制品库

常见问题

产品文档

腾讯云

常见问题

最近更新时间：2023-02-23 17:47:13

制品库支持什么类型?

支持包括 Docker、Maven、npm、Genric、Pypi、Helm 等常见制品库类型。

制品库的层级关系是怎样的?

制品库的层级关系为：**仓库 > 包 > 版本**，每个层级描述如下：

- 仓库：用于管理不同类型的仓库和仓库下的包资源，可以设置仓库对外的访问权限。
- 包：构建产物对外提供访问的基础单元，用于介绍当前构建产物的用途和使用指引。
- 版本：列出某个包下的所有构建产物，详细记录了每次构建产物的版本迭代更新变化。

制品库对外的权限是怎样的?

- 项目内：本项目成员可读和写。其他成员不可读和写。
- 团队内：本项目成员可读和写。企业内其他成员可读不可写。其他成员不可读和写。
- 公开：本项目成员可读和写。非本项目成员和匿名成员可读不可写。

制品库包名称的规则是什么?

包名仅支持1 - 31位英文、数字、下划线（_）、中划线（-）、点（.）的组合。不可与本仓库其他包名称重复，可以其他仓库内包名重复。

包的设置项包含什么?

设置项包含：许可证、包描述、成熟度、Web 站点 URL、问题跟踪 URL、版本控制 URL 等。

Maven 的 settings.xml 配置文件在哪?

在生成 Maven 类型制品时，您需要配置您的 settings 文件，通常这个文件存放的位置有如下几个地方，您都可以按需使用，只不过配置生效的范围和优先级不同：

1. 全局配置：${M2_HOME}/conf/settings.xml
   如果您不记得 Maven 的安装目录 ${M2_HOME}，您可以在终端中执行 echo ${M2_HOME} 或者 mvn -version 就可以看到 Maven home 的路径。
2. 用户配置：${user.home}/.m2/settings.xml
   您可以通过 echo 环境变量的方式找到该文件目录，有时候这个目录下是没有 settings.xml 文件，您可以去全部配置里拷贝一份 settings.xml 再进行修改。
3. 指定路径下的 settings.xml。

```
mvn deploy --settings settings.xml
```

> 说明：
> 在终端执行 mvn 相关命令时，settings.xml 配置生效的优先级：指定路径 > 用户配置 > 全局配置。

Maven 仓库如何配置私服账号密码?

在代码仓库中新增 settings.xml 文件，配置以下内容：

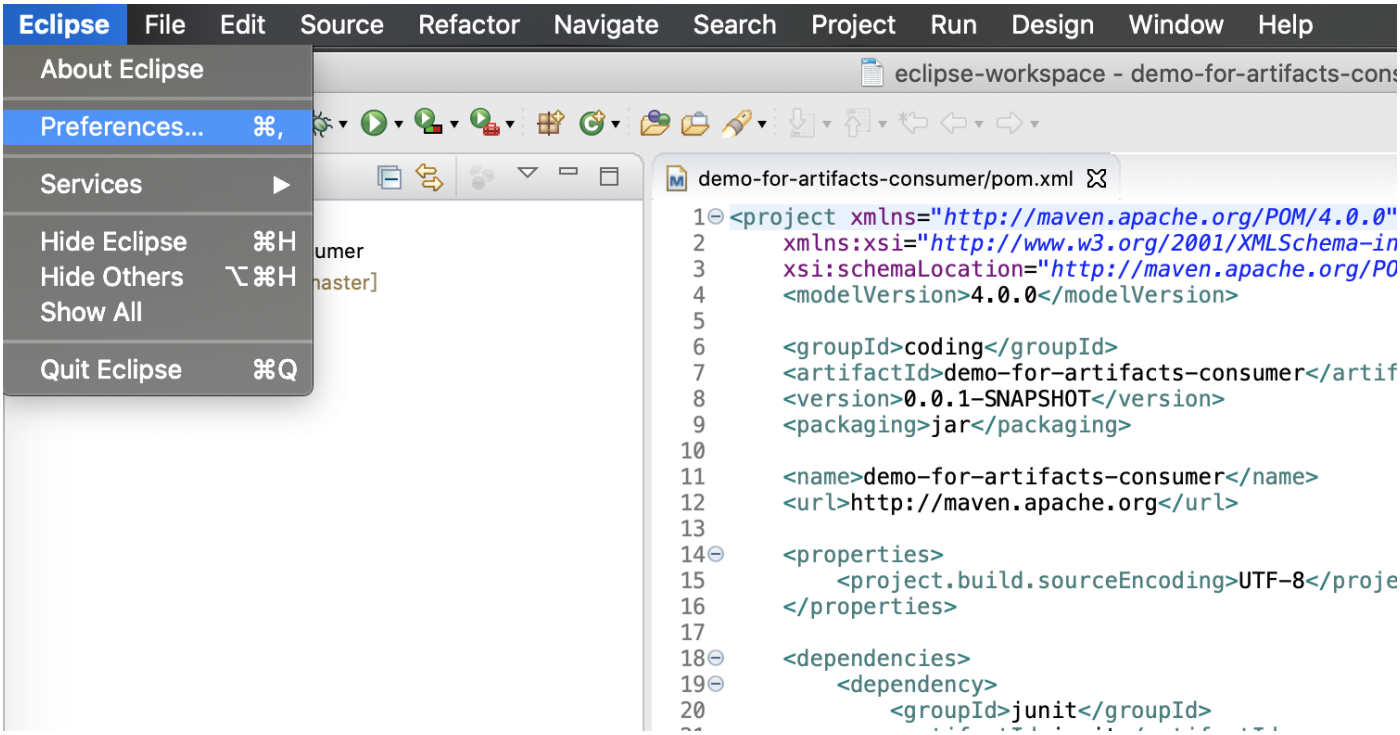
```xml BRACKET-FILTER
<!-- omitted xml -->
<servers>
<server>
<id>您的私服 ID</id>
<username>您的私服账号</username>
<password>您的私服密码</password>
</server>
</servers></settings>
```

在 pom.xml 中配置 repository 信息：
xml BRACKET-FILTER <repository> <!--必须与 settings.xml 的 id 一致--> <id>您的私服 ID</id> <name>私服仓库名</name> <url>私服仓库 URL</url></repository>

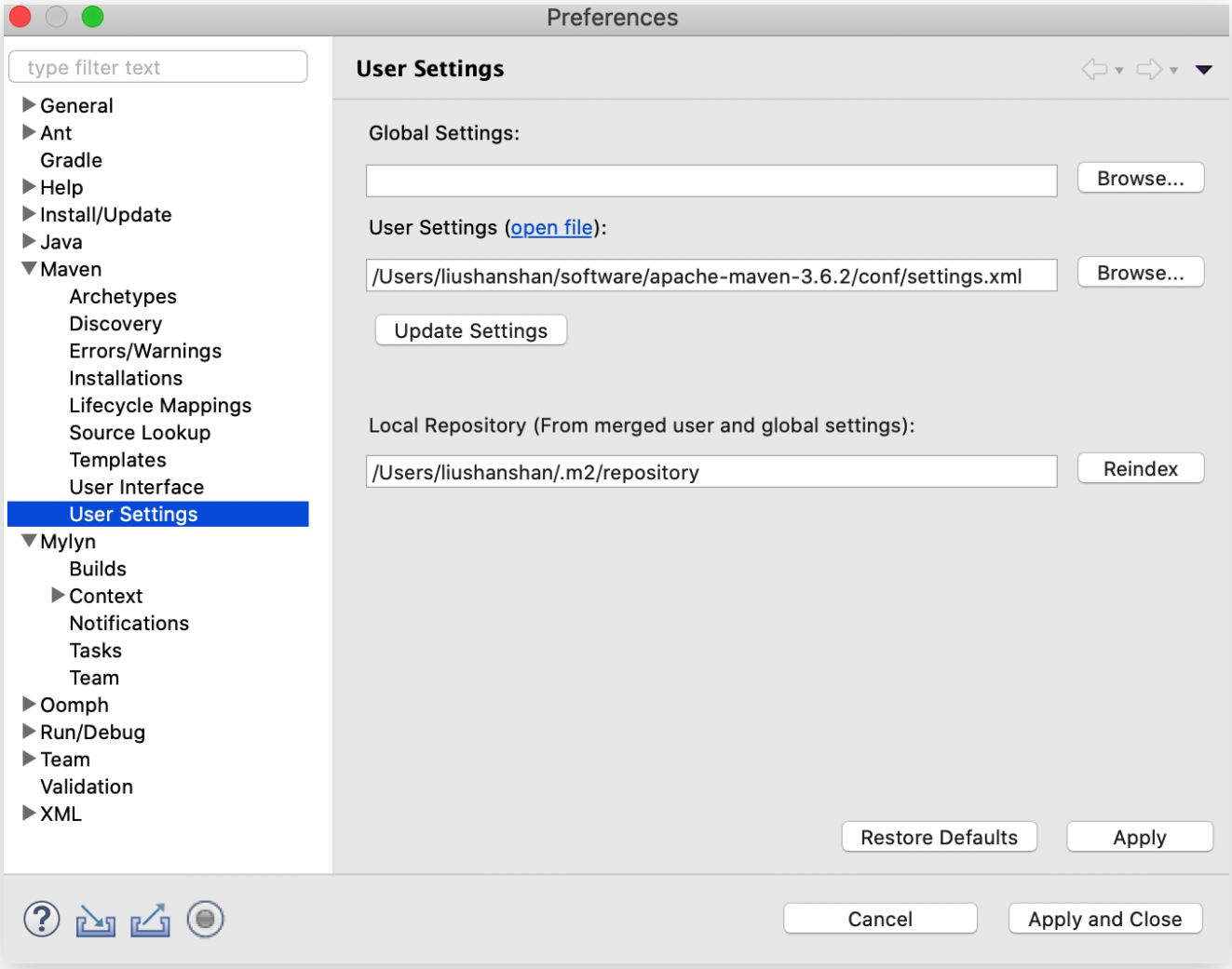
如何在 IDE 中修改 settings.xml 文件配置?

以 Eclipse 为例（其它类型 IDE 网上也有丰富的资料供参考）：

1. 单击Preferences进入偏好设置。



2. 在Maven > User Settings当中您就可以看到您使用的配置文件路径，并且修改配置文件。



如何将 npm @scope 指向 CODING 私有制品库?

1. 可以通过配置 **.npmrc** 来指定 @scope 的 registry。

   例如: 有一个 npm 包，位置信息如下：

   - 企业： my-team
   - 项目： my-project
   - 制品仓库： my-npm-repo
   - 名： @my-scope/my-pkg

   可以通过配置 .npmrc，让 package.json 中的 @my-scope/my-pkg 指向该链接地址：

   ```
   https://my-team-npm.pkg.coding.net/my-project/my-npm-repo/
   ```

2. 直接将 npm 包的 registry 指向 CODING 制品库。

   直接单击 npm 制品库指引页面中的**使用访问令牌生成配置**生成 .npmrc。

请妥善保管生成的配置：

```
registry=https://my-team-npm.pkg.coding.net/my-project/my-npm-repo/
always-auth=true
//my-team-npm.pkg.coding.net/my-project/my-npm-repo/:username=xxxxxx
//my-team-npm.pkg.coding.net/my-project/my-npm-repo/:_password=xxxxx
//my-team-npm.pkg.coding.net/my-project/my-npm-repo/:email=xxxx
```

由于 CODING 的 npm 制品库支持 代理功能，可以直接将 npm registry 设置为 CODING 制品库，公共制品也可以被拉取到。

> 说明：
> 关于如何在 CODING 持续集成中使用 npm 制品库，可参见 持续集成> 构建 npm 类型制品。

如何拉取其它 CODING 项目制品库的制品?

您可以通过 项目令牌 的方式拉取其它 CODING 项目制品库的制品。
为了方便您区分即将要操作的两个不同项目，我们统一将：

- 需要被拉取制品的制品库所在项目称为 “项目 A”。
- 执行拉取的项目称为 “项目 B”。

步骤一：在项目 A 内创建项目令牌

1. 进入项目 A **项目设置 > 开发者选项 > 项目令牌**，单击**新建项目令牌**。

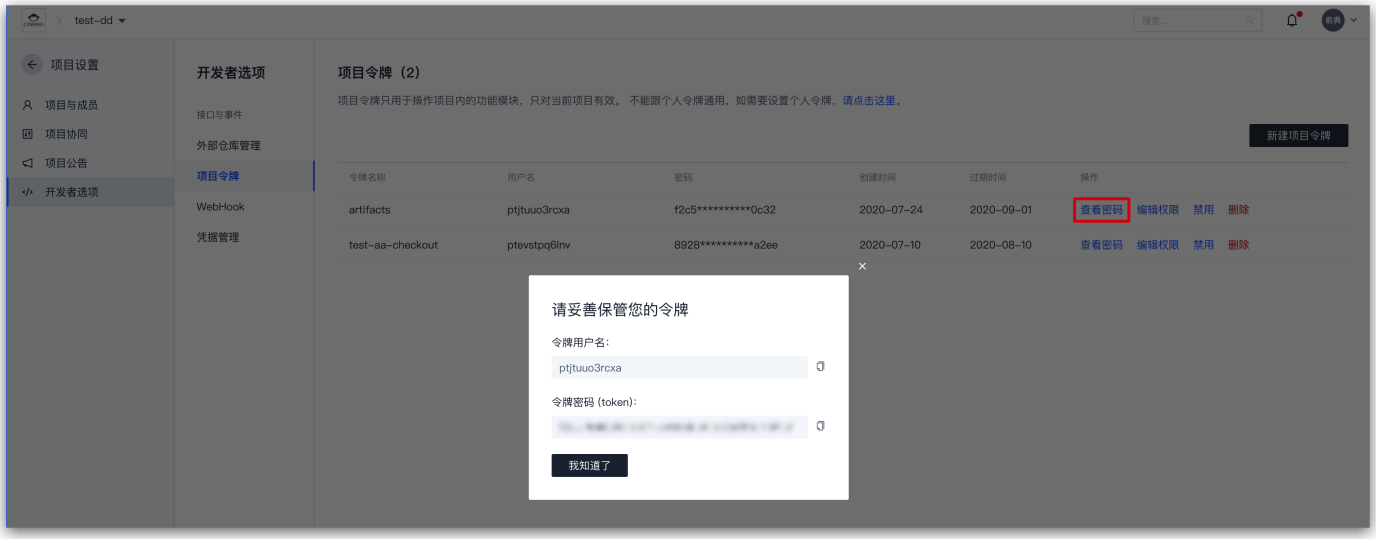
2. 配置制品库权限。

3. 单击**确定**后创建成功。

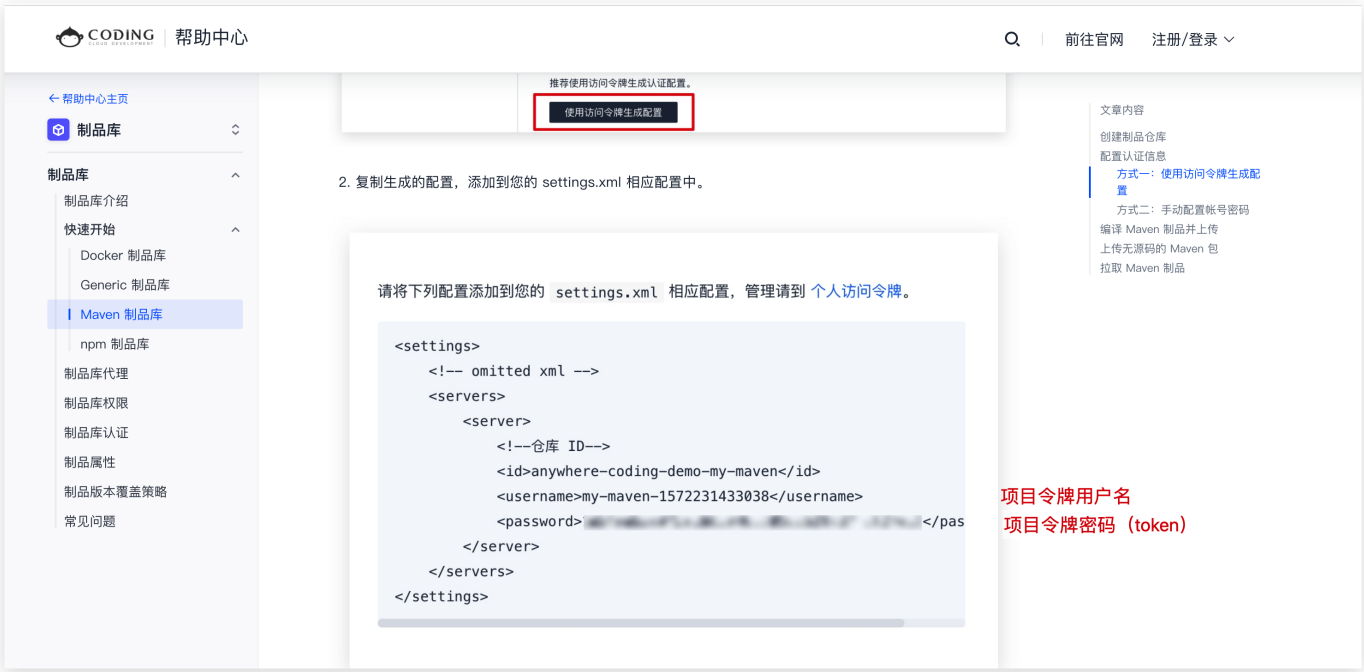
步骤二：在项目 B 中将刚才创建的项目 A 项目令牌作为用户名 + 密码拉取制品

1. 根据您的制品类型，配置认证信息，详情请参见 制品库认证。

2. 回到刚才创建的项目 A 项目令牌页面，单击**查看密码**。



3. 在项目 B 制品库配置认证信息时，将项目令牌用户名 + 项目令牌密码（token）作为用户名 + 密码填入信息。以 Maven 类型制品库为例：



4. 正确填入信息后，即可拉取成功其他 CODING 项目的制品库。

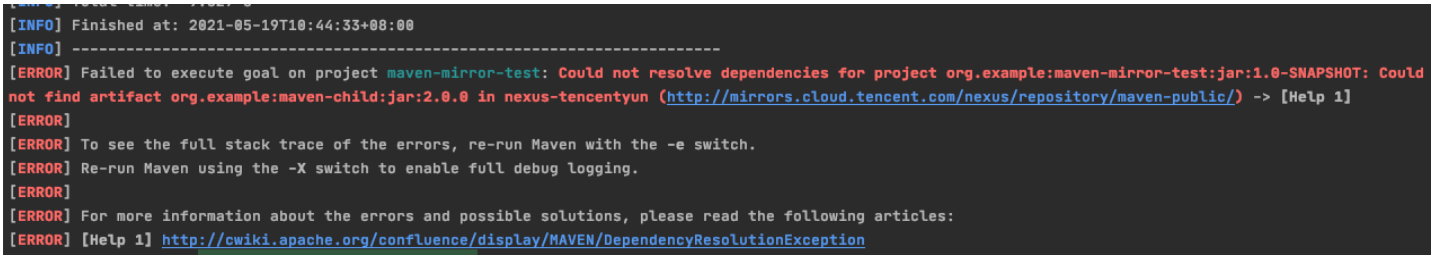
为什么制品仓库会出现没有主动推送的依赖包?

这是由于该制品仓库开启了 制品代理功能，当制品库代理功能开启时，制品仓库会作为统一入口帮助您管理依赖的第三方制品，因此这些依赖包会出现在制品仓库。您可以在 CODING 制品库内追踪该依赖包的团队内成员的使用情况，也可以通过 CODING 制品扫描统一检测该依赖包的安全漏洞，方便您的团队依赖制品审计。

为什么无法从制品库拉取依赖包?

问题描述：

使用 mirrors 参数配置镜像源加速后，无法从制品库拉取依赖包进行构建，问题截图如下：



处理办法：

导致此问题的原因可能是由于 <mirror> 配置中的 <mirrorOf>*</mirrorof> 将所有流量切换到了镜像源中拉取，而镜像源中并没有保存在 CODING 制品仓库中的依赖包。此问题有两个解决方案。

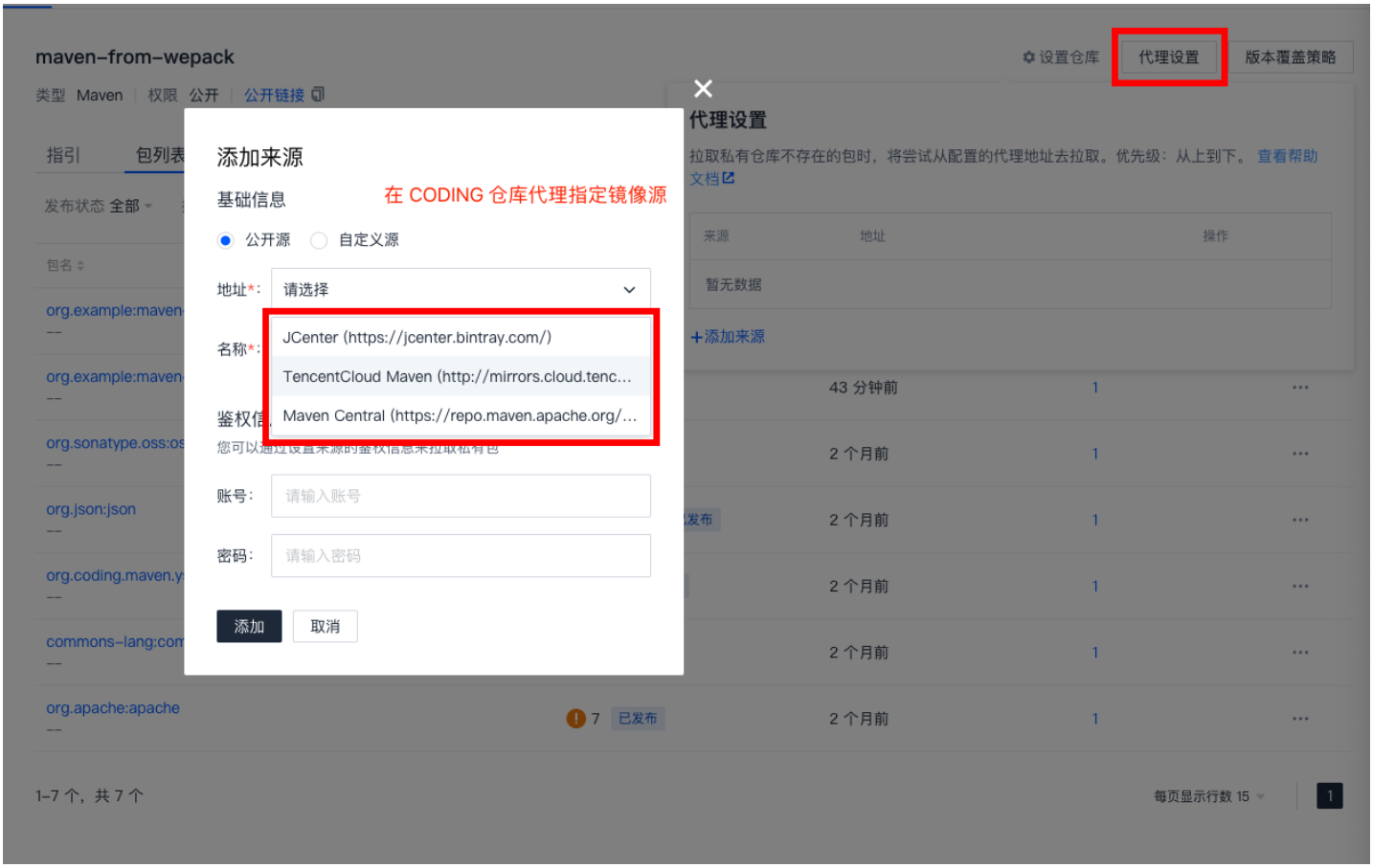
方法一：修改参数配置，仅允许非 CODING 制品仓库来源的制品从镜像源拉取。

```
<settings>
<!-- profiles 配置根据 CODING 仓库中的拉取指引配置 -->
<profiles>
<profile>
<id>Repository Proxy</id>
<activation>
<activeByDefault>true</activeByDefault>
</activation>
<repositories>
<repository>
<id>coding-maven-repo-id</id>
<name>coding-maven-repo-name</name>
<url>https://coding-maven-repo-url</url>
<releases>
<enabled>true</enabled>
</releases>
<snapshots>
<enabled>true</enabled>
</snapshots>
</repository>
</repositories>
</profile>
</profiles>
<mirrors>
<mirror>
<id>nexus-tencentyun</id>
```

```
<!-- 非 coding-maven-repo-id 来源的制品才会从镜像源中拉取制品 -->
<mirrorOf>!coding-maven-repo-id</mirrorOf>
<name>Nexus tencentyun</name>
<url>http://mirrors.cloud.tencent.com/nexus/repository/maven-public/</url>
</mirror>
</mirrors>
</settings>
```

方法二：删除 <mirrors> 镜像源配置，打开 CODING 制品仓库代理并指向镜像源。此方案会将所有开源依赖包代理并保存至 CODING 制品仓库。

```
<settings>
<!-- profiles 配置根据 CODING 仓库中的拉取指引配置 -->
<profiles>
<profile>
<id>Repository Proxy</id>
<activation>
<activeByDefault>true</activeByDefault>
</activation>
<repositories>
<repository>
<id>coding-maven-repo-id</id>
<name>coding-maven-repo-name</name>
<url>https://coding-maven-repo-url</url>
<releases>
<enabled>true</enabled>
</releases>
<snapshots>
<enabled>true</enabled>
</snapshots>
</repository>
</repositories>
</profile>
</profiles>
</settings>
```



注意：

持续集成中的镜像源加速配置写在了构建环境全局配置 ${M2_HOME}/conf/settings.xml 中，需要使用项目下 settings.xml 配置覆盖。

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

打包时指定了环境变量未生效

打包 Maven 制品时指定了环境变量，例如指定了 -Ptest，但上传至制品仓库时依然没有生效，依然为默认的 Dev 环境。若要解决此问题，请确认在使用 mvn deploy 命令推送到 CODING maven 制品库时是否有加 -P

参数指定环境。

dev配置文件： 编译： mvn clean package -DskipTests -Pdev -s settings.xml deploy到CODING maven制品库 mvn deploy -DskipTests -Pdev -s settings.xml

test配置文件： 编译： mvn clean package -DskipTests -Ptest -s settings.xml deploy到CODING maven制品库 mvn deploy -DskipTests -Ptest -s settings.xml

prod配置文件： 编译： mvn clean package -DskipTests -Pprod -s settings.xml deploy到CODING maven制品库 mvn deploy -DskipTests -Pdev -s settings.xml

上传制品时提示“请求参数中存在非法字符”

此问题常见于 Generic 类型制品上传。上传 Generic 制品包的名称仅支持 1-125 位的英文、数字、下划线(_)、中划线(-)、点(.)相互组合，请检查上传包的名称是否包含其他字符。

运行 docker build 生成镜像命令后报错如何处理?

问题详情:

docker build 生成镜像时提示 "for "-t, --tag" flag: invalid reference format" 错误码。

解决办法:

因默认版本变量 DOCKER_IMAGE_VERSION 的格式为 ${GIT_LOCAL_BRANCH:-branch}-${GIT_COMMIT}，此时若代码源中的分支带有 / 字符，类似含有命名为 release/1.0 的分支；因 docker tag 不支持带有 / 字符导致 docker build 命令运行失败。

在持续集成中的 enviroment 中添加分支变量:

```
DOCKER_IMAGE_VERSION = "${GIT_LOCAL_BRANCH.replace('/', '-')}-${GIT_COMMIT}"
```

具体操作如下图所示：


```
    stage('单元测试') {
      post {
        always {
          junit 'build/test-results/**/*.xml'

        }

      }
      steps {
        sh './gradlew test'
      }
    }
    stage('编译') {
      steps {
        sh './gradlew build'
      }
    }
    stage('构建镜像并推送到 CODING Docker 制品库') {
      steps {
        sh "docker build -t ${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION} -f ${DOCKERFILE_PATH} ${DOCKER_BUILD_CONTEXT}"
        useCustomStepPlugin(key: 'SYSTEM:artifact_docker_push', version: 'latest', params: [image:"${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_VERSION}",repo:"${DOCKER_REPO_NAME}"])
      }
    }
  }
  environment {
    CODING_DOCKER_REG_HOST = "${CCI_CURRENT_TEAM}-docker.pkg.${CCI_CURRENT_DOMAIN}"
    CODING_DOCKER_IMAGE_NAME = "${PROJECT_NAME.toLowerCase()}/${DOCKER_REPO_NAME}/${DOCKER_IMAGE_NAME}"
    DOCKER_IMAGE_VERSION = "${GIT_LOCAL_BRANCH.replace('/', '-')}-${GIT_COMMIT}"
  }
}
```