编程环境和软件设施安装手册

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

创建mysql用户和用户组

修改mysql目录的归属用户

准备MySQL的配置文件

正式开始安装MySQL

复制启动脚本到资源目录

设置MySQL系统服务并开启自启

启动mysqld

将 `mysql` 的 `bin` 目录加入 `PATH` 环境变量

首次登陆MySQL

接下来修改root账户密码

设置远程主机登录

最后利用Navicat等工具进行测试即可：

Redis缓存安装部署

首先准备Redis安装包

解压安装包

编译并安装

将 `Redis` 安装为系统服务并后台启动

查看Redis服务启动情况

启动Redis客户端并测试

设置允许远程连接

设置访问密码

消息队列RabbitMQ安装部署

首先安装erlang环境

安装RabbitMQ

设置RabbitMQ开机启动

启动RabbitMQ服务

开启web可视化管理插件：

访问可视化管理界面：

应用服务器Tomcat安装部署

准备安装包

解压并安装

启动Tomcat

配置快捷操作和开机启动

Web服务器Nginx安装部署

首先安装包并解压

预先安装额外的依赖

编译安装nginx

启动Nginx

浏览器验证启动情况

Docker环境安装

安装Docker

开启Docker服务

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

环境声明（重要！！！）

- 本文档所有实验、环境、工具、软件均基于 `CentOS 7.4 64bit` Linux操作系统进行
- 本文档所使用的 **Linux系统ISO镜像**、**软件安装包** 都已经提前下载并备好，需要的童鞋可以直接扫描下方二维码关注，然后回复“**软件包**”三个字即可自行领取：

> 本文档在 Github开源项目：github.com/hansonwang99/JavaCollection 中已收录，有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

> **另外，** 联系作者、提意见，可以参看本文档尾部，有相应联系方式。

接下来，我们就从 `Linux` 操作系统环境的安装开始。

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

安装配置LINUX系统环境

关于这一部分内容，之前已经出过「视频教程」和「图文教程」，可直接点击查看：

MACOS系统图文教程（点击可查看）

MACOS系统视频教程（点击可查看）

WINDOWS系统图文教程（点击可查看）

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

GIT工具安装

方式一：通过包管理器安装

在 `Linux` 上安装 `Git` 向来**仅需一行命令**即可搞定，因为各式各样的**包管理器**帮了我们大忙，所以对于 `CentOS` 系统来讲，直接执行如下命令即可安装：

```
yum install git
```

当然通过这种方式安装的 `Git` 可能不是较新版的 `Git`，以我们的实验环境 `CentOS 7.4` 来说，这种方式安装的 `Git` 版本为 `1.8.3.1`，不过一般来说是够用的。

方法二：通过源码编译安装

如果想安装较新版本的 `Git`，则需要自行下载 `Git` 源码来编译安装。

1、准备Git安装包

我这里选择安装的是 `2.26.2` 版，将下载好的安装包 `v2.26.2.tar.gz` 直接放在了 `root` 目录下

然后将其本地解压，得到 `git-2.26.2` 目录：

```
[root@localhost ~]# tar -zxvf v2.26.2.tar.gz
```

2、提前安装可能所需的依赖

```
yum install curl-devel expat-devel gettext-devel openssl-devel zlib-
devel gcc-c++ perl-ExtUtils-MakeMaker
```

3、编译安装Git

进入到对应目录，执行配置、编译、安装命令即可，如下所示：

```
[root@localhost ~]# cd git-2.26.2/
[root@localhost git-2.26.2]# make configure
[root@localhost git-2.26.2]# ./configure --prefix=/usr/local/git
[root@localhost git-2.26.2]# make profix=/usr/local/git
[root@localhost git-2.26.2]# make install
```

4、将Git加入环境变量

将 `Git` 的可执行程序加入环境变量，便于后续使用

编辑配置文件：

```
vim /etc/profile
```

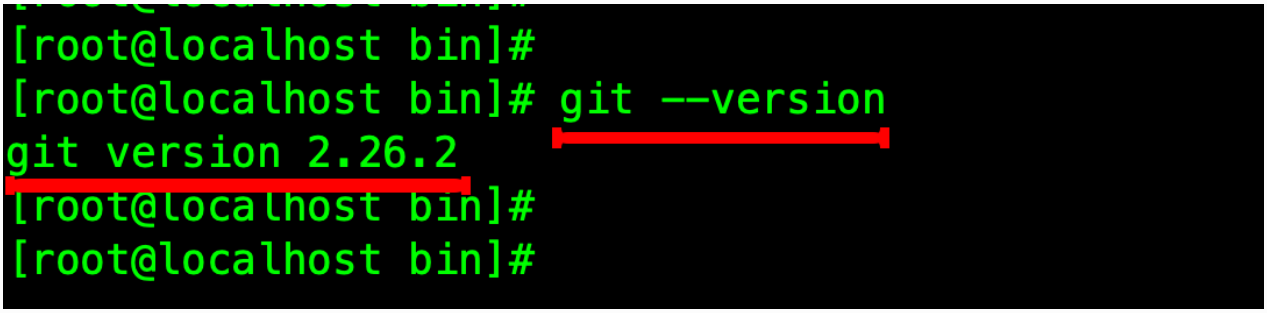
尾部加入 `Git` 的 `bin` 路径配置即可

```
export GIT_HOME=/usr/local/git
export PATH=$PATH:$GIT_HOME/bin
```

最后执行 `source /etc/profile` 使环境变量生效即可。

5、查看安装结果

执行 `git --version` 查看安装后的版本即可



本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

JDK（JAVA环境）安装

> 注意：这里安装的是Oracle JDK

准备JDK安装包

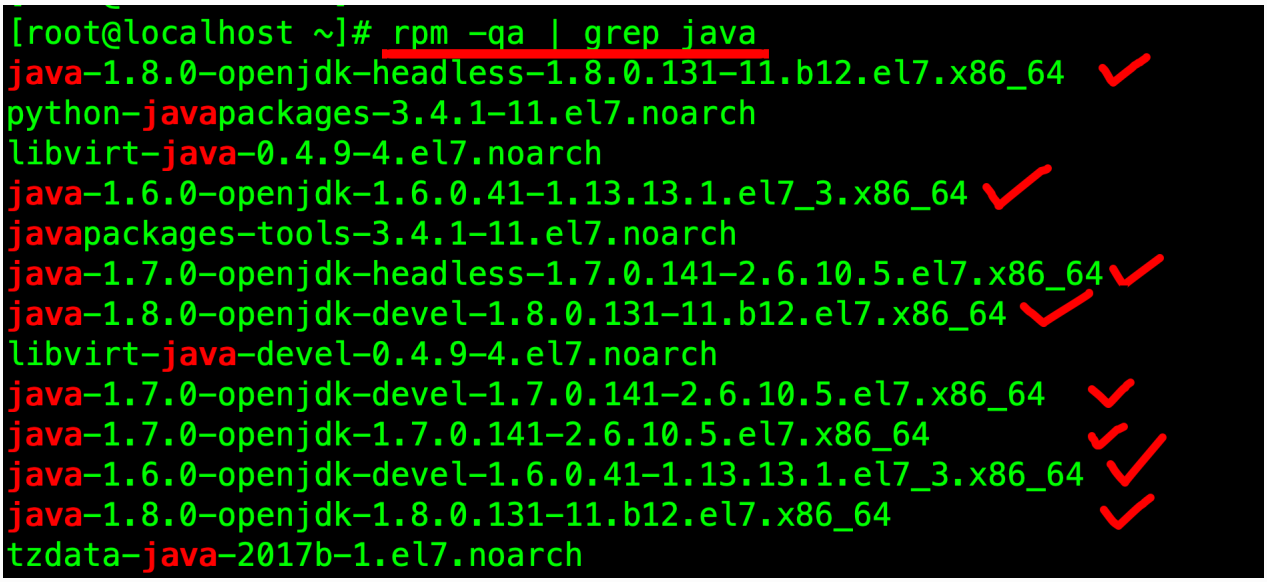
我这里下载的是 `jdk-8u161-linux-x64.tar.gz` 安装包，并将其直接放在了 `root` 目录下

卸载已有的OPENJDK（如果有）

如果系统自带有 `OpenJDK` ，可以按照如下步骤提前卸载之。

首先查找已经安装的 `OpenJDK` 包：

```
rpm -qa | grep java
```



接下来可以将 `java` 开头的安装包均卸载即可：

```
yum -y remove java-1.7.0-openjdk-1.7.0.141-2.6.10.5.el7.x86_64
yum -y remove java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86_64

... 省略 ...
```

创建目录并解压

1、在 `/usr/local/` 下创建 `java` 文件夹并进入

```
cd /usr/local/
mkdir java
cd java
```

2、将上面准备好的 `JDK` 安装包解压到 `/usr/local/java` 中即可

```
tar -zxvf /root/jdk-8u161-linux-x64.tar.gz -C ./
```

解压完之后， `/usr/local/java` 目录中会出现一个 `jdk1.8.0_161` 的目录

配置JDK环境变量

编辑 `/etc/profile` 文件，在文件尾部加入如下 `JDK` 环境配置即可

```
JAVA_HOME=/usr/local/java/jdk1.8.0_161
CLASSPATH=$JAVA_HOME/lib/
PATH=$PATH:$JAVA_HOME/bin
export PATH JAVA_HOME CLASSPATH
```

然后执行如下命令让环境变量生效

```
source /etc/profile
```

验证JDK安装结果

输入如下命令即可检查安装结果：

```
java -version

javac
```

```
[root@localhost java]#
[root@localhost java]# java -version
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)
[root@localhost java]#
[root@localhost java]#
[root@localhost java]# javac

用法: javac <options> <source files>
其中, 可能的选项包括:
  -g                         生成所有调试信息
  -g:none                    不生成任何调试信息
  -g:{lines,vars,source}     只生成某些调试信息
  -nowarn                    不生成任何警告
  -verbose                   输出有关编译器正在执行的操作的消息
  -deprecation               输出使用已过时的 API 的源位置
  -classpath <路径>            指定查找用户类文件和注释处理程序的位置
  -cp <路径>                   指定查找用户类文件和注释处理程序的位置
  -sourcepath <路径>           指定查找输入源文件的位置
  -bootclasspath <路径>        覆盖引导类文件的位置
```

By CodeSheep

NODE环境安装

准备NODE安装包

我这里下载的是 `node-v12.16.3-linux-x64.tar.xz` 安装包，并将其直接放在了 `root` 目录下

创建目录并解压

1、在 `/usr/local/` 下创建 `node` 文件夹并进入

```
cd /usr/local/
mkdir node
cd node
```

2、将 `Node` 的安装包解压到 `/usr/local/node` 中即可

```
[root@localhost node]# tar -xJvf /root/node-v12.16.3-linux-x64.tar.xz -
C ./
```

解压完之后， `/usr/local/node` 目录中会出现一个 `node-v12.16.3-linux-x64` 的目录

配置NODE系统环境变量

编辑 `~/.bash_profile` 文件，在文件末尾追加如下信息:

```
# Nodejs
export PATH=/usr/local/node/node-v12.16.3-linux-x64/bin:$PATH
```

```
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

export PATH=$PATH:/usr/local/mysql/bin

# Nodejs
export PATH=/usr/local/node/node-v12.16.3-linux-x64/bin:$PATH
~
~
```


刷新环境变量，使之生效即可：

```
source ~/.bash_profile
```

检查安装结果

```
node -v

npm version

npx -v
```

均有版本信息输出即可：

```
[root@localhost bin]#
[root@localhost bin]# node -v
v12.16.3
[root@localhost bin]#
[root@localhost bin]# npm version
{
  npm: '6.14.4',
  ares: '1.16.0',
  brotli: '1.0.7',
  cldr: '36.0',
  http_parser: '2.9.3',
  icu: '65.1',
  llhttp: '2.0.4',
  modules: '72',
  napi: '5',
  nghttp2: '1.40.0',
  node: '12.16.3',
  openssl: '1.1.1g',
  tz: '2019c',
  unicode: '12.1',
  uv: '1.34.2',
  v8: '7.8.279.23-node.35',
  zlib: '1.2.11'
}
[root@localhost bin]#
[root@localhost bin]# npx -v
6.14.4
[root@localhost bin]#
```


本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

PYTHON环境安装

`CentOS 7.4` 默认自带了一个 `Python2.7` 环境：

```
[root@localhost ~]#
[root@localhost ~]# python
Python 2.7.5 (default, Aug  4 2017, 00:39:18)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>>
```

然而现在主流都是 `Python3`，所以接下来再装一个 `Python3`，打造一个共存的环境。

准备PYTHON3安装包并解压

我这里下载的是 `Python-3.8.3.tgz` 安装包，并将其直接放在了 `/root` 目录下

执行如下命令解压之：

```
tar zxvf Python-3.8.3.tgz
```

则可以在当前目录得到文件夹：`Python-3.8.3`

安装相关预备环境

直接执行如下命令即可：

```
yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-
devel readline-devel tk-devel gcc make
```

编译并安装

这里指定了安装目录为 `/usr/local/python3`，有需要可以自定义

```
cd Python-3.8.3/
./configure prefix=/usr/local/python3
make && make install
```

等安装过程完毕，`/usr/local/python3`目录就会生成了

添加软链接

我们还需要将刚刚安装生成的目录`/usr/local/python3`里的`python3`可执行文件做一份软链接，链接到`/usr/bin`下，方便后续使用python3

```
ln -s /usr/local/python3/bin/python3 /usr/bin/python3
ln -s /usr/local/python3/bin/pip3 /usr/bin/pip3
```

验证安装

命令行输入`python3`，即可查看`Python3`版本的安装结果

```
[root@localhost python3]# python3
Python 3.8.3 (default, May 15 2020, 11:39:55)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more info
>>>
>>>
>>>
```


而输入`python`，依然还是`python2.7.5`环境

```
[root@localhost python3]#
[root@localhost python3]# python
Python 2.7.5 (default, Aug  4 2017, 00:39:18)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux2
Type "help", "copyright", "credits" or "license" for more
>>>
>>>
>>>
```

MAVEN项目构建和管理工具安装

准备MAVEN安装包并解压

这里下载的是 `apache-maven-3.6.3-bin.tar.gz` 安装包，并将其放置于提前创建好的 `/opt/maven` 目录下。

执行命令解压之：

```
tar zxvf apache-maven-3.6.3-bin.tar.gz
```

即可在当前目录得到 `/opt/maven/apache-maven-3.6.3` 目录

配置MAVEN加速镜像源

这里配置的是阿里云的maven镜像源。

编辑修改 `/opt/maven/apache-maven-3.6.3/conf/settings.xml` 文件，在 `<mirrors></mirrors>` 标签对里添加如下内容即可：

```
<mirror>
        <id>alimaven</id>
        <name>aliyun maven</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <mirrorOf>central</mirrorOf>
</mirror>
```

```
143     | repository, to be used as an alternate download site. The mirror site will be th
144     | server for that repository.
145     |-->
146   <mirrors>
147     <!-- mirror
148     | Specifies a repository mirror site to use instead of a given repository. The r
149     | this mirror serves has an ID that matches the mirrorOf element of this mirror.
150     | for inheritance and direct lookup purposes, and must be unique across the set
151     |
152     <mirror>
153       <id>mirrorId</id>
154       <mirrorOf>repositoryId</mirrorOf>
155       <name>Human Readable Name for this Mirror.</name>
156       <url>http://my.repository.com/repo/path</url>
157     </mirror>
158      -->
159
160     <mirror>
161         <id>alimaven</id>
162         <name>aliyun maven</name>
163         <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
164         <mirrorOf>central</mirrorOf>
165     </mirror>
166   </mirrors>
167
168   <!-- profiles
```

CodeSheep

配置环境变量

因为下载的是二进制版安装包，所以解压完，配置好环境变量即可使用了。

编辑修改 `/etc/profile` 文件，在文件尾部添加如下内容，配置 `maven` 的安装路径

```
export MAVEN_HOME=/opt/maven/apache-maven-3.6.3
export PATH=$MAVEN_HOME/bin:$PATH
```

接下来执行 `source /etc/profile` 来刷新环境变量，让 `maven` 环境的路径配置生效，

检验安装结果

执行 `mvn -v` ，能打印出 `maven` 版本信息说明安装、配置成功：

MYSQL数据库部署和安装

首先准备安装包

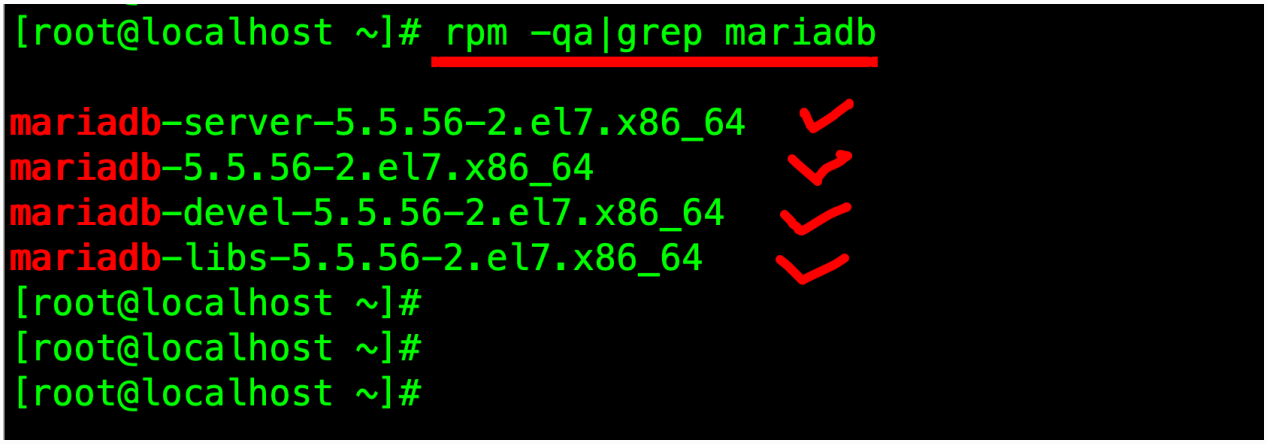
这里下载的是 `mysql-5.7.30-linux-glibc2.12-x86_64.tar.gz` 安装包，并将其直接放在了 `root` 目录下

卸载系统自带的MARIADB（如果有）

如果系统之前自带 `Mariadb`，可以先卸载之。

首先查询已安装的 `Mariadb` 安装包：

```
rpm -qa|grep mariadb
```



将其均卸载之：

```
yum -y remove mariadb-server-5.5.56-2.el7.x86_64
yum -y remove mariadb-5.5.56-2.el7.x86_64
yum -y remove mariadb-devel-5.5.56-2.el7.x86_64
yum -y remove mariadb-libs-5.5.56-2.el7.x86_64
```

解压MYSQL安装包

将上面准备好的 `MySQL` 安装包解压到 `/usr/local/` 目录，并重命名为 `mysql`

```
tar -zxvf /root/mysql-5.7.30-linux-glibc2.12-x86_64.tar.gz -C
/usr/local/
mv mysql-5.7.30-linux-glibc2.12-x86_64 mysql
```

创建MYSQL用户和用户组

```
groupadd mysql
useradd -g mysql mysql
```

同时新建 `/usr/local/mysql/data` 目录，后续备用

修改MYSQL目录的归属用户

(root@localhost mysql)# chown -R mysql:mysql ./

准备MYSQL的配置文件

在 `/etc` 目录下新建 `my.cnf` 文件

写入如下简化配置：

```
[mysql]
# 设置mysql客户端默认字符集
default-character-set=utf8
socket=/var/lib/mysql/mysql.sock

[mysqld]
skip-name-resolve
#设置3306端口
port = 3306
socket=/var/lib/mysql/mysql.sock
# 设置mysql的安装目录
basedir=/usr/local/mysql
# 设置mysql数据库的数据的存放目录
datadir=/usr/local/mysql/data
# 允许最大连接数
max_connections=200
# 服务端使用的字符集默认为8比特编码的latin1字符集
character-set-server=utf8
# 创建新表时将使用的默认存储引擎
default-storage-engine=INNODB
```

```
lower_case_table_names=1
max_allowed_packet=16M
```

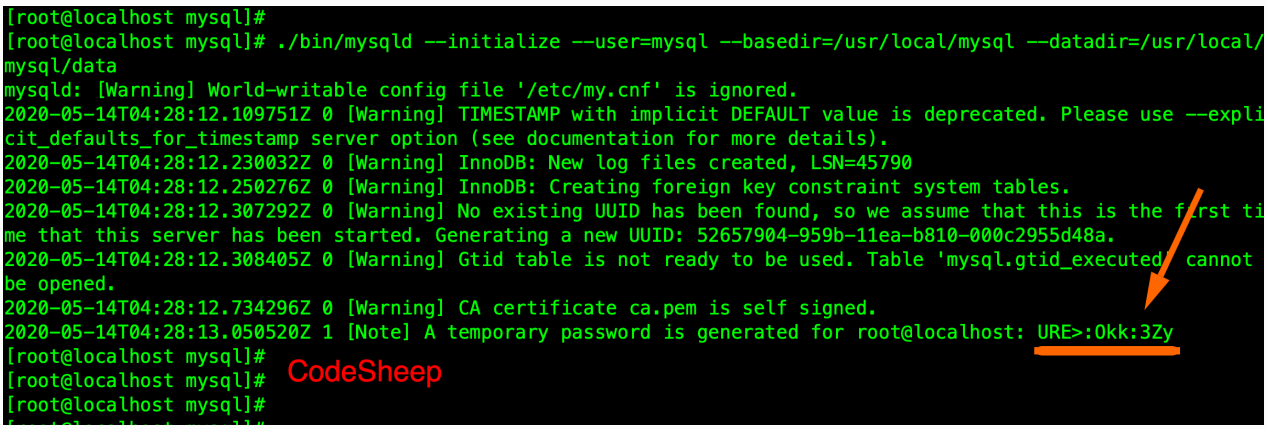
同时使用如下命令创建 `/var/lib/mysql` 目录，并修改权限：

```
mkdir /var/lib/mysql
chmod 777 /var/lib/mysql
```

正式开始安装MYSQL

执行如下命令正式开始安装：

```
cd /usr/local/mysql
./bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --
datadir=/usr/local/mysql/data
```



> 注意：记住上面打印出来的 `root` 的密码，后面首次登陆需要使用

复制启动脚本到资源目录

执行如下命令复制：

```
[root@localhost mysql]# cp ./support-files/mysql.server
/etc/init.d/mysqld
```

并修改 `/etc/init.d/mysqld`，修改其 `basedir` 和 `datadir` 为实际对应目录：

```
basedir=/usr/local/mysql
datadir=/usr/local/mysql/data
```

设置MYSQL系统服务并开启自启

首先增加 `mysqld` 服务控制脚本执行权限：

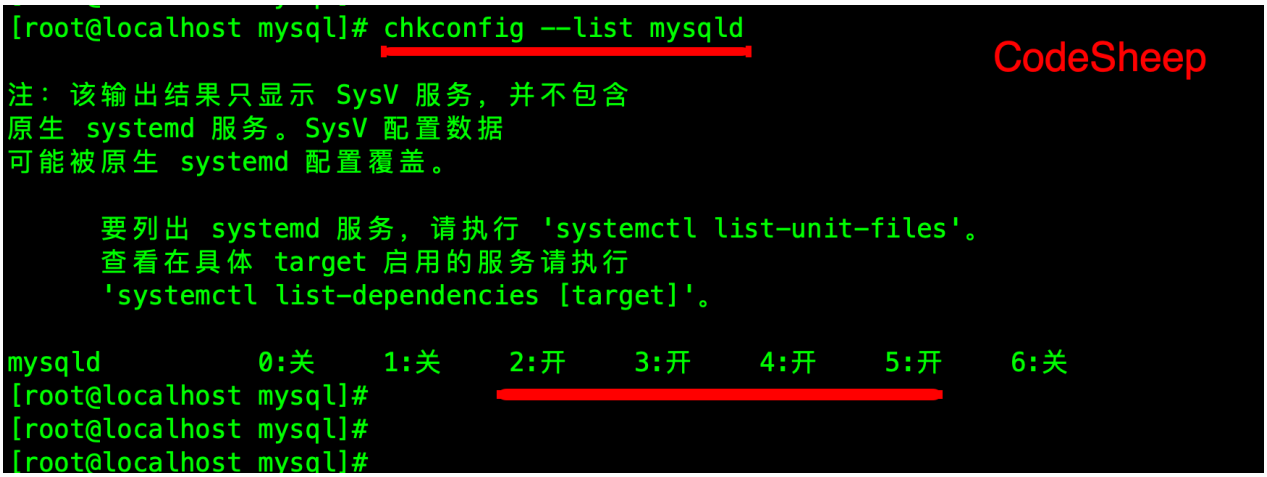
```
chmod +x /etc/init.d/mysqld
```

同时将 `mysqld` 服务加入到系统服务：

```
chkconfig --add mysqld
```

最后检查 `mysqld` 服务是否已经生效即可：

```
chkconfig --list mysqld
```



这样就表明 `mysqld` 服务已经生效了，在2、3、4、5运行级别随系统启动而自动启动，以后可以直接使用 `service` 命令控制 `mysql` 的启停。

启动MYSQLD

直接执行：

```
service mysqld start
```

```
[root@localhost mysql]#
[root@localhost mysql]#
[root@localhost mysql]#
[root@localhost mysql]# service mysqld start
Starting MySQL.

 SUCCESS!
[root@localhost mysql]#
[root@localhost mysql]#
[root@localhost mysql]#
[root@localhost mysql]#
[root@localhost mysql]#
```

CodeSheep

将 MYSQL 的 BIN 目录加入 PATH 环境变量

这样方便以后在任意目录上都可以使用 `mysql` 提供的命令。

编辑 `~/.bash_profile` 文件，在文件末尾处追加如下信息:

```
export PATH=$PATH:/usr/local/mysql/bin
```

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

export PATH=$PATH:/usr/local/mysql/bin
~
~
~
```

~/.bash_profile文件

CodeSheep

最后执行如下命令使环境变量生效

```
source ~/.bash_profile
```

首次登陆MYSQL

以 `root` 账户登录 `mysql`，使用上文安装完成提示的密码进行登入

```
mysql -u root -p
```

```
[root@localhost mysql]#
[root@localhost mysql]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.30

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All righ

Oracle is a registered trademark of Oracle Corporation and/or it
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current in
                                                    CodeSheep
mysql>                本地命令行登入成功
mysql>
```

接下来修改ROOT账户密码

在mysql的命令行执行如下命令即可，密码可以换成你想用的密码即可：

```
mysql>alter user user() identified by "111111";
mysql>flush privileges;
```

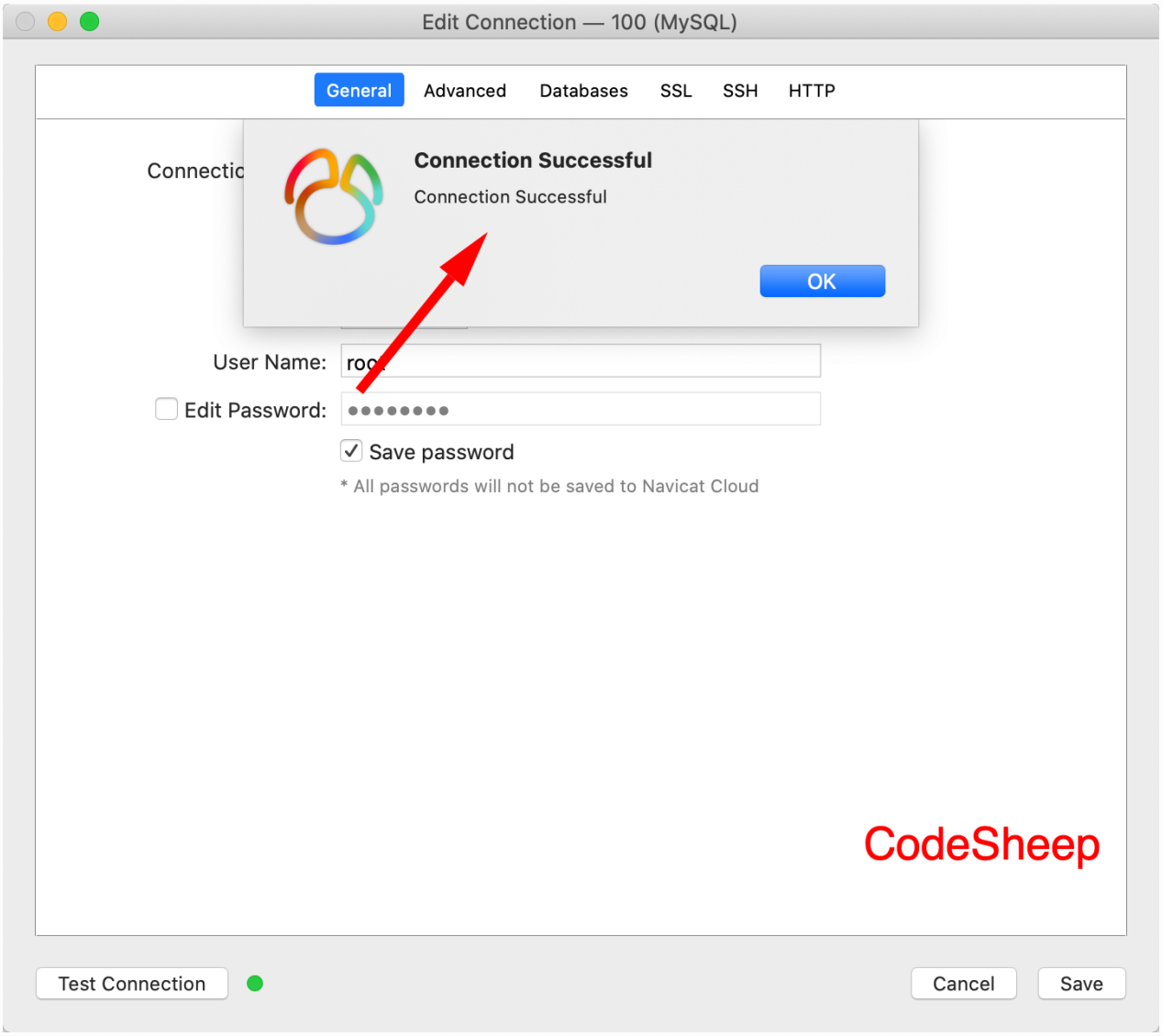
```
Type 'help;' or '\h' for help. Type '\c' to clear the current i

mysql>
mysql>
mysql> alter user user() identified by "111111";
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)
                                                    CodeSheep
mysql>
mysql>
```

比如这里将密码设置成简单的“111111”了。

设置远程主机登录

```
mysql> use mysql;
mysql> update user set user.Host='%' where user.User='root';
mysql> flush privileges;
```

最后利用NAVICAT等工具进行测试即可：



REDIS缓存安装部署

首先准备REDIS安装包

这里下载的是 `redis-5.0.8.tar.gz` 安装包，并将其直接放在了 `root` 目录下

解压安装包

1、在 `/usr/local/` 下创建 `redis` 文件夹并进入

```
cd /usr/local/
mkdir redis
cd redis
```

2、将 `Redis` 安装包解压到 `/usr/local/redis` 中即可

```
[root@localhost redis]# tar zxvf /root/redis-5.0.8.tar.gz -C ./
```

解压完之后， `/usr/local/redis` 目录中会出现一个 `redis-5.0.8` 的目录

编译并安装

```
cd redis-5.0.8/
make && make install
```

将 `REDIS` 安装为系统服务并后台启动

进入 `utils` 目录，并执行如下脚本即可：

```
[root@localhost redis-5.0.8]# cd utils/
[root@localhost utils]# ./install_server.sh
```

此处我全部选择的默认配置即可，有需要可以按需定制

```
[root@localhost utils]#
[root@localhost utils]# ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]
Selecting default: 6379
Please select the redis config file name [/etc/redis/6379.conf]
Selected default - /etc/redis/6379.conf
Please select the redis log file name [/var/log/redis_6379.log]
Selected default - /var/log/redis_6379.log
Please select the data directory for this instance [/var/lib/redis/6379]
Selected default - /var/lib/redis/6379
Please select the redis executable path [/usr/local/bin/redis-server]
Selected config:
Port           : 6379
Config file    : /etc/redis/6379.conf
Log file       : /var/log/redis_6379.log
Data dir       : /var/lib/redis/6379
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6379.conf => /etc/init.d/redis_6379
Installing service...
Successfully added to chkconfig!                    CodeSheep
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
[root@localhost utils]#
```

查看REDIS服务启动情况

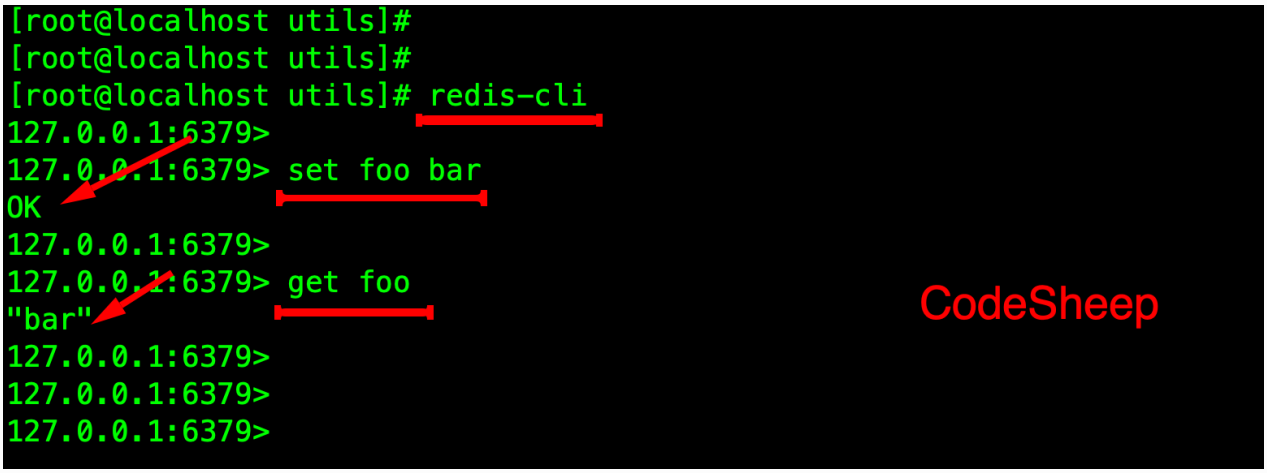
直接执行如下命令来查看Redis的启动结果：

```
systemctl status redis_6379.service
```

```
[root@localhost ~]#
[root@localhost ~]# systemctl status redis_6379.service
● redis_6379.service - LSB: start and stop redis_6379
   Loaded: loaded (/etc/rc.d/init.d/redis_6379; bad; vendor preset: disabled)
   Active: active (running) since 二 2020-05-19 21:52:16 CST; 49s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1915 ExecStart=/etc/rc.d/init.d/redis_6379 start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/redis_6379.service
           └─1949 /usr/local/bin/redis-server 0.0.0.0:6379

5月 19 21:52:16 localhost.localdomain systemd[1]: Starting LSB: start and stop redis_6379...
5月 19 21:52:16 localhost.localdomain redis_6379[1915]: Starting Redis server...
5月 19 21:52:16 localhost.localdomain systemd[1]: Started LSB: start and stop redis_6379.
[root@localhost ~]#
```

启动REDIS客户端并测试

启动自带的 `redis-cli` 客户端，测试通过：



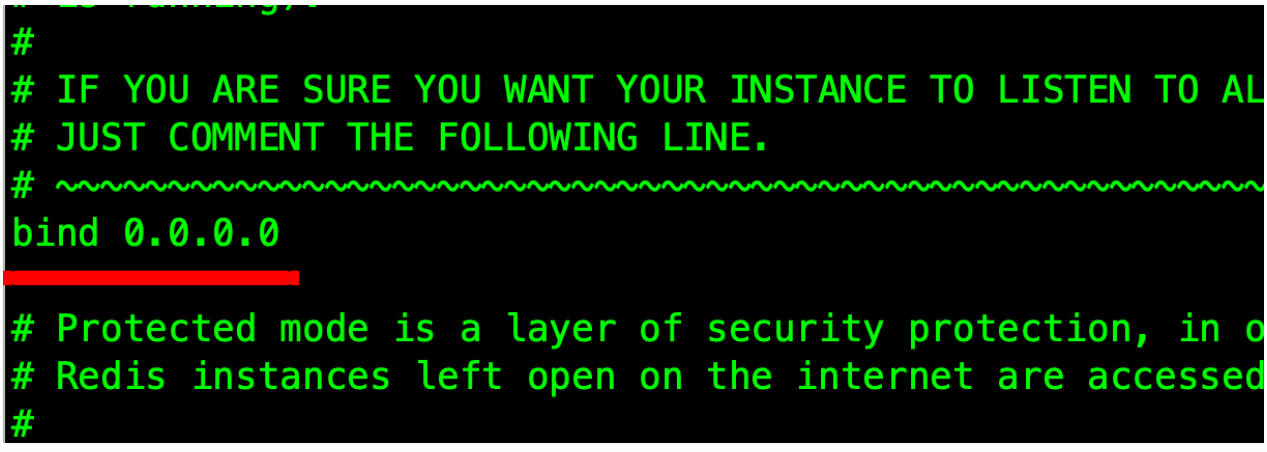
但是此时只能在本地访问，无法远程连接，因此还需要做部分设置

设置允许远程连接

编辑 `redis` 配置文件

```
vim /etc/redis/6379.conf
```

将 `bind 127.0.0.1` 修改为 `0.0.0.0`



然后重启 `Redis` 服务即可：

```
systemctl restart redis_6379.service
```

设置访问密码

编辑 `redis` 配置文件

```
vim /etc/redis/6379.conf
```

找到如下内容：

```
#requirepass foobared
```

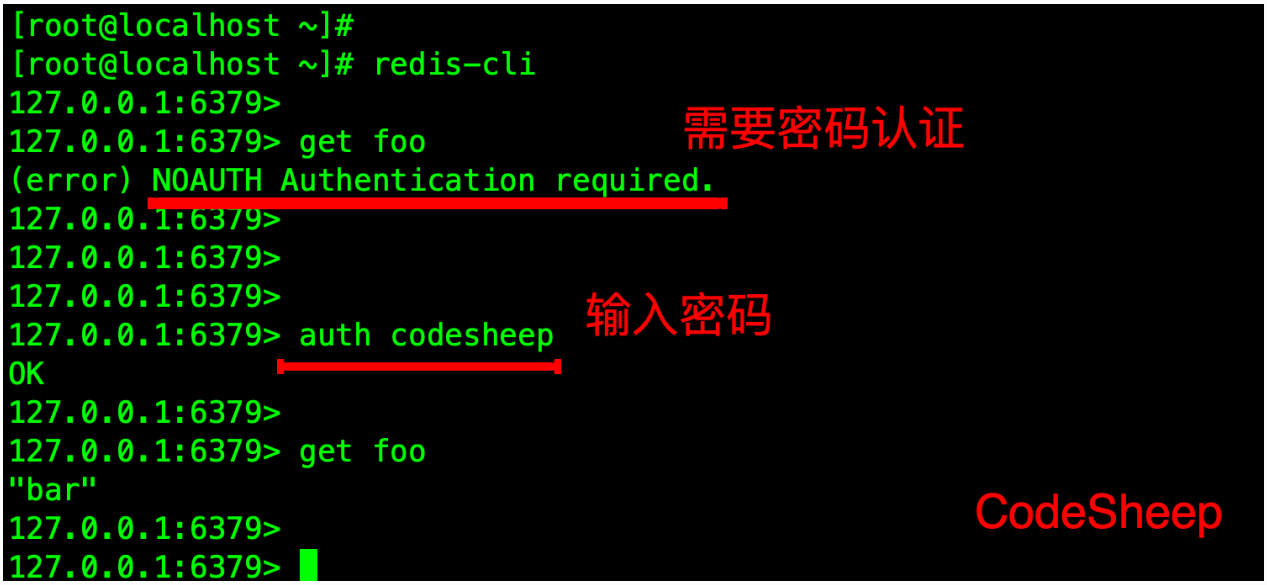
去掉注释，将 `foobared` 修改为自己想要的密码，保存即可。

```
requirepass codesheep
```

保存，重启 `Redis` 服务即可

```
systemctl restart redis_6379.service
```

这样后续的访问需要先输入密码认证通过方可：



消息队列RABBITMQ安装部署

首先安装ERLANG环境

因为 `RabbitMQ` 需要 `erlang` 环境的支持，所以必须先安装 `erlang` 。

我们这里要安装的是 `erlang-22.3.3-1.el7.x86_64.rpm`，所以首先执行如下命令来安装其对应的 `yum repo`：

```
curl -s
https://packagecloud.io/install/repositories/rabbitmq/erlang/script.rpm
.sh | sudo bash
```

```
Downloading packages:
No Presto metadata available for base
yum-utils-1.1.31-53.el7.noarch.rpm                                          | 122 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  正在更新    : yum-utils-1.1.31-53.el7.noarch                                                1/2
  清理        : yum-utils-1.1.31-42.el7.noarch                                                2/2
  验证中      : yum-utils-1.1.31-53.el7.noarch                                                1/2
  验证中      : yum-utils-1.1.31-42.el7.noarch                                                2/2

更新完毕：
  yum-utils.noarch 0:1.1.31-53.el7

完毕！
Generating yum cache for rabbitmq_erlang...
导入 GPG key 0xDF309A0B:
 用户ID     : "https://packagecloud.io/rabbitmq/erlang (https://packagecloud.io/docs#gpg_signing) <support@pack
gecloud.io>"
 指纹       : 2ebd e413 d3ce 5d35 bcd1 5b7c 71c6 3471 df30 9a0b
 来自       : https://packagecloud.io/rabbitmq/erlang/gpgkey
Generating yum cache for rabbitmq_erlang-source...
https://packagecloud.io/rabbitmq/erlang/el/7/SRPMS/repodata/other.xml.gz: [Errno 12] Timeout on https://package
loud.io/rabbitmq/erlang/el/7/SRPMS/repodata/other.xml.gz: (28, 'Operation timed out after 30000 milliseconds wi
h 0 out of 0 bytes received')
正在尝试其它镜像。
https://packagecloud.io/rabbitmq/erlang/el/7/SRPMS/repodata/other.xml.gz: [Errno 14] curl#7 - "Failed to connec
 to 2600:1f1c:2e5:6900:5df8:6a7b:6fb7:c697: Network is unreachable"
正在尝试其它镜像。

The repository is setup! You can now install packages.
[root@localhost ~]#
[root@localhost ~]#
```

接下来执行如下命令正式安装 `erlang` 环境：

```
yum install erlang-22.3.3-1.el7.x86_64
```

```
[root@localhost ~]# yum install erlang-22.3.3-1.el7.x86_64
已加载插件：fastestmirror, langpacks

Loading mirror speeds from cached hostfile
 * base: mirrors.nju.edu.cn
 * extras: mirrors.nju.edu.cn
 * updates: mirrors.nju.edu.cn
正在解决依赖关系
--> 正在检查事务
---> 软件包 erlang.x86_64.0.22.3.3-1.el7 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================
 Package              架构                 版本                       源                 大小
================================================================================
正在安装：
 erlang               x86_64               22.3.3-1.el7               rabbitmq_erlang    19 M

事务概要
================================================================================
安装  1 软件包

总下载量：19 M
安装大小：33 M
Is this ok [y/d/N]: Exiting on user command
您的事务已保存，请执行：
 yum load-transaction /tmp/yum_save_tx.2020-05-14.22-21.n0cwzm.yumtx 重新执行该事务
[root@localhost ~]#
```

CodeSheep

需要再次执行该命令

接着上面提示再次执行如下命令即可：

```
yum load-transaction /tmp/yum_save_tx.2020-05-14.22-21.n0cwzm.yumtx
```

```
================================================================================
 Package              架构                 版本                       源                 大小
================================================================================
正在安装：
 erlang               x86_64               22.3.3-1.el7               rabbitmq_erlang    19 M

事务概要
================================================================================
安装  1 软件包

总下载量：19 M
安装大小：33 M
Is this ok [y/d/N]: Y
Downloading packages:
erlang-22.3.3-1.el7.x86_64.rpm                                | 19 MB  00:00:58
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  正在安装    : erlang-22.3.3-1.el7.x86_64                                  1/1
  验证中      : erlang-22.3.3-1.el7.x86_64                                  1/1

已安装：
  erlang.x86_64 0:22.3.3-1.el7

完毕！
[root@localhost ~]#
```

接下来可以直接执行如下命令，测试 `erlang` 是否安装成功：

```
erl
```

```
[root@localhost ~]#
[root@localhost ~]# erl
Erlang/OTP 22 [erts-10.7.1] [source] [64-bit] [smp:2:2] [ds:2:2:10] [async-threads:1] [hipe]

Eshell V10.7.1  (abort with ^G)
1>
1>
1>
```

安装RABBITMQ

接下来正式开始安装 `rabbitmq`，首先依然是安装其对应的 `yum repo`：

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.rpm.sh | sudo bash
```

```
Generating yum cache for rabbitmq_rabbitmq-server...
导入 GPG key 0x4D206F89:
 用户ID     : "https://packagecloud.io/rabbitmq/rabbitmq-server (https://packageclou
 指纹       : 8c69 5b02 19af deb0 4a05 8ed8 f4e7 8920 4d20 6f89
 来自       : https://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey
https://packagecloud.io/rabbitmq/rabbitmq-server/el/7/x86_64/repodata/5bf2d7a2f580b2
hfq314.cloudfront.net/828/1033/el/7/x86_64/repodata/5bf2d7a2f580b26a717ad8c27979bf80
8, 'Operation timed out after 30005 milliseconds with 0 out of 0 bytes received')
正在尝试其它镜像。
Generating yum cache for rabbitmq_rabbitmq-server-source...

The repository is setup! You can now install packages.
[root@localhost ~]#
```

然后执行如下命令正式安装 `rabbitmq`：

```
yum install rabbitmq-server-3.8.3-1.el7.noarch
```

```
依赖关系解决

================================================================================
 Package                架构          版本                源              大小
================================================================================
正在安装:
 rabbitmq-server        noarch        3.8.3-1.el7         rabbitmq_rabbitmq-server  12 M
为依赖而安装:
 socat                  x86_64        1.7.3.2-2.el7       base             290 k

事务概要
================================================================================
安装  1 软件包 (+1 依赖软件包)

总下载量: 12 M
安装大小: 14 M
```

```
总计
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  正在安装     : socat-1.7.3.2-2.el7.x86_64
  正在安装     : rabbitmq-server-3.8.3-1.el7.noarch
  验证中       : socat-1.7.3.2-2.el7.x86_64
  验证中       : rabbitmq-server-3.8.3-1.el7.noarch

已安装:
  rabbitmq-server.noarch 0:3.8.3-1.el7

作为依赖被安装:
  socat.x86_64 0:1.7.3.2-2.el7

完毕!
[root@localhost ~]#
```

设置RABBITMQ开机启动

```
chkconfig rabbitmq-server on
```

启动RABBITMQ服务

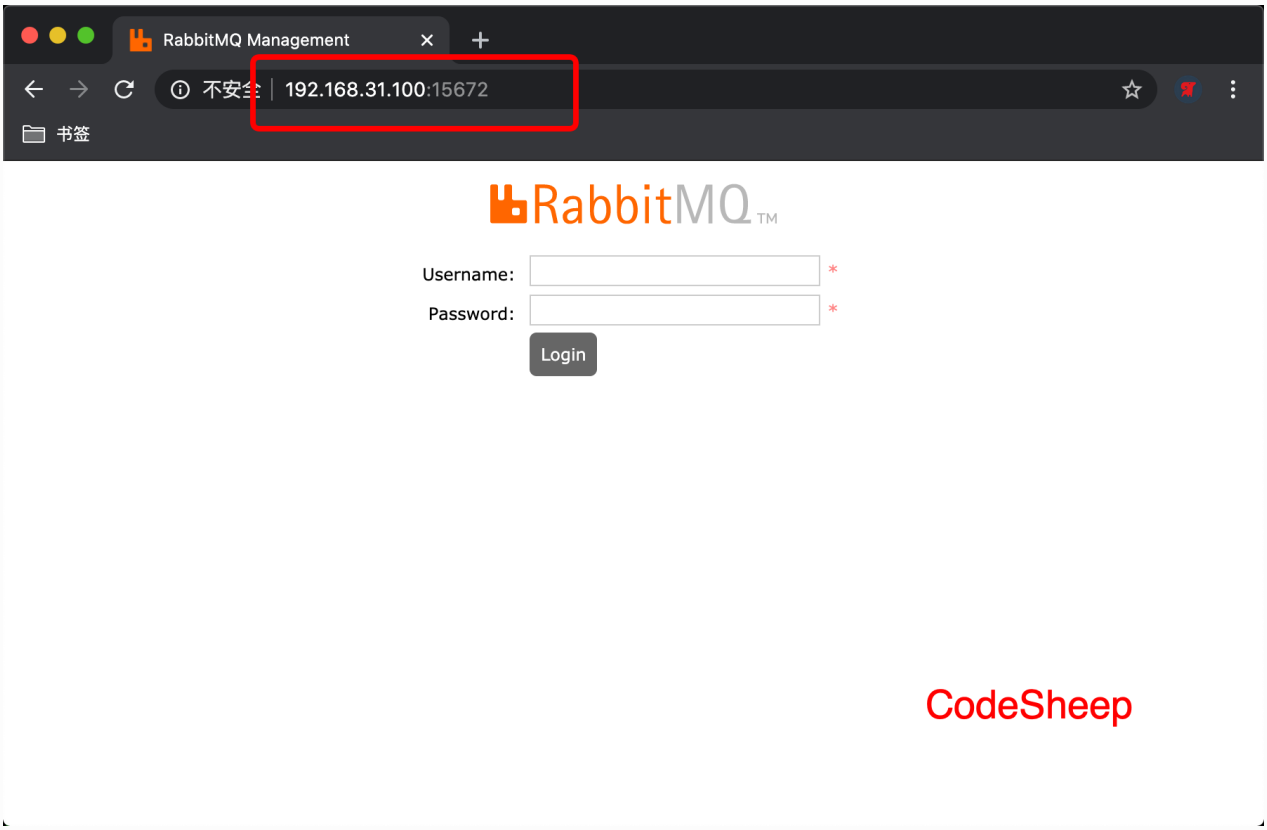
```
systemctl start rabbitmq-server.service
```

开启WEB可视化管理插件：

```
rabbitmq-plugins enable rabbitmq_management
```

访问可视化管理界面：

浏览器输入：`你的服务器IP:15672`

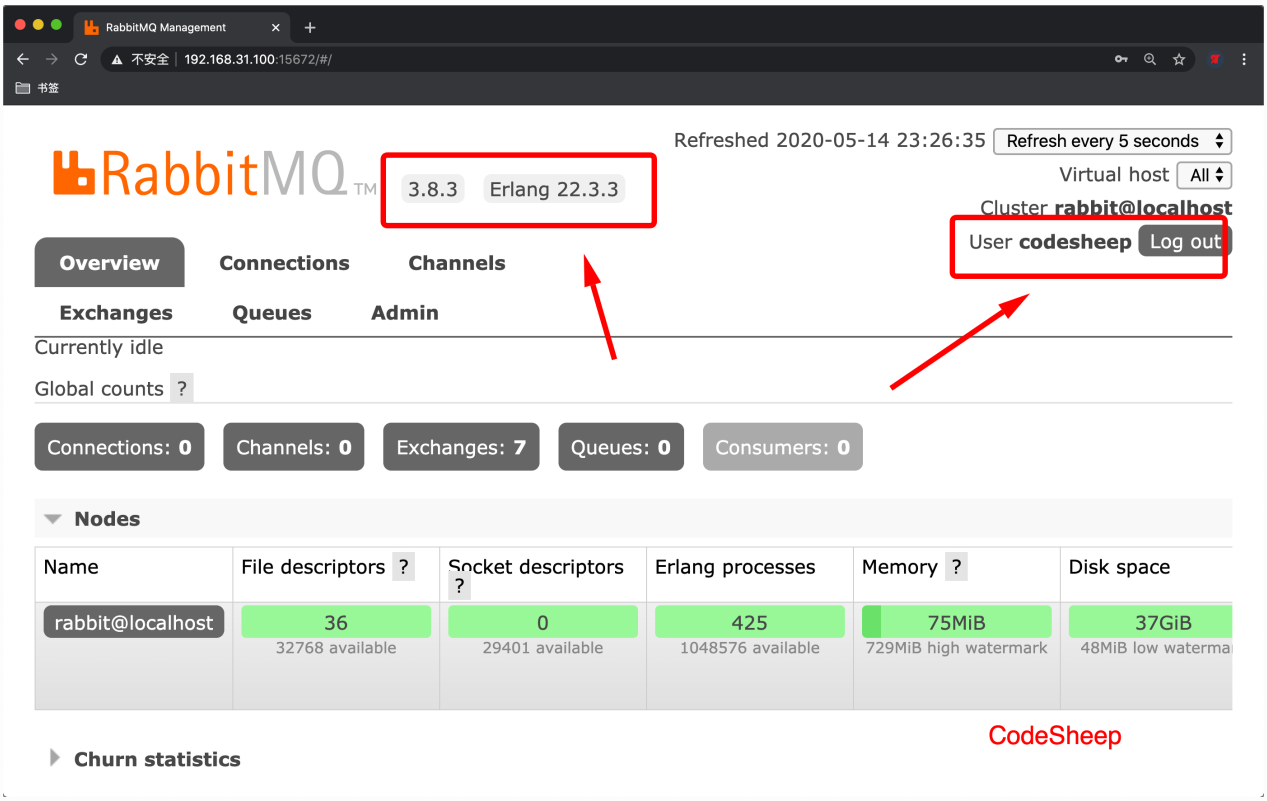


能看到网页登录入口即可。

我们可以在后台先添加一个用户/密码对：

```
rabbitmqctl add_user codesheep 123456
rabbitmqctl set_user_tags codesheep administrator
```

然后登录网页即可：



应用服务器TOMCAT安装部署

准备安装包

这里使用的是 `8.5.55` 版： `apache-tomcat-8.5.55.tar.gz` ，直接将其放在了 `/root` 目录下

解压并安装

在 `/usr/local/` 下创建 `tomcat` 文件夹并进入

```
cd /usr/local/
mkdir tomcat
cd tomcat
```

2、将 `Tomcat` 安装包解压到 `/usr/local/tomcat` 中即可

```
[root@localhost tomcat]# tar -zxvf /root/apache-tomcat-8.5.55.tar.gz -C
./
```

解压完之后， `/usr/local/tomcat` 目录中会出现一个 `apache-tomcat-8.5.55` 的目录

启动TOMCAT

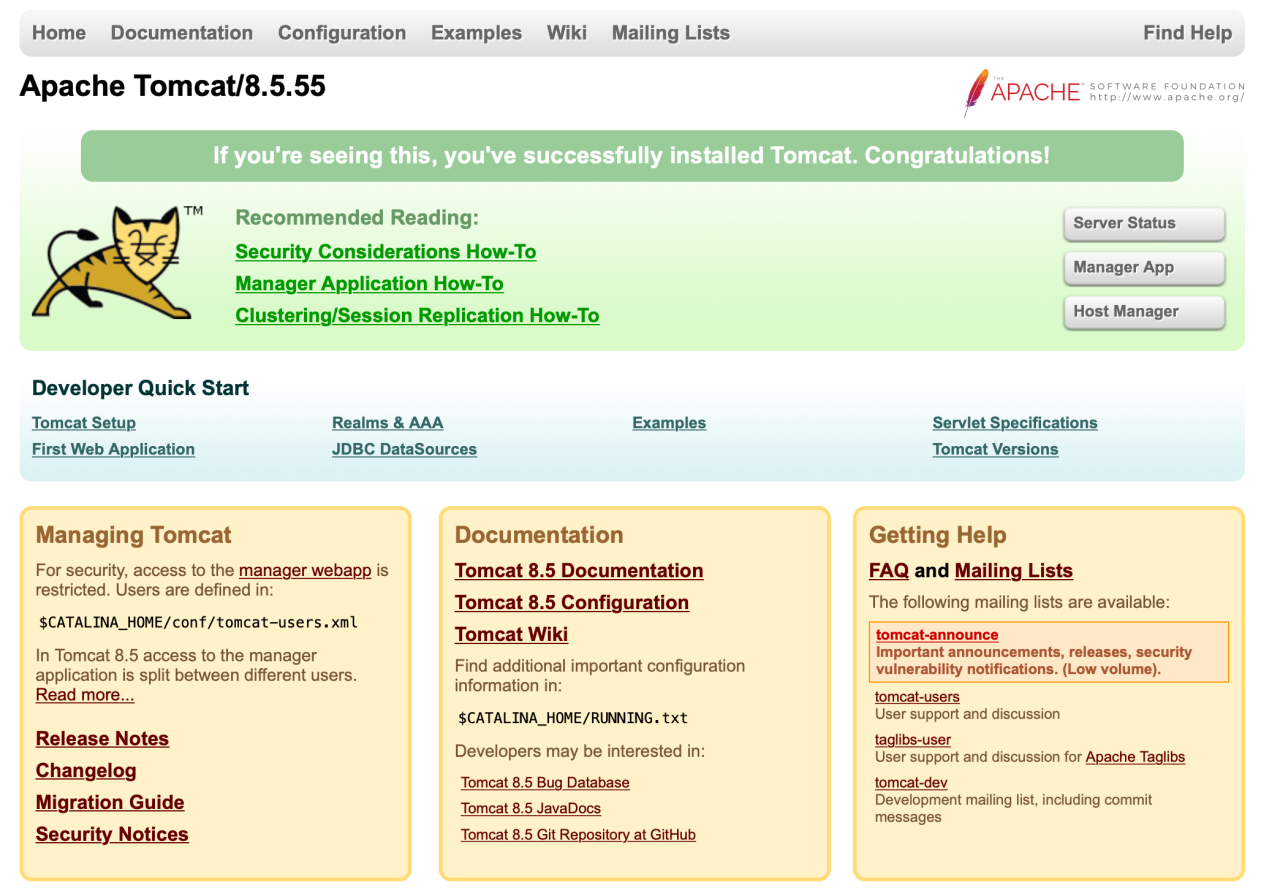
直接进 `apache-tomcat-8.5.55` 目录，执行其中 `bin` 目录下的启动脚本即可

```
[root@localhost apache-tomcat-8.5.55]# cd bin/
[root@localhost bin]# ./startup.sh
```

```
[root@localhost bin]#
[root@localhost bin]# ./startup.sh
Using CATALINA_BASE:   /usr/local/tomcat/apache-tomcat-8.5.55
Using CATALINA_HOME:   /usr/local/tomcat/apache-tomcat-8.5.55
Using CATALINA_TMPDIR: /usr/local/tomcat/apache-tomcat-8.5.55/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/tomcat/apache-tomcat-8.5.55/bin/boo
/local/tomcat/apache-tomcat-8.5.55/bin/tomcat-juli.jar
Tomcat started.
[root@localhost bin]#
```

CodeSheep

这时候浏览器访问：`你的主机IP:8080`，得到如下画面说明成功启动了



配置快捷操作和开机启动

首先进入`/etc/rc.d/init.d`目录，创建一个名为`tomcat`的文件，并赋予执行权限

```
[root@localhost ~]# cd /etc/rc.d/init.d/
[root@localhost init.d]# touch tomcat
[root@localhost init.d]# chmod +x tomcat
```

接下来编辑`tomcat`文件，并在其中加入如下内容：

```
#!/bin/bash
#chkconfig:- 20 90
#description:tomcat
#processname:tomcat
TOMCAT_HOME=/usr/local/tomcat/apache-tomcat-8.5.55
case $1 in
        start) su root $TOMCAT_HOME/bin/startup.sh;;
        stop) su root $TOMCAT_HOME/bin/shutdown.sh;;
        *) echo "require start|stop" ;;

esac
```

这样后续对于Tomcat的开启和关闭只需要执行如下命令即可：

```
service tomcat start
service tomcat stop
```

最后加入开机启动即可：

```
chkconfig --add tomcat
chkconfig tomcat on
```

WEB服务器NGINX安装部署

首先安装包并解压

这里下载的是 `nginx-1.17.10.tar.gz` 安装包，并将其直接放在了 `root` 目录下

1、在 `/usr/local/` 下创建 `nginx` 文件夹并进入

```
cd /usr/local/
mkdir nginx
cd nginx
```

2、将 `Nginx` 安装包解压到 `/usr/local/nginx` 中即可

```
[root@localhost nginx]# tar zxvf /root/nginx-1.17.10.tar.gz -C ./
```

解压完之后， `/usr/local/nginx` 目录中会出现一个 `nginx-1.17.10` 的目录

预先安装额外的依赖

```
yum -y install pcre-devel
yum -y install openssl openssl-devel
```

编译安装NGINX

```
cd nginx-1.17.10
./configure
make && make install
```

安装完成后，Nginx的可执行文件位置位于

```
/usr/local/nginx/sbin/nginx
```

启动NGINX

直接执行如下命令即可：

```
[root@localhost sbin]# /usr/local/nginx/sbin/nginx
```

如果想停止Nginx服务，可执行：

```
/usr/local/nginx/sbin/nginx -s stop
```

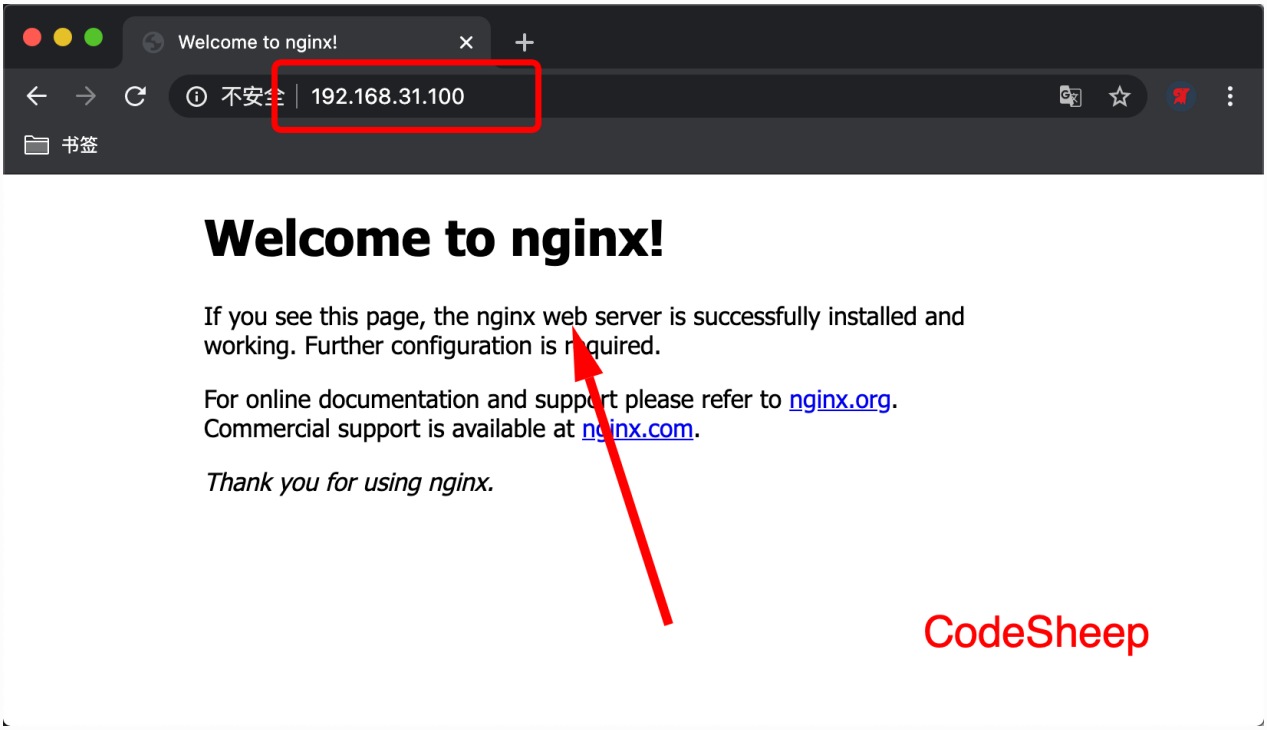
如果修改了配置文件后想重新加载Nginx，可执行：

```
/usr/local/nginx/sbin/nginx -s reload
```

注意其配置文件位于：

```
/usr/local/nginx/conf/nginx.conf
```

浏览器验证启动情况



DOCKER环境安装

安装DOCKER

```
yum install -y docker
```

静候安装完毕即可

开启DOCKER服务

```
systemctl start docker.service
```

```
[root@localhost ~]# docker version
Client:
 Version:         1.13.1
 API version:     1.26
 Package version: docker-1.13.1-161.git64e9980.el7_8.x86_64
 Go version:      go1.10.3
 Git commit:      64e9980/1.13.1
 Built:           Tue Apr 28 14:43:01 2020
 OS/Arch:         linux/amd64

Server:
 Version:         1.13.1
 API version:     1.26 (minimum version 1.12)
 Package version: docker-1.13.1-161.git64e9980.el7_8.x86_64
 Go version:      go1.10.3
 Git commit:      64e9980/1.13.1
 Built:           Tue Apr 28 14:43:01 2020
 OS/Arch:         linux/amd64
 Experimental:    false
[root@localhost ~]#
[root@localhost ~]#
```


查看安装结果

```
docker version
```

```
[root@localhost ~]#
[root@localhost ~]# docker version
Client:
 Version:         1.13.1
 API version:     1.26
 Package version:
```


设置开机启动

```
systemctl enable docker.service
```

配置DOCKER镜像下载加速

默认安装后的 `Docker` 环境在拉取 `Docker` 镜像时速度很慢：

```
[root@localhost ~]# docker pull mysql
Using default tag: latest
Trying to pull repository docker.io/library/mysql ...
latest: Pulling from docker.io/library/mysql
afb6ec6fdc1c: Downloading 1.403 MB/27.1 MB
0bdc5971ba40: Download complete
97ae94a2c729: Downloading 1.698 MB/4.178 MB
f777521d340e: Downloading 162.9 kB/437.1 kB
1393ff7fc871: Waiting
a499b89994d9: Waiting
7ebe8eefbafe: Waiting
597069368ef1: Waiting
ce39a5501878: Waiting
7d545bca14bf: Waiting
0f5f78cccacb: Waiting
623a5dae2b42: Waiting
```

因此我们需要手动配置镜像加速源，提升获取 `Docker` 镜像的速度。

配置方法非常简单。

直接编辑配置文件：

```
vim /etc/docker/daemon.json
```

在其中加入加速镜像源地址即可：

```
{
  "registry-mirrors": ["http://hub-mirror.c.163.com"]
}
```

比如这里使用的是 `网易` 的加速源，其他像 `阿里云` 、 `DaoCloud` 这些也都提供加速源，按需选择即可。

加完加速地址后，重新加载配置文件，重启 `docker` 服务即可：

```
systemctl daemon-reload
systemctl restart docker.service
```

这样镜像加速器就配置完成了，后续下载 `docker` 镜像速度将大增。

KUBERNETES集群部署

概 述

Kubernetes集群的搭建方法其实有多种，比如我在之前的文章《利用K8S技术栈打造个人私有云（连载之：K8S集群搭建）》中使用的就是二进制的安装方法。虽然这种方法有利于我们理解 k8s集群，但却过于繁琐。而 kubeadm是 Kubernetes官方提供的用于快速部署Kubernetes集群的工具，其历经发展如今已经比较成熟了，利用其来部署 Kubernetes集群可以说是非常好上手，操作起来也简便了许多，下面详细叙述之。

节点规划

本文准备部署一个 **一主两从** 的 **三节点** Kubernetes集群，整体节点规划如下表所示：

主机名	IP	角色
k8s-master	192.168.39.79	k8s主节点
k8s-node-1	192.168.39.77	k8s从节点
k8s-node-2	192.168.39.78	k8s从节点

下面介绍一下各个节点的软件版本：

- 操作系统： `CentOS-7.4-64Bit`
- Docker版本： `1.13.1`
- Kubernetes版本： `1.13.1`

所有节点都需要安装以下组件：

- `Docker`： 不用多说了吧
- `kubelet`： 运行于所有 Node上，负责启动容器和 Pod
- `kubeadm`： 负责初始化集群
- `kubectl`： k8s命令行工具，通过其可以部署/管理应用 以及CRUD各种资源

准备工作

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

- 所有节点关闭防火墙

```
systemctl disable firewalld.service
systemctl stop firewalld.service
```

- 禁用SELINUX

```
setenforce 0

vi /etc/selinux/config
SELINUX=disabled
```

- 所有节点关闭 swap

```
swapoff -a
```

- 设置所有节点主机名

```
hostnamectl --static set-hostname  k8s-master
hostnamectl --static set-hostname  k8s-node-1
hostnamectl --static set-hostname  k8s-node-2
```

- 所有节点 主机名/IP加入 hosts解析

编辑 `/etc/hosts` 文件，加入以下内容：

```
192.168.39.79 k8s-master
192.168.39.77 k8s-node-1
192.168.39.78 k8s-node-2
```

组件安装

0x01. Docker安装（所有节点）

> 不赘述了，参考上文Docker环境安装

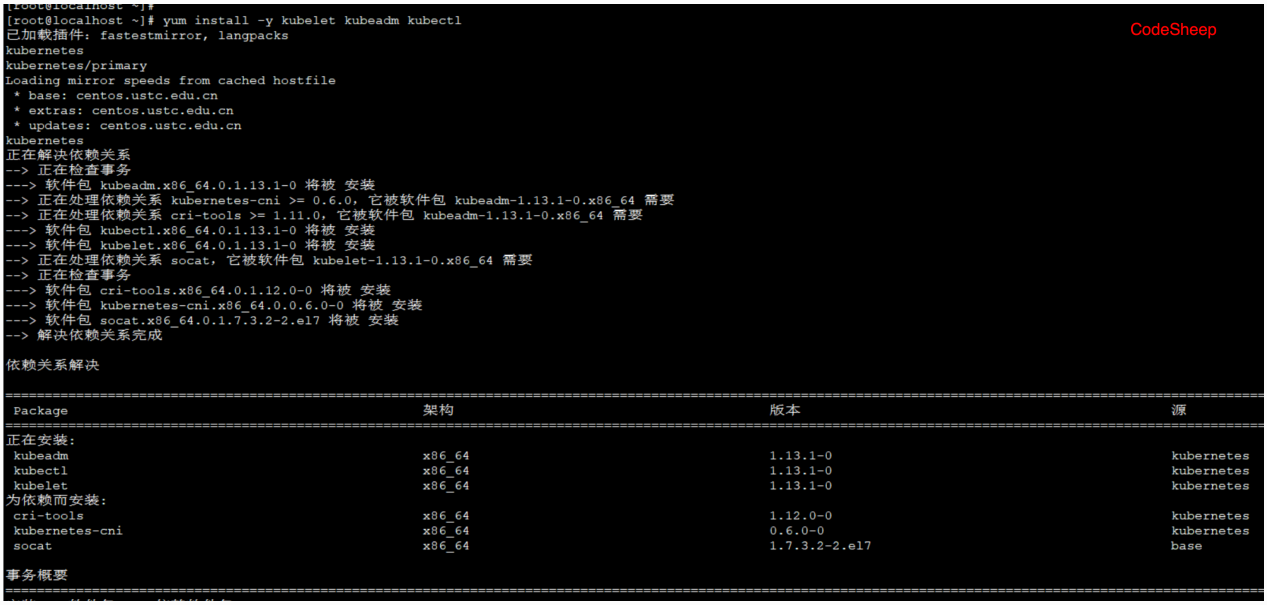
0x02. kubelet、kubeadm、kubectl安装（所有节点）

- 首先准备repo

```
cat>>/etc/yum.repos.d/kubrenetes.repo<<EOF
[kubernetes]
name=Kubernetes Repo
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-
x86_64/
gpgcheck=0
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
EOF
```

- 然后执行如下指令来进行安装

```
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX= disabled/' /etc/selinux/config

yum install -y kubelet kubeadm kubectl
systemctl enable kubelet && systemctl start kubelet
```



MASTER节点配置

0x01. 初始化 k8s集群

为了应对网络不畅通的问题，我们国内网络环境只能提前手动下载相关镜像并重新打 tag :

```
docker pull mirrorgooglecontainers/kube-apiserver:v1.13.1
docker pull mirrorgooglecontainers/kube-controller-manager:v1.13.1
docker pull mirrorgooglecontainers/kube-scheduler:v1.13.1
docker pull mirrorgooglecontainers/kube-proxy:v1.13.1
docker pull mirrorgooglecontainers/pause:3.1
docker pull mirrorgooglecontainers/etcd:3.2.24
docker pull coredns/coredns:1.2.6
```

```
docker pull registry.cn-shenzhen.aliyuncs.com/cp_m/flannel:v0.10.0-amd64

docker tag mirrorgooglecontainers/kube-apiserver:v1.13.1 k8s.gcr.io/kube-apiserver:v1.13.1
docker tag mirrorgooglecontainers/kube-controller-manager:v1.13.1 k8s.gcr.io/kube-controller-manager:v1.13.1
docker tag mirrorgooglecontainers/kube-scheduler:v1.13.1 k8s.gcr.io/kube-scheduler:v1.13.1
docker tag mirrorgooglecontainers/kube-proxy:v1.13.1 k8s.gcr.io/kube-proxy:v1.13.1
docker tag mirrorgooglecontainers/pause:3.1 k8s.gcr.io/pause:3.1
docker tag mirrorgooglecontainers/etcd:3.2.24 k8s.gcr.io/etcd:3.2.24
docker tag coredns/coredns:1.2.6 k8s.gcr.io/coredns:1.2.6
docker tag registry.cn-shenzhen.aliyuncs.com/cp_m/flannel:v0.10.0-amd64 quay.io/coreos/flannel:v0.10.0-amd64

docker rmi mirrorgooglecontainers/kube-apiserver:v1.13.1
docker rmi mirrorgooglecontainers/kube-controller-manager:v1.13.1
docker rmi mirrorgooglecontainers/kube-scheduler:v1.13.1
docker rmi mirrorgooglecontainers/kube-proxy:v1.13.1
docker rmi mirrorgooglecontainers/pause:3.1
docker rmi mirrorgooglecontainers/etcd:3.2.24
docker rmi coredns/coredns:1.2.6
docker rmi registry.cn-shenzhen.aliyuncs.com/cp_m/flannel:v0.10.0-amd64
```

```
[root@localhost ~]# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
k8s.gcr.io/kube-proxy                     v1.13.1             fdb321fd30a0        10 days ago         80.2 MB
k8s.gcr.io/kube-apiserver                 v1.13.1             40a63db91ef8        10 days ago         181 MB
k8s.gcr.io/kube-controller-manager        v1.13.1             26e6f1db2a52        10 days ago         146 MB
k8s.gcr.io/kube-scheduler                 v1.13.1             ab81d7360408        10 days ago         79.6 MB
k8s.gcr.io/coredns                        1.2.6               f59dcacceff4        7 weeks ago         40 MB
k8s.gcr.io/etcd                           3.2.24              3cab8e1b9802        3 months ago        220 MB
k8s.gcr.io/pause                          3.1                 da86e6ba6ca1        12 months ago       742 kB
[root@localhost ~]#
```

然后再在 Master节点上执行如下命令初始化 k8s集群：

```
kubeadm init --kubernetes-version=v1.13.1 --apiserver-advertise-address 192.168.39.79 --pod-network-cidr=10.244.0.0/16
```

- `--kubernetes-version`：用于指定 k8s版本
- `--apiserver-advertise-address`：用于指定使用 Master的哪个network interface进行通信，若不指定，则 kubeadm会自动选择具有默认网关的 interface
- `--pod-network-cidr`：用于指定Pod的网络范围。该参数使用依赖于使用的网络方案，本文将使用经典的flannel网络方案。

执行命令后，控制台给出了如下所示的详细集群初始化过程：

```
[root@localhost ~]# kubeadm init --config kubeadm-config.yaml
W1224 11:01:25.408209   10137 strict.go:54] error unmarshaling configuration schema.GroupVersionKind{Group:"kubeadm.k8s.io", Version:"v1beta1", Kind:"ClusterConfiguration"}: error unmarshaling JSON: while decoding JSON: json: unknown field "\u00a0 podSubnet"
```

```
[init] Using Kubernetes version: v1.13.1
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of
your internet connection
[preflight] You can also perform this action in beforehand using
'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names
[localhost.localdomain localhost] and IPs [192.168.39.79 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names
[localhost.localdomain localhost] and IPs [192.168.39.79 127.0.0.1 ::1]
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names
[localhost.localdomain kubernetes kubernetes.default
kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs
[10.96.0.1 192.168.39.79]
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-
manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in
"/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control
plane as static Pods from directory "/etc/kubernetes/manifests". This
can take up to 4m0s
[apiclient] All control plane components are healthy after 24.005638
seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-
config" in the "kube-system" Namespace
```

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

```
[kubelet] Creating a ConfigMap "kubelet-config-1.13" in namespace kube-
system with the configuration for the kubelets in the cluster
[patchnode] Uploading the CRI Socket information
"/var/run/dockershim.sock" to the Node API object
"localhost.localdomain" as an annotation
[mark-control-plane] Marking the node localhost.localdomain as control-
plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node localhost.localdomain as control-
plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: 26uprk.t7vpbwxojest0tvq
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap,
RBAC Roles
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens
to post CSRs in order for nodes to get long term certificate
credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover
controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation
for all node client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-
public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular
user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed
at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on
each node
as root:

  kubeadm join 192.168.39.79:6443 --token 26uprk.t7vpbwxojest0tvq --
discovery-token-ca-cert-hash
sha256:028727c0c21f22dd29d119b080dcbebb37f5545e7da1968800140ffe225b0123

[root@localhost ~]#
```

0x02. 配置 kubectl

在 Master上用 root用户执行下列命令来配置 kubectl：

```
echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> /etc/profile
source /etc/profile
echo $KUBECONFIG
```

0x03. 安装Pod网络

安装 Pod网络是 Pod之间进行通信的必要条件，k8s支持众多网络方案，这里我们依然选用经典的flannel方案

- 首先设置系统参数：

```
sysctl net.bridge.bridge-nf-call-iptables=1
```

- 然后在 Master节点上执行如下命令：

```
kubectl apply -f kube-flannel.yaml
```

`kube-flannel.yaml` 文件下载地址：

> 下载地址1（直接点击）
> 下载地址2（直接点击）

一旦 Pod网络安装完成，可以执行如下命令检查一下 CoreDNS Pod此刻是否正常运行起来了，一旦其正常运行起来，则可以继续后续步骤

```
kubectl get pods --all-namespaces -o wide
```

```
[root@localhost ~]#
[root@localhost ~]# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                          READY   STATUS    RESTARTS   AGE     IP              NODE
kube-system   coredns-86c58d9df4-2kh6s                      1/1     Running   0          5m24s   10.244.0.3      localhost.localdomain
kube-system   coredns-86c58d9df4-sh4cv                      1/1     Running   0          5m24s   10.244.0.2      localhost.localdomain
kube-system   etcd-localhost.localdomain                    1/1     Running   0          4m34s   192.168.199.79  localhost.localdomain
kube-system   kube-apiserver-localhost.localdomain          1/1     Running   0          4m29s   192.168.199.79  localhost.localdomain
kube-system   kube-controller-manager-localhost.localdomain 1/1     Running   0          4m23s   192.168.199.79  localhost.localdomain
kube-system   kube-flannel-ds-amd64-5hb9l                   1/1     Running   0          3m27s   192.168.199.79  localhost.localdomain
kube-system   kube-proxy-576j6                              1/1     Running   0          5m24s   192.168.199.79  localhost.localdomain
kube-system   kube-scheduler-localhost.localdomain          1/1     Running   0          4m20s   192.168.199.79  localhost.localdomain
[root@localhost ~]#
```

同时我们可以看到主节点已经就绪：`kubectl get nodes`

```
[root@localhost ~]# kubectl get nodes
NAME                    STATUS   ROLES    AGE     VERSION
localhost.localdomain   Ready    master   6m49s   v1.13.1
[root@localhost ~]#
[root@localhost ~]#
```

添加 SLAVE节点

在两个 Slave节点上分别执行如下命令来让其加入Master上已经就绪了的 k8s集群：

```
kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256:<hash>
```

如果 token忘记，则可以去 Master上执行如下命令来获取：

```
kubeadm token list
```

上述kubectl join命令的执行结果如下：

```
[root@localhost ~]# kubeadm join 192.168.39.79:6443 --token ynddd p.oamgloerxuune80q --discovery-token-ca-cert-hash sha256:7a45c40b5302aba7d8b9cbd3afc6d25c6bb8536dd6317aebcd2909b0427677c8
[preflight] Running pre-flight checks
[discovery] Trying to connect to API Server "192.168.39.79:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.39.79:6443"
[discovery] Requesting info from "https://192.168.39.79:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "192.168.39.79:6443"
[discovery] Successfully established connection with API Server "192.168.39.79:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "localhost.localdomain" as an annotation

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

效果验证

- 查看节点状态

```
kubectl get nodes
```

```
[root@k8s-master ~]# kubectl get nodes
NAME          STATUS    ROLES     AGE       VERSION
k8s-master    Ready     master    10m       v1.13.1
k8s-node-1    Ready     <none>    6m59s     v1.13.1
k8s-node-2    Ready     <none>    6m9s      v1.13.1
[root@k8s-master ~]#
```

- 查看所有 Pod状态

```
kubectl get pods --all-namespaces -o wide
```

```
[root@k8s-master ~]# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                 READY   STATUS    RESTARTS   AGE     IP               NODE
kube-system   coredns-86c58d9df4-4rds2             1/1     Running   0          9m20s   10.244.0.8       k8s-master
kube-system   coredns-86c58d9df4-rhtgq             1/1     Running   0          9m20s   10.244.0.9       k8s-master
kube-system   etcd-k8s-master                      1/1     Running   0          8m19s   192.168.199.79   k8s-master
kube-system   kube-apiserver-k8s-master            1/1     Running   0          8m39s   192.168.199.79   k8s-master
kube-system   kube-controller-manager-k8s-master   1/1     Running   0          8m43s   192.168.199.79   k8s-master
kube-system   kube-flannel-ds-amd64-8qzpx          1/1     Running   0          6m18s   192.168.199.77   k8s-node-1
kube-system   kube-flannel-ds-amd64-jvp59          1/1     Running   0          5m28s   192.168.199.78   k8s-node-2
kube-system   kube-flannel-ds-amd64-wztbk          1/1     Running   0          7m11s   192.168.199.79   k8s-master
kube-system   kube-proxy-crr7k                     1/1     Running   0          9m20s   192.168.199.79   k8s-master
kube-system   kube-proxy-gk5vf                     1/1     Running   0          6m18s   192.168.199.77   k8s-node-1
kube-system   kube-proxy-ktr27                     1/1     Running   0          5m28s   192.168.199.78   k8s-node-2
kube-system   kube-scheduler-k8s-master            1/1     Running   0          8m41s   192.168.199.79   k8s-master
[root@k8s-master ~]#
```

好了，集群现在已经正常运行了，接下来看看如何正常的拆卸集群。

拆卸集群

首先处理各节点：

```
kubectl drain <node name> --delete-local-data --force --ignore-
daemonsets
kubectl delete node <node name>
```

一旦节点移除之后，则可以执行如下命令来重置集群：

```
kubeadm reset
```

安装 DASHBOARD

就像给elasticsearch配一个可视化的管理工具一样，我们最好也给 k8s集群配一个可视化的管理工具，便于管理集群。

因此我们接下来安装 `v1.10.0` 版本的 kubernetes-dashboard，用于集群可视化的管理。

- 首先手动下载镜像并重新打标签：（所有节点）

```
docker pull registry.cn-qingdao.aliyuncs.com/wangxiaoke/kubernetes-dashboard-amd64:v1.10.0
docker tag registry.cn-qingdao.aliyuncs.com/wangxiaoke/kubernetes-dashboard-amd64:v1.10.0 k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0
docker image rm registry.cn-qingdao.aliyuncs.com/wangxiaoke/kubernetes-dashboard-amd64:v1.10.0
```

- 安装 dashboard：

```
kubectl create -f dashboard.yaml
```

`dashboard.yaml` 文件下载地址：

> 下载地址1（直接点击）
> 下载地址2（直接点击）

- 查看 dashboard的 pod是否正常启动，如果正常说明安装成功：

```
 kubectl get pods --namespace=kube-system
```

```
[root@k8s-master ~]# kubectl get pods --namespace=kube-system
NAME                                   READY   STATUS    RESTARTS
AGE
coredns-86c58d9df4-4rds2               1/1     Running   0
81m
coredns-86c58d9df4-rhtgq               1/1     Running   0
81m
etcd-k8s-master                        1/1     Running   0
80m
kube-apiserver-k8s-master              1/1     Running   0
80m
kube-controller-manager-k8s-master     1/1     Running   0
80m
kube-flannel-ds-amd64-8qzpx            1/1     Running   0
78m
kube-flannel-ds-amd64-jvp59            1/1     Running   0
77m
kube-flannel-ds-amd64-wztbk            1/1     Running   0
78m
```

```
kube-proxy-crr7k                          1/1     Running   0
81m
kube-proxy-gk5vf                          1/1     Running   0
78m
kube-proxy-ktr27                          1/1     Running   0
77m
kube-scheduler-k8s-master                 1/1     Running   0
80m
kubernetes-dashboard-79ff88449c-v2jnc     1/1     Running   0
21s
```

- 查看 dashboard的外网暴露端口

```
kubectl get service --namespace=kube-system
```

```
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)          AGE
kube-dns               ClusterIP   10.96.0.10      <none>
 53/UDP,53/TCP    5h38m
kubernetes-dashboard   NodePort    10.99.242.186   <none>
 443:31234/TCP    14
```

- 生成私钥和证书签名：

```
openssl genrsa -des3 -passout pass:x -out dashboard.pass.key 2048
openssl rsa -passin pass:x -in dashboard.pass.key -out dashboard.key
rm dashboard.pass.key
openssl req -new -key dashboard.key -out dashboard.csr【如遇输入，一路回车
即可】
```

- 生成SSL证书：

```
openssl x509 -req -sha256 -days 365 -in dashboard.csr -signkey
dashboard.key -out dashboard.crt
```

- 然后将生成的 `dashboard.key` 和 `dashboard.crt` 置于路径 `/home/share/certs` 下，该路径会配置到下面即将要操作的

`dashboard-user-role.yaml` 文件中

- 创建 dashboard用户

```
 kubectl create -f dashboard-user-role.yaml
```

`dashboard-user-role.yaml` 文件下载地址：

> 下载地址1（直接点击）
> 下载地址2（直接点击）

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

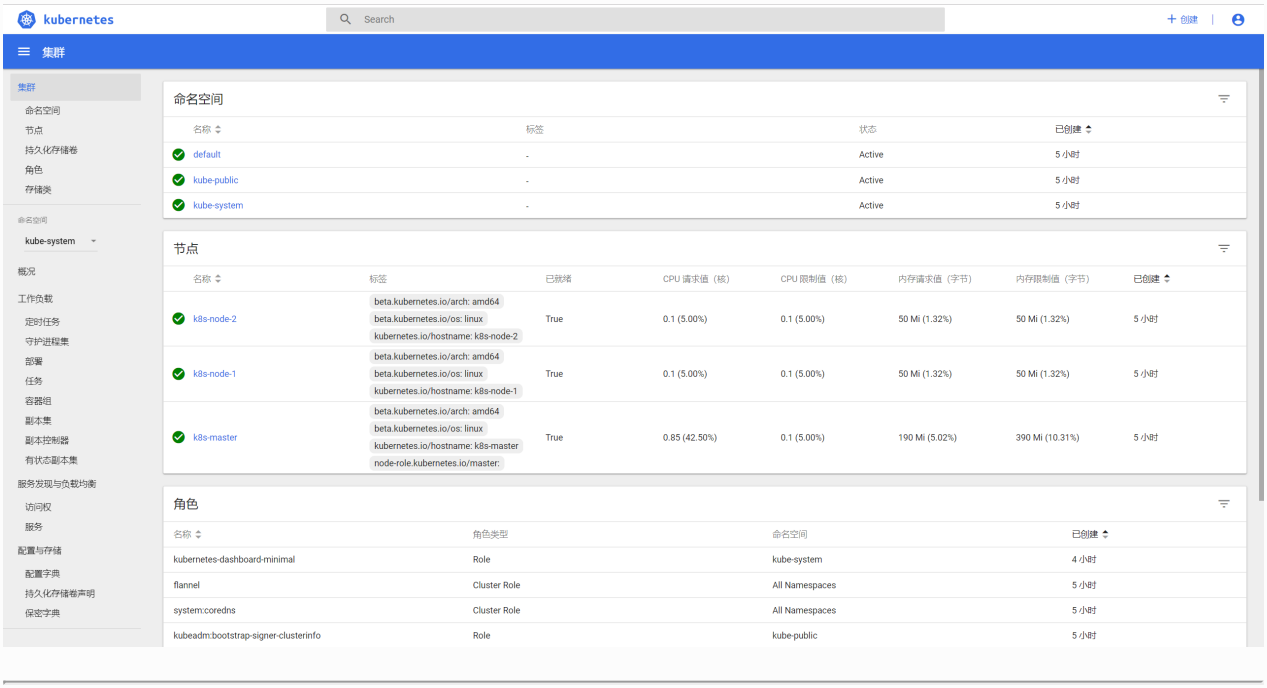
- 获取登陆token

```
kubectl describe secret/$(kubectl get secret -nkube-system |grep
admin|awk '{print $1}') -nkube-system
```

```
[root@k8s-master ~]# kubectl describe secret/$(kubectl get secret -
nkube-system |grep admin|awk '{print $1}') -nkube-system
Name:         admin-token-9d4vl
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: admin
              kubernetes.io/service-account.uid: a320b00f-07ed-11e9-
93f2-000c2978f207

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2
NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlL
XN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJh
ZG1pbi10b2tlbi05ZDR2bCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2Vydml
jZS1hY2NvdW50Lm5hbWUiOiJhZG1pbiIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bn
Qvc2VydmljZS1hY2NvdW50LnVpZCI6ImEzMjBiMDBmLTA3ZWQtMTFlOS05M2YyLTAwMGMyO
Tc4ZjIwNyIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTphZG1p
biJ9.WbaHx-
BfZEd0SvJwA9V_vGUe8jPMUHjKlkT7MWJ4JcQldRFY8Tdpv5GKCY25JsvT_GM3ob303r0yE
6vjQdKna7EfQNO_Wb2j1Yu5UvZnWw52HhNudHNOVL_fFRKxkSVjAILA_C_HvW6aw6TG5h7z
HARgl71I0LpW1VESeHeThipQ-pkt-Dr1jWcpPgE39cwxSgi-
5qY4ssbyYBc2aPYLsqJibmE-
KUhwmyOheF4Lxpg7E3SQEczsig2HjXpNtJizCu0kPyiR4qbbsusulH-
kdgjhmD9_XWP9k0BzgutXWteV8Iqe4-uuRGHZAxgutCvaL5qENv4OAlaArlZqSgkNWw
```

token既然生成成功，接下来就可以打开浏览器，输入 token来登录进集群管理页面：

CodeSheep



本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

ELASTICSEARCH集群部署

环境准备

- **节点准备**

本文准备搭建 **双节点** Elasticsearch集群（用双节点仅仅是做演示），因此这里准备了两台 **Linux CentOS 7.4 64bit** 机器：

> - 节点1：`192.168.31.8`
> - 节点2：`192.168.31.9`

- **Elasticsearch 安装包准备**

这里下载的是：`6.4.2` 版

```
wget
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-
6.4.2.tar.gz
```

- **安装目录准备**

这里拟将 Elasticsearch安装在 `/opt/elasticsearch` 目录下：

```
mkdir /opt/elasticsearch
将压缩包复制到该目录下并解压
```

ELASTICSEARCH 集群配置

需要修改两个节点上的配置文件 elasticsearch.yml

- **节点1 配置**

```
cluster.name: codesheep          # 集群名称
node.name: sheep1                  # 节点名
network.host: 192.168.31.8      # 绑定的节点1地址
network.bind_host: 0.0.0.0       # 此项不设置你试试本机可能访问不了啊
discovery.zen.ping.unicast.hosts: ["192.168.31.8","192.168.31.9"]  #
hosts列表
discovery.zen.minimum_master_nodes: 1

## 如下配置是为了解决 Elasticsearch可视化工具 dejavu的跨域问题！若不用可视化工具
则可省略之
http.port: 9200
http.cors.allow-origin: "http://192.168.199.76:1358"
http.cors.enabled: true
http.cors.allow-headers : X-Requested-With,X-Auth-Token,Content-
Type,Content-Length,Authorization
http.cors.allow-credentials: true
```

- **节点2 配置**

```
cluster.name: codesheep          # 集群名称
node.name: sheep1                  # 节点名
network.host: 192.168.31.9      # 绑定的节点2地址
network.bind_host: 0.0.0.0
discovery.zen.ping.unicast.hosts: ["192.168.31.8","192.168.31.9"]  #
hosts列表
discovery.zen.minimum_master_nodes: 1

## 如下配置是为了解决 Elasticsearch可视化工具 dejavu的跨域问题！若不用可视化工具
则可省略之
http.port: 9200
http.cors.allow-origin: "http://192.168.199.76:1358"
http.cors.enabled: true
http.cors.allow-headers : X-Requested-With,X-Auth-Token,Content-
Type,Content-Length,Authorization
http.cors.allow-credentials: true
```

集群启动前准备

- **创建用户及用户组**

由于 Elasticsearch不能以 root用户启动，因此需要添加非 root用户：

```
groupadd es
useradd es -g es
chown -R es:es ./elasticsearch-6.4.2
```

- **关闭防火墙**

```
systemctl stop firewalld
systemctl disable firewalld
```

启动 ELASTICSEARCH集群

- **切换用户**

```
su es
```

- **分别在 节点1和 节点2上启动ES服务**

```
cd bin
./elasticsearch   // 若要后台启动，则加-d参数
```

- **浏览器访问：http://ip:9200/ 查看启动效果**

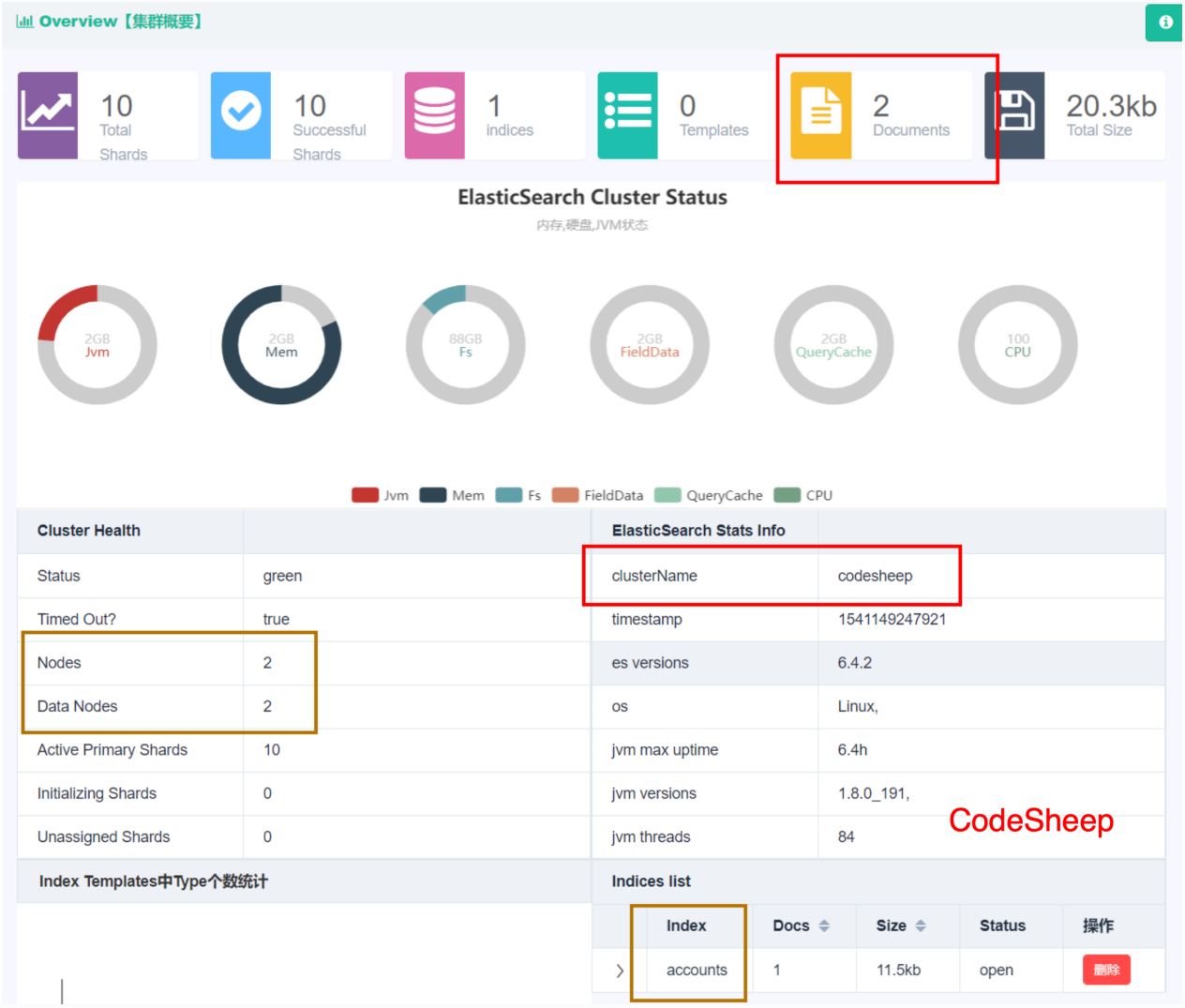
- **命令行查看集群信息**

```
[root@localhost ~]#
[root@localhost ~]# curl http://127.0.0.1:9200/_cat/allocation?v
shards disk.indices disk.used disk.avail disk.total disk.percent host          ip            node
     5        1.2kb     4.1gb     31.4gb     35.5gb           11 192.168.31.9 192.168.31.9 node2
     5        1.2kb     5.6gb     29.9gb     35.5gb           15 192.168.31.8 192.168.31.8 node1
[root@localhost ~]#
```

CodeSheep

- 利用可视化工具 dejavu查看集群信息

关于 Elasticsearch集群可视化管理工具的上手，可以参考我的前文：《一文上手 Elasticsearch常用可视化管理工具》



- 接下来插入两条数据

```
curl -X PUT 'localhost:9200/accounts/person/1' -d '
{
  "user": "张三",
  "title": "工程师",
  "desc": "数据库管理"
}'

curl -X PUT 'localhost:9200/accounts/person/1' -d '
{
  "user": "赵四",
  "title": "设计师",
  "desc": "UI设计"
}'
```

- **查看数据的入库效果**

```
- _shards: {
      failed: 0,
      skipped: 0,
      successful: 5,
      total: 5
  },
- hits: {
    - hits: [
        - {
            _id: "W0Gj02YBfRHOCCe7qdQH",
            _index: "accounts",
            _score: 1,
          - _source: {
                desc: "UI设计",
                title: "设计师",
                user: "赵四"
            },
            _type: "person"
        },
        - {
            _id: "1",
            _index: "accounts",
            _score: 1,
          - _source: {
                desc: "数据库管理",
                title: "工程师",
                user: "张三"
            },
            _type: "person"
        }
    ],
    max_score: 1,
    total: 2
  },
  timed_out: false,
  took: 4
}
```


OK，**索引 / 类型 / 文档** 一目了然！

> 若在 Elasticsearch集群 **安装/启动** 过程 中有任何奇葩 **问题/错误** 的话，可以参考这篇文章：《CentOS7上ElasticSearch安装填坑记》吧，里面记录了一些踩过的坑！

安装IK分词器

> 在 Elasticsearch的世界中，插件是很重要的一部分，很多功能都可以通过插件来实现，因此下面就以常用的 **IK分词器插件** 的安装为例，来操作一下 Elasticsearch插件的安装

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

分词技术是搜索技术的基石，而 IK分词器是比较经典的一个，接下来尝试安装一下吧

IK分词器版本与 ES版本对应，不能搞错，可在 这里查看

- **下载 IK分词器插件**

```
wget https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v6.4.2/elasticsearch-analysis-ik-6.4.2.zip
```

- **解压 / 安装**

新建目录 `/opt/elasticsearch/elasticsearch-6.4.2/plugins/elasticsearch-analysis-ik-6.4.2`

再将 zip包置于上述目录下并解压：

```
unzip elasticsearch-analysis-ik-6.4.2.zip
```

- **重启 Elasticsearch集群**

重启 Elasticsearch集群，若发现如下内容，这说明插件安装成功：

```
15,645][INFO ][o.e.p.PluginsService        ] [sheep1] loaded module [x-pack-security]
15,645][INFO ][o.e.p.PluginsService        ] [sheep1] loaded module [x-pack-sql]
15,646][INFO ][o.e.p.PluginsService        ] [sheep1] loaded module [x-pack-upgrade]
15,646][INFO ][o.e.p.PluginsService        ] [sheep1] loaded module [x-pack-watcher]
15,646][INFO ][o.e.p.PluginsService        ] [sheep1] loaded plugin [analysis-ik]
18,516][INFO ][o.e.x.s.a.s.FileRolesStore] [sheep1] parsed [0] roles from file [/opt/elasticsearch/e
19,064][INFO ][o.e.x.m.j.p.l.CppLogMessageHandler] [controller/11013] [Main.cc@109] controller (64 b
19,511][DEBUG][o.e.a.ActionModule         ] Using REST wrapper from plugin org.elasticsearch.xpack.sec
19,752][INFO ][o.e.d.DiscoveryModule      ] [sheep1] using discovery type [zen]
20,447][INFO ][o.e.n.Node                 ] [sheep1] initialized
20,447][INFO ][o.e.n.Node                 ] [sheep1] starting ...
20,577][INFO ][o.e.t.TransportService     ] [sheep1] publish_address {192.168.199.75:9300}, bound_addr
```

怎么样，很简单吧，说到底就是一个解压放置的过程而已。

ZOOKEEPER安装部署

准备安装包

这里使用的是 `3.6.1` 版，下载的是 `apache-zookeeper-3.6.1-bin.tar.gz` 压缩包，并将其放在了 `/root` 目录下

解压并安装

在 `/usr/local/` 下创建 `zookeeper` 文件夹并进入

```
cd /usr/local/
mkdir zookeeper
cd zookeeper
```

2、将 `ZooKeeper` 安装包解压到 `/usr/local/zookeeper` 中即可

```
[root@localhost zookeeper]# tar -zxvf /root/apache-zookeeper-3.6.1-
bin.tar.gz -C ./
```

解压完之后， `/usr/local/zookeerper` 目录中会出现一个 `apache-zookeeper-3.6.1-bin` 的目录

创建DATA目录

这里直接在 `/usr/local/zookeeper/apache-zookeeper-3.6.1-bin` 目录中创建一个 `data` 目录

```
codesheep  4096 5月   18 12:02 bin
codesheep  2692 4月   21 22:59 checkstyle-simple.xml
codesheep 17804 4月   21 22:59 checkstyle-strict.xml
codesheep  1538 4月   21 22:59 checkstyleSuppressions.xml
codesheep    77 5月   18 12:06 conf
root          6 5月   18 12:18 data
codesheep    20 4月   21 22:59 dev
codesheep   347 4月   21 22:59 excludeFindBugsFilter.xml
codesheep 11358 4月   21 22:59 LICENSE.txt
codesheep   432 4月   21 22:59 NOTICE.txt
codesheep  1795 4月   21 22:59 owaspSuppressions.xml
codesheep 33831 4月   21 22:59 pom.xml
codesheep  1963 4月   21 22:59 README.md
codesheep  3166 4月   21 22:59 README_packaging.md
codesheep 21474 4月   21 22:59 zk-merge-pr.py
codesheep    32 5月   18 12:02 zookeeper-assembly
codesheep    47 5月   18 12:02 zookeeper-client
codesheep  4096 5月   18 12:02 zookeeper-contrib
codesheep    32 5月   18 12:02 zookeeper-docs
codesheep    50 5月   18 12:02 zookeeper-it
codesheep    32 5月   18 12:02 zookeeper-jute
codesheep    57 5月   18 12:02 zookeeper-metrics-providers
codesheep   176 5月   18 12:02 zookeeper-recipes
codesheep    32 5月   18 12:02 zookeeper-server
```

等下该 `data` 目录地址要配到 `zooKeeper` 的配置文件中：

创建配置文件并修改

进入到 `zookeeper` 的 `conf` 目录，复制 `zoo_sample.cfg` 得到 `zoo.cfg`：

```
[root@localhost apache-zookeeper-3.6.1-bin]# cd conf/
[root@localhost conf]# cp zoo_sample.cfg zoo.cfg
```

修改配置文件 `zoo.cfg`，将其中的 `dataDir` 修改为上面刚创建的 `data` 目录，其他选项可以按需配置

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/usr/local/zookeeper/apache-zookeeper-3.6.1-bin/data
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
```


启动ZOOKEEPER

```
[root@localhost apache-zookeeper-3.6.1-bin]# ./bin/zkServer.sh start
```

```
[root@localhost apache-zookeeper-3.6.1-bin]# ./bin/zkServer.sh start
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/apache-zookeeper-3.6.1-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost apache-zookeeper-3.6.1-bin]#
[root@localhost apache-zookeeper-3.6.1-bin]#
```

启动后可以通过如下命令来检查启动后的状态：

```
[root@localhost apache-zookeeper-3.6.1-bin]# ./bin/zkServer.sh status
```

```
[root@localhost apache-zookeeper-3.6.1-bin]# ./bin/zkServer.sh status
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/apache-zookeeper-3.6.1-bin/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost.
Mode: standalone
[root@localhost apache-zookeeper-3.6.1-bin]#
[root@localhost apache-zookeeper-3.6.1-bin]#
```

从图中也可以看出zookeeper默认会绑定端口 `2181` 。

配置环境变量

编辑配置文件：

```
vim /etc/profile
```

尾部加入 `ZooKeeper` 的 `bin` 路径配置即可

```
export ZOOKEEPER_HOME=/usr/local/zookeeper/apache-zookeeper-3.6.1-bin
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

最后执行 `source /etc/profile` 使环境变量生效即可。

设置开机自启

首先进入 `/etc/rc.d/init.d` 目录，创建一个名为 `zookeeper` 的文件，并赋予执行权限

```
[root@localhost ~]# cd /etc/rc.d/init.d/
[root@localhost init.d]# touch zookeeper
[root@localhost init.d]# chmod +x zookeeper
```

接下来编辑 `zookeeper` 文件，并在其中加入如下内容：

```
#!/bin/bash
#chkconfig:- 20 90
#description:zookeeper
#processname:zookeeper
ZOOKEEPER_HOME=/usr/local/zookeeper/apache-zookeeper-3.6.1-bin
export JAVA_HOME=/usr/local/java/jdk1.8.0_161 # 此处根据你的实际情况更换对
应
case $1 in
        start) su root $ZOOKEEPER_HOME/bin/zkServer.sh start;;
        stop) su root $ZOOKEEPER_HOME/bin/zkServer.sh stop;;
        status) su root $ZOOKEEPER_HOME/bin/zkServer.sh status;;
        restart) su root $ZOOKEEPER_HOME/bin/zkServer.sh restart;;
        *) echo "require start|stop|status|restart" ;;

esac
```

最后加入开机启动即可：

```
chkconfig --add zookeeper
chkconfig zookeeper on
```

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

消息队列KAFKA安装部署

首先准备ZOOKEEPER服务

因为 `Kafka` 的运行环境依赖于 `ZooKeeper` ，所以首先得安装并运行 `ZooKeeper` 。

这个可以参考上面一节的内容。

准备KAFKA安装包

这里下载的是 `2.5.0` 版： `kafka_2.12-2.5.0.tgz` ，将下载后的安装包放在了 `/root` 目录下

解压并安装

在 `/usr/local/` 下创建 `kafka` 文件夹并进入

```
cd /usr/local/
mkdir kafka
cd kafka
```

2、将Kafka安装包解压到 `/usr/local/kafka` 中即可

```
[root@localhost kafka]# tar -zxvf /root/kafka_2.12-2.5.0.tgz -C ./
```

解压完之后， `/usr/local/kafka` 目录中会出现一个 `kafka_2.12-2.5.0` 的目录

创建LOGS目录

这里直接在 `/usr/local/kafka/kafka_2.12-2.5.0` 目录中创建一个 `logs` 目录

等下该 `logs` 目录地址要配到Kafka的配置文件中。

修改配置文件

进入到 `Kafka` 的 `config` 目录，编辑配置文件 `server.properties`

```
[root@localhost kafka_2.12-2.5.0]# cd config/
[root@localhost config]# vim server.properties
```

修改配置文件，一是将其中的 `log.dirs` 修改为上面刚创建的 `logs` 目录，其他选项可以按需配置

```
num.network.threads=3

# The number of threads that the server uses for processing r
e disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will a
 OOM)
socket.request.max.bytes=104857600


############################# Log Basics ####################

# A comma separated list of directories under which to store
log.dirs=/usr/local/kafka/kafka_2.12-2.5.0/logs

# The default number of log partitions per topic. More partit
# parallelism for consumption, but this will also result in m
# the brokers.
num.partitions=1
```


另外关注一下连接 `ZooKeeper` 的相关配置，根据实际情况进行配置：

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

```
############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000


############################# Group Coordinator Settings #############################
@
                                                                    130,0-1
```

启动KAFKA

执行如下命令即可：

```
./bin/kafka-server-start.sh ./config/server.properties
```

```
[2020-05-18 17:04:07,313] INFO [ProducerId Manager 0]: Acquired new producerId block (brokerId:0,blockStartProduce
ing to Zk with path version 1 (kafka.coordinator.transaction.ProducerIdManager)
[2020-05-18 17:04:07,341] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.Transacti
[2020-05-18 17:04:07,342] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.Tran
[2020-05-18 17:04:07,353] INFO [Transaction Marker Channel Manager 0]: Starting (kafka.coordinator.transaction.Tra
[2020-05-18 17:04:07,382] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$E
[2020-05-18 17:04:07,424] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotifica
)
[2020-05-18 17:04:07,489] INFO [SocketServer brokerId=0] Started data-plane processors for 1 acceptors (kafka.netw
[2020-05-18 17:04:07,520] INFO Kafka version: 2.5.0 (org.apache.kafka.common.utils.AppInfoParser)
[2020-05-18 17:04:07,520] INFO Kafka commitId: 66563e712b0b9f84 (org.apache.kafka.common.utils.AppInfoParser)
[2020-05-18 17:04:07,520] INFO Kafka startTimeMs: 1589792647489 (org.apache.kafka.common.utils.AppInfoParser)
[2020-05-18 17:04:07,521] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
```

> 如果需要后台启动，则加上`-daemon`参数即可

实验验证

首先创建一个名为`codesheep`的`topic`：

```
./bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --
replication-factor 1 --partitions 1 --topic codesheep
```

```
[root@localhost kafka_2.12-2.5.0]#
[root@localhost kafka_2.12-2.5.0]# ./bin/kafka-topics.sh --create
 --bootstrap-server localhost:9092 --replication-factor 1 --parti
tions 1 --topic codesheep
Created topic codesheep.
[root@localhost kafka_2.12-2.5.0]#
```

创建成功

创建完成以后，可以使用命令来列出目前已有的 `topic` 列表

```
./bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

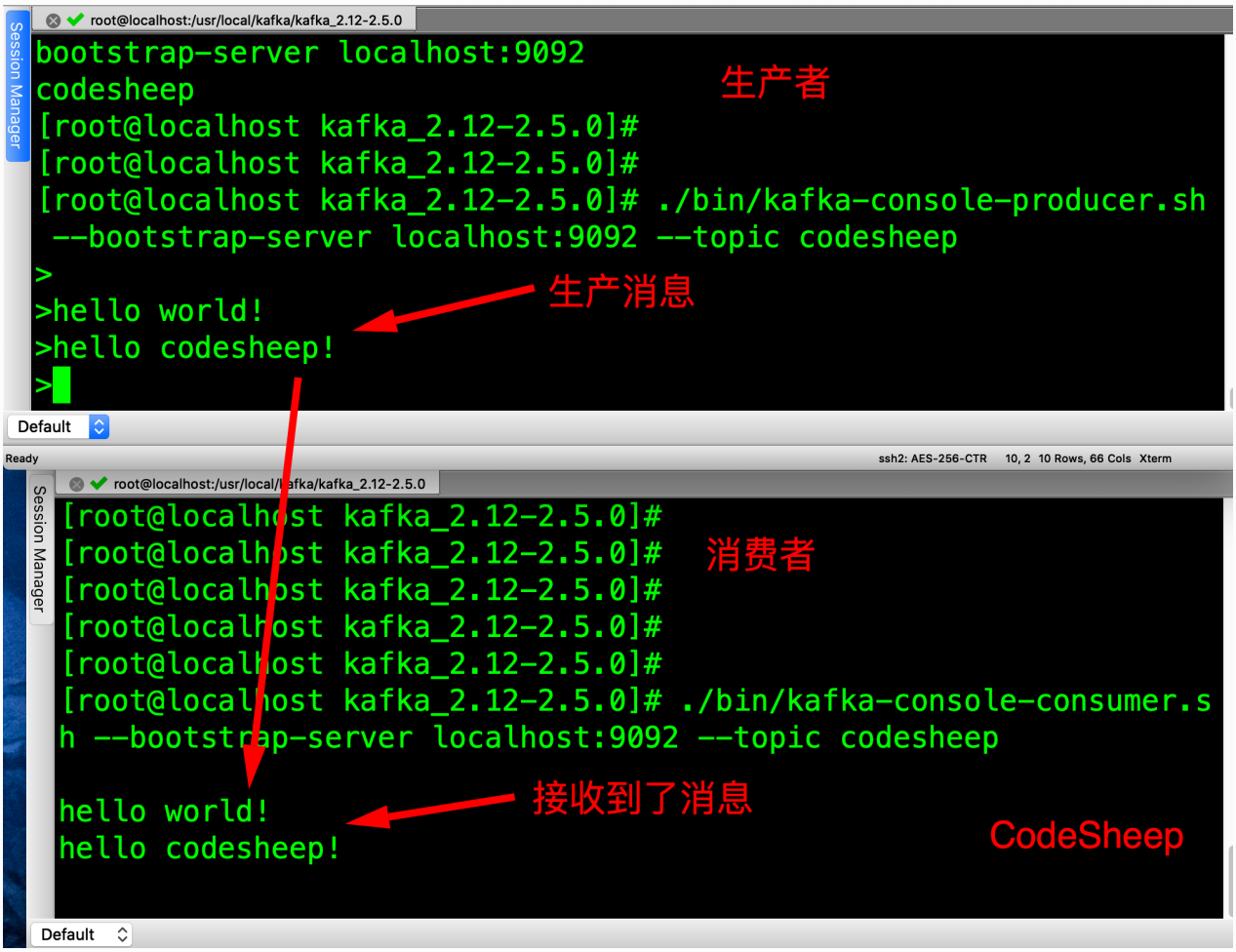
接下来创建一个**生产者**，用于在 `codesheep` 这个 `topic` 上生产消息：

```
./bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --
topic codesheep
```

而后接着创建一个**消费者**，用于在 `codesheep` 这个 `topic` 上获取消息：

```
./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --
topic codesheep
```

此时生产者发出的消息，在消费者端可以获取到：



本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

注意事项

实验过程中如果出现一些诸如客户端不能连通或访问等问题，可尝试考虑关闭防火墙：

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

持续更新中...

该PDF文档会持续更新，有新东西再往里加，有些东西也可以再继续优化。由于个人精力和能力有限，难免会有疏漏和不当的地方，还希望多多谅解。

另外， 联系作者、提意见，可直接扫码联系，一起交流进步：

> 本文档在 Github开源项目：github.com/hansonwang99/JavaCollection 中已收录，有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

本文档在 Github开源项目：https://github.com/hansonwang99/JavaCollection 中已收录，
有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...