软件开发生产线 CodeArts

编译构建用户指南

天翼云科技有限公司

目 录

1 使用前必读

编译构建是指将软件的源代码编译成目标文件，并和配置文件、资源文件等一起打包的过程。

编译构建服务（CodeArts Build）为开发者提供配置简单的混合语言构建平台，实现编译构建云端化，支撑企业实现持续交付，缩短交付周期，提升交付效率。支持编译构建任务一键创建、配置和执行，实现获取代码、构建、打包等活动自动化，实时监控构建状态，让您更加快速、高效地进行云端编译构建。

更多产品信息请参考产品介绍。

在使用编译构建服务前，用户需先了解角色权限和使用流程。

2 角色权限

编译构建中默认的用户角色类型及对构建任务的操作权限说明如下：

表 2-1 编译构建默认角色权限矩阵

项目角色	编辑	删除	查看	执行	复制	禁用	权限管理
任务创建者	√（*）	√（*）	√（*）	√（*）	√（*）	√（*）	√（*）
项目创建者	√（*）	√（*）	√（*）	√（*）	√（*）	√（*）	√（*）
项目经理	√	√	√	√	√	√	√
开发人员	√	√	√	√	√	√	×
测试经理	×	×	√	×	×（*）	×	×
测试人员	×	×	×	×	×（*）	×	×
参与者	×	×	×	×	×（*）	×	×
浏览者	×	×	√	×	×（*）	×	×

说明

- “√”表示默认有权限，“×”表示默认没有权限。
- 拥有“权限管理”权限的角色可以修改权限矩阵，但带“*”的权限不可修改。
- 项目创建者、项目经理和开发人员可以创建编译构建任务。

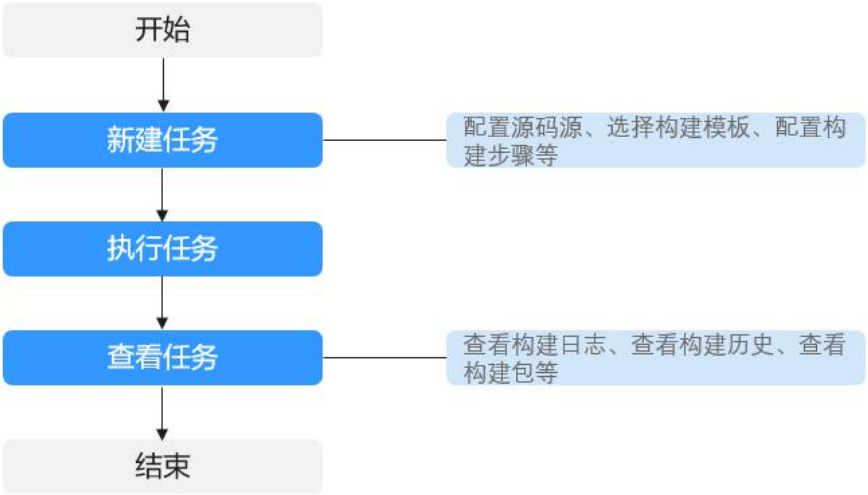
如果当前角色权限不满足用户需求，可参考配置角色权限进行设置。

3 使用流程

编译构建服务（CodeArts Build）为开发者提供配置简单的混合语言构建平台，实现编译构建云端化，支撑企业实现持续交付，缩短交付周期，提升交付效率。支持编译构建任务一键创建、配置和执行，实现获取代码、构建、打包等活动自动化，实时监控构建状态，让您更加快速、高效地进行云端编译构建。

流程介绍

介绍编译构建基本操作流程。



流程说明如下：

操作	说明
新建任务	新建构建任务，根据需要配置如下信息： • 源码源配置：可选择 Repo、GitHub、通用 Git 或来自流水线。 • 选择构建模板：编译构建支持 Maven/Ant/Gradle/CMake 等主流构建标准，预置了对应的构建模板，可根据需要选择工具版本；如果默认的工具版本满足不了用户需求，编

操作	说明
	译构建支持用户自定义构建环境，通过自定义制作镜像或者使用公共镜像进行构建，实现用户特殊构建需求。 • 配置构建步骤：编译构建预置了丰富的构建步骤，用户可以根据需要自定义组合。
执行任务	任务配置完成后，执行任务。具体操作参考执行构建任务。
查看任务	任务执行完成后，可查看任务详情以及执行结果，详见查看构建任务。

4 新建任务相关操作

4.1 新建构建任务

前提条件

- 已有可用项目，如果没有，请参考需求管理 CodeArts Req 服务的“用户指南 > Scrum 项目 > 新建 Scrum 项目”。
- 已在项目中新建可用代码仓库，如果没有，请参考代码托管 CodeArts Repo 服务的“用户指南 > 新版（推荐）创建 > 代码托管仓库”。

基本信息配置

1. 登录编译构建服务首页。
2. 单击“新建任务”，进入配置“基本信息”页面，填写构建任务基本信息。

表 4-1 基本信息

参数项	描述
任务名称	任务的名称。
归属项目	任务所属项目。
源码源	• Repo：默认从代码托管拉取代码进行构建，请选择已经创建的源码仓库及分支。 • 通用 Git：对于托管在其他服务上的代码，可以使用通用 Git 连接实现代码拉取，详见使用通用 Git 构建。 • 来自流水线：如果选择来自流水线，则只能通过流水线驱动执行，不能单独执行。
任务描述	对任务进行描述。

构建模板配置

1. 单击“下一步”，进入“构建模板”页面。

2. 选择适合自己项目的构建模板，然后单击“下一步”。
   也可以选择“空白构建模板”。
3. 进入“构建步骤”页签，页面展示所选模板的默认步骤组合，单击构建步骤上的 可根据需要添加构建步骤。
   详细步骤配置请参考配置步骤相关操作。

说明

若构建步骤中预置的工具版本无法满足使用需求，可以使用 SWR 公共镜像进行自定义环境构建。

配置其他信息

根据需要可在导航栏中配置其他页签信息。

- 基本信息：修改任务名称、归属项目、源码源、任务描述。
- 构建步骤：修改构建步骤信息。
- 参数设置：配置执行任务时的自定义参数。
- 执行计划：配置定时执行计划、持续集成触发策略。

4.2 源码源配置

4.2.1 背景

编译构建服务默认从代码托管服务拉取代码进行构建，服务扩展点（Endpoint）是软件开发生产线的一种扩展插件，为软件开发生产线提供链接第三方服务的能力。

编译构建可使用服务扩展点连接通用 Git 获取项目代码，可以提供对此类连接的新建、编辑、删除等操作。

说明

- 使用第三方代码仓库可能出现网络不稳定或其他问题，具体使用体验取决于第三方代码仓库网络环境和服务状态。
- 建议使用代码托管的代码导入功能，将代码导入到代码托管，实现安全、稳定、高效下载与构建。

4.2.2 使用通用 Git 构建

- 编译构建默认从代码托管拉取代码构建，对于托管在其它服务上的代码，可以使用通用 Git 连接实现代码拉取。
- 通用 Git 连接使用 AccessToken 授权，仅用于构建过程中拉取代码。

操作步骤

步骤 1 新建构建任务时，在“源码源”处选择“通用 Git”。首次使用通用 Git 连接，需要新建 Endpoint 实例。

步骤 2 单击“Endpoint 实例”右侧的“新建”，进入“服务扩展点管理”页面。

步骤 3 单击“新建服务扩展点”，在下拉列表选择“通用 Git”。

步骤 4 弹出“新建服务扩展点”对话框，请填写相应参数。

表 4-2 参数说明

参数名称	功能描述
连接名称	服务扩展点的名称。
Git 仓库 Url	Git 仓库的 Url（https 协议地址）。
用户名	Git 仓库用户名。
密码或 Access Token	Git 仓库密码或 Access Token。

步骤 5 参数设置完成后，单击“确定”，完成后续任务配置即可。

----结束

4.3 自定义模板

当预置的构建模板无法满足构建需求时，可以选择自定义构建模板。

步骤 1 登录编译构建服务首页。

步骤 2 在列表中选择构建任务，单击任务名称进入“构建历史”页面。

说明

若列表中没有任务，请新建构建任务。

步骤 3 单击页面右上角⋮，在下拉列表中选择“保存模板”。

图 4-2 保存模板

步骤 4 在弹框中输入模板名称与模板描述，单击“保存”。

步骤 5 单击页面右上角用户名，在下拉菜单中选择“租户设置”。

步骤 6 单击导航“编译构建 > 自定义模板”，即可在列表中看到已保存的构建模板。

对已保存的构建模板，可以完成以下操作：

表 4-3 管理自定义模板

操作	说明
收藏模板	单击☆，可以收藏该模板。
删除模板	单击🗑，在弹框中单击“确定”，即可删除该模板。

----结束

4.4 自定义构建镜像

背景信息

当常用的编译构建环境无法满足构建需求时，通过自定义构建环境提供的基础镜像，添加项目需要的依赖和工具，制作 Dockerfile 文件，然后制作 Docker 镜像并推送到 SWR 镜像仓，再通过使用 SWR 公共镜像即可实现自定义环境构建。

基础镜像

编译构建使用 centos7 和 ubuntu18 作为基础镜像，并提供多种构建常用的配置环境工具，用户可以根据需要配置自定义构建环境。

内置环境工具如下：

jdk 1.8、maven、git、ant、zip、unzip、gcc、cmake、make。

操作步骤

步骤 1 登录编译构建服务首页。

步骤 2 在编译构建首页右上角单击“更多”，在下拉列表选择“自定义构建环境”。

步骤 3 进入自定义构建环境页面，选择合适的基础镜像，单击即可下载 Dockerfile 模板。

步骤 4 编辑下载的 Dockerfile 文件。

可根据需要加入项目需要的其他依赖和工具，完成 Dockerfile 文件自定义，如下为添加了 jdk 和 maven 工具的示例。

```
RUN yum install -y java-1.8.0-openjdk.x86_64
RUN yun install -y maven
RUN echo 'hello world!'
RUN yun clean all
```

----结束

4.5 配置步骤相关操作

4.5.1 图形化构建

4.5.1.1 导读

编译构建预置了丰富的构建步骤，用户可以根据需要自定义组合。

如：maven 构建，内置了 maven、jdk 等工具，根据构建场景选择工具版本。如果预置的工具版本满足不了用户需求，可以通过自定义 Docker 镜像，加入项目需要的依赖和工具，将所需环境打包制作成 Docker 镜像并推送至 SWR 镜像仓库。可以随用随取，快速构建部署，减少项目对运行环境的依赖，详情请参考制作镜像并推送到 SWR 仓库和使用 SWR 公共镜像。

4.5.1.2 构建环境配置

配置构建任务全局运行环境。

说明

分为 X86 服务器与鲲鹏（ARM）服务器，在不同芯片架构上运行的软件，需要选择对应的环境主机。如软件最终在鲲鹏服务器上运行，则选择鲲鹏服务器。

配置说明

预置“构建环境配置”步骤，如图：

参数说明如下：

参数项	说明
构建环境主机类型	X86 服务器、鲲鹏（ARM）服务器。

4.5.1.3 代码下载配置

配置代码下载方式，若源码源为 Repo，可选择使用指定代码仓库 Tag 或 CommitID 构建，同时可选择开启子模块（submodules）自动更新。

配置说明

预置“代码下载配置”步骤，如图：

参数说明

参数项	说明

参数项	说明
使用指定代码仓库 Tag 或 CommitID 构建	不指定、指定 Tag 构建、指定 CommitID 构建。
子模块（submodules）自动更新	开启或不开启。
开启 Git LFS	根据需要选择是否开启“Git LFS”，构建默认不拉取音视频、图像等大型文件，开启“Git LFS”后，构建将会全量拉取文件。

指定 Tag 构建

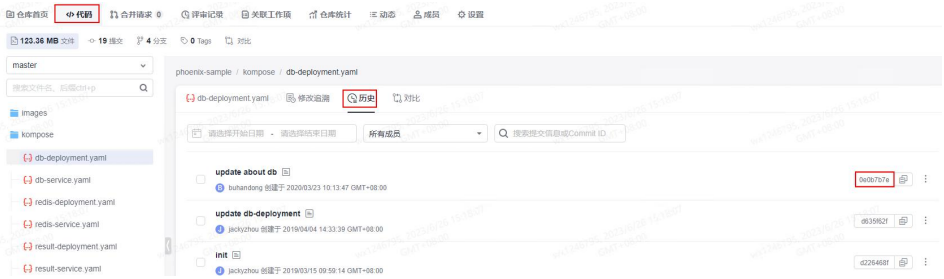
Tag 是指代码仓库中的标签，关于如何创建 Tag 请参见代码托管 CodeArts Repo 服务的“用户指南 > 新版（推荐） > 使用代码托管仓库 > 管理代码文件 > 标签管理”。

1. 在编译构建任务中，选择“指定 Tag 构建”，可以使用历史版本代码进行构建。

2. 执行任务时，会出现弹窗，输入标签名，单击“确定”，即可执行任务。

指定 CommitID 构建

CommitID 是指提交代码时生成的编号，在代码仓库中显示如下。



在编译构建任务中，可以通过指定 CommitID 来使用历史版本代码进行构建。

1. 选择“指定 CommitID 构建”，输入克隆深度，保存任务。

说明

克隆深度是指距离最近一次提交的提交次数，该值越大，检出代码的时间越长。深度为正整数，推荐最大深度为 25。

例如：克隆深度输入 5，那么在执行任务时，参数“CommitID”填写距离最近提交的前 5 个提交号中的任意一个都可以。

2. 执行任务时，会出现弹窗，按需要输入 CommitID，单击“确定”，即可启动任务执行。

子模块（submodules）自动更新

子模块属于 Git 的一个概念，是为了解决代码仓库包含并使用其他项目代码仓库的问题，详见代码托管 CodeArts Repo 服务的“用户指南 > 新版（推荐） > 配置代码托管仓库 > 仓库管理 > 子模块设置”。

在设置中，可以看到“子模块（submodules）自动更新”开关。打开开关，当代码仓库存在子模块时，系统在构建时会自动拉取子模块仓库的代码；反之关闭，系统不会自动拉取子模块仓库的代码。

4.5.1.4 Maven 构建

4.5.1.4.1 构建步骤：Maven 构建

背景信息

使用 Apache Maven 构建 Java 项目，主要包含以下特性：

- Maven 构建命令：执行 mvn package 命令、mvn deploy 命令或其他 shell 命令。
- 自定义公有依赖仓库：支持使用非软件开发生产线提供的公开依赖仓库构建。
- 自定义私有依赖仓库：支持添加其他私有依赖仓库。
- 发布私有依赖包到私有依赖库：自动在“pom.xml”文件增加 deploy 配置，配置后可使用 mvn deploy 发布私有依赖包到私有依赖库。
- 配置单元测试报告：支持构建后查看 junit 单元测试报告。

配置说明

添加“Maven 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。 **说明** 如果预置的工具版本满足不了用户需求，可以通过自定义 Docker 镜像，加入项目需要的依赖和工具，将所需环境打包制作成 Docker 镜像并推送至 SWR 镜像仓库，详情请参考制作镜像并推送到 SWR 仓库和使用 SWR 公共镜像。
命令	配置 Maven 命令，一般使用系统默认生成的命令即可。
setting 配置	可开启自动生成 setting 文件并配置依赖仓库，增加非 CodeArts 提供的依赖仓库：即如果需要的依赖无法在 CodeArts 开源镜像站、CodeArts 私有依赖仓库、SDK 仓库找到，则需要在此处添加。详见配置依赖仓库。
发布依赖包到 CodeArts 私有依赖库	选择“配置所有 pom”，则会自动在 pom.xml 文件增加 deploy 配置，可使用 mvn deploy 发布依赖包到 CodeArts 私有依赖仓库。详见配置发布依赖包到私有依赖库。
单元测试	选择处理单元测试结果并生成可视化报告，并使用报告结果控制任务的执行。详见配置单元测试。
缓存配置	选择是否使用缓存以提高构建速度，选择“使用缓存”后，每次构建时会把下载依赖包缓存起来，后续构建无需重复拉取，可有效提高构建速度。 **说明** maven 构建的依赖包存入缓存之后，只有当租户下面构建的项目有引进

参数项	说明
	新的依赖包时，才会更新缓存目录，并不支持对已有的依赖包缓存文件进行更新。

4.5.1.4.2 配置依赖仓库

编译构建支持使用非软件开发生产线提供的依赖仓库构建，为区分不同仓库，Maven构建将仓库按其来源、网络、权限等特征分为公有依赖仓库和私有依赖仓库。

- 公有依赖仓库
  - 开源镜像站：编译构建服务默认配置，无需任何修改即可在构建任务中使用。
  - 自定义公有依赖仓库：非软件开发生产线提供的公有依赖仓库（公开访问的），需在构建步骤“Maven 构建”中配置公有依赖仓库才能使用。
- 私有依赖仓库
  - 私有依赖库：用户开通后，编译构建服务默认配置，无需任何修改即可在构建任务中使用。
  - 自定义私有依赖仓库：非软件开发生产线提供的私有依赖仓库（企业私有，访问需要授权帐号认证），需在构建步骤“Maven 构建”中配置私有依赖仓库才能使用。

配置自定义公有依赖仓库

配置使用指定的公有依赖仓库构建。

1. 新建构建任务，其中，构建模板选择“Maven”。
2. 配置“Maven 构建”构建步骤，展开“setting 配置”。
3. 添加公有依赖仓库，输入仓库地址，根据需要勾选“release 仓库”和“snapshot 仓库”。
   - release 仓库：勾选后，构建过程将尝试从仓库中下载 release 版本依赖。
   - snapshot 仓库：勾选后，构建过程将尝试从仓库中下载 snapshot 版本依赖。

说明

release 仓库和 snapshot 仓库至少勾选一个，也可以同时勾选。

配置自定义私有依赖仓库

配置使用指定的私有依赖仓库构建。

1. 参考流水线 CodeArts Pipeline 服务的“用户指南 > 新版 UI（推荐） > 服务扩展点”，新建 nexus repository 服务扩展点。
2. 新建构建任务，其中，构建模板选择“Maven”。
3. 配置“Maven 构建”构建步骤，展开“setting 配置”。

   添加私有依赖仓库，选择第一步创建好的服务扩展点，根据需要勾选“release 仓库”和“snapshot 仓库”。

4.5.1.4.3 配置发布依赖包到私有依赖库

背景信息

编译构建服务默认配置私有依赖库作为私有依赖下载源，用户可通过手工上传或 Maven 构建方式将依赖上传到私有依赖仓库，如果需要将构建产物上传至私有依赖库，则需要添加此配置。

说明

- **软件发布库**主要用于归档用以部署或其他用途的软件包。
- **私有依赖库**主要用于存储供其他项目依赖的工具包等。

使用时要务必注意区分，避免出现如“将依赖上传到软件发布库但是构建时无法下载”此类场景。

操作步骤

步骤 1 参考制品仓库 CodeArts Artifact 服务的“用户指南 > 私有依赖库（新版） > 新建私有依赖库”。（如果已经创建，请忽略该步骤）。

步骤 2 使用 Maven 模板参考代码托管 CodeArts Repo 服务的“用户指南 > 新版（推荐） > 创建代码托管仓库 > 按模板新建仓库”新建代码仓库。

步骤 3 单击代码仓库名称，进入代码托管“文件”页，在“pom.xml”文件配置私有依赖坐标信息（groupId、artifactId、version）。

修改准备构建的私有依赖项目，“pom 文件”中指定坐标为：

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchem
```

```
a-instance"xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>come.test.demo</groupId>
    <artifactId>javaMavenDemo</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <name>maven_demo</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

步骤 4 新建构建任务，其中，构建模板选择“Maven”。

步骤 5 配置“Maven 构建”构建步骤，展开“发布依赖包到 CodeArts 私有依赖库”，选择“配置所有 pom”。

- 不配置 pom：表示无需发布私有依赖包到 CodeArts 私有依赖库。
- 配置所有 pom：表示在项目下所有“pom.xml”文件增加 deploy 配置，使用 **mvn deploy** 命令将构建出的依赖包上传到私有依赖仓库。

步骤 6 在命令窗口，使用“#”注释命令 **mvn package -Dmaven.test.skip=true -U -e -X -B**。

步骤 7 删除**#mvn deploy -Dmaven.test.skip=true -U -e -X -B** 命令前的“#”。

步骤 8 配置完成后执行构建任务。执行成功后即可将依赖包发布到私有依赖库。

步骤 9 进入软件开发生产线首页，单击导航栏“服务 > 制品仓库”，进入私有依赖库，搜索并查看上传的依赖。

上传成功后，在其他项目添加如下坐标即可引用。

```
<dependency>
    <groupId>com.test.demo</groupId>
    <artifactId>javaMavenDemo</artifactId>
    <version>1.0</version>
</dependency>
```

说明

私有依赖仓库分为 release 仓库和 snapshot 仓库，两种仓库对应的使用场景为：

- 对于以调试为目的发布的私有依赖包，一般会给依赖版本号增加-SNAPSHOT 后缀（如：1.0.0-SNAPSHOT），执行发布操作时，此类依赖会自动发布到 snapshot 仓库，发布时无需更新版本号，构建命令中增加-U 参数即可拉取最新版本。
- 对于正式发布的私有依赖包，版本号中不可带-SNAPSHOT 后缀（如：1.0.0），执行发布操作时，此类依赖会自动发布到 release 仓库，发布时必须更新版本号，否则会导致构建过程无法拉取最新依赖包。

----结束

4.5.1.4.4 配置单元测试

前提条件

使用“Maven 构建”构建步骤提供的单元测试功能，需要在项目中编写单元测试代码，且满足如下条件：

- 单元测试用例代码存放位置需满足 Maven 默认单元测试用例目录规范及命名规范，或自行在配置中指定用例位置。

  如：单元测试用例统一存放在路径“src/test/java/{{package}}/”时，单元测试将在 Maven 构建过程自动执行。
- 项目中不可存在忽略单元测试用例配置，即如下配置不可存在于项目“pom.xml”文件：

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-surefire-plugin</artifactId>
   <version>2.18.1</version>
   <configuration>
      <skipTests>true</skipTests>
   </configuration>
</plugin>
```

- 在“pom.xml”文件引入 junit 依赖。

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.7</version>
 </dependency>
```

操作步骤

步骤 1 新建代码仓库并参考代码托管服务的“用户指南 > 迁移到代码托管仓库 > 将本地代码上传到代码托管”，上传代码至代码仓库。

步骤 2 在 src 目录下创建单元测试类，如下图所示：

Demo 项目代码如下：

```
package test;

public class Demo {
   public String test(Integer i) {
       switch (i) {
           case 1:
               return "1";
           case 2:
               return "2";
           default:
               return "0";
       }
}
}
```

单元测试代码如下，其中，@Test 注解表示测试方法。

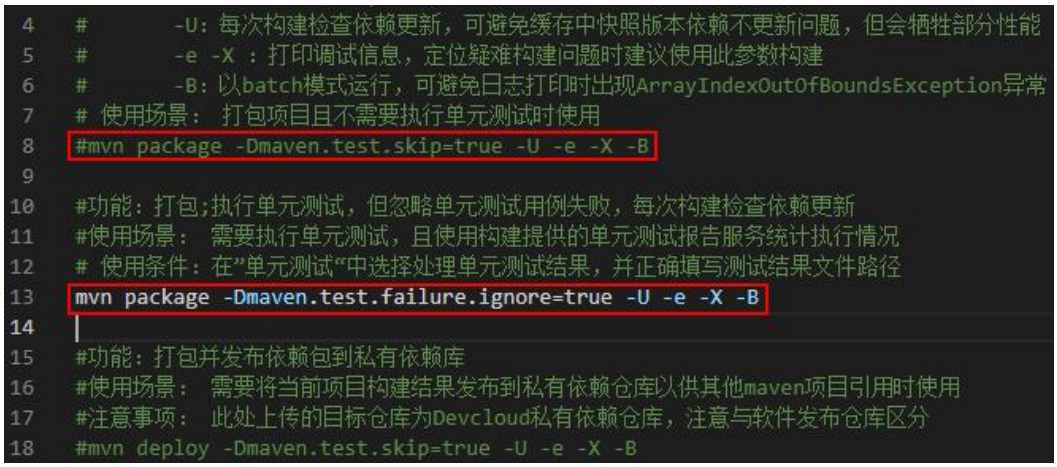
```
package test;

import org.junit.Test;

public class DemoTest {
    private Demo demo=new Demo();
    @Test
    public void  test(){
        assert demo.test(1).equals("1");
        assert demo.test(2).equals("2");
        assert demo.test(3).equals("0");
    }
}
```

步骤 3 新建构建任务，其中，构建模板选择“Maven”。

步骤 4 配置“Maven 构建”构建步骤，在命令窗口，使用“#”注释掉第 8 行的默认命令，并删除第 13 行命令前的“#”。



步骤 5 展开“单元测试”。

- 在“是否处理单元测试结果”处勾选“是”。
- 根据需要选择“是否忽略用例失败”。

  - 若勾选“是”，则用例失败时构建任务仍然成功。
  - 若勾选“否”，则用例失败时构建任务也失败。
- 配置单元测试结果文件路径。

  测试报告需要采集单元测试结果用以生成可视化报告，需在此处指明单元测试结果文件路径：
  - 多数情况下，保留默认路径“**/TEST*.xml”即可满足任务需求。
  - 为增加结果准确性，可根据实际情况制定精确的报告路径，如：“target/surefire-reports/TEST*.xml”。
- 根据需要选择“是否处理单元测试覆盖率结果”，配置方法请参见使用 JaCoCo 生成单元测试覆盖率报告。
- 配置单元测试覆盖率报告路径。

  请填写相对于项目根目录的相对路径，如：target/site/jacoco，选择处理单元测试覆盖率结果后，会将此目录下的所有文件进行打包上传。

步骤 6 配置完成后，执行编译构建任务。

执行成功后，即可在任务执行详情页面的“测试”页签查看测试报告，如果选择了处理单元测试覆盖率报告，会生成覆盖率测试报告，单击“覆盖率报告下载”即可下载。

----结束

使用 JaCoCo 生成单元测试覆盖率报告

- 单模块项目配置方法

  在项目中已添加 jacoco-maven-plugin 插件用于生成单元覆盖率报告，即在“pom.xml”文件中添加如下配置：

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.5</version>
    <executions>
       <execution>
          <goals>
             <goal>prepare-agent</goal>
          </goals>
       </execution>
       <execution>
          <id>report</id>
          <phase>test</phase>
          <goals>
             <goal>report</goal>
          </goals>
       </execution>
    </executions>
</plugin>
```

  JaCoCo 的 report 目标默认是在 verify 阶段，这里需要将 report 目标定义为 test 阶段，执行 *mvn test* 时，才会在代码的“target/site/jacoco”目录下生成单元测试的覆盖率报告。

- 多模块项目配置方法

  假设多模块项目代码结构如下，说明如何配置生成单元测试覆盖率报告。

```
├── module1
│   └── pom.xml
├── module2
│   └── pom.xml
├── module3
│   └── pom.xml
├── pom.xml
```

  a. 在项目下添加一个用来聚合的模块，自定义名称如：report，添加聚合模块后的代码结构如下：

```
├── module1
│   └── pom.xml
├── module2
│   └── pom.xml
├── module3
│   └── pom.xml
├── report
│   └── pom.xml
├── pom.xml
```

  b. 在项目根目录的“pom.xml”文件添加 jacoco-maven-plugin 插件。

```
<!-- 配置单元测试覆盖率-->
 <plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.3</version>
    <executions>
       <execution>
          <goals>
             <goal>prepare-agent</goal>
          </goals>
       </execution>
    </executions>
 </plugin>
```

  c. 配置聚合模块的“pom.xml”文件。

  以 dependency 形式引入所有依赖模块，并使用 report-aggregate 定义 JaCoCo 聚合目标。

```
<dependencies>
    <dependency>
       <groupId>${project.groupId}</groupId>
       <artifactId>module1</artifactId>
       <version>${project.version}</version>
    </dependency>
    <dependency>
       <groupId>${project.groupId}</groupId>
       <artifactId>module2</artifactId>
       <version>${project.version}</version>
    </dependency>
    <dependency>
       <groupId>${project.groupId}</groupId>
       <artifactId>module3</artifactId>
```

```
        <version>${project.version}</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.3</version>
            <executions>
                <execution>
                    <id>report-aggregate</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report-aggregate</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

配置完成后，在项目根目录下执行 *mvn test*，执行成功后会在“report/target/site/jacoco-aggregate”目录下生成各个模块的覆盖率报告。也可以在 outputDirectory 中自定义报告的输出路径：

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.3</version>
    <executions>
        <execution>
            <id>report-aggregate</id>
            <phase>test</phase>
            <goals>
                <goal>report-aggregate</goal>
            </goals>
            <configuration>
                <outputDirectory>target/site/jacoco</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

4.5.1.5 Android 构建

Android 构建系统编译应用资源和源代码，然后将它们打包成可供部署、签署和分发的 APK。

自定义安装

sdkmanager 命令（sdkmanager packages [options]）：安装需要的 Android 构建环境，如：sdkmanager "platform-tools" "platforms;android-28" --sdk_root=./，表示使用 sdkmanager 下载 platform-tools 和 platforms;android-28 到当前代码根目录下。

配置说明

添加“Android 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
Gradle	根据需要选择 Gradle 版本。
JDK	根据需要选择 JDK 版本。
NDK	根据需要选择 NDK 版本，也可以选择“不使用”。
命令	配置 Gradle 命令，一般使用系统默认给出的命令即可。

Android 版本说明

- SDK：用户项目构建 compileSdkVersion 版本。
- Build Tools：用户项目构建所需 buildToolsVersion 版本。

两个版本可以在项目下的“build.gradle”文件或是项目的全局配置文件（用户自定义）中找到。

app/build.gradle 大小: 951 bytes

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion '25.0.0'
    defaultConfig {
        applicationId "cn.bluemobi.dylan.step"
        minSdkVersion 11
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    lintOptions {
        abortOnError false
    }
}

```

说明

- 用户需要选择正确的 compileSdkVersion 版本和 buildToolsVersion 版本。
- 也支持 Gradle 的 wrapper 构建方式，如果提供的 gradle 版本没有满足您的要求，您也可以直接使用 gradlew 命令，使用 wrapper 去构建，会自动下载您所需要的 gradle 版本，构建命令例如：**./gradlew clean build**。

4.5.1.6 Android APK 签名

通过“Android APK 签名”构建步骤，使用 apksigner 对 Android APK 进行签名。

配置说明

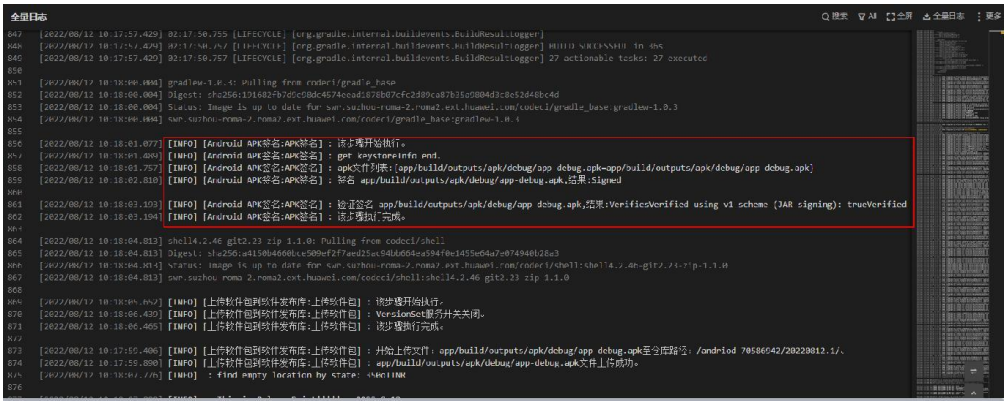
1. 在“Android 构建”步骤后添加“Android APK 签名”步骤。

   参数说明如下：

参数	说明
步骤显示名称	构建步骤的名称，可自定义修改。
需要签名的 APK 路径	Android 构建后生成要签名的.apk 文件位置，支持正则表达式，如：可以使用 build/bin/*.apk 匹配构建出来的 APK 包。
Keystore 文件	用于签名的 Keystore 文件，参考生成 Keystore 签名文件制作，单击下拉列表，展示文件管理页面已经上传的 Keystore 文件，请根据需要选择。
keystore password	密钥文件密码。
别名（Alias）	密钥别名。
key password	密钥密码。
apksigner 命令行	用户自定义签名参数，默认“--verbose”显示签名详细。

2. 验证签名是否成功。

   配置完成后执行构建任务，当显示任务执行成功后，查看构建日志，若“Android APK 签名”对应日志中显示“结果: Signed”即为签名成功。



4.5.1.7 Npm 构建

使用 Npm 工具管理软件包，能完成 vue 和 webpack 的构建。

配置说明

添加“Npm 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	配置 Npm 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.8 Gradle 构建

使用 Gradle 构建工具构建 Java，Groovy 和 Scala 项目。

配置说明

添加“Gradle 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
Gradle	根据需要选择 Gradle 版本。
JDK	根据需要选择 JDK 版本。
命令	配置 Gradle 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.9 Yarn 构建

使用 Yarn 构建 JavaScript 工程。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	配置 Yarn 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.10 gulp 构建

使用 gulp 工具构建前端集成开发环境。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	配置 gulp 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.11 Grunt 构建

使用 Grunt 构建 JavaScript 工程。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	配置 Grunt 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.12 mono

基于 mono-linux（X86 和 arm）平台完成 msbuild 和 dotnet 构建。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	配置 mono 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.13 PHP 构建

安装了 php 运行环境和 composer 工具，可以为声明项目所依赖的 php 代码库提供安装和打包环境。

参数说明如下：

参数项	说明

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	配置 PHP 命令，一般使用系统默认生成的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。 **说明** 第 2 行默认命令中“https://repo.packagist.org/”为官网仓库地址，如果用户访问不了该地址会导致构建失败，需替换成用户可以访问的仓库地址。

4.5.1.14 SetupTool 构建

使用 SetupTool 工具构建 Python 项目。

前提准备

使用 SetupTool 打包时，需要保证代码根目录下存在“setup.py”文件，关于 setup 文件写法请参见 Python 官方说明。

配置说明

添加“SetupTool 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	请根据需要选择工具版本。
命令	配置构建打包命令。 • 可以使用默认的命令打包为“egg”格式的文件。 • Python2.7 后建议使用 **python setup.py sdist bdist_wheel**，打包为源码包和 whl 格式的安装包，以便使用 pip 安装。

4.5.1.15 PyInstaller 构建

使用 PyInstaller 工具构建 Python 项目。

配置说明

添加“PyInstaller 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	请根据需要选择工具版本。
命令	配置构建打包命令，默认命令是将项目打包成一个可执行文件，PyInstaller 具体的命令可以查看官网文档。

4.5.1.16 执行 shell 命令

配置说明

添加“执行 shell 命令”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	请根据需要选择工具版本。
命令	请根据需要填写命令。

4.5.1.17 Gnu-arm 构建

使用 GNU-ARM 工具链设计、开发和使用 ARM 模拟器。

配置说明

添加“Gnu-arm 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	请根据需要选择 ARM 工具版本。
命令	配置 Gnu-arm 构建命令，一般使用系统默认给出的 make 命令即可。 • 如果 Makefile 不在代码根目录下，用户需要 cd 到正确的目录，再使用 make 命令。 • 用户不使用 make 命令，可以参考下列镜像自带的编译命令： – gnuarm201405 镜像 使用 **arm-none-linux-gnueabi-gcc** 命令，例如 arm-none-

参数项	说明
	linux-gnueabi-gcc -o main main.c – gnuarm-linux-gcc-4.4.3 镜像 使用 **arm-linux-gcc** 命令，例如 arm-linux-gcc -o main main.c – gnuarm-7-2018-q2-update 镜像 使用 **arm-none-eabi-gcc** 命令，例如 arm-none-eabi-gcc --specs=nosys.specs -o main main.c 说明 • Linux 下的 GNU 的 makefile 编写，请参见官网。 • 注意 Makefile 只有行注释“#”，如果要使用或者输出“#”字符，需要进行转义，"\#等。

4.5.1.18 CMake 构建

使用 CMake 构建工具构建跨平台项目工程。

配置说明

添加“CMake 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	请根据需要选择 CMake 构建工具版本。
命令	配置 CMake 命令，一般使用系统默认给出的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.19 Ant 构建

Apache Ant 是一个 Java 项目的构建工具，用于编译、测试和部署 Java 项目。

前提准备

项目为 Java 语言 Ant 结构，有正确的“build.xml”构建描述文件。

操作步骤

添加“Ant 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	默认使用推荐版本，可以根据需要选择和自己编译环境匹配的 Ant 与 JDK 镜像版本。
命令	配置 Ant 构建命令，一般使用系统默认给出的命令即可。如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.20 Go 语言构建

使用 Go 语言环境构建。

前置条件

项目为 Go 语言开发项目，代码中有构建描述文件。

配置说明

添加“Go 语言构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	请根据需要选择工具版本，默认使用推荐版本，可以根据需要选择和自己编译环境匹配的 Go 版本。
命令	配置 Go 项目构建命令，一般使用系统默认给出的命令即可，如有特殊构建要求，可以在文本域中输入自定义的构建脚本。

4.5.1.21 Android-Ionic 构建

背景信息

- Ionic 是一款基于 Angular、Cordova 的强大的 HTML5 移动应用框架，可以快速创建一个跨平台的移动应用。
- 支持快速开发移动 App、移动端 Web 页面、混合 App 和 Web 页面。

自定义安装 npm 组件

- 全局安装：**npm install －g xxx**。
- 项目下安装：**npm install xxx**。
- 如果发现镜像内安装的 npm 组件不符合要求，可以按照如下方式卸载，然后安装自己所需组件，以 cordova 为例：

a. 查看 cordova 版本：**cordova --version**，可知版本为 9.0.0。

```
[root@61e61df3003d /]# cordova --version
9.0.0 (cordova-lib@9.0.1)
```

b. 卸载 cordova：**npm uninstall －g cordova**。

```
[root@61e61df3003d /]# npm uninstall -g cordova
removed 438 packages in 5.106s
```

c. 重新安装需要的 cordova 版本：**npm install －g cordova@8.0.0**。

```
[root@localhost dist]# npm install -g cordova@8.0.0
npm WARN deprecated hawk@3.1.3: This module moved to @hapi/hawk. Please make sure to swit
npm WARN deprecated cryptiles@2.0.5: This version has been deprecated in accordance with
ou are unable to upgrade at this time, paid support is available for older versions (hapi
npm WARN deprecated sntp@1.0.9: This module moved to @hapi/sntp. Please make sure to swit
npm WARN deprecated hoek@2.16.3: This version has been deprecated in accordance with the
re unable to upgrade at this time, paid support is available for older versions (hapi.im/
npm WARN deprecated boom@2.10.1: This version has been deprecated in accordance with the
re unable to upgrade at this time, paid support is available for older versions (hapi.im/
/usr/local/node-v10.16.0-linux-x64/bin/cordova -> /usr/local/node-v10.16.0-linux-x64/lib/
npm WARN acorn-dynamic-import@4.0.0 requires a peer of acorn@^6.0.0 but none is installed

+ cordova@8.0.0
added 423 packages from 395 contributors in 130.31s
[root@localhost dist]# cordova --version
? May Cordova anonymously report usage statistics to improve the tool over time? Yes

Thanks for opting into telemetry to help us improve cordova.
8.0.0
[root@localhost dist]#
```

配置说明

步骤 1 确认 Ionic 项目已经上传到 CodeArts Repo 代码仓库。

项目中包含“ionic.config.json”、“package.json”和“andular.json”等项目编译描述文件。

步骤 2 添加“Ionic Android App 构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择 Gradle、JDK 和 NDK 版本。
命令	配置命令框中的打包脚本。

----结束

4.5.1.22 Android 快应用构建

本节主要介绍快应用签名的构建步骤-**Android 快应用构建**。

自定义安装

npm config set xxx 命令：配置 Npm 相关设置。

配置说明

添加“Android 快应用构建”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择构建工具版本。

命令	配置命令，以下是一个使用 debug 签名打包的例子。
	快应用签名操作：
	1. 通过 **openssl** 命令等工具生成签名文件“private.pem”、“certificate.pem”，例如：
	`openssl req -newkey rsa:2048 -nodes -keyout private.pem -x509 -days 3650 -out certificate.pem`
	在工程的“sign”目录下创建“release”目录，将私钥文件“private.pem”和证书文件“certificate.pem”复制进去。
	2. 发布程序包前需要增加 release 签名，然后在工程的根目录下运行：
	`npm run release`
	生成的应用路径为“/dist/.release.rpk”。
	3. 如果需要临时使用 debug 签名，可以使用：
	`npm run release -- --debug`
	说明
	由于 debug 签名是公开的，安全性无法保证，一定不要使用 debug 签名签发正式上线的应用。

4.5.1.23 制作镜像并推送到 SWR 仓库

背景信息

编译构建默认提供大量构建步骤、模板等，如果已有模板不能满足您的要求，如缺少必要的依赖包、工具等，您可以根据需要自定义 Docker 镜像，通过手动编写 Dockerfile 文件，加入项目需要的依赖和工具，然后将 Dockerfile 文件及所需的工具文件等上传到代码仓库，编译构建将根据 Dockerfile 文件制作镜像并推送至指定的 SWR 仓库。

- Dockerfile 是由一系列命令和参数构成的脚本，从 FROM 命令开始，紧接着跟随各种方法、命令和参数。这些命令应用于基础镜像并最终创建一个新的镜像。它

们简化了从头到尾的流程并极大的简化了部署工作。了解更多请参见 Docker 官网。

- 编译构建提供四种基础镜像：基于 centos7 包含各种常用工具的 X86、ARM 基础镜像和基于 ubuntu18 包含各种常用工具的 X86、ARM 基础镜像，可基于该基础镜像制作 Dockerfile 文件。

本节为您具体介绍如何使用 Dockerfile 定制一个简单的容器镜像并推送到 SWR 仓库：以 Maven 构建为例，在编译构建中完成 Maven 构建，并将构建生成的软件包制作为 Docker 镜像并推送到 SWR 仓库。

前提准备

制作 Docker 镜像前，需要做如下准备：

- **准备组织**

  在制作镜像并推送到 SWR 仓库时，需要指定 SWR 组织名，请提前在容器镜像服务创建组织。组织的约束与限制参考容器镜像服务的约束与限制。

- **准备项目代码**

  在代码托管服务基于 Java Maven Demo 模板创建代码仓库，请参见代码托管服务的“用户指南 > 创建代码托管仓库 > 按模板新建仓库”。

- **准备 Dockerfile 文件**

  基于编译构建基础镜像制作 Dockerfile 文件，本例使用 centos7 作为基础镜像。

  a. 在编译构建首页，单击右上角  ，选择“自定义构建环境”。

  b. 进入“自定义构建环境”页面，单击“基于 centos7 包含各种常用工具的 X86 基础镜像”，即可获取该基础镜像对应的 Dockerfile 文件。

  c. 获取构建包路径。

     Maven 构建包名格式为：artifactId-version.packaging，构建包默认生成于“./target”目录。

     单击代码仓库名称，进入代码托管“文件”页，查看代码仓库 pom.xml 文件坐标定义，如下图所示，则最终构建包路径为：./target/javaMavenDemo-1.0.jar。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>com        .demo</groupId>
  <artifactId>javaMavenDemo</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>maven_demo</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    </dependency>
  </dependencies>
```

d. 使用第 3 步获取的构建包路径编写第 2 步下载的 Dockerfile 文件，在基础镜像中添加 Maven 构建包，添加如下命令：

```
COPY ./target/javaMavenDemo-1.0.jar /demo/app.jar
```

该命令表示将构建包复制到镜像的“demo”目录，并将构建包命名为 app.jar。

e. 编写完成后，将 Dockerfile 文件及制作镜像过程中需要的其他文件上传到代码仓库根目录。

操作步骤

步骤 1 新建构建任务，其中，源码源选择前提准备中创建好的源码仓库，构建模板选择“Maven 构建”。

步骤 2 进入“构建步骤”页签，在“Maven 构建”步骤后新增“制作镜像并推送到 SWR 仓库”构建步骤。

“Maven 构建”构建步骤参数保持默认即可，“制作镜像并推送到 SWR 仓库”构建步骤参数配置说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	选择工具版本，使用默认版本即可。
镜像仓库	编译构建服务已经默认给出了各区域对应的 SWR 仓库地址，用户无需更改。 **说明** 支持推送到用户自定义镜像仓库。
授权用户	当前用户。请确保当前用户对组织内所有镜像享有编辑或管理权限。

参数项	说明
组织	SWR 仓库组织名，请填写准备工作中创建好的组织名。
镜像名字	制作完成后的镜像名称，可自定义。
镜像标签	用来标记镜像的版本，可自定义。通过“镜像名:标签”可以唯一指定镜像。
工作目录	docker build 命令中的“上下文路径”参数，该路径是 CodeArts Repo 代码仓库根目录的相对路径。 上下文路径，指的是 docker 在构建镜像时，docker build 命令将该路径下的所有内容打包给容器引擎帮助构建镜像。
Dockerfile 路径	Dockerfile 文件所在路径，请填写相对于工作目录的路径，如：工作目录为根目录， 且 Dockerfile 文件在根目录下，则此处填写为“./Dockerfile”。
添加构建元数据到镜像	将本次构建信息添加到镜像中，镜像制作完成后可以通过 docker inspect 命令查看镜像元数据。

步骤 3 配置完构建步骤，单击“新建并执行”，开始执行构建任务。

步骤 4 执行成功后，进入容器镜像服务。

步骤 5 单击导航栏“我的镜像”，选择“制作镜像并推送到 SWR 仓库”构建步骤中填写的组织，即可查看刚构建并上传的镜像。

----结束

4.5.1.24 使用 SWR 公共镜像

前提条件

使用 SWR 公共镜像构建前，需满足以下条件：

- 用户已将自定义镜像推送至 SWR 镜像仓。
- 用户在 SWR 镜像仓已将自定义镜像设置为公开。

配置方法

步骤 1 由于编译构建无法拉取用户在 SWR 私有仓中的镜像，因此，需要先将镜像设置为“公开”。

1. 登录 SWR 镜像服务。
2. 在导航单击“我的镜像”，然后单击镜像名称进入镜像详情页面，然后单击右上角“编辑”。
3. 在编辑框中，将“类型”设置为“公开”。

4. 获取完整的镜像地址：单击复制镜像下载指令，其中，docker pull 后面部分为镜像地址。

步骤 2 新建构建任务，在“新建任务 > 选择构建模板”页面，选择“空白构建模板”。

步骤 3 单击“下一步”，进入“构建步骤”页签。

步骤 4 添加“使用 SWR 公共镜像”构建步骤。

步骤 5 将获得的镜像地址粘贴到“镜像地址”输入框。

说明

将下载指令粘贴到“镜像地址”输入框时请去掉前面的“docker pull”。

步骤 6 在命令窗口输入构建命令，然后执行构建任务，即可完成构建。

----结束

4.5.1.25 上传软件包到软件发布库

将构建生成的软件包上传到软件发布库，在配置构建步骤时，添加“上传软件包到软件发布库”构建步骤即可。

- 仅支持上传单个文件、多个文件；不支持上传文件夹、自动创建路径。

  例如，“a”目录下有“aa”文件和“b”目录，“b”目录下有“bb”文件，构建包路径配置为“a/**”。

  即递归扫描“a”目录下所有文件，两个文件是同一个目录下，“aa”、“bb”两个文件将会上传到同一个目录下，系统不会在软件发布库里自动创建“b”目录。

- 如果用户有上传文件夹的需要，建议在“上传软件包到软件发布库”构建步骤之前先将待上传的文件夹打包为单文件后再上传。可以通过现有构建步骤执行打包命令，也可以新增“执行 shell 命令”构建步骤执行打包命令。
- 上传的软件包相关限制请参考制品仓库服务的约束与限制。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
构建包路径	构建结果所在路径，支持正则表达式。如：Maven 构建可以使用“**/target/*.?ar”匹配所有构建出来的 jar 包和 war 包。
发布版本号	构建包在软件发布库的存储目录，推荐不填，不填写时，自动使用构建编号作为目录名。
包名	上传到软件发布库的文件名都会改成包名，推荐不填，不填会保留原文件名。

参数配置

- **构建包路径**

  构建包路径支持正则匹配，“**”递归遍历当前目录，“*”匹配 0 或者多个字符，“?”匹配一个字符。

  系统文件分隔符使用“/”；路径对大小写不敏感。

  举例说明：

  - *.class

    当前目录下匹配“.class”结尾的文件。
  - **/*.class

    当前目录下递归匹配所有的“.class”结尾的文件。
  - test/a??.java

    匹配“test”目录下以“a”开头后跟两个字符的 java 文件。
  - **/test/**/XYZ*

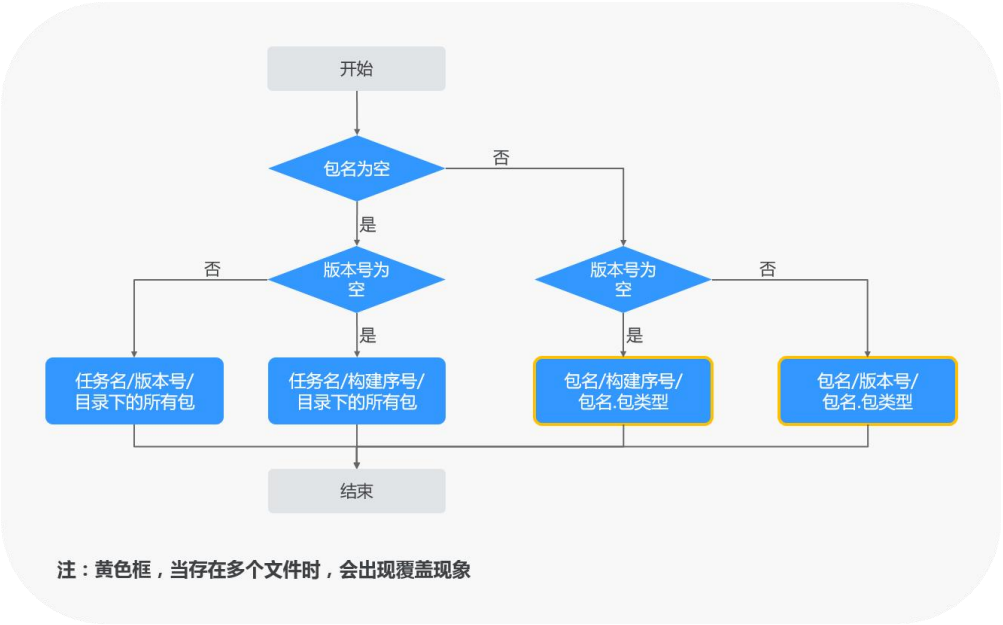
    递归匹配父目录为“test”文件是“XYZ”开头的所有文件，比如“abc/test/def/ghi/XYZ123”。

- **发布版本号及包名**

  包名推荐设置为空，这样可以上传构建包路径匹配的所有文件。

  设置包名后，一旦匹配多个文件时，会存在包覆盖的情况。假如包名需要设置且存在多个文件上传的情况，推荐增加多个上传软件包到软件发布库的构建步骤。

  发布版本号及包名是否为空对上传的影响如图：



4.5.1.26 上传文件到 OBS

将构建生成的软件包上传到 OBS，在配置构建步骤时，添加“上传文件到 OBS”构建步骤即可。

对象存储服务（OBS）的使用限制请参考约束与限制。

配置说明

添加“上传文件到 OBS”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
授权用户	• 当前用户：上传到当前租户的 OBS 桶。 • 其他用户：可以通过选择 IAM 帐号的方式上传到指定租户的 OBS 桶。
构建产物路径	构建结果所在路径，OBS 存储文件名为空时，可使用通配符上传多个文件。如：maven 可以使用**/target/*.?ar 匹配所有构建出来的 jar 包和 war 包。
桶名	目标 OBS 桶名（不支持跨 region 上传）。
OBS 存储目录	构建结果在 OBS 上的存储目录（如：application/version/），可留空，或填写“./”表示存储到 OBS 根目录。
OBS 存储文件名	构建结果在 OBS 上的存储文件名（不包含目录），留空时可上传多个文件，取构建产物文件名为 OBS 存储文件名；不为空时只能上传单个文件，如：application.jar。

参数项	说明
是否上传文件夹	可选择是否开启上传文件夹。
OBS 头域	上传文件时加入一个或多个自定义的响应头，当用户下载此对象或查询此对象元数据时，加入的自定义响应头会在返回消息的头域中出现。如：“键”填写成“x-frame-options”，“值”填写成“false”，即可禁止 OBS 中存放的网页被第三方网页嵌入。

4.5.1.27 执行 Docker 命令

介绍在 Docker 容器执行 Docker 命令的构建步骤：**执行 Docker 命令**。

配置说明

添加“执行 Docker 命令”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
命令	单击“添加”，新增一条命令行，请根据需要选择并配置命令，可参见常见的 Docker 命令。

常见的 Docker 命令

docker login

- **描述**：登录 docker 仓库
- **用法**：docker login [options] [server]
- **options**

名称，短名称	描述
--password, -p	密码。
--username, -u	用户名。
--password-stdin	从标准输入获取密码。

- **示例**

  docker login -u jack -p 12345 mydocker-registry.com

- **高级用法**

  从文件里将密码读入，cat ~/my_password.txt | docker login **--username** jack **--password-stdin**

docker build

- **描述**：通过 Dockerfile 或者上下文制作镜像
- **用法**：docker build [options] Path | URL | -
- **options**（只列几个常用的）

名称，短名称	描述
--file, -f	Dockerfile 名称，默认为./Dockerfile。
--tag, -t	“镜像名:标签”格式。

- **示例**

  docker build -t mydocker-registry.com/org/alpine:1.0 .

docker push

- **描述**：推送镜像到指定的地址
- **用法**：docker push [options] name[:tag]
- **示例**：docker push mydocker-registry.com/org/alpine:1.0

docker pull

- **描述**：从镜像仓库下载镜像到本地
- **用法**：docker pull [options] name[:tag|@digest]
- **options**

名称，短名称	描述
--all-tags, -a	下载镜像仓库所有有 tag 的镜像。

- **示例**

  docker pull mydokcer-registry.com/org/alpine:1.0

docker tag

- **描述**：创建一个镜像和标签指向原镜像
- **用法**：docker tag source_image[:tag] target_image[:tag]
- **示例**

  docker tag mydocker-registry.com/org/alpine:1.0 mydocker-registry/neworg/alpine:1.0

docker save

- **描述**：保存一个或者多个镜像到 tar 类型的文件，默认是标准输出流。
- **用法**：docker save [options] image [image ...]

- **options**

名称，短名称	描述
--output，-o	写文件，而不是使用标准输出。

- **示例**

  docker save **-o** alpine.tar mydocker-registry.com/org/alpine:1.0 mydocker-registry.com/org/alpine:2.0

docker logout

- **描述**：从镜像仓库登出
- **用法**：docker logout [server]
- **示例**：docker logout mydocker-registry.com

4.5.1.28 下载发布仓库包

背景信息

通过配置“下载发布仓库包”构建步骤，可以将发布仓库中的包或者其他文件下载到构建任务根目录，以便后续构建步骤使用这些包或者文件。

获取下载包地址

介绍从发布服务获取发布仓库包的下载地址。

步骤 1 登录软件开发生产线首页。

步骤 2 搜索目标项目并单击项目名称，在导航栏单击“制品仓库 > 软件发布库”。

步骤 3 进入软件发布库页面，查找待下载的仓库包。

步骤 4 单击待下载的仓库包名，弹出仓库包详情页面。

其中“下载地址”即为仓库包的下载地址，单击地址旁的，复制该地址。

----结束

配置下载发布仓库包

添加“下载发布仓库包”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
下载包地址	将步骤 4 复制的仓库包下载地址粘贴到输入框即可。

4.5.1.29 下载文件管理的文件

背景信息

通过配置“下载文件管理的文件”构建步骤，可以将文件管理的文件下载到工作目录。

配置说明

添加“下载文件管理的文件”构建步骤。

参数说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	根据需要选择工具版本。
下载文件	• 单击下拉列表，选择文件管理已上传的文件。 • 单击“上传”，可以将本地文件上传到文件管理。 • 单击“管理文件”，可跳转至“文件管理”页面对文件进行管理。

4.5.2 代码化构建

4.5.2.1 单任务配置

4.5.2.1.1 导读

编译构建支持通过 yaml 文件配置构建脚本，用户可以将构建过程需要用到的构建环境、构建参数、构建命令、构建步骤等操作通过 YAML 语法编写成 build.yml 文件，并且将 build.yml 文件随着被构建的代码一起纳入代码仓库，执行构建任务时，系统会以 build.yml 文件作为构建脚本执行构建任务，使构建过程可追溯、可还原，安全可信。

4.5.2.1.2 使用 yaml 构建任务

前提准备

- 已有可用项目，如果没有，请参考需求管理 CodeArts Req 服务的“用户指南 > Scrum 项目 > 新建 Scrum 项目”。
- 已在项目中新建可用代码仓库，如果没有，请参考代码托管 CodeArts Repo 服务的“用户指南 > 新版（推荐）创建 > 代码托管仓库”。
- 在代码仓库中，新建“.cloudbuild”目录，并将 yaml 文件存放在该目录下，yaml 文件编写方法及规范请参考 build.yml 文件的结构详解。

选择源码源

1. 。
2. 登录编译构建服务首页。
3. 单击“新建任务”，进入“基本信息”页面。
4. 选择“Repo”为源码源，并配置需要使用的源码仓库以及分支。

配置并执行 yaml 构建任务

1. 配置完源码源，单击“下一步”，进入“构建模板”页面。
2. 选择何种构建模板并不影响使用 yaml 构建，可以选择系统推荐模板，也可以不使用模板，选择“空白构建模板”。
3. 然后单击“下一步”。
4. 进入“构建步骤”页签，页面左上角选择“代码化”，系统会在选择源码源阶段配置的代码仓库及分支中，自动读取 yaml 文件，可在此处对 yaml 文件进行修改。
5. 修改完成后需单击右上角“新建并执行”，弹出“新建并执行”侧滑框。
6. 单击“新建并执行”，yaml 文件修改即可生效并运行 yaml 构建任务，构建脚本提交后将覆盖原 build.yml 文件。

4.5.2.1.3 yaml 文件结构详解

yaml 文件示例：

该示例是一个简单的通过 yaml 文件执行 Maven 构建模板：

```
version: 2.0 # 必须是 2.0
params: # 构建参数，可在构建过程中引用
  - name: paramA
    value: valueA
```

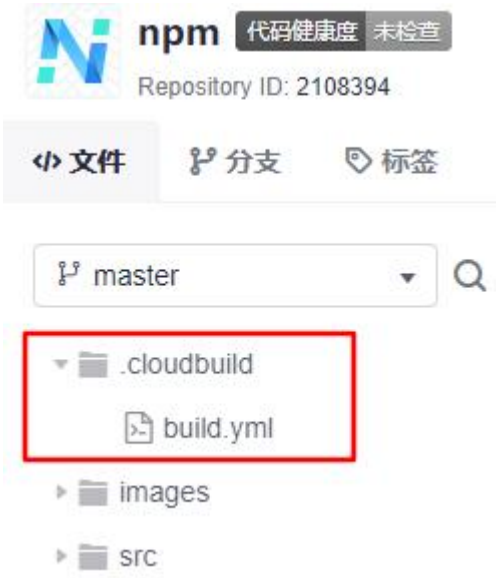
```
  - name: paramB
    value: valueB
env: # 非必填，但优先级最高，若在此定义了主机规格与类型，则不使用任务配置>基本信息里面选择的主机类型和规格
  resource:
    type:docker # 资源池类型：docker、Linux 或 MAC
    arch:X86 # 构建环境主机类型：X86 或 ARM
    class:8U16G # 规格：4U8G、8U16G 或 16U64G
steps:
  PRE_BUILD:
    - checkout:
        name: 代码下载 # 可选
        inputs: # 步骤参数
          scm: codehub # 代码来源:只支持 CodeArts Repo
          url: xxxxxxxxx # 拉取代码的 ssh 地址。
          branch: ${codeBranch} # 拉取的代码分支：支持参数化。
    - sh:
        inputs:
          command: echo ${paramA}
  BUILD:
    - maven: # 步骤关键字，仅支持特定关键字
      name: maven build # 可选
      image: xxx # 可以自定义镜像地址，请看下方说明
      inputs:
        command: mvn clean package
    - upload_artifact:
      inputs:
          path: "**/target/*.?ar"
  FINAL:
    - sh:
      inputs:
        command: echo hello world
```

在这个 yml 文件中，主要分为四个部分：

- **版本号（version）**：示例文件中定义了“version”为 2.0，该版本号必填且唯一。
- **构建环境（env）**：示例文件中定义了资源池类型、构建环境主机类型、主机规格。
- **构建参数（params）**：示例文件中定义了“paramA”和“paramB”两个参数，可在构建过程中引用被定义的参数，构建参数可不填，优先使用任务配置中的构建参数。
- **构建步骤(steps)**：示例文件中“steps”层级下又分为两个阶段：
  - **PRE_BUILD**：用于做构建前的准备，例如下载代码，执行 shell 等。
  - **BUILD**：用于构建，其下可定义 maven、npm、go、python、ant、CMake、mono、sbt、android、bazel 等主流工程构建。构建完成后，可定义制作镜像上传到 SWR 仓库、上传文件到 OBS、下载文件、上传二进制包至仓库、下载二进制包、执行 docker 命令等构建后操作。
  - **FINAL**：无论构建任务成功还是失败，都会在当前环境中执行此阶段中的步骤。用户可以自定义 shell 命令，仅支持配置一个 sh 步骤。

说明

- build.yml 文件定义好后，需严格遵循文件存放路径规则，将 build.yml 文件放在.cloudbuild 目录下。



- image 有两种格式：
- cloudbuild@maven3.5.3-jdk8-open ，以 cloudbuild 开始，@作为分隔符，后面是编译构建提供的默认镜像。
- 完整的 swr 镜像地址，例如：
  swr.xxx.xxx.com/codeci_test/demo:141d26c455abd6d7xxxxxxxxxxxxxxxxxxxx

构建步骤介绍

在 steps 中，共有 PRE_BUILD 和 BUILD 两个阶段，每一个阶段都可以定义一系列的构建步骤，支持定义的构建步骤，详见下表：

PRE_BUILD	说明	操作指导
- checkout	代码下载	使用 yaml 配置代码下载
- sh	执行 shell 命令	使用 yaml 配置执行 shell 命令

BUILD	说明	操作指导
- maven	Maven 构建	使用 yaml 配置 Maven 构建
- npm	NPM 构建	使用 yaml 配置 NPM 构建
- go	Golang 构建	使用 yaml 配置 Golang 构建
- python	Python 构建	使用 yaml 配置 Python 构建
- gradle	Gradle 构建	使用 yaml 配置 Gradle 构建

BUILD	说明	操作指导
- ant	Ant 构建	使用 yaml 配置 ant 构建
- cmake	CMake 构建	使用 yaml 配置 CMake 构建
- mono	mono 构建	使用 yaml 配置 mono 构建
- sbt	sbt 构建	使用 yaml 配置 sbt 构建
- android	Android 构建	使用 yaml 配置 Android 构建
- bazel	bazel 构建	使用 yaml 配置 bazel 构建
- build_image	制作镜像并上传到 SWR 仓库	使用 yaml 配置制作镜像并上传到 SWR 仓库
- upload_obs	上传文件至 OBS	使用 yaml 配置上传文件至 OBS
- download_file	下载文件	使用 yaml 配置下载文件
- upload_artifact	上传二进制包到仓库	使用 yaml 配置上传二进制包至仓库
- download_artifact	下载二进制包	使用 yaml 配置下载二进制包
- docker	执行 docker 命令	使用 yaml 配置执行 docker 命令

功能优势

整个构建过程都使用 YAML 的语法通过代码化的方式（Build As Code）定义在 build.yml 中，从而使得：

1. 清晰描述构建过程：构建参数、构建命令、构建步骤、以及构建后的操作，使构建过程可信。
2. 每次构建使用对应当前 commit 的 build.yml 配置，保证构建可还原可追溯，不必担心因修改了构建配置而不能重复执行之前的任务。
3. 如果新特性需要修改构建脚本，开发人员可以拉一个新的分支修改 build.yml 去测试，而不用担心影响其他分支。

4.5.2.1.4 使用 yaml 配置代码下载

配置参考如下:

```
version: 2.0 # 必须是 2.0
steps:
  PRE_BUILD:
    - checkout:
        name: checkout
```

```
inputs:
  scm: codehub # 代码来源:支持 CodeArts Repo 和 opensource
  url: xxxxxxxxx # 拉取代码的 ssh 地址。
  branch: ${codeBranch} # 任何时候都必填，支持参数化
  commit: ${commitId}
  lfs: true
  submodule: true
  depth: 100
  tag: ${tag}
  path: test
```

参数说明如下:

参数名	参数类型	描述	是否必填	默认值
scm	string	代码源：当前只支持 CodeArts Repo，如果 yaml 中没配置，则使用构建任务配置的代码仓信息。	否	codehub
url	string	拉取代码的 ssh 地址。	是	无
branch	string	拉取的代码分支：支持参数化。	是	无
commit	string	commitId 构建时拉取的 commitId：支持参数化。	否	无
tag	string	tag 构建时拉取的 tag：支持参数化，如果 commitId 和 tag 同时存在，优先执行 commitId 构建。	否	无
depth	int	浅克隆深度：当选择 commitId 构建时，depth 必须大于等于 commitId 所在深度。	否	1
submodule	bool	是否拉取子模块：true 为拉取；false 为不拉取。	否	false
lfs	bool	是否开启 git lfs：为 true 时会执行 git lfs pull。	否	false
path	string	clone 的子路径：代码将会下载到子目录下面。	否	无

4.5.2.1.5 使用 yaml 配置 manifest 多仓下载

在安卓、鸿蒙等场景下，一次构建需要同时集成数百甚至上千个代码仓，多代码仓的集成下载效率至关重要。

CodeArts Build 集成 Repo 下载工具，用户只需进行简单配置即可实现多个代码仓的联动集成。当前支持 CodeArts Repo、gerrit 两种类型的代码仓。

配置参考如下：

```
version: 2.0 # 必须是2.0
steps:
  PRE_BUILD:
  - manifest_checkout:
      name: "manifest"
      inputs:
        manifest_url: "https://xx.xx.xx-xxxxx-x.xx.xxx.com/xx/manifest.git"
        manifest_branch: "master"
        manifest_file: "default.xml"
        path: "dir/dir02"
        repo_url: "https://xx.xx.xx-xxxxx-x.xx.xxx.com/xx/git-repo.git"
        repo_branch: "master"
        username: "someone"
        password: "${PASSWD}"
```

参数说明如下：

参数名	参数类型	描述	是否必填	默认值
name	string	步骤名称。	否	manifest_checkout
manifest_url	string	指定 manifest 仓地址，包含 xml 文件的仓库。	是	无
manifest_branch	string	指定 manifest 分支或 revision。	否	HEAD
manifest_file	string	manifest 文件路径。	否	default.xml
path	string	自定义 manifest 所有子仓下载路径，为工作目录的相对路径 路径不能以“/”开头，不能包含“.”。	否	默认为工作目录。
repo_url	string	repo 仓库地址。	否	https://gerrit.googlesource.com/git-repo
repo_branch	string	repo 仓库分支。	否	stable
username	string	下载仓库时使用的用户名。	否； 下载非公开仓库时需填写。	无
password	string	下载仓库时使用的密码，https 密码。	否； 下载非公开仓库时需填写。	无

说明

1. manifest_file 中定义的多个仓库，必须为同一种源码源。
2. manifest_url 与 manifest_file 必须为同一种源码源；如果为非公开仓库，username&password 应该有下载权限。
3. repo_url 对应的 repo 仓库，需要有下载权限（仓库开源，或者仓库私有但配置了帐号密码）。
4. 以上非必填的参数，如果配置的值为空，则使用默认值。
5. 建议在使用非公开仓库时，用户名密码通过构建的私密参数进行配置，详情参考参数配置。

4.5.2.1.6 使用 yaml 配置执行 shell 命令

```
version: 2.0 # 必须是 2.0
steps:
  PRE_BUILD:
    - sh:
        inputs:
          command: echo ${a}
```

参数说明如下:

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.7 使用 yaml 配置 Maven 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - maven:
        image: cloudbuild@maven3.5.3-jdk8-open # 可以自定义镜像地址，请看下方说明
        inputs:
          settings:
            public_repos:
              - https://mirrors.xxx.com/maven
          cache: true # 是否开启缓存
          unit_test:
            coverage: true
            ignore_errors: false
            report_path: "**/TEST*.xml"
            enable: true
            coverage_report_path: "**/site/jacoco"
          command: mvn package -Dmaven.test.failure.ignore=true -U -e -X -B
```

说明

image 有两种格式：

1. cloudbuild@maven3.5.3-jdk8-open ，以 cloudbuild 开始，@作为分隔符，后面是编译构建提供的默认镜像。

2. 完整的 swr 镜像地址，例如:swr.xxx.xxx.com/codeci_test/demo:141d26c455abd6d7xxxxxxxxxxxxxxxxxxxx

配置说明

参数名	参数类型	描述	是否必填	默认值
settings	map	maven 构建的 setting 配置。	否	无
cache	bool	是否开启缓存。	否	false
command	string	执行命令。	是	无
unit_test	map	单元测试。	否	无

单元测试（unit_test）层级下参数说明:

参数名	参数类型	描述	是否必填	默认值
enable	bool	是否处理单元测试数据。	否	true
ignore_errors	bool	是否忽略单元测试错误。	否	true
report_path	String	单元测试数据路径。	是	无
converage	bool	是否处理覆盖率数据。	否	false
coverage_report_path	string	覆盖率数据路径。	否	无

4.5.2.1.8 使用 yaml 配置 NPM 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - npm:
        inputs:
          command: |
             export PATH=$PATH:~/.npm-global/bin
             npm config set registry https://repo.xxx.com/repository/npm/
             npm config set disturl https://repo.xxx.com/nodejs
             npm config set sass_binary_site https://repo.xxx.com/node-sass/
             npm config set phantomjs_cdnurl https://repo.xxx.com/phantomjs
             npm config set chromedriver_cdnurl https://repo.xxx.com/chromedriver
             npm config set operadriver_cdnurl https://repo.xxx.com/operadriver
             npm config set electron_mirror https://repo.xxx.com/electron/
             npm config set python_mirror https://repo.xxx.com/python
             npm config set prefix '~/.npm-global'
             npm install --verbose
             npm run build
```

参数名	参数类型	描述	是否必填	默认值

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.9 使用 yaml 配置 Golang 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - go:
        inputs:
          command: |
            export GO15VENDOREXPERIMENT=1
            export GOPROXY=https://goproxy.cn
            mkdir -p $GOPATH/src/example.com/demo/
            cp -rf . $GOPATH/src/example.com/demo/
            go install example.com/demo
            cp -rf $GOPATH/bin/ ./bin
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.10 使用 yaml 配置 Python 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - python:
        inputs:
          command: |
             pip config set global.index-url https://pypi.org/simple
             pip config set global.trusted-host repo.xxcloud.com
             python setup.py bdist_egg
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.11 使用 yaml 配置 Gradle 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - gradle:
```

```
      inputs:
         gradle: 4.8
         jdk: 1.8
         command: |
            # 使用 CodeArts 提供的 gradle wrapper,充分利用缓存加速
            cp /cache/android/wrapper/gradle-wrapper.jar ./gradle/wrapper/gradle-
wrapper.jar
            # 构建未签名的 APK
            /bin/bash ./gradlew build --init-
script  ./.codeci/.gradle/init_template.gradle -Dorg.gradle.daemon=false -
Dorg.gradle.internal.http.connectionTimeout=800000
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无
gradle	string	gradle 版本。	是	无
jdk	string	jdk 版本。	是	无

4.5.2.1.12 使用 yaml 配置 ant 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - ant:
        inputs:
          command:  ant -f build.xml
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.13 使用 yaml 配置 CMake 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - cmake:
        inputs:
          command: |
            # 新建 build 目录 切换到 build 目录、
            mkdir build && cd build
            # 生成 Unix 平台的 makefiles 文件并执行构建
            cmake -G 'Unix Makefiles' ../ && make -j
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.14 使用 yaml 配置 mono 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - mono:
        inputs:
           command: |
              nuget sources Disable -Name 'nuget.org'
              nuget sources add -Name 'xxcloud' -Source
'https://repo.xxcloud.com/repository/nuget/v3/index.json'
              nuget restore
              msbuild /p:OutputPath=../buildResult/Release/bin
              zip -rq ./archive.zip ./buildResult/Release/bin/*
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.15 使用 yaml 配置 sbt 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - sbt:
        inputs:
           command: |
              sbt package
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.16 使用 yaml 配置 Android 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - android:
        inputs:
```

```
        gradle: 4.8
        jdk: 1.8
        ndk: 17
        command: |
          cat ~/.gradle/init.gradle
          cat ~/.gradle/gradle.properties
          cat ~/.gradle/init_template.gradle
          rm -rf ~/.gradle/init.gradle
          rm -rf /home/build/.gradle/init.gradle
          # 使用 CodeArts 提供的 gradle wrapper,充分利用缓存加速
          cp /cache/android/wrapper/gradle-wrapper.jar ./gradle/wrapper/gradle-
wrapper.jar
          # 构建未签名的 APK
          /bin/bash ./gradlew assembleDebug -Dorg.gradle.daemon=false -d --
stacktrace
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无
gradle	string	gradle 版本。	是	无
jdk	string	jdk 版本。	是	无
ndk	string	ndk 版本。	是	无

4.5.2.1.17 使用 yaml 配置 bazel 构建

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - bazel:
        inputs:
          command: |
             cd java-maven
             bazel build //:java-maven_deploy.jar
             mkdir build_out
             cp -r  bazel-bin/*  build_out/
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。	是	无

4.5.2.1.18 使用 yaml 配置制作镜像并上传到 SWR 仓库

上传到 SWR 前，需了解 SWR（容器镜像服务）的约束与限制。

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - build_image:
        name: buildImage
        inputs:
          regions: ["x-x-x", "x-x-xxx"]
          organization: codeci_test
          image_name: demo
          image_tag: ${GIT_COMMIT}
          dockerfile_path: dockerfile/Dockerfile
          # set_meta_data: true
```

参数名	参数类型	描述	是否必填	默认值
regions	list	选择要上传的区域 SWR。默认上传到当前任务所在 region 的 SWR。	否	无
organization	string	上传到的 SWR 组织。	是	无
image_name	string	镜像名。	否	demo
image_tag	string	镜像标签。	否	v1.1
context_path	string	docker 的上下文路径。	否	.
dockerfile_path	string	dockerfile 文件相对 context_path 的路径。	否	./Dockerfile
set_meta_data	bool	是否添加构建元数据到镜像。	否	false

4.5.2.1.19 使用 yaml 配置上传文件至 OBS

对象存储服务（OBS）的使用限制请参考约束与限制。

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - upload_obs:
        inputs:
          artifact_path: "**/target/*.?ar"
          bucket_name: codecitest-obs
          obs_directory: test
        #   artifact_dest_name: ""
        #   upload_directory: true
        #   headers:
        #     x-frame-options: true
        #     test: test
        #     commit: ${commitId}
```

参数名	参数类型	描述	是否必填	默认值
artifact_path	string	要上传的产物路径，支持正则。	否	bin/*
bucket_name	string	要上传到的 obs 桶名。	是	无
obs_directory	string	要上传到的 obs 文件夹路径。默认上传到桶的根目录。	否	./
artifact_dest_name	string	上传到 obs 后的文件名。产物需要重命名时填写。	否	无
upload_directory	bool	是否上传文件夹。false 时会将匹配到的所有产物平铺上传到 obs_directory。	否	false
headers	map	上传的头域信息。	否	无

4.5.2.1.20 使用 yaml 配置下载文件

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - download_file:
        inputs:
          name: android22.jks
```

参数名	参数类型	描述	是否必填	默认值
name	string	文件名称。	是	无

4.5.2.1.21 使用 yaml 配置上传二进制包至仓库

上传的软件包相关限制请参考制品仓库服务的约束与限制。

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - upload_artifact:
        inputs:
          path: "**/target/*.?ar"
          version: 2.1
          name: packageName
```

参数名	参数类型	描述	是否必填	默认值
path	string	构建结果所在路径，支持正则表达式。如 maven 可以使用**/target/*.?ar 匹配所有构建	是	无

参数名	参数类型	描述	是否必填	默认值
		出来的 jar 包和 war 包。		
version	string	构建包在发布仓库的存储目录，推荐不填，不填写时，自动使用构建编号作为目录名。	否	无
name	string	上传发布仓的文件名都会改成包名，推荐不填，不填会保留原文件名。	否	无

4.5.2.1.22 使用 yaml 配置下载二进制包

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - download_artifact:
        inputs:
          url: xxxxxxxxxxxx
```

参数名	参数类型	描述	是否必填	默认值
url	string	下载 url（软件发布库中二进制包的部署下载地址）。	是	无

4.5.2.1.23 使用 yaml 配置执行 docker 命令

```
version: 2.0 # 必须是 2.0
steps:
  BUILD:
    - docker:
        inputs:
          command: |
            docker pull swr.xx-xxxxx-
x.myxxcloud.com/codeci/dockerindocker:dockerindocker18.09-1.3.2
```

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令，每个命令一行。支持的 docker 命令:build、tag、push、pull、login、logout、save。	是	无

4.5.2.2 多任务配置

4.5.2.2.1 导读

在编译构建中，构建任务是构建的最小单元，适用于业务比较简单的场景，但是在有些复杂的构建场景下，构建任务可能并不能满足复杂的构建要求。例如：

- 多仓工程需要分布到多个机器上去构建，并且构建工程之间还存在一定的依赖关系。
- 希望更模块化、更加细粒度的拆分构建任务，并按照依赖顺序进行构建。

对于上述这类比较复杂的构建场景，编译构建支持使用 BuildFlow 将多个存在依赖关系的构建任务按照有向无环图（DAG）的方式组装起来，BuildFlow 将会按照构建的依赖关系并发进行构建。

4.5.2.2.2 yaml 文件结构详解

BuildFlow 整体介绍

BuildFlow 示例：

```
version: 2.0 # 必须是 2.0
params:
  - name: buildFlowParam
    value: buildFlowValue
buildflow:
 strategy: lazy # 定义 buildFlow 运行的策略,lazy/eager
 jobs: # 构建任务
   - job: Job3
     depends_on: # 定义 job 的依赖,实例中 Job3 依赖 Job1,Job2
        - Job1
        - Job2
     build_ref: .cloudbuild/build3.yml # 定义 Job 在构建过程中需要运行的 yaml 构建脚本
   - job: Job1
     build_ref: .cloudbuild/build1.yml
   - job: Job2
     build_ref: .cloudbuild/build2.yml
```

在这个 BuildFlow 中，包含几个关键要素：

- **version**：版本号，必填且唯一，示例文件中定义的“version”为 2.0。
- **params**：构建参数，BuildFlow 的全局参数，该参数会被所有的 Job 共享。
- **strategy**：运行策略，共有两种运行模式，如果没有显式的定义，默认使用 Eager 模式。
  - Lazy：先触发优先级高的子任务构建，优先级高的子任务执行成功之后，再触发优先级低的子任务。

  说明

  构建时间相对较长，但是可以节省构建资源，推荐在并发数不足时使用。

  - Eager：同步触发所有子任务的构建，有依赖其它任务的子任务会先准备好环境和代码，等待所依赖的任务构建成功。

  说明

  可能造成资源空闲等待，但是可以缩短构建时间，推荐在并发数足够大的情况下使用。
- **Jobs**：需要进行编排的任务，示例文件中“Jobs”层级下又分出 3 个参数。
  - **job**：构建任务的名称，可自定义修改。
  - **depends_on**：该构建任务所依赖的构建任务。
  - **build_ref**：该构建任务在构建过程中需要运行的 yaml 构建脚本。

可以发现该 BuildFlow 示例中，共配置了三个构建任务：Job1、Job2、Job3，三个构建任务共享已被定义的参数 params，并且构建任务 Job3 依赖 Job1 和 Job2。

BuildFlow jobs 介绍

BuildFlow Jobs 用来定义 BuildFlow 中需要进行编排的任务，每一个 Job 都必须要有一个唯一的名字作为构建任务的唯一标识。

说明

- 若子任务 A 依赖子任务 B，则构建优先级 B > A。
- 优先级相同的子任务会同步触发。

BuildFlow jobs 示例：

```
buildflow:
  strategy: lazy
  jobs:
   - job: Job3
     depends_on:
        - Job1
        - Job2
     build_ref: .cloudbuild/build3.yml
   - job: Job1
     build_ref: .cloudbuild/build1.yml
   - job: Job2
     build_ref: .cloudbuild/build2.yml
```

如上所示，Job3 依赖于 Job1 和 Job2，即构建优先级 Job1、Job2 > Job3，且 Job1 和 Job2 同步触发。

BuildFlow params 介绍

BuildFlow params 可以定义全局参数，即所有 job 共享。但是有些时候全局参数的粒度可能太大了，只需要在部分的 Job 上定义参数。编译构建也支持只在部分 Job 上定义参数使用，例如：

```
buildflow:
 jobs:
   - job: Job3
     depends_on:
        - Build Job1
        - Build job2
     build_ref: .cloudbuild/build3.yml
   - job: Job1
     params:
       - name: isSubmodule
         value: true
     build_ref: .cloudbuild/build1.yml
   - job: Job2
     params:
       - name: isSubmodule
         value: true
     build_ref: .cloudbuild/build2.yml
```

如上所示，BuildFlow 中并未定义全局参数 params，而是将参数“isSubmodule”直接定义在 Job1 与 Job2 中。

说明

在使用 yaml 构建时，需注意参数使用的优先级：

运行时参数 > 构建任务参数设置里配置的参数 > BuildFlow 子任务 yaml 文件中定义的参数 > BuildFlow 父任务 yaml 文件中 Job 上定义的参数 > BuildFlow 父任务 yaml 文件中定义的全局参数。

4.5.3 文件管理

文件管理主要用来存储 Android APK 的签名文件和 Maven 构建 settings.xml 文件并提供对这类文件的管理（如：编辑、删除、权限设置），文件大小限制为 100k，文件类型限制为：**.xml、.key、.keystore、.jks、.crt、.pem。**

- 上传文件。
  a. 。
  b. 登录编译构建服务首页。
  c. 单击右上角“更多”，选择“文件管理”。
  d. 单击“上传文件”。
  e. 在弹出的窗口中选择文件，添加描述，勾选相关协议，然后单击“保存”。

- 文件管理。
  文件上传后，可以编辑文件、下载文件、删除文件、为用户配置文件操作权限。
  - 单击操作列，可修改文件名称，并设置是否允许租户内所有成员在编译构建中使用该文件。
  - 单击操作列，可以下载文件。
  - 单击操作列，请根据弹框提示确认是否删除。
  - 单击操作列，在弹出的界面配置用户操作文件的权限。

表 4-4 文件管理角色权限说明

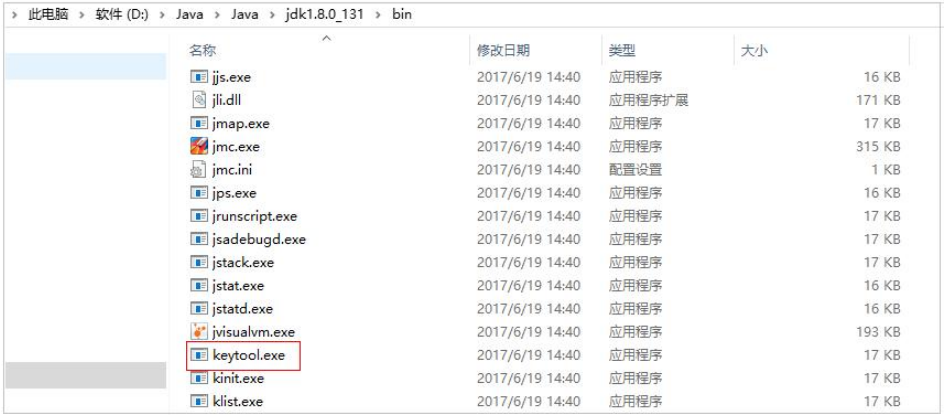
权限类型	拥有该权限的角色
创建	项目下所有用户。
查看	文件创建者、相同租户的用户。
使用	文件创建者、文件创建者配置了使用权限的用户。
更新	文件创建者、文件创建者配置了更新权限的用户。
删除	文件创建者、文件创建者配置了删除权限的用户。
权限配置	文件创建者。

说明

创建者默认有所有权限并且不可被删除和修改。

生成 Keystore 签名文件

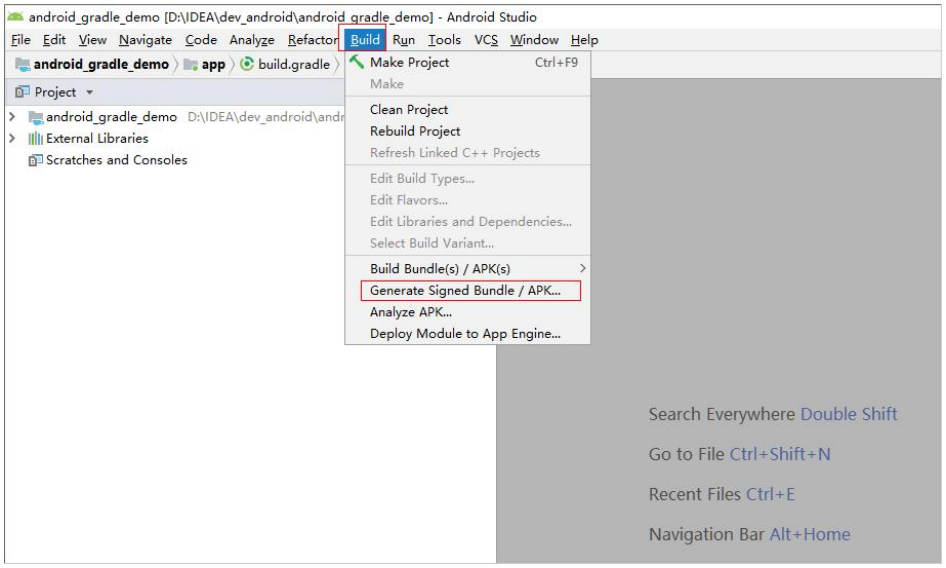
- **使用 JDK 的 keytool 工具生成签名文件**
  a. 找到 JDK 安装位置以及 keytool。



  b. 执行生成密钥命令，生成.jks 文件。

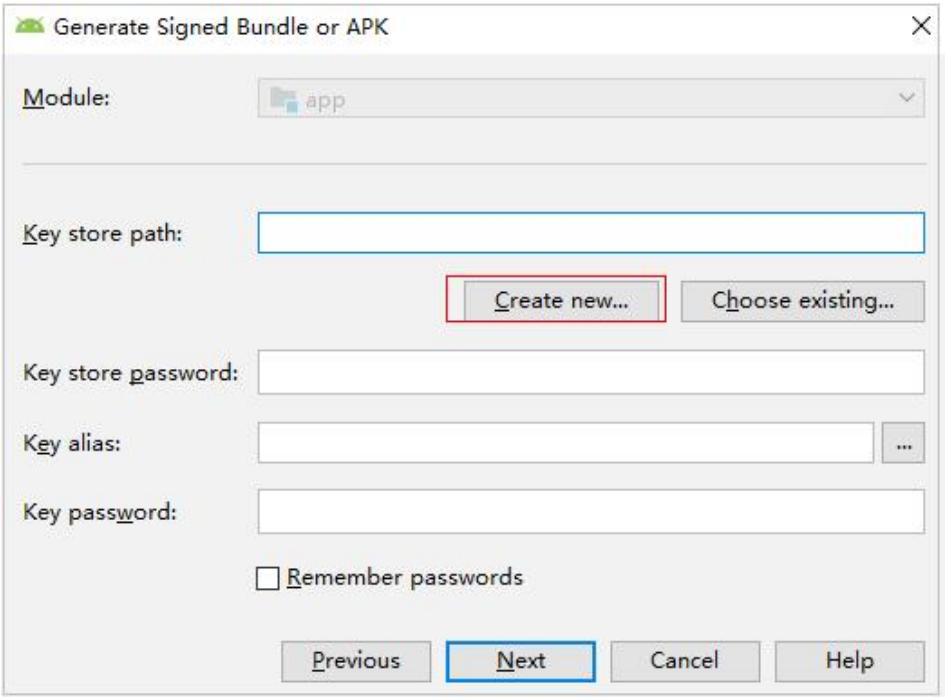
```
keytool -genkeypair -storepass 123456 -alias apksign -keypass 123456 -
keyalg RSA -validity 20000 -keystore D:/android.jks
```

- **使用 Android Studio 生成签名文件**
  a. 打开 Studio，选择“Build 下的 Generate Signed Bundle/APK”。

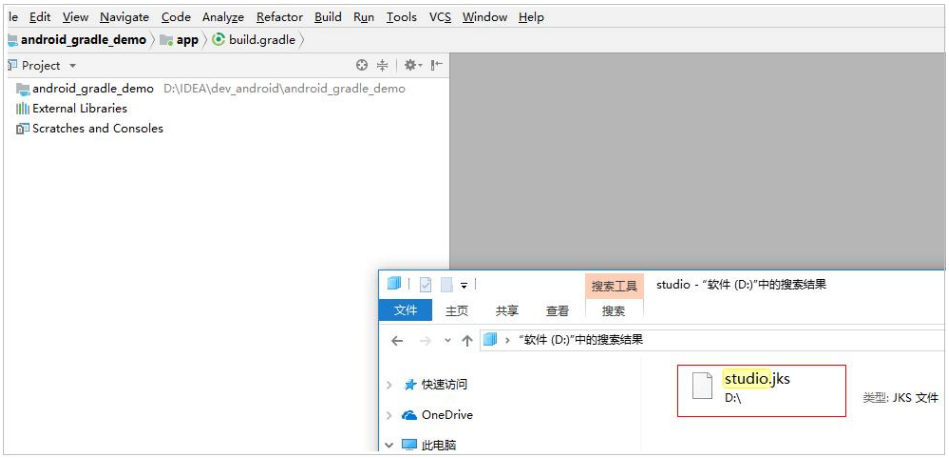


b. 选择“APK”，单击“Next”。

c. 单击“Create new...”，在弹出框填写相关信息，单击“OK”，然后单击“Next”。



d. 签名文件成功生成，查看文件。



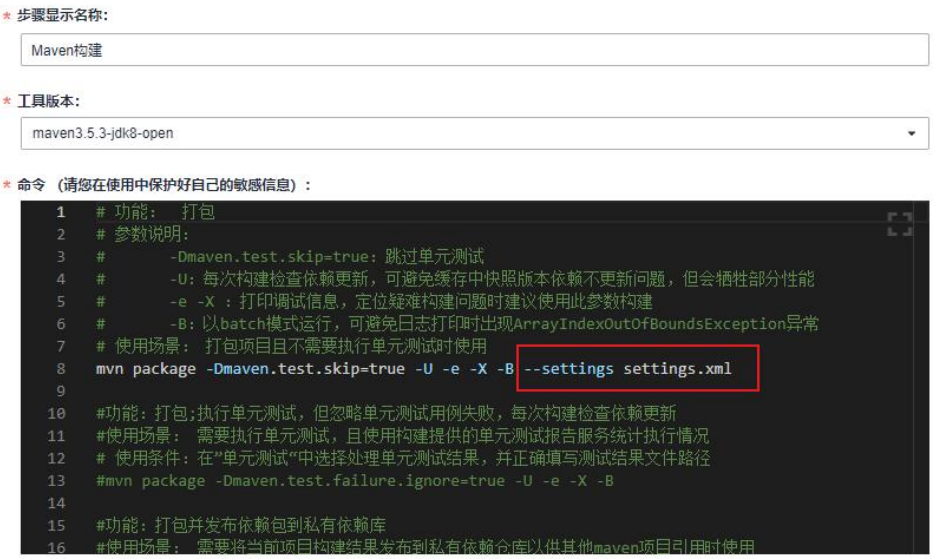
说明

生成的签名文件，可以上传到“文件管理”统一管理。

使用 settings.xml 文件

1. 新建或编辑 Maven 构建任务时，在“构建步骤”页签，添加“下载文件管理的文件”步骤，然后选择上传的 settings.xml 文件。

2. 在“Maven 构建”默认命令末尾添加“--settings settings.xml”，即可使用已添加的 settings.xml 文件执行 Maven 构建。



5 执行构建任务

前提条件

- 已新建构建任务，且用户具有执行/禁用构建任务的权限。
- 项目创建者、任务创建者、项目经理、开发人员可以执行/禁用编译构建任务。

执行任务

1. 登录编译构建服务首页。
2. 在编译构建服务首页搜索目标任务，单击构建任务所在行的 ▷ ，开始执行构建任务。

说明

如果构建任务配置了运行时参数且被引用，将弹出参数设置提示框，根据需要设置执行参数值。

禁用任务

1. 在编译构建首页搜索目标任务。
2. 单击构建任务所在行的 *** ，在下拉列表中选择“禁用”。
3. 弹出“任务禁用”提示框，输入禁用原因，单击“确定”。

说明

- 构建任务执行中无法禁用和删除。
- 构建任务被禁用后，构建任务名称后会出现“已禁用”标识，此时不能再执行构建任务；如需执行，请单击构建任务所在行的 *** ，在下拉列表中选择“取消禁用”。

6 查看构建任务

1. 登录编译构建服务首页。
2. 首页展示与当前用户相关的编译构建任务列表，列表项说明如下：

列表项	说明
名称	构建任务所属项目名及构建任务名，单击项目名可以进入到项目下编译构建列表，单击任务名可以进入到构建历史页面。构建成功标记为绿色、构建失败为红色、构建中止为黄色、未构建为浅灰色。
最近一次执行	任务执行人员、触发方式、所用仓库的分支、CommitID 等信息。
最近执行结果	从右到左显示最近执行结果，绿色为成功，蓝色为执行中，红色为失败。
启动时间 & 执行时长	构建任务启动时间和构建所用时长。
操作	 开始构建、  收藏任务、单击 ··· 展开下拉菜单（编辑、复制、禁用、删除任务，具体操作请参考编辑构建任务相关操作）。

3. 单击构建任务名称，进入“构建历史”列表页面，可以查看最近的构建历史记录（默认 30 天，可通过页面左上角的“日期选择组件”自定义时间周期）。
4. 单击“洞察”页签，以饼状图/折线图/柱状图的方式查看近 7 天的构建成功率与构建性能分布。
5. 单击构建历史下的构建编号，即可查看构建详情，包括代码源信息、触发来源、构建时间与时长、关联信息、
   - 单击左上角代码源链接，可进入对应代码仓库页面。
   - 单击“构建包下载”，可以下载本次构建成功的包。
   - 单击左侧构建步骤节点（如“代码检出”），可以查看对应编译构建日志。
   - 查看日志信息时，单击日志窗口右上角“全屏”，可最大化日志窗口；单击“退出全屏”，可退出最大化日志窗口；单击“日志”，可下载全量日志文件；单击左侧步骤节点，可查看对应步骤日志。

- 单击右上角“编辑”或“执行”按钮，可以编辑构建任务或执行构建任务，单击  ，可以根据需要复制任务、保存模板、查看徽标状态、禁用任务。

7 编辑构建任务相关操作

7.1 编辑/删除/复制/收藏构建任务

在操作编译构建任务前，需具备相应操作权限。

编辑构建任务

1. 登录编译构建服务首页。
2. 在编译构建任务列表搜索目标任务。
3. 单击编译构建任务所在行...，在下拉列表中选择“编辑”，进入“编辑任务”页面。
   - 基本信息：可修改任务名称、源码源、源码仓库、分支、任务描述等信息。
   - 构建步骤：可修改构建步骤、步骤参数等信息。
   - 参数设置：可配置执行任务时的自定义参数。
   - 执行计划：可配置触发事件（持续集成）和定时执行。
   - 修改历史：可查看构建任务的修改记录。
   - 权限管理：，可配置不同角色的权限。
   - 通知：可配置任务事件类型通知信息（包括任务构建成功、失败、删除、配置更新、被禁用）。
4. 根据需要选择对应页签并进行编辑，单击“保存并执行 > 保存”完成修改。

删除构建任务

1. 在编译构建任务列表搜索目标任务。
2. 单击编译构建任务所在行...，在下拉列表中选择“删除任务”，请根据实际情况确定是否删除对应构建任务。

复制构建任务

1. 在编译构建任务列表中搜索目标任务。
2. 单击编译构建任务所在行的...，在弹出的下拉列表选项单击“复制”，进入编译构建复制页面。
3. 根据需要修改任务信息，单击“复制”，即可复制该构建任务。

说明

复制任务会保留原任务的权限矩阵。

收藏构建任务

1. 在编译构建任务列表搜索目标任务。
2. 鼠标移至任务所在行，单击，图标变色即收藏成功。
3. （可选）单击，即可取消收藏。

说明

- 收藏构建任务后，刷新页面或下次进入任务列表时，该任务会在任务列表中置顶显示，收藏多个任务会按任务创建时间降序排列。
- 收藏非自己创建的任务，可以根据该任务设置的通知事件类型获取响应的通知。

7.2 配置构建参数

编译构建默认生成 **codeBranch** 参数和系统预定义参数。用户可以根据需要修改 **codeBranch** 参数类型和参数值，并添加其他自定义参数；系统预定义参数的参数值由系统自动生成，不需定义，可通过${参数名}引用。

参数配置

1. 登录编译构建服务首页。
2. 在编译构建任务列表搜索目标任务。
3. 单击编译构建任务所在行，在下拉列表中选择“编辑”，进入“编辑任务”页面。
4. 切至“参数设置”页签。

参数信息说明如下：

基本信息	说明
名称	参数名称。除系统默认生成的 **codeBranch** 参数和系统预定义参数，其余自定义新增的参数可自定义修改参数名称。
类型	参数可选类型。包括字符串类型和枚举类型。
默认值	参数的默认值。选择不同类型的参数系统都会自动生成对应的默认值，可根据需要修改参数值。
私密参数	参数为私密参数时，系统会将输入参数做加密存储，使用时进行解密，同时在运行日志里不可见。
运行时设置	打开表示单独执行构建任务时支持变更参数值，并且也会把该参数上报流水线。运行时参数需要执行时输入。

- 添加字符串类型参数

  单击“新增参数”，默认新增一条字符串类型参数，可根据需要修改参数名、参数类型、参数值，以及是否设置为私密参数或者运行时设置。

- 添加枚举类型参数

  i. 单击“新增参数”，默认新增一条字符串类型参数。

  ii. 单击参数类型旁的，在下拉列表选择“枚举”，弹出“枚举参数”对话框。

  iii. 为参数设置“可选取值”，每个参数值必须以英文分号结尾。

  iv. 设置完成后，在“参数值”列单击下拉列表选择其中一个值。

参数使用

分别举例介绍自定义参数和系统预定义参数的使用。

说明

在使用 yaml 构建时，需注意参数使用的优先级：

运行时参数 > 构建任务参数设置里配置的参数 > BuildFlow 子任务 yaml 文件中定义的参数 > BuildFlow 父任务 yaml 文件中 Job 上定义的参数 > BuildFlow 父任务 yaml 文件中定义的全局参数。

- **自定义参数**

  a. 配置执行参数。

  编辑构建任务，选择“参数设置”页签，添加一条参数，设置参数名称和参数值（本例参数名设置为“myparam”、默认值设置为“1.0.1.1”），打开“运行时设置”。

  b. 使用执行参数。

  切换到“构建步骤”页签，配置构建步骤，在发布版本号文本框里输入“${myparam}”,保存构建任务。

  c. 执行构建任务。

  弹出“设定参数并执行”框，根据实际情况输入值或者使用默认值。

d. 本构建任务是 Maven 构建并且开通了制品仓库服务，所以可以在制品仓库服务里查到该任务的构建包。

进入软件发布库，找到刚构建的构建包，即可看到版本号就是用户自定义的执行参数“myparam”值。

- **系统预定义参数**

a. 配置执行参数。

编辑构建任务，选择“构建步骤”页签，配置构建步骤，在发布版本号文本框里输入“${BUILDNUMBER}”，保存构建任务。

参数名	说明
BUILDNUMBER	构建编号。格式为“日期.今日该构建任务执行次数”，例如：20200312.3
GIT_COMMIT	代码提交号。例如：b6192120acc67074990127864d3fecaf259b20f5
TIMESTAMP	构建执行时间戳。例如：20190219191621
INCREASENUM	该任务构建执行总次数，从 1 开始自增长，每执行 1 次加 1。
PROJECT_ID	项目编号。
WORKSPACE	工作空间，源代码根目录。
GIT_TAG	代码 tag 名，使用 tag 构建时才有值。

b. 运行构建任务。

c. 本构建任务是 Maven 构建并且开通了制品仓库服务，所以可以在软件发布库里查到该任务的构建包。

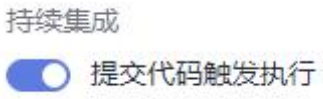
进入软件发布库，找到刚构建的构建包，即可看到版本号就是系统的执行参数“BUILDNUMBER”的值。

7.3 配置执行计划

编译构建支持用户配置触发事件和定时执行任务，从而使得开发者达到项目持续集成的目的。

持续集成

1. 登录编译构建服务首页。
1. 在编译构建任务列表搜索目标任务。
2. 单击操作列 ***，在下拉列表中选择“编辑”，进入“执行计划”页签。
3. 将“提交代码触发执行”按钮设置为开启状态。



4. 功能开启后，构建任务所引用的源码源发生提交代码行为时，则会触发构建任务。

定时执行

1. 将“启用定时执行”按钮设置为开启状态。

2. 选择需要构建任务定时执行的时间，并可按需开启是否“代码变化才执行”。
3. 功能开启后，构建任务会按照您设定的执行日与时间定时执行。
4. 若同时开启了“代码变化才执行”按钮，只有到达设定的执行日和时间，并且代码与上次构建有所变动时才会执行构建任务。

7.4 配置角色权限

编译构建支持为当前构建任务的各个角色配置权限。

操作步骤

1. 登录编译构建服务首页。

步骤 1 在编译构建任务列表搜索目标任务。

步骤 2 单击编译构建任务所在行的 *******，在下拉列表中选择“编辑”，进入到“权限管理”页签。

步骤 3 可根据实际需要配置不同角色的操作权限。

----结束

7.5 配置事件通知

编译构建支持给用户发送事件通知。任务构建成功、任务构建失败、任务被禁用、任务配置被更新和任务被删除时，可以给用户发送消息通知、邮件通知。

消息/邮件通知

1. 登录编译构建服务首页。
2. 在编译构建任务列表搜索目标任务。
3. 单击操作列 ***，在下拉列表中选择“编辑”，进入“编辑任务 > 构建步骤”页签。
4. 切换至“通知”页签，分别选择“消息”通知和“邮件”通知进行设置。

   默认所有事件都发送消息通知，构建任务失败发送邮件通知，请根据实际需要单击 开启通知，单击  关闭通知。

8 其他相关操作

8.1 构建任务回收站

构建任务被删除后，将会保存在构建任务回收站中。租户帐号可以对删除后的构建任务进行管理。

步骤 1 登录编译构建服务首页。

步骤 2 在编译构建首页右上角单击“更多”，在下拉列表选择“构建任务回收站”。

页面中展示已删除的构建任务，根据需要可以完成以下相关操作。

操作	说明
修改任务保留时间	单击“任务保留时间”下拉列表，根据需要选择时长，可选天数范围为 1~30 天。
搜索任务	在搜索框中输入待搜索内容，单击搜索，即可在页面中查看搜索结果。
删除任务	在列表中勾选待删除的任务，单击“删除”，即可将所选任务从回收站中删除。
恢复任务	在列表中勾选待恢复的任务，单击“恢复”，即可将所选任务恢复到编译构建服务的任务列表中。
清空回收站	单击“清空回收站”，可将回收站中所有任务删除。

----结束