软件开发生产线 CodeArts

制品仓库用户指南

天翼云科技有限公司

目 录

1 软件发布库

1.1 概述

软件发布库是一种通用软件制品库，用来统一管理不同格式的软件制品。除了基本的存储功能，还提供构建部署工具集成、版本控制、访问权限控制等重要功能，是一种企业处理软件开发过程中产生的所有制品包类型的标准化方式。

软件发布库中的主要操作包括：

- 基础操作：包括上传、下载、编辑、搜索、删除软件包，新建、编辑、搜索、删除文件夹。
- 查看/编辑软件包详情：软件包详情包括展示基本信息的概览、构建元信息、构建包归档信息，其中文件夹名称、软件包名称、软件包状态和发布版本可编辑。
- 管理回收站：软件包被删除后会转移至回收站，可以将软件包还原至删除前的文件夹中、或者从回收站中彻底删除。

1.2 进入软件发布库

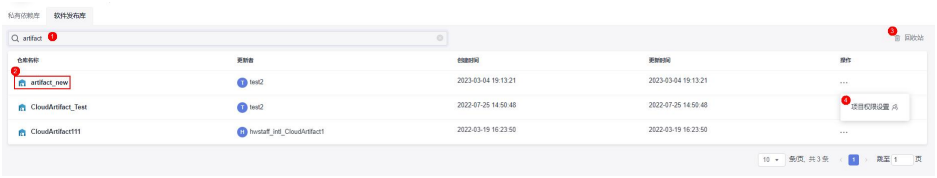
进入软件发布库页面有两种方式：首页入口和项目入口。

首页入口

步骤 1 登录软件开发生产线首页。

步骤 2 在功能菜单区单击“服务 > 制品仓库”。

步骤 3 页面中展示了当前租户下的项目名称列表，根据需要可完成以下操作。



序号	操作	说明
1	搜索仓库	在搜索栏内输入项目名称找到项目下的软件发布库。
2	查看文件夹详情	单击任一文件夹，即可查看文件夹中归档的软件包/文件夹列表。可以完成软件包/文件夹的上传、下载、编辑等管理操作。
3	管理回收站	单击“回收站”，进入回收站页面，可根据需要对软件包/文件夹进行删除或还原操作。
4	项目权限设置	单击…，进入项目权限设置，对成员进行权限编辑，详情请参考权限设置。

说明

从首页“服务”进入软件发布库时，页面展示项目列表，无法进行上传、创建文件夹等操作，请单击项目名进入具体项目下操作。

----结束

项目入口

步骤 1 登录软件开发生产线首页，单击项目卡片进入项目。

步骤 2 单击菜单栏“制品仓库 > 软件发布库”。

步骤 3 页面中将展示当前项目下归档的软件包/文件夹列表。

可根据需要完成后续章节中介绍的操作。

----结束

1.3 软件发布库基础操作

通过项目入口入软件发布库，可以完成上传、下载、编辑、搜索、删除软件包等操作。

新建文件夹

步骤 1 在软件发布库页面，单击“新建文件夹”可以创建新的文件夹，文件夹可以嵌套创建。

单击文件夹名称旁，可以修改文件夹名称。

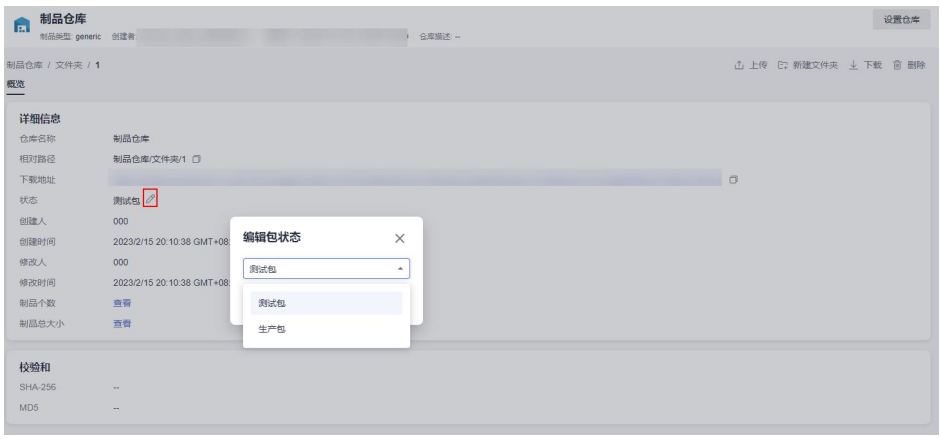
单击文件夹名称旁，可以删除文件夹和文件夹下的软件包，可以选择彻底删除或将删除的文件夹放入回收站。

步骤 2 选择文件夹单击“新建文件夹”，可以创建第二级文件夹。

----结束

设置生产包

步骤 1 进入项目的第一级文件夹后，可以修改第二级文件夹的状态（默认为“测试包”），单击“状态”列中的，在下拉栏中修改对应行文件夹的状态。



如果文件夹的状态为“生产包”，该文件夹状态不可修改、不可编辑（修改文件夹名称、以及修改文件夹下的文件名称、上传、修改版本号、新建文件夹），只能下载或删除。

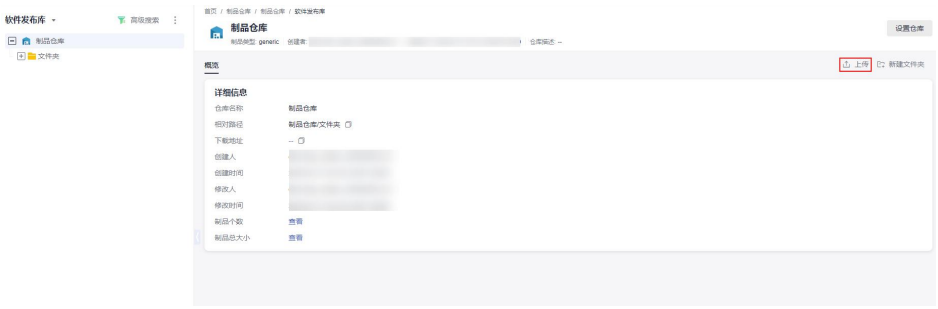
须知

文件夹的状态可由测试包变为生产包，状态转换不可逆，请谨慎操作。

----结束

上传软件包

步骤 1 单击页面右上方“上传”，可以手动上传本地软件包到软件发布库。



选择文件夹后，单击“上传”，可以手动上传本地软件包到对应的文件夹中。

步骤 2 单击软件包名称旁，可以修改软件包名称。

单击软件包名称旁，可以删除软件包，可以选择彻底删除或将删除的文件夹放入回收站。

----结束

高级搜索

步骤 1 通过项目入口进入软件发布库，单击“高级搜索”。

步骤 2 在搜索框中输入关键字（关键字可以为文件夹或文件名称），单击即可搜索出名称中有该关键字的软件包。

单击可以删除文件，可以选择彻底删除或将删除的文件放入回收站。

单击可以将文件下载到本地。

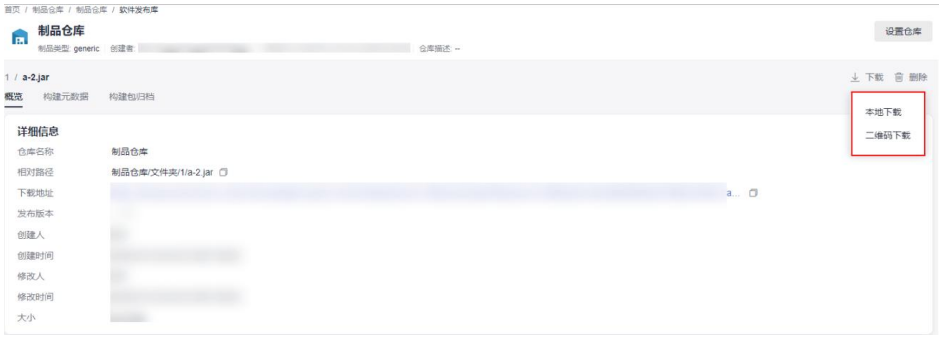
步骤 3 单击文件名即可跳转到该文件的详细信息页面。

----结束

下载软件包

步骤 1 通过项目入口进入软件发布库，选择需要下载的软件包，单击页面右侧“下载”。

步骤 2 在弹框中选择下载方式。



- 本地下载：将软件包下载到本地。
- 二维码下载：通过手机扫描二维码下载文件。

----结束

1.4 查看/编辑软件包详情

在软件发布库页面可以查看并编辑软件包详情，软件包详情包括三方面：概览、构建元数据、构建包归档。

通过项目入口进入软件发布库，单击软件包名称，页面展示所选软件包详情。通过三个页签“概览”、“构建元数据”、“构建包归档”展示软件包详情。

- 概览：展示仓库名称、相对路径、下载地址、发布版本、创建人、创建时间、大小、校验和等信息。

  单击，可以修改软件包的发布版本（由编译构建归档的软件包发布版本默认为构建序号）。
- 构建元数据：展示生成软件包的构建任务、大小、构建序号、构建者、代码库、代码分支。单击“构建任务的名称”可以链接到编译构建任务。
- 构建包归档：展示通过构建任务上传的软件包的归档记录，单击，可以下载软件包。

1.5 管理回收站

在软件发布库删除的软件包/文件夹都会移到回收站，可以对删除后的软件包/文件夹进行管理。

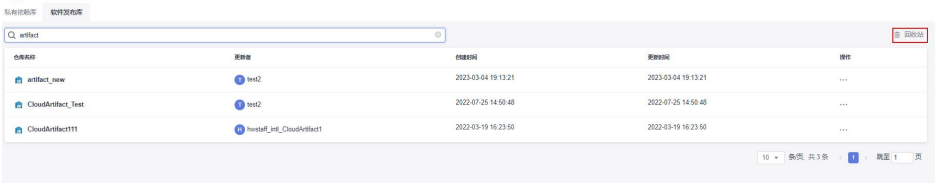
制品仓库服务提供“总回收站”和“项目内回收站”。

总回收站

总回收站可以对所有项目删除的软件包/文件夹进行管理。

步骤 1 登录软件开发生产线首页，在功能菜单区单击“服务 > 制品仓库”。

步骤 2 选择软件发布库页签，单击“回收站”。



步骤 3 页面展示不同项目下已经删除的文件，根据需要对软件包/文件夹进行如下操作。

序号	操作项	说明
1	单个还原	单击操作列，可以还原对应行软件包/文件夹。
2	批量还原	勾选多个软件包/文件夹，单击列表下方的“还原”，可以将所选的软件包/文件夹全部还原。
3	还原所有	单击“还原所有”，可以一键还原回收站所有软件包/文件夹。
4	单个删除	单击操作列，可以删除对应软件包/文件夹。
5	批量删除	勾选多个软件包/文件夹，单击列表下方的“删除”，可以将所选的软件包/文件夹全部删除。

序号	操作项	说明
6	清空回收站	单击“清空回收站”，可以一键删除回收站所有软件包/文件夹。

须知

1. 回收站的所有删除操作都将彻底删除对应软件包/文件夹，无法重新找回，请慎重操作。
2. 总回收站不支持跨项目还原或删除跨项目文件。

----结束

项目内回收站

用于处理处理项目内删除的软件包/文件夹。

步骤 1 通过项目入口进入软件发布库页面，单击页面左下方“回收站”。

步骤 2 页面展示当前项目下已经删除的文件，根据需要对软件包/文件夹进行如下操作。

序号	操作项	说明
1	单个还原	单击操作列，可以还原对应行软件包/文件夹。
2	批量还原	勾选多个软件包/文件夹，单击列表下方的“还原”，可以将所选的软件包/文件夹全部还原。
3	还原所有	单击“还原所有”，可以一键还原回收站所有软件包/文件夹。
4	单个删除	单击操作列，可以删除对应软件包/文件夹。
5	批量删除	勾选多个软件包/文件夹，单击列表下方的“删除”，可以将所选的软件包/文件夹全部删除。
6	清空回收站	单击“清空回收站”，可以一键删除回收站所有软件包/文件夹。

须知

回收站的所有删除操作都将彻底删除对应软件包/文件夹，无法重新找回，请慎重操作。

----结束

1.6 权限设置

在软件发布库中，不同的项目角色对应的操作权限不同。拥有“权限设置”操作权限的成员可以对权限范围进行编辑。

步骤 1 通过项目入口进入软件发布库。

步骤 2 单击页面左上方，在下拉栏中单击“项目权限设置”。

步骤 3 单击需要“配置权限的角色”，根据需要勾选权限，单击保存。

软件发布库提供的默认权限矩阵如下表所示。

操作/角色	项目创建者	项目经理	开发人员	测试经理	测试人员	运维经理	参与者	浏览者
权限配置	√	√	-	-	-	-	-	-

操作/角色	项目创建者	项目经理	开发人员	测试经理	测试人员	运维经理	参与者	浏览者
更改包状态	√	√	-	-	-	√	-	-
上传	√	√	√	√	√	√	-	-
删除/还原（测试包）	√	√	√	√	-	√	-	-
删除/还原（生产包）	√	-	-	-	-	√	-	-
编辑（测试包）	√	√	√	√	-	√	-	-
新建文件夹	√	√	√	√	√	√	-	-
下载	√	√	√	√	√	√	√	√
还原所有	√	√	-	√	-	-	-	-
清空回收站	√	√	-	√	-	-	-	-

说明

- 项目创建者默认拥有全部操作权限，且不显示在页面中，无法修改其权限范围。
- 浏览者只拥有下载权限，且无法修改其权限范围。

----结束

1.7 清理策略

软件发布库提供定时自动清理文件功能。可根据文件保留时长设置的自动将超时的文件从仓库移动至回收站、或者将从回收站内彻底清除。

步骤 1 通过项目入口进入软件发布库。

步骤 2 单击页面右上方“设置仓库”，显示清理策略页面。

步骤 3 根据需要打开“删除文件至回收站”或“从回收站彻底删除”的开关，在下拉列表中选择保存时间。

服务默认保留的时间为：

- 从发布库放入回收站：30 天。
- 从回收站彻底删除：30 天

如果列表中的选项不满足需要，可以自定义时间，单击“自定义”，输入数字，单击“ √ ”保存。

说明

以下为非必填项：

- 忽略“生产包”状态的文件：系统进行文件清理时将保留“生产包”状态的文件，请参考设置生产包。
- 忽略文件路径：系统进行文件清理时将保留匹配用户设置的文件路径的软件包，支持设置多个文件路径(以”/“开头，多路径之间用英文分号隔开)


设置仓库
清理策略
制品仓库清理策略支持自动/手动批量删除满足清理条件的制品，避免长时间未被使用的老旧制品占据过大存储空间。
删除文件至回收站
请设置发布库文件存储时长
天
忽略"生产包"状态的文件
忽略文件路径
从回收站彻底删除
30 天
自定义

----结束

2 私有依赖库

2.1 导读

私有依赖库用于管理私有组件（开发者通俗称之为私服），包括 Maven、npm、Go、PyPI、RPM、Debian、Conan、NuGet 制品仓库。

私有依赖库中的主要功能包括：

- 仓库管理：包括新建仓库、编辑仓库基本信息、管理仓库权限、配置仓库至本地开发环境等。
- 私有组件管理：包括上传、下载、搜索、删除私有组件，管理回收站等。

2.2 进入私有依赖库

进入私有依赖库页面有两种方式：首页入口和项目入口。

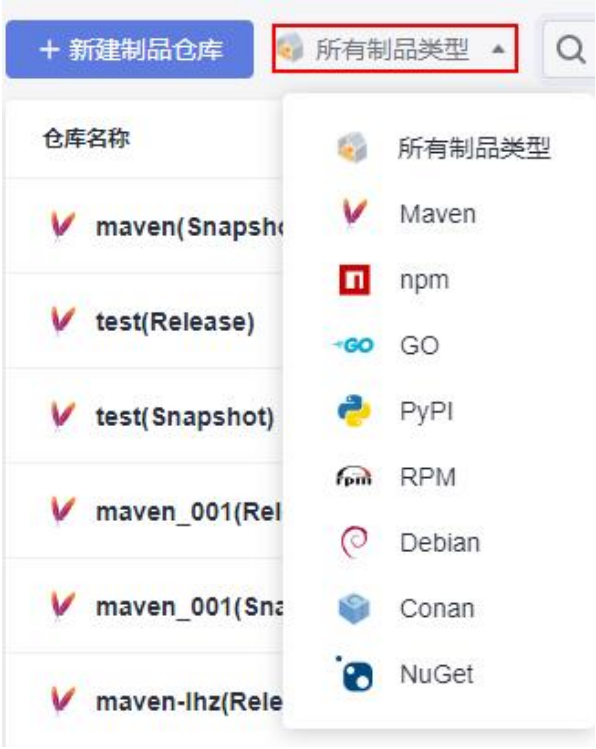
首页入口

步骤 1 登录软件开发生产线首页。

步骤 2 在功能菜单区单击“服务 > 制品仓库”。

步骤 3 选择“私有依赖库”页签，页面展示服务下已创建的所有私有依赖库。

单击上方的筛选下拉栏，可以按制品类型查看仓库。



单击某个仓库名称，跳转到该仓库所在的项目私有依赖库页面。

----结束

项目入口

步骤 1 登录软件开发流水线首页，单击项目卡片进入项目。

步骤 2 单击菜单栏“制品仓库 > 私有依赖库”。

步骤 3 页面中将展示当前项目下归档的不同类型仓库列表。

可根据需要完成后续章节中介绍的操作。

----结束

2.3 新建私有依赖库

首次使用私有依赖库时，需要新建仓库。只有租户管理员才有权限创建私有依赖库。

- 通过首页入口进入私有依赖库，用户可以单击页面左上方“新建制品仓库”。
- 通过项目入口进入私有依赖库，用户可以单击页面左上方 进行创建。

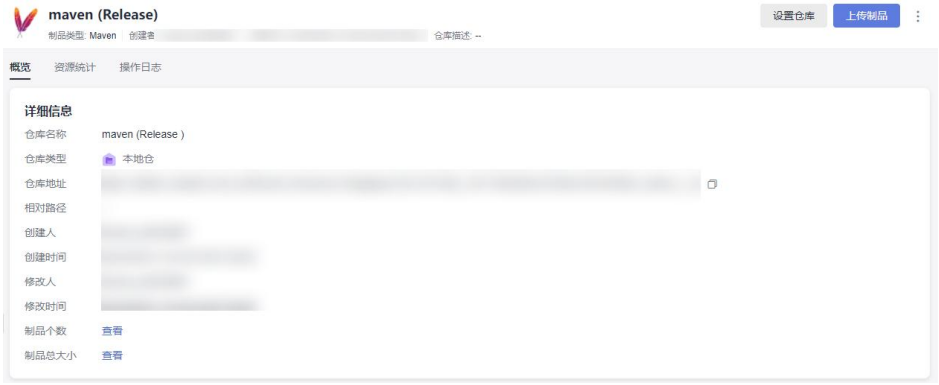
步骤 1 进入“新建私有依赖库”页面。

步骤 2 配置仓库基本信息，单击“确定”。

配置项	是否必填项	说明
仓库名称	是	仅支持英文，数字，下划线(_)，连字符(-)和点(.)，长度 20 字符以内。 **说明** 私有依赖库新建完成后，仓库的名称不支持修改。
制品类型	是	支持 Maven、npm、Go、PyPI、RPM、debian、Conan、NuGet 制品仓库。 选择不同格式的仓库，页面会展示对应的配置，请参照仓库配置说明完成进一步配置。
归属项目	是	为当前创建的仓库选择归属项目。设置完成后，所属项目无法更改。
描述	否	长度 200 字符以内。

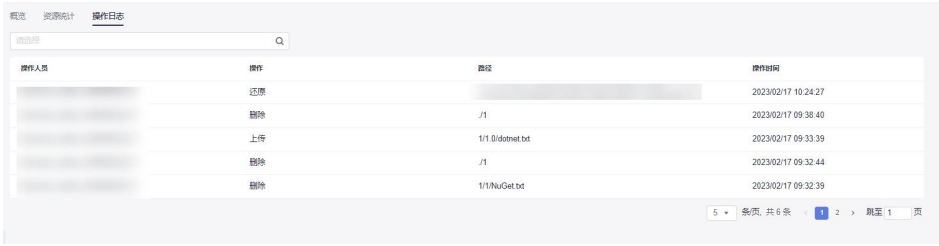
步骤 3 页面左侧列表展示已创建的私有依赖库名称，单击仓库名称显示仓库详情，分为“概览”、“资源统计”、“操作日志”三个页签。

- 概览：显示仓库的名称、仓库类型、仓库地址、相对路径、创建人、创建时间、修改人、修改时间、制品个数、制品总大小信息。



- 资源统计：按照“文件数量趋势”和“存储容量趋势”，对仓库上传制品动态进行统计。

- 操作日志：展示了在仓库中上传、删除、从回收站还原的操作历史。



----结束

仓库配置说明

除了公共配置信息外，每种格式仓库对应了不同的配置项，详情如下。

仓库格式	配置项	是否必填项	说明
Maven	版本策略	是	包括“Release”与“Snapshot”两个选项。 推荐全部选择，这样系统将生成“Release”和“Snapshot”两个仓库；也可以根据自己团队的需求至少选择一个，这样系统将生成一个“Release”或者是“Snapshot”仓库。
npm	添加路径	是	路径即 scope 值。scope 是将相关 npm 包组合在一起的一种方式。scope 与 npm 私有库是多对一的关系，一个 npm 私有库可以包含多个 scope，但是一个 scope 只能指向一个私有库。更多相关说明请参考 scope 官方文档。 构建时，scope 关联的组件可以从对应的私有库下载或者上传到对应私有库。 例如，在页面中添加路径“test”，那么只有以“test”开头的 npm 组件才能上传到此私有依赖库中。
Go	添加路径	否	输入需要添加的路径，单击“+”。 构建时，只允许以该路径开头的 go 文件上传到私有库。
PyPI	添加路径	否	输入需要添加的路径，单击“+”。 构建时，只允许在“setup.py”文件中的“name”值与添

仓库格式	配置项	是否必填项	说明
			加的路径匹配的 PyPI 依赖包上传到私有库。
RPM	添加路径	否	输入需要添加的路径，单击“+”。 构建时，只允许以该路径开头的 RPM 二进制文件上传到私有库。

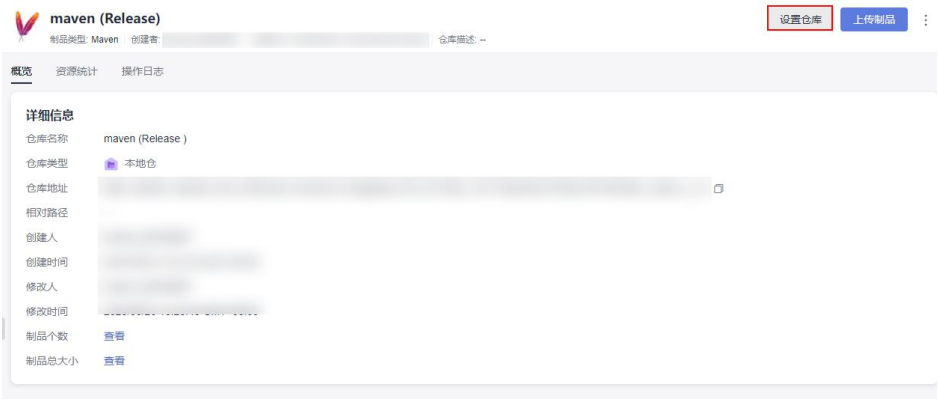
2.4 管理私有依赖库

仓库管理主要包括编辑仓库描述、添加路径、删除仓库、管理用户权限等操作。

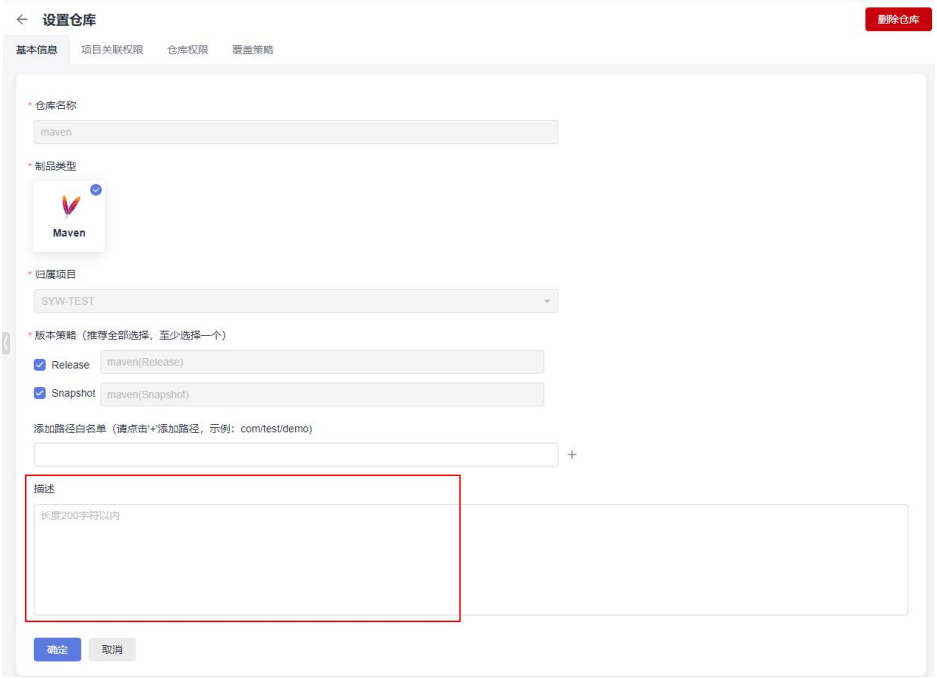
编辑仓库描述与路径

步骤 1 进入私有依赖库，在左侧边栏中单击待编辑信息的仓库名称。

步骤 2 单击页面右侧“设置仓库”，显示仓库的基本信息。



步骤 3 根据需要编辑仓库描述信息，单击“确定”。



说明

在基本信息页面中，仓库的名称、制品类型、归属项目、权限范围不能修改。

在仓库的基本信息页面，首先输入路径，单击 + 可以为 Maven、npm、Go、PyPI、RPM、Conan 仓库添加路径。

单击 可以删除路径。

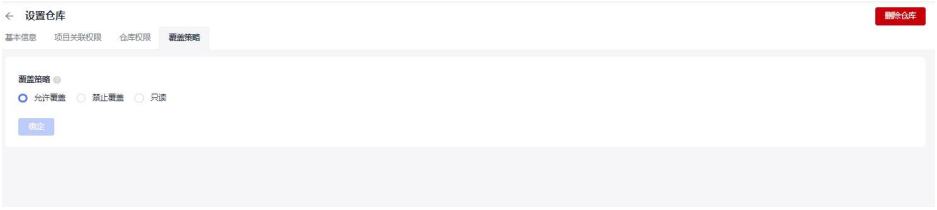
----结束

私有依赖库的覆盖策略

私有依赖库支持“允许覆盖”、“禁止覆盖”、“只读”三种版本策略，可以设置是否允许上传相同路径的制品并将原包覆盖。

步骤 1 进入私有依赖库，在左侧边栏中单击对应的仓库名称。

步骤 2 单击页面右侧“设置仓库”，显示仓库的基本信息，选择“覆盖策略”页签。



- 允许覆盖：允许上传相同路径的制品（默认选择），上传后将会覆盖原包。
- 禁止覆盖：禁止上传相同路径的制品。
- 只读：禁止上传、更新、删除制品。可以下载已上传的制品。

步骤 3 设置完成后，单击“确定”。

----结束

删除仓库

私有依赖库支持删除仓库，被删除的仓库将转移至回收站。

步骤 1 进入私有依赖库，在左侧边栏中单击要删除的仓库名称。

步骤 2 单击页面右侧“设置仓库”，显示仓库的基本信息。

步骤 3 单击页面右侧“删除仓库”。页面左侧仓库列表中将看不到已删除的仓库。

----结束

管理仓库权限

用户在创建仓库后，添加的项目成员和仓库角色对应关系如下：

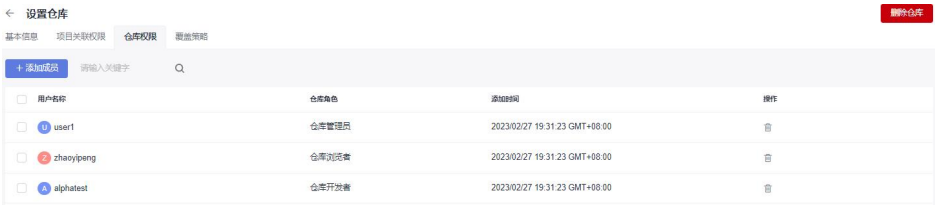
- 项目创建者、项目经理对应仓库管理员。
- 开发人员、测试经理、测试人员、运维经理对应仓库开发者。
- 参与者、浏览者、自定义角色对应仓库浏览者。

为私有依赖库成员添加/删除权限的操作步骤如下：

步骤 1 进入私有依赖库页面，在仓库列表中选择目标仓库。

步骤 2 在页面右侧单击“设置仓库”。

步骤 3 选择“仓库权限”页签，已经添加的仓库成员显示在列表中。



步骤 4 添加成员。

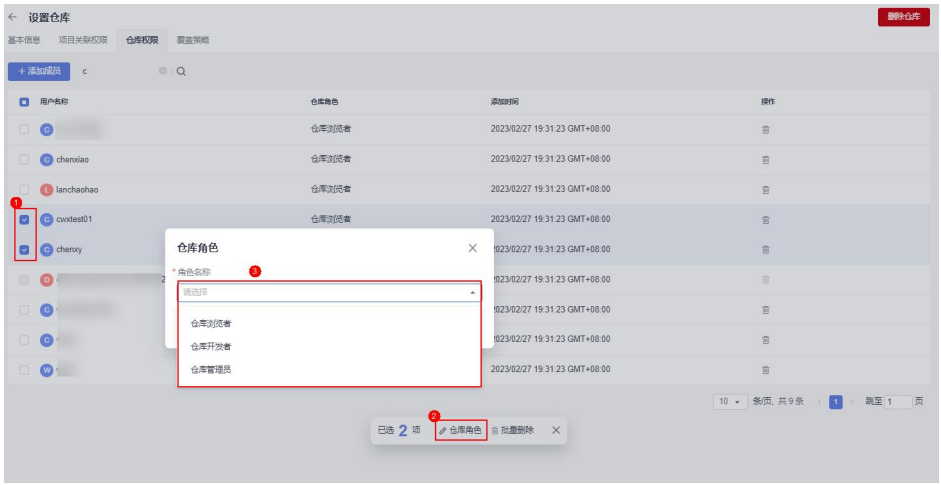
单击页面左上方“添加成员”，在弹框中勾选成员，单击“下一步”。

步骤 5 为成员分配仓库角色。

可以在仓库角色的下拉栏中选择“仓库管理者”、“仓库开发者”、“仓库浏览者”。

步骤 6 单击“确定”，完成添加仓库成员和仓库角色配置，新添加的成员将显示在列表中。

步骤 7 在成员列表中，勾选多个仓库成员，单击下方“仓库角色”可以批量配置仓库角色。



----结束

仓库权限对应的操作权限如下：

操作/角色	租户管理员			非租户管理员		
	仓库管理员	开发者	浏览者	仓库管理员	开发者	浏览者
新建私有依赖库	√	√	√	×	×	×
编辑私有依赖库	√	√	√	×	×	×
管理仓库与项目关联	√	√	√	×	×	×
上传私有组件	√	√	×	√	√	×
下载组件	√	√	√	√	√	√
删除组件	√	√	×	√	√	×
还原组件	√	√	×	√	√	×
彻底删除（组件）	√	√	×	√	√	×
删除仓库	√	×	×	×	×	×
还原仓库	√	√	×	√	√	×
彻底删除（仓库）	√	×	×	×	×	×
清空回收	√	√	√	×	×	×

站						
还原所有	√	√	√	×	×	×
管理用户权限	√	√	√	√	×	×

私有依赖库使用配置

私有依赖库支持与本地开发环境对接，在本地开发时可使用私有依赖库中的私有组件。

步骤 1 进入私有依赖库，在左侧边栏中单击待与本地环境对接的仓库名称。

步骤 2 单击页面右侧⋮，在下拉栏中单击“配置指导”。

步骤 3 在弹框中单击“下载配置文件”，下载配置文件至本地。

步骤 4 参考弹框中的说明，将下载的配置文件复制到相应目录中。

----结束

重置仓库密码

重置仓库密码即指的是私有依赖库配置文件中的密码，重置密码后需要重新下载配置文件，替换旧文件。

步骤 1 进入私有依赖库，单击页面左侧仓库列表上方图标⋮，在下拉列表中选择“重置仓库密码”。

步骤 2 在弹框中单击“是”。页面提示操作成功时表示密码重置成功。

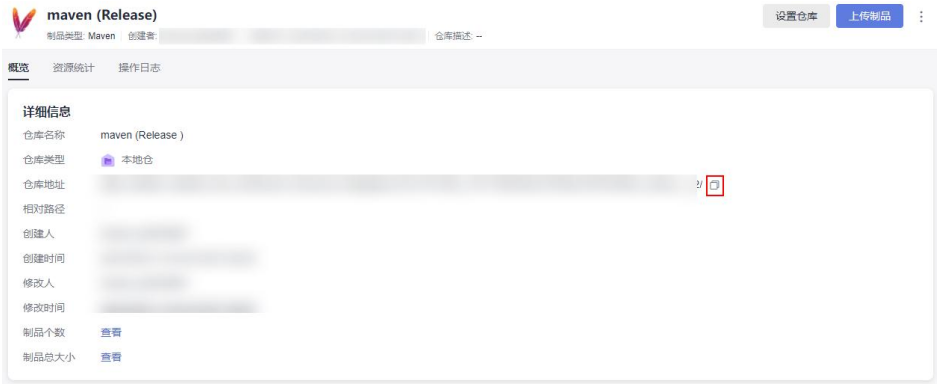
----结束

获取私有依赖库地址

配置本地开发环境对接私有依赖库时，会用到私有依赖库地址，通过以下操作可获取该地址。

步骤 1 进入私有依赖库，在左侧边栏中单击待获取地址的仓库名称。

步骤 2 页面中仓库的详细信息显示私有依赖库地址，单击，可以获取对应地址。

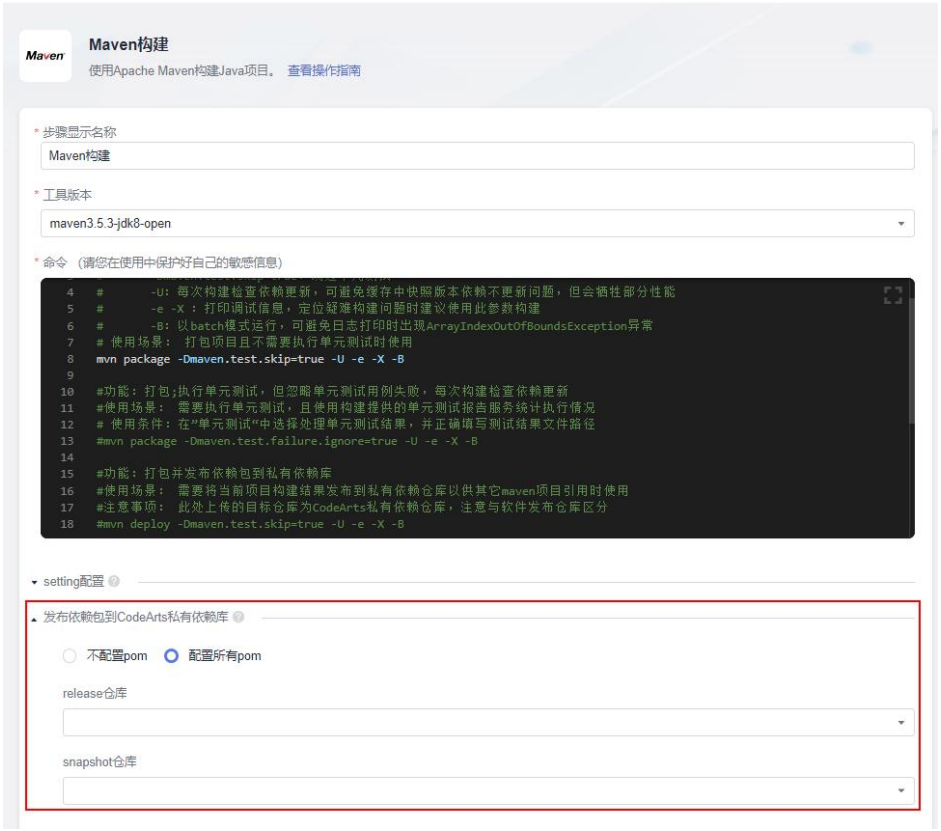


----结束

获取 Maven 仓库与项目关联

通过构建任务上传 Maven 组件到私有依赖库时，需要在构建步骤“Maven 构建”中指定仓库路径。

- 不配置 pom：依赖包不发布到私有依赖库中
- 配置所有 pom：若使用 mvn deploy 命令会将依赖包发布到指定的 release 仓库和 snapshot 仓库中。



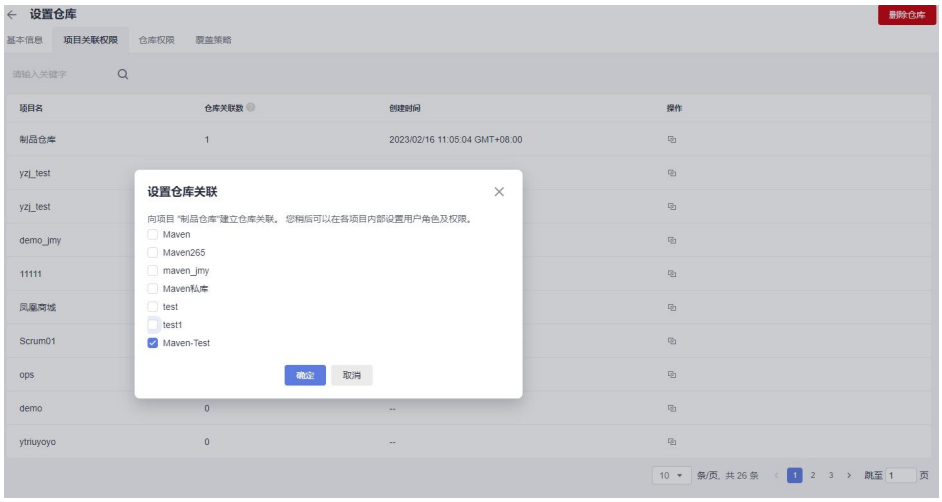
将 Maven 格式仓库与项目关联后，该项目中的构建任务即可完成在构建步骤中选择该仓库。

步骤 1 进入私有依赖库，在左侧仓库列表中单击 Maven 格式仓库。

步骤 2 单击页面右侧“设置仓库”，选择“项目关联权限”

步骤 3 在列表中找到待关联 Maven 仓库的项目，单击对应行中的图标 。

步骤 4 根据需要在弹框中勾选仓库名称，单击“确定”。



当页面提示操作成功时，列表中对应项目的仓库关联数量将显示为与所勾选的仓库数量

----结束

2.5 制品清理策略设置

制品仓库清理策略支持自动/手动批量删除满足清理条件的制品。用户在创新 Maven 类型仓库时，版本策略包括“Release”与“Snapshot”两个选项。

Maven 制品的快照（SNAPSHOT）是一种特殊的版本，指定了某个当前的开发进度的副本，不同于常规的版本，Maven 每次构建都会在远程仓库中检查新的快照，针对快照版本制品提供“快照版本最大保留个数”和“超期快照版本自动清理”功能。

制品仓库的制品清理策略减少了仓库存储空间的浪费，使仓库内制品清晰明了，有效保障了制品在开发、测试、部署、上线等步骤间的有序流转。

操作步骤

步骤 1 通过项目入口进入私有依赖库。

步骤 2 在左侧仓库列表中选择对应的“Snapshot”Maven 仓库，单击页面右上方“设置仓库”。

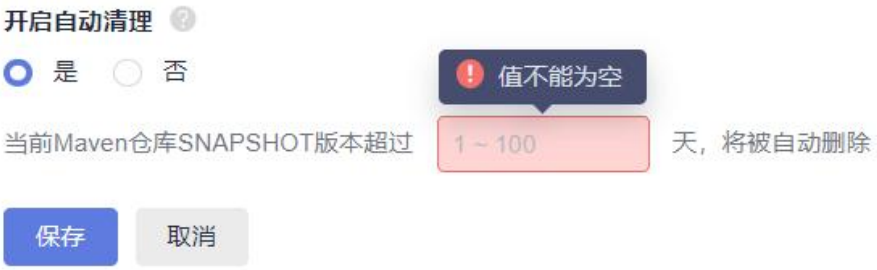
步骤 3 选择“清理策略”页签。

步骤 4 设置“快照版本数限制”，输入范围为 1~1000 个。

当该制品包的版本超过设置值时，最老版本的包将会被最新版本的包覆盖。

步骤 5 开启自动清理（默认为“否”），单击“是”并输入天数，超过指定天数的快照版本将被自动清理。

设置自动清理时间不能小于 1 天或者超过 100 天。



步骤 6 单击“保存”完成清理策略设置。

----结束

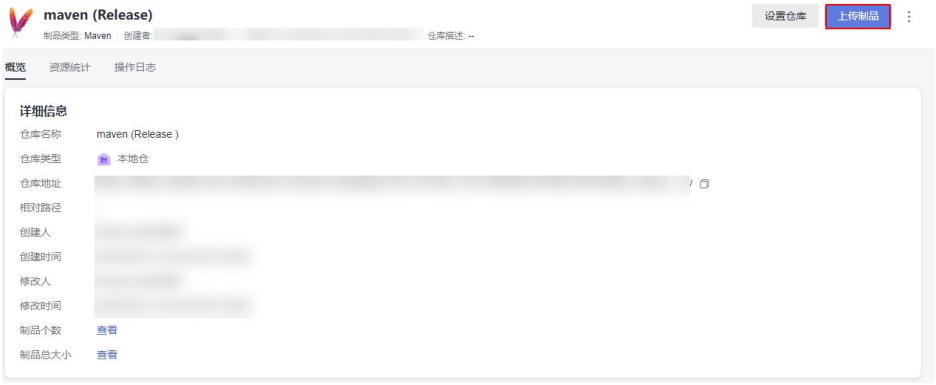
2.6 上传私有组件

只有仓库管理员与开发者角色才能够上传私有组件，在“独立用户权限”页面可设置仓库角色。

基础操作步骤

步骤 1 进入私有依赖库，在左侧边栏中单击待上传私有组件的目标仓库。

步骤 2 单击页面右侧“上传制品”。



步骤 3 弹框中输入组件参数，并上传文件，单击“上传”。每种类型组件的详细配置请参考以下各节中的说明

----结束

Maven 组件介绍

- POM：POM(Project Object Model，项目对象模型) 是 Maven 工程的基本工作单元，是一个 XML 文件，包含了项目的基本信息，用于描述项目如何构建，声明项目依赖等。执行构建任务时，Maven 会在当前目录中查找 POM，读取 POM，获取所需的配置信息，构建产生出目标组件。
- Maven 坐标：在三维空间中使用 X、Y、Z 唯一标识一个点。在 Maven 中通过 GAV 标识唯一的 Maven 组件包，GAV 是 **groupId**、**artifactId**、**version** 的缩写。

groupId 即公司或者组织，如 Maven 核心组件都是在 org.apache.maven 组织下。artifactId 是组件包的名称。version 是组件包的版本。

- Maven 依赖：依赖列表是 POM 的基石，大多数项目的构建和运行依赖于对其他组件的依赖，在 POM 文件中添加依赖列表。如 App 组件依赖 App-Core 和 App-Data 组件，配置如下：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.companyname.groupname</groupId>
    <artifactId>App</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <dependencies>
      <dependency>
        <groupId>com.companyname.groupname</groupId>
        <artifactId>App-Core</artifactId>
        <version>1.0</version>
      </dependency>
    </dependencies>
    <dependencies>
      <dependency>
        <groupId>com.companyname.groupname</groupId>
        <artifactId>App-Data</artifactId>
        <version>1.0</version>
      </dependency>
    </dependencies>
</project>
```

上传 Maven 组件

私有依赖库支持两种上传模式：POM 模式与 GAV 模式。

上传模式	说明
POM 模式	GAV 参数来自于 POM 文件，系统将保留组件的传递依赖关系。
GAV 模式	GAV，即 Group ID、Artifact ID、Version，是 jar 包的唯一标识。GAV 参数来源自于手动输入，系统将自动生成传递依赖的 POM 文件。

- POM 模式

  POM 模式可以只上传 pom 文件，也可上传 pom 文件与相关的组件，上传文件名称需要和 pom 文件中的 artifactId、version 一致。如下图，POM 中 artifactId 为 demo，version 为 1.0，上传的文件必须是 demo-1.0.jar。

pom 文件最基本结构如下：

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>demo</groupId>
  <artifactId>demo</artifactId>
  <version>1.0</version>
</project>
```

说明

modelVersion 这个标签必须存在，而且值必须是 4.0.0，这标志着使用的是 maven2。

当同时上传 POM 和 File 时，上传的 pom 文件中的 artifactId 和 version 需要与上传的 File 的名称对应，例如 pom 文件中的 artifactId 值为 demo，version 值为 1.0，则 File 文件名称必须为 demo-1.0，否则就会上传失败。

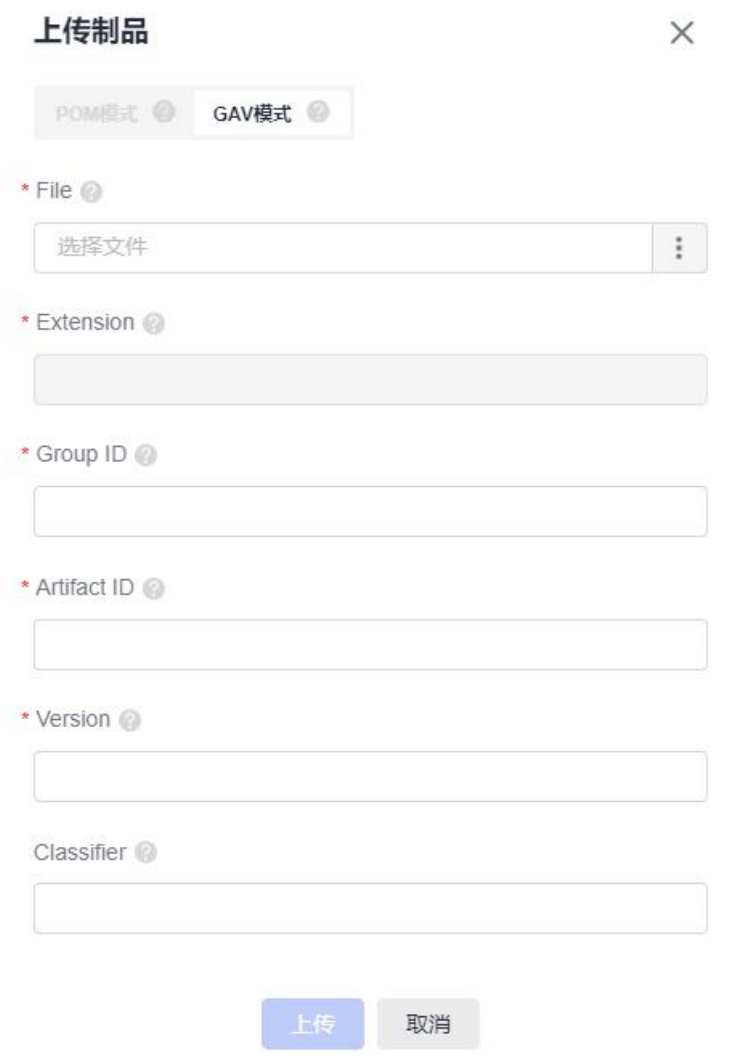
- GAV 模式

  GAV 模式模式中 Group ID、Artifact ID、Version 三个参数手动输入并决定上传文件名称，Extension 为打包类型，决定上传文件类型。

  Classifier 为分类，用于区分从同一 POM 构建出的具有不同内容的制品。该字段是可选的，支持大小写字母、数字、下划线(_)、连字符(-)和点(.)，如果输入会附加到文件名后。

  常见使用场景：

  - 区分不同版本：如 demo-1.0-jdk13.jar 和 demo-1.0-jdk15.jar。
  - 区分不同用途：如 demo-1.0-javadoc.jar 和 demo-1.0-sources.jar。



npm 组件介绍

npm 全称 Node Package Manager，是一个 JavaScript 包管理工具，npm 组件包就是 npm 管理的对象，而 npm 私有依赖库就是管理和存储 npm 组件包的一个私有仓库。

npm 组件包是由结构和文件描述组成：

- 包结构：是组织包中的各种文件，例如：源代码文件，资源文件等。
- 描述文件：描述包的相关信息，例如：package.json、bin、lib 等文件。

包中的 package.json 文件是对项目或模块包的描述文件，它主要包含名称、描述、版本、作者等信息，npm install 命令会根据这个文件下载所有依赖的模块。

package.json 示例如下：

```
{
  "name": "third_use",          //包名
  "version": "0.0.1",          //版本号
  "description": "this is a test project",  //描述信息
  "main": "index.js",           //入口文件
```

```
 "scripts": {                        //脚本命令
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "keywords": [                       //关键字
   "show"
 ],
 "author": "f",                 //开发者姓名
 "license": "ISC",                   //许可协议
 "dependencies": {                   //项目生产依赖
   "jquery": "^3.6.0",
   "mysql": "^2.18.1"
 },
 "devDependencies": {           //项目开发依赖
   "less": "^4.1.2",
   "sass": "^1.45.0"
 }
}
```

其中最重要的是 name 和 version 字段，这两个字段必须存在，否则当前包无法被安装，这两个属性一起形成了一个 npm 包的唯一标识。

name 是 package(包)的名称。名称的第一部分如@scope/在私有依赖库是必选的，用作名称空间。一般通过搜索 name 来安装使用需要的包。

```
{
 "name": "@scope/name"
}
```

version 是 package(包)的版本，一般为“x.y.z”格式。

```
{
 "version": "1.0.0"
}
```

上传 npm 组件

私有依赖库支持上传 tgz 格式的 npm 组件包，上传时需要配置以下两个参数。

参数	说明
PackageName	请与打包时的配置文件“package.json”中“name”保持一致。
Version	请与打包时的配置文件“package.json”中“version”保持一致。

说明

在上传组件时，PackageName 需要以创建仓库时添加的路径列表中的路径开头，详细可见帮助指导中的“仓库配置说明”。

例如：

创建 npm 仓库时，添加的路径为“@test”。

上传组件到该仓库时，“PackageName”中的“@test”存在于新建仓库时的路径列表中，可以成功上传。若使用其他不存在与列表中的路径，如“@npm”，则会上传失败。

上传成功之后，可在仓库组件列表中看到 tgz 格式的组件包，同时在路径“.npm”下生成对应的元数据。

上传 Go 组件

Go（又称 Golang）是 Google 开发的一种编程语言。GoLang1.11 开始支持模块化的包管理工具，模块是 Go 的源代码交换和版本控制的单元，mod 文件用来标识并管理一个模块，zip 文件是源码包。Go 模块主要分为两种：v2.0 以上版本，及 v2.0 以下版本，二者对 Go 模块的管理存在差异。

上传 Go 组件分为两步：上传 zip 文件与上传 mod 文件，需要分别输入以下参数。

参数	说明
zip path	zip 文件的完整路径。路径格式包括以下几种情况： • v2.0 以下版本：{moduleName}/@v/{version}.zip。 • v2.0 以上版本： – zip 包里有 go.mod 且路径里以/vN 结尾：{moduleName}/vX/@v/vX.X.X.zip。

参数	说明
	– zip 包里不含 go.mod 或 go.mod 第一行里不以/vN 结尾：{moduleName}/@v/vX.X.X+incompatible.zip。
zip file	zip 文件的目录结构。包括以下几种情况： • v2.0 以下版本：{moduleName}@{version}。 • v2.0 以上版本： – zip 包里有 go.mod 且路径里以/vN 结尾：{moduleName}/vX@{version}。 – zip 包里不含 go.mod 或 go.mod 第一行里不以/vN 结尾：{moduleName}@{version}+incompatible。
mod path	mod 文件的完整路径。路径格式包括以下几种情况： • v2.0 以下版本：{moduleName}/@v/{version}.mod。 • v2.0 以上版本： – zip 包里有 go.mod 且路径里以/vN 结尾：{moduleName}/vX/@v/vX.X.X.mod。 – zip 包里不含 go.mod 或 go.mod 第一行里不以/vN 结尾：{moduleName}/@v/vX.X.X+incompatible.mod。
mod file	mod 文件内容。包括以下几种情况： • v2.0 以下版本：module {moduleName} • v2.0 以上版本： – zip 包里有 go.mod 且路径里以/vN 结尾： module {moduleName}/vX – zip 包里不含 go.mod 或 go.mod 第一行里不以/vN 结尾：module {moduleName}

上传 PyPI 组件

建议进入工程目录（该目录下需含有配置文件 setup.py）执行以下命令将待上传组件打包成 wheel 格式（.whl）的安装包，安装包默认生成在工程目录的 dist 目录下；Python 软件包管理工具 pip 仅支持 wheel 格式安装包。

```
python setup.py sdist bdist_wheel
```

上传组件时需要配置以下两个参数。

参数	说明
PackageName	请与打包时的配置文件“setup.py”中“name”保持一致。
Version	请与打包时的配置文件“setup.py”中“version”保持一致。

上传成功之后，可在仓库组件列表中看到 whl 格式的安装包，同时在路径“.pypi”下生成对应的元数据，可用于 pip 安装。

上传 RPM 私有组件

RPM 简介

- RPM 是一种以数据库记录的方式来将所需要的软件安装到到 Linux 系统的一套软件管理机制。
- 一般建议使用以下规范打包命名 RPM 二进制文件。

软件名称-软件的主版本号.软件的次版本号.软件的修订号-软件编译次数.软件适合的硬件平台.rpm

例如：hello-0.17.2-54.x86_64.rpm。其中，“hello”是软件名称，“0”是软件的主版本号，“17”是软件的次版本号，“2”是软件的修订号，“54”是软件编译次数，“x86_64”是软件适合的硬件平台。

软件名称	主版本号	次版本号	修订号	编译次数	适合的硬件平台
hello	0	17	2	54	x86_64

注：上传组件时需要配置以下两个参数

参数	说明
Component	组件名称。
Version	RPM 二进制包的版本。

步骤 1 进入私有依赖库，在左侧边栏中单击待上传私有组件的目标仓库。

步骤 2 单击页面右侧“上传”。

步骤 3 在弹框中输入组件参数，并上传文件，单击“上传”。

----结束

上传成功之后，可在仓库组件列表中看到 RPM 二进制包，同时在组件名称路径下生成对应的元数据“repodata”目录，可用于 yum 安装。

上传 debian 私有组件

上传 debian 私有组件时，需要配置以下 5 个参数：

参数	参数说明
Distribution	软件包发行版本。
Component	软件包组件名称。
Architecture	软件包体系结构。
Path	软件包的存储路径，默认上传至根路径。
File	软件包的本地存储路径。

上传成功之后，可在仓库组件列表中看到 deb 格式的安装包，同时在路径“dists”下生成对应的元数据，可用于 debian 安装。

上传 NuGet 私有组件

NuGet 包是具有 .nupkg 扩展的单个 ZIP 文件，作为一种可共享的代码单元，开发人员可以把它发布到一个专用的服务器来共享给团队内其它成员。

制品仓库服务创建 NuGet 私有依赖库来托管 NuGet 包。

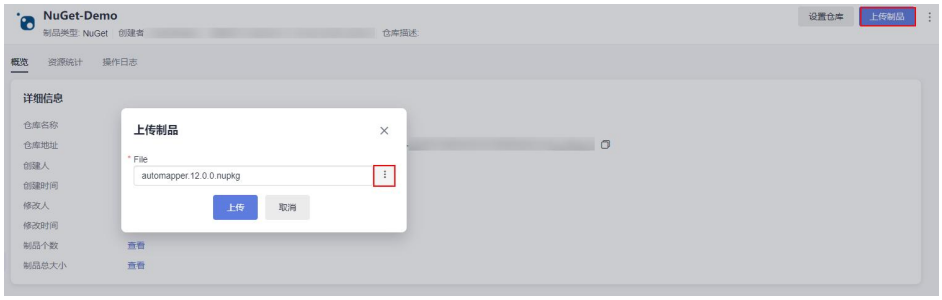
- 一般建议使用以下规范打包命名 NuGet 本地文件。

  软件名称-软件的主版本号.nupkg

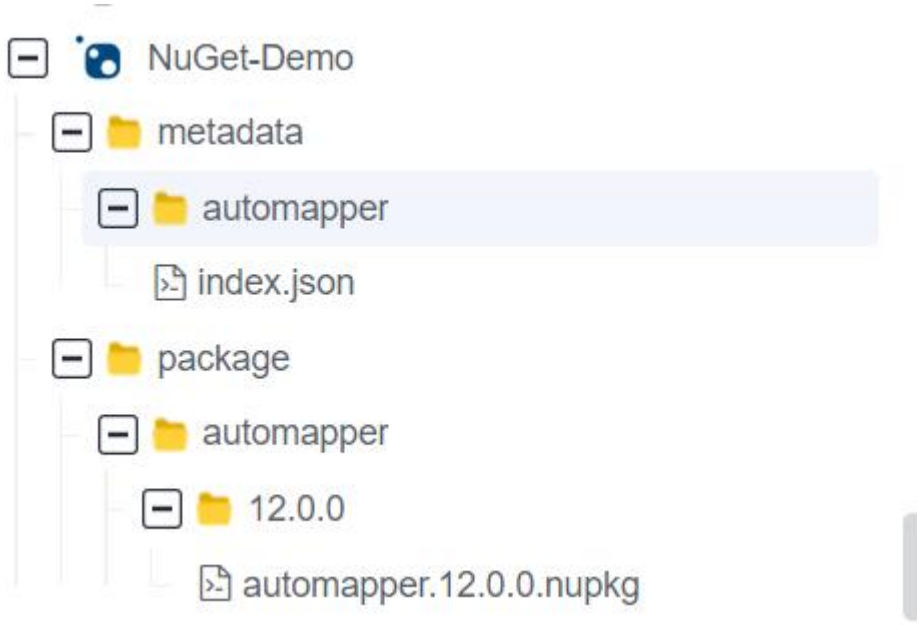
  例如：automapper.12.0.0.nupkg

步骤 1 进入私有依赖库，在左侧边栏中单击待上传私有组件的目标 NuGet 仓库。

步骤 2 单击“上传制品”，从本地选择待上传的 NuGet 文件，单击“上传”。



步骤 3 上传成功的组件显示在仓库列表中。



metadata 目录为元数据保存目录，由组件名称名称命名。元数据目录无法删除，会跟随对应组件的删除或还原进行删除或新增。

package 目录为组件保存目录。

----结束

2.7 客户端上传/下载私有组件

客户端上传 Maven 组件

- 使用客户端工具为 Maven，请确保已安装 JDK 和 Maven。
  a. 从私有依赖库页面下载 settings.xml 文件，将下载的配置文件直接替换或按提示修改 maven 的 settings.xml 文件。

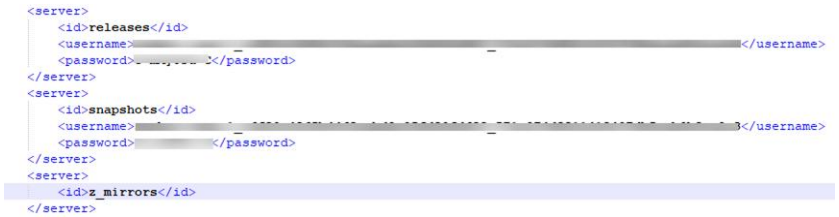
b. 使用以下命令进行客户端上传，命令示例如下：

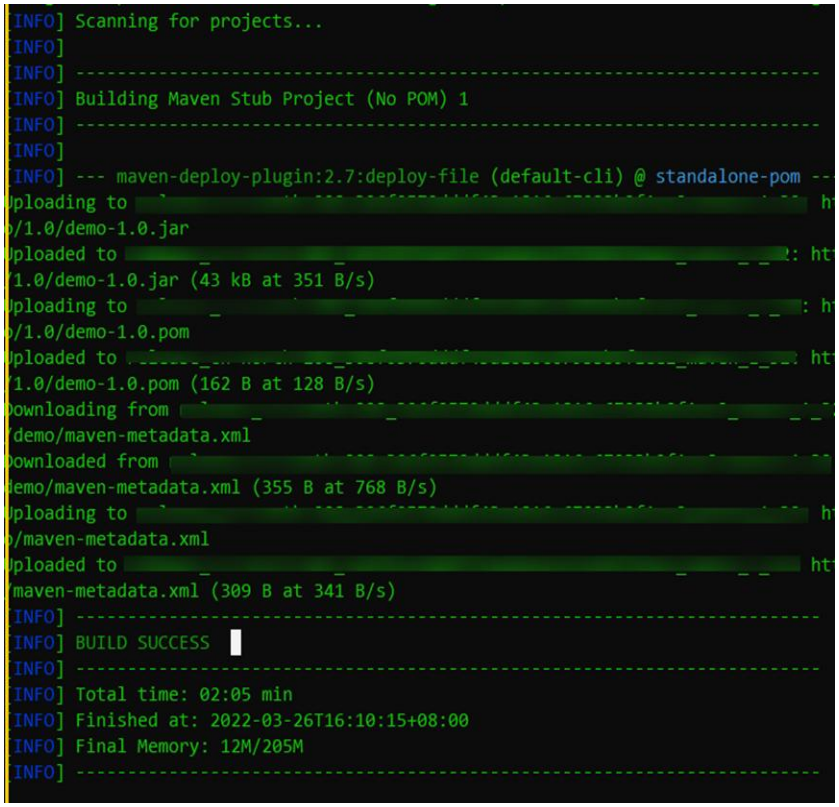
说明

上传时需要到上传的 pom 文件所在目录下执行命令

```
mvn deploy:deploy-file -DgroupId={groupId} -DartifactId={artifactId} -
Dversion={version} -Dpackaging=jar -Dfile={file_path} -DpomFile={pom_path}
-Durl={url} -DrepositoryId={repositoryId} -s {settings_path} -
Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true -
Dmaven.wagon.http.ssl.ignore.validity.dates=true
```

- **参数说明**
  - DgroupId ： 上传的 groupId
  - DartifactId ： 上传的 artifactId
  - Dversion ： 上传的版本 version
  - Dpackaging ： 上传包的类型（jar,zip,war 等）
  - Dfile ： 上传实体文件所在的路径
  - DpomFile : 上传实体 pom 文件所在的路径(Release 版本请注意：如果没有该参数，系统会自动生成 pom，pom 有特殊要求的请指定该参数)
  - pom 文件中的 DgroupId ， DartifactId ， Dversion 要与外面的一致，否则报 409。
  - DpomFile 和（DgroupId ， DartifactId ， Dversion） 可以二选一（即如果选择 DgroupId ， DartifactId ， Dversion，则可以不用 DpomFile）
  - Durl ： 上传文件到仓库的路径
  - DrepositoryId ： 这个是 settings 配置的用户名密码所对应的 id，如下图所示：





客户端下载 Maven 组件

- 使用客户端工具为 Maven，请确保已安装 JDK 和 Maven。

  1. 从私有依赖库页面下载 settings.xml 文件，将下载的配置文件直接替换或按提示修改 maven 的 settings.xml 文件。

  2. 使用以下命令进行客户端下载：

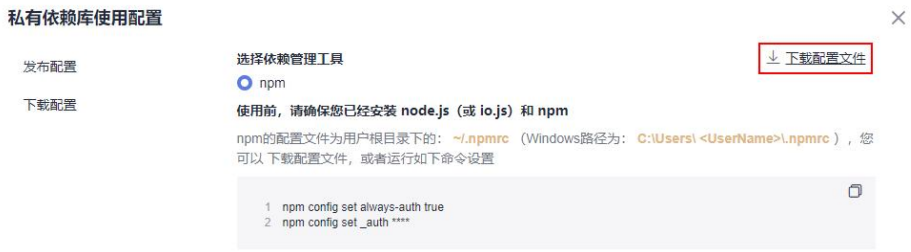
```
mvn dependency:get -DremoteRepositories={repo_url} -DgroupId={groupId} -
DartifactId={artifactId} -Dversion={version} -
Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true -
Dmaven.wagon.http.ssl.ignore.validity.dates=true
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-dependency-plugin:2.8:get (default-cli) @ standalone-pom ---
[INFO] Resolving demo:demo:jar:1.0 with transitive dependencies
Downloading from
/demo/1.0/demo-1.0.pom
Downloaded from
demo/1.0/demo-1.0.pom (0 B at 0 B/s)
Downloading from
/demo/1.0/demo-1.0.jar
Downloaded from
demo/1.0/demo-1.0.jar (0 B at 0 B/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.925 s
[INFO] Finished at: 2022-03-26T16:14:33+08:00
[INFO] Final Memory: 16M/194M
[INFO] ------------------------------------------------------------------------
```

客户端上传 npm 组件

- 使用客户端工具为 npm，请确保已安装 node.js（或 io.js）和 npm。

  1. 从私有依赖库页面下载“npmrc”文件，将下载的“npmrc”文件另存为“.npmrc”文件。



  2. 复制到用户目录下，路径为：Linux 系统路径为：~/.npmrc（C:\Users\<UserName>\.npmrc）。

  3. 进入 npm 工程目录(package.json 文件所在目录)，打开 package.json 文件，将创建仓库时填写的路径信息添加到 name 字段对应的值中。

```
{
  "name": "@test/demo",
  "version": "1.0.0",
  "description": "demo",
  "main": "index.js",
  "engines": {
    "node": ">=8.0.0",
    "npm": ">=5.0.0"
  },
```

  4. 执行以下命令将 npm 组件上传到仓库：

```
npm config set strict-ssl false
npm publish
```

客户端下载 npm 组件

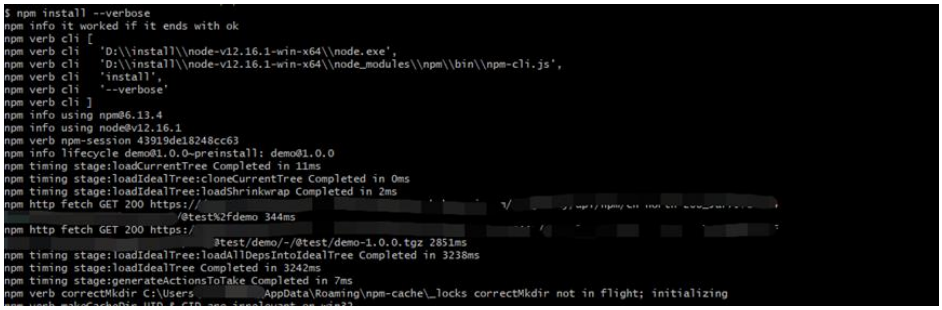
- 使用客户端工具为 npm，请确保已安装 node.js（或 io.js）和 npm。

  1. 从私有依赖库页面下载“npmrc”文件，将下载的“npmrc”文件另存为“.npmrc”文件。

  2. 复制到用户目录下，Linux 系统路径为：~/.npmrc (Windows 系统路径为：C:\Users\<UserName>\.npmrc)。

  3. 进入 npm 工程目录(package.json 文件所在目录)，执行以下命令下载 npm 依赖组件：

```
npm config set strict-ssl false
npm install --verbose
```



客户端上传 PyPI 组件

- 使用客户端工具为 python 和 twine，请确保已安装 python 和 twine。

1. 从私有依赖库页面下载“pypirc”文件，将下载的“pypirc”文件另存为“.pypirc”文件。

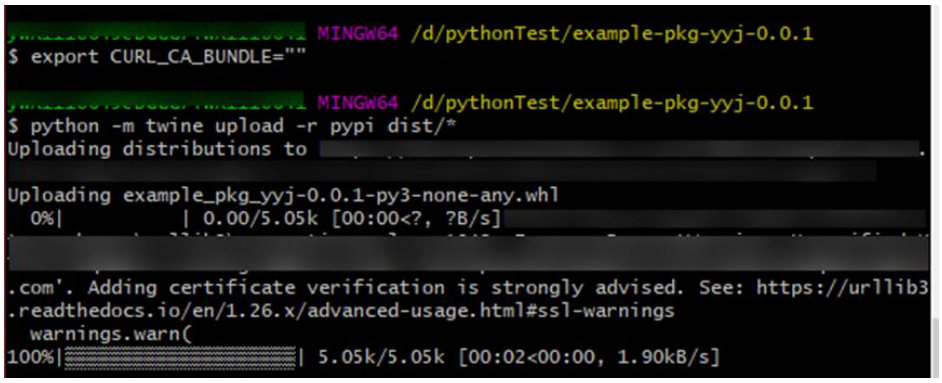
2. 复制到用户目录下，Linux 系统路径为：~/.pypirc (Windows 系统路径为：C:\Users\<UserName>\.pypirc)。

3. 进入 python 工程目录，执行以下命令将 python 工程打成 whl 包：

```
python setup.py bdist_wheel
```

4. 执行以下命令将文件上传到仓库：

```
python -m twine upload -r pypi dist/*
```



如果上传时报证书问题，请执行以下命令(Windows 系统请用 git bash 执行)设置环境变量跳过证书校验：

```
export CURL_CA_BUNDLE=""
```

说明

环境变量会因重新登录机器、切换用户、重新打开 bash 窗口等原因被清除，请在每次执行上传前添加环境变量。

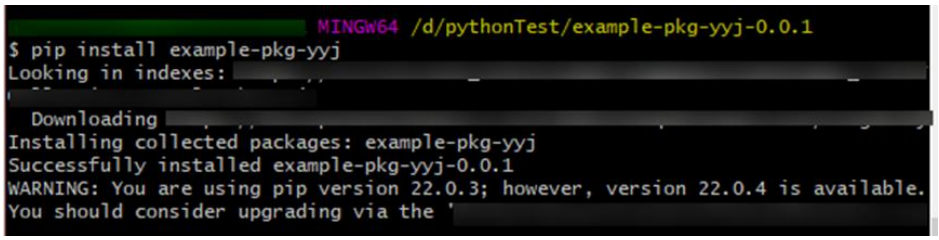
客户端下载 PyPI 组件

- 使用客户端工具为 python 和 pip，请确保已安装 python 和 pip。

  1. 从私有依赖库页面下载“pip.ini”文件，将“pip.ini”文件复制到用户目录下，Linux 系统路径为：~/.pip/pip.conf (Windows 系统路径为：C:\Users\<UserName>\pip\pip.ini)。

2. 执行以下命令安装 python 包：

```
pip install {包名}
```



客户端上传/下载 Go 组件

使用客户端工具为 go，请确保已安装 golang1.13 及以上版本，且工程为 go module 工程。

- Go Modules 打包方式简介及包上传。

  本文采用 Go Modules 打包方式完成 Go 组件的构建与上传。以下步骤中用到的 username 和 password 可以通过 Go 仓库的“配置指导”下载的配置文件中获取。

  打包命令主要包括以下几部分：

  a. 在工作目录中创建源文件夹。

  ```
  mkdir -p {module}@{version}
  ```

  b. 将代码源拷贝至源文件夹下。

  ```
  cp -rf . {module}@{version}
  ```

  c. 压缩组件 zip 包。

  ```
  zip -D -r [包名] [包根目录名称]
  ```

  d. 上传组件 zip 包与“go.mod”文件到私有依赖库中。

  ```
  curl -k -u {{username}}:{{password}} -X PUT {{repoUrl}}/{filePath} -T {{localFile}}
  ```

  根据打包的版本不同，组件目录结构有以下几种情况：

  - v2.0 以下版本：目录结构与“go.mod”文件路径相同，无需附加特殊目录结构。
  - v2.0 以上（包括 v2.0）版本：
    - “go.mod”文件中第一行以“/vX”结尾：目录结构需要包含“/vX”。例如，版本为 v2.0.1，目录需要增加“v2”。

□ “go.mod”文件中第一行不以“/vN”结尾：目录结构不变，上传文件名需要增加“+incompatible”。

下面分别对不同的版本举例说明。

v2.0 以下版本打包。

以下图所示“go.mod”文件为例。

go.mod

```
1  module example.com/demo
```

a. 在工作目录中创建源文件夹。

命令行中，参数“module”的值为“example.com/demo”，参数“version”自定义为 1.0.0。因此命令如下：

```
mkdir -p ~/example.com/demo@v1.0.0
```

b. 将代码源拷贝至源文件夹下。

参数值与上一步一致，命令行如下：

```
cp -rf . ~/example.com/demo@v1.0.0/
```

c. 压缩组件 zip 包。

首先，使用以下命令，进入组件 zip 包所在根目录的上层目录。

```
cd ~
```

然后，使用 zip 命令将代码压缩成组件包。命令行中，“包根目录名称”为“example.com”“包名”自定义为“v1.0.0.zip”，因此命令如下：

```
zip -D -r v1.0.0.zip  example.com/
```

d. 上传组件 zip 包与“go.mod”文件到私有依赖库中。

命令行中，参数“username”、“password”、“repoUrl”均可通过私有依赖库配置文件获取。

- 对于 zip 包，参数“filePath”为“example.com/demo/@v/v1.0.0.zip”，“localFile”为“v1.0.0.zip”。
- 对于“go.mod”文件，参数“filePath”为“example.com/demo/@v/v1.0.0.mod”，“localFile”为“example.com/demo@v1.0.0/go.mod”。

因此命令如下（参数 username、password、repoUrl 请参照私有依赖库配置文件自行修改）：

```
curl -k -u {{username}}:{{password}} -X PUT
{{repoUrl}}/example.com/demo/@v/v1.0.0.zip -T v1.0.0.zip
curl -k -u {{username}}:{{password}} -X PUT
{{repoUrl}}/example.com/demo/@v/v1.0.0.mod -T
example.com/demo@v1.0.0/go.mod
```

- **v2.0 以上版本打包，且“go.mod”文件中第一行以“/vX”结尾**。

以下图所示“go.mod”文件为例。

go.mod

```
1  module example.com/demo/v2
```

a. 在工作目录中创建源文件夹。

命令行中，参数“module”的值为“example.com/demo/v2”，参数“version”自定义为“2.0.0”。因此命令如下：

```
mkdir -p ~/example.com/demo/v2@v2.0.0
```

b. 将代码源拷贝至源文件夹下。

参数值与上一步一致，命令行如下：

```
cp -rf . ~/example.com/demo/v2@v2.0.0/
```

c. 压缩组件 zip 包。

首先，使用以下命令，进入组件 zip 包所在根目录的上层目录。

```
cd ~
```

然后，使用 zip 命令将代码压缩成组件包。命令行中，“包根目录名称”为“example.com”“包名”自定义为“v2.0.0.zip”，因此命令如下：

```
zip -D -r v2.0.0.zip  example.com/
```

d. 上传组件 zip 包与“go.mod”文件到私有依赖库中。

命令行中，参数“username”、“password”、“repoUrl”均可通过私有依赖库配置文件获取。

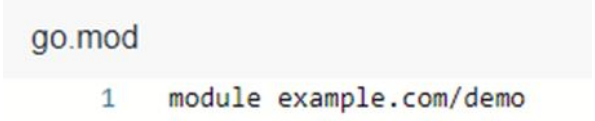
- 对于 zip 包，参数“filePath”为“example.com/demo/v2/@v/v2.0.0.zip”，“localFile”为“v2.0.0.zip”。
- 对于“go.mod”文件，参数“filePath”为“example.com/demo/v2/@v/v2.0.0.mod”，“localFile”为“example.com/demo/v2@v2.0.0/go.mod”。

因此命令如下（参数 username、password、repoUrl 请参照私有依赖库配置文件自行修改）：

```
curl -u {{username}}:{{password}} -X PUT
{{repoUrl}}/example.com/demo/v2/@v/v2.0.0.zip -T v2.0.0.zip
curl -u {{username}}:{{password}} -X PUT
{{repoUrl}}/example.com/demo/v2/@v/v2.0.0.mod -T
example.com/demo/v2@v2.0.0/go.mod
```

- **v2.0 以上版本打包，且“go.mod”文件中第一行不以“/vX”结尾。**

以下图所示“go.mod”文件为例。



a. 在工作目录中创建源文件夹。

命令行中，参数“module”的值为“example.com/demo”，参数“version”自定义为“3.0.0”。因此命令如下：

```
mkdir -p ~/example.com/demo@v3.0.0+incompatible
```

b. 将代码源拷贝至源文件夹下。

参数值与上一步一致，命令行如下：

```
cp -rf . ~/example.com/demo@v3.0.0+incompatible/
```

c. 压缩组件 zip 包。

首先，使用以下命令，进入组件 zip 包所在根目录的上层目录。

```
cd ~
```

然后，使用 zip 命令将代码压缩成组件包。命令行中，“包根目录名称”为“example.com”“包名”自定义为“v3.0.0.zip”，因此命令如下：

```
zip -D -r v3.0.0.zip  example.com/
```

d. 上传组件 zip 包与“go.mod”文件到私有依赖库中。

命令行中，参数“username”、“password”、“repoUrl”均可通过私有依赖库配置文件获取。

- 对于 zip 包，参数“filePath”为“example.com/demo/@v/v3.0.0+incompatible.zip”，“localFile”为“v3.0.0.zip”。
- 对于“go.mod”文件，参数“filePath”为“example.com/demo/@v/v3.0.0+incompatible.mod”，“localFile”为“example.com/demo@v3.0.0+incompatible/go.mod”。

因此命令如下（参数 username、password、repoUrl 请参照私有依赖库配置文件自行修改）：

```
curl -k -u {{username}}:{{password}} -X PUT
{{repoUrl}}/example.com/demo/@v/v3.0.0+incompatible.zip -T v3.0.0.zip
curl -k -u {{username}}:{{password}} -X PUT
{{repoUrl}}/example.com/demo/@v/v3.0.0+incompatible.mod -T
example.com/demo@v3.0.0+incompatible/go.mod
```

- **通过 go 客户端下载 Go 组件。**

  go 客户端无法忽略证书校验，需要先把私有依赖库对应的域名证书添加到本地证书信任列表里，执行以下步骤添加信任证书列表。

  1) 导出证书。

  ```
  openssl s_client -connect {host}:443 -showcerts </dev/null
  2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-
  /p' |openssl x509 -outform PEM >mycertfile.pem
  openssl x509 -outform der -in mycertfile.pem -out mycertfile.crt
  ```

  mycertfile.pem 和 mycertfile.crt 即为下载的证书。

  2) 把证书加入到根证书信任列表中。

  3) 执行 go 命令下载依赖包

  ```
  ##1.v2.0 以下版本包
  go get -v <moudlename>
  ##2.v2.0 以上(包含 2.0)版本包
  ##a.zip 包里有 go.mod 且路径以/vN 结尾的
  go get -v {{moduleName}}/vN@{{version}}
  ##b.zip 包里不含 go.mod 或 go.mod 第一行里不以/vN 结尾的
  go get -v {moduleName}}@{{version}}+incompatible
  ```

客户端上传/下载 Rpm 组件

使用 linux 系统和 yum 工具，请确保使用 linux 系统，且已安装 yum。

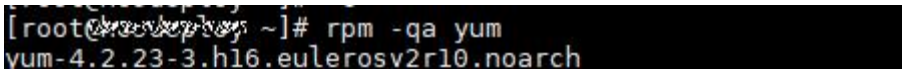
- **发布私有组件到 Rpm 私有依赖库**

步骤 1 检查 linux 下是否安装 yum 工具

在 linux 主机中输入

```
rpm -qa yum
```

如出现如下内容 则证明机器已安装 yum



步骤 2 登录软件开发生产线，进入 Rpm 私有依赖库。单击页面右侧⋮，在下拉栏中单击“配置指导”。

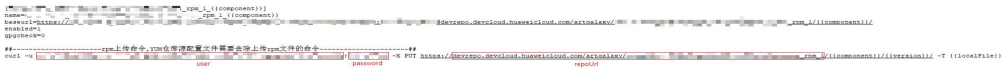
步骤 3 在弹框中单击“下载配置文件”。

步骤 4 在 Linux 主机中执行以下命令，上传 Rpm 组件

```
curl -k -u {{user}}:{{password}} -X PUT
https://{{repoUrl}}/{{component}}/{{version}}/ -T {{localFile}}
```

其中，“user”、“password”、“repoUrl”来源于上一步下载的配置文件中“rpm 上传命令”部分。

- user：位于 curl -u 与-X 之间、“:”之前的字符串。
- password：位于 curl -u 与-X 之间、“:”之后的字符串。
- repoUrl：“https://”与“/{{component}}”之间的字符串。



“component”、“version”、“localFile”来源于待上传的 Rpm 组件。以组件“hello-0.17.2-54.x86_64.rpm”为例。

  - component：软件名称，即“hello”。
  - version：软件版本，即“0.17.2”。
  - localFile：Rpm 组件，即“hello-0.17.2-54.x86_64.rpm”。

  完整的命令行如下图所示：

  命令执行成功，进入私有依赖库，可找到已上传的 Rpm 私有组件。

----结束

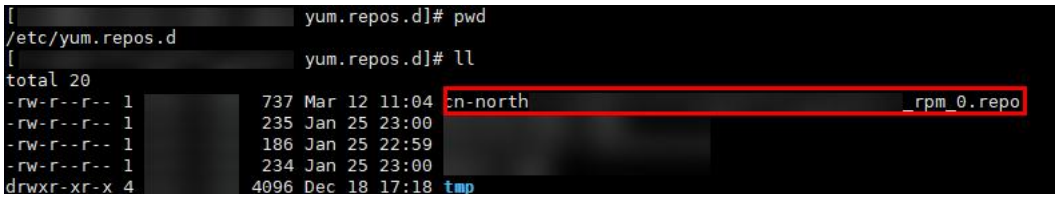
从 Rpm 私有依赖库获取依赖包

以•发布私有组件到 Rpm 私有依赖库中发布的 Rpm 私有组件为例，介绍如何从 Rpm 私有依赖库中获取依赖包。

步骤 1 参考发布 Rpm 私有组件的步骤 2、步骤 3，下载 Rpm 私有依赖库配置文件。

步骤 2 打开配置文件，将文件中所有“{{component}}”替换为上传 Rpm 文件时使用的“{{component}}”值（本文档中该值为“hello”），并删除“rpm 上传命令”部分，保存文件。

步骤 3 将修改后的配置文件保存到 Linux 主机的“/etc/yum.repos.d/”目录中。



步骤 4 执行以下命令，下载 Rpm 组件。其中，hello 为组件的“component”值，请根据实际情况修改。

```
yum install hello
```

----结束

客户端上传 Conan 组件

Conan 是 C/C++的包管理器，它适用于所有操作系统（Windows，Linux，OSX，FreeBSD，Solaris 等）。

前提条件：

- 已安装 Conan 客户端。
- 私有依赖库中已创建 Conan 仓库。

步骤 1 从私有依赖库页面选择对应的 Conan 仓库，单击“配置指导”，下载配置文件。

用户可以将得到的配置文件替换本地的 Conan 配置（Linux 路径为 ~/.conan/remotes.json，Windows 路径为 C:\Users\<UserName>\.conan\remotes.json）。

步骤 2 在使用配置页面复制并执行如下命令，将私有依赖库添加至本地 Conan 客户端中。

```
conan remote add Conan {repository_url}
conan user {user_name} -p={repo_password} -r=Conan
```

执行以下命令来查看远程仓库是否已经配置到 Conan 客户端中。

```
conan remote list
```

步骤 3 上传所有软件包至远程仓库，示例中 my_local_server 为远程仓库，实际使用过程中您可以替换为自己的仓库。

```
$ conan upload hello/0.1@demo/testing --all -r=my_local_server
```

步骤 4 查看远程仓库中已上传的软件包。

```
$ conan search hello/0.1@demo/testing -r=my_local_server
```

----结束

客户端下载 Conan 组件

步骤 1 从私有依赖库页面选择对应的 Conan 仓库，单击“配置指导”，下载配置文件。

用户可以将得到的配置文件替换本地的 Conan 配置（Linux 路径为 ~/.conan/remotes.json，Windows 路径为 C:\Users\<UserName>\.conan\remotes.json）。

步骤 2 执行安装命令来下载远程仓库中的 Conan 依赖包。

```
$ conan install ${package_name}/${package_version}@${package_username}/${channel} -r=cloud_artifact
```

步骤 3 执行搜索命令查看已下载的 Conan 软件包。

```
$ conan search "*"
```

步骤 4 执行删除命令移除本地缓存中的软件包。

```
$ conan remove ${package_name}/${package_version}@${package_username}/${channel}
```

----结束

NuGet 客户端上传组件

使用客户端工具为 NuGet，请确保已安装 NuGet。

步骤 1 从私有依赖库页面选择对应的 NuGet 仓库，单击“配置指导”，下载配置文件“NuGet.txt”。

步骤 2 打开下载的配置文件，使用如下命令，进行源的添加。

```
##----------------------NuGet add source----------------------##
nuget sources add -name {repo_name} -source {repo_url} -password {repo_password}
```

步骤 3 使用如下命令进行包的上传，替换<PATH_TO_FILE>为要上传文件的路径，执行上传语句（若有配置源，-source 后的参数可使用配置的源名称）。

```
##----------------------NuGet Download----------------------##
nuget install <PACKAGE>
```

----结束

dotnet 客户端上传组件

使用客户端工具为 dotnet，请确保已安装 dotnet。

说明

dotnet 客户端需要添加信任服务器证书，才可以使用。

- windows 信任证书步骤：

  1.导出服务器证书。

  ```
  openssl s_client -connect {host}:443 -showcerts </dev/null 2>/dev/null | sed
  -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' |openssl x509 -outform
  PEM >mycertfile.pem
  openssl x509 -outform der -in mycertfile.pem -out mycertfile.crt
  ```

  mycertfile.pem 和 mycertfile.crt 即为下载的证书。

  2.windows 需要使用 powershell 添加证书信任。

  添加证书

  ```
  Import-Certificate -FilePath "mycertfile.crt" -CertStoreLocation
  cert:\CurrentUser\Root
  ```

步骤 1 从私有库页面选择对应的 NuGet 仓库，单击“配置指导”，下载配置文件“dotnet.txt”。

步骤 2 打开配置文件，找到 dotnet add source 下的命令，进行源的添加。

```
##----------------------dotnet add source----------------------##
dotnet nuget add source {repo url} add -n {repo name} -u {user name} -p
{repo_password}
```

步骤 3 找到 dotnet upload 下的语句，替换<PATH_TO_FILE>为要上传文件的路径，执行上传语句。

```
##----------------------dotnet upload----------------------##
dotnet nuget push <PATH_TO_FILE> -s {repo_name}
```

----结束

NuGet 客户端下载组件

使用客户端工具为 NuGet，请确保已安装 NuGet。

步骤 1 从私有库页面选择对应的 NuGet 仓库，单击“配置指导”，下载配置文件“NuGet.txt”。

步骤 2 打开配置文件，找到 NuGet add source 下的命令，进行源的添加。

```
##----------------------NuGet add source----------------------##
nuget sources add -name {repo_name} -source{repo_url} -username {user_name} -
password {repo_password}
```

步骤 3 打开配置文件，找到 NuGet Download 下的语句，替换<PACKAGE>为要下载组件的名称，执行下载语句（若有配置源，-source 后的参数可使用配置的源名称）。

```
##----------------------NuGet Download----------------------##
nuget install <PACKAGE>
```

----结束

dotnet 客户端下载组件

使用客户端工具为 dotnet，请确保已安装 dotnet。

步骤 1 从私有库页面选择对应的 NuGet 仓库，单击“配置指导”，下载配置文件“dotnet.txt”。

步骤 2 打开配置文件，找到 dotnet add source 下的命令，进行源的添加。

```
##----------------------dotnet add source----------------------##
dotnet nuget add source {repo_url} add -n {repo_name} -u {user_name} -p
{repo_password}
```

步骤 3 找到 dotnet download 下的语句，替换< PACKAGE >为要下载组件的名称，执行下载语句。

```
##----------------------dotnet download----------------------##
dotnet add package <PACKAGE>
```

----结束

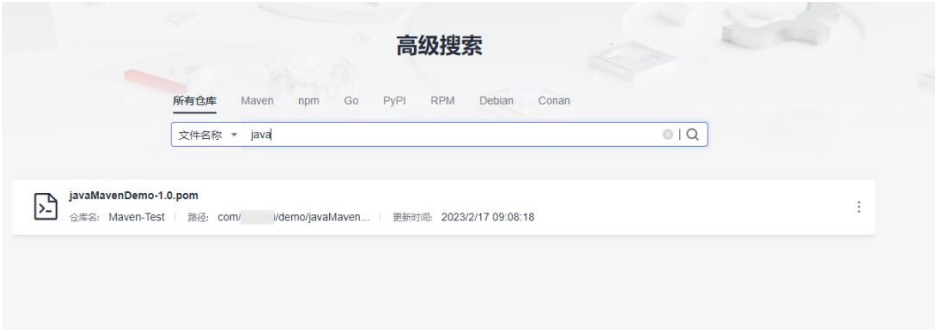
2.8 管理私有组件

搜索私有组件

步骤 1 进入私有依赖库，单击页面左上方“高级搜索”。

步骤 2 页面上方可以选择待查找组件所在的仓库（默认为所有仓库）。

步骤 3 在搜索框内输入文件名称的关键字，单击，即可搜索组件。

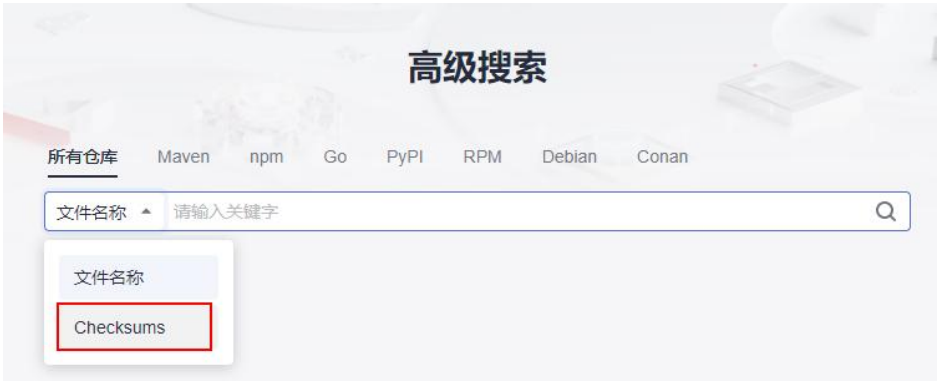


步骤 4 单击“文件名”可以查看组件的详细信息。

----结束

- 制品按照 checksums 搜索

1. 单击搜索框左侧的下拉列表，选择 Checksums（默认为文件名称）。



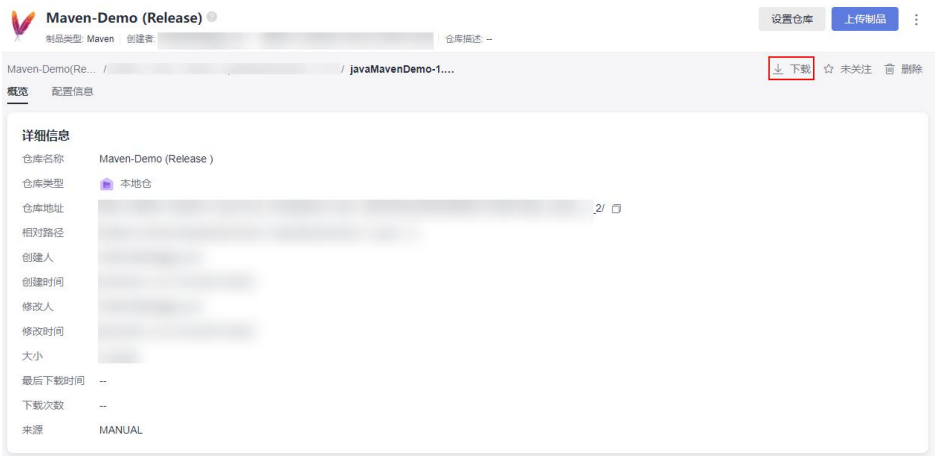
2. 输入“MD5/SHA-1/SHA-256/SHA-512 校验和”，单击也可以找到相应的组件。

下载私有组件

步骤 1 进入私有依赖库，在左侧边栏中找到需要下载的私有组件，单击组件名称。

若仓库或组件过多，可以通过搜索私有组件找到组件。

步骤 2 单击页面右侧“下载”。



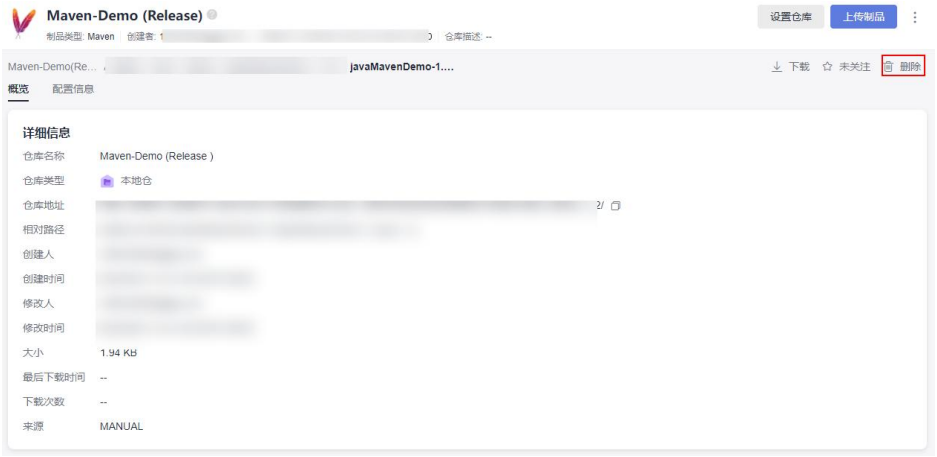
----结束

删除私有组件

步骤 1 进入私有依赖库，在左侧边栏中找到需要下载的私有组件，单击组件名称。

若仓库或组件过多，可以通过搜索私有组件找到组件。

步骤 2 单击页面右侧“删除”。



步骤 3 在弹框中单击“是”。

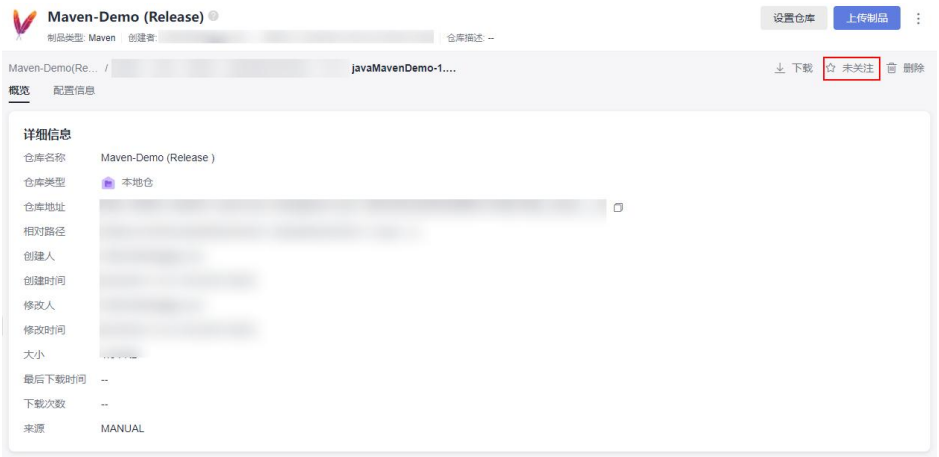
----结束

关注/取消关注

步骤 1 进入私有依赖库，在左侧边栏中找到需要下载的私有组件，单击组件名称。

若仓库或组件过多，可以通过搜索私有组件找到组件。

步骤 2 单击页面右侧“未关注”。



当图标变成  时，单击页面左侧最下方“我的关注”，即可查看已关注的组件列表。在列表中单击“path”值，页面将跳转至对应组件详情页。

----结束

2.9 管理回收站

在私有依赖库中被删除的仓库与组件都会移到回收站，可以对删除后的组件进行管理。

私有依赖库中分为“首页回收站”和“项目内回收站”。

首页回收站

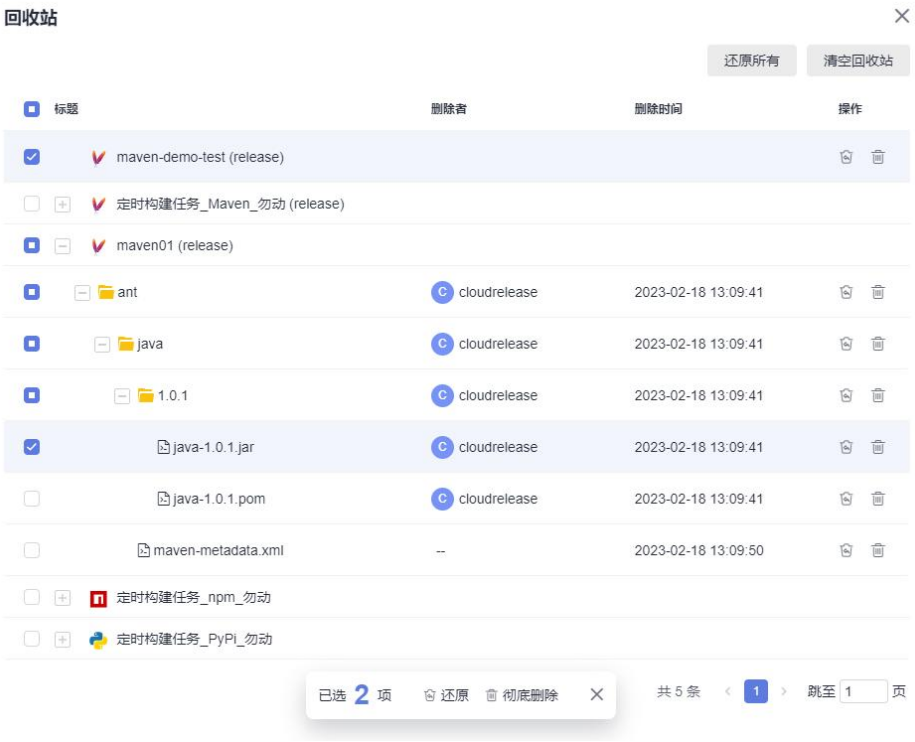
用户可以在制品仓库服务的首页回收站里，处理所有项目删除的组件。

步骤 1 通过首页入口进入私有依赖库页面。

步骤 2 单击页面右上方“回收站”，右侧滑出“回收站”页面。

步骤 3 根据需要对列表中的仓库与组件进行删除或还原操作。

列表中，若操作列中有 和 ，则表示此行是被删除的仓库；否则表示此行是被删除组件所在的仓库名称，单击仓库名称即可看到该仓库中被删除的组件。



可进行的操作如下：

操作类型	操作项	说明
还原	还原仓库	单击操作列的 ，可以还原对应仓库。

操作类型	操作项	说明
	还原单个组件	进入待还原组件所在仓库，在列表中单击操作列 ，可以还原对应组件。
	批量还原组件	进入待还原组件所在仓库，勾选多个组件，单击列表下方的“还原”，可以同时还原多个组件。
	还原所有	单击页面右上方“还原所有”，可以一键还原回收站中的所有仓库与组件。
删除	删除仓库	单击操作列，可以删除对应仓库。
	删除单个组件	进入待删除组件所在仓库，单击操作列，可以删除对应组件。
	批量删除组件	进入待删除组件所在仓库，勾选多个组件，单击列表下方的“彻底删除”，可以同时删除多个组件。
	清空回收站	单击页面右上方“清空回收站”，可以一键删除回收站中的所有仓库与组件。

----结束

项目内回收站

步骤 1 通过项目入口进入私有依赖库页面。

步骤 2 在页面左下方单击“回收站”，右侧滑出“回收站”页面。

步骤 3 根据需要对列表中的仓库与组件进行删除或还原操作。

列表中，若操作列中有 和 ，则表示此行是被删除的仓库；否则表示此行是被删除组件所在的仓库名称，单击仓库名称即可看到该仓库中被删除的组件。

可进行的操作如下：

操作类型	操作项	说明
还原	还原仓库	单击操作列的  ，可以还原对应仓库。
	还原单个组件	进入待还原组件所在仓库，在列表中单击操作列  ，可以还原对应组件。
	批量还原组件	进入待还原组件所在仓库，勾选多个组件，单击列表下方的“还原”，可以同时还原多个组件。
	还原所有	单击页面右上方“还原所有”，可以一键还原回收站中的所有仓库与组件。
删除	删除仓库	单击操作列  ，可以删除对应仓库。
	删除单个组件	进入待删除组件所在仓库，单击操作列  ，可以删除对应组件。
	批量删除组件	进入待删除组件所在仓库，勾选多个组件，单击列表下方的“彻底删除”，可以同时删除多个组件。
	清空回收站	单击页面右上方“清空回收站”，可以一键删除回收站中的所有仓库与组件。

----结束