

[前言](#)

[实验流程](#)

[环境交代](#)

[代码下载](#)

[修改配置](#)

[修改数据库配置](#)

[1、建库](#)

[2、建表](#)

[3、修改yml中数据库连接配置](#)

[修改缓存配置](#)

[修改日志配置](#)

[修改日志路径](#)

[修改字符集](#)

[本地运行](#)

[项目打包](#)

[前端项目打包](#)

[1、安装依赖](#)

[2、构建打包](#)

[后端项目打包](#)

[打jar包](#)

[打war包](#)

[前端项目部署](#)

[后端项目部署](#)

[jar包部署方式](#)

[war包部署方式](#)

[配置Nginx代理和转发](#)

[后台项目多实例部署](#)

[后记](#)

---

# 前言

关于「如何将一个前后端分离的开源项目部署到自己的服务器上」这个话题很早之前我就想做了。

一来是因为有很多初学的小伙伴们反馈说需要；而且经常也听到一部分小伙伴反馈说自己的云服务器买了不知道干啥，有什么办法可以玩起来，等等之类的话题。

所以今天这个坑必须给填上了，我这里又写了一个「**服务器项目部署PDF示例文档**」，希望对小伙伴们有用。

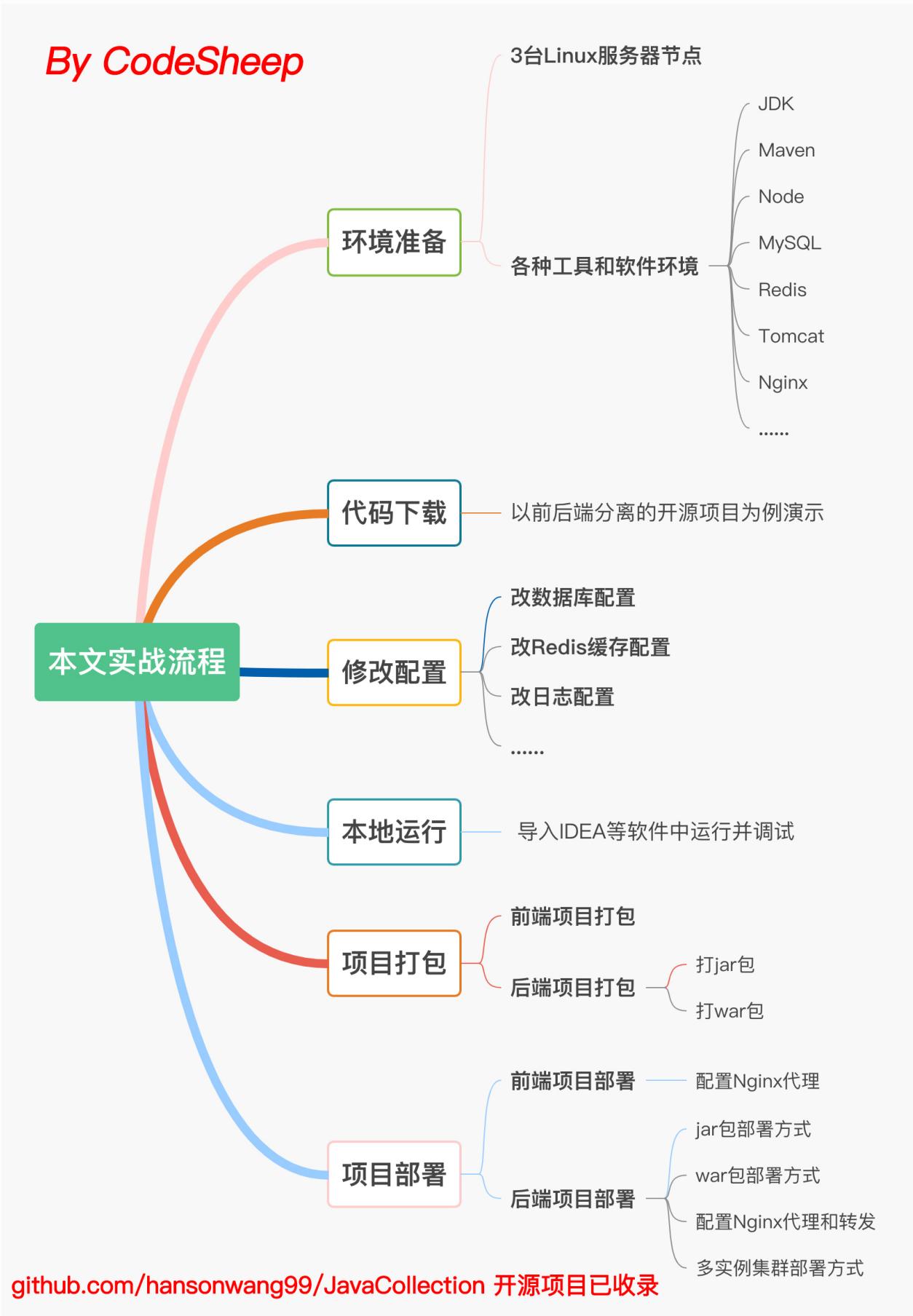
注：本文档在 Github开源项目：[github.com/hansonwang99/JavaCollection](https://github.com/hansonwang99/JavaCollection) 中已收录，有详细自学编程学习路线、面试题和面经、编程资料及系列技术文章等，资源持续更新中...

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。

---

# 实验流程

我还是先把本文的实验流程写在前面，并且画了个思维导图，这样好知道后面每一步在干嘛。



然后我们接下来一项项安排即可。

# 环境交代

部署离不开服务器节点、Linux系统和一系列软件环境以及基础设施，这个在之前就已经出过文章、文档以及视频聊过了，具体可以参看该系列文章：

- 服务器软件大科普
- 人手一套Linux环境搭建之：macOS版本教程
- 人手一套Linux环境搭建之：Windows版本教程
- 熬10天夜，肝出了这个PDF版「软件安装手册」

这些是本文实战的前提。

本实验投入了三台Linux主机节点，安排如下：

名称	IP地址	包含软件	用途
节点1	192.168.31.100	Nginx、Node、Git、MySQL、Redis	部署前端项目
节点2	192.168.31.101	Git、JDK、Maven、Tomcat	部署后端项目实例1
节点3	192.168.31.102	Git、JDK、Maven、Tomcat	部署后端项目实例2

顺便详细交代一下本文所用工具和软件的版本情况：

名称	版本	用途
Linux	CentOS 7.4 64bit	服务器操作系统
Git	1.8.3.1	代码下载
JDK	1.8.0_161	Java运行环境
Node	v12.16.3	
npm	6.14.4	前端项目构建和打包
Maven	3.6.3	后端项目构建和打包
MySQL	5.7.30	数据库
Redis	5.0.8	缓存
Nginx	1.17.10	Web服务器
Tomcat	8.5.55	应用服务器

# 代码下载

本文实验采用优秀开源项目Ruoyi的前后端分离版作为实验载体，我们将它完整的部署起来。

项目地址：[https://gitee.com/y\\_project/RuoYi-Vue](https://gitee.com/y_project/RuoYi-Vue)。

直接用 `Git` 命令下载到本地某个目录下即可。

```
git clone https://gitee.com/y_project/RuoYi-Vue
```

下载完成的代码目录里包含两个子目录，分别为**前端项目目录**和**后端项目目录**。

我这里在本地宿主机和三个 `Linux` 节点上都各下载了一份项目源码，路径分别为：

- 本地代码： `/Users/codesheep/IdeaProjects/RuoYi-Vue`
- 节点1代码路径： `/root/workspace/RuoYi-Vue`
- 节点2代码路径： `/root/workspace/RuoYi-Vue`
- 节点3代码路径： `/root/workspace/RuoYi-Vue`

本地代码是为了稍后导入 `IDEA` 中方便运行、调试和代码阅读及修改，`Linux` 系统节点上的三份代码待会直接打包部署使用。

---

# 修改配置

一个开源项目从网上下载之后，想直接立马就能完美运行起来不太现实，毕竟还有很多东西要根据自己的实际情况进行配置，比如数据库、缓存、日志路径等等，都需要根据自己实际情况修改。

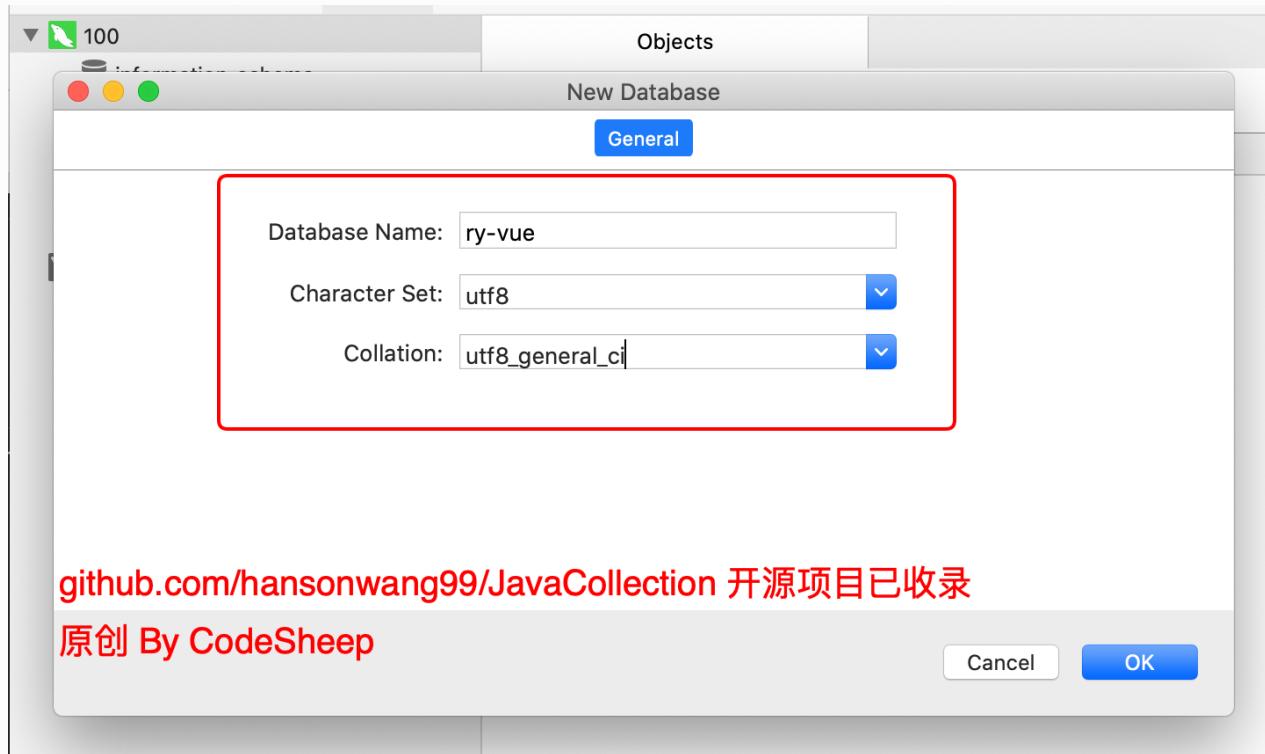
所以我们接下来一一配置。

---

## 修改数据库配置

### 1、建库

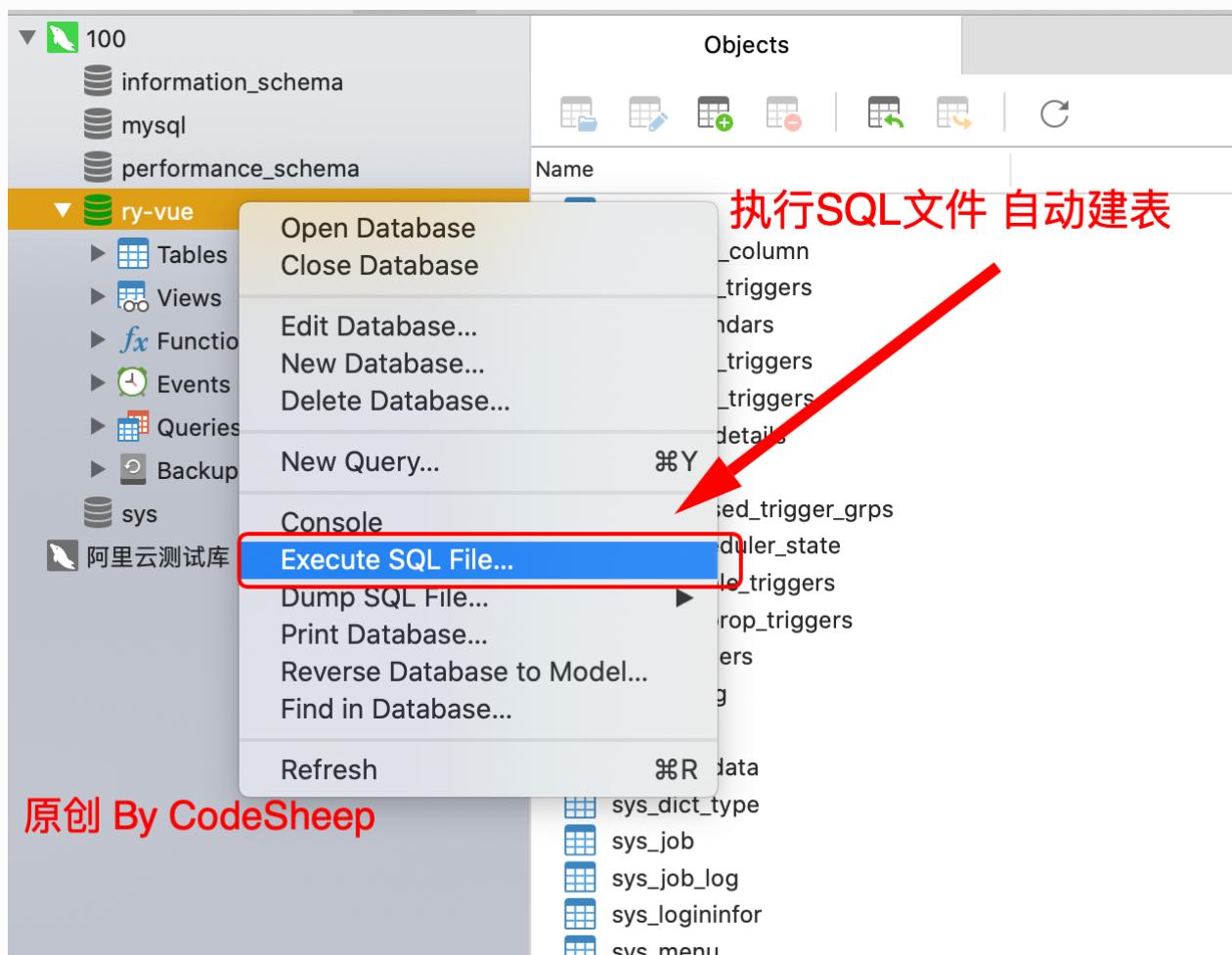
首先我们需要新建一个名为 `ry-vue` 的数据库，出于演示考虑，这里直接通过软件Navicat可视化创建：



### 2、建表

在上述新建的 `ry-vue` 的数据库里导入位于路径 `/Users/codesheep/IdeaProjects/RuoYi-Vue/ruoyi/sql` 下的两个 `.sql` 文件：

- `ry_20200415.sql`
- `quartz.sql`



原创 By CodeSheep

### 3、修改yml中数据库连接配置

修改路径 `/Users/codesheep/IdeaProjects/RuoYi-Vue/ruoyi/src/main/resources` 下的 `application-druid.yml` 配置文件，修改其中的数据库 `url`、`username`、`password` 等字段为你的实际的数据库链接：

```

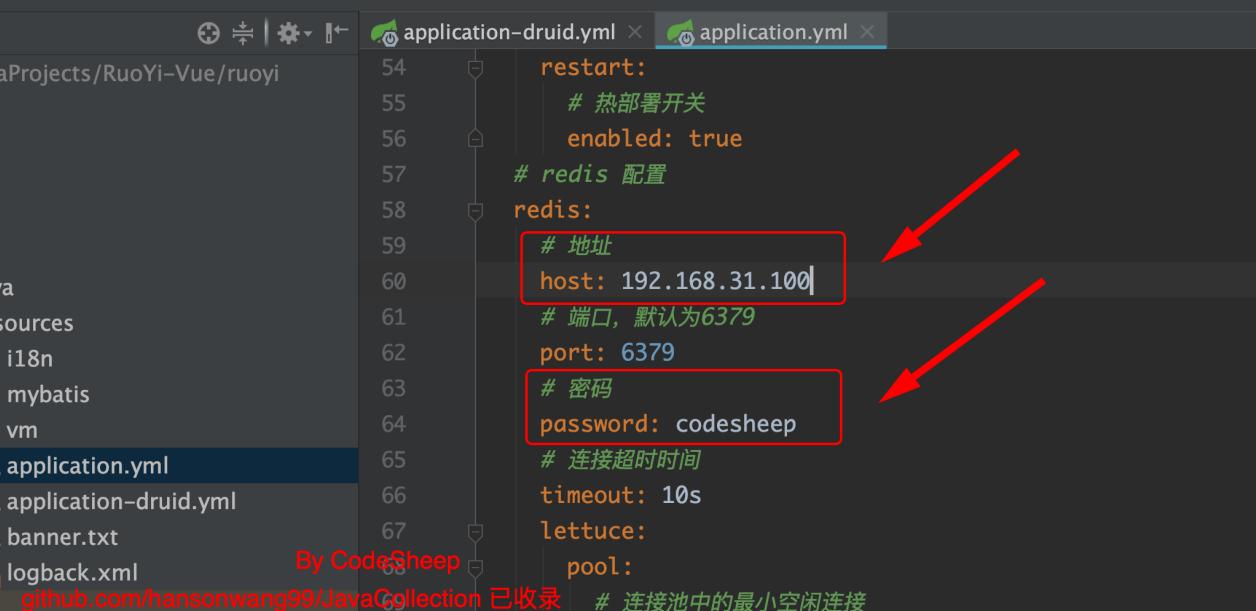
# 数据源配置
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    druid:
      # 主库数据源
      master:
        url: jdbc:mysql://192.168.31.100:3306/ry-vue?useUnicode=true&characterSet=utf8
        username: root
        password: 111111
      # 从库数据源
      slave:
        # 从数据源开关/默认关闭
        enabled: false
        url:
        username:
        password:

```

# 修改缓存配置

由于该项目使用了 Redis 缓存，因此和上面的 MySQL 数据库一样需要根据实际情况配置。

修改路径 /Users/codesheep/IdeaProjects/RuoYi-Vue/ruoyi/src/main/resources 下的 application.yml 配置文件，修改其中Redis缓存的 host、password 等字段为你实际的Redis链接：

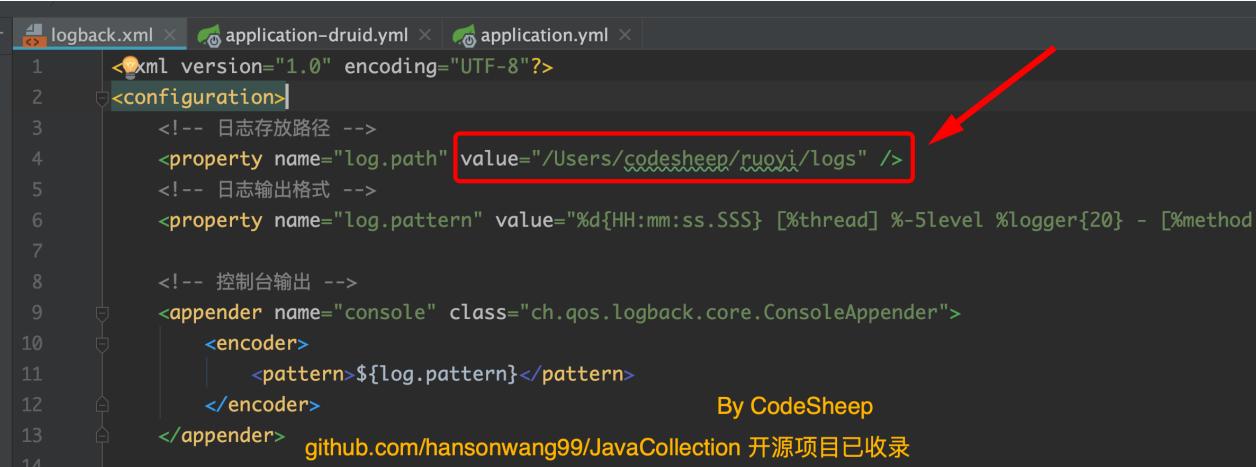


```
application-druid.yml x application.yml x
54      restart:
55          # 热部署开关
56          enabled: true
57      # redis 配置
58      redis:
59          # 地址
60          host: 192.168.31.100
61          # 端口, 默认为6379
62          port: 6379
63          # 密码
64          password: codesheep
65          # 连接超时时间
66          timeout: 10s
67      lettuce:
68          pool:
By CodeSheep
github.com/hansonwang99/JavaCollection 已收录 # 连接池中的最小空闲连接
```

# 修改日志配置

## 修改日志路径

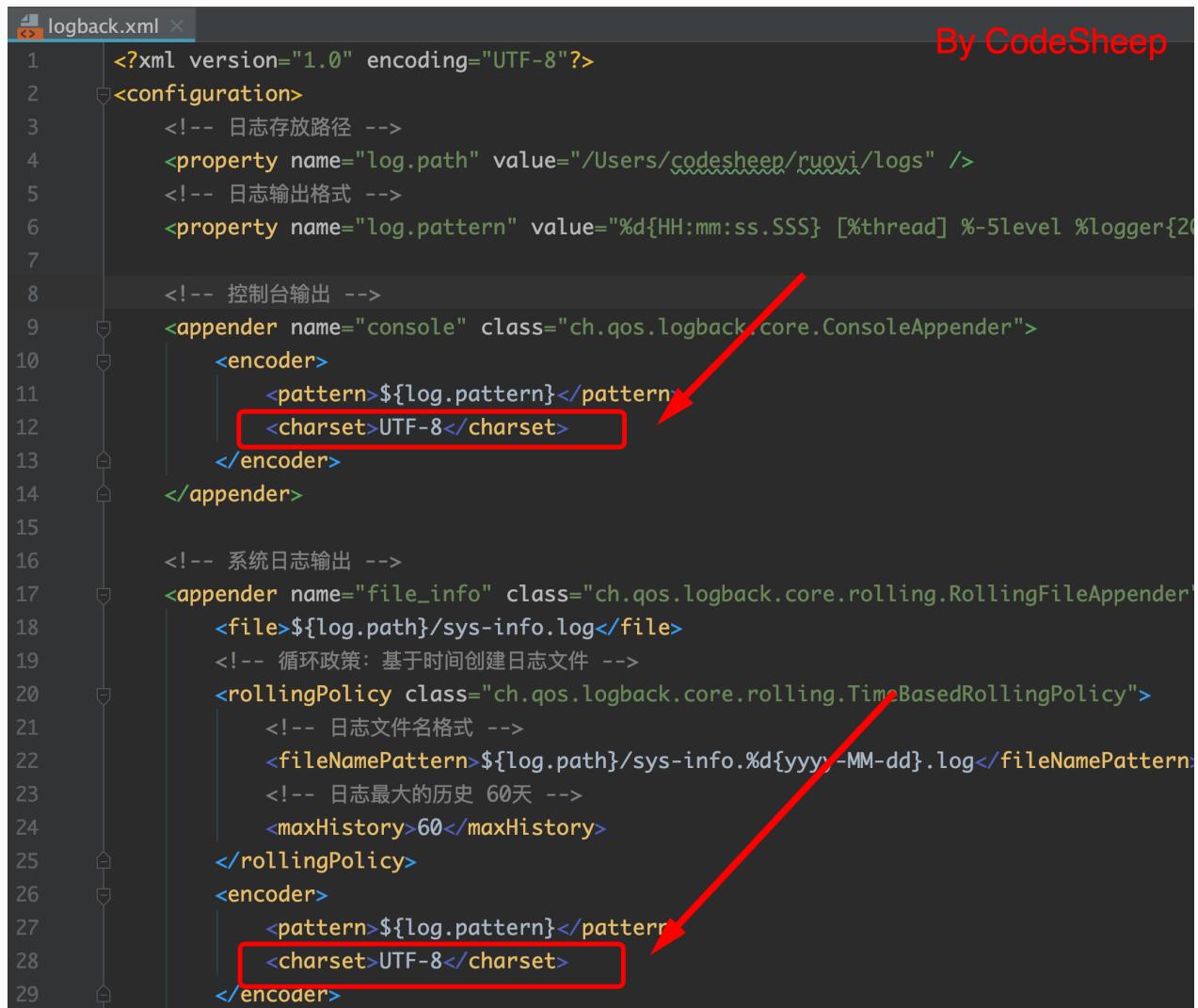
修改路径 /Users/codesheep/IdeaProjects/RuoYi-Vue/ruoyi/src/main/resources 下的 logback.xml 配置文件，首先修改其中的日志存放路径为自己定义的路径：



```
logback.xml x application-druid.yml x application.yml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration>
3     <!-- 日志存放路径 -->
4     <property name="log.path" value="/Users/codesheep/ruoyi/logs" />
5     <!-- 日志输出格式 -->
6     <property name="log.pattern" value="%d{HH:mm:ss.SSS} [%thread] %-5level %logger{20} - [%method,
7
8         <!-- 控制台输出 -->
9         <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
10            <encoder>
11                <pattern>${log.pattern}</pattern>
12            </encoder>
13        </appender>
By CodeSheep
github.com/hansonwang99/JavaCollection 开源项目已收录
14
```

# 修改字符集

另外则是在每一对 `<encoder></encoder>` 中加上 `UTF-8的` 字符集支持，避免后续服务器上看日志时可能出现乱码的现象：

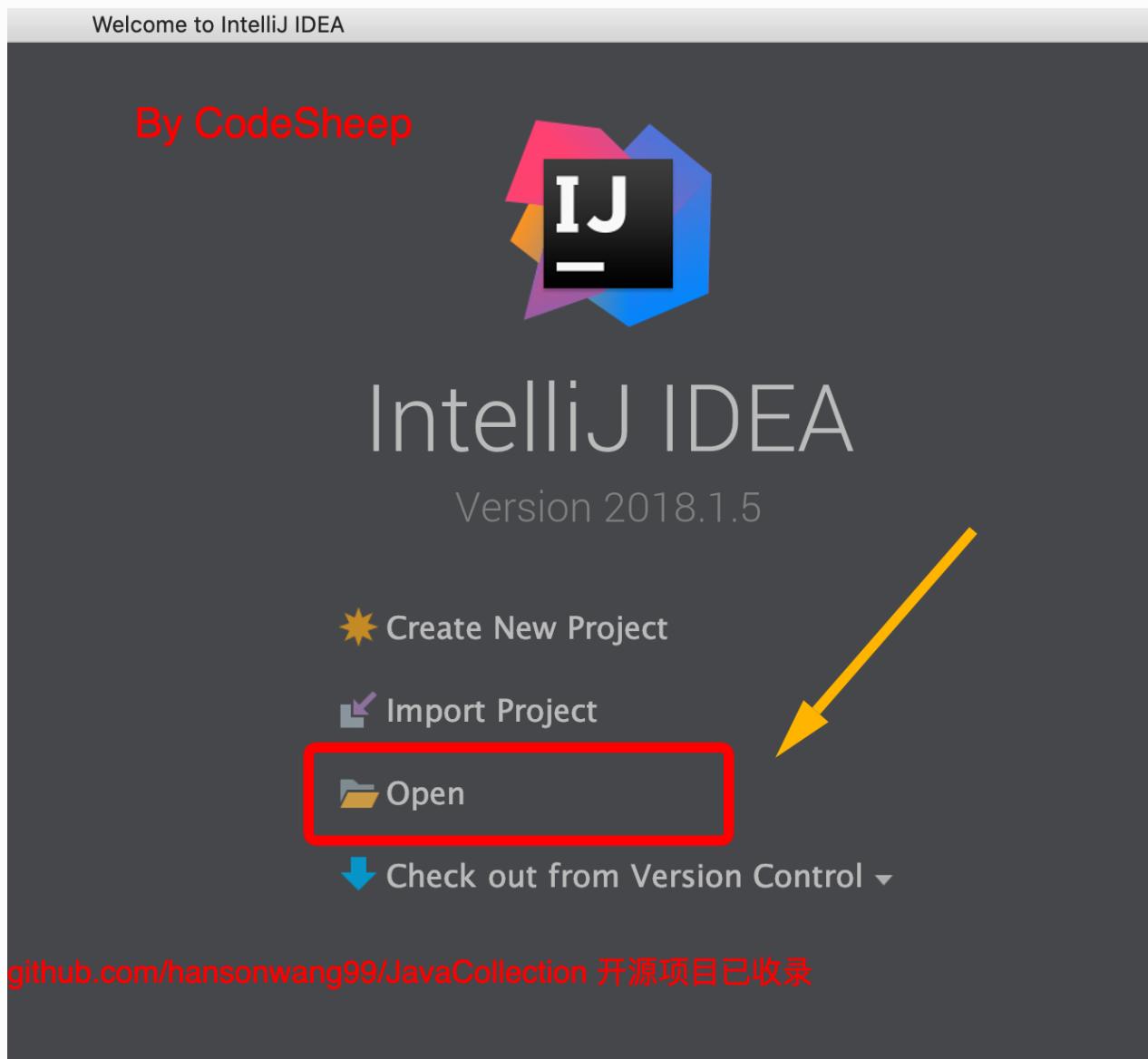


```
logback.xml x By CodeSheep
1  <?xml version="1.0" encoding="UTF-8"?>
2  <configuration>
3      <!-- 日志存放路径 -->
4      <property name="log.path" value="/Users/codesheep/ruoyi/logs" />
5      <!-- 日志输出格式 -->
6      <property name="log.pattern" value="%d{HH:mm:ss.SSS} [%thread] %-5level %logger{20} - %msg%n" />
7
8      <!-- 控制台输出 -->
9      <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
10         <encoder>
11             <pattern>${log.pattern}</pattern>
12             < charset >UTF-8</ charset >
13         </encoder>
14     </appender>
15
16     <!-- 系统日志输出 -->
17     <appender name="file_info" class="ch.qos.logback.core.rolling.RollingFileAppender">
18         <file>${log.path}/sys-info.log</file>
19         <!-- 循环政策：基于时间创建日志文件 -->
20         <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
21             <!-- 日志文件名格式 -->
22             <fileNamePattern>${log.path}/sys-info.%d{yyy-MM-dd}.log</fileNamePattern>
23             <!-- 日志最大的历史 60天 -->
24             <maxHistory>60</maxHistory>
25         </rollingPolicy>
26         <encoder>
27             <pattern>${log.pattern}</pattern>
28             < charset >UTF-8</ charset >
29         </encoder>

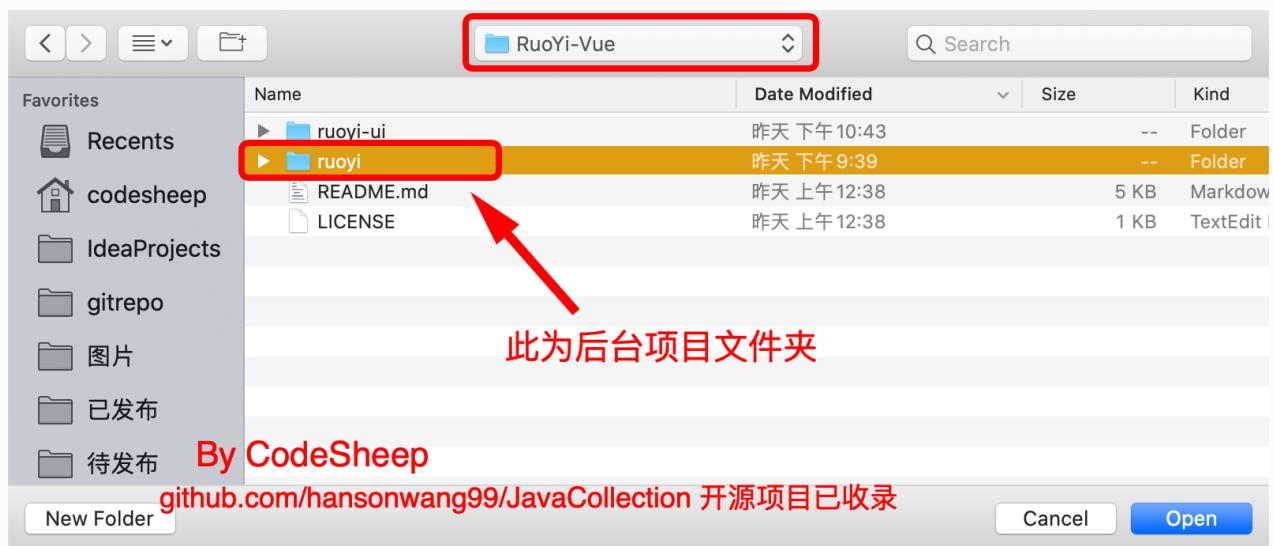
```

# 本地运行

打开 IntelliJ IDEA , 直接打开本地代码目录 Ruoyi-Vue 中的 ruoyi 文件夹即可:

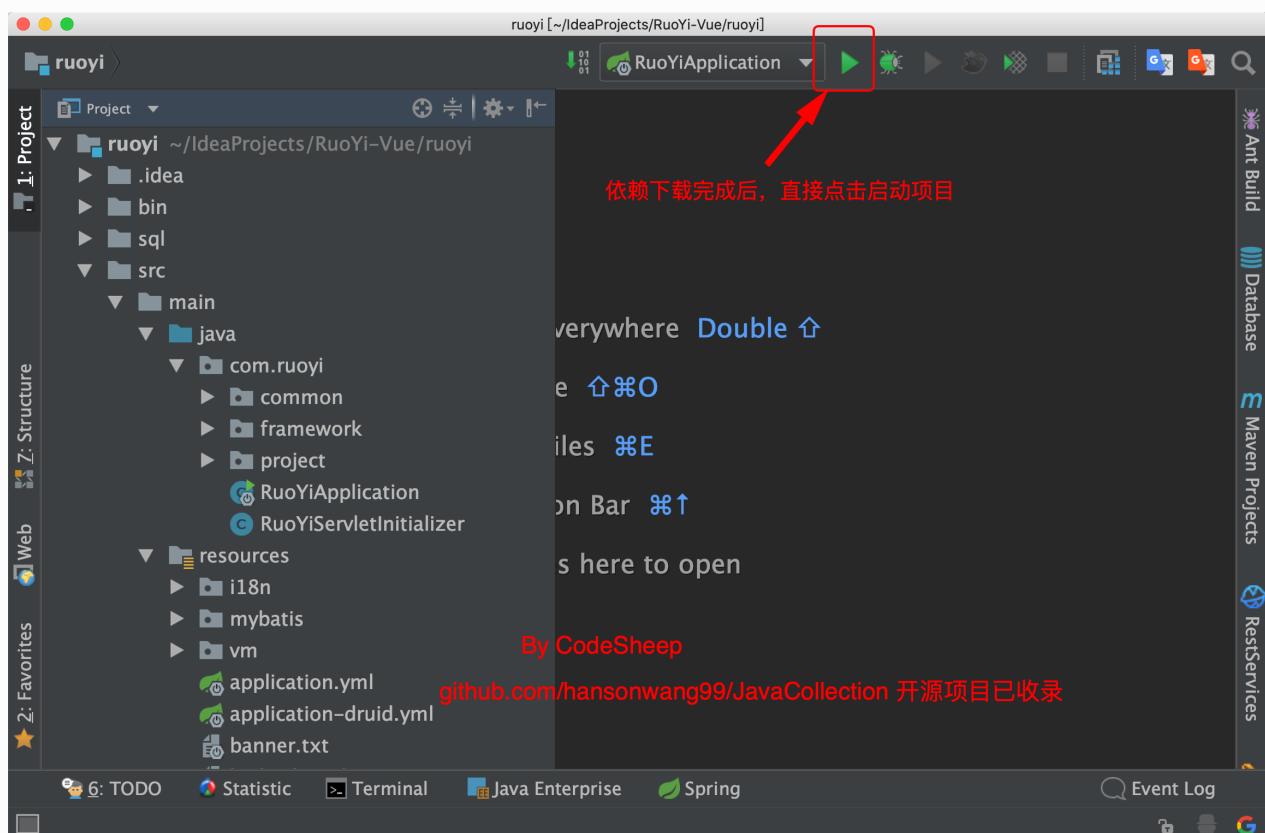


github.com/hansonwang99/JavaCollection 开源项目已收录



第一次导入，会自动下载项目 `pom.xml` 中各种所需的依赖，所以需要等待一段时间。

导入成功后，直接点击启动按钮，查看启动效果：



The screenshot shows the IntelliJ IDEA interface with the 'ruoyi' project open. The 'Run' tool window is active, displaying the 'Console' tab. The output window shows the application's startup logs:

```
23:18:25.441 [main] INFO o.a.t.u.n.NIOSelectorPool - [log,175] - Using a shared selector for servlet write/read
23:18:25.456 [main] INFO c.r.RuoYiApplication - [logStarted,59] - Started RuoYiApplication in 5.998 seconds
(JVM running for 6.747)
(♥～～)／ 若依启动成功  q(‘_,`q)
-----.
| _ _ \ \ \ / /
| ( ' ) | \ _ . / ,
|(_ o _) / _ ( )_ ..
| (.,_.)' __ __(_ o _)'
| \ \ \ | | | |(_.,_.)'
| | \ ` / | | ` ' /
| | \ / \ / \ /
'-' `-' `-' `-' `-'
```

A red arrow points from the text '(若依启动成功)' to the text '项目启动成功!' (Project started successfully!).

At the bottom right of the console output, there is a watermark: [github.com/hansonwang99/JavaCollection](https://github.com/hansonwang99/JavaCollection) 开源项目已收录

Other tabs in the Run tool window include 'Endpoints', 'Statistics', 'Terminal', 'Java Enterprise', and 'Spring'. The status bar at the bottom shows 'All files are up-to-date (moments ago)' and the time '109:1'.

项目运行成功，这样后续就可以对其进行调试和代码的研读了。

# 项目打包

## 前端项目打包

我们准备将前端项目部署到 `Linux 节点1` 上，所以进入 `Linux 节点1` 的代码目录：

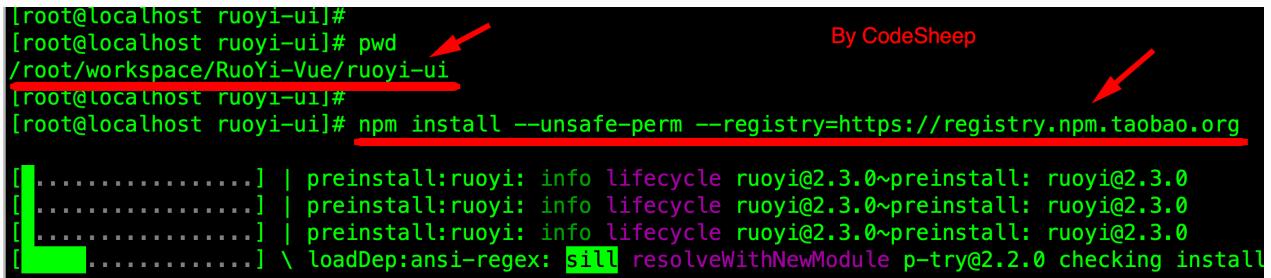
```
/root/workspace/RuoYi-Vue/ruoyi-ui
```

准备进行前端项目的打包操作。

### 1、安装依赖

进入前端项目子目录 `ruoyi-ui`，运行如下命令安装前端项目所需依赖：

```
npm install --unsafe-perm --registry=https://registry.npm.taobao.org
```

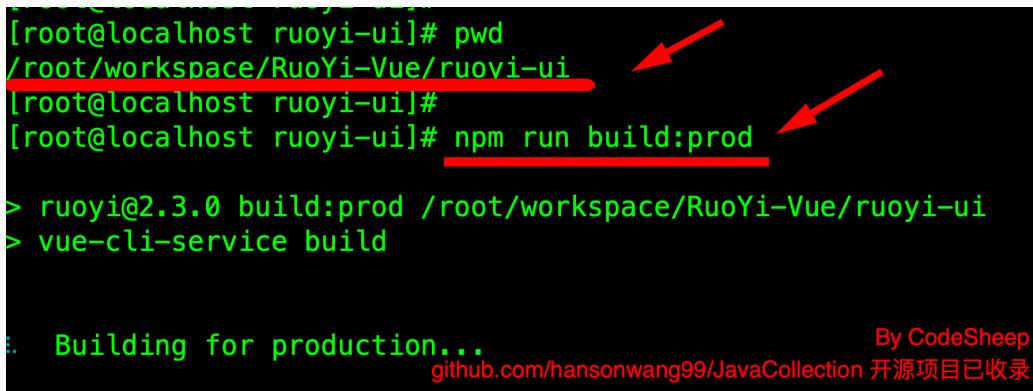


```
[root@localhost ruoyi-ui]# pwd
[root@localhost ruoyi-ui]# /root/workspace/RuoYi-Vue/ruoyi-ui
[root@localhost ruoyi-ui]# npm install --unsafe-perm --registry=https://registry.npm.taobao.org
[...]
[...] | preinstall:ruoyi: info lifecycle ruoyi@2.3.0~preinstall: ruoyi@2.3.0
[...] | preinstall:ruoyi: info lifecycle ruoyi@2.3.0~preinstall: ruoyi@2.3.0
[...] | preinstall:ruoyi: info lifecycle ruoyi@2.3.0~preinstall: ruoyi@2.3.0
[...] \ loadDep:ansi-regex: sill resolveWithNewModule p-try@2.2.0 checking install
```

### 2、构建打包

因为我们要部署，所以这里直接打包正式环境即可：

```
npm run build:prod
```



```
[root@localhost ruoyi-ui]# pwd
[root@localhost ruoyi-ui]# /root/workspace/RuoYi-Vue/ruoyi-ui
[root@localhost ruoyi-ui]# npm run build:prod
> ruoyi@2.3.0 build:prod /root/workspace/RuoYi-Vue/ruoyi-ui
> vue-cli-service build

... Building for production...
By CodeSheep
github.com/hansonwang99/JavaCollection 开源项目已收录
```

构建打包成功之后，会在根目录生成 `dist` 文件夹，里面就是构建打包好的前端项目文件：

```
DONE Build complete. The dist directory is ready to be deployed.  
INFO Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html  
  
[root@localhost ruoyi-ui]#  
[root@localhost ruoyi-ui]#  
[root@localhost ruoyi-ui]# ll  
总用量 748  
-rw-r--r--. 1 root root 58 6月 1 22:59 babel.config.js  
drwxr-xr-x. 2 root root 61 6月 1 22:59 bin  
drwxr-xr-x. 2 root root 22 6月 1 22:59 build  
drwxr-xr-x. 3 root root 57 6月 2 00:01 dist  
drwxr-xr-x. 1147 root root 32768 6月 1 23:52 node_modules  
-rw-r--r--. 1 root root 2907 6月 1 22:59 package.json  
-rw-r--r--. 1 root root 684594 6月 1 23:51 package-lock.json  
drwxr-xr-x. 2 root root 43 6月 1 22:59 public  
-rw-r--r--. 1 root root 460 6月 1 22:59 README.md  
drwxr-xr-x. 11 root root 203 6月 1 22:59 src  
-rw-r--r--. 1 root root 4455 6月 1 22:59 vue.config.js  
  
[root@localhost ruoyi-ui]# By CodeSheep  
[root@localhost ruoyi-ui]# github.com/hansonwang99/JavaCollection 开源项目已收录  
[root@localhost ruoyi-ui]#
```

该 `dist` 目录先留着，后面上部署前端项目时，直接拿来使用。

## 后端项目打包

我们准备将后端项目部署到 `Linux` 节点2和节点3，以节点2为例，进入代码目录：

```
cd /root/workspace/RuoYi-Vue/ruoyi
```

准备进行后端项目的打包操作。

### 打jar包

`Spring Boot` 由于自带 `Tomcat` 应用服务器，项目默认会打出可执行的 `jar` 包，非常方便。

直接在当前目录下执行 `mvn package` 命令即可构建打包：

```
[root@localhost ruoyi]# git clone https://github.com/hansonwang99/JavaCollection 开源项目已收录
[root@localhost ruoyi]#
[root@localhost ruoyi]# pwd      路径
/root/workspace/RuoYi-Vue/ruoyi
[root@localhost ruoyi]#
[root@localhost ruoyi]# mvn package 打包命令
[INFO] Scanning for projects...
Downloading from public: http://maven.aliyun.com/nexus/content/group
2.1.1.RELEASE.pom
Downloaded from public: http://maven.aliyun.com/nexus/content/group
.1.1.RELEASE.pom (12 kB at 12 kB/s)
Downloading from public: http://maven.aliyun.com/nexus/content/group
1.RELEASE.pom By CodeSheep
Downloaded from public: http://maven.aliyun.com/nexus/content/group
.RELEASE.pom (142 kB at 337 kB/s)
Downloading from public: http://maven.aliyun.com/nexus/content/group
Downloaded from public: http://maven.aliyun.com/nexus/content/group
Downloaded from public: http://maven.aliyun.com/nexus/content/group
```

该过程会延续一段时间，它会自动下载项目 `pom.xml` 中各种所需的依赖，并最终构建打包。

```
[INFO] Replacing main artifact with repackaged archive By CodeSheep
[INFO] -----
[INFO] BUILD SUCCESS ← 编译成功
[INFO] -----
[INFO] Total time:  07:43 min
[INFO] Finished at: 2020-06-02T09:35:59+08:00
[INFO] -----
[root@localhost ruoyi]#
[root@localhost ruoyi]#
[root@localhost ruoyi]# git clone https://github.com/hansonwang99/JavaCollection 开源项目已收录
```

构建成功如图所示，并且会在当前路径下生成一个 `target` 文件夹：

```
[root@localhost ruoyi]# pwd
/root/workspace/RuoYi-Vue/ruoyi
[root@localhost ruoyi]#
[root@localhost ruoyi]# ll
总用量 16
drwxr-xr-x. 2 root root   64 6月    1 23:00 bin
-rw-r--r--. 1 root root 8408 6月    1 23:00 pom.xml
-rw-r--r--. 1 root root 1697 6月    1 23:00 ry.sh
drwxr-xr-x. 2 root root   47 6月    1 23:00 sql
drwxr-xr-x. 3 root root   18 6月    1 23:00 src
drwxr-xr-x. 6 root root 131 6月    2 09:35 target
[root@localhost ruoyi]#
[root@localhost ruoyi]#
```

进入 `target` 文件夹，构建并打包出来可执行 `jar` 包即位于其中，大小约六十几兆：

```
[root@localhost ruoyi]# cd target/
[root@localhost target]#
[root@localhost target]#
[root@localhost target]#
[root@localhost target]# ll
总用量 67496
drwxr-xr-x. 6 root root      143 6月    2 09:33 classes
drwxr-xr-x. 3 root root      25 6月    2 09:33 generated-sources
drwxr-xr-x. 2 root root      28 6月    2 09:35 maven-archiver
drwxr-xr-x. 3 root root      35 6月    2 09:33 maven-status
-rw-r--r--. 1 root root 68690386 6月    2 09:35 ruoyi.jar
-rw-r--r--. 1 root root   418490 6月    2 09:35 ruoyi.jar.original
[root@localhost target]# By CodeSheep
[root@localhost target]#
```

## 打war包

上面说了，`Spring Boot` 由于默认内嵌 `Tomcat` 容器，所以打出来的 `jar` 包即可直接运行，但是我们有时候希望将其部署于外置的 `Tomcat` 服务器中，这时候就要打 `war` 包了。

打 `war` 包之前需要做几个代码上的配置：

### 1、修改 `pom.xml` 中的打包方式

修改 `/root/workspace/RuoYi-Vue/ruoyi/pom.xml` 文件，将：

```
<packaging>jar</packaging>
```

修改为：

```
<packaging>war</packaging>
```

### 2、剔除内嵌的Tomcat

我们只需要将 `Spring Boot` 内置的 `Tomcat` 在发布时去除即可。

修改 `/root/workspace/RuoYi-Vue/ruoyi/pom.xml` 文件，在原先的配置：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

下面加入如下配置即可：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

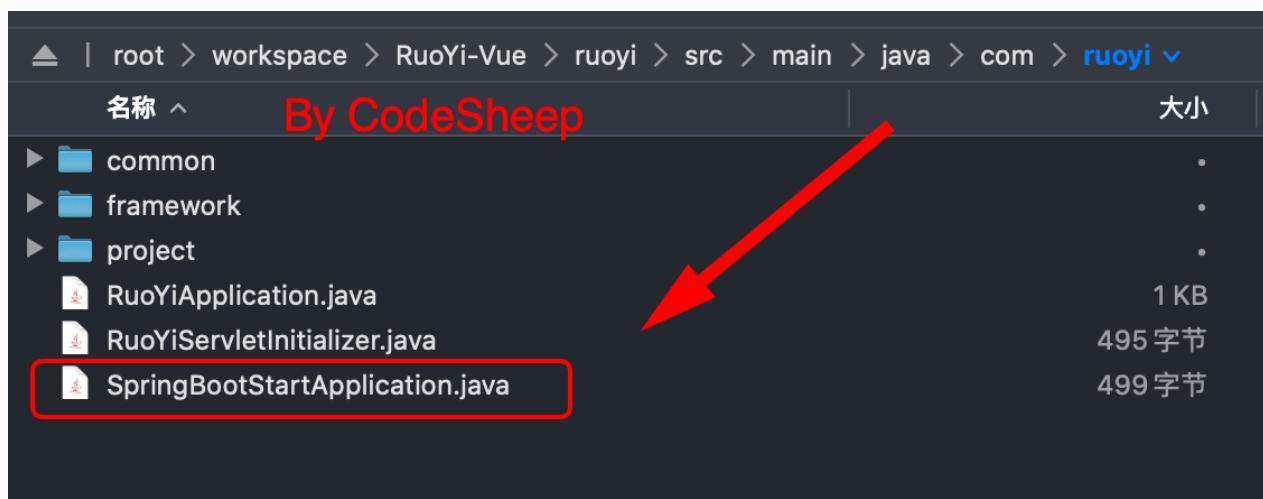
`provided` 表示该包只在编译和测试中使用，在发布时去除。

### 3、修改项目启动类

在 `Spring Boot` 中我们平常都用 `main` 方法启动的方式，都有一个 `SpringBootApplication` 的启动类，比如本项目的启动主类文件：

```
/Users/codesheep/IdeaProjects/RuoYi-
Vue/ruoyi/src/main/java/com/ruoyi/RuoYiApplication.java
```

而我们现在需要类似于 `web.xml` 的配置方式来启动 `Spring` 应用，为此，我们在 `Application` 类的同级目录下再添加一个 `SpringBootStartApplication` 类：



其代码如下：

```
public class SpringBootStartApplication extends
SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder
configure(SpringApplicationBuilder builder) {
        // 注意这里要指向原先用main方法执行的 RuoYiApplication启动类
        return builder.sources(RuoYiApplication.class);
    }
}
```

### 4、执行打包操作

切换到后台项目目录下：

```
/root/workspace/RuoYi-Vue/ruoyi
```

首先执行：

```
mvn clean
```

清理之前的打包结果：

```
[root@localhost ruoyi]# github.com/hansonwang99/JavaCollection 开源项目已收录
[root@localhost ruoyi]# pwd
/root/workspace/RuoYi-Vue/ruoyi 执行目录
[root@localhost ruoyi]#
[root@localhost ruoyi]#
[root@localhost ruoyi]# mvn clean 执行清理动作
[INFO] Scanning for projects...
[INFO]
[INFO] ----- < com.ruoyi:ruoyi > -----
[INFO] Building ruoyi 2.3.0
[INFO] ----- [ jar ] -----
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ ruoyi ---
[INFO]
[INFO] BUILD SUCCESS ← 清理成功!
[INFO]
[INFO] Total time: 0.370 s
```

然后再次执行 `mvn package` 命令即可构建打war包了，打包结果仍然位于生成的 `target` 目录下：

```
[INFO] Packaging webapp
[INFO] Assembling webapp [ruoyi] in [/root/workspace/RuoYi-Vue/ruoyi/target/ruoyi]
[INFO] Processing war project
[INFO] Webapp assembled in [276 msecs] 打包生成war包
[INFO] Building war: /root/workspace/RuoYi-Vue/ruoyi/target/ruoyi.war
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.1.RELEASE:repackage (repackage) @ ruoyi ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] BUILD SUCCESS ← 构建打包成功
[INFO]
[INFO] Total time: 57.298 s
[INFO] Finished at: 2020-06-02T12:38:56+08:00
[INFO]
[root@localhost ruoyi]# github.com/hansonwang99/JavaCollection 开源项目已收录
[root@localhost ruoyi]# By CodeSheep
```

# 前端项目部署

上文刚刚阐述过前端项目的打包过程，是在 `Linux` 节点1上进行的，打包好的前端项目成品位于 `dist` 目录，具体路径为：

```
/root/workspace/RuoYi-Vue/ruoyi-ui/dist
```

前端项目的部署非常简单，我们只需要将 `dist` 目录发布到你提前安装部署好的的 `Web` 服务器静态目录即可。

我们这里选用的是 `Nginx` 这款高性能Web服务器，接下来我们简单配置下 `Nginx`，并重启生效。

关于Nginx等基础设施和软件的安装部署过程，之前已经专门写过PDF了，可参考前文：[熬10天夜，肝出了这个PDF版「软件安装手册」](#)

我的 `Nginx` 安装于如下目录：

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

其配置文件位于目录：

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

简要配置 `nginx.conf` 文件如下所示：

```
user    root;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       mime.types;
```

```
default_type application/octet-stream;

#log_format main '$remote_addr - $remote_user [$time_local]
"$request" '
#                      '$status $body_bytes_sent "$http_referer" '
#                      '"$http_user_agent" "$http_x_forwarded_for"';

#access_log logs/access.log main;

sendfile          on;
#tcp_nopush      on;

#keepalive_timeout 0;
keepalive_timeout 65;

#gzip  on;

server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        root   /root/workspace/RuoYi-Vue/ruoyi-ui/dist;
        index index.html index.htm;
    }

    #error_page  404          /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass    http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on
127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass  127.0.0.1:9000;
    #    fastcgi_index index.php;
    #}
```

```
#      fastcgi_param  SCRIPT_FILENAME
/scripts$fastcgi_script_name;
#      include          fastcgi_params;
#}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny  all;
#}
}

# another virtual host using mix of IP-, name-, and port-based
configuration
#
#server {
#    listen      8000;
#    listen      somename:8080;
#    server_name somename alias another.alias;

#    location / {
#        root   html;
#        index  index.html index.htm;
#    }
#}

# HTTPS server
#
#server {
#    listen      443 ssl;
#    server_name localhost;

#    ssl_certificate      cert.pem;
#    ssl_certificate_key  cert.key;

#    ssl_session_cache    shared:SSL:1m;
#    ssl_session_timeout  5m;

#    ssl_ciphers  HIGH:!aNULL:!MD5;
#    ssl_prefer_server_ciphers  on;

#    location / {
#        root   html;
#        index  index.html index.htm;
#    }
#}
}
```

此处对原始 `nginx.conf` 文件只做了两处改动：

1、配置 `user root`

2、配置 `location /`

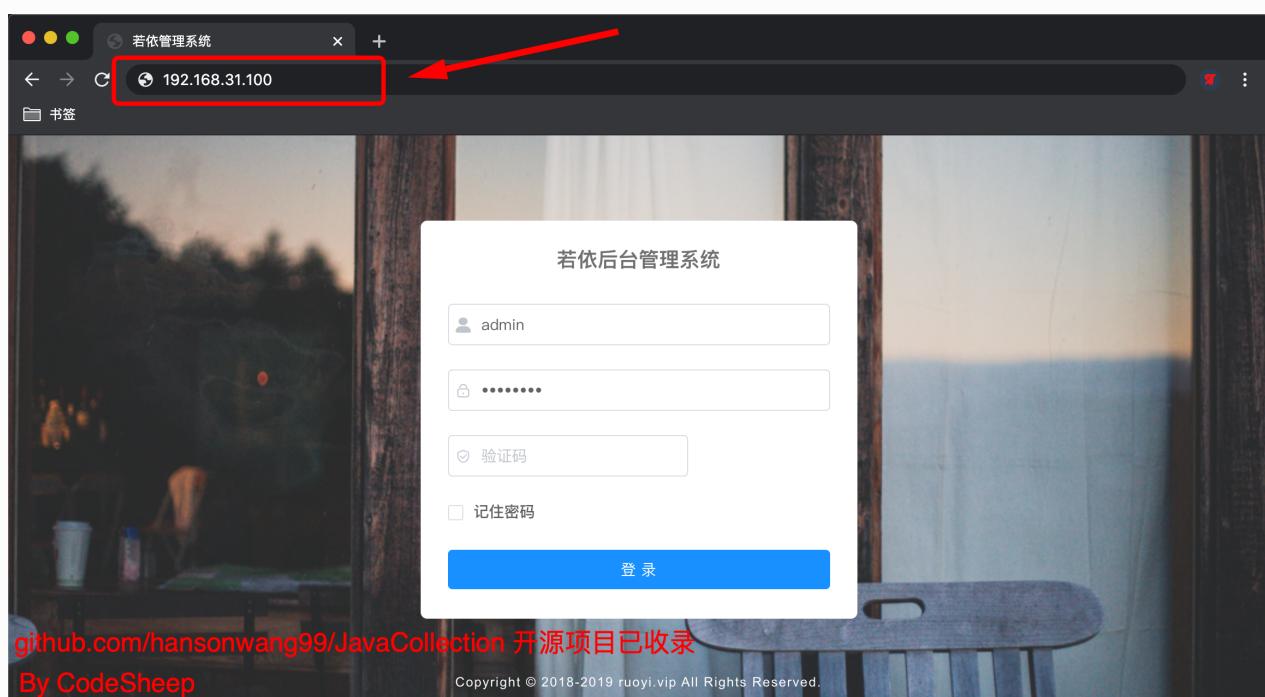
```
location / {  
    root /root/workspace/RuoYi-Vue/ruoyi-ui/dist;  
    index index.html index.htm;  
}
```

目的很简单，一眼就能看明白。

接下来执行如下命令重新加载 Nginx 即可生效：

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

然后打开浏览器，输入节点1的IP：`192.168.31.100` 即可访问网站前端页面：



至此前端部署结束，接下来进行后端项目的部署。

# 后端项目部署

## JAR包部署方式

`jar` 包部署方式非常简单，几乎只需要一行命令即可搞定，因为 `Spring Boot` 内嵌 `Tomcat`，开箱即用的感觉真的很爽。

上文阐述过后端项目的打 `jar` 包的过程，是在 `Linux` 节点2上进行的，打包好的后端项目 `jar` 包成品位于 `target` 目录，具体路径为：

```
/root/workspace/RuoYi-Vue/ruoyi/target/ruoyi.jar
```

我们只需要进入 `target` 目录中执行如下命令即可在后台部署项目：

```
nohup java -jar ruoyi.jar &
```

```
[root@localhost target]#  
[root@localhost target]# pwd  
/root/workspace/RuoYi-Vue/ruoyi/target 路径  
[root@localhost target]#  
[root@localhost target]# ll  
总用量 67496  
drwxr-xr-x. 6 root root 143 6月 2 11:50 classes  
drwxr-xr-x. 3 root root 25 6月 2 11:50 generated-sources  
drwxr-xr-x. 2 root root 28 6月 2 11:50 maven-archiver  
drwxr-xr-x. 3 root root 35 6月 2 11:50 maven-status  
-rw-r--r--. 1 root root 68690386 6月 2 11:50 ruoyi.jar  
-rw-r--r--. 1 root root 418490 6月 2 11:50 ruoyi.jar.original  
[root@localhost target]#  
[root@localhost target]# nohup java -jar ruoyi.jar &  
[1] 50135  
[root@localhost target]# nohup: 忽略输入并把输出追加到"nohup.out"  
[root@localhost target]# By CodeSheep  
[root@localhost target]#
```

## WAR包部署方式

war包部署方式就需要额外安装Tomcat应用服务器，我这里的Tomcat安装于如下目录：

```
/usr/local/tomcat/apache-tomcat-8.5.55
```

关于Tomcat等基础设施和软件的安装部署过程，之前已经专门写过PDF了，可参考前文：[熬10天夜，肝出了这个PDF版「软件安装手册」](#)

`war` 包的部署方式也非常简单，我们只需要将上文后端项目打包好的 `war` 包直接放入 `Tomcat` 的 `webapps` 目录即可。

上文阐述过后端项目的打 `war` 包的过程，是在 `Linux` 节点2上进行的，打包好的后端项目 `war` 包成品位于 `target` 目录，具体路径为：

```
/root/workspace/RuoYi-Vue/ruoyi/target/ruoyi.war
```

这里我们直接使用 `cp` 命令拷贝一份到 `Tomcat` 的 `webapps` 目录里即可：

```
cp /root/workspace/RuoYi-Vue/ruoyi/target/ruoyi.war  
/usr/local/tomcat/apache-tomcat-8.5.55/webapps/
```

这时候后端理论上就部署完毕，可以访问了，只不过现在还需要加项目名进行访问，就像这样：

```
http://192.168.31.101:8080/ruoyi/
```

很明显这样有点不方便。

所以我们接下来设置去掉项目名，直接根目录访问。

我们只需要修改一下 `Tomcat` 的配置文件 `server.xml`：

```
/usr/local/tomcat/apache-tomcat-8.5.55/conf/server.xml
```

在其 `<Host>` 标签下面加入如下内容即可：

```
<Context path="/" docBase="/usr/local/tomcat/apache-tomcat-  
8.5.55/webapps/ruoyi" reloadable="false"></Context>
```

```
148     <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
149         |   resourceName="UserDatabase"/>
150     </Realm>
151
152     <Host name="localhost" appBase="webapps"
153         |   unpackWARs="true" autoDeploy="true">
154
155         <Context path="/" docBase="/usr/local/tomcat/apache-tomcat-8.5.55/webapps/ruoyi" reloadable="false"></Context>
156         <!-- SingleSignOn valve, share authentication between web applications
157             Documentation at: /docs/config/valve.html -->
158         <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
159         <!--
160             Documentation at: /docs/config/valve.html
161             Note: The pattern used is equivalent to using pattern="common" -->
162             <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
163                 |   prefix="localhost_access_log" suffix=".txt"
164                 |   pattern="%h %l %u %t "%r" %s %b" />
165
166             By CodeSheep
167         </Valve>
168     </Host>
169 </Engine> github.com/hansonwang99/JavaCollection 开源项目已收录
170
```

最后重启 Tomcat 即可生效：

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

## 配置NGINX代理和转发

接下来我们继续配置节点1上的 Nginx， 在其配置文件中加上如下配置：

```
location /prod-api/ {
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header REMOTE-HOST $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://192.168.31.101:8080/;
}
```

可以看出来，我们的目的是想将所有发往节点1上的 /prod-api/ 请求，全部转发到节点2上部署的后端项目里。

至于为什么这里配基础转发路径 /prod-api/，这个需要看一下前端项目根目录里的隐藏文件 .env.production 就明白了：

```

[root@localhost ruoyi-ui]# pwd
/root/workspace/RuoYi-Vue/ruoyi-ui 路径
[root@localhost ruoyi-ui]#
[root@localhost ruoyi-ui]# ls -al
总用量 784
drwxr-xr-x.  8 root root  4096 6月   2 00:01 .
drwxr-xr-x.  5 root root    79 6月   1 22:59 ..
-rw-r--r--.  1 root root    58 6月   1 22:59 babel.config.js
drwxr-xr-x.  2 root root   61 6月   1 22:59 bin
drwxr-xr-x.  2 root root   22 6月   1 22:59 build
drwxr-xr-x.  3 root root   57 6月   2 00:01 dist
-rw-r--r--.  1 root root  536 6月   1 22:59 .editorconfig
-rw-r--r--.  1 root root 172 6月   1 22:59 .env.development
-rw-r--r--.  1 root root 111 6月   1 22:59 .env.production
-rw-r--r--.  1 root root 134 6月   1 22:59 .env.staging
-rw-r--r--.  1 root root 307 6月   1 22:59 .eslintignore
-rw-r--r--.  1 root root 5342 6月   1 22:59 .eslintrc.js
-rw-r--r--.  1 root root 279 6月   1 22:59 .gitignore
drwxr-xr-x. 1147 root root 32768 6月   1 23:52 node_modules
-rw-r--r--.  1 root root 2907 6月   1 22:59 package.json
-rw-r--r--.  1 root root 684594 6月   1 23:51 package-lock.json
drwxr-xr-x.  2 root root   43 6月   1 22:59 public
-rw-r--r--.  1 root root   460 6月   1 22:59 README.md
drwxr-xr-x.  11 root root  203 6月   1 22:59 src
-rw-r--r--.  1 root root 4455 6月   1 22:59 vue.config.js
[root@localhost ruoyi-ui]# By CodeSheep
[root@localhost ruoyi-ui]#

```

其内容如下：

```

# 生产环境配置
ENV = 'production'

# 生产环境
VUE_APP_BASE_API = '/prod-api'

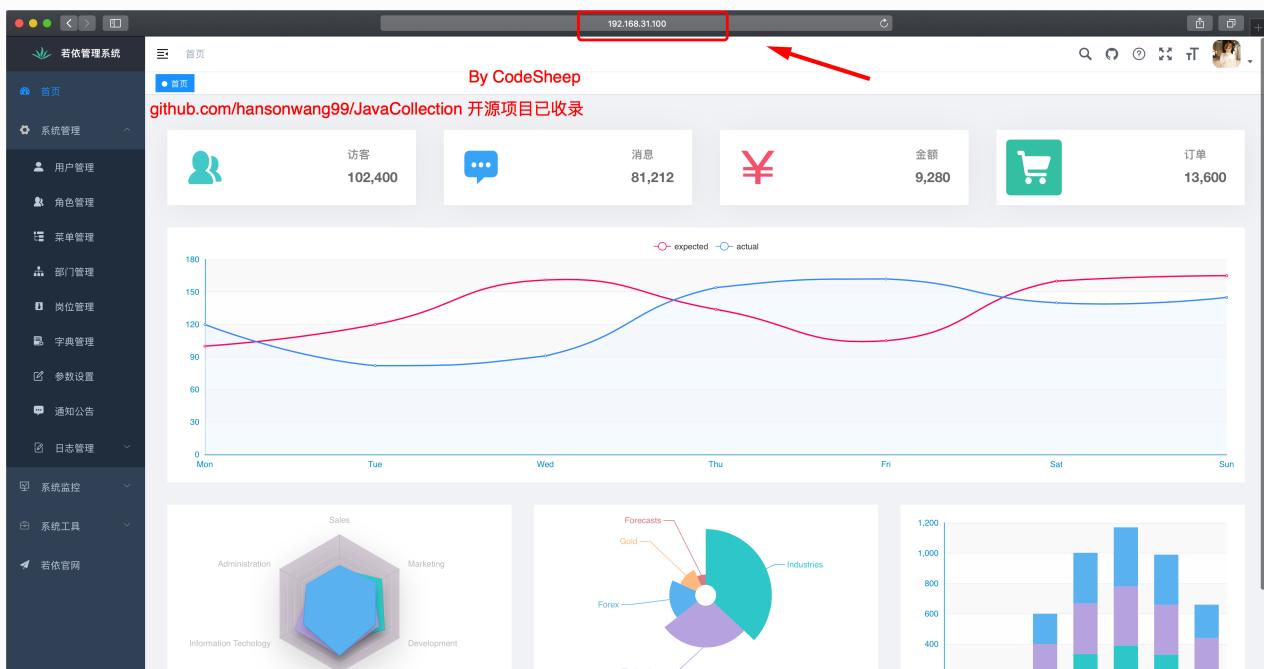
```

这就是需要设置的基础转发路径。

然后接下来执行如下命令重新加载 Nginx 即可生效：

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

然后打开浏览器，访问节点1的IP： `192.168.31.100`，即可正常登录，进入项目的后台系统：



至此所有的部署全部结束！

# 后台项目多实例部署

对于上面的部署，我们对后台项目只部署了一个运行实例，位于**节点2**上。

但实际情况我们处于提升性能或稳定性等考虑，我们希望对后台项目在多个节点上**部署多实例**，所以接下来简要演示一下。

正好我们还准备了一个**节点3**机器，重复上文的后端项目部署过程，在**节点3**上也将后端项目给同样地运行起来。

然后我们继续配置**节点1**上的**Nginx**，在其配置文件中做如下配置：

```
upstream ruoyi{
    server 192.168.31.101:8080 weight=5;
    server 192.168.31.102:8080 weight=3;
}

server {
    # ... 省略 ...

    location /prod-api/ {
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For
        $proxy_add_x_forwarded_for;
        proxy_pass http://ruoyi/;
    }

    # ... 省略 ...
}
```

首先我们配置了一个名为**ruoyi** 的服务器集群地址，其包含两个实例（分别对应**节点2**和**节点3**），并且分配了不同的权重 **weight**（理论上来说权重 **weight** 越大，对应**Tomcat** 上分配到的请求也会越多）。

同时将 **location /prod-api/** 里的代理转发路径指向了 **ruoyi** 多实例集群，这样当我们访问前端时，**Nginx** 会自动将我们的请求转发到两个实例中的任何一个中去处理。

这样即使后端实例down掉一个，还有其他实例可以支撑系统的运行，稳定性和性能均要更高一些。

---

至此，本文档实验过程完全结束！



# 后记

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：



另外：本文档在 Github开源项目：[github.com/hansonwang99/JavaCollection](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...

# 编程环境和软件设施安装手册

## 编程环境和软件设施安装手册

### 环境声明（重要！！！）

### 安装配置Linux系统环境

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](#)）

### Git工具安装

方式一：通过包管理器安装

方法二：通过源码编译安装

### JDK（Java环境）安装

准备JDK安装包

卸载已有的OpenJDK（如果有）

创建目录并解压

配置JDK环境变量

验证JDK安装结果

### Node环境安装

准备Node安装包

创建目录并解压

配置Node系统环境变量

检查安装结果

### Python环境安装

准备python3安装包并解压

安装相关预备环境

编译并安装

添加软链接

验证安装

### Maven项目构建和管理工具安装

准备Maven安装包并解压

配置Maven加速镜像源

配置环境变量

检验安装结果

### MySQL数据库部署和安装

首先准备安装包

卸载系统自带的Mariadb（如果有）

解压MySQL安装包

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

查看安装结果

设置开机启动

配置Docker镜像下载加速

## Kubernetes集群部署

概述

节点规划

准备工作

组件安装

0x01. Docker安装（所有节点）

0x02. kubelet、kubeadm、kubectl安装（所有节点）

### Master节点配置

0x01. 初始化 k8s集群

0x02. 配置 kubectl

0x03. 安装Pod网络

添加 Slave节点

效果验证

拆卸集群

安装 dashboard

## ElasticSearch集群部署

环境准备

Elasticsearch 集群配置

集群启动前准备

启动 Elasticsearch集群

安装IK分词器

## ZooKeeper安装部署

准备安装包

解压并安装

创建data目录

创建配置文件并修改

启动ZooKeeper

配置环境变量

设置开机自启

## 消息队列Kafka安装部署

首先准备ZooKeeper服务

准备Kafka安装包

解压并安装

创建logs目录

修改配置文件

启动Kafka

实验验证

## 注意事项

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](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...

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。

## 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）](#)

---

# 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

```
[root@localhost bin]#  
[root@localhost bin]# git --version  
git version 2.26.2  
[root@localhost bin]#  
[root@localhost bin]#
```

# 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 `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
```

```
[root@localhost ~]# rpm -qa | grep java
java-1.8.0-openjdk-headless-1.8.0.131-11.b12.el7.x86_64 ✓
python-javapackages-3.4.1-11.el7.noarch
libvirt-java-0.4.9-4.el7.noarch
java-1.6.0-openjdk-1.6.0.41-1.13.13.1.el7_3.x86_64 ✓
javapackages-tools-3.4.1-11.el7.noarch
java-1.7.0-openjdk-headless-1.7.0.141-2.6.10.5.el7.x86_64 ✓
java-1.8.0-openjdk-devel-1.8.0.131-11.b12.el7.x86_64 ✓
libvirt-java-devel-0.4.9-4.el7.noarch
java-1.7.0-openjdk-devel-1.7.0.141-2.6.10.5.el7.x86_64 ✓
java-1.7.0-openjdk-1.7.0.141-2.6.10.5.el7.x86_64 ✓
java-1.6.0-openjdk-devel-1.6.0.41-1.13.13.1.el7_3.x86_64 ✓
java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86_64 ✓
tzdata-java-2017b-1.el7.noarch
```

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	By CodeSheep
-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	

# 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
~
```

CodeSheep

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]#
```

CodeSheep

# 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  
>>>  
>>>  
>>> CodeSheep
```

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 info  
>>>  
>>>
```

# 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 the
144 | server for that repository.
145 |-->
146 <mirrors>
147   <!-- mirror
148   | Specifies a repository mirror site to use instead of a given repository. The re
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 o
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
```



## 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：

```
[root@izbp1gl6m86wbyqvme2gs8z conf]# mvn -v  
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)  
Maven home: /opt/maven/apache-maven-3.6.3  
Java version: 1.8.0_161, vendor: Oracle Corporation, runtime: /usr/local/jdk1.8.0_161/jre  
Default locale: en_US, platform encoding: UTF-8  
OS name: "linux", version: "3.10.0-693.2.2.el7.x86_64", arch: "amd64",  
[root@izbp1gl6m86wbyqvme2gs8z conf]# [root@izbp1gl6m86wbyqvme2gs8z conf]# [root@izbp1gl6m86wbyqvme2gs8z conf]#
```

# 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
```

```
[root@localhost ~]# rpm -qa|grep mariadb
mariadb-server-5.5.56-2.el7.x86_64    ✓
mariadb-5.5.56-2.el7.x86_64            ✓
mariadb-devel-5.5.56-2.el7.x86_64      ✓
mariadb-libs-5.5.56-2.el7.x86_64       ✓
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
```

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
```

```
[root@localhost mysql]#  
[root@localhost mysql]# ./bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/  
mysql/data  
mysqld: [Warning] World-writable config file '/etc/my.cnf' is ignored.  
2020-05-14T04:28:12.109751Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --expli-  
cit_defaults_for_timestamp server option (see documentation for more details).  
2020-05-14T04:28:12.230032Z 0 [Warning] InnoDB: New log files created, LSN=45790  
2020-05-14T04:28:12.250276Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.  
2020-05-14T04:28:12.307292Z 0 [Warning] No existing UUID has been found, so we assume that this is the first ti-  
me that this server has been started. Generating a new UUID: 52657904-959b-11ea-b810-000c2955d48a.  
2020-05-14T04:28:12.308405Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot  
be opened.  
2020-05-14T04:28:12.734296Z 0 [Warning] CA certificate ca.pem is self signed.  
2020-05-14T04:28:13.050520Z 1 [Note] A temporary password is generated for root@localhost: URE>:0kk:3Zy  
[root@localhost mysql]# CodeSheep  
[root@localhost mysql]#
```

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
```

```
[root@localhost mysql]# chkconfig --list mysqld
```

CodeSheep

注：该输出结果只显示 SysV 服务，并不包含  
原生 `systemd` 服务。SysV 配置数据  
可能被原生 `systemd` 配置覆盖。

要列出 `systemd` 服务，请执行 '`systemctl list-unit-files`'。  
查看在具体 target 启用的服务请执行  
'`systemctl list-dependencies [target]`'。

```
mysqld      0:关    1:关    2:开    3:开    4:开    5:开    6:关
[root@localhost mysql]#
[root@localhost mysql]#
[root@localhost mysql]#
```

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 rights reserved.  
Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
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 input statement.  
  
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>
```

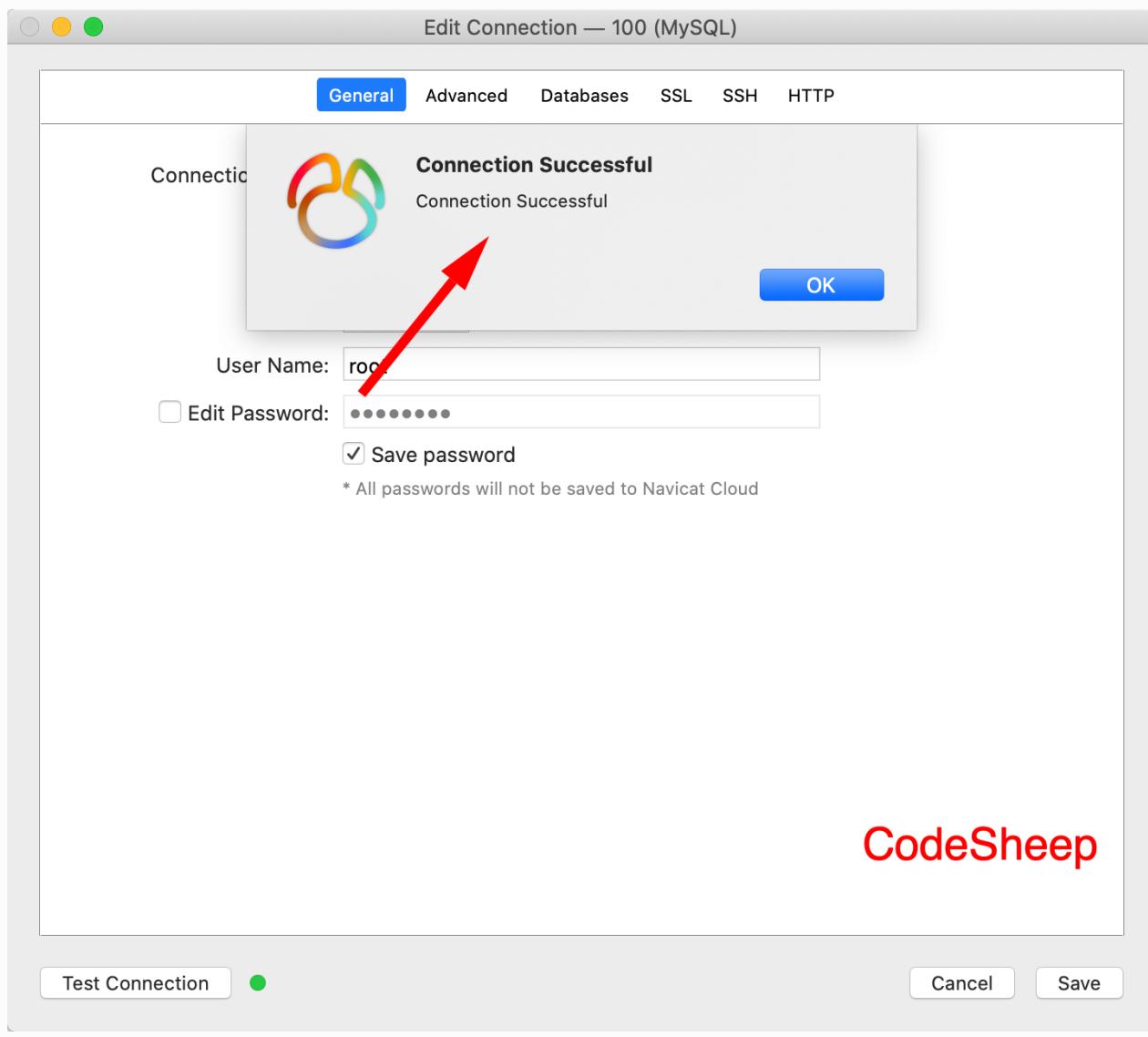
比如这里将密码设置成简单的“111111”了。



## 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：



CodeSheep

# 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]# ./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!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
[root@localhost utils]#
```

CodeSheep

## 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：

```
[root@localhost utils]#  
[root@localhost utils]#  
[root@localhost utils]# redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379> set foo bar  
OK  
127.0.0.1:6379>  
127.0.0.1:6379> get foo  
"bar"  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379>
```

CodeSheep

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

```
#  
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL  
# JUST COMMENT THE FOLLOWING LINE.  
# ~~~~~  
bind 0.0.0.0  
  
# Protected mode is a layer of security protection, in order  
# Redis instances left open on the internet are accessed  
#
```

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：

```
[root@localhost ~]#  
[root@localhost ~]# redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379> get foo  
(error) NOAUTH Authentication required. 需要密码认证  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379> auth codesheep 输入密码  
OK  
127.0.0.1:6379>  
127.0.0.1:6379> get foo  
"bar"  
127.0.0.1:6379>  
127.0.0.1:6379> █ CodeSheep
```

# 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
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
正在更新    : yum-utils-1.1.31-53.el7.noarch
清理        : yum-utils-1.1.31-42.el7.noarch
验证中      : yum-utils-1.1.31-53.el7.noarch
验证中      : yum-utils-1.1.31-42.el7.noarch
                                         1/2
                                         2/2
                                         1/2
                                         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@packagecloud.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/repo/other.xml.gz: [Errno 12] Timeout on https://packagecloud.io/rabbitmq/erlang/el/7/SRPMS/repo/other.xml.gz: (28, 'Operation timed out after 30000 milliseconds with 0 out of 0 bytes received')
正在尝试其它镜像。
https://packagecloud.io/rabbitmq/erlang/el/7/SRPMS/repo/other.xml.gz: [Errno 14] curl#7 - "Failed to connect 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_64-22.3.3-1.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

事务摘要
=====
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

事务摘要
=====
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>
```

## 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://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey)"
  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/5bf2d7a2f580b2hfq314.cloudfront.net/828/1033/el/7/x86_64/repodata/5bf2d7a2f580b26a717ad8c27979bf808, '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

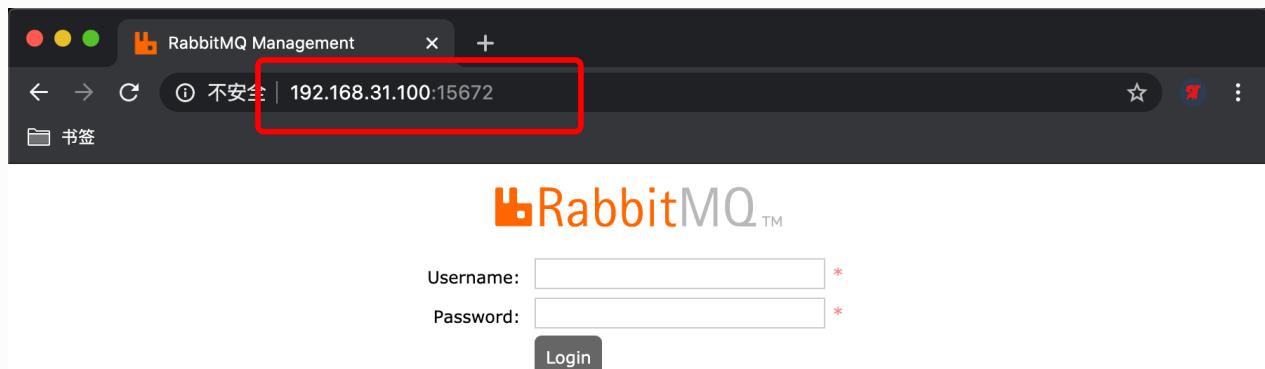
```
systemctl start rabbitmq-server.service
```

## 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



CodeSheep

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：

The screenshot shows the RabbitMQ Management interface. At the top, it displays the version "3.8.3 Erlang 22.3.3". On the right, it shows the cluster name "Cluster rabbit@localhost" and the user "User codesheep" with a "Log out" button. Two red arrows point from the text above to these specific areas. Below the header, there are tabs for Overview, Connections, Channels, Exchanges, Queues, and Admin. The Overview tab is selected. In the main area, it says "Currently idle" and shows global counts for Connections (0), Channels (0), Exchanges (7), Queues (0), and Consumers (0). A section titled "Nodes" lists the node "rabit@localhost" with its resource usage: File descriptors (36, 32768 available), Socket descriptors (0, 29401 available), Erlang processes (425, 1048576 available), Memory (75MiB, 729MiB high watermark), and Disk space (37GiB, 48MiB low watermark). A red arrow points from the text "User codesheep" to the "User codesheep" button in the top right.

# 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]# ./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/bootstrap.jar:/local/tomcat/apache-tomcat-8.5.55/bin/tomcat-juli.jar
Tomcat started.                                     CodeSheep
[root@localhost bin]#
```

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：

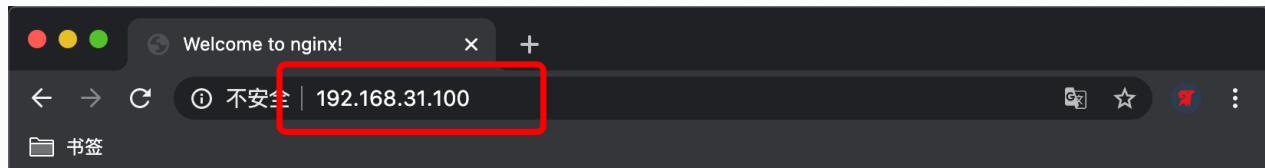


```
/usr/local/nginx/sbin/nginx -s reload
```

注意其配置文件位于：

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

## 浏览器验证启动情况



### Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

CodeSheep

# DOCKER环境安装

## 安装DOCKER

```
yum install -y docker
```

静候安装完毕即可

```
[root@centos ~]# yum install -y docker
[...]
验证中 : selinux-policy-3.13.1-166.el7.noarch
验证中 : libselinux-devel-2.5-11.el7.x86_64
验证中 : python-dmidecode-3.12.2-1.el7.x86_64
验证中 : libsepol-2.5-6.el7.x86_64
验证中 : policycoreutils-2.5-17.1.el7.x86_64
验证中 : libsemanage-python-2.5-8.el7.x86_64
验证中 : libselinux-python-2.5-11.el7.x86_64
验证中 : selinux-policy-targeted-3.13.1-166.el7.noarch
验证中 : setools-libs-3.3.8-1.1.el7.x86_64
验证中 : libsemanage-2.5-8.el7.x86_64

已安装:
docker.x86_64 2:1.13.1-161.git64e9980.el7_8
```

CodeSheep

## 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 ~]#
```

CodeSheep

## 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:
```

CodeSheep

## 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
7ebe8eeefbaf: Waiting
597069368ef1: Waiting
ce39a5501878: Waiting
7d545bca14bf: Waiting
0f5f78cccacb: Waiting
623a5dae2b42: Waiting
```



下载速度太慢

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

- 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
```

```
[root@localhost ~]# yum install -y kubelet kubeadm kubectl
已加载插件: fastestmirror, langpacks
CodeSheep
kubernetes
kubernetes/primary
Loading mirror speeds from cached hostfile
 * base: centos.ustc.edu.cn
 * extras: centos.ustc.edu.cn
 * updates: centos.ustc.edu.cn
kubernetes
正在解决依赖关系
正在检查事务
--> 正在检查事务
--> 软件包 kubeadm.x86_64 0.1.13.1-0 将被 安装
--> 正在处理依赖关系 kubernetes-cni >= 0.6.0, 它被软件包 kubeadm-1.13.1-0.x86_64 需要
--> 正在处理依赖关系 cri-tools >= 1.11.0, 它被软件包 kubeadm-1.13.1-0.x86_64 需要
--> 软件包 kubelet.x86_64 0.1.13.1-0 将被 安装
--> 软件包 socat.x86_64 0.1.13.1-0 将被 安装
--> 正在处理依赖关系 socat, 它被软件包 kubelet-1.13.1-0.x86_64 需要
正在检查事务
--> 软件包 cri-tools.x86_64 0.1.12.0-0 将被 安装
--> 软件包 kubernetes-cni.x86_64 0.6.0-0 将被 安装
--> 软件包 socat.x86_64 0.1.7.3.2-2.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
=====
正在安装:
kubeadm          x86_64     1.13.1-0      kubernetes
kubectl          x86_64     1.13.1-0      kubernetes
kubelet          x86_64     1.13.1-0      kubernetes
为依赖而安装:
cri-tools         x86_64     1.12.0-0      kubernetes
kubernetes-cni   x86_64     0.6.0-0       kubernetes
socat            x86_64     1.7.3.2-2.el7  base

事务摘要
=====
```

## 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 mirror.googlecontainers/kube-apiserver:v1.13.1
docker pull mirror.googlecontainers/kube-controller-manager:v1.13.1
docker pull mirror.googlecontainers/kube-scheduler:v1.13.1
docker pull mirror.googlecontainers/kube-proxy:v1.13.1
docker pull mirror.googlecontainers/pause:3.1
docker pull mirror.googlecontainers/etcfd: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/etcfd:3.2.24 k8s.gcr.io/etcfd: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/etcfd: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/etcfd         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
```

```
[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 "kubernetes-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 ~]# [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 yndddp.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 ~]#
[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-wztbk  1/1     Running   0          5m28s   192.168.199.78  k8s-node-2
kube-system  kube-proxy-crr7k          1/1     Running   0          7m11s   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/wangxiao/ke/kubernetes-
dashboard-amd64:v1.10.0
docker tag registry.cn-qingdao.aliyuncs.com/wangxiao/ke/kubernetes-
dashboard-amd64:v1.10.0 k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0
docker image rm registry.cn-qingdao.aliyuncs.com/wangxiao/ke/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）](#)

- 获取登陆token

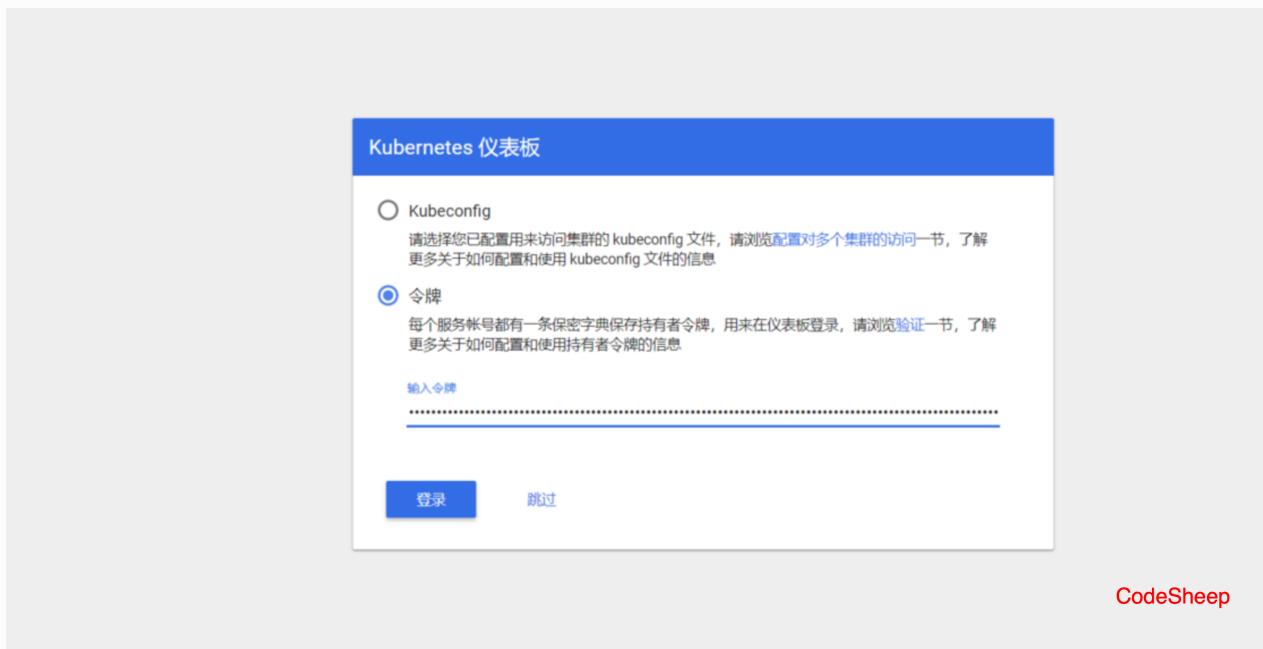
```
kubectl describe secret/$(kubectl get secret -nkube-system |grep admin|awk '{print $1}') -nkube-system
```

```
[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.eyJpc3MiOiJrdWJlcmlldGVzL3NlcnpY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pbv9zZXJ2aWN1YWnb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIIsImt1YmVybmv0ZXMuaw8vc2VydmljZWfjY291bnQvc2Vjcmv0Lm5hbWUiOijhZG1pbv10b2tlbi05ZDR2bCIsImt1YmVybmv0ZXMuaw8vc2VydmljZWfjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOijhZG1pbvIsImt1YmVybmv0ZXMuaw8vc2VydmljZWfjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImEzMjBiMDBmLTA3ZWQtMTF1OS05M2YyLTAvMGMyOTc4ZjIwNyIsInN1YiI6InN5c3RlbTpzzXJ2aWN1YWnb3VudDprdWJlLXN5c3RlbTphZG1pbvij9.WbaHx-
BfZEd0SvJwA9V_vGUe8jPMUhjk1kt7MWJ4JcQldRFY8Tdpv5GKCY25JsvT_GM3ob303r0yE6vjQdKna7EfQNO_Wb2j1Yu5UvZnWw52HhNudHNOVL_fFRKxksVjaILA_C_HvW6aw6TG5h7zHARgl71I0LpW1VESeHeThipQ-pkt-DrljWcpPgE39cwxSgi-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

The screenshot shows the Kubernetes Dashboard cluster overview page for the "kube-system" namespace. The left sidebar has a "集群" tab selected, showing options like 命名空间, 节点, 持久化存储卷, 角色, 存储类, and others. The main content area has three tables: 1) 命名空间 (Namespaces) table with rows for default, kube-public, and kube-system. 2) 节点 (Nodes) table with rows for k8s-node-2, k8s-node-1, and k8s-master. 3) 角色 (Roles) table with rows for kubernetes-dashboard-minimal, flannel, system:coredns, and kubeadm:bootstrap-signer-clusterinfo. Each table includes columns for name, labels, status, and creation time.

# 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 ~]# 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》

ElasticSearch Cluster Status  
内存,硬盘,JVM状态

Cluster Health		ElasticSearch Stats Info	
Status	green	clusterName	codesheep
Timed Out?	true	timestamp	1541149247921
Nodes	2	es versions	6.4.2
Data Nodes	2	os	Linux,
Active Primary Shards	10	jvm max uptime	6.4h
Initializing Shards	0	jvm versions	1.8.0_191,
Unassigned Shards	0	jvm threads	84

Index Templates中Type个数统计

Indices list				
Index	Docs	Size	Status	操作
accounts	1	11.5kb	open	<button>删除</button>

CodeSheep

- 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: "WOGj02YBfRHOCe7qdQH",
            _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
}
程序羊:www.codesheep.cn版权所有
```

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

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：

```
[5,645][INFO ][o.e.p.PluginsService      ] [sheep1] loaded module [x-pack-security]
[5,645][INFO ][o.e.p.PluginsService      ] [sheep1] loaded module [x-pack-sql]
[5,646][INFO ][o.e.p.PluginsService      ] [sheep1] loaded module [x-pack-upgrade]
[5,646][INFO ][o.e.p.PluginsService      ] [sheep1] loaded module [x-pack-watcher]
[5,646][INFO ][o.e.p.PluginsService      ] [sheep1] loaded plugin [analysis-ik]
[8,516][INFO ][o.e.x.s.a.s.FileRolesStore] [sheep1] parsed [0] roles from file [/opt/elasticsearch/e
[9,064][INFO ][o.e.x.m.j.p.l.CppLogMessageHandler] [controller/11013] [Main.cc@109] controller (64 b
[9,511][DEBUG ][o.e.a.ActionModule       ] Using REST wrapper from plugin org.elasticsearch.xpack.sec
[9,752][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/zookeeper` 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	<b>data</b>
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
```

CodeSheep

## 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
```



# 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
```

CodeSheep

另外关注一下连接 ZooKeeper 的相关配置，根据实际情况进行配置：

```
##### 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,blockStartProducing 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.TransactionCoordinator)
[2020-05-18 17:04:07,342] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2020-05-18 17:04:07,353] INFO [Transaction Marker Channel Manager 0]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2020-05-18 17:04:07,382] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpirationReaper)
[2020-05-18 17:04:07,424] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZKNodeChangeNotificationProcessor)
[2020-05-18 17:04:07,489] INFO [SocketServer brokerId=0] Started data-plane processors for 1 acceptors (kafka.network.Processor)
[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]# ./bin/kafka-topics.sh --create  
--bootstrap-server localhost:9092 --replication-factor 1 --parti  
tions 1 --topic codesheep  
Created topic codesheep. 创建成功  
[root@localhost kafka_2.12-2.5.0]#
```

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
```

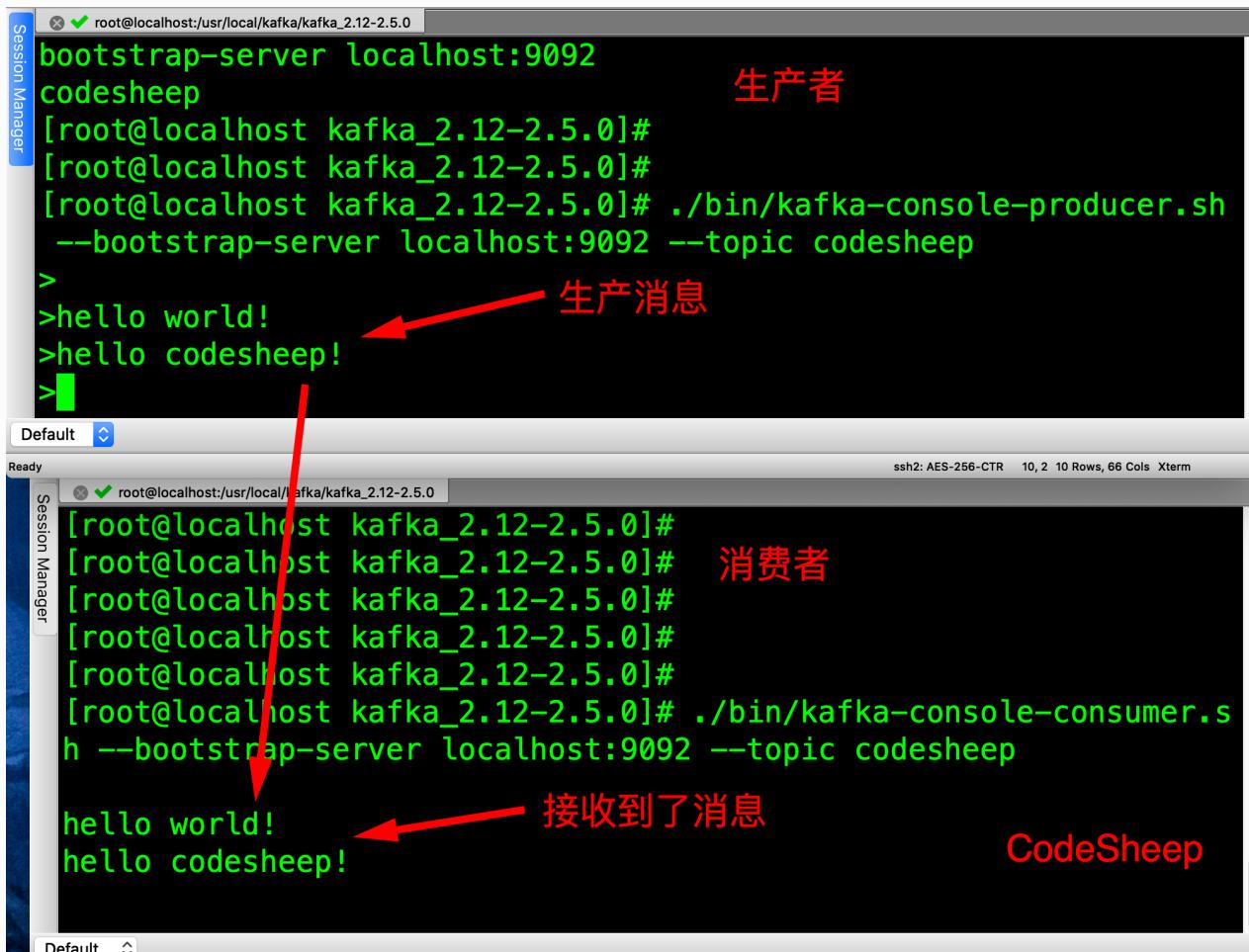
接下来创建一个生产者，用于在 `codesheep` 这个 `topic` 上生产消息：

```
./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：





## 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
```

## 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](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...

# 《常用LINUX命令备忘手册》

- 由于时间仓促和个人精力有限，难免会有疏漏和不当的地方，还望多多谅解
- 该PDF文档会持续更新，有新东西再往里加
- 另外，联系交流，可直接微信私信，一起进步提高 ↓



## 附：其他干货文档笔记下载

(点击标题链接可跳转)

- [数据结构和算法刷题笔记.pdf下载](#)
- [LeetCode算法刷题C/C++版答案pdf下载](#)
- [LeetCode算法刷题Java版答案pdf下载](#)
- [找工作简历模板集\(word格式\)下载](#)
- [Java基础核心知识大总结.pdf 下载](#)
- [C/C++常见面试题（含答案）下载](#)
- [设计模式学习笔记.pdf下载](#)
- [Java后端开发学习路线+知识点总结](#)
- [前端开发学习路线+知识点总结](#)
- [大数据开发学习路线+知识点总结](#)
- [C/C++\(后台\)学习路线+知识点总结](#)
- [嵌入式开发学习路线+知识点总结](#)

# 关机/重启/注销

常用命令	作用
shutdown -h now	即刻关机
shutdown -h 10	10分钟后关机
shutdown -h 11:00	11: 00关机
shutdown -h +10	预定时间关机 (10分钟后)
shutdown -c	取消指定时间关机
shutdown -r now	重启
shutdown -r 10	10分钟之后重启
shutdown -r 11:00	定时重启
reboot	重启
init 6	重启
init 0	立刻关机
telinit 0	关机
poweroff	立刻关机
halt	关机
sync	buff数据同步到磁盘
logout	退出登录Shell

# 系统信息和性能查看

常用命令	作用
uname -a	查看内核/OS/CPU信息
uname -r	查看内核版本
uname -m	查看处理器架构
arch	查看处理器架构
hostname	查看计算机名
who	显示当前登录系统的用户
who am i	显示登录时的用户名
whoami	显示当前用户名
cat /proc/version	查看linux版本信息
cat /proc/cpuinfo	查看CPU信息
cat /proc/interrupts	查看中断
cat /proc/loadavg	查看系统负载
uptime	查看系统运行时间、用户数、负载
env	查看系统的环境变量
lsusb -tv	查看系统USB设备信息
lspci -tv	查看系统PCI设备信息
lsmod	查看已加载的系统模块

grep MemTotal /proc/meminfo	查看内存总量
grep MemFree /proc/meminfo	查看空闲内存量
free -m	查看内存用量和交换区用量
date	显示系统日期时间
cal 2021	显示2021日历表
top	动态显示cpu/内存/进程等情况
vmstat 1 20	每1秒采一次系统状态，采20次
iostat	查看io读写/cpu使用情况
sar -u 1 10	查询cpu使用情况（1秒一次，共10次）
sar -d 1 10	查询磁盘性能

# 磁盘和分区

常用命令	作用
fdisk -l	查看所有磁盘分区
swapon -s	查看所有交换分区
df -h	查看磁盘使用情况及挂载点
df -hl	同上
du -sh /dir	查看指定某个目录的大小
<code>du -sk *   sort -rn</code>	从高到低依次显示文件和目录大小
mount /dev/hda2 /mnt/hda2	挂载hda2盘
mount -t ntfs /dev/sdc1 /mnt/usbhd1	指定文件系统类型挂载 (如ntfs)
mount -o loop xxx.iso /mnt/cdrom	挂载iso文件
mount /dev/sda1 /mnt/usbdisk	挂载usb盘/闪存设备
umount -v /dev/sda1	通过设备名卸载
umount -v /mnt/mymnt	通过挂载点卸载
fuser -km /mnt/hda1	强制卸载(慎用)

# 用户和用户组

常用命令	作用
useradd codesheep	创建用户
userdel -r codesheep	删除用户
usermod -g group_name user_name	修改用户的组
usermod -aG group_name user_name	将用户添加到组
usermod -s /bin/ksh -d /home/codepig -g dev codesheep	修改用户codesheep的登录Shell、主目录以及用户组
groups test	查看test用户所在的组
groupadd group_name	创建用户组
groupdel group_name	删除用户组
groupmod -n new_name old_name	重命名用户组
su - user_name	完整切换到一个用户环境
passwd	修改口令
passwd codesheep	修改某用户的口令
w	查看活动用户
id codesheep	查看指定用户codesheep信息
last	查看用户登录日志
crontab -l	查看当前用户的计划任务
cut -d: -f1 /etc/passwd	查看系统所有用户
cut -d: -f1 /etc/group	查看系统所有组

# 网络和进程管理

常用命令	作用
ifconfig	查看网络接口属性
ifconfig eth0	查看某网卡的配置
route -n	查看路由表
netstat -lntp	查看所有监听端口
netstat -antp	查看已经建立的TCP连接
netstat -lntp	查看TCP/UDP的状态信息
ifup eth0	启用eth0网络设备
ifdown eth0	禁用eth0网络设备
iptables -L	查看iptables规则
ifconfig eth0 192.168.1.1 netmask 255.255.255.0	配置ip地址
dhclient eth0	以dhcp模式启用eth0
route add -net 0/0 gw Gateway_IP	配置默认网关
route add -net 192.168.0.0 netmask 255.255.0.0 gw 192.168.1.1	配置静态路由到达网 络'192.168.0.0/16'
route del 0/0 gw Gateway_IP	删除静态路由
hostname	查看主机名

host <a href="http://www.codesheep.cn">www.codesheep.cn</a>	解析主机名
nslookup <a href="http://www.codesheep.cn">www.codesheep.cn</a>	查询DNS记录，查看域名解析是否正常
ps -ef	查看所有进程
<code>ps -ef   grep codesheep</code>	过滤出你需要的进程
kill -s name	kill指定名称的进程
kill -s pid	kill指定pid的进程
top	实时显示进程状态
vmstat 1 20	每1秒采一次系统状态，采20次
iostat	查看io读写/cpu使用情况
sar -u 1 10	查询cpu使用情况（1秒一次，共10次）
sar -d 1 10	查询磁盘性能

# 常见系统服务命令

常用命令	作用
chkconfig --list	列出系统服务
service <服务名> status	查看某个服务
service <服务名> start	启动某个服务
service <服务名> stop	终止某个服务
service <服务名> restart	重启某个服务
systemctl status <服务名>	查看某个服务
systemctl start <服务名>	启动某个服务
systemctl stop <服务名>	终止某个服务
systemctl restart <服务名>	重启某个服务
systemctl enable <服务名>	开启自启动
systemctl disable <服务名>	关闭自启动

# 文件和目录操作

常用命令	作用
cd <目录名>	进入某个目录
cd ..	回上级目录
cd ../../	回上两级目录
cd	进个人主目录
cd -	回上一步所在目录
pwd	显示当前路径
ls	查看文件目录列表
ls -F	查看目录中内容 (显示是文件还是目录)
ls -l	查看文件和目录的详情列表
ls -a	查看隐藏文件
ls -lh	查看文件和目录的详情列表 (增强文件大小易读性)
ls -lSr	查看文件和目录列表 (以文件大小升序查看)
tree	查看文件和目录的树形结构
mkdir <目录名>	创建目录
mkdir dir1 dir2	同时创建两个目录
mkdir -p /tmp/dir1/dir2	创建目录树
rm -f file1	删除'file1'文件
rmdir dir1	删除'dir1'目录

rm -rf dir1	删除'dir1'目录和其内容
rm -rf dir1 dir2	同时删除两个目录及其内容
mv old_dir new_dir	重命名/移动目录
cp file1 file2	复制文件
cp dir/* .	复制某目录下的所有文件至当前目录
cp -a dir1 dir2	复制目录
cp -a /tmp/dir1 .	复制一个目录至当前目录
ln -s file1 link1	创建指向文件/目录的软链接
ln file1 lnk1	创建指向文件/目录的物理链接
find / -name file1	从跟目录开始搜索文件/目录
find / -user user1	搜索用户user1的文件/目录
find /dir -name *.bin	在目录/dir中搜带有.bin后缀的文件
locate <关键词>	快速定位文件
locate *.mp4	寻找.mp4结尾的文件
whereis <关键词>	显示某二进制文件/可执行文件的路径
which <关键词>	查找系统目录下某的二进制文件
chmod ugo+rwx dir1	设置目录所有者(u)、群组(g)及其他用户(o)的读(r)写(w)执行(x)权限
chmod go-rwx dir1	移除群组(g)及其他用户(o)对目录的读写执行权限
chown user1 file1	改变文件的所有者属性
chown -R user1 dir1	改变目录的所有者属性
chgrp group1 file1	改变文件群组

```
chown  
user1:group1 file1
```

改变文件的所有人和群组

## 文件查看和处理

常用命令	作用
cat file1	查看文件内容
cat -n file1	查看内容并标示行数
cat xxx.txt	awk 'NR%2==1'
tac file1	从最后一行开始反看文件内容
more file1	查看一个长文件的内容
less file1	类似more命令，但允许反向操作
head -2 file1	查看文件前两行
tail -2 file1	查看文件后两行
tail -f /log/msg	实时查看添加到文件中的内容
grep codesheep hello.txt	在文件hello.txt中查找关键词codesheep
grep ^sheep hello.txt	在文件hello.txt中查找以sheep开头的内容

grep [0-9] hello.txt	选择hello.txt文件中所有包含数字的行
sed 's/s1/s2/g' hello.txt	将hello.txt文件中的s1替换成s2
sed '/^\$/d' hello.txt	从hello.txt文件中删除所有空白行
sed '/ *#/d; /^\$/d' hello.txt	从hello.txt文件中删除所有注释和空白行
sed -e '1d' hello.txt	从文件hello.txt 中排除第一行
sed -n '/s1/p' hello.txt	查看只包含关键词"s1"的行
sed -e 's/ *\$//' hello.txt	删除每一行最后的空白字符
sed -e 's/s1//g' hello.txt	从文档中只删除词汇s1并保留剩余全部
sed -n '1,5p;5q' hello.txt	查看从第一行到第5行内容
sed -n '5p;5q' hello.txt	查看第5行
paste file1 file2	合并两个文件或两栏的内容
paste -d '+' file1 file2	合并两个文件或两栏的内容，中间用"+"区分
sort file1 file2	排序两个文件的内容
sort file1 file2	uniq
sort file1 file2	uniq -u
sort file1 file2	uniq -d
comm -1 file1 file2	比较两个文件的内容(去除'file1'所含内容)
comm -2 file1 file2	比较两个文件的内容(去除'file2'所含内容)
comm -3 file1 file2	比较两个文件的内容(去除两文件共有部分)

# 打包和解压

常用命令	作用
zip xxx.zip file	压缩至zip包
zip -r xxx.zip file1 file2 dir1	将多个文件+目录压成zip包
unzip xxx.zip	解压zip包
tar -cvf xxx.tar file	创建非压缩tar包
tar -cvf xxx.tar file1 file2 dir1	将多个文件+目录打tar包
tar -tf xxx.tar	查看tar包的内容
tar -xvf xxx.tar	解压tar包
tar -xvf xxx.tar -C /dir	将tar包解压至指定目录
tar -cvfj xxx.tar.bz2 dir	创建bz2压缩包
tar -jxvf xxx.tar.bz2	解压bz2压缩包
tar -cvfz xxx.tar.gz dir	创建gzip压缩包
tar -zxvf xxx.tar.gz	解压gzip压缩包
bunzip2 xxx.bz2	解压bz2压缩包
bzip2 filename	压缩文件
gunzip xxx.gz	解压gzip压缩包
gzip filename	压缩文件
gzip -9 filename	最大程度压缩

# RPM包管理命令

常用命令	作用
rpm -qa	查看已安装的rpm包
rpm -q pkg_name	查询某个rpm包
rpm -q --whatprovides xxx	显示xxx功能是由哪个包提供的
rpm -q --whatrequires xxx	显示xxx功能被哪个程序包依赖的
rpm -q --changelog xxx	显示xxx包的更改记录
rpm -qi pkg_name	查看一个包的详细信息
rpm -qd pkg_name	查询一个包所提供的文档
rpm -qc pkg_name	查看已安装rpm包提供的配置文件
rpm -ql pkg_name	查看一个包安装了哪些文件
rpm -qf filename	查看某个文件属于哪个包
rpm -qR pkg_name	查询包的依赖关系
rpm -ivh xxx.rpm	安装rpm包
rpm -ivh --test xxx.rpm	测试安装rpm包
rpm -ivh --nodeps xxx.rpm	安装rpm包时忽略依赖关系
rpm -e xxx	卸载程序包
rpm -Fvh pkg_name	升级确定已安装的rpm包
rpm -Uvh pkg_name	升级rpm包(若未安装则会安装)
rpm -V pkg_name	RPM包详细信息校验

# YUM包管理命令

常用命令	作用
yum repolist enabled	显示可用的源仓库
yum search pkg_name	搜索软件包
yum install pkg_name	下载并安装软件包
yum install --downloadonly pkg_name	只下载不安装
yum list	显示所有程序包
yum list installed	查看当前系统已安装包
yum list updates	查看可以更新的包列表
yum check-update	查看可升级的软件包
yum update	更新所有软件包
yum update pkg_name	升级指定软件包
yum deplist pkg_name	列出软件包依赖关系
yum remove pkg_name	删除软件包
yum clean all	清除缓存
yum clean packages	清除缓存的软件包
yum clean headers	清除缓存的header

# DPKG包管理命令

常用命令	作用
dpkg -c xxx.deb	列出deb包的内容
dpkg -i xxx.deb	安装/更新deb包
dpkg -r pkg_name	移除deb包
dpkg -P pkg_name	移除deb包(不留配置)
dpkg -l	查看系统中已安装deb包
dpkg -l pkg_name	显示包的大致信息
dpkg -L pkg_name	查看deb包安装的文件
dpkg -s pkg_name	查看包的详细信息
dpkg –unpack xxx.deb	解开deb包的内容

# APT软件工具

常用命令	作用
apt-cache search pkg_name	搜索程序包
apt-cache show pkg_name	获取包的概览信息
apt-get install pkg_name	安装/升级软件包
apt-get purge pkg_name	卸载软件 (包括配置)
apt-get remove pkg_name	卸载软件 (不包括配置)
apt-get update	更新包索引信息
apt-get upgrade	更新已安装软件包
apt-get clean	清理缓存

# 花 累

该PDF文档会持续更新，有新东西再往里加。另外由于时间仓促和个人精力有限，难免会有疏漏和不当的地方，还望多多谅解。

另外，联系交流，可直接微信私信，一起进步提高 ↓



本文档 GitHub <https://github.com/rd2coding/Road2Coding> 已经收录，里面有我整理的**6大编程方向的自学路线+知识点大梳理**、**我的简历**、**面试考点**、**几本硬核pdf笔记**，以及**我的程序员人生**，欢迎star。

另附：其他干货文档笔记下载（点击标题链接可跳转）：

- 数据结构和算法刷题笔记.pdf下载
- LeetCode算法刷题C/C++版答案pdf下载
- LeetCode算法刷题Java版答案pdf下载
- 找工作简历模板集(word格式)下载
- Java基础核心知识大总结.pdf 下载
- C/C++常见面试题（含答案）下载
- 设计模式学习笔记.pdf下载
- Java后端开发学习路线+知识点总结

- 前端开发学习路线+知识点总结
- 大数据开发学习路线+知识点总结
- C/C++(后台)学习路线+知识点总结
- 嵌入式开发学习路线+知识点总结

本文 GitHub <https://github.com/rd2coding/Road2Coding> 已经收录，里面有我整理的**6大编程方向(岗位)的自学路线+知识点大梳理、面试考点、我的简历、几本硬核pdf笔记**，以及**我的程序员人生**。

小伙伴们 大家好！

之前很多朋友问过我，作为一个UP主和一个程序员会用到哪些软件和工具。

正好趁着我这次开发机转移，在新机上安装各种开发环境和软件，就这个机会跟大家分享一下好了！小伙伴们有什么好用的软件或工具也欢迎评论区安利一下，大家一起交流进步，我就先抛砖引玉了。

这次装机，我把我的所有常用软件分为了**5大类**：

- 社交/娱乐/上网
- 办公软件
- 效率和美化
- UP主必备
- 程序员生产力工具

接下来一一介绍。

---

## \*0X01.\*社交/娱乐/上网

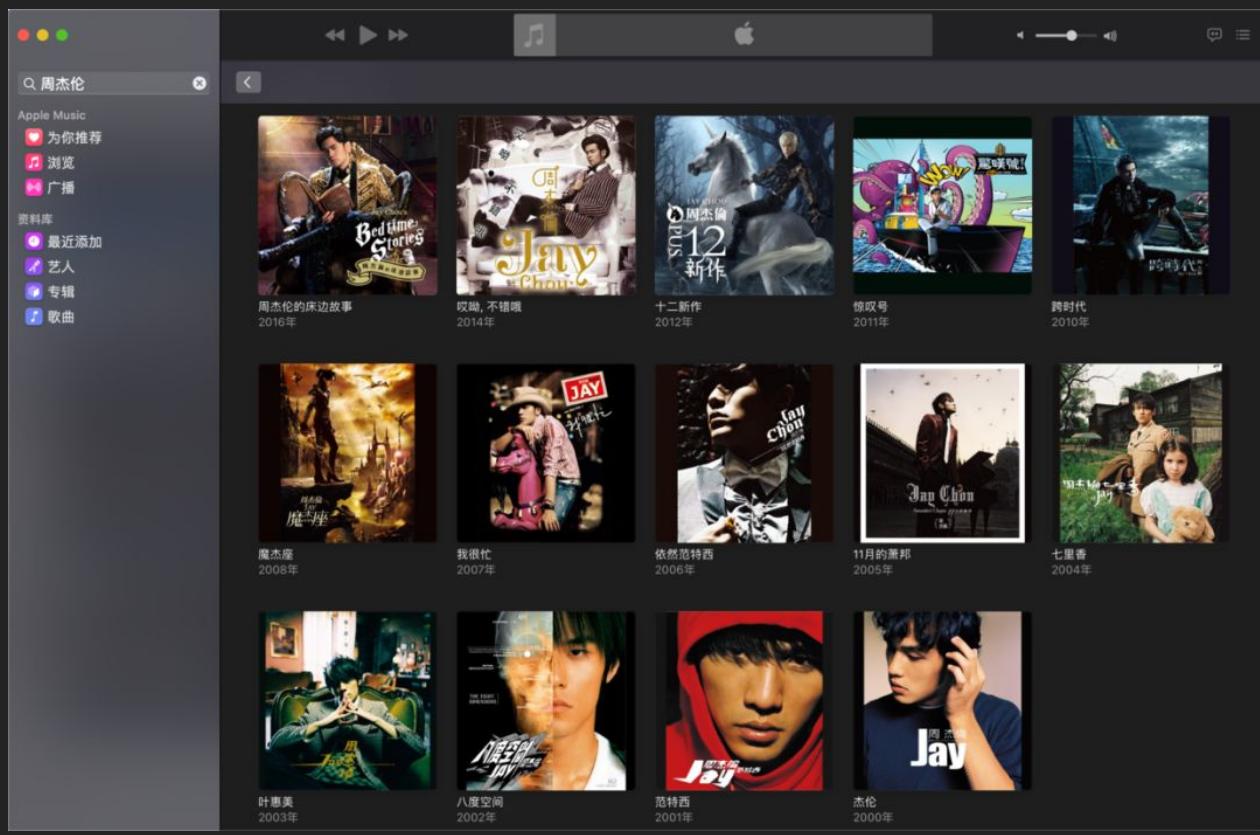


社交、娱乐、上网等软件估计大家都大同小异，各有各的使用习惯和取舍。

这地方我必须要提的，可以说极大提升了我休闲时间幸福感的软件那就是：

## Apple Music

不得不说，正版专辑真的挺香，音质也非常好，我沉迷于其中无法自拔！



## 0X02.办公软件

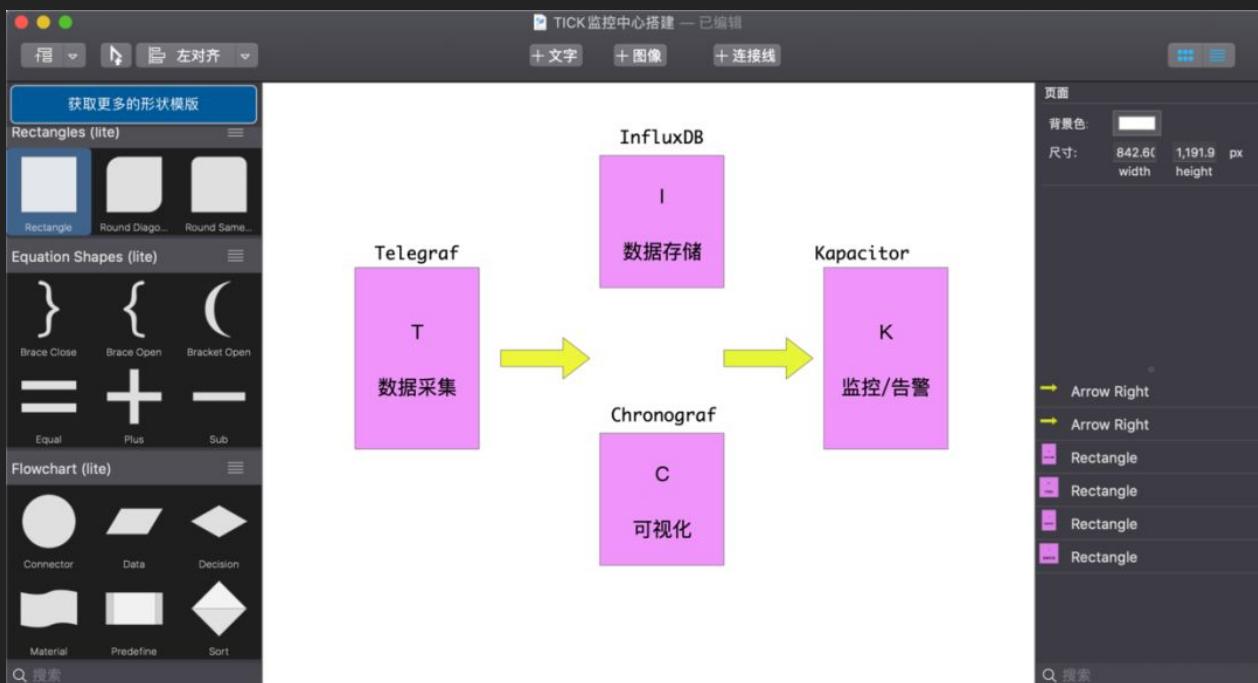
### 1、微软Office家族

这个我想不用多介绍了

### 2、PDF Professional

一直以来用得非常顺手的一款PDF阅读器，不管是编辑、做笔记、目录、书签、合并/分离、转换PDF文件都非常方便。

### 3、Flowchart Designer



平时写技术博客，文中的原理图、流程图等都是用它来完成的，基本满意

## 4、有道云笔记

本项目内容为《Spring Boot实战合集》系列文章，代码已开源，并持续保持更新。

另外，所有内容都在我的个人微信公众号 **CodeSheep** 最先推出，有问题也欢迎大家来这里提问交流

### 实战经验相关

- Spring Boot工程集成全局唯一ID生成器 UidGenerator
- Spring Boot 工程集成全局唯一ID生成器 Vesta
- Spring Boot优雅编码之：Lombok加持
- Spring Boot应用Docker化
- Spring Boot热部署加持
- 基于Spring Boot实现图片上传/加水印一把梭操作
- 从Spring Boot到SpringMVC
- 自然语言处理工具包 HanLP在 Spring Boot中的应用
- Spring Boot应用部署于外置Tomcat容器
- 初探 Kotlin + Spring Boot联合编程

### 数据库/缓存相关

- Guava Cache本地缓存在 Spring Boot应用中的实践

作为一个跨平台、跨终端的笔记软件，还算非常好用的，我的所有技术文章基本都放在里面。

## 5、百度云网盘

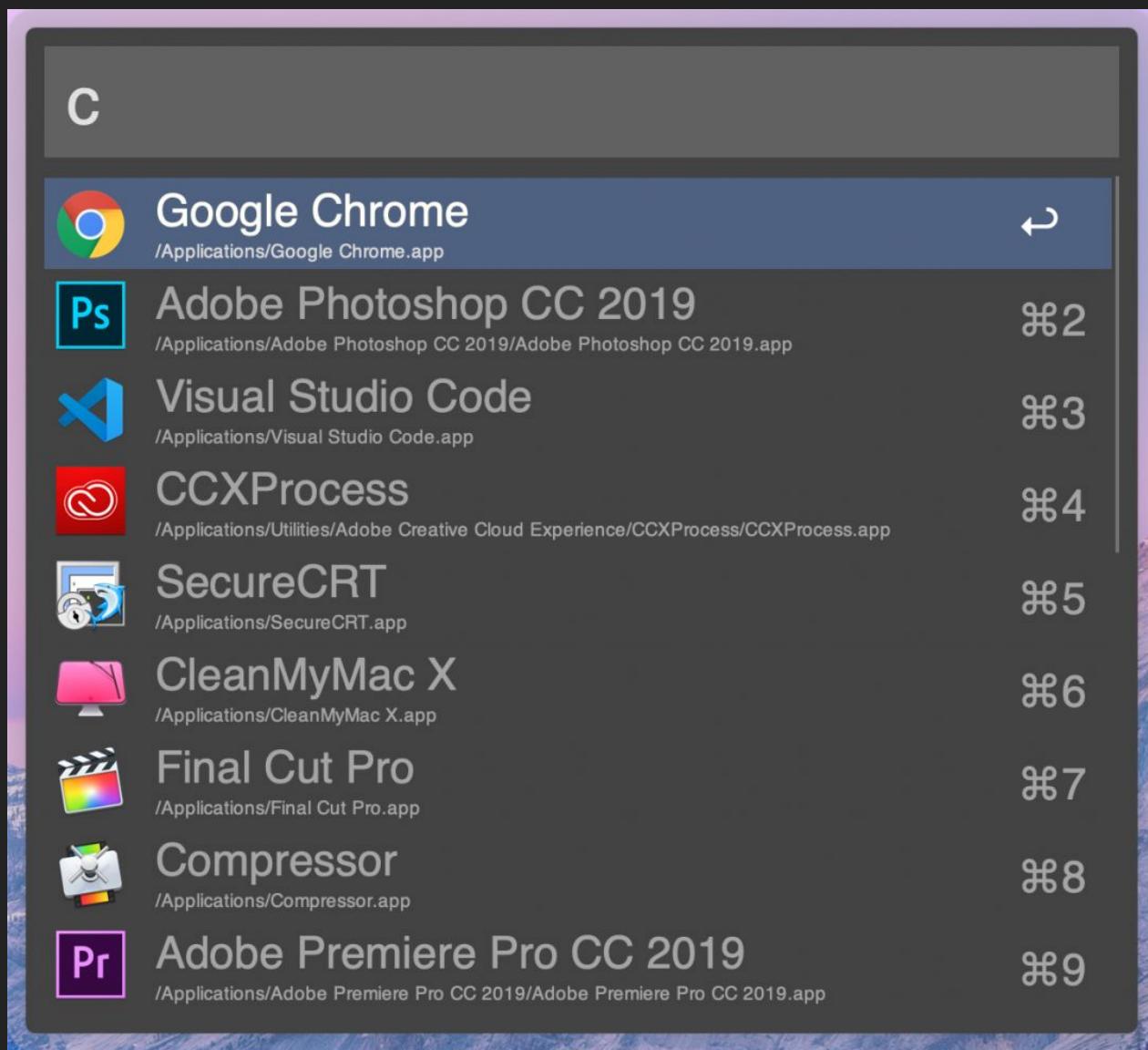
百度你6，我充会员还不行嘛！

---

## 0X03.效率和美化



1、Alfred



用Mac的没有不用Alfred的吧，我想应该不用多介绍了，是一个高效的启动器，谁用谁知道。

## 2、Magnet

## Magnet 偏好设置

左  ⌘← ⌘↖×

右  ⌘→ ⌘↗×

上  ⌘↑ ⌘↑↗×

下  ⌘↓ ⌘↓↗×

左上  ⌘↖U ⌘↖U×

右上  ⌘↖I ⌘↖I×

左下  ⌘↖J ⌘↖J×

右下  ⌘↖K ⌘↖K×

左 1/3  ⌘↖D ⌘↖D×

左 2/3  ⌘↖E ⌘↖E×

中 1/3  ⌘↖F ⌘↖F×

右 2/3  ⌘↖T ⌘↖T×

右 1/3  ⌘↖G ⌘↖G×

下一个显示  ⌘→ ⌘→×

上一个显示  ⌘← ⌘←×

最大化  ⌘↔ ⌘↔×

居中  ⌘C ⌘C×

还原  ⌘↶ ⌘↶×

登录时启动

拖动以对齐窗口

取消对齐，回到原始尺寸

用Windows系统的小伙伴都知道，可以自由拖动窗口在屏幕周边进行吸附从而分屏，但Mac默认是没有这个功能的。Magnet是我用的一个还不错的分屏软件，直接支持窗口拖动吸附，非常方便。

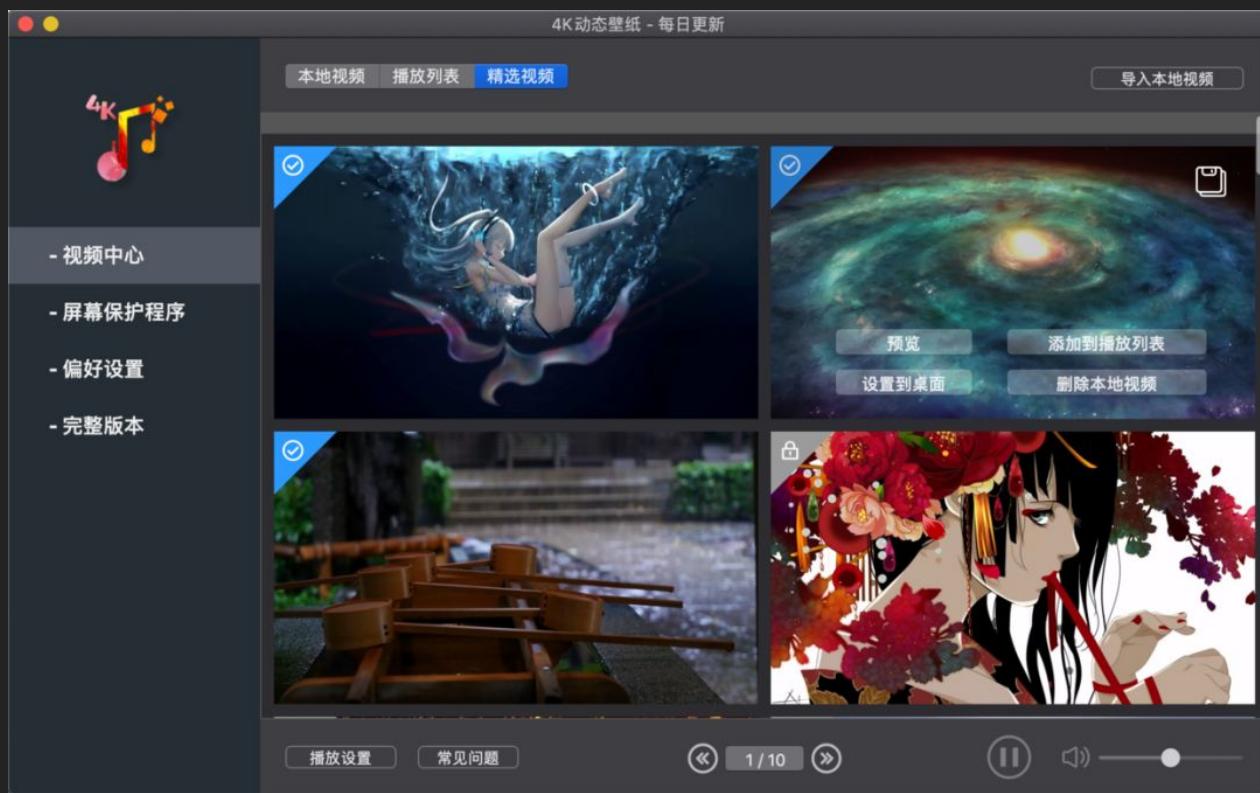
### 3、CleanMyMac X



非常好用的系统管理软件，系统清理、程序卸载、健康监控、文件粉碎都能完成，关键是界面真的很舒适。

### 4、4K Live Wallpaper

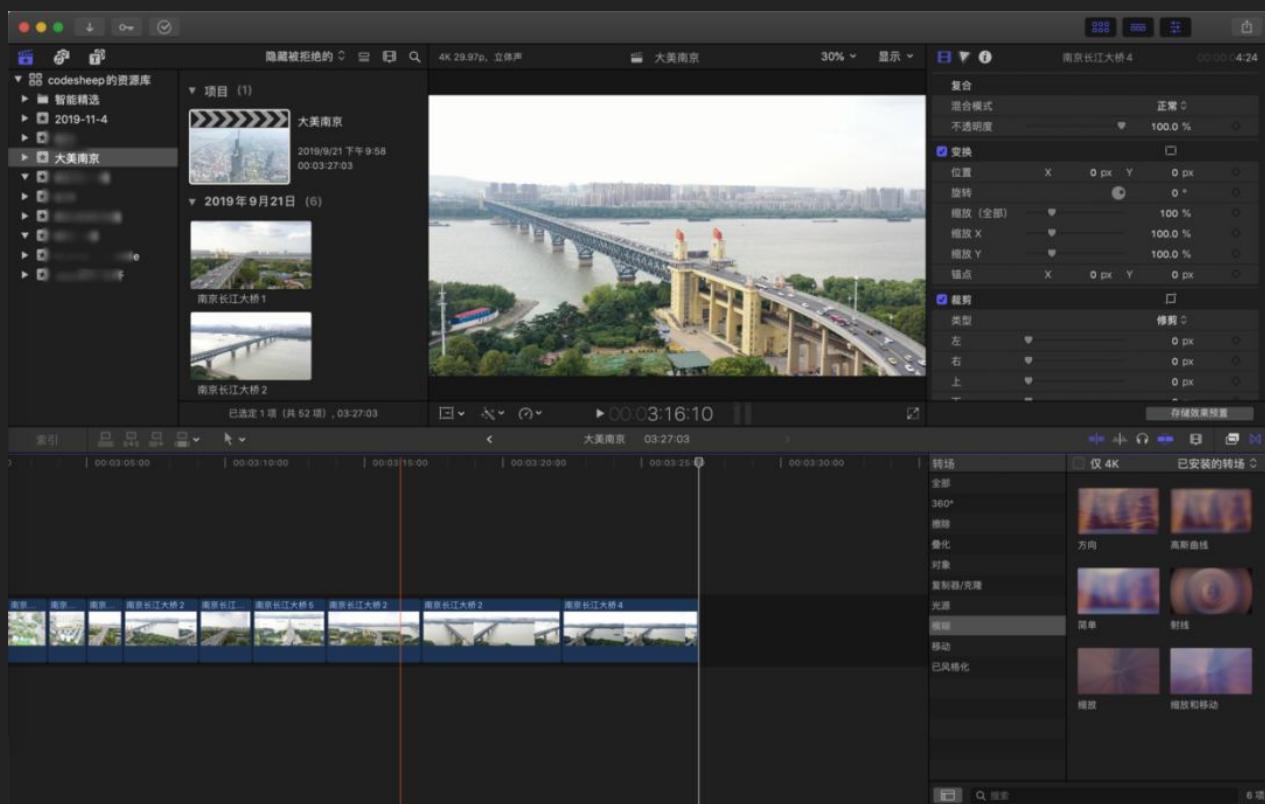
4K动态壁纸软件，纯粹为了酷炫的效果，非常适合视网膜屏幕



# \*0X04.\*UP主必备



## 1、Final Cut Pro X



作为一个程序员UP猪，我频道里所有的视频大部分都是用Final Cut Pro X剪辑完成的，着实的UP主生产力工具。

## 2、Premiere



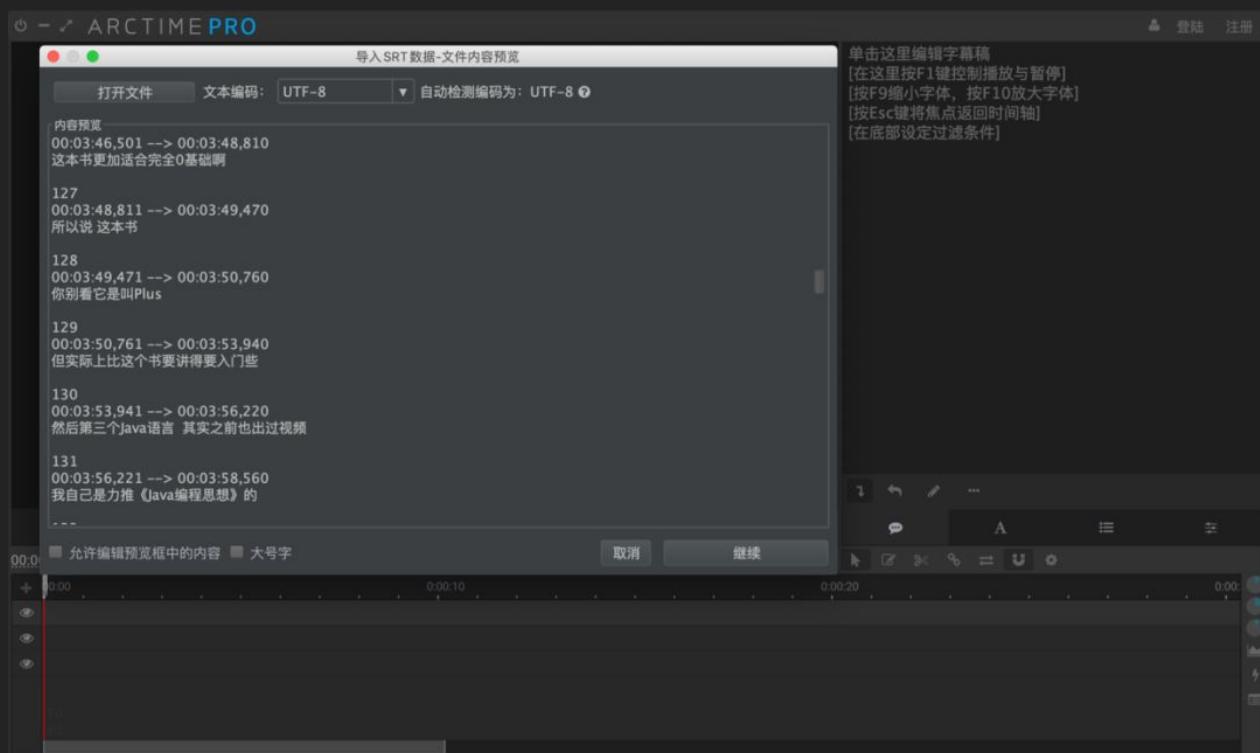
除了上面的Final Cut Pro X可以剪视频，Premiere则更加专业和复杂，强大肯定是不用说了，关键有个问题是这玩意经常性地崩溃。。。

### 3、DaVinci Resolve



达芬奇数字调色软件，我正好最近偶尔学一学，挺高深的，目前仍处于踩坑摸底阶段。

## 4、ArcTime Pro



给视频批量做字幕的必备软件。

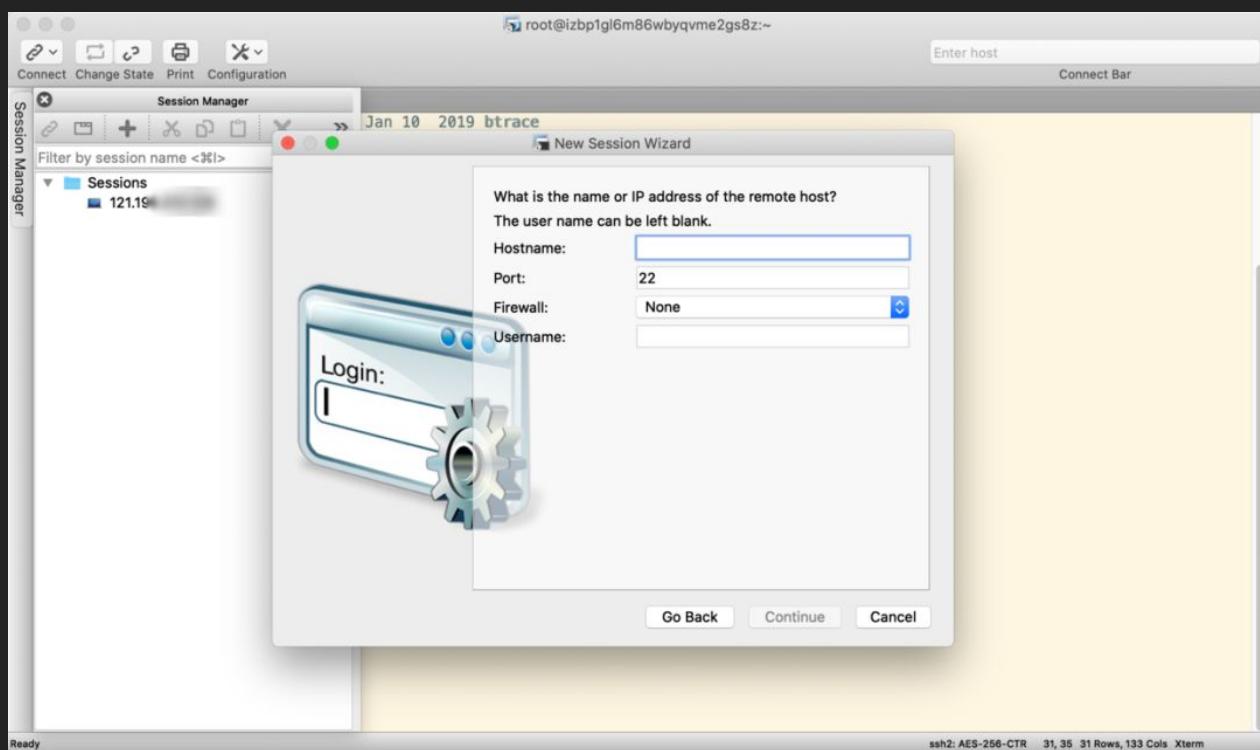
## 5、Ps 和 Lr

Photoshop 和 Lightroom应该也不用多介绍了，处理图片，做图片封面，我用的都是它们。

# 0X05.程序员生产力工具

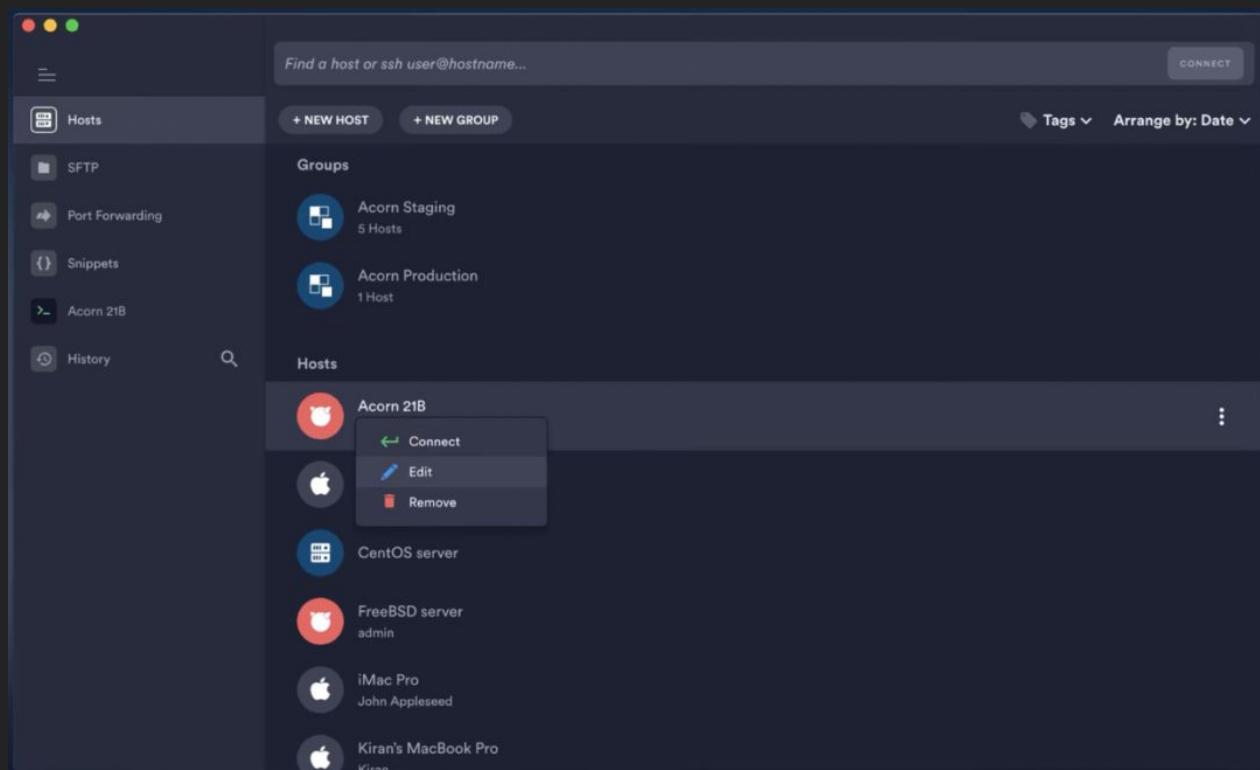


## 1、SecureCRT



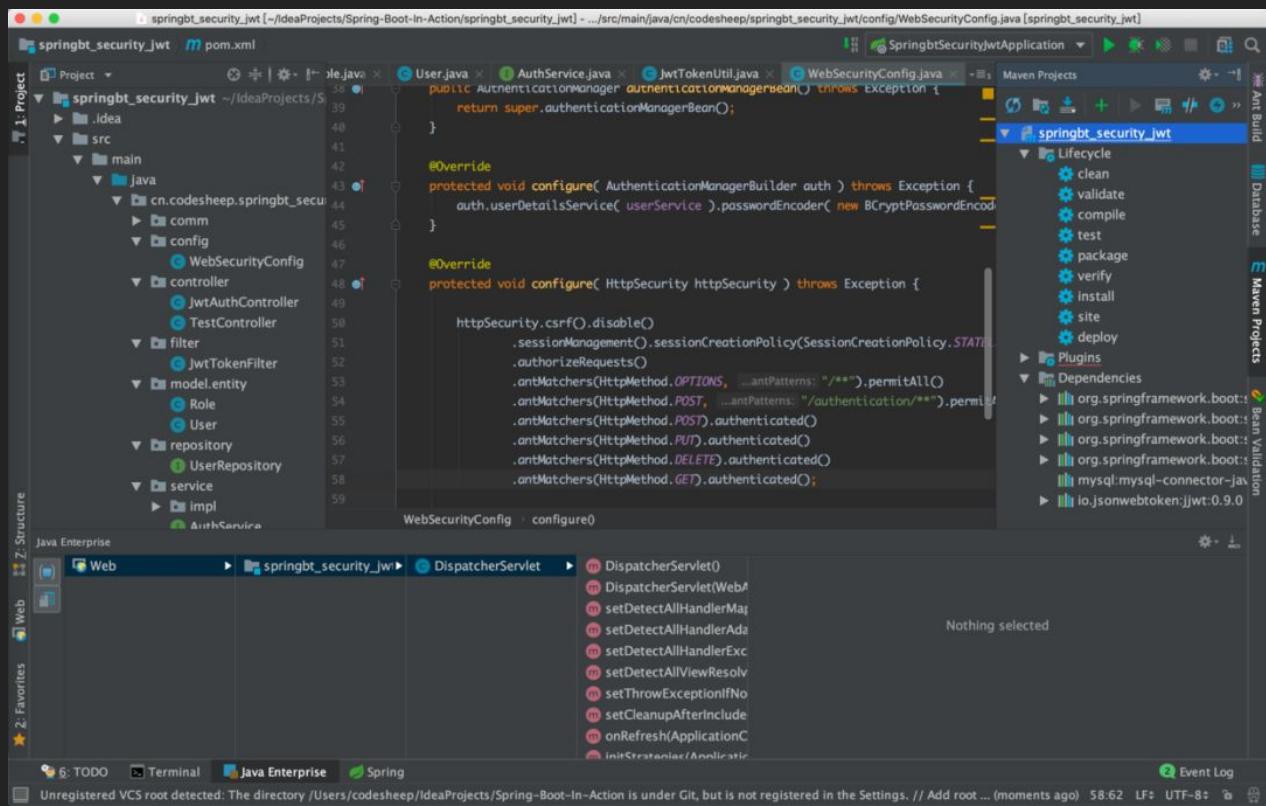
SecureCRT是一款SSH远程连接服务器的终端软件，同时也支持[Telnet]和[rlogin]等多种协议，我已经用了多年了，形影不离。

## 2、Termius



Termius和上面的SecureCRT一样，也是一个SSH终端软件，功能也非常强大，App Store里即可直接安装，很方便。

### 3、IntelliJ IDEA



这软件可真是吃饭家伙。作为一个Java后端开发，所有的Java代码基本都是在里面写的，不愧是Java领域最炫酷IDE！

### 4、Chrome浏览器

这个我想不用多介绍了，程序员必备浏览器。

### 5、iTerm2

Mac OS 自带的终端其实也可以用，只不过有时候不太方便，定制自由度也不够。iTerm2则是一个可以完全替代系统自带终端的神器，尤其配合 `zsh` 这个 Shell 来使用，可以说非常地舒适。

## 6、Visual Studio Code

这个应该也不用多介绍，微软出品，跨所有平台的文本编辑器，我一般也就临时用来看看代码，写写 Markdown 格式的文章，当然它的功能远不止这些。

## 7、VMware Fusion

## 选择安装方法



从光盘或映像中安装

将您的 ISO 文件拖到此处以开始安装



迁移您的 PC



从恢复分区中安装 macOS



导入现有虚拟机



从 Boot Camp 安装



创建自定虚拟机



在远程服务器上创建虚拟机

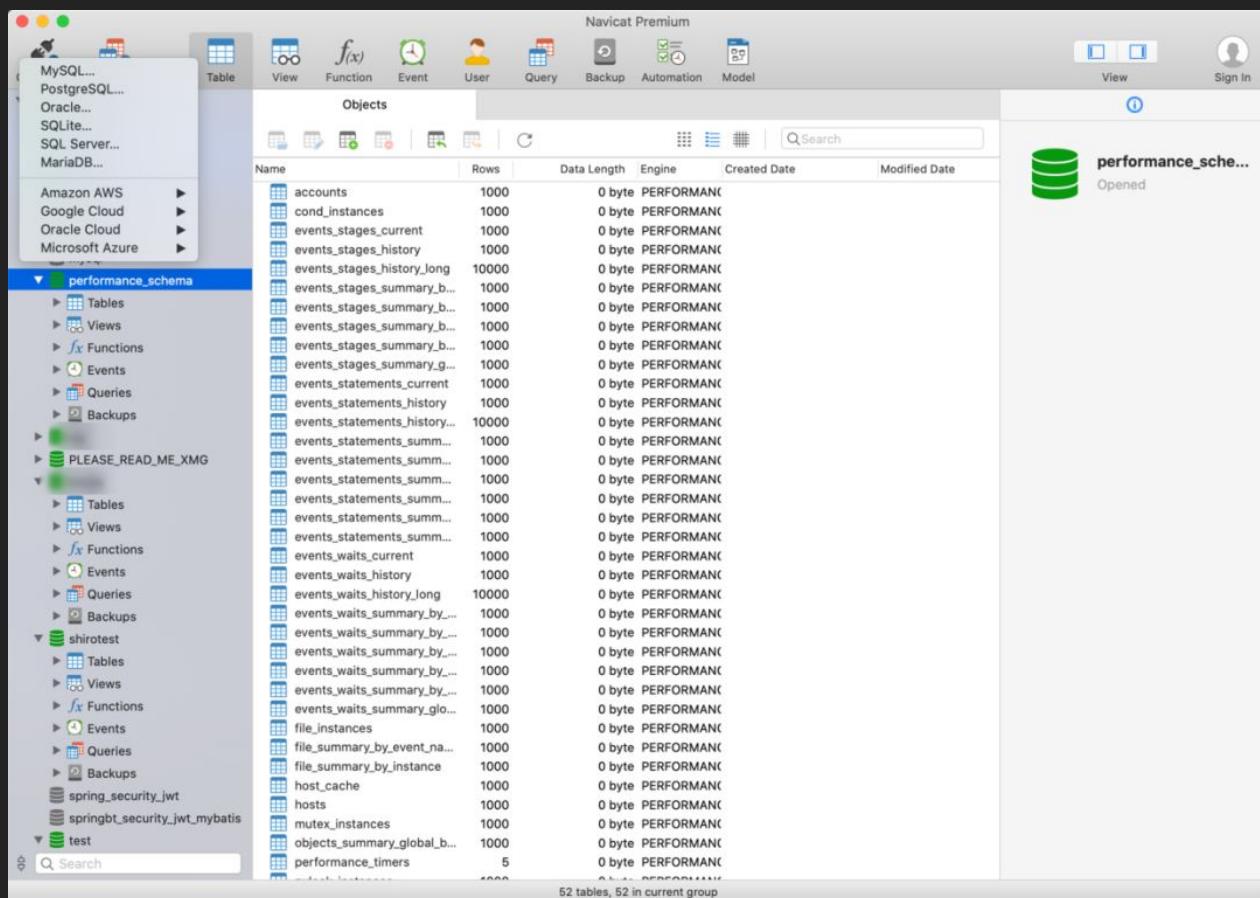


取消

继续

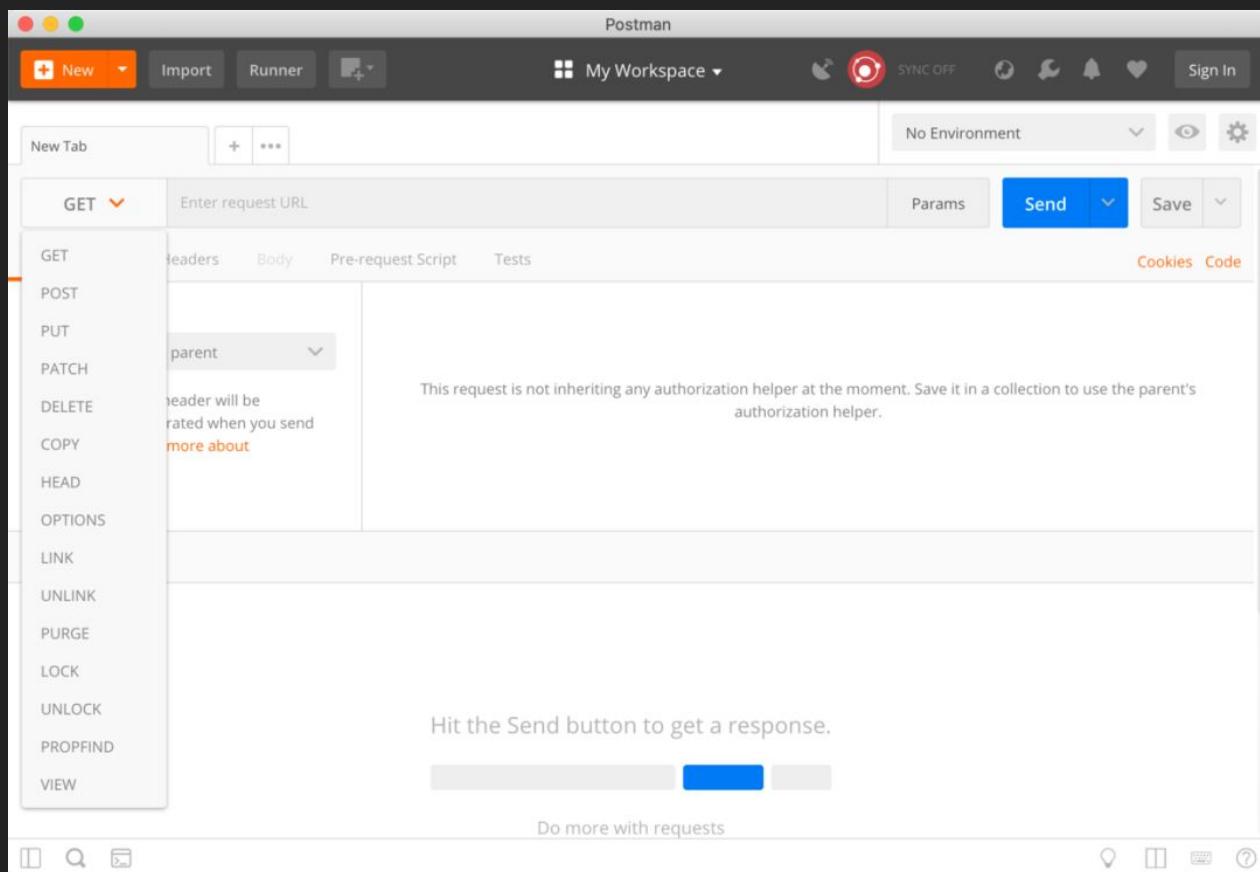
VMware Fusion是一款可以媲美Parallels Desktop的虚拟机软件，我一直用的这个。

### 8、Navicat



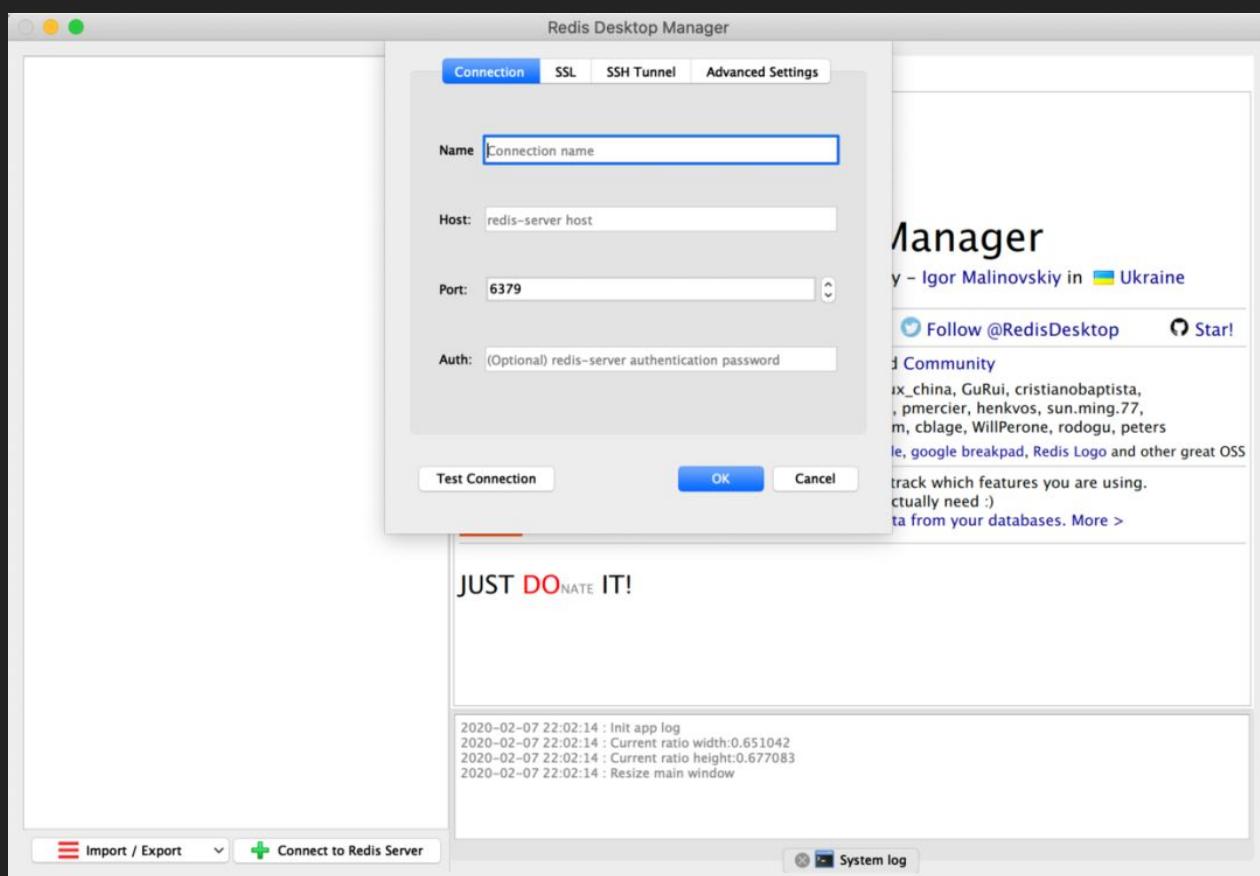
我重度依赖的一款数据库可视化软件，支持多种数据库，支持大部分在现今数据库管理系统中使用的功能，包括存储过程、事件、触发器、函数、视图等。

## 9、Postman



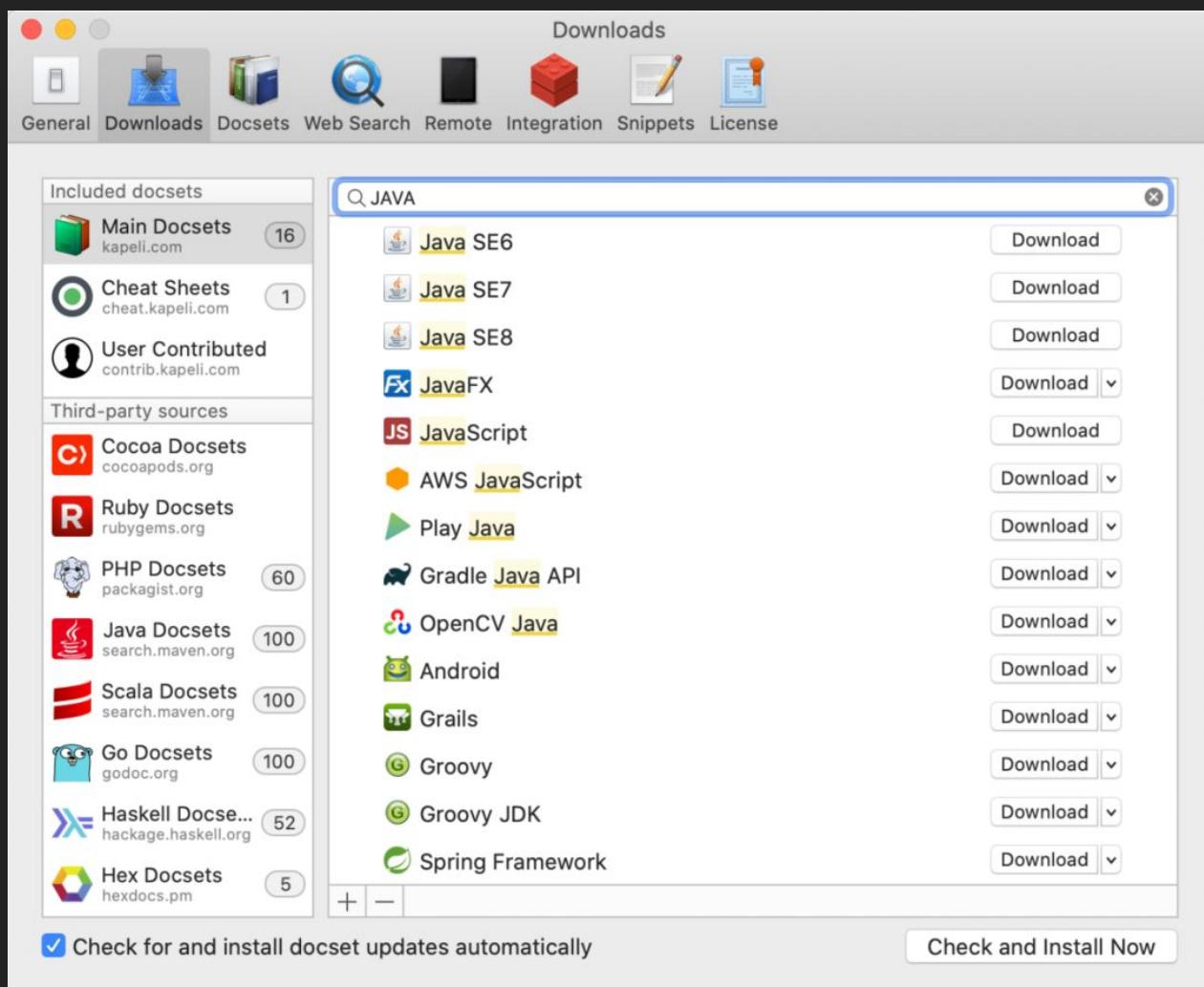
作为一个后端开发，API接口的测试少不了它。

## 10、Redis Desktop Manager



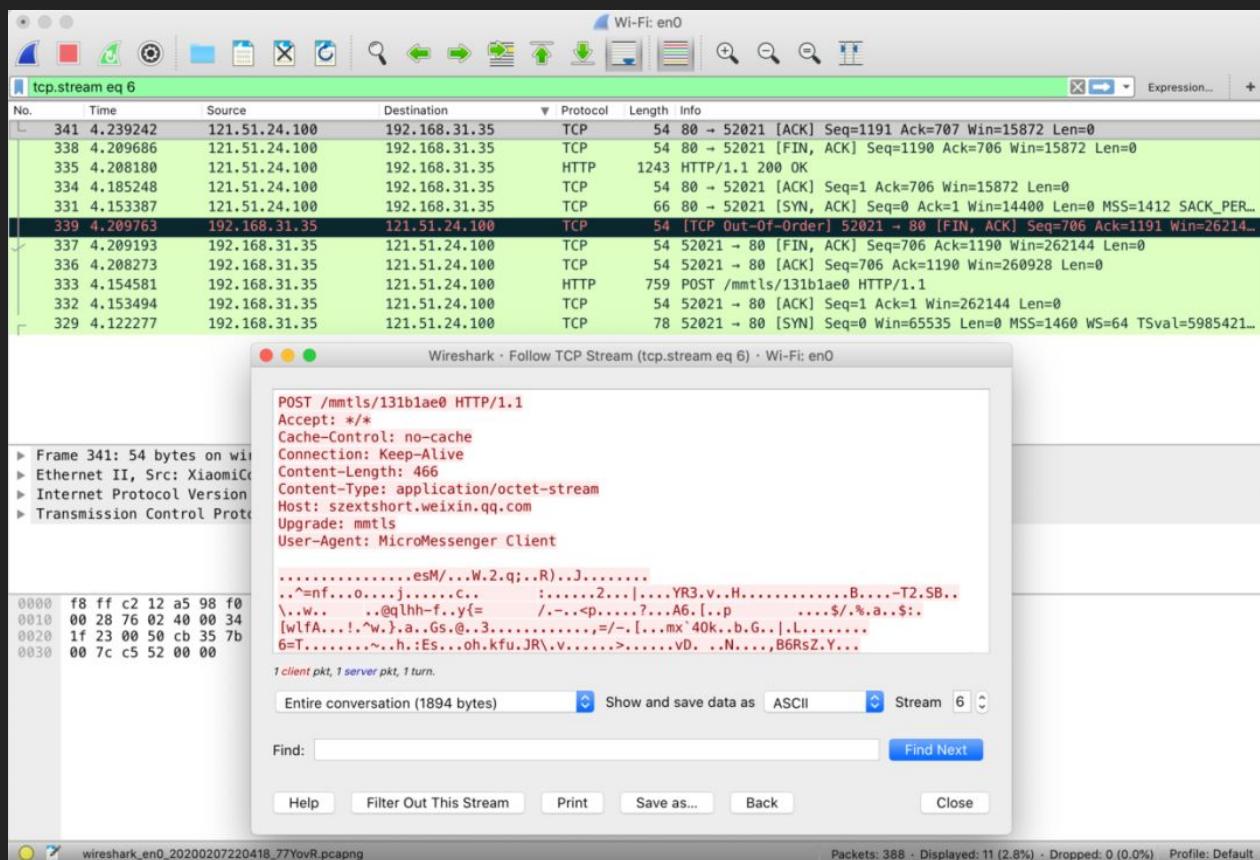
RDM和上面的 **Navicat** 差不多，是一款 Redis 缓存的可视化软件，平时可视化操作 Redis 少不了它。

## 11、Dash



Dash既是一个强大的程序员开发文档管理器、也是一个方便的代码片段管理工具，可以说解决了程序员的两大痛点。

## 12、Wireshark



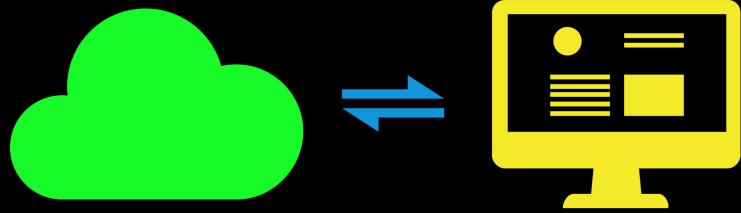
比较好用的抓包工具、看报文的工具。有了这个软件，TCP/IP协议学起来可以说非常具象了，有啥不懂的，直接抓个包分析一下报文，一目了然。

## 写在最后

联系我，直接微信扫码，给我私信即可 ↓



本文 GitHub <https://github.com/rd2coding/Road2Coding> 已经收录，里面有我整理的**6大编程方向的自学路线+知识点大梳理、我的简历、面试考点、几本硬核pdf笔记**，以及**我的程序员人生**，欢迎star。



本文 GitHub <https://github.com/rd2coding/Road2Coding> 已经收录，里面有我整理的**6大编程方向(岗位)的自学路线+知识点大梳理、面试考点、我的简历、几本硬核pdf笔记**，以及我的程序员人生。

小伙伴们大家好，本期再托一次家底，和大家分享十几个自己日常工作学习过程中的常用在线工具和网站，希望能有帮助。

也欢迎小伙伴们分享和安利出你们觉得有用的工具，不要停啊。



# 啊！不要停止对知识的深索

注：本文 Github 开源项目：[github.com/hansonwang99/JavaCollection](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...

## 在线绘图神器

很多小伙伴咨询说博客文章里的技术图怎么画出来的，这里透个底，大部分都是通过 processon 画出来的，在线画图十分方便，几乎可以画出你想要的任何技术图，包括：流程图、思维导图、原型图、UML 图、网络拓扑图、组织结构图等等。

The screenshot shows the ProcessOn web application interface. At the top, there's a navigation bar with 'ProcessOn - 我的文件' (My Files), a search bar, and user profile icons. Below the navigation is a header with 'ProcessOn' and tabs for '团队' (Team), '个人' (Personal), '推荐' (Recommend), and '模版' (Template). On the left, there's a sidebar with '个人文件' (Personal Files) sections for '我的文件' (My Files), '最近修改' (Recently Modified), '与我协作' (Collaborating with me), '我的收藏' (My Favorites), and '回收站' (Recycle Bin). The main area displays a grid of thumbnail previews for various diagrams and files, including '博文配图草稿', '技术图库草稿', '技术图库成品', '临时', '实验集群架构图', '书单目录', '思维导图', '未命名文件', '未命名文件', '学习路线', '学习路线-备份1', '学习路线-备份2', '鱼骨图2', 'Elasticsearch 集群', 'Java编程思想学习录...', 'Java后端开发学习路线', 'Java容器继承关系', 'k8s集群', 'Linux发行版', and 'SheepLog系统架构'. At the bottom, there are links for '个人版', '查看特权', '帮助手册', '反馈', '服务条款', '关于我们', and a red banner with the URL 'github.com/hansonwang99/JavaCollection 开源项目已收录 By CodeSheep'.

- <https://processon.com/>

# 代码图片神器

很多小伙伴常问，公众号文章里的类似这样的代码图片是如何做出来的：



这就得用到这个专门做代码图片的神器工具网站了，比如这里的dute：

在线代码截图，源代码生成图片

dute.org/code-snapshot

By CodeSheep

## 在线代码截图

在线代码截图工具，根据输入的源代码，可以生成漂亮的代码图片。本工具可自动识别输入的代码属于哪种程序语言，包括 C、C++、Java、Go、Python、PHP、JavaScript、TypeScript、CSS、HTML 等几十种开发语言，并提供了若干高亮主题，以及相关图片生成选项，以生成不同风格的源码截图。生成的图片可用于公众号文章、博客、微博、论坛等需要展示代码的场合。

高亮主题 Seti 程序语言 Java

```
1 package cn.codesheep;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         System.out.println("自学编程之路：学习/面试指南+资源分享+技术文章");
7         System.out.println("github.com/hansonwang99/JavaCollection");
8         System.out.println("By CodeSheep");
9     }
10 }
```

背景色

代码窗口 显示窗口按钮  
使用圆角效果  
显示阴影效果

代码字体 Consolas

字体大小 16px

编辑器 显示行号  
添加水印

保存代码截图

- <https://www.dute.org/code-snapshot>

## 在线任务管理工具

日常记录需求、灵感和待办事项没有称手的小工具？想做详细的学习任务规划和管理没有好地方？我想 Trello 应该能够帮到你。



从图中可以看出，这种泳道配合任务卡片的拖动方式使用起来还是非常可以的。

事实上Trello的功能远不止此，它甚至是一个非常好用的团队协作工具，而且Trello均配有客户端的App，跨平台同步使用非常方便。

- <https://trello.com/>

## 图片超分辨率神器

如果现在手里有一张低分辨率的小图，如果让它变成高清大图呢，就像这样：

原图/a small & noisy image



用 bigjpg.com 8x倍放大，降噪程度高

Using bigjpg.com 8x Upscaling and High Noise

Using bigjpg.com for upscaling and high noise

## Reduction



放大前后对比

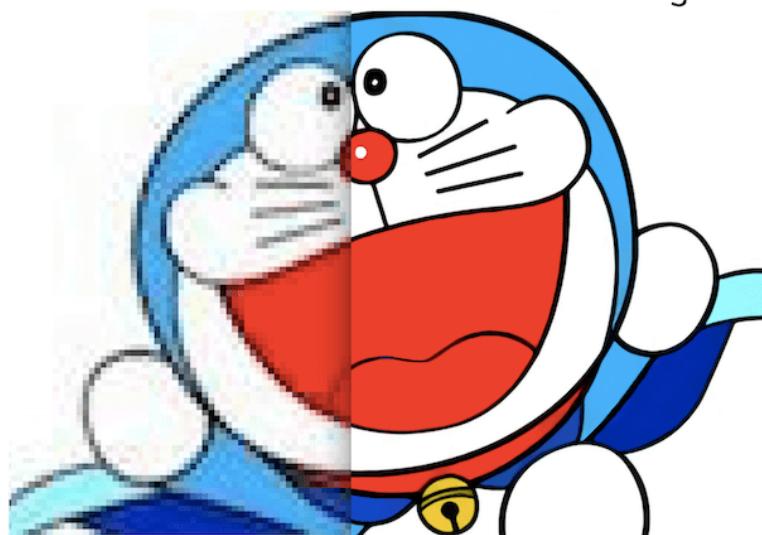
before and after enlarge

原图

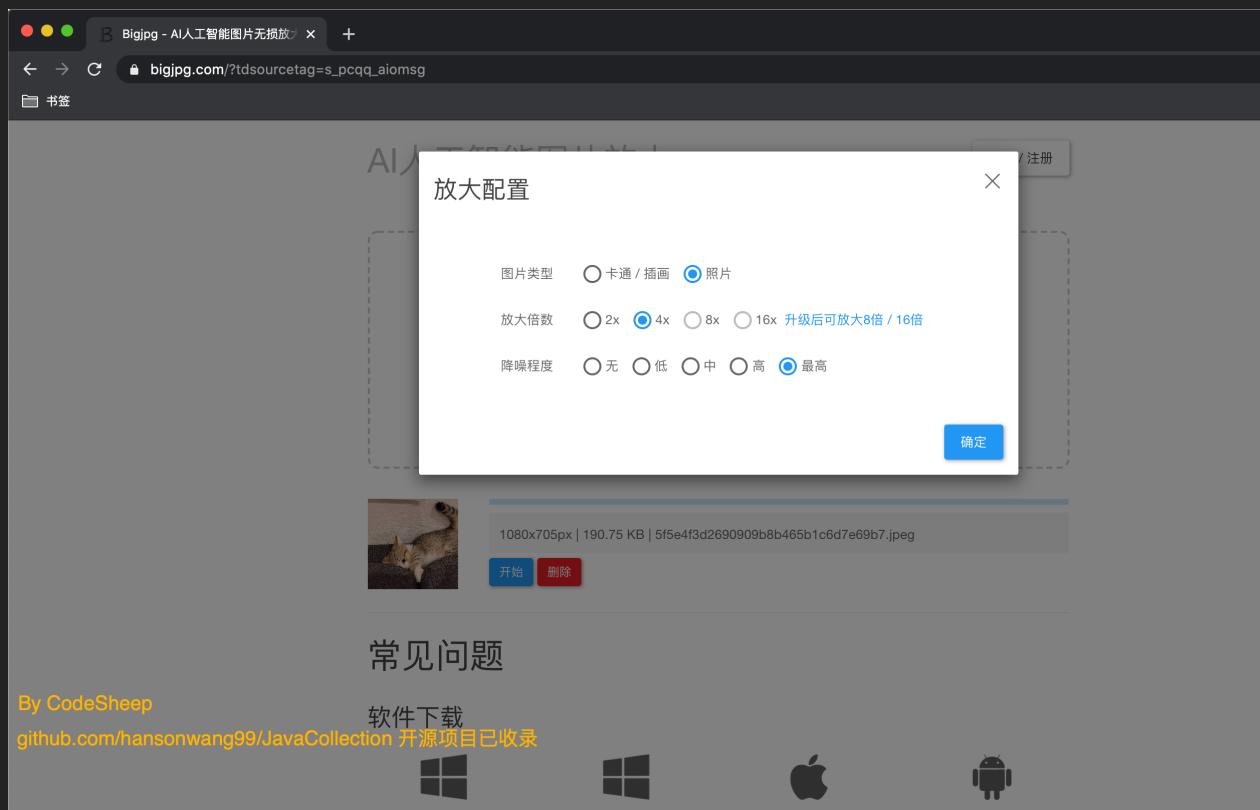
before

bigjpg.com 无损放大后

after enlarge



那么就可以借助于bigpig这款在线图片超分辨率神器，它使用的是人工智能方面的深度学习技术进行图片超分辨率，挺有特色的。



## 在线PS神器

uupoop是一款好用的在线图片处理工具站，堪称在线PS神器，支持编辑psd、xd、sketch等多种格式的文件处理，非常强大。除此之外，上面还集成了各种封面制作、海报制作和GIF动图制作等在线工具，爱了。



- <https://www.uupoop.com/>

## 在线音频剪辑神器

这款在线工具网站可以让您直接在浏览器中剪辑音轨，并且支持多种文件格式，有点香。

The screenshot shows a dark-themed online audio cutter interface. At the top, it displays the URL "weixinsyt.com" and the page title "在线音频剪辑器 - 音频剪辑软件". The main area features a waveform visualization of an MP3 file titled "(科学科普) 编程软件和环境大科普.mp3". The waveform is green and spans from 00:00.0 to 06:17.4. Below the waveform, there are controls for playback (play/pause), volume, and trim. A timeline indicates the start at 00:00.0 and end at 06:17.4. To the right, there are buttons for saving ("另存为") and starting the cut ("开始剪辑").

**音频在线剪辑**  
此网站应用是一款在线应用，它可以让您直接在浏览器中剪辑音轨。该应用运行速度快、稳定，支持 300 多种文件格式、淡入淡出功能和铃声质量预设。

**在线剪辑歌曲**  
使用我们的应用，您无需在电脑上安装音频编辑软件。只需在浏览器窗口点击几下，就可以剪辑音轨。您只要上传文件、剪辑需要的部分，然后再保存至硬盘即可。

**创建 iPhone 铃声**  
只要轻轻操作一下，本应用就可以为您的 iPhone 制作铃声，将其剪裁为 40 秒并保存为 m4r 格式，然后通过 iTunes 上传到手机中。

**淡入淡出功能**  
本款应用可以让您平滑淡入淡出音轨。该功能在制作铃声时非常有用。

**操作方便**  
使用该应用无需具备特殊技能。它的使用非常简单：上传文件，使用滑块选择音频片段，然后点击“剪辑”。

**支持所有格式**  
我们的应用现在支持 300 多种不同的格式，以后将陆续增加更多格式支持。

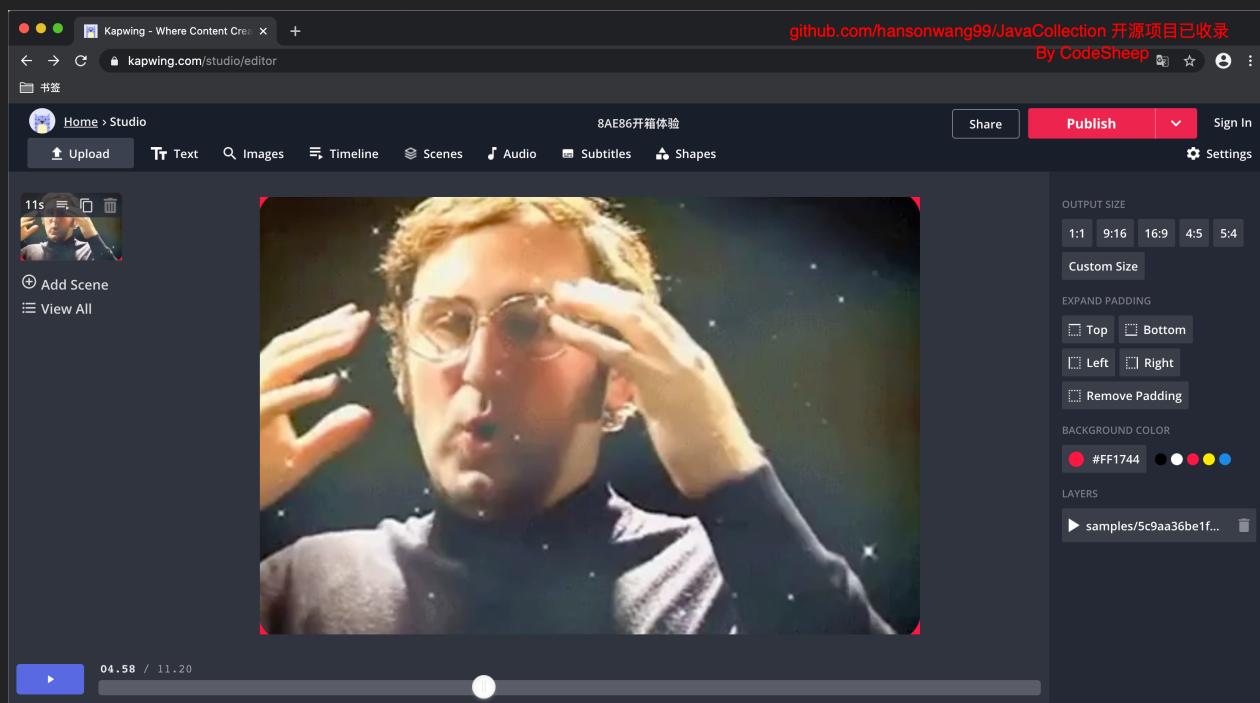
**从视频中提取音频**  
您可以使用此网站功能从视频中提取音频。从电影或 MTV 中获取音轨时，该功能非常有用。

**安全保障**  
只有您能访问上传后的文件。该流程保证全面安全。文件处理完毕后，将自动从我们的服务器中删除。

• <https://www.weixinsyt.com/>

## 在线视频剪辑神器

Kapwing 是一款在线轻量级的视频编辑 studio，除了支持日常的视频编辑之外，还有很多工具集供使用，上手很容易。



- <https://www.kapwing.com/>
- 

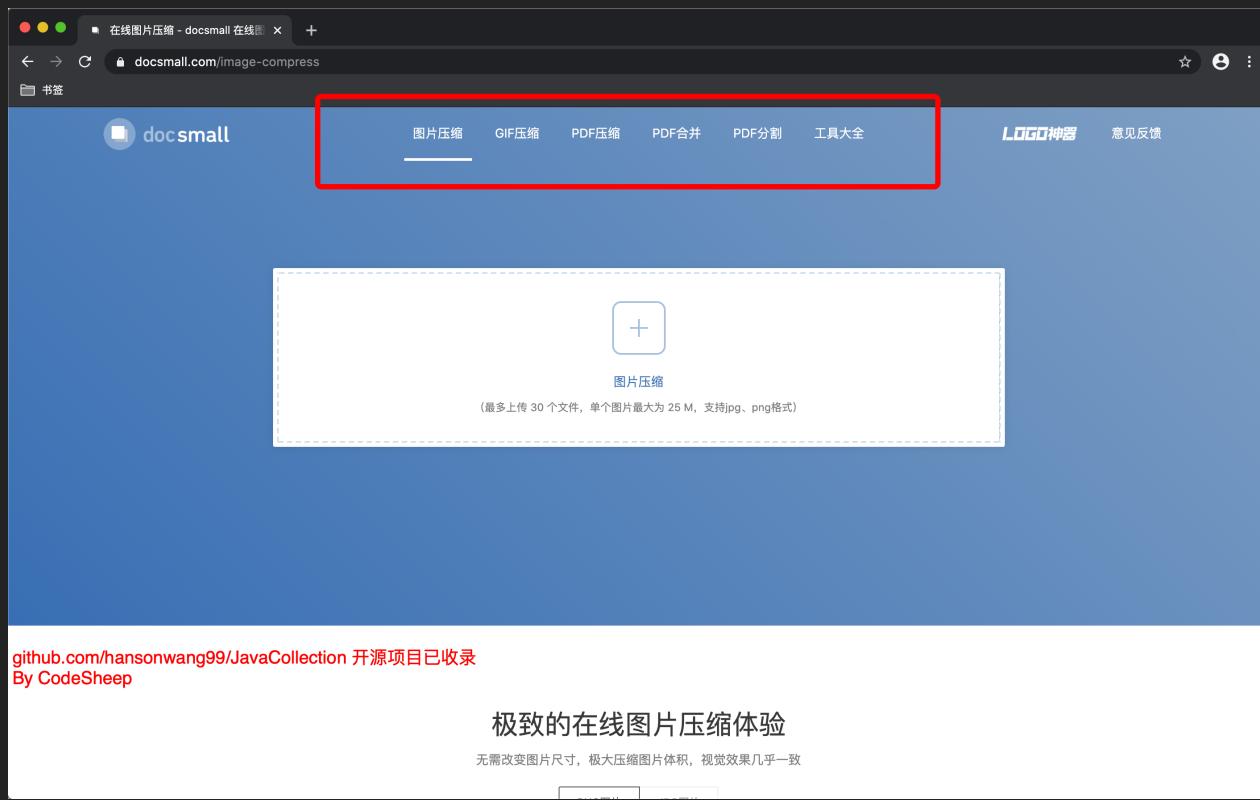
# 在线PDF神器

PDF派，一个包含了20个好用PDF在线工具的网站，各种转换都不在话下。

The screenshot shows the homepage of PDF派 (PDFpai.com). At the top, there's a navigation bar with icons for back, forward, search, and user profile. The URL 'pdfpai.com' is visible. A red banner at the top right reads 'By CodeSheep' and 'github.com/hansonwang99/JavaCollection 开源项目已收录'. The main header 'PDF派' is accompanied by a blue circular logo with a white 'PDF' icon. Below the header, a sub-header says '20个好用的PDF在线工具，完全免费！'. A section titled '把PDF转换成Office, iWork, 图片等其他格式' contains six buttons: 'PDF转Word' (Word icon), 'PDF转Excel' (Excel icon), 'PDF转PPTX' (PowerPoint icon), 'PDF转图片' (Image icon), 'PDF转Pages' (Pages icon), and 'PDF转Numbers' (Numbers icon). Another section titled '把Office文件或图片转换成PDF' contains four buttons: 'Word转换成PDF' (Word icon), 'Excel转换成PDF' (Excel icon), 'PPTX转换成PDF' (PowerPoint icon), and '图片转换成PDF' (Image icon). A third section titled '其他好用的免费PDF工具' contains four buttons: '加密PDF' (Lock icon), '合并PDF' (Merge icon), '添加水印到PDF' (Watermark icon), 'PDF页码' (Page Number icon), '解锁PDF' (Unlock icon), '拆分PDF' (Split icon), '旋转PDF' (Rotate icon), and '压缩PDF' (Compress icon). At the bottom left, there's a bullet point list: • <https://www.pdfpai.com/>

## 在线压缩神器

图片压缩、GIT压缩、PDF压缩等，都可以借助在线网站docsmaill完成。上传、处理、下载，三步搞定，很便利。



- <https://docsmall.com/>

# 在线短链接神器

长链接转短链接工具，而且提供API接口供编程操作，生成的短链接还能在后台检测访问数据。

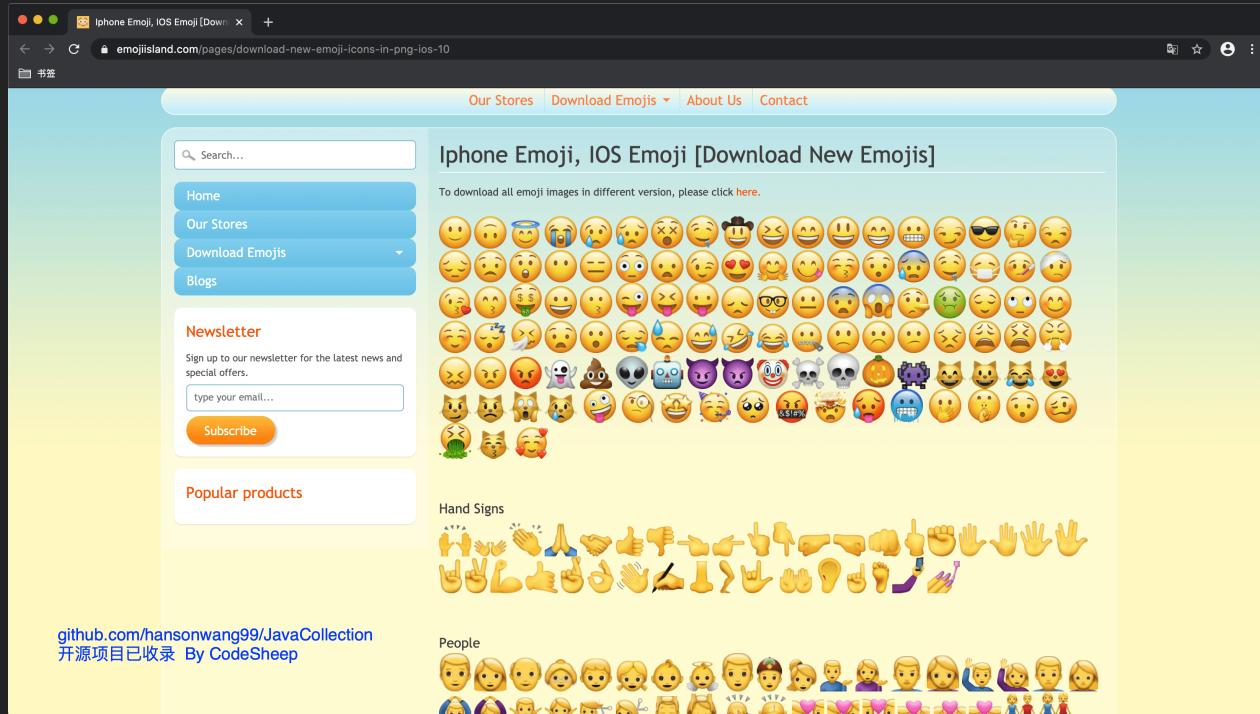


- <http://mrw.so/>
- 

## 在线EMOJI神器

有时候我们做封面或者配图，需要点emoji表情来点缀，那素材哪里找呢？

emojiisland是一个不错的去处，emoji表情成套，PNG格式的可以免费下载。



- <https://emojiisland.com/>

好啦，这次先整理出个人在平时工作和学习工程中常用的工具网站，后面遇到好玩的、有用的也会持续分享，也欢迎小伙伴们分享和安利出你们觉得有用的工具。

## 写在最后

联系我，直接微信扫码，给我私信即可↓



本文 GitHub <https://github.com/rd2coding/Road2Coding> 已经收录，里面有我整理的**6大编程方向的自学路线+知识点大梳理、我的简历、面试考点、几本硬核pdf笔记**，以及**我的程序员人生**，欢迎star。

每天进步一点点，慢一点才能更快