编译构建

用户指南

文档版本 01

发布日期 2023-11-15

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 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。

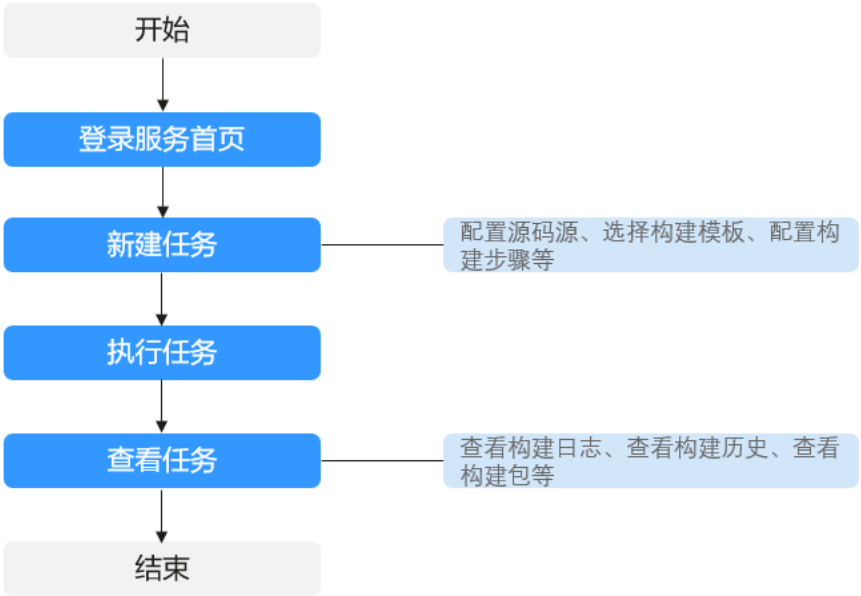
如果当前角色权限不满足用户需求，可参考**配置角色权限**进行设置。

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
登录服务首页	访问编译构建服务首页。

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 登录服务首页

前提条件

- 已**注册华为账号并开通华为云**。

操作步骤

步骤1 **登录华为云控制台页面**。

步骤2 单击页面左上角 ☰ ，在服务列表中选择“开发与运维 > 编译构建CodeArts Build”。

步骤3 单击“立即使用”即可进入服务首页。

- 单击页面左上角，根据需要选择区域。
- 单击“更多”，可使用以下功能。
  - **自定义模板**
  - **自定义构建环境**
  - **文件管理**
  - **构建任务回收站**
  - **构建资源池管理**

----结束

5 新建构建任务

前提条件

- 已有可用项目，如果没有，请**新建项目**。
- 已在项目中新建可用代码仓库，如果没有，请**新建代码仓库**。

配置基本信息

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 5-1 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。 • GitHub：对于托管在GitHub上的代码，可以使用GitHub连接实现代码拉取，详见**使用GitHub仓库构建**。 • 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。
源码仓库	选择实际使用的源码仓库。
分支	选择仓库分支。
任务描述	对任务进行描述。

配置构建模板

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”。
   如果没有合适的构建模板，可**自定义模板**。

配置构建步骤

1. 单击“下一步”，进入“构建步骤”页签，页面展示所选模板的默认步骤组合。
2. 单击构建步骤上的可根据是需要添加构建步骤。
   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、源码仓库、分支和任务描述。
- 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。

6 配置构建步骤

6.1 构建步骤导读

编译构建服务支持**图形化构建**和**代码化构建**。

图形化构建

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。如果预置的构建工具版本无法满足用户使用需求，也可以自定义构建环境，将所需环境打包制作成Docker镜像并推送至SWR镜像仓库后使用，详情可参考**制作镜像并推送到SWR仓库**和**使用SWR公共镜像**。

代码化构建

（代码化构建仅支持**源码源为Repo**。）

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。功能优势如下：

- 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。
- 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。
- 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。

代码化构建支持**单任务配置**和**多任务配置**。

6.2 图形化构建

6.2.1 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。

Mac执行机：

- 当前编译构建支持使用Mac执行机构建，支持当前在使用的所有Mac版本。
- 当选择Mac执行机时，构建步骤仅支持**执行shell命令**、**上传软件包到软件发布库**和**下载发布仓库包**。

配置说明

预置“构建环境配置”步骤。

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。
执行主机	用来执行编译构建任务的计算资源，在编译构建服务中，该计算资源为虚拟机。执行主机包括内置执行机和自定义执行机。 • 内置执行机：编译构建服务自身提供的执行主机，用户无需配置即可开箱即用。 • 自定义执行机：用户自行提供的计算资源，通过注册的方式托管到编译构建服务中，通过编译构建服务进行调度并执行构建任务。 可根据实际情况选择内置执行机或自定义执行机，自定义执行机为在资源池中添加的代理执行机，具体自定义操作可参考**资源池管理**。

6.2.2 代码下载配置

配置代码下载方式，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）自动更新与Git LFS。

配置说明

预置“代码下载配置”步骤。

参数说明如下：

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	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**子模块管理（Git Submodule操作）**。 • 开启：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。 • 不开启：系统不会自动拉取子模块仓库的代码。
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，若源码源选择Repo，那么关于如何创建Tag可参见**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，若源码源选择Repo，则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。

6.2.3 Maven 构建

6.2.3.1 操作指导

内置了Maven、jdk等工具，可根据构建场景选择工具版本。

使用Maven构建Java项目，主要包含以下功能：

- 可执行mvn package命令、mvn deploy命令或其他shell命令进行构建。
- 支持使用非软件开发生产线提供的公开依赖仓库构建。
- 支持添加其他私有依赖仓库。
- 支持自动在“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。
- 支持构建后查看Junit单元测试报告。

配置说明

在**配置构建步骤**中，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	自动生成setting文件并配置依赖仓库：可根据用户的IP不同，自动识别最优站点访问方式，国内用户使用“国内站点”，国际用户使用“国际站点”。建议使用默认配置。 如果需要的依赖无法在华为开源镜像站、CodeArts私有依赖仓库、HuaweiSDK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	编译构建服务默认使用私有依赖库作为私有依赖下载源，如果需要将构建产物上传至私有依赖库供其他项目依赖使用，则需要添加此配置。详见**配置发布依赖包到私有依赖库**。
单元测试	如果用户需要对单元测试结果进行处理，可配置此项。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。

6.2.3.2 配置依赖仓库

配置说明

如果用户需要使用非软件开发生产线提供的依赖仓库进行构建，可通过本页指导配置依赖仓库。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**配置自定义公有依赖仓库**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，访问需要授权账号认证），需在构建步骤“Maven构建”中**配置自定义私有依赖仓库**才能使用。

配置自定义公有依赖仓库

1. 在**Maven构建**步骤中，展开“setting配置”。
2. 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

1. **新建nexus repository服务扩展点**，如：test01。
2. 在**Maven构建**步骤中，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”。

6.2.3.3 配置发布依赖包到私有依赖库

配置说明

编译构建服务默认使用私有依赖库作为私有依赖下载源，如果需要将构建产物上传至私有依赖库供其他项目依赖使用，则需要添加此配置。

- **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。

  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。

说明

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 **创建私有依赖库**（如果已经创建，请忽略该步骤）。

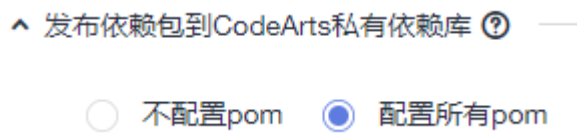
步骤2 使用Maven模板**新建代码仓库**。

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/XMLSchema-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>
```

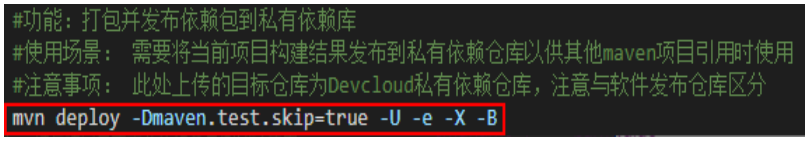
步骤4 在**Maven构建**步骤中，展开“发布依赖包到CodeArts私有依赖库”，选择“配置所有pom”。



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

步骤6 删除**#mvn deploy -Dmaven.test.skip=true -U -e -X -B**命令前的“#”。



步骤7 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8 单击导航栏“制品仓库 > 私有依赖库”，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>
```

----结束

6.2.3.4 配置单元测试

配置说明

如果用户需要对单元测试结果进行处理，可对**Maven构建**提供的单元测试功能进行配置，需要在项目中编写单元测试代码，且需满足如下条件：

- 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为“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的配置代码。

  单击代码仓库名称，进入代码托管“文件”页，确保以下代码未存在于项目“pom.xml”文件中。

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

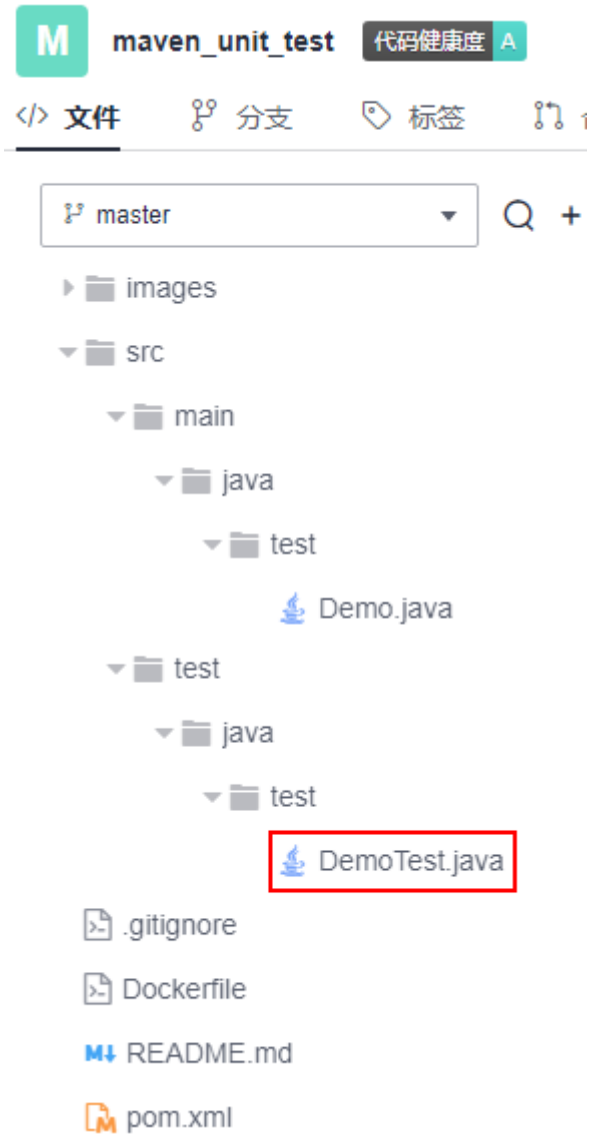
- 单击代码仓库名称，进入代码托管“文件”页，“pom.xml”文件中需引入junit依赖，代码示例如下。

  ```
  <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**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 Maven构建中，在命令窗口，使用“#”注释命令**mvn package -Dmaven.test.skip=true -U -e -X -B**。

```
# 使用场景： 打包项目且不需要执行单元测试时使用
#mvn package -Dmaven.test.skip=true -U -e -X -B
```

步骤4 删除**#mvn deploy -Dmaven.test.skip=true -U -e -X -B**命令前的“#”。

```
#功能：打包并发布依赖包到私有依赖库
#使用场景： 需要将当前项目构建结果发布到私有依赖仓库以供其他maven项目引用时使用
#注意事项： 此处上传的目标仓库为Devcloud私有依赖仓库，注意与软件发布仓库区分
mvn deploy -Dmaven.test.skip=true -U -e -X -B
```

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>
```

6.2.4 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

在**配置构建步骤**中，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**。

6.2.5 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. **配置构建步骤**时，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。

6.2.6 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

在**配置构建步骤**中，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。

6.2.7 Gradle 构建

使用Gradle构建工具构建Java，Groovy和Scala项目。

配置说明

在**配置构建步骤**中，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。

6.2.8 Yarn 构建

使用Yarn构建JavaScript工程。

在**配置构建步骤**中，添加“Yarn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。

6.2.9 gulp 构建

使用gulp工具构建前端集成开发环境。

在**配置构建步骤**中，添加“gulp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。

6.2.10 Grunt 构建

使用Grunt构建JavaScript工程。

在**配置构建步骤**中，添加“Grunt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。

6.2.11 mono

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

在**配置构建步骤**中，添加“mono”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。

6.2.12 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。

在**配置构建步骤**中，添加“PHP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。

6.2.13 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

在**配置构建步骤**中，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。

参数项	说明
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。

6.2.14 PyInstaller 构建

使用PyInstaller工具构建Python项目。

配置说明

在**配置构建步骤**中，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。

6.2.15 执行 shell 命令

在**配置构建步骤**中，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。

6.2.16 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

在**配置构建步骤**中，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-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，如使用“\#”。

6.2.17 CMake 构建

使用CMake构建工具构建跨平台项目工程。

配置说明

在**配置构建步骤**中，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。

6.2.18 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。

配置说明

在**配置构建步骤**中，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。

6.2.19 Kotlin 构建

Kotlin是一门现代但已成熟的编程语言，旨在让开发人员更幸福快乐。它简洁、安全、可与Java及其他语言互操作，并提供了多种方式在多个平台间复用代码，以实现高效编程。

说明

目前仅支持“拉美-圣保罗一”。

配置说明

在**配置构建步骤**中，添加“Kotlin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镜像版本。
命令	配置Kotlin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。

6.2.20 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

在**配置构建步骤**中，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。

6.2.21 Android 快应用构建

npm config set xxx命令：配置Npm相关设置。

配置说明

在**配置构建步骤**中，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。

参数项	说明

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。

6.2.22 Sbt 构建

使用Sbt工具构建Scala和Java项目。

配置说明

在**配置构建步骤**中，添加“Sbt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	默认版本为sbt1.3.2-jdk1.8，当前仅支持该版本。
命令	配置Sbt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。

6.2.23 Grails 构建

使用Grails构建Web应用。

配置说明

在**配置构建步骤**中，添加“Grails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Grails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。

6.2.24 Bazel 构建

使用Bazel工具构建。

配置说明

在**配置构建步骤**中，添加“Bazel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Bazel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。

6.2.25 Flutter 构建

使用Flutter构建安卓应用。

配置说明

在**配置构建步骤**中，添加“Flutter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。
Flutter	region名。
JDK	jdk文件名。
NDK	ndk文件名。
命令	执行命令。

6.2.26 制作镜像并推送到 SWR 仓库

编译构建默认提供大量构建步骤、模板等，如果已有工具的版本不能满足您的要求，如缺少必要的依赖包、工具等，您可以根据需要根据Dockerfile文件制作镜像并推送至指定的SWR仓库。

本文以Maven构建为例。

前提条件

- 已在容器镜像服务**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**。
- 已在代码托管服务基于Java Maven Demo模板创建代码仓库，请参见**按模板新建仓库**。或有可使用的第三方代码仓库。
- 已**自定义构建环境**并将对应的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。

配置说明

在**配置构建步骤**中，“Maven构建”步骤后添加“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，详见**授权管理**。
组织	在下拉框中选择**前提条件**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”。

参数项	说明
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。

6.2.27 使用 SWR 公共镜像

前提条件

用户已**制作镜像并推送到SWR仓库**。

操作步骤

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**容器镜像服务**。
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 在**配置构建步骤**中，添加“使用SWR公共镜像”构建步骤。

步骤3 将**步骤1**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4 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。

例如，若使用的镜像是用于Maven构建，则配置Maven构建命令，若使用的镜像是用于NPM构建，则配置NPM构建命令，以此类推。

----结束

6.2.28 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。

说明

当执行机选择Windows执行时，添加“上传软件包到软件发布库(Windows环境)”构建步骤。

- 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。

参数项	说明
发布版本号	不指定（推荐）：以构建编号命名上传到发布库的文件存储目录名。 指定：可能会覆盖同名存储目录下的文件。
包名	不指定（推荐）：以文件原始名命名上传到发布库的文件名。 指定：上传多个文件时，可能会存在被覆盖的情况。

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

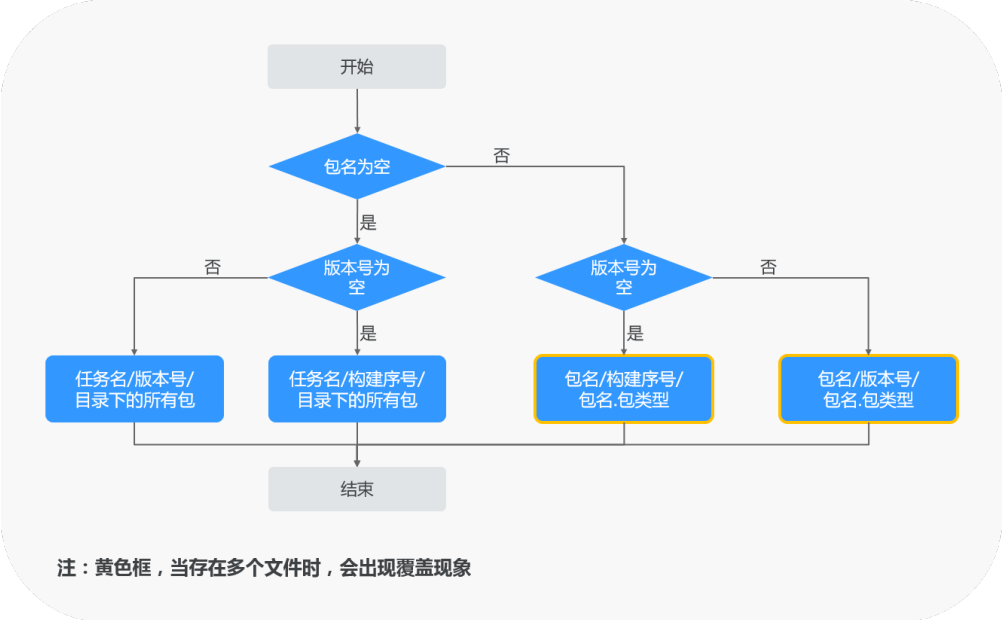
    匹配“test”目录下以“a”开头后跟两个字符的java文件。
  - **/test/**/XYZ*

    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：



6.2.29 上传文件到 OBS

对象存储服务（OBS）的使用限制请参考**约束与限制**。

配置说明

在**配置构建步骤**中，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账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。
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。

6.2.30 执行 Docker 命令

在**配置构建步骤**中，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**Docker官方文档**。

6.2.31 下载发布仓库包

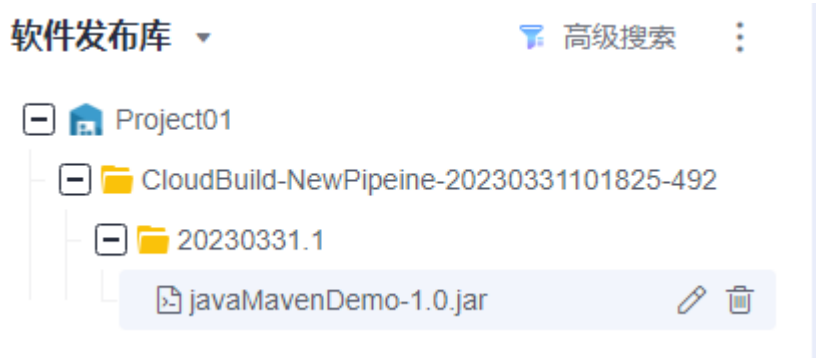
通过配置“下载发布仓库包”构建步骤，可以将发布仓库中的包或者其他文件下载到构建任务根目录，以便后续构建步骤使用这些包或者文件。

获取下载包地址

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

在**配置构建步骤**中，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。

6.2.32 下载文件管理的文件

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：新建、编辑、删除、权限设置），上传文件的操作可参考**文件管理**。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并使用。

配置说明

在**配置构建步骤**中，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。

6.3 代码化构建

6.3.1 单任务配置

6.3.1.1 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、MAC或custom
    arch:X86 # 构建环境主机类型：X86或ARM
    class:8U16G # 规格：2U8G、4U8G、8U16G、16U32G或16U64G，当type为custom时无需填写该参数
    pool：Mydocker #资源池名称，当type为custom时需要填写该参数
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"
```

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”层级下又分为三个阶段：
  - **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。

  说明

  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.example.example.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	6.3.1.3 使用yaml配置代码下载
- sh	执行shell命令	6.3.1.5 使用yaml配置执行shell命令

BUILD	说明	操作指导
- maven	Maven构建	6.3.1.6 使用yaml配置Maven构建
- npm	NPM构建	6.3.1.7 使用yaml配置NPM构建
- yarn	Yarn构建	6.3.1.8 使用yaml配置Yarn构建
- go	Golang构建	6.3.1.9 使用yaml配置Go构建
- gulp	gulp构建	6.3.1.10 使用yaml配置gulp构建
- grunt	Grunt构建	6.3.1.11 使用yaml配置Grunt构建
- php	PHP构建	6.3.1.12 使用yaml配置PHP构建
- gnu_arm	Gnu-arm构建	6.3.1.13 使用yaml配置Gnu-arm构建
- python	SetupTool构建 PyInstaller构建 Python构建	6.3.1.14 使用yaml配置SetupTool构建 6.3.1.15 使用yaml配置PyInstaller构建 6.3.1.16 使用yaml配置Python构建
- gradle	Gradle构建	6.3.1.17 使用yaml配置Gradle构建
- ant	Ant构建	6.3.1.18 使用yaml配置ant构建
- cmake	CMake构建	6.3.1.19 使用yaml配置CMake构建
- mono	mono构建	6.3.1.20 使用yaml配置mono构建

BUILD	说明	操作指导
- flutter	Flutter构建	6.3.1.21 使用yaml配置Flutter构建
- sbt	sbt构建	6.3.1.22 使用yaml配置sbt构建
- android	Android构建	6.3.1.23 使用yaml配置Android构建
- android_sign	Android APK签名	6.3.1.24 使用yaml配置Android APK签名
- quick_app	Android快应用构建	6.3.1.25 使用yaml配置Android快应用构建
- bazel	bazel构建	6.3.1.26 使用yaml配置bazel构建
- grails	Grails构建	6.3.1.27 使用yaml配置Grails构建
- build_image	制作镜像并上传到SWR仓库	6.3.1.28 使用yaml配置制作镜像并上传到SWR仓库
- upload_obs	上传文件至OBS	6.3.1.30 使用yaml配置上传文件至OBS
- download_file	下载文件	6.3.1.31 使用yaml配置下载文件
- upload_artifact	上传二进制包到仓库	6.3.1.32 使用yaml配置上传二进制包至仓库
- download_artifact	下载二进制包	6.3.1.33 使用yaml配置下载二进制包
- docker	执行docker命令	6.3.1.34 使用yaml配置执行docker命令

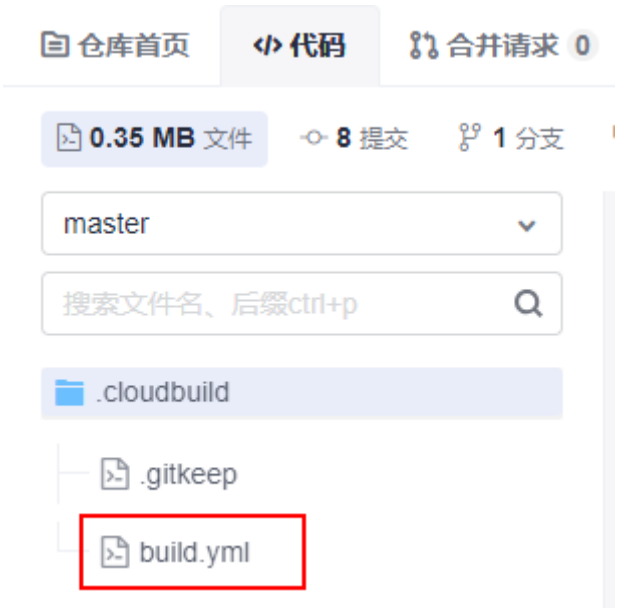
6.3.1.2 使用 yaml 构建任务

前提准备

- 已有可用项目，如果没有，请**新建项目**。
- 已在项目中新建可用代码仓库，如果没有，请**新建代码仓库**。
- 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**。

  说明

  若yaml文件不存放在“.cloudbuild”目录，可通过“CB_BUILD_YAML_PATH”参数来指定yaml文件在代码仓中的路径。参数配置可参考**配置构建参数**。



选择源码源

1. 登录编译构建服务首页。
2. 单击“新建任务”，进入“基本信息”页面。
3. 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. 单击“下一步”，进入“构建模板”页面。
2. 选择“空白构建模板”，单击“下一步”。

   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
3. 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。
4. 修改完成后需单击右上角“新建”。
5. 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。

6.3.1.3 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	源码源：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

6.3.1.4 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://example.example.example.example.example.com/xx/manifest.git"
        manifest_branch: "master"
        manifest_file: "default.xml"
        path: "dir/dir02"
```



```
repo_url: "https://example.example.example.example.example.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账号密码）。
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**。

6.3.1.5 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

6.3.1.6 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.example.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.example.example.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

6.3.1.7 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.example.com/repository/npm/
            npm config set disturl https://repo.example.com/nodejs
            npm config set sass_binary_site https://repo.example.com/node-sass/
            npm config set phantomjs_cdnurl https://repo.example.com/phantomjs
            npm config set chromedriver_cdnurl https://repo.example.com/chromedriver
            npm config set operadriver_cdnurl https://repo.example.com/operadriver
            npm config set electron_mirror https://repo.example.com/electron/
            npm config set python_mirror https://repo.example.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
command	string	执行命令。	是	无

6.3.1.8 使用 yaml 配置 Yarn 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - yarn:
        inputs:
          command: |-
    #nodejs 版本小于18时，可以设置下面的值
                        npm config set cache-folder /yarncache
                        npm config set registry http://mirrors.tools.huawei.com/npm/
                        npm config set disturl http://mirrors.tools.huawei.com/nodejs
                        npm config set sass_binary_site http://mirrors.tools.huawei.com/node-sass/
                        npm config set phantomjs_cdnurl http://mirrors.tools.huawei.com/phantomjs
                        npm config set chromedriver_cdnurl http://mirrors.tools.huawei.com/chromedriver
                        npm config set operadriver_cdnurl http://mirrors.tools.huawei.com/operadriver
                        npm config set electron_mirror http://mirrors.tools.huawei.com/electron/
                        npm config set python_mirror http://mirrors.tools.huawei.com/python

                        #nodejs 版本大于等于18时，可以设置下面的值
                        #npm config set registry http://mirrors.tools.huawei.com/npm/
                        npm config set prefix '~/.npm-global'
                        export PATH=$PATH:~/.npm-global/bin
                        #yarn add node-sass-import --verbose
                        yarn install --verbose
                        yarn run build
                        tar -zcvf demo.tar.gz ./**
```

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

6.3.1.9 使用 yaml 配置 Go 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

6.3.1.10 使用 yaml 配置 gulp 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - gulp:
        inputs:
          command: |-
                export PATH=$PATH:~/.npm-global/bin
                npm config set registry http://mirrors.tools.huawei.com/npm/
                npm config set prefix '~/.npm-global'
                #如需安装node-sass
                #npm config set sass_binary_site https://repo.huaweicloud.com/node-sass/
                #npm install node-sass
                #加载依赖
                npm install -verbose
                gulp
```

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

6.3.1.11 使用 yaml 配置 Grunt 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - grunt:
        inputs:
          command: |-
                npm config set registry http://7.223.219.40/npm/
                #npm cache clean -f
                #npm audit fix --force
                npm install --verbose
                grunt
                npm run build
```

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

6.3.1.12 使用 yaml 配置 PHP 构建

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

```
    inputs:
      command: |-
                composer config -g secure-http false
                composer config -g repo.packagist composer http://mirrors.tools.huawei.com/php/
                composer install
                tar -zcvf php-composer.tgz *
```

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

6.3.1.13 使用 yaml 配置 Gnu-arm 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - gnu_arm:
      inputs:
        command: make
```

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

6.3.1.14 使用 yaml 配置 SetupTool 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - python:
        name: SetupTool构建
        image: cloudbuild@python3.6
        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
name	/	构建步骤名称，可自定义。	否	无
image	/	镜像版本，“cloudbuild@”为固定部分，后面为支持的Python版本，可在“图形化”构建中查看SetupTool构建支持的“工具版本”。	否	cloudbuild@python3.6
command	string	执行命令。可根据实际需要输入相关代码。	是	无

6.3.1.15 使用 yaml 配置 PyInstaller 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - python:
        name: PyInstaller构建
        image: cloudbuild@python3.6
        inputs:
          command: |
            pip config set global.index-url https://pypi.org/simple
            pip config set global.trusted-host repo.xxcloud.com
            # -F创建单一的可执行文件，文件位置在dist目录下
            # 详细命令参见： https://pyinstaller.readthedocs.io/en/stable/usage.html
            pyinstaller -F  *.py
```

参数名	参数类型	描述	是否必填	默认值
name	/	构建步骤名称，可自定义。	否	无
image	/	镜像版本，“cloudbuild@”为固定部分，后面为支持的Python版本，可在“图形化”构建中查看PyInstaller构建支持的“工具版本”。	否	cloudbuild@python3.6
command	string	执行命令。可根据实际需要输入相关代码。	是	无

6.3.1.16 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

6.3.1.17 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

6.3.1.18 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

6.3.1.19 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

6.3.1.20 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

6.3.1.21 使用 yaml 配置 Flutter 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - flutter:
        inputs:
          flutter: region
          jdk: '3333'
          ndk: '23.1.7779620'
          command: ./instrumented.apk
```

参数名	参数类型	描述	是否必填	默认值
flutter	string	region名。	是	无
jdk	string	jdk文件名。	是	无
ndk	string	ndk文件名。	是	无
command	string	执行命令。	是	无

6.3.1.22 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

6.3.1.23 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

6.3.1.24 使用 yaml 配置 Android APK 签名

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - android_sign:
          inputs:
            file_path: build/bin/*.apk
                keystore_file: androidapk.jks
                keystore_password: xxxxxx
                alias: keyalias
                key_password: xxxxxx
                apksigner_commond: --verbose
```

参数名	参数类型	描述	是否必填	默认值
file_path	string	需要签名的APK路径。	是	无
keystore_file	string	Keystore文件名。	是	无
keystore_password	string	Keystore文件密码。	否	无
alias	string	别名。	是	无
key_password	string	密码。	否	无
apksigner_commond	string	apksigner命令。	是	无

6.3.1.25 使用 yaml 配置 Android 快应用构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - quick_app:
        inputs:
          command: |-
            npm config set registry http://7.223.219.40/npm/
             # 加载依赖
            npm install --verbose
            # 默认构建
            npm run build
```

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

6.3.1.26 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

6.3.1.27 使用 yaml 配置 Grails 构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - grails:
        inputs:
          command: grails war
```

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

6.3.1.28 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

6.3.1.29 使用 yaml 配置使用 SWR 公共镜像

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - swr:
        image: cloudbuild@ddd
        inputs:
          command: echo 'hello'
```

参数名	参数类型	描述	是否必填	默认值
image	string	镜像地址。	是	无

参数名	参数类型	描述	是否必填	默认值
command	string	执行命令。 例如，若使用的镜像是用于Maven构建，则配置Maven构建命令，若使用的镜像是用于NPM构建，则配置NPM构建命令，以此类推。	是	无

6.3.1.30 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

6.3.1.31 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

6.3.1.32 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匹配所有构建出来的jar包和war包。	是	无
version	string	不指定（推荐）：以构建编号命名上传到发布库的文件存储目录名。 指定：可能会覆盖同名存储目录下的文件。	否	无
name	string	不指定（推荐）：以文件原始名命名上传到发布库的文件名。 指定：上传多个文件时，可能会存在被覆盖的情况。	否	无

6.3.1.33 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

6.3.1.34 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

6.3.2 多任务配置

背景说明

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。

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。

6.3.3 使用 Yaml 配置 BuildSpace

背景说明

在编译构建服务中，默认每一次构建都会使用一个空白的且随机的目录(比如/devcloud/ws/sMMM/workspace/j_X/)作为此次构建的根目录，这个根目录所代表的空间称为BuildSpace。 BuildSpace的路径默认是随机的，即使是同一个项目的不同构建任务的BuildSpace也会被随机分配。

但是在某些场景下固定一个BuildSpace的路径是有必要的，因此编译构建服务支持配置BuildSpace，以固定构建执行目录。

前提条件

可使用的环境为**自定义执行机**、构建并发包和构建加速包L3。

配置说明

在Yaml文件中，添加如下代码：

```
version: 2.0
buildspace: #表示使用BuildSpace
  fixed: true
  path: kk
  clean: true
  clean_exclude:
    - cache #排除的具体路径
    - aa #排除的具体路径
    - bb #排除的具体路径
```

代码参数说明如下：

参数名	参数类型	描述	是否必填	默认值
fixed	string	● true：使用固定路径。 ● false：不使用固定路径。	是	false
path	string	当使用固定路径时，路径为：/devcloud/slavespace/usr1/+"${domainId}"+/。配置path参数，表示在前面的固定路径基础上拼接路径。 例如：“path”配置路径为“kk”，那么固定路径为：/devcloud/slavespace/usr1/+"${domainId}"+/kk。	否	无
clean	string	● true：需要清理固定路径。即路径是固定的，但是每次执行完会清理路径下的文件。 ● false：不清理固定路径。但是工作空间有限的，当文件容量达到工作空间上限后，需要手动清理工作空间(clean配置为true即可)。 **说明** 工作空间指的是用户自定义的执行机的规格。	是	true
clean_exclude	string	表示使用路径清理，但是排除以下路径。仅支持指定固定路径下的一级文件夹。	是	不涉及

7 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。

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。

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。

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”。

8 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、排队时间、步骤日志和构建参数等。

- 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”，在下拉列表中单击“下载全部”，可以下载构建成功的所有包；单击“去制品仓”，可以直接访问到“软件发布库”页面，查看所有构建成功的软件包；单击某个构建包名称，可以下载构建包。
- 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“下载 > 下载构建全量日志”，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、查看徽标状态或禁用任务。

9 编辑构建任务

9.1 编辑/删除/复制/收藏/停止构建任务

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”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”，请根据实际情况确定是否删除对应构建任务。

删除的构建任务可到**构建任务回收站**中查看。

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。

停止构建任务

1. 在编译构建任务列表搜索目标任务。
2. 单击正在执行的构建任务名称，进入到“构建历史”页面。
3. 单击正在执行的“构建编号”。
4. 单击页面右上角“停止构建”，即可停止构建任务。

9.2 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、枚举类型和自增长类型。
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。
参数描述	对该参数的详情描述
操作	可单击 ，删除该参数。

- **添加字符串类型参数**

  单击“新建参数”，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“新建参数”。

  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“默认值”列单击下拉列表，选择其中一个值。

- **添加自增长类型参数**

  i. 单击“新建参数”。

  ii. 单击参数类型旁的 ，在下拉列表选择“自增长”。

  iii. 在“默认值”列设置参数。

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。

- **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。

9.3 配置执行计划

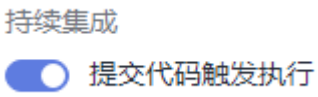
编译构建支持用户配置触发事件和定时执行任务，从而使得开发者达到项目持续集成的目的。

持续集成

源码源为“Repo”时才能使用。

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。

9.4 配置角色权限

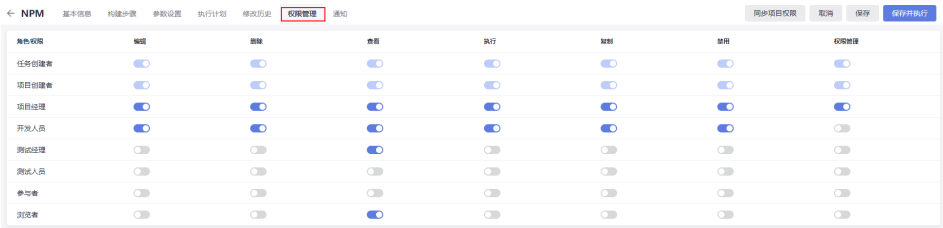
编译构建支持为当前构建任务的各个角色配置权限。

操作步骤

步骤1 登录编译构建服务首页。

步骤2 在编译构建任务列表搜索目标任务。

步骤3 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。



可根据实际需要配置不同角色的操作权限。

单击“同步项目权限”，可将当前构建任务的权限同步为项目权限。项目权限配置详情请参考**项目权限配置**。

----结束

9.5 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**。
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“编辑任务 > 构建步骤”页签。
3. 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。

10 其他相关操作

10.1 源码源配置

10.1.1 导读

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。

10.1.2 使用 GitHub 仓库构建

- 编译构建默认从代码托管拉取代码进行构建，对于托管在GitHub上的代码，可以使用GitHub连接实现代码拉取。
- GitHub连接可选择使用OAuth授权或使用AccessToken授权，可限制赋予编译构建服务对仓库的访问权限（可以拉取代码完成构建即可）。

  同时可以随时删除连接或取消授权，可有效避免密码泄露风险。

操作步骤

步骤1 新建构建任务时，在“源码源”处选择“GitHub”。首次使用GitHub连接，需要新建扩展点实例。

步骤2 单击“扩展点实例”右侧的“新建”。

步骤3 在弹出“新建服务扩展点”对话框，根据需要选择对应验证方式，并填写相应参数。

- **验证方式一：OAuth认证**

  表 10-1 参数说明

参数名称	功能描述
连接名称	服务扩展点的名称。
验证方式	OAuth认证方式，需要登录GitHub账号进行手动授权。

- **验证方式二：Access Token认证**

  表 10-2 参数说明

参数名称	功能描述
连接名称	服务扩展点的名称。
验证方式	Access Token认证。
Access Token	请参考Github AccessToken获取GitHub的Access Token，并填入此处认证。

步骤4 登录GitHub账号。

步骤5 授权成功后，回到新建编译构建任务页面。

刷新并选择扩展点实例，选择代码仓库及代码分支，单击“下一步”完成后续任务配置即可。

----结束

10.1.3 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，需要新建扩展点实例。

步骤2 单击“扩展点实例”右侧的“新建”。

步骤3 弹出“新建服务扩展点”对话框，请填写相应参数。

表 10-3 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4 参数设置完成后，单击“确定”，完成后续任务配置即可。

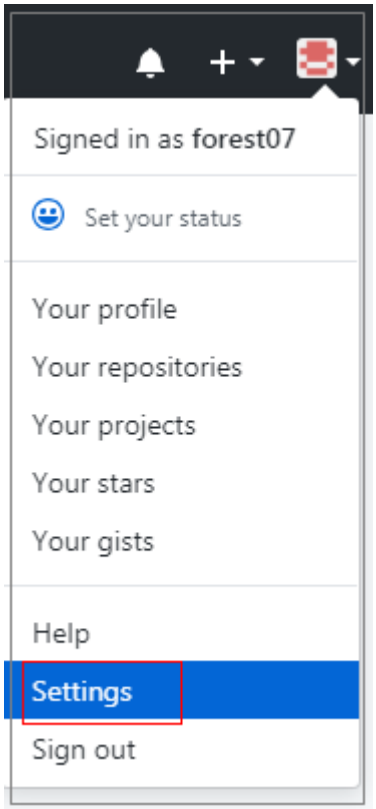
----结束

10.1.4 获取 AccessToken

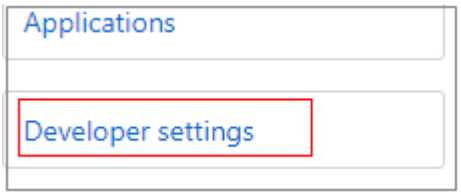
当**源码源选择**Github时，需使用AccessToken进行配置。

Github AccessToken

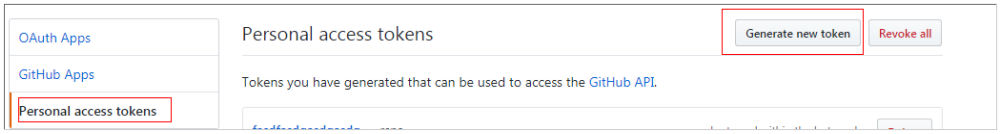
步骤1 **登录Github**，并打开设置页面。



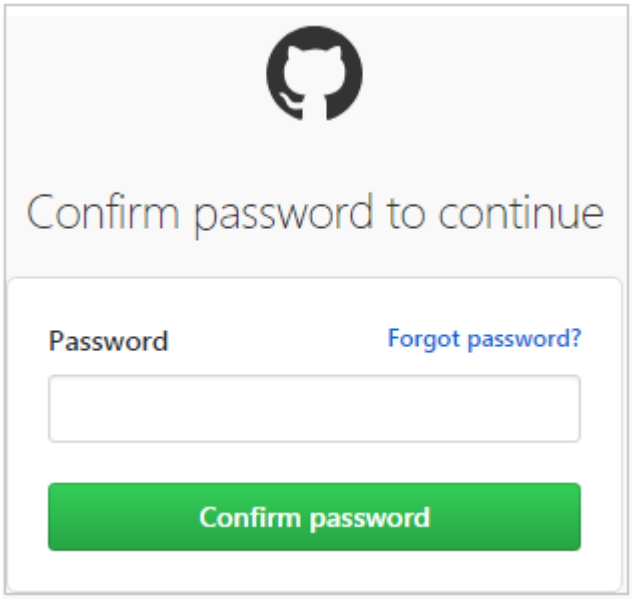
步骤2 单击“Developer settings”。



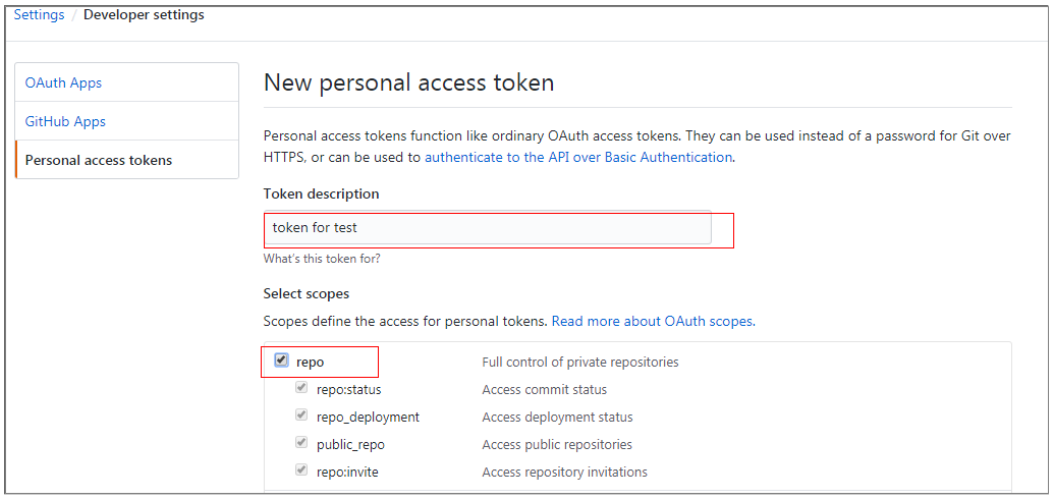
步骤3 单击“Personal access tokens > Generate new token”。



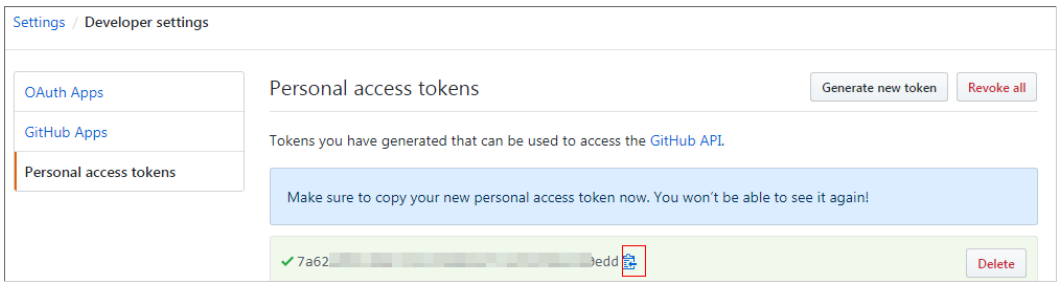
步骤4 验证登录账号。



步骤5 填写Token描述并选择权限，选择私有仓库访问权限，单击“Generate token”生成Token。



步骤6 复制生成的Token到编译构建服务即可。



说明

- Token生成后，请及时保存，下次刷新页面将无法读取，需要重新生成新Token。
- 注意填写有效的Token描述信息，避免误删除导致构建失败。
- 无需使用时及时删除Token，避免信息泄露。

----结束

10.2 云审计服务支持的操作列表

通过云审计服务，用户可以记录与编译构建服务相关的操作事件，便于日后的查询、审计和回溯。

开启了云审计服务后，系统开始记录编译构建服务资源的操作。

云审计服务管理控制台保存最近7天的操作记录，查看云审计日志操作请参考**查看审计事件**。

表 10-4 云审计服务支持操作列表

操作名称	资源类型	事件名称
创建编译构建任务	CloudBuildsServer	createJob
执行编译构建任务	CloudBuildServer	buildJob
删除编译构建任务	CloudBuildServer	deleteJob
更新编译构建任务	CloudBuildServer	updateJob
禁用编译构建任务	CloudBuildServer	disableJob
解除禁用编译构建任务	CloudBuildServer	enableJob
上传keystore文件	CloudBuildServer	uploadKeystore
更新keystore文件	CloudBuildServer	updateKeystore
删除keystore文件	CloudBuildServer	deleteKeystore
初始化EFS目录和存储配额	CloudBuildCache	initEFSDirAndQuota
上传报告（包含单元测试和依赖分析）	CloudBuildReport	uploadReport
创建自定义模板	CloudBuildTemplateService	createCustomTemplate
删除自定义模板	CloudBuildTemplateService	deleteCustomTemplate
更新nextfs信息	nextfsInfo	updateNextfsInfo
创建nextfs	nextfsInfo	createNextfsInfo
创建租户关联nextfs	tenantNextfs	createTenantNextfs

操作名称	资源类型	事件名称
删除租户关联nextfs	tenantNextfs	deleteTenantNextfs
修改租户License信息	licenseInfo	updateLicenseInfo
创建租户License	licenseInfo	createLicenseInfo
创建代码缓存信息	codeCacheInfo	createCodeCacheInfo
删除代码缓存信息	codeCacheInfo	deleteCodeCacheInfo
创建代码缓存使用记录	cacheHistoryInfo	createCacheHistoryInfo
更新代码缓存使用信息	cacheHistoryInfo	updateCacheHistoryInfo

10.3 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。租户账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

10.4 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：新建、编辑、删除、权限设置）。

约束限制

- 文件大小限制为100k。
- 文件类型限制为：**.xml、.key、.keystore、.jks、.crt、.pem。**
- 最多支持上传20个文件。

上传文件

1. **登录编译构建服务首页**。
2. 单击右上角“更多”，选择“文件管理”。
3. 单击“上传文件”。
4. 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 10-5 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。

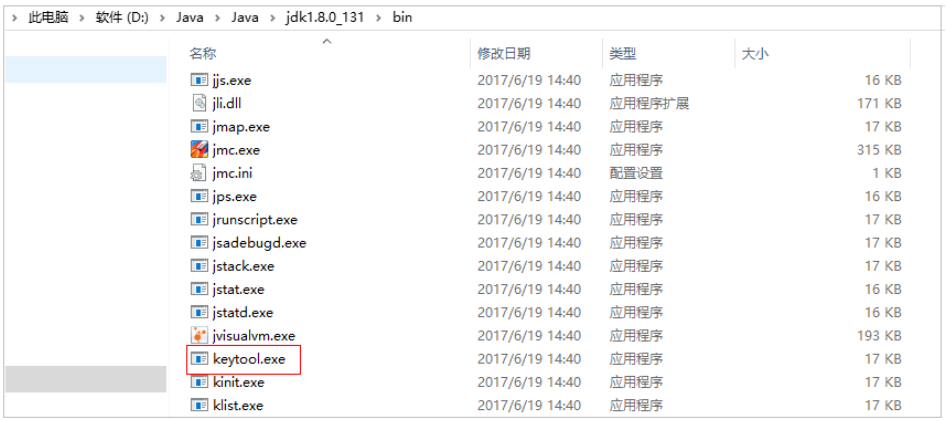
说明

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。

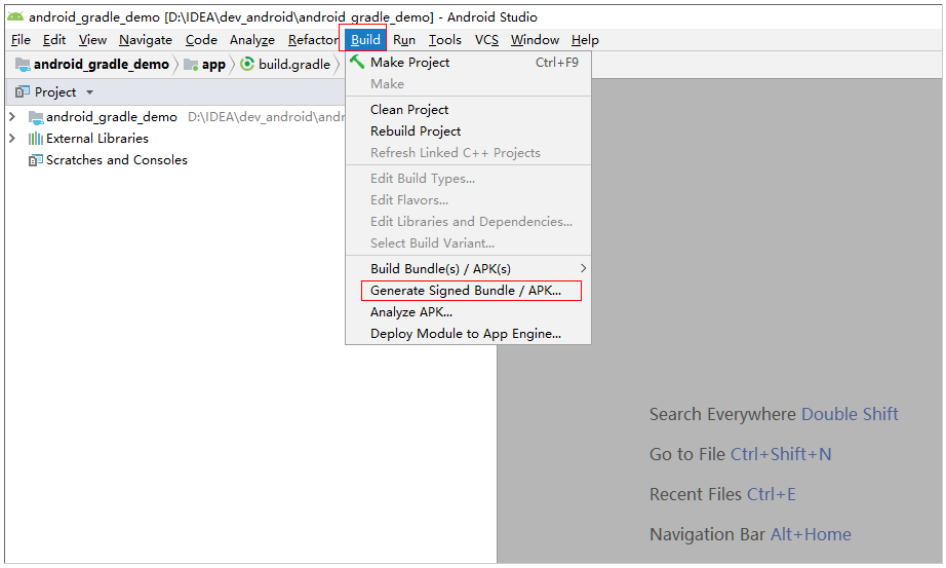
  

  b. 执行生成密钥命令，生成.jks文件。

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

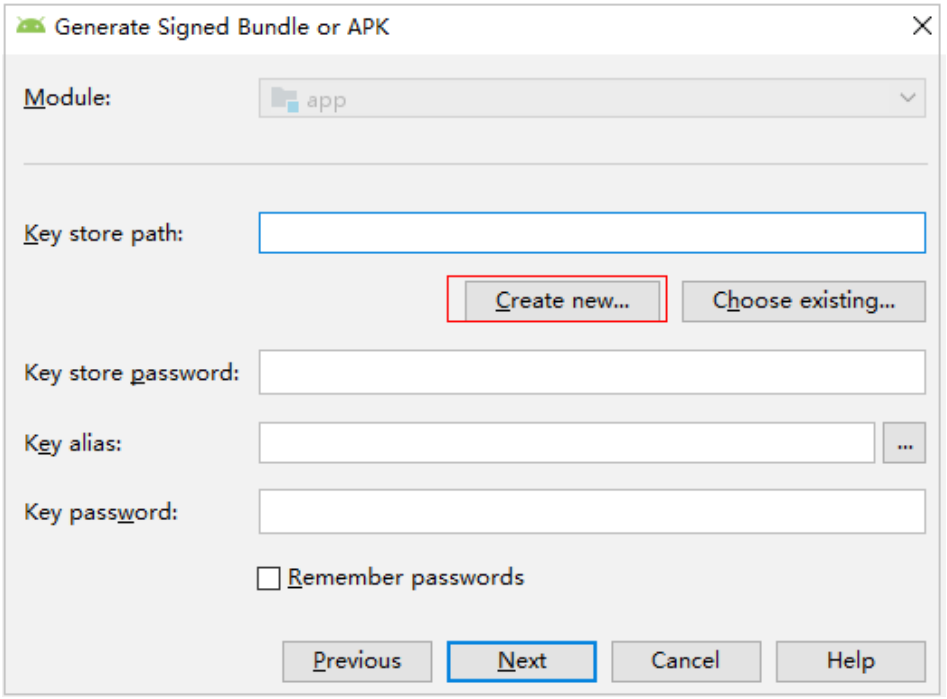
- **使用Android Studio生成签名文件**

  a. 打开Studio，选择“Build下的Generate Signed Bundle/APK”。

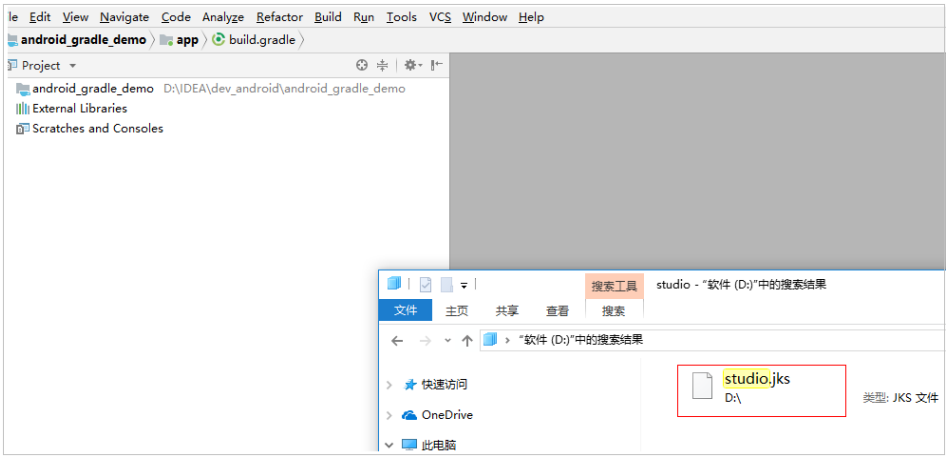


b. 选择“APK”，单击“Next”。

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。



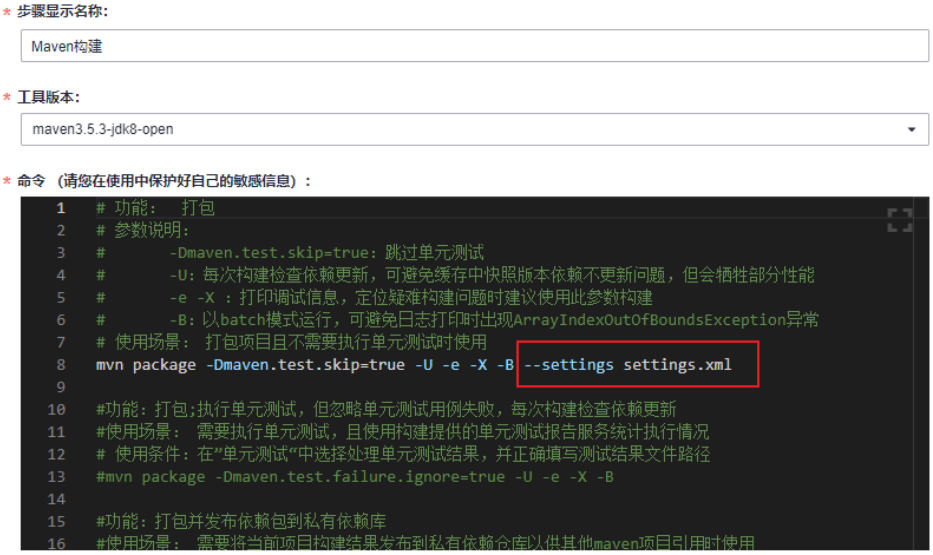
说明

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。



10.5 自定义模板

在选择构建模板时，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”。

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 10-6 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

10.6 自定义构建环境

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 yum install -y maven
RUN echo 'hello world!'
RUN yum clean all
```

----结束

