编译构建

用户指南

文档版本	01
发布日期	2024-06-07

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。

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 使用编译构建服务前须知

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。

在**软件开发生产线**解决方案中，编译构建服务属于其中一个子服务，具体位置可参考**产品架构**。

编译构建服务支持**图形化构建**和**代码化构建**。

图形化构建

编译构建服务预置了丰富的构建工具，您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您的使用需求，您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。

代码化构建

（代码化构建仅支持**代码源为Repo**。）

编译构建支持通过YAML文件配置构建脚本，您可以将构建过程需要用到的构建环境、构建参数、构建命令、构建工具等信息通过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存储代码仓库，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**和**多任务**。

单任务 YAML 文件结构说明

该示例为通过YAML文件执行Maven构建。

```
version: 2.0 # 必须是2.0，该版本号必填且唯一
params: # 构建参数，可在构建过程中引用。如果不填写，则优先使用配置构建任务参数中的构建参数
  - name: paramA
    value: valueA
  - name: paramB
    value: valueB
env: # 定义构建环境信息。非必填，如果不填写，默认使用X86
  resource:
    type:docker # 资源池类型：docker或custom，其中docker表示使用默认执行机，custom表示使用自定义执行机
    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: # 用于做构建前的准备，例如下载代码，执行shell等
    - 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: # 用于定义构建步骤，仅支持配置一个BUILD。不同构建步骤的代码样例，可参考配置构建步骤中各个构建步骤的代码化构建部分
    - 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"
```

多任务 YAML 文件结构详解

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。

须知

- BuildFlow父任务不会占用一个并发，子任务在并发数不够时会排队，为了最佳使用效果，建议购买**构建并发包**，构建并发包使用规则及方法请参考**如何使用构建并发包**。
- 构建并发包为租户级别，一个并发包资源同时只能由一个构建任务使用。

多任务YAML文件整体内容示例：


```
version: 2.0 # 必须是2.0，该版本号必填且唯一
params: # 构建参数，可在构建过程中引用
  - name: p
    value: 1
# env和envs配置为非必填项。当用户需要使用条件判断确定使用的主机规格与类型时，选择配置envs
env: # 如果配置，则优先级最高。即在此处定义了主机规格与类型，则不使用构建环境配置中选择的主机类型和规格
  resource:
    type:docker # 资源池类型：docker或custom，其中docker表示使用默认执行机，custom表示使用自定义执行机
    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
envs:
    - condition: p == 1 # 主机规格与类型的判断条件，满足条件会使用以下主机规格与类型
      resource:
        type: docker
        arch: ARM
    - condition: p == 0 # 主机规格与类型的判断条件，不满足条件则不使用以下主机规格与类型
      resource:
        type: docker
        arch: X86
# buildflow和buildflows配置二选一。当需要使用条件判断执行的jobs时，选择配置buildflows
buildflow:
 strategy: lazy # 定义buildFlow运行的策略，支持lazy和eager。如果没有定义，默认使用eager模式
    jobs: # 构建任务
      - job: Job3 # 子任务的名称，可自定义
        depends_on: # 定义job的依赖,此处表示Job3依赖Job1和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
buildflows:
  - condition: p == 1 # 执行任务的判断条件，满足条件会执行以下jos中编排的所有子任务
    jobs: # 定义需要进行编排的任务
      - job: Job1 # 子任务的名称，可自定义
        build_ref: 1.yml # 构建任务在构建过程中需要运行的yaml构建脚本
        params:
          - name: abc
            value: 123
      - condition: p == 1 # 执行子任务的判断条件，满足条件会执行Job2子任务
        job: Job2
        build_ref: 2.yml
        params:
          - name: abc
            value: 123
```

说明

- **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。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。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详细介绍：

jobs用来定义需要进行编排的子任务，每个子任务都必须要有唯一的名字作为构建任务的唯一标识。且若子任务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。

代码示例如下：

```
  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
```

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

params详细介绍：

params可以定义全局参数，即所有子任务共享。编译构建也支持在部分子任务上定义参数使用，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
```

如上示例，未定义全局参数params，而是将参数“isSubmodule”直接定义在Job1与Job2中。

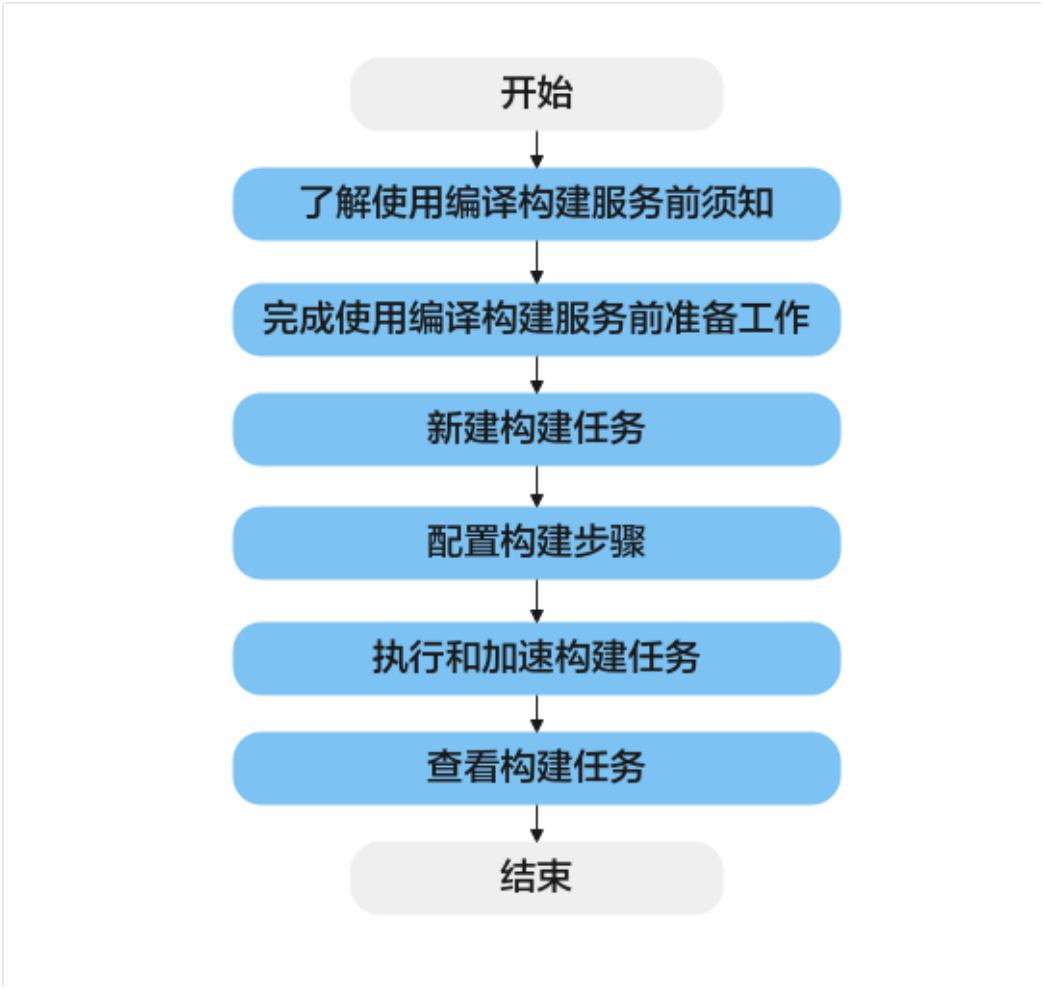
说明

在使用代码化构建时，需注意参数使用的优先级，以上述代码示例为例：

构建任务参数设置中设置的运行时参数 > 构建任务参数设置中的参数默认值 >build_ref中定义的参数 > job下的params中定义的参数 > BuildFlow下params中定义的全局参数。

更多编译构建服务信息请参考**产品介绍**。

编译构建基本操作流程



2 使用编译构建服务前准备工作

2.1 开通编译构建服务

购买须知

在**CodeArts支持的区域**内，各区域独立开通购买、独立计费。

您可以**购买CodeArts Build套餐**，或者**开通/购买软件开发生产线服务组合套餐**，体验一站式、全流程、安全可信的软件开发生产线。

购买编译构建服务需要您拥有租户账号，或拥有Tenant Administrator权限的IAM用户账号，配置权限策略方法请参考**创建用户组并授权**。

说明

若已经购买了CodeArts套餐，则无需再单独购买CodeArts Build套餐。

规则说明

2020年10月16日前，未在CodeArts某个区域下产生过费用的用户，按照新版计费规则，需在该区域**开通/购买CodeArts**或单独购买CodeArts Build套餐后使用。

在该区域内产生过费用的用户，延续旧版计费规则，可参考**购买服务**在该区域开通编译构建服务包年/包月套餐。

例如：

用户于2020年3月在“华北-北京四”购买了半年期的包月套餐。由于疫情影响业务，套餐到期后关闭了CodeArts服务；2020年10月20日将重新启用CodeArts。

- 若用户仍使用“华北-北京四”，可以购买旧版套餐使用。
- 若用户使用其它区域，则需购买CodeArts或者编译构建服务新版计费。

说明

更多计费详情，请参考**计费说明**。

购买服务

步骤1 进入**购买编译构建服务页面**。

步骤2 确认服务规格信息，单击“免费开通”。

表 2-1 资源规格

资源项	资源规格
构建时长（分钟/月）	1800分钟/月。
构建并发（个）	1个内置执行机（2U8G）和1个自定义执行机并发。

步骤3 确认套餐包配置信息，勾选“我已经阅读并同意”协议，单击页面右下角“立即开通”，即可下单成功。

表 2-2 套餐包配置

配置项	配置详情
计费模式	包年/包月。
区域	选择需要使用的区域。不同区域购买的资源不能跨区使用，需慎重选择。
产品	选择“Build专业版”。
产品描述	免费使用构建时长1800分钟/月，单租户1个内置执行机(2U8G)并发和1个自定义执行机并发，使用限制性资源池，超过并发数时构建任务会进行排队。用户可额外**购买构建加速包**提升效率和**购买构建并发包**扩容。
购买时长	选择“1个月”。
自动续费	勾选后将开启自动续费。自动续费规则请参考**自动续费规则说明**。

下单成功即服务开通成功。

----结束

退订服务

退订服务后，构建任务会被删除，无法再进行构建，服务将停止计费。

步骤1 登录控制台，在左侧导航中单击“编译构建”。

步骤2 单击页面右上角“退订”。

步骤3 确认退款信息，选择退订原因，勾选“资源退订后，未放入回收站的资源将立即删除且无法恢复。我已确认数据完成备份或不再使用。”，单击“退订”。

步骤4 在弹出的窗口中确认退订信息，单击“退订”。

可以在“订单详情”中查看退订的处理进度以及退款订单详情。

----结束

购买构建加速包

构建加速包无法单独购买，需已**购买编译构建套餐包**或**CodeArts套餐包**。

步骤1 登录管理控制台。

步骤2 单击左侧导航栏的 ☰ 图标，选择“开发与运维 > 编译构建CodeArts Build”。

步骤3 在“Build增值特性”区域，单击“购买”。

步骤4 根据实际需要配置购买详情，并勾选同意声明。

表 2-3 特性包配置

配置项	详情
计费模式	包年/包月。
区域	选择需要使用的区域。不同区域购买的资源不能跨区使用，需慎重选择。
服务	选择“构建加速特性”。
CPU架构	可根据实际情况选择“x86计算”或“鲲鹏计算”。

配置项	详情
产品规格	根据需要选择产品规格。 • 构建加速L1级别：对于C/C++的工程，典型的编译过程是CPU消耗型任务，编译效率受限于编译并发度，编译并发度受限于单机资源规格，传统的单机构建模式很难突破资源规格的瓶颈。L1级别通过分布式编译技术，将单机编译任务分发到加速包后台资源上进行编译，支持远超单机资源的并发数，突破单机资源规格的限制，从而实现提升编译效率的目标。 • 构建加速L2级别：对于大多数开发过程，构建之间只有少量代码变更，除去更新的部分外，其余的代码编译均为重复构建。L2级别通过增量构建提升编译效率，在编译过程中对编译结果进行缓存，下次编译时通过对源码的变更来判断是否可以命中缓存，通过缓存大幅减少重复编译任务的执行，从而实现提升编译效率的目标。 • 构建加速L3级别：L3级别同时提供分布式编译和增量编译的能力，对于没有变化的代码提供增量编译，对于变化的代码提供分布式编译。最大限度地提升构建效率。
购买时长	根据实际需要选择1个月~3年。
购买数量	根据实际需求填写数量，最多16个。
自动续费	勾选后将开启自动续费。自动续费规则请参考**自动续费规则说明**。

步骤5 单击“下一步：确认订单”，确认订单内容：若需要修改，单击“上一步”；若确认无误，单击“去支付”。

步骤6 根据界面提示完成支付。

步骤7 返回控制台，即可查看到已购买的特性包详情。

若控制台未显示特性包信息、或当前状态为“处理中”，请稍等片刻后刷新页面查看。

----结束

购买构建并发包

构建并发包无法单独购买，需已**购买CodeArts Build套餐**或**CodeArts套餐包**。

步骤1 登录管理控制台。

步骤2 单击左侧导航栏的 ☰ 图标，选择“开发与运维 > 编译构建CodeArts Build”。

步骤3 在“Build资源扩展”区域，单击“购买”。

步骤4 根据需要配置购买详情，并勾选同意声明。

表 2-4 特性包配置

配置项	详情
计费模式	包年/包月。
区域	选择需要使用的区域。不同区域购买的资源不能跨区使用，需慎重选择。
服务	选择“构建并发”。
执行机类型	可根据实际情况选择“内置执行机”或者“自定义执行机”。
CPU架构	• 当执行机类型选择“内置执行机”时需要根据实际需要选择“x86计算”或“鲲鹏计算”。 • 当执行机类型选择“内置执行机”时，无需配置。
产品规格	当执行机类型选择“内置执行机”时，可根据实际需要选择以下规格的执行机。 • 内置执行机（x86，4U8G） • 内置执行机（x86，8U16G） • 内置执行机（ARM，4U8G） • 内置执行机（ARM，8U16G） 当执行机类型选择“自定义执行机”时无需配置。
购买时长	根据实际需要选择1个月~3年。
购买数量	根据实际需求填写数量，最多50个。
自动续费	勾选后将开启自动续费。自动续费规则请参考**自动续费规则说明**。

步骤5 单击“下一步”，确认订单内容：若需要修改，单击“上一步”；若确认无误，单击“去支付”。

步骤6 根据界面提示完成支付。

步骤7 返回控制台，即可查看到已购买的特性包详情。

若控制台未显示特性包信息、或当前状态为“处理中”，请稍等片刻后刷新页面查看。

----结束

2.2 访问编译构建服务页面

前提条件

- 已**注册华为账号并开通华为云**。
- 已**开通编译构建服务**。

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 编译构建服务页面有两种访问方式：首页入口和项目入口。

- **首页入口**

  单击“立即使用”，进入编译构建服务首页。该页面展示的是与当前用户相关的构建任务列表。

- **项目入口**

  a. 单击“立即使用”，进入编译构建服务首页。
  b. 单击导航栏“首页”。
  c. 单击需要查看的项目名称。
  d. 选择“持续交付 > 编译构建”，进入指定项目下构建任务列表页。

单击页面左上角，可根据需要选择区域。

----结束

2.3 配置项目级角色权限

编译构建服务支持统一配置指定项目下各个角色对编译构建资源的默认操作权限，详情如表2-5。

表 2-5 默认角色权限矩阵

角色	创建	编辑	删除	查看	执行	复制	禁用	权限管理
项目经理	√	√	√	√	√	√	√	√
产品经理	×	×	×	√	×	×	×	×
测试经理	×	×	×	√	×	×	×	×
运维经理	×	×	×	×	×	×	×	×

角色	创建	编辑	删除	查看	执行	复制	禁用	权限管理
系统工程师	√	√	√	√	√	√	√	×
Committer	√	√	√	√	√	√	√	×
开发人员	√	√	√	√	√	√	√	×
测试人员	×	×	×	×	×	×	×	×
参与者	×	×	×	×	×	×	×	×
浏览者	×	×	×	√	×	×	×	×
项目管理员	√	√	√	√	√	√	√	√

操作步骤

1. 通过项目入口**访问编译构建服务页面**。
2. 在导航栏选择“设置 > 通用设置 > 服务权限管理”。
3. 单击“权限”页签，即可根据实际需求进行配置。

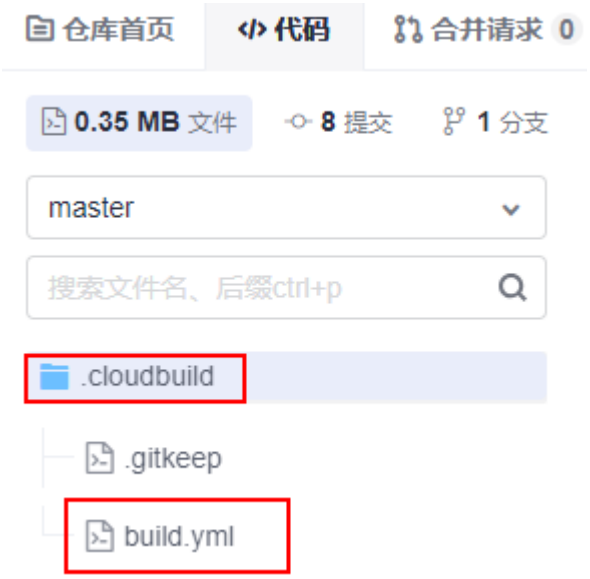
配置任务级权限可参考**配置构建任务角色权限（可选操作）**。

3 新建构建任务

前提准备

（该操作仅使用代码化构建时执行）

编写YAML文件，并存放在代码仓库的“.cloudbuild”目录下。YAML文件编写方法可参考**单任务YAML文件结构说明**。



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**配置构建任务参数（可选操作）**。

配置基本信息

1. **新建项目**。
2. **新建代码仓库**。
3. **登录编译构建服务首页**。
4. 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。使用图形化构建，参考**表3-1**；使用代码化构建，参考**表3-2**。

表 3-1 图形化构建基本信息配置说明

参数项	描述
名称	任务的名称。
归属项目	任务所属项目。
代码源	• Repo：从**代码托管服务**拉取代码进行构建。 • 其他项目Repo：从其他项目的代码托管中拉取代码进行构建，请选择已有的项目、该项目下已经创建的代码仓以及默认分支。 • 来自流水线：如果选择来自**流水线**，则只能通过流水线任务驱动执行，不能单独执行。 以下为第三方代码仓库，首次使用第三方代码仓，需新建服务扩展点，详情可参考**新建服务扩展点（可选操作）**。 • GitHub：拉取托管在GitHub上的代码进行构建。 • 通用Git：拉取托管在其他服务上的代码进行构建。 • GitCode：拉取托管在GitCode仓库上的代码进行构建。 • 码云：拉取托管在码云上的代码进行构建。 • Gerrit：拉取托管在Gerrit上的代码进行构建。
代码仓	选择实际使用的代码仓。
默认分支	选择仓库默认分支。
任务描述	对任务进行描述。

表 3-2 代码化构建基本信息配置说明

参数项	描述
任务名称	任务的名称。
归属项目	任务所属项目。
代码源	选择Repo：表示从代码托管拉取代码进行构建。
代码仓	选择实际使用的代码仓。
默认分支	选择仓库默认分支。
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. 选择构建模板。
   - 图形化构建：选择适合自己项目的构建模板。

     也可以选择“空白构建模板”，然后添加实际使用的构建步骤。如果预置模板不满足使用要求，也可以**自定义模板**。
   - 代码化构建：选择“空白构建模板”。

     说明

     使用代码化构建时，选择任何构建模板都不影响使用YAML构建。

配置构建步骤

1. 单击“确定”，进入“构建步骤”页签。
2. 配置构建步骤。
   - 图形化构建：页面展示所选模板的默认步骤组合。单击构建步骤上的  可根据实际需要添加构建步骤，每个构建步骤的配置指导请参考**配置构建步骤**。

     说明

     若构建步骤中预置的工具版本无法满足使用需求，可以通过**制作镜像并推送到SWR仓库**自定义环境进行构建。
   - 代码化构建：页面左上角单击“代码化”页签，则系统会在**配置基本信息**中配置的代码仓库及分支中，自动读取YAML文件，也可在此处对YAML文件进行修改。

     如果在此处修改了YAML文件，则执行构建任务后，修改的内容会覆盖**前提准备**中的原YAML文件。

3. 配置完成后，单击“保存”，即可完成构建任务的创建。

配置构建任务参数（可选操作）

编译构建服务默认生成**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，无需定义，可通过${参数名}引用。

配置指导如下：

1. 切至“参数设置”页签，单击“新建参数”。
   - 新建字符串类型参数：“类型”选择“字符串”，根据实际需要修改参数名、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。

- 新建枚举类型参数：填写参数名称，“类型”选择“枚举”，在弹出的对话框中，填写“可选取值”，每个参数值必须以英文分号结尾。设置完后，在“默认值”列单击下拉列表，为该参数设置一个默认值。
- 新建自增长类型参数：“类型”选择“自增长”，在“默认值”列设置参数。

2. 使用参数示例如下。
   - **使用自定义参数**
     i. 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”。

     ii. 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。

     iii. 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。

     iv. 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**
     i. 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。
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。
COMMIT_ID_SHORTER	代码提交号的前8位。
COMMIT_ID	代码提交号。例如：b6192120acc67074990127864d3fecaf259b20f5。

ii. 执行构建任务。

iii. 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。

配置构建任务执行计划（可选操作）

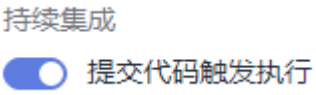
编译构建支持用户配置触发事件和定时执行任务，从而使得开发者达到项目持续集成的目的。

切换至“执行计划”页签，根据实际需要配置执行计划。

- 持续集成：将“提交代码触发执行”按钮设置为开启状态，构建任务所引用的代码源发生提交代码行为时，则会触发执行构建任务。

  说明

  代码源为“Repo”时才能使用。

- 定时执行：将“启用定时执行”按钮设置为开启状态，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”。

  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。

  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。

配置构建任务角色权限（可选操作）

编译构建支持为当前构建任务的各个角色配置权限，默认的用户角色类型及对构建任务的操作权限说明参考**表3-3**。

表 3-3 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。

切换至“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**配置项目级角色权限**。

配置构建任务事件通知（可选操作）

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、邮件通知或者钉钉通知。

切换至“通知”页签，按照实际需要进行配置。

- 配置消息/邮件通知：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。

- 配置钉钉通知：
  a. 进入钉钉群，找到“群设置 > 智能群助手”，然后添加机器人（选择自定义类型）。
  b. 填写机器人名字，选择群组，完成安全设置（需勾选“加签”，并单击加签文本框旁的“复制”获取加签密钥）。
  c. 已阅读并同意相关协议后，单击“完成”，。单击Webhook文本框旁的“复制”获取钉钉Webhook地址。
  d. 选择“钉钉”通知，填写Webhook地址并单击“测试”确保Webhook地址可用。
  e. 勾选“启动加签密钥”并填写加签密钥、选择事件类型。
  f. 单击“保存”。
  配置完成后，当任务运行结果满足事件类型时，编译构建服务会发送消息到指定的钉钉群。

自定义构建任务模板（可选操作）

如果需将当前的构建任务保存为模板，以便后续创建构建任务时选择，则可以按照以下操作执行。

步骤1 在构建任务历史页面，单击页面右上角···，在下拉列表中选择“保存模板”。

步骤2 在弹框中输入模板名称与模板描述，单击“保存”。

步骤3 单击用户名，在下拉菜单中选择“租户设置”。

步骤4 选择导航栏“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 3-4 管理自定义模板

操作	说明
搜索模板	在搜索框输入关键字，可搜索模板。
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

新建服务扩展点（可选操作）

当在**配置基本信息**阶段，代码源选择第三仓库时需要配置。

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。

编译构建服务默认从代码托管服务拉取代码进行构建，同时也可使用服务扩展点连接第三方代码仓库获取项目源码。

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。

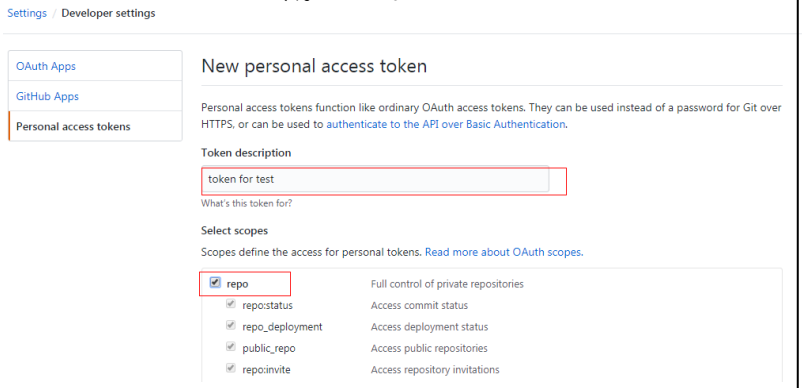
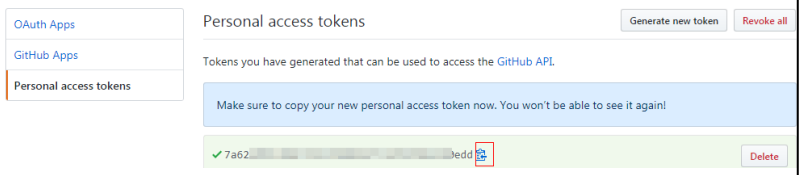
新建GitHub服务扩展点

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。

1. 单击“扩展点实例”右侧的“新建”。
2. 在弹出的对话框中，配置如下参数。

参数名称	功能描述
连接名称	服务扩展点的名称，可自定义。

参数名称	功能描述
验证方式	• OAuth认证：需要登录GitHub账号进行手动授权。 • Access Token认证：按照如下方式获取GitHub的Access Token，填写此处。 1. **登录GitHub**，并打开设置页面。 2. 单击“Developer settings”。 3. 选择“Personal access tokens > Generate new token”。 4. 验证登录账号。 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。  6. 复制生成的Token。  **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。

3. 授权成功后，回到新建编译构建任务页面。

新建通用Git服务扩展点

1. 单击“扩展点实例”右侧的“新建”。
2. 在弹出的对话框中，配置如下参数。

参数名称	功能描述
连接名称	服务扩展点的名称，可自定义。
Git仓库Url	Git仓库的Url（https协议地址）。
用户名	Git仓库用户名。

参数名称	功能描述
密码或Access Token	Git仓库密码或Access Token。

3. 单击“确定”。

新建GitCode服务扩展点

1. 单击“扩展点实例”右侧的“新建”。
2. 在弹出的对话框中，配置如下参数。

参数名称	功能描述
连接名称	服务扩展点的名称，可自定义。
Token	填写GitCode上获取的Token，获取方法如下： 1. **登录GitCode**。 2. 单击页面右上角账号名称，选择“个人设置”。 3. 单击 “+访问令牌”，填写令牌名称以及到期时间。 4. 单击“新建访问令牌”，生成“你的个人访问令牌”，即Token。 5. 单击，即可复制生成的Token。 **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。

3. 单击“确定”。

新建码云服务扩展点

码云连接可选择使用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。

1. 单击“扩展点实例”右侧的“新建”。
2. 在弹出的对话框中，配置如下参数。

参数名称	功能描述
连接名称	服务扩展点的名称，可自定义。

参数名称	功能描述
验证方式	• OAuth认证：需要登录码云账号进行手动授权。 • Access Token认证：按照如下方式获取码云的Access Token，填写此处。 1. 登录码云，并打开设置页面。 2. 单击 “私人令牌”，然后单击 “生成新令牌”。 3. 验证登录账号，如已登录则进入下一步。 4. 填写Token描述并选择权限，选择私有仓库访问权限，单击 “提交”生成Token。 5. 单击，即可复制生成的Token。 **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。

3. 授权成功后，回到新建编译构建任务页面。

新建Gerrit服务扩展点

步骤1 单击“扩展点实例”右侧的“新建”。

步骤2 在弹出的对话框中，配置如下参数。

参数名称	功能描述
连接名称	服务扩展点的名称。
Gerrit地址	Gerrit仓库地址（https协议地址）。
用户名	Gerrit仓库用户名。
密码	Gerrit仓库密码。

步骤3 单击“确定”。

----结束

4 配置构建步骤

4.1 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。

编译构建服务支持使用自定义执行机，支持的自定义执行机类型有LINUX、LINUX_DOCKER、WINDOWS和MAC，各个类型支持的构建场景可参考**表4-1**，用户可根据实际需求选择使用的执行机类型。

表 4-1 各个类型执行机的使用说明

执行机类型	使用说明
LINUX	• 执行构建任务时，可通过执行shell命令，在Linux虚拟机上执行构建任务，拥有更高的自由度。 • 在使用编译构建服务前用户需要在自定义执行机上自行安装构建工具，例如Maven、Gradle等。 • 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**。
LINUX_DOCKER	• 执行构建任务时，编译构建服务将拉起一个Linux Docker容器，构建任务在容器中执行。 • 整个构建过程在容器中运行，运行后容器会自动清理构建镜像，包括构建过程中拉取的代码、过程数据、构建产物等。 • 支持用户宿主机目录与容器目录映射，即可在镜像内共享宿主机目录。 • 除MSBuild构建步骤外，支持所有构建步骤，无需自行安装构建环境。

执行机类型	使用说明
WINDOWS	• 执行构建任务时，构建任务在Windows执行机上执行，支持用户执行Windows相关的构建任务。 • 通过Git Bash工具执行Shell脚本实现构建。 • 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** • 支持Windows7、Windows10、Windows Server2012和Windows Server2016。 • 自定义Windows执行机前，需已安装JDK和Git。 • 编译工具需自行安装。例如：使用Maven构建，则需要安装Maven工具。
MAC	• 执行构建任务时，构建任务在MAC执行机上执行Shell命令，支持用户执行Windows相关的构建任务。 • 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** • 支持当前在使用的所有Mac版本。

图形化配置

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**。

代码化构建

代码示例如下：

```
version: 2.0 # 必须是2.0，该版本号必填且唯一
env: # 定义构建环境信息。非必填，如果不填写，默认使用X86
```



```
resource:
  type:docker # 资源池类型：docker或custom，其中docker表示使用默认执行机，custom表示使用自定义执行机
  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
```

4.2 配置代码下载

配置代码下载方式。

图形化构建

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。

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：需按照以下操作执行。

Tag是指代码仓库中的标签，若代码源选择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：需按照以下操作执行。

CommitID是指提交代码时生成的编号，若代码源选择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。

代码化构建（单仓下载）

```
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	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

代码化构建（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，多个代码仓的集成下载效率至关重要。

编译构建集成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，必须为同一种代码源。
2. manifest_url与manifest_file必须为同一种代码源；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**配置构建任务参数（可选操作）**。
6. 该功能目前仅支持北京四区域使用，其余区域后续上线。

4.3 使用 Maven 构建

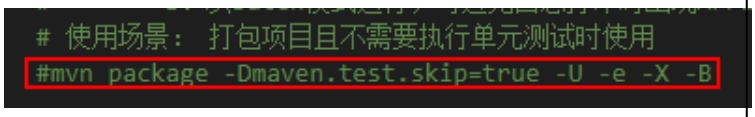
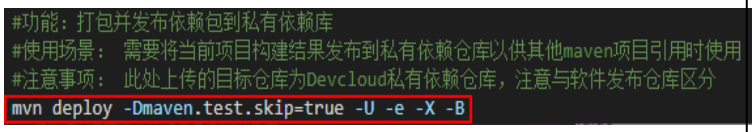
图形化构建

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。
命令	配置Maven命令，一般使用系统默认生成的命令即可。

参数项	说明
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。 • 公有依赖仓库：默认已添加华为开源镜像站，同时配置了HuaweiSDK仓库。此配置仅在需要添加非软开云提供的公有依赖仓库时使用，添加方法如下： 1. 单击“添加”。 2. 填写公有依赖仓库地址，根据需要勾选“release仓库”和“snapshot仓库”。release仓库和snapshot仓库至少勾选一个，也可以同时勾选。 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。 • 私有依赖库：默认已配置软件开发生产线提供的私有依赖仓库。此配置仅在需要添加其它私有依赖仓库时使用，添加方法如下： 1. **新建nexus repository服务扩展点**，如：test01。 2. 单击“添加”，选择上一步创建的服务扩展点，并根据需要勾选“release仓库”和“snapshot仓库”。 **说明** “release仓库”和“snapshot仓库”两种仓库对应的使用场景区分如下，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。 • “snapshot仓库”：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。

参数项	说明
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。配置前，需已**创建私有依赖库**。配置方法如下： • 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。 配置后，需在命令窗口，使用“#”注释命令**mvn package -Dmaven.test.skip=true -U -e -X -B**，如下图：  删除**#mvn deploy -Dmaven.test.skip=true -U -e -X -B**命令前的“#”，如下图：  上传的私有依赖包，在其他项目添加pom.xml文件中的groupId、artifactId、version坐标即可引用。
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。

代码化构建

```
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
        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
```

参数名	参数类型	描述	是否必填	默认值
image	string	镜像地址，有以下两种格式。 ● 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	是	无
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

配置单元测试（可选）

图形化构建时，如果需要配置单元测试，需执行以下操作。

1. 配置单元测试前，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。

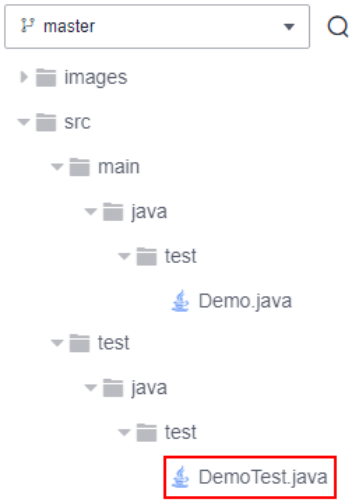
  单击代码仓库名称，进入代码托管“文件”页，确保以下代码未存在于项目“pom.xml”文件中。

```
<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>
```

2. 新建代码仓库并上传代码至代码仓库。
3. 在src目录下创建单元测试类，如下图所示。



Demo代码样例如下：

```
package test;

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

DemoTest代码样例如下：

```
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");
    }
}
```

4. 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
```

5. 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
```

6. 展开“单元测试”，根据实际需求配置如下参数。
   - 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。

   配置完成后，如果构建任务执行成功，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。

配置使用 JaCoCo 生成单元测试覆盖率报告（可选）

配置单元测试时，如果选择处理单元测试覆盖率结果，则需要按照如下指导进行配置。

- 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>
```

4.4 使用 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。

图形化构建

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。

代码化构建

```
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

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 Android APK 签名

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

图形化构建

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。

参数	说明
apksigner命令行	用户自定义签名参数，默认“--verbose”显示签名详情。

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。

代码化构建

```
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

4.6 使用 Npm 构建

使用Npm工具管理软件包，能完成vue和webpack的构建。

图形化构建

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。

参数项	说明
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。

代码化构建

```
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

4.7 使用 Gradle 构建

使用Gradle构建工具构建Java，Groovy和Scala项目。

图形化构建

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。

代码化构建

```
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

4.8 使用 Yarn 构建

使用Yarn构建JavaScript工程。

图形化构建

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。

代码化构建

```
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

4.9 使用 gulp 构建

使用gulp工具构建前端集成开发环境。

图形化构建

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。

代码化构建

```
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

4.10 使用 Grunt 构建

使用Grunt构建JavaScript工程。

图形化构建

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。

代码化构建

```
ersion: 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

4.11 使用 mono 构建

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

图形化构建

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。

代码化构建

```
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

4.12 使用 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。

图形化构建

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。

参数项	说明
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3行默认命令中“http://mirrors.huaweicloud.com/repository/php/”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。

代码化构建

```
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

4.13 使用 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**。

图形化构建

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。

代码化构建

```
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

4.14 使用 PyInstaller 构建

使用PyInstaller工具构建Python项目。

图形化构建

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。

参数项	说明
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。

代码化构建

```
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

4.15 使用 shell 命令执行构建

图形化构建

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。

代码化构建

```
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

4.16 使用 Gnu-arm 构建

使用GNU-ARM工具链设计、开发和使用ARM模拟器。

图形化构建

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“\#”。

代码化构建

```
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

4.17 使用 Msbuild 构建

编译构建服务提供了常用的构建模板（构建环境），不同的构建模板中预装了对应构建所需工具集，MSBuild构建镜像一般预装了msbuild、nuget、.NET Framework等常用工具。

使用msbuild构建工具执行引擎、构造工程，支持.NET框架构建，包含.NET Core和.NET Frameworks。

说明

- 该步骤仅支持图形化构建。
- 该步骤仅可单独使用，如果构建任务中已有其他构建步骤，将无法添加“Msbuild构建”。

配置说明

在**配置构建步骤**中，添加“Msbuild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Msbuild构建工具版本。 • 请选择相应的.NET版本。 • msbuild15-all版本的工具提供更多的构建支持，如nant、nodejs的构建命令支持等。
powershell命令	配置Msbuild命令，一般使用系统默认给出的命令即可。 • nuget restore命令会还原csharp项目依赖。 • msbuild命令构建项目： – OutputPath会指定生成路径，此路径设置会以csproject文件为相对路径。 – 项目存在多个csproject时可能会因此导致构建失败，删除路径指定参数后可构建成功。 • .NET Core项目请尝试使用.NET相关命令构建。

Msbuild 系统规格说明

为方便使用，编译构建服务提供的Msbuild构建环境原则上尽可能保持与本地环境一致，但因Windows系统与编译构建服务本身一些系统限制，少数场景下可能会导致构建失败。

使用前建议仔细阅读以下规格说明。

- **不支持带空格的文件路径**

  C#项目中目录或文件名中包含空格会导致构建失败，目录/文件的命名请使用字母、数字、下划线的组合，勿使用其他特殊字符，避免不必要的构建失败。

- **文件全路径长度不得超过260个字符**
  - Windows系统中，文件全路径的最大长度限制为260个字符，超过此长度会导致Msbuild构建失败。
  - 编译构建服务约定在指定目录下执行命令，即您的构建场景实际与以下步骤类似：

    ```
    cd C:\编译构建的默认路径\您的项目路径
    msbuild
    ```

    说明

    - 项目文件全路径长度实际为项目下文件相对路径长度与编译构建服务默认路径长度之和。
    - 编译构建服务默认路径长度为45字符。

      因此，在使用Msbuild构建的过程中，您的项目文件路径需满足：项目下文件相对路径（以代码仓库为根目录）长度不可大于215字符。
    - 一些特殊场景（如构建时指定输出目录为“Output/release”)下，可能会额外占用路径长度。

      建议您的项目下文件相对路径（以代码仓库为根目录）长度保持在200个字符以下，原则上尽可能短。

- **不可直接引用系统不具备的组件**
  - 部分场景下，解决方案中可能不使用NuGet等管理工具，直接引用默认路径下的程序集。

    但构建时环境中不具备此程序集，导致编译告警，如果项目代码中使用了此引用，甚至会直接导致失败。
  - 由于Windows系统特殊性，通常情况下，此类程序集默认安装于本地系统，无需指定程序集位置，VS构建时会从默认配置的几个程序集路径查找，可以构建成功。

    而云端构建环境对应目录无此程序集，进而导致云端构建环境与本地不一致带来的失败。
  - 为解决此类场景，Msbuild集成了NuGet，可以在构建时从远程仓库下载对应程序集，此时只需于项目中指定“packages.config”，并于其中声明依赖的程序集即可。
  - 特殊情况下，项目引用的程序集可能无法在远程仓库找到，此时需要手工保存程序集至代码仓库中，并显式指定程序集路径。

    详细解决方案请参见**找不到程序集（**.dll）**。

- **命令行中路径分隔符使用'/'而非'\'**

部分场景下，可能需要在命令行中使用路径参数，此处需注意，Msbuild构建环境要求路径分隔符统一使用“ / ”格式。

错误示例：

```
cd test\test1或cd test\\test1
```

正确示例：

```
cd test/test1
```

- **避免直接指定低版本SDK路径**
  - 编译构建服务提供了“.NET Framework”的4.7.2版本和3.5版本（详细说明请参见已支持的镜像版本及工具集）。

    一般来讲，4.7.2版本可以兼容4.0以上版本SDK，3.5版本可以兼容3.5版本以下SDK，项目中可以引用兼容版本的SDK内容。
  - 在某些场景下，用户可能将引用直接指向了某个低版本的sdk路径，此时会因找不到SDK导致构建失败。

    如果您的项目出现此类场景，建议：

    - 尝试更改您的引用路径，尽可能使用兼容版本SDK。
    - 如果您的项目因为不可回避的原因，必须指向低版本SDK路径，请尝试联系客服。

Msbuild 构建场景

- 已支持场景

场景类型	说明
无外部依赖	参考镜像版本及对应工具版本，对于仅使用了环境预装依赖库的项目，选择合适的镜像版本即可直接使用**msbuild**或**.NET**命令进行构建。 例如：项目使用了dotnetframework4.7.2的SDK和Office操作的相关官方依赖库（MSOffice）。可选用“msbuild15-dotnetframework4.7.2”版本镜像，使用**msbuild**命令构建。
使用Nuget进行依赖管理	对于使用了环境预装依赖库以外的项目，但使用了Nuget对**所有依赖库**进行管理的项目，选择合适的镜像版本后，可先使用**nuget restore**命令下载所有依赖，此后使用**msbuild**命令进行构建。**.NET**命令无需先执行**nuget**命令。 例如：项目使用了“dotnetframework4.7.2”的SDK，依赖了Myget上某Package并使用Nuget添加了该依赖。可选用“msbuild15-dotnetframework4.7.2”版本镜像，使用**nuget restore && msbuild**命令构建。
其他	对于有其他命令需求的项目，如Git、JDK、Nant、Nodejs等，请参见msbuild15-all版本镜像构建工具集的说明，使用支持的命令进行操作。

- 未支持场景

场景类型	说明
未使用Nuget管理依赖库	依赖了本地安装的依赖库，且没有使用nuget对依赖进行管理。详细解决方案请参见**找不到程序集（**.dll）**。 例如：某项目使用“dotnetframework4.7.2”的SDK，本地安装了Nunit依赖库，但没有使用Nuget对其进行管理。 此时使用**msbuild**命令对其进行构建时会出现找不到库的错误，导致构建失败。
解决方案版本低于VS2015（不包含）	对于使用VS2015（不含）以前版本创建的解决方案，会出现版本过低不兼容的情况，导致构建失败。请尝试升级解决方案。

Powershell 命令使用

- **Powershell使用操作**

  如果在构建任务中使用**powershell**命令，在命令行窗口按如下格式输入命令即可：

  ```
  powershell –Command Powershell命令参数列表
  ```

- **常用Powershell命令**

命令类型	说明
Compress-Archive（压缩命令）	powershell –Command Compress-Archive -Path [*SourcePath*] -DestinationPath [*Target.zip*] • SourcePath 指定需要压缩的文件或文件夹，支持通配符和相对路径。 • Target.zip 输出的压缩文件名，可用于上传到软件发布库时填写文件名。
Expand-Archive（解压缩命令）	powershell –Command Expand-Archive -Path [*SourcePath*] -DestinationPath [*TargetPath*] • SourcePath 指定需要解压的文件，如“demo.zip”。 • TargetPath 要解压到的目标路径，支持通配符和相对路径。
Copy-Item（复制命令）	powershell –Command Copy-Item -Recurse -Path [*SourcePath*] -DestinationPath [*TargetPath*] • SourcePath 指定需要复制的文件或文件夹，支持通配符和相对路径。 • TargetPath 要复制到的目标路径，支持通配符和相对路径。 **说明** -Recurse选项为循环复制子文件夹，但若在**SourcePath**中使用了通配符，此开关会失效，不会复制指定目录下的子文件夹。

说明

文档只介绍了构建常用的Powershell命令，更多Powershell命令请参见**微软官方文档**。

4.18 使用 CMake 构建

使用CMake构建工具构建跨平台项目工程。

图形化构建

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。

代码化构建

```
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

4.19 使用 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。

图形化构建

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。

代码化构建

```
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

4.20 使用 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。

图形化构建

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。

代码化构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - kotlin:
        inputs:
          command:  gradle 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

4.21 使用 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。

图形化构建

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。

代码化构建

```
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

4.22 使用 Ionic Android App 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]#
```

图形化构建

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“angular.json”等项目编译描述文件。

步骤2 在**配置构建步骤**中，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

代码化构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - ionic_android_app:
        inputs:
          gradle: '4.8'
          jdk: '3333'
          ndk: '17'
          command: ./instrumented.apk
```

参数名	参数类型	描述	是否必填	默认值
gradle	string	gradle版本。	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

4.23 使用 Android 快应用构建

npm config set xxx命令：配置Npm相关设置。

图形化构建

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。

代码化构建

```
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

4.24 使用 GFortran 构建

安装了GFortran编译环境的gcc、cmake、gfortran和openmpi等工具，支持编译单个GFortran源码或者整个GFortran项目。

图形化构建

在**配置构建步骤**中，添加“GFortran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GFortran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。

代码化构建

```
version: 2.0 # 必须是2.0
steps:
  BUILD:
    - fortran:
        inputs:
          command: |-
            gfortran -c -fpic helloworld.f90
            gfortran -shared -o helloworld.so helloworld.o
```

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

4.25 使用 Sbt 构建

使用Sbt工具构建Scala和Java项目。

图形化构建

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。

代码化构建

```
ersion: 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

4.26 使用 Grails 构建

使用Grails构建Web应用。

图形化构建

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。

代码化构建

```
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

4.27 使用 Bazel 构建

使用Bazel工具构建。

图形化构建

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。

代码化构建

```
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

4.28 使用 Flutter 构建

使用Flutter构建安卓应用。

图形化构建

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。

代码化构建

```
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

4.29 使用构建方舟编译器构建

使用ubuntu操作系统编译方舟编译器进行构建。

图形化构建

在**配置构建步骤**中，添加“构建方舟编译器”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。

代码化构建

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

4.30 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。

图形化构建

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。

参数项	说明
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。

代码化构建

```
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

4.31 上传软件包到软件发布库

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**。

图形化构建

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。

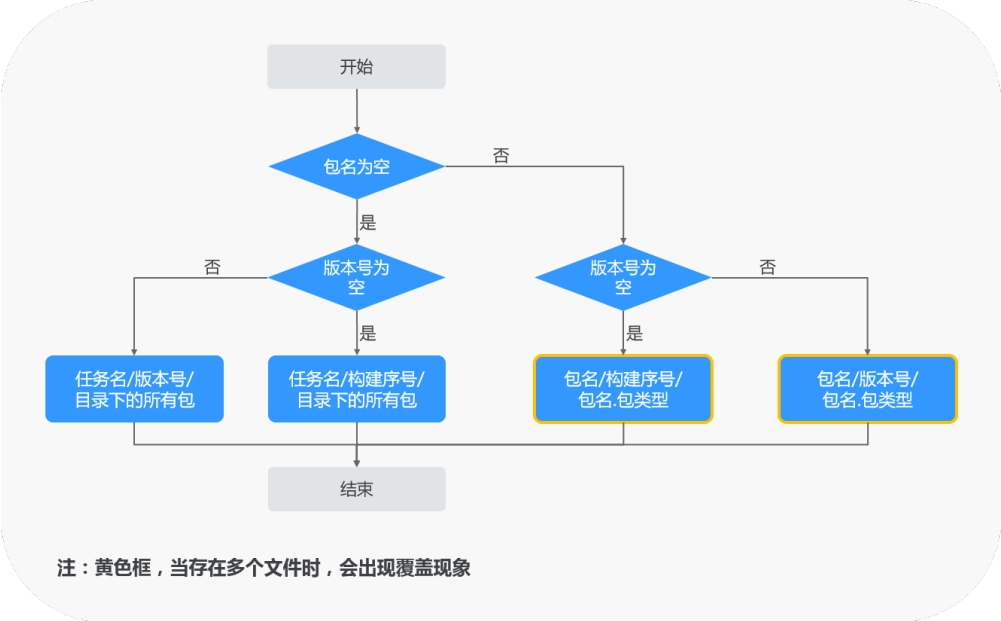
  例如，“a”目录下有“aa”文件和“b”目录，“b”目录下有“bb”文件，构建包路径配置为“a/**”。

  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。

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。
构建包路径	构建结果所在路径。 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	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。包名推荐设置为空，可以上传构建包路径匹配的所有文件。 指定：上传多个文件时，可能会存在被覆盖的情况。如果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：



代码化构建

```
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

4.32 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**。

镜像设置为“公开”

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。

图形化构建

步骤1 在**配置构建步骤**中，添加“使用SWR公共镜像”构建步骤。

步骤2 将4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3 在命令窗口输入构建命令。

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。

----结束

代码化构建

```
ersion: 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。 镜像地址有两种格式： • 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:141d26c455abd6d7xxxxxxxxxxxxxxxxxx	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

4.33 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**。

图形化构建

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。
是否上传文件夹	可选择是否开启上传文件夹。
忽略文件夹路径	忽略的文件夹路径。选择上传文件夹时，会根据此路径忽略部分文件夹，不上传到OBS。 如产物路径填写为“target/api/api.jar”，忽略文件夹路径填写为“target”，obs存储目录为“./”，则会将“api.jar”上传到OBS桶的“api/api.jar”路径下。若路径无法匹配，则默认不忽略路径中的文件夹。

参数项	说明
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。

代码化构建

```
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

4.34 通过 Docker 命令操作镜像

图形化构建

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。

代码化构建

```
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

4.35 下载软件发布库中的软件包

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

图形化构建

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。

代码化构建

```
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

4.36 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。

图形化构建

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。

代码化构建

```
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

4.37 单元测试报告

仅支持图形化构建。

配置说明

在配置构建步骤中，添加“单元测试报告”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。

参数项	说明
工具版本	根据需要选择工具版本。
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。 • 测试报告类型：选择单元测试的框架，目前仅支持junit。 • 单元测试结果文件：填写相对于项目根目录的相对路径，如 target/surefire-reports/TEST*.xml。目前只支持标准的.xml格式单元测试报告。 • 是否处理单元测试覆盖率结果：根据实际需要选择。若选“是”， 需确认项目中有使用jacoco-maven-plugin插件生成单元覆盖率报告。

4.38 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。	否	false

参数名	参数类型	描述	是否必填	默认值
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** • 如果未配置清理固定路径，当文件容量达到工作空间上限后，会自动清理当前租户下的固定路径中所有文件。 • 工作空间指的是用户自定义的执行机的规格。	否	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。	否	不涉及

5 执行和加速构建任务

5.1 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。
- 若当前构建任务并发数无法满足用户的需求，可**购买构建并发包**，增加构建任务并发数。并发包的使用规则可参考**如何使用构建并发包**。

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。

5.2 Gcc/Clang 构建加速

Gcc/Clang构建加速是指通过分布式编译、增量编译等技术，实现对软件编译过程的效率提升，支撑企业研发过程的快速迭代，缩短产品的上市周期。

说明

- 目前该功能仅支持C/C++语言的编译加速。
- 使用构建加速能力需要额外购买配套构建加速包，构建加速包因加速原理以及效果的不同，共有三种规格以供购买，规格介绍及购买指南请参考**购买增值特性**。
- 用户基于自定义执行机的构建，无法使用构建加速能力。

前提条件

- 已有可用项目，如果没有，请**新建项目**。

- 已在项目中有可用代码仓库，如果没有，请**新建代码仓库**。
- **购买构建加速包**。

配置基本信息

1. **新建项目**。
2. **新建代码仓库**。
3. **登录编译构建服务首页**。
4. 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。使用图形化构建，参考**表5-1**；使用代码化构建，参考**表5-2**。

表 5-1 图形化构建基本信息配置说明

参数项	描述
名称	任务的名称。
归属项目	任务所属项目。
代码源	• Repo：从**代码托管服务**拉取代码进行构建。 • 其他项目Repo：从其他项目的代码托管中拉取代码进行构建，请选择已有的项目、该项目下已经创建的代码仓以及默认分支。 • 来自流水线：如果选择来自**流水线**，则只能通过流水线任务驱动执行，不能单独执行。 以下为第三方代码仓库，首次使用第三方代码仓，需新建服务扩展点，详情可参考**新建服务扩展点（可选操作）**。 • GitHub：拉取托管在GitHub上的代码进行构建。 • 通用Git：拉取托管在其他服务上的代码进行构建。 • GitCode：拉取托管在GitCode仓库上的代码进行构建。 • 码云：拉取托管在码云上的代码进行构建。 • Gerrit：拉取托管在Gerrit上的代码进行构建。
代码仓	选择实际使用的代码仓。
默认分支	选择仓库默认分支。
任务描述	对任务进行描述。

表 5-2 代码化构建基本信息配置说明

参数项	描述
任务名称	任务的名称。

参数项	描述
归属项目	任务所属项目。
代码源	选择Repo：表示从代码托管拉取代码进行构建。
代码仓	选择实际使用的代码仓。
默认分支	选择仓库默认分支。
任务描述	对任务进行描述。

构建模板配置

步骤1 单击“下一步”，进入“构建模板”配置页面。

步骤2 选择“CMake模板”，单击“下一步”，进入“构建步骤”页签，**配置CMake构建加速**。

----结束

配置 CMake 构建加速

步骤1 添加“CMake构建”构建步骤。

步骤2 工具版本选择“cmake3.16.5-gcc7.3.0”，根据加速原理以及效果的不同，构建加速分为L1/L2/L3三种模式，请根据购买的加速规格使用加速命令，下方示例为开启L1模式加速：

```
cmake -G'Unix Makefiles'../&& BuildAccelerateL1 make -j8
//开启构建加速只需在make前添加加速命令：BuildAccelerateL1
//切换模式只需将BuildAccelerate后的L1替换为L2/L3。
//最大并发CPU核数，即make -j后面的数字，最大256。
```

```
#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' ../&& BuildAccelerateL1 make -j8
```

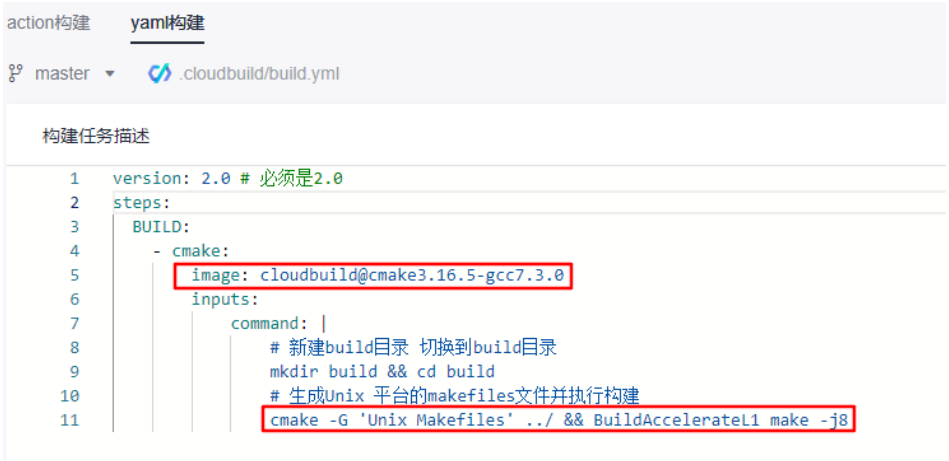
步骤3 单击“新建”，开始执行构建任务。

⚠ 注意

- 加速命令只允许写在最外层，不允许通过shell脚本等调用。
- 同一次构建任务中禁止混用不同级别的加速命令，将导致任务无法保存与执行，例如：

```
BuildAccelerateL1 make -j8
BuildAccelerateL2 make -j8
```

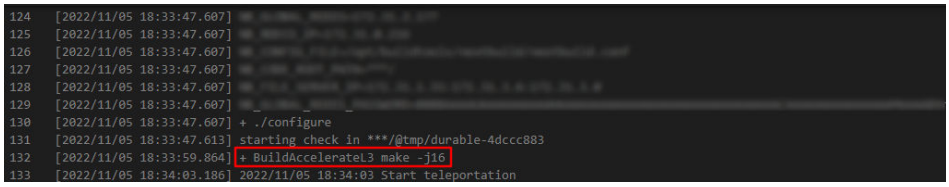
- 使用yaml配置CMake构建支持使用构建加速，与图形化构建相同，修改工具版本并添加加速命令即可，如何使用yaml配置CMake构建请参考**使用CMake构建**。



----结束

如何判断构建加速是否生效

使用CMake构建加速成功后，日志会打印出相应加速模式的信息，如下图即为L3模式加速生效：



5.3 鸿蒙构建加速

5.3.1 导读

构建加速服务通过解析鸿蒙构建工程的内部依赖关系，将其拆解分发至多台机器并发执行，结合增量编译技术，实现对软件编译过程的效率提升，支撑企业研发过程的快速迭代，缩短产品的上市周期。

说明

- 增量编译技术需结合L3级别加速使用。
- 使用鸿蒙构建加速能力需要购买配套构建加速包，购买方法请参考**购买增值特性**。
- 该功能目前仅支持北京四区域使用，其余区域后续上线。
- 本章节配置仅支持鸿蒙L1、L3级别的加速，L2级别待支持。
- 构建过程中不允许直接覆盖LD_LIBRARY_PATH和LD_PRELOAD环境变量，建议通过追加的方式使用，例如：
  ```
  "export LD_LIBRARY_PATH=new_path:${LD_LIBRARY_PATH}"
  ```

5.3.2 加速前准备

在一般的构建工程中，其构建过程大致分为构建前准备（工具链、代码仓）、构建依赖件准备（ninja文件生成）、编译构建、构建后操作（打包、检查等）。其中，构建加速介入编译构建阶段，对此前的构建过程中生成的构建依赖件进行解析，并执行编译。

在配置构建加速前，需如下准备：

- 找到构建依赖件准备的节点，以OpenHarmony为例，一个形态的编译命令如下：
  ```
  ./build.sh --product-name rk3568 --build-target make_all --build-target make_test --ccache false -v
  ```
- 准备好构建使用的docker镜像，基于该docker镜像制作新镜像：在dockerfile中增加“/opt/buildtools”目录供加速工具部署，并确保构建用户对“/opt/buildtools”目录有权限写入。参考命令如下：
  ```
  RUN mkdir -p /opt/buildtools && chmod -R 777 /opt/buildtools
  ```

5.3.3 配置详解

通过 BuildFlow 组织加速构建

构建加速需要结合多任务代码化构建使用，可参考**多任务YAML文件结构详解**中的部分配置。

BuildFlow配置方法如下样例：

```
buildflow:
  jobs_resolver: # 必配
    provider: tbuild_jobs_resolver # 必配，固定值
    jobs: # 需要进行编排的任务集
        - job: distribute_job # 构建任务名称
          build_ref: accelerate.yml # 指定构建加速脚本，脚本名称可自定义
          worker: 2 # 指定为16vCPU的倍数，例如2就代表使用了32vCPU进行加速
```

参数说明如下：

- jobs_resolver：buildflow的子节点，必配。
- provider：此处使用的provider为jobs_resolver的高级选项，意为指定job对应的任务解析器，取值固定为tbuild_jobs_resolver。
- jobs：需要进行编排的任务集，此处的jobs作为jobs_resolver的子节点，与普通构建场景buildflow下的jobs子节点有所区别，配置时请注意缩进。
- job：构建任务名称，可自定义。
- build_ref：该构建任务在构建过程中需要运行的加速构建脚本。
- worker：指定为16vCPU的倍数，例如2就代表使用了32vCPU进行加速。

配置示例 1：依赖解析模式

步骤1 获取json文件。

在分支A编写BuildFlow配置中build_ref指定的accelerate.yml，示例如下：

```
version: 2.0

params:
  - name: TB_GET_ORI_TRACE
    value: "1"

steps:
  PRE_BUILD:
  - checkout:
      name: "checkout"
      inputs:
        scm: "codehub"
        url: "git@codehub.devcloud.example.example.com:example.git"
        branch: "master"
        lfs: false
        submodule: false
  BUILD:
  - tbuild_execute:
      inputs:
        image: "swr.example.example.com/buildimage:ohos-x86-v1"
        command: "cd OpenHarmony && BuildAccelerateL3 -HarmonyOS ./build.sh --product-name rk3568 --
build-target make_all --ccache false -v && post_build.sh && mv out/TBTrace_make_all_1.json /example/
TB1.json"
```

步骤2 执行分布式构建。

在分支B编写BuildFlow配置中build_ref指定的accelerate.yml，示例如下：

```
version: 2.0

params:
  - name: TB_BUILDTRACE_ALL
    value: "1"
  - name: TB_RSYNC
    value: ${WORKSPACE}/OpenHarmony/:out/rk3568

steps:
  PRE_BUILD:
  - checkout:
      name: "checkout"
      inputs:
        scm: "codehub"
        url: "git@codehub.devcloud.example.example.com:example.git"
        branch: "master"
        lfs: false
        submodule: false
  BUILD:
  - tbuild_execute:
      inputs:
        image: "swr.example.example.com/buildimage:ohos-x86-v1"
        command: "mv /example/TB1.json OpenHarmony/TB1.json && cd OpenHarmony &&
BuildAccelerateL3 -HarmonyOS ./build.sh --product-name rk3568 --build-target make_all --ccache false -v
&& post_build.sh"
```

----结束

配置示例 2：产物分类模式

在分支C编写BuildFlow配置中build_ref指定的accelerate.yml，示例如下：

```
version: 2.0
```



```
params:
  - name: TB_RSYNC
    value: ${WORKSPACE}/OpenHarmony/:out/rk3568
  - name: TB_SEARCH_TARGETS_ALL
    value: "obj/build/ohos/common/generate_src_installed_info.stamp"

steps:
  PRE_BUILD:
  - checkout:
      name: "checkout"
      inputs:
        scm: "codehub"
        url: "git@codehub.devcloud.example.example.com:example.git"
        branch: "master"
        lfs: false
        submodule: false
  BUILD:
  - tbuild_execute:
      inputs:
        image: "swr.example.example.com/buildimage:ohos-x86-v1"
        command: "cd OpenHarmony && BuildAccelerateL3 -HarmonyOS ./build.sh --product-name rk3568 --
build-target make_all --ccache false -v && post_build.sh"
```

params 参数项详解

params配置项指定了一些使用鸿蒙构建加速必配的参数，参数说明如下：

参数项	是否必填	说明	示例
TB_RSYNC	是	需要同步的产物文件目录，“:”前为根目录，“:”后为若干个以“,”分隔的子目录，子目录前带“!”代表此目录不同步，不带“!”代表此目录的所有文件会被同步，带“!”的优先级更高。在鸿蒙构建场景下，需要拼接为“${WORKSPACE}/实际目录”。	"${WORKSPACE}/OpenHarmony/:out/rk3568,out/kernel,!out/rk3568/exe.unstripped,!out/rk3568/lib.unstripped,!out/rk3568/innerkits,!out/rk3568/.ninja_,!out/rk3568/Makefile,!out/rk3568/NOTICE_FILES,!out/rk3568/clang_x64/exe.unstripped,!out/rk3568/clang_x64/lib.unstripped,!out/rk3568/mingw_x86_64/lib.unstripped,+out/rk3568/obj/base/security/selinux_adapter"

参数项	是否必填	说明	示例
TB_GET_ORI_TRACE	配置示例1：依赖解析模式 必填	依赖解析模式下获取当前工程的依赖json文件开关。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_BUILDTRACE_ALL	配置示例1：依赖解析模式 必填	依赖解析模式开关，不设置时默认使用配置示例2：产物分类模式。 ● 1：开启。 ● 不设置：关闭（默认）。 ● 非“1”的其他字符串：开启，字符串视为json文件的自定义路径和名字。	OpenHarmony/harmony.json
TB_SEARCH_TARGETS_ALL	配置示例2：产物分类模式 必填	该值填写工程中汇总了各个模块的target，如鸿蒙的：parts_test.stamp 、generate_src_installed_info.stamp。一般的，这样的target的下一层直接依赖是工程中的多个小模块，如鸿蒙的ark模块、ace模块。这些对应的target在同一个工程里一般不会变化。	"obj/build/ohos/common/generate_src_installed_info.stamp"

steps 参数项详解

steps配置项定义了构建过程，示例中包含如下两个步骤：PRE_BUILD（构建前准备）和BUILD（编译构建）。

- PRE_BUILD

  此阶段主要做代码下载，参数解释如下：

```
PRE_BUILD:
- checkout: # 代码下载步骤
    name: "代码下载" # 步骤名称，可自定义
    inputs: # 步骤参数
      scm: "codehub " # 代码来源:只支持Repo
      url: "git@codehub.devcloud.example.example.com:test/python3.git" # 拉取代码的ssh地址。
      branch: "master" # 拉取的代码分支。
      lfs: false # 选择是否开启Git LFS，false关闭、true开启。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。
      submodule: false # false关闭、true开启。开启该功能，系统在构建时会自动拉取子模块仓库的代码。
```

- BUILD

  此阶段主要定义了download_artifact插件、tbuild_execute插件和upload_artifact插件，参数解释如下：

```
BUILD:
- tbuild_execute: # 鸿蒙加速场景下固定配置，定义tbuild_execute插件
    inputs: # 固定配置
      image: "swr.xx-xx-x.myxxcloud.com/buildimage:ohos-x86-v1" # 构建使用的镜像，参考加速前准备
```



```
章节制作docker镜像。
        command: "mv /example/TB1.json OpenHarmony/TB1.json && cd OpenHarmony &&
BuildAccelerateL3 -HarmonyOS ./build.sh --product-name rk3568 --build-target make_all --ccache
false -v && post_build.sh"
# command为构建使用的命令，此处将构建分解为两个段落，准备和执行
# mv /example/TB1.json OpenHarmony/TB1.json是依赖解析模式独有的准备步骤，文件名字固定，如果
工程中存在多个ninja构建，则文件的下标依次增加，例如TBTrace_target_2.json和TB2.json，以此类推。
# 准备阶段:使用加速前准备章节中获取的./build.sh --product-name rk3568 --build-target make_all --
build-only-gn --ccache false -v
# 构建阶段:依照加速级别调用加速命令（BuildAccelerateL1 BuildAccelerateL3）的鸿蒙模式（-
HarmonyOS）直接执行构建，此处样例取值BuildAccelerateL3 -HarmonyOS
# 后处理阶段:以实际工程需要为准，该示例仅使用post_build.sh
# 依赖解析模式实际命令最终拼接为"mv /example/TB1.json OpenHarmony/TB1.json && cd
OpenHarmony && BuildAccelerateL3 -HarmonyOS ./build.sh --product-name rk3568 --build-target
make_all --ccache false -v && post_build.sh"
# 产物分类模式实际命令最终拼接为"cd OpenHarmony && BuildAccelerateL3 -HarmonyOS ./build.sh --
product-name rk3568 --build-target make_all --ccache false -v && post_build.sh"
```

build.sh样例：

```
source build/envsetup.sh
lunch aosp_x86_64-eng
make -j64
```

高级选项（可选）

高级选项均为非必填选项，在构建过程中有工程无法执行需要特殊适配或优化性能时配置，若随意配置可能会导致构建失败。

表 5-3 通用选项

参数项	说明	示例
TB_GET_TRACE	构建结束后获取依赖json文件文件的开关。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_NINJA_RULE_ALL	分割规则配置，target按数量比值分组。不设置时会自动配置合适值。	"1:2:3:4"
TB_TARGETS_LIST_ALL	人工指定分发的target进行编译，每个逗号隔开一个worker的target。使用星号分隔多个ninja工程的配置。不设置时会自动配置合适值。	"harmony_target_1 harmony_target_2,harmony_target_3 harmony_target_4"
TB_HOOK_LOCK	对软链接也进行文件同步。若构建过程中发生软链接文件未同步导致的报错，需要开启此选项。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_APPEND_PATH	构建时可向PATH环境变量中追加的参数。 ● 1：开启。 ● 0：关闭（默认）。	1

参数项	说明	示例
TB_SHUT_DOWN_SAME_TIME	所有worker都等待主节点执行完毕后再结束构建释放资源。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_RSYNC_LOCK	构建加速的同时worker向构建执行机实时传输文件。开启后效率会进一步提升，但会存在概率性编译失败的情况。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_MAKE_J	设置构建并发数。默认为worker核数。	16
TB_REFER_NINJA_FILE	如果存在串行执行一个一模一样的ninja工程时，可以使用此变量优化构建速度 ● 1：开启。 ● 0：关闭（默认）。	1
CCACHE_DIR	自定义编译缓存的本地目录。默认为/tmp/xcache目录。	${WORKSPACE}/TBcache
TB_CACHE_SIZE	使用自定义执行机自定义编译缓存的本地目录存储大小上限。默认为100G。	100G
CCACHE_MAXSIZE	自定义编译缓存的本地目录存储大小上限。默认为20G。	100G
TB_ONE_WORKER	使用自定义执行机且只使用一个worker进行构建时可以使用此变量打开编译缓存开关。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_CLIENT	该变量在主节点client自动设置，可以通过比较此变量是否等于1判断该节点是否为主节点client。	不需要配置
TB_NET_INTERFACE_NAME	指定获取IP时读取的网卡名，在多网卡情况下获取IP使用，默认为空，多个网卡名通过逗号分隔，配置在前的网卡名有更高的优先级。	eth0,eth1
TB_ACC_PREPARE	使用自定义执行机时必须配置此变量。	false
TB_OUTPUT_PATH	自定义产物目录路径，默认设置为out。	output

参数项	说明	示例
TB_SELF_ENV	worker编译target时使用本地环境变量，不使用client传递的变量。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_RSYNC_FLAG	增加同步文件时rsync命令的参数	--ignore-existing -a
TBUILD_PLUGIN_PKG_TYPE	指定使用的TBuild版本为snapshot版本还是release版本。不设置时默认使用最新的release版本。	release
TBUILD_PLUGIN_PKG_VERSION	指定使用的TBuild版本号。不设置时默认使用最新的release版本。	1.0.1

表 5-4 依赖解析模式

参数项	说明	示例
TB_CPU_NUM	分发任务时，所有机器都以此值计算可分配任务量。默认为空。	16
TB_LOCAL_CONTAIN_PATTERN	根据关键字指定必须分发在client编译的target，此target生成的产物不需要文件传输回主节点。默认为空。	file_contexts.bin,com.android.vndk.current.,out/soong/host/linux-x86/bin/fileslist
TB_LOCAL_NOT_CONTAIN_PATTERN	根据关键字指定必须不分发在client编译的target。默认为空。	Bluetooth.so
TB_CAPACITY_ALL	单台机器可分配的任务权值上限，单位是分钟，以json文件的时间为参考，此时间为单纯的构建时间，不包括cpu空闲时间，实际构建时间会大于此值。可以指定为小数。超过此上限的target会分发至client，在所有agent编译完成后在本地最后编译。不设置时会根据json文件自动设置。	5.5
TB_TASK_SIZE_ALL	人工指定分割多少份target，可以大于机器数量，建议设置的值略大于机器数量，不建议少于机器数量，会导致机器浪费。不设置时会根据机器数量自动设置。	8

表 5-5 ninja 文件缓存选项

参数项	说明	示例
TB_CACHE_SERVER_IP	ninja文件缓存开关，和TB_CACHE_ARCHIVE_PATH同时设置时缓存才会开启。 ● 非空值：开启，如果开启了远端缓存，该值视为远端服务器IP。 ● 空值：关闭（默认）。	172.example.example.example
TB_CACHE_ARCHIVE_PATH	ninja文件缓存本地归档目录，和TB_CACHE_SERVER_IP同时设置时缓存才会开启。	AOSP/ninja_cache
TB_CACHE_RECACHE	本次构建会重新生成ninja文件缓存，不会命中历史缓存。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_CACHE_REMOTE	ninja文件缓存远端开关，命中时从远端获取缓存，生成缓存时也会归档至远端。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_CACHE_LOCAL	ninja文件缓存本地开关，命中时从本地获取缓存，优先级高于远端缓存，生成缓存时也会归档至本地。 ● 1：开启（默认）。 ● 0：关闭。	1
TB_CACHE_DEPENDS	增加指定的文件作为缓存命中的依赖文件，如果该文件产生变化，会使缓存不命中。默认为空。多个文件使用逗号分隔。	build.sh,test.sh
TB_CACHE_VERSION	为缓存增加指定的版本号，如果版本号产生变化，会使缓存不命中。默认为空。	1.0
TB_CACHE_EXCLUDE_KEY	缓存时过滤掉带关键字的文件，多个关键字用逗号分隔，默认设置为".glob"，可以用空字符串重置。	.so,.glob
TB_NINJA_FILE_CACHE	设置被缓存的目录，多个目录使用逗号分隔，默认设置为"out"。	out,test
TB_CACHE_NINJA_CACHE_SIZE	ninja文件缓存本地归档目录空间上限，超过此上限会根据算法自动清理历史缓存。默认设置为50，单位是G。	100

参数项	说明	示例
TB_SKIP_TARGET	归档ninja文件缓存时跳过此target，即使命中ninja文件缓存此target也会重新编译。	update-api

5.4 AOSP 构建加速

5.4.1 导读

构建加速服务通过解析AOSP构建工程的内部依赖关系，将其拆解分发至多台机器并发执行，结合增量编译技术，实现对软件编译过程的效率提升，支撑企业研发过程的快速迭代，缩短产品的上市周期。

说明

- 增量编译技术需结合L3级别加速使用。
- 使用AOSP构建加速能力需要购买配套构建加速包，购买方法请参考**购买增值特性**。
- 该功能目前仅支持北京四区域使用，其余区域后续上线。
- 本章节配置仅支持鸿蒙L1、L3级别的加速，L2级别待支持。
- 构建过程中不允许直接覆盖LD_LIBRARY_PATH和LD_PRELOAD环境变量，建议通过追加的方式使用，例如：
  ```
  "export LD_LIBRARY_PATH=new_path:${LD_LIBRARY_PATH}"
  ```

5.4.2 加速前准备

在一般的构建工程中，其构建过程大致分为构建前准备（工具链、代码仓）、构建依赖件准备（ninja文件生成）、编译构建、构建后操作（打包、检查等）。其中，构建加速介入编译构建阶段，对此前的构建过程中生成的构建依赖件进行解析，并执行编译。

在配置构建加速前，需如下准备：

- 找到构建依赖件准备的节点，以AOSP为例，一个形态的编译命令如下：
  ```
  source build/envsetup.sh
  lunch aosp_x86_64-eng
  make -j64
  ```
- 准备好构建使用的docker镜像，基于该docker镜像制作新镜像：在dockerfile中增加“/opt/buildtools”目录供加速工具部署，并确保构建用户对“/opt/buildtools”目录有权限写入。参考命令如下：
  ```
  RUN mkdir -p /opt/buildtools && chmod -R 777 /opt/buildtools
  ```

5.4.3 配置详解

通过 BuildFlow 组织加速构建

构建加速需要结合多任务代码化构建使用，可参考**多任务YAML文件结构详解**中的部分配置。

BuildFlow配置方法如下样例：

```
buildflow:
  jobs_resolver: # 必配
    provider: tbuild_jobs_resolver # 必配，固定值
    jobs: # 需要进行编排的任务集
        - job: distribute_job # 构建任务名称
          build_ref: accelerate.yml # 指定构建加速脚本，脚本名称可自定义
          worker: 2 # 指定为16vCPU的倍数，例如2就代表使用了32vCPU进行加速
```

参数说明如下：

- jobs_resolver：buildflow的子节点，必配。
- provider：此处使用的provider为jobs_resolver的高级选项，意为指定job对应的任务解析器，取值固定为tbuild_jobs_resolver。
- jobs：需要进行编排的任务集，此处的jobs作为jobs_resolver的子节点，与普通构建场景buildflow下的jobs子节点有所区别，配置时请注意缩进。
- job：构建任务名称，可自定义。
- build_ref：该构建任务在构建过程中需要运行的加速构建脚本。
- worker：指定为16vCPU的倍数，例如2就代表使用了32vCPU进行加速。

配置示例 1：依赖解析模式

步骤1 获取json文件。

在分支A编写BuildFlow配置中build_ref指定的accelerate.yml，示例如下：

```
version: 2.0

params:
  - name: TB_GET_ORI_TRACE
    value: "1"

steps:
  PRE_BUILD:
  - checkout:
      name: "checkout"
      inputs:
        scm: "codehub"
        url: "git@codehub.devcloud.example.example.com:example.git"
        branch: "master"
        lfs: false
        submodule: false
  BUILD:
  - tbuild_execute:
      inputs:
        image: "swr.example.example.com/buildimage:AOSP"
        command: "cd AOSP && chmod a+x build.sh && BuildAccelerateL3 -AOSP ./build.sh && mv out/
TBTrace_droid_1.json /example/TB1.json"
```

步骤2 执行分布式构建。

在分支B编写BuildFlow配置中build_ref指定的accelerate.yml，示例如下：

```
version: 2.0

params:
  - name: TB_BUILDTRACE_ALL
    value: "1"
  - name: TB_RSYNC
    value: ${WORKSPACE}/AOSP/:out/target/product/generic_x86_64

steps:
  PRE_BUILD:
  - checkout:
```



```
    name: "checkout"
    inputs:
      scm: "codehub"
      url: "git@codehub.devcloud.example.example.com:example.git"
      branch: "master"
      lfs: false
      submodule: false
  BUILD:
  - tbuild_execute:
      inputs:
        image: "swr.example.example.com/buildimage:AOSP"
        command: "mv /example/TB1.json AOSP/TB1.json && cd AOSP && chmod a+x build.sh &&
BuildAccelerateL3 -AOSP ./build.sh && ./post_build.sh"
```

----结束

配置示例 2：产物分类模式

在分支C编写BuildFlow配置中build_ref指定的accelerate.yml，示例如下：

```
version: 2.0

params:
  - name: TB_RSYNC
    value: ${WORKSPACE}/AOSP/:out/target/product/generic_x86_64

steps:
  PRE_BUILD:
  - checkout:
      name: "checkout"
      inputs:
        scm: "codehub"
        url: "git@codehub.devcloud.example.example.com:example.git"
        branch: "master"
        lfs: false
        submodule: false
  BUILD:
  - tbuild_execute:
      inputs:
        image: "swr.example.example.com/buildimage:AOSP"
        command: "cd AOSP && chmod a+x build.sh && BuildAccelerateL3 -AOSP ./build.sh && ./post_build.sh"
```

params 参数项详解

params配置项指定了一些使用AOSP构建加速必配的参数，参数说明如下：

参数项	是否必填	说明	示例
TB_RSYNC	是	需要同步的产物文件目录，“:”前为根目录，“:”后为若干个以“,”分隔的子目录，子目录前带“!”代表此目录不同步，不带“!”代表此目录的所有文件会被同步，带“!”的优先级更高。在AOSP构建场景下，需要拼接为“${WORKSPACE}/实际目录”。	"${WORKSPACE}/AOSP/:out/target/product/generic_x86_64,\!out/target/product/generic_x86_64/obj,\!out/target/product/generic_x86_64/symbols,\!out/target/product/generic_x86_64/obj_x86,\!out/target/product/generic_x86_64/obj_arm,\!out/target/product/generic_x86_64/gen"
TB_GET_ORI_TRACE	**配置示例1：依赖解析模式**必填	依赖解析模式下获取当前工程的依赖json文件开关。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_BUILD TRACE_ALL	**配置示例1：依赖解析模式**必填	依赖解析模式开关，不设置时默认使用**配置示例2：产物分类模式**。 ● 1：开启。 ● 不设置：关闭（默认）。 ● 非“1”的其他字符串：开启，字符串视为json文件的自定义路径和名字。	AOSP/aosp.json

steps 参数项详解

steps配置项定义了构建过程，示例中包含如下两个步骤：PRE_BUILD（构建前准备）和BUILD（编译构建）。

- PRE_BUILD

  此阶段主要做代码下载，参数解释如下：

```
PRE_BUILD:
- checkout: # 代码下载步骤
    name: "代码下载" # 步骤名称，可自定义
    inputs: # 步骤参数
      scm: "codehub " # 代码来源:只支持Repo
      url: "git@codehub.devcloud.example.example.com:test/python3.git" # 拉取代码的ssh地址。
      branch: "master" # 拉取的代码分支。
      lfs: false # 选择是否开启Git LFS，false关闭、true开启。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。
      submodule: false # false关闭、true开启。开启该功能，系统在构建时会自动拉取子模块仓库的代码。
```


- BUILD

  此阶段主要定义了download_artifact插件、tbuild_execute插件和upload_artifact插件，参数解释如下：

```
BUILD:
  - tbuild_execute: # AOSP加速场景下固定配置，定义tbuild_execute插件
      inputs: # 固定配置
        image: "swr.xx-xx-x.myxxcloud.com/buildimage:AOSP" # 构建使用的镜像，参考加速前准备章节制作docker镜像。
        command: "mv /example/TB1.json AOSP/TB1.json && cd AOSP && chmod a+x build.sh && BuildAccelerateL3 -AOSP ./build.sh && ./post_build.sh"
# command为构建使用的命令，此处将构建分解为两个段落，准备和执行
# mv /example/TB1.json AOSP/TB1.json 是依赖解析模式独有的准备步骤，文件名字固定，如果工程中存在多个ninja构建，则文件的下标依次增加，例如TBTrace_droid_2.json和TB2.json，以此类推。
# 准备阶段:在代码仓根目录新建build.sh，内容见build.sh样例
# 构建阶段:依照加速级别调用加速命令（BuildAccelerateL1 BuildAccelerateL3）的AOSP模式（-AOSP）直接执行构建，此处样例取值BuildAccelerateL3 -AOSP
# 后处理阶段:以实际工程需要为准，该示例仅使用post_build.sh
# 依赖解析模式实际命令最终拼接为"mv /example/TB1.json AOSP/TB1.json && cd AOSP && chmod a+x build.sh && BuildAccelerateL3 -AOSP ./build.sh && ./post_build.sh"
# 产物分类模式实际命令最终拼接为"cd AOSP && chmod a+x build.sh && BuildAccelerateL3 -AOSP ./build.sh && ./post_build.sh"
```

  build.sh样例：

```
source build/envsetup.sh
lunch aosp_x86_64-eng
make -j64
```

高级选项（可选）

高级选项均为非必填选项，在构建过程中有工程无法执行需要特殊适配或优化性能时配置，若随意配置可能会导致构建失败。

表 5-6 通用选项

参数项	说明	示例
TB_GET_TRACE	构建结束后获取依赖json文件文件的开关。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_NINJA_RULE_ALL	用于产物分类模式自定义Target切割及分发，使用逗号分隔需要切割至不同分组的编译产物类别，使用冒号分隔需要切割至同一分组的编译产物类别。使用星号分隔多个ninja工程的配置。不设置时会自动配置合适值。	nonSystem:fonts:media:usr:system_ext,bin_other1:bin:bin_other2:lib:lib64,apex:system_ext_apex:apex_1:system_ext_apex_1:fake_packages:packaging_script,framework:priv-app:priv-app2:priv-app3:app,host,apex_0:system_ext_apex_0,apex_2:system_ext_apex_2:vendor,product:etc*nonSystem

参数项	说明	示例
TB_TARGETS_LIST_ALL	人工指定分发的target进行编译，每个逗号隔开不同worker的target，每个空格隔开同一个worker的不同target。使用星号分隔多个ninja工程的配置。不设置时会自动配置合适值。	"nonSystem_target,framework_target,lib_target*lib_target"
TB_HOOK_LOCK	对软链接也进行文件同步。若构建过程中发生软链接文件未同步导致的报错，需要开启此选项。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_APPEND_PATH	构建时可向PATH环境变量中追加的参数。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_SHUT_DOWN_SAME_TIME	所有worker都等待主节点执行完毕后再结束构建释放资源。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_RSYNC_LOCK	构建加速的同时worker向构建执行机实时传输文件。开启后效率会进一步提升，但会存在概率性编译失败的情况。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_MAKE_J	设置构建并发数。默认为worker核数。	16
TB_REFER_NINJA_FILE	如果存在串行执行一个一模一样的ninja工程时，可以使用此变量优化构建速度 ● 1：开启。 ● 0：关闭（默认）。	1
CCACHE_DIR	自定义编译缓存的本地目录。默认为/tmp/xcache目录。	${WORKSPACE}/TBcache
TB_CACHE_SIZE	使用自定义执行机时自定义编译缓存的本地目录存储大小上限。默认为100G。	100G
CCACHE_MAXSIZE	自定义编译缓存的本地目录存储大小上限。默认为20G。	100G
TB_ONE_WORKER	使用自定义执行机且只使用一个worker进行构建时可以使用此变量打开编译缓存开关。 ● 1：开启。 ● 0：关闭（默认）。	1

参数项	说明	示例
NT	该变量在主节点client自动设置，可以通过比较此变量是否等于1判断该节点是否为主节点client。	不需要配置。
TB_NET_INTERFACE_NAME	指定获取IP时读取的网卡名，在多网卡情况下获取IP使用，默认为空，多个网卡名通过逗号分隔，配置在前的网卡名有更高的优先级。	eth0,eth1
TB_ACC_PREPARE	使用自定义执行机时必须配置此变量。	false
TB_OUTPUT_PATH	自定义产物目录路径，默认设置为out。	output
TB_SELF_ENV	worker编译target时使用本地环境变量，不使用client传递的变量。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_RSYNC_FLAG	增加同步文件时rsync命令的参数	--ignore-existing -a
TBUILD_PLUGIN_PKG_TYPE	指定使用的TBuild版本为snapshot版本还是release版本。不设置时默认使用最新的release版本。	release
TBUILD_PLUGIN_PKG_VERSION	指定使用的TBuild版本号。不设置时默认使用最新的release版本。	1.0.1

表 5-7 依赖解析模式

参数项	说明	示例
TB_CPU_NUM	分发任务时，所有机器都以此值计算可分配任务量。默认为空。	16
TB_LOCAL_CONTAIN_PATTERN	根据关键字指定必须分发在client编译的target，此target生成的产物不需要文件传输回主节点。默认为空。	file_contexts.bin,com.android.vndk.current.,out/soong/host/linux-x86/bin/fileslist
TB_LOCAL_NOT_CONTAIN_PATTERN	根据关键字指定必须不分发在client编译的target。默认为空。	Bluetooth.so

参数项	说明	示例
TB_CAPACITY_ALL	单台机器可分配的任务权值上限，单位是分钟，以json文件的时间为参考，此时间为单纯的构建时间，不包括cpu空闲时间，实际构建时间会大于此值。可以指定为小数。超过此上限的target会分发至client，在所有agent编译完成后在本地最后编译。不设置时会根据json文件自动设置。	5.5
TB_TASK_SIZE_ALL	人工指定分割多少份target，可以大于机器数量，建议设置的值略大于机器数量，不建议少于机器数量，会导致机器浪费。不设置时会根据机器数量自动设置。	8

表 5-8 ninja 文件缓存选项

参数项	说明	示例
TB_CACHE_SERVER_IP	ninja文件缓存开关，和TB_CACHE_ARCHIVE_PATH同时设置时缓存才会开启。 ● 非空值：开启，如果开启了远端缓存，该值视为远端服务器IP。 ● 空值：关闭（默认）。	172.example.example.example
TB_CACHE_ARCHIVE_PATH	ninja文件缓存本地归档目录，和TB_CACHE_SERVER_IP同时设置时缓存才会开启。	AOSP/ninja_cache
TB_CACHE_RECACHE	本次构建会重新生成ninja文件缓存，不会命中历史缓存。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_CACHE_REMOTE	ninja文件缓存远端开关，命中时从远端获取缓存，生成缓存时也会归档至远端。 ● 1：开启。 ● 0：关闭（默认）。	1
TB_CACHE_LOCAL	ninja文件缓存本地开关，命中时从本地获取缓存，优先级高于远端缓存，生成缓存时也会归档至本地。 ● 1：开启（默认）。 ● 0：关闭。	1

参数项	说明	示例
TB_CACHE_DEPENDS	增加指定的文件作为缓存命中的依赖文件，如果该文件产生变化，会使缓存不命中。默认为空。多个文件使用逗号分隔。	build.sh,test.sh
TB_CACHE_VERSION	为缓存增加指定的版本号，如果版本号产生变化，会使缓存不命中。默认为空。	1.0
TB_CACHE_EXCLUDE_KEY	缓存时过滤掉带关键字的文件，多个关键字用逗号分隔，默认设置为“.glob”，可以用空字符串重置。	.so,.glob
TB_NINJA_FILE_CACHE	设置被缓存的目录，多个目录使用逗号分隔，默认设置为“out”。	out,test
TB_CACHE_NINJA_CACHE_SIZE	ninja文件缓存本地归档目录空间上限，超过此上限会根据算法自动清理历史缓存。默认设置为50，单位是G。	100
TB_SKIP_TARGET	归档ninja文件缓存时跳过此target，即使命中ninja文件缓存此target也会重新编译。	update-api

5.5 代码缓存

概述

代码缓存是指通过一致性HASH、分布式文件存储、增量更新等技术，实现构建时代码下载效率的提升。

说明

- 使用代码缓存能力需要购买配套构建加速包。
- 构建缓存只提供文件缓存的上传和下载检出功能，支持用户自定义脚本更新。
- 该功能目前仅支持北京四区域使用，其余区域后续上线。

前提条件

- 已有可用项目或下载源。
- 已购买配套加速特性包。

配置方法

目前仅支持通过YAML配置使用代码缓存，配置在env中，如何编写YAML文件请参考**单任务YAML文件结构说明**。

```
params: # 构建参数，可在构建过程中引用
- name: CLOUD_BUILD_UPLOAD_FLAG  # 参数为有值和为空两种状态，可控制跳过缓存上传至文件服务器
  value: true
- name: CLOUD_BUILD_REMOTE_CACHE # 参数为有值和为空两种状态，可控制会从其他执行机获取缓存
```

```
  value: true
env:
  cache: # 使用代码缓存
    - type:code #必填，使用缓存开关
      local_path:code # 必填，代码在构建执行机上存放的相对路径
      command:dos2unix build.sh && sh build.sh # 必填
      branch:master # 选填，分支名，可自定义，与url一起确定缓存标签
      url:git@codehub.devcloud.example.example.com:test/python3.git # 选填，可自定义，与branch一起确定
缓存标签
```

表 5-9 配置参数表

参数项	参数类型	描述	是否必填
CLOUD_BUILD_UPLOAD_FLAG	String	用于控制是否跳过缓存上传文件服务器。 True：是。 为空：否。	否
CLOUD_BUILD_REMOTE_CACHE	String	用于控制是否从其他执行机获取缓存。 True：是。 为空：否。	否
type	String	使用缓存开关。	是
local_path	String	代码在构建执行机上存放的相对路径。	是
command	String	运行下载/更新代码的脚本命令。	是
branch	String	分支名，可自定义，与url一起确定缓存标签。	否
url	String	链接，可自定义，与branch一起确定缓存标签。	否

工作模式介绍

1. 代码缓存下载。本地没有缓存的情况下，从服务器下载缓存代码到本地；有缓存的情况下，使用本地缓存并还原目录结构。
2. 代码缓存更新。代码检出有变化时，会增量扫描目录树，缓存差异文件和目录树，加速下次构建缓存效率。

效果示例

在缓存盘有缓存的情况下，200万个文件（190G）还原时间在2到3分钟（仅供参考，实际效率受执行机规格、负载等影响）。

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。
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，黄色为执行中的任务被中止，灰色为任务未被执行。
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，具体操作请参考管理构建任务生命周期）。

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。

7 管理构建任务生命周期

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、代码源、代码仓、默认分支、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

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。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。

操作	说明
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。

复制构建任务

1. 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。
2. 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。

禁用任务

1. 单击构建任务所在行的 ，在下拉列表中选择“禁用”。
2. 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”。

收藏构建任务

1. 鼠标移至任务所在行，单击，图标变色即收藏成功。
2. （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. 单击正在执行的构建任务名称，进入到“构建历史”页面。
2. 单击正在执行的“构建编号”。
3. 单击页面右上角“停止构建”，即可停止构建任务。

8 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**请参考。

表 8-1 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

操作名称	资源类型	事件名称
更新nextfs信息	nextfsInfo	updateNextfsInfo
创建nextfs	nextfsInfo	createNextfsInfo
创建租户关联nextfs	tenantNextfs	createTenantNextfs
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

9 其他相关操作

9.1 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“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。

- 在搜索框输入关键字，可搜索文件。
- 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，在下拉框中选择“删除”，可根据弹框提示确认是否删除。
- 单击操作列，在下拉框中选择“编辑权限”，可在弹出的界面配置用户操作文件的权限。

表 9-1 文件管理角色权限说明

权限类型	拥有该权限的角色
添加用户	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。
编辑权限	文件创建者。

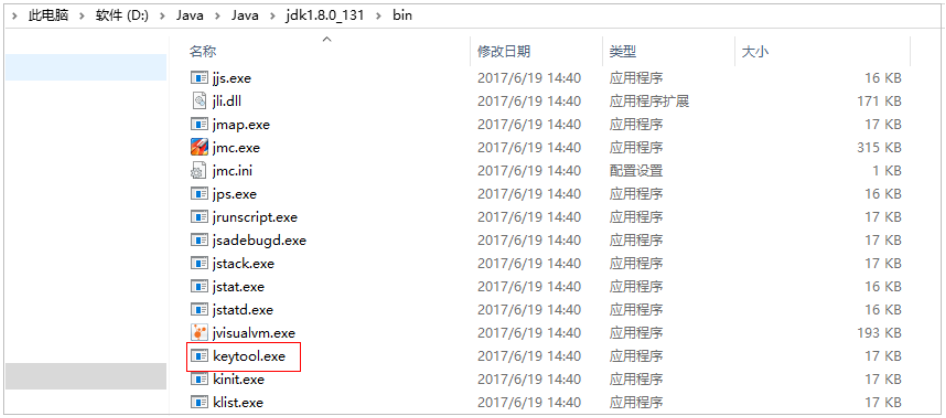
说明

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

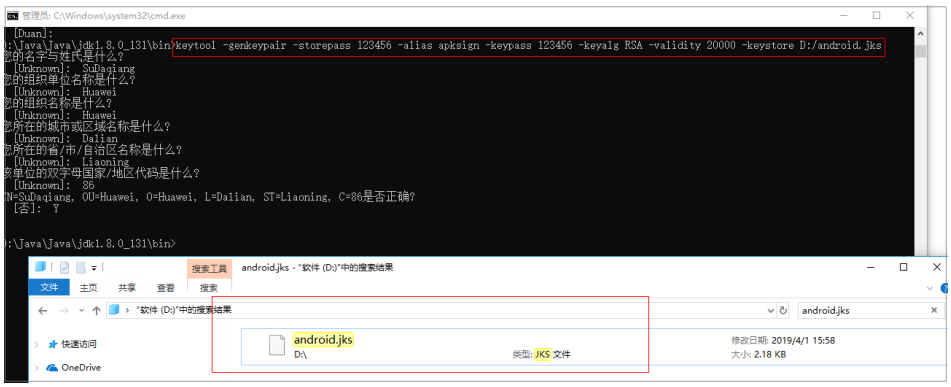
- **使用JDK的keytool工具生成签名文件**

  a. 找到JDK安装位置以及keytool。



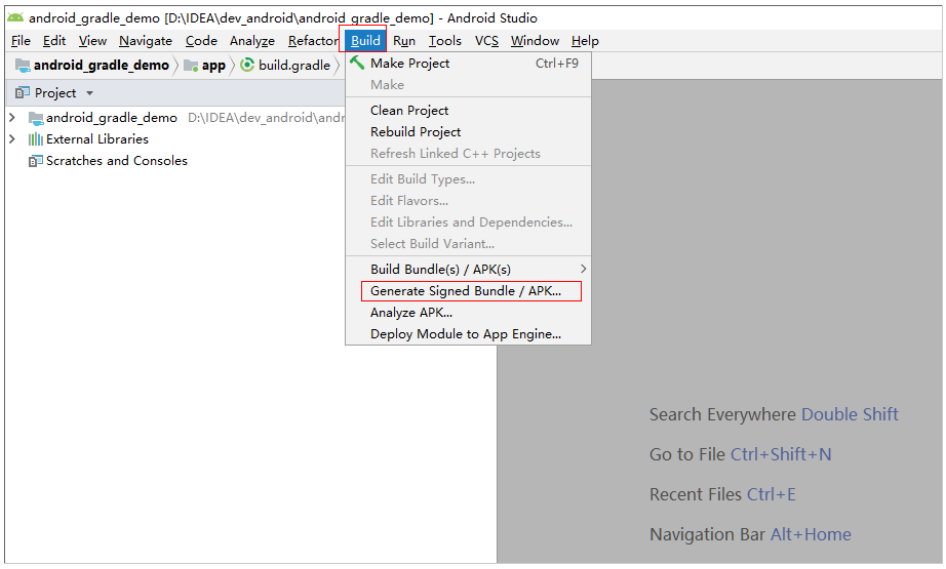
b. 执行生成密钥命令，生成.jks文件。

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



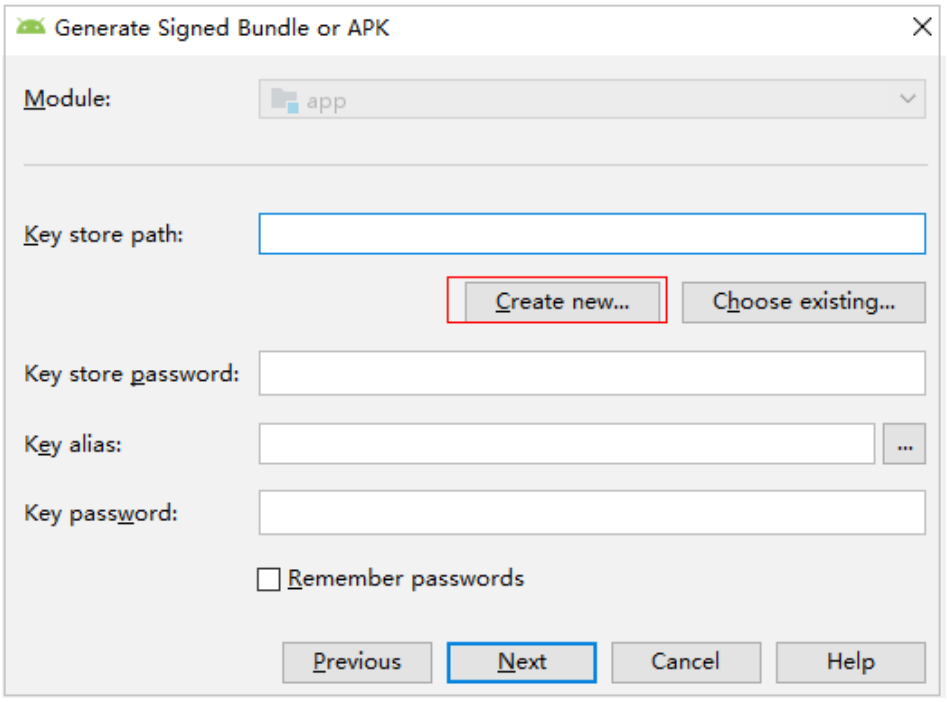
- **使用Android Studio生成签名文件**

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

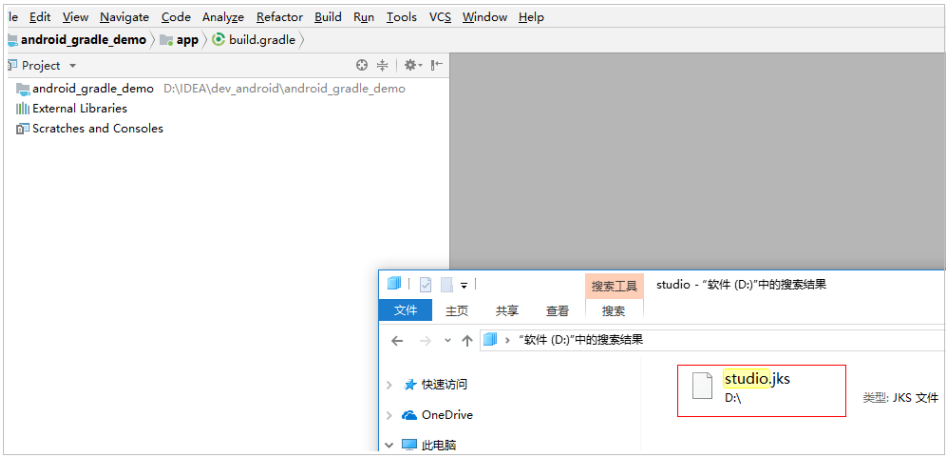


b. 选择“APK”，单击“Next”。

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。



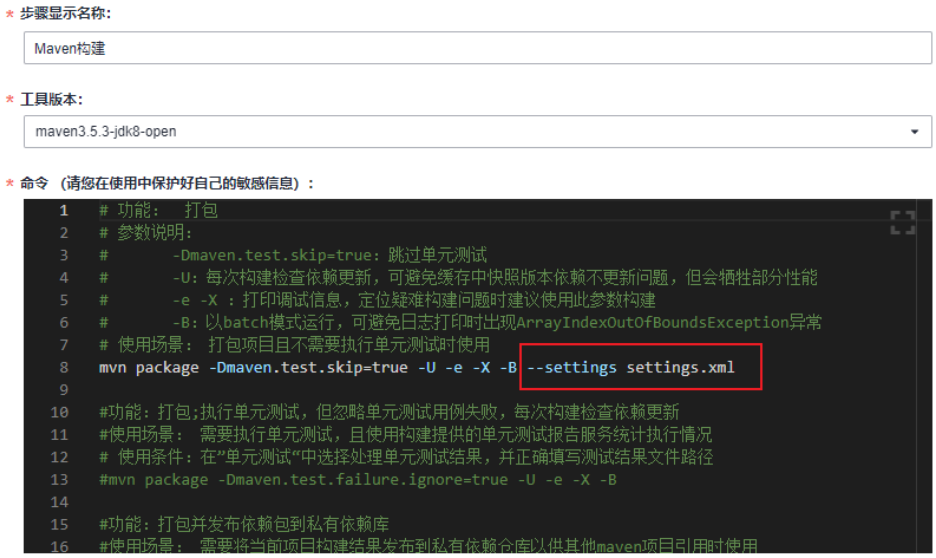
说明

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。



9.2 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

