sszy6789

13719442657友善陈工

g++ -fPIC -shared -o libRfidcom.so SLR11002.cpp

ls -l /dev/tty\*

spyder3

1. 查看python 32还是64位 >>> **import** platform
2. >>> platform.architecture()

rm -rf 目录名字

serial module安装

我使用的是Ubuntu16.04 64bit系统，其他系统应该也都类似的。

首先需要在终端输入：

sudo apt install python-pip //python2

sudo apt install python3-pip //python3

如果系统已经安装过了pip，就不需要这一步了。

然后通过：

sudo pip install pyserial //python2

sudo pip3 install pyserial //python3

就安装好serial module在本机了。

切换到root用户： sudo su

退出root用户：exit，退回到user用户

sudo mount -t vfat /dev/sda1 /mnt 挂接sd

umount /dev/sdb1

/opt/pycharm-community-2017.3.3/bin# sh /opt/pycharm-community-2017.3.3/bin/pycharm.sh &

./bin/emqttd console

sqlitebrowser

tar xzf redis-2.8.17.tar.gz

sudo apt-get install sqlitebrowser

#uwsgi --ini /opt/RfidProject/uwsgi9090.ini

Python3 manage.py runserver 192.168.1.105:9090

. /opt/redis-4.0.6/src/redis-server /opt/redis-4.0.6/src/redis.conf

Redis的安装很简单:

wget http://download.redis.io/releases/redis-3.0.6.tar.gz

tar xzf redis-3.0.6.tar.gz

cd redis-3.0.6

make

启动服务端:

src/redis-server &

启动客户端并操作:

src/redis-cli

redis> set foo bar

OK

redis> get foo

"bar"

把daemonize设置为yes后台运行

确认是否启动，可以使用 ps -ef|grep redis

redis操作 <http://doc.redisfans.com/> 命令

/opt/redis-4.0.6/src/

redis-cli 客户端 redis-server 服务端 redis-server /opt/redis-4.0.6/src/redis.conf

AUTH 密码

#CONFIG GET \*

#redis> CONFIG GET slowlog-max-len

#1) "slowlog-max-len"

#2) "1024"

#redis> CONFIG SET slowlog-max-len 10086

#OK

#redis> CONFIG GET slowlog-max-len

#1) "slowlog-max-len"

#2) "10086"

import redis

import configparser

CONFIG = configparser.ConfigParser()

CONFIG.read("config/system.ini")

redis\_host = CONFIG.get("redis", "REDIS\_HOST")

redis\_port = CONFIG.get("redis","REDIS\_PORT")

redis\_db=CONFIG.get("redis","REDIS\_DB")

redis\_pwd=CONFIG.get("redis","REDIS\_PASSWORD")

redisConnect = redis.Redis(redis\_host,redis\_port,redis\_db,redis\_pwd)

class RedisTool:

@staticmethod

def hexists(name,key):

return redisConnect.hexists(name,key)

@staticmethod

def hget(name, key):

return redisConnect.hget(name, key)

@staticmethod

def getset(name, value):

return redisConnect.getset(name, value)

@staticmethod

def hdel(name, \*keys):

return redisConnect.hdel(name, \*keys)

@staticmethod

def hgetall(name):

return redisConnect.hgetall(name)

@staticmethod

def hkeys(name):

return redisConnect.hkeys(name)

@staticmethod

def hlen(name):

return redisConnect.hlen(name)

#Set key to value within hash name Returns 1 if HSET created a new field, otherwise 0

@staticmethod

def hset(name, key, value):

return redisConnect.hset(name, key, value)

@staticmethod

def setex(name, time, value):

return redisConnect.setex(name, time, value)

@staticmethod

def get(name):

return redisConnect.get(name)

@staticmethod

def exists(name):

return redisConnect.exists(name)

@staticmethod

def set(name, value):

return redisConnect.set(name, value)

首先去python官网下载python3的源码包，网址：https://www.python.org/点进去之后页面下方可以看到下载链接

选择Download Gzipped source tarball，下载下来后就是Python-3.6.1.tgz打开终端，利用命令cd 进入文件所在文件夹里

释放文件

tar xfz Python-3.6.1.tgz

这里使用xfz命令，而不建议使用-xvzf命令，因为其释放的文件夹需要root权限才可以更改或者删除。

进入释放的文件夹

cd Python-3.6.1/

添加配置

./configure –prefix=/usr/bin/python3.6

要配置的是安装Python需要的文件夹。配置完成后，终端会显示下面一段话，可以选择无视它，不影响后续操作。

If you want a release build with all optimizations active (LTO, PGO, etc),

please run ./configure –enable-optimizations

编译源码

sudo make

执行安装

sudo make install

安装过程持续几分钟，安装成功后，安装的Python就在 计算机/usr/python3.6 文件夹之中。

修改默认Python版本

Ubuntu16.04版本集成了Python2.7及3.5版本，系统中原来的python在/usr/bin/python，通过ls -l可以看到，python是一个软链接，链接到本目录下的python2.6

我们需要先把这个默认链接删除，再新建一个python3.6的软链接.

首先要确保终端的指针已经退回根目录，再进行以下操作

需要删除原有的Python链接文件:

rm /usr/bin/python

注意如果终端提示：rm: 无法删除’/usr/bin/python’: 权限不够，那么就需要打开root权限删除链接。

建立指向Python3.6的链接：

ln -s /usr/bin/python3.6/bin/python3.6 /usr/bin/python

查询Python版本

python –version

如果显示是python3.6.1则说明安装成功！

如果ubuntu修改文件的时候一直提示：

sudo：unable to resolve host abc

* 1

那么就需要修改/etc/hosts文件：   
如果之前为

127.0.0.1 localhost

* 1

变更为：

127.0.0.1 localhost

127.0.1.1 hostname

* 1
* 2

hostname 为主机名，查看文件：/etc/hostname 获得

////////////////////////////////////////////////////////////////////////////////

在Ubuntu中，有时候运用sudo  apt-get install 安装软件时，会出现一下的情况

E: Could not get lock /var/lib/dpkg/lock - open (11: Resource temporarily unavailable)  
E: Unable to lock the administration directory (/var/lib/dpkg/), is another process using it?

在这个时候，主要是因为apt还在运行，此时的解决方案是

**1、找到并且杀掉所有的apt-get 和apt进程**

    运行下面的命令来生成所有含有 apt 的进程列表，你可以使用ps和grep命令并用管道组合来得到含有apt或者apt-get的进程。

ps -A | grep apt

找出所有的 apt 以及 apt-get 进程

$ sudo kill -9 processnumber

或者

$ sudo kill -SIGKILL processnumber

比如，下面命令中的9是 SIGKILL 的信号数，它会杀掉第一个 apt 进程

$ sudo kill -9 进程ID

或者

$ sudo kill -SIGKILL 进程ID

2、删除锁定文件

锁定的文件会阻止 Linux 系统中某些文件或者数据的访问，这个概念也存在于 Windows 或者其他的操作系统中。

一旦你运行了 apt-get 或者 apt 命令，锁定文件将会创建于 /var/lib/apt/lists/、/var/lib/dpkg/、/var/cache/apt/archives/ 中。

这有助于运行中的 apt-get 或者 apt 进程能够避免被其它需要使用相同文件的用户或者系统进程所打断。当该进程执行完毕后，锁定文件将会删除。

   当你没有看到 apt-get 或者 apt 进程的情况下在上面两个不同的文件夹中看到了锁定文件，这是因为进程由于某个原因被杀掉了，因此你需要删除锁定文件来避免该错误。

首先运行下面的命令来移除 /var/lib/dpkg/ 文件夹下的锁定文件：

$ sudo rm /var/lib/dpkg/lock

之后像下面这样强制重新配置软件包：

$ sudo dpkg --configure -a

也可以删除 /var/lib/apt/lists/ 以及缓存文件夹下的锁定文件：

$ sudo rm /var/lib/apt/lists/lock

$ sudo rm /var/cache/apt/archives/lock

接下来，更新你的软件包源列表：

$ sudo apt update

或者

$ sudo apt-get update

总结一下，对于 Ubuntu（以及它的衍生版）用户在使用 apt-get 或者 apt 也叫 [aptitude 命令](http://www.tecmint.com/difference-between-apt-and-aptitude/)[7]时遇到的问题，我们已经用两种方法来解决了。

Linux 技巧：让进程在后台可靠运行的几种方法

申 毅

2008 年 5 月 29 日发布

Weibo

Google+

用电子邮件发送本页面

Comments

11

我们经常会碰到这样的问题，用 telnet/ssh 登录了远程的 Linux 服务器，运行了一些耗时较长的任务， 结果却由于网络的不稳定导致任务中途失败。如何让命令提交后不受本地关闭终端窗口/网络断开连接的干扰呢？下面举了一些例子， 您可以针对不同的场景选择不同的方式来处理这个问题。

nohup/setsid/&

场景：

如果只是临时有一个命令需要长时间运行，什么方法能最简便的保证它在后台稳定运行呢？

hangup 名称的来由

在 Unix 的早期版本中，每个终端都会通过 modem 和系统通讯。当用户 logout 时，modem 就会挂断（hang up）电话。 同理，当 modem 断开连接时，就会给终端发送 hangup 信号来通知其关闭所有子进程。

解决方法：

我们知道，当用户注销（logout）或者网络断开时，终端会收到 HUP（hangup）信号从而关闭其所有子进程。因此，我们的解决办法就有两种途径：要么让进程忽略 HUP 信号，要么让进程运行在新的会话里从而成为不属于此终端的子进程。

1. nohup

nohup 无疑是我们首先想到的办法。顾名思义，nohup 的用途就是让提交的命令忽略 hangup 信号。让我们先来看一下 nohup 的帮助信息：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

NOHUP(1) User Commands NOHUP(1)

NAME

nohup - run a command immune to hangups, with output to a non-tty

SYNOPSIS

nohup COMMAND [ARG]...

nohup OPTION

DESCRIPTION

Run COMMAND, ignoring hangup signals.

--help display this help and exit

--version

output version information and exit

可见，nohup 的使用是十分方便的，只需在要处理的命令前加上 nohup 即可，标准输出和标准错误缺省会被重定向到 nohup.out 文件中。一般我们可在结尾加上"&"来将命令同时放入后台运行，也可用">filename 2>&1"来更改缺省的重定向文件名。

nohup 示例

1

2

3

4

5

6

7

[root@pvcent107 ~]# nohup ping www.ibm.com &

[1] 3059

nohup: appending output to `nohup.out'

[root@pvcent107 ~]# ps -ef |grep 3059

root 3059 984 0 21:06 pts/3 00:00:00 ping www.ibm.com

root 3067 984 0 21:06 pts/3 00:00:00 grep 3059

[root@pvcent107 ~]#

2。setsid

nohup 无疑能通过忽略 HUP 信号来使我们的进程避免中途被中断，但如果我们换个角度思考，如果我们的进程不属于接受 HUP 信号的终端的子进程，那么自然也就不会受到 HUP 信号的影响了。setsid 就能帮助我们做到这一点。让我们先来看一下 setsid 的帮助信息：

1

2

3

4

5

6

7

8

9

10

SETSID(8) Linux Programmer’s Manual SETSID(8)

NAME

setsid - run a program in a new session

SYNOPSIS

setsid program [ arg ... ]

DESCRIPTION

setsid runs a program in a new session.

可见 setsid 的使用也是非常方便的，也只需在要处理的命令前加上 setsid 即可。

setsid 示例

1

2

3

4

5

[root@pvcent107 ~]# setsid ping www.ibm.com

[root@pvcent107 ~]# ps -ef |grep www.ibm.com

root 31094 1 0 07:28 ? 00:00:00 ping www.ibm.com

root 31102 29217 0 07:29 pts/4 00:00:00 grep www.ibm.com

[root@pvcent107 ~]#

值得注意的是，上例中我们的进程 ID(PID)为31094，而它的父 ID（PPID）为1（即为 init 进程 ID），并不是当前终端的进程 ID。请将此例与nohup 例中的父 ID 做比较。

3。&

这里还有一个关于 subshell 的小技巧。我们知道，将一个或多个命名包含在“()”中就能让这些命令在子 shell 中运行中，从而扩展出很多有趣的功能，我们现在要讨论的就是其中之一。

当我们将"&"也放入“()”内之后，我们就会发现所提交的作业并不在作业列表中，也就是说，是无法通过jobs来查看的。让我们来看看为什么这样就能躲过 HUP 信号的影响吧。

subshell 示例

1

2

3

4

5

[root@pvcent107 ~]# (ping www.ibm.com &)

[root@pvcent107 ~]# ps -ef |grep www.ibm.com

root 16270 1 0 14:13 pts/4 00:00:00 ping www.ibm.com

root 16278 15362 0 14:13 pts/4 00:00:00 grep www.ibm.com

[root@pvcent107 ~]#

从上例中可以看出，新提交的进程的父 ID（PPID）为1（init 进程的 PID），并不是当前终端的进程 ID。因此并不属于当前终端的子进程，从而也就不会受到当前终端的 HUP 信号的影响了。

disown

场景：

我们已经知道，如果事先在命令前加上 nohup 或者 setsid 就可以避免 HUP 信号的影响。但是如果我们未加任何处理就已经提交了命令，该如何补救才能让它避免 HUP 信号的影响呢？

解决方法：

这时想加 nohup 或者 setsid 已经为时已晚，只能通过作业调度和 disown 来解决这个问题了。让我们来看一下 disown 的帮助信息：

1

2

3

4

5

6

7

8

9

10

11

disown [-ar] [-h] [jobspec ...]

Without options, each jobspec is removed from the table of

active jobs. If the -h option is given, each jobspec is not

removed from the table, but is marked so that SIGHUP is not

sent to the job if the shell receives a SIGHUP. If no jobspec

is present, and neither the -a nor the -r option is supplied,

the current job is used. If no jobspec is supplied, the -a

option means to remove or mark all jobs; the -r option without

a jobspec argument restricts operation to running jobs. The

return value is 0 unless a jobspec does not specify a valid

job.

可以看出，我们可以用如下方式来达成我们的目的。

灵活运用 CTRL-z

在我们的日常工作中，我们可以用 CTRL-z 来将当前进程挂起到后台暂停运行，执行一些别的操作，然后再用 fg 来将挂起的进程重新放回前台（也可用 bg 来将挂起的进程放在后台）继续运行。这样我们就可以在一个终端内灵活切换运行多个任务，这一点在调试代码时尤为有用。因为将代码编辑器挂起到后台再重新放回时，光标定位仍然停留在上次挂起时的位置，避免了重新定位的麻烦。

用disown -h jobspec来使某个作业忽略HUP信号。

用disown -ah 来使所有的作业都忽略HUP信号。

用disown -rh 来使正在运行的作业忽略HUP信号。

需要注意的是，当使用过 disown 之后，会将把目标作业从作业列表中移除，我们将不能再使用jobs来查看它，但是依然能够用ps -ef查找到它。

但是还有一个问题，这种方法的操作对象是作业，如果我们在运行命令时在结尾加了"&"来使它成为一个作业并在后台运行，那么就万事大吉了，我们可以通过jobs命令来得到所有作业的列表。但是如果并没有把当前命令作为作业来运行，如何才能得到它的作业号呢？答案就是用 CTRL-z（按住Ctrl键的同时按住z键）了！

CTRL-z 的用途就是将当前进程挂起（Suspend），然后我们就可以用jobs命令来查询它的作业号，再用bg jobspec来将它放入后台并继续运行。需要注意的是，如果挂起会影响当前进程的运行结果，请慎用此方法。

disown 示例1（如果提交命令时已经用“&”将命令放入后台运行，则可以直接使用“disown”）

1

2

3

4

5

6

7

8

9

[root@pvcent107 build]# cp -r testLargeFile largeFile &

[1] 4825

[root@pvcent107 build]# jobs

[1]+ Running cp -i -r testLargeFile largeFile &

[root@pvcent107 build]# disown -h %1

[root@pvcent107 build]# ps -ef |grep largeFile

root 4825 968 1 09:46 pts/4 00:00:00 cp -i -r testLargeFile largeFile

root 4853 968 0 09:46 pts/4 00:00:00 grep largeFile

[root@pvcent107 build]# logout

disown 示例2（如果提交命令时未使用“&”将命令放入后台运行，可使用 CTRL-z 和“bg”将其放入后台，再使用“disown”）

1

2

3

4

5

6

7

8

9

10

11

12

[root@pvcent107 build]# cp -r testLargeFile largeFile2

[1]+ Stopped cp -i -r testLargeFile largeFile2

[root@pvcent107 build]# bg %1

[1]+ cp -i -r testLargeFile largeFile2 &

[root@pvcent107 build]# jobs

[1]+ Running cp -i -r testLargeFile largeFile2 &

[root@pvcent107 build]# disown -h %1

[root@pvcent107 build]# ps -ef |grep largeFile2

root 5790 5577 1 10:04 pts/3 00:00:00 cp -i -r testLargeFile largeFile2

root 5824 5577 0 10:05 pts/3 00:00:00 grep largeFile2

[root@pvcent107 build]#

screen

场景：

我们已经知道了如何让进程免受 HUP 信号的影响，但是如果有大量这种命令需要在稳定的后台里运行，如何避免对每条命令都做这样的操作呢？

解决方法：

此时最方便的方法就是 screen 了。简单的说，screen 提供了 ANSI/VT100 的终端模拟器，使它能够在一个真实终端下运行多个全屏的伪终端。screen 的参数很多，具有很强大的功能，我们在此仅介绍其常用功能以及简要分析一下为什么使用 screen 能够避免 HUP 信号的影响。我们先看一下 screen 的帮助信息：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

SCREEN(1) SCREEN(1)

NAME

screen - screen manager with VT100/ANSI terminal emulation

SYNOPSIS

screen [ -options ] [ cmd [ args ] ]

screen -r [[pid.]tty[.host]]

screen -r sessionowner/[[pid.]tty[.host]]

DESCRIPTION

Screen is a full-screen window manager that multiplexes a physical

terminal between several processes (typically interactive shells).

Each virtual terminal provides the functions of a DEC VT100 terminal

and, in addition, several control functions from the ISO 6429 (ECMA

48, ANSI X3.64) and ISO 2022 standards (e.g. insert/delete line and

support for multiple character sets). There is a scrollback history

buffer for each virtual terminal and a copy-and-paste mechanism that

allows moving text regions between windows.

使用 screen 很方便，有以下几个常用选项：

用screen -dmS session name来建立一个处于断开模式下的会话（并指定其会话名）。

用screen -list 来列出所有会话。

用screen -r session name来重新连接指定会话。

用快捷键CTRL-a d 来暂时断开当前会话。

screen 示例

1

2

3

4

5

6

7

[root@pvcent107 ~]# screen -dmS Urumchi

[root@pvcent107 ~]# screen -list

There is a screen on:

12842.Urumchi (Detached)

1 Socket in /tmp/screens/S-root.

[root@pvcent107 ~]# screen -r Urumchi

当我们用“-r”连接到 screen 会话后，我们就可以在这个伪终端里面为所欲为，再也不用担心 HUP 信号会对我们的进程造成影响，也不用给每个命令前都加上“nohup”或者“setsid”了。这是为什么呢？让我来看一下下面两个例子吧。

1. 未使用 screen 时新进程的进程树

1

2

3

4

5

6

7

8

9

[root@pvcent107 ~]# ping www.google.com &

[1] 9499

[root@pvcent107 ~]# pstree -H 9499

init─┬─Xvnc

├─acpid

├─atd

├─2\*[sendmail]

├─sshd─┬─sshd───bash───pstree

│ └─sshd───bash───ping

我们可以看出，未使用 screen 时我们所处的 bash 是 sshd 的子进程，当 ssh 断开连接时，HUP 信号自然会影响到它下面的所有子进程（包括我们新建立的 ping 进程）。

2. 使用了 screen 后新进程的进程树

1

2

3

4

5

6

7

8

9

[root@pvcent107 ~]# screen -r Urumchi

[root@pvcent107 ~]# ping www.ibm.com &

[1] 9488

[root@pvcent107 ~]# pstree -H 9488

init─┬─Xvnc

├─acpid

├─atd

├─screen───bash───ping

├─2\*[sendmail]

而使用了 screen 后就不同了，此时 bash 是 screen 的子进程，而 screen 是 init（PID为1）的子进程。那么当 ssh 断开连接时，HUP 信号自然不会影响到 screen 下面的子进程了。

总结

现在几种方法已经介绍完毕，我们可以根据不同的场景来选择不同的方案。nohup/setsid 无疑是临时需要时最方便的方法，disown 能帮助我们来事后补救当前已经在运行了的作业，而 screen 则是在大批量操作时不二的选择了

一、添加开机自启服务

在CentOS 7中添加开机自启服务非常方便，只需要两条命令(以Jenkins为例)：

systemctl enable jenkins.service #设置jenkins服务为自启动服务

sysstemctl start jenkins.service #启动jenkins服务

二、添加开机自启脚本

在centos7中增加脚本有两种常用的方法，以脚本autostart.sh为例：

#!/bin/bash

#description:开机自启脚本

/usr/local/tomcat/bin/startup.sh #启动tomcat

方法一

1、赋予脚本可执行权限（/opt/script/autostart.sh是你的脚本路径）

chmod +x /opt/script/autostart.sh

2、打开/etc/rc.d/rc/local文件，在末尾增加如下内容

/opt/script/autostart.sh

3、在centos7中，/etc/rc.d/rc.local的权限被降低了，所以需要执行如下命令赋予其可执行权限

chmod +x /etc/rc.d/rc.local

方法二

1、将脚本移动到/etc/rc.d/init.d目录下

mv /opt/script/autostart.sh /etc/rc.d/init.d

2、增加脚本的可执行权限

chmod +x /etc/rc.d/init.d/autostart.sh

3、添加脚本到开机自动启动项目中

cd /etc/rc.d/init.d

chkconfig --add autostart.sh

chkconfig autostart.sh on

<http://www.runoob.com/django/django-nginx-uwsgi.html>

安装 计划任务crond

sudo apt-get install cron

重启

检测你的service路径

执行命令 which service

root@ubuntu4146:/data/gitlabData/backups# which service

/usr/sbin/service

比如执行命令 sudo /usr/sbin/service cron start ，启动 cron 服务

上面命令返回 Job is already running: cron 表示 cron命令早已经在运行中，   
通常你可以通过执行命令 ps -ef | grep cron查看 cron命令是否正在运行

执行命令 sudo /usr/sbin/service cron status , 查看crontab服务状态

执行命令 sudo /usr/sbin/service cron reload 重新 cron 服务配置

执行命令 sudo /usr/sbin/service cron restart , 重新启动 cron 服务

执行命令 sudo /usr/sbin/service cron stop , 停止 cron 服务

/usr/lib/python3.6/site-packages/

Python EmbeddedCom.py

Cd /opt/redis-4.0.8/src

./redis-server /opt/redis-4.0.8/src/redis.conf &