

制品仓库

# 快速入门

文档版本 01  
发布日期 2024-10-30



版权所有 © 华为技术有限公司 2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



HUAWEI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。

# 安全声明

## 漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：

<https://www.huawei.com/cn/psirt/vul-response-process>

如企业客户须获取漏洞信息，请参见如下网址：

<https://securitybulletin.huawei.com/enterprise/cn/security-advisory>

---

## 目录

---

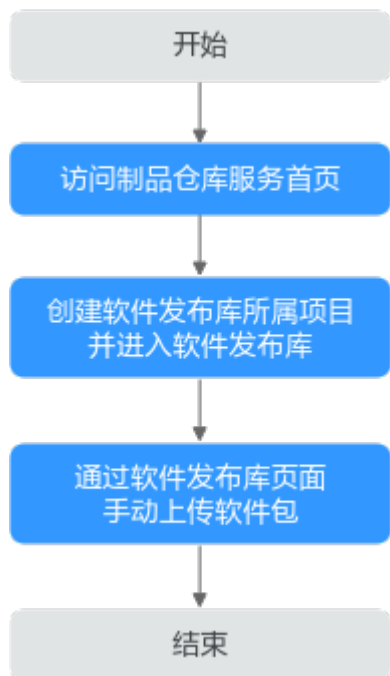
1 上传软件包到 CodeArts Artifact 软件发布库.....	1
2 上传私有组件到 CodeArts Artifact 中的 Maven 私有依赖库.....	3
3 通过编译构建任务发布/获取 Maven 私有组件.....	6
4 通过编译构建任务发布/获取 NPM 私有组件.....	11
5 通过编译构建任务发布/获取 Go 私有组件.....	17
6 通过编译构建任务发布/获取 PyPI 私有组件.....	22
7 通过 Linux 命令行上传/获取 Rpm 私有组件.....	26
8 通过 Linux 命令行上传/获取 Debian 私有组件.....	28

# 1 上传软件包到 CodeArts Artifact 软件发布库

软件包是软件开发过程中编译构建生成的中间产物，是持续集成和持续交付的中间环节。通过将软件包上传至软件发布库进行存储和管理，可以实现文件的可信存储，有效支撑软件开发活动，为部署实施提供了安全可靠的软件包来源，也为构建任务提供了重要依赖。

本文档向您介绍如何上传软件包到CodeArts Artifact软件发布库，帮助您快速建立对软件发布库的整体印象。主要操作流程如图1-1所示。

图 1-1 上传软件包到 CodeArts Artifact 软件发布库流程




## 准备工作

- 已注册华为账号并开通华为云。
- 已开通制品仓库服务。

## 访问制品仓库服务首页

**步骤1** [登录华为云控制台页面](#)。

**步骤2** 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3** 单击“立即使用”，进入制品仓库服务首页。

----结束

## 创建软件发布库所属项目并进入软件发布库

**步骤1** 单击导航栏“首页”。

**步骤2** 单击“新建项目”。

**步骤3** 单击“Scrum”项目模板。

**步骤4** 项目名称填写“Scrum01”，其它保持默认即可。

**步骤5** 单击“确定”，进入到“Scrum01”项目下。

**步骤6** 单击导航栏“制品仓库 > 软件发布库”，即可进入到该项目下的软件发布库。

### 说明

软件发布库无需新建，用户创建项目后，该项目下会自动创建同项目名称的软件发布库。

----结束

## 通过软件发布库页面手动上传软件包

**步骤1** 进入与已创建项目同名的仓库，单击页面右上方“上传制品”。

**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，默认即可。
- 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。

----结束

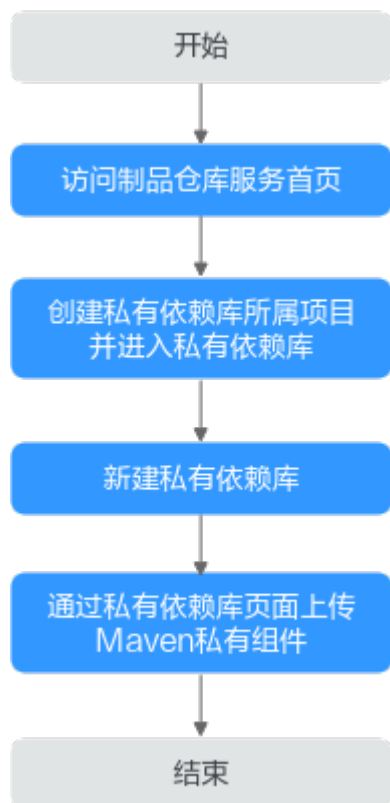
制品仓库服务支持从页面上上传软件包，也支持与编译构建服务对接，将构建生成的软件包上传到软件发布库，请参考[上传软件包到软件发布库](#)。

# 2 上传私有组件到 CodeArts Artifact 中的 Maven 私有依赖库

在日常开发过程中，开发人员时常需要共享一些组件给团队其他人员，私有依赖库则作为类似共享站的作用（同时也能储存组件），将需要共享的私有组件上传到私有依赖库，其他人员则可以便捷的从私有依赖库获取到这些组件。

本文向您介绍如何上传私有组件到CodeArts Artifact中的Maven私有依赖库，帮助您快速建立对私有依赖库的整体印象。主要操作流程如[图2-1](#)所示。

图 2-1 上传私有组件到 CodeArts Artifact 中的 Maven 私有依赖库流程




## 前提条件

- 已注册华为账号并开通华为云。
- 已开通制品仓库服务。

## 访问制品仓库服务首页

步骤1 登录华为云控制台页面。

步骤2 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3 单击“立即使用”，进入制品仓库服务首页。

----结束

## 创建私有依赖库所属项目并进入私有依赖库

步骤1 单击导航栏“首页”。

步骤2 单击“新建项目”。

步骤3 单击“Scrum”项目模板。

步骤4 项目名称填写“Scrum01”，其它保持默认即可。

步骤5 单击“确定”，进入到“Scrum01”项目下。

步骤6 单击导航栏“制品仓库 > 私有依赖库”，即可进入到该项目下的私有依赖库。

----结束

## 新建私有依赖库

步骤1 在制品仓库服务首页，选择“私有依赖库”页签。

步骤2 单击“新建制品仓库”。

步骤3 配置以下仓库基本信息，单击“确定”按钮。

- 仓库类型：分为“本地仓”和“聚合仓”，默认选择本地仓。
- 仓库名称：输入仓库名称。
- 制品类型：选择Maven类型。
- 归属项目：默认为当前项目，用户可以在下拉栏中选择其他目标项目。
- 添加路径白名单（非必选）：为私有依赖库配置路径白名单，不填即可。
- 版本策略：同时勾选后，Maven私有依赖库将生成Release和Snapshot两种类型仓库，默认即可。
- 描述（非必填）：用户可以输入长度不超过200字符的描述信息。

步骤4 新建成功的Maven私有依赖库将显示在仓库视图中。

----结束



## 通过私有依赖库页面上传 Maven 私有组件

- 步骤1** 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** 在弹框中上传模式选择“POM模式”。
- 步骤4** 在POM下，单击“选择文件”，从本地上传结尾是pom.xml或者以.pom的私有组件。
- 步骤5** 单击“上传”。
- 步骤6** 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。

### ----结束

制品仓库服务支持从页面上传私有组件至私有依赖库，也支持与编译构建服务对接，将Maven构建生成的软件包上传到私有依赖库，请参考[使用Maven构建](#)的参数表格。

# 3 通过编译构建任务发布/获取 Maven 私有组件

本文档介绍如何通过编译构建任务发布Maven私有组件至私有依赖库、及如何从私有依赖库获取Maven组件完成编译构建任务。

## 前提条件

- 已有可用项目。如果没有项目，请先[创建项目](#)。
- 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](#)。
- 已创建Maven格式私有依赖库，并[关联CodeArts Artifact中的Maven仓库与项目](#)。

## 发布 Maven 私有组件到私有依赖库

### 步骤1 配置代码仓库。

1. 登录软件开发生产线，进入已创建的项目。单击顶部菜单“服务 > 代码托管”，进入代码托管服务。
2. 创建Maven类型代码仓库（操作步骤请参考[创建云端仓库](#)）。本文中使用仓库模板“Java Maven Demo”创建代码仓库。
3. 进入代码仓库，在“pom.xml”中查看组件配置。

```

pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>com.huawei.demo</groupId>
4   <artifactId>javaMavenDemo</artifactId>
5   <packaging>jar</packaging>
6   <version>1.0</version>
7   <name>maven_demo</name>
8   <url>http://maven.apache.org</url>
9   <dependencies>
10    <dependency>
11     <groupId>junit</groupId>
12     <artifactId>junit</artifactId>
13     <version>3.8.1</version>
14     <scope>test</scope>
15    </dependency>
16  </dependencies>

```

**步骤2** 配置并执行编译构建任务。

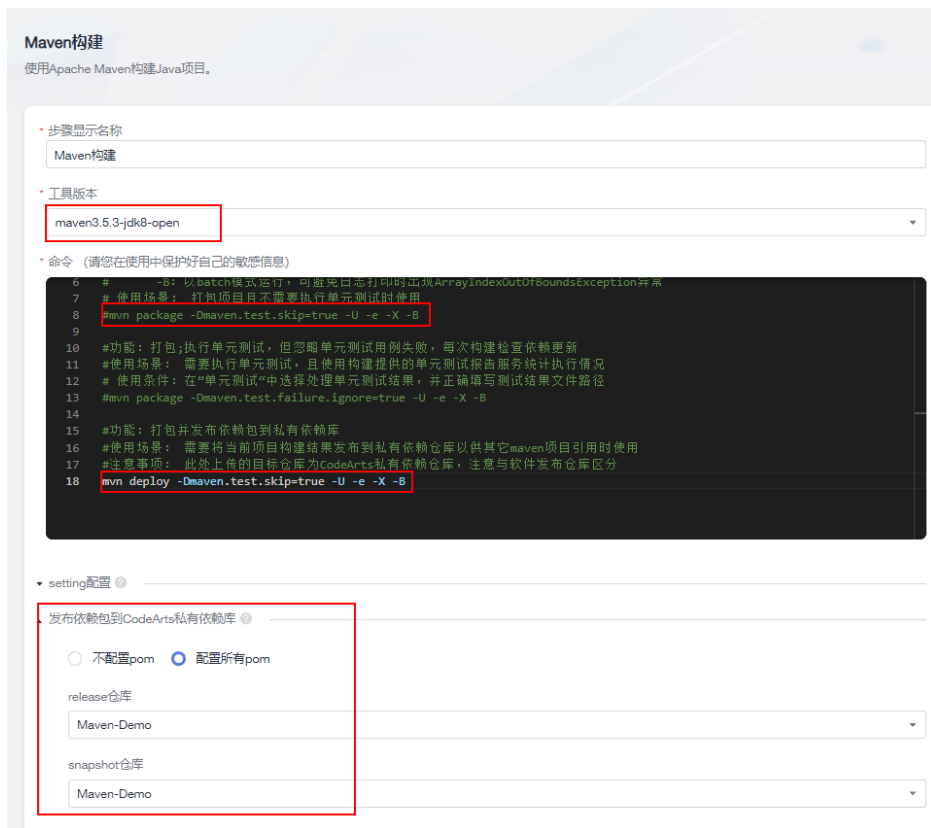
1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。  
在页面中选择“空白构建模板”，单击“下一步”。
2. 添加步骤“Maven构建”。



3. 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。

**说明**

若下拉列表中没有选项，请参考[管理Maven仓库与项目的关联](#)将Maven私有依赖库关联至构建任务所在的项目。



**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，进入私有依赖库，可找到通过构建任务上传的Maven私有组件。

----结束

## 从私有依赖库获取 Maven 组件作为依赖包

以[发布Maven私有组件到私有依赖库](#)中发布的Maven私有组件为例，介绍如何从私有依赖库中获取Maven组件作为依赖包。

**步骤1** 配置代码仓库。

1. 进入Maven私有依赖库，在仓库中找到Maven组件。单击与组件同名的“.pom”文件，在页面右侧单击“下载”。
2. 在本地打开下载的文件，找到<groupId>、<artifactId>、<version>代码行。



3. 进入代码托管服务。创建Maven类型代码仓库（操作步骤请参考[创建云端仓库](#)）。本文中使用仓库模板“Java Maven Demo”创建代码仓库。
4. 进入代码仓库，编辑文件“pom.xml”：将复制的dependency代码段粘贴在dependencies代码段中，并修改版本号<version>（例如2.0）。

```
pom.xml 大小: 1.41 KB 全
```

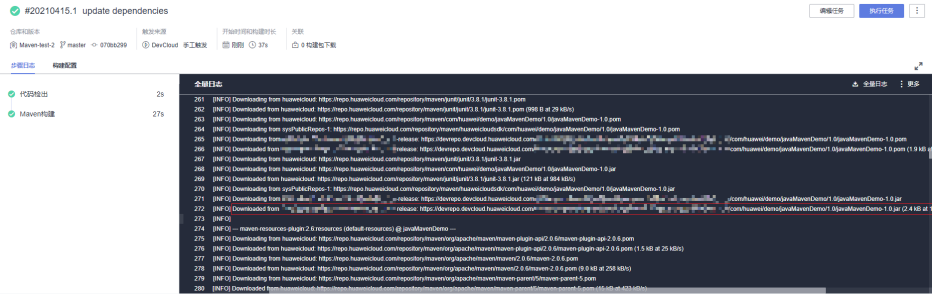
```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>com.huawei.demo</groupId>
4   <artifactId>javaMavenDemo</artifactId>
5   <packaging>jar</packaging>
6   <version>2.0</version>
7   <name>maven_demo</name>
8   <url>http://maven.apache.org</url>
9   <dependencies>
10    <dependency>
11      <groupId>junit</groupId>
12      <artifactId>junit</artifactId>
13      <version>3.8.1</version>
14      <scope>test</scope>
15    </dependency>
16    <dependency>
17      <groupId>com.huawei.demo</groupId>
18      <artifactId>javaMavenDemo</artifactId>
19      <version>1.0</version>
20    </dependency>
21  </dependencies>
```

## 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“Maven构建”。



3. 单击“新建并执行”，启动构建任务。  
当任务执行成功时，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

# 4 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。

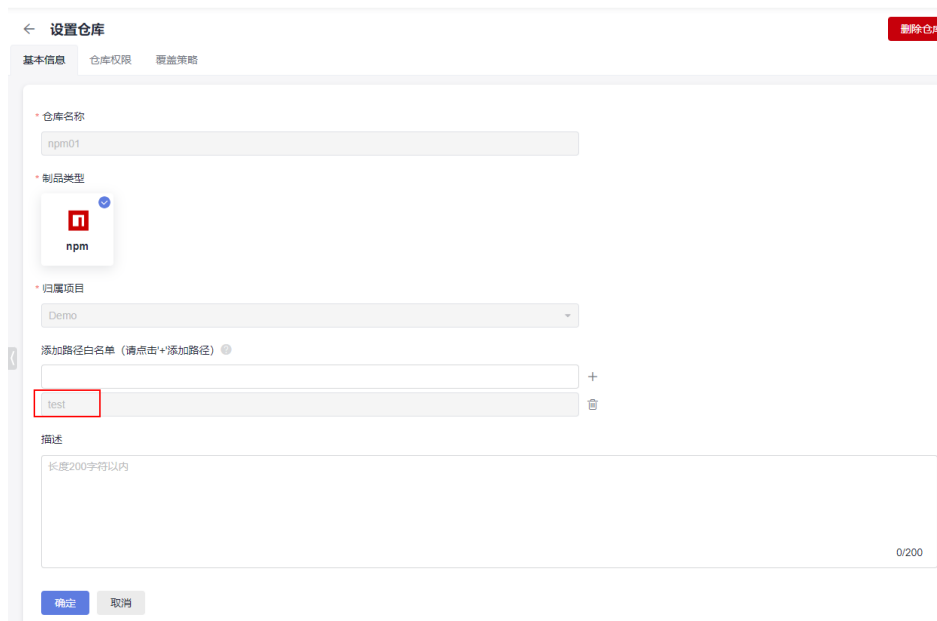
## 前提条件

- 已有可用项目。如果没有项目，请先[创建项目](#)。
- 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。



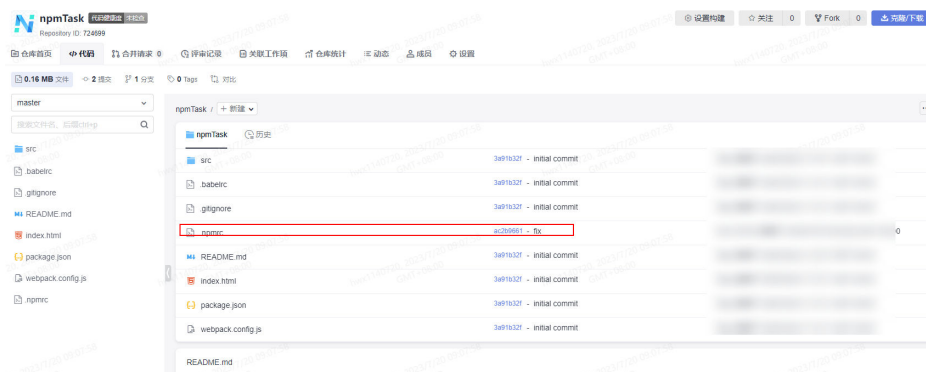
2. 单击“取消”返回私有依赖库页面，单击页面右侧“操作指导”。
3. 在弹框中单击“下载配置文件”。



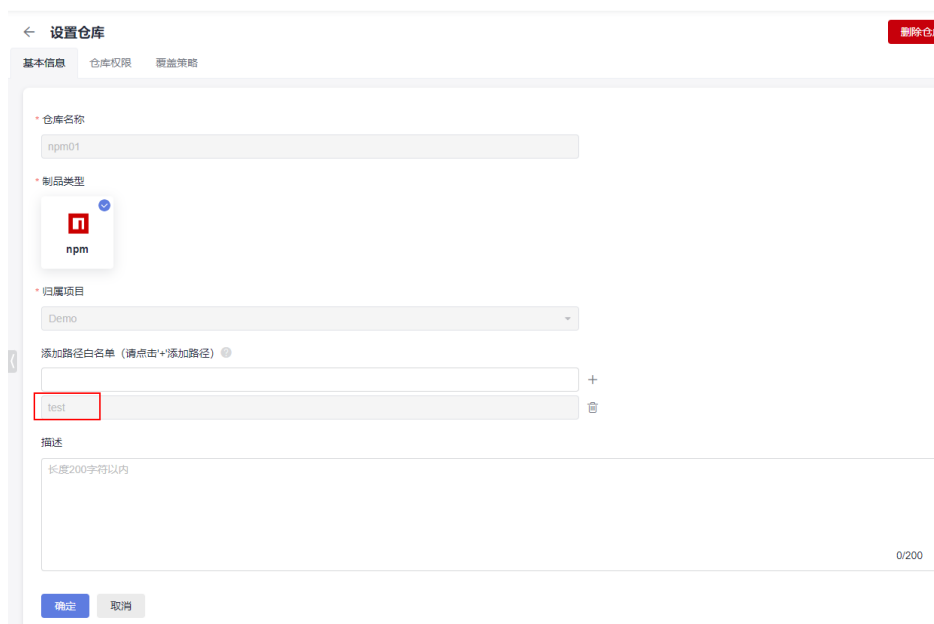
4. 在本地将下载的“npmrc”文件另存为“.npmrc”文件。

### 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。





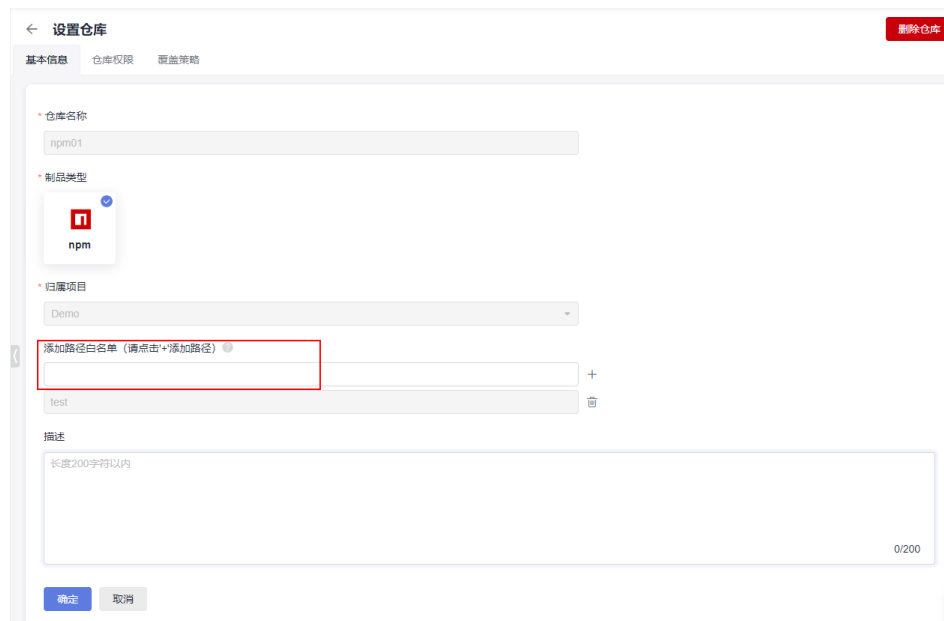
```

package.json 修改追溯 历史 783 Bytes Rep
1 {
2   "name": "test-vue-demo",
3   "description": "",
4   "version": "1.0.0",
5   "author": "",
6   "private": false,
7   "scripts": {
8     "dev": "cross-env NODE_ENV=development webpack-dev-server --open --hot",
9     "rm": "rm -rf node_modules",
10    "tar": "tar cvf vue_demo.tar *",
11    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules",
12    "all:prod": "npm run build && npm run rm && npm run tar"
13  },

```

### 说明

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。



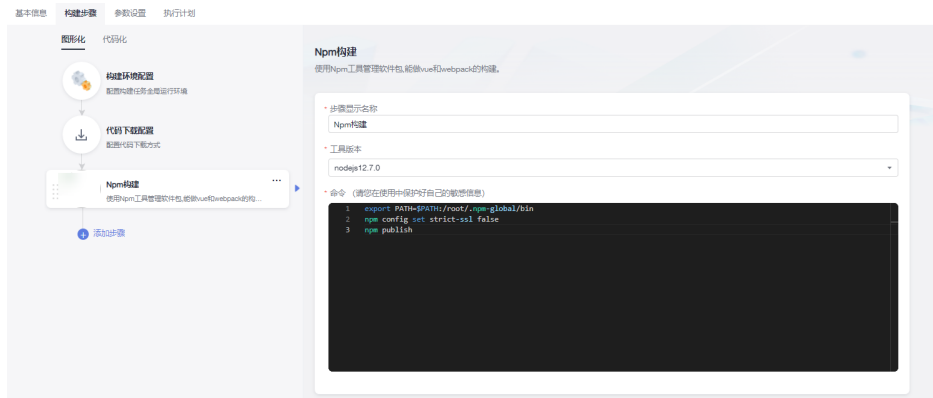
### 步骤3 配置并执行编译构建任务。

- 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”。
- 添加步骤“Npm构建”。



- 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"

```

package.json 修改追溯 历史
1  {
2     "name": "vue-demo",
3     "description": "",
4     "version": "1.0.0",
5     "author": "",
6     "private": false,
7     "scripts": {
8       "dev": "cross-env NODE_ENV=development webpack-dev-server --open --hot",
9       "rm": "rm -rf node_modules",
10      "tar": "tar cvf vue_demo.tar *",
11      "build": "cross-env NODE_ENV=production webpack --progress --hide-modules",
12      "all:prod": "npm run build && npm run rm && npm run tar"
13    },
14    "dependencies": {
15      "vue": "^2.2.1",
16      "@test/vue-demo": "^1.0.0"
17    }
  
```

### 步骤2 配置并执行编译构建任务。

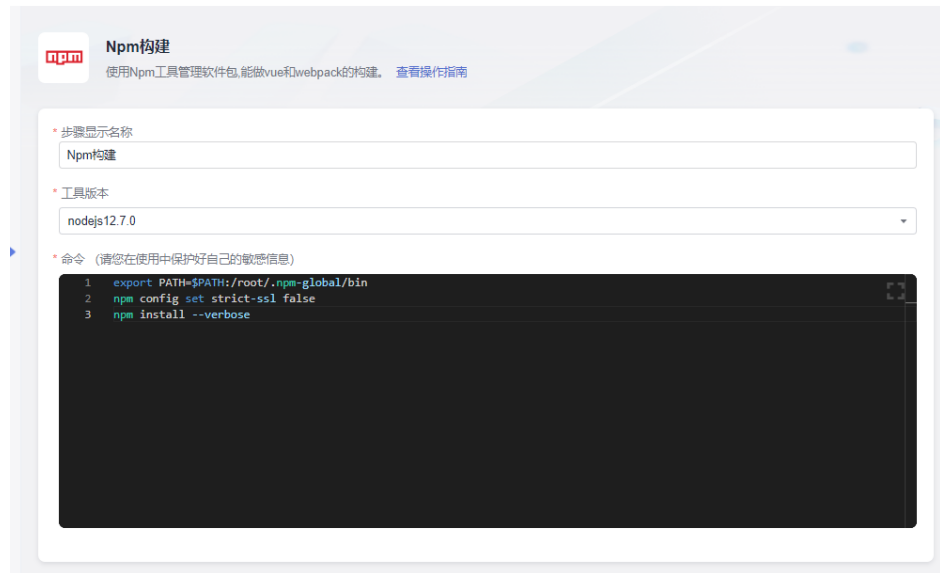
1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。  
在页面中选择“空白构建模板”，单击“下一步”。

## 2. 添加步骤“Npm构建”。



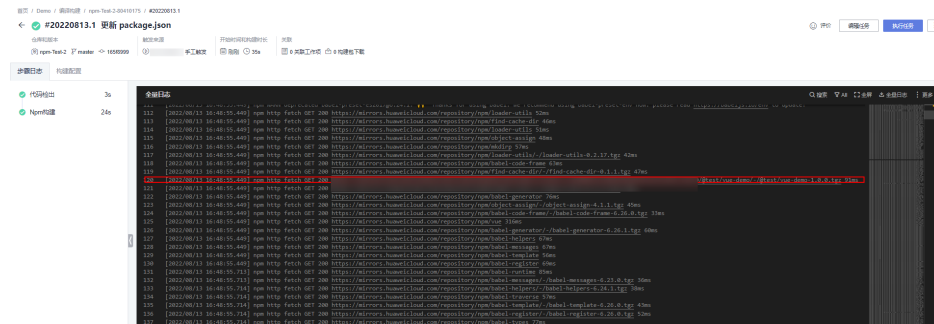
## 3. 编辑步骤“Npm构建”。

- 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
```

# 5 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[创建项目](#)。
- 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**中下载的配置文件中，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](#)）。本文中使用的仓库模板“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。

```
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。

```
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
```

# 6 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[创建项目](#)。
- 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。

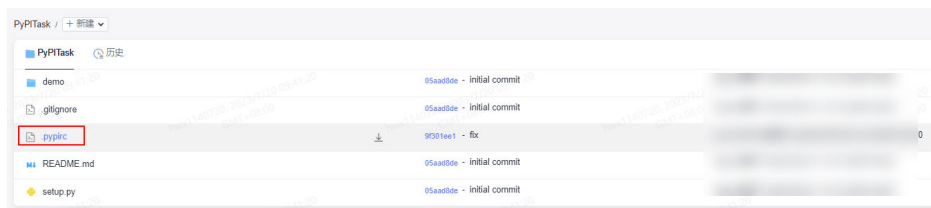
1. 登录制品仓库，进入PyPI私有依赖库。单击页面右侧“操作指导”。
2. 在弹框中找到“发布配置”，单击“下载配置文件”。



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/*
```

#### 说明

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。

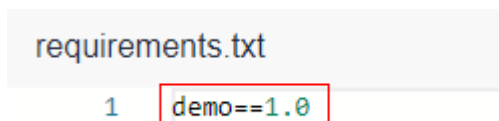
1. 进入PyPI私有依赖库，单击页面右侧“操作指导”。
2. 在弹框中找到“下载配置”，单击“下载配置文件”。



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

# 7 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。

```
#####rpm上传命令，yml仓库配置文件需要去掉上传rpm文件的命令#####  
curl -u user:password -X PUT https://repoUrl/component/version/ -T localFile
```

“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:

```
curl -u [redacted]@devrepo.devcloud.huaweicloud.com: [redacted] -X PUT  
https://devrepo.devcloud.huaweicloud.com/artgalaxy/[redacted]/rpm_1/hello/0.17.2/ -T hello-0.17.2-54.x86_64.rpm
```

**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。

```
[redacted] yum.repos.d# pwd  
/etc/yum.repos.d  
[redacted] yum.repos.d# ll  
total 20  
-rw-r--r-- 1 [redacted] 737 Mar 12 11:04 [redacted]-n-north [redacted]-rpm_0.repo  
-rw-r--r-- 1 [redacted] 235 Jan 25 23:00 [redacted]  
-rw-r--r-- 1 [redacted] 186 Jan 25 22:59 [redacted]  
-rw-r--r-- 1 [redacted] 234 Jan 25 23:00 [redacted]  
drwxr-xr-x 4 [redacted] 4096 Dec 18 17:18 tmp
```

**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

# 8 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上传命令-----##  
curl -u <USERNAME>:<PASSWORD> -X PUT "https://<devrepo.devcloud>:<ARTGALAXY>/artgalaxy/<ARTGALAXY>  
<REPOURL>/<DEBIAN_PACKAGE_NAME>.<DEBIAN_PACKAGE_NAME>.deb.distribution=<DISTRIBUTION>.<DEBIAN_PACKAGE_NAME>.deb.component=<COMPONENT>.<DEBIAN_PACKAGE_NAME>.deb.architecture=<ARCHITECTURE>" -T <PATH_TO_FILE>  
<FILE>
```

“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:

```
##-----debian上传命令-----##  
curl -u <USERNAME>:<PASSWORD> -X PUT "https://<devrepo.devcloud>:<ARTGALAXY>/artgalaxy/<ARTGALAXY>  
<REPOURL>/a2jmidid_8_dfsg0-1_amd64.deb.distribution=trusty.<DEBIAN_PACKAGE_NAME>.deb.component=main.<DEBIAN_PACKAGE_NAME>.deb.architecture=amd64" -T /root/a2jmidid_8_dfsg0-1_amd64.deb
```

**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pgp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”。

```
root@szvphispre01726:/debian# gpg --import artifactory.gpg.public
gpg: key 8595129: public key "devcloud-artifact (artifact debian key pair) <devcloud-artifact@huawei.com>" imported
gpg: Total number processed: 1
gpg:      imported: 1
```

## 步骤3 apt导入公钥。

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

```
root@szvphispre01726:/# gpg --export --armor 8595129 | apt-key add -
OK
root@szvphispre01726:/#
```

## 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的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
```

```
root@szvphispre01726:/tmp# apt-get update
Get:1 https://devrepo.devcloud.cn/artifactory/secure/trusty InRelease [2,212 B]
Fetched 2,212 B in 0s (8,201 B/s)
Reading package lists... Done
W: Usage of apt_auth.conf(5) should be preferred over embedding login information directly in the sources.list(5) entry for 'https://devrepo.devcloud.cn/artifactory/secure/trusty'
root@szvphispre01726:/tmp#
```

## 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
```

### 说明

<PACKAGE>获取方法

- 下载Debian组件的Packages源数据，以a2jmidid包为例：

The screenshot shows a web-based package manager interface. On the left, a tree view displays the package structure: 'All' > 'debian01' > 'dists' > 'demo' > '0ad-dbg\_0.0.17-1\_amd64.deb'. On the right, a '配置信息' (Configuration Information) panel shows the following details:

Distribution	demo
Component	huawei
Architecture	arm

----结束