软件开发生产线 CodeArts

编译构建常见问题

天翼云科技有限公司

目 录

1 公共问题

1.1 执行构建任务时，能否指定在某一台/一种配置的服务器上运行？

否。

目前编译构建服务采取空闲服务器随机分配的方式，暂不支持指定特定机器执行构建任务。

1.2 构建环境中有 Android，对于 IOS 是否有支持计划？

否。

编译构建功能目前不支持 IOS，暂无明确的支持计划。

1.3 单次编译构建上传的构建包是否有容量限制？

有。

基于安全考虑，单次编译构建时，对上传的构建包容量做了限制，具体如下：

上传构建包场景	容量大小
编译构建任务上传到软件发布库	< 5GB
编译构建任务上传到私有依赖库	< 300MB

1.4 构建找不到必须的项目文件

问题现象

使用 Maven 等工具构建时，通常会依赖特定的构建文件，如：pom.xml 文件等。如果工具找不到相应的构建文件，则会失败并报“xxx 工程找不到 xxx 文件”此类错误，常见的错误信息如下：

工具	构建文件	错误信息
Maven	pom.xml	The goal you specified requires a project to execute but there is no POM in this directory (). Please verify you invoked Maven from the correct directory
Ant	build.xml	Buildfile: build.xml does not exist!
NPM	package.json	npm ERR! enoent ENOENT: no such file or directory, open '/package.json'
Yarn	package.json	error Couldn't find a package.json file in ""

原因分析

以上报错可能的原因有：

- 代码工程目录下没有相应的构建文件。
- 代码工程目录存在嵌套，执行构建命令时所在目录没有相应文件。

处理方法

- 检查项目中是否丢失构建工具需要的构建文件。
- 确认构建文件是否位于项目根目录（或是否在构建命令指定的构建文件路径），如有必要先执行“cd”命令进入子目录。

  示例：

  ```
  cd demo-root/demo
  mvn package -Dmaven.test.failure.ignore=true
  ```

1.5 上传软件包时找不到文件

问题现象

构建任务的“上传软件包到软件发布库”步骤中构建包路径填写错误时，在执行任务时会报错，日志中会出现如下错误信息：

```
[ERROR] [上传软件包到软件发布库:external_nexus_artifact_uploader] : 错误信息:
DEV.CB.0220021,未找到文件,可能文件路径不对:**/target/bb.war。
```

原因分析

上传软件包到软件发布库的构建步骤，构建包路径配置错误，导致系统找不到对应的文件。如上配置的路径为“**/target/bb.war”，实际 target 目录下是不存在“bb.war”这个包的。

处理方法

- 确定 target 目录下有 war 包，只是名字可能不是“bb.war”。

  这种情况下修改构建包路径为“**/target/*.war”，正则匹配 war 包。
- 无法确定 target 目录下有哪些文件。

  在构建执行的步骤 shell 里最后增加“ls -al target”，再次执行构建，就会打印出 target 目录下的所有文件。找到需要的文件位置后，再重写构建包路径配置。

说明

- 如果构建结果不在 workspace 目录（构建命令在 workspace 目录或其子目录下执行），则在下一个 Action 中将丢失此构建包，因此需要提前拷贝构建包到 workspace 目录，如： mv /usr/bin/nginx ./。

1.6 权限不足，无法获取信息

问题现象

执行编译构建任务失败，异常信息为：权限不足，无法获取信息。

原因分析

用户不知道自己的角色或者角色被修改时，导致执行编译构建的权限不足，无法操作该任务。

处理方法

步骤 1 联系任务的管理员（任务创建者、项目创建者）配置任务的操作权限。

步骤 2 进入任务的“权限管理”页面，开启对应操作权限。

----结束

1.7 任务不存在

问题现象

执行流水线失败，流水线上挂载的构建任务报错，异常信息为：任务不存在。

原因分析

该报错构建任务被删除，导致流水线执行失败。

处理方法

步骤 1 检查该任务是否被人为删除，且不可以从用户侧恢复。

步骤 2 尝试重新配置构建任务和流水线。

步骤 3 如果仍然未能解决，请联系技术支持工程师。

----结束

1.8 任务执行中止

问题现象

构建任务被中止，异常信息如下：

原因分析

编译构建单个构建任务单次构建最大时长限制为：1 小时（非付费用户）/4 小时（付费用户），构建时长如果超过了系统限定值，系统会强制中止任务执行。

1.9 Eclipse 普通 Java 项目上云

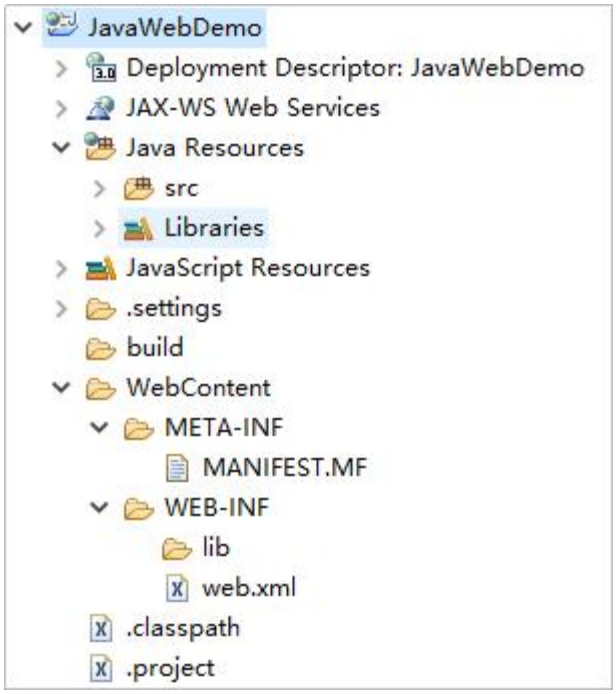
背景信息

Eclipse 开发的 Java web 项目无法在软开云上构建出包，需要转换项目。本文档将指导您如何将项目改造成 Ant 项目，在软开云上使用 Ant 工具进行构建出包。

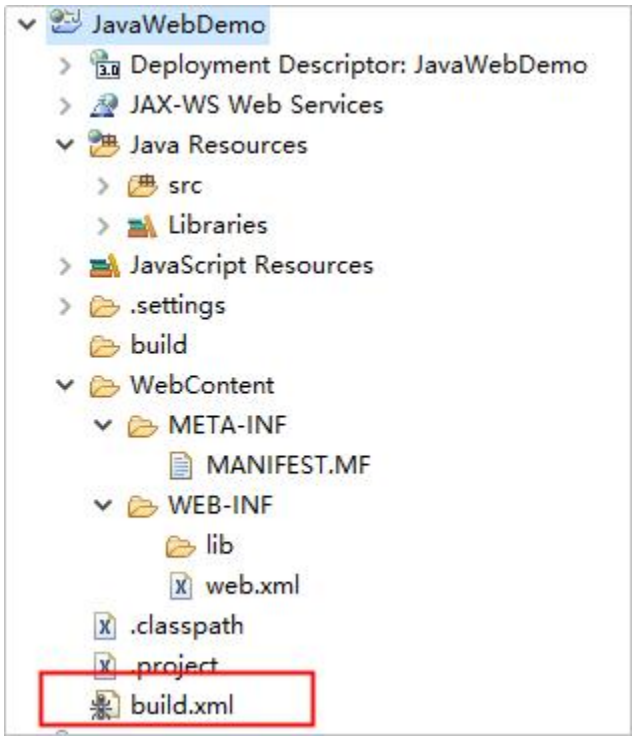
操作步骤

步骤 1 在项目中创建一个 build.xml 文件。

下图为一个 Eclipse 创建出来的 web 项目的 demo 的目录结构。



在根目录创建 build.xml 文件，目录结构变为下图。



下面是更加详细的说明，说明后有一个完整的 build.xml 例子，只要将例子中的各个属性修改成真正的项目对应的内容即可。

1. **定义属性部分**

   - 定义项目名称

```
<property name="project.name" value="JavaWebTest" />
```

- 定义包名：打包时生成的.war 文件名，这里用到项目名的定义 project.name，生成的 war 包名就是项目名.war。

```
<property name="package.name" value="${project.name}.war" />
```

- war 包输出路径：上传软件包时以此路径+包名作为构建包路径,value 值就是路径，这里可以自己定义。

```
<property name="dist.war.dir" value="./targets" />
```

注意：如果“./targets”目录不存在，请在 init 步骤中创建。

- 源代码(*.java)路径，这里的 value 值指向项目 java 代码存放的路径。

```
<property name="src.dir" value="src" />
```

- 源码中 WebContent 目录路径，这里的 value 指向项目 WebContent 代码存放的路径。

```
<property name="webcontent.dir" value="./WebContent" />
```

- 源码中 WEB-INF 目录路径，这里的 value 指向项目 WEB-INF 代码存放的路径，通常都是在 WebContent 目录下，所以引用上面 WebContent 的路径。

```
<property name="webcontent.webinf.dir" value="${webcontent.dir}/WEB-INF"
/>
```

- class 文件输出目录：通常编译后的 class 文件放在 WEB-INF 下。

```
<property name="webcontent.webinf.classes.dir"
value="${webcontent.webinf.dir}/classes" />
```

注意：如果 classes 目录不存在，请在 init 步骤中创建。

- 定义 lib 路径，这里是你引用的依赖包的存放路径，通常在 WEB-INF 下。

```
<property name="webcontent.webinf.lib.dir"
value="${webcontent.webinf.dir}/lib" />
```

注意：如果 lib 目录不存在，请在 init 步骤中创建。

- 定义 java 版本。

```
<property name="source.version" value="1.8" />
<property name="target.version" value="1.8" />
```

- 定义 web.xml 路径，不是 web 项目可以不用定义。

```
<property name="webxml.path" value="${webcontent.webinf.dir}/web.xml" />
```

2. **定义路径部分**

- 定义 classpath 路径，如 A 引用 B，A.java 的编译在这里寻找 B.class 文件。

```
<path id="classpath">
    <!-- 项目的 jar 包 -->
    <fileset dir="${webcontent.webinf.lib.dir}">
        <include name="**/*.jar" />
    </fileset>

    <!-- 项目的 classes 文件 -->
    <pathelement location="${webcontent.webinf.classes.dir} " />

    <!--这里的需要用到的 web 服务器的包，可自行下载添加-->
    <!-- web 服务器的 jar 包 -->
    <!-- <fileset dir="${localWebServer.home}/lib">
        <include name="**/*.jar" />
```

```
    </fileset>    -->
</path>
```

3. **定义构建过程**

- 初始化步骤 （init），包含清空 war 包输出目录、创建 classes 路径等步骤。

属性	描述
<delete>标签	删除动作，dir 属性就是要删除的目录路径
<mkdir>标签	创建目录动作，dir 属性就是要创建的目录路径
<echo>标签	打印动作，message 属性就是要打印的内容

上面提到的 war 包目录（dist.war.dir），编译存放 class 文件目录（webcontent.webinf.classes.dir），存放依赖包的 lib 目录（webcontent.webinf.lib.dir），如果原来项目中没有，就要在这一步创建。

```
<target name="init">
    <echo message="删除 targets 目录(war 包输出目录)" />
    <delete dir="${dist.war.dir}" />
    <echo message="创建 targets 目录(war 包输出目录)" />
    <mkdir dir="${dist.war.dir}" />
    <echo message="创建 classes 目录" />
    <!-- WebContent 下的 classes -->
    <mkdir dir="${webcontent.webinf.classes.dir}" />
</target>
```

- 编译 java 文件，使用<javac>标签将 java 文件编译到“dist.classes”下，编译步骤依赖于 init 步骤创建的 classes 目录。

属性	描述
depends	depends="init"声明当前步骤需要在 init 步骤后使用
srcdir	srcdir 属性指定上面定义好的 java 代码的路径属性 src.dir
destdir	destdir 属性指定定义的存放编译完的 class 文件的 classes 目录
source，target	指定编译时使用的 jdk 版本

```
<target name="compile" depends="init">
    <echo message="使用指定的 classpath 编译源代码,输出到 classes 目录" />
    <!-- 这里指定 jdk 版本为 1.8 -->
    <javac encoding="utf-8" listfiles="true" srcdir="${src.dir}"
        destdir="${webcontent.webinf.classes.dir}" debug="on"
deprecation="false"
        optimize="true" failonerror="true" source="${source.version}"
target="${target.version}">
        <classpath refid="classpath" />
    </javac>
</target>
```

- 将编译完的项目打成 war 包，首先使用<delete>标签清理掉原有的 war 包，再使用<war>标签打包。

属性	描述
warfile	该属性定义打出来的 war 包的包名，包含路径
webxml	指定 web.xml 的路径
<fileset>子标签	指定 webContent 路径

```
<target name="war" depends="compile" description="将工程打成 war 包">
    <echo message="生成 war 包" />
    <delete file="${dist.war.dir}/${package.name}" />
    <war warfile="${dist.war.dir}/${package.name}" webxml="${webxml.path}">
       <fileset dir="${webcontent.dir}">
       </fileset>
    </war>
</target>
```

说明

如果是要打 jar 包，这里就不能用<war>标签，而是用<jar>标签，示例如下：

- jarfile 属性和 war 包的属性类似，是存放打好的 jar 包存放的路径，需要在上面属性定义阶段定义，basedir 属性是编译后的 class 的目录，就是上面的打 war 包时定义的 webcontent.webinf.classes.dir 属性。
- jar 包就不需要 webxml 属性了。
- 如果要打的 jar 包是需要使用 java -jar 来执行的可执行 jar 包，则需要定义 manifest，如果只是一个功能性的，被依赖的 jar 包就不需要了。

  Main-Class 指定 main 函数所在的类。

```
<target name="jar" depends="compile" description="make jar file">
  <!--jar 操作，jarfile 指定 jar 包存放路径，basedir 为编译后的 class 的目录-->
  <jar jarfile="${jarfilename}" basedir="${classes}">
       <!--为 jar 包指定 manifest，当然，如果 jar 包不需要打成 runnable 的形式，manifest 可以不要-->
       <manifest>
            <!--指定 main-class-->
            <attribute name="Main-Class" value="demo.SayHello" />
       </manifest>
  </jar>
</target>
```

下面是完整的 build.xml 示例，供参考，通常只需要把属性定义阶段里面那些路径按照实际项目的路径填充好就可以。

```
<?xml version="1.0"?>
<project default="war" basedir=".">
      <echo message="pulling in property files" /> <property
file="build.properties" />
      <property name="project.name" value="JavaWebDemo" />
      <property name="package.name" value="${project.name}.war" />
      <property name="dist.war.dir" value="./targets" />
      <property name="src.dir" value="src" />
```

```
      <property name="webcontent.dir" value="./WebContent" />
      <property name="webcontent.webinf.dir" value="${webcontent.dir}/WEB-INF" />
      <property name="webcontent.webinf.classes.dir"
value="${webcontent.webinf.dir}/classes" />
      <property name="webcontent.webinf.lib.dir"
value="${webcontent.webinf.dir}/lib" />
      <property name="source.version" value="1.8" />
      <property name="target.version" value="1.8" />
      <property name="webxml.path" value="${webcontent.webinf.dir}/web.xml" />

      <path id="classpath">
        <fileset dir="${webcontent.webinf.lib.dir}">
            <include name="**/*.jar" />
        </fileset>
        <pathelement location="${webcontent.webinf.classes.dir} " />
    </path>

     <target name="init">
         <echo message="删除 targets 目录(war 包输出目录)" />
         <delete dir="${dist.war.dir}" />
         <echo message="创建 targets 目录(war 包输出目录)" />
         <mkdir dir="${dist.war.dir}" />
         <echo message="创建 classes 目录" />
         <mkdir dir="${webcontent.webinf.classes.dir}" />
     </target>


    <target name="compile" depends="init">
        <echo message="使用指定的 classpath 编译源代码,输出到 classes 目录" />
        <javac encoding="utf-8" listfiles="true" srcdir="${src.dir}"
            destdir="${webcontent.webinf.classes.dir}" debug="on"
deprecation="false"
            optimize="true" failonerror="true" source="${source.version}"
target="${target.version}">
            <classpath refid="classpath" />
        </javac>
    </target>

     <target name="war" depends="compile" description="将工程打成 war 包">
         <echo message="生成 war 包" />
         <delete file="${dist.war.dir}/${package.name}" />
         <war warfile="${dist.war.dir}/${package.name}" webxml="${webxml.path}">
             <fileset dir="${webcontent.dir}">
             </fileset>
         </war>
     </target>
</project>
```

步骤 2 将改好的 build.xml 提交到代码仓库，创建 Ant 类型的构建任务。

上传软件包到软件发布库中的构建包路径就可以按照上面 build.xml 说明的那样填写 war 包输出路径加上包名的格式。

步骤 3 保存任务，执行构建，构建成功之后就可以在软件发布库看到编译打包好的 war 包。

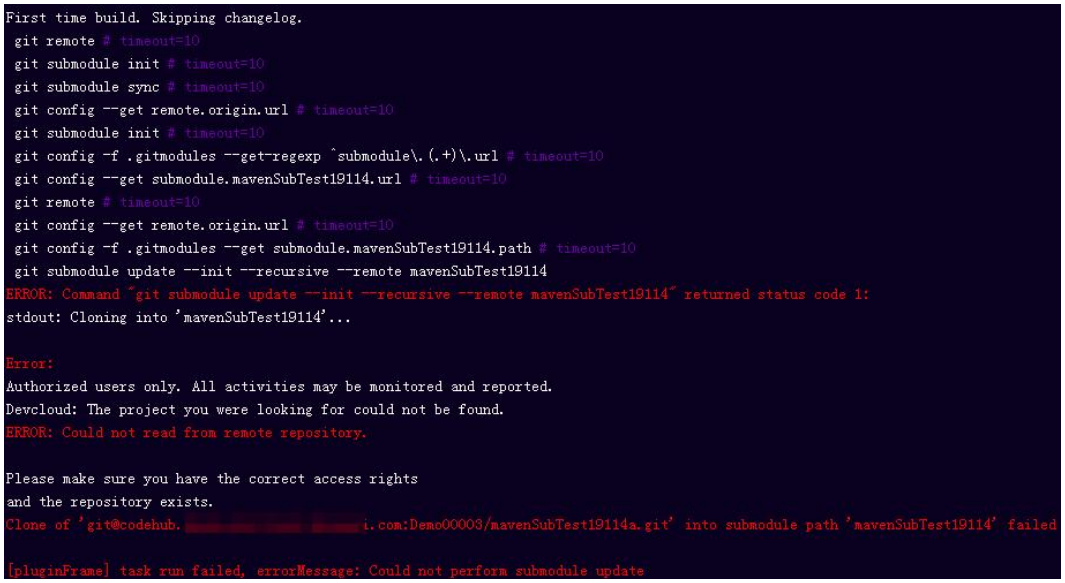
----结束

2 拉取代码

2.1 拉取子模块代码出错

问题现象

执行构建任务时，报如下异常信息：



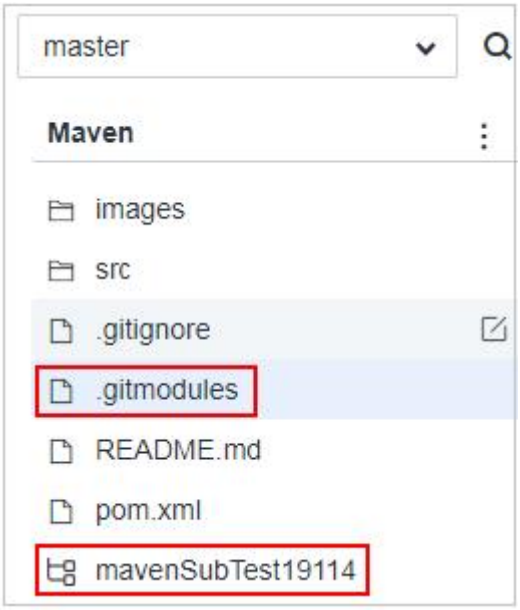
原因分析

Git 从 CodeArts Repo 拉取子模块时出现错误“Could not read from remote repository”，可能是没有权限或者“.gitmodules”文件配置错误。

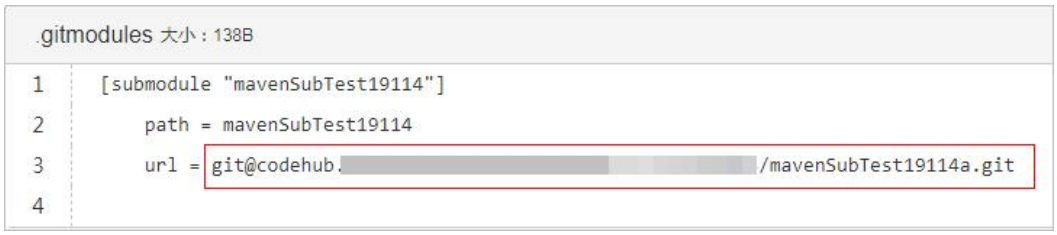
处理办法

步骤 1 打开主代码仓库，选择“设置 > 子模块设置”，部署秘钥没有同步，单击同步按钮，之后再尝试编译构建。

步骤 2 如果步骤 1 已同步，很可能是主仓库的“.gitmodules”文件配置出错，先检查存在“.gitmodules”文件且子模块是“mavenSubTest19114”。



步骤 3 打开“.gitmodules”文件，修改成正确的子模块配置“mavenSubTest19114a.git”。



步骤 4 重新修改好“.gitmodules”，再尝试构建。

----结束

2.2 Git 拉取子模组失败，找不到子模组的修订版本

问题现象

异常信息如下：

```
[2019-07-02 08:29:23.179] ERROR: Command "git submodule update --init --recursive --remote asae-feign" returned status code 1:
[2019-07-02 08:29:23.179] stdout: Cloning into 'asae-feign'...
[2019-07-02 08:29:23.179]
[2019-07-02 08:29:23.179] Error: ERROR: Needed a single revision
[2019-07-02 08:29:23.179] Unable to find current origin/develop revision in submodule path 'asae-feign'
[2019-07-02 08:29:23.179]
[2019-07-02 08:29:23.202] [INTERNAL]  : [pluginFrame] step run failed, errorMessage: Could not perform submodule update
[2019-07-02 08:29:23.250] [INFO] [代码检出] : StagePostExecution started
[2019-07-02 08:29:23.251] [INFO] [代码检出] : StagePostExecution finished
```

原因分析

原先检出的目录有问题，本例的目录为“asae-feign”，这个问题属于 Git 本身的 bug。

处理办法

在代码仓库删除该目录，然后重新执行 **git submodule update**，然后重新执行构建任务。

2.3 Git 不拉取子模块

问题现象

构建拉取 Repo 代码时，存在“.gitmodules”文件且确认配置正确，但是没有去拉取子模块。

原因分析

此问题一般为没开启子模块自动更新。

处理办法

编辑构建任务，选择“代码下载配置”构建步骤，将“子模块（submodules）自动更新”开关打开。

3 Maven 构建

3.1 license 信息检查不通过

问题现象

异常信息如下：

```
[ERROR] Failed to execute goal org.apache.rat:apache-rat-plugin:0.12:check (rat-check) on project maven: Too many files with unapproved license: 7 See RAT report in: /xxx/slave1/workspace/job_4f1501a3-542c-4f3d-a2bb-8fdbd4d76678_1534924094266/target/rat.txt -&gt; [Help 1]
```

原因分析

文件 License 信息检查不通过。

处理方法

在 mvn 命令中添加参数：

```
apache-rat:check -Drat.numUnapprovedLicenses=600
```

3.2 使用 maven deploy 命令上传包失败

问题现象

通过执行 Maven 构建任务上传依赖到私有依赖库时，执行任务时日志报如下异常信息：

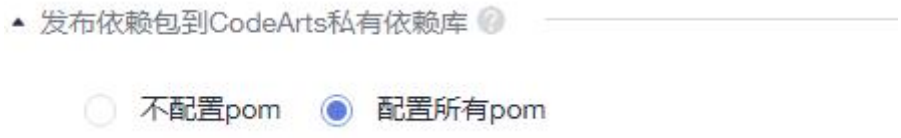
```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-deploy-plugin:2.7:deploy (default-deploy) on project javaMavenDemo: Deployment failed: repository element was not specified in the POM inside distributionManagement element or in -DaltDeploymentRepository=id::layout::url parameter -&gt; [Help 1]
```

原因分析

“pom.xml”文件没有正确配置“distributionManagement”。

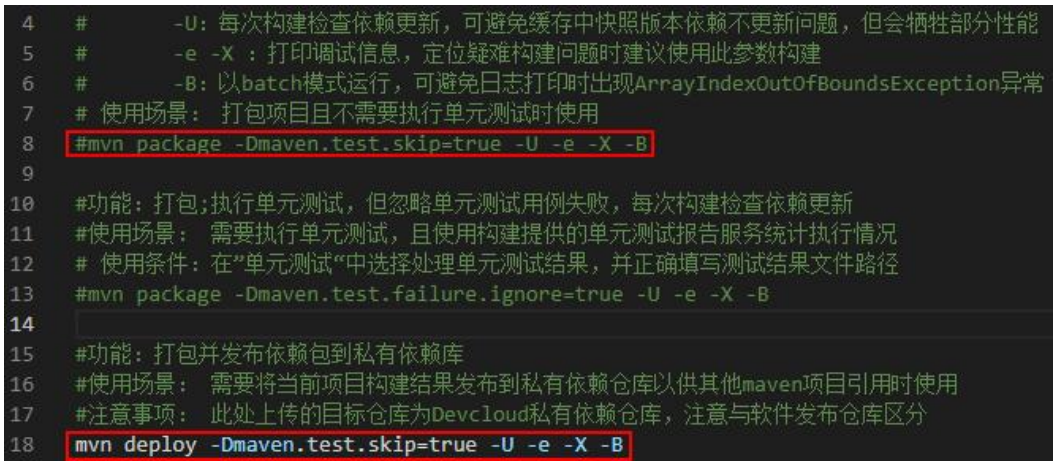
处理方法

步骤 1 配置“Maven 构建”构建步骤，展开“发布依赖包到 CodeArts 私有依赖库”，选择“配置所有 pom”。



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

步骤 2 在命令窗口，使用“#”注释掉第 8 行的默认命令，并删除第 18 行命令前的“#”。



步骤 3 配置完成后执行构建任务。执行成功后即可将依赖包发布到私有依赖库。

----结束

3.3 找不到 pom 文件

问题现象

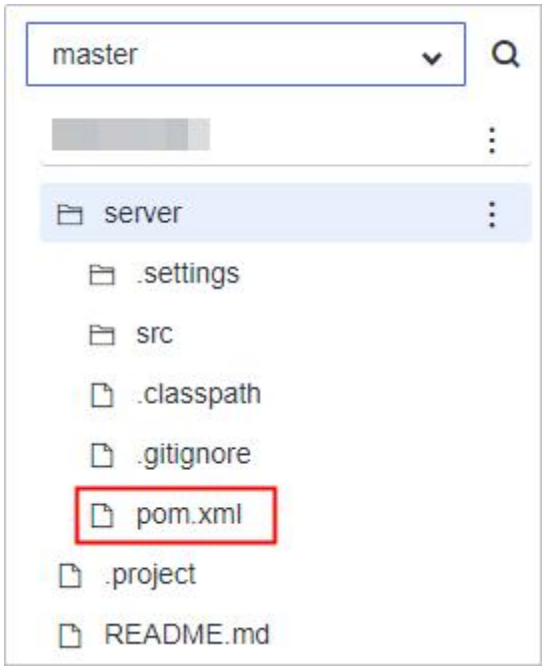
异常信息如下：

```
[ERROR] The goal you specified requires a project to execute but there is no POM in this directory (/xxx/slavespace/slave3/workspace/job_4a1d5be4-b273-4ac8-8d5d-2ee583e71832_1544498089095). Please verify you invoked Maven from the correct directory.
```

原因分析

异常信息显示找不到 POM 文件。系统默认的编译命令是找源码根目录下的 POM 文件，这个错误就是说源码根目录下不存在 POM 文件。

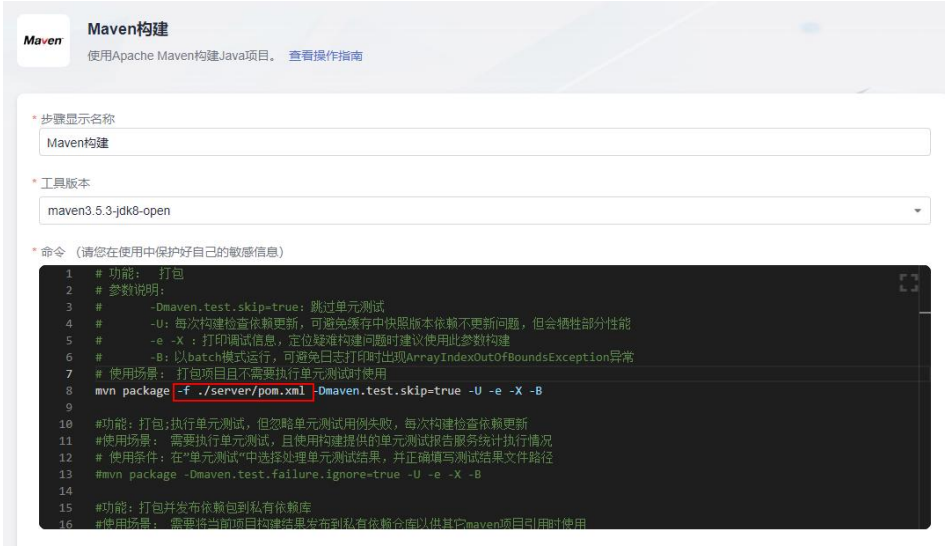
例如：下图中源码根目录下显然不存在 POM 文件的，进入 server 目录下才发现 POM 文件。



处理方法

这种情况下，需要修改系统默认构建命令。以上面的源码结构为例，解决办法两种，两者选其一即可：

- 先执行 **cd server** 进入 server 目录，然后执行 **mvn** 编译命令。
- 在 maven 编译命令后增加**-f ./server/pom.xml**，指定 pom 文件的路径。



3.4 找不到 package/symbol

问题现象

执行 Maven 构建任务时，日志报异常信息提示找不到 package 或 symbol，例如：

```
com/xxx/xxx/configserver/encryptor/xxx.java:[11,40] package
com.sun.jersey.api.client.config does not exist
```

原因分析

分析日志可知，项目中引用了“com.sun.jersey.api.client.config”包下面的内容，但构建时无法从项目中以及所有解析出的依赖包中找到此包导致。导致此结果的原因一般有两大类：

- 代码问题：代码中包引用不正确，此类问题较易排查，如有遇到可优先排查代码。
- 环境/组件问题：依赖包损坏或不一致，此类问题常表现为本地可编译而云端构建失败；此章节主要为此类问题提供一些可能的解决方案。其中可能的环境/组件问题有：
  - 依赖包冲突
  - 依赖范围错误
  - 使用 GAV 模式上传依赖包
  - 依赖包损坏
  - 其他

依赖包冲突

部分场景下，因为操作失误或一些第三方依赖被动引入，项目中可能同时存在同一依赖的多个版本。同时引入不同版本可能会导致实际使用的版本与预期不符，进而导致找不到指定的包。处理此类问题操作步骤：

步骤 1 确认使用的依赖包版本，有两种方式可参考。

1. 参考 Maven 定义的两个依赖调解原则：
   - 第一原则：路径最近者优先， 如：A->B->C->X(1.0)、A->D->X(2.0)，则最终引用 X 版本为 2.0。
   - 第二原则：第一声明者优先， 两个引入路径长度相同时，先引入的版本为最终版本。
2. 使用 dependency 插件，在构建任务执行 **mvn dependency:tree** 查看。

步骤 2 如果确定构建使用的依赖与预期不符，导致构建失败，在 pom 最外层引入需要的依赖并重试：

```
<dependencys>
    <dependency>
        <groupId></groupId>
        <artifactId></artifactId>
        <version></version>
```

```
    </dependency>
</dependencys>
```

----结束

依赖范围错误

使用 Maven 管理依赖时，Maven 坐标中 scope 属性定义了依赖的有效范围。错误地指定依赖范围会导致该依赖在 compile 时无效，如果此时在项目主代码中使用了此依赖中的包，则会导致编译错误。处理步骤：

步骤 1 使用 **mvn dependency:tree** 查看项目使用的依赖及依赖范围。

步骤 2 对比依赖范围以及项目中使用依赖的位置。

若在主代码中使用了依赖中的包，且要求依赖在编译时有效，则依赖的范围需要为以下之一：

- compile
- provided
- system：系统依赖范围必须通过 systemPath 指定依赖文件位置，且依赖文件必须存在于指定目录。

----结束

使用 GAV 模式上传依赖包

- 在私有依赖仓库上传自研依赖包 A 时，如果选择 GAV 模式，只需要上传 jar 文件，系统会自动生成对应的 pom 文件；但是，此 pom 文件中只包含此依赖本身的坐标定义，原来的**<dependencies>**节点则会全部丢失。
- 假设当前构建项目 D，使用了项目 A 构建的依赖 A，依赖 A 引入了第三方依赖 B（D > A > B），此时，在构建 D 项目时，因为以上原因，Maven 解析依赖 A 时，无法识别引入的依赖 B，最终导致项目 D 中找不到依赖 B 的内容，遇到此场景时，可尝试按以下步骤排查：

步骤 1 查看项目 D 的依赖树，确定缺失的内容是否由项目 A 的 pom 文件引入，如果是则进入下一步，否则请尝试其他解决方案。

步骤 2 从私有依赖仓库下载依赖 A 的 pom 文件，与项目 A 中 pom 对比，如果线上 pom 缺失了 B 依赖的引入，进入下一步，否则请尝试其他解决方案。

步骤 3 更新依赖 A 的版本号并重新上传，此处提供两种解决方案：

- 使用编译构建服务构建项目 A，使用 deploy 命令将依赖 A 上传到 Maven 私有仓库（推荐：可集成于流水线中实现自动化）。
- 在 Maven 私有仓库重新上传依赖 A，此时选择 POM 模式，分别上传 jar 文件和 pom 文件。

步骤 4 更新完成后，尝试重新构建项目 D。

----结束

依赖包损坏

依赖包损坏可能会导致依赖包中某些文件缺失，此时构建可以找到相应依赖包，但无法找到其中的 class 文件或者 package，导致此类问题。此场景下可按包类型分不同方式处理：

- 第三方依赖包：直接联系技术支持处理。
- 自研（手动上传到 Maven 私有仓库）的依赖包，按如下步骤排查：
  a. 从 Maven 私有依赖仓库下载依赖包。
  b. 解压缩并查看依赖包内容是否正常。
  c. 若依赖包内容异常，再分两种情况排查：
     - 如果是第三方提供的包手动上传到 maven 私有依赖仓库，确认包文件无误并尝试重新上传（注意同时上传 pom 与 jar 文件）。
     - 如果是自己构建（本地/云端构建）的依赖包，且已确认代码无误，则检查是否是多任务同时构建导致构建生成 jar 包内容缺失。

3.5 多任务同时构建导致构建生成 jar 包内容缺失

问题现象

构建环境异常或不适当的构建方式可能会导致生成的 jar 包内容有缺失，但构建结果是成功，导致问题难以定位。

- 前置条件：A 项目依赖 B 项目，同时构建并上传依赖 A 和依赖 B（多人同时构建或流水线设置构建任务并行执行）
- 构建结果：构建任务 B 结果为成功，构建任务 A 结果为成功
- 问题描述：依赖 B 无异常，依赖 A 偶现内容缺失

原因分析

A 依赖 B 且 A、B 项目同时构建时，可能出现 B 正在上传且未上传完时，A 开始下载 B 依赖，导致 A 项目无法完整获取依赖 B 内容。

处理办法

步骤 1 确定 A 项目所有依赖的自研项目 B1、B2……Bn。

步骤 2 排查相关流水线，确认是否有项目 A 与项目 Bn 并行构建。

步骤 3 如果找到，修改流水线配置，将 A、B 项目构建方式改为串行。

步骤 4 如果没有，对比 A、B 项目构建历史，或与相关责任人确认构建时间，确认是否同时构建。

----结束

3.6 多个子项目和父项目之间的引用问题

问题现象

Maven 构建任务，pom 文件存在多个子项目和父项目之间的引用，在执行任务时，日志报如下异常信息：

```
[ERROR] Project 'xxx.xxx:xxx1:1.0-SNAPSHOT' is duplicated in the reactor @
[2022-03-02 14:02:52.656] [ERROR] Project 'xxx.xxx:xxx2:1.0-SNAPSHOT' is duplicated
in the reactor -> [Help 1]
[2022-03-02 14:02:52.656] [ERROR]
[2022-03-02 14:02:52.656] [ERROR] To see the full stack trace of the errors, re-run
Maven with the -e switch.
[2022-03-02 14:02:52.656] [ERROR] Re-run Maven using the -X switch to enable full
debug logging.
```

原因分析

在 Maven 中，parent 模块组织好 childA 和 childB，叫做"聚合"。多个模块联合编译实现起来很简单，按照以下方式即可：

1. 在 parent 的 pom 文件里加入以下内容：

   ```
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.demo</groupId>
   <artifactId>parent</artifactId>
   <version>1.0</version>
   <modules>
     <module>childA</module>
     <module>childB</module>
   </modules>
   ```

2. 在 childA 和 childB 的 pom 文件中添加相应的标签来标记父模块。
   - childA：

     ```
     <modelVersion>4.0.0</modelVersion>
     <groupId>com.demo</groupId>
     <artifactId>childA</artifactId>
     <version>1.0</version>
     <parent>
       <groupId>com.demo</groupId>
       <artifactId>parent</artifactId>
       <version>1.0</version>
     </parent>
     ```

   - childB：

     ```
     <modelVersion>4.0.0</modelVersion>
     <groupId>com.demo</groupId>
     <artifactId>childB</artifactId>
     <version>1.0</version>
     <parent>
       <groupId>com.demo</groupId>
       <artifactId>parent</artifactId>
       <version>1.0</version>
     </parent>
     ```

在上述的配置形式中指定了一个父项目，下面有两个同级的子项目 A 和 B，如果 A 项目的 pom 文件中把 B 项目当做自己的子项目来引用或者把 parent 项目作为子项目就会引起冲突，构建时就是出现上面的报错。

处理办法

检查项目的 pom 的引用情况 ，如果要 B 项目作为 A 的子项目，则需要从 parent 的 pom 中把 B 项目的引用去掉，把 B 项目的父标签指向 A 项目，如下：

- parent 项目：

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.demo</groupId>
<artifactId>parent</artifactId>
<version>1.0</version>
<modules>
  <module>childA</module>
</modules>
```

- A 项目：

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.demo</groupId>
<artifactId>childA</artifactId>
<version>1.0</version>
<parent>
  <groupId>com.demo</groupId>
  <artifactId>parent</artifactId>
  <version>1.0</version>
</parent>
<modules>
  <module>childB</module>
</modules>
```

- B 项目：

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.demo</groupId>
<artifactId>childA</artifactId>
<version>1.0</version>
<parent>
  <groupId>com.demo</groupId>
  <artifactId>childA</artifactId>
  <version>1.0</version>
</parent>
```

3.7 Maven 构建缓存配置及清理步骤

编译构建提供了构建缓存功能，构建时可将依赖缓存于用户私有存储空间，下次构建时直接使用，无需重复下载，可极大提高构建效率。

构建缓存配置

新建编译构建任务时，默认选择使用缓存加速构建，用户可以在配置“Maven 构建”步骤时选择是否使用缓存。

缓存清理步骤

由于网络抖动、并发构建或其他极端情况，可能出现缓存内容异常导致构建异常，本章节介绍异常缓存的清理过程。

须知

执行缓存清理操作前，请务必仔细阅读以下缓存清理风险以及注意事项：

- 由于缓存目录为同租户共享，频繁清理缓存会概率性导致同租户用户构建异常（常表现为“xxx 文件不存在”），故只可在缓存异常时清理，任务执行成功后务必再次编辑任务，删除清理命令，并且在执行清理缓存操作的同时，不要执行其他的使用缓存的编译构建任务。
- 清理缓存时需要使用精确的文件路径，如：清理 XXX 厂商 demo 1.0.0 版本，请使用命令 rm -rf /path/com/*xxx*/demo/1.0.0。尽量避免删除目录层级过高，导致下次构建缓慢或因网络问题导致依赖异常。
- 出于安全考虑，缓存清理命令只可在“Maven 构建”步骤执行，在其他步骤执行此命令会导致“目录不存在”或”清理无效”等报错。

步骤 1 单击构建任务列表操作列，进入“编译构建编辑”页面。

步骤 2 选择“构建步骤 > Maven 构建”，找到命令行 **mvn xxxx**。

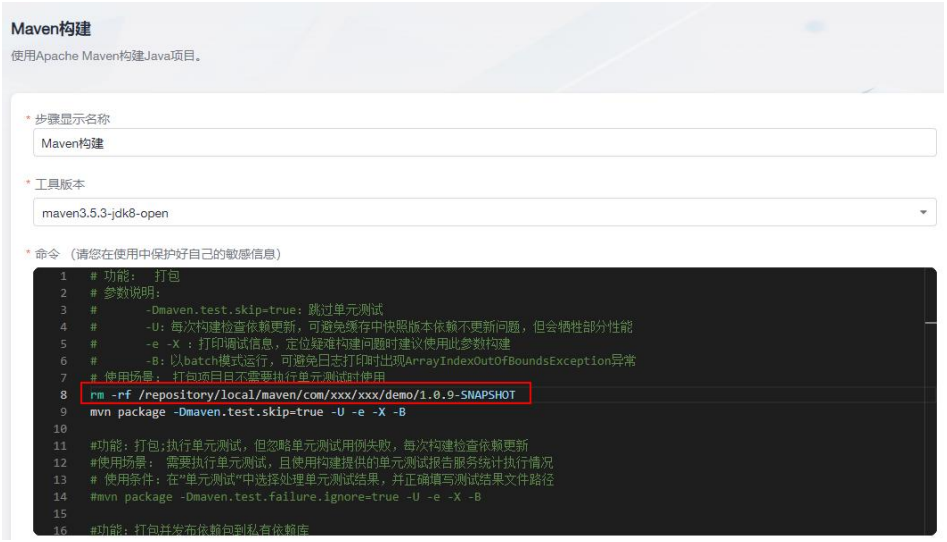
步骤 3 在命令行“mvn xxx”前输入缓存清理命令，单击“保存”。

缓存清理命令格式为：rm -rf /repository/local/maven/{groupId}/{artifactId}/{version}，需填入的参数分别对应依赖包坐标中的 groupId、artifactId、version，其中，groupId 中的点会被自动分割为层级目录。

若依赖包如下：

```
<dependency>
<groupId>com.xxx.xxx</groupId>
<artifactId>demo</artifactId>
<version>1.0.9-SNAPSHOT</version>
</dependency>
```

则清理该依赖包所需命令为：rm -rf /repository/local/maven/com/xxx/xxx/demo/1.0.9-SNAPSHOT。



步骤 4 重新执行构建任务，执行成功后按照上面步骤再次编辑任务，移除清理缓存命令。

----结束

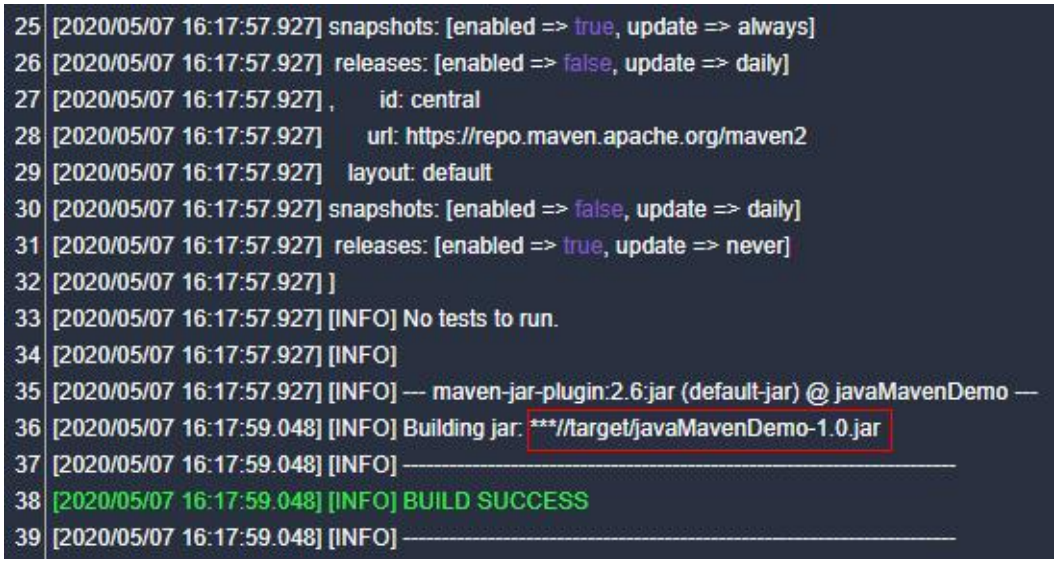
3.8 查找准确的构建包路径

步骤 1 新建 Maven 构建任务，在“Maven 构建”构建步骤后增加“上传软件包到软件发布库”构建步骤。

步骤 2 配置构建包路径，填写任意路径并保存。

步骤 3 执行构建任务，在日志中找到 BUILD SUCCESS 信息。

往上几行找到形如“/target/****.war”的信息，即为准确的构建包路径。



----结束

3.9 使用 jib-maven-plugin 插件构建 maven 工程制作镜像

背景说明

由于软开云官方提供的 maven 镜像中没有 docker 环境，所以，使用 docker-maven-plugin 插件构建的项目通过软开云的编译构建制作镜像时会报错，比如：

```
INFO: I/O exception (java.io.IOException) caught when processing request to {}->unix://localhost:80: No such file or directory
```

本文将指导用户在没有 docker 的 maven 环境使用 jib-maven-plugin 插件制作出带有镜像的 tar 文件。

操作步骤

1. 修改需要制作镜像的项目代码。

   找到 pom 文件并引入插件，内容如下：

```
<!--使用 jib 插件-->
          <plugin>
             <groupId>com.google.cloud.tools</groupId>
             <artifactId>jib-maven-plugin</artifactId>
             <version>1.3.0</version>
             <configuration>
                <!--from 节点用来设置镜像的基础镜像，相当于 Docerkfile 中的 FROM 关键字-->
                <from>
                   <!—建议使用 swr 公开镜像，下载速度快，更稳定 -->
                   <image>swr.cn-north-5.myxxcloud.com/xxxx/xxx:xxxxx</image>
                </from>
                <to>
                   <!--镜像名称和 tag，使用了 mvn 内置变量${project.version}，表示当前工程的 version-->
                   <image> hellojib:${project.version}</image>
                </to>
                <!--容器相关的属性-->
                <container>
                   <!--jvm 内存参数-->
                   <jvmFlags>
                      <jvmFlag>-Xms4g</jvmFlag>
                      <jvmFlag>-Xmx4g</jvmFlag>
                   </jvmFlags>
                   <!--要暴露的端口-->
                   <ports>
                      <port>8080</port>
                   </ports>
                </container>
             </configuration>
          </plugin>
```

   - From 标签：设置基础镜像，相当于 dockerfile 中的 FROM 关键字，这里推荐使用 SWR 中的镜像，构建时下载速度快并且稳定。
   - To 标签：设置制作出来的镜像的镜像名称和 tag。
   - Container 标签：设置容器的相关属性，jvm 内存参数，端口等。

- mainClass 标签：设置项目启动的主程序，也就是 Spring Boot 的 Application 类。

2. 创建构建任务并执行。
   a. 添加两个构建步骤：Maven 构建和上传软件包到软件发布库，并将 Maven 构建默认命令修改为：

```
mvn compile jib:buildTar -Dmaven.test.skip=true -U -e -X -B
```

说明

jib 构建工具主要包含了四个强大的功能，由于编译构建是在没有 docker 环境的情况下构建，所以使用 build 命令和 dockerBuild 命令并不能制作出镜像，只能使用 buildTar 命令制作出一个包含镜像的 tar 文件。

- build 提供了创建镜像并推送到远程仓库功能。
- buildTar 提供创建一个包含镜像的 tar 文件功能。
- dockerBuild 提供创建 docker 镜像到本地功能。
- exportDockerContext 提供创建 dockerfile 功能。

构建成功后，日志显示如下信息：

```
35  [2021/01/04 15:51:33.154] [INFO]
36  [2021/01/04 15:51:33.154] [INFO] --- jib-maven-plugin:1.3.0:buildTar (default-cli) @ javaMavenDemo ---
37  [2021/01/04 15:51:36.709] [INFO]
38  [2021/01/04 15:51:36.709] [INFO] Containerizing application to file at '***//target/jib-image.tar'...
39  [2021/01/04 15:51:36.709] [INFO] Getting base image**********
40  [2021/01/04 15:51:36.709] [INFO] Building classes layer...
41  [2021/01/04 15:51:37.528] [INFO] The base image requires auth. Trying again for**********
42  [2021/01/04 15:51:37.529] [INFO] Retrieving registry credentials for swr.cn-north-5.myhuaweicloud.com...
43  [2021/01/04 15:51:55.378] [INFO]
44  [2021/01/04 15:51:55.378] [INFO] Container entrypoint set to [java, -Xms4g, -Xmx4g, -cp, /app/resources:/app/classes:/app/libs/*, HelloWorld]
45  [2021/01/04 15:51:55.378] [INFO] Building image to tar file...
46  [2021/01/04 15:51:58.647] [INFO]
47  [2021/01/04 15:51:58.647] [INFO] Built image tarball at ***//target/jib-image.tar
48  [2021/01/04 15:51:58.647] [INFO]
49  [2021/01/04 15:51:58.647] [INFO] ------------------------------------------------------------
50  [2021/01/04 15:51:58.647] [INFO] BUILD SUCCESS
51  [2021/01/04 15:51:58.647] [INFO] ------------------------------------------------------------
52  [2021/01/04 15:51:58.648] [INFO] Total time: 30.526 s
53  [2021/01/04 15:51:58.648] [INFO] Finished at: 2021-01-04T07:51:57Z
```

   b. 在 java 工程的 target 目录下，可以看到生成了名为 jib-image.tar 的文件，同时任务会通过上传软件到发布库步骤上传到发布库。

3. 使用 tar 镜像。

   通过执行脚本或下载命令从发布库中将 tar 文件下载到要部署应用的服务器上，执行 docker load 命令将 tar 文件的镜像加载到本地镜像仓库，再使用 docker run 等命令启动镜像即可。

3.10 代码更新后构建打出来的包还是旧的

问题现象

本地提交了代码到远程仓库，并且确认远程仓库代码已经更新，但是构建后打出来的包，解压并反编译后发现还是旧的代码。

原因分析

这种问题一般是用户不小心将本地编译后的文件（“target”目录文件）上传到远程仓库，同时打包前没有执行 clean 操作导致。

处理方法

- 方法一：删除远程仓库的“target”目录。
- 方法二：打包命令增加“clean”参数，如：原先打包命令为：mvn package -Dmaven.test.skip=true -U -e -X -B，增加“clean”参数后如下：

```
mvn clean package -Dmaven.test.skip=true -U -e -X -B
```

3.11 对应的扩展点不存在

问题现象

构建任务执行失败，日志提示“对应的服务扩展点不存在”。

原因分析

服务扩展点数据丢失，构建任务如果关联了该服务扩展点，则执行时会报错。

处理方法

重新在服务扩展点页面新建服务扩展点，并将服务扩展点重新关联到构建任务中，以构建任务中的“通用 Git”服务扩展点丢失为例。

1. 单击顶部导航栏“设置 > 通用设置 > 服务扩展点管理”。
2. 新建通用 Git 服务扩展点。
3. 返回执行失败的构建任务，编辑该任务，在“源码选择”页签重新关联新建的通用 Git 服务扩展。
4. 重新执行构建任务。

4 Android 构建

4.1 项目配置的 Jcenter()不稳定

问题现象

执行过构建任务日志报错信息如下：

```
Caused by: org.gradle.internal.resource.transport.http.HttpErrorStatusCodeException:
Could not GET 'https://jcenter.bintray.com/org/apache/commons/commons-
compress/1.8.1/commons-compress-1.8.1.pom'. Received status code 504 from server:
Gateway Time-out
```

原因分析

- 网络异常无法连接依赖镜像仓。
- 依赖镜像仓异常。

处理方法

建议配置开源镜像站，稳定、快速，配置方法如下：

进入构建任务依赖的代码仓库，在“build.gradle”文件中添加如下代码，即可配置开源镜像仓。

```
allprojects {
repositories {
maven {
url 'https://repo.xxcloud.com/repository/maven/'
}
jcenter()
}
}
```

4.2 lint 检查出错终止任务执行

问题现象

处理方法

可以在命令行中的 gradle 命令后加上-xlint 参数，跳过 lint 检查。如：

```
/bin/bash ./gradlew assembleDebug -Dorg.gradle.daemon=false -d --stacktrace -xlint
```

或

```
gradle assembleDebug -Dorg.gradle.daemon=false -d --stacktrace --init-script
/root/.gradle/init.gradle -xlint
```

4.3 无法下载 com.android.tools.build:gradle:3.0.1 依赖

问题现象

错误信息如下：

```
Could not find com.android.tools.build:gradle:3.0.1
```

处理方法

根据日志提示，对“app”目录下的“build.gradle”文件添加 google()仓库，进行如下修正：

```
allprojects {
repositories {
google()
jcenter()
```

```
}
}
```

4.4 Javadoc generation failed

问题现象

Gradle 在构建过程中执行了 **javadoc** 检查，可能会报“Javadoc generation failed”的错误：

处理方法

避免 **javadoc** 的检查，在项目根目录下的 Gradle 下面就要添加如下配置：

```
allprojects {
repositories {
jcenter()
}
tasks.withType(Javadoc) {
options.addStringOption('Xdoclint:none', '-quiet')
options.addStringOption('encoding', 'UTF-8')
}
}
```

4.5 Could not find method google()

问题现象

Gradle 插件升级到 Gradle Plugin Build Tool 3.0 版本后，对应的 Gradle 需要升级到 4.1 版本。如果编译插件找不到对应的 Gradle 4.1 版本，就会报如下错误：

```
Could not find method google() for arguments [] on repository container
```

处理方法

构建工具 Gradle 选择 4.1 以上版本即可。

4.6 Gradle 版本过低

问题现象

执行 Android 构建后出现如下所示提示，需要 Gradle 最低版本是 3.3，当前是 2.10。

原因分析

编译环境的 Gradle 版本较低不满足编译要求。

处理方法

- 如果是 Gradle 构建，则选择符合条件的 Gradle 版本。
- 如果是 Gradlew 构建，则修改“gradle/wrapper/gradle-wrapper.properties”文件，修改“gradle-*.*-all.zip”的版本。

gradle-wrapper.properties 大小：231B

```
#Mon Dec 28 10:00:20 PST 2015
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-2.10-all.zip

```

4.7 Android APK 签名失败

问题现象

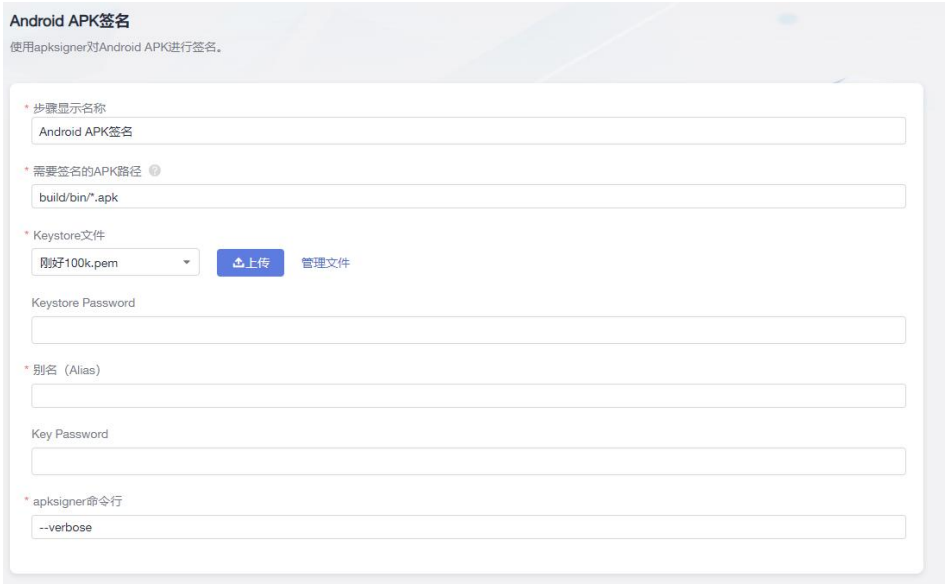
Android 构建时报签名错误：

错误信息类似于 “Execution failed for task ':app:validateSigningDebug” 或者“Execution failed for task ':app:validateSigningRelease”。

处理方法

在 Android 构建过程中推荐使用“Android APK 签名”构建步骤完成 APK 签名，编译构建提供了 Android APK 签名构建步骤，配置方法如下：

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



参数说明如下：

参数	说明
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，单击下拉列表，展示已经上传的 Keystore 文件，请根据需要选择。
keystore password	密钥文件密码。
别名（Alias）	密钥别名。
key password	密钥密码。
apksigner 命令行	用户自定义签名参数，默认“--verbose”显示签名详细。

2. 验证签名是否成功。

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

5 Gradle 构建

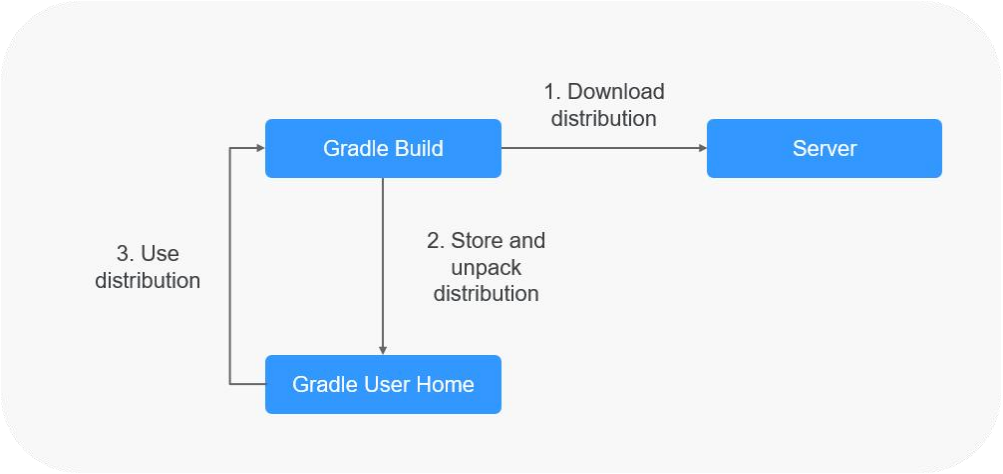
5.1 找不到指定版本的 Gradle 工具

问题现象

编辑 Action 时找不到想要的 Gradle 工具版本。

原因分析

- 如果您需要编译的工程依赖的 Gradle 版本不在列表中，可以使用 gradlew(gradle wrapper)封装 Gradle 命令。
- Gradlew 封装了 Gradle 命令，将首先安装指定版本的 Gradle，再执行构建命令。
- Gradle 官方建议所有 Gradle 项目中都创建 Wrapper 文件，方便没有安装 Gradle 的用户使用。



Gradle Wrapper 使用教程

步骤 1 在本地环境中，进入代码根目录，执行 gradle wrapper 命令。命令执行完毕后，可以发现代码仓库中新增了以下文件：

- gradlew (Unix Shell 脚本)
- gradlew.bat (Windows 批处理文件)

- gradle/wrapper/gradle-wrapper.jar (Wrapper JAR 文件)
- gradle/wrapper/gradle-wrapper.properties (Wrapper 属性文件)

步骤 2 提交代码到代码仓库。

步骤 3 修改构建任务中命令行里的语句，将 gradle 替换成./gradlew，如将 gradle build 替换为./gradlew build。

----结束

6 Npm 构建

6.1 JavaScript heap out of memory

问题现象

执行 Npm 构建任务时，日志报如下异常信息：

```
FATAL ERROR: CALL_AND_RETRY_LAST Allocation failed - JavaScript heap out of memory.
```

原因分析

Nodejs 运行时使用内存是有大小限制的，64 位系统约为 1.4GB，32 位系统约为 0.7GB，该次构建内存使用超出了默认大小。

处理方法

方法一：升级 nodejs 版本。

方法二：启动 Node 时设置“--max_old_space_size”或“--max_new_space_size”参数来调整内存大小的使用限制。

```
node --max_old_space_size=1700 test.js // 单位为MB 修改老生代内存限制
node --max_new_space_size=1024 test.js // 单位为KB 修改新生代内存限制
```

针对前端三大框架的解决方法如下：

框架类型	解决方法
Vue	只需要修改“package.json”文件中“build”属性值，在命令中加入带参数的 node 命令即可，例如： `"build": "node --max_old_space_size=4096 build/build.js"`

React	举例说明“package.json”里面“scripts”字段的内容如下：
	`"scripts": {`
	`"start": "react-scripts start",`
	`"build": "react-scripts build",`
	`"test": "react-scripts test --env=jsdom",`

框架类型	解决方法
	"eject": "react-scripts eject" } 运行 npm run build 的时候跑的实际代码是 react-scripts build，项目根目录下“node_modules”文件夹，找到.bin 目录并打开它找到“react-scripts”文件，打开这个文件，把--max_old_space_size=4096 这行代码写在 #!/usr/bin/env node 后面： #!/usr/bin/env node --max_old_space_size=4096

Angular	举例说明“package.json”里面“scripts”字段的内容如下：
	"scripts": {
	"ng": "ng",
	"start": "ng serve",
	"build": "ng build",
	"test": "ng test",
	"lint": "ng lint",
	"e2e": "ng e2e"
	}
	这里的 ng 命令也和 React 一样，在项目根目录“mode_modules”文件夹下的.bin 目录里面存在名为 ng 的文件，修改该文件的首行：
	#!/usr/bin/env node --max_old_space_size=4096

6.2 Unexpected end of JSON ...

问题现象

执行 npm install 时，提示异常信息如下：

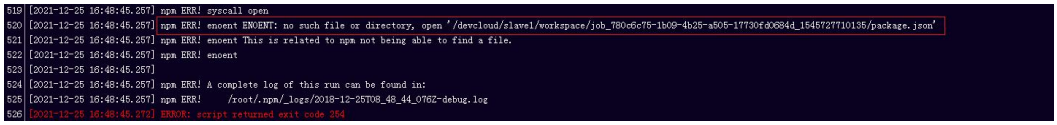
原因分析

解析文件中的 json 字符串失败，有可能从镜像仓下载的文件不完整。

6.3 enoent ENOENT: no such file or directory

问题现象

异常信息如下：



原因分析

项目缺少关键文件。

上图中 520 行的错误日志，“npm ERR! enoent ENOENT: no such file or directory, open '/xxx/slave1/workspace/job_780c6c75-1b09-4b25-a505-17730fd0684d_1545727710135/package.json'”，表示缺少“package.json”文件。

处理方法

补充错误日志中提示缺失的文件。比如缺少“package.json”文件的情况，需要在代码根目录下增加“package.json”文件。

6.4 Module not found: Error: Can't resolve ...

问题现象

执行 Npm 构建任务时，日志报如下异常信息：

原因分析

找不到需要的文件。

上图中 6068 行的错误日志，“Module not found: Error: Can't resolve './App.Vue' in '/xxx/slave1/workspace/job_d5d70df6-9b64-4faa-ba67-93c06d4a1972_1545727944134/src'”，在“src”文件夹下找不到“./App.Vue”文件。可能原因如下：

- 对应文件夹下，没有所需文件。
- 文件路径大小写配置有误。图中代码配置的是“'./App.Vue'”，实际文件名是“'./App.vue'”，导致找不到所需文件。因为 Windows 系统不区分大小写，而 Linux 系统区分，所以可能本地能构建成功，在编译构建服务上却构建失败。

处理方法

步骤 1 在代码项目中的相应文件夹下，补充错误日志中提示缺失的文件。

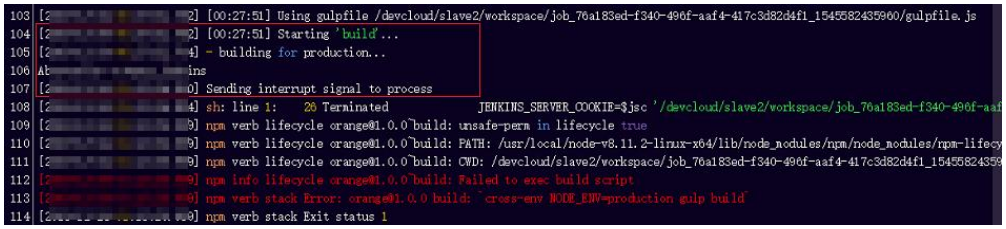
步骤 2 修改出错的代码中配置的文件路径。

----结束

6.5 NPM 构建失败，但不显示错误日志

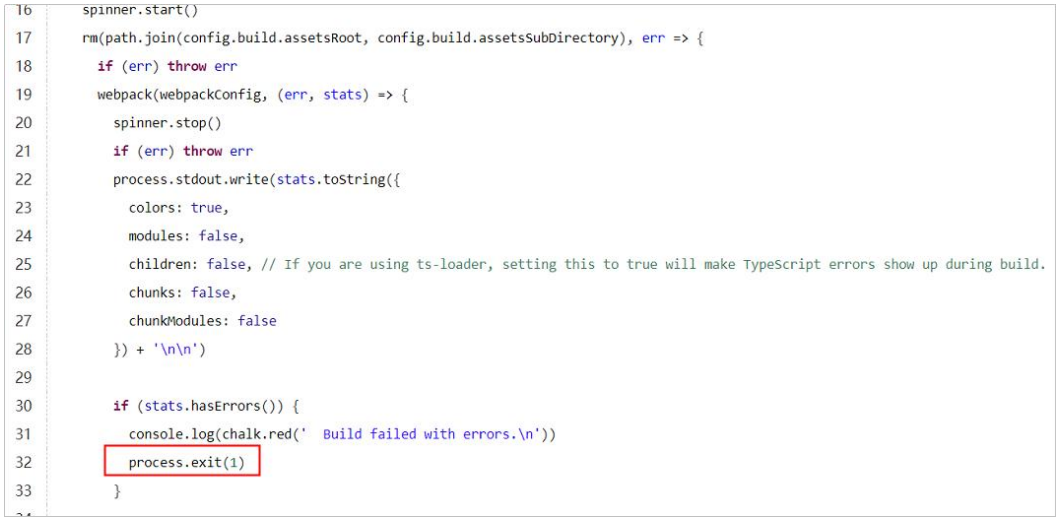
问题现象

Npm 构建失败，但不显示错误日志，异常信息如下：



原因分析

在构建脚本中，设置了出现错误时，直接退出构建。



```
spinner.start()
rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {
  if (err) throw err
  webpack(webpackConfig, (err, stats) => {
    spinner.stop()
    if (err) throw err
    process.stdout.write(stats.toString({
      colors: true,
      modules: false,
      children: false, // If you are using ts-loader, setting this to true will make TypeScript errors show up during build.
      chunks: false,
      chunkModules: false
    }) + '\n\n')

    if (stats.hasErrors()) {
      console.log(chalk.red('  Build failed with errors.\n'))
      process.exit(1)
    }
```

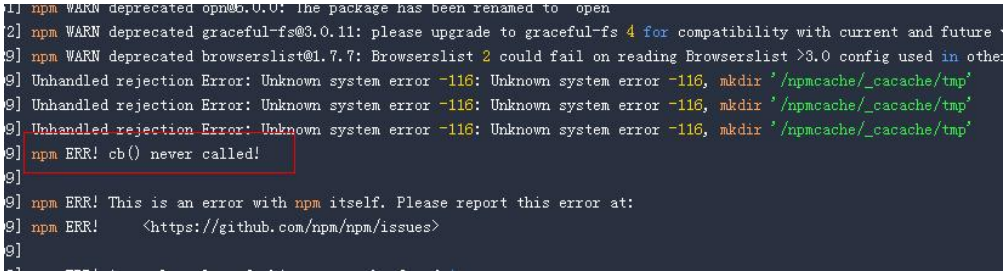
处理方法

检查构建脚本中对错误情况的处理，删除“process.exit(1) ”等可能导致构建出错时直接退出的情况。

6.6 npm cb() never called

问题现象

执行 Npm 构建任务时，日志报如下异常信息：

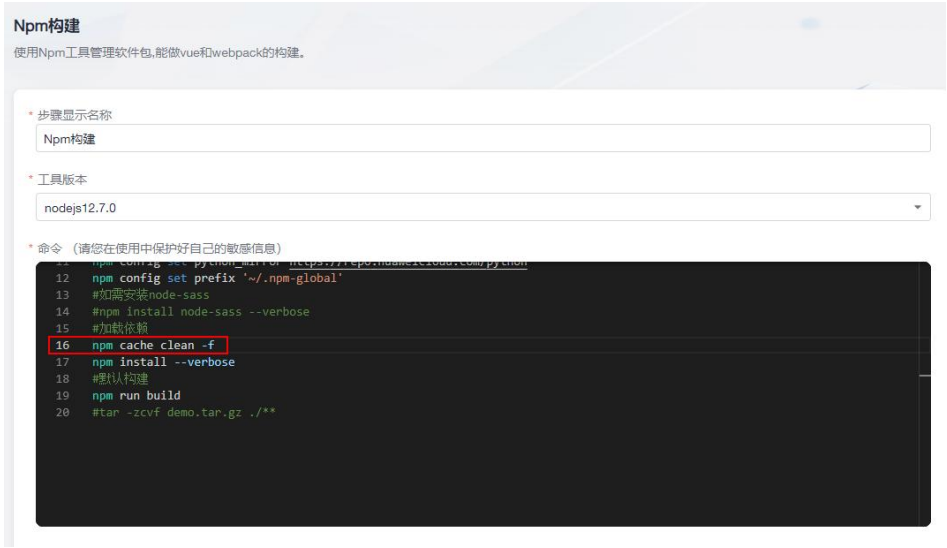


原因分析

NPM 缓存发生异常，需要清理缓存。

处理方法

编辑任务，在命令行 **npm install** 命令之前添加命令 **npm cache clean -f**，然后保存任务重新执行。



6.7 gyp ERR! stack Error: EACCES: permission denied

问题现象

执行 Npm 构建任务时，日志报如下异常信息：

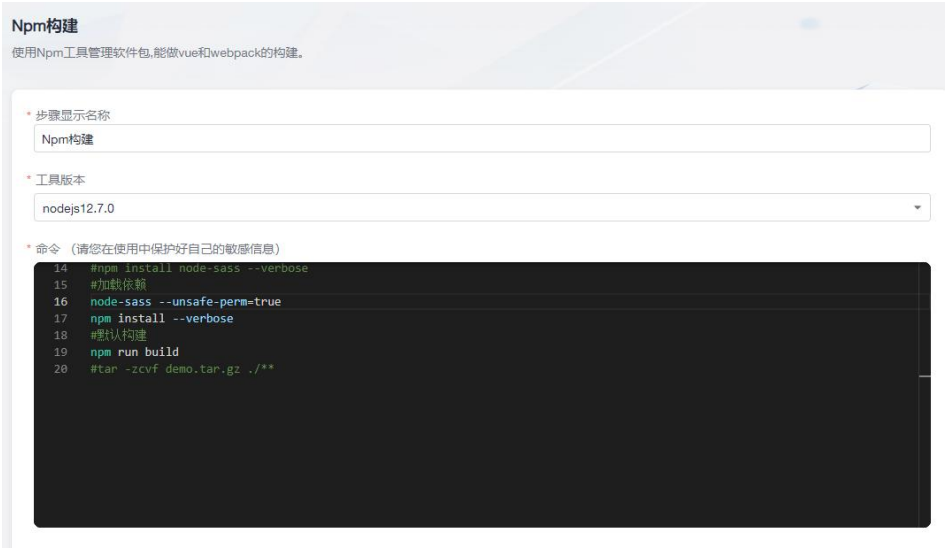
```
gyp ERR! stack Error: EACCES: permission denied, mkdir '******/job_1451ba57-0c35-4daa-99c2-21425404f61c_1564043318112/saas_shop/node_modules/node-sass/.node-gyp'
```

原因分析

当前目录没有读写权限。

处理办法

编辑任务，在命令行 **npm install** 命令之后添加 **node-sass --unsafe-perm=true**，保存任务重新执行。



6.8 eslint: error 'CLODOP' is not defined

问题现象

执行 Npm 构建任务时，日志报如下异常信息：

```
Module Error (from ./node_modules/@vue/cli-plugin-eslint/node_modules/eslint-
loader/index.js):

***//public/LodopFuncs.js
  79:25  error  'getCLodop' is not defined  no-undef
  80:27  error  Empty block statement       no-empty
  89:21  error  'CLODOP' is not defined     no-undef
```

原因分析

如上异常报 LodopFuncs.js 文件中函数未声明 is not defined，可先排查文件；文件正常则可能是不符合 eslint 规范导致报错。

处理方法

1. 检查 LodopFuncs.js 文件中 getCLodop 函数是否已定义。
2. 如果文件正常，可以在 eslint 检查不通过的文件头部添加如下命令行忽略 eslint 的检查。

   ```
   /* eslint-disable */
   ```

6.9 node-sass 下载失败

问题现象

执行 Npm 构建任务时，日志报如下异常信息：

```
Downloading binary from https://github.com/sass/node-
sass/releases/download/v4.14.1/linux-x64-72_binding.node
Cannot download "https://github.com/sass/node-sass/releases/download/v4.14.1/linux-
x64-72_binding.node":

read ECONNRESET

...

npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! node-sass@4.14.1 postinstall: `node scripts/build.js`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the node-sass@4.14.1 postinstall script.
npm ERR! This is probably not a problem with npm. There is likely additional
logging output above.
```

原因分析

node-sass 的镜像源需要单独设置，如果没有设置，npm 默认会去 github 下载。从软件开发生产线到 github 的网络不太稳定，容易下载失败。

处理方法

在默认命令 npm install 之前先加上如下命令，重新执行构建即可。

```
npm config set sass_binary_site https://repo.xxcloud.com/node-sass/
```

6.10 error: could not write config file

问题现象

执行 Npm 构建任务时，日志报异常信息：error: could not write config file /npmcache/_cacache/tmp/git-clone-b0ba91a1/.git/config: Disk quota exceeded

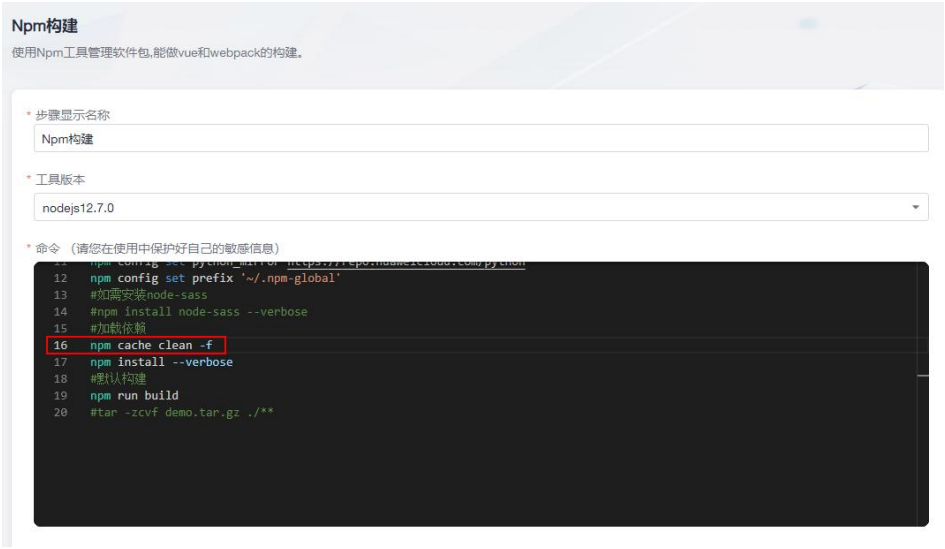
原因分析

NPM 缓存空间已满，需要清理缓存。

处理方法

1. 进入编译构建服务首页。

2. 选择对应的构建任务，单击任务所在行的 *** ，单击“编辑”。
3. 在“构建步骤”页面编辑“NPM 构建”。
4. 在命令行 **npm install** 命令之前添加命令 **npm cache clean -f**，然后保存任务重新执行。



6.11 npm 构建耗时且安装依赖缓慢

原因分析

默认的镜像仓地址因网络原因可能导致网络下行效率低。

处理方法

步骤 1 进入编译构建服务首页。

步骤 2 选择对应的构建任务，单击任务所在行的 *** ，单击“编辑”。

步骤 3 在“构建步骤”页面编辑“NPM 构建”。

步骤 4 在 NPM 构建步骤里，添加如下命令，修改 Npm 镜像仓地址：

```
npm config set registry https://repo.xxcloud.com/repository/npm/
```

或

```
npm config set registry https://registry.npm.taobao.org
```

步骤 5 单击“保存并执行”，重新执行构建任务。

----结束

7 Docker 构建

7.1 使用 Dockerfile 制作镜像失败

使用步骤“制作镜像并推送到 SWR”或“执行 Docker 命令”制作镜像时，docker build 阶段可能会制作镜像失败，可参考各场景对应解决方案处理：

- COPY 或者 ADD 指令找不到文件
- 制作镜像时拉取基础镜像失败
- 执行命令失败

COPY 或者 ADD 指令找不到文件

问题现象

构建任务中有“制作镜像并推送 SWR”或“执行 Docker 命令”构建步骤，执行任务时日志报如下异常信息：

```
ADD failed: stat /var/lib/docker/tmp/docker-builder154037010/temp: no such file or directory
[ERROR][制作镜像并推送到 SWR 仓库]：错误信息：DEV.CB.0210043，制作 Docker 镜像失败。
COPY failed: stat /var/lib/docker/tmp/docker-builder076130522/test.txt: no such file or directory
```

原因分析

ADD 指令的源文件为“./temp”，而当前工作目录下没有 temp 文件。

处理方法

假设当前目录的结构如下：

```
+ target
   - temp
- Dockerfile
```

target 目录下有 temp 文件，而 Dockerfile 文件和 target 同级。

- 方法一：将 ADD 指令的源文件改为“./target/temp”。
- 方法二：target 目录作为工作目录，将“制作镜像并推送到 SWR 仓库”构建步骤的工作目录改为“target”，Dockerfile 路径改为“../Dockerfile”。

制作镜像时拉取基础镜像失败

使用 Dockerfile 制作镜像时，如果指定的基础镜像参数有误，会导致镜像拉取失败，主包括以下两个场景：

- 指定的镜像不存在或无权限

  错误日志

  ```
  pull access denied for java1, repository does not exist or may require 'docker login'
  ```

  分析处理

  镜像仓库中找不到指定的镜像或当前用户对该镜像没有 pull 权限时，会出现该错误。

  此例的 Dockerfile 中，*FROM java1:8ull-jdk-alpine* 命令指定的镜像“java1”无法在镜像仓库中找到，故出现此错误，请核对并修正镜像名后重试即可。

- 指定的镜像标签不存在

  错误日志

  ```
  manifest for java:8ull-jdk-alpine not found
  ```

  分析处理

  镜像仓库中存在指定镜像，但不存在镜像的对应版本/标签时会出现“manifest not found”错误。此例的 Dockerfile 中，*FROM java:8ull-jdk-alpine* 命令指定了镜像“java:8ull-jdk-alpine”，镜像仓库中存在“java”镜像，但没有对应的版本/标签“8ull-jdk-alpine”，故出现此错误，请核对并修正镜像版本后重试即可。

执行命令失败

问题现象

使用 Dockerfile 制作镜像时，在执行 docker build 阶段报如下错：

```
exec user process caused "exec format error"
```

原因分析

此问题出现的原因一般有两个：

- 制作镜像的基础镜像和执行机不匹配，如：镜像为 arm 的，但是执行机是 x86 的。
- Dockerfile 文件内容从其他地方复制过来时出现问题。

处理方法

1. 先确认镜像和执行机是否匹配，如果镜像是 x86 的镜像，就只能用 x86 的执行机。
2. 重新执行构建，查看是否成功，如果不成功，手动输入 Dockerfile 后再重新执行。

7.2 推送镜像到 SWR 失败

使用步骤“制作镜像并推送到 SWR”或“执行 Docker 命令”时，因参数错误、环境问题等，可能会出现推送镜像失败，可参考各场景对应解决方案处理。

- 推送镜像提示无权限（denied: you do not have the permission）
- 推送镜像提示组织数达到上限（denied: The number of namespaces exceeds the upper limit）
- 推送镜像提示未登录（denied: You may not login yet）
- 推送镜像提示认证失败（denied: Authenticate Error）
- 推送镜像提示组织名非法（invalid reference format）
- 推送镜像提示本地镜像不存在（An image does not exist locally with the tag: ***）
- 推送镜像提示非法摘要（digest invalid: Invalid digest）

推送镜像提示无权限

错误日志

上传镜像到 SWR 仓库，提示如下错误：

```
denied: you do not have the permission

 [ERROR]  : [pluginFrame] step run failed, errorMessage: DEV.CB.0210044, Docker push failed
```

分析处理

此错误表示当前用户对目标组织没有权限，请逐步排查以下可能的原因：

1. 编辑构建任务，单击“制作镜像并推送到 SWR 仓库”构建步骤，查看组织名。
2. 登录容器镜像服务，在组织管理里查看组织是否存在。
   - 组织不存在，创建组织即可（组织数不可超过上限）。
   - 组织存在，但当前用户对该组织没有编辑权限，推送镜像时仍然会出现此错误，管理员可参考容器镜像服务 SWR 的“用户指南 > 授权管理”选择性为当前用户授权。
   - 组织存在，且用户对该组织有编辑权限，那么请进入统一身份认证服务，检查该用户是不是在只读权限的用户组里，如果是，请移除该用户。

推送镜像提示组织数达到上限

错误日志

```
denied: The number of namespaces exceeds the upper limit
```

```
 [ERROR] : [pluginFrame] step run failed, errorMessage: DEV.CB.0210044, Docker push
failed
```

分析处理

推送镜像提示未登录

错误日志

```
denied: You may not login yet
 [ERROR] : [pluginFrame] step run failed, errorMessage: fail to execute docker
command
```

分析处理

此类错误发生的原因一般有如下两种：

- push 操作前未使用“docker login”命令登录，此时添加对应登录命令即可。
- 执行了登录命令，但是登录命令中 SWR 地址错误，导致执行没报错但实际登录未生效，需要核对登录命令是否正确。

推送镜像提示认证失败

错误日志

```
Error response from daemon: Get https://swr.xxx.xxx.com/v2/: denied: Authenticate
Error
 [ERROR] : [pluginFrame] step run failed, errorMessage: fail to execute docker
command.
```

分析处理

此类错误一般为 SWR 登录命令中帐号/密码填写错误或临时登录帐号信息已过期导致，参考容器镜像服务 SWR 的“用户指南 > 镜像管理客户端上传镜像（推荐）”获取有效登录指令重试即可。

推送镜像提示组织名非法

错误日志

```
invalid reference format
 [ERROR] : [pluginFrame] step run failed, errorMessage: fail to execute docker
command.
```

分析处理

SWR 服务对“组织”命名有相应格式要求，推送镜像时，如果使用的组织名不满足其格式要求，则会出现此错误。

推送镜像提示本地镜像不存在

错误日志

```
[2022-03-05 17:01:05.816] An image does not exist locally with the tag:
swr.xxx.xxx.com/demo/faqdemo1
```

```
[ERROR] : [pluginFrame] step run failed, errorMessage: fail to execute docker
command.
```

分析处理

此类错误一般为镜像制作失败或 push 命令中镜像名、标签等信息填写错误，导致 push 命令中期望的镜像与 build/tag 命令中实际生成的镜像不一致，需要检查镜像制作过程或 push 参数是否正确。

此例中镜像 docker push swr.xxx.xxx.com/demo/faqdemo1:v1.1 中 faqdemo1 填写错误，build 参数中指定的镜像名为 faqdemo，修正 push 参数后再试即可。

推送镜像提示非法摘要

错误日志

```
digest invalid: Invalid digest
```

分析处理

此问题一般为 SWR 网络不稳定导致，重试几次即可。

7.3 拉取镜像失败

问题现象

执行构建任务时，日志报如下异常信息：

```
ERROR: docker pull image failed, dockerImage
```

原因分析

镜像拉取失败的原因可能有以下几种：

- 网络异常导致拉取超时。
- 拉取的镜像不存在。
- 拉取的镜像为私有镜像。

处理方法

- 网络异常导致，可以通过以下方法处理：
  a. 重试确认是否能解决。
  b. 如频繁出现或重试仍然失败请通过官网留言联系支持人员。
- 镜像不存在：请确保镜像已经上传至镜像仓，且镜像名称、镜像版本正确。
- 镜像为私有镜像：请将镜像设置为公开，或者先执行 *docker login* 鉴权通过后再执行 *docker pull* 操作。

7.4 镜像仓库登录异常

问题现象

异常信息如下：

```
Error response from daemon: login attempt to https://hub.docker.com/v2/ failed with
status: 404 Not Found
```

原因分析

镜像仓库地址填写有误，编译构建不支持自定义 https 请求的镜像仓库。

处理方法

镜像仓库地址保持系统提供的默认值。

8 制作镜像并推送到 SWR 仓库

8.1 如何推送到其他租户

问题现象

制作镜像并推送到 SWR 仓库时，提示错误信息“DEV.CB.0210043”，并提示制作 Docker 镜像失败。

处理方法

1. 进入编译构建服务首页。
2. 选择对应的构建任务，单击任务所在行的 *** ，单击“编辑”。
3. 在“构建步骤”页面编辑“制作镜像并推送到 SWR 仓库”。
4. 单击“管理 IAM 账号”。

5. 单击“新建扩展服务点”，选择“IAM 账户”。

6. 在弹出的窗口中填写参数信息。

Access Key Id 和 Secret Access Key 获取方式如下：

a. 单击“控制台”。

b. 单击页面右上角账号名称，选择“我的凭证”。

c. 单击“访问密钥”。

d. 单击“新增访问密钥”，填写相关描述，单击“确定”。

e. 在弹出的窗口中单击“立即下载”，可将密钥信息下载到本地。

A	B	C
User Name	Access Key Id	Secret Access Key
[redacted]	[redacted]	[redacted]

7. 步骤 4 中的 IAM 账号选择步骤 6 中新建的服务扩展点。

8.2 构建时拉取 dockerhub 镜像超时/次数限制

问题现象

执行构建任务时，拉取 dockerhub 镜像超时/次数限制，日志报如下异常信息：

```
Error response from daemon: Get https://registry.docker-cn.com/v2/: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

或

```
toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: https://www.docker.com/increase-rate-limit
```

原因分析

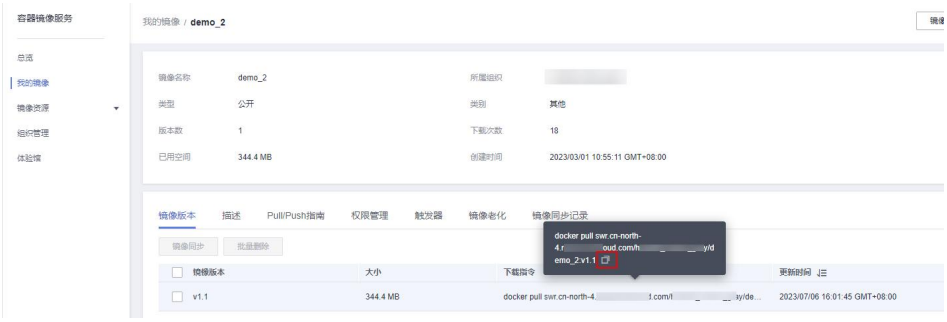
可能是因为 dockerhub 镜像源的网络不稳定并且存在频率限制，容易导致拉取超时或失败。可以将 dockerhub 镜像源的镜像迁移到 SWR 上，再拉取镜像。

处理方法

步骤 1 下载 dockerhub 镜像源的镜像到本地。

步骤 2 参考页面上传镜像页面，上传镜像到 SWR。

步骤 3 在镜像详情页面中，单击对应镜像版本“下载指令”列的复制图标，复制镜像下载指令。



步骤 4 修改代码仓中 Dockerfile 文件，将文件中镜像地址修改为步骤 3 中拷贝的地址。


dockerdemo0412 / Dockerfile
Dockerfile
历史
修改追溯
1 FROM swr. l.com/ 2/demo:v1.1
3 RUN mkdir -p /run/nginx && apk --update add nginx && sed -i "s#return 404;#root /usr/share/nginx/html;#g" /etc/nginx/conf.d/default.conf
5 COPY 2048 /usr/share/nginx/html
7 EXPOSE 80
9 CMD ["nginx", "-g", "daemon off;"]

----结束