



软件开发生产线 CodeArts

制品仓库快速入门

天翼云科技有限公司

目 录

1 快速上手软件发布库	3
2 快速上手私有依赖库	7
3 通过编译构建任务发布/获取 Maven 私有组件	9
4 通过编译构建任务发布/获取 Npm 私有组件	14
5 通过编译构建任务发布/获取 Go 私有组件	20
6 通过编译构建任务发布/获取 PyPI 私有组件	26
7 通过 Linux 命令行上传/获取 Rpm 私有组件	30
8 通过 Linux 命令行上传/获取 Debian 私有组件	33

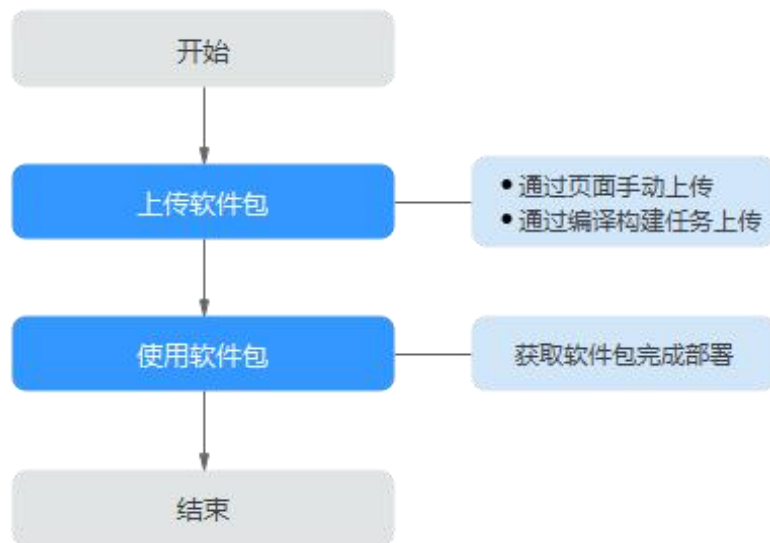
1 快速上手软件发布库

本文档向您介绍软件发布库的基本操作流程，帮助您快速建立对软件发布库的整体印象。

使用软件发布库前，需要已有可用的项目。如果没有项目，请先新建项目。

软件发布库的主要使用流程如图 1-1 所示：

图 1-1 软件发布库使用流程



通过软件发布库页面手动上传软件包

步骤 1 登录软件开发生产线，单击页面功能菜单区“服务 > 制品仓库”，选择“软件发布库”页签。

步骤 2 进入与已创建项目同名的仓库，单击页面右上方“上传”。

步骤 3 在弹窗中选中待上传的软件包，单击“打开”。

---结束

通过编译构建任务发布软件包到软件发布库

以 Maven 构建任务为例，介绍如何通过编译构建任务发布软件包到软件发布库。

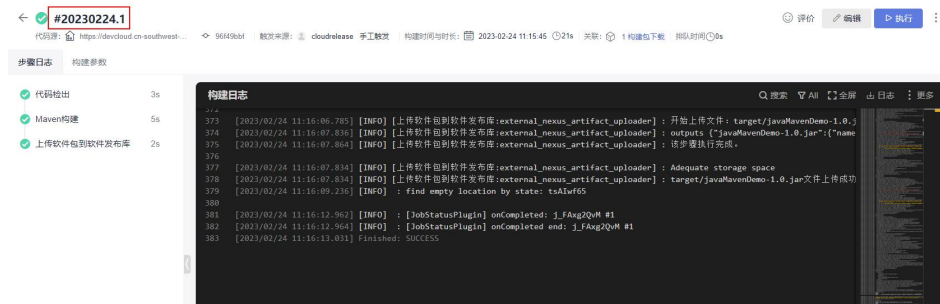
步骤 1 准备代码仓库。

1. 登录软件开发生产线，进入已创建的项目。
2. 进入代码托管服务，创建 Maven 类型代码仓库。本文中使用的仓库模板“Java Web Demo”创建代码仓库。

步骤 2 配置并执行编译构建任务。

1. 进入代码仓库，单击页面右上角“设置构建”，页面跳转至“新建构建任务”页面。
在页面中选择“Maven”，单击“下一步”。
2. 根据需要编辑构建步骤，本文中采用模板中的默认配置值。
3. 单击“新建并执行”，启动构建任务执行。
待任务执行成功时，记录构建任务页面左上角“#”之后的数字串，如图 1-2 所示。

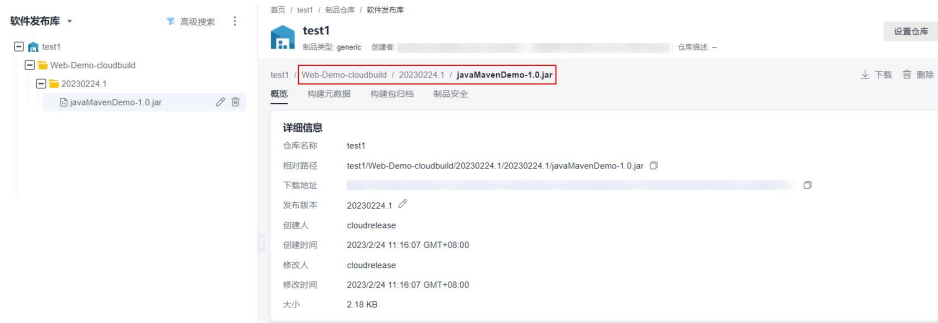
图 1-2 编译构建任务



步骤 3 查看软件包。

1. 进入项目下的软件发布库。
2. 进入与构建任务名称同名的文件夹。
3. 找到与构建任务页面中记录的数字串同名的文件夹，进入该文件夹即可找到生成的软件包，如图 1-3 所示。

图 1-3 查看软件包



说明

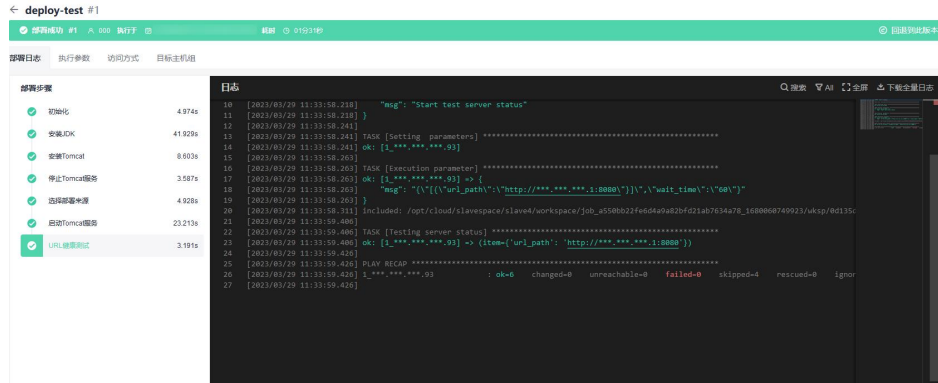
若在编译构建任务的步骤“上传软件包到软件发布库”中设置了“发布版本号”，软件包在将保存在与发布版本号同名的文件夹中。

---结束

从软件发布库获取软件包完成部署

以[通过编译构建任务发布软件包到软件发布库](#)中发布的软件包为例，介绍如何从软件发布库中获取软件包完成部署操作。

- 步骤 1 进入软件开发生产线首页，单击目标项目名称，进入项目。
- 步骤 2 单击菜单“设置 > 通用设置 > 基础资源管理”，默认进入“主机集群”页面。
- 步骤 3 单击“新建主机集群”，填写相关信息后，单击“保存”，完成主机集群的创建并在主机集群中新增目标主机。
- 步骤 4 进入部署服务，单击“新建应用”。根据实际情况配置部署任务信息，本文使用的主要配置如下：
 - 部署模板选择“Tomcat 应用部署”。
 - 部署步骤“选择部署来源”中，根据[通过编译构建任务发布软件包到软件发布库](#)中的存放路径，选择软件包。
- 步骤 5 在“环境管理”页签中，新建环境并将主机添加到环境中。
- 步骤 6 单击“保存并部署”开始部署应用，当页面显示“部署成功”时，说明部署应用成功从软件发布库中获取软件包、并部署到目标主机上。



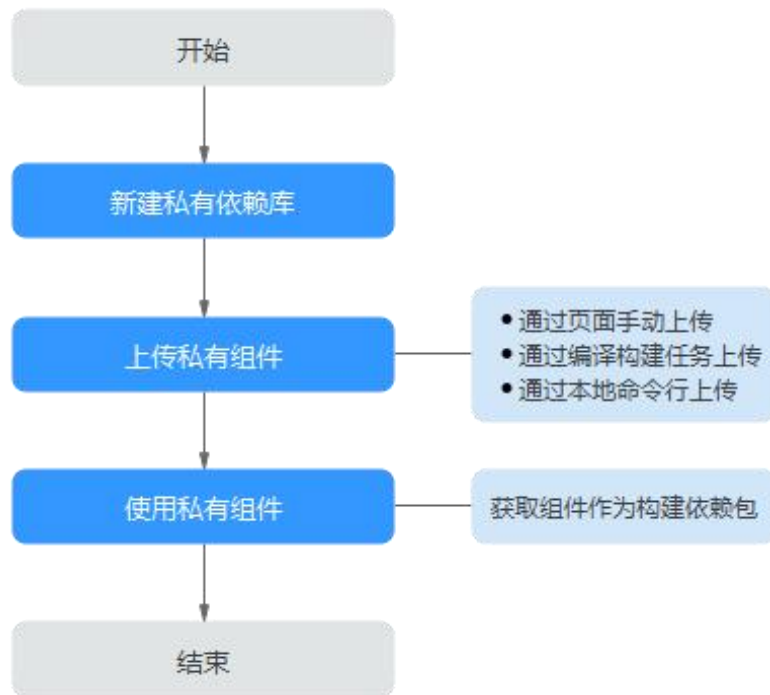
----结束

2 快速上手私有依赖库

本文档向您介绍私有依赖库的基本操作流程，帮助您快速建立对私有依赖库的整体印象。

私有依赖库的主要使用流程如图 2-1 所示：

图 2-1 私有依赖库使用流程



新建私有依赖库

- 步骤 1** 登录软件开发生产线首页，单击页面功能菜单区“服务 > 制品仓库”，选择“私有依赖库”页签。
- 步骤 2** 单击页面左侧“新建制品仓库”。
- 步骤 3** 配置仓库基本信息，单击“确定”按钮。

---结束

通过私有依赖库页面上传私有组件

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

通过编译构建任务上传/获取私有组件

软件开发生产线支持通过编译构建任务上传 Maven、Npm、Go、PyPI 组件到私有依赖库中，并支持从私有依赖库中获取组件作为构建依赖包。

详细操作请参考：

- [通过编译构建任务发布/获取 Maven 私有组件](#)
- [通过编译构建任务发布/获取 Npm 私有组件](#)
- [通过编译构建任务发布/获取 Go 私有组件](#)
- [通过编译构建任务发布/获取 PyPI 私有组件](#)

通过 Linux 命令行上传/获取 Rpm 私有组件

通过 Linux 命令行，可以上传 Rpm、Debian 私有组件至私有依赖库，也可以从私有依赖库中下载 Rpm 组件。

详细操作请参考[通过 Linux 命令行上传/获取 Rpm 私有组件](#)。

详细操作请参考[通过 Linux 命令行上传/获取 Debian 私有组件](#)。

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

1. 已有可用项目。如果没有项目，请先新建项目。
2. 请添加当前账号对当前私有库的权限。
3. 已创建 Maven 格式私有依赖库，并与项目关联。

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。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.██████████.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
```

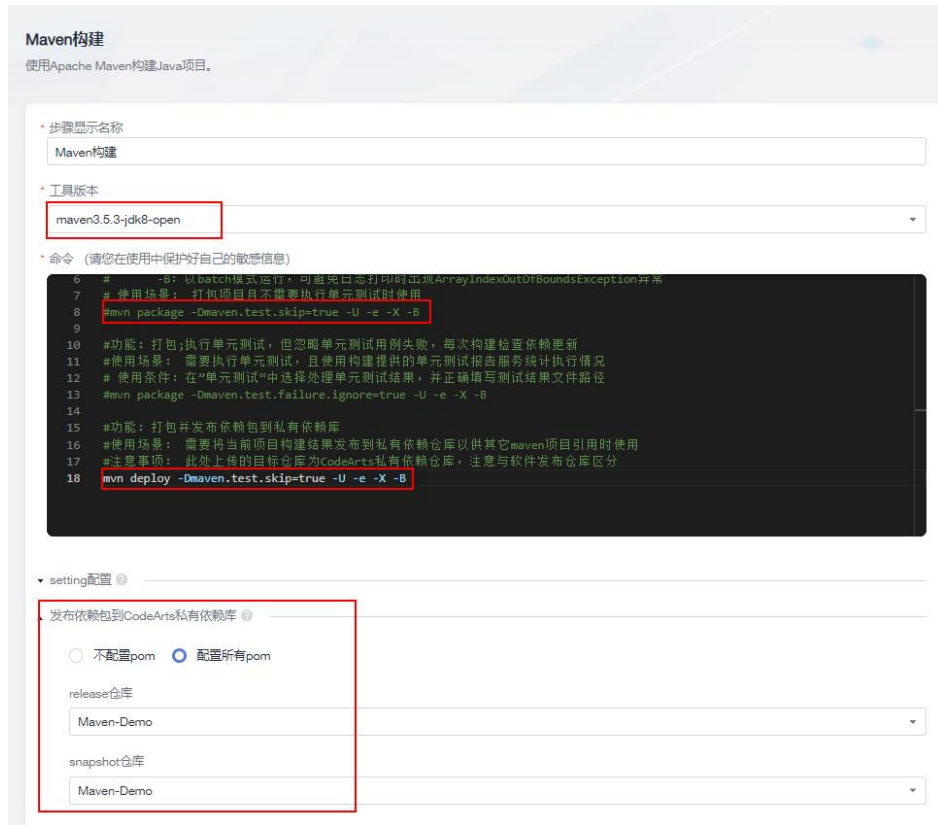
找到以下命令行，在命令行前添加#。

```
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。

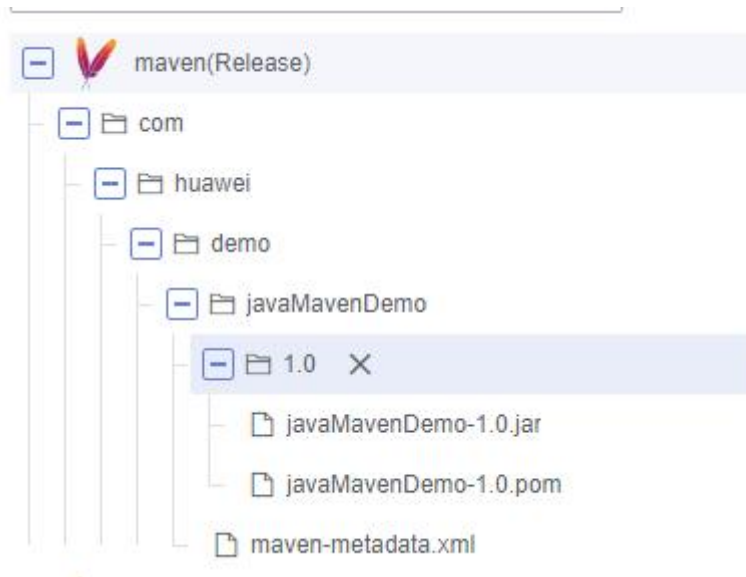
📖 说明

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。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）。

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。

---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

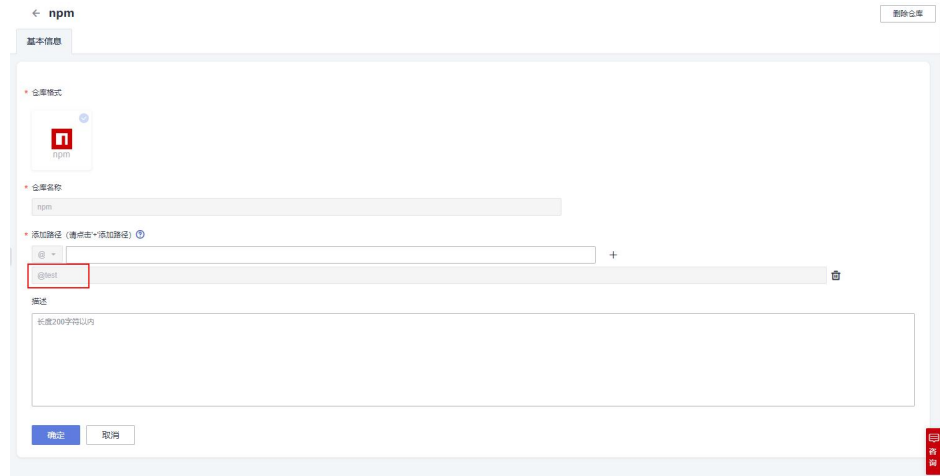
1. 已有可用项目。如果没有项目，请先新建项目。
2. 已创建 Npm 格式私有依赖库。
3. 请添加当前账号对当前私有库的权限。


发布私有组件到 Npm 私有依赖库

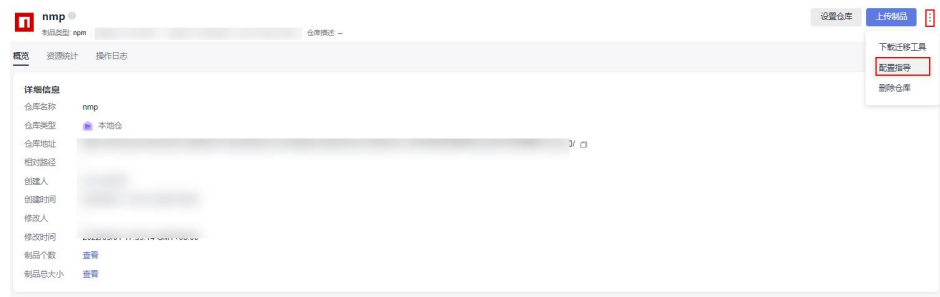
步骤 1 下载私有依赖库配置文件。

1. 登录软件开发生产线，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，在下拉栏中单击“配置指导”。



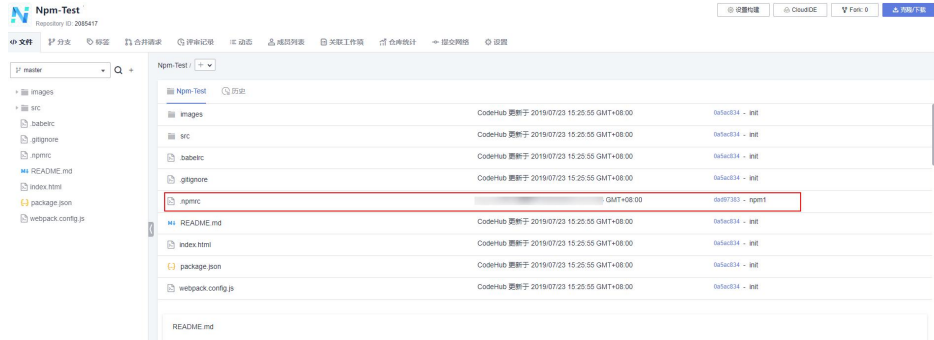
3. 在弹框中单击“下载配置文件”。



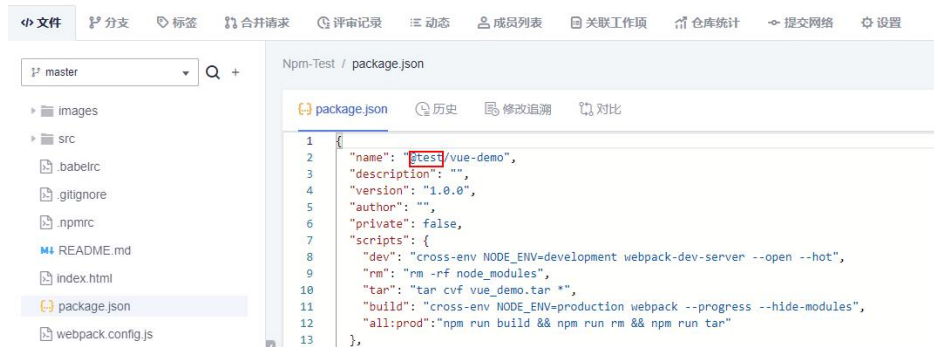
4. 在本地将下载的“npmrc”文件另存为“.npmrc”文件。

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。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。



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。

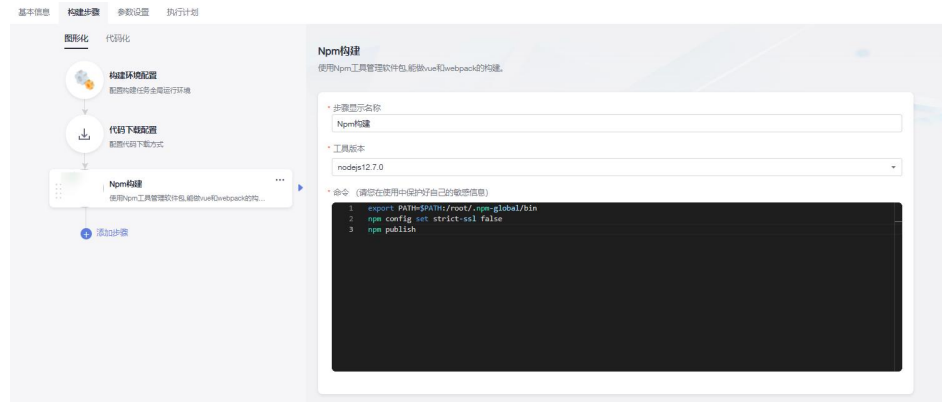
步骤 3 配置并执行编译构建任务。

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

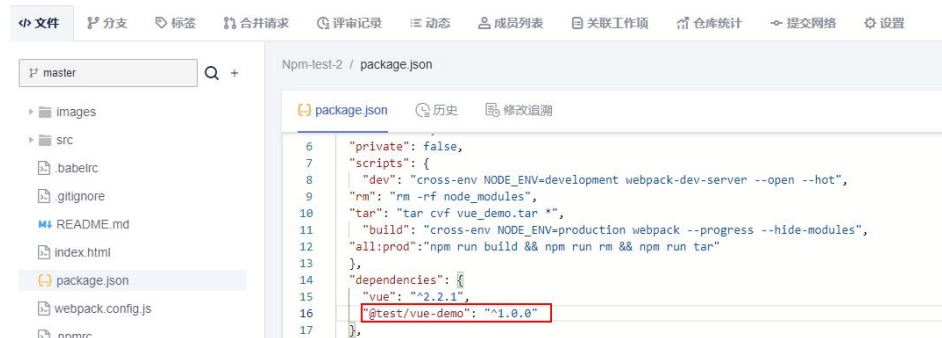
从 Npm 私有依赖库获取依赖包

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。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。

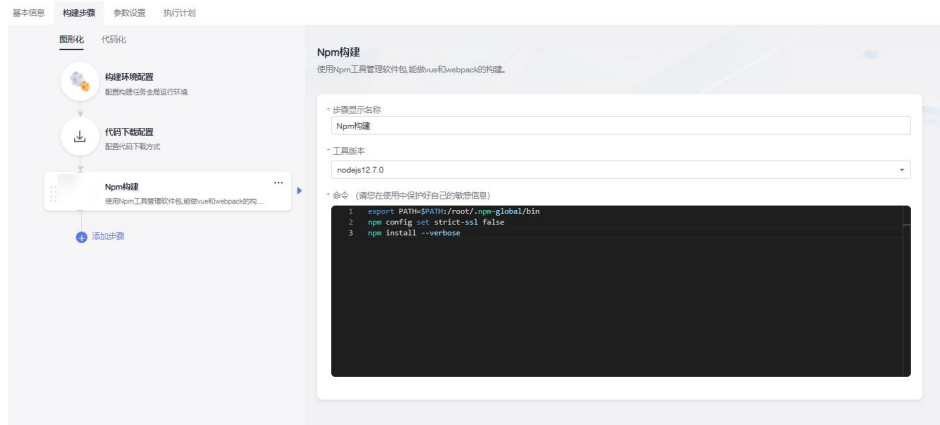
1. 在代码仓库中，单击页面右上角“设置构建”，页面跳转至“新建编译构建任务”页面。
在页面中选择“空白构建模板”，单击“下一步”。
2. 添加步骤“Npm 构建”。



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。

----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 @scope/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。

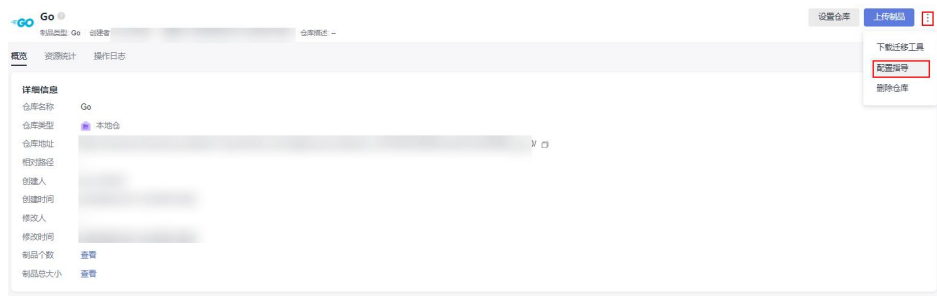
前提条件

1. 已有可用项目。如果没有项目，请先新建项目。
2. 已创建 Go 格式私有依赖库。
3. 请添加当前账号对当前私有库的权限。

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



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。本文中使用的仓库模板“GoWebDemo”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，本文中选择了“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。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“<module name>”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。

---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。

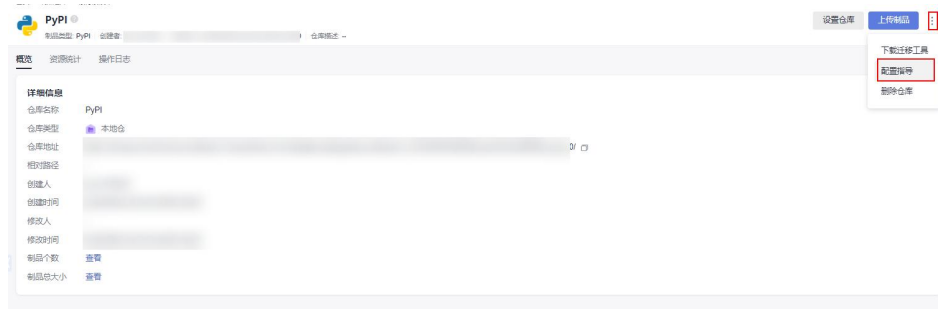
前提条件

1. 已有可用项目。如果没有项目，请先新建项目。
2. 已创建 PyPI 格式私有依赖库。
3. 请添加当前账号对当前私有库的权限。

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



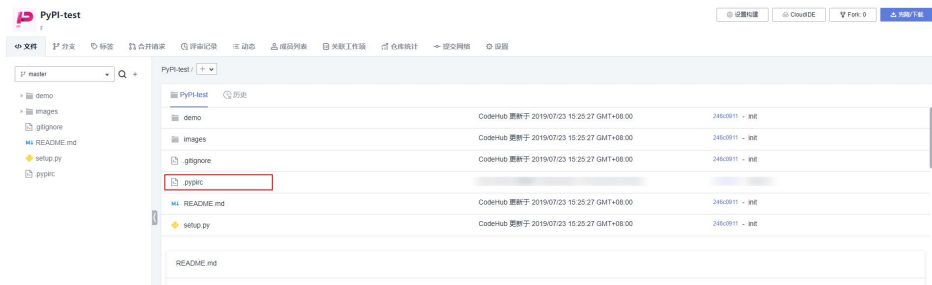
2. 在弹框中找到，单击“下载配置文件”。



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。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“Setuptools”。



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“私有依赖库使用配置”。
2. 在弹框中找到，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。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
```

requirements.txt

```
1 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“Setuptools”。



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。

----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

1. 已有可用的 Rpm 组件。
2. 已有可连通公网的 Linux 系统主机。
3. 已创建 Rpm 格式私有依赖库
4. 请添加当前账号对当前私有库的权限。

发布私有组件到 Rpm 私有依赖库

步骤 1 登录软件开发生产线，进入 Rpm 私有依赖库。单击页面右侧，在下拉栏中单击“配置指导”。



步骤 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。

```
name: rpm_1({{component}})
baseurl: https://devrepo.{{component}}/rpm_1({{component}})
enablegpg: true
gpgcheck: 1
# rpm 上传命令, rpm 仓库配置文件中包含上传 rpm 文件的命令
curl -u {{username}}:{{password}} -X PUT https://devrepo.{{component}}/rpm_1({{component}}/{{component}}/ -T (localFile)
```

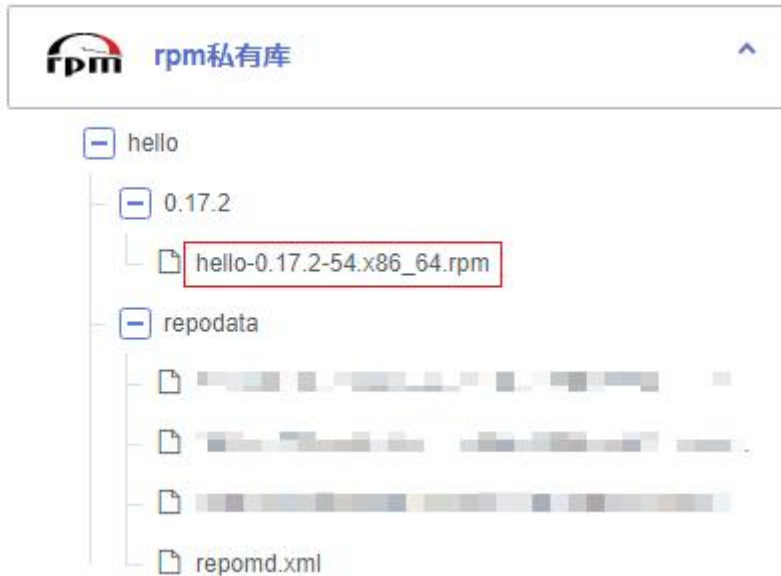
“component”、“version”、“localFile” 来源于待上传的 Rpm 组件。以组件 “hello-0.17.2-54.x86_64.rpm” 为例。

- 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 {{username}}:{{password}} -X PUT https://devrepo.{{component}}/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。

```
[ yum.repos.d]# pwd
/etc/yum.repos.d
[ yum.repos.d]# ll
total 20
-rw-r--r-- 1 737 Mar 12 11:04 cn-north _rpm_0.repo
-rw-r--r-- 1 235 Jan 25 23:00
-rw-r--r-- 1 186 Jan 25 22:59
-rw-r--r-- 1 234 Jan 25 23:00
drwxr-xr-x 4 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

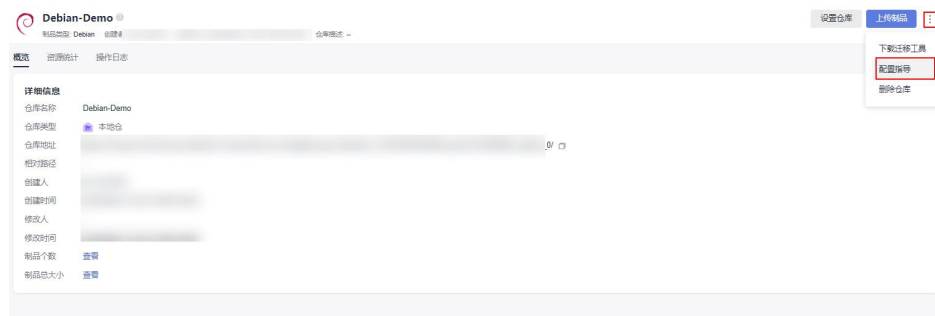
本文档介绍如何通过 Linux 命令行上传私有组件到 Debian 私有依赖库、如何从 Debian 私有依赖库获取依赖包。

前提条件

1. 已有可用的 Debian 组件。
2. 已有可连通公网的 Linux 系统主机。
3. 已创建 Debian 格式私有依赖库。
4. 请添加当前账号对当前私有库的权限。

发布私有组件到 Debian 私有依赖库

步骤 1 登录软件开发生产线，进入 Debian 私有依赖库。单击页面右侧，在下拉栏中单击“配置指导”。



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://<repoUrl>/<DEBIAN_PACKAGE_NAME>;deb.distribution=<DISTRIBUTION>;deb.component=<COMPONENT>;deb.architecture=<ARCHITECTURE>" -T <PATH_TO_FILE>
```

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

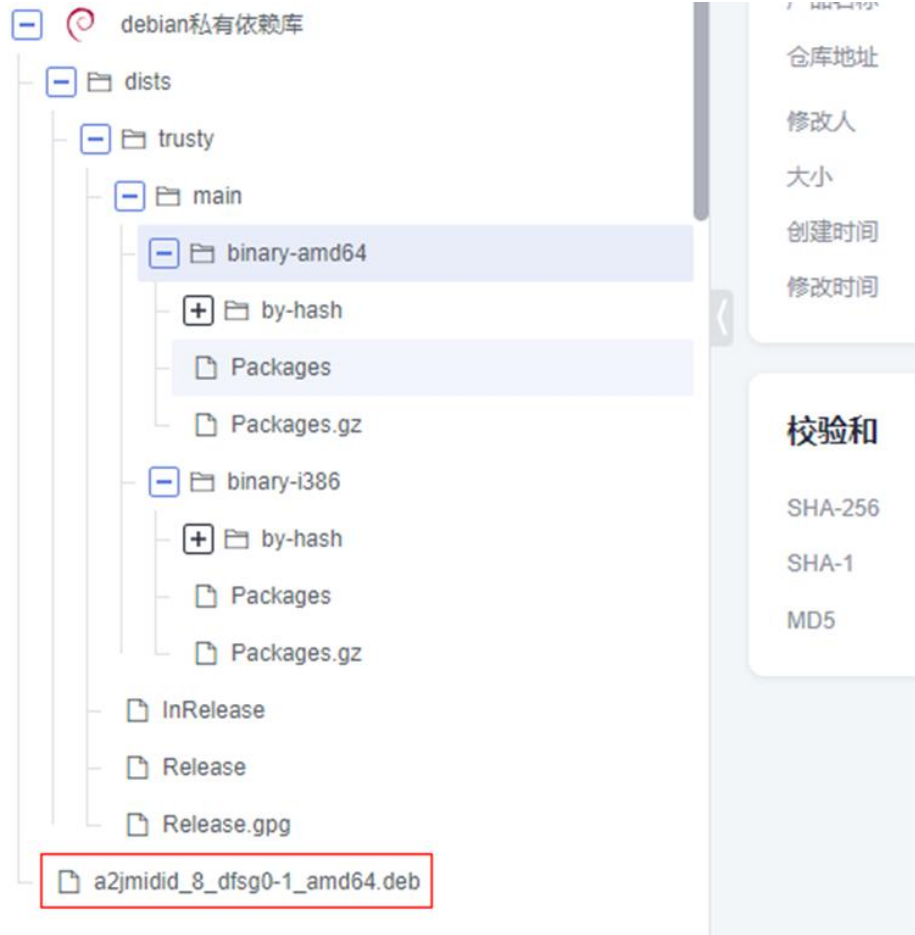
以组件“a2jmidid_8_dfsg0-1_amd64.deb”为例。

- 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://<repoUrl>/a2jmidid_8_dfsg0-1_amd64.deb;deb.distribution=trusty;deb.component=main;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。

私有依赖库使用配置

发布配置

选择依赖管理工具

apt

下载配置

使用前请确保您已安装 Linux 系统，apt 仓库源配置文件为用户根目录下的：`/etc/apt/sources.list`

公共配置

下载软件包前将获取的公钥信息保存到 public.asc 文件中，并添加到系统密码列表。 [下载公钥](#)

```
1 gpg --import <PUBLIC_KEY>
2 gpg --list-signatures
3 gpg --export --armor <SIG_ID> | apt-key add -
```

上传命令

运行以下命令上传软件包：

```
1 curl -k -u "<USERNAME>:<PASSWORD>" -X PUT "https://"
```

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”。

```
root@szvphispre01726:/debian# gpg --import artifactory.gpg.public
gpg: key 22F2232222222222: public key "devcloud-artifact (artifact debian key pair) <XXXXXXXXXXXXXXXXXXXX@XXXXXXXXXX>" 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 XXXXXXXXXXXXXXXXXXXX | 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 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.com/artifactory/XXXXXXXXXXXXXXXXXXXX/ trusty InRelease [2,212 B]
Fetched 2,212 B in 0s (8,201 B/s)
Reading package lists... Done
W: Issues of apt.auth.conf(5) should be preferred over embedding login information directly in the sources.list(5) entry for 'https://devrepo.devcloud.com/artifactory/XXXXXXXXXXXXXXXXXXXX/Ar
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：



---结束