制品仓库

用户指南

文档版本	01
发布日期	2024-07-02

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

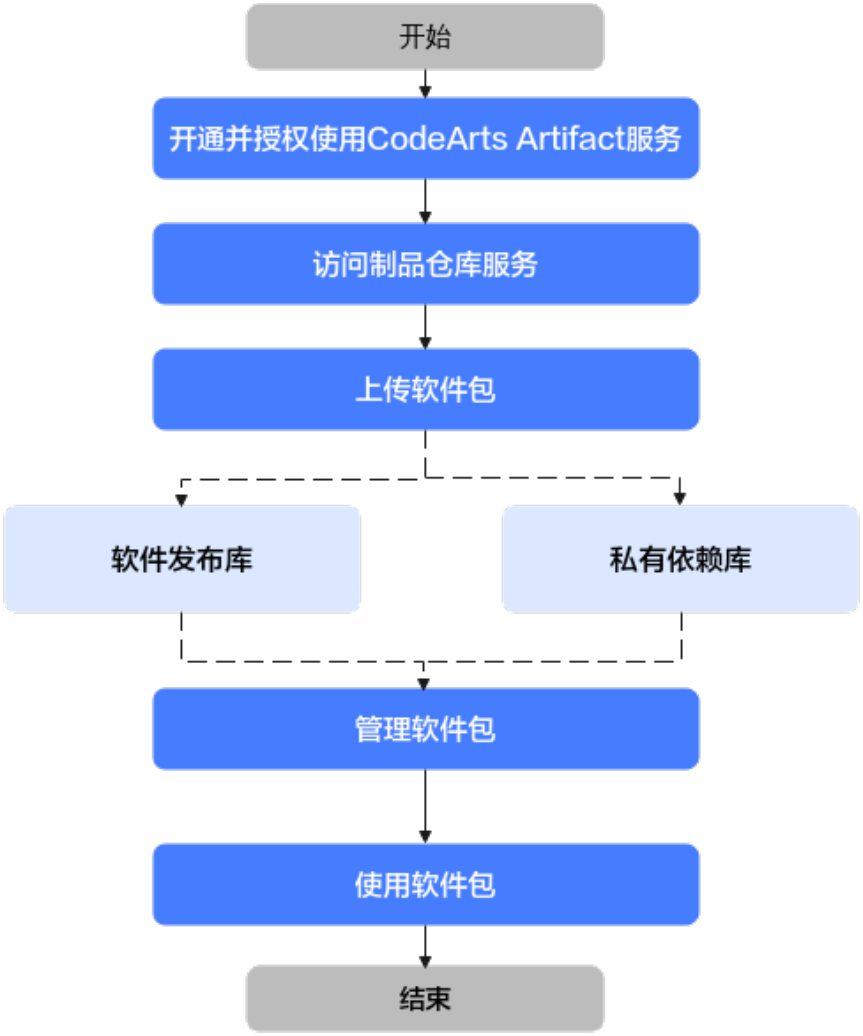
目 录

1 制品仓库服务(CodeArts Artifact)使用流程

制品仓库服务帮助开发者统一管理各种开发语言在开发、构建过程中的依赖，构建成果（二进制制品）以及交付过程关键信息的重要组件，支持Maven、npm等常见制品包类型。可以与本地构建工具和云上的持续集成、持续部署无缝对接，同时支持制品包版本管理、细粒度权限控制、安全扫描等重要功能，实现软件包生命周期管理，提升发布质量和效率。

在**软件开发生产线**解决方案中，制品仓库服务属于其中一个子服务，具体位置可参考**产品架构**。

制品仓库服务基本操作流程



2 购买并授权使用制品仓库(CodeArts Artifact)服务

前提条件

已**注册华为账号并开通华为云**。

购买须知

在**CodeArts支持的区域**内，各区域独立开通购买、独立计费。

您可以**购买制品仓库单服务套餐**，或者**开通/购买软件开发生产线服务组合套餐**，体验一站式、全流程、安全可信的软件开发生产线。

购买制品仓库服务需要您拥有租户账号，或拥有Tenant Administrator权限的IAM用户账号，配置权限策略方法请参考**创建用户组并授权**。

说明

若已经购买了CodeArts套餐，则不能再单独购买制品仓库服务。

规则说明

2020年10月16日前，未在CodeArts某个区域下产生过费用的用户，按照新版计费规则，需在该区域**开通/购买CodeArts**或单独购买制品仓库服务后使用。

在该区域内产生过费用的用户，延续旧版计费规则，可参考**购买服务**在该区域开通制品仓库服务包年/包月套餐。

例如：

用户于2020年3月在“华北-北京四”购买了半年期的包月套餐。由于疫情影响业务，套餐到期后关闭了CodeArts服务；2020年10月20日将重新启用CodeArts。

- 若用户仍使用“华北-北京四”，可以购买旧版套餐使用。
- 若用户使用其它区域，则需购买CodeArts或者制品仓库服务新版计费。

购买服务

步骤1 进入**购买制品仓库服务页面**。

步骤2 根据需要选择区域、商品、购买时长、是否自动续费，勾选同意声明后单击“下一步：确认订单”。

说明

建议根据您业务所在物理区域就近选择，以减少网络延时。购买的套餐只在对应的区域生效，不能跨区域使用。

步骤3 确认订单内容：若需要修改，单击“上一步”；若确认无误，单击“下一步”。

步骤4 选择支付方式后，确认付款。

在制品仓库控制台页可查看到购买的套餐信息。

----结束

变更 CodeArts Artifact 套餐规格

CodeArts Artifact支持变更套餐规格，变更影响请参见**变更配置后对计费的影响**。

步骤1 登录制品仓库控制台。

步骤2 找到CodeArts Artifact套餐，单击操作列中的“变更”。

步骤3 根据需要选择变更商品、变更类型，勾选同意声明，单击“下一步：确认订单”。

说明

若变更类型选择“续费变更”，则还需要选择续费时长。

步骤4 确认订单内容：若需要修改，单击“上一步”；若确认无误，单击“下一步”。

步骤5 根据页面提示完成支付。

----结束

购买资源扩展

制品仓库支持对存储和流量扩展，详情介绍请参见**资源扩展**。

步骤1 进入购买资源扩展页面。

步骤2 根据需要选择区域、商品、购买时长、是否自动续费相关配置项，勾选同意声明后单击“下一步：确认订单”。

步骤3 确认订单内容：若需要修改，单击“上一步”；若确认无误，单击“下一步”。

步骤4 根据页面提示完成支付。

----结束

退订服务

退订服务后，制品仓库任务会被删除，无法再进行使用，服务将停止计费。

步骤1 登录控制台，在左侧导航中单击“制品仓库”。

步骤2 单击页面右上角“退订”。在弹框中确认“退订服务后将即时清理服务中的数据，请提前做好备份并谨慎操作”信息，单击“确定”。

步骤3 确认退款信息，选择退订原因，勾选“我已确认本次退订金额和相关费用。”、“资源退订后，未放入回收站的资源将立即删除且无法恢复。我已确认数据完成备份或不再使用。”，单击“退订”。

步骤4 在弹出的窗口中确认退订信息，单击“退订”。

可以在“费用中心>我的订单>订单管理”中查看退订的处理进度以及退款订单详情。

----结束

3 管理软件发布库 2.0

3.1 软件发布库 2.0 的简介

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、安全扫描等重要功能，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。

说明

制品仓库服务（CodeArts Artifact）于2023年3月进行服务升级，老用户在该时间前的存量软件发布库及软件发布库内的资源存放在旧版软件发布库中。

软件发布库无需新建，用户创建项目后，会在新版软件发布库下自动创建同项目名称的软件发布库。

访问 CodeArts Artifact 的软件发布库 2.0

步骤1 开通制品仓库服务。

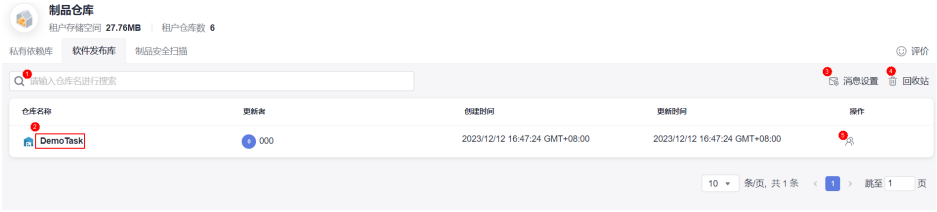
步骤2 添加成员并授权成员角色，请参考配置软件发布库2.0的权限。

步骤3 登录华为云控制台页面。

步骤4 单击页面左上角 ☰ ，在服务列表中选择“开发与运维 > 制品仓库 CodeArts Artifact”。

步骤5 单击“立即使用”，进入制品仓库服务首页。

步骤6 选择“软件发布库”页签，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可以找到该项目的同名软件发布库。
2	查看仓库	单击任一仓库名称，即可查看对应的软件发布库中归档的软件包/文件夹列表。可以完成软件包的上传、下载、编辑等管理操作。
3	消息设置	用户可以设置软件发布库的“容量阈值”，当仓库容量超过阈值时，通过发送邮件通知进行提醒。 **说明** 在回收站中的软件包仍然占用软件发布库的使用容量，当软件包从回收站彻底删除时释放所占用的容量。
4	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。
5	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。

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。

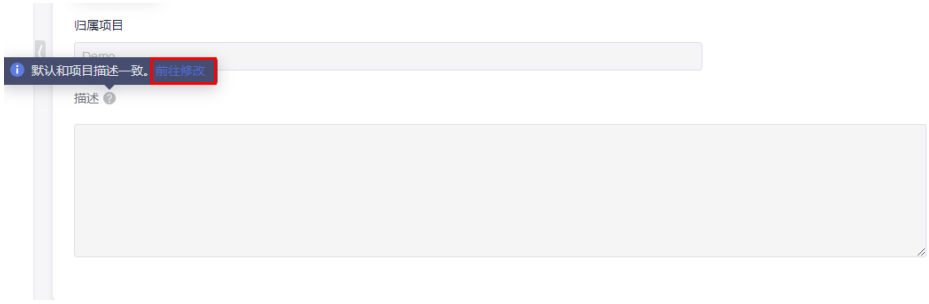
----结束

查看软件发布库基本信息

步骤1 在软件发布库页面，单击页面右上方“设置仓库”。

步骤2 页面显示软件发布库的仓库名称、制品类型、描述。

- 软件发布库的名称同所属项目的名称一致，无法修改。
- 软件发布库的描述会同步所属项目的描述，如下图，单击“前往修改”，可以跳转到项目基本信息页面修改描述。



----结束

3.2 配置软件发布库 2.0 的权限

新增的成员需赋予指定的角色，才可以正常的使用制品仓库服务。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 参考需求管理服务CodeArts Req的“用户指南 > 通用设置 > 服务权限管理 > 成员”章节**添加成员**添加成员，为新增的成员**赋予角色**。

步骤2 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤3 单击页面左上方 ···，在下拉栏中单击“项目权限设置”。

步骤4 单击需要“配置权限的角色”，选择“二进制仓”，根据需要单击编辑并勾选权限，单击保存。

软件发布库提供的默认权限矩阵如下表所示。

表 3-1 项目级权限

项目级权限说明									
角色/动作	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
项目经理	√	√	√	×	√	√	√	√	√
产品经理	×	×	√	×	√	√	√	×	×
测试经理	×	√	√	×	√	√	√	√	√
系统工程师	×	√	√	×	√	√	√	×	×
Committer	×	√	√	×	√	√	√	×	×
开发人员	×	√	√	×	√	√	√	×	×
测试人员	×	√	×	×	×	√	√	×	×
参与者	×	×	×	×	×	×	√	×	×
浏览者	×	×	×	×	×	×	×	×	×
项目管理者	√	√	√	√	√	√	√	√	√

说明

- 项目管理者默认拥有全部操作权限，无法修改其权限范围。
- 自定义角色无预置权限，可以联系管理员添加当前角色类型对应资源的相应操作权限。
- 项目管理者、项目经理、测试经理默认具有“权限配置”的权限。其他角色之前如有“权限配置”权限，可以继续在软件发布库内给其他角色配置权限。

----结束

3.3 上传软件包到软件发布库 2.0

新建文件夹

步骤1 在软件发布库页面，单击页面右侧“新建文件夹”或仓库视图中图标，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”或图标，可以创建第二级文件夹。

----结束

上传软件包

步骤1 单击页面右上方“上传制品”，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“上传制品”，可以手动上传本地软件包到对应的文件夹中。

步骤2 在弹框中配置如下信息后，单击“上传”。

- 目标仓库：当前软件发布库。
- 版本：用户可以为软件包设置版本号。
- 上传方式：选择“单个文件上传”或“多个文件上传”，本章节默认“单个文件上传”。
- 路径：用户设置路径名称后，仓库视图中会创建改名称的文件夹，上传的软件包会存放在该文件夹内。
- 文件：从本地选择需要上传到软件发布库的软件包。

步骤3 在仓库视图中，单击已上传的软件包名称，页面将展示该软件包的详细信息。

步骤4 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。

说明

不建议用户将带有明文账号密码等敏感信息的文件上传至软件发布库。

----结束

制品仓库服务支持从页面上传软件包，也支持与编译构建服务对接，将构建生成的软件包上传到软件发布库，请参考**上传软件包到软件发布库**。

搜索

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 在页面左侧搜索框中输入关键字（关键字可以为文件夹或文件名称)即可搜索出名称中有该关键字的软件包。

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

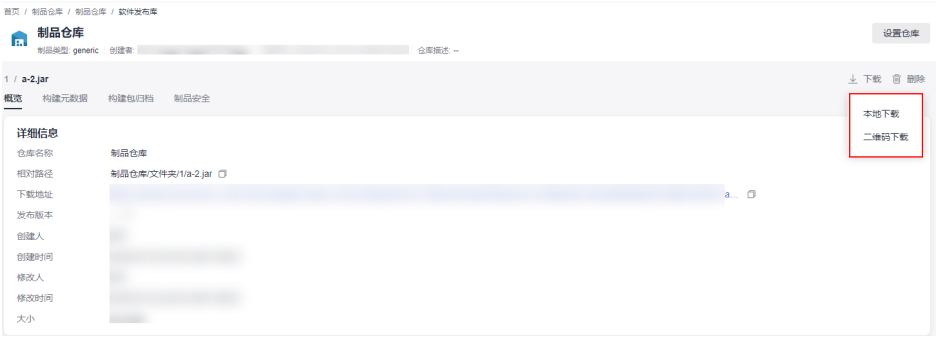
下载软件包

场景1

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 单击页面右侧“下载”。

步骤3 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

场景2

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 鼠标悬浮在选择需要下载的软件包，单击软件包右侧  。

----结束

3.4 管理软件发布库 2.0 中的软件包

在仓库视图中查看软件包

在仓库视图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、制品安全信息。

进入软件发布库，选择“仓库视图”页签，单击软件包名称，页面展示所选软件包详情。通过四个页签“概览”、“构建元数据”、“构建包归档”、“制品安全”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、修改人、修改时间、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。
- 制品安全：页面展示了对该软件包进行安全扫描的结果，详情请参考**制品安全扫描**。

在版本视图中查看软件包

制品仓库服务支持将软件包按照版本维度进行归类展示。在版本视图的列表中，提供按制品包名称和版本号进行展示，提供按照更新时间对文件进行排序。

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 用户需要为已上传的软件包编辑版本号（由编译构建归档的软件包发布版本默认为执行构建任务时设置的版本号）。

步骤3 在页面左上方选择“版本视图”页签，页面展示已设置版本的软件包列表。

步骤4 软件发布库将不同版本的同名软件包放在一个文件下。单击“文件名”，页面将显示该软件包最新版本的概览信息。

步骤5 单击“版本数”，页面将展示对应软件包的版本列表。

单击“版本号”，页面将显示该软件包的概览信息和文件列表。在文件列表中，单击“文件名称”，页面将跳转到软件包所在的存储位置。

步骤6 用户设置软件包的版本后，版本状态默认为“未发布”，可以修改版本状态。

- 在文件列表中，将文件的版本状态设置为“已发布”，该文件下最新版本的软件包会被设置为“已发布”。
- 单击“版本数”进入版本列表，可以分别将不同版本号的版本状态设置为“已发布”。

  说明

  版本状态由“未发布”变为“已发布”，状态转换不可逆，请谨慎操作。已发布状态的文件不可修改、不可编辑（修改名称、修改版本号），只能下载或删除。

----结束

搜索

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 在页面左侧搜索框中输入关键字（关键字可以为文件夹或文件名称)即可搜索出名称中有该关键字的软件包。

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

设置软件发布库中文件夹的发布状态

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“未发布”），单击“状态”列中的，在下拉栏中修改对应层级文件夹的状态。

如果文件夹的状态为“已发布”，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

文件夹的状态可由未发布变为已发布，状态转换不可逆，请谨慎操作。

----结束

下载软件包

场景1

步骤1 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

场景2

步骤1 鼠标悬浮在选择需要下载的软件包。

步骤2 单击软件包右侧  。

----结束

3.5 设置软件发布库 2.0 的清理策略

软件发布库提供定时自动清理文件功能。可根据设置文件的保留时长，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 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 单击页面右上方“设置仓库”，选择“清理策略”页签。

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**设置生产包的发布状态**。
- 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)。

----结束

3.6 管理软件发布库 2.0 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。

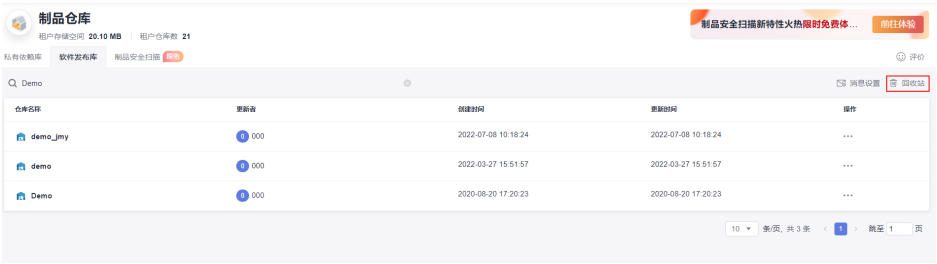
制品仓库服务提供“总回收站”和“项目内回收站”。

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 **访问制品仓库服务首页**。

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	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。
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	单个删除	单击操作列，可以删除对应软件包/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. 在总回收站内，勾选多个文件进行批还原除或删除时，不支持跨项目还原或删除跨项目文件。

----结束

项目内回收站

用于处理项目内删除的软件包/文件夹。

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”。

步骤2 单击页面左下方“回收站”。

步骤3 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	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。
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	单个删除	单击操作列  ，可以删除对应软件包/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

4 管理私有依赖库 2.0

4.1 私有依赖库 2.0 的简介

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、NuGet、PyPI、RPM、Debian、Conan、、原生-Docker、CocoaPods制品仓库。

说明

制品仓库服务（CodeArts Artifact）于2023年3月进行服务升级，老用户在该时间前的存量私有依赖库没有绑定项目，私有依赖库及私有依赖库内的资源存放在旧版私有依赖库中。

访问 CodeArts Artifact 的私有依赖库

步骤1 开通制品仓库服务。

步骤2 添加成员并授权成员角色，请参考配置软件发布库2.0的权限。

步骤3 登录华为云控制台页面。

步骤4 单击页面左上角 ☰ ，在服务列表中选择“开发与运维 > 制品仓库 CodeArts Artifact”。

步骤5 单击“立即使用”，进入制品仓库服务首页。

步骤6 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 所有制品类型 ,在下拉栏中可以按制品类型查看仓库。

步骤7 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

4.2 新建私有依赖库 2.0

首次使用私有依赖库时，需要新建仓库。私有依赖库分为“本地仓”、“聚合仓”。

本地仓：托管在服务端的制品仓库，是实际物理仓库，用户可以在本地仓上传不同类型制品。

聚合仓：用户在聚合仓中可以设置代理源与本地三方依赖仓库对接，也具备本地仓的功能，提供统一制品仓库入口，简化客户配置。

步骤1 单击项目卡片进入项目（若没有项目，请**新建项目**）。

步骤2 单击菜单栏“制品仓库 > 私有依赖库”，进入私有依赖库，用户可以单击页面左上方 ＋ 新建进行创建。

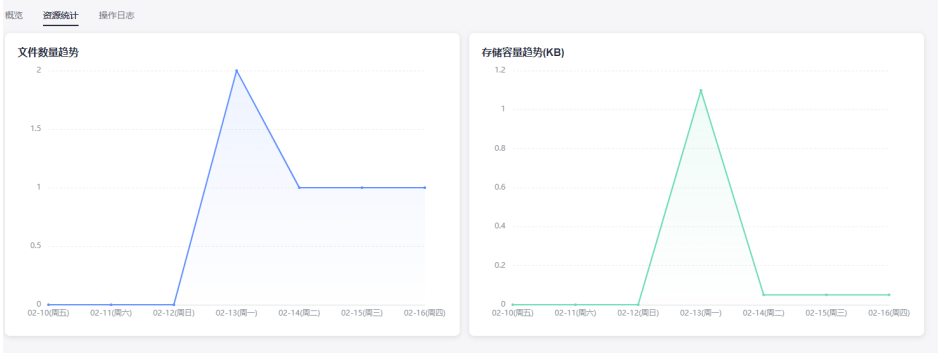
步骤3 进入“新建私有依赖库”页面。

步骤4 配置仓库基本信息，单击“确定”。

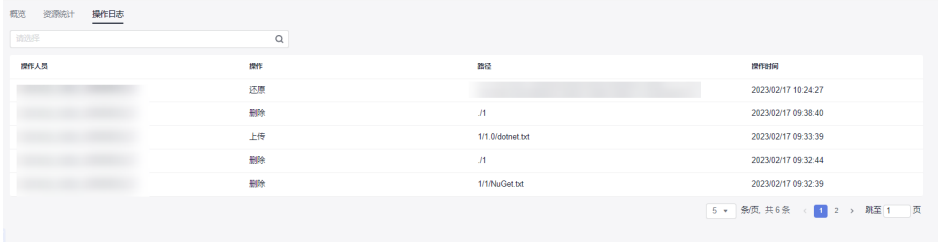
配置项	是否必填项	说明
仓库类型	是	用户可以选择“本地仓”或“聚合仓”。
仓库名称	是	仅支持中文，英文，数字，下划线(_)，连字符(-)和点(.)，长度20字符以内。 **说明** 私有依赖库新建完成后，仓库的名称不支持修改。
制品类型	是	本地仓支持Maven、npm、Go、PyPI、RPM、debian、Conan、NuGet、原生-Docker、CocoaPods制品仓库。 聚合仓支持Maven、npm、PyPI、原生-Docker制品仓库。 选择不同格式的仓库，页面会展示对应的配置，请参照**仓库配置说明**完成进一步配置。
归属项目	是	为当前创建的仓库选择归属项目。设置完成后，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

步骤5 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。
	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Maven文件上传到私有库。
npm	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npm文件上传到私有库。
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。
Conan	添加路径	否	输入需要添加的路径，单击“+”。 只允许在本地客户端上传以该路径开头的Conan文件到私有库。
原生-Docker	添加路径	否	输入需要添加的路径，单击“+”。 镜像推送时，只允许以该路径开头的镜像文件推送到私有库。

仓库格式	配置项	是否必填项	说明
CocoaPods	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CocoaPods类型文件上传到私有库。

为私有依赖库聚合仓设置代理

CodeArts Artifact新增自定义代理仓功能，允许用户创建自定义代理仓库来代理开源社区仓库和三方依赖仓库，通过代理仓下载文件后，支持将对应文件缓存到制品仓库，解决三方依赖下载慢的用户痛点，下载三方依赖达到和本地仓库一样的极致下载体验。

说明

私有依赖库支持Maven、npm、PyPI三种类型仓库进行代理设置。

在私有依赖库中，用户可以向Maven、npm、PyPI、原生-Docker虚仓中添加自定义镜像源。自定义镜像源配置方式如下：

步骤1 登录软件开发生产线首页，单击页面右上角用户名，在下拉菜单中选择“租户设置”。

步骤2 在页面左侧导航栏选择“镜像仓管理 > 镜像仓”。

步骤3 选择“自定义代理源”页签，单击页面右上方“新增代理”。

步骤4 在弹框中，选择制品类型，输入镜像仓名称（必填）、镜像仓代理地址（必填）、PyPI索引代理地址（制品类型为PyPI时必填）、代理账号、代理密码。

说明

1. 镜像仓代理地址请填写以“https://”、“http://”为开头的地址，否则会报“URL不合法”。
2. 不填写代理密码，默认使用上次设置的密码。

步骤5 单击“确定”，完成自定义代理源添加。

步骤6 对于已添加的自定义代理源，可以进行以下操作。

操作	说明
编辑	操作列中的，可以修改镜像仓名称、代理账号、代理密码。
删除	操作列中的，可以删除该自定义代理源。 若待删除的自定义代理源已关联了私有依赖库，需要在对应仓库的“代理设置”页面中移除该代理源后，返回本页面完成删除操作。

----结束

为私有依赖库聚合仓添加代理：

步骤1 已创建聚合仓，可参考**新建私有依赖库**。

步骤2 进入私有依赖库，在左侧边栏中选择对应聚合仓的仓库名称。

步骤3 单击页面右侧“代理设置”。

步骤4 单击“添加代理”，选择“公开源”或“自定义源”。

用户可以在“自定义源”中选择“第三方仓库”或“华为本地仓库”两种代理类型。

- 第三方仓库（设置第三方仓库或者由用户自行创建的仓库为代理源）：
  用户选择第三方仓库后，单击“代理名称”的下拉列表，在下拉列表中选择自定义代理源。新建自定义代理源请参考**自定义代理源**。
- 华为本地仓库（设置华为本地仓库为代理源，用户只能选择自己作为仓库管理员的本地仓库）
  用户在镜像仓名称的下拉列表中，可以选择私有依赖库中的本地仓库。

步骤5 单击“确定”，完成添加代理。

- 单击操作列中的，可以修改镜像仓名称、代理账号、代理密码。

  说明

  用户无法编辑华为本地仓库的代理源。

  单击操作列中的，可以删除对应的代理。

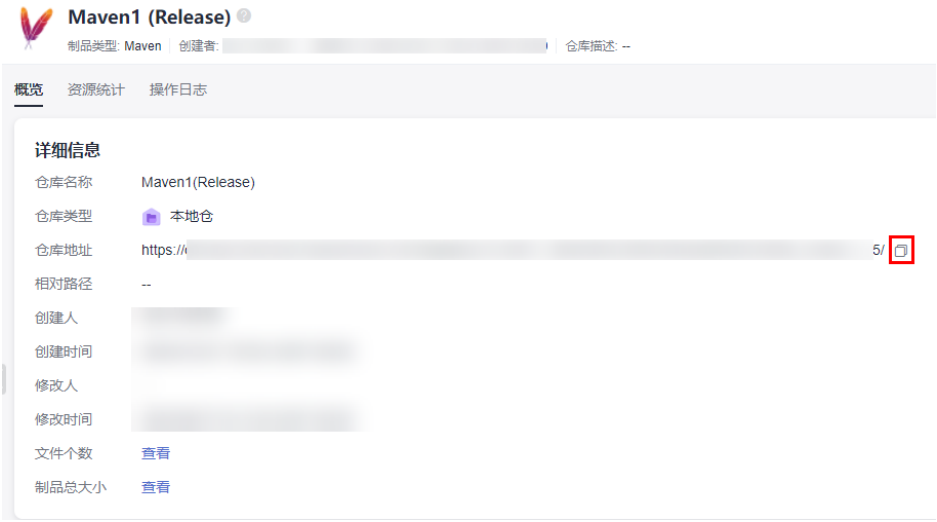
----结束

获取私有依赖库地址

用户新建成功的私有依赖库都会生成一个对应的私有依赖库地址，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

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

4.3 配置私有依赖库 2.0 权限

添加成员并授权成员角色

新增的成员需赋予指定的角色，才可以正常的使用制品仓库服务。

在私有依赖库中，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 登录CodeArts首页。

步骤2 单击目标项目名称，进入项目。

步骤3 单击菜单**“设置 > 通用设置 > 服务权限管理”**，进入项目权限管理界面。

步骤4 单击需要“配置权限的角色”，选择“二进制仓”，根据需要单击编辑并勾选权限，单击保存。

----结束

表 4-1 项目级权限

项目级权限说明										
角色/动作	创建仓库	编辑仓库	删除仓库	还原	彻底删除	还原所有	清空回收站	上传私有库组件	下载/查看私有库组件	删除/覆盖上传私有库组件
项目经理	√	√	√	√	√	√	√	√	√	√
产品经理	×	√	×	√	√	×	×	√	√	×
测试经理	×	√	×	√	×	√	×	√	√	×
运维经理	×	√	×	√	√	√	√	√	√	×

项目级权限说明										
系统工程师	×	√	×	√	√	×	×	√	√	×
Committer	×	√	×	√	√	×	×	√	√	×
开发人员	×	√	×	√	√	×	√	√	√	×
测试人员	×	×	×	√	×	×	×	√	√	×
参与者	×	×	×	×	×	×	×	×	√	×
浏览者	×	×	×	×	×	×	×	×	√	×
项目管理员	√	√	√	√	√	√	√	√	√	√

说明

- 租户管理员（拥有**Tenant Administrator**权限的IAM用户账号）可以对租户下所有的项目进行管理设置，非租户管理员的项目创建者拥有上述表格列出的操作权限。
- 如果管理员在创建IAM用户时，没有将其加入任何用户组，新创建的IAM没有任何权限，管理员可以在IAM控制台为其授予权限。授权后，用户即可根据权限使用账户中的云服务资源，请参考**创建用户并授权**。
- 自定义角色无预置权限，可以联系管理员添加当前角色类型对应资源的相应操作权限。

在私有依赖库中管理仓库权限

制品仓库服务支持在项目下统一配置项目各角色对当前项目下私有依赖库的默认操作权限，请参考**添加成员并授权成员角色**。

用户也可以单独配置对应私有依赖库的仓库权限。

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，当前项目下的角色显示在页面中。

步骤4 在角色列表中，单击需要修改权限的角色，勾选或取消勾选相关权限，单击“保存”。

----结束

说明

1. 新建成功的私有依赖库默认对接项目下“通用设置 > 权限管理”的角色权限，在“权限管理”修改后的角色权限会同步到私有依赖库的仓库权限。
2. 当用户没有在对应私有库下修改相关角色的仓库权限，在“权限管理”修改该角色的权限会同步到对应私有依赖库的仓库权限。
3. 当用户在对应私有库下修改了相关角色的仓库权限，在“权限管理”修改该角色的权限将不会同步到对应私有依赖库的仓库权限，请在仓库下进行后续该角色的权限修改。

4.4 管理私有依赖库 2.0

4.4.1 查看私有依赖库基本信息并配置仓库路径

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、版本策略不能修改。

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、原生-Docker、CocoaPods添加路径。

单击 🗑 可以删除路径。

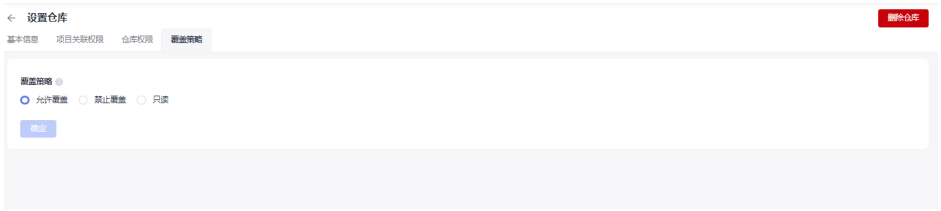
----结束

4.4.2 配置私有依赖库覆盖策略

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，系统将自动保存。

----结束

4.4.3 配置 CodeArts Artifact 中 Maven 仓库的清理策略

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。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。

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 私有依赖库”，进入私有依赖库。

步骤2 在左侧仓库列表中选择对应的“Snapshot”类型的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

4.4.4 关联 CodeArts Artifact 中的 Maven 仓库与项目

在私有依赖库中，将Maven格式仓库与多个项目关联后，项目中的构建任务即可完成在构建步骤中选择该仓库，将构建出来的产物存放到Maven仓库中。

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

在编译构建服务中，需要将构建产物上传至私有依赖库的配置方法，请参见**发布依赖包到CodeArts私有依赖库**。

4.5 通过私有依赖库页面上传下载私有组件

仓库管理员与开发者角色能够上传私有组件，可以在“仓库权限”中可设置仓库角色。

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

说明

1. 私有依赖库通过页面上传单文件大小限制：Maven/npm/PyPI/RPM/Debian类型最大限制为100MB， NuGet最大限制为20MB。
2. 建议不要将带有明文账号密码等敏感信息的文件上传至私有依赖库。

----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。

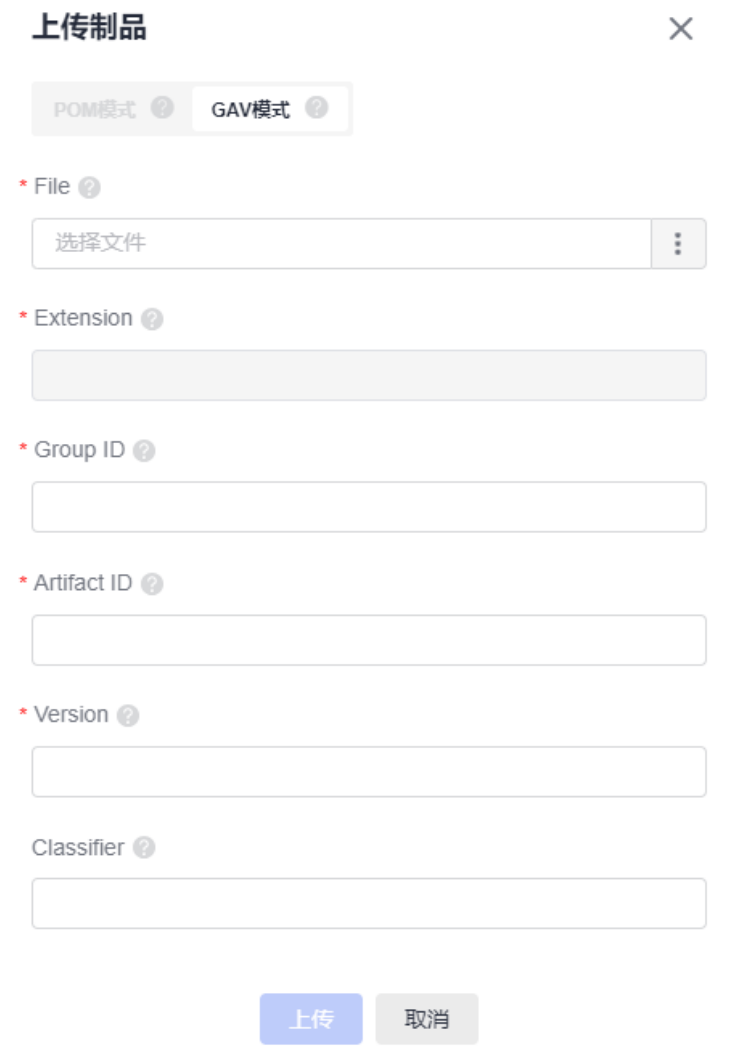
- GAV模式

  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”用作名称空间；另一部分“name”，一般通过搜索该“name”字段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。 – 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 全名 RedHat Package Manager，是由Red Hat公司提出，被众多Linux发行版本所采用，是一种以数据库记录的方式来将所需要的软件安装到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“上传制品”。

步骤3 在弹框中输入组件参数，并上传文件，单击“上传”。

----结束

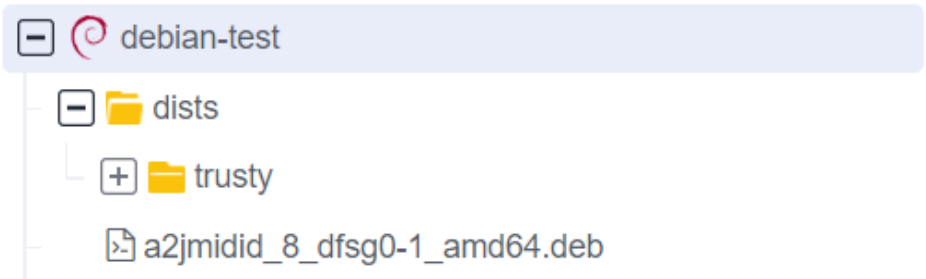
上传成功之后，可在仓库组件列表中看到RPM二进制包，同时在组件名称路径下生成对应的元数据“repodata”目录，可用于yum安装。

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

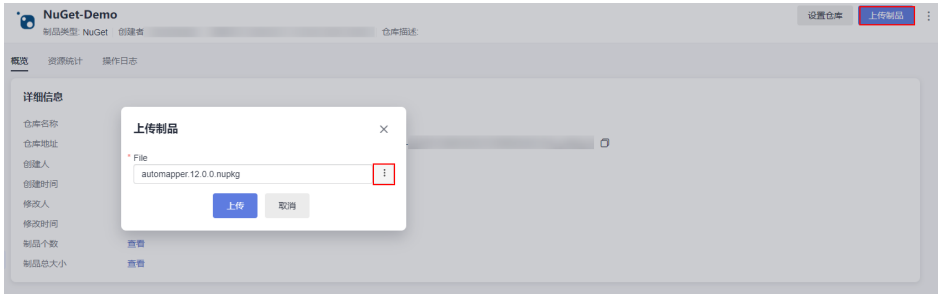
NuGet 包是具有 .nupkg 扩展的单个 ZIP 文件，作为一种可共享的代码单元，开发人员可以把它发布到一个专用的服务器来共享给团队内其它成员。

制品仓库服务创建NuGet私有依赖库来托管NuGet包。

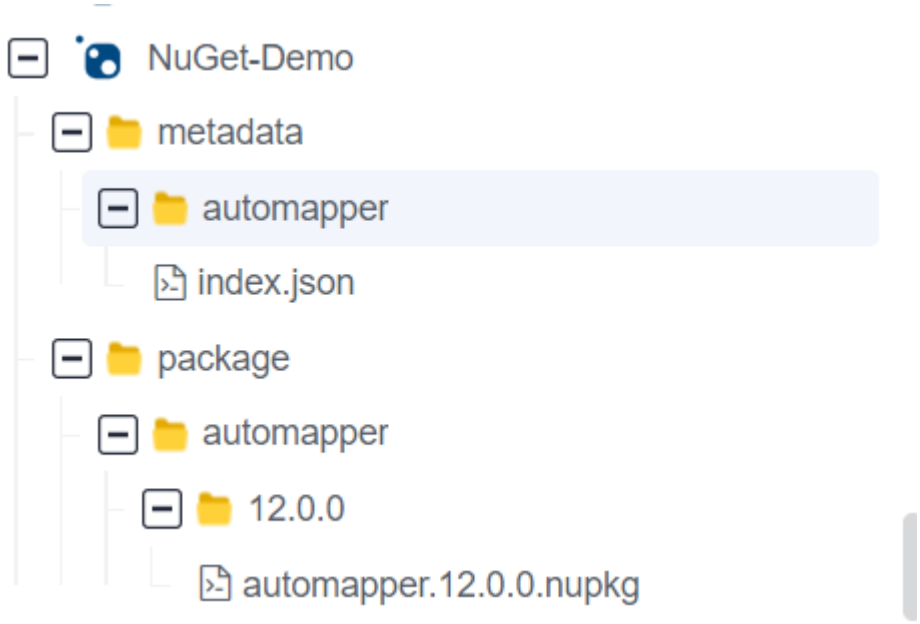
- 一般建议使用以下规范打包命名NuGet本地文件。
  软件名称-软件的主版本号.nupkg
  例如：automapper.12.0.0.nupkg

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

上传原生-Docker 组件

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Docker仓库。

步骤2 单击“上传制品”，配置页面信息（详见下表）。

参数	说明
上传方式	默认为“单个文件上传”。
PackageName	输入制品包名称。
文件	从本地选择待上传的Docker文件。

步骤3 单击“上传”。

----结束

上传 CocoaPods 组件

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CocoaPods仓库。

步骤2 单击“上传制品”，配置页面信息（详见下表）。

参数	说明
上传方式	默认为“单个文件上传”。
PackageName	输入制品包名称。
文件	从本地选择待上传的CocoaPods文件。

步骤3 单击“上传”。

----结束

4.6 通过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。

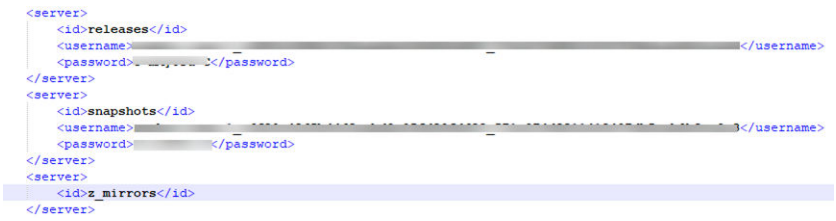
  b. 使用以下命令进行客户端上传，命令示例如下：

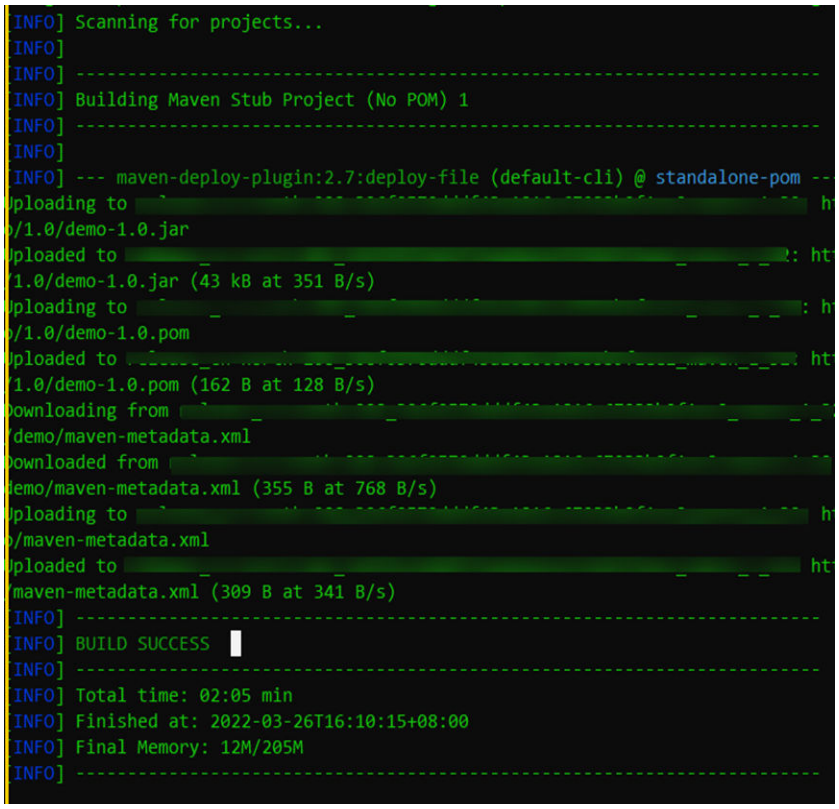
说明

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
```

```
npm notice === Tarball Details ===
npm notice name:          @test/demo
npm notice version:       1.0.0
npm notice package size:  8.7 MB
npm notice unpacked size: 10.6 MB
npm notice shasum:
npm notice integrity:
npm notice total files:   102
npm notice
+ @test/demo@1.0.0
```

客户端下载 npm 组件

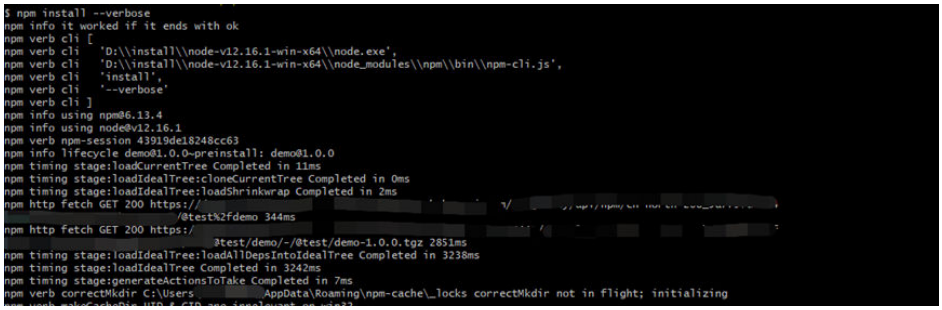
- 使用客户端工具为npm，请确保已安装node.js（或io.js）和npm。

  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。

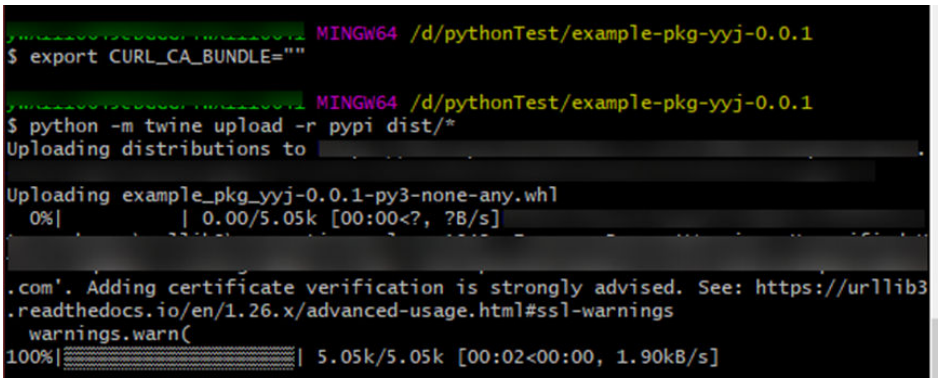
  2. 复制到用户目录下，Linux系统路径为：~/.pypirc (Windows系统路径为：C:\Users\<UserName>\.pypirc)。

  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。

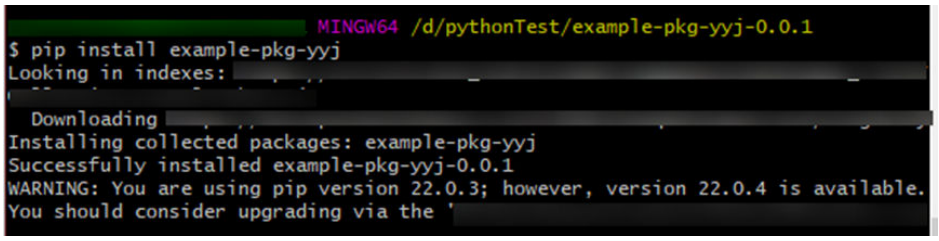
客户端下载 PyPI 组件

- 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“操作指导”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”。

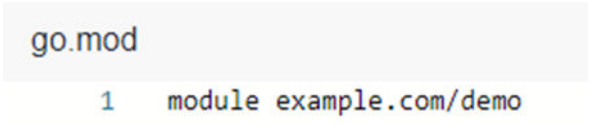
- 对于“go.mod”文件，参数“filePath”为“example.com/demo/v2/@v/v2.0.0.mod”，“localFile”为“example.com/demo/v2@v2.0.0/go.mod”。

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

```
[root@xxxxxxxxx ~]# rpm -qa yum
yum-4.2.23-3.h16.eulerosv2r10.noarch
```

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“操作指导”。

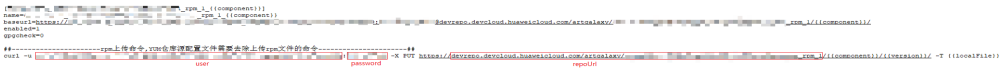
步骤3 在弹框中单击“下载配置文件”。

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

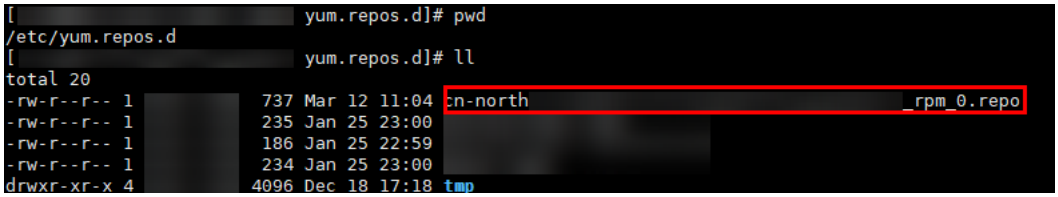
从 Rpm 私有依赖库获取依赖包

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“操作指导”，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“操作指导”，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“操作指导”，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} -username {user_name} -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 push <PATH_TO_FILE> -source <SOURCE_NAME>
```

----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“操作指导”，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“操作指导”，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“操作指导”，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

客户端上传/下载 Cocoapods 组件

前提条件：

- 已安装Ruby客户端与cocoapods-art插件。
- 私有依赖库中已创建CocoaPods仓库。

通过“下载配置文件替换”上传CocoaPods组件至私有依赖库：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“下载配置文件替换”，单击“下载配置文件”，下载配置文件“cocoapods.txt”。

步骤3 获取已下线的配置文件中的**{username}**、**{password}**。

步骤4 在本地客户端执行以下命令，将本地CocoaPods软件包上传至对应的私有依赖库。

```
curl -k -u "<USERNAME>:<PASSWORD>" -XPUT "{url}/<TARGET_FILE_PATH>/" -T <PATH_TO_FILE>/<FILE_NAME>
```

USERNAME：步骤3中获取的**{username}**。

PASSWORD：步骤3中获取的**{password}**。

url：对应CocoaPods私有依赖库的仓库地址。

TARGET_FILE_PATH：需要存放的私有依赖库文件夹名称。

PATH_TO_FILE：需要上传组件的本地路径。

FILE_NAME：设置组件上传至私有依赖库的名称。

步骤5 在CocoaPods私有依赖库中查看已上传的组件。

----结束

通过“下载配置文件替换”，下载CocoaPods私有依赖库的组件：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“下载配置文件替换”。

步骤3 在选择用途中，单击“下载”。

步骤4 执行以下命令进行本地客户端下载等相关操作：

执行如下命令下载远程私有依赖库中组件。

```
pod repo-art add {package_name} {url}
```

package_name：用户设置下载cocoapods依赖包名称。

url：对应私有依赖库的仓库地址。

执行修改命令（修改cocoapods私有依赖库地址）

```
pod repo-art update {package_name} {url}
```

package_name：cocoapods依赖包无法修改。

url：输入目标私有依赖库的仓库地址。

执行查询命令展示已下载的组件。

```
pod repo-art list
```

执行删除命令移除本地软件依赖包。

```
pod repo-art remove <repo-name>//repo_name:cocoapods依赖包名称。
```

----结束

通过“按照命令行配置”上传CocoaPods组件至私有依赖库：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“按照命令行配置”。

步骤3 执行以下命令来确认已安装Rudy客户端。

```
rudy -v
```

步骤4 执行以下安装命令来安装cocoapods-art插件。

```
sudo gem install cocoapods-art
```

步骤5 执行以下命令将私有依赖库添加至您的CocoaPods客户端中。

```
pod repo-art add <repo_name> "{url}"
```

repo_name：设置本地客户端存放私有依赖库组件的文件夹名称。

url：CocoaPods私有依赖库的仓库地址

步骤6 在选择用途中，单击“发布”。

步骤7 执行以下命令将本地客户端组件上传至对应的CocoaPods私有依赖库。

```
curl -k -u "<USERNAME>:<PASSWORD>" -XPUT "{url}/<TARGET_FILE_PATH>/" -T <PATH_TO_FILE>/<FILE_NAME>
```

USERNAME：**步骤3**中获取的**{username}**。

PASSWORD：**步骤3**中获取的**{password}**。

url：对应CocoaPods私有依赖库的仓库地址。

TARGET_FILE_PATH：需要存放的私有依赖库文件夹名称。

PATH_TO_FILE：需要上传组件的本地路径。

FILE_NAME：设置组件上传至私有依赖库的名称。

步骤8 在CocoaPods私有依赖库中查看已上传的组件。

----结束

通过“按照命令行配置”下载CocoaPods私有依赖库的组件：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“按照命令行配置”。

步骤3 执行以下命令来确认已安装Rudy客户端。

```
rudy -v
```

步骤4 执行以下安装命令来安装cocoapods-art插件。

```
sudo gem install cocoapods-art
```

步骤5 执行以下命令将私有依赖库添加至您的CocoaPods客户端中。

```
pod repo-art add <repo_name> "{url}"
```

repo_name：设置本地客户端存放私有依赖库组件的文件夹名称。

url：CocoaPods私有依赖库的仓库地址

步骤6 在选择用途中，单击“下载”。

步骤7 执行以下命令进行本地客户端下载等相关操作：

执行刷新命令下载远程私有依赖库中组件。

```
pod repo-art update {package_name}//package_name：组件名称。
```

执行查询命令展示已下载的组件。

```
pod repo-art list
```

执行删除命令移除本地软件依赖包。

```
pod repo-art remove <repo-name>//repo-name：cocoapods依赖包名称
```

----结束

4.6.1 设置私有依赖库 2.0 与本地开发环境对接

下载私有依赖库配置文件

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“操作指导”。

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

重置私有依赖库密码

用户可以下载配置文件，实现私有依赖库与本地开发环境对接，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

4.6.2 通过客户端上传私有组件至私有依赖库

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：

```
<server>
    <id>releases</id>
    <username>[redacted]</username>
    <password>[redacted]</password>
</server>
<server>
    <id>snapshots</id>
    <username>[redacted]</username>
    <password>[redacted]</password>
</server>
<server>
    <id>z_mirrors</id>
</server>
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-deploy-plugin:2.7:deploy-file (default-cli) @ standalone-pom ---
Uploading to [redacted]
/1.0/demo-1.0.jar
Uploaded to [redacted]
1.0/demo-1.0.jar (43 kB at 351 B/s)
Uploading to [redacted]
/1.0/demo-1.0.pom
Uploaded to [redacted]
1.0/demo-1.0.pom (162 B at 128 B/s)
Downloading from [redacted]
demo/maven-metadata.xml
Downloaded from [redacted]
emo/maven-metadata.xml (355 B at 768 B/s)
Uploading to [redacted]
/maven-metadata.xml
Uploaded to [redacted]
maven-metadata.xml (309 B at 341 B/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:05 min
[INFO] Finished at: 2022-03-26T16:10:15+08:00
[INFO] Final Memory: 12M/205M
[INFO] ------------------------------------------------------------------------
```

- 使用客户端工具为Gradle，请确保已安装JDK和Gradle。
  a. 从私有依赖库页面下载inti.gradle文件。

  b. 在本地项目下找到“build.gradle”文件，需要在gradle文件下添加以下命令，命令示例如下：

```
uploadArchives {
    repositories {
        mavenDeployer {repository(url:"***") {
                authentication(userName: "{repo_name}", password: "{repo_password}")
            }
            //构造项目的Pom文件
            pom.project {
                name = project.name
                packaging = 'jar'
                description = 'description'
            }
        }
    }
}
```

- url：上传文件到仓库的路径，可在对应Maven私有依赖库界面，单击获取。
- {repo_name}：从对应Maven仓库页面下载inti.gradle文件中获取username。
- {repo_password}：从对应Maven仓库页面下载inti.gradle文件中获取password。

c. 到本地项目所在目录下执行命令：

```
gradle uploadArchives
```

d. 返回对应的Maven仓库查看已上传的组件。

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
```

```
npm notice === Tarball Details ===
npm notice name:          @test/demo
npm notice version:       1.0.0
npm notice package size:  8.7 MB
npm notice unpacked size: 10.6 MB
npm notice shasum:
npm notice integrity:
npm notice total files:   102
npm notice
+ @test/demo@1.0.0
```

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。

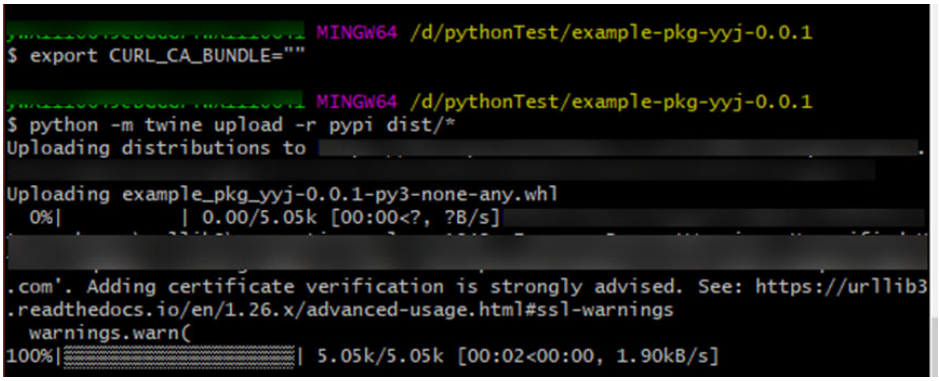
2. 复制到用户目录下，Linux系统路径为：~/.pypirc (Windows系统路径为：C:\Users\<UserName>\.pypirc)。

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。

客户端上传 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“操作指导”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。

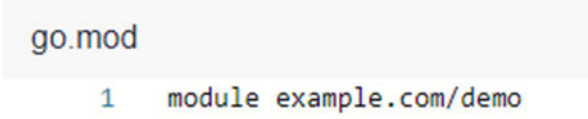
- 对于zip包，参数“filePath”为“example.com/demo/v2/@v/v2.0.0.zip”，“localFile”为“v2.0.0.zip”。
- 对于“go.mod”文件，参数“filePath”为“example.com/demo/v2/@v/v2.0.0.mod”，“localFile”为“example.com/demo/v2@v2.0.0/go.mod”。

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
```

客户端上传 RPM

使用linux系统和yum工具，请确保使用linux系统，且已安装yum。

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

```
[root@xxxxxxxx ~]# rpm -qa yum
yum-4.2.23-3.h16.eulerosv2r10.noarch
```

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“操作指导”。

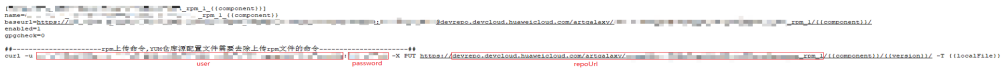
步骤3 在弹框中单击“下载配置文件”。

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

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“操作指导”，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

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“操作指导”，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} -username {user_name} -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 push <PATH_TO_FILE> -source <SOURCE_NAME>
```

----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“操作指导”，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

客户端上传 Cocoapods 组件

前提条件：

- 已安装Ruby客户端与cocoapods-art插件。
- 私有依赖库中已创建CocoaPods仓库。

通过“下载配置文件替换”上传CocoaPods组件至私有依赖库：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“下载配置文件替换”，单击“下载配置文件”，下载配置文件“cocoapods.txt”。

步骤3 获取已下线的配置文件中的**{username}**、**{password}**。

步骤4 在本地客户端执行以下命令，将本地CocoaPods软件包上传至对应的私有依赖库。

```
curl -k -u "<USERNAME>:<PASSWORD>" -XPUT "{url}/<TARGET_FILE_PATH>/" -T <PATH_TO_FILE>/<FILE_NAME>
```

USERNAME：**步骤3**中获取的**{username}**。

PASSWORD：**步骤3**中获取的**{password}**。

url：对应CocoaPods私有依赖库的仓库地址。

TARGET_FILE_PATH：需要存放的私有依赖库文件夹名称。

PATH_TO_FILE：需要上传组件的本地路径。

FILE_NAME：设置组件上传至私有依赖库的名称。

步骤5 在CocoaPods私有依赖库中查看已上传的组件。

----结束

通过“按照命令行配置”上传CocoaPods组件至私有依赖库：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“按照命令行配置”。

步骤3 执行以下命令来确认已安装Rudy客户端。

```
rudy -v
```

步骤4 执行以下安装命令来安装cocoapods-art插件。

```
sudo gem install cocoapods-art
```

步骤5 执行以下命令将私有依赖库添加至您的CocoaPods客户端中。

```
pod repo-art add <repo_name> "{url}"
```

repo_name：设置本地客户端存放私有依赖库组件的文件夹名称。

url：CocoaPods私有依赖库的仓库地址

步骤6 在选择用途中，单击“发布”。

步骤7 执行以下命令将本地客户端组件上传至对应的CocoaPods私有依赖库。

```
curl -k -u "<USERNAME>:<PASSWORD>" -XPUT "{url}/<TARGET_FILE_PATH>/" -T <PATH_TO_FILE>/<FILE_NAME>
```

USERNAME：**步骤3**中获取的**{username}**。

PASSWORD：**步骤3**中获取的**{password}**。

url：对应CocoaPods私有依赖库的仓库地址。

TARGET_FILE_PATH：需要存放的私有依赖库文件夹名称。

PATH_TO_FILE：需要上传组件的本地路径。

FILE_NAME：设置组件上传至私有依赖库的名称。

步骤8 在CocoaPods私有依赖库中查看已上传的组件。

----结束

4.6.3 通过客户端从私有依赖库下载私有组件

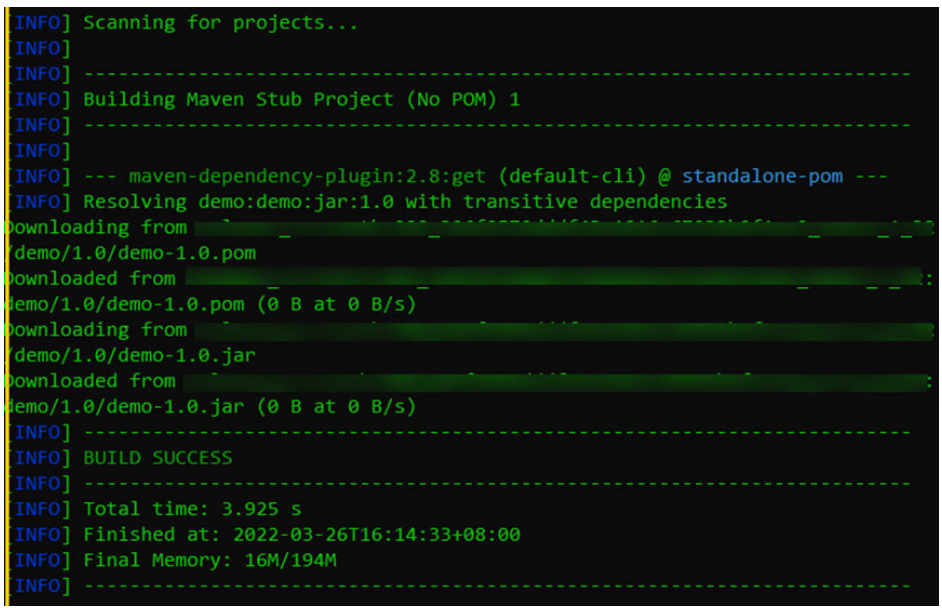
客户端下载 Maven 组件

- 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
```



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
```

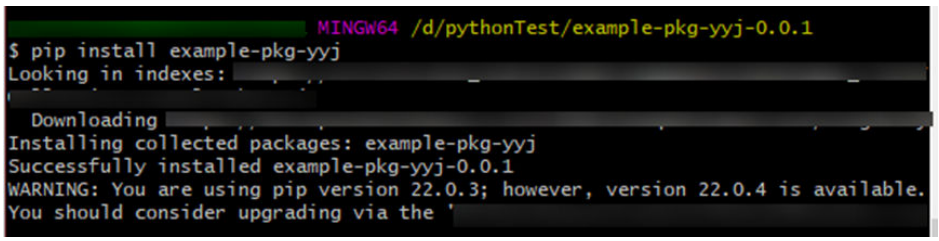
客户端下载 PyPI 组件

- 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
```

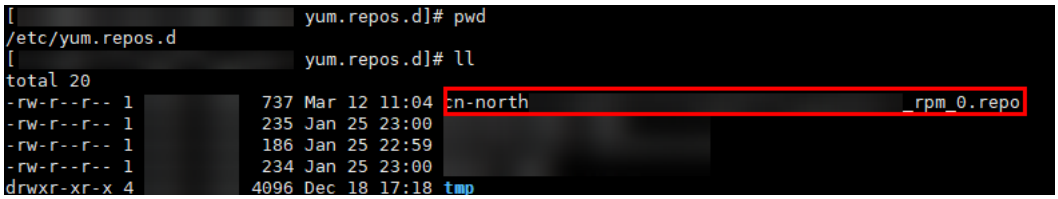

客户端下载 RPM

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**2**、**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“操作指导”，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“操作指导”，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“操作指导”，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

客户端下载 Docker 组件

前提条件:

- 已安装Docker客户端。
- 私有依赖库中已创建Docker仓库。
- 在Docker客户端需要执行如下命令修改配置，忽略证书。

  ```
  vi /etc/docker/daemon.json
  {
    "insecure-registries": ["url"]
  }
  ```

  {url}：仓库地址，如下图所示。

客户端下载Docker组件：

在本地客户端执行以下命令，下载Docker组件。

```
sudo docker pull {url}/${image_name}:${image_version}
```

url：仓库地址。

image_name：组件名称。

image_version：组件版本。

客户端下载 Cocoapods 组件

前提条件：

- 已安装Ruby客户端与cocoapods-art插件。
- 私有依赖库中已创建CocoaPods仓库。

通过“下载配置文件替换”，下载CocoaPods私有依赖库的组件：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“下载配置文件替换”。

步骤3 在选择用途中，单击“下载”。

步骤4 执行以下命令进行本地客户端下载等相关操作：

执行如下命令下载远程私有依赖库中组件。

```
pod repo-art add {package_name} {url}
```

package_name：用户设置下载cocoapods依赖包名称。

url：对应私有依赖库的仓库地址。

执行修改命令（修改cocoapods私有依赖库地址）

```
pod repo-art update {package_name} {url}
```

package_name：cocoapods依赖包无法修改。

url：输入目标私有依赖库的仓库地址。

执行查询命令展示已下载的组件。

```
pod repo-art list
```

执行删除命令移除本地软件依赖包。


```
pod repo-art remove <repo-name>//repo_name:cocoapods依赖包名称。
```

----结束

通过“按照命令行配置”下载CocoaPods私有依赖库的组件：

步骤1 从私有库页面选择对应的CocoaPods仓库，单击“操作指导”。

步骤2 选择“按照命令行配置”。

步骤3 执行以下命令来确认已安装Rudy客户端。

```
rudy -v
```

步骤4 执行以下安装命令来安装cocoapods-art插件。

```
sudo gem install cocoapods-art
```

步骤5 执行以下命令将私有依赖库添加至您的CocoaPods客户端中。

```
pod repo-art add <repo_name> "{url}"
```

repo_name：设置本地客户端存放私有依赖库组件的文件夹名称。

url：CocoaPods私有依赖库的仓库地址

步骤6 在选择用途中，单击“下载”。

步骤7 执行以下命令进行本地客户端下载等相关操作：

执行刷新命令下载远程私有依赖库中组件。

```
pod repo-art update {package_name}//package_name：组件名称。
```

执行查询命令展示已下载的组件。

```
pod repo-art list
```

执行删除命令移除本地软件依赖包。

```
pod repo-art remove <repo-name>//repo-name：cocoapods依赖包名称
```

----结束

4.7 管理私有依赖库 2.0 中的组件

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（默认为所有制品类型）。

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

4.7.1 通过私有依赖库查看私有组件

通过仓库视图查看私有组件

用户进入私有依赖库后，页面默认展示私有依赖库的仓库视图，上传成功的组件将保存在仓库视图中对应的文件夹下。

步骤1 进入私有依赖库，单击仓库及文件夹前的图标，找到私有组件。

步骤2 单击组件名称，页面将显示所在私有依赖库的仓库详细信息以及组件的校验和信息。

单击校验和信息的，可以复制该信息，在搜索栏内根据粘贴的校验和信息找到目标组件。

----结束

通过版本视图查看私有组件

私有依赖库支持将不同类型私有组件按照版本维度进行归类展示。在版本视图的列表中，提供按制品包名称和版本号进行过滤展示，提供按照更新时间对文件进行排序。

步骤1 进入私有依赖库页面。

步骤2 在页面左上方选择“版本视图”页签，单击左侧列表中的仓库名称，页面展示该类型仓库下的软件包版本列表。为不同类型私有组件设置版本请参考**通过私有依赖库页面上传下载私有组件**。

步骤3 不同版本的同名软件包放在一个文件下。单击“包名”，页面将显示该软件包最新版本的概览信息。

步骤4 单击“版本数”，页面将展示对应软件包的版本列表。

单击“版本号”，页面将显示该软件包的概览信息和文件列表。在文件列表中，单击“文件名称”，页面将跳转到软件包所在的存储位置。

----结束

4.7.2 通过私有依赖库编辑私有组件

4.8 管理私有依赖库 2.0 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

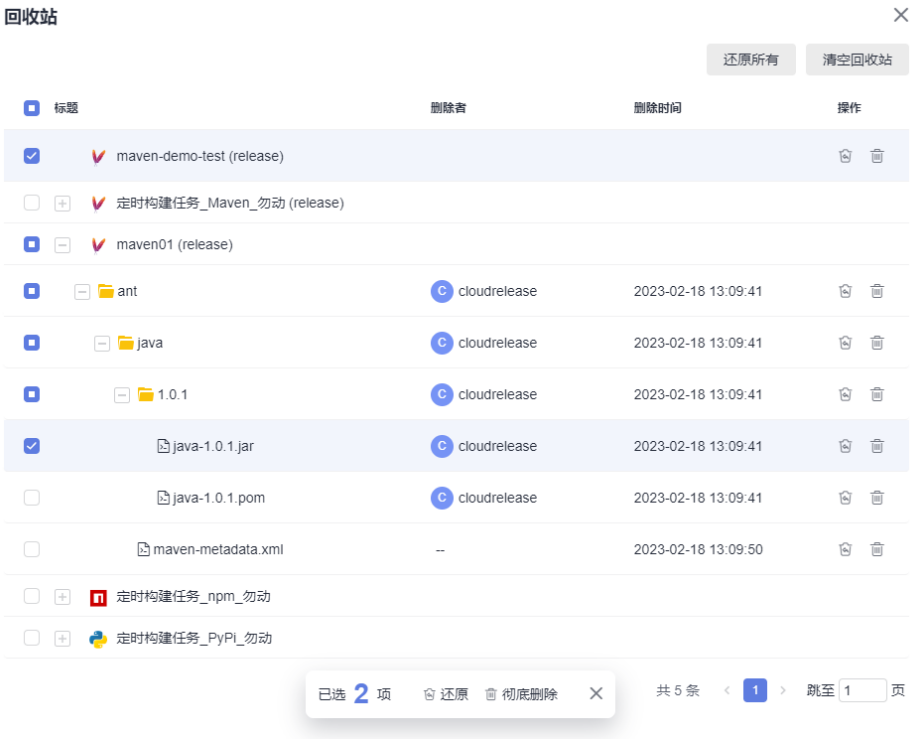
用户可以在制品仓库服务的首页回收站里，处理所有项目删除的组件。

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。
	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。

操作类型	操作项	说明
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 单击项目卡片进入项目，单击菜单栏“制品仓库 > 私有依赖库”，单击目标仓库名称。

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。

操作类型	操作项	说明
	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

5 扫描软件包/私有组件

5.1 对软件发布库中的软件包进行安全扫描

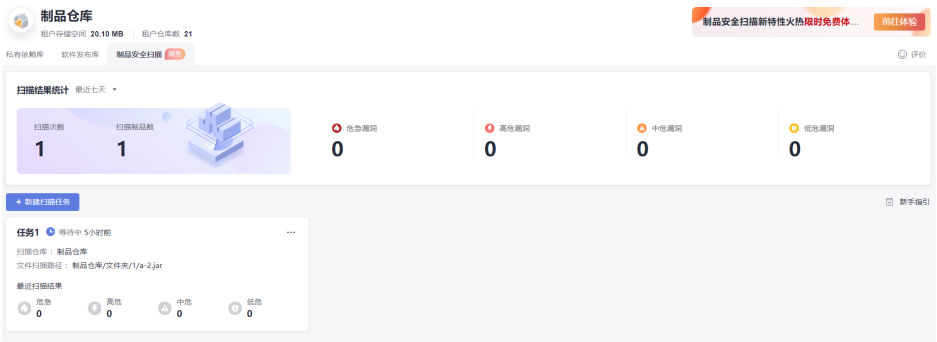
制品安全扫描支持开源合规和漏洞检测，无需上传源码检测，整个检测为分四个级别，误报低，针对新风险响应迅速。

支持按照仓库、选择制品创建扫描方案，可以执行、暂停任务。

步骤1 登录软件开发流水线首页。

步骤2 在功能菜单区单击“服务 > 制品仓库”，选择“制品安全扫描”页签。

页面上方展示了扫描次数、扫描制品数、分四个安全漏洞等级（危急漏洞、高危漏洞、中危漏洞、低危漏洞）及漏洞数量。



用户可以单击扫描结果统计右侧的下拉栏，设置时间，页面将显示该时间范围内的统计结果。

步骤3 单击“新建扫描任务”，在弹框中配置以下扫描信息。

配置项	是否必填	描述
仓库类型	是	选择需要扫描的制品所在的仓库类型。
制品	是	单击并在搜索栏中输入关键字，选择需要扫描的制品。
任务名称	是	为创建的扫描任务命名。

步骤4 配置完成后，单击“扫描”，完成漏洞扫描的创建，生成的任务显示在页面中。

生产的任务卡片中展示了“扫描仓库”、“文件扫描路径”和“最近扫描结果”。扫描结果显示四个漏洞等级（危急漏洞、高危漏洞、中危漏洞、低危漏洞）和数量。

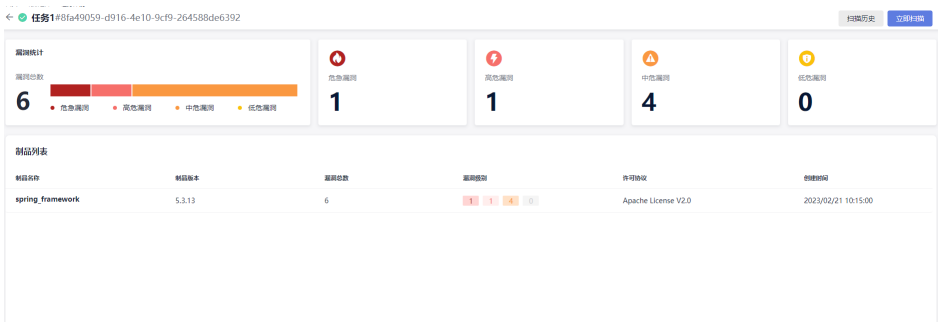
- 单击◎可以停止扫描中的任务。

- 单击任务卡片···，单击“删除任务”可以将当前任务删除。

- 单击任务卡片…，单击“执行任务”可以重新执行当前任务。

步骤5 单击任务卡片，页面展示扫描任务详情。

页面上方展示四个漏洞等级（危急漏洞、高危漏洞、中危漏洞、低危漏洞）和漏洞数量。



- 单击右上方“扫描历史”，用户可以设置时间筛选对应时间范围内的扫描历史。
- 单击右上方“立即扫描”可以重新启动该扫描任务。

步骤6 在制品列表中单击“制品名称”，页面右侧将展示该扫描任务下的制品详情。

单击漏洞列表中的“CVE”，页面右侧将展示该条漏洞的详情。

----结束

5.2 对私有依赖库中的私有组件进行安全扫描

步骤1 登录软件开发流水线首页。

步骤2 在功能菜单区单击“服务 > 制品仓库”，选择“制品安全扫描”页签。

步骤3 单击“新建扫描任务”，在弹框中配置以下扫描信息。

配置项	是否必填	描述
仓库类型	是	选择需要扫描的制品所在的仓库类型，发布库或私有库。
制品	是	在下拉栏中选择制品类型，输入组件名的关键字，找到相应的组件。
任务名称	是	为创建的扫描任务命名。

步骤4 配置完成后，单击“扫描”，完成漏洞扫描的创建，生成的任务显示在页面中。

步骤5 单击任务卡片即可查看任务详情。

步骤6 后续安全扫描的基础操作可以参考**制品安全扫描**。

----结束

6 管理软件发布库 1.0

6.1 访问软件发布库 1.0

制品仓库服务（CodeArts Artifact）于2023年3月进行服务升级，老用户在该时间前的存量软件发布库及软件发布库内的资源存放在旧版软件发布库中。

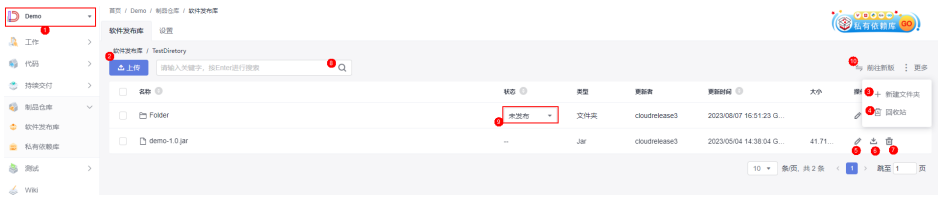
对于有存量仓库和资源的老用户，您还可以继续在旧版软件发布库中上传、管理软件包，创建项目后，会在新版软件发布库下自动创建同项目名称的软件发布库。

进入 CodeArts Artifact 的软件发布库 1.0

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”，进入新版软件发布库页面。

步骤2 单击页面左下方“返回旧版”进入旧版软件发布库。

步骤3 页面中展示了当前项目下的存量文件列表，根据需要可完成以下操作。



序号	操作项	说明
1	切换项目	单击项目名称的下拉列表，可以切换项目查看旧版的软件发布库。
2	上传	单击列表左上方“上传”，可以手动上传本地软件包到软件发布库。 单次可上传单文件的最大值为2GB。 **说明** 建议不要将带有明文账号密码等敏感信息的文件上传至软件发布库。

序号	操作项	说明
3	新建文件夹	单击列表右上方“更多 > 新建文件夹”，可以在当前页面下新建文件夹。
4	回收站	单击列表右上方“更多 > 回收站”，可以进入软件发布库回收站，对删除后的软件包/文件夹进行管理。
5	编辑	单击操作列，可编辑对应行软件包/文件夹信息： ● 软件包：可修改软件包的名称和发布版本（由编译构建归档的软件包发布版本默认为构建序号）。 ● 文件夹：可修改文件夹名称。
6	下载	单击操作列，可以下载对应行软件包。
7	删除	● 单击操作列，弹出对话框。单击“放入回收站”可以将对应行的软件包/文件夹删除至回收站；单击“彻底删除”可以将对应行的软件包/文件夹彻底删除。 ● 勾选多个软件包/文件夹，在列表下方单击“放入回收站”可以同时将多个软件包/文件夹删除至回收站；单击“彻底删除”可以同时将多个软件包/文件夹彻底删除。 **说明** 在回收站中的软件包仍然占用软件发布库的使用容量，当软件包从回收站彻底删除时释放所占用的容量。
8	搜索	在列表左上方的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。
9	修改状态	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，单击“状态”列中的下拉列表，可以修改对应行文件夹的状态。 状态分为“未发布”和“已发布”两种。文件夹的状态可由未发布变为已发布，状态转换不可逆；如果文件夹的状态为“已发布”，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。
10	前往新版	单击“前往新版”，可以返回新版的软件发布库。

----结束

6.2 管理软件发布库 1.0 中的软件包

上传软件包到软件发布库 1.0

步骤1 访问软件发布库1.0。

步骤2 单击“上传”。

步骤3 在弹框中配置如下信息后，单击“上传”。

- 目标仓库：当前软件发布库。
- 版本：用户可以为软件包设置版本号。
- 上传方式：选择“单个文件上传”或“多个文件上传”，本章节默认“单个文件上传”。
- 路径：用户设置路径名称后，仓库视图中会创建改名称的文件夹，上传的软件包会存放在该文件夹内。
- 文件：从本地选择需要上传到软件发布库的软件包。

步骤4 上传成功的软件包将保存在软件发布库的文件列表中。

单击软件包所在操作列中的，可以修改软件包名称。

单击软件包所在操作列中的，可以将软件包下载到本地。

单击软件包所在操作列中的，可以删除软件包。

----结束

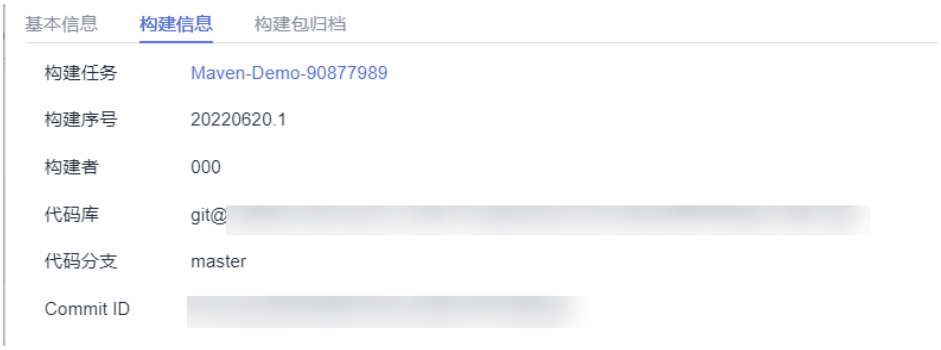
在软件发布库 1.0 中查看/编辑软件包

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：基本信息、构建信息、构建包归档信息。

进入软件发布库，单击软件包名称，页面右侧滑出对话框展示所选软件包详情。通过三个tab页签“基本信息”、“构建信息”、“构建包归档”展示软件包的详情。

- 基本信息：展示软件包名称、发布版本、大小、归档路径、部署下载地址、创建者、创建时间、校验和等信息。

  单击右上角，可以修改软件包的名称和发布版本（由编译构建归档的软件包发布版本默认为构建序号）。
- 构建信息：展示生成软件包的构建任务、构建序号、构建者、代码库、代码分支和Commit ID。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。

6.3 设置软件发布库 1.0 的清理策略

软件发布库提供定时自动清理文件功能。可根据设置文件的保留时长，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 单击项目卡片进入项目，单击菜单栏“制品仓库 > 软件发布库”，进入新版软件发布库页面。

步骤2 单击页面左下方“返回旧版”进入旧版软件发布库。

步骤3 选择“设置”页签。

步骤4 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：15天。
- 从回收站彻底删除：15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。
  - 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)。

----结束

6.4 管理软件发布库 1.0 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。

步骤1 进入旧版软件发布库。

步骤2 单击“更多 > 回收站”，进入“回收站”页面。

步骤3 根据需要对软件包/文件夹进行删除或还原操作。

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“更多操作 > 还原所有”，可以一键还原回收站所有软件包/文件夹。
4	清空回收站	单击“更多操作 > 清空回收站”，可以一键删除回收站所有软件包/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	单个删除	单击操作列，可以删除对应软件包/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

7 管理私有依赖库 1.0

7.1 访问私有依赖库 1.0

制品仓库服务（CodeArts Artifact）于2023年3月进行服务升级，老用户在该时间前的存量私有依赖库没有绑定项目，私有依赖库及私有依赖库内的资源存放在旧版私有依赖库中。

私有依赖库2.0支持的功能，也可以在私有依赖库1.0中实现。

进入私有依赖库 1.0

步骤1 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”，进入私有依赖库，。

步骤3 单击页面左下方“返回旧版”进入旧版私有依赖库，用户已创建的存量旧版私有依赖库将显示在仓库视图中。

----结束

7.2 配置私有依赖库 1.0

新增的成员需赋予指定的角色，才可以正常的使用制品仓库服务，请参考**添加成员并授权成员角色**。

在私有依赖库中管理仓库权限

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：

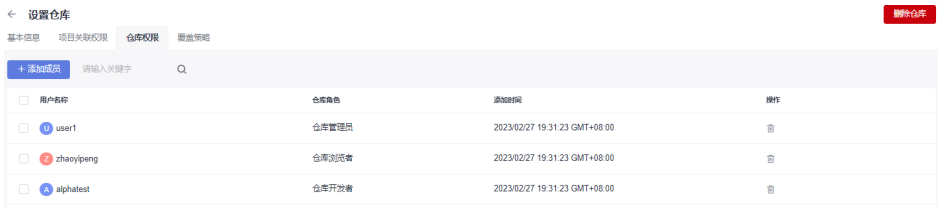
- 项目创建者、项目经理对应仓库管理员。
- 开发人员、测试经理、测试人员、运维经理对应仓库开发者。
- 参与者、浏览者、自定义角色对应仓库浏览者。

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。

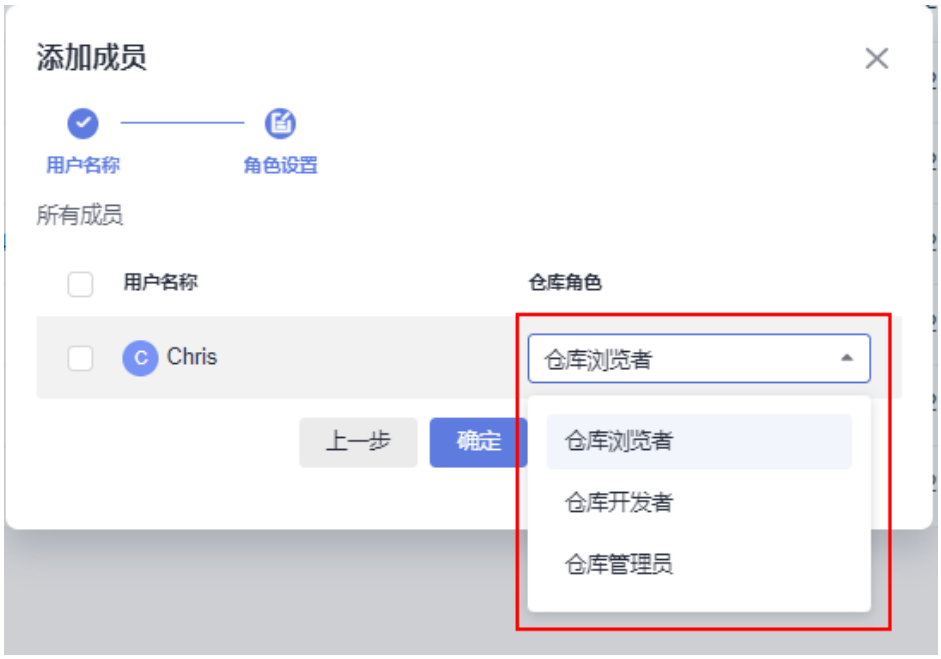


步骤4 添加成员。

单击页面左上方“添加成员”，在弹框中勾选成员，单击“下一步”。

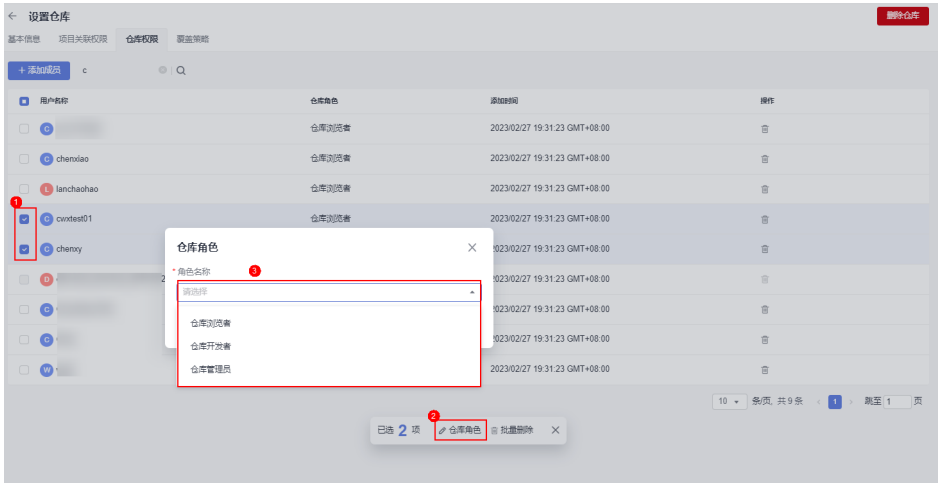
步骤5 为成员分配仓库角色。

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：

表 7-1 私有依赖库角色权限表

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	√	√	√	√	×	×

租户账号及仓库管理员批量管理用户权限

租户账号可以向私有依赖库中添加/删除成员。各仓库的管理员可以管理对应仓库中成员的角色。

步骤1 单击页面右上角用户名，在下拉菜单中选择“租户设置”。

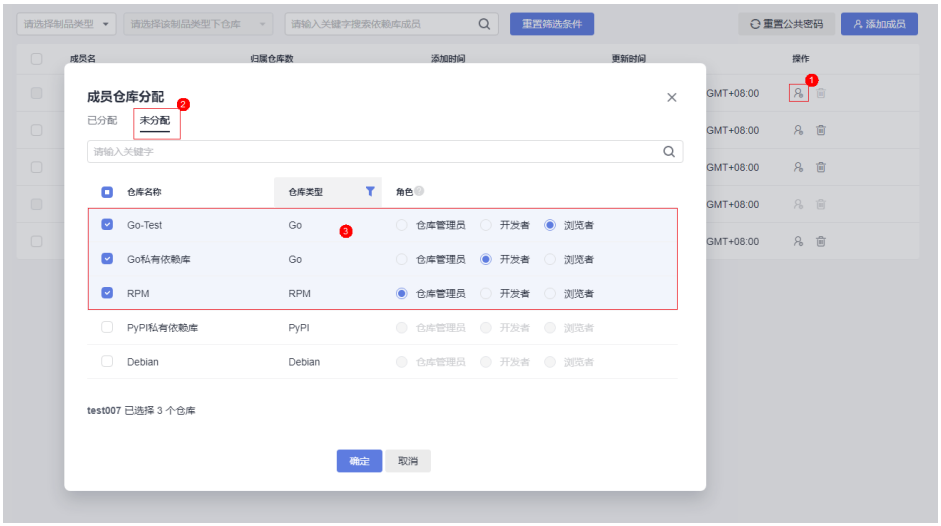
步骤2 单击导航“制品仓库 > 独立用户权限”。

步骤3 单击“添加成员”，在弹框中勾选成员，单击“确定”。

步骤4 为成员分配仓库角色。

1. 在列表中找到待分配角色的成员，单击操作列中。
2. 在弹框中选择“未分配”页签。
3. 根据需要勾选仓库，并选择对应角色，单击“确定”。

图 7-1 成员仓库分配



----结束

在“独立用户权限”页面中还可完成以下操作。

操作	说明
删除成员	单击，可删除对应行的成员；勾选多个成员，单击“批量删除”，可以删除全部所选成员。
修改成员权限	单击，在弹框中勾选仓库名称，单击“确定”。
查看仓库成员	依次单击页面左上角两个下拉列表，选择制品类型和仓库名称，页面中自动显示该仓库的成员列表。 单击页面上方“重置筛选条件”，可查看全部成员列表。
移出成员	选择一个仓库，在成员里列表中单击，可将对应行的成员从该仓库中移出。 **说明** 移出成员是将该成员从某个仓库中移出，不影响该成员在其它仓库中的角色与权限。 删除成员是将该成员从所有仓库中移出，对应的仓库权限也会相应被删除。
搜索仓库成员	在页面上方搜索框中输入成员名或关键字，单击，即可查找仓库成员。
重置公共密码（该功能只有租户管理员可见）	公共密码用于编译构建服务向私有依赖库上传下载组件，页面不可见。单击页面右上方“重置公共密码”即可重置该密码。

7.3 管理私有依赖库 1.0

7.3.1 查看私有依赖库基本信息并配置仓库路径

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、版本策略不能修改。

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添加路径。

单击 可以删除路径。

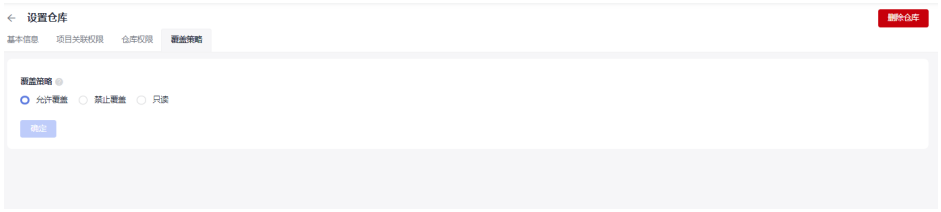
----结束

7.3.2 配置私有依赖库覆盖策略

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，系统将自动保存。

----结束

7.3.3 配置 CodeArts Artifact 中的 Maven 仓库的清理策略

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。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。

步骤1 单击项目卡片进入项目，单击菜单栏“制品仓库 > 私有依赖库”，进入私有依赖库。

步骤2 在左侧仓库列表中选择对应的“Snapshot”类型的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

7.3.4 关联 CodeArts Artifact 中的 Maven 仓库与项目

在私有依赖库中，将Maven格式仓库与多个项目关联后，项目中的构建任务即可完成在构建步骤中选择该仓库，将构建出来的产物存放到Maven仓库中。

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

在编译构建服务中，需要将构建产物上传至私有依赖库的配置方法，请参见**发布依赖包到CodeArts私有依赖库**。

7.4 管理私有依赖库 1.0 中的私有组件

7.4.1 通过私有依赖库上传私有组件

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

上传组件的操作步骤如下。

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 在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。

说明

不建议用户将带有明文账号密码等敏感信息的文件上传至私有依赖库。

----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。 – 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。

参数	说明
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 全名 RedHat Package Manager，是由Red Hat公司提出，被众多Linux发行版本所采用，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“上传制品”。

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。

参数	参数说明
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

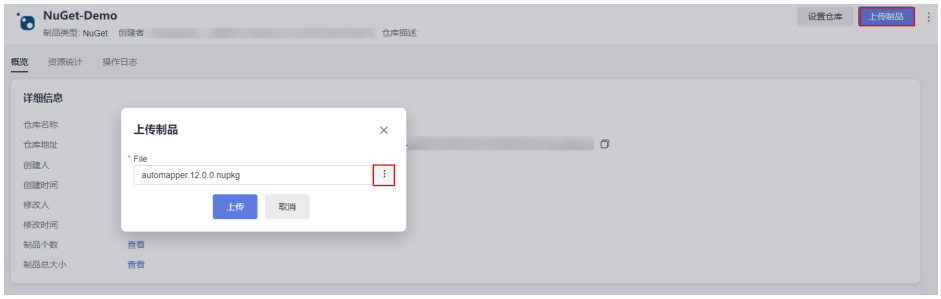
NuGet 包是具有 .nupkg 扩展的单个 ZIP 文件，用户可以使用 NuGet 包来共享组织或工作组专用的代码。

制品仓库服务支持将本地的NuGet包上传至私有依赖库。

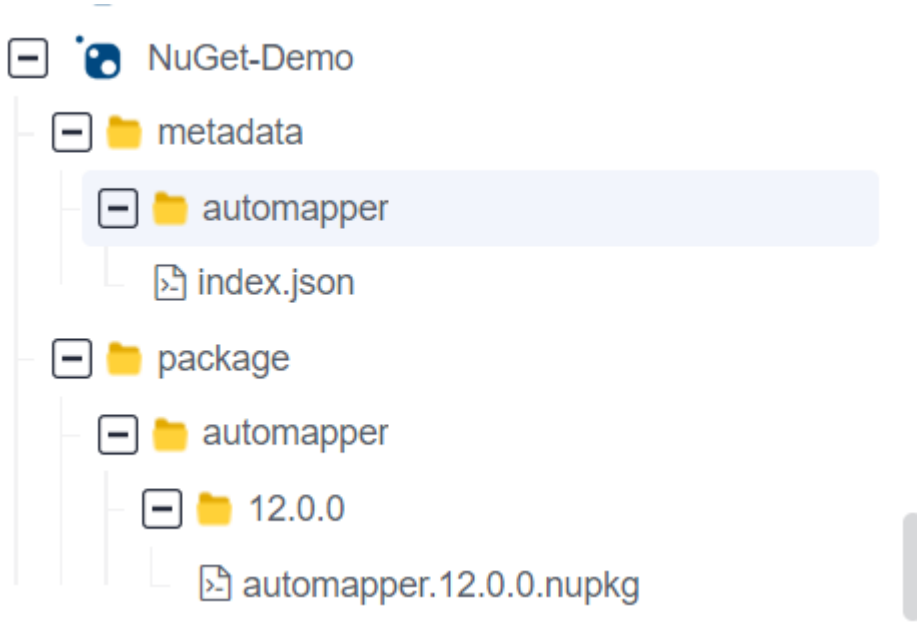
- 一般建议使用以下规范打包命名NuGet本地文件。
  软件名称-软件的主版本号.nupkg
  例如：automapper.12.0.0.nupkg

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

7.4.2 通过私有依赖库查看私有组件

通过仓库视图查看私有组件

用户进入私有依赖库后，页面默认展示私有依赖库的仓库视图，上传成功的组件将保存在仓库视图中对应的文件夹下。

步骤1 进入私有依赖库，单击仓库及文件夹前的图标，找到私有组件。

步骤2 单击组件名称，页面将显示所在私有依赖库的仓库详细信息以及组件的校验和信息。

单击校验和信息的，可以复制该信息，在搜索栏内根据粘贴的校验和信息找到目标组件。

----结束

通过版本视图查看私有组件

私有依赖库支持将不同类型私有组件按照版本维度进行归类展示。在版本视图的列表中，提供按制品包名称和版本号进行过滤展示，提供按照更新时间对文件进行排序。

步骤1 进入私有依赖库页面。

步骤2 在页面左上方选择“版本视图”页签，单击左侧列表中的仓库名称，页面展示该类型仓库下的软件包版本列表。为不同类型私有组件设置版本请参考**通过私有依赖库页面上传下载私有组件**。

步骤3 不同版本的同名软件包放在一个文件下。单击“包名”，页面将显示该软件包最新版本的概览信息。

步骤4 单击“版本数”，页面将展示对应软件包的版本列表。

单击“版本号”，页面将显示该软件包的概览信息和文件列表。在文件列表中，单击“文件名称”，页面将跳转到软件包所在的存储位置。

----结束

7.4.3 通过私有依赖库编辑私有组件

组件管理主要包括下载、搜索、删除私有组件、关注/取消关注等操作。

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”。页面将显示“高级搜索”页面。

步骤2 页面上方可以选择待查找组件的类型（默认为所有制品类型）。

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**找到私有组件。

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删除的私有组件，单击组件名称。

若仓库或组件过多，可以通过**搜索私有组件**找到私有组件。

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关注的私有组件，单击组件名称。

若仓库或组件过多，可以通过**搜索私有组件**找到私有组件。

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

7.5 管理私有依赖库 1.0 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。

步骤1 进入私有依赖库。

步骤2 单击“回收站”，页面右侧滑出“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。

须知

回收站的所有删除操作都将彻底删除仓库与组件，无法重新找回，请慎重操作。

----结束

7.6 设置私有依赖库 1.0 与本地开发环境对接

下载私有依赖库配置文件

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“操作指导”。

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

重置私有依赖库密码

用户可以下载配置文件，实现私有依赖库与本地开发环境对接，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

私有依赖库1.0也支持通过客户端上传、下载私有组件，操作步骤同私有依赖库2.0，请参考**通过客户端上传私有组件至私有依赖库**、**通过客户端从私有依赖库下载私有组件**。

8 租户级 IP 白名单

IP白名单是通过设置IP白名单的IP范围和访问权限，限制用户的访问和上传下载权限，大大增强仓库的安全性。

用户可以通过创建“租户级IP白名单”设置私有依赖库的访问权限。

步骤1 登录软件开发生产线首页，单击页面右上角用户名，在下拉菜单中选择“租户设置”。

步骤2 在页面左侧导航栏选择“制品仓库 > 租户级IP白名单”。

步骤3 单击页面右上方“新增IP白名单”。

步骤4 在弹框中选择“指定IP”或“CIDR”，输入IP地址。

表 8-1 IP 白名单格式

格式	说明
指定IP	这是最简单的一种IP白名单格式，如将您的个人家庭电脑的IP添加到白名单中，比如：100.*.*.123。
CIDR格式（无类别域间路由）	• 当您的服务器在一个局域网内并使用CIDR路由时，您可以指定局域网的32位出口IP以及一个指定网络前缀的位数。 • 从同一个IP发起的请求，只要网络前缀同您设定的前缀部分相同，即可视为来自同一授信范围从而被接受。

步骤5 勾选“我已阅读并同意《隐私政策说明》与《CodeArts服务使用声明》”，单击“确定”。

----结束