

CBIB Ubuntu Server Manual

version	author	date	diff	other
V1.0	王彬	2018.09.06	Setup	initialization

CBIB Ubuntu Server Manual

前言

第一章 服务器介绍

1.1 硬件

1.2 软件

1.3 IP地址

第二章 服务器配置

2.1 深度学习框架配置

2.1.1 深度学习库依赖关系

2.1.2 Linux加载库过程简介

2.1.3 Linux环境变量的配置

2.1.4 Driver、CUDA、cudnn下载

2.1.5 Driver、CUDA、cudnn安装

2.1.6 Tensorflow/Keras的安装

2.2 固定IP配置

2.2.1 校园网的组成

2.2.2 CBIB实验室局域网组成

2.2.3 固定IP的配置

2.3 SSH 服务配置

2.4 远程桌面配置

2.5 远程传输配置(ftp/samba/nfs)

2.6 sublime+sftp 远程文件同步

第三章 Linux运维

3.1 Linux用户管理

3.1.1 Linux下的三类用户

3.1.2 linux 文件权限

第四章 服务器管理

第五章 常见问题及解决方案

前言

此指导书作为CBIB实验室公用Ubuntu服务器指导资料，包含但不局限于深度学习环境搭建、服务器运维、服务器日常维修、硬件维护、问题定位以及解决措施，仅供参考。除此之外还配备常见FAQ，见[README.md](#)。希望以此传承下去，为后续师兄弟减少环境搭建工作量，节约学习时间。

请历届管理员严格把守以下规则：

- 服务其ip地址不得以任何形式对外泄露

- 不得将服务器账号租、借给实验室意外人员
- 不使用GPU（仅占用CPU）的项目不得使用服务器
- 实验室老成员毕业离校后，及时回收账号
- 同一项目同一程序一次不得使用超过4块GPU，特殊情况除外
- 任何人不得私自删除、升级**公用库**（cuda、cudnn、tensorflow、keras、pytorch、caffa等），需经过管理员和大家一致同意后方可升级
- 普通用户（非管路员）不得修改 `/etc/profile`、`/etc/bashrc` 等公共配置文件
- 每人限制一个服务器账号，不使用远程桌面（或使用频率低于2次/天）的用户，远程桌面应该处于常关状态
- 除管理员和负责人(金人超老师)以外，任何人不得使用root登录，root密码不得向任何人泄露
- 服务器机房每天必须检查一次，确保空调处于开启状态，室内温度不得超过40度
- 每隔一段时间要检查服务器机架后侧电线，防止温度过高造成电线老化漏电

此项目自2018.9.6发起，手册开源于github，由每一届服务器运维管理员负责维护。

2017.9~2018.9 管理员codewang， github地址<https://github.com/deeper-code/deeper-server>

第一章 服务器介绍

Note: 后文中Linux发行版特指Ubuntu.本手册只提供命令行操作指导，不提供图像界面操作指导。

1.1 硬件

实验室目前共2台大服务器（10-GPU），2台小服务器（2-GPU）。其硬件配置如表1-1所示。

服务器编号	俗名	GPU数量	GPU型号	显存	内存	硬盘	CPU型号	负责人
1	老服务器	10	GeForce GTX 1080 Ti	12GB	128GB	12TB	Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz (2CPU-28核-64bit)	曹海潮
2	新服务器	10	GeForce GTX 1080 Ti	12GB	128GB	4TB	Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz (2CPU-28核-64bit)	王彬
3	小服务器1	2	GeForce GTX 1080 Ti	12GB	64GB	2TB	Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz (1GPU-6核-64bit)	王彬
4	小服务器2		GeForce GTX 1080 Ti	12GB	64GB	2TB	Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz (1GPU-6核-64bit)	陆建国

表1-1 服务器硬件配置表

其中：

- 1号**老服务器**位于机房-服务器机架从上往下第一台（无外接显示器，无键鼠，与2号公用）
- 2号**新服务器**位于机房-服务器机架从上往下第二台（有外接显示器，有键鼠）
- 3号**小服务器1**位于实验室-进门第4排-右数4号桌（有外接显示器，有键鼠）

- 4号小服务器2位于机房-进门空调旁边（无外接显示器，无键鼠）

1.2 软件

软件配置涉及到大家的深度学习平台，格外重要。请管理员及时维护此表（表1-2），一旦出现私自更新、删除公用库可按照此表进行恢复。这里只统计公用数据库，公用库，公用应用程序，*表示未统计项，或不重要

软件名称	1号	2号	3号	4号	表项维护人@维护日期
OS	Ubuntu LTS 14.04(出厂)	Ubuntu LTS 14.04(出厂)	Ubuntu LTS 16.04（王彬 @2018.6）	Ubuntu LTS 16.04（陆建国 @2018.6）	codewang@2018.9.6
CUDA	9.0	8.0	9.0	9.0	codewang@2018.9.6
cudnn	*	*	8.5	*	codewang@2018.9.6
nvidia-driver	384.130	384.69	384.130	*	codewang@2018.9.6
python3	3.4.3	3.4.3	3.6.5	*	codewang@2018.9.6
Tensorflow-gpu	1.3.0	1.4.0	1.8.0	*	codewang@2018.9.6
Tensorboard	0.1.8	0.4.0	1.8.0	*	codewang@2018.9.6
Keras	2.0.8	2.1.6	2.2.0	*	codewang@2018.9.6
Pytorch	无	0.1.0	*	*	codewang@2018.9.6
Caffa	*	*	无	*	codewang@2018.9.6
Theano	*	1.0.2	无	*	codewang@2018.9.6
Mxnet	*	1.1.0.post0	无	*	codewang@2018.9.6
xgboost	0.6a2	0.71	无	*	codewang@2018.9.6
ipython	6.2.1	6.2.1	6.4.0	*	codewang@2018.9.6
Cython	0.27.3	0.27.3	0.28.2	*	codewang@2018.9.6

表1-2 服务器软件配置表

1.3 IP地址

IP地址是向学校申请的固定IP地址，如表1-3所示，校内任何地方均可以访问。切记不可将服务IP地址告诉他人。

编号	IP地址
1号	xxx.xxx.xxx.253
2号	xxx.xxx.xxx.232
3号	xxx.xxx.xxx.226
4号	xxx.xxx.xxx.225

表1-3 服务器IP配置表

第二章 服务器配置

本章内容包括：

- 如何安装cuda、cudnn、tensorflow-gpu等深度学习框架，以及其配置过程
- 固定IP配置
- 远程桌面配置<需配合第三章 Linux运维，可先阅读第三章>

2.1 深度学习框架配置

由于实验室项目不尽相同，不同的项目组可能会使用不同的深度学习框架，管理员只需用维护tensorflow和keras即可。但是cuda、cudnn是所有框架公用的，所以升级和维护需要实现与各项目组沟通协调。目前实验室所使用的深度学习框架包括：`keras with tensorflow-backend` (刘老师、马老师项目组)、`caffe` (金老师项目组)、`matlab` (许老师项目组)、`other`，下面主要介绍cudnn、cuda以及tensorflow和keras的安装配置。

2.1.1 深度学习库依赖关系

我们知道CPU也好、GPU也好、硬盘、内存都是物理设备都属于硬件，那么操作系统或者计算机软件想要使用这些硬件，就需要驱动程序，这和Window是一样的。（虽然windows上有许多设备是免驱的，但是实质上他们使用的是公用驱动，万能驱动。比如鼠标、蓝牙键盘等，其实他们都是使用默认的USB设备驱动程序，所以可以免驱），所以我们想要使用GPU首先需要安装**驱动程序**。

除此之外，tensorflow这样的深度学习框架也不可能直接和GPU驱动程序打交道，不然tensorflow的软件体量就太大了，所以NVIDIA公司为这些上层应用软件提供了一些辅助程序，称之为**cuda**和**cudnn**。

然后就是我们熟悉的深度学习框架了，比如tensorflow、pytorch、mxnet、caffe等,注意，**Keras**严格意义上来说并不是深度学习框架，它是以**tensorflow**或者**theaon**后端的高级封装API库。下面我们用一副图来了解它们的关系。如图2-1所示：

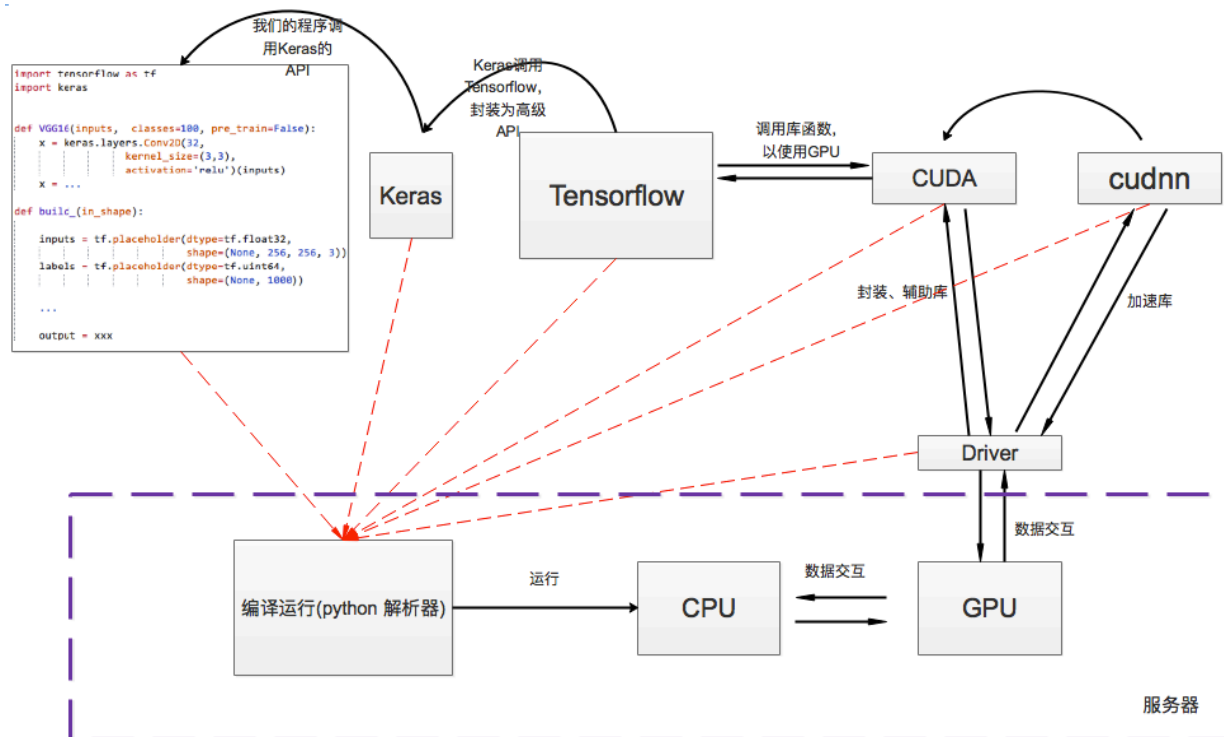


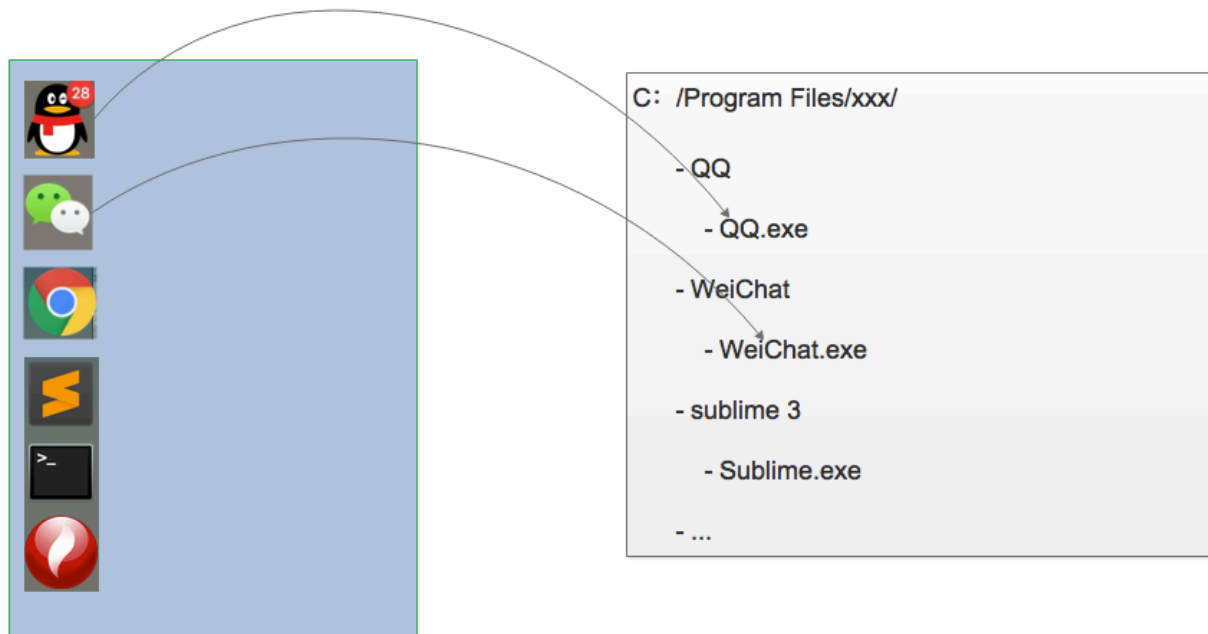
图2-1 深度学习库依赖关系

2.1.2 Linux加载库过程简介

本节，我将通过类比windows应用程序来解释Linux系统是如何启动应用程序，如何加载动态链接库的。假设我们在windows上安装一个QQ软件，其步骤如下：

- 下载QQ安装包，一般为 .exe
- 安装
 - 选择安装位置，默认为 C:/Program files/xxx
 - 创建桌面快捷方式
 - 安装完成
- 启动应用程序

实质上我们选择安装路径时，安装程序会将QQ安装包的内容，所需要的库文件，数据库等拷贝到我们指定的目录下，同时还会设置注册表（这个不清楚没关系）。然后创建桌面快捷方式比如 /Desktop/qq ,当我们点击运行 qq(快捷方式) 时系统会找到此快捷方式所指定的应用程序，也就是 C:/Program Files/xxx/qq.exe ,然后就完成启动了。



Linux下的程序启动与Window类似，但也有些不同，首先linux不存在快捷方式，那么问题就来,linux是如何找到对应的应用程序的呢？

Linux下启动应用程序，系统会根据环境变量 **PATH** 指定的路径，取对应的路径下查找应用程序。

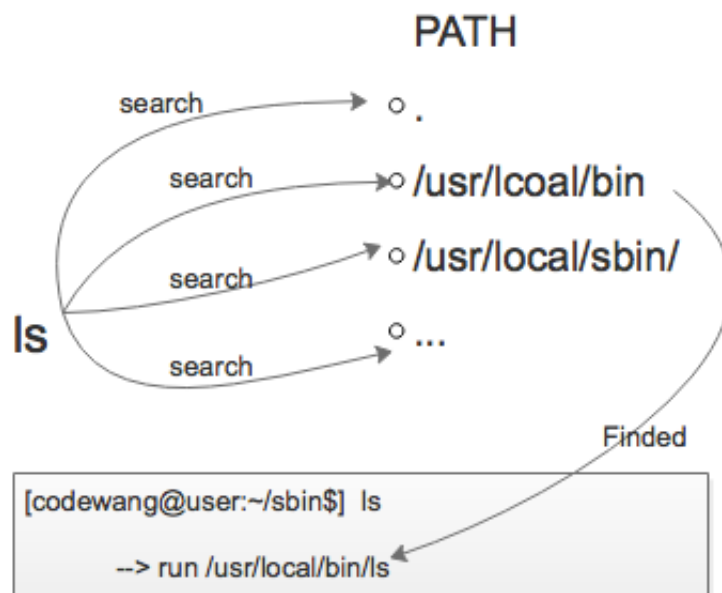
我们可以使用echo命令查看环境变量：

```
echo $PATH <--- $ 表示读取环境变量， echo是输出显示命令
```

(2号服务器)输出为：

```
/usr/local/MATLAB/R2014b/bin:/usr/local/cuda-8.0/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin
```

我们可以看到PATH环境变量中有很多路径，用:(冒号)隔开，那么假设当我们在终端键入 `ls` 命令时，实质上Linux系统会去这些路径下查找名称为 `ls` 的应用程序（默认优先查找当前路径 `.`），最终会在 `/usr/local/bin/` 找到 `ls` 程序，然后调用它，所以说我们运行 `ls` 其实是运行 `/usr/local/bin/ls`。



那么同样的道理，如果程序需要加载某个动态链接库，那么它会去哪些路径下查找呢？答案是 **LD_LIBRARY_PATH**

```
codewang@user:~/sbin$ echo $LD_LIBRARY_PATH
/usr/local/cuda-8.0/lib64:
codewang@user:~/sbin$
```

2.1.3 Linux环境变量的配置

上面我们讲了Linux运行应用程序和加载动态链接库时所依赖的两个环境变量，下面我们讲解如何设置环境变量，每个用户的环境变量有什么区别？

首先我要强调一点，**Linux系统不是桌面系统**，其设计之初就不是为日常办公而设计，它是一种多用户的服务器型的操作系统，既然Linux是多用的操作系统，那么问题就随之而来，假如某台服务器上有AB两个用户，A用户安装了一个QQ轻量版并且设置了 **PATH** 环境变量，B用户又想使用完整版的QQ，于是他也下载安装了完整版的QQ，也设置了环境变量 **PATH**，那么这就造成的冲突。因此 **Linux中每个用户的环境变量都是独立的！**

a) 如何设置环境变量 ---> export

Linux中我们使用 **export** 命令来设置环境变量，但是要主要环境变量有它自己的生命周期。设置的环境变量有效期直至当机器**重新启动**或者该环境变量被**覆盖**、被**删除**时。我们可以再命令行中使用 **export** 命令来设置环境变量，下面我们看一个例子：

```
[codewang$] echo $HELLO_WORLD    --> 查看环境变量 `HELLO_WORLD`
                                --> 该用户的该环境变量不存在，或者为空

[codewang$] export HELLO_WORLD='never fight alone'    --> 定义环境变量
[codewang$] echo $HELLO_WORLD
never fight alone    --> 可以看到已经设置成功

[codewang$] logout    --> 退出登录

-----

我们切换登录另一个用户 user
[user$] echo $HELLO_WORLD    --> 查看环境变量 `HELLO_WORLD`
                                --> 不存在，这也说明了用户的环境变量是独立的。

[user$]
```

b) 脚本设置环境变量 --> /home/username/.bashrc

上面介绍了使用 `export` 来设置环境变量，但是也面临着一个问题，一旦机器重启那我们就得重新设置环境变量，这岂不是很麻烦，好在Linux系统在启动时会自动运行一些脚本(有兴趣的同学可以查阅相应资料)，在系统启动、或者用户登录时系统自动运行该用户 `home` 下的 `.bashrc` 脚本（shell 脚本）。所以每个用户都可以自己定制自己的启动脚本，那么我们就可以在该脚本中设置环境变量，就不要再操心系统重新的问题了。

```
[codewang$] vim ~/.bashrc    --> ~的意思是当前用户的home目录，等价于/home/codewang

-+-+--+ 以下为 /home/codewang/.bashrc 文件的内容 -+-+--+
export PATH=xxx
export LD_LIBRARY_PATH=xxx
export HELLO_WORLD=xxx

关于vim的使用，读者自行查阅资料。
-+-+--+
```

Note: 当我们编辑保存该脚本后，它要在系统重新或者用户登录时才会被调用，如果你编辑了该文件想立即生效那么我们可以运行 `source` 命令： `[codewang$] source ~/.bashrc`

C) 通用环境变量配置 --> /etc/profile

服务器上有很多大家公用的软件，公用的库文件，难道每个用户都要自己去一一配置环境变量吗？一旦某个库更新了，管理员要一个用户一个用户的取修改他们的 `.bashrc` 文件吗？这当然是不能忍受的。Linux在启动时还会调用一个脚本文件 `/etc/profile` ,它和每个用户目录下的 `.bashrc` 类似，区别在于它设置的环境变量对所有用户有效!!! 并且先于每个用户的`.bashrc`运行

- 对所有用户有效

我们可以很方便的为所有用户设置公共软件路径，公共库路径，例如tensorflow、cuda等公共库，管理员就可以设置 `/etc/profile` 来为所有用户设置 `PATH`和`LD_LIBRARY_PATH` 环境变量

- 先于每个用户的 `.bashrc` 运行

这也说明了，如果 `profile` 中为每个用户设定了 `PATH`，而某个用户自己的 `.bashrc` 也设置了 `PATH` 环境变量，那么后者会覆盖前者。

好了，设置环境变量就介绍这么多，其实linux设置环境变量的方式不止以上几种，但是我们常用的就是上面的方式，有兴趣的同学可以自行查阅相关资料。下面我们通过实例说明该如何有效的设置环境变量。

让我们先做以下假设：

- 管理员在 `/usr/local/cuda` 下安装了 `cuda8.0`，其中 `cuda` 的应用程序在 `bin` 目录下，库文件在 `lib64` 目录下。
- `codewang` 觉得 `cuda8.0` 太老旧了，自己想使用 `cuda9.0`，并下载安装了 `cuda9.0` 安装在了 `/usr/local/cuda-9.0` 下面，同样的应用程序在 `bin` 目录下，库文件在 `lib64` 目录下。

管理员

作为管理员，只要用户不破坏公共库，公共资源，原则上用户有使用任何版本的软件的自由，所以管理员

只能操作 `/etc/profile` 文件，不可擅自改动某个用户的 `/home/username/.bashrc` 文件，普通用户也不可以擅自修改 `/etc/profile` 文件。

----- 以下为标准的 `/etc/profile` 文件配置 -----

```
# modified by codewang @2018.09.07    --> 修改文件添加备注，这是管理员的责任
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64

# 解释
# export PATH=$PATH:/usr/local/cuda/bin
# 首先PATH=$PATH 也就是等于原内容，然后后面追加内容，我们之前讲过多个路径用冒号隔开
# 因此该命令的含义为：给PATH后面追加:/usr/local/cuda/bin
# 这样就不会覆盖原来的环境变量了
```

codewang

上面管理员为所有用户配置了 `PATH` 和 `LD_LIBRARY_PATH`，但是我不想用 `cuda8.0`，所以我可以通过设置我自己的环境变量来实现，同时还不影响其他用户。因此我只需用设置我自己的 `.bashrc` 文件即可。

----- 以下为 `codewang` 的个人定制 `~/.bashrc` 文件内容 -----

```
export PATH=$PATH:/usr/local/cuda-9.0/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-9.0/lib64

# 解释
# 管理员设置了两个环境变量，我又在后面追加我自己想用的cuda9.0
```

```
# 这样，我的环境变量就包含了cuda8.0和cuda9.0， tensorflow依赖
# 哪个版本它会自己查找的。
#
# 如果 codewang想强制只使用cuda9.0，不想使用管理员设置的cuda8.0怎么办？
# 其实很简单：
# 1. 首先 echo $PATH 查看当前环境变量内容
# 2. 替换掉对于内容即可(cuda替换为cuda-9.0)
# 或者：
export PATH=/usr/local/cuda-9.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-9.0/lib64:$LD_LIBRARY_PATH
#我把自己想用的库放在环境变量的最前面，那么系统在搜索库时就会优先所有我指定的啦。
```

实质上，指定不同版本的python、不同版本的tensorflow或者是anaconda等都是这个原理。

2.1.4 Driver、CUDA、cudnn下载

Note: 在安装CUDA、cudnn之前要先确认操作系统版本、将要安装的tensorflow版本或者其他深度学习框架的版本。这里以tensorflow为例。

我们可以先去 [github](#) 上查看想要安装的tensorflow版本所需要的cudnn版本,或者google。这里假设我们要安装的tensorflow版本为 1.9.0，其需要的cuda版本为 cuda9.0 以上，keras对应的版本为 2.1.6,同时cuda9.0对应的cudnn版本为 8.0,除此之外还有GPU驱动程序驱动程序的版本假设为 384.69

驱动程序、cuda、cudnn均从NVIDIA官网下载：<https://www.nvidia.com/zh-cn/>

a) 下载驱动程序

- 前往NVIDIA官网，找到驱动程序->所有驱动程序



- 根据我们的GPU硬件选择相应的版本，点击搜索（最好选择英文版，一劳永逸）

NVIDIA 驱动程序下载

选项 1: 手动查找适用于我的 NVIDIA 产品的驱动程序。

[帮助](#)

产品类型: GeForce

产品系列: GeForce 10 Series

产品家族: GeForce GTX 1080 Ti

操作系统: Linux 64-bit

语言: English (US)

搜索

- 点击下载即可，然后上传到服务器，建议将历史版本的下载包存储下来以备。也可以直接在服务器上下载。

LINUX X64 (AMD64/EM64T) DISPLAY DRIVER

Version: 390.87
Release Date: 2018.8.27
Operating System: Linux 64-bit
Language: English (US)
File Size: 78.86 MB

DOWNLOAD

RELEASE HIGHLIGHTS

SUPPORTED PRODUCTS

ADDITIONAL INFORMATION

- Fixed a resource leak introduced in the 390 series of drivers that could lead to reduced performance after starting and stopping several OpenGL and/or Vulkan applications.

- 然后我们就得到了名称为 `NVIDIA-Linux-x1080ti-384.69.run`

假设我将此文件存放在 `/home/user/Download/NVIDIA-Linux-x1080ti-384.69.run` 下

b) 下载CUDA

- 同样的，我们在首页找到：开发者->CUDA



- 进入页面后，点击 `Download Now`，然后根据机器配置勾选相应选项，最好下载 `.run` 文件。这个页面默认是下载最新版本的CUDA，如果需要旧版本自行查找，方法类似。

Select Target Platform ⓘ

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX
Architecture ⓘ	x86_64	ppc64le	
Distribution	Fedora	OpenSUSE	RHEL CentOS SLES Ubuntu
Version	17.10	16.04	
Installer Type ⓘ	runfile (local)	deb (local)	deb (network) cluster (local)

Download Installers for Linux Ubuntu 16.04 x86_64

The base installer is available for download below.
There is 1 patch available. This patch requires the base installer to be installed first.

> Base Installer

Download (1.7 GB)

Installation Instructions:

- Run `sudo sh cuda_9.2.148_396.37_linux.run`
- Follow the command-line prompts

- 最终我们会得到一个 `cuda_9.3.148_396.37_linux.run` 的文件，同样的上传至服务器，建议备份。

假设我存放在了 `/home/user/Download/cuda_9.3.148_396.37_linux.run`

c) 下载cuDnn <https://developer.nvidia.com/cuDNN>

- 下载cuDNN需要登录，所以你要现在NVIDIA官网上注册。这里就略过了，下载方法与上面类似。

The NVIDIA CUDA® Deep Neural Network library [cuDNN] is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN is part of the NVIDIA Deep Learning SDK.

Deep learning researchers and framework developers worldwide rely on cuDNN for high-performance GPU acceleration. It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning. cuDNN accelerates widely used deep learning frameworks, including Caffe2, MATLAB, Microsoft Cognitive Toolkit, TensorFlow, Theano, and PyTorch. For access to NVIDIA optimized deep learning framework containers, visit NVIDIA GPU CLOUD to learn more and get started.

Download cuDNN >

Introductory Webinar >

Developer Guide >

Forums >

- 最终我们可以得到一个 `cuda-8.0-linux-x64-xxx.tgz`

假设我存在： `/home/user/Download/cudnn-8.0-linux-x64-xxx.tgz`

2.1.5 Driver、CUDA、cudnn安装

上述三个安装包下载完成后，我们就有了：

```
/home/user/Download/NVIDIA-Linux-x1080ti-384.69.run    --> NVIDIA显卡驱动程序
/home/user/Download/cuda_9.3.148_396.37_linux.run      --> CUDA
/home/user/Download/cudnn-8.0-linux-x64-xxx.tgz        --> cdDNN
```

a) 安装驱动程序

GPU也就是我们所说的显卡，它有可能已经应用在我们服务器配置的显示器上了，而出厂默认安装的是用于高性能显示的驱动程序,或者老旧的驱动程序，所以我们要先做一下清理工作。

- 卸载原来的驱动程序

```
sudo apt-get remove --purge nvidia*
```

- 禁用 nouveau

```
sudo vim /etc/modprobe.d/blacklist-nouveau.conf
```

输入以下内容并保存，即可将nouveau添加到黑名单，该驱动程序将不会再被加载。

```
blacklist nouveau
options nouveau modeset=0
```

- 刷新

```
sudo update-initramfs -u
```

然后重启系统

- 重启后，检查是否禁用nouveau成功

```
lsmod | grep nouveau
```

，如果没有输出内容则表示禁用成功。

- 切换到命令行界面

因为我们要重新安装GPU驱动，所以图形界面需要暂时关闭。

```
# 切换到终端界面
Ctrl + Alt + F1 (Ctrl + Alt + Fx 表示切换到x终端，其中F7表示图像界面)
# 关闭图形界面
user$ sudo service lightdm stop
```

- 开始安装NVIDIA显卡驱动(runfile)

```
# 进入驱动程序对应的目录
user$ cd /home/user/Download/
# 增加可执行权限
user$ sudo chmod +x NVIDIA-Linux-x1080ti-384.69.run
# 执行.run，开始安装
user$ sudo ./NVIDIA-Linux-x1080ti-384.69.run --no-x-check --no-nouveau-check --no-opengl-files
# --no-opengl-files 只安装驱动文件，不安装 OpenGL 文件。这个参数最重要
# --no-x-check 安装驱动时不检查 X 服务
# --no-nouveau-check 安装驱动时不检查 nouveau
```

- 启动图像界面

```
# 启动图像界面
sudo service lightdm start
# 切换到图像界面
Ctrl + Alt + F7
```

- 检查驱动程序是否安装成功

```
user$ nvidia-smi
```

如果安装成功，会出现 类似如下的界面,可以看到驱动版本为 384.69：

```
Tue Sep 18 15:20:59 2018
+-----+
| NVIDIA-SMI 384.69                  Driver Version: 384.69          |
+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+
|    0  GeForce GTX 108...    Off   | 00000000:04:00.0 Off  |          N/A         |
| 27%   48C    P2      77W / 250W | 10795MiB / 11172MiB |      0%      Default |
+-----+-----+
```

b) 安装 CUDA

和安装驱动程序类似，我们的CUDA库选择的也是run文件形式来安装，CUDA不需关闭图形界面，所以可以直接在终端里执行安装。

```
# 增加可执行权限
user$ sudo chmod +x cuda_9.3.148_396.37_linux.run
# 开始安装
user$ ./cuda_9.3.148_396.37_linux.run

# Description
#
# This package includes over 100+ CUDA examples that demonstrate
# various CUDA programming principles, and efficient CUDA
# implementation of algorithms in specific application domains.
# The NVIDIA CUDA Samples License Agreement is available in
# Do you accept the previously read EULA?
# accept/decline/quit: accept
#
# Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 367.48?
# (y)es/(n)o/(q)uit: n ---> 是否安装驱动程序？ 我们安装过了，所以选择 no
#
# Install the CUDA 9.0 Toolkit?
# (y)es/(n)o/(q)uit: y ---> 是否安装CUDA 工具箱， 选择 yes
#
# Enter Toolkit Location
# [ default is /usr/local/cuda-9.0 ]: ---> 安装位置，默认即可
```

```
#
# Do you want to install a symbolic link at /usr/local/cuda?
# (y)es/(n)o/(q)uit: y ---> 是否安装动态链接文件，选择yes
#
# Install the CUDA 9.0 Samples?
# (y)es/(n)o/(q)uit: y ---> cuda自带了一些示例程序，我们选择 yes。之后可以做一些验证工作。
#
# Enter CUDA Samples Location
# [ default is /xxxx ]: ---> 示例程序的安装位置， 默认即可
#
# Installing the CUDA Toolkit in /usr/local/cuda-9.0 ...
# Installing the CUDA Samples in /xxx ...
# Copying samples to /xxxx now...
# Finished copying samples.
```

至此，run文件已经将我们需要的CUDA动态链接库安装到了 `/usr/local/cuda-9.0/`（默认）下面。相信你还记得我们在上一节中讲到的，Linux系统需要我们设定环境变量才能让其他应用程序能够找到我们安装的动态链接库，所以下来我们要设置环境变量。

如果你是管理员，请为所有人设置环境变量（`/etc/profile`），如果你是普通用户安装了自己需要的定制版CUDA那么请设置（`~/.bashrc`），这里我们给出 `/etc/profile` 的设置示例，具体原理不再赘述，参考2.1.3小节：

```
# Add CUDA-9.0 Library.  modify by codewang @2019.9.16
export PATH=/usr/local/cuda-9.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-9.0/lib64:$LD_LIBRARY_PATH
```

c) 安装cudnn加速库

因为cudnn是用来做动态加速运算的库，但是并不是所有NVIDIA显卡用户都要用到此部分功能，所以NVIDIA讲此部分库从CUDA中独立出来，形成了现在的cudnn，因此与其说是安装cudnn库，不如说是给CUDA打补丁，所以可以看到我们下载的cudnn库时压碎文件，并不需要安装。我们只需用将其解压，将对应问价拷贝至CUDA安装目录下即可。

```
# 解压 cudnn
user$ tar -xvzf cudnn-8.0-linux-x64-xxx.tgz --> 假设解压得到 cuda 目录
# 拷贝动态链接库到CUDA安装目录下
user$ sudo cp ./cuda/lib64/libcudnn* /usr/local/cuda-9.0/lib64/
# 拷贝头文件到CUDA安装目录下
user$ sudo cp ./cuda/include/cudnn.h /usr/local/cuda-9.0/include/
# 因为是刚解压出来的文件，其权限是不确定的，为了以防万一，我们手动为其增加可读权限
user$ sudo chmod a+r /usr/local/cuda-9.0/lib64/libcudnn*
user$ sudo chmod a+r /usr/local/cuda-9.0/include/cudnn.h
```

d) 验证

这里我们可以验证前面三个库的安装是否正确，利用上面提到的CUDA的示例程序。路径为 `/usr/local/cuda-9.0/samples/1_Uutilities/deviceQuery`

```
# 进入示例程序目录
user$ cd /usr/local/cuda-8.0/samples/1_Uutilities/deviceQuery
# 编译示例程序
user$ sudo make --> 我们可以看到生成了 deviceQuery 的可执行程序
# 运行观察结果
user$ ./deviceQuery
#
# ----- 只作为示例，仅供参考 -----
#
# CUDA Device Query (Runtime API) version (CUDART static linking)
# Detected 10 CUDA Capable device(s)
# Device 0: "GeForce GTX 1080 Ti"
# CUDA Driver Version / Runtime Version          9.0 / 9.0
# CUDA Capability Major/Minor version number:    6.1
# Total amount of global memory:                 11172 MBytes
(11715084288 bytes)
# (28) Multiprocessors, (128) CUDA Cores/MP:     3584 CUDA Cores
# GPU Max clock rate:                            1582 MHz (1.58 GHz)
# Memory Clock rate:                             5505 Mhz
# Memory Bus Width:                              352-bit
# L2 Cache Size:                                2883584 bytes
# Maximum Texture Dimension Size (x,y,z)        1D=(131072), 2D=(131072,
65536), 3D=(16384, 16384, 16384)
# Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
# Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048
layers
# Total amount of constant memory:                65536 bytes
# Total amount of shared memory per block:        49152 bytes
# Total number of registers available per block:  65536
# Warp size:                                     32
#
# ...
# ...
#
# deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.0,
# CUDA Runtime Version = 9.0, NumDevs = 10, Device0 = GeForce GTX 1080 Ti,
Device1 =
# GeForce GTX 1080 Ti, Device2 = GeForce GTX 1080 Ti, Device3 = GeForce
GTX 1080 Ti,
# Device4 = GeForce GTX 1080 Ti, Device5 = GeForce GTX 1080 Ti, Device6 =
GeForce GTX
# 1080 Ti, Device7 = GeForce GTX 1080 Ti, Device8 = GeForce GTX 1080 Ti,
Device9 =
# GeForce GTX 1080 Ti
# Result = PASS ----> 表示安装正确
```


2.1.6 Tensorflow/Keras的安装

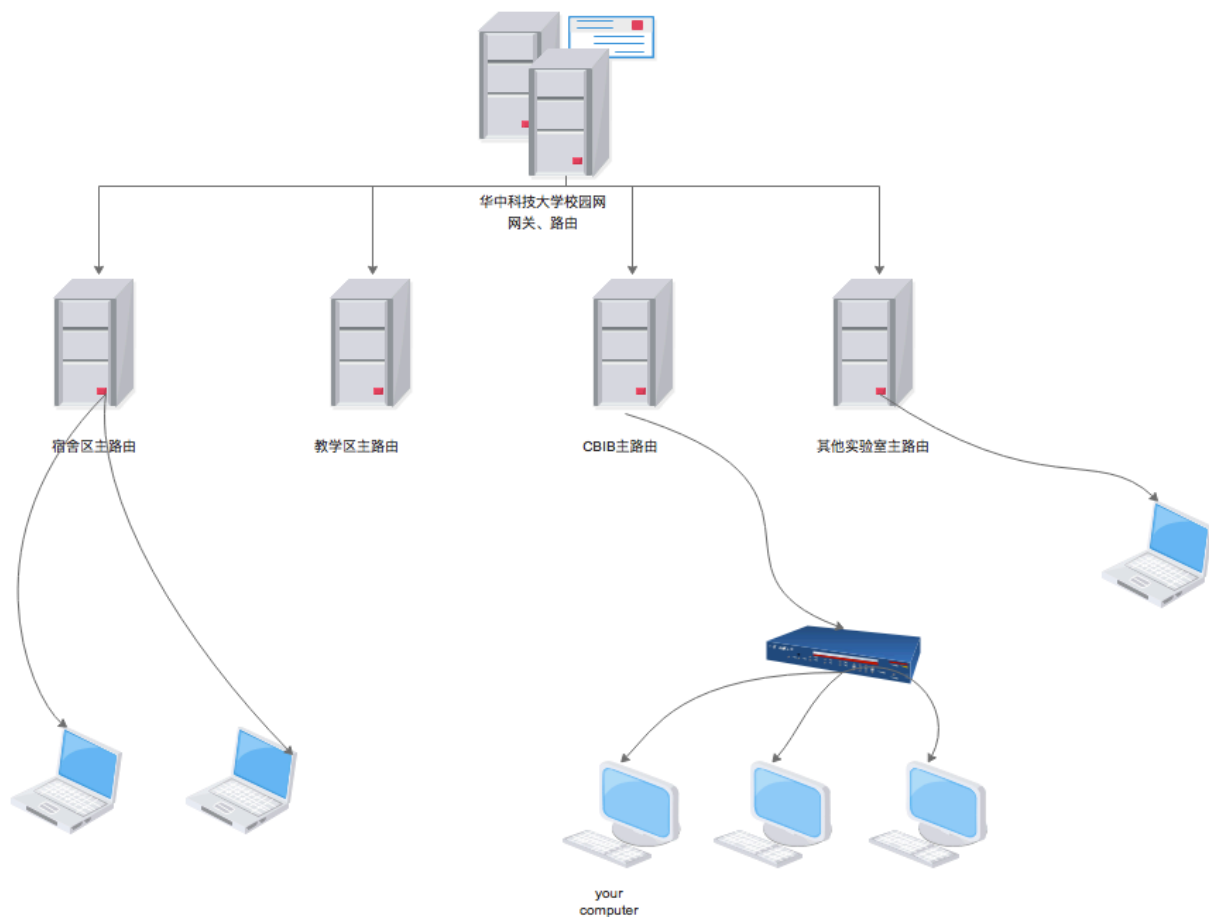
安装tensorflow、keras、pytorch、caffe等框架就相对简单很多了，使用pip安装即可，或者conda安装。

```
# 安装tensorflow-gpu
user$ sudo pip3 install tensorflow-gpu # ---> 安装最新版tensorflow gpu版本
# 也可以指定安装版本
user$ sudo pip3 install tensorflow-gpu==1.6.0 # ---> 安装1.6.0版本
# keras 安装
user$ sudo pip3 install keras==2.2.0
#
# 这里做一点补充， 由于我们的设备一般自带python2.x， 我们也安装了python3.x
# 我们一般使用python3.x(python2.x已经不再维护了)， 所以安装库时要用对应的pip3,而不是
pip
```

2.2 固定IP配置

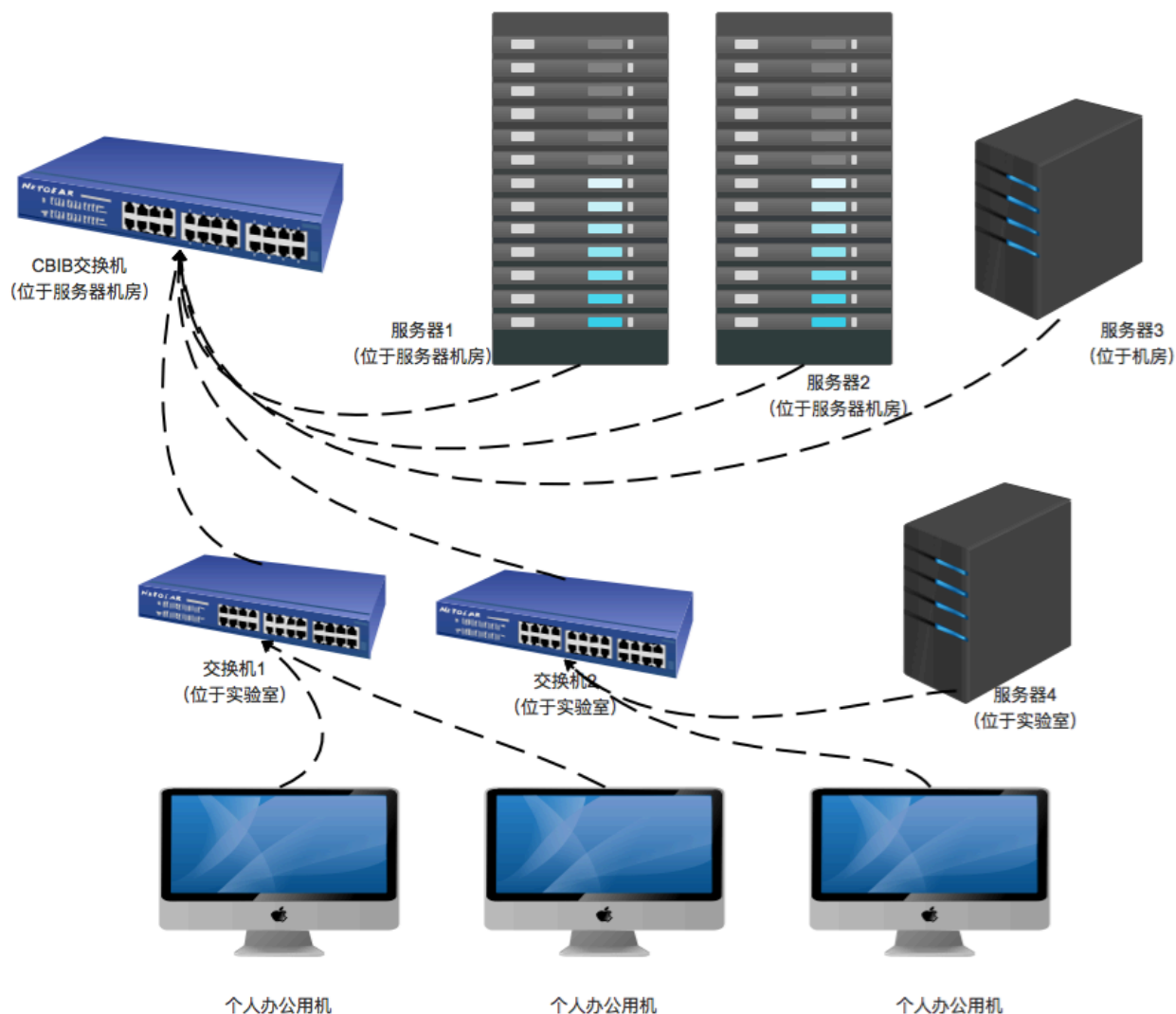
本节将介绍申请固定IP的过程，和如何给服务器配置固定IP地址。

2.2.1 校园网的组成



无论你在校园内的任何地方，只要使用校园网那么最终都会通过学校的总路由、网关而访问英特网，所以无论你在学校的何处只要你有一根网线能够连接到校园网（可以不登录），或者使用无线网连接到校园网，都可以访问学校内的公开IP地址（相当于外网）。

2.2.2 CBIB实验室局域网组成



而我们实验室的网络步骤也是一个局域网，所以你可以在不连接校园网的情况下访问服务器，下面我们分情况讨论为什么一定要给服务器配置校内固定IP。

- **服务器不连接校园网，固定IP地址**

上面我们说了，因为实验室内所有机器都在一个局域网内，所以就算服务器不连接校园网，只要将网线连接至CBIB的交换机上，我们就可以在局域网内访问服务器了，但是问题在于我们的服务器是不能访问外网的，那么我们先更新软件，下载安装包都无法实现，只能使用自己的电脑下载好安装包，然后上传至服务器安装相当的麻烦。

- **服务器登录校园网账号**

另一种方式就是给服务器登录校园网账号，这样服务器便可以访问外网了，解决了上述的问题，但是使用校园网账号登录时其IP地址是不固定的，我们每次登录服务器都要先查看其IP地址，因此此方法也不行。

- **服务器使用校内固定IP，免登录**

最终的解决方案就是向学校校园网中心申请固定IP。首先固定IP地址是不需要登录的，可以直接访问外网，其次其IP地址不会发生变化。具体申请固定IP的流程可请教@金老师或者@马老师。申请固定IP需要向校园网中心提供机器的MAC地址，校园网中心会下发一个全校可访问的公共IP地址与我们提交的MAC地址绑定，然后我们的服务器就可以免费上网了。

如何查看MAC地址

```
user$ ifconfig
```



```
eth0      Link encap:Ethernet  HWaddr xx:xx:xx:xx:xx:xx
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Memory:fb300000-fb37ffff

eth1      Link encap:Ethernet  HWaddr xx:xx:xx:xx:xx:xx
          inet addr:xxx.xxx.xxx.xxx  Bcast:xxx.xxx.xxx.xxx
Mask:255.255.252.0
          inet6 addr: fe80::ae1f:6bff:fe14:4a4f/64 Scope:Link
          inet6 addr: 2001:250:4000:8240:ae1f:6bff:fe14:4a4f/64
Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:117381703 errors:0 dropped:0 overruns:0 frame:0
          TX packets:449580661 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:62143220880 (62.1 GB)  TX bytes:645669845490 (645.6
GB)
          Memory:fb200000-fb27ffff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:2271267728 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2271267728 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:1
          RX bytes:6361168372943 (6.3 TB)  TX bytes:6361168372943 (6.3
TB)
```

```
# 其中 eth0表示服务器的第一块网卡， eth1同理， lo表示本地回环地址
# 一般我们只使用一块网卡即可，在服务器2中我们使用的是eth1
# 其中：
#   HWaddr(hardware address) 就是该网卡的MAC地址
#   inet addr : 表示该网卡的ip地址
#   其他
```

申请单下发后大致是这个样子：

设备信息		
设备名称	MAC 地址	操作系统
服务器		Ubuntu
设备接入交换机	设备接入端口号	用户 IP 地址
		
子网掩码	网关	DNS 服务器
255.255.252.0	200.1.1.254	200.1.1.1、200.1.1.2

2.2.3 固定IP的配置

我们从申请单可以得到

```
MAC地址:  MAC:MAC:MAC:MAC:MAC:MAC
IP地址 :  ip.ip.ip.ip
子网掩码:  255.255.252.0
网关    :  way.way.way.way
DNS     :  DNS1.DNS1.DNS1.DNS1, DNS2.DNS2.DNS2.DNS2
```

请注意在你申请IP时提交的MAC 地址是哪一张网卡的，接下来的配置就要在哪一张网卡上进行。
MAC地址不匹配
是不能上网的。

a) 修改 /etc/network/interface

```
user$ sudo vim /etc/network/interface

# ----- 以下为示例内容 -----
# The loopback network interface
auto lo      ---> 这是本地回环的配置，我们不用管
iface lo inet loopback

# The primary network interface
auto eth1    # ---> 先让个自动获取IP，激活网卡
iface eth1 inet static # ---> IPv4 设置为固定IP模式
address ip.ip.ip.ip    # ---> 固定IP
netmask 255.255.252.0  # ---> 子网掩码
gateway way.way.way.way # ---> 网关

# 保存退出
```

b) 修改 /etc/resolvconf/resolv.conf.d/base

这一步的目的是配置DNS

```
user$ sudo vim /etc/resolvconf/resolv.conf.d/base
```

```
# ----- 以下为示例内容 -----  
nameserver DNS1.DNS1.DNS1.DNS1  
nameserver DNS2.DNS2.DNS2.DNS2  
  
# 保存退出
```

c) 刷新配置，重新网卡

```
user$ sudo resolvconf -u # ----> 刷新网络配置  
user$ sudo /etc/init.d/networking restart # ----> 重启网络服务
```

然后终端检查IP地址是否设置正确，是否可以访问外网，实验室内个人PC是否可以访问服务器。
如果重启网络服务不能生效的话可以试一试重启服务器。

2.3 SSH 服务配置

2.4 远程桌面配置

此部分挪动到第三章

2.5 远程传输配置(ftp/samba/nfs)

2.6 sublime+sftp 远程文件同步

第三章 Linux运维

为了能更有条理地讲解服务器远程桌面配置，我将此部分内容移动至本章（第三章）。本章涉及常用的Linux运维知识，例如 文件权限、用户权限、用户的增删改查、分组，远程桌面配置等。Linux运维是一门大学科，这里不可能面面俱到，我只是把实验室服务器运维常用的一些知识点和个人的一些小建议编写在此，难免有纰漏，望指正。

如果有对Linux运维有兴趣的同学，推荐阅读《鸟哥的Linux私房菜》系列书籍

3.1 Linux用户管理

Linux作为多用户操作系统，用户的管理、用户之间的隔离、用户的权限是必不可少的关键因素。这里将简要介绍Linux中常用的一些用户管理的知识，让我们一起领略一下Linux是怎么做到让多个用户在一台机器上工作而又不会互相冲突且能相互合作的。当然！这些都是理想情况（每个用户都是老鸟），对于萌新一直是让管理员头疼的存在，所以说管理一群对linux不了解的用户着实是一个难题，

特别是对我这种运维半吊子来说，不懂高级的运维操作，只能用传统的入门级方法苟且偷生，还要被人鄙视不懂还装逼（骑虎难下）。

3.1.1 Linux下的三类用户

说到用户大家首先想到的肯定是 管理员即root用户、普通用户，那怎么会出现第三个用户呢？其实是软件、应用程序，我们安装一个QQ程序，那么这个QQ程序想要读写某个文件、访问我们的摄像头怎么办？比如是张三安装的QQ，难道QQ就不能访问系统目录了吗？所以linux系统还存在第三个用户即 伪用户。

a) root用户

root用户是唯一的，一台机器只能有一个root用户，root用户拥有神一般的权限，哪怕是删除系统的权限它都有。因为root用户的权限太高，所以一般我们建议不能使用远程登录的方式登录root账号，root密码也是绝对不能随便告诉其他人的，可以想象一下如果被某个黑客、破坏者拿到root密码，远程登录到服务器那会是多么可怕。**root用户的UID（user id）为0**。我们常说的管理员！不是指root用户，root用户是账号、管理员是人！，通常管理员也是登录自己的账号来维护服务器，而不是登录root账号

b) 伪用户

上面我们也说了，伪用户是为了方便系统管理、应用程序、满足相应的系统进程文件属主的要求而设立的。伪用户不能登录系统，其UID为1~499

c) 普通用户

普通用户就是我们平常使用的登录账号了，一般情况下包括管理员在内所有服务器使用者都应该是普通用户的账号。其UID在500~6000,具有有限的权限，比如只能写自己的所属文件、只能修改删除自己创建的文件夹等，不能安装应用程序到系统目录等。关乎权限 问题后面我们会详细介绍。

关于用户的其他信息，linux系统中有个叫做 `/etc/passwd` 的文件，其中记录了每一个用户的信息，我们打开看一看其具体内容：

```
# ----- /etc/passwd 文件样例内容 -----  
-----  
  
# root:x:0:0:root:/root:/bin/bash  
# daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
# bin:x:2:2:bin:/bin:/usr/sbin/nologin  
# sys:x:3:3:sys:/dev:/usr/sbin/nologin  
# sync:x:4:65534:sync:/bin:/bin/sync  
# games:x:5:60:games:/usr/games:/usr/sbin/nologin  
# man:x:6:12:man:/var/cache/man:/usr/sbin/nologin  
# mail:x:8:8:mail:/var/mail:/usr/sbin/nologin  
# syslog:x:101:104::/home/syslog:/bin/false  
# lightdm:x:112:118:Light Display Manager:/var/lib/lightdm:/bin/false  
#
```

```
# ...
#
# user:x:1000:1000:user,,,:/home/user:/bin/bash
# sshd:x:116:65534::/var/run/ssh:/usr/sbin/nologin
# codewang:x:1001:1024::/home/codewang:/bin/bash
# jy:x:1004:1024::/home/jy:/bin/bash
# ll:x:1005:1024::/home/ll:/bin/bash
# jrc:x:1007:1024::/home/jrc:/bin/bash
# xrdp:x:120:128::/var/run/xrdp:/bin/false
#
# ...
#
# wzd:x:1030:1024::/home/wzd:/bin/bash
# lp:x:1031:1024::/home/lp:/bin/bash
# cookcoder:x:1006:1002:,,,:/home/cookcoder:/bin/bash
# jjl:x:1032:1024::/home/jjl:/bin/bash
# czm:x:1033:1024::/home/czm:/bin/bash
...

```

这是2号服务器的 `/etc/passwd` 的部分内容。可以看到此文件的每一行对应一个用户，每一行分为7个部分用冒号隔开，分别为 账号名称:用户密码:用户标识码（UID）:组标识码(GID):用户相关信息:用户home目录:用户环境

这里我们就三种用户分别看一下其内容分别代表什么。

- `root:x:0:0:root:/root:/bin/bash`

第一行是root用户的信息：

用户名称 ： root，

密码： 是加密的所以显示为x，

用户ID (UID) ： 0

组ID (GID) ： 0 ， 关于组(group)后面会讲解

用户相关信息 ： root

home目录 ： /root

用户环境 ： /bin/bash ,用户环境是指bash解析器，我们在命令行写命令，或者写脚本都需要一个解析器来解析我们输入的内容，一般包括/bin/bash、/bin/sh等，一般推荐使用/bin/bash

- `sshd:x:116:65534::/var/run/ssh:/usr/sbin/nologin`

sshd 就是我们用于远程登录的ssh服务，从这也可以证实为什么Linux需要第三个用户类型了，我们每个人使用ssh登录后，所有发送给服务器的命令，都是通过ssh服务转发的，所以sshd需要非常特殊的权限。

同时我们注意到其 `UID=116` 确实是伪用户的UID，其home目录是 `/var/run/sshd` 指向了其运行时需要的一些临时使用的目录，特别的其用户环境是 `/usr/sbin/nologin` 这也说明了伪用户是不可以登录的。

- `codewang:x:1001:1024::/home/codewang:/bin/bash`

这是我的账号的相关系，可以看到UID是1001，我们实验室的用户ID是从1001开始的，组1024是组ID表示CBIB组（后面详解），home目录是/home/codewang。其实每个用户的home目录是可以更改的，而且不一定非要在 `/home/` 下面，可以放在硬盘上或者其他地方，而当我们更改home目录后此文件也会有相应的变换。

我们回顾一下，前一章讲到一个小技巧：如何快速的进入我的home目录，我们讲到使用 `cd ~/` 其中 `~` 就表示当前用户的home目录，那么当解析器 `/bin/bash` 看到 `~` 之后，就回去查询 `/etc/passwd` 中找到对应用户从而得到其home目录的路径。

3.1.2 linux 文件权限

Linux中有一句非常重要的定理：**一切皆文件！**，在Window中可能存在驱动程序，应用程序，各种注册表等，但是在linux中无论是什么都是文件！比如USB鼠标，安装驱动后其在用户空间的表示形式就是设备文件（设备节点），一切都是文件。所以文件的权限也是Linux的一个组成部分。

我们在命令行可以使用 `ls -al` 或者 `ll` 来查看文件的具体信息。

```
user$ ls -al

# total 312
# drwxrwxr-x 6 user user 4096 7月 30 19:12 .
# drwxrwxr-x 9 user user 4096 8月 5 12:56 ..
# -rw-rw-r-- 1 user user 270208 7月 30 19:12 Breast Segmentation
Checking.ipynb
# drwxrwxr-x 2 user user 4096 7月 17 10:45 checkpoints
# drwxrwxr-x 2 user user 4096 7月 17 18:08 .ipynb_checkpoints
# drwxrwxr-x 2 user user 4096 7月 17 18:09 __pycache__
# drwxrwxr-x 2 user user 4096 7月 10 13:41 record
# -rw-rw-r-- 1 user user 10518 8月 5 16:11 unet.py
# -rw-rw-r-- 1 user user 10591 7月 17 10:41 unet_without_lrs.py

user$ ls -al /dev/nvidia*
# crw-rw-rw- 1 root root 195, 0 9月 25 09:56 /dev/nvidia0
# crw-rw-rw- 1 root root 195, 1 9月 25 09:56 /dev/nvidia1
# crw-rw-rw- 1 root root 195, 255 9月 25 09:56 /dev/nvidiactl
# crw-rw-rw- 1 root root 195, 254 9月 25 09:56 /dev/nvidia-modeset
# crw-rw-rw- 1 root root 243, 0 9月 25 09:56 /dev/nvidia-uvm
```

我们用空格分开，可以看到文件信息包括9个部分。

- 第一个部分：

`drwxrwxr-x` -->

- 第一位d表示是文件夹， -表示是普通文件， c表示字符设备设备节点， b表示块设备等
- 剩下的9位分别表示所属用户的权限、组用户的权限、其他用户的权限。
比如 `rw` 表示可读可写可执行， -表示不具有该权限。

例子： `drwxrwxr-x`

d 表示该文件是一个目录

`rw` 表示这个文件的拥有者(所属用户)拥有读、写、执行权限

`rw` 表示该文件所在的分组的其他用户拥有读、写、执行权限

`r-x` 表示其他用户拥有读权限、执行权限，但是没有写权限。

假设有甲、乙、丙和root四个用户，甲和丙在用户组G1中，乙在G2分组内

现在有一个文件A 其所属用户为甲、所属组为G1， 其权限为 `drwxrwxr-x`。那么：

对于该文件来说，

- 甲是所属用户和组用户 所以甲对该文件有 `rw`权限
- 乙不是所属用户那么就是其他用户、但是乙个该文件在同一个分组内所以乙是该文件的

组用户

乙作为其他用户的权限是 `r-x`，作为组用户其权限是 `rw`，我们对这些权限取与运算，

所以乙用户

对该文件的权限为 `rw`。

- 丙用户对该文件来说是其他用户，所以拥有读权限和可执行权限
- root用户拥有一切权限。

第四章 服务器管理

第五章 常见问题及解决方案

此部分单独成册，见README.md