在请求正文中定义权限。

您也可以使用 Orchestrator REST API 检索有关对象权限的信息或删除现有权限。

REST API 权限

使用 Orchestrator REST API 设置权限时，您必须使用一组字符来定义权限。

您可以在 POST 请求的请求正文的 <rights> 标记中添加一系列字符，从而定义元素的权限。

您通过 Orchestrator REST API 用来设置权限的字符包含具体含义。

表 2-1. Orchestrator REST API 权限字符集

字符	描述
r	授予查看权限。
x	授予执行权限。
i	授予检查权限。
c	授予编辑权限。
a	授予管理权限。

示例：用于设置权限的语法

在 Orchestrator 元素权限的 URL 所发起 POST 请求的请求正文中，您可以使用以下示例语法。

```
<permissions xmlns="http://www.vmware.com/vco">
    <permission>
        <principal>cn=vcousers,ou=vco,dc=appliance</principal>
        <rights>ric</rights>
    </permission>
</permissions>
```

通过在请求正文的 <rights> 标记中设置 ric 权限，您可以允许 vcousers 用户组的成员查看、检查和编辑 Orchestrator 元素。

检索工作流的权限

您可以使用 Orchestrator REST API 检索工作流权限的信息。

步骤

1 提出 GET 请求，并从返回的工作流列表中检索该工作流的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

2 在工作流权限的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/permissions/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了有关工作流权限的信息。

删除工作流的权限

您可以使用 Orchestrator REST API 删除某个工作流的权限。您可以先删除工作流的现有权限，然后再设置新权限。

步骤

1 提出 GET 请求，并从返回的工作流列表中检索该工作流的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

2 在工作流权限的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/permissions/
```

如果 DELETE 请求成功，您会收到状态代码 204，并且响应正文为空。

设置工作流权限

您可以使用 Orchestrator REST API 设置某个工作流的相应权限。

前提条件

查看您可以设置的权限类型以及可在请求正文中使用的语法。请参见 REST API 权限。

步骤

1 提出 GET 请求，并从返回的工作流列表中检索该工作流的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/workflows/
```

2 在 REST 客户端应用程序中，添加请求标头为想要设置权限的工作流定义属性。

3 在请求正文中，指定要设置的权限。

4 在工作流权限的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/workflows/{workflowID}/permissions/
```

如果 POST 请求成功，您会收到状态代码 201。响应正文中提供了有关工作流权限的信息。

检索操作的权限

您可以使用 Orchestrator REST API 检索操作权限的信息。

步骤

1 提出 GET 请求，并从返回的操作列表中检索该操作的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

2 在操作权限的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/permissions/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了有关操作权限的信息。

删除操作的权限

您可以使用 Orchestrator REST API 删除某个操作的权限。您可以先删除操作的现有权限，然后再设置新权限。

步骤

1 提出 GET 请求，并从返回的操作列表中检索该操作的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

2 在操作权限的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/permissions/
```

如果 DELETE 请求成功，您会收到状态代码 204，并且响应正文为空。

设置操作权限

您可以使用 Orchestrator REST API 设置某个操作的相应权限。

前提条件

查看您可以设置的权限类型以及可在请求正文中使用的语法。请参见 REST API 权限。

步骤

1 提出 GET 请求，并从返回的操作列表中检索该操作的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/actions/
```

2 在 REST 客户端应用程序中，添加请求标头为想要设置权限的操作定义属性。

3 在请求正文中，指定要设置的权限。

4 在操作权限的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/actions/{actionID}/permissions/
```

如果 POST 请求成功，您会收到状态代码 201。响应正文中提供了有关操作权限的信息。

检索软件包的权限

您可以使用 Orchestrator REST API 检索软件包权限的信息。

步骤

1 提出 GET 请求，并从返回的软件包列表中检索该软件包的名称：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

2 在软件包权限的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/permissions/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了有关软件包权限的信息。

删除软件包的权限

您可以使用 Orchestrator REST API 删除某个软件包的权限。您可以先删除软件包的现有权限，然后再设置新权限。

步骤

1 提出 GET 请求，并从返回的软件包列表中检索该软件包的名称：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

2 在软件包权限的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/permissions/
```

如果 DELETE 请求成功，您会收到状态代码 204，并且响应正文为空。

设置软件包权限

您可以使用 Orchestrator REST API 设置某个软件包的相应权限。

前提条件

查看您可以设置的权限类型以及可在请求正文中使用的语法。请参见 REST API 权限。

步骤

1 提出 GET 请求，并从返回的软件包列表中检索该软件包的名称：

```
GET http://{orchestrator_host}:{port}/vco/api/packages/
```

2 在 REST 客户端应用程序中，添加请求标头为想要设置权限的软件包定义属性。

3 在请求正文中，指定要设置的权限。

4 在软件包权限的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/packages/{package_name}/permissions/
```

如果 POST 请求成功，您会收到状态代码 201。响应正文中提供了有关软件包权限的信息。

检索资源的权限

您可以使用 Orchestrator REST API 检索资源权限的信息。

步骤

1 提出 GET 请求，并从返回的资源列表中检索该资源的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

2 在资源权限的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/permissions/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了有关资源权限的信息。

删除资源的权限

您可以使用 Orchestrator REST API 删除某个资源的权限。您可以先删除资源的现有权限，然后再设置新权限。

步骤

1 提出 GET 请求，并从返回的资源列表中检索该资源的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

2 在资源权限的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/permissions/
```

如果 DELETE 请求成功，您会收到状态代码 204，并且响应正文为空。

设置资源权限

您可以使用 Orchestrator REST API 设置某个资源的相应权限。

前提条件

查看您可以设置的权限类型以及可在请求正文中使用的语法。请参见 REST API 权限。

步骤

1 提出 GET 请求，并从返回的资源列表中检索该资源的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/resources/
```

2 在 REST 客户端应用程序中，添加请求标头为想要设置权限的资源定义属性。

3 在请求正文中，指定要设置的权限。

4 在资源权限的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/resources/{resourceID}/permissions/
```

如果 POST 请求成功，您会收到状态代码 201。响应正文中提供了有关资源权限的信息。

检索配置元素的权限

您可以使用 Orchestrator REST API 检索配置元素权限的信息。

步骤

1 提出 GET 请求，并从返回的配置元素列表中检索该配置元素的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

2 在配置元素权限的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/{configuration_elementID}/permissions/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了有关配置元素权限的信息。

删除配置元素的权限

您可以使用 Orchestrator REST API 删除某个配置元素的权限。您可以先删除配置元素的现有权限，然后再设置新权限。

步骤

1 提出 GET 请求，并从返回的配置元素列表中检索该配置元素的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

2 在配置元素权限的 URL 发起 DELETE 请求：

```
DELETE http://{orchestrator_host}:
{port}/vco/api/configurations/{configuration_elementID}/permissions/
```

如果 DELETE 请求成功，您会收到状态代码 204，并且响应正文为空。

设置配置元素的权限

您可以使用 Orchestrator REST API 设置某个配置元素的权限。

前提条件

查看您可以设置的权限类型以及可在请求正文中使用的语法。请参见 REST API 权限。

步骤

1 提出 GET 请求，并从返回的配置元素列表中检索该配置元素的 ID：

```
GET http://{orchestrator_host}:{port}/vco/api/configurations/
```

2 在 REST 客户端应用程序中，添加请求标头为想要设置权限的配置元素定义属性。

3 在请求正文中，指定要设置的权限。

4 在配置元素权限的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:
{port}/vco/api/configurations/{configuration_elementID}/permissions/
```

如果 POST 请求成功，您会收到状态代码 201。响应正文中提供了有关配置元素权限的信息。

执行插件操作

您可以使用 Orchestrator REST API 提供的 Web 服务执行各种插件操作。

检索关于插件的信息

您可以使用 Orchestrator REST API 检索所有已安装插件的元数据信息。

步骤

1 在 REST 客户端应用程序中，添加请求标头以定义插件的属性。

2 在插件对象的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/plugins/
```

如果 GET 请求成功，您会收到状态代码 200。

导入插件

您可以使用 Orchestrator REST API 导入某个插件。

根据 REST 客户端应用程序库的具体情况，您可以使用定义了插件属性的自定义代码。

注 如果已经安装了同名插件，则您无法再导入同名插件。

前提条件

插件二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要导入的插件定义属性。

2 在插件对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/plugins/
```

如果 POST 请求成功，您会收到状态代码 200。

导出插件

您可以使用 Orchestrator REST API 导出某个插件。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**accept**
- 值：**application/dar**

2 在想要导出的插件的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/plugins/{plug-in_name}/
```

如果 GET 请求成功，您会收到状态代码 200。响应正文中提供了插件内容。

启用或禁用插件

您可以使用 Orchestrator REST API 启用或禁用插件。

您可以在插件的 URL 中设置 PUT 请求，从而将某个插件的状态从启用改为禁用或从禁用改为启用。您可以检索有关 Orchestrator 插件的信息，从而查看插件的当前状态。请参见检索关于插件的信息。

前提条件

插件二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要启用或禁用的插件定义属性。

2 在想要启用或禁用的插件的 URL 中设置 PUT 请求：

```
PUT http://{orchestrator_host}:{port}/vco/api/plugins/{plug-in_name}/state/
```

如果 PUT 请求成功，您会收到状态代码 200。

执行服务器配置操作

您可以使用 Orchestrator REST API 提供的 Web 服务执行与 Orchestrator 服务器配置相关的各种操作。

检索有关 Orchestrator 服务器配置的信息

您可以使用 Orchestrator REST API 检索有关 Orchestrator 服务器配置的信息。

步骤

1 在 REST 客户端应用程序中，添加请求标头为想要检索信息的服务器定义属性。

2 在插件对象的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/server-configuration/
```

如果 GET 请求成功，您会收到状态代码 200。

导入 Orchestrator 服务器配置

您可以使用 Orchestrator REST API 导入某个已保存的配置。

前提条件

配置二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**content-type**
- 值：**multipart/form-data**

2 在服务器配置的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/server-configuration/
```

如果 POST 请求成功，您会收到状态代码 200。

导出 Orchestrator 服务器配置

您可以使用 Orchestrator REST API 导出服务器配置。

前提条件

配置二进制内容应可用作多部分内容。有关详细信息，请参见 RFC 2387。

步骤

1 在 REST 客户端应用程序中，使用以下值添加请求标头。

- 名称：**content-type**
- 值：**multipart/form-data**

2 使用以下值添加另一请求标头。

- 名称：**accept**
- 值：***/***

3 在服务器配置的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/server-configuration/
```

如果 POST 请求成功，您会收到状态代码 200。

执行标记操作

您可以使用 Orchestrator REST API 提供的 Web 服务在 Orchestrator 中使用标记来执行各种操作，使对象更容易被搜索到。

您可以为对象附加标记，以使其更容易被搜索到。标记是字符串，长度在 3 到 64 个字符之间，并且不得包含空格字符。

您可以添加全局标记和专用标记。全局标记会对所有 Orchestrator 用户显示，而专用标记仅针对创建该标记的用户显示。全局标记只能由具有管理权限的用户创建和移除。

标记对象

您可以使用 Orchestrator REST API 将标记分配给某个对象。

您可以同时创建专用标记和全局标记。您可以在请求正文中指定标记是专用还是全局。

注 要创建全局标记，必须以具有管理员特权的用户身份登录。

您还可以将值分配给创建的标记。值为可用来筛选标记的可选参数。

步骤

1 使用以下语法定义请求正文。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tag-instance xmlns="http://www.vmware.com/vco" global="false">
  <name>tag_name</name>
  <value>tag_value</value>
</tag-instance>
```

注 您可以将**全局**变量设置为 **"true"** 以创建全局标记。

2 在对象的 URL 发起 POST 请求：

```
POST http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tags
```

如果 POST 请求成功，您会收到状态代码 200。

取消标记对象

您可以使用 Orchestrator REST API 移除分配给某个对象的标记。

您可以同时移除专用标记和全局标记。

注 要移除全局标记，必须以具有管理员特权的用户身份登录。

步骤

- 发起 DELETE 请求以移除专用标记或全局标记。
  - 若要移除专用标记，请使用以下语法在对象的 URL 发起 DELETE 请求：

    ```
    DELETE http://{orchestrator_host}:
    {port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tag/{tag_name}
    ```

  - 若要移除全局标记，请使用以下语法在对象的 URL 发起 DELETE 请求：

    ```
    DELETE http://{orchestrator_host}:
    {port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tag/:{tag_name}
    ```

如果 DELETE 请求成功，您会收到状态代码 200。

列出对象标记

您可以使用 Orchestrator REST API 检索分配给某个对象的标记列表。

步骤

- 在对象的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/{objectId}/tags
```

如果 GET 请求成功，您会收到状态代码 200。

按类型列出已标记的对象

您可以使用 Orchestrator REST API 检索标记了特定标记的对象列表，并按对象类型进行筛选。

步骤

- 在对象类型的 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/catalog/{namespace}/{objectType}/?
tags=tag1&tags=:tag2=value
```

如果 GET 请求成功，您会收到状态代码 200。

列出标记所有者

您可以使用 Orchestrator REST API 检索标记所有者的列表。标记所有者即至少创建了一个标记的用户。

步骤

- 在以下 URL 发起 GET 请求：

```
GET http://{orchestrator_host}:{port}/vco/api/tags
```

如果 GET 请求成功，您会收到状态代码 200。检索的列表包含至少创建了一个标记的用户。全局标记列在系统用户名 __GLOBAL__ 下。

按用户列出标记

您可以使用 Orchestrator REST API 检索特定用户所创建标记的列表。

您还可以检索全局标记。全局标记列在系统用户名 __GLOBAL__ 下。

步骤

- 在用户的 URL 发起 GET 请求。
  - 若要检索特定用户所创建标记的列表，请使用以下语法发起 GET 请求：

    ```
    GET http://{orchestrator_host}:{port}/vco/api/tags/{user_name}
    ```

  - 若要检索全局标记的列表，请使用以下语法发起 GET 请求：

    ```
    GET http://{orchestrator_host}:{port}/vco/api/tags/__GLOBAL__
    ```

如果 GET 请求成功，您会收到状态代码 200。

按用户列出按标记名称筛选的标记

您可以使用 Orchestrator REST API 检索由指定用户创建的标记实例的列表，并可按标记名称筛选标记。

您还可以检索全局标记实例。全局标记列在系统用户名 __GLOBAL__ 下。

步骤

- 在用户的 URL 发起 GET 请求。
  - 若要检索特定用户创建的标记实例的筛选列表，请使用以下语法发起 GET 请求：

    ```
    GET http://{orchestrator_host}:{port}/vco/api/tags/{user_name}/{tag_name}
    ```

  - 若要检索全局标记实例的筛选列表，请使用以下语法发起 GET 请求：

    ```
    GET http://{orchestrator_host}:{port}/vco/api/tags/__GLOBAL__/{tag_name}
    ```

如果 GET 请求成功，您会收到状态代码 200。您检索的信息包含了对已标记对象、标记名称、标记值的引用，以及标记实例是全局还是专用的说明。

按用户移除标记

您可以使用 Orchestrator REST API 移除特定用户创建的所有标记。

您还可以移除全局标记。全局标记列在系统用户名 __GLOBAL__ 下。

注 要移除全局标记，必须以具有管理员特权的用户身份登录。

步骤

- 在用户的 URL 发起 DELETE 请求。
  - 若要移除特定用户创建的标记，请使用以下语法发起 DELETE 请求：

    ```
    DELETE http://{orchestrator_host}:{port}/vco/api/tags/{user_name}
    ```

  - 若要移除全局标记，请使用以下语法发起 DELETE 请求：

    ```
    DELETE http://{orchestrator_host}:{port}/vco/api/tags/__GLOBAL__
    ```

如果 DELETE 请求成功，您会收到响应代码 204。