CODING DevOps

制品库

腾讯云

文档目录

制品库
快速开始
基础操作

最近更新时间：2023-09-11 16:05:25

在正式进行制品库的操作前，您可以参考以下内容进行功能初始化。下述的步骤与准备并非必须选项，可以按照实际需求有选择性地阅读。

创建项目

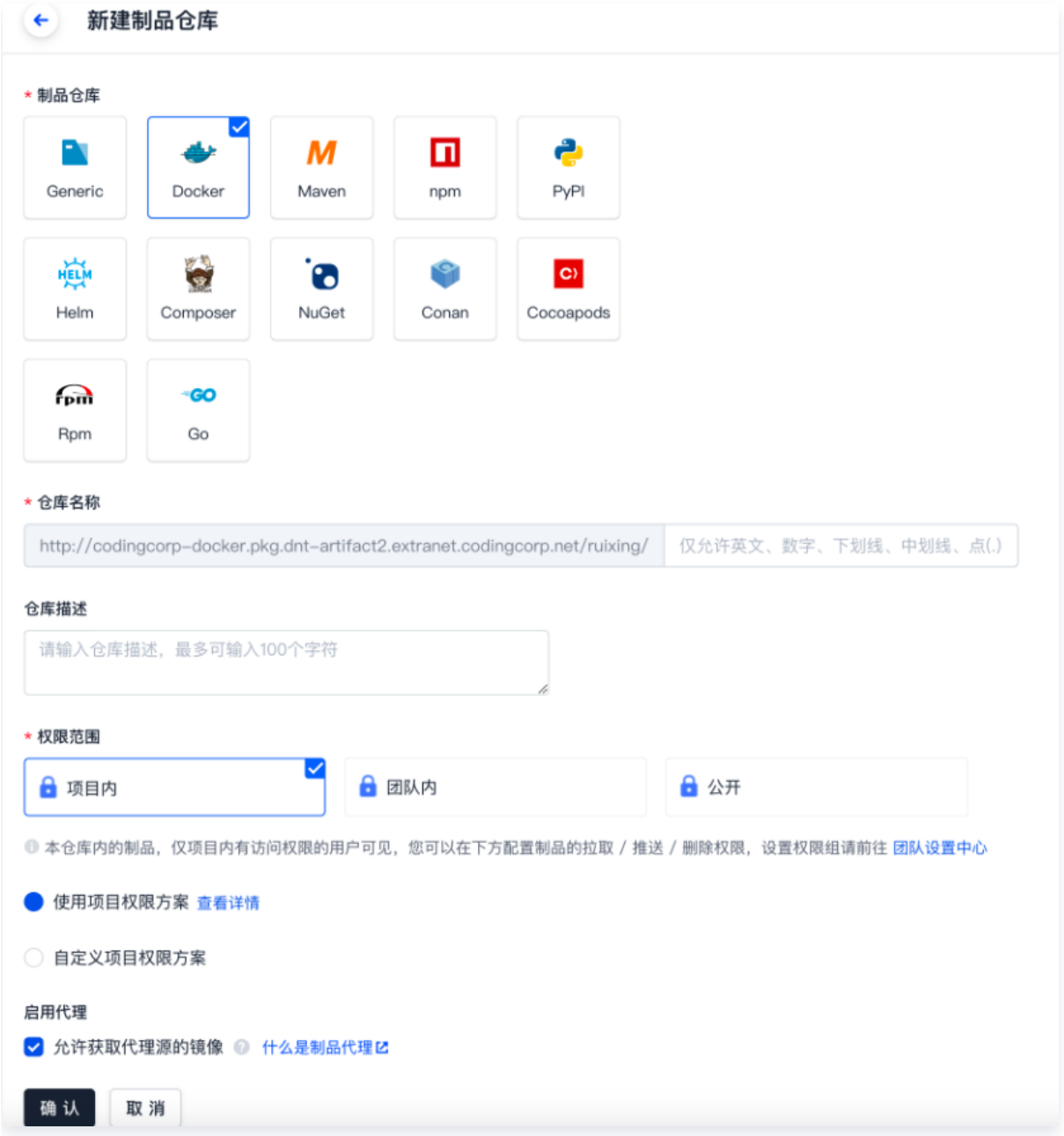
使用制品仓库前需新建项目，选择 DevOps 项目。具体操作请参见 创建项目 。
项目创建完成后，可进入项目，在左侧菜单中进入制品仓库功能。若功能被隐藏，则需点击左下角的**项目设置 > 项目与成员 > 菜单管理**开启制品库功能。

新建制品仓库

进入制品仓库后，单击右上角**创建制品仓库**并选择仓库类型。

- 提供一个仓库名称。
- 选择制品仓库类型。
- 权限范围：决定当前创建的制品仓库对不同类型角色的操作权限，默认将对当前项目成员开放**推送**和**拉取**操作。

> **说明：**
> 团队实名制后才能选择**公开**制品权限范围，由团队负责人或管理员前往团队设置中心进行实名认证。



创建制品仓库后，您可以在左侧文档目录列表参考各制品类型的快速开始。

删除制品仓库

单击任意仓库中的设置仓库，在基本信息中进行删除操作。

制品筛选

在制品搜索框中输入关键词或调整搜索条件可以快捷查找所需要的制品，搜索完成后还可以将搜索条件另存为筛选器方便下次查询。

Generic

最近更新时间：2025-04-14 17:03:52

该文档介绍如何将通用文件类型的制品存储在 CODING 制品库中。其内容包括创建制品库、推送、拉取和删除制品。

创建制品仓库

参见 基础操作，在项目中创建 Generic 类型制品仓库。

推送制品

> **说明：**
> 高级版团队的 Generic 类型制品的下载限制容量为每日 50G。

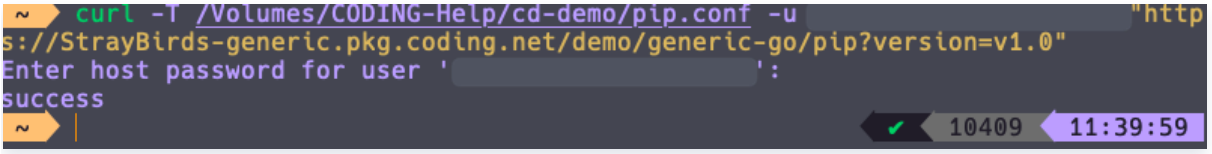
支持以两种方式推送制品：

- 命令行方式
- 通过页面直接上传

通过命令行上传

输入本地路径、制品名称、制品版本后将自动生成推送命令，复制后直接在终端中运行即可。制品上传过程支持断点续传，按照图示的提示命令安装插件后再进行推送。

输入服务密码，认证后完成推送。



通过页面直接上传

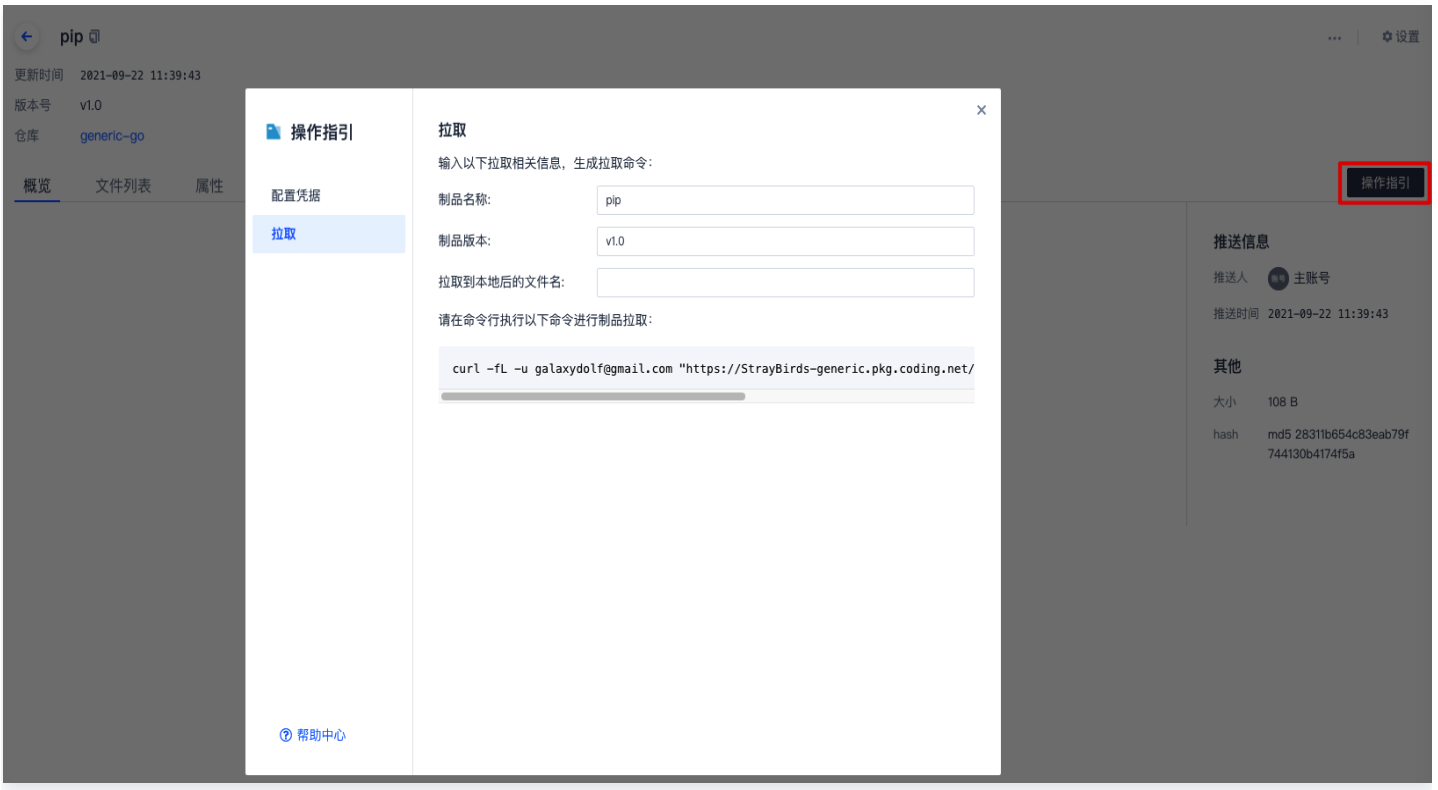
直接将制品文件拖拽至页面按钮也可以完成上传操作。

查看制品

上传成功后，在包列表处即可查看版本号信息。

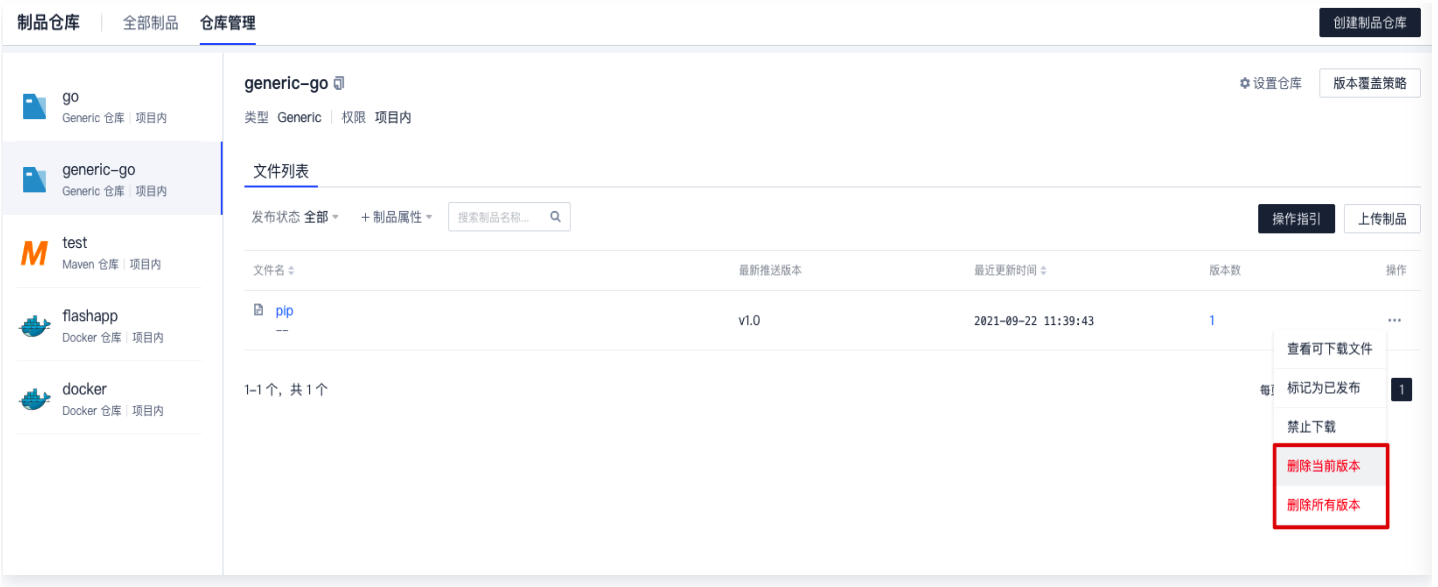
拉取制品

单击已上传制品的**操作指引**，输入版本号、名称与文件名后自动生成命令，在终端中运行命令进行拉取：



删除制品

单击页面上的删除版本按钮进行制品版本删除。



若需删除制品仓库，单击 基础操作 进行了解。

Docker

最近更新时间：2023-09-13 10:16:13

本文档介绍如何将 Docker 镜像存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含如何进行镜像制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Docker
> - 已创建项目和制品仓库，具体操作可参见 基础操作 指引中的创建项目、新建制品库。
> - 制品仓库选择 Docker 类型。

制作镜像（可选阅读）

本章节提供两种方法快速创建一个 Demo Docker 镜像，若已熟悉 Docker 镜像制作可以跳过本节。

方法一：本地制作镜像

1. 在本地任意目录创建文件，名称为 Dockerfile，并写入以下内容：

```
FROM coding-public-docker.pkg.coding.net/public/docker/nodejs:12
```

2. 在所在目录中调出终端，运行命令构建镜像。

```
docker build -t hello-world .
```

镜像制作成功，tag 默认为 `hello-world:latest` 。您可以按照《Docker——官方手册》自定义 tag 内容，格式为 `<镜像名>:<版本>` ，本章节不再展开。

```
/Volumes/CODING/Docker-learning docker build -t hello-world .
Sending build context to Docker daemon  2.048kB
Step 1/1 : FROM fanvinga/docker-ssrmu
 ---> 90ec15c0d38d
Successfully built 90ec15c0d38d
Successfully tagged hello-world:latest
```

方法二：从 Docker Hub 拉取镜像

1. 在终端中直接执行命令拉取镜像。

```
docker pull hello-world
```

2. 执行命令，查看已拉取的镜像。

```
docker images
```

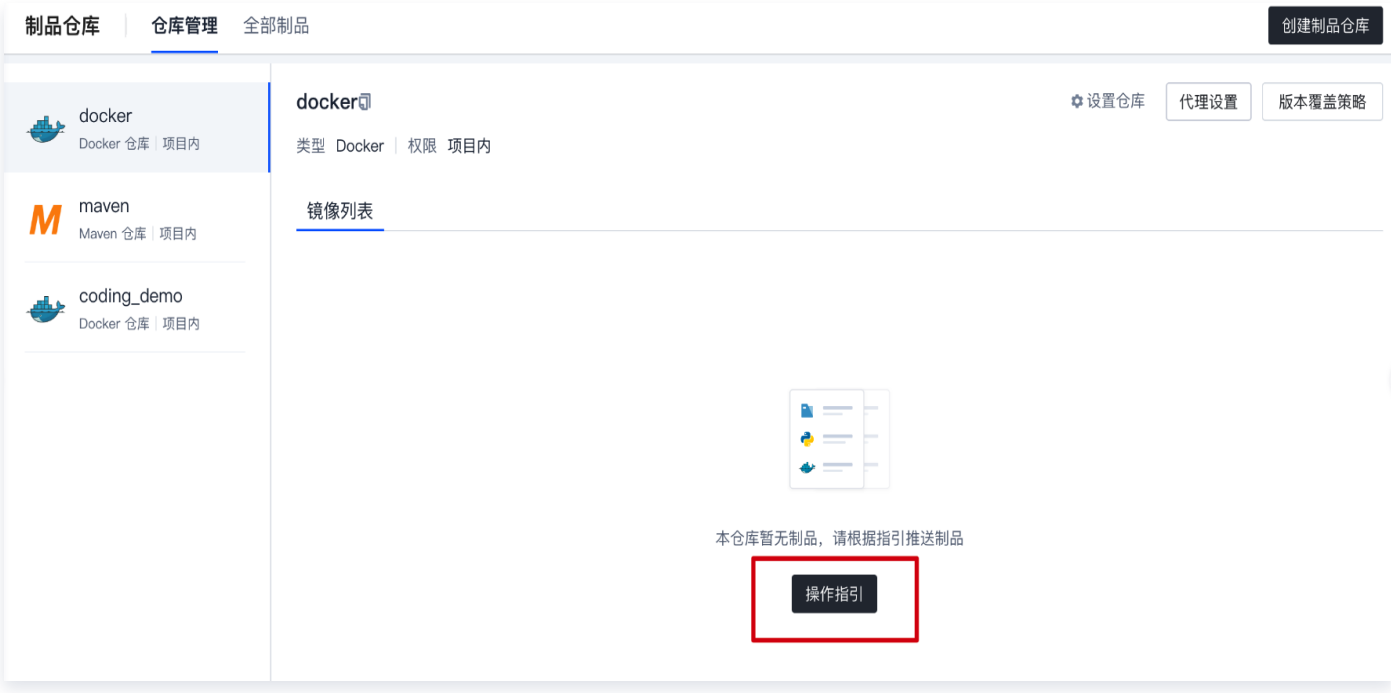
配置认证信息

当您已在本地完成制品编译后，就可以将制品推送至远端制品仓库。推送之前需在本地配置远端仓库的认证信息。

访问令牌

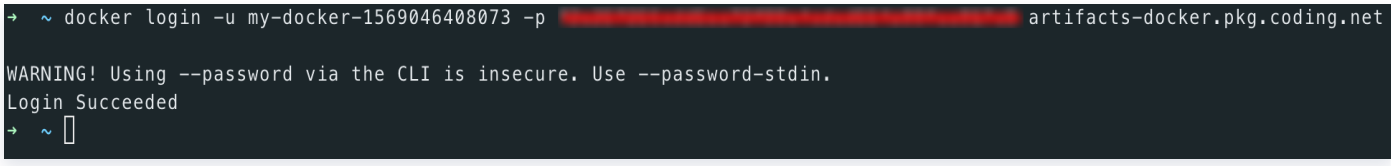
推荐使用**访问令牌**生成认证的配置信息。

1. 在仓库页面单击**操作指引**。



2. 输入账号的登录密码或两步验证码后确认，复制生成的命令。

3. 在本地 Docker 环境中的命令行中粘贴生成后的命令并执行，即可完成认证。



推送镜像

> **说明：**
> 希望自动化推送 Docker 镜像？请参见如何在持续集成中使用 制品库插件 。

以下命令行仅作示例，命令行会因项目差异而发生改变，请复制项目内制品仓库中直接生成的命令。

1. 给 上文 拉取至本地的 hello-world 镜像打标签。

```
docker tag hello-world straybirds-docker.pkg.coding.net/coding-demo/coding-demo/hello-world
```

2. 推送您的 docker 镜像至制品仓库。

```
docker push straybirds-docker.pkg.coding.net/coding-demo/coding-demo/hello-world
```

成功推送后将看到如下内容。

```
/Volumes/CODING/Docker-learning docker push straybirds-docker.pkg.coding.net/coding-demo/coding-demo/hello-world
The push refers to repository [straybirds-docker.pkg.coding.net/coding-demo/coding-demo/hello-world]
cc471758abdf: Pushed
a25d159becc3: Pushed
06cfd7503045: Pushed
beee9f30bc1f: Pushed
latest: digest: sha256:44d5f1eb23a6227332267f15811f8833ecf65f3a52dd57844d30bd4353401f98 size: 1157
```

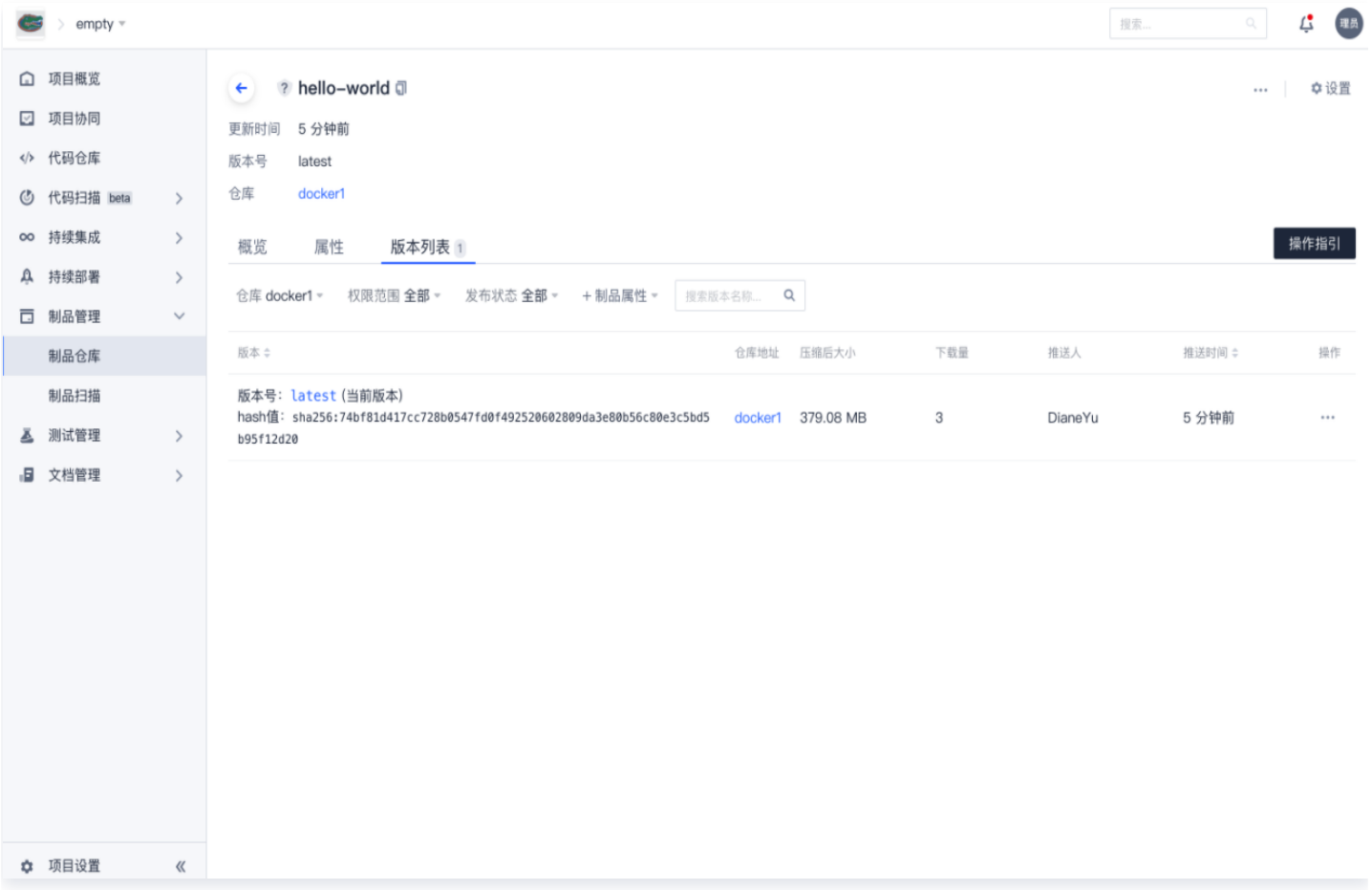
上述操作命令，均会直接显示在操作指引中，可输入替换值后复制命令。

查看镜像

推送完毕后，左侧菜单处的项目概览会在项目内广播推送动态。

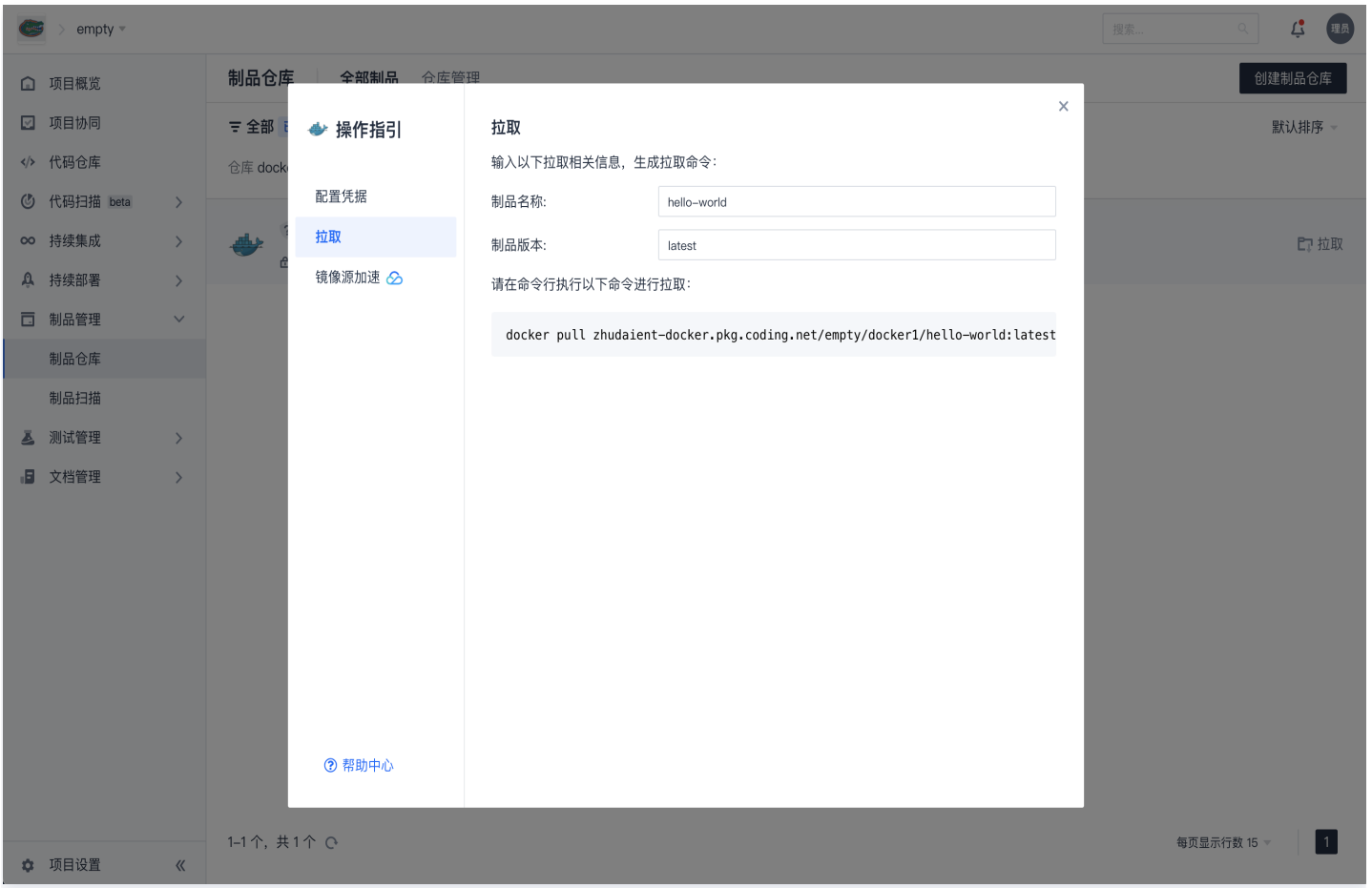
项目的制品列表中，可以看到推送的 **hello-world** 镜像。

单击镜像名，可以在右侧栏查看到该包的完整信息，内含概览、指引、属性、版本列表等信息。



拉取镜像

使用 `docker pull` 命令可以拉取在 CODING 制品库中托管的 Docker 镜像。指引页面会自动生成相对应的拉取命令。


empty
项目概览
项目协同
代码仓库
代码扫描 beta
持续集成
持续部署
制品管理
制品仓库
制品扫描
测试管理
文档管理
项目设置
制品仓库
全部制品
仓库管理
创建制品仓库
默认排序
拉取
操作指引
配置凭据
拉取
镜像源加速
帮助中心
拉取
输入以下拉取相关信息，生成拉取命令：
制品名称:
hello-world
制品版本:
latest
请在命令行执行以下命令进行拉取：
docker pull zhudaient-docker.pkg.coding.net/empty/docker1/hello-world:latest
1-1 个，共 1 个
每页显示行数 15
1

Maven

最近更新时间：2023-09-11 16:05:25

该文档介绍如何将 Maven 类型的制品存储在 CODING 制品库中。其内容包括创建制品库、推送和拉取制品。

说明：

阅读该篇文档需要准备好以下内容：

- 《基础操作》——创建项目。
- 制品仓库选择 Maven 类型。

Maven 类型仓库支持 Apache Maven、Gradle Groovy、Gradle Kotlin DSL 三种格式文件。

配置认证信息

在对 Maven 制品仓库进行推送或拉取操作之前，需要配置认证信息。单击操作指引中的**生成个人令牌作为凭据**，将其添加至 `settings.xml` 文件中。

编译 Maven 制品并上传

以一个简单的 demo 为例，我们希望把这个 demo 的 Maven 包推送到上述步骤中创建好的 Maven 仓库中。

```
[redacted]:0.0.1-SNAPSHOT [redacted] ls -l
total 32
-rw-r--r--  1 [redacted]  staff   220 10 28 13:42 _remote.repositories
-rw-r--r--  1 [redacted]  staff  2254 10 28 13:42 demo-for-artifacts-0.0.1-SNAPSHOT.jar
-rw-r--r--  1 [redacted]  staff   766 10 28 11:03 demo-for-artifacts-0.0.1-SNAPSHOT.pom
-rw-r--r--  1 [redacted]  staff   710 10 28 13:42 maven-metadata-local.xml
[redacted]0.1-SNAPSHOT [redacted] pwd
[redacted]/.m2/repository/coding/demo-for-artifacts/0.0.1-SNAPSHOT
```

1. 在仓库的指引页面，复制下列配置到项目的 pom.xml 文件当中。

通常一个 Maven 项目当中已有 groupId、artifactId、version 的配置，只将 distributionManagement 拷贝进去即可。

demo-for-artifacts/pom.xml

```xml
<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.x
    <modelVersion>4.0.0</modelVersion>

    <groupId>coding</groupId>
    <artifactId>demo-for-artifacts</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>demo-for-artifacts</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <distributionManagement>
        <repository>
            <!--必须与 settings.xml 的 id 一致 -->
            <id>anywhere-coding-demo-my-maven</id>
            <name>my-maven</name>
            <url>https://anywhere-maven.pkg.coding.net/repository/coding-demo/my-maven/</url>
        </repository>
    </distributionManagement>
</project>
```

2. 执行 mvn deploy 命令。

```
mvn deploy
```

若提示未找到 settings.xml 文件，在命令末尾加上 -s 参数来设置您放置 settings 文件的路径，代入参数后：

```
 mvn deploy -s "/Users/somebody/software/apache-maven-
3.6.2/conf/settings.xml"
```

3. mvn 命令提示 build success 后，刷新制品仓库页面，即可看到最新推送上来的制品。

上传无源码的 Maven 包

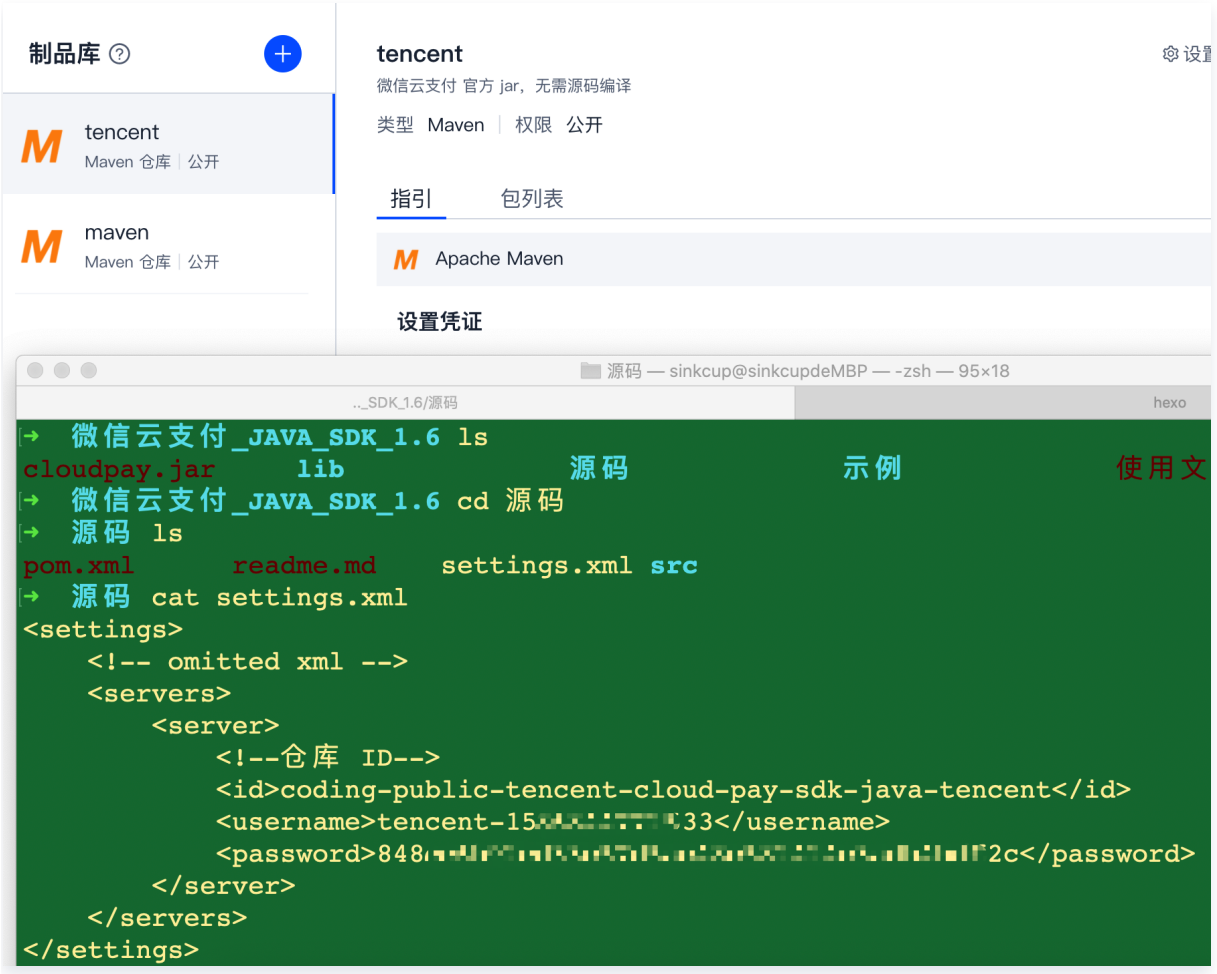
如果第三方 Maven 包未正规发布到网络仓库，而且仅提供 jar 包，未提供源码或者源码编译报错，那我们可以把 jar 包直接上传到仓库，命令如下：

```
mvn deploy:deploy-file -Durl=file://C:\m2-repo \
```

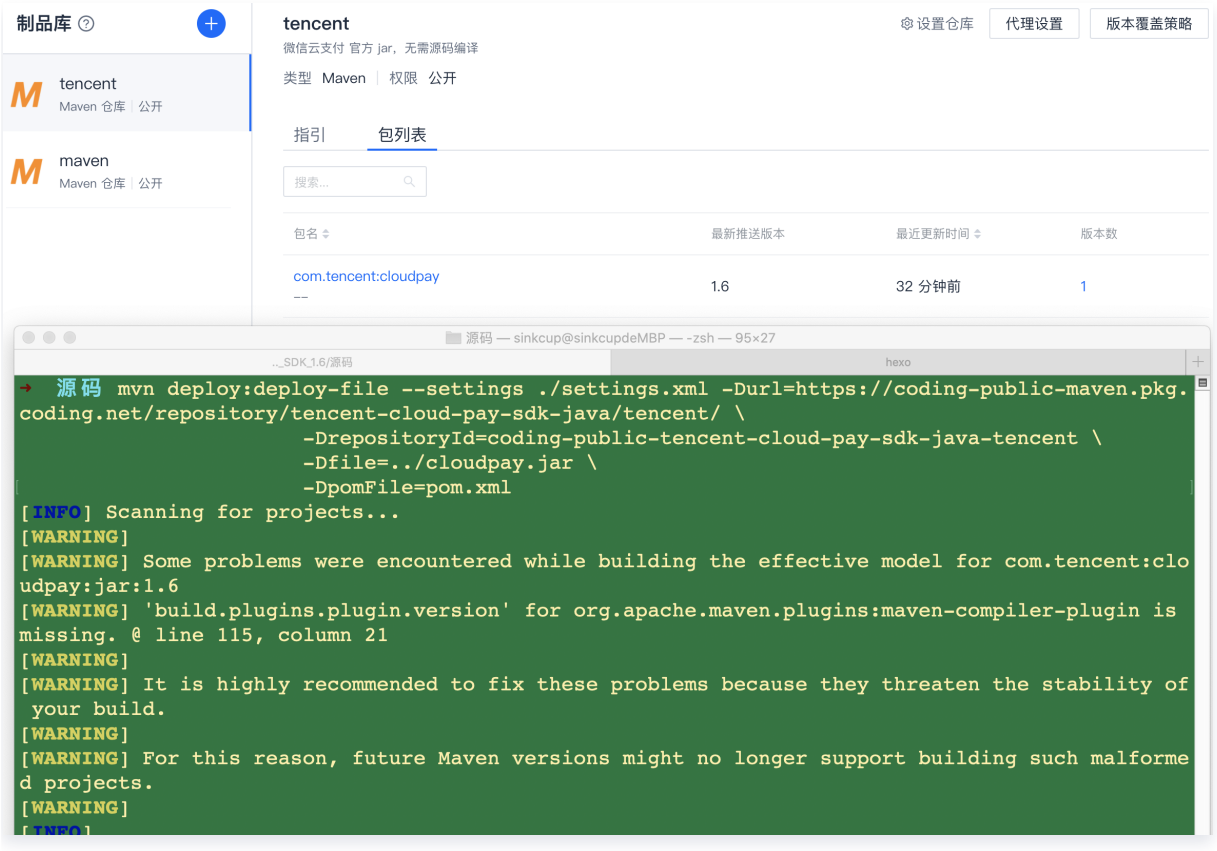
```
                        -DrepositoryId=some.id \
                        -Dfile=your-artifact-1.0.jar \
                        [-DpomFile=your-pom.xml] \
                        [-DgroupId=org.some.group] \
                        [-DartifactId=your-artifact] \
                        [-Dversion=1.0] \
                        [-Dpackaging=jar] \
                        [-Dclassifier=test] \
                        [-DgeneratePom=true] \
                        [-DgeneratePom.description="My Project
Description"] \
                        [-DrepositoryLayout=legacy]
```

如果第三方提供了 `pom.xml` ，可以从中读取 group、artifact、version 等字段，例如 微信云支付 Java SDK 使用下列命令：

```
mvn deploy:deploy-file --settings ./settings.xml -Durl=https://coding-
public-maven.pkg.coding.net/repository/tencent-cloud-pay-sdk-
java/tencent/ \
                        -DrepositoryId=coding-public-tencent-cloud-pay-
sdk-java-tencent \
                        -Dfile=../cloudpay.jar \
                        -DpomFile=pom.xml
```



微信云支付 Java SDK 上传 jar 包列表页面。



拉取 Maven 制品

1. 在仓库的指引页面，复制配置到 settings.xml 当中，例如 微信云支付 Java SDK 的配置如下：

```
<settings>
    <!--   omitted xml -->
    <profiles>
        <profile>
            <id>Repository Proxy</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <repositories>
                <repository>
                    <id>coding-public-tencent-cloud-pay-sdk-java-
tencent</id>
                    <name>tencent</name>
                    <url>https://coding-public-
maven.pkg.coding.net/repository/tencent-cloud-pay-sdk-
java/tencent/</url>
                    <releases>
                        <enabled>true</enabled>
                    </releases>
                    <snapshots>
                        <enabled>true</enabled>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
    </profiles>
</settings>
```

2. 在您的 Java 项目的 pom.xml 中配置依赖包（dependencies 标签），例如依赖微信云支付 Java SDK 的配置如下：

```
<project>
    <dependencies>
        <dependency>
            <groupId>com.tencent</groupId>
            <artifactId>cloudpay</artifactId>
            <version>1.6</version>
        </dependency>
```

```
    </dependencies>
</project>
```

3. 编译项目。

```
mvn install -s ./settings.xml
```

执行过程您可以看到包被正常拉取下来，也可以在执行完成后在本地 maven 缓存看到拉取下来的包：

```
pom.xml src     target
P[redacted]emo-for-artifacts-consumer [redacted]$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< coding:demo-for-artifacts-consumer >-----------------
[INFO] Building demo-for-artifacts-consumer 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from anywhere-coding-demo-my-maven: https://anywhere-maven.pkg.coding.net/repos
itory/coding-demo/my-maven/coding/demo-for-artifacts/0.0.1-SNAPSHOT/maven-metadata.xml
Downloaded from anywhere-coding-demo-my-maven: https://anywhere-maven.pkg.coding.net/reposi
tory/coding-demo/my-maven/coding/demo-for-artifacts/0.0.1-SNAPSHOT/maven-metadata.xml (774
B at 332 B/s)
```

Npm

最近更新时间：2023-09-11 16:05:25

该文档介绍如何将 npm 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含如何进行制品制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Node.js。
> - 《基础操作》——创建项目。
> - 制品仓库选择 npm 类型。

初始化本地 npm 项目（可选）

若您已熟悉 npm 制品的操作，则可以跳过此章节。

1. 新建 Demo 目录作为 npm 的项目地址。

```
mkdir npm-demo
```

2. 初始化 npm 项目。

```
cd npm-demo && npm init
```

根据提示在新增的 `package.json` 填入该 npm 包的配置文件。
参考内容：

```
{
    "name": "example",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "author": "",
    "license": "MIT"
}
```

3. 新建 `.npmrc` 文件。

```
touch .npmrc
```

配置认证信息

在对制品进行推送或拉取操作之前，需要配置认证信息。

有两种方式可以配置认证信息：

- 使用配置文件设置凭证
- 使用交互式命令行设置凭证

方法一：使用配置文件设置凭证

1. 在制品仓库的指引页面，输入密码后单击生成个人令牌作为凭据。

2. 复制页面中新增的交互式命令行设置凭据，将其添加到您项目的 `package.json` 同一级目录下的 `.npmrc` 文件。

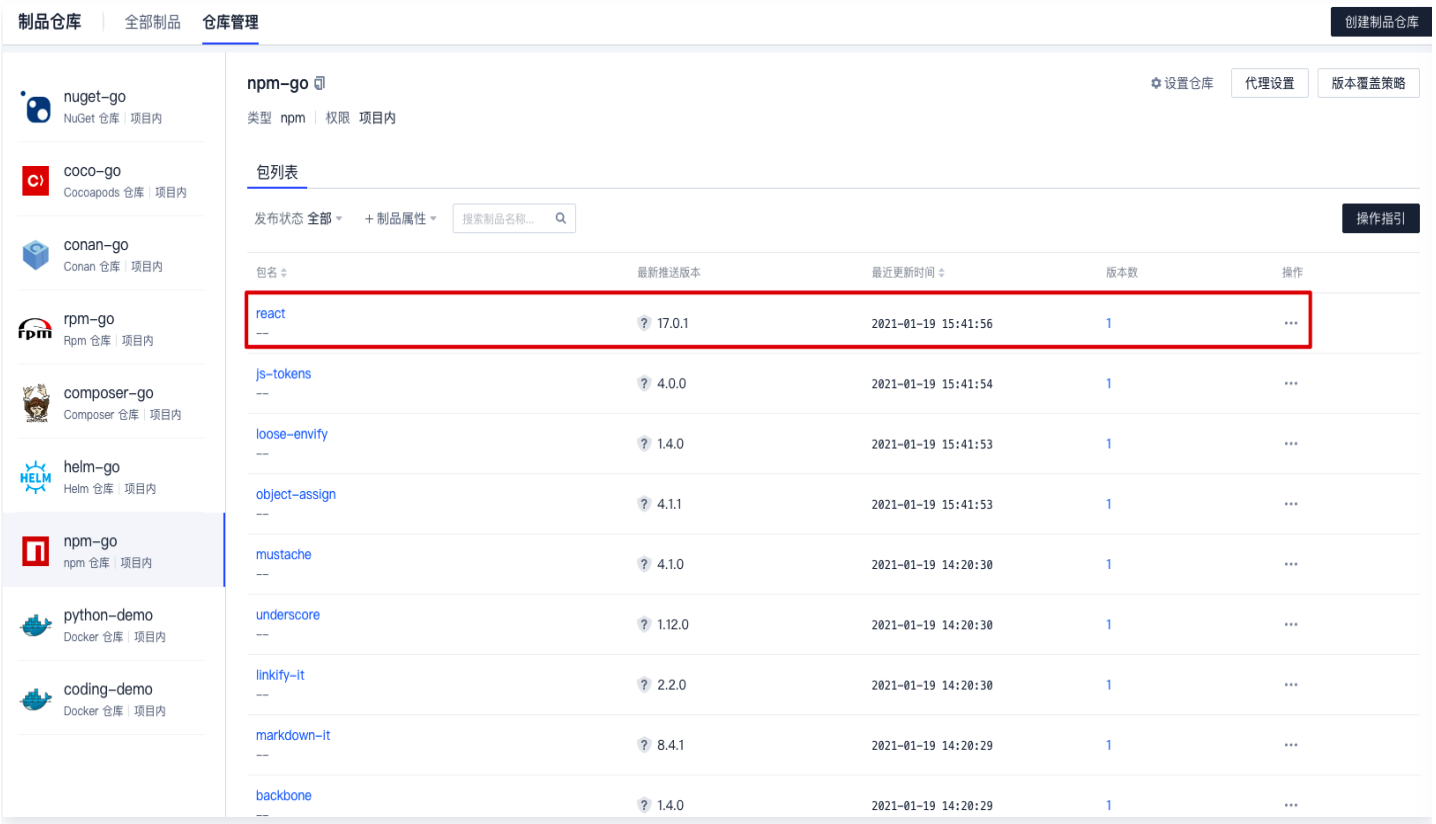
方法二：使用交互式命令行设置凭证

按照页面提示在本地制品库中设置相关凭证。

推送制品

在网页上的操作指引中输入制品名称与版本号，按照提示先初始化后再推送制品。

推送成功后，刷新仓库页面，您可以看到最新推送上来的制品。



拉取制品

在操作指引中输入制品名称与版本号，自动生成拉取命令。

执行完毕后，您可以看到拉取成功的信息提示：

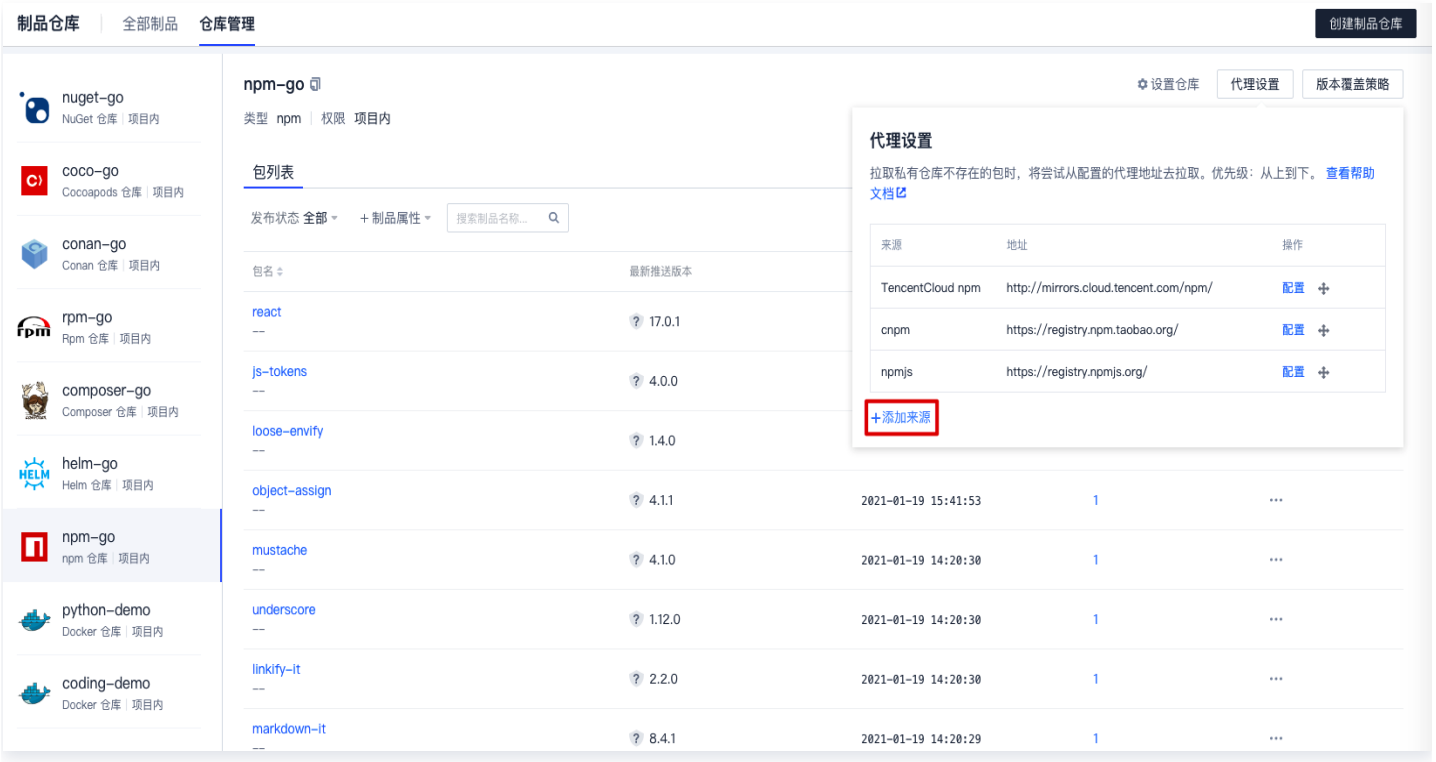
```
/Volumes/CODING/node  npm install hello-world --registry=https://straybirds-npm.pkg
.coding.net/coding-demo/npm-go/
npm WARN node@1.0.0 No description
npm WARN node@1.0.0 No repository field.

+ hello-world@0.0.2
updated 1 package in 35.936s

3 packages are looking for funding
  run `npm fund` for details
```

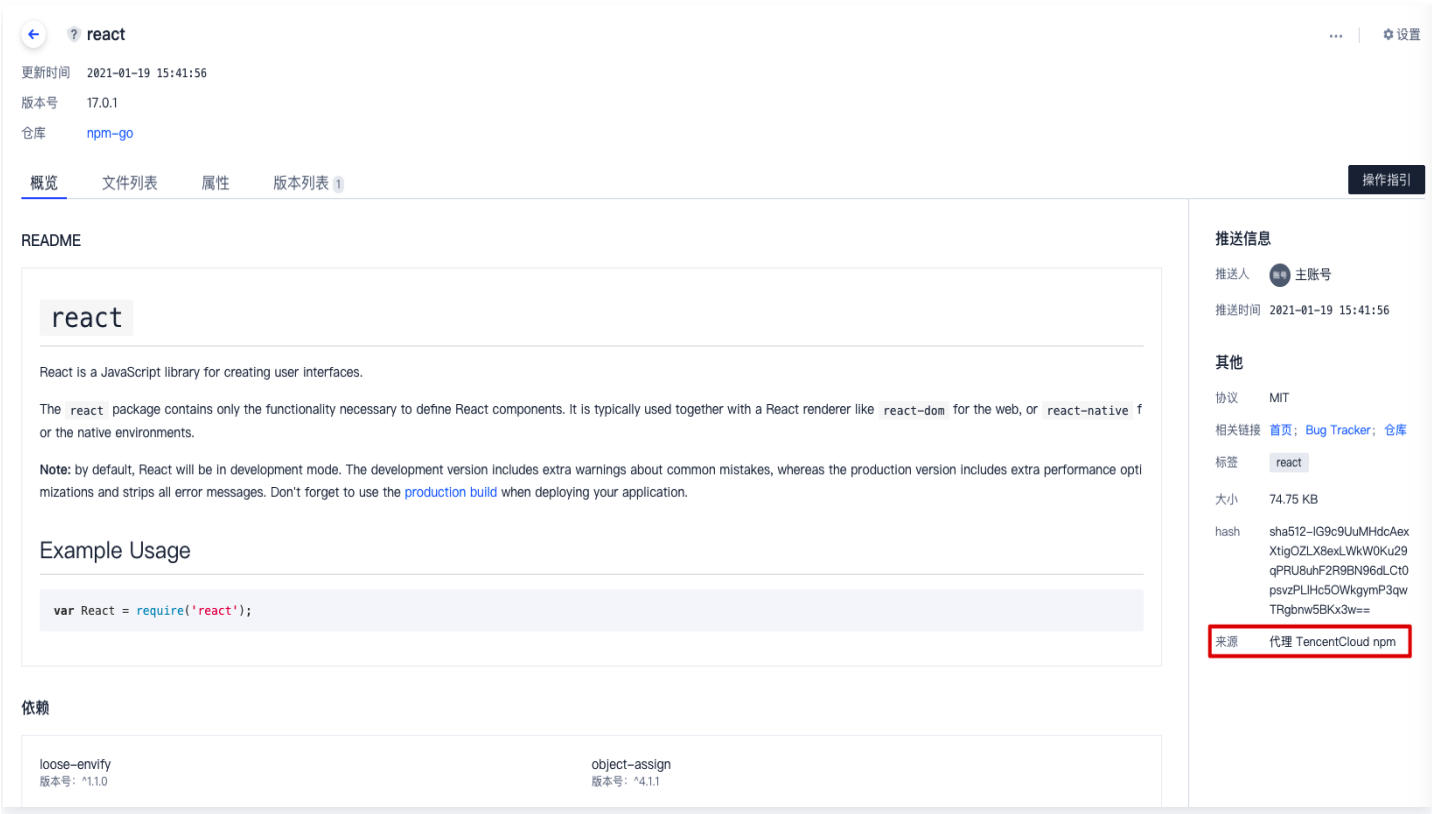
设置代理

当 CODING 制品仓库尚未托管想要拉取的制品时，将尝试从配置的代理地址拉取。您可以添加第三方制品源，用以获取特定仓库中的制品。无需额外设置，制品仓库将会按照顺序从上到下依次检索相应的制品包。



使用操作指引中的拉取命令，替换 `<package>` 的包名，完成拉取。

拉取的制品及依赖会成功拉取到本地，并且还会同步至 CODING 制品仓库中，详情页会显示包的来源。



更多请参见 制品库代理 。

Helm

最近更新时间：2023-09-11 16:05:25

该文档介绍如何将 Helm 类型的制品存储在 CODING 制品库中。其内容包括创建制品库、推送、拉取和删除制品。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Helm 。
> - 《基础操作》——创建项目。
> - 制品仓库选择 Helm 类型。

制作包（可选阅读）

本章节提供两种方法快速创建一个 Helm Chart，若您已熟悉制作方法可跳过本节。

方法一：本地制作镜像

1. 在本地任意目录创建 Helm Chart 并自定义包名。

```
helm create [name]
```

2. 打包。

```
helm package [name]
```

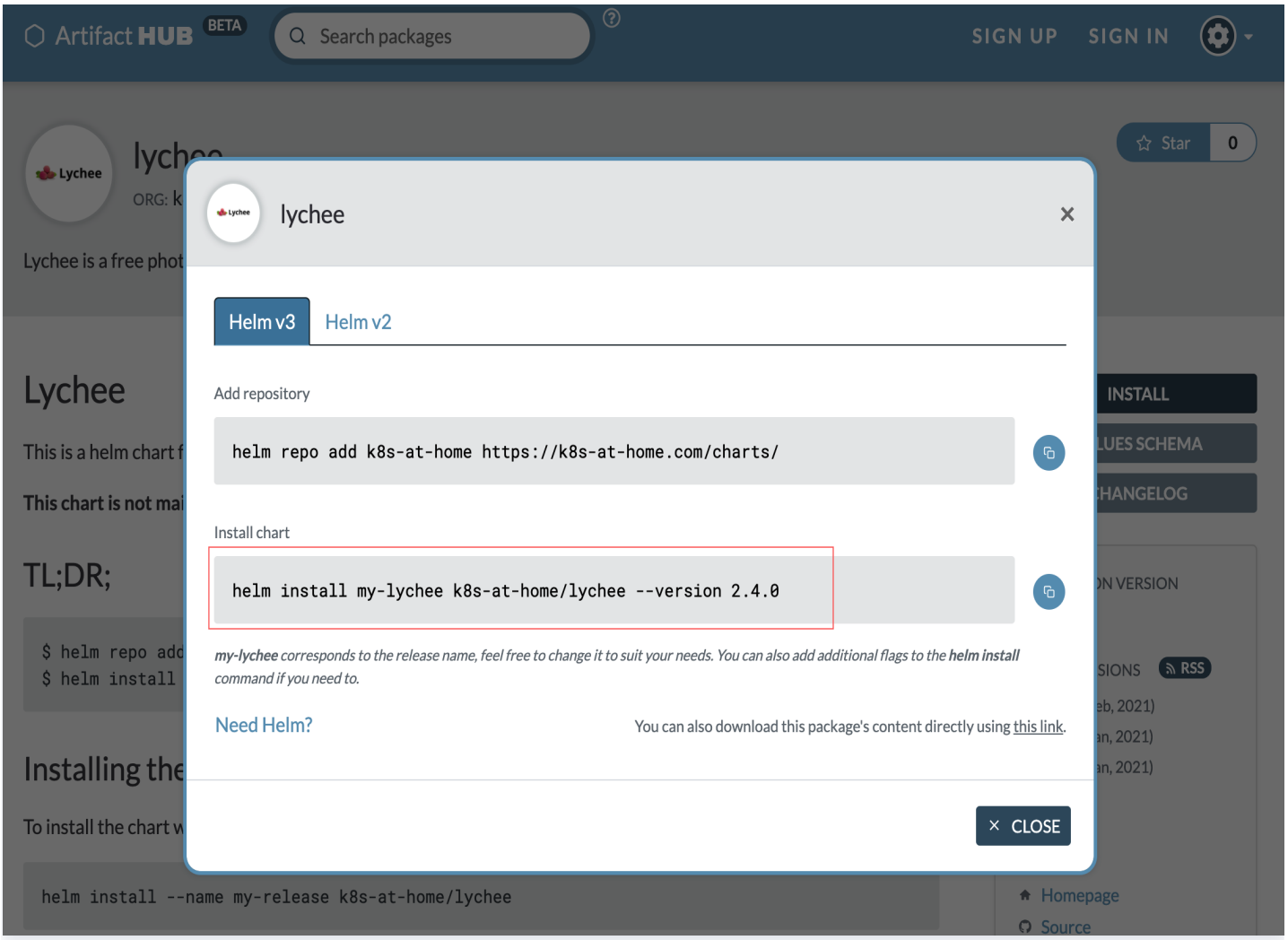
方法二：直接拉取 artifacthub 中的制品

搜索任意 Helm Chart 并在本地自定义目录下运行下载命令。

```
$ helm repo add [远程仓库名] [远程仓库地址]

$ helm fetch [helm chart 在远程仓库的地址>]--version [版本]
```

将 install 替换为 fetch，并删除 my-lychee 后复制此链接。



运行成功后本地会出现相关制品。

```
/Volumes/CODING/制品库写作/helm ls
hello                   hello-0.1.0.tgz         lychee-2.4.0.tgz        sample-golang-app-1.0.2.tgz
```

配置认证信息

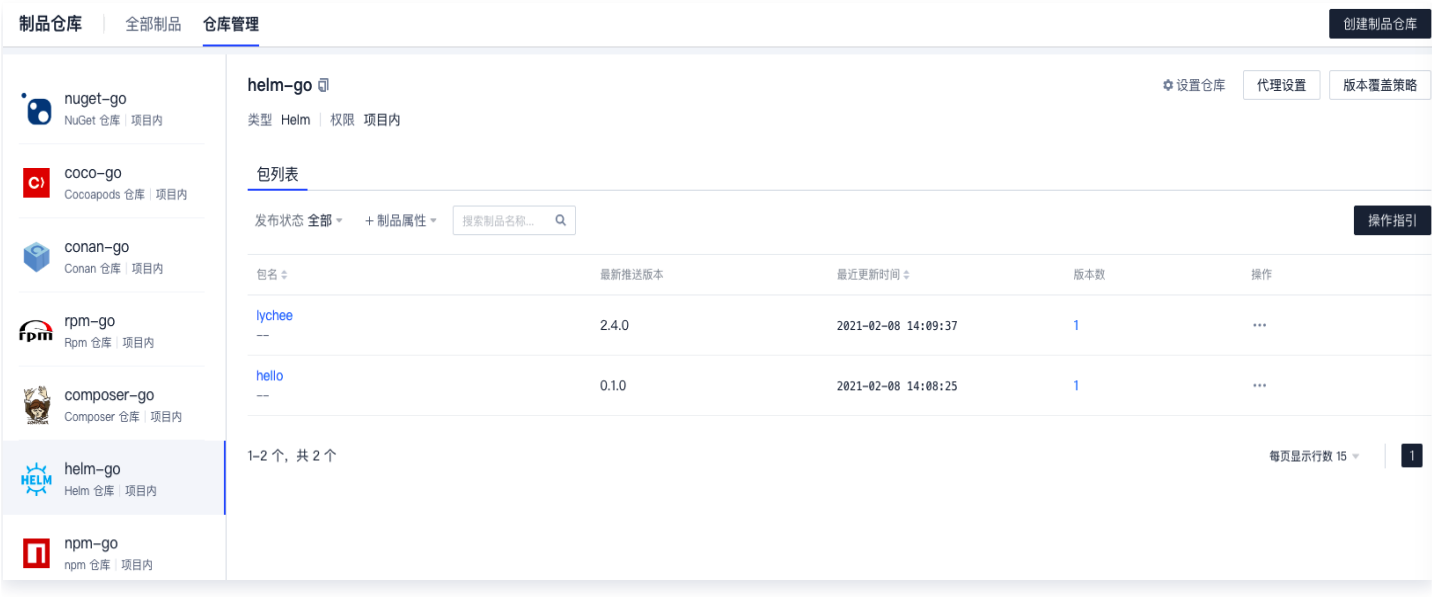
在本地完成制品编译后，就可以将制品推送至远端制品仓库。您可以选择 Helm+ cURL 或 Helm + CODING Helm 插件两种方法进行制品推拉。

按照指引提示，输入密码后进行信息认证。

推送制品

在操作指引页输入制品名称，自动生成推送命令，复制后使用终端进入 Helm Chart 所在目录执行。

推送成功后，刷新仓库页面即可看到最新推送的制品。



拉取制品

如果您的制品仓库有更新，在操作指引中生成拉取命令进行更新。

PyPI

最近更新时间：2023-09-11 16:05:25

该文档介绍如何将 PyPI 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含如何进行制品制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Python3。
> - 《基础操作》——创建项目。
> - 制品仓库选择 PyPI 类型。

初始化

1. 新建 Demo 目录作为本地 PyPI 包的地址，在终端中运行命令创建 Demo 项目文件夹。

```
mkdir -p demo/example_pkg/__init__.py
```

2. 进入 demo 目录，创建 setup.py 文件。

```
cd demo && touch setup.py
```

3. 在 setup.py 文件中粘贴配置内容。

```
import setuptools

setuptools.setup(
    name="example-pkg-YOUR-USERNAME-HERE", # Replace with your own
username
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    url="https://github.com/pypa/sampleproject",
    packages=setuptools.find_packages(),
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
```

```
    ],
    python_requires='>=3.6',
)
```

4. 安装 setuptools 和 wheel 工具。

```
python3 -m pip install --user --upgrade setuptools wheel
```

5. 打包项目。

```
python3 setup.py sdist bdist_wheel
```

打包项目后，会在 `/dist` 目录下生成以下两个文件，用于推送到制品仓库。

```
└─dist
    ├─example_pkg_YOUR_USERNAME_HERE-0.0.1-py3-none-any.whl
    ├─example_pkg_YOUR_USERNAME_HERE-0.0.1.tar.gz
```

配置制品库认证信息

推送至 CODING 制品仓库之前，需在本地文件中添加相应的认证信息。您可以通过**自动生成配置**或**手动配置**两种方式进行认证。在进行操作前，请使用命令 `cd /` 前往根目录，输入 `ls -a` 查看是否存在 `.pypirc` 和 `pip.conf` 文件。
如果没有的话，输入以下命令以新建两个文件。

```
touch .pypirc && touch pip.conf
```

自动生成配置

1. 单击网页上的**生成个人令牌作为凭据**，系统会自动帮您生成访问凭据。若需查看个人令牌，则前往**个人账户设置 > 访问令牌**处进行管理。

2. 输入登录密码后确认，得到配置内容。

   将配置内容复制进入根目录的 `.pypirc` 和 `pip.conf` 文件中。

推送

进入项目目录，如上文中新建的 Demo 目录，复制网页上的命令后在终端执行，即可把 `Demo/dist` 目录下的所有制品推送至制品库。

```
twine upload -r coding-pypi dist/*
```

```
/Volumes/CODING/pypi/packaging_tutorial twine upload -r coding-pypi dist/*
Uploading distributions to https://codingcorp-pypi.pkg.coding.net/coding-help-generator/pypi-go
Uploading example_pkg_YOUR_USERNAME_HERE-0.0.1-py3-none-any.whl
100%|████████████████████████████████| 4.41k/4.41k [00:00<00:00, 6.96kB/s]
Uploading example-pkg-YOUR-USERNAME-HERE-0.0.1.tar.gz
```

推送成功后，刷新仓库页面，您可以看到最新推送的制品。

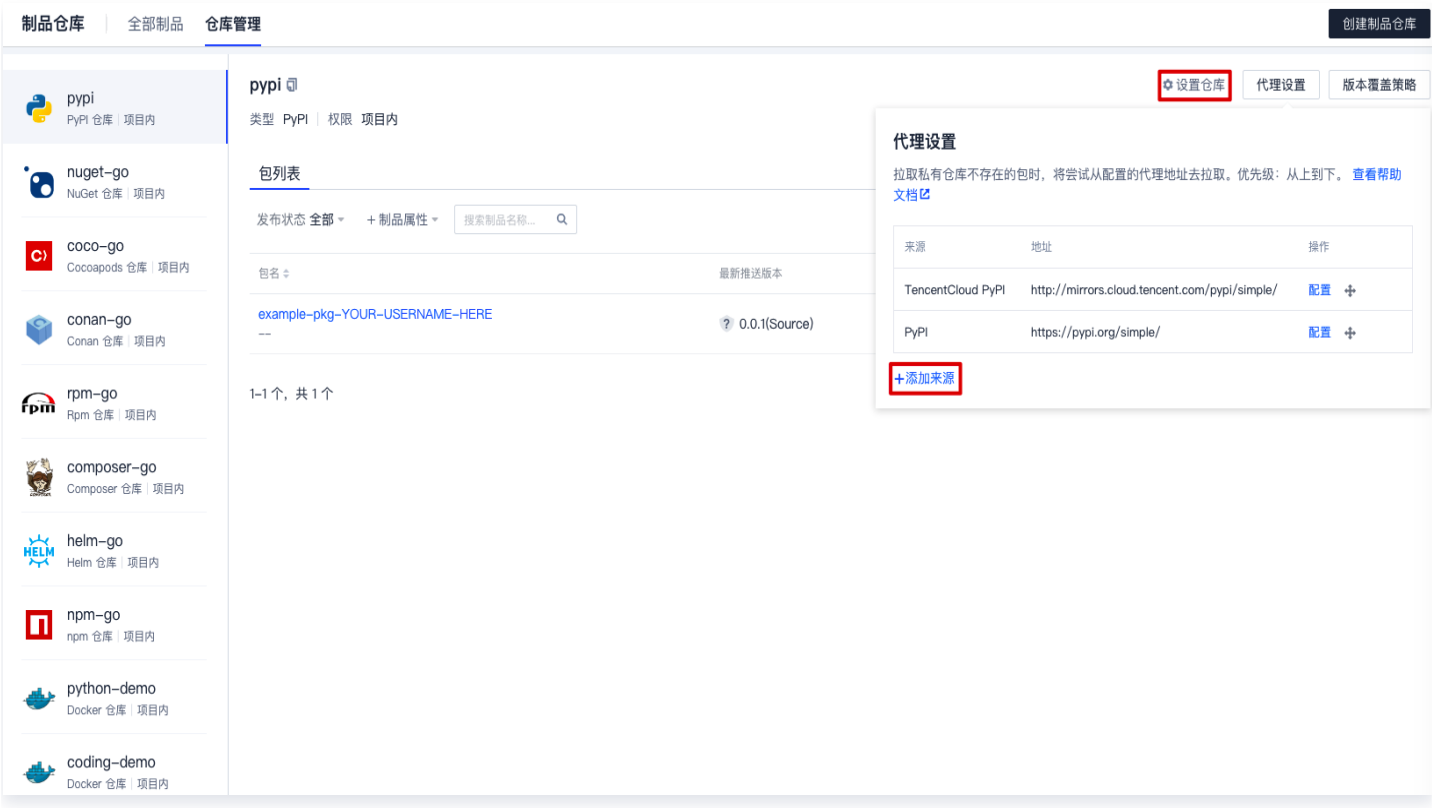
拉取

根据 PyPI 制品仓库中具体制品的拉取指引可以执行 `pip install` 拉取制品。

```
pip install <制品名>
```

设置代理

当 CODING 私有制品仓库不存在想要拉取的制品时，将尝试从配置的代理地址拉取。您可以添加第三方制品源，用以获取特定仓库中的制品。无需额外设置，CODING 将会按照顺序从上到下依次检索相应的制品包。



使用网页上生成的命令，替换 `<package>` 的包名，完成拉取。拉取的制品及依赖会成功拉取到本地，并且还会同步至 CODING 制品仓库中，详情页会显示包的来源。
代理设置的详细说明，请参见 制品库代理 。

Composer

最近更新时间：2024-11-15 12:04:02

本文档介绍如何快速使用 Composer 制品仓库，方便团队在项目进行统一的制品管理与版本控制。内容包含如何进行制品包制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Composer。
> - 《基础操作》——创建项目。
> - 制品仓库选择 Composer 类型。

制作 Composer 包（可选阅读）

安装 Composer

在终端中执行下载 Composer 命令。

```
curl -sS https://getcomposer.org/installer | php
```

添加至环境变量，方便全局运行命令。

```
mv composer.phar /usr/local/bin/composer
```

初始化

新建 Demo 目录。

```
mkdir composer-demo && cd composer-demo
```

初始化 Composer 包，按照提示输入初始化信息。

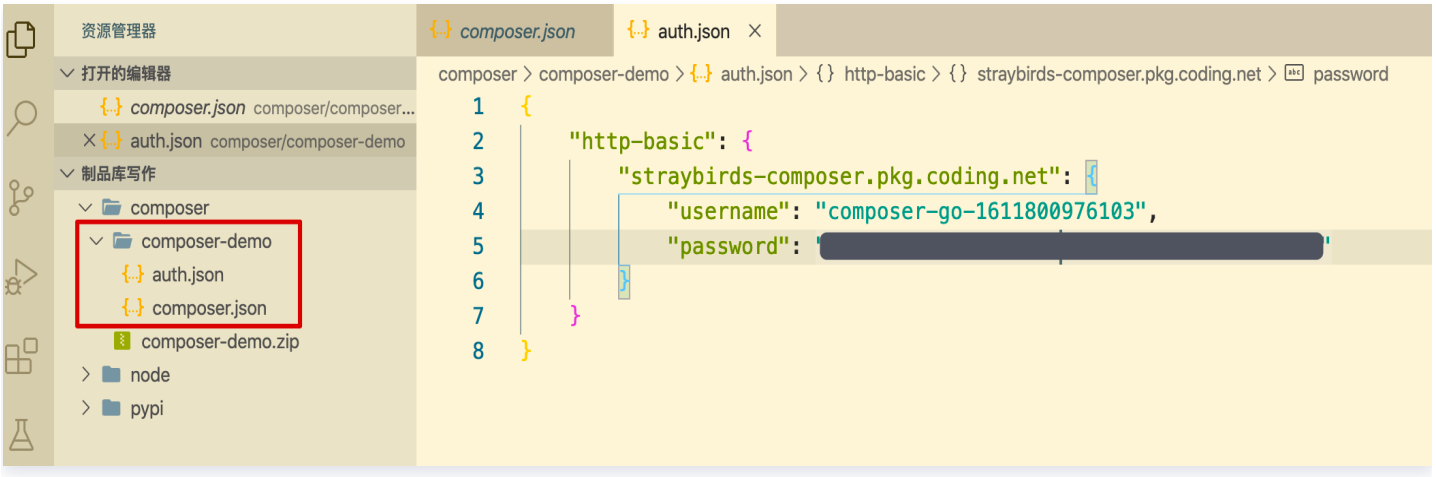
```
composer inits
```

初始化完成后会在同一目录下新增 composer.json 文件作为该 Composer 包的配置文件。

配置认证文件

前往 Composer 包的文件目录，新建 auth.json 文件。输入密码，单击操作指引页面中的**生成个人令牌作为凭据**自动生成推送凭据。

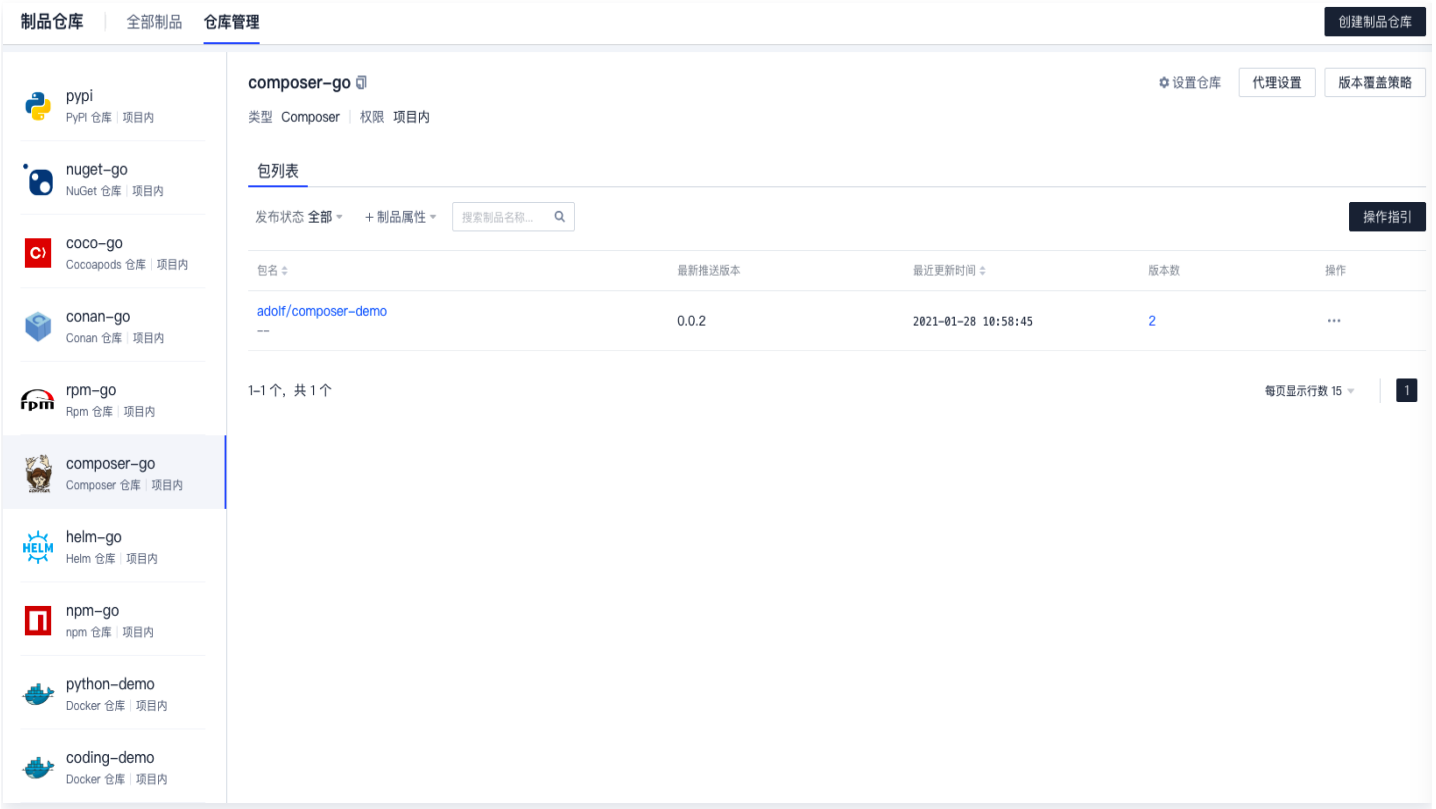
复制凭据后粘贴至 auth.json 文件内。



推送

在操作指引页中输入制品名称与版本自动生成推送命令后在终端中运行。

推送完成后即可在 CODING 制品库看到已推送的包。



拉取

在操作指引页中输入制品名称与版本自动生成拉取命令。

设置代理

当 CODING 私有制品仓库不存在想要拉取的制品时，将尝试从配置的代理地址拉取。您可以添加第三方制品源，用以获取特定仓库中的制品。无需额外设置，CODING 将会按照顺序从上到下依次检索相应的制品包。

代理设置的详细说明，请参见 制品库代理 。

Nuget

最近更新时间：2023-09-11 16:05:26

该文档介绍如何将 NuGet 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含制品仓库创建、NuGet 包制作、制品推拉与使用代理等。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 NuGet。
> - 《基础操作》——创建项目。
> - 制品仓库选择 NuGet 类型。

初始化 NuGet 制品（可选阅读）

访问官网下载并安装 NuGet。

本地生成

若您已熟悉 NuGet 制品的操作，则可以跳过此章节。

1. 新建 Demo 目录。

```
mkdir nuget-demo && cd nuget-demo
```

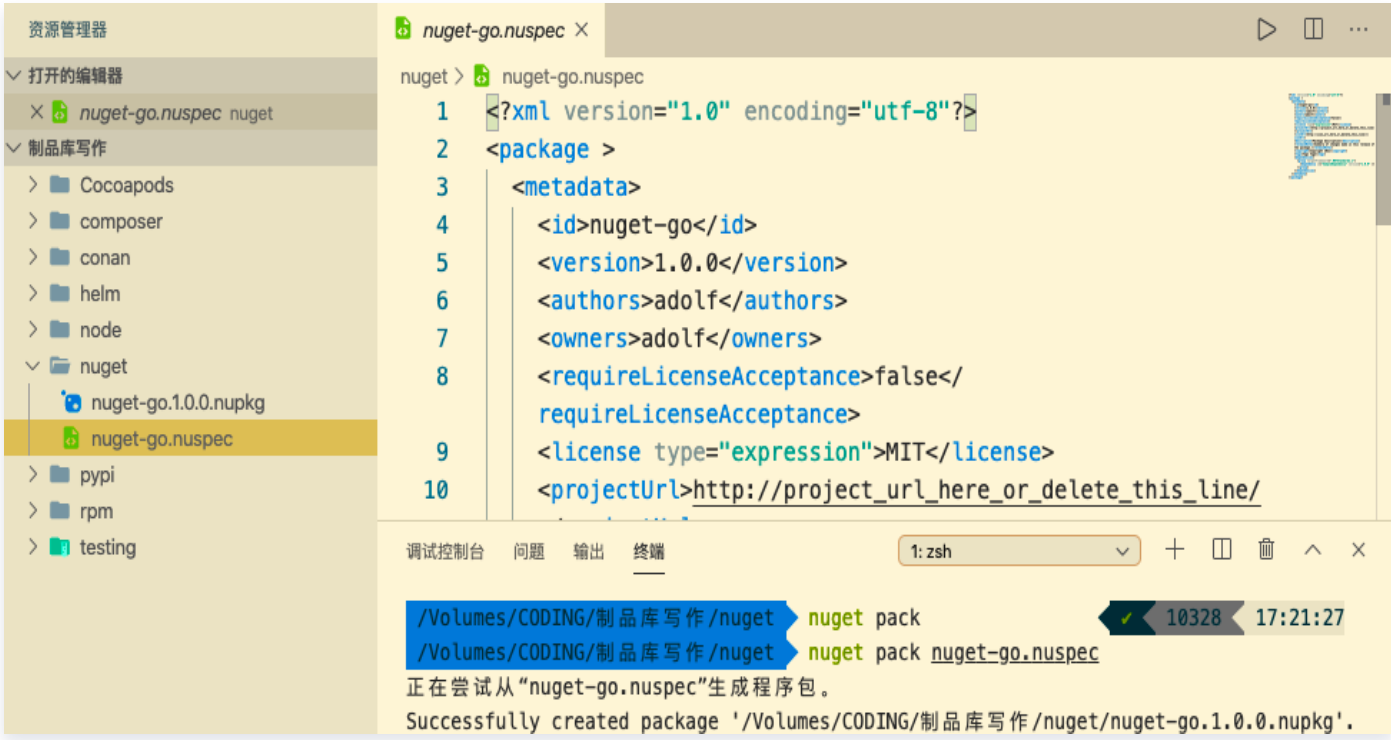
2. 创建 `.nuspec` 包。

```
nuget spec [制品名称]
```

3. 打包制品。

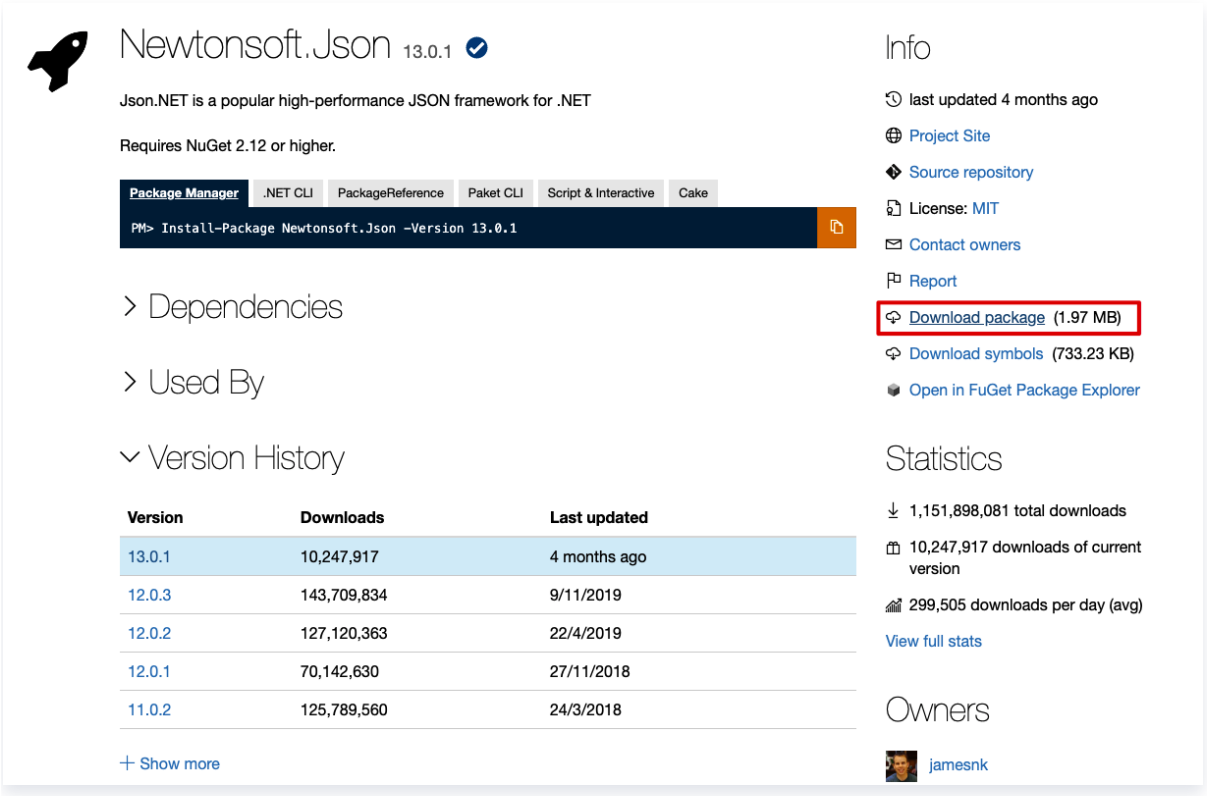
```
nuget pack <制品名称>.nuspec
```

4. 打包完成后即可在本地目录中看到生成的包文件。



在线拉取

单击访问 官网，搜索任意 NuGet 制品并通过在线链接或命令行下载。



通过命令行拉取：

```
nuget install [制品名称] -OutputDirectory packages
```

配置制品仓库认证

您需要在本地配置认证信息，用以访问 CODING 中的 NuGet 类型制品仓库。此处我们使用**自动生成配置**完成认证过程。
输入密码后，单击操作指引上的**生成个人令牌作为凭据**，输入密码后得到配置命令，复制后在需要推送的 NuGet 制品的所在目录执行配置命令即可。此过程的**权限机制**使用到了**个人访问令牌**功能，若希望对其进行控制，更多详细内容请参见 访问令牌 。

推送制品

输入命令行，将相应的名称替换为本地内容即可完成推送。

```
nuget push -ApiKey api -Source "nuget-go" [制品名称].nupkg
```

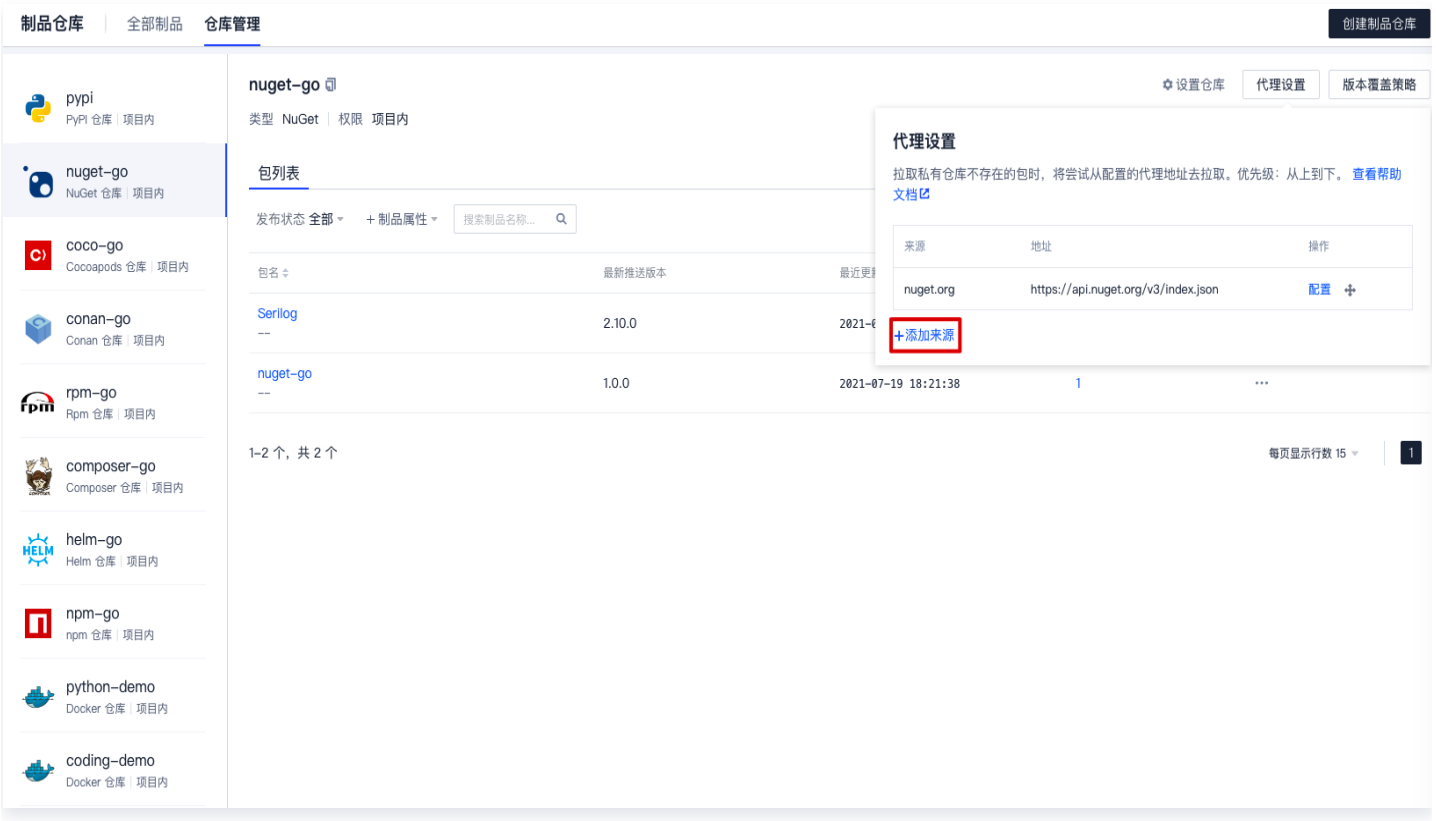
拉取制品

输入命令行，将相应的名称替换为本地内容即可完成拉取。

```
nuget install -Source "nuget-go" -Version [制品版本] [制品名称]
```

设置为代理

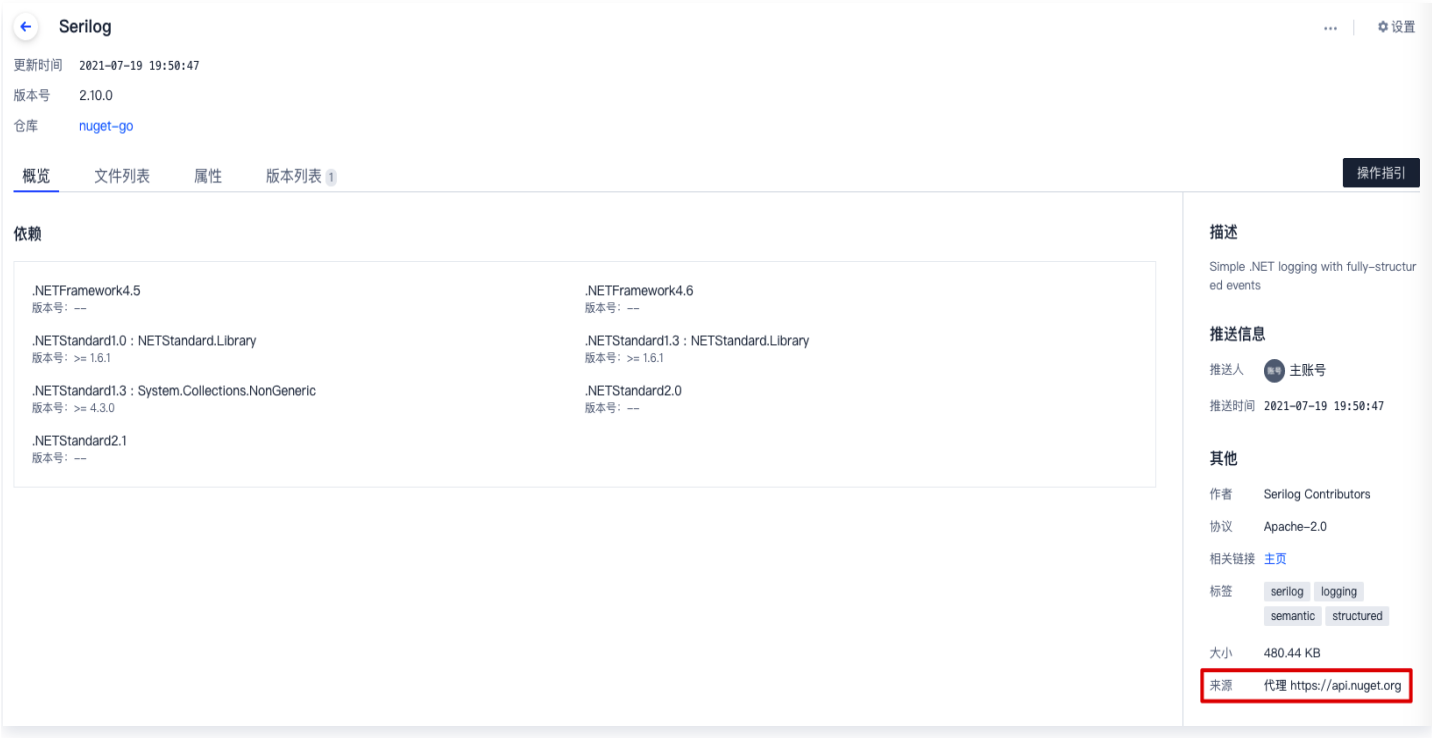
当 CODING 私有制品仓库不存在想要拉取的制品时，将尝试从配置的代理地址拉取。您可以添加第三方制品源，用以获取特定仓库中的制品。无需额外设置，CODING 将会按照顺序从上到下依次检索相应的制品包。



使用命令拉取制品：

```
nuget install -Source "nuget-go" -Version [制品版本] [制品名称]
```

拉取的制品及依赖会成功拉取到本地，并且还会同步至 CODING 制品仓库中，详情页会显示包的来源。



若 CODING 制品仓库中没有自动储存由代理拉取的 NuGet 制品，可能由于以下两点问题导致：

1. **您没有该仓库的推送权限。**
2. **您的本地缓存中已有该制品。**

代理设置的详细说明，请参见 制品库代理 。

Rpm

最近更新时间：2024-06-04 18:10:11

该文档介绍如何将 rpm 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含如何进行制品制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - Linux 环境。
> - 基础操作——创建项目。
> - 制品仓库选择 rpm 类型。

初始化

Linux 系统自带 rpm，您可以直接在运行 Linux 系统的终端直接运行命令，若置于其他操作系统，则可以使用 Docker 安装 Centos：

```
docker run -it --name centos centos:8 /bin/bash
```

下载 Demo 项目

进入 rpm 制品下载地址，搜索制品包并下载至本地后进行安装。
例如：

```
wget -N --no-check-certificate
"https://www.rpmfind.net/linux/fedora/linux/development/rawhide/Everythi
ng/aarch64/os/Packages/h/hello-2.12.1-4.fc40.aarch64.rpm"
```

配置仓库认证信息

单击**操作指引**上的**生成个人令牌作为凭据**将会自动生成设置凭证。

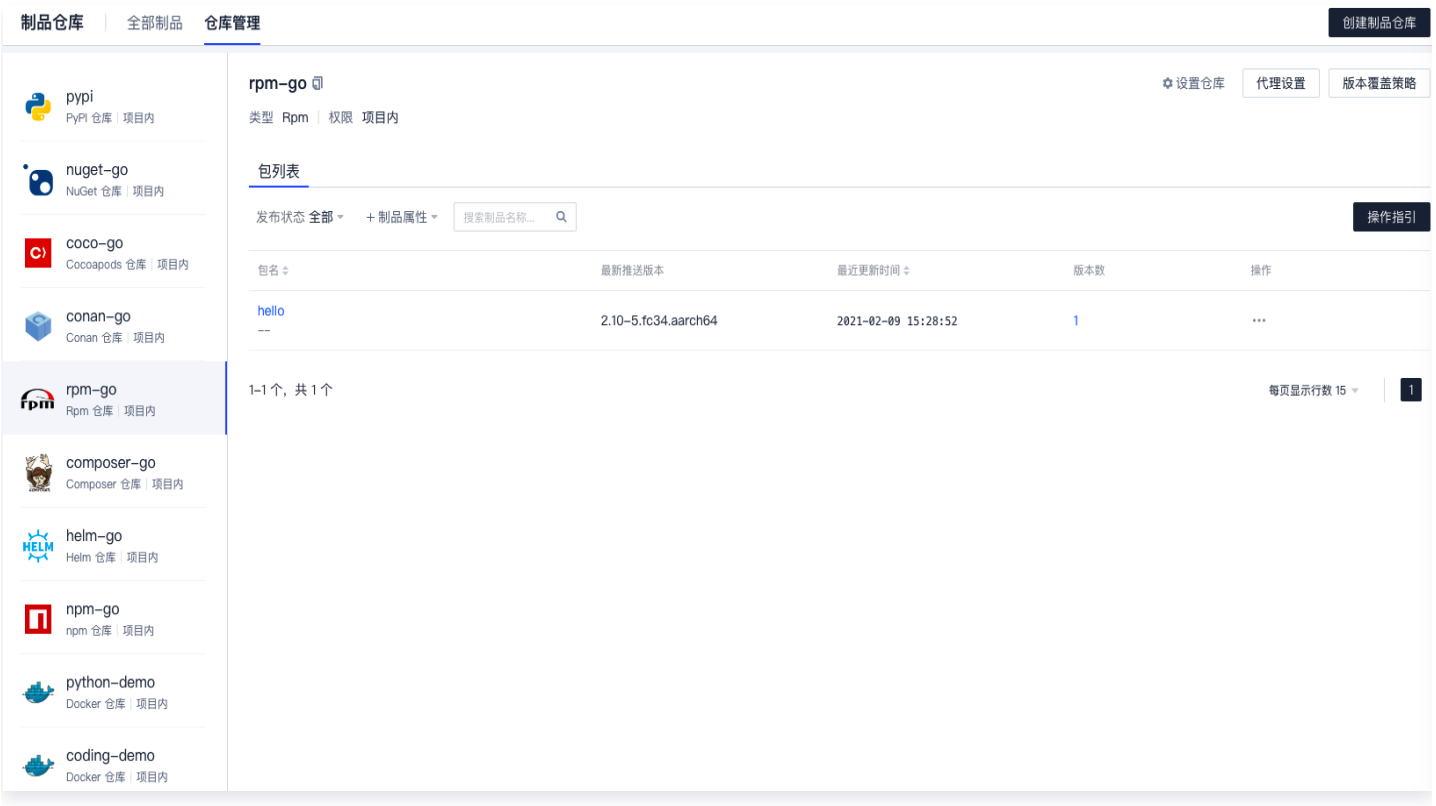
将生成的代码复制至本地的 `/etc/yum.repos.d/rpm-go.repo` 文件中，如果没有该文件请新建。

推送制品

执行 rpm publish 命令推送 rpm 包。

```
curl -u [用户名/邮箱] -X POST [推送指引中提供的仓库地址信息] -T [制品名称].rpm
```

推送成功后，刷新仓库页面，您可以看到最新推送上来的制品。

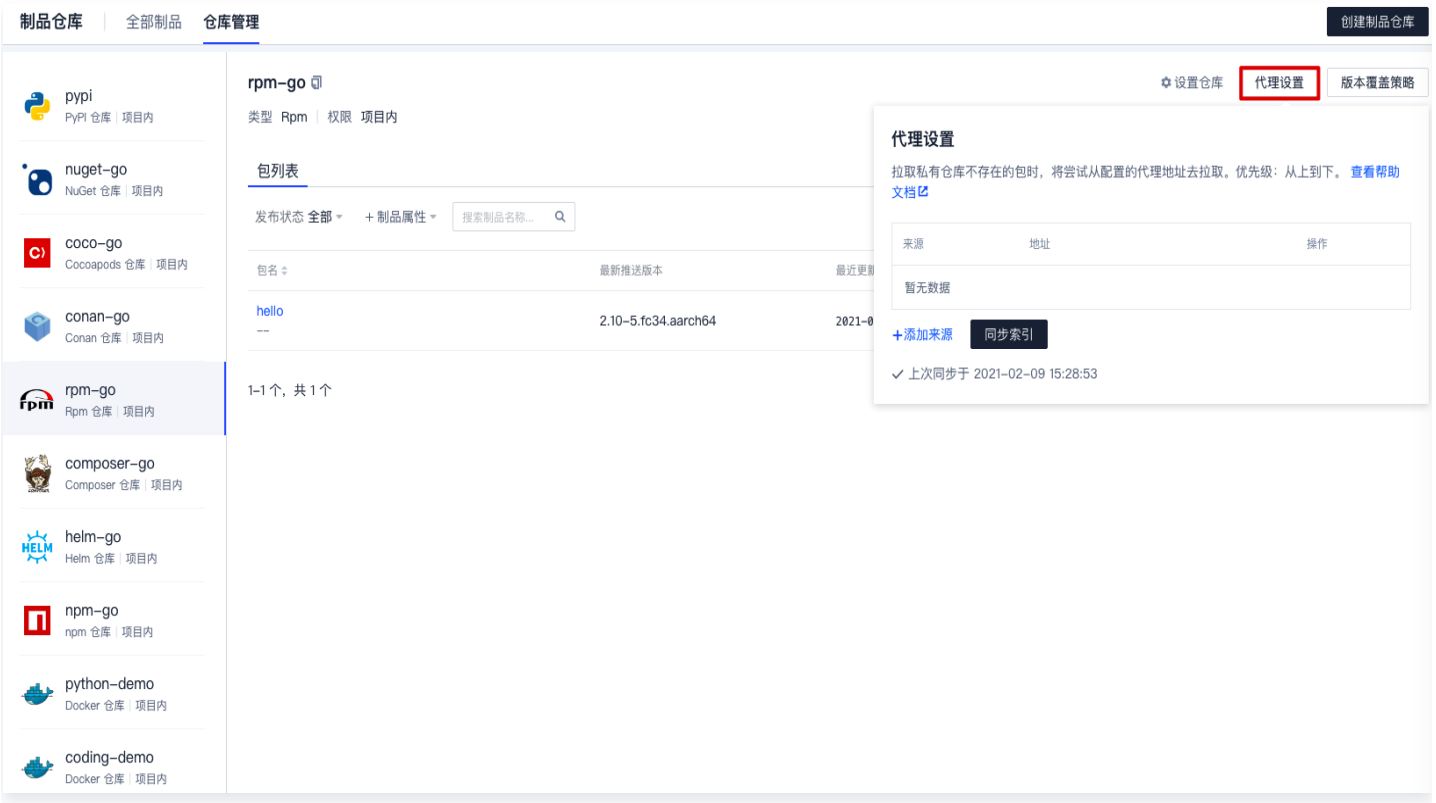


拉取制品

运行页面指引上的命令，完成拉取操作。

制品代理

rpm 仓库已有默认代理地址，可以自定义配置其他地址。



配置需要代理的远程仓库地址，拉取仓库中的制品至本地后，将自动备份至 CODING 制品仓库列表。

如果 rpm 制品仓库中没有储存代理的 rpm 制品，可能是因为以下两点原因：

1. 您没有该仓库的推送权限。
2. 您的本地缓存中已有该制品包。

Conan

最近更新时间：2023-09-11 16:05:26

该文档介绍如何将 conan 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含如何进行制品制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Python3。
> - 基础操作——创建项目。
> - 制品仓库选择 conan 类型。

安装 Conan

使用 pip 安装，需要 python3.5 及以上的版本。

```
pip3 install conan
```

使用 brew 安装。

```
brew install conan
```

新建 conan 包

在本地新建 Demo 目录。

```
mkdir mypkg && cd mypkg
```

创建 Demo 项目。

```
conan new hello/0.1 -t
```

将项目打包成二进制包。

```
conan create . demo/testing
```

如果执行报错 `/bin/sh: cmake: command not found`，需要执行命令安装 cmake。

```
$ pip3 install cmake
# 或
$ brew install cmake
```

配置仓库认证信息

在操作指引中输入密码后，单击**生成个人令牌作为凭据**将会自动生成执行命令。

推送制品

运行页面指引上的命令行，将变量替换为拟推送的制品名称与版本号。

```
conan upload [包名称]/[自定义版本号] --all -r=conan-go
```

拉取制品

执行页面指引中的拉取命令，从当前仓库拉取制品。

```
conan install <包名称>/<自定义版本号>@ -r conan-go
```

Cocoapods

最近更新时间：2023-09-11 16:05:26

该文档介绍如何将 Cocoapods 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含如何进行制品制作、认证配置与制品推拉。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Cocoapods。
> - 基础操作——创建项目。
> - 制品仓库选择 Cocoapods 类型。

安装

执行命令安装 Cocoapods。

```
$ sudo gem install cocoapods
-- 或
$ brew install cocoapods
```

新建 Demo 项目

在任意目录执行创建命令，并根据命令行提示选择需要的示例模版：

```
pod lib create <自定义 pod 名称>
```

该命令会将 Cocoapods 官方的示例代码从 GitHub 仓库克隆至本地。

配置认证信息

首先安装 CODING Cocoapods 插件。

```
sudo gem install cocoapods-coding-ar
```

在操作指引中输入密码，单击**生成个人令牌作为凭据**，系统将自动生成配置凭证。

复制命令至您的 `~/.netrc` 文件中。若没有该文件，运行 `vim ~/.netrc` 新建并粘贴内容。

推送制品

在操作指引中输入制品名称，按照指引步骤执行命令。

拉取制品

根据 Cocoapods 制品仓库中具体制品的拉取指引可以拉取指定的 Cocoapods 制品与版本。


操作指引
配置凭据
推送
拉取
帮助中心
拉取
输入以下拉取相关信息，生成拉取配置：
制品名称:
制品版本:
1. 请将以下配置添加到您的 Podfile 文件中：
source 'https://StrayBirds-cocoapods.pkg.coding.net/coding-demo/coco-go'
target 'MyApp' do
pod '<PACKAGE>', '~> <VERSION>'
end
2. 请在命令行执行以下命令进行拉取：
pod install

Go

最近更新时间：2023-09-13 10:16:11

该文档介绍如何将 Go 类型制品存储在 CODING 制品库中，方便团队在项目进行统一的制品管理与版本控制。下文包含 Go 制品仓库创建、本地打包 Go 制品、制品推拉使用等功能介绍。

> **说明：**
> 阅读该篇文档需要准备好以下内容：
> - 安装 Go 1.13 以上版本。
> - 已创建项目和制品仓库，具体操作可参见 基础操作 指引中的创建项目、新建制品库。
> - 制品仓库选择 Go 类型。

配置认证信息

在对制品进行推送或拉取操作之前，需要配置认证信息。

1. 在制品仓库的**仓库管理**页签，单击**操作指引**。

2. 在**操作指引**的**配置凭据**页面输入个人账号的登录密码，单击**生成个人令牌作为凭证**。

3. 根据操作系统添加 GOPROXY 配置，复制弹窗中的命令在终端执行。

推送 Go 制品

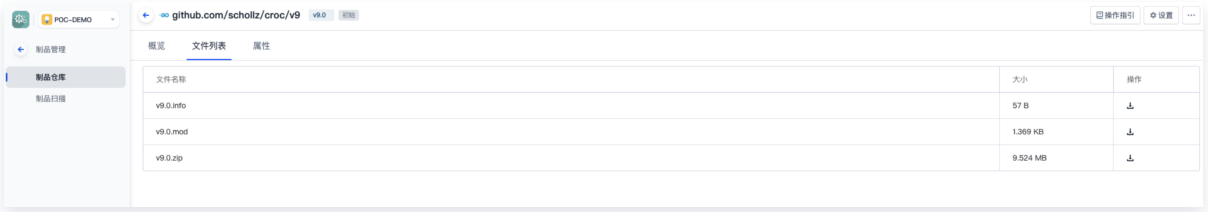
使用 CLI 工具推送 Go 制品。

1. 在**操作指引**的**推送**页面，根据不同操作系统复制并在终端执行对应的命令，完成 CLI 工具下载安装。

2. 输入**制品模块**、**制品版本**，系统自动生成推送命令。根据不同操作系统，复制并在终端执行对应的命令，完成 Go 制品包推送至制品仓库。

3. 制品包推送成功后终端界面显示如下信息。

制品仓库页面制品列表中能查看已经推送成功的 Go 制品。



拉取 Go 制品

使用以下一种方式拉取 Go 制品。

- 在**操作指引**的**推送**页面，输入 Go 制品的 module 信息，复制并在终端执行命令来拉取 Go 制品。

> **说明：**
>
> - 填写的**制品模块**信息必须与 Go 制品 `go.mod` 文件中 moudle 信息保持一致，否则会导致拉取 Go 制品库失败。

- 在非 https 环境使用 `go get` 命令时，要求 Go proxy 不附带鉴权信息，且必须是公开仓库。

- 在终端的 `go.mod` 文件所在目录下执行以下命令，拉取依赖制品。

制品包拉取成功后终端界面显示如下信息。

```
→  croc git:(main) ✗ go get github.com/schollz/croc/v9
go: added github.com/cespare/xxhash v1.1.0
```

权限配置

最近更新时间：2024-09-04 14:25:11

制品作为团队的重要资产，它的权限管理是至关重要的：例如某些制品需要开放给第三方外部使用（例如开源组件包）、某些制品需要开放给团队内的其他项目组成员使用（例如基础公共组件包）、某些制品只需开放给本项目成员使用（例如应用安装包）。CODING 制品库提供了完善的权限管理，满足不同场景下的权限需求。

权限规则

1. CODING 制品库提供了三种权限范围。
   - 项目内
   - 团队内
   - 公开

   公开制品需团队进行实名操作，由团队负责人/管理员前往团队设置中心进行实名认证。

2. 不同成员对制品的操作。
   - 拉取：从制品库拉取任意指定的制品。
   - 推送：推送任意制品到制品库。
   - 代理：从代理中同步不在缓存中的制品。
3. 权限方案
   - 项目权限方案：

     若选择项目权限方案，用户在制品库内的权限取决于其关联的项目权限组所拥有的制品库权限。

- **自定义权限方案：**

  若选择自定义权限方案，以指定的自定义权限为准。

  > **说明：**
  > **项目管理员默认具备所有制品操作权限，不受自定义权限方案与项目权限方案限制。**

设置权限

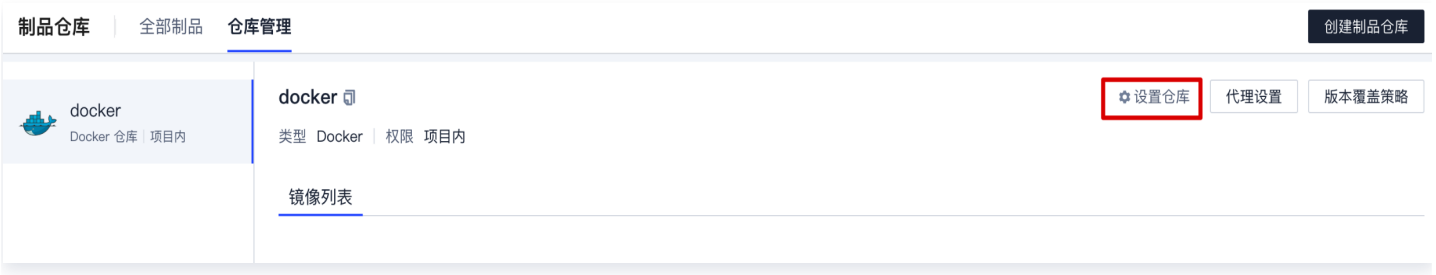
了解上述权限对应的鉴权规则后，您可以在创建制品仓库时按需设置权限范围，对于已经创建好的制品仓库您也可以进行权限范围修改。

创建仓库时设置权限

在制品仓库页面，新建制品仓库时即可设置权限范围。

对已存在的仓库修改权限

单击仓库右上方的设置仓库。



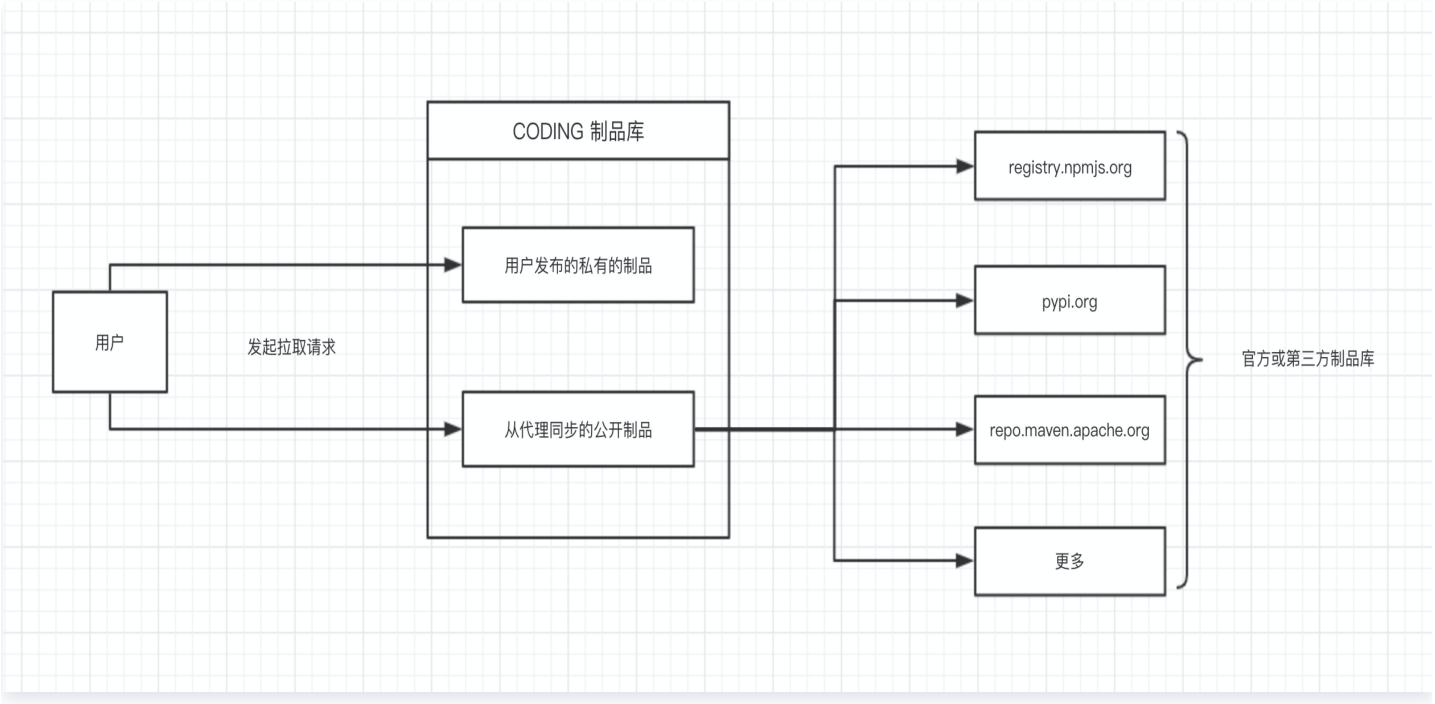
在**基本信息**中即可对权限范围进行更新。

制品库代理

最近更新时间：2024-08-02 15:23:01

功能介绍

制品库的代理功能支持用户配置仓库代理多个源，当私有仓库内找不到对应的包时，会尝试去配置的源拉取对应的包返回给用户。同时也支持用户配置代理源认证的账号信息。制品库代理功能可作为统一入口帮助用户管理依赖的第三方制品。



制品拉取的顺序如下：

1. 优先获取私有仓库内的包。
2. 私有仓库内无法找到时，再从配置代理的源按照从上到下顺序查找。

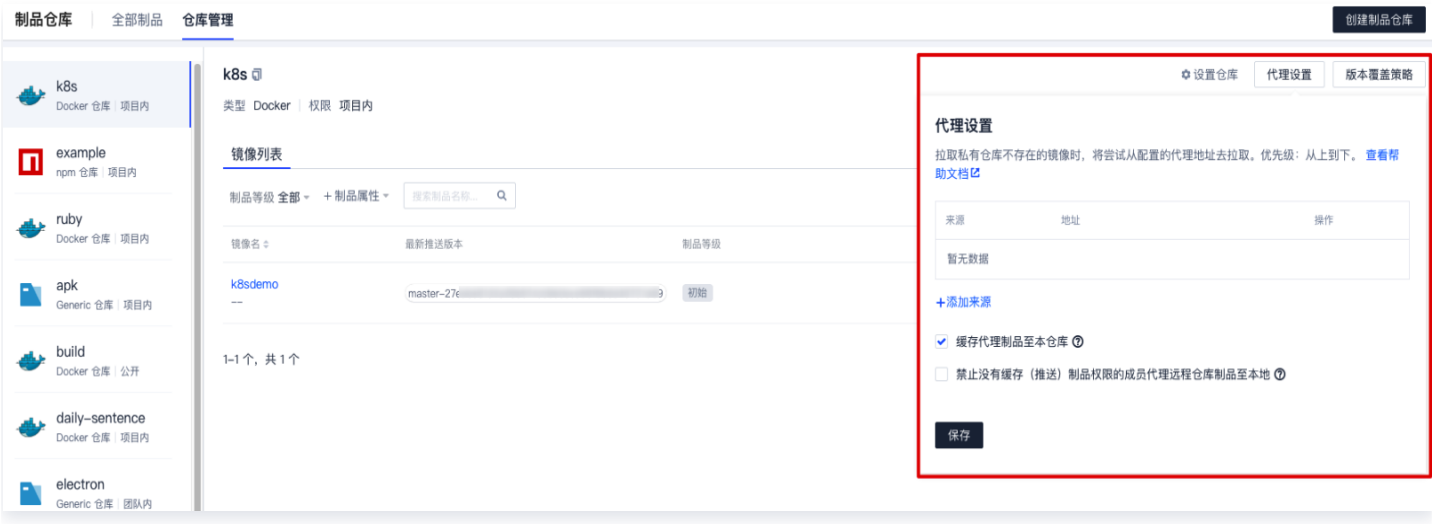
开启代理

在**制品库**下新建制品库时，可选择**启用代理**，默认此项打开。


* 制品仓库
Generic
Docker
Maven
npm
PyPI
Helm
Composer
NuGet
Conan
Cocoapods
Rpm
Go
* 仓库名称
https://zzz et/coding-demo/
仅允许英文、数字、下划线、中划线、点(.)
仓库描述
请输入仓库描述，最多可输入100个字符
* 权限范围
项目内
团队内
公开
本仓库内的制品，仅项目内有访问权限的用户可见，您可以在下方配置制品的拉取 / 推送 / 删除权限，设置权限组请前往 团队设置中心
使用项目权限方案 查看详情
自定义项目权限方案
启用代理
允许获取代理源的镜像
什么是制品代理
确认
取消

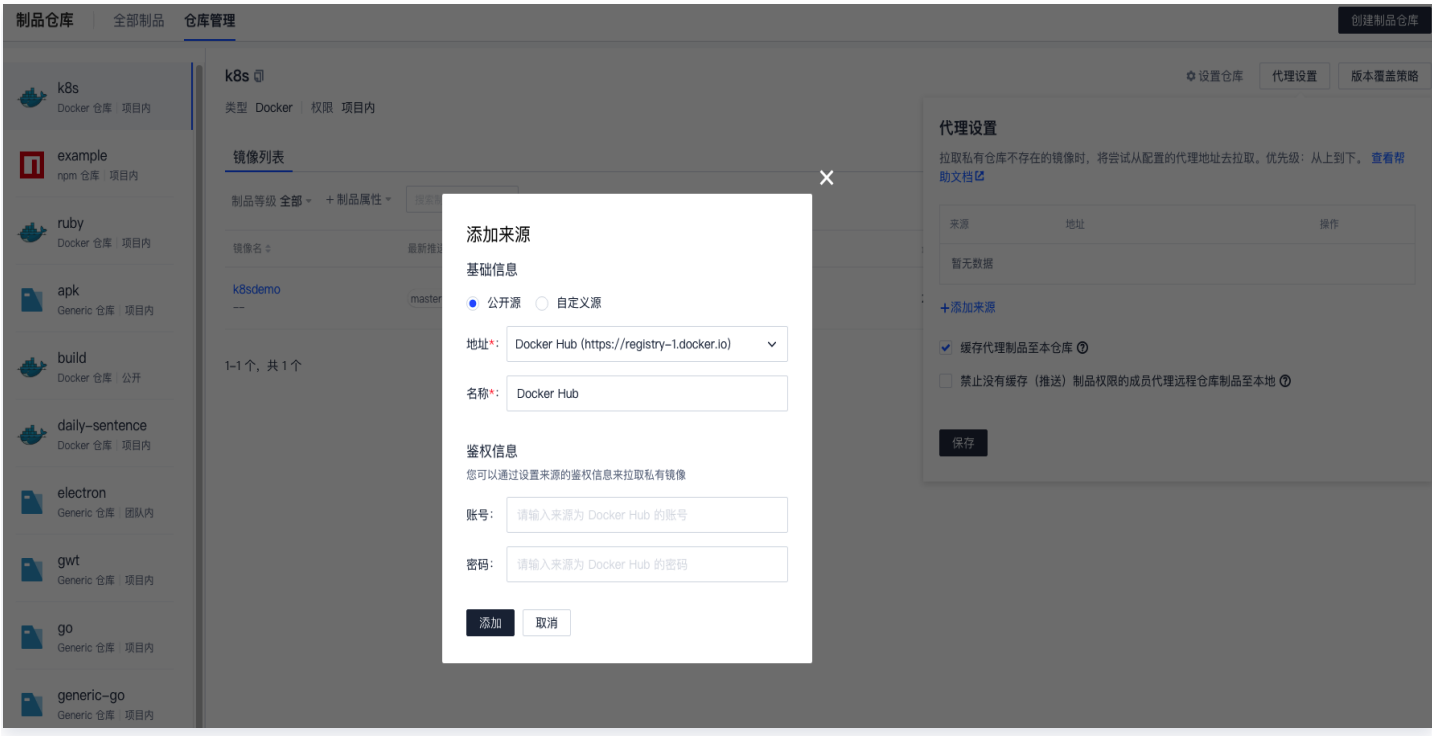
配置代理

选择指定制品库，单击**代理设置**，可以添加或删除代理来源、调整代理来源优先级、勾选是否缓存至远端制品仓库等操作。



添加来源

在单击**添加来源**后，进入创建来源页面，填写地址、名称，如有必要再填写配置鉴权信息。单击**添加**即可。



修改鉴权

如需修改代理源的鉴权信息，在代理源列表页面，单击**配置**，即可进行修改。制品库内置的代理地址如下：

类型	名称	地址
npm	npmjs	https://registry.npmjs.org
npm	TencentCloud npm	https://mirrors.cloud.tencent.com/npm
PyPI	PyPI	https://pypi.org/simple
PyPI	TencentCloud PyPI	https://mirrors.cloud.tencent.com/pypi/simple
Maven	Maven Central	https://repo.maven.apache.org/maven2
Maven	TencentCloud Maven	https://mirrors.cloud.tencent.com/nexus/repository/maven-public
Composer	TencentCloud Composer	https://mirrors.cloud.tencent.com/composer/

放行代理 IP

若出现代理拉取异常的情况，有可能是因为目标制品源拦截了来自 CODING IP 的网络请求。放行相关 IP 即可解决此问题，CODING 平台所使用的服务 IP 出口为：

```
212.129.144.0/24
212.64.105.0/24
49.234.127.0/24
49.235.224.0/24
49.234.65.0/24
81.69.101.0/24
```

使用代理

在代理配置成功后就可以使用代理源拉取依赖了。

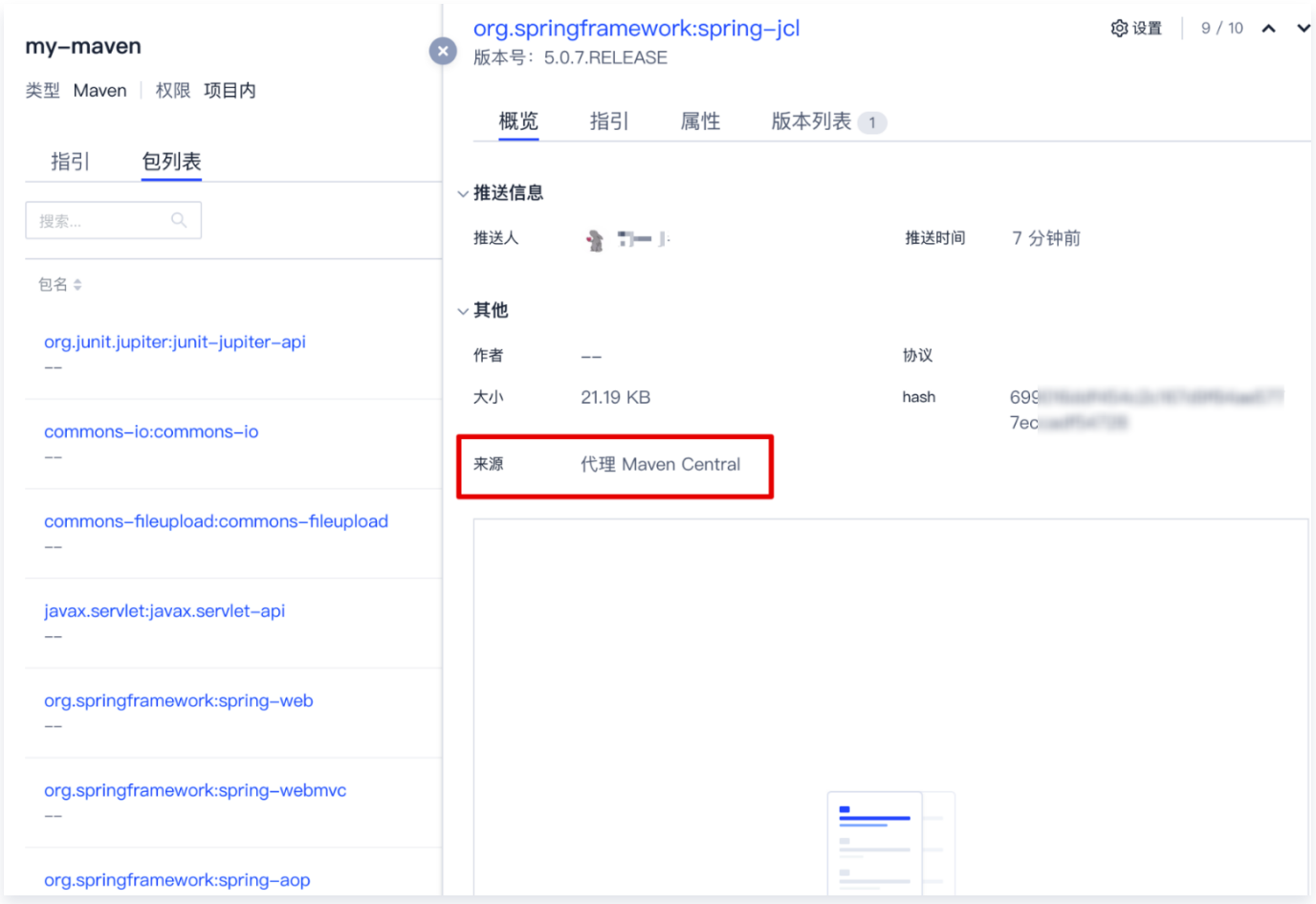
如何识别制品库中的制品来源是不是从代理同步而来的?

1. 以 Maven 类型制品为例，您可以在本地执行 maven install 时看到类似如下的制品拉取日志：

```
[INFO] Downloading from : https://xxxxxxxx-
maven.pkg.coding.net/repository/coding-demo/my-
maven/org/springframework/spring-jcl/5.0.7.RELEASE/spring-jcl-
5.0.7.RELEASE.pom
[INFO] Downloaded from : https://xxxxxxxx-
maven.pkg.coding.net/repository/coding-demo/my-
```

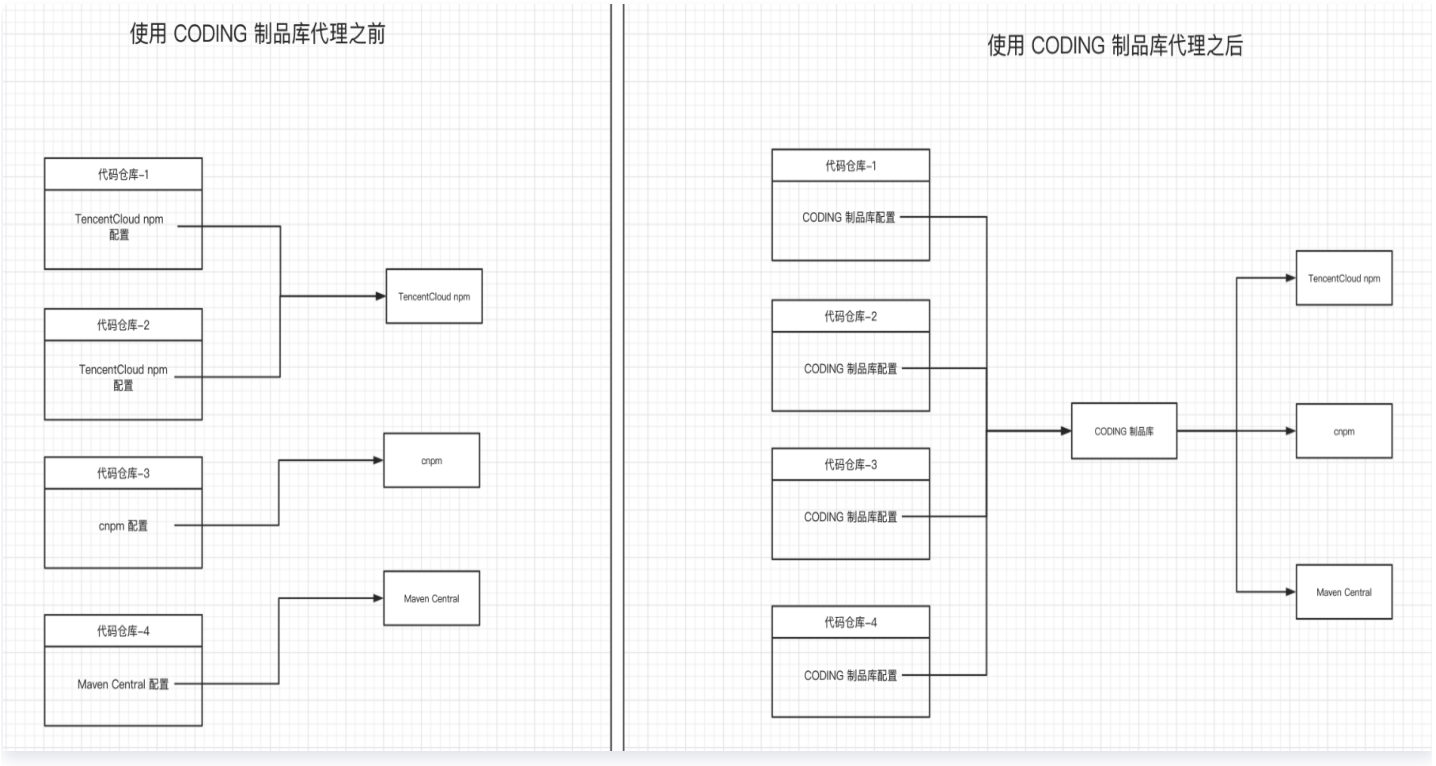
```
maven/org/springframework/spring-jcl/5.0.7.RELEASE/spring-jcl-
5.0.7.RELEASE.pom (1.9 kB at 735 B/s)
```

2. 同时，在 CODING 制品库上，您也可以看到该制品的来源。



直接从第三方制品源拉取制品和通过 CODING 制品库代理拉取有什么区别?

制品库可以帮助您统一管理团队内的制品源配置，您可以在 CODING 制品库内团队内成员的使用情况，也可以通过 CODING 制品扫描统一检测出有安全缺陷，直接对团队内的制品安全进行审计。


使用 CODING 制品库代理之前
代码仓库-1
TencentCloud npm 配置
代码仓库-2
TencentCloud npm 配置
TencentCloud npm
代码仓库-3
cnpm 配置
cnpm
代码仓库-4
Maven Central 配置
Maven Central
使用 CODING 制品库代理之后
代码仓库-1
CODING 制品库配置
代码仓库-2
CODING 制品库配置
代码仓库-3
CODING 制品库配置
代码仓库-4
CODING 制品库配置
CODING 制品库
TencentCloud npm
cnpm
Maven Central

制品库认证

最近更新时间：2024-08-02 15:23:01

功能介绍

当用户访问制品库时，制品库会对用户提供的凭证进行鉴权，以确保用户对制品库拥有操作权限。
制品库支持多种鉴权方式：

- 个人访问令牌
- 用户账号密码
- 项目令牌

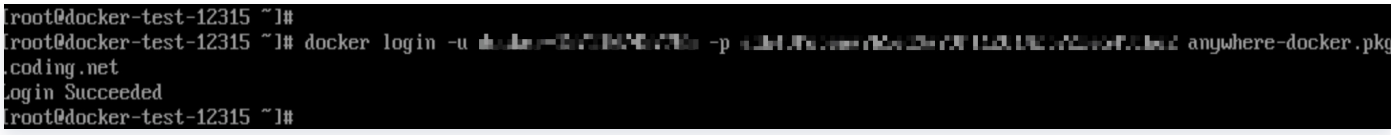
每种制品库在本地配置凭证的命令会有区别，但逻辑相似，并且在制品库页面都有设置凭证的指引。本文以 Docker 制品库为例，演示用户配置鉴权凭证的三种方式。

个人访问令牌

个人访问令牌（Access token）包含了用户的安全认证信息，利用访问令牌可以拥有查看或操作相应资源的权限。推荐使用个人访问令牌进行制品库认证，相比直接配置用户账号密码更加安全。
1. 单击制品仓库指引页面的**生成个人令牌作为凭据**。

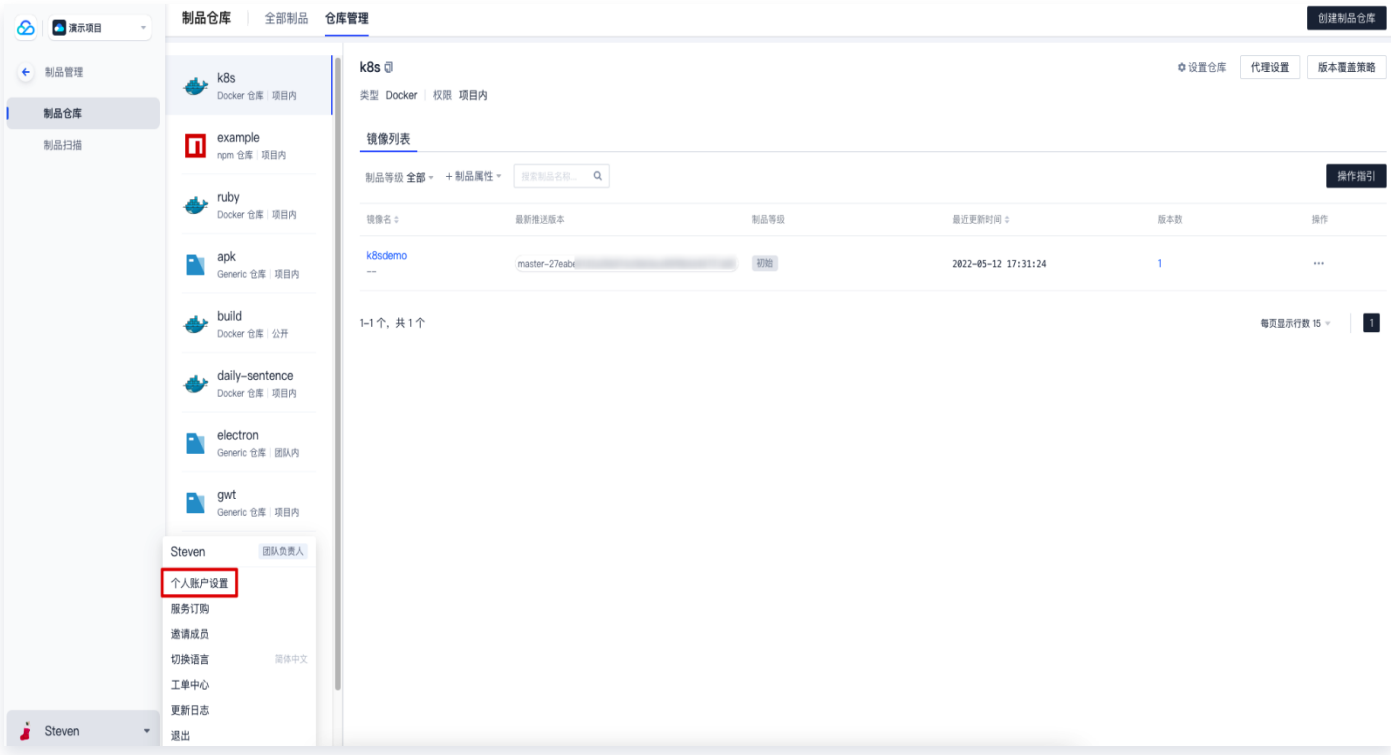
2. 输入认证信息后，即可看到已携带新访问令牌的执行命令，单击 **copy** 复制命令。

3. 在本地 docker 环境中，执行刚刚复制的 docker login 命令，提示登录成功即可进行下一步的推送/拉取操作。



查看个人令牌

1. 单击左下角头像框中的**个人账户设置**。



2. 在个人账户页面，单击**访问令牌**，可以看到上述步骤中通过制品库新建的个人访问令牌信息，在此页面您也可以看到个人访问令牌被使用的记录。

3. 单击令牌右侧的**编辑**，可以查看或修改该访问令牌的权限信息。

用户账号密码

通过用户账号（手机号或邮箱）密码也可设置凭证信息。

1. 在制品仓库指引页面，直接复制默认提供的 Docker 命令，将其中的 password 替换为个人账户登录密码。

2. 在本地 docker 页面执行命令，输入密码，即可认证成功进行下一步推送或拉取操作。

项目令牌

CODING 提供项目令牌功能，项目管理员可根据实际情况配置项目内制品仓库的推拉权限。具体操作请参见 项目令牌 。

制品晋级

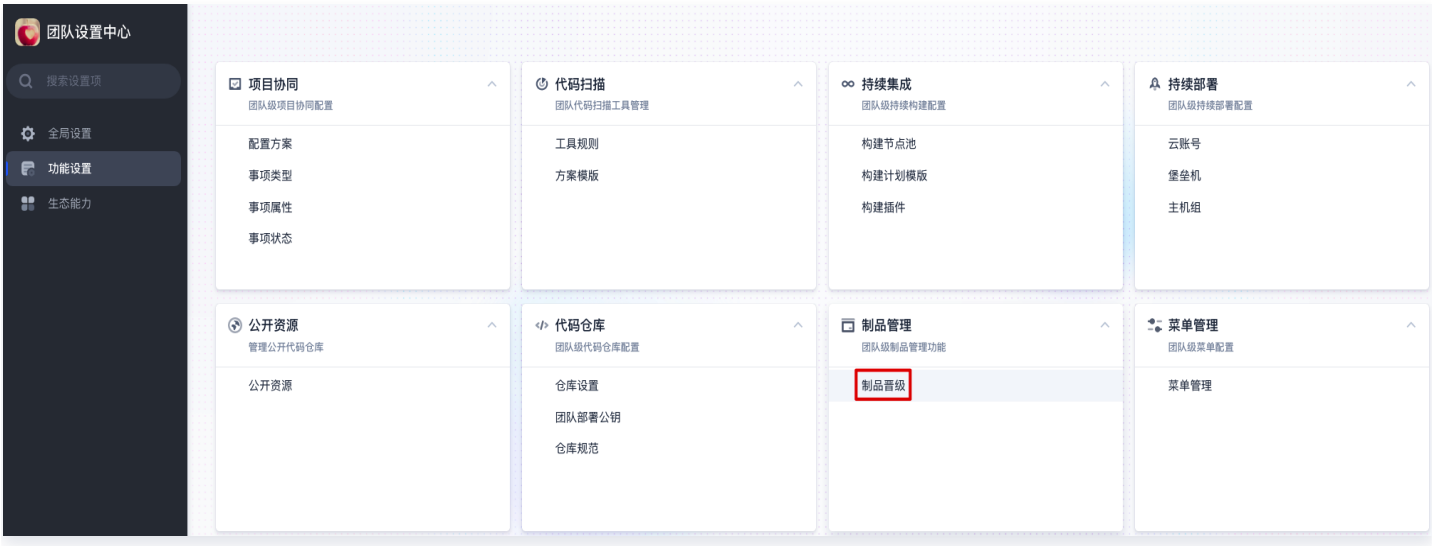
最近更新时间：2024-08-02 15:23:01

功能介绍

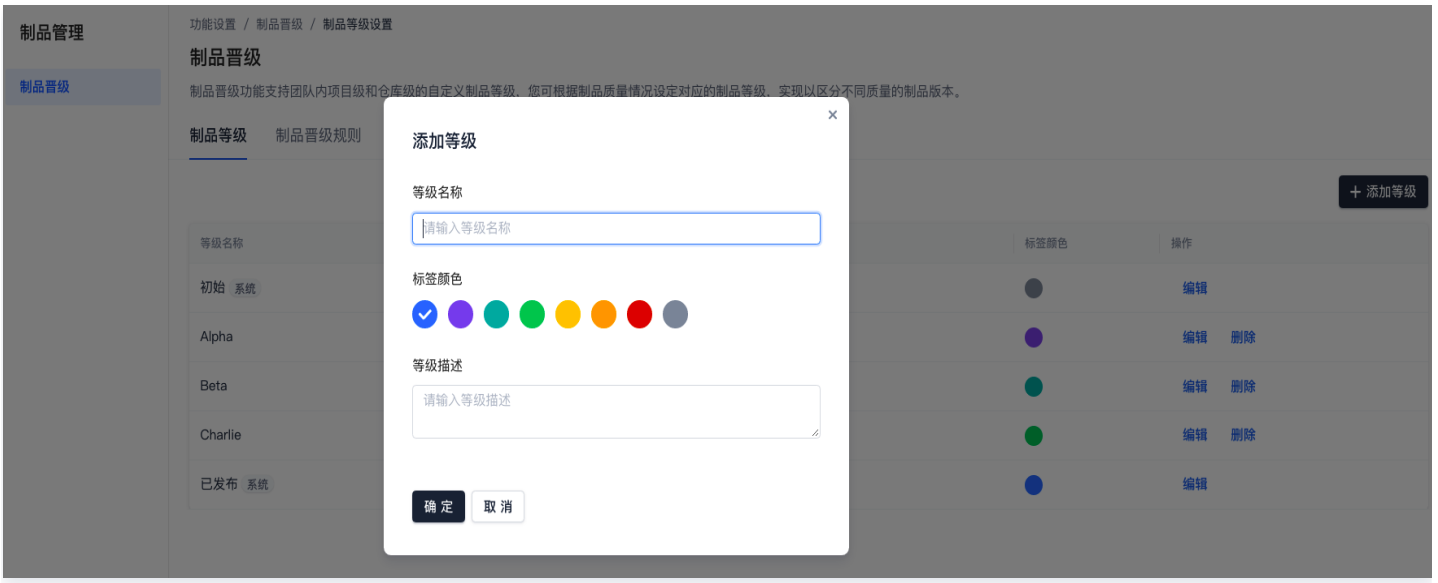
仅通过简单的制品版本号无法有效判断制品是否达到了可交付水平。制品晋级功能提供自定义制品等级和晋级规则，帮助研发团队有效且直观地区分制品版本的成熟度情况，使得最终交付的制品版本是合格且可信赖的。团队成员能够根据业务需求，为制品仓库配置不同的晋级规则，完善制品质量及版本区分。

设定制品等级

类似于项目协同中的事项流转，制品晋级本质上是在直观描述制品的交付状态。前往**团队设置中心 > 功能设置 > 制品晋级**创建制品等级。



您可以按照测试交付流程定义制品等级，例如初始 > A 测 > B 测 > 待发布 > 已发布；或根据团队内部对于制品质量的共识制定等级，输入名称并勾选颜色后完成创建。

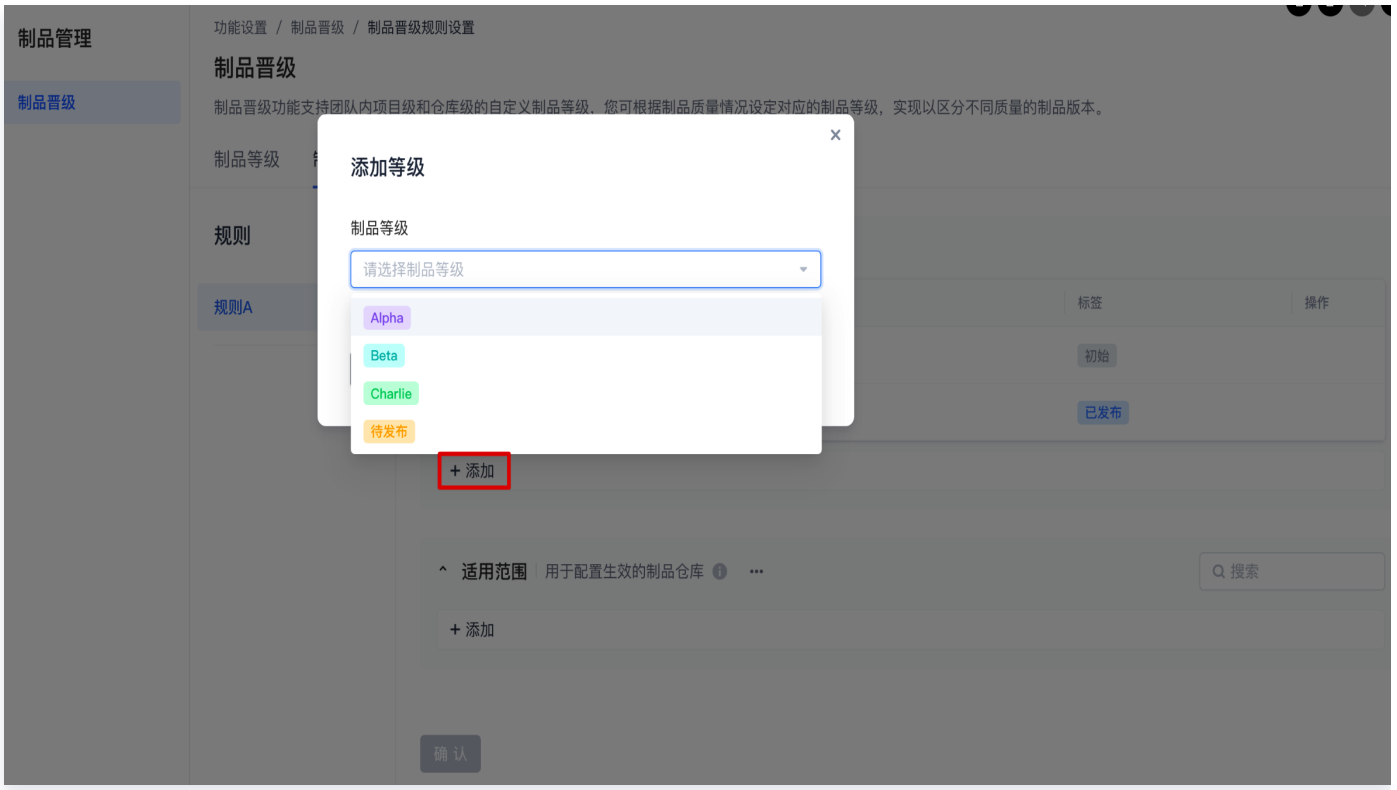


等级创建完成后，还需要继续设定晋级规则才能够应用至制品仓库。

设定制品晋级规则

晋级规则用于定义制品等级的流转顺序，例如规定制品状态需从**初始**等级历经各项测试后才能达到**待发布**等级。进入制品晋级选项后，前往**制品晋级规则**页创建晋级规则。

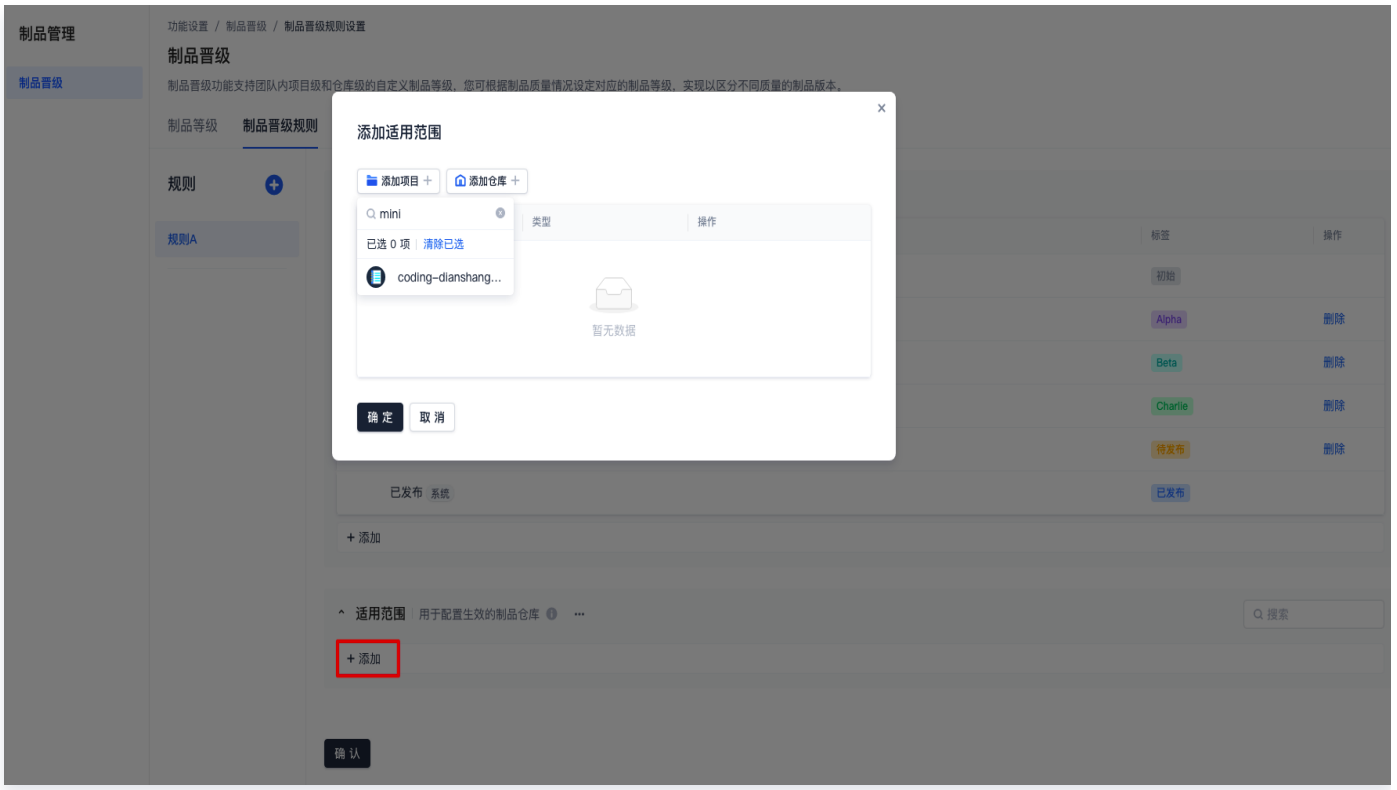
1. 将制品等级圈选至**晋级规则**中。



2. 拖拽左侧按钮调整等级的流转顺序。

3. 勾选适用该晋级规则的制品仓库。完成后单击下方的**确认**进行保存即可。

> **说明：**
> 单个项目或制品仓库支持应用多项规则，在实际的制品流转过程中任意选择一项等级。

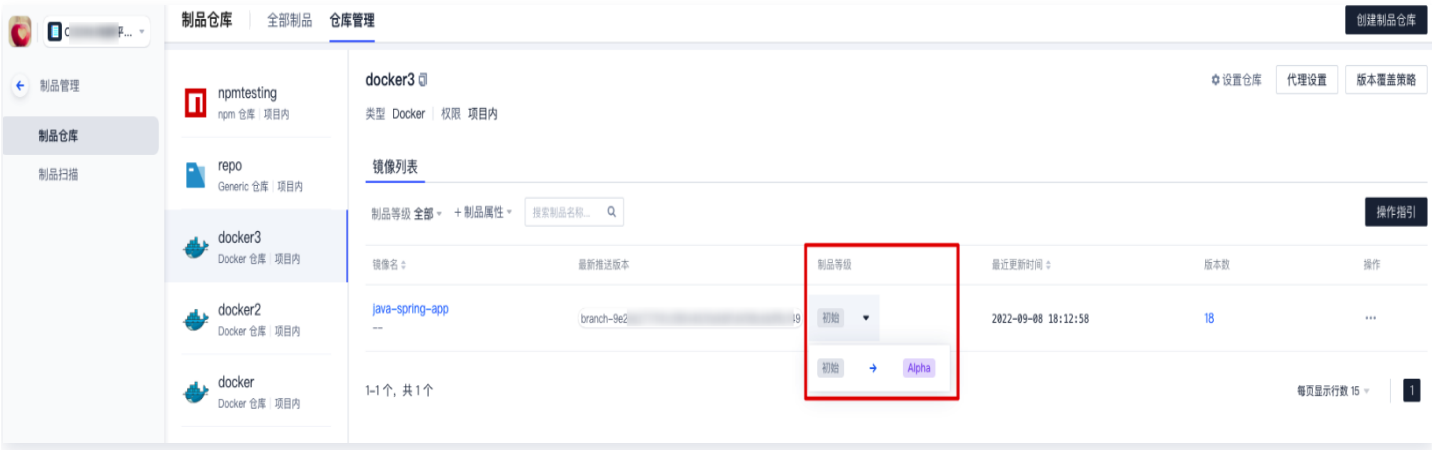


应用制品晋级规则

在制品管理中的制品列表与制品历史版本页调整制品等级，方便团队成员掌握制品目前的所属状态，快速了解当前版本的制品质量是否达到了待发布等级。

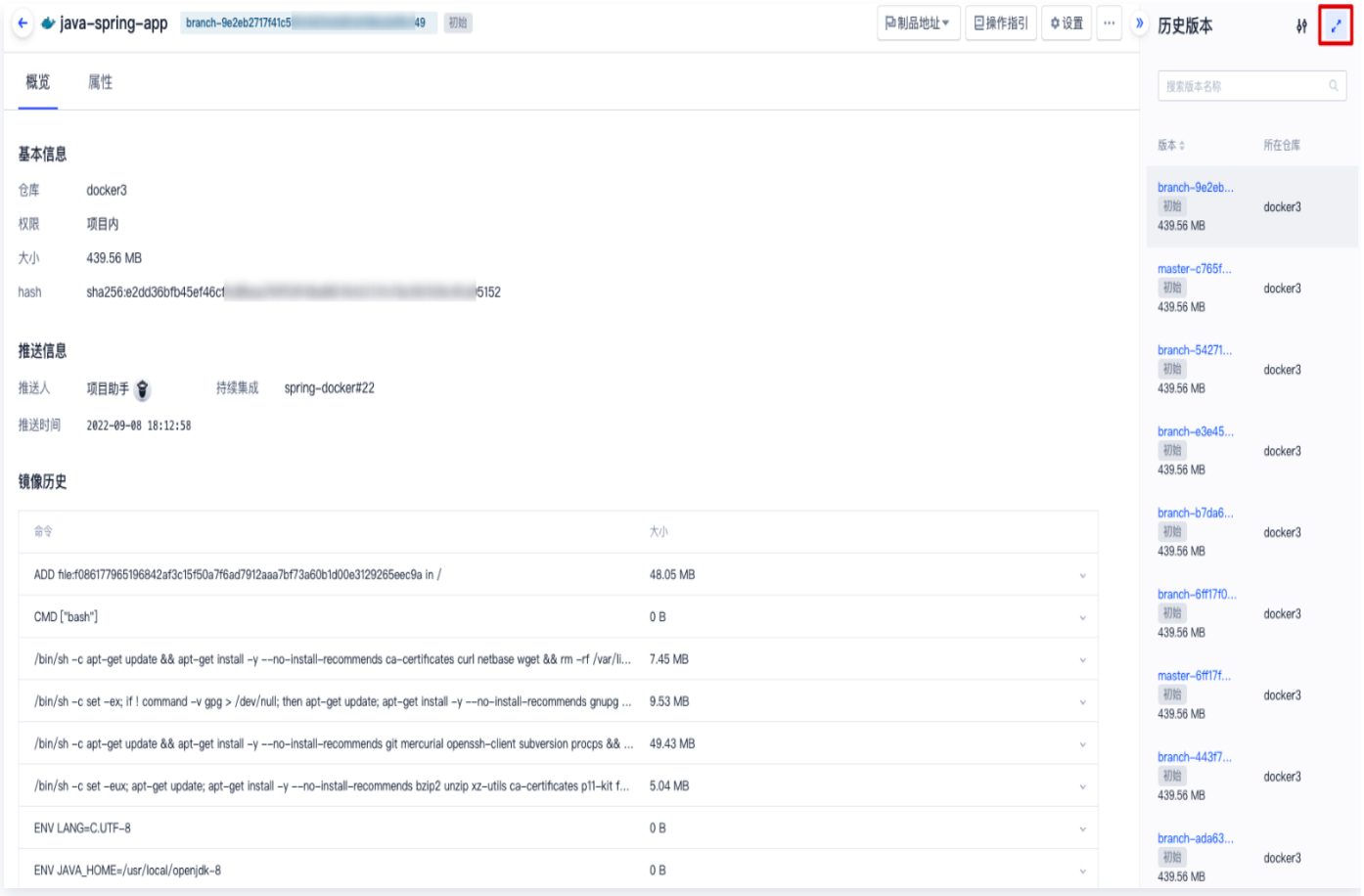
制品列表

进入仓库管理页，在列表中手动调整制品等级。

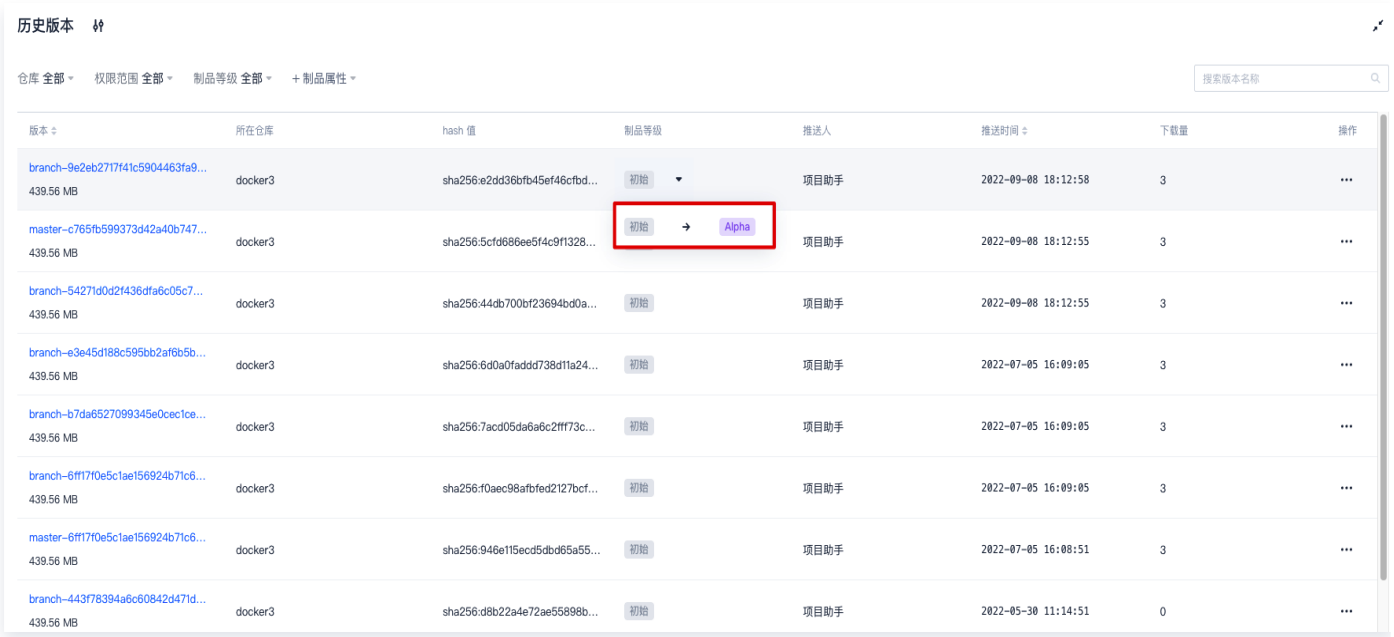


历史版本

1. 单击制品概览右上角的全屏按钮，展开查看制品的历史版本。



2. 在制品等级中手动调整制品状态。



制品版本覆盖策略

最近更新时间：2024-08-02 15:23:01

在开发阶段，每一次的代码修改都有可能会产生新的制品。开发人员可能需要依赖方频繁修改版本号来使用最新的版本。这将会非常不利于开发调试，因为如果在生产阶段随意覆盖同一个制品的版本，可能会带来管理上的混乱。**让制品拥有唯一的版本号，可以保障同一个版本的制品永远保持相同的行为，这对于部署及应用生命周期管理而言都非常有意义。**

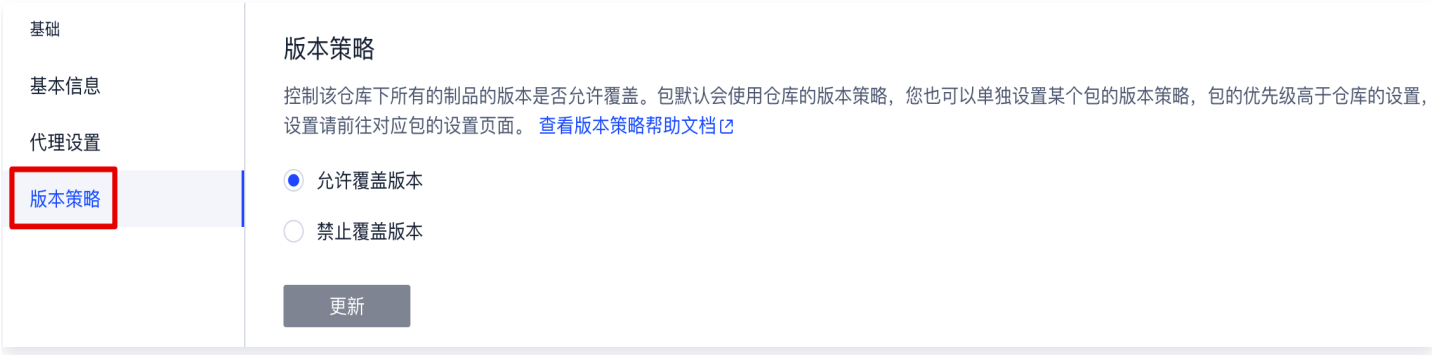
CODING 制品库提供了灵活的版本覆盖策略，可以保障 Docker 镜像版本的唯一，也可以重复发布同一个 npm 包的版本。您可以根据需要，针对制品的生命周期，设置**仓库/包/版本** 的策略。

接下来本文按照这三个层级来介绍如何设置制品版本覆盖策略、以及制品库提供的默认版本覆盖策略。

仓库版本覆盖策略

单击制品库 > 设置仓库。

单击版本策略，此处可设置该仓库下所有制品的版本是否允许覆盖。

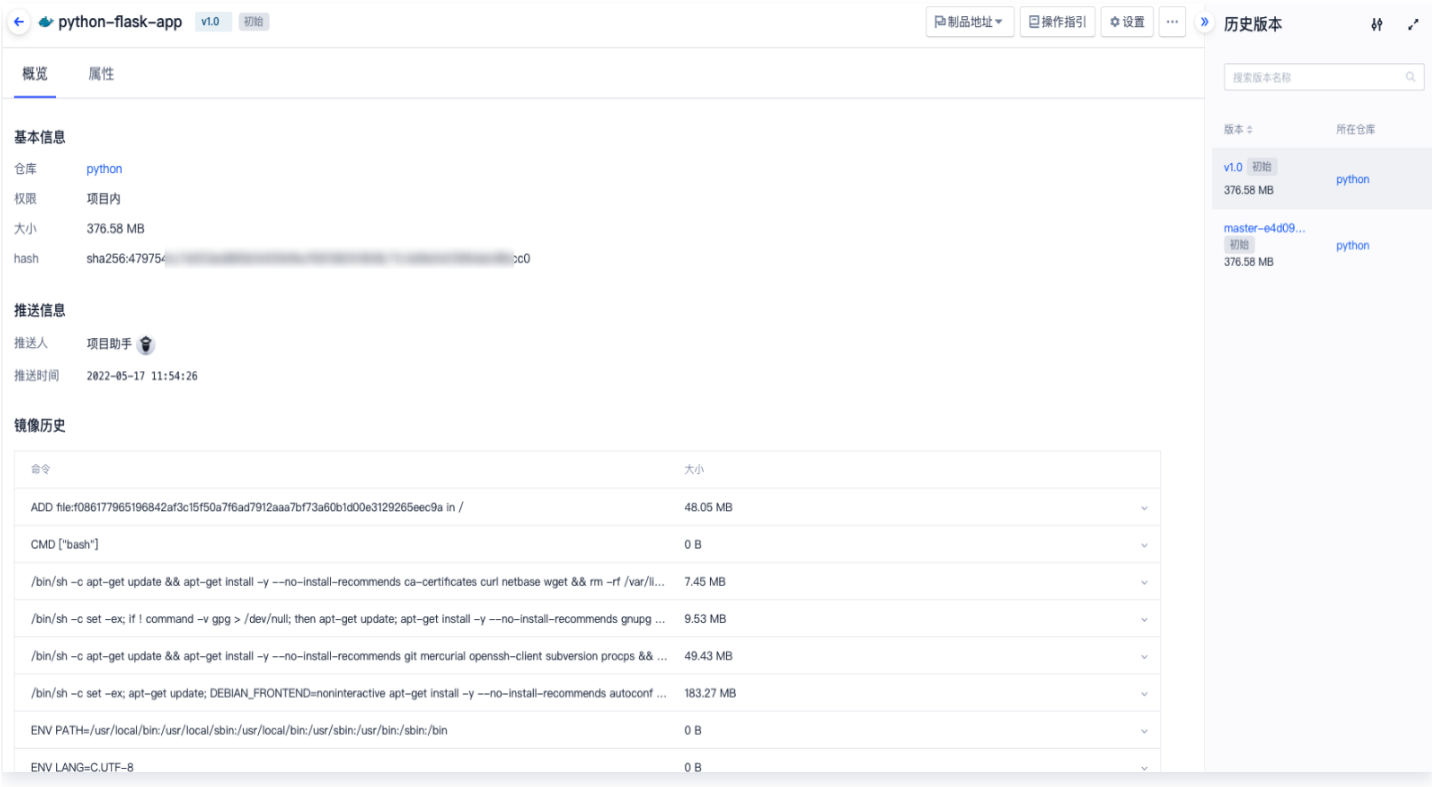


说明：

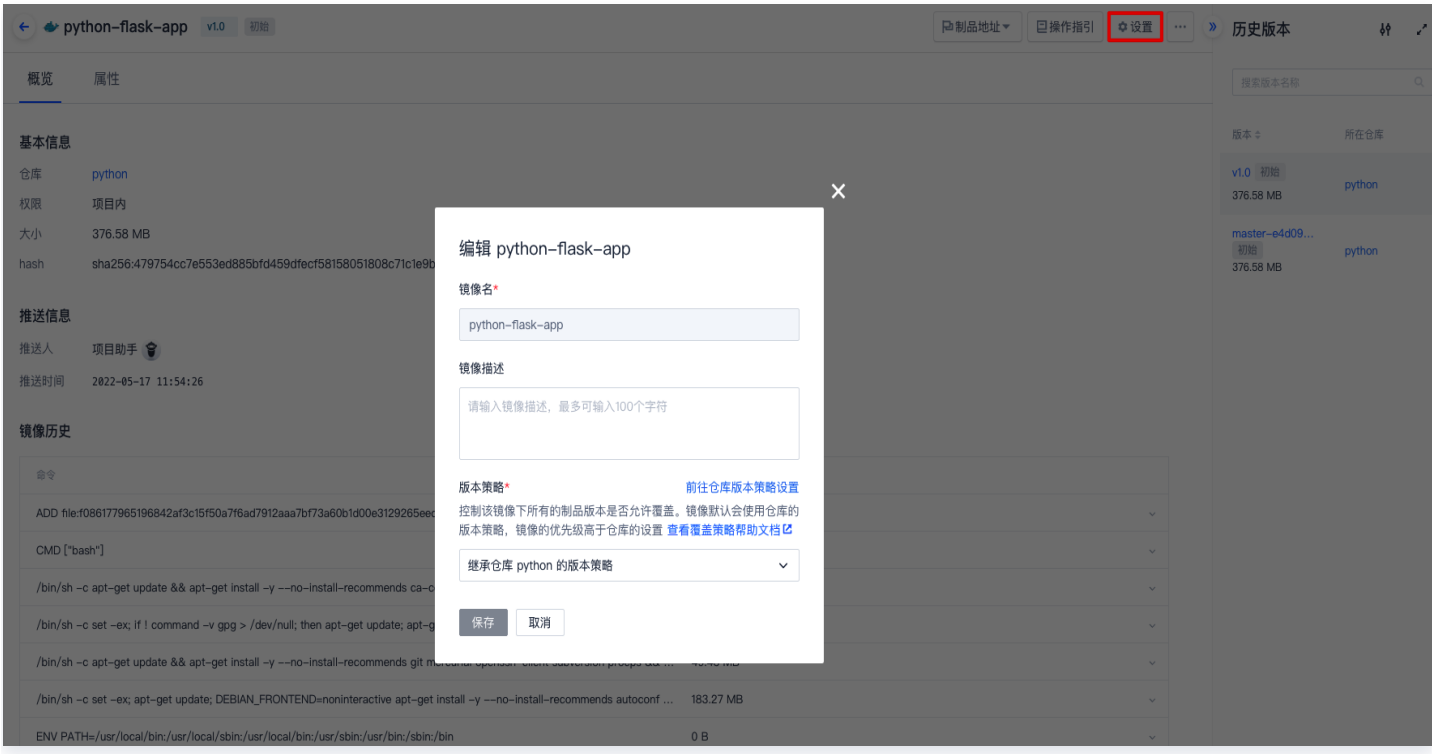
目前 Maven 较为特殊，多了一项：使用 Maven SNAPSHOT（快照）。

包的版本覆盖策略

单击具体包名可看到右侧的包详情页面。



单击**设置**即可对包的版本策略进行选择，默认情况下包使用本仓库的版本覆盖策略。



默认的版本覆盖策略

制品库按照该制品类型的原生逻辑，提供了默认的版本覆盖策略，详情如下：

制品类型	仓库	包	版本
Docker	允许发布相同版本	继承仓库规则	未发布
Maven	Maven SNAPSHOT	继承仓库规则	未发布
npm	不允许发布相同版本	继承仓库规则	未发布
PyPI	不允许发布相同版本	继承仓库规则	未发布
Generic	允许发布相同版本	继承仓库规则	未发布
Helm	允许发布相同版本	继承仓库规则	未发布
Composer	不允许发布相同版本	继承仓库规则	未发布
NuGet	不允许发布相同版本	继承仓库规则	未发布
Conan	允许发布相同版本	继承仓库规则	未发布

制品清理策略

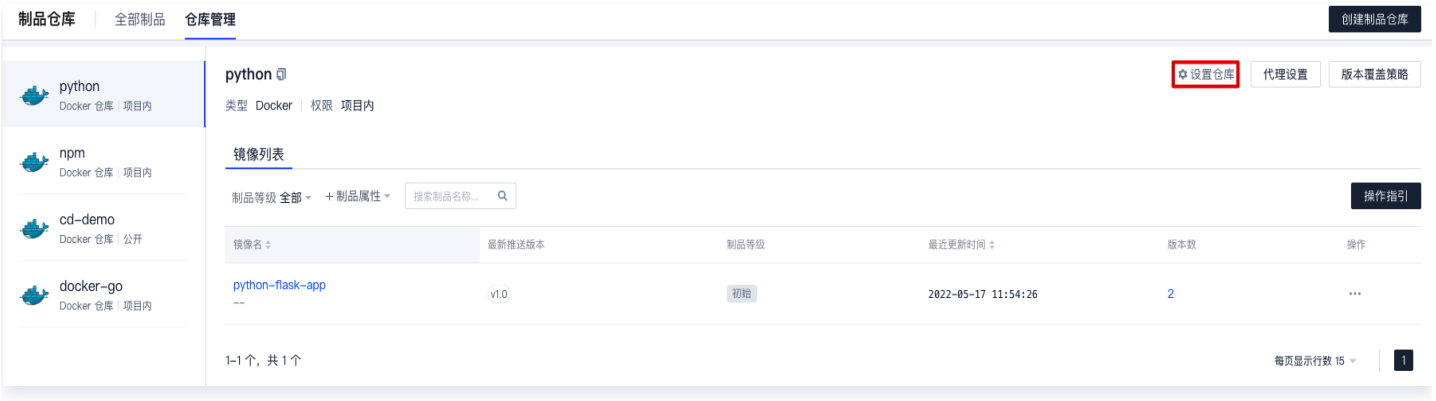
最近更新时间：2023-09-11 16:05:27

制品仓库的清理策略能够及时清理老旧版本的制品。您可以通过设置清理策略快速清理多余制品，释放储存空间。目前支持清理策略设置的制品类型：

- Docker
- Generic
- Gradle
- Helm
- npm

配置清理策略

进入上述类型的制品仓库页后，单击**设置仓库 > 清理策略**。

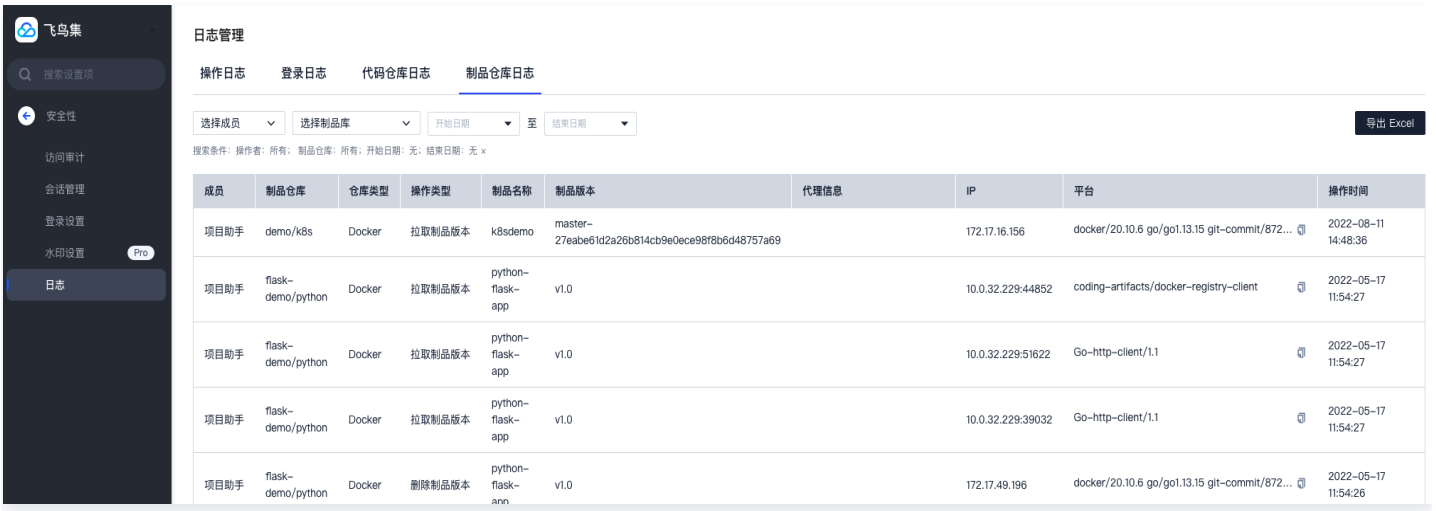


您需要先填写清理设置中的两个触发条件，再选择执行自动或手动清理，只有同时符合两个触发清理条件的制品才会被纳入清理列表。

查看清理记录

进入团队设置中心，在日志中查看制品仓库的操作日志。

在**制品仓库日志**页将显示近期的制品仓库日志。


飞鸟集
搜索设置项
安全性
访问审计
会话管理
登录设置
水印设置
Pro
日志
日志管理
操作日志
登录日志
代码仓库日志
制品仓库日志
选择成员
选择制品库
开始日期
至
结束日期
导出 Excel
搜索条件：操作者：所有； 制品仓库：所有；开始日期：无；结束日期：无 ×
成员 制品仓库 仓库类型 操作类型 制品名称 制品版本 代理信息 IP 平台 操作时间
项目助手 demo/k8s Docker 拉取制品版本 k8sdemo master-27eabe61d2a26b814cb9e0ece98f8b6d48757a69 172.17.16.156 docker/20.10.6 go/go1.13.15 git-commit/872... 2022-08-11 14:48:36
项目助手 flask-demo/python Docker 拉取制品版本 python-flask-app v1.0 10.0.32.229:44852 coding-artifacts/docker-registry-client 2022-05-17 11:54:27
项目助手 flask-demo/python Docker 拉取制品版本 python-flask-app v1.0 10.0.32.229:51622 Go-http-client/1.1 2022-05-17 11:54:27
项目助手 flask-demo/python Docker 拉取制品版本 python-flask-app v1.0 10.0.32.229:39032 Go-http-client/1.1 2022-05-17 11:54:27
项目助手 flask-demo/python Docker 删除制品版本 python-flask-app v1.0 172.17.49.196 docker/20.10.6 go/go1.13.15 git-commit/872... 2022-05-17 11:54:26

制品扫描
功能简介

最近更新时间：2025-03-17 15:33:52

> **说明：**
> CODING DevOps 于2025年9月1日起更新 CODING 订购方案，取消原标准版套餐，下线部分功能（**制品安全扫描、测试管理、测试协同、仪表盘、研发度量**），新注册团队用户界面无**持续部署、应用管理**功能，为确保您的使用权益和资产数据安全，请及时关注并处理，了解更多详情 。

CODING 制品仓库的扫描功能可以在不访问源代码的情况下，通过扫描二进制组件及其元数据，找寻组件中存在的漏洞。制品扫描功能支持与持续集成/持续部署模块相集成。您可以在方案中预设质量红线标准，杜绝问题组件发布至生产环境。同时，扫描方案还支持提供详细扫描记录和缺陷统计。

产品功能

扫描方案

用于规定扫描规则、质量红线，以及被执行扫描的制品包。扫描方案只能应用于当前项目内的制品仓库，每个扫描方案都有唯一的扫描 ID。扫描方案可以应用于当前项目下的任意仓库、任意制品包与任意制品版本。触发方式可以选择制品包更新时自动触发或选择制品版本手动触发。

扫描规则

用于规定扫描时关注的制品缺陷内容，可以指定关注的漏洞等级并设置漏洞白名单。

- 漏洞危险等级规则：

  定义该扫描方案关注的漏洞危险等级。若不勾选**低危**等级，则所有低危等级的漏洞都不会被扫描出来，不计入漏洞统计。

- 漏洞白名单（仅安全扫描）：

  将具体的 CVE 漏洞编号纳入至白名单后，所有公共组件中如果存在该编号的 CVE 漏洞都不会被扫描；将具体所属依赖组件列入白名单后，该组件中的所有漏洞都不会被扫描。

质量红线

用于规定扫描结果是否通过的标准。您可以根据团队的安全需求定义红线标准，包括对漏洞、开源许可证等级和数量限制。

组成成分分析（SBOM）

目前仅针对 Generic、Maven、Docker、npm 类型制品提供分析功能。
组成成分分析（SBOM）功能可以帮助企业分析制品中的依赖组件，并提供各组件的版本、漏洞、license 统计等信息。与此同时，系统还会分析制品与依赖的关联关系，包括组件依赖的组件及组件所关联的制品，为软件供应链风险追溯和安全治理提供有效依据。

开源许可证风险

开发者在使用开源软件的过程中需注意许可证是否存在违规风险。制品扫描在执行漏洞扫描的同时也会对开源组件许可证进行扫描，展示许可证风险等级、来源、约束与关联组件等信息。

方案应用

单个扫描方案可以复用于各类型的制品仓库中的任意制品版本，触发方式支持制品包更新时自动触发与手动触发。

方案类型

扫描方案支持安全漏洞扫描与移动端安装包质量检查两种类型。移动端安装包质量检查方案由**腾讯云安装包质量检查 IPT** 提供主要能力。除了两种类型外，制品扫描还将继续拓展更多制品扫描能力。

- **安全漏洞扫描：**

  扫描制品及其依赖的公共组件中存在的安全漏洞。漏洞的定义采用 CVE 国际通用标准及评级；制品扫描采用的组件漏洞特征库由云鼎实验室维护，为腾讯安全提供超过 20 年的行业经验积累，由专门的安全运营团队实时更新。

  支持以下类型制品仓库：Docker、Maven、npm、PyPI、Generic。
- **移动端安装包质量扫描：**

  对 Android / iOS 操作系统的安装包进行安装包大小、重复文件、图片等进行规范检查。

  支持 Generic 类型制品仓库中 .ipa 和 .apk 格式的制品文件。

安全漏洞

漏洞的定义采用 CVE 国际通用标准定义，其危险等级遵循 CVSS 标准。CODING 制品扫描通过对制品及其依赖解析，暴露该制品存在的安全漏洞。漏洞信息包括 CVE 编号、引入依赖组件、危险等级及漏洞描述等。
漏洞的定义由 CVE 编号及其存在的公共组件共同决定。例如，我们在公共组件 Log4j 2.17 版本中，发现了编号为 CVE-2021-45105 的安全漏洞，则这条漏洞在扫描结果中会显示为：

- CVE 编号：CVE-2021-45105
- 所属依赖：Log4j
- 引入版本：2.17

制品漏洞库

CODING 制品扫描采用腾讯安全漏洞特征库。
腾讯安全漏洞特征库是腾讯科恩实验室自研的、长期维护的商业化制品漏洞库。在漏洞信息方面，包含了国内外的开源漏洞库信息（包括：CVE、NVD、CNVD、 CNNVD 等）与腾讯自身发现的独有商业漏洞信息。腾讯安全团队对开源漏洞信息进行了误报校验与中文翻译，同时提供修复优先级与修复建议信息，扫描准确性行业领先。对于自身发现的独有商业漏洞信息，提供独有漏洞 ID、漏洞详情描述、漏洞修复建议等内容，涵盖漏洞基本信息。
对于私有化客户，我们提供了实时更新和离线更新两种更新方式。

实时更新

在实时更新方面，支持在本地漏洞库配置 国内互联网更新地址 和腾讯云账号密钥认证，在每次扫描执行的时候，会检查漏洞库是否为最新版本。如果不是最新版本，则会先更新漏洞库再执行扫描。

离线更新

在离线更新方面，也支持将漏洞库离线打包部署至企业内网，提供更新服务。

快速开始

最近更新时间：2024-08-02 15:23:01

本文以示例制品 fastjson 作为扫描对象，若项目内已有制品，可以直接进行扫描。
得益于 CODING 制品代理功能，在拉取制品至本地时将自动上传至制品仓库。单击 示例制品 获取相关信息，若不清楚如何使用 Maven 制品仓库，请参见 快速开始 Maven 制品库 。

创建扫描方案

前往**制品管理 > 制品扫描**，点击左上角的蓝色 + 号创建一个新的扫描方案，输入扫描方案名称、描述、扫描规则和质量红线标准，即可完成创建。

方案类型

扫描方案支持安全漏洞扫描与移动端安装包质量检查两种类型；使用安全漏洞扫描类型能够筛选漏洞等级与 CVE 漏洞白名单。移动端安装包质量检查方案由**腾讯云安装包质量检查 IPT** 提供主要能力。

扫描规则

扫描规则决定了该扫描方案中能够被检查出来的漏洞。若仅勾选了**高危**的漏洞等级，则其他等级的漏洞将不会被统计，即使存在高于**高危**等级的漏洞也不会被纳入检查。

CVE 漏洞白名单

用于在扫描过程中忽略特定类型的漏洞。您可以给某种类型的漏洞定义一个 CVE 漏洞编号并在白名单处填写，若制品存在 CVE-2020-9794 类型的漏洞则不会被纳入统计范围，单击查看 CVE 漏洞收录 。

> **说明：**
> 在扫描过程中，漏洞扫描仅会在扫描规则中勾选的等级范围中进行，其他等级均会被过滤，然后才会判断 CVE 漏洞白名单中规定的具体漏洞编号。例如用户在漏洞扫描中只选择了**危急**的漏洞等级，而又在 CVE 漏洞白名单中填入了一个**低危**等级的漏洞，此等级的漏洞会被忽略。

高级选项

在高级配置中可以自动禁止未扫描完成或存在质量问题的制品被下载，防止存在漏洞的制品被意外使用。

编辑扫描方案

对于已创建的扫描方案，单击右上角的**设置**或**规则配置**进入设置页面。

您可以在此处编辑方案名称或描述、查看重新勾选扫描规则中的漏洞等级和质量红线标准。

触发扫描方案

扫描方案支持自动与手动两种触发方式。

自动扫描

单击右上角的•••，进入方案应用页，您可以在此处配置自动触发扫描方案。

打开自动扫描开关后，当前方案所应用的制品仓库有更新时将自动触发扫描。
扫描筛选支持全部和按条件：

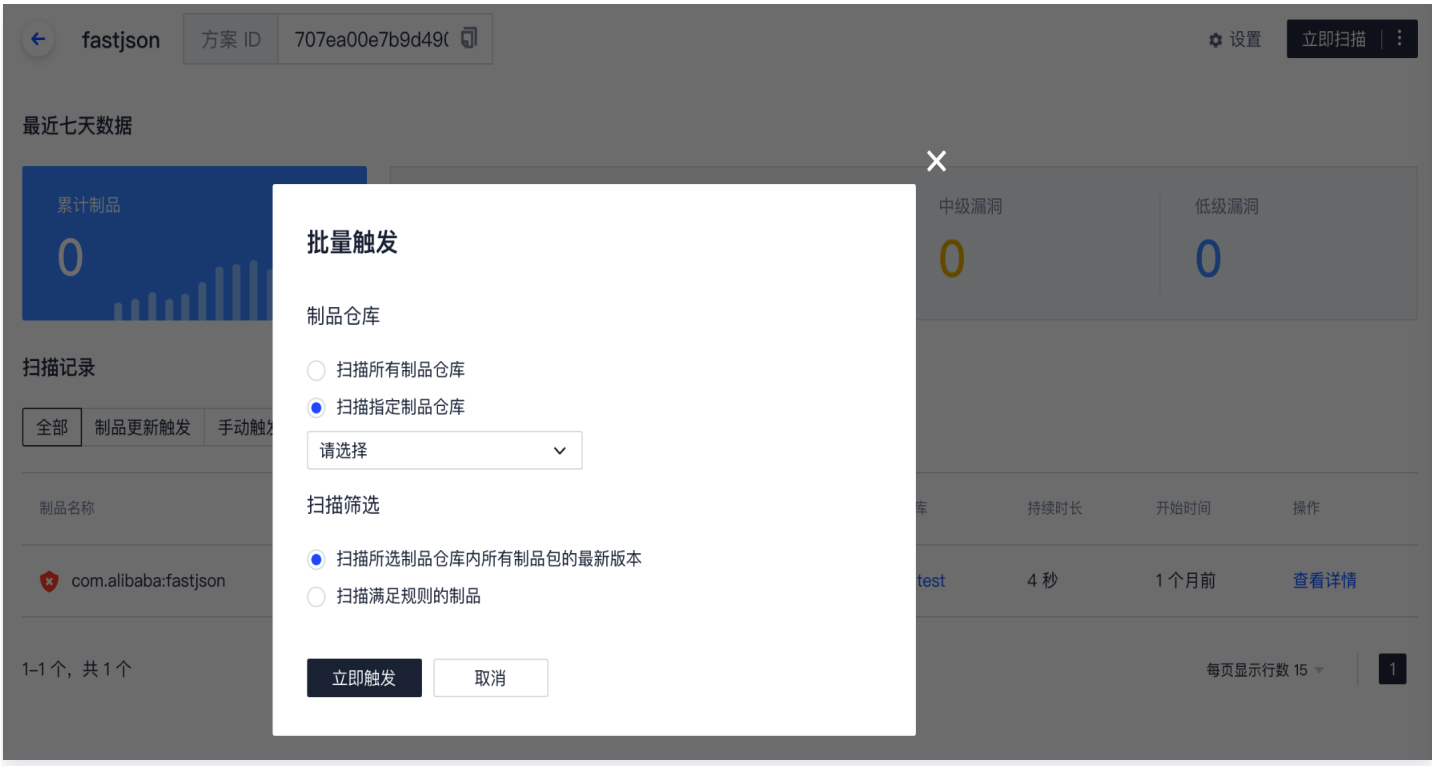
- 默认为**全部**，即任一制品仓库有更新时会自动触发此扫描方案。
- 勾选**按条件筛选**后，可以为扫描方案设置应用范围与触发条件。例如下图，当 pypi-go 和 write-go 制品仓库有任何制品更新时就会自动触发该扫描方案。

您还可以对扫描方案添加更加细致的制品筛选条件，如下图当 test 制品的 release 版本有更新时，将对此制品单独进行扫描。

手动扫描

有三种手动触发扫描的方式：触发单个扫描方案、批量触发制品扫描与在持续集成流水线中添加制品扫描。

单击扫描方案右上角的批量扫描可以在全部制品仓库中执行扫描，或设置扫描范围与扫描筛选条件。



在持续集成流水线中也支持手动添加制品扫描单元。

分析扫描结果

图标标识

各标示结果的图标意义：

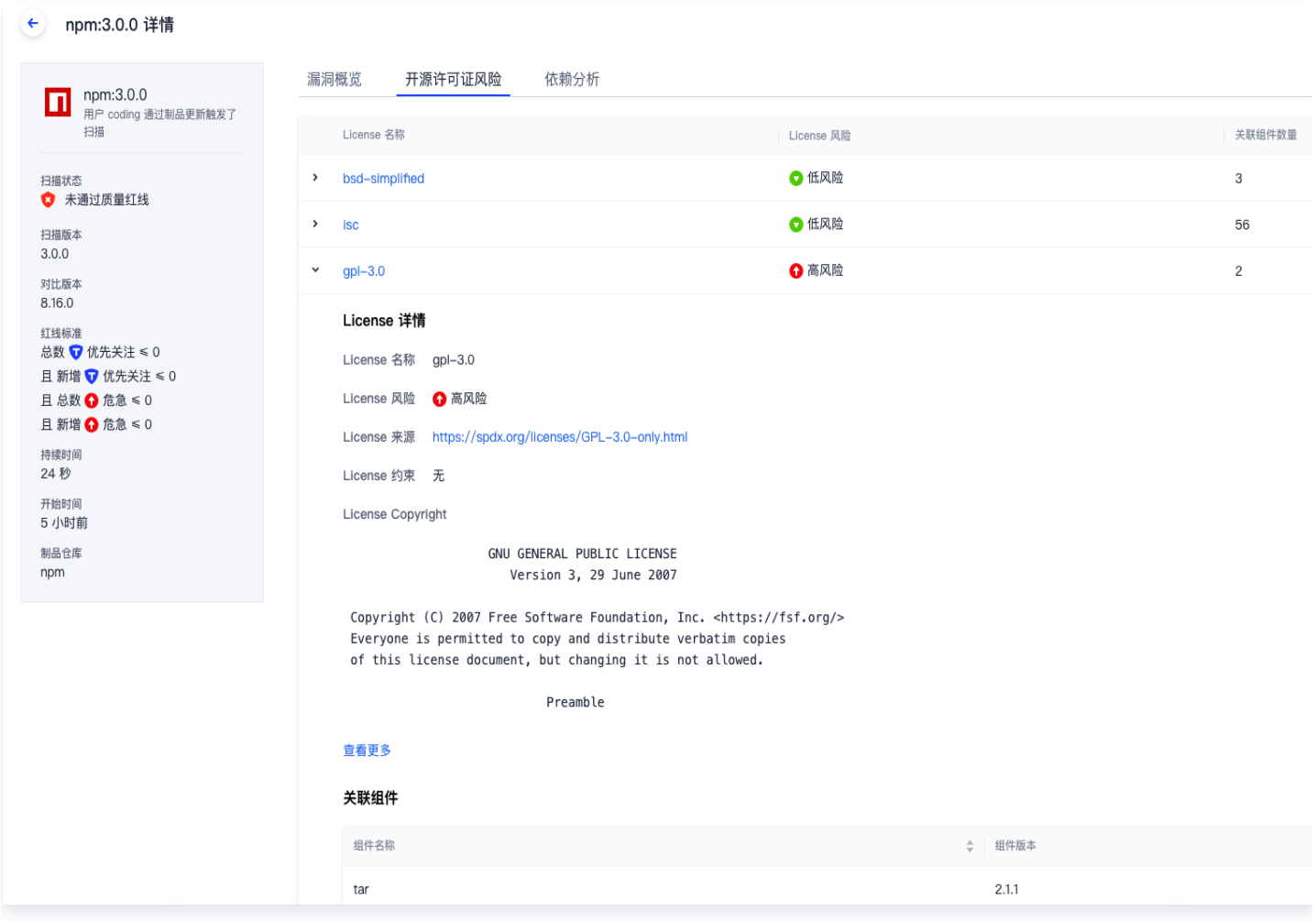
将鼠标移至标示处将会弹窗扫描结果。

查看扫描结果

进入扫描方案详情，您可以看到该扫描方案被应用的所有历史累计数据和扫描记录。

开源许可证风险

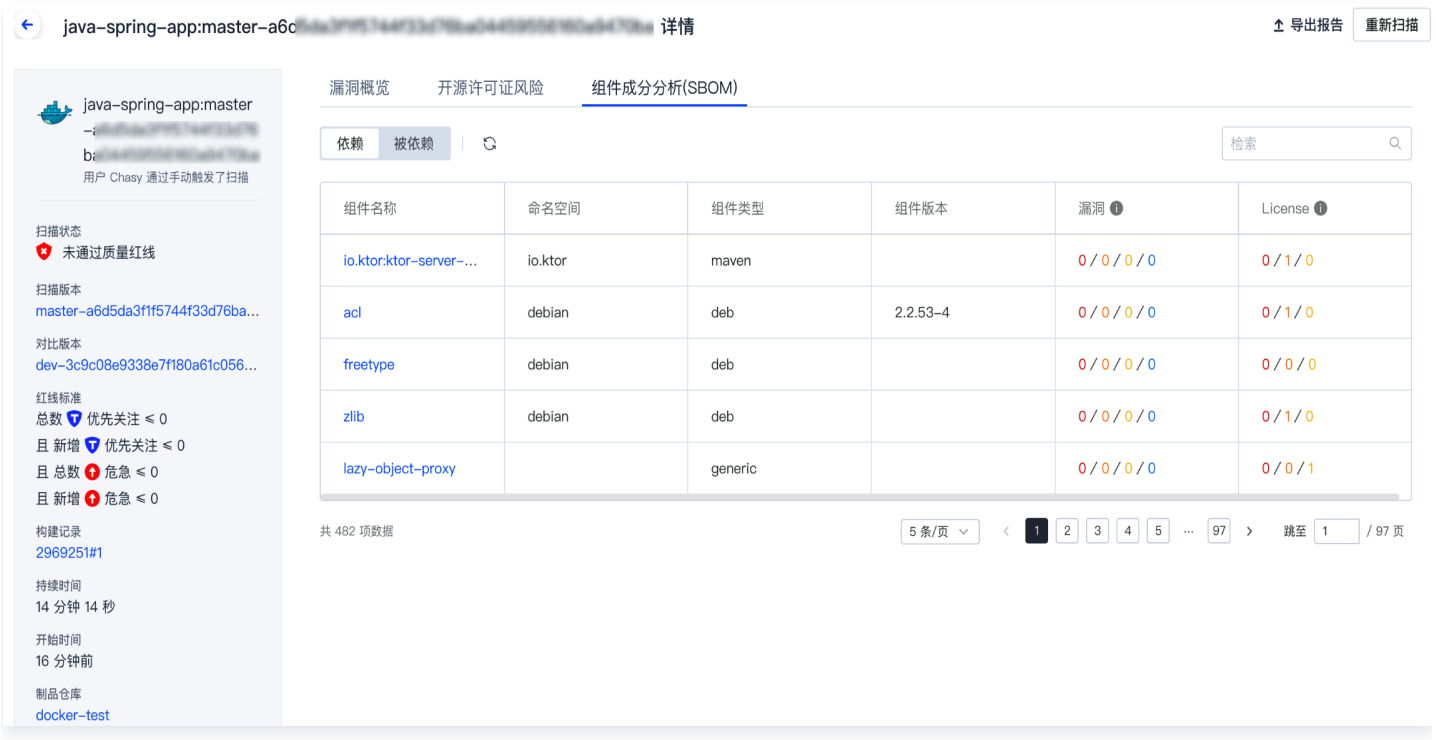
单击**开源许可证风险**可以查看该制品所有组件的开源 license 名称、风险等级、来源信息与关联组件等。



组成成分分析（SBOM）

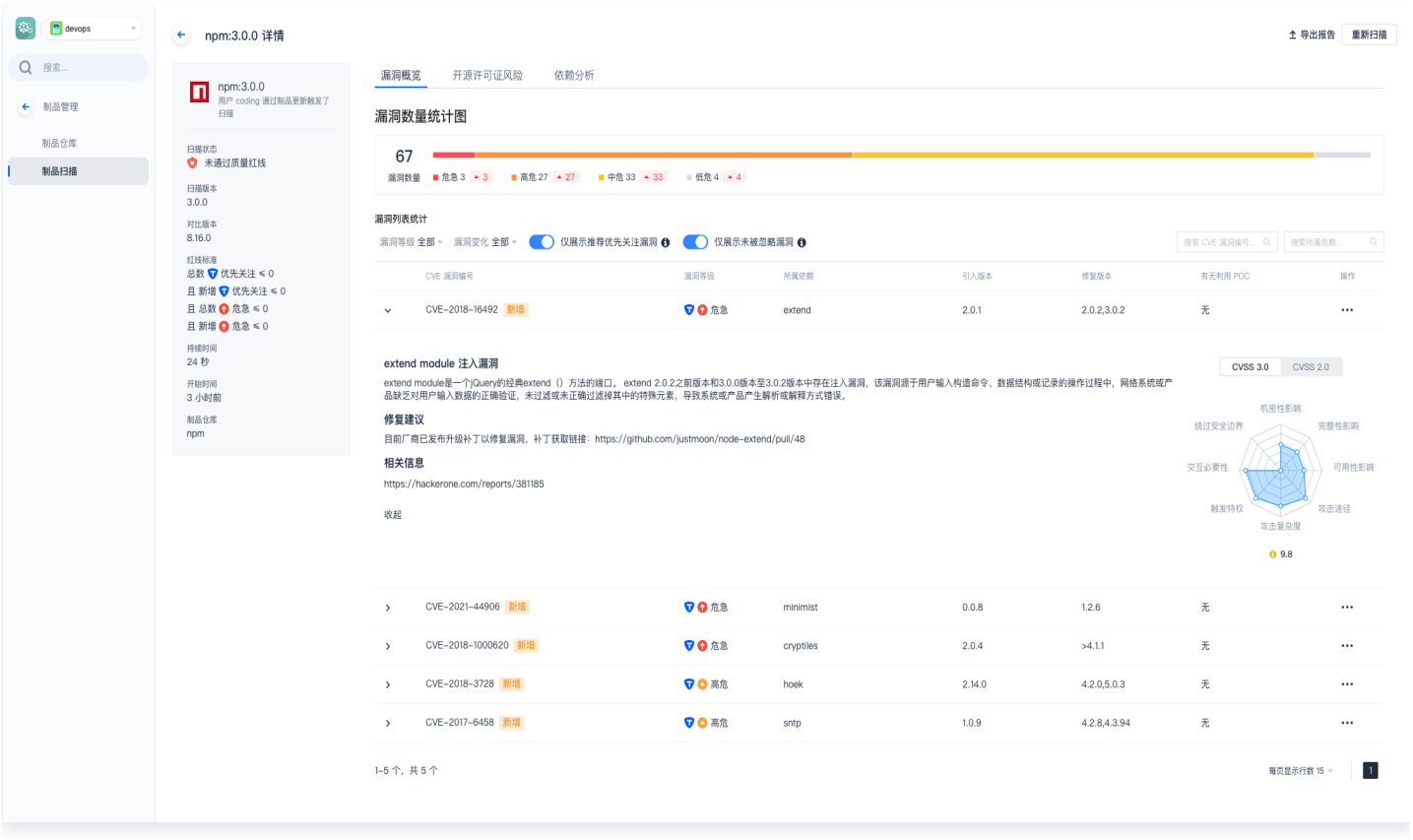
目前仅针对 Generic、Maven、Docker、npm 类型制品提供分析功能。

组成成分分析（SBOM）功能可以帮助企业分析制品中的依赖组件，并提供各组件的版本、漏洞、license 统计等信息。与此同时，系统还会分析制品与依赖的关联关系，包括组件依赖的组件及组件所关联的制品，为软件供应链风险追溯和安全治理提供有效依据。



解决漏洞并记录

单击**查看详情**或制品名称即可查看该制品版本的漏洞详情。详情页展示了所有漏洞的详细信息，包括漏洞编码、等级、所属依赖、版本、修复建议等。



针对已暴露的漏洞，您可以按照漏洞概览中的修复建议进行漏洞修复。

若漏洞并不会造成较为严重的后果，可以将其记录为忽略；若已对漏洞采取修补动作，可以将其记录为修复。