制品仓库

最佳实践

文档版本	01
发布日期	2025-01-24

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 CodeArts Artifact 最佳实践汇总

本文汇总了基于制品仓库（CodeArts Artifact）常见应用场景的操作实践，为每个实践提供详细的方案描述和操作指导，帮助用户深入了解CodeArts Artifact的各个功能。

表 1-1 CodeArts Artifact 最佳实践一览表

最佳实践	说明
通过编译构建任务发布Maven组件并按照版本归档至私有依赖库	相对于开发过程中的“源代码”，制品仓库服务关注和管理开发产生的待部署的软件包。软件包通常是由源码编译构建或打包而成，其中涉及生命周期的元数据（如名称、大小等基本属性、代码库地址、代码分支信息、构建任务、构建者、构建时间）。在开发过程中，软件包会根据不同版本不断生成改进。 软件包及其属性的管理是发布过程管理的基础，也是软件开发过程中的重要资产，而能够及时查看软件包的版本记录也成为开发者面临的诉求。本实践介绍如何通过编译构建任务发布Maven组件并按照版本归档至私有依赖库。
通过编译构建任务发布/获取NPM私有组件	私有依赖库管理各种开发语言对应的私有组件包（开发者通俗称之为私服）。由于不同的开发语言组件通常有不同的归档格式要求，私有依赖库目的就在于管理私有开发语言组件并在企业或团队内共享给其他开发者开发使用。 本实践介绍如何通过编译构建任务发布私有组件到NPM私有依赖库、如何从NPM私有依赖库获取依赖包完成编译构建任务。
通过编译构建任务发布/获取Go私有组件	本实践介绍如何通过编译构建任务发布私有组件到Go私有依赖库、如何从Go私有依赖库获取依赖包完成编译构建任务。
通过编译构建任务发布/获取PyPI私有组件	本实践介绍如何通过编译构建任务发布私有组件到PyPI私有依赖库、如何从PyPI私有依赖库获取依赖包完成编译构建任务。
通过Linux命令行上传/获取RPM私有组件	本实践介绍如何通过Linux命令行上传私有组件到RPM私有依赖库、如何从RPM私有依赖库获取依赖包。

最佳实践	说明
通过Linux命令行上传/获取Debian私有组件	本实践介绍如何通过Linux命令行上传私有组件到Debian私有依赖库、如何从Debian私有依赖库获取依赖包
批量迁移Maven/NPM/PyPI组件至私有依赖库	制品仓库服务的私有依赖库支持通过页面手动上传下载私有组件，还支持与本地开发环境对接，通过本地开发环境上传下载私有组件。如果待上传的包太多，单个上传会很繁琐。通过私有依赖库提供的批量迁移工具可以提高上传效率，在易用性上能够更加便捷地从Nexus或其他类型仓库批量上传组件至私有依赖库。
批量迁移JFrog仓库至私有依赖库	JFrog仓库是一个用于存储和管理软件包的中央存储库，提供了一种集中式的方式来管理软件包，支持各种软件包管理工具，如Maven、Gradle、NPM、NuGet等。CodeArts Artifact的私有依赖库提供了批量迁移工具，支持将JFrog仓库迁移至私有依赖库。本实践介绍如何批量迁移JFrog仓库至私有依赖库。

2 通过编译构建任务发布 Maven 组件并按照版本归档至私有依赖库

背景信息

相对于开发过程中的“源代码”，制品仓库服务关注和管理开发产生的待部署的软件包。软件包通常是由源码编译构建或打包而成，其中涉及生命周期的元数据（如名称、大小等基本属性、代码库地址、代码分支信息、构建任务、构建者、构建时间）。在开发过程中，软件包会根据不同版本不断生成改进。

软件包及其属性的管理是发布过程管理的基础，也是软件开发过程中的重要资产，而能够及时查看软件包的版本记录也成为开发者面临的诉求。

准备工作

- 已有可用项目。如果没有项目，请先**新建CodeArts项目**。
- 已添加当前账号对当前私有库的权限，请参考**配置私有依赖库权限**。

创建 Maven 类型私有依赖库并关联项目

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 > 私有依赖库”。

步骤3 单击 ＋ 新建，选择“本地仓库”，输入仓库名称，选择“Maven制品类型”。

步骤4 单击“确定”。新建成功的Maven私有依赖库将显示在仓库视图中。

步骤5 在仓库视图中，单击对应的仓库名称，单击“设置仓库”。

步骤6 选择“项目关联权限”页签，单击对应项目名所在操作列的图标，在弹框中勾选目标私有依赖库的名称。

步骤7 单击“确定”。

----结束

在代码仓库中设置组件的版本

步骤1 登录软件开发生产线，进入已创建的项目。

步骤2 单击顶部菜单“服务 > 代码托管”，进入代码托管服务。

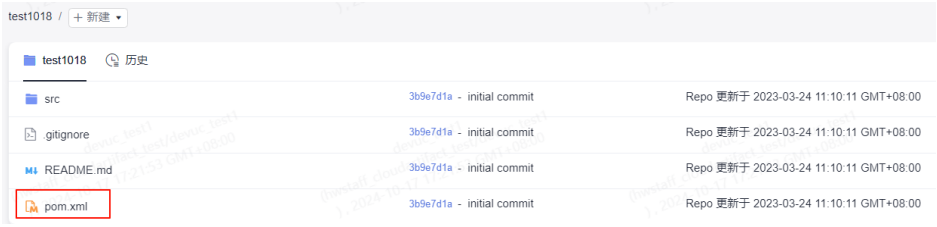
步骤3 单击“新建仓库”。

步骤4 选择“归属项目”，单击“模板仓库”，单击“下一步”。

步骤5 搜索并勾选“Java Maven Demo”类型模板，单击“下一步”。

步骤6 输入“代码仓库名称”，单击“确定”。

步骤7 进入代码仓库，单击“pom.xml”，查看组件配置。



步骤8 在组件配置页面中，<version>代码行中为当前组件的版本号（默认为1.0）。

单击页面右上方，可以修改版本号，修改完成后单击“确定”。

----结束

通过编译构建发布 Maven 私有组件到私有依赖库

步骤1 在代码仓库完成设置组件版本后，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

步骤2 在页面中选择“空白构建模板”，单击“下一步”。

步骤3 单击“点击添加构建步骤”。搜索并添加步骤“Maven构建”。

步骤4 编辑步骤“Maven构建”。

- 工具版本按照实际选择，本文中选择“maven3.5.3-jdk8-open”。
- 找到以下命令行，删除命令行前的#。
  ```
  #mvn deploy -Dmaven.test.skip=true -U -e -X -B
  ```
  找到以下命令行，在命令行前添加#。
  ```
  mvn package -Dmaven.test.skip=true -U -e -X -B
  ```
- 在“发布依赖包到CodeArts私有依赖库”一栏勾选“配置所有pom”，并在下拉列表中选择与已项目关联的Maven私有依赖库。

步骤5 单击“新建并执行”，执行构建任务。

----结束

在 Maven 私有依赖库的版本视图中查看归档的组件

步骤1 进入目标私有依赖库，找到通过构建任务上传的Maven私有组件。

参考上述步骤，在代码仓库中**设置组件版本**，可将多个版本组件归档至私有依赖库。

步骤2 单击“版本视图”。

在包列表中，可以查看从编译构建中获取软件包的版本数和最新版本。

步骤3 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 选择“文件列表”页签，在列表中可以单击目标组件操作列中的，可将组件下载到本地。

步骤5 用户在本地对组件修改并设置新的版本号后，在目标私有依赖库中，单击“上传组件”，可将最新版本的组件上传至私有依赖库。

版本视图中的包列表显示对应组件最新上传的版本并统计版本归档过的数量。

----结束

3 通过编译构建任务发布/获取 NPM 私有组件

本文档介绍如何通过编译构建任务发布私有组件到NPM私有依赖库、如何从NPM私有依赖库获取依赖包完成编译构建任务。

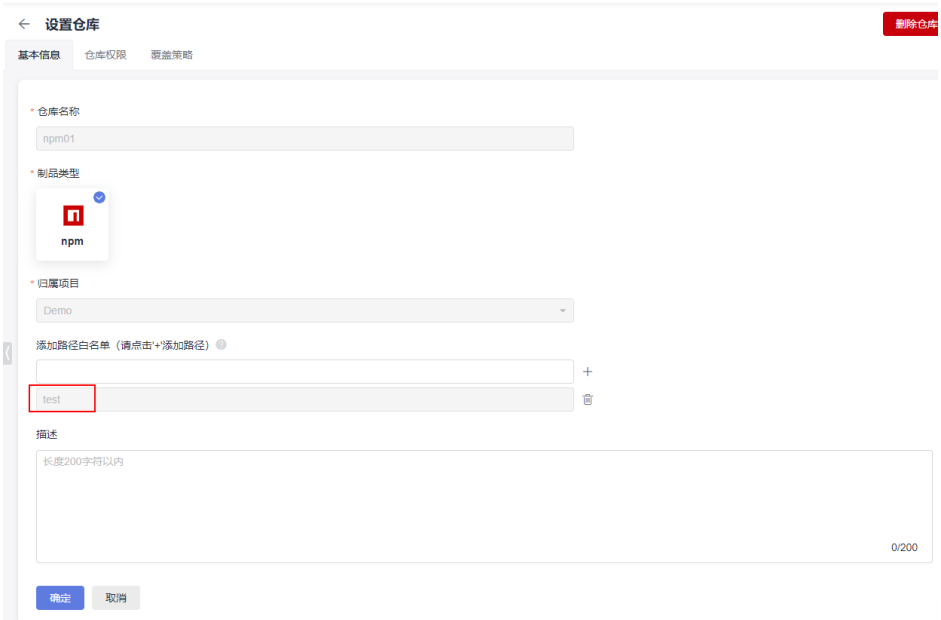
前提条件

- 已有可用项目。如果没有项目，请先**新建CodeArts项目**。
- 已创建NPM格式私有依赖库。
- 添加当前账号对当前私有库的权限，请参考**配置私有依赖库权限**。

发布私有组件到 NPM 私有依赖库

步骤1 下载私有依赖库配置文件。

1. 登录制品仓库，进入NPM私有依赖库。单击页面右侧“设置仓库”，记录仓库的路径。

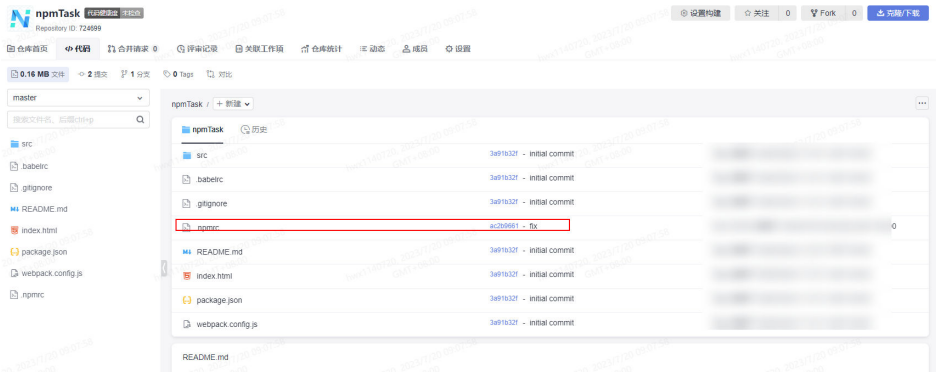


2. 单击“取消”返回私有依赖库页面，单击页面右侧“操作指导”。
3. 在弹框中单击“下载配置文件”。

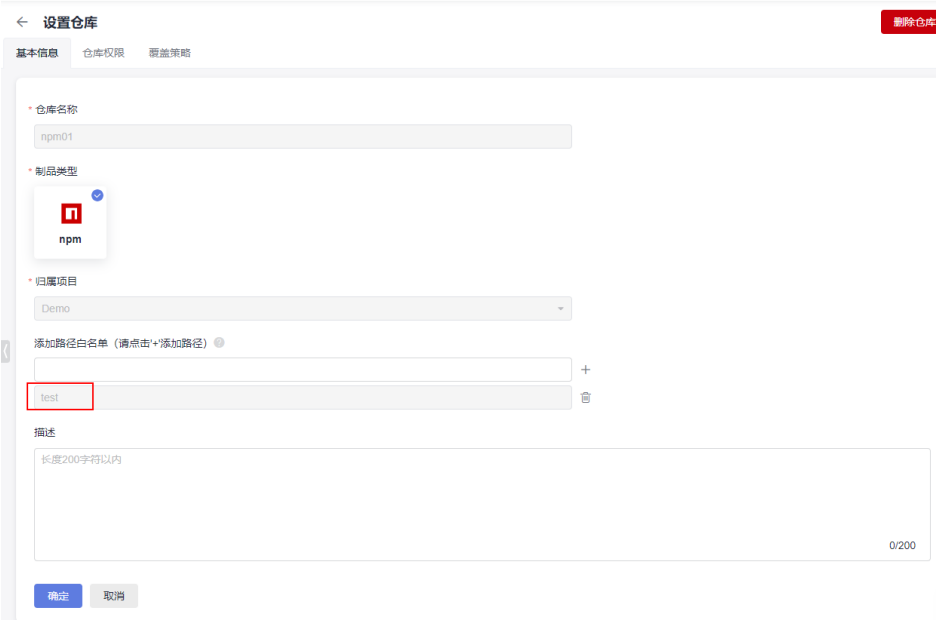
4. 在本地将下载的“npmrc”文件另存为“.npmrc”文件。

步骤2 配置代码仓库。

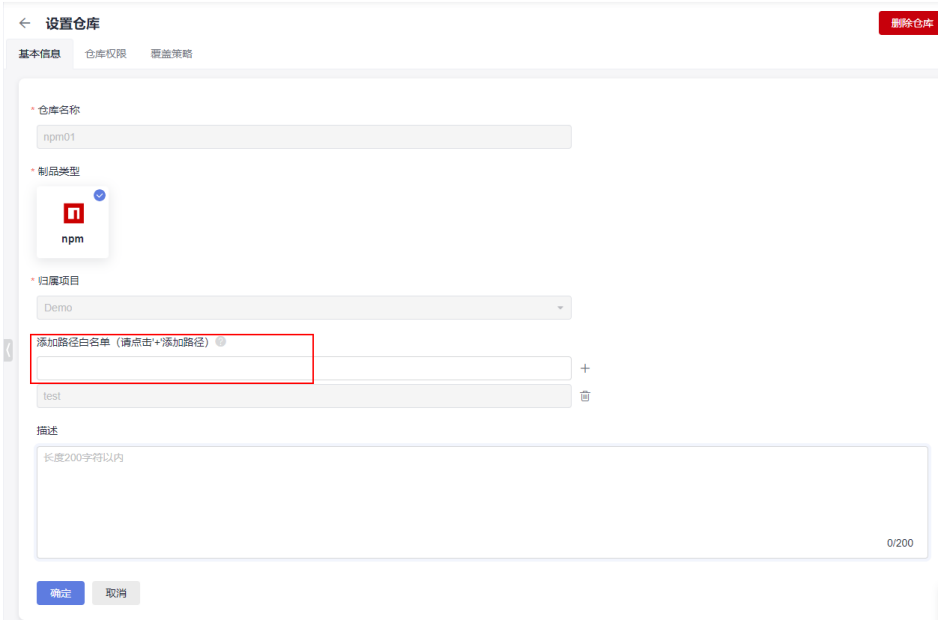
1. 进入代码托管服务，创建Node.js代码仓库（操作步骤请参考**创建云端仓库**）。本文使用模板“nodejs Webpack Demo”创建代码仓库。
2. 进入代码仓库，将“.npmrc”文件**上传至代码仓库**的根目录中。



3. 在代码仓库中找到“package.json”文件并打开，将在“编辑私有依赖库”页面中记录的路径信息添加到**name**字段对应的值中。



实际操作中，若出现**name**字段的值固定且不便修改的情况，则可以在“编辑私有依赖库”页面将该值配置到“添加路径”字段中。



步骤3 配置并执行编译构建任务。

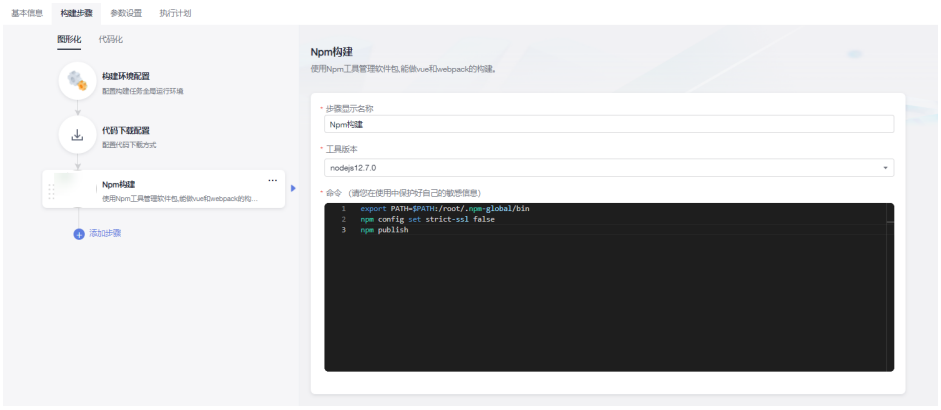
1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

   在页面中选择“空白构建模板”，单击“下一步”。

2. 添加步骤“Npm构建”。

3. 编辑步骤“Npm构建”。
   - 工具版本按照实际选择，本文中选择“nodejs12.7.0”。
   - 删除已有命令行，输入以下命令：

```
export PATH=$PATH:/root/.npm-global/bin
npm config set strict-ssl false
npm publish
```

4. 单击“新建并执行”，启动构建任务执行。

   待任务执行成功时，进入私有依赖库，可找到通过构建任务上传的NPM私有组件。

----结束

从 NPM 私有依赖库获取依赖包

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

步骤1 配置代码仓库。

1. 进入代码托管服务，创建Node.js代码仓库（操作步骤请参考**创建云端仓库**）。本文使用模板“nodejs Webpack Demo”创建代码仓库。
2. 参考**发布私有组件到NPM私有依赖库**，获取“.npmrc”文件并上传至需要使用Npm依赖包的代码仓库根目录中。
3. 在代码仓库中找到“package.json”文件并打开，将依赖包配置到**dependencies**字段中，本文中配置的值为：

   ```
   "@test/vue-demo": "^1.0.0"
   ```

步骤2 配置并执行编译构建任务。

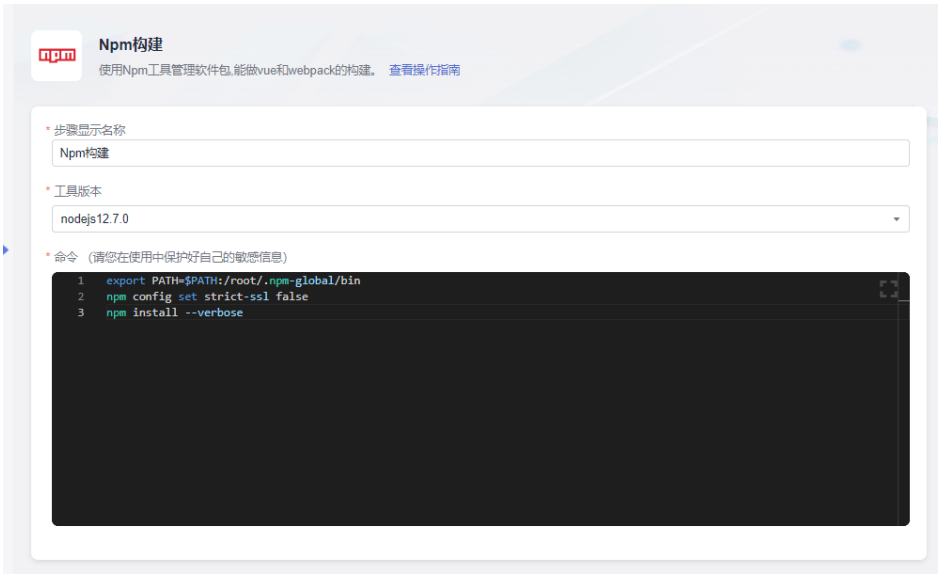
1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

   在页面中选择“空白构建模板”，单击“下一步”。

2. 添加步骤“Npm构建”。

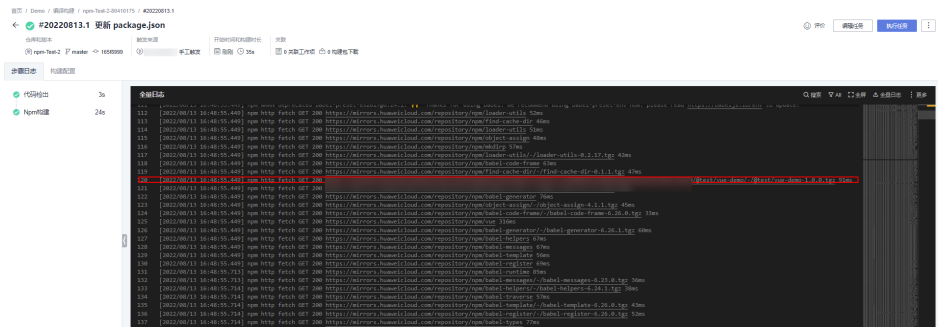
3. 编辑步骤“Npm构建”。
   - 工具版本按照实际选择，本文中选择“nodejs12.7.0”。
   - 删除已有命令行，输入以下命令：

```
export PATH=$PATH:/root/.npm-global/bin
npm config set strict-ssl false
npm install --verbose
```



步骤3 单击“新建并执行”，启动构建任务执行。

待任务执行成功时，查看构建任务详情，在日志中找到类似如下内容，说明编译构建任务从私有依赖库完成了依赖包下载并构建成功。



----结束

NPM 命令简介

在编译构建任务命令行中，还可以配置如下NPM命令，以完成其它功能：

- 删除私有依赖库中已存在的私有组件
  ```
  npm unpublish @socpe/packageName@version
  ```
- 获取标签列表
  ```
  npm dist-tag list @scope/packageName
  ```
- 新增标签
  ```
  npm dist-tag add @scope/packageName@version tagName --registry registryUrl --verbose
  ```
- 删除标签
  ```
  npm dist-tag rm @scope/packageName@version tagName --registry registryUrl --verbose
  ```

命令行参数说明：

- scope：私有依赖库路径，查看方法请参考**发布私有组件到NPM私有依赖库**。
- packageName：“package.json”文件中，**name**字段中scope之后的部分。
- version：“package.json”文件中，**version**字段对应的值。
- registryUrl：私有库配置文件中的对应scope的私有库地址url。
- tagName：标签名称。

以**发布私有组件到NPM私有依赖库**发布的私有组件为例：

- **scope**对应的值为“test”。
- **packageName**对应的值为“vue-demo”。
- **version**对应的值为“1.0.0”。

因此，删除此组件的命令应为：

```
npm unpublish @test/vue-demo@1.0.0
```

4 通过编译构建任务发布/获取 Go 私有组件

本文档介绍如何通过编译构建任务发布私有组件到Go私有依赖库、如何从Go私有依赖库获取依赖包完成编译构建任务。

前提条件

- 已有可用项目。如果没有项目，请先**新建CodeArts项目**。
- 已创建Go格式私有依赖库。
- 添加当前账号对当前私有库的权限，请参考**配置私有依赖库权限**。

发布私有组件到 Go 私有依赖库

步骤1 下载私有依赖库配置文件。

1. 登录制品仓库，进入Go私有依赖库。单击页面右侧“操作指导”。
2. 在弹框中单击“下载配置文件”。

步骤2 配置代码仓库。

1. 进入代码托管服务。创建Go语言代码仓库（操作步骤请参考**创建云端仓库**）。本文中使用仓库模板“Go Web Demo”创建代码仓库。
2. 准备“go.mod”文件，并**上传至代码仓库**的根目录中。本文中使用的“go.mod”文件如下所示：

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


步骤3 配置并执行编译构建任务。

1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

   在页面中选择“空白构建模板”，单击“下一步”。

2. 添加步骤“Go语言构建”。

3. 编辑步骤“Go语言构建”。
   - 工具版本按照实际选择，本文中选择“go-1.13.1”。
   - 删除已有命令行，打开在步骤**步骤1**中下载的配置文件，将文件中的“LINUX下配置go环境变量命令”复制到命令框中。
   - 将配置文件中go上传命令代码段复制到命令框中，并参考**Go Modules打包方式简介**替换命令行中的参数信息（本文打包版本为“v1.0.0”）。
4. 单击“新建并执行”，启动构建任务执行。

   待页面提示“构建成功”时，进入私有依赖库，可找到通过构建任务上传的Go私有组件。

----结束

从 Go 私有依赖库获取依赖包

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

步骤1 参考**发布私有组件到Go私有依赖库**，下载私有依赖库配置文件。

步骤2 进入代码托管服务，创建Go语言代码仓库（操作步骤请参考**创建云端仓库**）。本文中使用仓库模板“GoWebDemo”创建代码仓库。

步骤3 配置并执行编译构建任务。

1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

   在页面中选择“空白构建模板”，单击“下一步”。

2. 添加步骤“Go语言构建”。
3. 编辑步骤“Go语言构建”。
   - 工具版本按照实际选择，本文中选择“go-1.13.1”。
   - 删除已有命令行，打开已下载的私有依赖库配置文件，将文件中的“LINUX下配置go环境变量命令”代码段复制到命令框中。
   - 根据下载版本，选择配置文件中“go下载命令”相应的命令行复制到命令框中，并将“<modulename>”参数值。（本文中为“example.com/demo”）。

步骤4 单击“新建并执行”，启动构建任务执行。

待页面提示“构建成功”时，查看构建任务详情，在日志中找到类似如下内容，说明编译构建任务从私有依赖库完成了依赖包下载并构建成功。

----结束

Go Modules 打包方式简介

本文采用Go Modules打包方式完成Go组件的构建与上传。

打包命令主要包括以下几部分：

1. 在工作目录中创建源文件夹。
   ```
   mkdir -p {module}@{version}
   ```
2. 将代码源拷贝至源文件夹下。
   ```
   cp -rf . {module}@{version}
   ```
3. 压缩组件zip包。
   ```
   zip -D -r [包名] [包根目录名称]
   ```
4. 上传组件zip包与“go.mod”文件到私有依赖库中。
   ```
   curl -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。

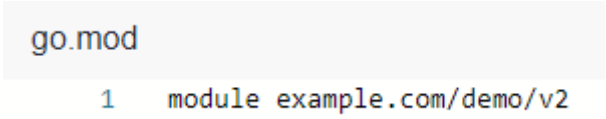
- 对于zip包，参数“filePath”为“example.com/demo/@v/v1.0.0.zip”，“localFile”为“v1.0.0.zip”。
- 对于“go.mod”文件，参数“filePath”为“example.com/demo/@v/v1.0.0.mod”，“localFile”为“example.com/demo@v1.0.0/go.mod”。

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



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。

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

5 通过编译构建任务发布/获取 PyPI 私有组件

本文档介绍如何通过编译构建任务发布私有组件到PyPI私有依赖库、如何从PyPI私有依赖库获取依赖包完成编译构建任务。

前提条件

- 已有可用项目。如果没有项目，请先**新建CodeArts项目**。
- 已创建PyPI格式私有依赖库。
- 添加当前账号对当前私有库的权限，请参考**配置私有依赖库权限**。

发布私有组件到 PyPI 私有依赖库

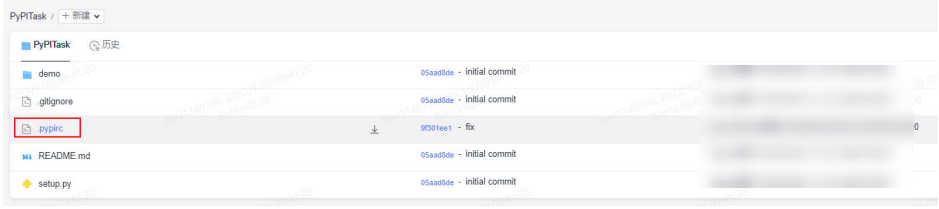
步骤1 下载私有依赖库配置文件。

1. 登录制品仓库，进入PyPI私有依赖库。单击页面右侧“操作指导”。
2. 在弹框中找到“发布配置”，单击“下载配置文件”。

3. 在本地将下载的“pypirc”文件另存为“.pypirc”文件。

步骤2 配置代码仓库。

1. 进入代码托管服务，创建Python代码仓库（操作步骤请参考**新建仓库**）。本文使用模板“Python3 Demo”创建代码仓库。
2. 进入代码仓库，将“.pypirc”文件**上传至代码仓库**的根目录中。



步骤3 配置并执行编译构建任务。

1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

   在页面中选择“空白构建模板”，单击“下一步”。

2. 添加步骤“SetupTool构建”。

3. 编辑步骤“SetupTool构建”。
   - 工具版本按照实际选择，本文中选择“python3.6”。
   - 删除已有命令行，输入以下命令：

     ```
     # 请保证代码根目录下有setup.py文件,下面命令将把工程打为whl包
     python setup.py bdist_wheel
     # 设置当前项目根目录下的.pypirc文件为配置文件
     cp -rf .pypirc ~/
     # 上传组件至pypi私有库
     twine upload -r pypi dist/*
     ```

     如果上传时报证书问题，请在上述命令首行添加以下命令，设置环境变量跳过证书校验：

     ```
     export CURL_CA_BUNDLE=""
     ```

4. 单击“新建并执行”，启动构建任务执行。

   待任务执行成功时，进入私有依赖库，可找到通过构建任务上传的PyPI私有组件。

----结束

从 PyPI 私有依赖库获取依赖包

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

步骤1 下载私有依赖库配置文件。

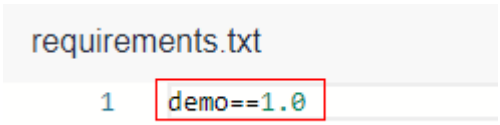
1. 进入PyPI私有依赖库，单击页面右侧“操作指导”。
2. 在弹框中找到“下载配置”，单击“下载配置文件”。

3. 在本地将下载的“pip.ini”文件另存为“pip.conf”文件。

步骤2 配置代码仓库。

1. 进入代码托管服务，创建Python代码仓库（操作步骤请参考**新建仓库**）。本文使用模板“Python3 Demo”创建代码仓库。
2. 进入代码仓库，将“pip.conf”文件上传至需要使用PyPI依赖包的代码仓库根目录中。
3. 在代码仓库中找到“requirements.txt”文件并打开（若没有请**新建文件**），将依赖包配置添加到此文件中，本文中配置的值为：

```
demo ==1.0
```



步骤3 配置并执行编译构建任务。

1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。

   在页面中选择“空白构建模板”，单击“下一步”。
2. 添加步骤“Setup Tool构建”。

3. 编辑步骤“SetupTool构建”。
   - 工具版本按照实际选择，本文中选择“python3.6”。
   - 删除已有命令行，输入以下命令：

```
# 可以通过此命令设置当前项目根目录下的pip.conf文件为配置文件
export PIP_CONFIG_FILE=./pip.conf
# 下载pypi组件
pip install -r requirements.txt --no-cache-dir
```


步骤4 单击“新建并执行”，启动构建任务执行。

待任务执行成功时，查看构建任务详情，在日志中找到类似如下内容，说明编译构建任务从私有依赖库完成了依赖包下载并构建成功。

----结束

6 通过 Linux 命令行上传/获取 Rpm 私有组件

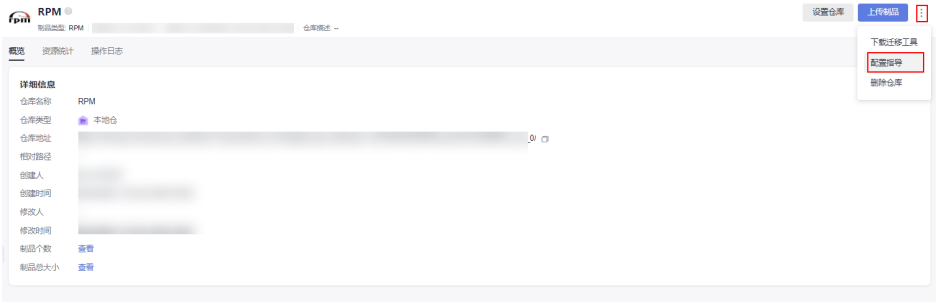
本文档介绍如何Linux命令行上传私有组件到Rpm私有依赖库、如何从Rpm私有依赖库获取依赖包。

前提条件

- 已有可用的Rpm组件。
- 已有可连通公网的Linux系统主机。
- 已创建Rpm格式私有依赖库。
- 添加当前账号对当前私有库的权限，请参考**配置私有依赖库权限**。

发布私有组件到 Rpm 私有依赖库

步骤1 登录制品仓库，进入Rpm私有依赖库。单击页面右侧“操作指导”。



步骤2 在弹框中单击“下载配置文件”。

步骤3 在Linux主机中执行以下命令，上传Rpm组件。

```
curl -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：

步骤4 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

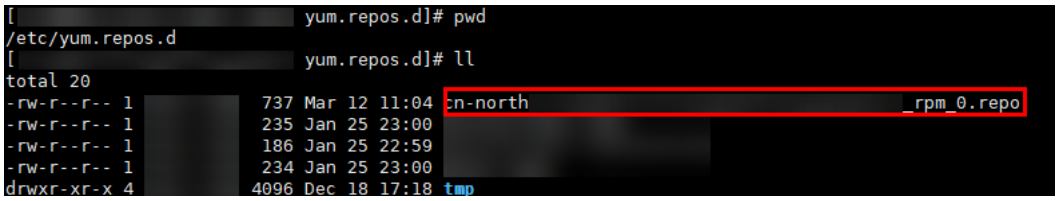
从 Rpm 私有依赖库获取依赖包

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**发布私有组件到Rpm私有依赖库**，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

7 通过 Linux 命令行上传/获取 Debian 私有组件

本文档介绍如何通过Linux命令行上传私有组件到Debian私有依赖库、如何从Debian私有依赖库获取依赖包。

前提条件

- 已有可用的Debian组件。
- 已有可连通公网的Linux系统主机。
- 已创建Debian格式私有依赖库。
- 添加当前账号对当前私有库的权限，请参考**配置私有依赖库权限**。

发布私有组件到 Debian 私有依赖库

步骤1 登录制品仓库，进入Debian私有依赖库。单击页面右侧“操作指导”。

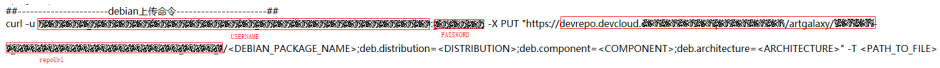
步骤2 在弹框中单击“下载配置文件”。

步骤3 在Linux主机中执行以下命令，上传Debian组件。

```
curl -u <USERNAME>:<PASSWORD> -X PUT "https:// <repoUrl>/
<DEBIAN_PACKAGE_NAME>;deb.distribution=<DISTRIBUTION>;deb.component=<COMPONENT>;deb.architecture=<ARCHITECTURE>" -T <PATH_TO_FILE>
```


其中“USERNAME”、“PASSWORD”、“repoUrl”来源于**上一步**下载的配置文件中“Debian上传命令”部分。

- USERNAME：上传文件使用的用户名，可以从Debian配置文件中获取，参考示例图片。
- PASSWORD：上传文件使用的密码，可以从Debian配置文件中获取，参考示例图片。
- repoUrl：上传文件使用的url，可以从Debian配置文件中获取，参考示例图片。

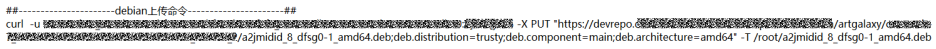


“DEBIAN_PACKAGE_NAME”、“DISTRIBUTION”、“COMPONENT”、“ARCHITECTURE”来源于待上传的Debian组件。

以组件“a2jmidid_8_dfsg0-1_amd64.deb”为例。

- DEBIAN_PACKAGE_NAME：软件包名称，例如：“a2jmidid_8_dfsg0-1_amd64.deb”。
- DISTRIBUTION：发行版本，例如：“trusty”。
- COMPONENT：组件名称，例如：“main”。
- ARCHITECTURE：体系结构，例如：“amd64”。
- PATH_TO_FILE：Debian组件的本地存储路径，例如：“/root/a2jmidid_8_dfsg0-1_amd64.deb”。

完整的命令如下图所示：



步骤4 命令执行成功，进入私有依赖库，可找到已上传的Debian私有组件。

----结束

从 Debian 私有依赖库获取依赖包

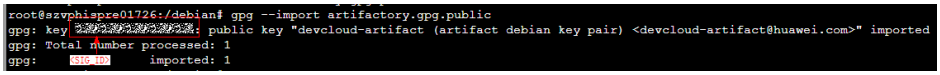
以**发布私有组件到Debian私有依赖库**中发布的Debian私有组件为例，介绍如何从Debian私有依赖库中获取依赖包。

步骤1 参考**发布私有组件到Debian私有依赖库**，下载Debian私有依赖库的“公钥”文件。

步骤2 导入gpg公钥。

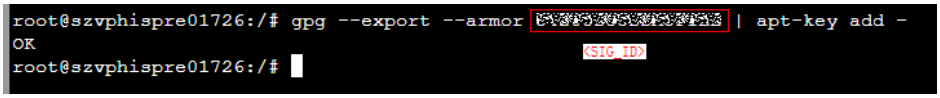
```
gpg --import <PUBLIC_KEY_PATH>
```

PUBLIC_KEY_PATH：Debian公钥的本地存储路径，例如：“artifactory.gpg.public”。



步骤3 apt导入公钥。

```
gpg --export --armor <SIG_ID> | apt-key add -
```



步骤4 apt仓库源添加。

打开配置文件（获取方法参考**发布私有组件到Debian私有依赖库**），将文件中所有“DISTRIBUTION”替换为上传Debian文件时使用的“COMPONENT”值（例如“main”），并根据下载的配置文件sources.list执行仓库源添加。

步骤5 仓库源添加后，使用如下命令更新仓库源。

```
apt-get update
```

步骤6 执行以下命令，下载Debian包。其中a2jmidid为包的“PACKAGE”值，请根据实际情况修改。

```
apt download a2jmidid
```

<PACKAGE>获取方法如下：

下载Debian组件的Packages源数据，以a2jmidid包为例。

----结束

8 批量迁移 Maven/NPM/PyPI 组件至私有依赖库

背景信息

制品仓库服务的私有依赖库支持通过页面手动上传下载私有组件，还支持与本地开发环境对接，通过本地开发环境上传下载私有组件。通过私有依赖库上传组件的操作请参考**通过私有依赖库页面上传/下载私有组件**。

如果待上传的包太多，单个上传会很繁琐。通过私有依赖库提供的批量迁移工具可以提高上传效率，在易用性上能够更加便捷地从Nexus或其他类型仓库批量上传组件至私有依赖库。

准备工作

- 已创建对应格式的私有依赖库。
- 运行环境为Python3。
- 若使用Nexus，迁移工具必须和Nexus运行在同一台Linux主机上，且必须和CodeArts服务网络连通，该主机必须安装Python3。

迁移 Maven 组件

步骤1 从Maven本地仓库（例如：C:\Users\xxxxx\.m2\repository）找到需迁移的组件，复制到指定目录（用户自己指定）。

步骤2 进入私有依赖库，在左侧边栏中选择目标Maven私有依赖库。

步骤3 单击仓库名称，页面中仓库的详细信息显示“仓库地址”，单击 即可复制该地址。

步骤4 单击页面右上方“操作指导”，在弹框中单击“下载配置文件”，将配置文件settings.xml下载到本地。

在本地打开配置文件，在文件中搜索并找到用户名与密码。


```
<server>
    <id>releases</id>
    <username>                                                    </username>
    <password>          </password>
</server>
<server>
    <id>snapshots</id>
    <username>                                                    </username>
    <password>          </password>
</server>
<server>
    <id>z_mirrors</id>
</server>
```

步骤5 单击页面右上方···，单击“下载迁移工具”将迁移工具压缩包（脚本uploadArtifact2.py、配置文件artifact.conf）下载到本地。

步骤6 配置artifact.conf。

```
[artifact]
packageType = 组件类型，设置为Maven
userInfo = username:password（步骤4中获取的用户名与密码）
repoRelease = 仓库类型为 Release地址（步骤3中获取的仓库地址）
repoSnapshot = 仓库类型为 Snapshot地址（步骤3中获取的仓库地址）
srcDir = 组件的目录路径（用户自己指定），如存放在步骤1下载组件的目标路径。

[nexus]
nexusAddr=nexus地址
nexusPort=nexus端口
repoName=待迁移的nexus仓库名称
userName=nexus用户名
passwd=nexus密码
```

步骤7 执行迁移脚本python uploadArtifact2.py。

步骤8 进入对应的私有依赖库，查看组件包是否上传成功。

----结束

迁移 npm 组件

步骤1 进入私有依赖库，在左侧边栏中选择目标npm私有依赖库。

步骤2 单击仓库名称，页面中仓库的详细信息显示“仓库地址”，单击 即可复制该地址。

步骤3 单击页面右上方“操作指导”，在弹框中单击“下载配置文件”，将npmrc文件下载到本地。

在本地打开配置文件，在文件中找到“_auth”字段的值并进行base64解码。

步骤4 单击页面右上方···，单击“下载迁移工具”将迁移工具压缩包（脚本uploadArtifact2.py、配置文件artifact.conf）下载到本地。

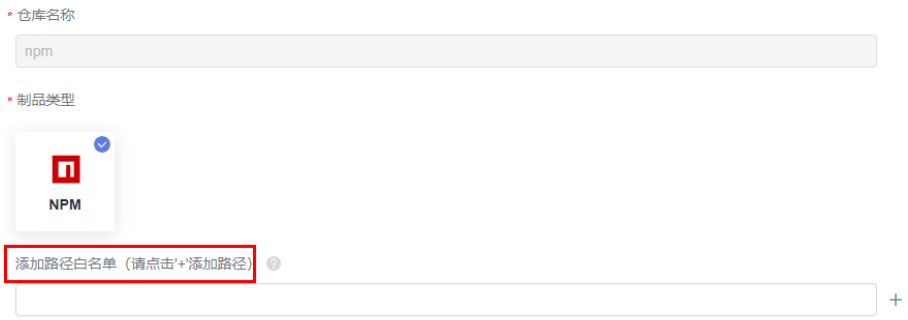
步骤5 配置artifact.conf。

```
[artifact]
packageType = 组件类型，设置为npm
userInfo = npm仓库下的配置文件npmrc中通过base64 解密后的_auth字段的值（参考步骤3）
repoRelease = 私有依赖库地址（步骤2中获取的仓库地址）
repoSnapshot = 保留为空
srcDir = 组件的目录路径，例如：C:\Users\xxxxxx\repository，用户自己指定


[nexus]
nexusAddr=nexus地址
nexusPort=nexus端口
repoName=待迁移的nexus仓库名称
userName=nexus用户名
passwd=nexus密码
```


步骤6 检查对应npm仓是否配置了路径白名单。

请确认package.json中私有二进制包是否在白名单内，只有与白名单内匹配的二进制包才能上传成功；没有配置白名单，则package.json中私有二进制包都可以上传成功。



步骤7 执行迁移脚本python uploadArtifact2.py。

步骤8 进入对应的私有依赖库，查看组件包是否上传成功。

----结束

迁移 PyPI 组件

步骤1 进入私有依赖库，在左侧边栏中选择目标PyPI私有依赖库。

步骤2 单击仓库名称，页面中仓库的详细信息显示“仓库地址”，单击即可复制该地址。

步骤3 单击页面右上方“操作指导”，在弹框中单击“下载配置文件”，将配置文件pypirc下载到本地。

在本地打开配置文件，在文件中搜索并找到用户名与密码。

步骤4 单击页面右上方，单击“下载迁移工具”将迁移工具压缩包（脚本uploadArtifact2.py、配置文件artifact.conf）下载到本地。

步骤5 配置artifact.conf。

```
[artifact]
packageType = 组件类型，设置为pypi
userInfo = username:password（步骤3中获取的用户名与密码）
repoRelease = 私有依赖库地址（步骤2中获取的仓库地址）
repoSnapshot = 保留为空
srcDir = 组件的目录路径，例如：C:\Users\xxxxxx\repository，用户自己指定

[nexus]
nexusAddr=nexus地址
nexusPort=nexus端口
repoName=待迁移的nexus仓库名称
userName=nexus用户名
passwd=nexus密码
```

步骤6 执行迁移脚本python uploadArtifact2.py。

步骤7 进入私有库页面查看二进制包是否上传成功。

----结束

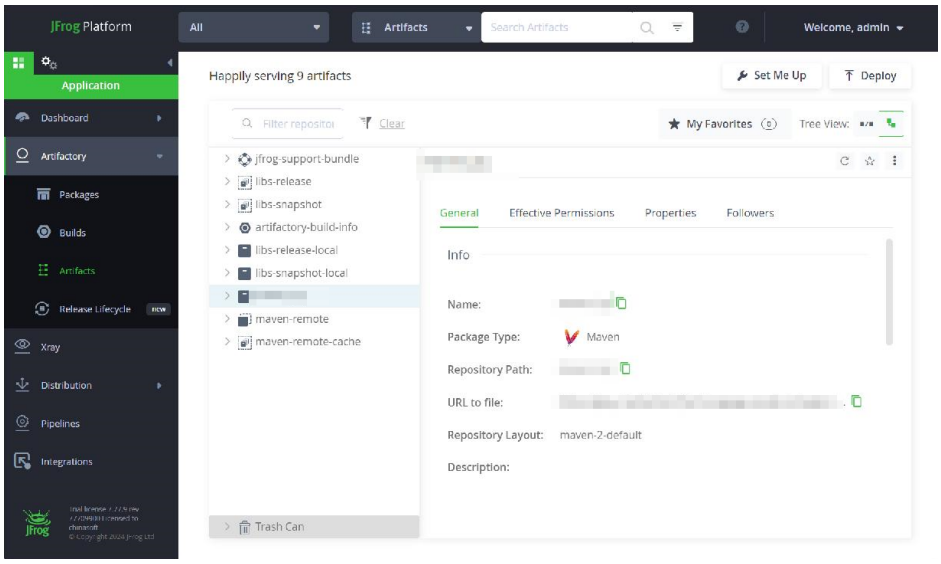
9 批量迁移 JFrog 仓库至私有依赖库

背景信息

JFrog仓库是一个用于存储和管理软件包的中央存储库，提供了一种集中式的方式来管理软件包，支持各种软件包管理工具，如Maven、Gradle、npm、NuGet等。CodeArts Artifact的私有依赖库提供了批量迁移工具，支持将JFrog仓库迁移至私有依赖库。本节介绍如何批量迁移JFrog仓库至私有依赖库。

待迁移的JFrog仓库示例如图9-1所示。

图 9-1 待迁移的 JFrog 仓库



准备工作

依赖Java运行环境，需要安装JRE，请参考**安装JRE**。

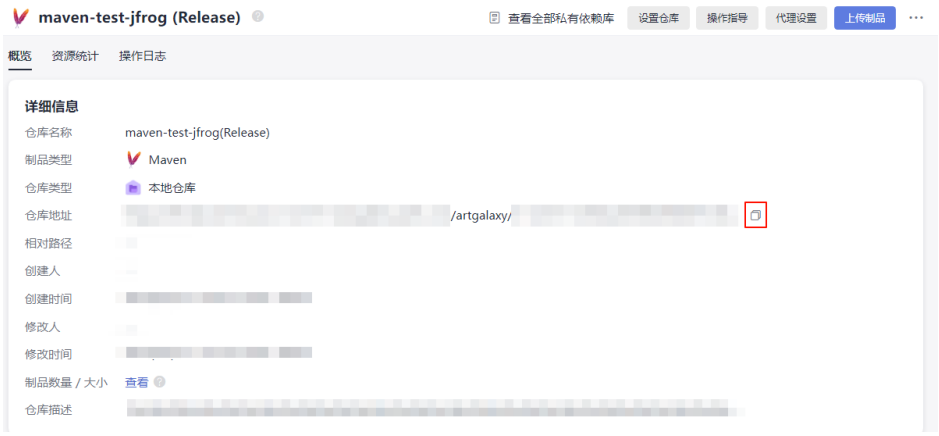
步骤一：获取私有依赖库地址与配置

步骤1 获取私有依赖库地址。

1. 进入私有依赖库，选择“仓库视图”，并在左侧边栏中选择目标Maven私有依赖库。

2. 单击仓库名称，右侧页面中仓库的“概览”页签中详细信息显示“仓库地址”。单击即可复制仓库地址。

图 9-2 获取私有依赖库地址



步骤2 获取私有依赖库配置。

1. 单击页面右上方“操作指导”。
2. 在“操作指导”对话框中单击“下载配置文件”，下载配置文件**settings.xml**至本地。

图 9-3 下载配置文件

3. 在本地打开配置文件**settings.xml**，在文件中搜索并找到如下红框中的用户名与密码。

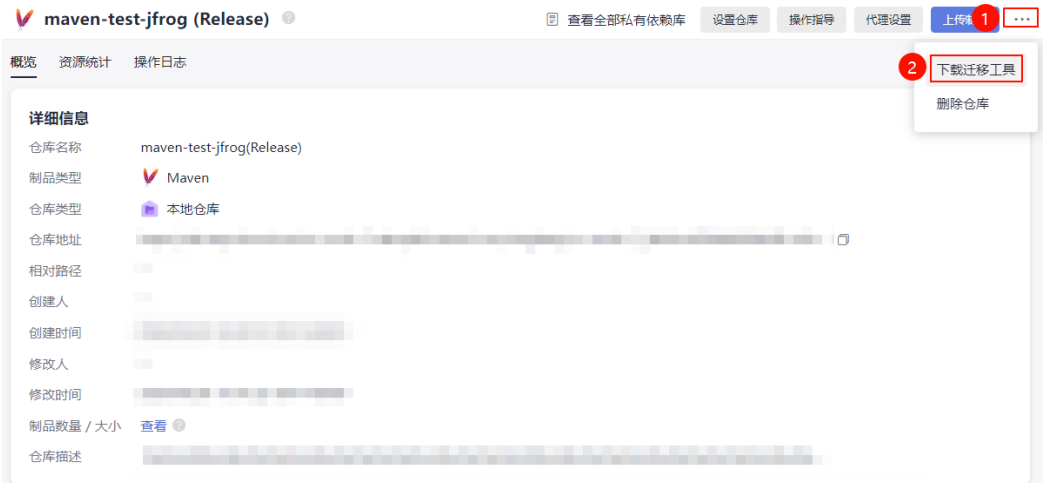
```
<server>
    <id>releases</id>
    <username>                                                    </username>
    <password>          </password>
</server>
<server>
    <id>snapshots</id>
    <username>                                                    </username>
    <password>          </password>
</server>
<server>
    <id>z_mirrors</id>
</server>
```

----结束

步骤二：配置迁移工具

步骤1 返回私有依赖库，单击页面右侧 ···并在下拉列表选择“下载迁移工具”。

图 9-4 下载迁移工具



步骤2 将迁移工具**MigrateTool.rar**包下载到本地，并执行以下命令，将**MigrateTool.rar**包解压并进入解压后的目录中。

```
unrar x MigrateTool.rar
cd MigrateTool/
```

步骤3 用记事本打开**MigrateTool.rar**包解压后目录中的**application.yaml**文件，配置**表9-1**所示参数。

表 9-1 配置迁移工具参数

参数名称	参数说明
package_type	JFrog源仓库类型，配置为“maven”。
repo_type	JFrog源仓库类型，配置为“jfrog”。
domain	JFrog源仓库地址，例如“http://*本地JFrog仓库IP.本地JFrog仓库端口*/artifactory”。
repo	需要迁移的JFrog源仓库名称，根据实际名称填写。
user_name	登录JFrog源仓库的账号，根据实际情况填写。
password	登录JFrog源仓库的密码，根据实际情况填写。

参数名称	参数说明
target_repo_type	迁移后的目标仓库类型，配置为“artifactory”。
target_domain	迁移后的目标仓库地址，配置为图9-5中“/artgalaxy/”前半段的1的信息。
target_repo	迁移后的目标仓库ID，配置为图9-5中“/artgalaxy/”后半段的2的信息。
target_user_name	迁移后的目标仓库账号，配置为从步骤2.3中获取的**username**。
target_password	迁移后的目标仓库密码，配置为从步骤2.3中获取的**password**。

图 9-5 迁移后的目标仓库详细信息

----结束

步骤三：执行迁移

执行以下命令迁移JFrog仓库至私有依赖库。

```
nohup java -jar /tools/relocation-jfrog.jar --spring.config.additional-location=./application-product.yaml
> /log/relocation-jfrog.log 2>&1 &
```

图 9-6 执行迁移

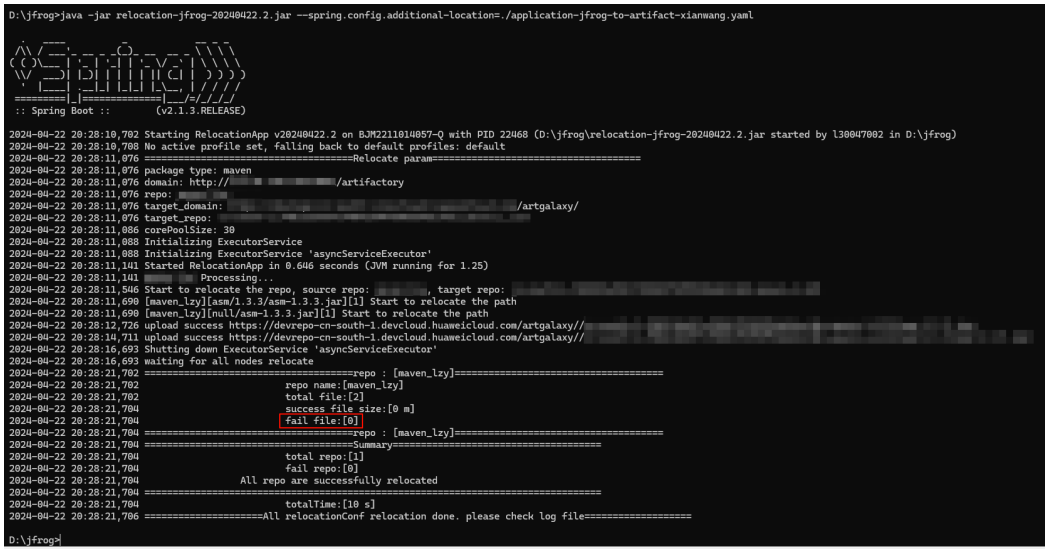


图9-6中“fail file”值为**0**时则表示迁移成功；否则迁移失败，可尝试重新执行迁移或联系客服寻求技术支持。

说明

- 该命令会处于后台运行。
- /tools/relocation-jfrog.jar：指定迁移工具路径。
- --spring.config.additional-location=./application-product.yaml：指定配置文件路径。
- /log/relocation-jfrog.log：指定迁移工具执行日志路径，可通过该日志查看迁移情况。