

**SeawayEdge应用开发指南**

中科海微（北京）科技有限公司

Seaway Technologies Co. Ltd

## **目录**

[目录 1](#_Toc871125752)

[文档修订目录 1](#_Toc1614492265)

[文档版本 1](#_Toc1315618010)

[Version3.2 1](#_Toc1100264558)

[第一章 开发环境准备 4](#_Toc164741989)

[1.1安装软件包 4](#_Toc712188140)

[1.2使用IDE进行Java开发 5](#_Toc1819704249)

[1.3 网络设置 6](#_Toc1454696016)

[1.4 用户管理 11](#_Toc2122103464)

[1.5 仓库设置 13](#_Toc784510072)

[1.6 开发工具安装 15](#_Toc1858671171)

[1.7 虚拟化环境安装 15](#_Toc1389241735)

[1.8 编译环境配置 15](#_Toc1543629961)

[第二章 使用GCC编辑 16](#_Toc38815120)

[2.1 基本规则 16](#_Toc1678176799)

[2.2库 19](#_Toc67241095)

[第三章 使用make编译 22](#_Toc544685343)

[3.1 基本规则 22](#_Toc1951256287)

[3.2 Makefile 25](#_Toc541642272)

[3.2.1Makefile结构 25](#_Toc198485871)

[3.2.2 Makefile主要内容 26](#_Toc909930106)

[3.2.3 示例 26](#_Toc964241255)

[第四章 使用JDK编译 27](#_Toc1091172523)

[4.1 基本规则 27](#_Toc1973732328)

[4.2 库类 31](#_Toc339341487)

[4.3示例 32](#_Toc1743289224)

[第五章 构建RPM包 34](#_Toc1342591747)

[5.1打包说明 34](#_Toc1328812800)

[5.2本地构建 37](#_Toc1674284447)

[第六章 调试程序 40](#_Toc1220474088)

[6.1启动程序debug 40](#_Toc1891684519)

[6.2使用gdb调试程序 40](#_Toc46316998)

[6.3应用程序交互监控 41](#_Toc1060705172)

[第七章 FAQ 43](#_Toc1010072057)

[7.1部分依赖java-devel的应用程序自编译失败 43](#_Toc422832464)

[第八章 安装obs工具 44](#_Toc521834525)

[8.1介绍 44](#_Toc149647327)

[8.2架构适配 44](#_Toc419274252)

[8.3安装 44](#_Toc848507557)

[8.4部署 46](#_Toc1575094419)

# 文档修订目录

## 文档版本

## Version3.2

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号 | 修订日期 | 修订内容 | 修订人 |
| V3.1 | 2023/04/29 | 初订 | 陈海鑫 |

**前言**

在使用本手册之前，请您认真阅读以下使用许可协议，只有在同意以下使用许可协议的情况下方能使用本手册中介绍的产品。

**版权声明**

中科海微（北京）科技有限公司版权所有，并保留对本文档及本声明的最终解释权和修改权。本文档中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明外，其著作权或其他相关权利均属于中科海微（北京）科技有限公司。未经中科海微（北京）科技有限公司书面同意，任何人不得以任何方式或形式对本手册内的任何部分进行复制、摘录、备份、修改、传播、翻译成其它语言、将其全部或部分用于商业用途**。**

**免责条款**

本文档依据现有信息制作，其内容如有更改，恕不另行通知。中科海微（北京）科技有限公司在编写该文档的时候已尽最大努力保证其内容准确可靠，但中科海微（北京）科技有限公司不对本文档中的遗漏、不准确、或错误导致的损失和损害承担责任。

**技术支持与信息反馈**

如果您在使用我们的产品时遇到问题,或者您认为我们的产品有某些功能缺陷,请访问我们的官网www.haiwei.tech联系我们的客服,我们将为您解决问题和反馈;或者需要技术支持指导以及有任何宝贵意见，也请您通过官网或者电话联系我们：

**文档简介**

本文档简要介绍应用程序开发需要的常用工具，以指导用户进行应用程序开发。

* 在SeawayEdge系统中安装和使用GCC编译器，并完成一个简单代码的开发、编译和执行。
* 在SeawayEdge系统中使用JDK自带工具完成代码的编译和执行。
* 在SeawayEdge系统中安装IntelliJ IDEA进行Java开发。
* 在本地或使用OBS（Open Build Service）创建RPM（The RPM Package Manager）软件包的方法。

**应用开发指南**

本文档主要适用于进行代码开发的用户。用户需要具备以下经验和技能：

* 具备Linux操作系统得基础知识
* 了解Linux命令行的基本操作

# 开发环境准备

## 1.1安装软件包

安装开发过程中需要用到的软件。不同的开发需要的软件不一样，但安装方法相同，本章以安装常用的几个软件包（JDK，rpm-build）为例。有些开发软件SeawayEdge操作系统已默认自带，如GCC、GNU make。

* + 1. 安装JDK软件包

执行dnf list installed | grep jdk 查询JDK软件是否已安装：



清除缓存：



创建缓存:



查询可安装的JDK软件包：



在root权限下安装JDK软件包，以安装java-1.8.0-openjdk-devel软件包为例:



查询JDK软件版本:



* + 1. 安装rpm-build软件包

执行dnf list installed | grep rpm-build 查询rpm-build软件是否已安装：



清除缓存：



创建缓存:



在root权限下安装rpm-build软件包：



查询rpm-build软件版本：



## 1.2使用IDE进行Java开发

对于小型的Java程序，可以直接参考“使用JDK编译”章节得到可运行Java应用。但是对于大中型Java应用，这种方式已经无法满足开发者的需求。因此您可以参考如下步骤安装IDE并进行使用，以方便您在SeawayEdge系统上的Java开发工作。

* 简介

IntelliJ IDEA是一款非常流行的Java IDE，其社区版可以免费下载使用。目前支持使用IntelliJ IDEA集成开发环境（IDE）进行Java程序的开发，从而可以提升开发人员的工作效率。

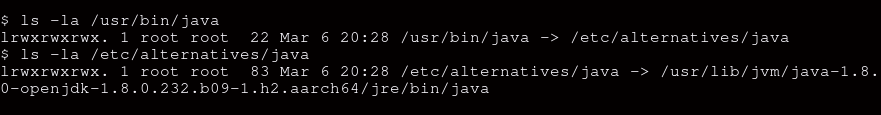
* 使用MobaXterm登录服务器：

MobaXterm是一款非常优秀的SSH客户端，其自带X Server，可以轻松解决远程GUI显示问题。您需要提前下载安装好MobaXterm并打开，然后SSH登录您的服务器并进行以下操作。

* 设置JDK环境

在设置JAVA\_HOME之前您需要先找到JDK的安装路径。在“开发环境准备 > 安装软件包 > 安装JDK软件包”章节中您已经学会了如何安装JDK，如果您还没安装JDK，请提前安装好。查看java路径，命令如下：

查看软链接的实际指向目录，命令如下：



发现JDK的真实路径为/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.232.b09-1.h2.aarch64，设置JAVA\_HOME和PATH，命令如下：



* 下载安装GTK库

运行如下命令：

如果显示gtk2或者gtk3，则表示您已安装该库，可以直接跳过进入下一步，否则需要在root权限下运行如下命令自动下载安装gtk库：



* 设置X11 Forwarding

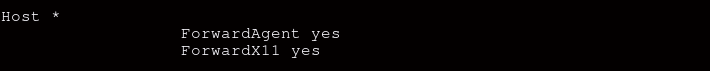
切换到sshd配置目录：



如果该目录不存在，则创建目录后再进行切换，创建目录命令如下：



然后在.ssh目录下编辑config文件并保存：（将以下内容添加到文件末尾并保存）



* 下载并运行IntelliJ IDEA

在执行如上环境配置后，您就可以下载使用IntelliJ IDEA了。鉴于最新版的IntelliJ IDEA和SeawayEdge系统在部分功能上有兼容性问题，建议您从此链接下载2018版本linux压缩包。下载好后把压缩包移到您想要安装该软件的目录，对压缩包进行解压：



解压后切换到IntelliJ IDEA的目录下并运行：



## 1.3 网络设置

* + 1. 配置网络

在海微服务器操作系统中，一般使用配置网络接口的命令行工具：ifconfig、ifup、ifdown。

* + 1. ifconfig命令

管理员用户可以使用ifconfig命令查看网络设备状态信息以及对网络接口进行配置。

■ 命令格式：ifconfig [网络设备] [参数]

■ 常用参数

参数 描述

add <地址> 设置网络设备的IP地址。

del <地址> 删除网络设备的IP地址。

down 关闭指定的网络设备。

netmask <子网掩码> 设置网络设备的子网掩码。

说明：在终端界面，执行man ifconfig，查看其他更多参数及其含义

■ 执行示例

◆ 执行 ifconfig ，显示网络设备信息。

◆ 执行 ifconfig eth1 down，关闭网络接口eth1。

◆ 执行 ifconfig -a ，显示全部当前有效的接口信息。

◆ 执行 ifconfig eth1 192.168.1.88 netmask 255.255.255.0，设置网络接口eth1的ip地址，子网掩码。

* + 1. ifup命令

管理员用户可以使用ifup命令激活一个指定的网络接口。

执行 ifup eth1，激活网络接口eth1。

* + 1. ifdown命令

管理员用户可以使用ifdown命令关闭一个指定网络接口。

执行 ifdown eth1，关闭网络接口。

* + 1. 管理ip

海微服务器操作系统中，一般使用管理IP的命令行工具：ip、dhclient、route。

ip命令

ip命令用来显示或操纵Linux主机的路由、网络设备、策略路由和隧道，是Linux下较新的功能强大的网络配置工具。

使用ip命令，只需一个命令，你就能很轻松地执行一些网络管理任务。相对于ifconfig，ip命令功能更强大。

海微服务器操作系统已经预装了iproute，可以轻松的设置IP。

■ 命令格式：ip [参数] OBJECT { COMMAND | help }

■ 常用参数

常用参数 对象说明

link 网络设备。

address 设备上的协议（IP或IPv6）地址。

addrlabel 协议地址选择的标签配置。

route 路由表条目。

rule 路由策略数据库中的规则。

-V -Version：显示指令版本信息。

-s -stats，statistics：输出详细信息。

-h -human，-human-readable：输出人类可读的统计信息和后缀。

-o -oneline：将每条记录输出到一行，用‘\’字符替换换行符。

■ 执行示例

◆ 执行 ip addr show，或 ip address show，显示网卡及配置的地址信息。

◆ 执行 ip addr add 192.168.0.123/24 dev eth1，设置IP地址。24表示子网掩码是24表示子网掩码的长度是24位，换算成十进制是255.255.255.0。

◆ 执行 ip addr del 192.168.0.123/24 dev eth1，删除ip地址。

◆ 执行 ip link set eth1 up ，启用被禁用的网卡eth1。

◆ 执行 ip link set eth1 down，禁用网卡。

◆ 执行 ip -s link list，显示详细的设备信息。

◆ 执行 ip route list，显示核心路由表。

dhclient命令

dhclient命令使用动态主机配置协议动态的配置网络接口的网络参数。

■ 命令格式：dhclient [参数]

■ 常用参数

参数 说明

-p 指定dhcp客户端监听的端口号。

-d 总是以前台方式运行程序。

-r 释放ip地址。

■ 执行示例

◆ 执行 dhclient -4 eth0指定dhclient支持ipv4协议。

◆ 执行 dhclient -r，释放当前IP。

route命令

Linux系统的route命令用于显示和操作IP路由表，要实现两个不同的子网之间的通信，需要一台连接两个网络的路由器，或者同时位于两个网络的网关来实现。

备注：直接在命令行下执行route命令来添加路由，不会永久保存，当网卡重启或者机器重启之后，该路由就失效了；可以在/etc/rc.local中添加route命令来设置路由永久有效。

■ 命令格式：route [参数]

■ 常用参数

参数 描述

-n 不解析名字

-v 显示详细的处理信息

-F 显示发送信息

-C 显示路由缓存

add 添加一条新路由。

del 删除一条路由。

-net 目标地址是一个网络。

-host 目标地址是一个主机。

netmask 当添加一个网络路由时，需要使用网络掩码。

metric 设置路由跳数。

gw 路由数据包通过网关。注意，你指定的网关必须能够达到。

■ 执行示例

◆ 执行 route add -net 192.168.2.0 netmask 255.255.255.0 dev eth0， 添加一条到达192.168.2.0网络的路由，指定网络掩码为255.255.255.0，数据包通过网络接口eth0。

◆ 执行 route add -net 192.168.66.0 netmask 255.255.255.0 gw 192.168.2.1， 添加一条到达192.168.66.0网络的路由，指定网络掩码为255.255.255.0，数据包通过网关地址192.168.2.1。

◆ 执行 route add -host 192.168.66.200 gw 192.168.2.1，所有去往192.168.66.200主机的数据包发往网关地址192.168.2.1。

◆ 执行 route add default gw 192.168.1.1 ，添加一条默认网关，所有的数据包将被转发到192.168.1.1。

◆ 添加或设置网关，执行route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0命令。

◆ 删除路由记录，执行route del -net 224.0.0.0 netmask 240.0.0.0命令。

* + 1. 网络诊断

ping命令

ping命令是常用的网络命令，也属于一个通信协议，是TCP/IP协议的一部分。使用ping命令可以检查网络是否连通，能够帮助管理员分析和判定网络故障。

执行ping指令会使用ICMP传输协议，发出要求回应的信息，若远端主机的网络功能没有问题，就会回应该信息，因而得知该主机运作正常。ICMP协议是Internet控制报文协议。是TCP/IP协议簇的一个子协议，用于在IP主机、路由器之间传递控制消息。

■ 命令格式：ping [参数] 地址

■ 常用参数

参数 描述

-d 使用Socket的SO\_DEBUG功能。

-c <完成次数> 设置完成要求回应的次数。

-f 极限检测。

-i <间隔秒数> 指定收发信息的间隔时间。

-l <前置载入> 设置在送出要求信息之前，先行发出的数据包。

-n 只输出数值。

-p <范本样式> 设置填满数据包的范本样式。

-q 不显示指令执行过程，开头和结尾的相关信息除外。

-r 忽略普通的Routing Table，直接将数据包送到远端主机上。

-R 记录路由过程。

-s <数据包大小> 设置数据包的大小。

-t <存活数值> 设置存活数值TTL的大小。

■ 执行示例

◆ 执行 ping 192.168.0.12，查看本机的网络状态。

◆ 执行 ping -i 3 -s 1024 -t 255 www.baidu.com，-i 3 发送周期为 3秒， -s 设置发送包的大小 -t 设置TTL值为 255。

◆ 执行 ping -c 2 www.baidu.com，收到两次包后，自动退出。

◆ 执行 ping -i 0.5 -c 10 www.baidu.com，每0.5秒发送一次网络封装包并限制发送10次。

traceroute命令

通过traceroute我们可以知道信息从你的计算机到互联网另一端的主机是走的什么路径。当然每次数据包由某一同样的出发点（source）到达某一同样的目的地(destination)走的路径可能会不一样，但基本上来说大部分时候所走的路由是相同的。

traceroute主要用于追踪网络数据包的路由途径，默认值数据包大小是40Bytes，用户可自定义大小。

注意：在海微服务器操作系统中，默认未集成traceroute，手动执行yum install traceroute，安装即可使用。

■ 命令格式：traceroute [参数] [地址]

■ 常用参数

参数 描述

-d 使用Socket层级的排错功能。

-f 设置第一个检测数据包的存活数值TTL的大小。

-F 设置勿离断位。

-g 设置来源路由网关，最多可设置8个。

-i 使用指定的网络界面送出数据包。

-I 使用ICMP回应取代UDP资料信息。

-m 设置检测数据包的最大存活数值TTL的大小。

-n 直接使用IP地址而非主机名称。

-p 设置UDP传输协议的通信端口。

-r 忽略普通的Routing Table，直接将数据包送到远端主机上。

-s 设置本地主机送出数据包的IP地址。

-t 设置检测数据包的TOS数值。

-w 设置等待远端主机回报的时间。

■ 执行示例

◆ 执行 traceroute www.baidu.com，显示到达目的地的数据包路由。

◆ 执行traceroute -m 4 www.baidu.com，显示到达目的地的数据包路由，设置数据存活数值为4。

* + 1. 域名管理

在海微服务器操作系统中，一般使用管理域名的命令行工具：host、nslookup。

host命令

host 是一个常用的DNS查询工具，经常用来查询域名、检查域名解析是否正确。

■ 命令格式：host [参数] name [server]

■ 常用参数

参数 描述

-a 查询所有的信息。

-c 设置查询类型。

-C 查询完整的SOA记录。

-d | -v 显示详细过程。

-l 列表模式。

-t 选择查询类型：CNAME NS SOA SIG KEY AXFR。

-w 永久等待。

-W 设置等待超时。

■ 执行示例：执行 host -a www.baidu.com，查看域名地址的详细信息。

nslookup命令

nslookup是一个用于查询 Internet域名信息或诊断DNS服务器问题的工具。主要用来诊断域名系统的基础信息。

■ 命令格式：nslookup [参数] [dns-server]

■ 执行示例

◆ 执行 nslookup www.baidu.com ，查询网站域名信息。

◆ 执行 nslookup -q=mx mail.qq.com，查看邮件服务器的信息。

* + 1. 网络状态

netstat命令

在海微服务器操作系统中，一般通过netstat命令来查看网络连接状态，当设置不同命令参数时，可以显示网络连接状态、路由表、接口状态、无效连接和多播成员。也可以用于查看某个服务是否运行。

■ 命令格式：netstat [参数]

■ 常用参数

参数 描述

-a (all)显示所有选项，netstat默认不显示LISTEN相关。

-t (tcp)仅显示tcp相关选项。

-u (udp)仅显示udp相关选项。

-n 拒绝显示别名，能显示数字的全部转化成数字。(重要)

-l 仅列出有在 Listen (监听) 的服务状态。

-p 显示建立相关链接的程序名。

-r 显示路由信息，路由表。

-e 显示扩展信息，例如uid等。

-s 按各个协议进行统计。(重要)

-c 连续打印所选信息。

说明：在终端界面，执行man netstat ，查看其他更多参数及其含义

■ 执行示例

◆ 执行 netstat -a，列出所有网路端口。

◆ 执行 netstat -i，显示网卡列表。

◆ 执行 netstat -apu ，查看UDP端口的使用情况。

◆ 执行 netstat -s，显示网络统计信息。

## 1.4 用户管理

用户管理常用的功能：添加用户、修改用户、删除用户、口令管理、添加用户组、修改用户组、删除用户组。

* + 1. 添加用户

useradd命令用来创建或更新用户信息。

■ 命令格式：useradd [参数] 用户名

■ 常用参数

参数 描述

-c comment给新用户添加备注。

-d home\_dir为主目录指定一个名字（如果不想用登陆名作为主目录名的话）。

-m 创建用户的HOME目录。

-M 不创建用户的HOME目录（当默认设置里指定创建时，才用到）。

-N 不要创建与用户同名的组。

-r 创建系统用户。

-p passwd为用户指定默认密码。

-s shell指定默认登陆shell。

-u uid为帐户指定一个唯一的UID。

-U --user-group创建与用户同名的组。

说明：执行man useradd命令，查看其他更多参数及其含义。

■ 执行示例：以root用户身份登陆系统，打开终端界面。

◆ 执行useradd testuser1添加用户testuser1。

◆ 执行useradd -m testuser2命令，创建用户主目录。

◆ 执行useradd -m -s /bin/bash testuser1命令，指定用户登陆后使用的shell。

◆ 执行useradd -U testuser2命令，创建一个和用户同名的组，并将用户添加到组中。

说明：

■ 执行less /etc/passwd命令，可以查看新增加用户testuser1的信息。

■ 执行cat /etc/shells可以查看系统安装了哪些shell。

* + 1. 修改用户

usermod命令可用来修改用户帐号的各项设定。

■ 命令格式：usermod [参数] 用户名

■ 常用参数

参数 描述

-a --append 把用户追加到某些组中，仅与-G一起使用。

-d --home用户的新登陆目录。如果给了-m选项，当前主目录的内容会移动到新主目录中，如果不存在，则创建。

-l --login用户的名称将会从原用户修改为新用户。

-G --groups GROUP1[,GROUP2,...[,GROUPN]]]用户属于的附加组列表。组之间使用逗号分隔，没有空格。

-u --uid UID设置用户ID的新数值。

说明：执行man usermod命令，查看其他更多参数及其含义。

■ 执行示例

◆ 执行usermod -aG testgrp testuser多个组之间用空格隔开，把testuser用户加入testgrp组。可以使用id testuser查看是否追加到组成功。

说明：执行id testuser命令，查看用户加入的组。

◆ 执行usermod -l testuser testuser1命令，修改testuser1的用户名为testuser。

◆ 执行usermod -d /home/testuser4 -m testuser命令，修改用户名为testuser的home名。

◆ 执行usermod -u 10002 testuser命令，修改用户名为testuser的UID。

* + 1. 删除用户

■ 命令格式：userdel [参数] 用户名

■ 常用参数

参数 描述

-h --help显示帮助信息。

-r --remove用户主目录中的文件将随用户主目录和用户邮箱一起删除。在其它文件系统中的文件需要手动搜索并删除。

-Z --selinux-user为用户删除所有的SELinux用户映射。

说明：执行man userdel命令，查看其他更多参数及其含义。

■ 执行示例：以root用户身份登陆系统，打开终端界面。

◆ 执行userdel testuser1命令，将删除testuser1相关的用户数据文件，如 /etc/passwd、/etc/group、/etc/shadow等都将被删除。但不删除home目录及文件。

◆ 执行userdel -r testuser1命令，删除用户目录及其它相关文件，home目录及文件也将删除。

注意：请不要轻易用-r参数；他会删除用户的同时删除用户所有的文件和目录，切记如果用户目录下有重要的文件，在删除前请备份。

* + 1. 口令管理

■ 命令格式：passwd [参数] 用户名

■ 常用参数

表1.12 passwd参数表

参数 描述

-l 锁定口令，即禁用帐号。

-u 解锁指定用户的密码。

-d 删除用户密码，这是禁用一个用户密码的快速方法。

说明：执行man passwd命令，查看其他更多参数及其含义

■ 执行示例

◆ 执行passwd testuser3，修改testuser3这个用户的密码。输入新密码，确认新密码。

◆ 执行passwd -l testuser3，锁定testuser3这个用户。

◆ 执行passwd -u testuser3，解锁testuser3这个用户。

说明：远程终端，执行ssh testuser3@192.168.0.20，输入密码，验证操作是否生效。192.168.0.20是服务器的IP地址。

■ 认证模块：passwd具体调用的认证模块与/etc/pam.d/system-auth 和/etc/pam.d/password-auth 中的相关配置有关。

## 1.5 仓库设置

讲述如何配置操作系统软件仓库，使操作系统可以通过yum/dnf等命令进行在线安装、卸载或升级应用软件开发所需要的软件包。

在海微服务器操作系统安装完成后，首先对软件源配置文件进行配置并升级。简单来说就是你安装软件时，程序从哪个仓库地址获取软件包。

* + 1. 配置软件源

软件源的配置一般有两种方式，一种是直接配置 /etc/dnf/dnf.conf 文件，另外一种是在 /etc/yum.repos.d 目录下增加 .repo 文件。

/etc/dnf/dnf.conf 文件包含的“main”部分，配置文件示例如下：

[main]

gpgcheck=0

installonly\_limit=3

clean\_requirements\_on\_remove=True

best=True

Skip\_if\_unavailable=False

常用选项说明：

参数 说明

cachedir 缓存目录，该目录用于存储RPM包和数据库文件。

keepcache 可选值是1和0，表示是否要缓存已安装成功的那些RPM包及头文件，默认值为0，即不缓存。

debuglevel 设置dnf生成的debug信息。取值范围：[0-10]，数值越大会输出越详细的debug信息。默认值为2，设置为0表示不输出debug信息。

clean\_requirements\_on\_remove 删除在dnf remove期间不再使用的依赖项，如果软件包是通过DNF安装的，而不是通过显式用户请求安装的，则只能通过clean\_requirements\_on\_remove删除软件包，即它是作为依赖项引入的。 默认值为True。

best 升级包时，总是尝试安装其最高版本，如果最高版本无法安装，则提示无法安装的原因并停止安装。 默认值为True。

obsoletes 可选值1和0，设置是否允许更新陈旧的RPM包。默认值为1，表示允许更新。

gpgcheck 可选值1和0，设置是否进行gpg校验。默认值为1，表示需要进行校验。

plugins 可选值1和0，表示启用或禁用dnf插件。默认值为1，表示启用dnf插件。

installonly\_limit 设置可以同时安装“installonlypkgs”指令列出包的数量。默认值为3，不建议降低此值。

repository部分允许您定义定制化的软件源仓库，各个仓库的名称不能相同，否则会引起冲突。下面是[repository]部分的一个最小配置示例：

[repository]

name=repository\_name

baseurl=repository\_url

repository参数说明：

参数 说明

name=repository\_name 软件仓库（repository ）描述的字符串

baseurl=repository\_url 软件仓库（repository ）的地址。

■ 使用http协议的网络位置：例如<http://path/to/repo>

■ 使用ftp协议的网络位置：例如<ftp://path/to/repo>

■ 本地位置：例如file:///path/to/local/repo

海微服务器操作系统中有默认的repo文件SeawayEdgeOS.repo，其中包含AppStream、BaseOS、PowerTools、Plus、Update等源。

SeawayEdgeOS.repo文件的AppStream配置如下所示：

[SeawayEdgeOS-$releasever-AppStream]

name = SeawayEdgeOS$releasever AppStream

baseurl= https://enterprise-c-packages.chinauos.com/server-enterprise-c/kongzi/1050/AppStream/$basearch

enabled = 1

username=$auth\_u

password=$auth\_p

gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-uos-release

gpgcheck = 0

skip\_if\_unavailable = 1

说明：

■ SeawayEdgeOS默认已经配置好了镜像源仓库，需要激活授权后才能使用。

■ SeawayEdgeOS默认配置中有认证相关的配置，请勿修改repo默认配置文件，否则会导致软件包更新失败,username和password为认证所需要的用户名和密码。

* + 1. 安装软件包

■ 执行以下命令安装特定软件包。

dnf install PKGNAME

■ 执行以下命令更新所有已安装的软件包。

dnf upgrade

注意：更新软件包将被下载到/var/cache/dnf/目录下缓存，然后再进行安装。升级的记录会写在日志文件/var/log/dnf.log、/var/log/dnf.librepo.log、/var/log/dnf.rpm.log中。

## 1.6 开发工具安装

* + 1. 代码编辑工具安装

讲述vim等软件工具的安装

在安装软件前需要完成系统软件仓库的配置，软件仓库具体配置方法参考“仓库设置”章节的描述。

Linux系统图像化模式下，系统默认会安装vi、vim等编译关键，在Linux系统最小化安装或docker环境中，可以使用yum install vim命令来安装vim。

* + 1. 代码调试工具安装

在安装软件前需要完成系统软件仓库的配置，软件仓库具体配置方法参考“仓库设置”章节的描述。

gdb：调试器

Valgrind：是一款用于内存调试、内存泄漏检测以及性能分析的软件开发工具。

ltrace：用来跟踪进程调用库函数的情况；

strace：监控用户空间进程和内核的交互,比如系统调用、信号传递、进程状态。

通过yum install命令来安装gdb valgrind ltrace strace。

yum install gdb valgrind ltrace strace

* + 1. 版本管理工具安装

在安装软件前需要完成系统软件仓库的配置，软件仓库具体配置方法参考“仓库设置”章节的描述。

git和svn都是版本控制管理工具，通过采用分支管理系统的高效管理，实现共享资源，实现最终集中式的管理。

通过yum install命令来安装git和svn。

## 1.7 虚拟化环境安装

在安装软件前需要完成系统软件仓库的配置，软件仓库具体配置方法参考“仓库设置”章节的描述。

虚拟化相关软件属于“container-tools”module，可以通过module形式安装。执行dnf module install container-tools即可完成安装。

## 1.8 编译环境配置

在开始开发C程序前，还需要完成基本开发库和相关编译器的安装部署。例如：

通过dnf module install llvm-toolset命令安装clang开发工具集；

通过dnf install glibc glibc-devel glibc-headers命令安装glib相关库和头文件；

通过dnf install kernel-headers命令安装kernel头文件；

通过dnf module install mariadb mariadb-devel 命令安装mariadb数据库和数据库相关库文件；

通过dnf module install virt-devel命令安装虚拟化相关开发库和头文件。

# 第二章 使用GCC编辑

GCC（GNU Compiler Collection）是GNU推出的功能强大、性能优越的多平台编译器。GCC编译器能将C、C++语言源程序、汇编程序和目标程序编译、链接成可执行文件。SeawayEdge操作系统中已默认安装了GCC软件包。

## 2.1 基本规则

* + 1. 文件类型

对于任何给定的输入文件，文件类型决定进行何种编译。GCC常用的文件类型如表1所示。

| 扩展名（后缀） | 说明 |
| --- | --- |
| .c | C语言源代码文件。 |
| .C，.cc或.cxx | C++源代码文件。 |
| .m | Objective-C源代码文件。 |
| .s | 汇编语言源代码文件。 |
| .i | 已经预处理的C源代码文件。 |
| .ii | 已经预处理的C++源代码文件。 |
| .S | 已经预处理的汇编语言源代码文件。 |
| .h | 程序所包含的头文件。 |
| .o | 编译后的目标文件。 |
| .so | 动态链接库，它是一种特殊的目标文件。 |
| .a | 静态链接库。 |
| .out | 可执行文件，但可执行文件没有统一的后缀，系统从文件的属性来区分可执行文件和不可执行文件。如果没有给出可执行文件的名字，GCC将生成一个名为a.out的文件。 |

* + 1. 编译流程

使用GCC将源代码文件生成可执行文件，需要经过预处理、编译、汇编和链接。

* 预处理：将源程序（如 .c 文件）预处理，生成 .i 文件。
* 编译：将预处理后的 .i 文件编译成为汇编语言，生成 .s 文件。
* 汇编：将汇编语言文件经过汇编，生成目标文件 .o 文件。
* 链接：将各个模块的 .o 文件链接起来生成一个可执行程序文件。

其中 .i 文件、.s文件、.o文件是中间文件或临时文件，如果使用GCC一次性完成C语言程序的编译，则这些文件会被删除。

* + 1. 编译选项

GCC编译的命令格式为：gcc [options] filenames其中：options ：编译选项。filenames ：文件名称。GCC是一个功能强大的编译器，其 options 参数取值很多，但大部分并不常用，常用的 options 取值如表2所示。

| options取值 | 说明 | 示例 |
| --- | --- | --- |
| -c | 编译、汇编指定的源文件生成目标文件，但不进行链接。通常用于编译不包含主程序的子程序文件。 | #使用-c选项编译test1.c、test2.c源文件  gcc -c test1.c test2.c |
| -S | 编译指定的源文件生成以.s作为后缀的汇编语言文件，但不进行汇编。 | #编译器预处理 circle.c，将其翻译成汇编语言，并将结果存储在 circle.s 文件中。  gcc -S circle.c |
| -E | 预处理指定的源文件，但不进行编译。  默认情况下，预处理器的输出会被导入到标准输出流（如显示器），可以利用-o选项把它导入到某个输出文件。 | #预处理的结果导出到 circle.i 文件。  gcc -E circle.c -o circle.i |
| -o file | 用在生成可执行文件时，生成指定的输出文件file。同时该名称不能和源文件同名。如果不给出这个选项，gcc就给出预设的可执行文件a.out。 | #将源文件作为输入文件，将可执行文件作为输出文件，也即完整地编译整个程序。  gcc main.c func.c -o app.out |
| -g | 在可执行程序中包含标准调试信息。 | - |
| -L library\_path | 在库文件的搜索路径列表中添加library\_path路径。 | - |
| -Ilibrary | 链接时搜索指定的函数库library。  使用 GCC 编译和链接程序时，GCC 默认会链接 libc.a 或者 libc.so，但是对于其他的库（例如非标准库、第三方库等），就需要手动添加。 | #使用-l选项，以链接数学库。  gcc main.c -o main.out -lm  说明：  数学库的文件名是 libm.a。前缀lib和后缀.a是标准的，m是基本名称，GCC 会在-l选项后紧跟着的基本名称的基础上自动添加这些前缀、后缀，本例中，基本名称为 m。 |
| -I head\_path | 在头文件的搜索路径列表中添加head\_path路径。 | - |
| -shared | 进行静态编译，及链接静态库，禁止链接动态库。 | - |
| -shared | 默认选项，可省略。   * 可以生成动态库文件。 * 进行动态编译，优先链接动态库，只有没有动态库时才会链接同名的静态库。 | - |
| -fPIC（或-fpic） | 生成使用相对地址的位置无关的目标代码。通常使用-static选项从该PIC目标文件生成动态库文件。 | - |

预编译：使用-E参数可以让GCC在预处理结束后停止编译过程

gcc -E hello.c -o hello.i

预处理的宏定义插入到hello.i中

生成目标代码：将hello.i编译为目标代码，通过使用-c参数来完成

gcc -c hello.i -o hello.o

也可以通过源文件直接生成

gcc -c hello.c

生成可执行文件：

gcc hello.o -o hello

也可以通过源文件直接生成

gcc -o hello hello.c

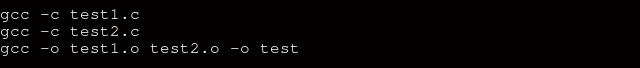
* + 1. 多源文件编译

多个源文件的编译方法有2种：

* 多个源文件一起编译。编译时需要所有文件重新编译



* 分别编译各个源文件，之后对编译后输出的目标文件链接。编译时只重新编译修改的文件，未修改的文件不用重新编译：



## 2.2库

库是写好的、现有的、成熟的、可以复用的代码。每个程序都要依赖很多基础的底层库。

库文件在命名时约定，以lib为前缀，以.so（动态库）或.a（静态库）为后缀，中间为库文件名。如libfoo.so或libfoo.a。由于所有的库文件都遵循了同样的规范，因此当在链接库时，-l 选项指定链接的库文件名时可以省去lib前缀，即GCC 在对-lfoo 进行处理时，会自动去链接名为libfoo.so 或libfoo.a的库文件。而当在创建库时，必须指定完整文件名libfoo.so或libfoo.a。

当使用gcc在编译一个C程序是，可以使用“-l”选项指定要依赖的库，在使用库是可用通过库名称来指定，不需要加后缀。如：gcc ...... -lfoo

当在生成一个库时，需要指定全名，如：gcc ...... -o libfoo.so.x.x

如果在某个目录下有编译时所需要的头文件，为了让GCC能够顺利地找到它们，就可以使用-L选项，如：gcc ...... -L /usr/local/lib/

根据链接时期的不同，库分为静态库和动态库。静态库是在链接阶段，将汇编生成的目标文件.o与引用到的库一起链接打包到可执行文件中；而动态库是在程序编译时并不会被链接到目标代码中，而是在程序运行时才被载入。二者有如下差异。

* 资源利用不一样

静态库为生成的可执行文件的一部分，而动态库为单独的文件。所以使用静态库和动态库的可执行文件大小和占用的磁盘空间大小不一样，导致资源利用不一样。

* 扩展性与兼容性不一样

静态库中某个函数的实现变了，那么可执行文件必须重新编译，而对于动态链接生成的可执行文件，只需要更新动态库本身即可，不需要重新编译可执行文件。

* 依赖不一样

静态库的可执行文件不需要依赖其他的内容即可运行，而动态库的可执行文件必须依赖动态库的存在。所以静态库更方便移植。

* 加载速度不一样

静态库在链接时就和可执行文件在一块了，而动态库在加载或者运行时才链接，因此，对于同样的程序，静态链接的要比动态链接加载更快。

* + 1. 动态链接库

使用-shared选项 和-fPIC选项，可直接使用源文件、汇编文件或者目标文件创建一个动态库。其中-fPIC 选项作用于编译阶段，在生成目标文件时就需要使用该选项，以生成位置无关的代码。

* 示例1：从源文件生成动态链接库。



* 示例2：从目标文件生成动态链接库。



将一个动态库链接到可执行文件，需要在命令行中列出动态库的名称。

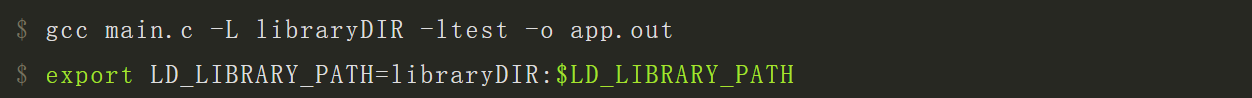
示例：将main.c和libtest.so一起编译成 app.out，当 app.out 运行时，会动态地加载链接库 libtest.so。



这种方式是直接指定使用当前目录下的libtest.so文件。若使用下面搜索动态库的方式，则为了确保程序在运行时能够链接到动态库，需要通过如下三种方法中的任一种实现。1. 将动态库保存在标准目录下，例如 /usr/lib。

2. 把动态库所在路径libraryDIR添加到环境变量LD\_LIBRARY\_PATH中。

3. 把动态库所在路径libraryDIR添加到 /etc/ld.so.conf中然后执行ldconfig或者以动态库所在路径libraryDIR为参数执行ldconfig。



* + 1. 静态链接库

创建一个静态链接库，需要先将源文件编译为目标文件，然后再使用ar命令将目标文件打包成静态链接库。

示例：将源文件test1.c，test2.c，test3.c编译并打包成静态库。



其中ar是一个备份压缩命令，可以将多个文件打包成一个备份文件（也叫归档文件），也可以从备份文件中提取成员文件。ar最常见的用法是将目标文件打包为静态链接库。

ar将目标文件打包成静态链接库的命令格式为：

* Sllfilename ：静态库文件名。
* Targetfilelist ：目标文件列表。
* r： 替换库中已有的目标文件，或者加入新的目标文件。
* c： 创建一个库，不管库是否存在，都将创建。
* s： 创建目标文件索引，在创建较大的库时能提高速度。

示例：创建一个main.c文件来使用静态库



其中libraryDIR为libtest.a库的路径。

* + 1. 示例
* 使用GCC编译C程序示例

cd到代码目录，以目录“~/code”进行举例。如下所示：



编写Hello World程序，保存为helloworld.c，此处以编译Hello World程序进行举例说明。示例如下：



在代码目录，执行编译，使用命令：



编译完成后，会生成helloworld文件，查看编译结果，示例如下：

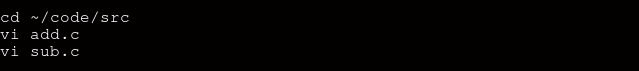


* 使用GCC创建和使用动态链接库示例

cd到代码目录，以目录“~/code”进行举例。并在该目录下创建src，lib，include子目录，分别用于存放源文件，动态库文件和头文件。



cd到~/code/src目录，创建2个函数add.c、sub.c，分别实现加、减。



将add.c、sub.c源文件创建为动态库libmath.so，并将该动态库存放在~/code/lib目录。



cd到~/code/include目录，创建1个头文件math.h，声明函数的头文件。



cd到~/code/src目录，创建一个调用add()和sub()的main.c函数。



将main.c和libmath.so一起编译成math.out。



将动态链接库所在的路径加入到环境变量中。



执行math.out。

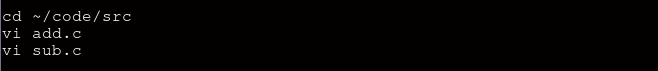


* 使用GCC创建和使用静态链接库示例

cd到代码目录，以目录“~/code”进行举例。并在该目录下创建src，lib，include子目录，分别用于存放源文件，静态库文件和头文件。



cd到~/code/src目录，创建2个函数add.c、sub.c，分别实现加、减。



将add.c、sub.c源文件编译为目标文件add.o、sub.o。



将add.o、sub.o目标文件通过ar命令打包成静态库libmath.a，并将该静态库存放在~/code/lib目录。



cd到~/code/include目录，创建1个头文件math.h，声明函数的头文件。



cd到~/code/src目录，创建一个调用add()和sub()的main.c函数。



将main.c和libmath.a一起编译成math.out。



执行math.out。



# 第三章 使用make编译

GNU make实用程序（通常缩写为make）是一种用于控制从源文件生成可执行文件的工具。 make会自动确定复杂程序的哪些部分已更改并需要重新编译。 make使用称为Makefile的配置文件来控制程序的构建方式。

## 3.1 基本规则

* + 1. 文件类型

makefile文件中可能用到的文件类型如表所示。

| 扩展名（后缀） | 说明 |
| --- | --- |
| .c | C语言源代码文件。 |
| .C，.cc或.cxx | C++源代码文件。 |
| .m | Objective-C源代码文件。 |
| .s | 汇编语言源代码文件。 |
| .i | 已经预处理的C源代码文件。 |
| .ii | 已经预处理的C++源代码文件。 |
| .S | 已经预处理的汇编语言源代码文件。 |
| .h | 程序所包含的头文件。 |
| .o | 编译后的目标文件。 |
| .so | 动态链接库，它是一种特殊的目标文件。 |
| .a | 静态链接库。 |
| .out | 可执行文件，但可执行文件没有统一的后缀，系统从文件的属性来区分可执行文件和不可执行文件。如果没有给出可执行文件的名字，GCC将生成一个名为a.out的文件。 |

* + 1. make工作流程

使用make由源代码文件生成可执行文件，需要经过如下步骤：1. make命令会读入Makefile文件，包括当前目录下命名为“GNUmakefile” 、“makefile” 、“Makefile”的文件、被include的makefile文件、参数-f、--file、--makefile指定的规则文件。2. 初始化变量。3. 推导隐含规则，分析依赖关系，并创建依赖关系链。4. 根据依赖关系链，决定哪些目标需要重新生成。5. 执行生成命令，最终输出终极文件。

* + 1. make选项

make命令格式为：make [option]... [target]...其中：option ：参数选项。target ：Makefile中指定的目标。常用make的 option 取值如表2所示。

| options取值 | 说明 |
| --- | --- |
| -C dir，--directory=dir | 指定make在开始运行后的工作目录为dir。  当存在多个 -C 选项的时候，make 的最终工作目录是第一个目录的相对路径。 |
| -d | make在执行的过程中打印出所有的调试信息。使用-d选项可以显示make构造依赖关系链、重建目标过程中的所有信息。 |
| -e，--environment-overrides | 使用环境变量定义覆盖Makefile中的同名变量定义。 |
| -f file，--file=file，--makefile=file | 指定file文件为make 执行的Makefile文件。 |
| -h，--help | 打印帮助信息。 |
| -i，--ignore-errors | 执行过程中忽略规则命令执行的错误。 |
| -k，--keep-going | 执行命令错误时不终止make的执行，make 尽最大可能执行所有的命令，直至出现致命的错误才终止。 |
| -n，--just-print，--dry-run | 按实际运行时的执行顺序模拟执行命令(包括用@开头的命令)，没有实际执行效果，仅仅用于显示执行过程。 |
| -o file，--old-file=file，--assume-old=file | 指定file文件不需要重建，即使它的依赖已经过期，同时不重建此依赖文件的任何目标。 |
| -p，--print-data-base | 命令执行之前，打印出make读取的Makefile的所有数据，同时打印出 make的版本信息。如果只需要打印这些数据信息，可以使用 “make -qp”命令，查看 make 执行之前预设的规则和变量，可使用命令“make -p -f /dev/null”。 |
| -r，--no-builtin-rules | 忽略内嵌的隐含规则的使用，同时忽略所有后缀规则的隐含后缀列表。 |
| -R，--no-builtin-variables | 忽略内嵌的隐含变量。 |
| -s，--silent，--quiet | 取消命令执行过程中的打印。 |
| -S，--no-keep-going，--stop | 取消 "-k" 的选项在递归的 make 过程中子 make 通过 "MAKEFLAGS" 变量继承了上层的命令行选项那个。我们可以在子 make 中使用“-S”选项取消上层传递的 "-k" 选项，或者取消系统环境变量 "MAKEFLAGS" 中 "-k"选项。 |
| -t，--touch | 更新所有的目标文件的时间戳到当前系统时间。防止 make 对所有过时目标文件的重建。 |
| -v，--version | 查看make的版本信息。 |

## 3.2 Makefile

make是通过Makefile文件获取如何编译、链接和安装、清理的方法，从而实现将源代码文件生成可执行文件和其他相关文件的工具。因此，Makefile中描述了整个工程的编译和链接等规则，其中包含了哪些文件需要编译，哪些文件不需要编译，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重建等等。Makefile文件让工程编译实现了自动化，不需要每次都手动输入一堆源文件和参数。

本章简单介绍Makefile文件的结构和主要内容，更多Makefile的内容请通过info make命令查询。

### 3.2.1Makefile结构

Makefile文件结构如下所示：targets:prerequisitescommand或者是：targets:prerequisites;commandcommand其中：

* targets ：目标，可以是目标文件、可执行文件或标签。
* prerequisites ：依赖文件，生成targets需要的文件或者是目标。可以是多个，也可以没有。
* command ：make需要执行的命令（任意的 shell 命令）。可以有多条命令，每一条命令占一行。
* 目标和依赖文件之间要使用“:”分隔，命令的开始一定要按“Tab”。

Makefile文件结构表明了输出的目标，输出目标的依赖对象和生成目标需要执行的命令。

### 3.2.2 Makefile主要内容

一个Makefile文件主要由以下内容组成：

* 显示规则

明确写出来的依赖关系，如要生成的文件，文件的依赖文件，生成的命令。

* 隐含规则

由make自动推导的规则，make命令支持自动推导功能。

* 变量的定义
* 文件指示

文件指示包括三部分：include 其他 Makefile，如include xx.md。选择执行，如#ifdef。定义多行命令，如define...endef。

* 注释

以 “#” 开头。

### 3.2.3 示例

* 使用Makefile实现编译的示例

cd到代码目录，以目录“~/code”进行举例。



创建1个头文件hello.h和2个函数hello.c、main.c。



创建Makefile文件。



执行make命令。



执行./main目标。



命令执行后，打印如下信息：The 1th say hello.The 2th say hello.The 3th say hello.The 4th say hello.

# 第四章 使用JDK编译

JDK（Java Development Kit）是 Java 开发者进行 Java 开发所必需的软件包，包含 JRE（Java Runtime Environment）和编译、调测工具。SeawayEdge在OpenJDK 的基础上进行了 GC 优化、并发稳定性增强、安全性增强等修改，提高了 Java 应用程序在 ARM 上的性能和稳定性。

## 基本规则

4.1.1 文件类型及工具

对于任何给定的输入文件，文件类型决定采用何种工具进行处理。JDK常用的文件类型如表所示.

| 扩展名（后缀） | 说明 |
| --- | --- |
| .java | java语言源代码文件。 |
| .class | java的字节码文件，是一种和任何具体机器环境及操作系统环境无关的中间代码。它是一种二进制文件，是Java 源文件由 Java 编译器编译后生成的目标代码文件。 |
| .jar | java的jar压缩文件。 |

JDK常用的工具如表所示

| 工具名称 | 说明 |
| --- | --- |
| java | Java运行工具，用于运行.class字节码文件或.jar文件。 |
| javac | Java编程语言的编译器，将.java的源代码文件编译成.class的字节码文件。 |
| jar | 创建和管理jar文件。 |

4.1.2 java程序生成流程

通过JDK将java源代码文件生成并运行Java程序，需要经过编译和运行。编译：是指使用Java编译器（javac）将java源代码文件（.java文件）编译为.class的字节码文件。运行：是指在Java虚拟机上执行字节码文件

* + 1. JDK常用工具选项
* javac编译选项

javac编译的命令格式为：javac [options] [sourcefiles] [classes] [@argfiles]其中：options ：命令选项。sourcefiles ：一个或多个需要编译的源文件。classes ：一个或多个要为注释处理的类。@argfiles ：一个或多个列出选项和源文件的文件。这些文件中不允许有-J选项。javac是java编译器，其 options 参数取值很多，但有些大部分并不常用，常用的 options 取值如表所示。

| options取值 | 说明 | 示例 |
| --- | --- | --- |
| -d path | 指定存放生成的类文件的路径。  默认情况下，编译生成的类文件与源文件在同一路径下。使用-d选项可以将类文件输出到指定路径。 | #使用-d选项将所有类文件输出到bin路径下  javac /src/\*.java -d /bin |
| -s path | 指定存放生成的源文件的路径。 | - |
| -cp path或-classpath path | 搜索编译所需的class文件，指出编译所用到的class文件的位置。 | #在Demo中要调用GetStringDemo类中的getLine()方法，而GetStringDemo类编译后的文件，即.class文件在bin目录下。  javac -cp bin Demo.java -d bin |
| -verbose | 输出关于编译器正在执行的操作的消息，如加载的类信息和编译的源文件信息。 | #输出关于编译器正在执行的操作的消息。  javac -verbose -cp bin Demo.java |
| -source sourceversion | 指定查找输入源文件的位置。 | - |
| -sourcepath path | 用于搜索编译所需的源文件（即java文件），指定要搜索的源文件的位置，如jar、zip或其他包含java文件的目录。 | - |
| -target targetversion | 生成特定JVM版本的类文件。取值为1.1，1.2，1.3，1.4，1.5（或5），1.6（或6），1.7（或7），1.8（或8）。targetversion的默认取值与-source选项的sourceversion有关。sourceversion取值：   * 1.2，targetversion为1.4； * 1.3，targetversion为1.4； * 1.5、1.6、1.7、未指定，targetversion为1.8。 * 其他值，targetversion与sourceversion取值相同。 | - |

* java运行选项

java运行的格式为：运行类文件：java [options] classesname [args]运行jar文件：java [options] -jar filename [args]其中：options ：命令选项，选项之间用空格分隔。classname ：运行的.class文件名。filename ：运行的.jar文件名。args：传递给main()函数的参数，参数之间用空格分隔。java是运行java应用程序的工具，其 options 参数取值很多，但有些大部分并不常用，常用的 options 取值如表所示。

| options取值 | 说明 | 示例 |
| --- | --- | --- |
| -cp path或-classpath path | 指定要运行的文件所在的位置以及需要用到的类路径，包括jar、zip和class文件目录。  当路径有多个时，使用“:”分隔。 | - |
| -verbose | 输出关于编译器正在执行的操作的消息，如加载的类信息和编译的源文件信息。 | #输出关于编译器正在执行的操作的消息。  java -verbose -cp bin Demo.java |

* Jar打包选项

jar的命令格式为：jar {c | t | x | u}[vfm0M] [jarfile] [manifest] [-C dir] file...jar命令参数说明如表所示。

| 参数 | 说明 | 示例 |
| --- | --- | --- |
| c | 创建jar文件包。 | #把当前目录的hello.class文件打包到Hello.jar，且不显示打包的过程。如果Hello.jar文件还不存在，就创建它，否则首先清空它。  jar cf Hello.jar hello.class |
| t | 列出jar文件包的内容列表。 | #列出Hello.jar包含的文件清单。  jar tf Hello.jar |
| x | 展开jar文件包的指定文件或者所有文件。 | #解压Hello.jar 到当前目录，不显示任何信息。  jar xf Hello.jar |
| u | 更新已存在的jar文件包，如添加文件到jar文件包中。 | - |
| v | 生成详细报告并打印到标准输出。 | #把当前目录的hello.class文件打包到Hello.jar，并显示打包的过程。如果Hello.jar文件还不存在，就创建它，否则首先清空它。  jar cvf Hello.jar hello.class |
| f | 指定jar文件名，通常这个参数是必须的。 | - |
| m | 指定需要包含的manifest清单文件。 | - |
| 0 | 只存储，不压缩，这样产生的jar文件包会比不用该参数产生的体积大，但速度更快。 | - |
| M | 不产生所有项的manifest清单文件，此参数会忽略m参数 | #把当前目录的hello.class文件打包到Hello.jar，并显示打包的过程。如果Hello.jar文件还不存在，就创建它，否则首先清空它。但在创建Hello.jar时不产生manifest 文件。  jar cvfM Hello.jar hello.class |
| jarfile | .jar文件包，它是f参数的附属参数。 | - |
| manifest | .mf的manifest清单文件，它是m参数的附属参数。 | - |
| -C dir | 转到指定dir下执行jar命令，只能配合参数c、t使用。 | - |
| file | 指定文件/路径列表，文件或路径下的所有文件（包括递归路径下的）都会被打入jar文件包中，或解压jar文件到路径下。 | #把当前目录的所有class文件打包到Hello.jar，并显示打包的过程。如果Hello.jar文件还不存在，就创建它，否则首先清空它。  jar cvf Hello.jar \*.class |

## 库类

java类库是以包的形式实现的，包是类和接口的集合。java编译器为每个类生成一个字节码文件，且文件名与类名相同，因此同名的类之间就有可能发生冲突。java语言中，把一组类和接口封装在一个包内，包可以有效地管理类名空间，位于不同包中的类即使同名也不会冲突，从而解决了同名类之间可能发生的冲突问题，为管理大量的类和接口提供了方便，也有利于类和接口的安全。

除java提供的许多包外，开发者也可以自定义包，把自己编写的类和接口等组成程序包的形式，以便后续使用。自定义包需要先声明包，然后再使用包。

* + 1. 包的声明

包的声明格式为：package pkg1[.pkg2[.pkg3...]];

为了声明一个包，首先必须建立一个相应的目录结构，子目录与包名一致，然后在需要放入该包的类文件开头声明包，表示该文件的全部类都属于这个包。包声明中的“.”指明了目录的层次。如果源程序文件中没有package语句，则指定为无名包。无名包没有路径，一般情况下，java仍然会把源文件中的类存储在当前工作目录（即存放java源文件的目录）下。

包声明语句必须被加到源程序文件的起始部分，而且前面不能有注释和空格。如果在不同源程序文件中使用相同的包声明语句，就可以将不同源程序文件中的类都包含在相同的包中。

* + 1. 包的引用

在 Java 中，为了能使用java提供的包中的公用类，或者使用自定义的包中的类，有两种方法。

* 在要引用的类名前带上包名。

如：name.A obj=new name.A ();其中，name为包名，A为类名，obj为对象。表示程序中用name包中的A类定义一个对象obj。示例：新建一个example包中Test类的test对象。

* 在文件开头使用import来导入包中的类。

import语句的格式为：import pkg1[.pkg2[.pkg3...]].(classname | \*);其中，pkg1[.pkg2[.pkg3...]]表明包的层次，classname为所要导入的类。如果要从一个包中导入多个类，则可以使用通配符“\*”来替代。示例：导入example包中的Test类。

示例：将example 整个包导入。



## 4.3示例

* + 1. 编译不带包的java程序示例

cd到代码目录，此处以“~/code”进行举例。如下所示：



编写Hello World程序，保存为HelloWorld.java，此处以编译Hello World程序进行举例说明。示例如下：



在代码目录，执行编译，使用命令：

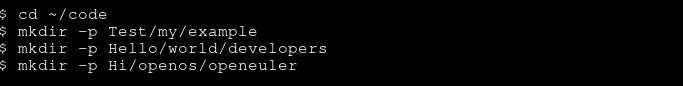


编译完成后，会生成 HelloWorld.class 文件，通过java命令可执行查看结果，示例如下：



* + 1. 编译带包的java程序示例

cd到代码目录，此处以“~/code”进行举例。并在该目录下创建“~/code/Test/my/example”、“~/code/Hello/world/developers”、“~/code/Hi/openos/SeawayEdge”子目录，分别用于存放源文件。



cd到~/code/Test/my/example目录，创建Test.java。



cd到~/code/Hello/world/developers目录，创建Hello.java。



cd到~/code/Hi/openos/SeawayEdge目录，创建Hi.java。



cd到~/code目录，使用javac编译源文件。



cd到~/code目录，使用java运行Test程序。



执行结果如下所示：



# 第五章 构建RPM包

## 5.1打包说明

* + 1. 原理介绍

RPM打包的时候需要编译源码，需要把编译好的配置文件、二进制命令文件等放到合适的位置，还要根据需要对RPM的包进行测试，这些都需要先有一个“工作空间”。rpmbuild命令使用一套标准化的“工作空间”：



rpmdev-setuptree这个命令就是安装 rpmdevtools 带来的。可以看到运行了这个命令之后，在“/root”目录（非root用户为“/home/用户名”目录）下多了一个 rpmbuild 的文件夹，目录结构如下：



内容相关的说明如下：

| **目录** | **宏代码** | **名称** | **功能** |
| --- | --- | --- | --- |
| ~/rpmbuild/BUILD | %\_builddir | 构建目录 | 源码包被解压至此，并在该目录的子目录完成编译 |
| ~/rpmbuild/RPMS | %\_rpmdir | 标准 RPM 包目录 | 生成/保存二进制 RPM 包 |
| ~/rpmbuild/SOURCES | %\_sourcedir | 源代码目录 | 保存源码包（如 .tar 包）和所有 patch 补丁 |
| ~/rpmbuild/SPECS | %\_specdir | Spec 文件目录 | 保存 RPM 包配置（.spec）文件 |
| ~/rpmbuild/SRPMS | %\_srcrpmdir | 源代码 RPM 包目录 | 生成/保存源码 RPM 包(SRPM) |

* + 1. 打包流程

打包的过程主要分为如下步骤：

* 把源代码放到%\_sourcedir中。
* 进行编译，编译的过程是在%\_builddir中完成的，一般情况下，源代码是压缩包格式，需要先进行解压。
* 进行“安装”，类似于预先组装软件包，把软件包应该包含的内容（比如二进制文件、配置文件、man文档等）复制到%\_buildrootdir中，并按照实际安装后的目录结构组装，比如二进制命令可能会放在/usr/bin下，那么就

在%\_buildrootdir下也按照同样的目录结构放置。

* 做一些必要的配置，比如在实际安装前的准备，安装后的清理等等。这些都是通过配置在SPEC文件中来告诉rpmbuild命令。
* 检查软件是否正常运行。
* 生成的RPM包放置到%\_rpmdir，源码包放置到%\_srcrpmdir下。

在SPEC文件中，各个阶段说明如下：

| **阶段** | **读取的目录** | **写入的目录** | **具体动作** |
| --- | --- | --- | --- |
| %prep | %\_sourcedir | %\_builddir | 读取位于 %\_sourcedir 目录的源代码和 patch 之后，解压源代码至 %\_builddir 的子目录并应用所有 patch。 |
| %build | %\_builddir | %\_builddir | 编译位于 %\_builddir 构建目录下的文件。通过执行类似 ./configure && make 的命令实现。 |
| %install | %\_builddir | %\_buildrootdir | 读取位于 %\_builddir 构建目录下的文件并将其安装至 %\_buildrootdir 目录。这些文件就是用户安装 RPM 后，最终得到的文件。 |
| %check | %\_builddir | %\_builddir | 检查软件是否正常运行。通过执行类似 make test 的命令实现。 |
| bin | %\_buildrootdir | %\_rpmdir | 读取位于 %\_buildrootdir 最终安装目录下的文件，以便最终在 %\_rpmdir 目录下创建 RPM 包。在该目录下，不同架构的 RPM 包会分别保存至不同子目录， noarch 目录保存适用于所有架构的 RPM 包。这些 RPM 文件就是用户最终安装的 RPM 包。 |
| src | %\_sourcedir | %\_srcrpmdir | 创建源码 RPM 包（简称 SRPM，以.src.rpm 作为后缀名），并保存至 %\_srcrpmdir 目录。SRPM 包通常用于审核和升级软件包。 |

* + 1. 打包选项

通过rpmbuild命令构建软件包。rpmbuild构建软件包一般可以通过构建SPEC文件、tar文件、source文件实现。rpmbuild命令格式为：rpmbuild [option...]

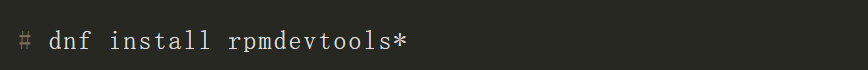
常用的rpmbuild打包选项如表所示。

| option取值 | 说明 |
| --- | --- |
| -bp specfile | 从specfile文件的%prep段开始构建（解开源码包并打补丁）。 |
| -bc specfile | 从specfile文件的%build段开始构建。 |
| -bi specfile | 从specfile文件的%install段开始构建。 |
| -bl specfile | 从specfile文件的%files段开始检查。 |
| -ba specfile | 通过specfile文件构建源码包和二进制包。 |
| -bb specfile | 通过specfile文件构建二进制包。 |
| -bs specfile | 通过specfile文件构建源码包。 |
| -rp sourcefile | 从sourcefile文件的%prep段开始构建（解开源码包并打补丁）。 |
| -rc sourcefile | 从sourcefile文件的%build段开始构建。 |
| -ri sourcefile | 从sourcefile文件的%install段开始构建。 |
| -rl sourcefile | 从sourcefile文件的%files段开始检查。 |
| -ra sourcefile | 通过sourcefile文件构建源码包和二进制包。 |
| -rb sourcefile | 通过sourcefile文件构建二进制包。 |
| -rs sourcefile | 通过sourcefile文件构建源码包。 |
| -tp tarfile | 从tarfile文件的%prep段开始构建（解开源码包并打补丁）。 |
| -tc tarfile | 从tarfile文件的%build段开始构建。 |
| -ti tarfile | 从tarfile文件的%install段开始构建。 |
| -ta tarfile | 通过tarfile文件构建源码包和二进制包。 |
| -tb tarfile | 通过tarfile文件构建二进制包。 |
| -ts tarfile | 通过tarfile文件构建源码包。 |
| --buildroot=DIRECTORY | 在构建时，使用DIRECTORY目录覆盖默认的/root目录。 |
| --clean | 完成打包后清除BUILD目录下的文件。 |
| --nobuild | 不执行任何实际的构建步骤。可用于测试spec文件。 |
| --noclean | 不执行spec文件的"%clean"阶段(即使它确实存在)。 |
| --nocheck | 不执行spec文件的"%check"阶段(即使它确实存在)。 |
| --dbpath DIRECTORY | 使用DIRECTORY中的数据库，而不是默认的 /var/lib/rpm。 |
| --root DIRECTORY | 使DIRECTORY为最高级别的路径，默认“/”为最高路径。 |
| --rebuild sourcefile | 将安装指定的源代码包sourcefile，然后进行准备、编译、安装。 |
| --recompile sourcefile | 在--recompile 的基础上额外构建一个新的二进制包。在构建结束时，构建目录、源代码和 spec 文件都将被删除(就好像用了 --clean)。 |
| -?，--help | 打印详细的帮助信息。 |
| --version | 打印详细的版本信息。 |

## 5.2本地构建

* + 1. 构建开发环境
* 操作步骤

用户可以直接使用DNF工具安装 rpmdevtools，其中包含 rpm-build 等命令以及相关依赖（例如make、gdb）。使用如下命令：



* + 1. 创建Hello World RPM包

这里以GNU “Hello World” 项目的打包过程作为示例，包含了典型的FOSS（Free and Open Source Software） 软件项目相关的最常用的外围组件，其中包括配置/编译/安装环境、文档、国际化等等

* 下载源码

我们直接下载官方例子的源码，使用如下命令：



* 编辑SPEC文件

在\*\*~/rpmbuild/SPECS\*\*目录下新建spec文件，参考命令如下：



在文件中写入对应内容后保存文件。文件内容示例如下，请根据实际情况修改相应字段。

Name: hello

Version: 2.10

Release: 1%{?dist}

Summary: The "Hello World" program from GNU

Summary(zh\_CN): GNU "Hello World" 程序

License: GPLv3+

URL: http://ftp.gnu.org/gnu/hello

Source0: http://ftp.gnu.org/gnu/hello/%{name}-%{version}.tar.gz

BuildRequires: gettext

Requires(post): info

Requires(preun): info

%description

The "Hello World" program, done with all bells and whistles of a proper FOSS

project, including configuration, build, internationalization, help files, etc.

%description -l zh\_CN

"Hello World" 程序, 包含 FOSS 项目所需的所有部分, 包括配置, 构建, 国际化, 帮助文件等。

%prep

%setup -q

%build

%configure

make %{?\_smp\_mflags}

%install

make install DESTDIR=%{buildroot}

%find\_lang %{name}

rm -f %{buildroot}/%{\_infodir}/dir

%post

/sbin/install-info %{\_infodir}/%{name}.info %{\_infodir}/dir || :

%preun

if [ $1 = 0 ] ; then

/sbin/install-info --delete %{\_infodir}/%{name}.info %{\_infodir}/dir || :

fi

%files -f %{name}.lang

%doc AUTHORS ChangeLog NEWS README THANKS TODO

%license COPYING

%{\_mandir}/man1/hello.1.\*

%{\_infodir}/hello.info.\*

%{\_bindir}/hello

%changelog

\* Thu Dec 26 2019 Your Name <youremail@xxx.xxx> - 2.10-1

- Update to 2.10

\* Sat Dec 3 2016 Your Name <youremail@xxx.xxx> - 2.9-1

- Update to 2.9

Name 标签是软件名，Version 标签是版本号，而 Release 标签是发布编号。

Summary 标签是简要说明，英文的话第一个字母应大写，以避免 rpmlint 工具（打包检查工具）警告。

License 标签说明软件包的协议版本，审查软件的 License 状态是打包者的职责，这可以通过检查源码或 LICENSE 文件，或与作者沟通来完成。

Group 标签过去用于按照 /usr/share/doc/rpm/GROUPS 分类软件包。目前该标记已丢弃，vim的模板还有这一条，删掉即可，不过添加该标记也不会有任何影响。%changelog 标签应包含每个 Release 所做的更改日志，尤其应包含上游的安全/漏洞补丁的说明。%changelog 条目应包含版本字符串，以避免 rpmlint 工具警告。

多行的部分，如 %changelog 或 %description 由指令下一行开始，空行结束。

一些不需要的行 (如 BuildRequires 和 Requires) 可在行首使用 ‘#’ 注释。

%prep、%build、%install、%file暂时用默认的，未做任何修改。

* 构建RPM包

构建源码、二进制和包含调试信息的软件包，在spec文件所在目录执行如下命令：



执行成功后，查看结果，使用如下命令：



# 第六章 调试程序

## 6.1启动程序debug

如果要使用gdb调试程序，需要在使用gcc编译c程序时使用-g选项开启程序debug模式。

## 6.2使用gdb调试程序

在使用gdb调试程序前，需要使用dnf install gdb命令安装gdb，gdb安装后，使用gdb appname进行调试程序，appname为实际要调试的程序。

gdb常用的子命令有设置断点、查看变量、重启/退出调试等等。

run(r) ： 重新开始运行文件（run-text：加载文本文件，run-bin：加载二进制文件）

run argv[1] argv[2]：调试时命令行传参

start ：单步执行，运行程序，停在第一行执行语句

next(n) ：单步调试（逐过程，函数直接执行）

step(s)：单步调试（逐语句：跳入自定义函数内部执行）

finish：结束当前函数，返回到函数调用点

continue( c )：继续运行

list(l)：查看源代码（list-n,从第n行开始查看代码。list+ 函数名：查看具体函数）

info(i)：查看函数内部局部变量的数值

print( p )：打印值及地址

display：追踪查看具体变量值

undisplay：取消追踪观察变量

info breakpoints：查看当前设置的所有断点

watch：被设置观察点的变量发生修改时，打印显示

i watch：显示观察点

x：查看内存x/20xw 显示20个单元，16进制，4字节每单元

break+num(b)：在第num行设置断点

info breakpoints：查看当前设置的所有断点

delete breakpoints num(d)：删除第num个断点

enable breakpoints：启用断点

disable breakpoints：禁用断点

backtrace(bt) ：查看函数的调用的栈帧和层级关系 frame(f)：切换函数的栈帧 info(i)：查看函数内部局部变量的数值 finish：结束当前函数，返回到函数调用点

set：设置变量的值 display：追踪查看具体变量值 undisplay：取消追踪观察变量 watch：被设置观察点的变量发生修改时，打印显示 i watch：显示观察点

## 6.3应用程序交互监控

讲述对程序的系统调用和交互的监控

* + 1. ltrace

ltrace是一个简单地运行指定命令的程序，直到它退出。它拦截并记录动态库调用其由执行的过程调用由该过程接收。它还可以拦截和打印由程序执行的系统调用。

-a, --align column

对齐特定列中的返回值（默认列是屏幕宽度的5/8）。

-A maxelts

之前要打印的最大数组元素数用省略号（“…”）抑制其余部分。这也是限制递归结构扩展的数量。

-b, --no-signals

禁止打印被跟踪者接收的信号过程

-c计算每次库调用的时间和调用，并报告程序退出时的摘要。

-C, --demangle

将低级符号名称解码（解映射）为用户级名称。除了删除任何初始下划线前缀这使得C++函数名可读。

-D, --debug mask

显示ltrace本身的调试输出。mask是一个数字描述应显示哪些调试消息。使用选项-Dh查看可以使用的内容，但请注意目前唯一可靠的调试掩码是77，它显示所有调试消息。

-e filter

修改库调用的限定表达式（即通过PLT插槽进行的调用从主二进制文件到库或库间文件的调用调用）进行跟踪。用法示例和语法描述出现在以下过滤器规格和过滤器部分表达。如果在命令行，匹配其中任何一个的库调用是跟踪。如果未给出-e，则假定@MAIN为默认值。

-f跟踪当前创建的子进程

由于分叉（2）或克隆（2）而跟踪的进程系统调用。新流程将立即附加。

-F、 --config pathlist

包含以冒号分隔的路径列表。如果路径指目录，当搜索原型库（请参见原型库发现）。如果它指的是文件，该文件将隐式导入所有加载的原型库。

-i在库运行时打印指令指针调用

-l、--library library\_pattern

仅显示对库实现的函数的调用匹配library\_patter。这就像您指定的指定库中实现的每个符号一个-e模式。可以使用多种库模式使用此选项的多个实例指定。用法librarypattern的示例和语法描述出现在以下过滤器规格和过滤器部分表达。请注意，虽然此选项选择可能指向选定的库保证由于以下原因，电话不会被转接到其他地方e、 g.LD\_PRELOAD或简单的依赖排序。如果你希望确保给定库中的符号实际调用时，请改用-x@library\_path。

-L当没有给定-e选项时，不要假定默认值@MAIN的动作。实际上，这意味着图书馆调用将不会被跟踪。

-n、 --indent nr

按每个调用级别的nr个空格缩进跟踪输出嵌套。使用此选项可使程序流动可视化易于理解。这种缩进也是无用的永不返回的函数，例如的服务函数在C++运行时引发异常。

-o、--output filename

将跟踪输出写入文件名，而不是写入标准错误。

-p pid附加到具有进程ID pid的进程并开始跟踪。此选项可以与传递要执行的命令。可以连接到多个通过传递多个选项-p来处理。

-r打印跟踪的每一行的相对时间戳，这记录了连续行。

-s strsize指定要打印的最大字符串大小（默认值为32).

-S显示系统调用和库调用

-t在跟踪的每行前面加上一天的时间。

-tt如果给出两次，打印的时间将包括微秒。

-ttt如果给三次，打印的时间将增加

* + 1. strace

strace命令是一个集诊断、调试、统计于一体的工具,常用来跟踪进程执行时的系统调用和所接收的信号，我们可以用它来监控用户空间进程和内核的交互。如对应用程序的系统调用、信号传递与进程状态变更等进行跟踪与分析，以达到解决问题的目的。 strace常用来跟踪进程执行时的系统调用和接收的信号。 在Linux中，进程不能直接访问硬件设备，当进程需要访问硬件设备(比如，读取磁盘文件，接收网络数据等等)时，必须由用户态模式切换至内核态模式，通过系统调用访问硬件设备。strace可以跟踪到一个进程产生的系统调用，包括参数，返回值，执行消耗的时间。

-c 统计每一系统调用的所执行的时间,次数和出错的次数等；

-d 输出strace关于标准错误的调试信息

-f 跟踪由fork调用所产生的子进程

-ff 如果提供-o filename,则所有进程的跟踪结果输出到相应的filename.pid中,pid是各进程的进程号

-F 尝试跟踪vfork调用.在-f时,vfork不被跟踪

-h 输出简要的帮助信息

-i 输出系统调用的入口指针

-q 禁止输出关于脱离的消息

-r 打印出相对时间关于每一个系统调用

-t 在输出中的每一行前加上时间信息

-tt 在输出中的每一行前加上时间信息,微秒级

-ttt 微秒级输出,以秒了表示时间

-T 显示每一调用所耗的时间

-v 输出所有的系统调用，一些调用关于环境变量,状态,输入输出等调用由于使用频繁,默认不输出

-V 输出strace的版本信息

-x 以十六进制形式输出非标准字符串

-xx 所有字符串以十六进制形式输出

-a column设置返回值的输出位置.默认 为40

-e expr指定一个表达式,用来控制如何跟踪.格式如下:

[qualifier=][!]value1[,value2]...

qualifier只能是 trace,abbrev,verbose,raw,signal,read,write其中之一，value是用来限定的符号或数字，默认的 qualifier是 trace，感叹号是否定符号，例如:-eopen等价于 -e trace=open,表示只跟踪open调用，而-etrace!=open表示跟踪除了open以外的其他调用.有两个特殊的符号 all 和 none.注意有些shell使用!来执行历史记录里的命令,所以要使用\\

-e trace=set只跟踪指定的系统调用.例如:

-e trace=open,close,rean,write表示只跟踪这四个系统调用.默认的为set=all

-e trace=file只跟踪有关文件操作的系统调用

-e trace=process只跟踪有关进程控制的系统调用

-e trace=network跟踪与网络有关的所有系统调用

-e strace=signal跟踪所有与系统信号有关的系统调用

-e trace=ipc跟踪所有与进程通讯有关的系统调用

-e abbrev=set设定 strace输出的系统调用的结果集

-v 等与 abbrev=none，默认为abbrev=all

-e raw=set将指 定的系统调用的参数以十六进制显示

-e signal=set指定跟踪的系统信号.默认为all，如signal=!SIGIO(或者signal=!io),表示不跟踪SIGIO信号

-e read=set输出从指定文件中读出 的数据.例如:

-e read=3,5-e write=set输出写入到指定文件中的数据

-o filename将strace的输出写入文件filename-p pid跟踪指定的进程pid

-s strsize指定输出的字符串的最大长度.默认为32.文件名一直全部输出

-u username以username 的UID和GID执行被跟踪的命令

# 第七章 FAQ

## 7.1部分依赖java-devel的应用程序自编译失败

* 问题描述

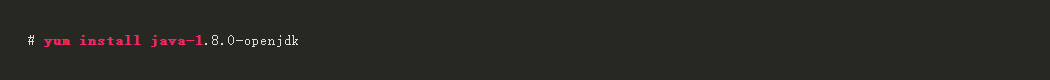
部分依赖java-devel的应用程序会出现使用rpmbuild命令自编译失败的问题。

* 原因分析

为了提供更新的openjdk特性和对广大java应用程序的兼容，openEuler同时提供了openjdk-1.8.0、openjdk-11等多个版本的openjdk。部分应用程序在编译时需要依赖java-devel包，安装java-devel包时系统会默认安装更高版本的java-11-openjdk，从而导致这些应用的编译失败。

* 解决方法

用户需手动使用如下命令安装java-1.8.0-openjdk后再使用rpmbuild命令进行自编译。



# 第八章 安装obs工具

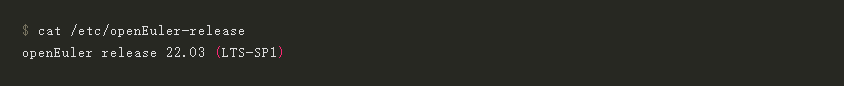
## 8.1介绍

OBS（Open Build Service）是通用编译工具，用于将源码包构建为 RPM 软件包或 Linux 镜像。 obs-server 是 OBS 对应的软件包。

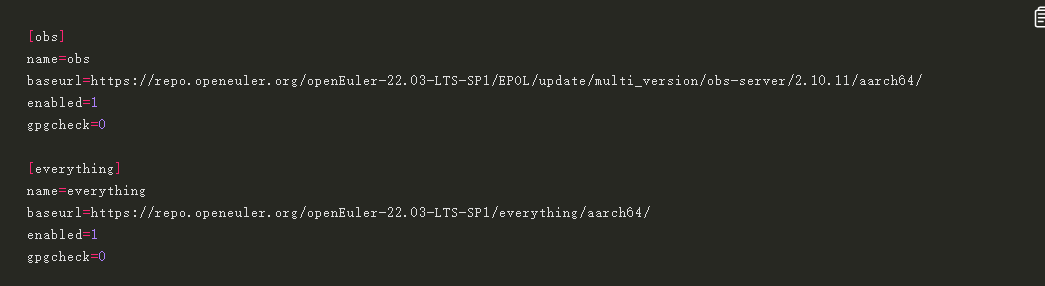
## 8.2架构适配

支持在多种架构平台（x86\_64、AArch64 等）上使用。

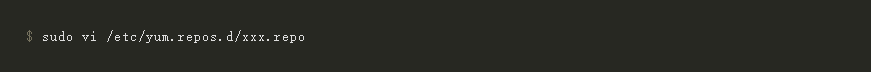
## 8.3安装

此处以 openEuler 22.03 LTS SP1版本的 AArch64 架构为例，介绍 obs-server 多版本分支软件包的安装操作。1. 确认设备已安装操作系统 openEuler 22.03 LTS SP1。

2. 配置 yum 源，obs-server 多版本的 repo 源要放置于 everything 的 repo 源前面。yum 源配置内容参考如下：



执行下述命令拷贝上述文本到 repo 源中：



3. 使 yum 源生效。



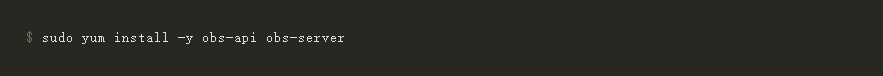
4. 检查是否已安装其他版本的软件包。

$ sudo rpm -qa obs-server obs-common obs-api mod\_passenger obs-api-deps obs-bundled-gems passenger ruby ruby-help ruby-irb rubygem-bundler rubygem-io-console rubygem-json rubygem-openssl rubygem-psych rubygem-rake rubygem-rdoc rubygems rubygem-bigdecimal rubygem-did\_you\_mean

5. （可选）如果有其他版本的软件包，请卸载软件包防止安装冲突。

$ sudo yum remove -y obs-server obs-common obs-api mod\_passenger obs-api-deps obs-bundled-gems passenger ruby ruby-help ruby-irb rubygem-bundler rubygem-io-console rubygem-json rubygem-openssl rubygem-psych rubygem-rake rubygem-rdoc rubygems rubygem-bigdecimal rubygem-did\_you\_mean

6. 安装 obs-server 相关软件包。



7. 检查 obs-server 软件包安装成功。



## 8.4部署

获取部署脚本，具体请参见获取链接： https://gitee.com/openeuler/infrastructure/tree/master/obs/tf/startup执行 restart\_service.sh 脚本部署 OBS 工具。