

七的笔记

The Seven Notes

[Star 0](#) [Follow 1](#) [Fork 0](#)

Release 2021-11-23 02:26:39 build failing License Apache 2.0 Words 61,654

Author [Seven](#)



1. 🚀Introduction

1.1. 关于作者 30 字

1.2. 关于6+1 当天更新 147 字

1.3. 如何开始 当天更新 603 字

2. ⚒Backend

2.1. Java

2.1.1. Install 当天更新 177 字

2.1.2. Java问题排查 当天更新 567 字

2.2. JUC

2.2.1. JUC笔记 当天更新 467 字

2.3. JVM

2.3.1. JVM 当天更新 713 字

2.3.2. JVM笔记 当天更新 210 字

2.4. Nginx

2.4.1. Install 当天更新 382 字

2.4.2. nginx.conf 当天更新 392 字

2.4.3. Commands 87 字

2.5. ZooKeeper

2.5.1. Install 当天更新 427 字

2.5.2. ZK笔记 当天更新 155 字

2.6. SysIO

2.6.1. IO笔记 当天更新 616 字

3. ⚓Frontend

3.1. NPM 当天更新 529 字

3.2. Yarn 当天更新 120 字

3.3. VuePress 当天更新 284 字



4. Database

4.1. MySQL

4.1.1. 基础数据 当天更新 343 字

4.1.2. MySQL笔记 当天更新 276 字

4.2. MariaDB

4.2.1. Install 当天更新 464 字

4.2.2. Documentation 当天更新 603 字

4.2.3. FAQ 87 字

4.3. Redis

4.3.1. Install 当天更新 856 字

4.3.2. Install Windows 65 字

4.3.3. Commands 当天更新 452 字

4.3.4. Redis笔记 当天更新 304 字

4.4. MongoDB

4.4.1. Install 当天更新 729 字

4.4.2. Commands 95 字

4.4.3. 备份恢复 当天更新 493 字

5. Operating System

5.1. 计算机基础

5.1.1. 常识 当天更新 818 字

5.2. Docker

5.2.1. Install 当天更新 144 字

5.2.2. Commands 当天更新 499 字

5.2.3. Docker Maven Plugin 当天更新 221 字

5.2.4. Dockerfile 24 字

5.3. Kubernetes



5.4. Linux

5.4.1. Commands	当天更新	1,221 字
5.4.2. 文件目录操作	当天更新	1,298 字
5.4.3. 用户和组	当天更新	291 字
5.4.4. chkconfig、systemctl	当天更新	284 字
5.4.5. 防火墙	当天更新	551 字

5.5. Windows

5.5.1. Windows	当天更新	681 字
----------------	------	-------

6. □Reading Notes

6.1. 学习中...	当天更新	1,329 字
6.2. Why技术	当天更新	3,193 字
6.3. 博客	当天更新	145 字
6.4. On Java 8	当天更新	998 字
6.5. 深入理解Java虚拟机(第3版)	当天更新	3,404 字

7. □Appendix

7.1. Markdown基本语法	当天更新	1,471 字
7.2. Halo	当天更新	181 字
7.3. Git	当天更新	415 字
7.4. 乱七八糟	当天更新	165 字
7.5. 专有名词解释		32 字

[在 GitHub 上编辑此页](#) ↗

总字数: 56 字 上次更新: 2021-11-23 01:19:59



关于作者

Seven

- 90后 程序员
- 转眼工作6年，你怎么还这么菜？！

[在 GitHub 上编辑此页](#) ↗

总字数: 30 字 上次更新: 2021-11-22 23:56:32

[关于6+1](#) →



关于6+1

记记笔记

- 记记笔记
- 记记生活
- 少年，别太懒

喂，铲屎官！

- 性格超级好的斗鸡眼 萝卜
- 间歇性拉屎的小胆鬼 丸子
- 傲娇又黏人的小姑娘 巧克力

巧克力(Chocolate)

- 暹罗 蓝色 母猫
- 2016年08月01日
- 5年3月22天2小时29分钟39秒 【1940天】



小丸子(Meatball)

- 布偶 蓝双 公猫
- 2018年06月01日
- 3 年 5 月 23 天 2 小时 29 分钟 39 秒 【1271 天】



萝卜(Robo)

- 布偶 海双 公猫
- 2019年08月22日
- 2 年 3 月 2 天 2 小时 29 分钟 39 秒 【824 天】



[在 GitHub 上编辑此页](#) ↗

总字数: 147 字 上次更新: 2021-11-22 23:56:32

← [关于作者](#)

[如何开始](#) →



如何开始

注意

请确保你的 Node.js 版本 ≥ 8 。

参考

[VuePress官网的快速上手](#)

安装

创建工作文件夹seven，进入文件夹。

```
1 # 快速初始化package.json  
2 npm init -y  
3  
4 # 将 VuePress 作为一个本地依赖安装  
5 yarn add -D vuepress  
6 # 或者：  
7 npm install -D vuepress  
8  
9 # 创建 README.md文件  
10 mashibing.md
```

[sh 复制代码](#)

在 package.json 里加一些脚本：

```
1 {  
2   "scripts": {  
3     "docs:dev": "vuepress dev .",  
4     "docs:build": "vuepress build ."
```

[json 复制代码](#)



然后就可以开始写作了：

```
1  yarn dev  
2  # 或者：  
3  npm run dev
```

sh 复制代码

要生成静态的 HTML 文件，运行：

```
1  yarn build  
2  # 或者：  
3  npm run build
```

sh 复制代码

默认情况下，文件将会被生成在 .vuepress/dist，当然，你也可以通过 .vuepress/config.js 中的 dest 字段来修改，生成的文件可以部署到任意的静态文件服务器上，参考 部署 来了解更多。

使用插件

一个插件可以在以 vuepress-plugin-xxx 的形式发布到 npm，你可以这样使用它：

```
1  module.exports = {  
2      plugins: ['vuepress-plugin-xx']  
3 }
```

js 复制代码

如果你的插件名以 vuepress-plugin- 开头，你可以使用缩写来省略这个前缀，和上面等价：

```
1  module.exports = {  
2      plugins: ['xxx']  
3 }
```

js 复制代码



```
1 # 代码块复制按钮                                         sh 复制代码
2 npm install -D vuepress-plugin-nuggets-style-copy
3
4 # 回到顶部(选一个即可)
5 npm install -D @vuepress/plugin-back-to-top
6 npm install -D vuepress-plugin-go-top
7
8 # markdown插件
9 npm install -D markdown-it-task-lists
```

.vuepress/config.js 添加插件

```
1 module.exports = {
2   markdown: {
3     plugins: ['task-lists']
4   },
5   plugins: [
6     ['@vuepress/back-to-top'],
7     ["nuggets-style-copy", {
8       copyText: '复制代码',
9       tip: {
10         time: '3000',
11         content: '复制成功',
12         title: 'Tips'
13       }
14     }]
15   ]
16 }
```

部署

生成静态的 HTML 文件

```
1 npm run build                                         sh 复制代码
```



```
1 # 拷贝  
2 cd .vuepress/dist  
3 # 到linux目录  
4 cd /fobgochod/frontend/vuepress/seven/dist
```

[sh 复制代码](#)

修改 nginx.config 文件，添加server

```
1 vi /etc/nginx/nginx.config  
2  
3 # 默认服务添加  
4 server {  
5     location /seven {  
6         alias /fobgochod/frontend/vuepress/seven/dist;  
7         index index.html index.htm;  
8         autoindex on;  
9     }  
10 }  
11  
12 # 没有设置base(/)  
13 server {  
14     listen 8081;  
15     location / {  
16         root /fobgochod/frontend/vuepress/seven/dist;  
17         index index.html index.htm;  
18         try_files $uri $uri/ /index.html;  
19     }  
20 }
```

[sh 复制代码](#)

重启nginx

```
1 systemctl stop nginx.service  
2 systemctl start nginx.service  
3 # 重启  
4 systemctl restart nginx.service
```

[sh 复制代码](#)

Travis



- GitHub Pages Deployment ↗

```
language: node_js
os: linux
dist: xenial
node_js:
  - 10
before_install:
  - export TZ='Asia/Shanghai'
install:
  - npm install
script:
  - npm run build
deploy:
  - provider: pages
    strategy: git
    skip_cleanup: true
    local_dir: dist
    token: $gh_token
    keep_history: true
    target_branch: gh-pages
on:
  branch: master
```

yaml 复制代码

访问

- <https://fobgochod.github.io> ↗
- <https://fobgochod.com> ↗

[在 GitHub 上编辑此页](#) ↗

总字数: 603 字 上次更新: 2021-11-22 23:56:32

← [关于6+1](#)

[Install](#) →



Install

文档

- <https://www.oracle.com/java/technologies/javase-downloads.html>
- <https://www.oracle.com/java/technologies/oracle-java-archive-downloads.html>
- <http://jdk.java.net/java-se-ri/8-MR3>
- [OpenJDK Download](#)
- [JDK Bugs](#)

Linux

目录结构

```
1   /
2   |   └── opt
3   |   |   └── install
4   |   |   |   └── java
5   |   |   |   |   └── null
6   |   |   └── package
7   |   |       └── java
8   |   |           ├── jdk-8u301-linux-x64
9   |   |           └── jdk-8u301-linux-x64.tar.gz
10  |   └── root(me)
11  └── usr
```

[复制代码](#)

源码安装

参考

安装



```
3 cd /opt/package/java
4 wget https://download.oracle.com/otn/java/jdk/8u301-b09/d3cl
5 # wget https://download.java.net/openjdk/jdk8u41/ri/openjdk
6 # wget https://builds.openlogic.com/downloadJDK/openlogic-o|
7 tar -zxvf jdk-8u301-linux-x64.tar.gz
8
9 # 安装
10 vi /etc/profile
11 #Java
12 export JAVA_HOME=/opt/package/java/jdk1.8.0_301
13 export CLASSPATH=. :${JAVA_HOME}/lib/rt.jar:${JAVA_HOME}/lib.
14 export PATH=$PATH:${JAVA_HOME}/bin
15
16 # 让修改生效：. 或者 source
17 . /etc/profile
18 source /etc/profile
```

检查

```
1 java -version
```

sh 复制代码

YUM安装

参考：

```
1 # 查找
2 yum list java*
3 # 安装
4 yum install java-1.8.0-openjdk.x86_64
5 # 检查
6 java -version
```

sh 复制代码

[在 GitHub 上编辑此页](#)

总字数: 177 字 上次更新: 2021-11-22 23:56:32





Java问题排查

Linux

```
1 # shift+M 按内存排序 f 进入设置界面  
2 top  
3  
4 # 监控内存和CPU  
5 vmstat  
6  
7  
8 # 查看内存  
9 free -th  
10 # 3s刷新一次  
11 free -th -s 3  
12  
13 # 磁盘  
14 df -h
```

[sh 复制代码](#)

Java

jps

```
1 # 查找进程号  
2 jps -lv
```

[sh 复制代码](#)

jstat

```
1 # 每秒打印一次gc，总共5次  
2 jstat -gc [pid] 1000 5  
3 jstat -gcutil [pid] 1000 5
```

[sh 复制代码](#)



jstack

```
1 jstack -l 3333 #进程号  
2 jstack -m [pid]  
3 jstack -F [pid]
```

[sh 复制代码](#)

jmap

```
1 # 查看整个JVM内存状态  
2 jmap -heap [pid]  
3 # 查看JVM堆中对象详细占用情况  
4 jmap -histo [pid]  
5 jmap -histo 3333 | head -10  
6  
7 # 出现OOM时生成堆dump  
8 -XX:+HeapDumpOnOutOfMemoryError  
9 # 生成堆文件地址  
10 -XX:HeapDumpPath=/DAP/logs/heap_dump.hprof  
11  
12 # 直接生成当前JVM的dump文件，所有对象在堆中的分布情况  
13 jmap -dump:format=b,file=/DAP/logs/heap_dump.hprof [pid]  
14 # 加:live 参数 dump 存活对象在队中的分布情况  
15 jmap -dump:live,format=b,file=/DAP/logs/heap_dump.hprof [pid]  
16
```

[sh 复制代码](#)

jinfo

```
1 # 查看JVM参数和系统参数  
2 jinfo pid  
3  
4 # 查看JVM参数，其中 Non-default VM flags 是虚拟机默认设置的参数，  
5 jinfo -flags [pid]  
6 # 可以查看指定参数的值，比如查看堆的最大值(-XX:MaxHeapSize也就是-X  
7 jinfo -flag MaxHeapSize [pid]  
8 # 查看系统参数  
9 jinfo -sysprops [pid]
```

[sh 复制代码](#)



Docker

```
1 # 进入容器  
2 docker exec -it containerName /bin/sh  
3 # 日志  
4 docker logs containerName  
5 # 查找  
6 docker ps -f "name=test"  
7 docker images -f=reference='test'
```

sh 复制代码

DB

```
1 # @formatter:off  
2  
3 # 只列出前100条  
4 show processlist;  
5 # 全列出  
6 show full processlist;  
7 # 指定数据库  
8 select * from information_schema.processlist a where a.db =  
9  
10  
11 # 查看正在锁的事务  
12 select * from information_schema.innodb_locks;  
13 # 查看等待锁的事务  
14 select * from information_schema.innodb_lock_waits;  
15  
16  
17 # 当前运行的所有事务  
18 select * from information_schema.innodb_trx;  
19 select concat('kill ', trx_mysql_thread_id, ';') from information_schema.innodb_trx;  
20 # kill trx_mysql_thread_id;  
21  
22 show open tables where in_use > 0;  
23 show open tables from iam;
```

sh 复制代码



```
1 # 查看innodb引擎的运行时信息  
2 show engine innodb status;  
3  
4 # 查看服务器状态  
5 show status like '%lock%';  
6 show status like 'Threads%';  
7 show global status LIKE '%connections';  
8  
9 # 查看variables  
10 show variables like '%timeout%';  
11 show variables like '%open_files_limit%';  
12 show variables like '%table_open_cache%';  
13 # max_used_connections / max_connections * 100% (理想值≈ 85%)  
14  
15 # 慢日志  
16 show global variables like '%slow%';  
17 # 日志  
18 show global variables where value like '%log%';
```

[sh 复制代码](#)[在 GitHub 上编辑此页](#)

总字数: 567 字 上次更新: 2021-11-22 23:56:32

[← Install](#)[JUC笔记 →](#)



JUC笔记

@formatter:off

多线程与高并发（进程内高并发）

1. 单机高并发应该掌握的线程基础：线程状态，异常与锁等 视频41
2. 解析自旋锁CAS操作与volatile 视频46
3. JUC包下AtomicXXX类与新的同步机制：Latch Semaphore等 视频52
4. LockSupport，高频面试题，AQS源码，以及源码阅读方法论 视频58
5. 强软弱虚四种引用以及ThreadLocal的原理与源码 视频62
6. 线程池可用的各种高并发容器详解：CopyOnWriteList，BlockingQueue等 视频64
7. 详解线程池：自定义线程池，JDK自带线程池，ForkJoin，源码解析等(一) 视频69
8. 详解线程池：自定义线程池，JDK自带线程池，ForkJoin，源码解析等(二) 视频74
9. 单机压测工具JMH，单机最快MQ - Disruptor原理解析 视频76

@formatter:on

1. 单机高并发应该掌握的线程基础：线程状态，异常与锁等
2. 解析自旋锁CAS操作与volatile
3. JUC包下AtomicXXX类与新的同步机制：Latch Semaphore等
4. LockSupport，高频面试题，AQS源码，以及源码阅读方法论
5. 强软弱虚四种引用以及ThreadLocal的原理与源码
6. 线程池可用的各种高并发容器详解：CopyOnWriteList，BlockingQueue等
7. 详解线程池：自定义线程池，JDK自带线程池，ForkJoin，源码解析等(一)
8. 详解线程池：自定义线程池，JDK自带线程池，ForkJoin，源码解析等(二)
9. 单机压测工具JMH，单机最快MQ - Disruptor原理解析



总字数: 467 子 上次更新: 2021-11-23 00:32:13

← Java问题排查

JVM →



JVM

- Java Language and Virtual Machine Specifications ↗
- JDK8启动参数-Windows ↗
- JDK8启动参数-UNIX ↗
- JDK17启动参数 ↗

JVM配置参数分类

Standard Options(-)

These are the most commonly used options that are supported by all implementations of the JVM.

Non-Standard Options(-X)

These options are general purpose options that are specific to the Java HotSpot Virtual Machine.

Advanced Runtime Options(-XX)

These options control the runtime behavior of the Java HotSpot VM.

查看参数

```
1 # 是打印所有的默认参数设置  
2 -XX:+PrintFlagsInitial  
3 # 是打印最终值，如果某个默认值被新值覆盖，显示新值  
4 -XX:+PrintFlagsFinal  
5 # 是打印那些被新值覆盖的项  
6 -XX:+PrintCommandLineFlags  
7  
8 java -XX:+PrintFlagsInitial
```

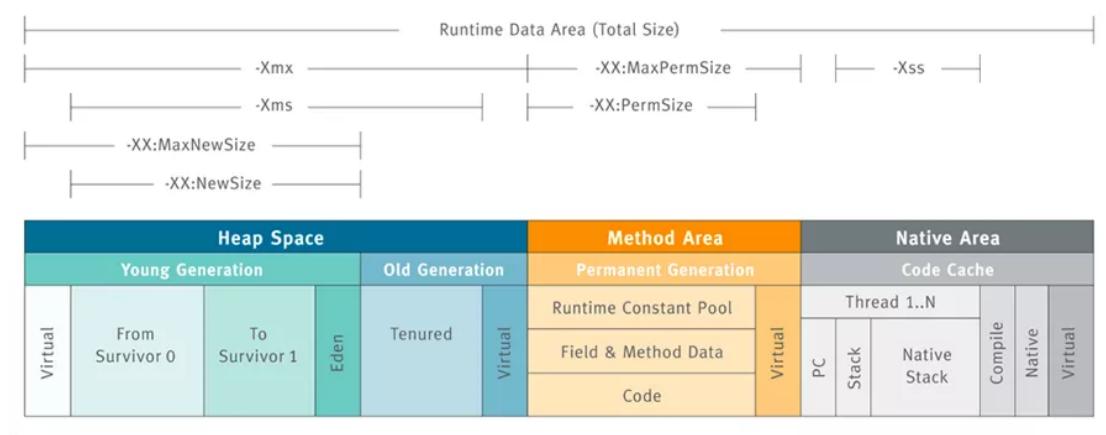
复制代码



格式如下： Type | Name | Operator | Value | Application

- product – 官方支持, JVM内部选项
- rw – 可动态写入的.
- C1 – Client JIT 编译器
- C2 – Server JIT 编译器
- pd – platform Dependent 平台独立
- lp64 – 仅 64 位JVM
- manageable – 外部定义的并且是可动态写入的.
- diagnostic – 用于虚拟机debug的
- experimental – 非官方支持的

内存区域



- 控制参数详解
 - -Xms设置堆的最小空间大小。
 - -Xmx设置堆的最大空间大小。
 - -Xmn堆中新生代初始及最大大小 (NewSize和MaxNewSize为其细化) 。
 - -XX:NewSize设置新生代最小空间大小。
 - -XX:MaxNewSize设置新生代最大空间大小。
 - -XX:PermSize设置永久代最小空间大小。
 - -XX:MaxPermSize设置永久代最大空间大小。
 - -Xss设置每个线程的堆栈大小。

-XX:NewRatio=2



-XX:SurvivorRatio=8

- 默认 -XX:SurvivorRatio=8 Eden:S0:S1=8:1:1
- 假如 -XX:SurvivorRatio=4 Eden:S0:S1=4:1:1

-XX:-UseAdaptiveSizePolicy

- 开启 : -XX:+UseAdaptiveSizePolicy
- 关闭 : -XX:-UseAdaptiveSizePolicy

说明 : -XX:NewRatio=默认值是2、 -Xmn优先级高于-XX:NewRatio

```

1   java -Xms30M -Xmx30m -XX:+PrintFlagsFinal -version | grep -i 复制代码
2   java -Xms30M -Xmx30m -XX:NewRatio=2 -XX:+PrintFlagsFinal -version
3   java -Xms30M -Xmx30m -XX:NewRatio=2 -Xmn20M -XX:+PrintFlagsFinal

```

```

1   -verbose:gc 复制代码
2   -XX:+PrintGCDetails
3   -XX:+PrintGCDateStamps
4   -Xloggc:E:\gc\gc.log

```

JVM问题排查

查看进程ID

```

1   [root@fobgochod ~]# jps -l 复制代码
2   1413 sun.tools.jps.Jps

```

```

1   # 堆内存分布
2   jmap -heap 1413 复制代码
3   # GC信息
4   jstat -gcutil 1413 1000 5
5
6   # 导出堆信息

```



```
1 # 查看进程1413中最耗cpu的子线程  
2 top -p 1413 -H  
3  
4 # 将最耗cpu的线程ID转换为16进制输出  
5 [root@fobgochod ~]# printf "%x \n" 1413  
6 585  
7  
8 # 查询具体出现问题的代码位置  
9 jstack 23219 | grep 585 -A 30
```

[复制代码](#)

```
1 # 查看java线程数  
2 ps -elf | grep java | wc -l  
3 # 监控网络客户连接数  
4 netstat -n | grep tcp | grep 8080 | wc -l  
5 # 查一下tcp连接情况  
6 netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
```

[复制代码](#)

[在 GitHub 上编辑此页](#) ↗

总字数: 713 字 上次更新: 2021-11-22 23:56:32

← JUC笔记

JVM笔记 →



JVM笔记

@formatter:off

JVM从入门到精通

1. JVM入门级class文件格式 视频80
2. 详解Class加载过程 视频86
3. Java内存模型 视频93
4. 内存屏障与JVM指令 视频97
5. Java运行时数据区和常用指令 视频102
6. JVM调优必备理论知识-GC Collector-三色标记 视频110
7. JVM调优实战 视频114
8. JVM实战调优 视频119
9. JVM实战调优 视频124
10. 垃圾回收算法串讲 视频128
11. JVM常见参数总结 视频132

@formatter:on

1. JVM入门级class文件格式
2. 详解Class加载过程
3. Java内存模型
4. 内存屏障与JVM指令
5. Java运行时数据区和常用指令

[在 GitHub 上编辑此页](#)

总字数: 210 字 上次更新: 2021-11-23 00:32:13

← JVM

Install →



Install

文档

- <http://nginx.org/en/download.html>

Linux

目录结构

```
1   /
2   |   opt
3   |   |   install
4   |   |   |   nginx
5   |   |   |   |   conf
6   |   |   package
7   |   |   |   nginx
8   |   |   |   |   nginx-1.18.0
9   |   |   |   |   nginx-1.18.0.tar.gz
10  |   root(me)
11  |   usr
```

复制代码

源码安装

参考

- 【官网】[Building nginx from Sources](#)
- <https://www.cnblogs.com/chenxiaochan/p/7253407.html>
- <https://blog.csdn.net/t8116189520/article/details/81909574>

安装



```

3   cd /opt/package/nginx
4   wget http://nginx.org/download/nginx-1.18.0.tar.gz
5   tar -zxvf nginx-1.18.0.tar.gz

6
7   # 一键安装四个依赖
8   yum -y install gcc zlib zlib-devel pcre-devel openssl
9
10  # 安装
11  cd nginx-1.18.0
12  # 生成Makefile
13  ./configure --prefix=/opt/install/nginx
14  # 编译
15  make
16  # 安装
17  make install PREFIX=/opt/install/nginx

```

启动

```

1  # 打开配置文件，修改默认端口
2  vi /opt/install/nginx/conf/nginx.conf
3  # 启动
4  cd /opt/install/nginx/sbin
5  ./nginx
6  # 停止
7  ./nginx -s stop
8  # 重启
9  ./nginx -s reload
10 # 查看nginx进程是否启动：
11 ps -ef | grep nginx

```

sh 复制代码

构建systemctl服务

```

1  # 关闭之前启动的nginx服务
2  pkill -9 nginx
3
4  # systemctl管理(内容下面代码块)
5  vi /usr/lib/systemd/system/nginx.service
6

```

sh 复制代码



```

10 systemctl start nginx

sh 复制代码

1 # /usr/lib/systemd/system/nginx.service
2 [Unit]
3 Description=nginx - high performance web server
4 Documentation=http://nginx.org/en/docs/
5 After=network-online.target remote-fs.target nss-lookup.target
6 Wants=network-online.target

7
8 [Service]
9 Type=forking
10 PIDFile=/opt/install/nginx/logs/nginx.pid
11 ExecStart=/opt/install/nginx/sbin/nginx -c /opt/install/nginx/conf/nginx.conf
12 ExecReload=/bin/kill -s HUP $MAINPID
13 ExecStop=/bin/kill -s TERM $MAINPID

14
15 [Install]
16 WantedBy=multi-user.target

```

防火墙

```

sh 复制代码

1 firewall-cmd --permanent --zone=public --add-port=80/tcp
2 firewall-cmd --permanent --zone=public --remove-port=80/tcp
3 firewall-cmd --reload
4 firewall-cmd --list-ports

```

YUM安装

参考：

- 【官网】[yum install](#)

```

sh 复制代码

1 # 安装
2 yum install nginx
3 # 配置文件路径
4 vi /etc/nginx/nginx.conf

```



```
8 # 重启
9 nginx -s reload
10 # 开机启动
11 systemctl enable nginx
12 # 验证是否启动
13 ps -ef | grep nginx
```

[在 GitHub 上编辑此页](#)

总字数: 382 字 上次更新: 2021-11-22 23:56:32

← [JVM笔记](#)

[nginx.conf](#) →



nginx.conf

Full Example Configuration ↗

目录结构

```
1   /
2   └── opt
3       ├── install
4       ├── package
5       └── source
6           └── backend
7               └── hold-on
8           └── frontend
9               ├── GitBook
10              └── OnJava8
11               ├── vue
12               │   └── admin
13               └── vuepress
14                   ├── docs
15                   └── fobgochod
16   └── root(me)
17   └── usr
```

复制代码

创建目录

```
1   mkdir -p /opt/{install,package,source}/{backend/hold-on,fron
```

sh 复制代码

修改nginx.conf添加一行 `include /opt/install/nginx/conf.d/*.conf;`
保证了 `/opt/install/nginx/conf.d/` 下,所有以.conf结尾的配置文件,都会
被主配置文件nginx.conf引入并生效

```
1   mkdir -p /opt/install/nginx/conf.d
2   vi /opt/install/nginx/nginx.conf
```

sh 复制代码



```
6     default_type application/octet-stream;
7
8     include /opt/install/nginx/conf.d/*.conf;
9 }
```

静态站点(vuepress、 gitbook)

注：

1. 网站部署到 `http://zhouxiao.co/docs/`, 那么 config.js 的 base 应该被设置成 `"/docs/"`

```
sh 复制代码
1 server {
2     listen      80;
3     server_name localhost;
4
5     access_log  logs/access.localhost.log  main;
6
7     location / {
8         root    html;
9         index   index.html index.htm;
10    }
11
12    # 1
13    location /docs {
14        alias /opt/source/frontend/vuepress/fobgochod/dist;
15        try_files $uri $uri/ /index.html;
16        index  index.html index.htm;
17    }
18
19    location /onjava8 {
20        alias /opt/source/frontend/GitBook/OnJava8/_book;
21        index  index.html index.htm;
22        try_files $uri $uri/ /index.html;;
23    }
24
25    location /docs/ {
26        proxy_set_header X-Real-IP $remote_addr;
```



30 }

vue

```
1 server {  
2     listen      80;  
3     server_name localhost;  
4  
5     access_log  logs/access.localhost.log;  
6  
7     location /admin {  
8         alias /opt/source/frontend/vue/admin/dist;  
9         try_files $uri $uri/ @router;  
10        index index.html index.htm;  
11    }  
12  
13    location @router {  
14        rewrite ^.*$ /admin/index.html last;  
15    }  
16  
17    location /admin/ {  
18        proxy_set_header X-Real-IP $remote_addr;  
19        proxy_set_header Host      $http_host;  
20        proxy_pass http://localhost:7002/;  
21    }  
22}
```

sh 复制代码

后端服务

注：

1. 访问 <http://zhouxiao.co/api/v1/> ,会实际访问
<http://localhost:7003/api/v1>
2. 访问 <http://zhouxiao.co/console/api/v1/> ,会实际访问
<http://localhost:7004/api/v1>



```
3     server_name  localhost;
4
5     access_log    logs/access.localhost.log;
6
7     # 1
8     location /api/v1 {
9         proxy_pass http://localhost:7003;
10    }
11
12    # 2 多了一个/
13    location /console/ {
14        proxy_pass http://localhost:7004/;
15    }
16 }
```

二级域名

```
1      server {
2          listen      80;
3          server_name docs.zhouxiao.co;
4
5          access_log  logs/access.docs.zhouxiao.co.log;
6
7          root /opt/source/frontend/vuepress/docs/dist;
8
9          location / {
10             try_files $uri $uri/ /index.html;
11             index  index.html index.htm;
12         }
13     }
```

sh 复制代码

[在 GitHub 上编辑此页](#)

总字数: 392 字 上次更新: 2021-11-22 23:56:32





Commands

Linux

Windows

常用命令

```
1 # 启动nginx
2 start nginx
3 # 验证配置文件是否正确
4 nginx -t -c nginx.conf
5 # 重新加载配置文件
6 nginx -s reload
7 # 快速停止nginx
8 nginx -s stop
9 # 完整的停止nginx
10 nginx -s quit
11 # 查看Nginx版本：
12 nginx -v
13 # 重新打开日志文件：
14 nginx -s reopen
```

[sh 复制代码](#)

[在 GitHub 上编辑此页](#)

总字数: 87 字 上次更新: 2021-11-22 23:56:32

← [nginx.conf](#)

[Install](#) →



Install

文档

- <https://zookeeper.apache.org/index.html>
- [Download](#)
- [Getting Started](#)
- [ZooKeeper CLI](#)

Linux

目录结构

```
1   /
2   |   opt
3   |   |   install
4   |   |   |   zookeeper
5   |   |   |       data
6   |   |   package
7   |   |       zookeeper
8   |   |           apache-zookeeper-3.6.3-bin
9   |   |           apache-zookeeper-3.6.3-bin.tar.gz
10  |   root(me)
11  |   usr
```

复制代码

源码安装

安装

```
1   sh 复制代码
2   mkdir -p /opt/{install/zookeeper/data,package/zookeeper}
3   cd /opt/package/zookeeper
4   wget https://dlcdn.apache.org/zookeeper/zookeeper-3.6.3/apache-zookeeper-3.6.3-bin.tar.gz
```



```
7 cp zoo_sample.cfg zoo.cfg
8 vi zoo.cfg
9 # 修改dataDir目录
10 dataDir=/opt/install/zookeeper/data
11
12 # 设置环境变量
13 vi /etc/profile
14 # ZooKeeper
15 export ZOOKEEPER_HOME=/opt/package/zookeeper/apache-zookeeper-3.6.3
16 export PATH=${ZOOKEEPER_HOME}/bin:$PATH
17 # 让修改生效：. 或者 source
18 . /etc/profile
```

集群安装

准备4台主机

- 172.16.2.91
- 172.16.2.92
- 172.16.2.93
- 172.16.2.94

1. 创建基础目录(4台)

```
1 mkdir -p /opt/{install/zookeeper/data,package/zookeeper} sh 复制代码
```

2. 172.16.2.91执行

```
1 cd /opt/package/zookeeper sh 复制代码
2 wget https://dlcdn.apache.org/zookeeper/zookeeper-3.6.3/apache-zookeeper-3.6.3-bin.tar.gz
3 tar xf apache-zookeeper-3.6.3-bin.tar.gz
4
5 # 修改配置文件 zoo.cfg
6 cd apache-zookeeper-3.6.3-bin/conf/
7 cp zoo_sample.cfg zoo.cfg
8 vi zoo.cfg
```



```

12 server.1=172.16.2.91:2888:3888
13 server.2=172.16.2.92:2888:3888
14 server.3=172.16.2.93:2888:3888
15 server.4=172.16.2.94:2888:3888
16 ---
17
18 # 拷贝到其它三台主机
19 scp -r /opt/package/zookeeper/apache-zookeeper-3.6.3-bin/ 1:
20 scp -r /opt/package/zookeeper/apache-zookeeper-3.6.3-bin/ 1:
21 scp -r /opt/package/zookeeper/apache-zookeeper-3.6.3-bin/ 1:

```

3. 在dataDir目录下添加myid(4台)

```

1 echo 1 > /opt/install/zookeeper/data/myid
2 echo 2 > /opt/install/zookeeper/data/myid
3 echo 3 > /opt/install/zookeeper/data/myid
4 echo 4 > /opt/install/zookeeper/data/myid

```

sh 复制代码

4. 设置环境变量(4台)

```

1 vi /etc/profile
2 # ZooKeeper
3 export ZOOKEEPER_HOME=/opt/package/zookeeper/apache-zookeeper-3.6.3-bin
4 export PATH=${ZOOKEEPER_HOME}/bin:$PATH
5 # 让修改生效：. 或者 source
6 . /etc/profile

```

sh 复制代码

启动服务端、客户端

```

1 zkServer.sh help
2 # 前台启动 ( start默认后台启动 )
3 zkServer.sh start-foreground
4 # 启动客户端
5 zkCli.sh
6 [zk: localhost:2181(CONNECTED) 0] ls /
7 [zookeeper]

```

sh 复制代码



```
--  
11 [hello, zookeeper]  
12 [zk: localhost:2181(CONNECTED) 3] get /hello  
13 null  
14 [zk: localhost:2181(CONNECTED) 4] set /hello "world"  
15 [zk: localhost:2181(CONNECTED) 5] get /hello  
16 world  
17 [zk: localhost:2181(CONNECTED) 6] get -s /hello  
18 world  
19 cZxid = 0x10000000f  
20 ctime = Sun Sep 05 07:56:56 EDT 2021  
21 mZxid = 0x100000011  
22 mtime = Sun Sep 05 07:58:16 EDT 2021  
23 pZxid = 0x10000000f  
24 cversion = 0  
25 dataVersion = 1  
26 aclVersion = 0  
27 ephemeralOwner = 0x0  
28 dataLength = 5  
29 numChildren = 0  
30 [zk: localhost:2181(CONNECTED) 7] help
```

YUM安装

[在 GitHub 上编辑此页](#) ↗

总字数: 427 字 上次更新: 2021-11-22 23:56:32

← Commands

ZK笔记 →



ZK笔记

@formatter:off

ZooKeeper

1. zookeeper介绍、安装、shell cli 使用，基本概念验证 [视频55](#)
2. zookeeper原理知识，paxos、zab、角色功能、API开发基础 [视频56](#)
3. zookeeper案例：分布式配置注册发现、分布式锁、ractive模式编程 [视频56](#)

@formatter:on

1. zookeeper介绍、安装、shell cli 使用，基本概念验证
2. zookeeper原理知识，paxos、zab、角色功能、API开发基础
3. zookeeper案例：分布式配置注册发现、分布式锁、ractive模式编程

[在 GitHub 上编辑此页](#)

总字数: 155 字 上次更新: 2021-11-23 00:32:13

← [Install](#)

[IO笔记](#) →



IO笔记

@formatter:off

内存与IO，磁盘IO，网络IO

1. 虚拟文件系统，文件描述符，IO重定向 视频166
2. 内核中PageCache、mmap作用、java文件系统io、nio、内存中缓冲区作用 视频174
3. Socket编程BIO及TCP参数 视频177
4. C10K问题及NIO精讲和IO模型性能压测 视频184
5. 网络编程之多路复用器及Epoll精讲 视频185
6. 网络编程java API 实战多路复用器开发 视频187
7. 全手写急速理解Netty模型及IO模型应用实战 视频189
8. Netty之IO模型开发本质手写部分实现推导篇 视频191
9. 全手写基于Netty的RPC框架自定义协议，连接池 视频196
10. 全手写基于Netty的RPC框架 协议编解码问题 粘包拆包与内核关系 视频198
11. 全手写基于Netty的RPC框架 provider端简单dispatcher实现RPC调用全流程 视频203
12. 全手写基于Netty的RPC框架 简单重构框架分层及RPC传输的本质及有无状态的RPC区别 视频207
13. 自定义HTTP协议解析和HTTPserver调用实现 视频216

@formatter:on

1. 虚拟文件系统，文件描述符，IO重定向
2. 内核中PageCache、mmap作用、java文件系统io、nio、内存中缓冲区作用
3. Socket编程BIO及TCP参数
4. C10K问题及NIO精讲和IO模型性能压测
5. 网络编程之多路复用器及Epoll精讲
6. 网络编程java API 实战多路复用器开发
7. 全手写急速理解Netty模型及IO模型应用实战



-
- 10. 全手写基于Netty的RPC框架 协议编解码问题 粘包拆包与内核关系
 - 11. 全手写基于Netty的RPC框架 provider端简单dispatcher实现RPC调用全流程
 - 12. 全手写基于Netty的RPC框架 简单重构框架分层及RPC传输的本质及有无状态的RPC区别
 - 13. 自定义HTTP协议解析和HTTPserver调用实现

[在 GitHub 上编辑此页](#) ↗

总字数: 616 字 上次更新: 2021-11-23 00:32:13

← ZK笔记

NPM →



NPM

下载：<https://nodejs.org/en/download/releases>

仓库：<https://www.npmjs.com>

安装

版本 node-v10.16.3-x64.msi npm 是 Node.js 的包管理工具，用来安装各种 Node.js 的扩展

```
1 | node -v  
2 | npm -v
```

sh 复制代码

1.临时使用

```
1 | npm --registry https://registry.npm.taobao.org install express
```

sh 复制代码

2.持久使用

```
1 | npm config set registry https://registry.npm.taobao.org
```

sh 复制代码

3.验证是否成功

```
1 | npm config get registry
```

sh 复制代码

4.通过cnpm使用

```
1 | npm install -g cnpm --registry=https://registry.npm.taobao.org
```

sh 复制代码



配置

参考：<https://www.cnblogs.com/liaojie970/p/9296177.html>

1. 配置npm的全局模块的存放路径以及cache的路径

```
1  npm config set prefix "D:\install\nodejs\node_global"      sh 复制代码
2  npm config set cache "D:\install\nodejs\node_cache"
```

2. 系统环境变量添加系统变量path

```
1  D:\install\nodejs\node_global      sh 复制代码
```

3. 安装express

注：“-g”这个参数意思是装到global目录下，也就是上面说设置的“D:\install\nodejs\node_global”

```
1  npm install -g express      sh 复制代码
2  # express 4.x版本中将命令工具分出来，安装一个命令工具，执行命令：
3  npm install -g express-generator
```

4. 验证

```
1  express --version      sh 复制代码
2  npm info express
```

npm安装时-S -D作用及区别

参考：<https://www.cnblogs.com/web-record/p/10904907.html>



安装

```
1 # 安装  
2 npm install -D vuepress  
3 # 代码复制按钮插件  
4 npm install -D vuepress-plugin-nuggets-style-copy  
5 # 回到顶部插件  
6 npm install -D @vuepress/plugin-back-to-top  
7 npm install -D vuepress-plugin-go-top
```

sh 复制代码

markdown插件

```
1 npm install -D markdown-it-task-lists  
2  
3 npm install -D vuepress-plugin-comment  
4 npm install -D @vuepress/plugin-google-analytics
```

sh 复制代码

卸载

```
1 # 删除模块，但不删除模块留在package.json中的对应信息  
2 npm uninstall module_name  
3 # 删除模块，同时删除模块留在package.json中dependencies下的对应信息  
4 npm uninstall module_name --save  
5 # 删除模块，同时删除模块留在package.json中devDependencies下的对应信息  
6 npm uninstall module_name --save-dev
```

sh 复制代码

-S、-D、-g

-S

- --save-prod
- 包名会被注册在package.json的dependencies
- 在生产环境下这个包的依赖依然存在



-D

- --save-dev
- 包名会被注册在package.json的devDependencies
- 里面的插件只用于开发环境，不用于生产环境
- npm install -d module_name
- npm install --save-dev module_name

-g

- npm install -g module_name
- 全局安装

不写

- npm install module_name
- 本地安装，将安装包放在 ./node_modules 下
- 包名不会进入package.json里面

[在 GitHub 上编辑此页](#)

总字数: 529 字 上次更新: 2021-11-22 23:56:32

← IO笔记

Yarn →



Yarn

参考：<https://yarn.bootcss.com/docs/usage/>

初始化一个新项目

```
1 | yarn init
```

sh 复制代码

添加依赖包

```
1 | yarn add [package]  
2 | yarn add [package]@[version]  
3 | yarn add [package]@[tag]
```

sh 复制代码

将依赖项添加到不同依赖项类别中 分别添加到 devDependencies、
peerDependencies 和 optionalDependencies 类别中：

```
1 | yarn add [package] --dev  
2 | yarn add [package] --peer  
3 | yarn add [package] --optional
```

sh 复制代码

升级依赖包

```
1 | yarn upgrade [package]  
2 | yarn upgrade [package]@[version]  
3 | yarn upgrade [package]@[tag]
```

sh 复制代码

移除依赖包

```
1 | yarn remove [package]
```

sh 复制代码

安装项目的全部依赖



3

yarn install

[在 GitHub 上编辑此页](#)

总字数: 120 字 上次更新: 2021-11-22 23:56:32

← NPM

VuePress →



VuePress

- 参考：
 - https://segmentfault.com/a/1190000015237352?utm_source=tag-newest#articleHeader11
 - <https://www.jianshu.com/p/7a2cc8a7f40c>
- 插件：<https://vuepress.github.io/zh/>
- 支持语言：<https://prismjs.com/#languages-list>

安装

```
1 # 安装  
2 npm install -g vuepress  
3 # 卸载  
4 npm uninstall -g vuepress
```

sh 复制代码

插件

```
1 # 代码复制按钮插件  
2 npm install -g vuepress-plugin-nuggets-style-copy  
3 # 回到顶部插件  
4 npm install -g @vuepress/plugin-back-to-top  
5 npm install -g vuepress-plugin-go-top
```

sh 复制代码

FAQ

npm install -D vuepress-plugin-export
报错：ERROR: Failed to download Chromium r686378! Set
"PUPPETEER_SKIP_CHROMIUM_DOWNLOAD" env variable to skip
download.



PUPPETEER_SKIP_CHROMIUM_DOWNLOAD=1 阻止下载 Chromium

入门

目录结构如下

```
1   sample
2     └── docs
3       ├── .vuepress
4       └── README.md
5   └── package.json
```

[复制代码](#)

1. 初始化

- 创建工作文件夹 sample
- 执行初始化命令

```
1   sample>npm init -y
```

[sh 复制代码](#)

2. 配置

- 新建文件夹/docs
- 创建/docs/README.md文件
- 修改/package.json，添加下述兩行

```
1   {
2     "name": "sample",
3     "version": "1.0.0",
4     "description": "",
5     "main": "index.js",
6     "scripts": {
7       "test": "echo \\\"Error: no test specified\\\" && exit 1",
8       "docs:dev": "vuepress dev docs",
9       "docs:build": "vuepress build docs"
10    },
11    "keywords": []
```

[json 复制代码](#)



3.运行

```
1 | docs>npm run docs:dev
```

sh 复制代码

4.编译

```
1 | docs>npm run docs:build
```

sh 复制代码

[在 GitHub 上编辑此页](#) ↗

总字数: 284 字 上次更新: 2021-11-22 23:56:32

← Yarn

GitBook →



GitBook

安装

```
1 npm install -g gitbook-cli
```

sh 复制代码

常用命令

```
1 gitbook init //初始化目录文件  
2 gitbook help //列出gitbook所有的命令  
3 gitbook --help //输出gitbook-cli的帮助信息  
4 gitbook build //生成静态网页  
5 gitbook serve //生成静态网页并运行服务器  
6 gitbook build --gitbook=2.0.1 //生成时指定gitbook的版本，本地没有  
7 gitbook ls //列出本地所有的gitbook版本  
8 gitbook ls-remote //列出远程可用的gitbook版本  
9 gitbook fetch 标签/版本号 //安装对应的gitbook版本  
10 gitbook update //更新到gitbook的最新版本  
11 gitbook uninstall 2.0.1 //卸载对应的gitbook版本  
12 gitbook build --log=debug //指定log的级别  
13 gitbook buildid --debug //输出错误信息
```

sh 复制代码

入门

目录结构如下

```
1 sample  
2   └── _book  
3     ├── README.md  
4     └── SUMMARY.md
```

复制代码



- 的进阶入门入 Sample

- 执行初始化命令

```
1 | gitbook init
```

sh 复制代码

2.运行

```
1 | gitbook serve  
2 | gitbook serve --port 8080
```

sh 复制代码

3.编译

```
1 | gitbook build
```

sh 复制代码

4.生成PDF文件

```
1 | gitbook pdf ./ ./sample.pdf
```

sh 复制代码

[在 GitHub 上编辑此页](#) ↗

总字数: 263 字 上次更新: 2021-11-22 23:56:32

← VuePress

基础数据 →



基础数据

Sakila

下载地址：<https://dev.mysql.com/doc/index-other.html>

Oracle Scott

```
1  DROP SCHEMA IF EXISTS scott;                                复制代码
2  CREATE SCHEMA scott;
3  USE scott;
4
5  -- DROP TABLE IF EXISTS emp;
6  -- DROP TABLE IF EXISTS dept;
7  -- DROP TABLE IF EXISTS salgrade;
8  -- DROP TABLE IF EXISTS bonus;
9
10 CREATE TABLE IF NOT EXISTS dept (
11     deptno INT(4) NOT NULL,
12     dname VARCHAR(14) DEFAULT NULL,
13     loc VARCHAR(13) DEFAULT NULL,
14     PRIMARY KEY (deptno)
15 );
16
17 CREATE TABLE IF NOT EXISTS emp (
18     empno INT(4) NOT NULL,
19     ename VARCHAR(10) DEFAULT NULL,
20     job VARCHAR(9) DEFAULT NULL,
21     mgr INT(4) DEFAULT NULL,
22     hiredate DATE DEFAULT NULL,
23     sal DECIMAL(7,2) DEFAULT NULL,
24     comm DECIMAL(7,2) DEFAULT NULL,
25     deptno INT(2) DEFAULT NULL,
26     PRIMARY KEY (empno), KEY fk_deptno (deptno),
27     CONSTRAINT fk_deptno FOREIGN KEY (deptno) REFERENCES de
28 );
```



```
31
32     losal DECIMAL(7,2) DEFAULT NULL,
33     hisal DECIMAL(7,2) DEFAULT NULL
34 );
35
36 CREATE TABLE IF NOT EXISTS bonus (
37     ename VARCHAR(10) DEFAULT NULL,
38     job VARCHAR(9) DEFAULT NULL,
39     sal DECIMAL(7,2) DEFAULT NULL,
40     comm DECIMAL(7,2) DEFAULT NULL
41 );
42
43 INSERT IGNORE INTO dept (deptno, dname, loc) VALUES
44     (10, 'ACCOUNTING', 'NEW YORK'),
45     (20, 'RESEARCH', 'DALLAS'),
46     (30, 'SALES', 'CHICAGO'),
47     (40, 'OPERATIONS', 'BOSTON');
48
49 INSERT IGNORE INTO emp (empno, ename, job, mgr, hiredate, sal)
50     (7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800.00, NULL),
51     (7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600.00),
52     (7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250.00),
53     (7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975.00),
54     (7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250.00),
55     (7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850.00),
56     (7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450.00),
57     (7788, 'SCOTT', 'ANALYST', 7566, '1987-04-19', 3000.00),
58     (7839, 'KING', 'PRESIDENT', NULL, '1981-11-17', 5000.00),
59     (7844, 'TURNER', 'SALESMAN', 7698, '1981-09-08', 1500.00),
60     (7876, 'ADAMS', 'CLERK', 7788, '1987-05-23', 1100.00, NULL),
61     (7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950.00, NULL),
62     (7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000.00, NULL),
63     (7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300.00, NULL);
64
65 INSERT IGNORE INTO salgrade (grade, losal, hisal) VALUES
66     (1, 700.00, 1200.00),
67     (2, 1201.00, 1400.00),
68     (3, 1401.00, 2000.00),
69     (4, 2001.00, 3000.00),
70     (5, 3001.00, 9999.00);
```



← [GitBook](#)

[MySQL笔记](#) →



MySQL笔记

```
@formatter:off
```

MySQL调优

1. mysql调优--使用profiles,performance_schema性能监控 [视频107](#)
2. mysql调优--数据类型和schema优化 [视频108](#)
3. mysql调优--索引基本实现原理及索引优化 [视频111](#)
4. mysql调优--mysql索引优化实现细节 [视频112](#)
5. mysql调优--mysql查询优化分析 [视频113](#)
6. mysql调优--mysql分区设计及分区优化 [视频115](#)
7. mysql调优--mysql分区优化2及参数设计优化 [视频116](#)
8. mysql调优--mysql参数设计优化及总结 [视频117](#)

```
@formatter:on
```

1. mysql调优--使用profiles,performance_schema性能监控
2. mysql调优--数据类型和schema优化
3. mysql调优--索引基本实现原理及索引优化
4. mysql调优--mysql索引优化实现细节
5. mysql调优--mysql查询优化分析
6. mysql调优--mysql分区设计及分区优化
7. mysql调优--mysql分区优化2及参数设计优化
8. mysql调优--mysql参数设计优化及总结

[在 GitHub 上编辑此页](#)

总字数: 276 字 上次更新: 2021-11-23 00:32:13

← [基础数据](#)

[Install](#) →



Install

文档

<https://downloads.mariadb.org/mariadb/repositories>

Linux

源码安装

YUM安装

```
1 # 查看系统发型版本  
2 yum -y install redhat-lsb  
3 lsb_release -a
```

sh 复制代码

```
1 vi /etc/yum.repos.d/MariaDB.repo
```

sh 复制代码

CentOS 7



- SLES
- openSUSE
- Arch Linux
- Mageia
- Fedora
- **CentOS**
- RedHat
- Mint
- Ubuntu
- Debian

- CentOS 8 (ARM64)
- CentOS 8 (ppc64le)
- CentOS 8 (x86_64)
- CentOS 7 (ppc64le)
- CentOS 7 (ppc64)
- **CentOS 7 (x86_64)**
- CentOS 6 (x86_64)
- CentOS 6 (x86)

- **10.5 [Stable]**
- 10.6 [RC]
- 10.4 [Old Stable]
- 10.3 [Old Stable]
- 10.2 [Old Stable]

Here is your custom MariaDB YUM repository entry for CentOS. Copy and paste it into a file under /etc/yum.repos.d/ (we suggest naming the file MariaDB.repo or something similar).

```
# MariaDB 10.5 CentOS repository list - created 2021-06-26 14:50 UTC
# http://downloads.mariadb.org/mariadb/repositories/
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.5/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

After the file is in place, install MariaDB with:

```
sudo yum install MariaDB-server MariaDB-client
```

If you haven't already accepted the MariaDB GPG key, you will be prompted to do so. See "[Installing MariaDB with yum](#)" for detailed information.

Please see [Installing OQGraph](#) for details on additional install steps needed for that storage engine.

```
1 # MariaDB 10.5 CentOS repository list - created 2021-04-08 : sh 复制代码
2 # http://downloads.mariadb.org/mariadb/repositories/
3 [mariadb]
4 name = MariaDB
5 baseurl = http://yum.mariadb.org/10.5/centos7-amd64
6 gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
7 gpgcheck=1
```

```
1 # 中国科学技术大学 sh 复制代码
2 [mariadb]
3 name = MariaDB
4 baseurl = http://mirrors.ustc.edu.cn/mariadb/yum/10.2/cento:
5 gpgkey=http://mirrors.ustc.edu.cn/mariadb/yum/RPM-GPG-KEY-Mi:
6 gpgcheck=1
```

```
8 # 清华大学开源软件镜像站
9 [mariadb]
10 name = MariaDB
11 baseurl = https://mirrors.tuna.tsinghua.edu.cn/mariadb/yum/:
12 gpgkey=https://mirrors.tuna.tsinghua.edu.cn/mariadb/yum/RPM
13 gpgcheck=1
```



CentOS 8

To generate the entries select an item from each of the boxes below. Once an item is selected in each box, your customized repository configuration will appear below.

1. Choose a Distro

- SLES
- openSUSE
- Arch Linux
- Mageia
- Fedora
- **CentOS**
- RedHat
- Mint
- Ubuntu
- Debian

2. Choose a Release

- CentOS 8 (ARM64)
- CentOS 8 (ppc64le)
- **CentOS 8 (x86_64)**
- CentOS 7 (ppc64le)
- CentOS 7 (ppc64)
- CentOS 7 (x86_64)
- CentOS 6 (x86_64)
- CentOS 6 (x86)

3. Choose a Version

- **10.5 [Stable]**
- 10.6 [RC]
- 10.4 [Old Stable]
- 10.3 [Old Stable]

Here is your custom MariaDB YUM repository entry for CentOS. Copy and paste it into a file under /etc/yum.repos.d/ (we suggest naming the file MariaDB.repo or something similar).

```
# MariaDB 10.5 CentOS repository list - created 2021-06-26 12:14 UTC
# http://downloads.mariadb.org/mariadb/repositories/
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.5/centos8-amd64
module_hotfixes=1
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

The configuration item `module_hotfixes=1` is a workaround for what we have been told is a dnf bug. See [MDEV-20673](#) for more details.

After the file is in place, install and start MariaDB with:

```
sudo dnf install MariaDB-server
sudo systemctl start mariadb
```

If you haven't already accepted the MariaDB GPG key, you will be prompted to do so during the install. See "[Installing MariaDB with yum](#)" for detailed information.

```
1 # MariaDB 10.5 CentOS repository list - created 2021-06-26 : sh 复制代码
2 # http://downloads.mariadb.org/mariadb/repositories/
3 [mariadb]
4 name = MariaDB
5 baseurl = http://yum.mariadb.org/10.5/centos8-amd64
6 module_hotfixes=1
7 gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
8 gpgcheck=1
```

```
1 dnf install MariaDB-server sh 复制代码
```

启动



配置

```

1 mysql_secure_installation                                         sh 复制代码
2
3 # 首先是设置密码，会提示先输入密码
4 Enter current password for root (enter for none):<-初次运行直
5
6 # 设置密码
7 Set root password? [Y/n] <- 是否设置root用户密码，输入y并回车或]
8 New password: <- 设置root用户的密码
9 Re-enter new password: <- 再输入一次你设置的密码
10
11
12 # 其他配置
13 Remove anonymous users? [Y/n] <- 是否删除匿名用户，回车
14Disallow root login remotely? [Y/n] <-是否禁止root远程登录，回车
15 Remove test database and access to it? [Y/n] <- 是否删除test数据库
16 Reload privilege tables now? [Y/n] <- 是否重新加载权限表，回车

```

配置字符集

```

1 show variables like '%character%';                                复制代码
2 show variables like '%collation%';

```

```

1 #文件/etc/my.cnf                                              复制代码
2 vi /etc/my.cnf
3 #在[mysqld]标签下添加
4 init_connect='SET collation_connection=utf8_general_ci'
5 init_connect='SET NAMES utf8'
6 character_set_server=utf8
7 collation_server=utf8_general_ci
8 skip-character-set-client-handshake
9 lower_case_table_names=1
10
11 #文件/etc/my.cnf.d/client.cnf

```



```
15  
16 #文件/etc/my.cnf.d/mysql-clients.cnf  
17 vi /etc/my.cnf.d/mysql-clients.cnf  
18 #在[mysql]中添加  
19 default-character-set=utf8
```

授予外网登陆权限

```
1 # 登陆mysql  
2 mysql -uroot -proot  
3 # 授权  
4 MariaDB [mysql]> grant all privileges on *.* to root@'%' identified by '123456';  
5 MariaDB [mysql]> flush privileges;
```

[复制代码](#)

防火墙

```
1 firewall-cmd --permanent --zone=public --add-port=3306/tcp  
2 firewall-cmd --permanent --zone=public --remove-port=3306/tcp  
3 firewall-cmd --reload  
4 firewall-cmd --list-ports
```

[sh 复制代码](#)[在 GitHub 上编辑此页](#)

总字数: 464 字 上次更新: 2021-11-22 23:56:32

[← MySQL笔记](#)[Documentation →](#)



Documentation

MariaDB Server Documentation

官网帮助文档地址：

- <https://mariadb.com/kb/en/documentation>
- <https://mariadb.com/kb/zh-cn/mariadb-documentation>

参数介绍：https://mariadb.com/kb/en/server-system-variables/#sql_mode

数据类型：<https://mariadb.com/kb/en/library/data-types>

数据类型字节：<https://mariadb.com/kb/en/library/data-type-storage-requirements>

UPDATE语法：<https://mariadb.com/kb/zh-cn/update>

SQL Statements：<https://mariadb.com/kb/en/library/sql-statements>

快速插入：<https://mariadb.com/kb/zh-cn/how-to-quickly-insert-data-into-mariadb>

备份和恢复：<https://mariadb.com/kb/en/library/backing-up-and-restoring-databases>

错误码：<https://mariadb.com/kb/en/library/mariadb-error-codes>

SQLSTATE：<https://mariadb.com/kb/en/sqlstate>

Error Codes：<https://mariadb.com/kb/en/mariadb-error-codes>

Varchar

```
1 # mysql语句最大长度  
2 show variables like 'max_allowed_packet'
```

sh 复制代码



参考：<https://mariadb.com/kb/en/library/innodb-strict-mode/>

1. Mysql每行记录的最大值为64k (BLOB and TEXT数据类型不纳入统计) , 即65535个字节 (表所有字段加起来的大小)
2. 而varchar要用1-2字节来存储字段长度 , 小于255的1字节 , 大于255的2字节。
3. Mysql 5.0后,英文字符固定都是一个字节 , 汉字字符根据编码方式占不同字节 , UTF-8占3个字节 , GBK占了2个字节。

```

1 # 统计字节个数
2 select length();
3 # 统计字符个数
4 select char_length();
5 # 例子
6 select length('aa'),length('悟空'),char_length('aa'),char_le

```

sh 复制代码

结果 #1 (4x1)			
LENGTH('AA')	LENGTH('悟空')	CHAR_LENGTH('AA')	CHAR_LENGTH('悟空')
2	6	2	2

- UTF-8编码：最大长度是 21844 。根据上面信息可以推算出(65535-2)/3=21844余1。
- GBK编码: 最大长度是 32766 。根据上面信息可以推算出(65535-2)/2=32766余1。

timeout

[connection_timeout](#)

```

1 connection_timeout
2 The connection_timeout parameter is used to disconnect sess.
3 The session timeouts are disabled by default. To enable them
4 section. A value of zero is interpreted as no timeout, the :

```

sh 复制代码



```
8 TABLE) either do them with a direct connection to the serve
9
10 Example:
11
12 [Test Service]
13 connection_timeout=300
```

wait_timeout ↗

```
1 wait_timeout
2   · Description: Time in seconds that the server waits for a
3     session value is initialized when a thread starts up from
4     or from the interactive_timeout value, if the connection :
5   · Commandline: --wait-timeout=#  

6   · Scope: Global, Session
7   · Dynamic: Yes
8   · Type: numeric
9   · Default Value: 28800
10  · Range: (Windows): 1 to 2147483
11  · Range: (Other): 1 to 31536000
```

sh 复制代码

interactive_timeout ↗

```
1 interactive_timeout
2   · Description: Time in seconds that the server waits for an
3     mysql_real_connect() CLIENT_INTERACTIVE option) to become
4   · Commandline: --interactive-timeout=#  

5   · Scope: Global, Session
6   · Dynamic: Yes
7   · Data Type: numeric
8   · Default Value: 28800
9   · Range: (Windows): 1 to 2147483
10  · Range: (Other): 1 to 31536000
```

sh 复制代码

MySQL : 参数wait_timeout和interactive_timeout以及空闲超时的实现



[在 GitHub 上编辑此页](#)

总字数: 603 字 上次更新: 2021-11-22 23:56:32

[← Install](#)

[FAQ →](#)



FAQ

[1932]Table doesn't exist in engine 的解决方法 参考：

<http://chenxuhou.com/detail/121.html>

```
1 # 这三个文件存放在数据库data目录下,将这三个文件补全就OK了      sh 复制代码
2 ib_logfile0
3 ib_logfile1
4 ibdata1
```

参考：<https://www.cnblogs.com/heyongboke/p/10437510.html>

```
1 SQL Error (1558): Column count of mysql.proc is wrong. Expected 33 but found 34. This is probably because MySQL's tmpdir was not set correctly when it was created. Created with MariaDB 5.0.56, now running 10.0.11-MariaDB. Please use mysql_upgrade to fix this error.      sh 复制代码
2
3 mysql_upgrade -u root -p
```

```
[root@pt ~]# mysql_upgrade -u root -p
Enter password:
Phase 1/7: Checking and upgrading mysql database
Processing databases
mysql                                         OK
mysql.columns_priv                         OK
mysql.db                                    OK
mysql.event                                 OK
mysql.func                                  OK
mysql.help_category                        OK
mysql.help_keyword                          OK
mysql.help_relation                         OK
mysql.help_topic                           OK
mysql.host                                 OK
mysql.ndb_binlog_index                     OK
mysql.plugin                               OK
mysql.proc                                 OK
mysql.procs_priv                           OK
mysql.proxies_priv                         OK
mysql.servers                             OK
```



总字数: 87 字 上次更新: 2021-11-22 23:56:32

← Documentation

Install →



Install

文档

- <https://redis.io/download>
- <https://download.redis.io/releases/redis-6.0.10.tar.gz>

Linux

目录结构

```
1   /
2   |   opt
3   |   |   install
4   |   |   |   redis
5   |   |   |   |   bin
6   |   |   |   |   etc
7   |   |   |   |   |   redis.conf
8   |   |   |   |   lib
9   |   |   |   log
10  |   |   package
11  |   |   |   redis
12  |   |   |   |   redis-6.2.4
13  |   |   |   |   |   src
14  |   |   |   |   |   utils
15  |   |   |   |   |   |   install_server.sh
16  |   |   |   |   |   redis.conf
17  |   |   |   |   sentinel.conf
18  |   |   |   redis-6.2.4.tar.gz
19  |   |   root(me)
20  |   |   usr
```

复制代码

源码安装



安装

```
1   sh 复制代码  
2   mkdir -p /opt/{install/redis/{lib,etc,log},package/redi:  
3   cd /opt/package/redis  
4   # 下载  
5   wget https://download.redis.io/releases/redis-6.2.4.tar.gz  
6   # 解压  
7   tar -zxvf redis-6.2.4.tar.gz  
8   cd redis-6.2.4  
9   # 编译  
10  make  
11  # 不成功可以再试试  
12  # yum install gcc  
13  # make distclean  
14  # make  
15  # 安装到/opt/install/redis  
16  make install PREFIX=/opt/install/redis
```

复制redis.conf

```
1   sh 复制代码  
2   cp /opt/package/redis/redis-6.2.4/redis.conf /opt/install/r
```

启动服务端、客户端

```
1   sh 复制代码  
2   # 可以cd到PREFIX指定的安装目录  
3   cd /opt/install/redis/bin  
4   # 或者redis-6.2.4/src目录  
5   cd /opt/package/redis/redis-6.2.4/src  
6  
6   # 1.启动服务端  
7   ./redis-server  
8   # 后台启动  
9   ./redis-server &  
10  # 指定配置文件启动  
11  ./redis-server /opt/install/redis/etc/redis.conf  
12  #./redis-server /opt/package/redis/redis-6.2.4/redis.conf  
13
```



```
17 # 3. 关闭服务端  
18 ./redis-cli shutdown  
19 # 或者  
20 ps -aux|grep redis-server  
21 kill -9 pid
```

设定环境变量

```
1 # 设定环境变量  
2 vi /etc/profile  
3 export REDIS_HOME=/opt/install/redis  
4 export PATH=$PATH:${REDIS_HOME}/bin  
5 # 使环境变量生效  
6 source /etc/profile  
7 echo $PATH  
8  
9 # 设定后可以直接启动  
10 redis-server  
11 redis-cli
```

sh 复制代码

注册服务

执行install_server.sh

多次执行 `install_server.sh` 可安装多个redis

```
1 # 注册服务(service redis status)  
2 [root@fobgochod ~]# cd /opt/package/redis/redis-6.2.4/utils  
3 [root@fobgochod utils]# ./install_server.sh  
4 Welcome to the redis service installer  
5 This script will help you easily set up a running redis ser  
6  
7 Please select the redis port for this instance: [6379]  
8 Selecting default: 6379  
9 Please select the redis config file name [/etc/redis/6379.conf]  
10 Please select the redis log file name [/var/log/redis_6379.log]  
11 Please select the data directory for this instance [/var/lib/redis]
```

sh 复制代码



```
15 Config file      : /opt/install/redis/etc/6379.conf
16 Log file        : /opt/install/redis/log/6379.log
17 Data dir         : /opt/install/redis/lib/6379
18 Executable       : /opt/install/redis/bin/redis-server
19 Cli Executable  : /opt/install/redis/bin/redis-cli
20 Is this ok? Then press ENTER to go on or Ctrl-C to abort.
21 Copied /tmp/6379.conf => /etc/init.d/redis_6379
22 Installing service...
23 Successfully added to chkconfig!
24 Successfully added to runlevels 345!
25 Starting Redis server...
26 Installation successful!
27 [root@fobgochod utils]#
```

Redis运行命令目录

```
1 cd /etc/init.d/
```

sh 复制代码

配置修改

/opt/install/redis/etc/6379.conf

```
1 # 方式一
2 # 注释掉配置文件中的bind 【ip address】
3 # bind 127.0.0.1
4 # 关闭Redis的服务保护模式
5 protected-mode no

6

7 # 方式二
8 bind 0.0.0.0
9 # PS 该模式下Redis的服务保护模式不需要关闭
10 protected-mode yes

11

12 # Redis后台启动
13 daemonize yes
```

sh 复制代码



FAQ

gcc版本低了

```

1  yum -y install centos-release-scl
2  yum -y install devtoolset-9-gcc devtoolset-9-gcc-c++ devtoo.
3  scl enable devtoolset-9 bash
4  echo "source /opt/rh/devtoolset-9/enable" >> /etc/profile
5  gcc -v

```

sh 复制代码

以前安装过低版本redis

```

1  service redis start
2  # 里面的redis_service路径是老版本5.0的redis，但是/etc/redis/637
3  vi /etc/init.d redis
4
5  Starting Redis server...
6
7  *** FATAL CONFIG FILE ERROR ***
8  Reading the configuration file, at line 356
9  >>> 'rdb-del-sync-files no'
10 Bad directive or wrong number of arguments

```

sh 复制代码

redis版本较高，如redis-6.2.4，注册服务会提示如下

```

1  [root@fobgochod utils]# ./install_server.sh
2  Welcome to the redis service installer
3  This script will help you easily set up a running redis ser
4
5  This systems seems to use systemd.
6  Please take a look at the provided example service unit file
7
8
9  # 方案一，注释安装文件install_server.sh中对应检查
10 #bail if this system is managed by systemd
11 #_pid_1_exe=$(readlink -f /proc/1/exe)"
12 #if [ "${_pid_1_exe##*/}" = systemd ]

```

sh 复制代码



```
16 #      exit 1
17 #fi
18 #unset _pid_1_exe
```

注册服务后启动失败

```
1 # 问题
2 [root@fobgochod utils]# service redis_6379 start
3 /var/run/redis_6379.pid exists, process is already running
4
5 # 解决
6 [root@fobgochod utils]# rm -f /var/run/redis_6379.pid
7 [root@fobgochod utils]#
```

[sh 复制代码](#)

YUM安装

[在 GitHub 上编辑此页](#)

总字数: 856 字 上次更新: 2021-11-22 23:56:32

[← FAQ](#)[Install Windows →](#)



Install Windows

文档

- <https://www.runoob.com/redis/redis-install.html>
- <https://github.com/tporadowski/redis/releases>

测试

启动服务端

```
1     redis-server.exe redis.windows.conf
```

[sh 复制代码](#)

启动客户端

```
1 # 另起一个cmd窗口
2 redis-cli.exe -h 127.0.0.1 -p 6379
3 # 设置键值对
4 set myKey abc
5 # 取出键值对
6 get myKey
7 # 退出
8 quit
```

[sh 复制代码](#)

[在 GitHub 上编辑此页](#)

总字数: 65 字 上次更新: 2021-11-22 23:56:32

← [Install](#)

[Commands](#) →



Commands

文档

- <http://doc.redisfans.com>
- <https://redis.io/commands>
- <https://redis.io/documentation>

切换数据库

```
1 172.16.2.141:0>select 1  
2 "OK"  
3 172.16.2.141:1>
```

[sh](#) [复制代码](#)

返回当前数据库的 key 的数量

```
1 172.16.2.141:1>dbsize  
2 "4"  
3 172.16.2.141:1>
```

[sh](#) [复制代码](#)

将当前数据库的 key 移动到给定的数据库 db 当中

```
1 172.16.2.141:0>select 1  
2 "OK"  
3 172.16.2.141:1>setex key1 100 value1  
4 "OK"  
5 172.16.2.141:1>move key1 2  
6 "1"  
7 172.16.2.141:1>select 2  
8 "OK"  
9 172.16.2.141:2>get key1  
10 "value1"  
11 172.16.2.141:2>
```

[sh](#) [复制代码](#)



删除给定的一个或多个 key

```
1 172.16.2.141:1>mset key1 value1 key2 value2 key3 value3      sh 复制代码
2 "OK"
3 172.16.2.141:1>del key1 key2 key3
4 "3"
5 172.16.2.141:1>
```

检查给定 key 是否存在

```
1 172.16.2.141:1>set key1 value1      sh 复制代码
2 "OK"
3 172.16.2.141:1>exists key1
4 "1"
5 172.16.2.141:1>del key1
6 "1"
7 172.16.2.141:1>exists key1
8 "0"
9 172.16.2.141:1>
```

为给定 key 设置生存时间，当 key 过期时(生存时间为 0)，它会被自动删除

```
1 172.16.2.141:1>set key1 value1      sh 复制代码
2 "OK"
3 172.16.2.141:1>ttl key1
4 "-1"
5 172.16.2.141:1>expire key1 100
6 "1"
7 172.16.2.141:1>ttl key1
8 "96"
9 172.16.2.141:1>
```

移除给定 key 的生存时间



```

3 172.16.2.141:1>expire key1 100
4 "1"
5 172.16.2.141:1>ttl key1
6 "95"
7 172.16.2.141:1>persist key1
8 "1"
9 172.16.2.141:1>ttl key1
10 "-1"
11 172.16.2.141:1>

```

查找所有符合给定模式 pattern 的 key

```

1 172.16.2.141:1>mset one 1 two 2 three 3 four 4 sh 复制代码
2 "OK"
3 172.16.2.141:1>keys *o*
4 1) "two"
5 2) "four"
6 3) "one"
7 172.16.2.141:1>

```

GET

```

1 # 获取key对应value sh 复制代码
2 get key1
3 # 将给定 key 的值设为 value , 并返回 key 的旧值(old value)。
4 getset key value

```

SET

```

1 set key1 value1 sh 复制代码
2 set key2 value2 EX 100
3 set key3 value3 PX 100000
4
5 # key存在则不能修改
6 set key4 value4 NX

```



```
10 set key5 value5 XX
11 set key5 value5
12 set key5 new-value5 XX
13
14 # 将值 value 关联到 key , 并将 key 的生存时间设为 seconds (以秒为
15 setex key6 60 value6
16
17 # 将 key 中储存的数字值加减一。如果 key 不存在 , 那么 key 的值会先被
18 incr key7
19 incrby key7 99
20
21 # 将 key 中储存的数字值减一
22 decr key7
23 decrby key7 99
24
25 # 将 key 的值设为 value , 当且仅当 key 不存在。
26 setnx key8 value8
```

[在 GitHub 上编辑此页](#)

总字数: 452 字 上次更新: 2021-11-22 23:56:32

← [Install Windows](#)

[Redis笔记](#) →



Redis笔记

@formatter:off

Redis

1. redis 介绍及NIO原理介绍 视频43 [↗](#)
2. redis的string类型&bitmap 视频44 [↗](#)
3. redis的list、set、hash、sorted_set、skipList 视频45 [↗](#)
4. redis的消息订阅、pipeline、事务、modules、布隆过滤器、缓存 LRU 视频48 [↗](#)
5. redis的持久化RDB、fork、copyonwrite、AOF、RDB&AOF混合使用 视频49 [↗](#)
6. redis的集群：主从复制、CAP、PAXOS、cluster分片集群01 视频50 [↗](#)
7. redis的集群：主从复制、CAP、PAXOS、cluster分片集群02 视频53 [↗](#)
8. redis开发：spring.data.redis、连接、序列化、high/low api 视频54 [↗](#)

@formatter:on

1. redis 介绍及NIO原理介绍
2. redis的string类型&bitmap
3. redis的list、set、hash、sorted_set、skipList
4. redis的消息订阅、pipeline、事务、modules、布隆过滤器、缓存 LRU
5. redis的持久化RDB、fork、copyonwrite、AOF、RDB&AOF混合使用
6. redis的集群：主从复制、CAP、PAXOS、cluster分片集群01
7. redis的集群：主从复制、CAP、PAXOS、cluster分片集群02
8. redis开发：spring.data.redis、连接、序列化、high/low api



← Commands

Install →



Install

文档

- https://www.mongodb.com/try/download/community?tck=docs_server

Linux

目录结构

```
1   /
2   └── opt
3       ├── install
4           ├── mongodb
5               ├── bin
6               ├── data
7               ├── etc
8               │   └── mongod.conf
9               └── logs
10      └── package
11          └── mongodb
12              ├── mongodb-linux-x86_64-rhel80-4.4.6
13              └── mongodb-linux-x86_64-rhel80-4.4.6.tgz
14      └── root(me)
15      └── usr
```

复制代码

源码安装

参考：

- [【官网】】Install MongoDB Community Edition using .tgz Tarball](#)

安装



```

3  yum -y install libcurl openssl xz-libs
4  # 下载
5  cd /opt/package/mongodb
6  wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-
7  # 解压
8  tar -zxvf mongodb-linux-x86_64-rhel80-4.4.6.tgz
9  cp ./mongodb-linux-x86_64-rhel80-4.4.6/bin/* /opt/install/m
10 # 添加conf文件
11 vi /opt/install/mongodb/etc/mongod.conf

```

[mongod参数说明 : Options](#)

[MongoDB配置说明 : mongod.conf](#)

```

1  # /opt/install/mongodb/etc/mongod.conf
2  systemLog:
3      destination: file
4      logAppend: true
5      path: /opt/install/mongodb/logs/mongodb.log
6  storage:
7      dbPath: /opt/install/mongodb/data
8      journal:
9          enabled: true
10     directoryPerDB: true
11 processManagement:
12     fork: true
13     pidFilePath: /opt/install/mongodb/bin/mongod.pid
14     timeZoneInfo: /usr/share/zoneinfo
15 net:
16     port: 27017
17     bindIp: 0.0.0.0
18 setParameter:
19     enableLocalhostAuthBypass: false
20 # 登陆密码授权(enabled)
21 security:
22     authorization: disabled

```

[yaml 复制代码](#)

```

1  # 添加环境变量
2  vi /etc/profile

```

[sh 复制代码](#)



```
6 source /etc/profile  
7 # 查看Path  
8 echo $PATH
```

启动

```
1 # 启动服务  
2 mongod -f /opt/install/mongodb/etc/mongod.conf  
3 # 关闭服务  
4 mongod --shutdown -f /opt/install/mongodb/etc/mongod.conf  
5 # 或者  
6 pkill mongod  
7 # 或进入 mongo shell  
8 mongo  
9 db.version()  
10 use admin  
11 db.shutdownServer()
```

sh 复制代码

卸载

```
1 ## 卸载  
2 sudo yum erase $(rpm -qa | grep mongodb-org)  
3 sudo rm -r /opt/package/mongodb
```

sh 复制代码

防火墙

```
1 firewall-cmd --permanent --zone=public --add-port=27017/tcp  
2 firewall-cmd --permanent --zone=public --remove-port=27017/  
3 firewall-cmd --reload  
4 firewall-cmd --list-ports
```

sh 复制代码

YUM安装

参考：



```
1 vi /etc/yum.repos.d/mongodb-org-4.4.repo                                         sh 复制代码  
  
1 [mongodb-org-4.4]                                                               sh 复制代码  
2   name=MongoDB Repository  
3   baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongo  
4   gpgcheck=1  
5   enabled=1  
6   gpgkey=https://www.mongodb.org/static/pgp/server-4.4.asc  
  
1 # 最新稳定版本                                                               sh 复制代码  
2 yum install -y mongodb-org  
3 # 或者指定版本  
4 yum install -y mongodb-org-4.4 mongodb-org-server-4.4 mongo  
  
1 # 修改默认配置                                                               sh 复制代码  
2 vi /etc/mongod.conf
```

[mongod参数说明 : Options](#)

[MongoDB配置说明 : mongod.conf](#)

```
1 # /etc/mongod.conf                                                               yaml 复制代码  
2  
3 # for documentation of all options, see:  
4 #   http://docs.mongodb.org/manual/reference/configuration-options  
5  
6 # where to write logging data.  
7 systemLog:  
8   destination: file  
9   logAppend: true  
10  path: /var/log/mongodb/mongod.log  
11  
12 # Where and how to store data.  
13 storage:  
14   dbPath: /var/lib/mongo  
15   journal:
```



```
18
19     # wiredTiger:
20
21     # how the process runs
22     processManagement:
23         fork: true # fork and run in background
24         pidFilePath: /var/run/mongodb/mongod.pid # location of p.
25         timeZoneInfo: /usr/share/zoneinfo
26
27     # network interfaces
28     net:
29         port: 27017
30         bindIp: 0.0.0.0 # Enter 0.0.0.0,:: to bind to all IPv4 a
31
32     # 登陆密码授权(enabled)
33     security:
34         authorization: disabled
```

启动

```
1 # /usr/lib/systemd/system/mongod.service
2 systemctl daemon-reload
3 systemctl enable mongod
4 systemctl start mongod
```

sh 复制代码

卸载

```
1 # 卸载
2 systemctl stop mongod
3 yum erase $(rpm -qa | grep mongodb-org)
4 rm -r /var/log/mongodb
5 rm -r /var/lib/mongo
```

sh 复制代码

Windows

下载 <https://www.mongodb.com/download-center>



```
3 mongod --dbpath "d:\my text\data"  
4  
5 # 移除 MongoDB 服务  
6 D:\install\MongoDB\Server\bin>mongod --remove  
7 # 指定服务名  
8 mongod --remove --serviceName "MongoDB"  
9  
10 # 安装 MongoDB服务  
11 D:\install\MongoDB\Server\bin>mongod --config "D:\install\Mi  
12 # 指定服务名  
13 mongod --config "D:\install\MongoDB\Server\bin\mongo.config  
14  
15  
16 # 配置本地Windows MongoDB 服务 CMD启动服务 , 指定存储位置启动服务  
17 D:\install\MongoDB\Server\bin>mongod --dbpath "D:\install\Mi  
18  
19 MongoDB已经开启 , 浏览器访问http://localhost:27017  
20  
21 # 服务启动和关闭 ( 服务名 : MongoDB )  
22 net stop MongoDB  
23 net start MongoDB  
24  
25  
26 # 客户端 : 管理员模式打开命令行窗口 ,  
27 D:\install\MongoDB\Server\bin>mongo
```

[在 GitHub 上编辑此页](#)

总字数: 729 字 上次更新: 2021-11-22 23:56:32

← Redis笔记

Commands →



Commands

Command Helpers ↗

```
1 mongo                                         sh 复制代码
2 db.version()
3
4 show dbs
5 use <db>
6 show collections
7
8 db.help()
9 db.<collection>.help()
10
11 db.stats()
12 db.<collection>.stats()
```

```
1 # 查询集合                                         sh 复制代码
2 db.collection.find()
3 # 集合数目
4 db.collection.count()
5 db.collection.find().count()
6 # 分页
7 db.collection.find().limit(10).skip(1)
8 # 条件查询
9 db.buckets.find({ _id: ObjectId("5fa0c50274af09562c7e9992") })
10 db.getCollection("ddd.fileInfos").find({ _id: JUUID("ab8a6a1")) })
```

```
1 # 建立文本索引                                         sh 复制代码
2 db.collection.createIndex({
3     "describe": "text"
4 }, {
5     name: "text_describe"
6 })
7
8 # Like (MongoDB 支持正则表达式)
```



```
--  
12 | select * from users where name like "hurry%";  
13 | db.users.find({name:/^hurry/});
```

[在 GitHub 上编辑此页](#) ↗

总字数: 95 字 上次更新: 2021-11-22 23:56:32

← [Install](#)

[备份恢复](#) →



备份恢复

文档

The MongoDB Database Tools Documentation [↗](#)

Sample Data

Import Data into MongoDB [¶ ↗](#)

inventory.crud.json

```
1  use test                                         sh 复制代码
2
3  db.dept.drop();
4  db.dept.insert({ deptno: 10, dname: 'ACCOUNTING', loc: 'NEW YORK' });
5  db.dept.insert({ deptno: 20, dname: 'RESEARCH', loc: 'DALLAS' });
6  db.dept.insert({ deptno: 30, dname: 'SALES', loc: 'CHICAGO' });
7  db.dept.insert({ deptno: 40, dname: 'OPERATIONS', loc: 'BOSTON' });
8
9  db.emp.drop();
10 db.emp.insert({ empno: 7369, ename: 'SMITH', job: 'CLERK', hiredate: '1980-12-17' });
11 db.emp.insert({ empno: 7499, ename: 'ALLEN', job: 'SALESMAN', hiredate: '1981-02-22' });
12 db.emp.insert({ empno: 7521, ename: 'WARD', job: 'SALESMAN', hiredate: '1981-02-22' });
13 db.emp.insert({ empno: 7566, ename: 'JONES', job: 'MANAGER', hiredate: '1981-04-01' });
14 db.emp.insert({ empno: 7654, ename: 'MARTIN', job: 'SALESMAN', hiredate: '1981-09-28' });
15 db.emp.insert({ empno: 7698, ename: 'BLAKE', job: 'MANAGER', hiredate: '1981-05-01' });
16 db.emp.insert({ empno: 7782, ename: 'CLARK', job: 'MANAGER', hiredate: '1981-04-01' });
17 db.emp.insert({ empno: 7788, ename: 'SCOTT', job: 'ANALYST', hiredate: '1983-09-17' });
18 db.emp.insert({ empno: 7839, ename: 'KING', job: 'PRESIDENT', hiredate: '1981-11-17' });
19 db.emp.insert({ empno: 7844, ename: 'TURNER', job: 'SALESMAN', hiredate: '1982-08-20' });
20 db.emp.insert({ empno: 7876, ename: 'ADAMS', job: 'CLERK', hiredate: '1982-07-01' });
21 db.emp.insert({ empno: 7900, ename: 'JAMES', job: 'CLERK', hiredate: '1981-12-03' });
22 db.emp.insert({ empno: 7902, ename: 'FORD', job: 'ANALYST', hiredate: '1981-12-03' });
23 db.emp.insert({ empno: 7934, ename: 'MILLER', job: 'CLERK', hiredate: '1982-01-15' });
```



备份

```
1 # 备份数据库test到/DAP/DB/MongoDB/  
2 mongodump --db=test --out=/DAP/DB/MongoDB/  
3 # 备份数据库test下的表emp到/DAP/DB/MongoDB/  
4 mongodump --db=test --collection=emp --out=/DAP/DB/MongoDB/
```

sh 复制代码

恢复

```
1 # 恢复数据到原库  
2 mongorestore /DAP/DB/MongoDB/  
3  
4 # 恢复数据库test中的表emp  
5 mongorestore --nsInclude=test.emp /DAP/DB/MongoDB/  
6  
7 # 通过json文件恢复数据到指定数据库、指定表  
8 mongorestore --db=testStore --collection=empStore /DAP/DB/Mi
```

sh 复制代码

FAQ

```
1 # 阿里云MongoDB快照恢复数据  
2  
3 # 1. 删除dbpath目录下的mongod.lock  
4 rm -f /var/lib/mongo/mongod.lock  
5 # 2. 将data目录下所有文件授权给当前用户 root忽略 ,  
6 chown -R user:user  
7 # 3. 清除log下目录  
8 rm -rf /var/log/mongodb/*  
9 # 4. 修复数据  
10 ./mongod --repair --dbpath /var/lib/mongo/  
11 # 5. 按配置启动服务端  
12 ./mongod --config mongodb.config  
13 # 6. 启动客户端  
14 mongo
```

sh 复制代码



[在 GitHub 上编辑此页](#)

总字数: 493 字 上次更新: 2021-11-22 23:56:32

← [Commands](#)

[常识](#) →



常识

@formatter:off

0x7C00

为什么主引导记录的内存地址是0x7C00 ?

- <http://www.ruanyifeng.com/blog/2015/09/0x7c00.html>
- <https://www.glamenv-septzen.net/en/view/6>

MSL

参考：<https://www.cnblogs.com/ytys/p/9993535.html>

MSL是Maximum Segment Lifetime英文的缩写，中文可以译为“报文最大生存时间”

RFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等。

2MSL即两倍的MSL，TCP的TIME_WAIT状态也称为2MSL等待状态，当TCP的一端发起主动关闭，在发出最后一个ACK包后，即第3次握手完成后发送了第四次握手的ACK包后就进入了TIME_WAIT状态，必须在此状态下停留两倍的MSL时间，等待2MSL时间主要目的是怕最后一个ACK包对方没收到，那么对方在超时后将重发第三次握手的FIN包，主动关闭端接到重发的FIN包后可以再发一个ACK应答包。在TIME_WAIT状态时两端的端口不能使用，要等到2MSL时间结束才可继续使用。当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。不过在实际应用中可以通过设置SO_REUSEADDR选项达到不必等待2MSL时间结束再使用此端口。

并行和并发的区别



- **并行**：同一时刻多条指令在多个处理器同时执行，不管是微观，还是宏观上，都是同时执行的。
- 举个例子，并发就是一个家庭主妇既要烧饭，也要带娃，也要打扫房间，如果每个事情只做一分钟，然后轮换，从宏观上来说，会造成同时执行的错觉。并行就是该家庭主妇请了两个保姆，一个专职负责烧饭，一个专职负责带娃，自己专职负责打扫卫生，不管从宏观还是微观上来看，他们都是同时执行的。
- 某位大佬曾经说两者的区别，**并发是同一时间应对多件事情的能力，并行是同一时间去做多件事情的能力。**

带宽5Mbps、100Mbps的网速

- Mbps=Mbit/s即兆比特每秒（ Million bits per second的缩写）
- MB/s是Mbps的8倍
- $5\text{Mbps} = 5 * 1024\text{Kbps} / 8 = 640\text{KB/s}$
- $100\text{Mbps} = 100 * 1024\text{Kbps} / 8 = 12.5\text{MB/s}$

ASCII , Unicode 和 UTF-8

- [Unicode对照表](#)
- [Unicode查询对应字符](#)
- [ASCII , Unicode 和 UTF-8](#)
- [Unicode与JavaScript详解](#)

原码、反码、补码

- 正数：原码=反码=补码
- 负数：
 - 反码：原码除符号位外，按位取反
 - 补码：反码+1（负数的补码等于他的原码自低位向高位，尾数的第一个‘1’及其右边的‘0’保持不变，左边的各位按位取反，符号位不变。）



正数	负数
0 0000	0 0000
1 0001	-1 1111
2 0010	-2 1110
3 0011	-3 1101
4 0100	-4 1100
5 0101	-5 1011
6 0110	-6 1010
7 0111	-7 1001
	-8 1000

正数	负数
-0 1111	-0 1000
-1 1110	-1 1001
-2 1101	-2 1010
-3 1100	-3 1011
-4 1011	-4 1100
-5 1010	-5 1101
-6 1001	-6 1110
-7 1000	-7 1111

https://blog.csdn.net/zhiwen_a[在 GitHub 上编辑此页](#)

总字数: 818 字 上次更新: 2021-11-22 23:56:32

[← 备份恢复](#)[Install →](#)



Install

文档

Linux

源码安装

YUM安装

参考：

- Install Docker Engine on CentOS

```
1 ## 卸载旧版本                                         sh 复制代码
2 yum remove docker \
3       docker-client \
4       docker-client-latest \
5       docker-common \
6       docker-latest \
7       docker-latest-logrotate \
8       docker-logrotate \
9       docker-engine
10
11 # Install the yum-utils package (which provides the yum-con:
12 yum install -y yum-utils
13 yum-config-manager \
14   --add-repo \
15   https://download.docker.com/linux/centos/docker-ce.repo
16
17 # Install the latest version of Docker Engine and container
18 yum -y install docker-ce docker-ce-cli containerd.io
19
20 # 启动和测试
21 systemctl start docker
22 docker run hello-world
```



Portainer安装

参考：

<https://www.portainer.io/installation>

```
1 docker search portainer
2 docker pull portainer/portainer
3 docker volume create portainer_data
4
5 docker run -d -p 8000:8000 -p 9000:9000 \
6   --name=portainer \
7   --restart=always \
8   -v /var/run/docker.sock:/var/run/docker.sock \
9   -v portainer_data:/data \
10  portainer/portainer-ce
```

sh 复制代码

[在 GitHub 上编辑此页](#)

总字数: 144 字 上次更新: 2021-11-22 23:56:32

← 常识

Commands →



Commands

docker commandline ↗

```
1 docker version  
2 docker info
```

sh 复制代码

images

- [build ↗](#)
- [images ↗](#)
- [rmi ↗](#)

```
1 # 构建镜像  
2 docker build -t nginx:v1 .  
3 docker build -f Dockerfile.debug .  
4  
5 # 查看  
6 docker images  
7 docker images -a  
8  
9 ## 删除  
10 docker rmi [IMAGE ID]  
11 docker rmi REPOSITORY:TAG
```

sh 复制代码

```
1 # 导出镜像  
2 docker save -o ~/hello-world.tar hello-world:latest  
3 docker save hello-world > ~/hello-world.tar  
4 # 导入镜像  
5 docker load -i ~/hello-world.tar  
6 docker load < ~/hello-world.tar
```

sh 复制代码

Prune unused Docker objects ↗



```
3 # 删除未使用镜像  
4 docker image prune
```

container

```
1 docker run -idt --name docker_nginx_v1 -p 80:80 nginx:v1      sh 复制代码  
2 # --name="docker_nginx_v1": 为容器指定一个名称 ;  
3 # -p: 端口映射 , 格式为 : 主机(宿主)端口:容器端口 ;  
4 # -d: 后台运行容器 , 并返回容器ID ;  
5 # -i: 以交互模式运行容器 , 通常与 -t 同时使用 ;  
6 # -t: 为容器重新分配一个伪输入终端 , 通常与 -i 同时使用 ;  
7  
8 cd /var/lib/docker  
9  
10 # 容器自动启动  
11 docker run --restart=always  
12 # 如果已经启动了则可以使用如下命令 :  
13 docker update --restart=always <CONTAINER ID>
```

```
1 # 查看  
2 docker ps  
3 docker ps --no-trunc  
4 docker ps -a  
5  
6 docker start [CONTAINER ID]  
7 docker stop [CONTAINER ID]  
8 docker restart [CONTAINER ID]  
9  
10 # 删除  
11 docker rm [CONTAINER ID]  
12 docker rm -f [CONTAINER ID]  
13  
14 # 进入容器  
15 docker exec -it CONTAINER_ID /bin/sh  
16 exit
```



```
3  
4      # 网络  
5      docker network ls  
6      # 默认的网络模式是 bridge 。在该模式下，docker 创建了一个 bridge ,  
7      ifconfig docker0
```

数据拷贝

```
1      # 1) 将主机/www/runoob目录拷贝到容器96f7f14e99ab的/www目录下。  
2      docker cp /www/runoob 96f7f14e99ab:/www/  
3      # 2) 将主机/www/runoob目录拷贝到容器96f7f14e99ab中，目录重命名为  
4      docker cp /www/runoob 96f7f14e99ab:/www  
5  
6      # 将容器96f7f14e99ab的/www目录拷贝到主机的/tmp目录中。  
7      docker cp 96f7f14e99ab:/www /tmp/
```

volume

```
1      # 创建卷hello  
2      docker volume create hello  
3      # 查看卷  
4      docker volume ls  
5      # 查看卷[hello]的信息  
6      docker volume inspect hello  
7      # 查看无效卷  
8      docker volume ls -qf dangling=true  
9      # 删除卷  
10     docker volume rm $(docker volume ls -qf dangling=true)
```

[在 GitHub 上编辑此页](#)

总字数: 499 字 上次更新: 2021-11-22 23:56:32





Docker Maven Plugin

Docker容器设定

```
1  vi /usr/lib/systemd/system/docker.service           sh 复制代码
2
3  # 在ExecStart=/usr/bin/dockerd 后面加上
4  -H tcp://0.0.0.0:2375 -H unix://var/run/docker.sock
5
6  # 重启
7  systemctl daemon-reload
8  systemctl restart docker
9
10 # Verify is correctly
11 curl http://localhost:2375/version
```

```
1  # 防火墙开放2375端口                         sh 复制代码
2  firewall-cmd --zone=public --add-port=2375/tcp --permanent
3  firewall-cmd --reload
```

Maven pom插件

```
1 <plugin>                                         复制代码
2   <groupId>com.spotify</groupId>
3   <artifactId>docker-maven-plugin</artifactId>
4   <configuration>
5     <!--指定生成的镜像名-->
6     <imageName>${project.artifactId}</imageName>
7     <!--指定标签-->
8     <imageTags>
9       <imageTag>${project.version}</imageTag>
10    </imageTags>
11    <!-- 指定 Dockerfile 路径-->
12    <dockerDirectory>${project.basedir}/src/main/resources</dockerDirectory>
```



```
16      <resources>
17          <resource>
18              <targetPath>/</targetPath>
19                  <!--jar 包所在的路径 此处配置的 即对应 target
20                  <directory>${project.build.directory}</direc
21                  <!-- 需要包含的 jar包 ,这里对应的是 Dockerfile
22                  <include>${project.build.finalName}.jar</in
23          </resource>
24      </resources>
25  </configuration>
26  <executions>
27      <execution>
28          <id>build-image</id>
29              <!--将插件绑定在package这个phase上。也就是说，用户只
30              <phase>package</phase>
31              <goals>
32                  <goal>build</goal>
33              </goals>
34      </execution>
35  </executions>
36 </plugin>
```

[在 GitHub 上编辑此页](#)

总字数: 221 字 上次更新: 2021-11-22 23:56:32

[← Commands](#)

[Dockerfile →](#)



Dockerfile

- [Docker镜像仓库](#)
- [openjdk](#)
- [DockerFile指令介绍](#)
- [Best practices for writing Dockerfiles](#)

[在 GitHub 上编辑此页](#)

总字数: 24 字 上次更新: 2021-11-22 23:56:32

← [Docker Maven Plugin](#)

[Install](#) →



Install

文档

- [minikube start](#)
- [Accessing Dashboard](#)
- [kubectl](#)
 - [install-kubectl](#)
 - [kubectl-commands](#)
- [minikube安装dashboard](#)

Linux

安装

安装Docker

Docker Install

```
1  vi /usr/lib/systemd/system/docker.service
2
3  # 在ExecStart=/usr/bin/dockerd 后面加上
4  --exec-opt native.cgroupdriver=systemd
5
6  # 重启
7  systemctl daemon-reload
8  systemctl restart docker
```

sh 复制代码

创建用户、组

```
1  # 查看用户、组
2  cat /etc/passwd
3  cat /etc/group
```

sh 复制代码



kubectl安装

```
1 curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/VERSION) | sh 复制代码  
2 install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl  
3 kubectl version --client  
4  
5 alias kubectl="kubectl -s http://localhost:39215"  
6 kubectl -s http://localhost:39215 version
```

minikube安装

切换到非root用户

```
1 su admin sh 复制代码  
  
1 curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 | sh 复制代码  
2 install minikube-linux-amd64 /usr/local/bin/minikube  
3  
4 # minikube manager  
5 yum install conntrack  
6 minikube start --driver=docker  
7 minikube config set driver docker  
8 minikube config set memory 16384  
9  
10 minikube version  
11 minikube status  
12 minikube pause  
13 minikube stop  
14 minikube delete --all  
15  
16 minikube service tomcat-service  
17 minikube ssh -- docker info  
18 minikube service list
```



dashboard安装

```
1 # 启用dashboard  
2 minikube addons list  
3 minikube addons enable dashboard  
4  
5 # 查看如下信息  
6 # Labels: k8s-app=kubernetes-dashboard  
7 # Port: 9090/TCP  
8 kubectl get pod -A  
9 kubectl describe pod kubernetes-dashboard-968bcb79-hb58p -n  
10  
11  
12 # 方法一：直接修改  
13 kubectl -n kubernetes-dashboard edit service kubernetes-dash  
14 kubectl -n kubernetes-dashboard get service kubernetes-dash  
15 minikube service list  
16  
17 # 方法二：新增yml，创建service  
18 vi minikube-dashboard.yaml  
19 kubectl apply -f minikube-dashboard.yaml  
20 minikube service list
```

sh 复制代码

minikube-dashboard.yaml

```
1 # minikube-dashboard.yaml  
2 apiVersion: v1  
3 kind: Service  
4 metadata:  
5   name: minikube-dashboard  
6   namespace: kubernetes-dashboard  
7   labels:  
8     addonmanager.kubernetes.io/mode: Reconcile  
9     k8s-app: kubernetes-dashboard  
10    kubernetes.io/minikube-addons: dashboard  
11 spec:  
12   selector:  
13     k8s-app: kubernetes-dashboard  
14   type: NodePort
```

yaml 复制代码



```
17  
18     targetPort: 9090  
19     nodePort: 30001
```

打包部署

```
1 ## 创建部署 pod  
2 kubectl create -f k8s-dahboard.yml  
3 ## 更新部署配置  
4 kubectl apply -f k8s-dahboard.yml  
5 ## 删除  
6 kubectl delete -f k8s-dahboard.yml  
7 ## 查看已经部署的pod 显示信息全面[-o wide]  
8 kubectl get pod [-o wide]  
9 ## 查看pod详细信息  
10 kubectl describe pod podName  
11 ## 查看pod输出日志  
12 kubectl logs [-f] podName
```

sh 复制代码

tomcat部署

```
1 # tomcat-deploy.yaml  
2 apiVersion: apps/v1  
3 kind: Deployment  
4 metadata:  
5   name: tomcat-deploy  
6   labels:  
7     k8s-app: tomcat-cluster  
8 spec:  
9   replicas: 2  
10  selector:  
11    matchLabels:  
12      k8s-app: tomcat-cluster  
13  template:  
14    metadata:  
15      labels:  
16        k8s-app: tomcat-cluster  
17  spec:  
18    containers:
```

yaml 复制代码



```
--  
22      - containerPort: 8080  
  
1 # tomcat-service.yaml  
2 apiVersion: v1  
3 kind: Service  
4 metadata:  
5   name: tomcat-service  
6   labels:  
7     k8s-app: tomcat-cluster  
8 spec:  
9   type: NodePort  
10  selector:  
11    k8s-app: tomcat-cluster  
12  ports:  
13    - port: 8000  
14      targetPort: 8080  
15      nodePort: 30002  
  
1 vi tomcat-deploy.yaml  
2 # 部署  
3 kubectl create -f tomcat-deploy.yaml  
4 # 查看  
5 kubectl get deployment  
6 kubectl get pod -o wide  
7 kubectl describe pod tomcat-deploy-749b7f4dbf-98zhz  
8  
9 vi tomcat-service.yaml  
10 # 服务  
11 kubectl create -f tomcat-service.yaml  
12 # 查看  
13 kubectl get service  
14 kubectl describe service tomcat-service  
15  
16 minikube service list  
17 minikube service tomcat-service --url  
yaml 复制代码  
sh 复制代码
```



← Dockerfile

Commands →



Commands

minikube

```
1 minikube start --driver=none  
2 minikube config set driver none  
3 minikube config set memory 16384  
4  
5 minikube status  
6 minikube pause  
7 minikube stop  
8 minikube delete --all  
9  
10 minikube service tomcat-service  
11 minikube ssh -- docker info  
12 minikube service list  
13  
14 minikube addons list  
15 minikube addons enable dashboard
```

sh 复制代码

kubectl

缩写：pod (po)、 service (svc)、 replicationcontroller (rc)、 deployment (deploy)、 replicaset (rs)

```
1 kubectl get pod  
2 kubectl describe pod [pod]  
3 # 创建一个服务 暴露444端口  
4 kubectl expose pod [pod] --port=444 --name=test-service  
5 # 对主机暴露8080端口  
6 kubectl port-forward [pod] 8080  
7 # 连接到容器  
8 kubectl attach pod -c container  
9 # 进入容器 [-c]
```

sh 复制代码



```
13 # 运行一个shell, use for debugging  
14 kubectl run -i --tty busybox --image=busybox --restart=Never
```

```
1 # 将本地 docker 与 K8S 依赖的 docker 进行绑定  
2 eval $(minikube docker-env)  
3 # 取消与 minikube 中的 docker 进行绑定  
4 eval $(minikube docker-env -u)  
5  
6 # 导出导入镜像  
7 docker save -o /opt/package/docker/image/fobgochod-admin.tar  
8 docker load -i /opt/package/docker/image/fobgochod-admin.tar
```

```
1 # 导出已有的Pod为YAML格式  
2 kubectl get deployment nginx-deploy -o yaml  
3 kubectl get deployment nginx-deploy -o yaml > nginx.yaml  
4 # 使用dry-run干跑模式导出YAML  
5 kubectl create deployment test --image=nginx --dry-run -o yaml  
6  
7 # deployment.yaml , 主要有以下五个字段 , 而需要自己维护的实际只有4个  
8  
9  
10 # 1. apiVersion : 定义API组名和版本 , 如v1。  
11 kubectl api-versions #查看K8S所支持API版本  
12  
13 # 2. kind : 定义资源类别 , 要创建的是POD就写为pod、Deployment、StatefulSet等  
14 kubectl api-resources #查看K8S所有资源类型  
15  
16 # 3. metadata : 元数据信息 , 包含资源名称、namespace等。namespace是必填项  
17 kubectl get namespaces #查看namespace  
18 kubectl create namespace myns #创建一个名为myns的namespace  
19  
20 # 4. spec (核心) : 声明资源的属性状态 , 也就是说希望deployment是什么样的  
21  
22  
23 # 5. status (核心) : 资源当前状态 , 应该与spec接近才对 , 本字段无需配置  
24  
25 # 查看每个字段支持的子字段  
26 kubectl explain deployment.metadata  
27 kubectl explain deployment.spec.template.spec.containers
```



[在 GitHub 上编辑此页](#)

总字数: 519 字 上次更新: 2021-11-22 23:56:32

← [Install](#)

[Commands](#) →



Commands

Linux命令大全：<http://man.linuxde.net>

常用命令

ps

```
1 # ps: process status  
2 # System V 风格  
3 ps -ef | grep java  
4 # BSD风格  
5 ps -aux | grep java  
6  
7 # 删除执行中的程序或工作  
8 kill -9 pid
```

sh 复制代码

```
1 # 查看某一端口的占用情况  
2 lsof -i:8080  
3 netstat -tunlp | grep 8080
```

sh 复制代码

压测

```
1 # 修改TIME_WAIT超时时间(建议小于30秒)  
2 vi /etc/sysctl.conf  
3 net.ipv4.tcp_fin_timeout = 30  
4 # 执行如下命令，使配置生效(-p 从指定的文件加载系统参数，如不指定即从  
5 sysctl -p  
6 # 查看当前系统中生效的所有参数  
7 sysctl -a  
8  
9 # Linux  
10 cat /proc/sys/net/ipv4/ip_local_port_range
```

sh 复制代码



```

14  sysctl -p
15
16  # 查看TCP连接状态的数量
17  netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
```

top

```

1  # 显示当前系统未使用的和已使用的内存数目
2  free -m
3  # top , uptime , w等命令都可以查看系统负载
4  top
5  uptime
6  w
7
8  [root@fobgochod ~]# top
9  # 敲击 F 进入编辑视图
```

[sh 复制代码](#)

Fields Management for window 1:Def, whose current sort field is PID

- Navigate with Up/Dn
- Right selects for move then 'Enter' or Left commits
- 'd' or 'Space' toggles display
- 's' sets sort
- Use 'q' or 'Esc' to end

- f -> 进入编辑界面
 - ↑ ↓ : 切换字段
 - → 选择字段 -> ↑ ↓ 移动字段 -> ← or Enter 确认
 - d or Space : 显示隐藏字段
 - s : 指定排序字段
 - q or Esc : 退出，完成设置

磁盘



```
3 # 以指定的区块大小来显示区块数目
4 df -B 1G
5 # 查看全部文件系统
6 df -a
7
8 # 显示当前目录的大小
9 du -sh
10 # 显示某个目录或文件的大小
11 du -sh dirName
12 # 显示当前目录下所有文件的大小
13 du -sh ./*
14 # 显示mysql所有数据库文件大小
15 du -sh /var/lib/mysql/*
```

mount

```
1 mount
2 # 将 /dev/hda1 挂在 /mnt 之下
3 #mount /dev/hda1 /mnt
4
5
6 umount
7 # 通过设备名卸载
8 umount -v /dev/sda1
9 /dev/sda1 umounted
10 # 通过挂载点卸载
11 umount -v /mnt/mymount/
12 /tmp/diskboot.img umounted
```

sh 复制代码

Yum

全称为 Yellow dog Updater, Modified 是一个在Fedora和RedHat以及 CentOS中的Shell前端软件包管理器

YUM的配置方式是基于分段配置的



```
3 # Yum的片段配置：  
4 /etc/yum.repos.d/*.repo
```

说明

- 若无@或不是install，则表示尚未安装
- base，表示未安装，包位于base仓库中
- updates，表示未安装，包位于updates仓库中
- -y，当安装过程提示选择全部为"yes"

```
1 # yum安装：  
2 yum install packageName  
3 # yum卸载：  
4 yum -y remove packageName  
5  
6 # 查看yum仓库中指定包名的软件包，可以使用通配符  
7 yum list all mariadb*  
8  
9 # 只显示已安装的包  
10 yum list installed  
11 # 只显示没有安装，但可安装的包  
12 yum list available  
13 # 查看所有可更新的包  
14 yum list updates  
15  
16 # 显示不属于任何仓库的，额外的包  
17 yum list extras  
18 # 显示被废弃的包  
19 yum list obsoletes  
20 # 新添加进yum仓库的包  
21 yum list recent
```

sh 复制代码

lrzs

文件传输



- rz中的r意为received（接收），告诉客户端，我（服务器）要接收文件 received by client，就等同于客户端在上传。

安装

```
1 # 查看  
2 yum list all lrzs  
3 # 安装  
4 yum install -y lrzs.x86_64
```

sh 复制代码

上传下载

```
1 # 不覆盖原文件  
2 rz  
3 # 覆盖原文件  
4 rz -y  
5  
6 # 下载一个文件  
7 sz filename  
8 # 下载多个文件  
9 sz filename1 filename2 ...
```

sh 复制代码

hostname

```
1 # 修改主机名  
2 hostname  
3  
4 vi /etc/sysconfig/network  
5 NETWORKING=yes  
6 HOSTNAME=fobgochod  
7 # 重启  
8 reboot  
9 # 查看  
10 hostnamectl  
11 cat /etc/hostname  
12 cat /etc/hosts
```

sh 复制代码



IP

```
1 # 查看
2 ifconfig
3 ip addr show
4
5 # 修改
6 vi /etc/sysconfig/network-scripts/ifcfg-eth0
7
8 DEVICE=eth0
9 ONBOOT=yes # 开机启动
10 IPADDR=172.16.2.141 # IP
11 PREFIX=24
12 GATEWAY=172.16.2.1 # 网关
13 DNS1=172.16.1.250 # DNS
14 IPV6_PRIVACY=no
15 ZONE=public
```

[复制代码](#)

其它

```
1 # 操作系统
2 cat /etc/redhat-release
3 # 内核
4 uname -s
5 # 内核版本
6 uname -r
7 # 硬件架构
8 uname -i
9 # 主机名称
10 uname -n
11 hostname
12 # CPU信息
13 cat /proc/cpuinfo
14 # 内存信息
15 cat /proc/meminfo
```

[复制代码](#)



```
3 # 下载  
4 wget https://downloads.mariadb.org/interstitial/mariadb-10.5.11-1.el7.x86_64.rpm  
5 wget https://downloads.mariadb.org/interstitial/mariadb-10.5.11-1.el7.x86_64.rpm.asc
```

```
1 # CPU型号  
2 cat /proc/cpuinfo | grep 'model name' | uniq  
3 # CPU个数  
4 cat /proc/cpuinfo | grep "physical id" | uniq | wc -l  
5 # CPU核数  
6 cat /proc/cpuinfo | grep "cpu cores" | uniq  
7 # 内存  
8 cat /proc/meminfo | grep MemTotal  
9 # CPU大小  
10 cat /proc/cpuinfo | grep 'model name' && cat /proc/cpuinfo  
11  
12 # 查看cpu 核数命令  
13 grep 'model name' /proc/cpuinfo | wc -l
```

```
1 systemctl enable redis  
2 systemctl list-unit-files | grep enable  
3  
4 chkconfig --list  
5 chkconfig redis off
```

[在 GitHub 上编辑此页](#)

总字数: 1,221 字 上次更新: 2021-11-22 23:56:32

← Commands

文件目录操作 →



文件目录操作

cd

1	cd	进入用户主目录；	sh 复制代码
2	cd ~	进入用户主目录；	
3	cd -	返回进入此目录之前所在的目录；	
4	cd .	还在当前目录	
5	cd ..	返回上级目录（若当前目录为“/”，则执行完后还在“/”；“..”为上级	
6	cd /	进入根目录（“/”）	
7			
8	pwd	查看当前目录	

vi

命令模式（Command mode）

1	a : 在当前字符后添加文本；	sh 复制代码
2	A : 在行末添加文本；	
3	i : 在当前字符前插入文本；	
4	I : 在行首插入文本；	
5	o : 在当前行后面插入一空行；	
6	O : 在当前行前面插入一空行；	
7		
8	x或X : 删除一个字符，x删除光标后的，而X删除光标前的；	
9	D : 删除从当前光标到光标所在行尾的全部字符；	
10	dd : 删除光标行正行内容	
11		
12	# 如果修改过，保存当前文件，然后退出！效果等同于（保存并退出）	
13	ZZ	
14	# 不保存，强制退出。效果等同于 :q!	
15	ZQ	
16		
17	# 翻页	
18	Ctrl+u : 向文件首翻半屏；	



输入模式 (Insert mode)

```
1 # 从编辑模式切换到命令模式  
2 Esc  
3 # 回车键，换行  
4 Enter  
5 # 退格键，删除光标前一个字符  
6 Backspace  
7 # 删除键，删除光标后一个字符  
8 DEL  
9 # 移动光标到行首/行尾  
10 HOME/END  
11 # 上/下翻页  
12 Page Up/Page Down
```

sh 复制代码

底线命令模式 (Last line mode)

```
1 # 保存并退出  
2 :wq  
3  
4 # 保存  
5 :w  
6  
7 # 强制保存  
8 :w!  
9  
10 # 退出  
11 :q  
12  
13 # 强制退出  
14 :q!  
15  
16 # 显示行号  
17 :set nu  
18  
19 # 不显示行号
```

sh 复制代码



```
23 :行号
24
25 # 光标跳转到最后一行的行首
26 :$ 
27
28 # 从当前光标所在位置开始向文件尾部查找
29 # 小写n匹配下一个
30 /word
31
32 # 从当前光标所在位置开始向文件头部查找
33 ?word
```

复制

复制一行

- 把光标移动到要复制的行上 按yy
- 把光标移动到要复制的位置 按p

从光标行复制到最后一行

\$代表最后一行，也可以直接输入行号数字 .,10/y

- 光标定位到起始行
- shift+: 进入命令模式
- 依次输入 .,\$y (.光标位置 \$代表最后一行 y复制)
- 按回车 完成copy
- 光标移动到要复制的位置 按p 粘贴

批量替换(#->\$)

- 光标定位到起始行
- shift+: 进入命令模式
- 依次输入 .,\$s/#/\$/ (.光标位置 \$代表最后一行 s替换 把#替换成\$)
- 按回车 完成替换



cat、more、less

```
1 cat -n /etc/profile | more  
2  
3 # -n 显示行号  
4  
5 # 按Enter键：下一行  
6 # 按Space键：下一页  
7 # 按Q键：退出  
8  
9 cat -n /etc/profile | less  
10  
11 # 用PageUp键向上翻页，用PageDown键向下翻页  
12 # 按Enter键：下一行。  
13 # 按Q键：退出  
14  
15 # 查找指定内容PATH、"export PATH"，有空格要用""  
16 cat /etc/profile | grep PATH  
17 cat /etc/profile | grep "export PATH"
```

sh 复制代码

tail、head

```
1 # 查找文件的开头的内容  
2 # 前5行  
3 head -5 /etc/profile  
4 head -n +5 /etc/profile  
5 # 显示第一行到倒数5行  
6 head -n -5 /etc/profile  
7  
8 # -n 头部内容的行数  
9 # -v 显示文件名  
10  
11 # 查找文件的结尾的内容  
12 # 最后5行  
13 tail -5 /etc/profile  
14 tail -n -5 /etc/profile  
15 # 从第5行至文件末尾
```

sh 复制代码



```
19 # -v 显示文件名
```

统计

```

1 # 统计某个字符串出现的次数
2 grep -o objStr filename|wc -l
3
4 # 如果是多个字符串出现次数，直接用\| 链接起来
5 grep -o 'objStr1\|objStr2' filename|wc -l
6
7 # wc命令介绍：l表示行数； w表示英文单词数； m表示字符数
8 wc [-lwm]
9
10 # 统计home目录下文件/目录数(只查一级)
11 # 查找文件数量
12 ls -l /home | grep '^-' | wc -l
13 # 查找目录数量
14 ls -l /home | grep '^d' | wc -l
15
16 # 统计home目录下所有文件/目录数(递归查所有，含子子孙孙)
17 # 查找文件数量
18 ls -lR /home | grep '^-' | wc -l
19 # 查找目录数量
20 ls -lR /home | grep '^d' | wc -l

```

[sh 复制代码](#)

directory

```

1 # 批量创建目录
2 mkdir -p /opt/{install,package,source/{backend,frontend}}
3
4 # 远程拷贝source目录到node02相同路径
5 [root@node01 opt]# scp -r ./source/ node02:`pwd`/
6 # 远程拷贝profile文件到node02的etc目录
7 [root@node01 ~]# scp /etc/profile node02:/etc

```

[sh 复制代码](#)



```
3 lost connection
4
5 [root@localhost zookeeper]# scp -r ./apache-zookeeper-3.6.3
6 ssh_exchange_identification: Connection closed by remote host
7 lost connection
8
9
10 vi /etc/hosts.allow
11 ## 允许所有ip主机均能连接本机
12 sshd: ALL
13 ## 或者指定网段
14 sshd:192.168.*:allow
15 sshd:172.16.*:allow
16 sshd:10.*:allow
17 # 重启sshd
18 systemctl restart sshd
```

[在 GitHub 上编辑此页](#)

总字数: 1,298 字 上次更新: 2021-11-22 23:56:32

[← Commands](#)

[用户和组 →](#)



用户和组

用户

```
1 # 查看用户  
2 cat /etc/passwd  
3 cat /etc/gshadow  
  
1 # 查看当前用户  
2 whoami  
3 who am i  
  
1 groupadd test-group  
2  
3 # 添加用户到组test-group  
4 useradd -g test-group test-user && cat /etc/passwd | grep t  
5 useradd -g test-group -u 1111 test-user && cat /etc/passwd  
6  
7 # 修改用户  
8 useradd -g test-group test-user  
9 usermod -l test-user-new test-user && cat /etc/passwd | grep  
10  
11 # 删除用户  
12 # 删除用户zhou , 但不删除其家目录及文件 ;  
13 userdel test-user  
14 # 删除用户zhou , 及其目录及文件一并删除 ;  
15 userdel -r test-user
```

复制代码

sh 复制代码

sh 复制代码

组

```
1 # 查看组  
2 cat /etc/group  
3
```

sh 复制代码



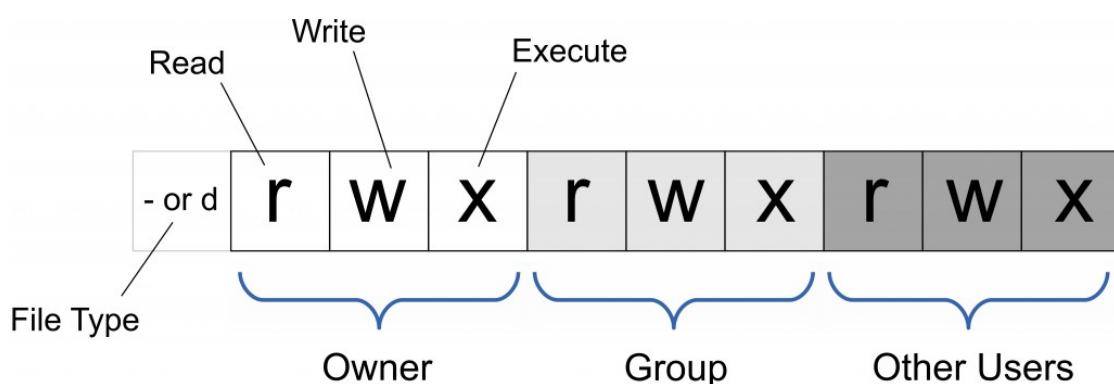
```

1 groupadd test-group
2 groupdel test-group
3
4 # -g : 指定新建工作组的id ;
5 groupadd -g 1111 test-group
6 cat /etc/group | grep test-group
7 groupdel test-group
8
9
10 # 添加测试组
11 groupadd test-group && cat /etc/group | grep test-group
12 # 修改组id
13 groupmod -g 1111 test-group && cat /etc/group | grep test-g
14 # 修改组名称
15 groupmod -n test-group-new test-group && cat /etc/group | g
16 # 删除新测试组
17 groupdel test-group-new
18
19 # 查看
20 groups
21 # 查看用户nobody所在组
22 groups nobody

```

sh 复制代码

权限

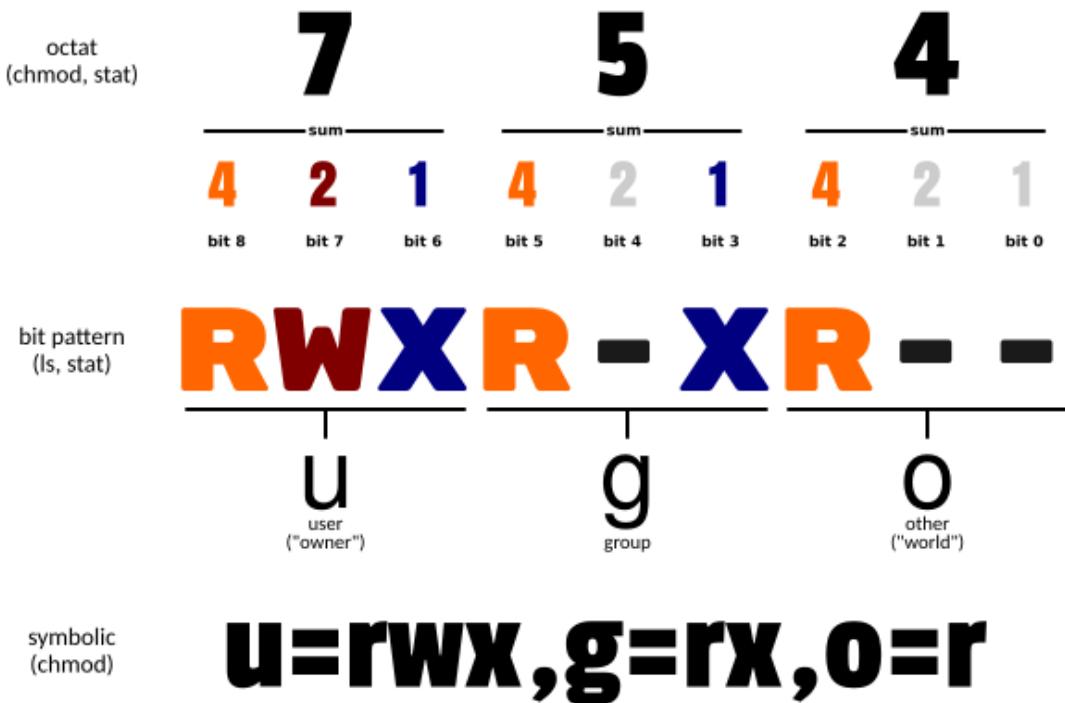


```

1 # 修改文件权限 用户 组
2 vi test
3 chmod +x test
4 chmod 755 test

```

sh 复制代码



[在 GitHub 上编辑此页](#)

总字数: 291 字 上次更新: 2021-11-22 23:56:32

← 文件目录操作

chkconfig、systemctl →



chkconfig、systemctl

<https://www.cnblogs.com/yadongliang/p/12561541.html>

CentOS 7.x开始，CentOS开始使用systemd服务来代替daemon，原来管理系统启动和管理系统服务的相关命令全部由systemctl命令来代替。

service 命令与 systemctl 命令对比

daemon命令	systemctl命令	说明
service [服务] start	systemctl start [unit type]	启动服务
service [服务] stop	systemctl stop [unit type]	停止服务
service [服务] restart	systemctl restart [unit type]	重启服务

此外还是二个systemctl参数没有与service命令参数对应

- status：参数来查看服务运行情况
- reload：重新加载服务，加载更新后的配置文件（并不是所有服务都支持这个参数，比如network.service）

chkconfig 命令与 systemctl 命令对比

daemon命令	systemctl命令	说明
chkconfig [服务] on	systemctl enable [unit type]	设置服务开机启动
chkconfig [服务] off	systemctl disable [unit type]	设备服务禁止开机启动
chkconfig --list	systemctl list-unit-files	查看系统上所有的服务



← 用户和组

防火墙 →



防火墙

firewall-cmd

安装

```
1  yum install -y firewalld          复制代码  
  
1  # 开启服务  
2  systemctl start firewalld  
3  # 关闭防火墙  
4  systemctl stop firewalld  
5  
6  # 开机自动启动  
7  systemctl enable firewalld  
8  # 关闭开机制动启动  
9  systemctl disable firewalld      sh 复制代码
```

常用命令

```
1  # 查看版本  
2  firewall-cmd --version           sh 复制代码  
3  # 查看帮助  
4  firewall-cmd --help  
5  # 显示状态(关闭-notrunning , 开启-running)  
6  firewall-cmd --state  
7  # 查看防火墙规则  
8  # 只显示/etc/firewalld/zones/public.xml中防火墙策略  
9  firewall-cmd --list-all  
10 # 显示/etc/firewalld/zones/下的所有策略  
11 firewall-cmd --list-all-zones  
12 # 更新防火墙规则  
13 firewall-cmd --reload  
14 # 查看区域信息
```



```

18 # 拒绝所有包
19 firewall-cmd --panic-on
20 # 取消拒绝状态
21 firewall-cmd --panic-off
22 # 查看是否拒绝
23 firewall-cmd --query-panic

```

帮助

```

1 # 查看帮助
2 man firewall-cmd | grep query-port
3 # 列出支持的zone
4 firewall-cmd --get-zones
5 # 支持的服务
6 firewall-cmd --get-services

```

[sh 复制代码](#)

参数解释

- `--permanent` #永久生效，没有此参数重启后失效
- `--zone` #作用域
- `--add-service` #添加的服务
- `--add-port=80/tcp` #添加端口，格式为：端口/通讯协议

端口

```

1 # 查看所有打开的端口 [--permanent] [--zone=zone] --list-ports
2 firewall-cmd --list-ports
3 firewall-cmd --permanent --list-ports
4 # 开启端口-永久
5 firewall-cmd --permanent --zone=public --add-port=3306/tcp
6 firewall-cmd --permanent --zone=public --add-port=8080-8082
7 # 删除端口
8 firewall-cmd --permanent --zone=public --remove-port=3306/tcp
9 firewall-cmd --permanent --zone=public --remove-port=8080-8082
10 # 重新载入配置
11 firewall-cmd --reload

```

[sh 复制代码](#)



服务

```
1 # 查看所有打开的服务 [--permanent] [--zone=zone] --list-services      sh 复制代码
2 firewall-cmd --list-services
3 # 开启服务
4 firewall-cmd --permanent --zone=public --add-service=http
5 # 删除服务
6 firewall-cmd --permanent --zone=public --remove-service=smt|sh 复制代码
7
8 # 设置某个ip访问某个端口
9 firewall-cmd --permanent --add-rich-rule='rule family="ipv4" port port=8080'
10 # 删除配置
11 firewall-cmd --permanent --remove-rich-rule 'rule family=ip'
12
13 # 设置某个ip访问某个服务
14 firewall-cmd --permanent --zone=public --add-rich-rule='rule service=8080'
15 # 删除配置
16 firewall-cmd --permanent --zone=public --remove-rich-rule='rule service=8080'|sh 复制代码
```

iptables(Centos7以下版本)

1.开放8080端口

```
1 /sbin/iptables -I INPUT -p tcp --dport 8080 -j ACCEPT      sh 复制代码
```

2.保存

```
1 /etc/rc.d/init.d/iptables save      sh 复制代码
```

3.查看打开的端口

```
1 chkconfig --list      sh 复制代码
2
3 /etc/init.d/iptables status
```

[SST 复制代码](#)

```
1 # 永久性生效，重启后不会复原  
2 # 开启  
3 chkconfig iptables on  
4 # 关闭  
5 chkconfig iptables off  
6  
7 # 即时生效，重启后复原  
8 # 开启  
9 service iptables start  
10 # 关闭  
11 service iptables stop
```

[在 GitHub 上编辑此页](#)

总字数: 551 字 上次更新: 2021-11-22 23:56:32

[← chkconfig、systemctl](#)[Windows →](#)



Windows

常用

bat

```
1 # 退回上一级目录
2 cd..
3 # 退回到根目录
4 cd\

5
6 # md 目录名 建立指定文件夹
7 md d:\file

8
9 # rd 目录名 删除指定文件夹
10 rd d:\file

11
12 # 清屏
13 cls

14
15 # 复制
16 copy 路径\文件名 路径\文件名

17
18 # 剪切
19 move 路径\文件名 路径\文件名

20
21 # 删除D:\file目录下文件1.txt
22 del d:\file\1.txt /p
23 del d:\file\1.txt

24
25 # 删除当前文件夹下所有文件
26 del *.*

27
28 # 删除D盘file文件夹下所有文件
29 del d:\file\
30 del d:\file\*
31 del d:\file\*.*

32
```



```
36  
37 # 重命名  
38 ren 旧文件名 新文件名  
39  
40 # 注释  
41 rem 注释内容  
42 :: 注释内容  
43  
44 # 暂停  
45 pause  
46  
47 # 日期  
48 date  
49 time  
50  
51 # 查看命令帮助  
52 help  
53 help cd  
54 cd /?  
55  
56 # 退出dos窗口  
57 exit
```

cmd

```
1 # 版本信息  
2 ver  
3 winver  
4  
5 # 查看系统信息  
6 systeminfo  
7  
8 # 调出任务管理器  
9 taskmgr  
10  
11 # 修改dos窗口标题  
12 title 我是标题  
13  
14 # 颜色属性由两个十六进制数字指定：第一个为背景，第二个则为前景。
```



```
18      2 = 绿色      A = 淡绿色
19      3 = 浅绿色    B = 淡浅绿色
20      4 = 红色      C = 淡红色
21      5 = 紫色      D = 淡紫色
22      6 = 黄色      E = 淡黄色
23      7 = 白色      F = 亮白色
24
25 # 背景红色
26 color 4
27 # 在亮白色上产生亮红色
28 color fc
29 # 恢复默认
30 color
31
32 # 改变窗口大小
33 mode con cols=80 lines=25
34
35
36 ping
37 net
38 telnet
39 ftp
40 netstat
41 nbtstat
42 tracert
43 ipconfig
```

命令

```
1 # 查看端口
2 netstat -ano
3 netstat -ano | findstr 8080
4
5 # 查看进程
6 tasklist | findstr java
7
8 # 结束进程
9 taskkill /f /t /im java.exe
```



快捷键

```
1 # 讲述人  
2 Windows+Enter  
3 # 我的电脑  
4 Windows+E  
5 # 运行  
6 Windows+R  
7 # 远程桌面连接  
8 mstsc  
9 # 服务  
10 services.msc  
11 # 注册表编辑器  
12 regedit  
13 # 打开系统属性  
14 sysdm.cpl  
15 # 查看win10系统版本信息  
16 dxdiag  
17 # 记事本  
18 notepad
```

sh 复制代码

路径

```
1 # 任务栏图片  
2 %APPDATA%\Microsoft\Internet Explorer\Quick Launch\User Pini  
3 # 开始菜单：  
4 %APPDATA%\Microsoft\Windows\Start Menu\Programs  
5 # hosts  
6 %windir%\System32\drivers\etc
```

压测

```
1 # 界面修改、重启生效  
2 regedit>计算机\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\S  
3 添加 DWORD(32位)值(D)
```

sh 复制代码



```
7   REG ADD "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WANP\Parameters" /v DynamicPortRange /t REG_DWORD /d 0x00000000 /f  
8   REG ADD "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WANP\Parameters" /v DynamicPortRange /t REG_DWORD /d 0x00000000 /f  
9   REG ADD "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WANP\Parameters" /v DynamicPortRange /t REG_DWORD /d 0x00000000 /f  
10  REG ADD "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WANP\Parameters" /v DynamicPortRange /t REG_DWORD /d 0x00000000 /f  
11  
12  # 查看动态端口范围  
13  netsh interface ipv4 show dynamicportrange protocol=tcp  
14  netsh int ipv4 show dynamicportrange tcp  
15  
16  # 设置动态端口  
17  netsh int ipv4 set dynamicport tcp start=1024 num=64512  
18  
19  # 查看TCP连接状态的数量  
20  netstat -an|find "ESTABLISHED" /c  
21  netstat -an|find "TIME_WAIT" /c
```

[在 GitHub 上编辑此页](#)

总字数: 681 字 上次更新: 2021-11-22 23:56:32

← 防火墙

学习中... →



学习中...

- 马士兵教育 [↗](#)
- 课程、资料链接 [↗](#)
- git.mashibing.com/bjmashibing [↗](#)
- 直播课预告链接 [↗](#)

学习计划

@formatter:off

多线程与高并发（进程内高并发）

1. 单机高并发应该掌握的线程基础：线程状态，异常与锁等 视频41 [↗](#)
2. 解析自旋锁CAS操作与volatile 视频46 [↗](#)
3. JUC包下AtomicXXX类与新的同步机制：Latch Semaphore等 视频52 [↗](#)
4. LockSupport，高频面试题，AQS源码，以及源码阅读方法论 视频58 [↗](#)
5. 强软弱虚四种引用以及ThreadLocal的原理与源码 视频62 [↗](#)
6. 线程池可用的各种高并发容器详解：CopyOnWriteList，BlockingQueue等 视频64 [↗](#)
7. 详解线程池：自定义线程池，JDK自带线程池，ForkJoin，源码解析等(一) 视频69 [↗](#)
8. 详解线程池：自定义线程池，JDK自带线程池，ForkJoin，源码解析等(二) 视频74 [↗](#)
9. 单机压测工具JMH，单机最快MQ - Disruptor原理解析 视频76 [↗](#)

JVM从入门到精通

1. JVM入门级class文件格式 视频80 [↗](#)
2. 详解Class加载过程 视频86 [↗](#)



5. Java运行时数据区和常用指令 视频102
6. JVM调优必备理论知识-GC Collector-三色标记 视频110
7. JVM调优实战 视频114
8. JVM实战调优 视频119
9. JVM实战调优 视频124
10. 垃圾回收算法串讲 视频128
11. JVM常见参数总结 视频132

高并发负载均衡

1. 网络协议原理 视频24
2. LVS的DR,TUN,NAT模型推导 视频25
3. LVS的DR模型试验搭建 视频26
4. 基于keepalived的LVS高可用搭建 视频28

Redis

1. redis 介绍及NIO原理介绍 视频43
2. redis的string类型&bitmap 视频44
3. redis的list、set、hash、sorted_set、skipList 视频45
4. redis的消息订阅、pipeline、事务、modules、布隆过滤器、缓存 LRU 视频48
5. redis的持久化RDB、fork、copyonwrite、AOF、RDB&AOF混合使用 视频49
6. redis的集群：主从复制、CAP、PAXOS、cluster分片集群01 视频50
7. redis的集群：主从复制、CAP、PAXOS、cluster分片集群02 视频53
8. redis开发：spring.data.redis、连接、序列化、high/low api 视频54

ZooKeeper



3. zookeeper案例：分布式配置注册发现、分布式锁、reactive模式编程
视频56

Java程序员应该掌握的底层知识

1. java程序员需要了解的底层知识第一课 视频150
2. 硬件和操作系统的底层知识 视频155
3. 操作系统之进程管理 视频158
4. Linux系统的内存映射 视频161
5. 内核同步方法及用汇编启动内核 视频165

内存与IO，磁盘IO，网络IO

1. 虚拟文件系统，文件描述符，IO重定向 视频166
2. 内核中PageCache、mmap作用、java文件系统io、nio、内存中缓冲区作用 视频174
3. Socket编程BIO及TCP参数 视频177
4. C10K问题及NIO精讲和IO模型性能压测 视频184
5. 网络编程之多路复用器及Epoll精讲 视频185
6. 网络编程java API 实战多路复用器开发 视频187
7. 全手写急速理解Netty模型及IO模型应用实战 视频189
8. Netty之IO模型开发本质手写部分实现推导篇 视频191
9. 全手写基于Netty的RPC框架自定义协议，连接池 视频196
10. 全手写基于Netty的RPC框架 协议编解码问题 粘包拆包与内核关系
视频198
11. 全手写基于Netty的RPC框架 provider端简单dispatcher实现RPC调用全流程 视频203
12. 全手写基于Netty的RPC框架 简单重构框架分层及RPC传输的本质及有无状态的RPC区别 视频207
13. 自定义HTTP协议解析和HTTPserver调用实现 视频216

MySQL调优



-
- 3. mysql调优--索引基本实现原理及索引优化 视频111 [↗](#)
 - 4. mysql调优--mysql索引优化实现细节 视频112 [↗](#)
 - 5. mysql调优--mysql查询优化分析 视频113 [↗](#)
 - 6. mysql调优--mysql分区设计及分区优化 视频115 [↗](#)
 - 7. mysql调优--mysql分区优化2及参数设计优化 视频116 [↗](#)
 - 8. mysql调优--mysql参数设计优化及总结 视频117 [↗](#)

[在 GitHub 上编辑此页 ↗](#)

总字数: 1,329 字 上次更新: 2021-11-22 23:56:32

← [Windows](#)

[Why技术](#) →



Why技术

推书

- [19]【荒腔走板】推荐十本书，全是原创书评。 ↗
- [34]《深入理解Java虚拟机》第2版挖的坑终于在第3版中被R大填平了 ↗
- [44]盘点那些在我文章中出现过的书籍|文末有福利哦 ↗
- [47]我告诉你这书的第3版到底值不值得买？ ↗
- [XX]没事，那啥，就通知一下，这一波羊毛该薅了。 ↗

多线程

- [03]有的线程它死了，于是它变成一道面试题 ↗
- [42]如何设置线程池参数？美团给出了一个让面试官虎躯一震的回答。 ↗
- [45]每天都在用，但你知道Tomcat的线程池有多努力吗？ ↗
- [60]笑了，面试官问我知不知道异步编程的Future。 ↗
- [61]呵呵，面试官问我知不知道CompletionService？ ↗
- [66]关于多线程中抛异常的这个面试题我再说最后一次！ ↗
- [98]面试官一个线程池问题把我问懵逼了。 ↗

Redis

- [X05]当周杰伦把QQ音乐干翻的时候，作为程序猿我看到了什么？ ↗
- [38]【求锤得锤的故事】Redis锁从面试连环炮聊到神仙打架。 ↗
- [40]面试时遇到『看门狗』脖子上挂着『时间轮』，我就问你怕不怕？ ↗

JVM

- [35]面试官：你说你熟悉jvm？那你讲一下并发的可达性分析 ↗
- [36]G1回收器：我怎么知道你是什么时候的垃圾？ ↗
- [37]面试官：你回去等通知吧！ ↗



面试

- [07]面试了15位来自985/211高校的2020届研究生之后的思考 [↗](#)
- [10]这道Java基础题真的有坑！我求求你，认真思考后再回答。 [↗](#)
- [12]这道Java基础题真的有坑！我也没想到还有续集。 [↗](#)
- [65]why哥被阿里一道基础面试题给干懵了，一气之下写出万字长文。 [↗](#)
- [X04]普通二本，毕业三年，北漂之后，我是怎么成为程序猿的。 [↗](#)

Dubbo

- [X01]Dubbo 2.7新特性之异步化改造 [↗](#)
- [11]参加Dubbo社区开发者日成都站后，带给我的一点思考。 [↗](#)
- [16]一文讲透Dubbo负载均衡之最小活跃数算法 [↗](#)
- [X17]Dubbo一致性哈希负载均衡的源码和Bug，了解一下？ [↗](#)
- [18]Dubbo加权轮询负载均衡的源码和Bug，了解一下？ [↗](#)
- [30]Dubbo 2.7.5在线程模型上的优化 [↗](#)
- [33]Dubbo Cluster集群那点你不知道的事。 [↗](#)
- [X50]吐血输出：2万字长文带你细细盘点五种负载均衡策略。 [↗](#)
- [63]没想到吧！关于Dubbo的『消费端线程池模型』官网也写错了。 [↗](#)
- [64]why哥这里有一道Dubbo高频面试题，请查收。 [↗](#)

荒腔走板

- [08]荒腔走板，我想在这个节日里推荐一本书 [↗](#)
- [13]订阅号做了77天，我挣了xxx元... [↗](#)
- [21]2013，我这一年。 [↗](#)
- [22]2014，我这一年。 [↗](#)
- [23]2015，我这一年。 [↗](#)
- [24]2016，我这一年。 [↗](#)
- [25]2017，我这一年。 [↗](#)
- [26]2018，我这一年。 [↗](#)
- [27]2019，我这一年。 [↗](#)
- [31]【荒腔走板】我的高三，绝地反击。 [↗](#)



- [49]是爱情呀。|狗粮、长文、谨慎点击。 ↗
- [54]生而不养，何以为家？ ↗
- [67]中国女排，猛男落泪|why哥出镜，视频彩蛋 ↗
- [68]我回到高中教室写下了：Hello World。 ↗
- [77]房租，强奸过每一个漂泊的人。 ↗
- [80]要不，瞅一眼why哥的家？ ↗
- [82]2020，我这一年。 ↗
- [84]我希望这才是35岁危机的真正原因。 ↗
- [92]卧槽，这年轻人。 ↗

Why

2019

- [X01]Dubbo 2.7新特性之异步化改造 ↗
- [X02]事务没回滚？来，我们从现象到原理一起分析一波！ ↗
- [X03]有的线程它死了，于是它变成一道面试题 ↗
- [X04]普通二本，毕业三年，北漂之后，我是怎么成为程序猿的。 ↗
- [X05]当周杰伦把QQ音乐干翻的时候，作为程序猿我看到了什么？ ↗
- [X06]当朋友圈疯狂要国旗的时候，作为程序猿我看到了什么？ ↗
- [07]面试了15位来自985/211高校的2020届研究生之后的思考 ↗
- [08]荒腔走板，我想在这个节日里推荐一本书 ↗
- [09]讲真，我发现这本书有个地方写错了！ ↗
- [10]这道Java基础题真的有坑！我求求你，认真思考后再回答。 ↗
- [11]参加Dubbo社区开发者日成都站后，带给我的一点思考。 ↗
- [12]这道Java基础题真的有坑！我也没想到还有续集。 ↗
- [13]订阅号做了77天，我挣了xxx元... ↗
- [14]很开心，在使用mybatis的过程中我踩到一个坑。 ↗
- [15]这道面试题我真不知道面试官想要的回答是什么 ↗
- [16]一文讲透Dubbo负载均衡之最小活跃数算法 ↗
- [X17]Dubbo一致性哈希负载均衡的源码和Bug，了解一下？ ↗
- [18]Dubbo加权轮询负载均衡的源码和Bug，了解一下？ ↗
- [19]【荒腔走板】推荐十本书，全是原创书评。 ↗
- [20]http请求中加号被替换为空格？源码背后的秘密 ↗



- [23]2015，我这一年。 ↗
- [24]2016，我这一年。 ↗
- [25]2017，我这一年。 ↗
- [26]2018，我这一年。 ↗
- [27]2019，我这一年。 ↗

2020

- [28]够强！一行代码就修复了我提的Dubbo的Bug。 ↗
- [29]快速失败机制&失败安全机制 ↗
- [30]Dubbo 2.7.5在线程模型上的优化 ↗
- [31]【荒腔走板】我的高三，绝地反击。 ↗
- [32]《代码整洁之道》&《程序员的职业素养》 ↗
- [33]Dubbo Cluster集群那点你不知道的事。 ↗
- [34]《深入理解Java虚拟机》第2版挖的坑终于在第3版中被R大填平了 ↗
- [35]面试官:你说你熟悉jvm?那你讲一下并发的可达性分析 ↗
- [36]G1回收器：我怎么知道你是什么时候的垃圾？ ↗
- [37]面试官：你回去等通知吧！ ↗
- [38]【求锤得锤的故事】Redis锁从面试连环炮聊到神仙打架。 ↗
- [39]一个困扰我122天的技术问题，我好像知道答案了。 ↗
- [40]面试时遇到『看门狗』脖子上挂着『时间轮』，我就问你怕不怕？ ↗
- [41]一个成都程序猿写于离开北京一周年与26岁生日的这一天。 ↗
- [42]如何设置线程池参数？美团给出了一个让面试官虎躯一震的回答。 ↗
- [43]对不起，我可能要放弃这个号了。 ↗
- [44]盘点那些在我文章中出现过的书籍|文末有福利哦 ↗
- [45]每天都在用，但你知道 Tomcat 的线程池有多努力吗？ ↗
- [46]【荒腔走板】纪录片《生活万岁》 ↗
- [47]我告诉你这书的第3版到底值不值得买？ ↗
- [48]mybatis开发，你用xml还是注解？我pick... ↗
- [49]是爱情呀。|狗粮、长文、谨慎点击。 ↗
- [X50]吐血输出：2万字长文带你细细盘点五种负载均衡策略。 ↗
- [51]mybatis逆向工程使用姿势不对，把表清空了，心里慌的一比，于是写了个插件。 ↗



- [53]我从LongAdder中窥探到了高并发的秘籍，上面只写了两个字...[↗](#)
- [54]生而不养，何以为家？[↗](#)
- [55]快来！我从源码中学习到了一招Dubbo的骚操作！[↗](#)
- [56]我的程序跑了60多小时，就是为了让你看一眼JDK的BUG导致的内存泄漏。[↗](#)
- [57]JDK的BUG导致的内存溢出！反正我是没想到还能有续集。[↗](#)
- [58]一个成都程序猿眼中的成都和天府软件园，先从蚂蚁金服说起...[↗](#)
- [59]我靠！Semaphore里面居然有这么一个大坑！[↗](#)
- [60]笑了，面试官问我知不知道异步编程的Future。[↗](#)
- [61]呵呵，面试官问我知不知道CompletionService？[↗](#)
- [62]听说你也想做公众号？我写了一年了，谈谈得与失。[↗](#)
- [63]没想到吧！关于Dubbo的『消费端线程池模型』官网也写错了。[↗](#)
- [64]why哥这里有一道Dubbo高频面试题，请查收。[↗](#)
- [65]why哥被阿里一道基础面试题给干懵了，一气之下写出万字长文。[↗](#)
- [66]关于多线程中抛异常的这个面试题我再说最后一次！[↗](#)
- [67]中国女排，猛男落泪|why哥出镜，视频彩蛋[↗](#)
- [68]我回到高中教室写下了：Hello World。[↗](#)
- [69]Doug Lea在J.U.C包里面写的BUG又被网友发现了。[↗](#)
- [70]这玩意比ThreadLocal叼多了，吓得why哥赶紧分享出来。[↗](#)
- [71]一人血书，想让why哥讲一下这道面试题。[↗](#)
- [72]why哥悄悄的给你说几个HashCode的破事。[↗](#)
- [73]请大家帮why哥看看，我交了多少智商税。[↗](#)
- [74]要我说，多线程事务它必须就是个伪命题！[↗](#)
- [75]这个Map你肯定不知道，毕竟存在感确实太低了。[↗](#)
- [76]面试官问我：什么是高并发下的请求合并？[↗](#)
- [77]房租，强奸过每一个漂泊的人。[↗](#)
- [78]一个基于运气的数据结构，你猜是啥？[↗](#)
- [X79]我叫你不要重试，你非得重试。这下玩坏了吧？[↗](#)
- [80]要不，瞅一眼why哥的家？[↗](#)
- [81]其实吧，LRU也就那么回事。[↗](#)
- [82]2020，我这一年。[↗](#)

2021



- [85]可以，很牛逼。 ↴
- [86]布隆，牛逼！布谷鸟，牛逼！ ↴
- [87]基础送分题，why哥只说这一次。 ↴
- [88]我一般在B站看这些破玩意... ↴
- [89]凉了呀，面试官叫why哥设计一个排行榜。 ↴
- [90]我错了，打我可以，别打脸。 ↴
- [91]我给Apache顶级项目贡献了点源码。 ↴
- [92]卧槽，这年轻人。 ↴
- [92]这个Bug的排查之路，真的太有趣了。 ↴
- [X93]有哪些道理是我当了程序员后才知道的？ ↴
- [XX]没事，那啥，就通知一下，这一波羊毛该薅了。 ↴
- [X94]生活，万岁！！！ ↴
- [X95]深夜的一点碎碎念，无关痛痒。 ↴
- [96]知乎的一次29.7元的咨询。 ↴
- [XX]哭了，假期好像啥都没干，就要结束了？！ ↴
- [97]当我看技术文章的时候，我在想什么？ ↴
- [98]面试官一个线程池问题把我问懵逼了。 ↴

[在 GitHub 上编辑此页](#) ↴

总字数: 3,193 字 上次更新: 2021-11-22 23:56:32

← 学习中...

博客 →



博客

美团技术团队 ↗

<https://tech.meituan.com> ↗

- 日志级别动态调整——小工具解决大问题 ↗
- 卫星系统——酒店后端全链路日志收集工具介绍 ↗
- 不可不说的Java“锁”事 ↗
- 从ReentrantLock的实现看AQS的原理及应用 ↗
- Java线程池实现原理及其在美团业务中的实践(美团技术团队) ↗

Idea Buffer ↗

<http://www.ideabuffer.cn> ↗

- 深入理解Java线程池：ThreadPoolExecutor ↗

其它

- GC垃圾回收——总结 ↗

[在 GitHub 上编辑此页](#) ↗

总字数: 145 字 上次更新: 2021-11-22 23:56:32

← Why技术

On Java 8 →



On Java 8

书籍简介

- 本书作者为 [美] Bruce Eckel，即《Java 编程思想》的作者。
- 本书是事实上的《Java 编程思想》第五版。
- 《Java 编程思想》第四版基于 JAVA 5 版本；《On Java 8》 基于 JAVA 8 版本。

类初始化和加载

初始化和加载

Java中每个类的编译代码都存在于它自己独立的文件中。该文件只有在使用程序代码时才会被加载。一般可以说“类的代码在首次使用时加载”。这通常是指创建类的第一个对象，或者是访问了类的 static 属性或方法。构造器也是一个 static 方法尽管它的 static 关键字是隐式的。因此，准确地说，一个类当它任意一个 static 成员被访问时，就会被加载。

首次使用时就是 static 初始化发生时。所有的 static 对象和 static 代码块在加载时按照文本的顺序（**在类中定义的顺序**）依次初始化。static 变量只被初始化一次。

在类中变量定义的顺序决定了它们初始化的顺序。即使变量定义散布在方法定义之间，它们仍会在任何方法（包括构造器）被调用之前得到初始化。

概括一下创建对象的过程，假设有个名为 Dog 的类：

1. 即使没有显式地使用 static 关键字，构造器实际上也是静态方法。所以，当首次创建 Dog 类型的对象或是首次访问 Dog 类的静态方法或属性时，Java



态初始化的所有动作都会执行。因此，静态初始化只会在首次加载 Class 对象时初始化一次。

3. 当用 new Dog() 创建对象时，首先会在堆上为 Dog 对象分配足够的存储空间。
4. 分配的存储空间首先会被清零，即会将 Dog 对象中的所有基本类型数据设置为默认值（数字会被置为 0，布尔型和字符型也相同），引用被置为 null。
5. 执行所有出现在字段定义处的初始化动作。
6. 执行构造器。你将会在“复用”这一章看到，这可能会牵涉到很多动作，尤其当涉及继承的时候。

```
1  class Insect {  
2  
3      static {  
4          System.out.println("Insect static block");  
5      }  
6  
7      private static int x1 = printInit("static Insect.x1 ini-  
8          static int x3 = 3;  
9          protected int x5 = printInit("Insect.x5 initialized");  
10         public int x7;  
11  
12     Insect() {  
13         System.out.println("Insect constructor");  
14         System.out.println("Insect.x5 = " + x5);  
15         System.out.println("Insect.x7 = " + x7);  
16         this.x7 = 7;  
17     }  
18  
19     static int printInit(String s) {  
20         System.out.println(s);  
21         return 47;  
22     }  
23 }  
24  
25 /**  
26 * The full process of initialization  
27 * <p>  
28 * 1、父类静态代码块、静态变量按文本顺序初始化  
29 * 2、子类静态代码块、静态变量按文本顺序初始化
```



```
32
33     * 6、子类构造函数
34     *
35     * @author zhouxiao
36     * @date 2020/8/1
37     */
38     public class Beetle extends Insect {
39
40         public static void main(String[] args) {
41             System.out.println("***** main method begin ***");
42             Beetle b = new Beetle();
43         }
44
45         private static int x2 = printInit("static Beetle.x2 ini");
46         static int x4 = 6;
47         protected int x6 = printInit("Beetle.x6 initialized");
48         public int x8 = 8;
49
50         public Beetle() {
51             super();
52             System.out.println("Beetle constructor");
53             System.out.println("Beetle.x6 = " + x6);
54             System.out.println("Beetle.x8 = " + x8);
55             System.out.println("Insect.x7 = " + x7);
56         }
57
58         static {
59             System.out.println("Beetle static block");
60         }
61     }
```

输出：

```
1  Insect static block
2  static Insect.x1 initialized
3  static Beetle.x2 initialized
4  Beetle static block
5  ***** main method begin *****
6  Insect.x5 initialized
7  Insect constructor
```

[复制代码](#)



```
11  Beetle constructor  
12  Beetle.x6 = 47  
13  Beetle.x8 = 8  
14  Insect.x7 = 7
```

[在 GitHub 上编辑此页](#)

总字数: 998 字 上次更新: 2021-11-22 23:56:32

← 博客

深入理解Java虚拟机（第3版）→



深入理解Java虚拟机(第3版)

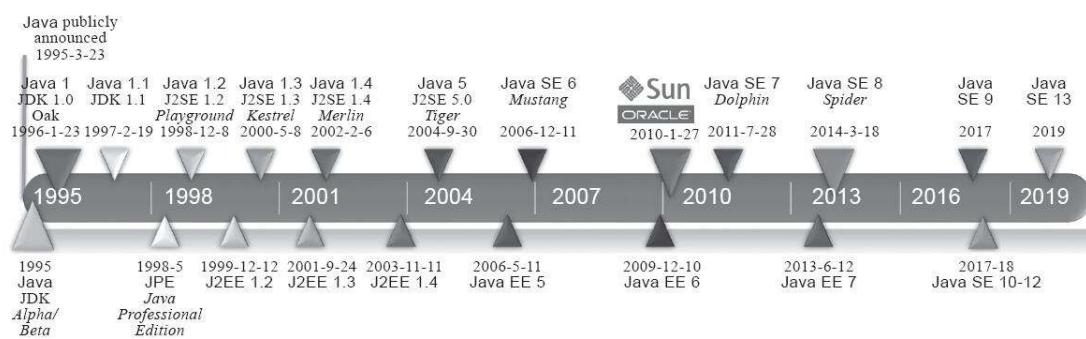
书籍简介

- 本书作者为 [中] 周志明。
- 本书是《深入理解Java虚拟机》系列第3版。
- 这是一部从工作原理和工程实践两个维度深入剖析JVM的著作。

RednaxelaFX

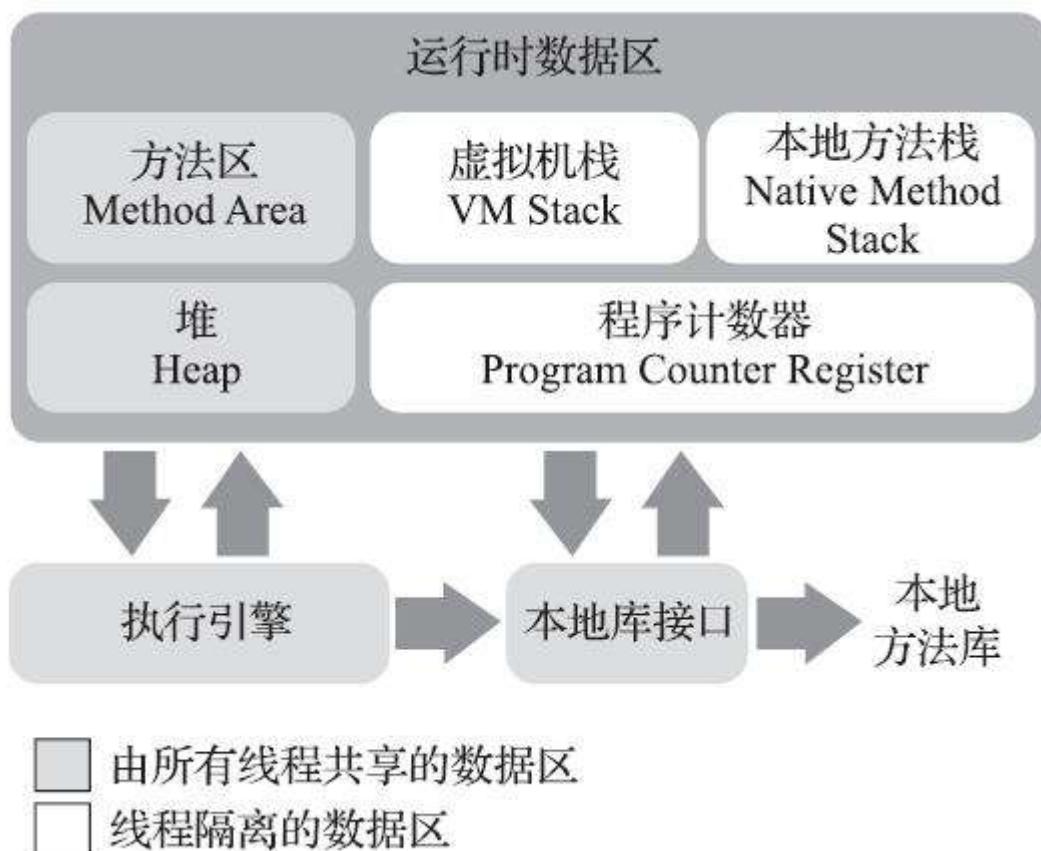
- Major GC和Full GC的区别是什么 ↗
- JVM full GC的奇怪现象 ↗
- JVM默认老年代回收是 PSMarkSweep(Serial-Old) 还是Parallel Old ↗

走近Java



自动内存管理

Java与C++之间有一堵由内存动态分配和垃圾收集技术所围成的高墙，墙外面的人想进去，墙里面的人却想出来。Java与C++之间有一堵由内存动态分配和垃圾收集技术所围成的高墙，墙外面的人想进去，墙里面的人却想出来。



- 程序计数器
 - 线程私有
 - 如果线程正在执行的是一个Java方法，这个计数器记录的是正在执行的虚拟机字节码指令的地址
 - 如果正在执行的是本地（Native）方法，这个计数器值则应为空（Undefined）
 - 唯一一个在《Java虚拟机规范》中没有规定任何OutOfMemoryError情况的区域
- Java虚拟机栈
 - 线程私有，生命周期与线程相同
 - 虚拟机栈描述的是Java方法执行的线程内存模型：每个方法被执行的时候，Java虚拟机都会同步创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态连接、方法出口等信息。每一个方法被调用直至执行完毕的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。
 - 变量槽（Slot）



- 如果Java虚拟机栈容量可以动态扩展，当栈扩展时无法申请到足够的内存会抛出OutOfMemoryError异常
- 本地方法栈
 - 虚拟机栈为虚拟机执行执行Java方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到本地（Native）方法服务
 - 譬如Hot-Spot虚拟机直接就把本地方法栈和虚拟机栈合二为一
 - 与虚拟机栈一样，本地方法栈也会在栈深度溢出或者栈扩展失败时分别抛出StackOverflowError和OutOfMemoryError异常
- Java堆（GC堆）
 - 被所有线程共享
 - Java世界里“几乎”所有的对象实例都在这里分配内存
 - 由于现代垃圾收集器大部分都是基于分代收集理论设计的，所以Java堆中经常会出现“新生代”“老年代”“永久代”“Eden空间”“From Survivor空间”“To Survivor空间”等名词
 - 在十年之前（以G1收集器的出现为分界），作为业界绝对主流的HotSpot虚拟机，它内部的垃圾收集器全部都基于“经典分代”来设计，需要新生代、老年代收集器搭配才能工作
 - Java堆既可以被实现成固定大小的，也可以是可扩展的，不过当前主流的Java虚拟机都是按照可扩展来实现的（通过参数-Xmx和-Xms设定）
 - 如果在Java堆中没有内存完成实例分配，并且堆也无法再扩展时，Java虚拟机将会抛出OutOfMemoryError异常
- 方法区（Method Area）
 - 被所有线程共享
 - 在JDK 6的时候HotSpot开发团队就有放弃永久代，逐步改为采用本地内存（Native Memory）来实现方法区的计划了
 - 到了JDK 7的HotSpot，已经把原本放在永久代的字符串常量池、静态变量等移出
 - 而到了JDK 8，终于完全废弃了永久代的概念，改用与JRockit、J9一样在本地内存中实现的元空间（Meta-space）来代替，把JDK 7中永久代还剩余的内容（主要是类型信息）全部移到元空间中。
 - 根据《Java虚拟机规范》的规定，如果方法区无法满足新的内存分配需求时，将抛出OutOfMemoryError异常
- 运行时常量池（Runtime Constant Pool）
 - 是方法区的一部分



- 并不是虚拟机运行时数据区的一部分
- 但是这部分内存也被频繁地使用，而且也可能导致OutOfMemoryError异常出现

HotSpot虚拟机对象探秘

- 给对象分配内存
 - 指针碰撞 (Bump The Pointer)
 - 空闲列表 (Free List)
- 本地线程分配缓冲 (Thread Local Allocation Buffer , TLAB)
- 对象的访问定位
 - 句柄访问
 - 直接指针

经典垃圾收集器

术语

- 垃圾收集器 (Garbage Collection , 简称GC)
- 判断对象是否存活算法
 - 引用计数算法 (Reference Counting) : 微软COM技术 (Component Object Model)
 - 可达性分析算法 (Reachability Analysis) : 主流Java虚拟机
 - 字符串常量池 (String Table)
 - GC Roots集合
- 分代收集和局部回收(Partial GC)
- 从如何判定对象消亡的角度出发，垃圾收集算法可分为
 - 引用计数式垃圾收集 (Reference Counting GC) , 又称：直接垃圾收集
 - 追踪式垃圾收集 (Tracing GC) , 又称：间接垃圾收集
- 当前商业虚拟机的垃圾收集器大多遵循了"分代收集" (Generational Collection) 的理论进行设计
 - 两个分代假说
 - 弱分代假说 (Weak Generational Hypothesis) : 绝大多数对象都是朝生夕灭的



- 第三条经验法则
 - 跨代引用假说 (Intergenerational Reference Hypothesis) : 跨代引用相对于同代引用来说仅占极少数
- Java堆划分区域
 - 新生代 (Young Generation) 和老年代 (Old Generation) : HotSpot虚拟机，也是现代业界主流命名方式
 - 婴儿区 (Nursery) 和长存区 (Tenured) : IBM J9虚拟机
- 分代统一定义
 - 部分收集 (Partial GC) : 指目标不是完整收集整个Java堆的垃圾收集，其中又分为：
 - 新生代收集 (Minor GC/Young GC) : 指目标只是新生代的垃圾收集。
 - 老年代收集 (Major GC/Old GC) : 指目标只是老年代的垃圾收集。目前只有CMS收集器会有单独收集老年代的行为。另外请注意“Major GC”这个说法现在有点混淆，在不同资料上常有不同所指，读者需按上下文区分到底是指老年代的收集还是整堆收集。
 - 混合收集 (Mixed GC) : 指目标是收集整个新生代以及部分老年代的垃圾收集。目前只有G1收集器会有这种行为。
 - 整堆收集 (Full GC) : 收集整个Java堆和方法区的垃圾收集。
- 垃圾收集器算法
 - 标记-清除算法 (Mark-Sweep)
 - 标记-复制算法
 - 半区复制 (Semispace Copying)
 - 分配担保 (Handle Promotion)
 - 标记-整理算法 (Mark-Compact)
- 并发标记对象消失问题
 - 两个条件同时满足，才会产生“对象消失”问题
 - 赋值器插入了一条或多条从黑色对象到白色对象的新引用；
 - 赋值器删除了全部从灰色对象到该白色对象的直接或间接引用。
 - 两种解决方案
 - 增量更新 (Incremental Update) : CMS
 - 原始快照 (Snapshot At The Beginning , SATB) : G1、Shenandoah
- 衡量垃圾收集器的三项最重要的指标 (不可能三角)
 - 内存占用 (Footprint)

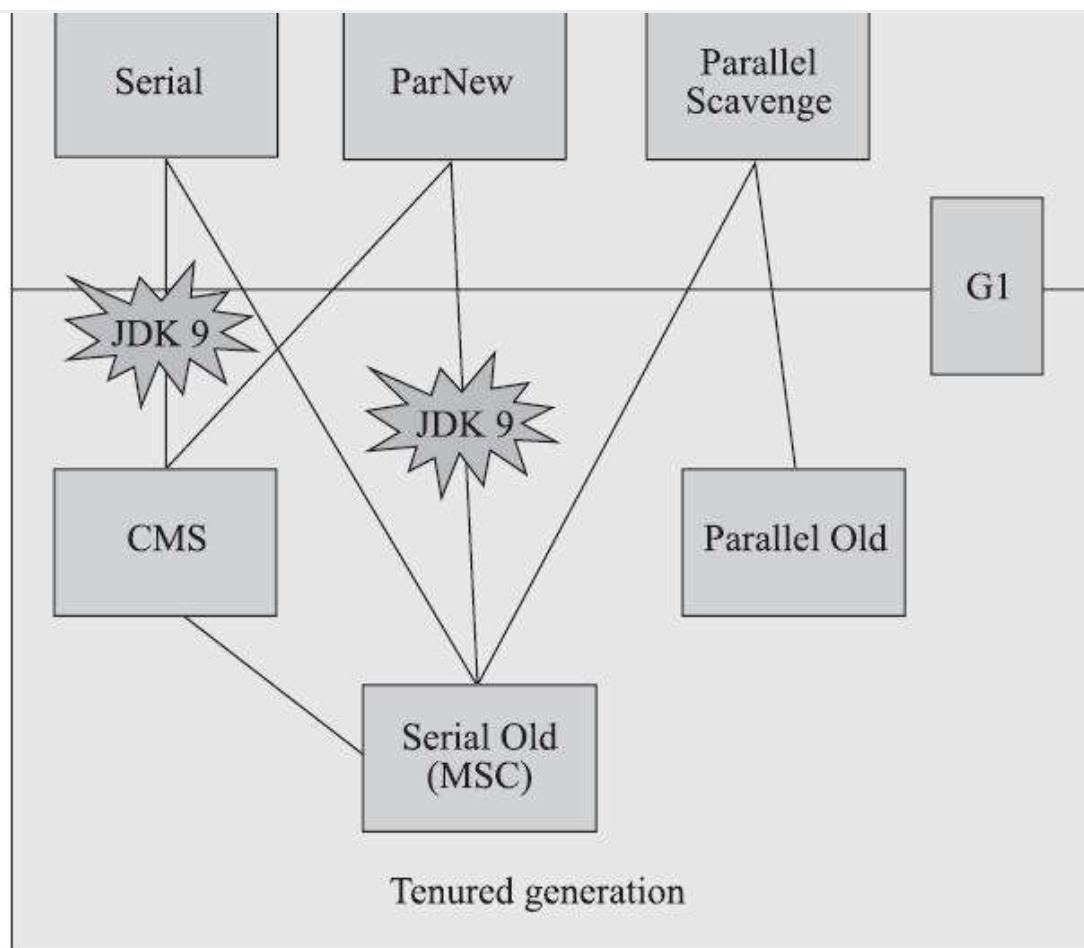


垃圾收集器

"经典"：指在JDK 7 Update 4之后（在这个版本中正式提供了商用的G1收集器，此前G1仍处于实验状态）、JDK 11正式发布之前，OracleJDK中的HotSpot虚拟机所包含的全部可用的垃圾收集器。

注意：并行和并发都是并发编程中的专业名词，在谈论垃圾收集器的上下文语境中，它们可以理解为

- 并行（Parallel）：并行描述的是多条垃圾收集器线程之间的关系，说明同一时间有多条这样的线程在协同工作，通常默认此时用户线程是处于等待状态。
- 并发（Concurrent）：并发描述的是垃圾收集器线程与用户线程之间的关系，说明同一时间垃圾收集器线程与用户线程都在运行。由于用户线程并未被冻结，所以程序仍然能响应服务请求，但由于垃圾收集器线程占用了部分系统资源，此时应用程序的处理的吞吐量将受到一定影响。



← Young GC → ← Old GC →

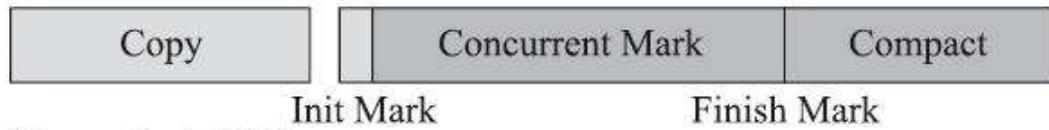
Serial, Parallel:



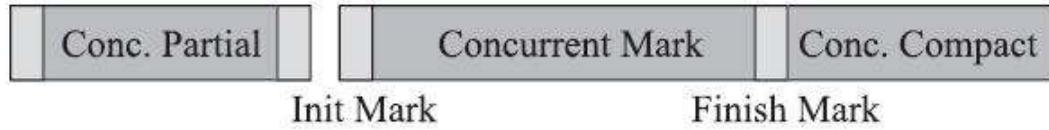
CMS:



G1:



Shenandoah, ZGC:





Serial	新生代	标记-复制	单线程(STW)	1. JDK1.3.1之前 代唯一选择 2. HotSpot客户端 的新生代收集器 3. 简单高效(与其 程相比),它是收 消耗(Memory Footprint)
ParNew	新生代	标记-复制	多线程并行(STW)	1. Serial多线程并 余行为包括控制方 法、STW、对象策 略等都与Serial 2. JDK7之前遗留 的新生代收集器 3. 除了Serial收集 只有它能与CMS
Parallel Scavenge	新生代	标记-复制	多线程并行(STW)	1. 目标是达到一 吐量(Throughput) 吐量优先收集器 2. -XX:+MaxGCPause 控制最大垃圾收 大于0毫秒 3. -XX : GCTime 设置吞吐量大小 100的整数 4. - XX:+UseAdaptive 虚拟机会根据当 情况收集性能监 调整这些参数以 停顿时间或者最 这种调节方式称 自适应的调节策 Ergonomics)



Serial Old (MSC) (MarkSweepCompact)	老年代	标记-整理	单线程 (STW)	1. 是Serial收集器 2. 主要意义是提升HotSpot虚拟机 3. JDK5之前与Parallel Scavenge搭配使用 4. 作为CMS收集的后备预案
Parallel Old	老年代	标记-整理	多线程并发 (STW)	1. JDK6才开始提供Scavenge收集器 2. 吞吐量优先(Parallel Scavenge+Parallel Old)
CMS (Concurrent Mark Sweep)	老年代	标记-清除	并发标记、并发清除	1. 四个步骤，1)并发标记(增量更新) 2)并发清除 3)并发低停顿收敛 4)并发清除 HotSpot虚拟机第一次成功尝试 3. 只有CMS会有这种行为 4. 三个明显缺点 1)常敏感，应为占用CPU而导致应用程序吞吐量 2)无法处理浮动垃圾(Floating Garbage) 3)并发清除"算法必然会带来停顿"
G1 (Garbage First)	Region	标记-整理、标记-复制	并发标记	1. 垃圾收集器技术里程碑式的成功 2. 基于Region的垃圾回收 3. JDK 7 Update引入
Shenandoah	Region		并发标记、并发整理	是一款只有Oracle公司含，而OracleJDK没有的收集器



ZGC	Region	并发整理	2018年Oracle创建 ZGC提交给OpenJDK 进入OpenJDK发
-----	--------	------	--

垃圾收集相关常用参数

参 数	描 述
UseSerialGC	虚拟机运行在 Client 模式下的默认值，打开此开关后，使用 Serial + Serial Old 的收集器组合进行内存回收
UseParNewGC	打开此开关后，使用 ParNew + Serial Old 的收集器组合进行内存回收，在 JDK 9 后不再支持
UseConcMarkSweepGC	打开此开关后，使用 ParNew + CMS + Serial Old 的收集器组合进行内存回收。Serial Old 收集器将作为 CMS 收集器出现“Concurrent Mode Failure”失败后的后备收集器使用
UseParallelGC	JDK 9 之前虚拟机运行在 Server 模式下的默认值，打开此开关后，使用 Parallel Scavenge + Serial Old (PS MarkSweep) 的收集器组合进行内存回收
UseParallelOldGC	打开此开关后，使用 Parallel Scavenge + Parallel Old 的收集器组合进行内存回收
SurvivorRatio	新生代中 Eden 区域与 Survivor 区域的容量比值，默认为 8，代表 Eden : Survivor=8 : 1
PretenureSizeThreshold	直接晋升到老年代的对象大小，设置这个参数后，大于这个参数的对象将直接在老年代分配
MaxTenuringThreshold	晋升到老年代的对象年龄。每个对象在坚持过一次 Minor GC 之后，年龄就增加 1，当超过这个参数值时就进入老年代
UseAdaptiveSizePolicy	动态调整 Java 堆中各个区域的大小以及进入老年代的年龄
HandlePromotionFailure	是否允许分配担保失败，即老年代的剩余空间不足以应付新生代的整个 Eden 和 Survivor 区的所有对象都存活的极端情况
ParallelGCThreads	设置并行 GC 时进行内存回收的线程数
GCTimeRatio	GC 时间占总时间的比率，默认值为 99，即允许 1% 的 GC 时间。仅在使用 Parallel Scavenge 收集器时生效
MaxGCPauseMillis	设置 GC 的最大停顿时间。仅在使用 Parallel Scavenge 收集器时生效
CMSInitiatingOccupancyFraction	设置 CMS 收集器在老年代空间被使用多少后触发垃圾收集。默认值为 68%，仅在使用 CMS 收集器时生效
UseCMSCompactAtFullCollection	设置 CMS 收集器在完成垃圾收集后是否要进行一次内存碎片整理。仅在使用 CMS 收集器时生效，此参数从 JDK 9 开始废弃
CMSFullGCsBeforeCompaction	设置 CMS 收集器在进行若干次垃圾收集后再启动一次内存碎片整理。仅在使用 CMS 收集器时生效，此参数从 JDK 9 开始废弃
UseG1GC	使用 G1 收集器，这个是 JDK 9 后的 Server 模式默认值
G1HeapRegionSize=n	设置 Region 大小，并非最终值
MaxGCPauseMillis	设置 G1 收集过程目标时间，默认值是 200ms，不是硬性条件
G1NewSizePercent	新生代最小值，默认值是 5%
G1MaxNewSizePercent	新生代最大值，默认值是 60%
ParallelGCThreads	用户线程冻结期间并行执行的收集器线程数



总字数: 3,404 字 上次更新: 2021-11-22 23:56:32

← On Java 8

Markdown基本语法 →



Markdown基本语法

你好！这是你第一次使用 **Markdown编辑器** 所展示的欢迎页。如果你想学习如何使用Markdown编辑器，可以仔细阅读这篇文章，了解一下Markdown的基本语法知识。

标题

输入

```
1 # 一级标题  
2 ## 二级标题  
3 ### 三级标题  
4 ##### 四级标题  
5 ##### 五级标题  
6 ##### 六级标题
```

[复制代码](#)

输出

输入1次 `#`，并按下 `space` 后，将生成1级标题。

输入2次 `##`，并按下 `space` 后，将生成2级标题。

以此类推，我们支持6级标题。有助于使用 `TOC` 语法后生成一个完美的目录。

文本样式

输入

```
1 `斜体：` *强调文本* _强调文本_  
2 `加粗：` **加粗文本** __加粗文本__  
3 `加粗+斜体：` ***加粗文本*** ___加粗文本___  
4 `删除线：` ~~删除文本~~  
5 > 引用文本  
6 >> 引用文本
```

[复制代码](#)



```
10 -----  
11 ***  
12 *****  
13 `下标：` H<sub>2</sub>0  
14 `上标：` 2<sup>10</sup>  
15 `键盘文本：` <kbd>Ctrl+Shift</kbd>
```

输出

斜体：**强调文本 强调文本**

加粗：**加粗文本 加粗文本**

加粗+斜体：**加粗文本 加粗文本**

删除线：~~删除文本~~

引用文本

引用文本

引用文本

分割线： 三个或者三个以上的 - 或者 * 都可以，例如：

下标： H_2O

上标： 2^{10}

键盘文本：**Ctrl+Shift**

链接与图片

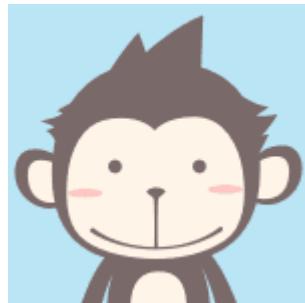
输入



```
3 | 图片: .



图片:



图片:

## 代码片

### 输入

```
1 | `` `java {3}
2 | public class HelloWorld {
3 | public static void main(String[] args) {
4 | System.out.println("Hello World");
5 | }
6 | }
7 | `` `
```

复制代码

### 输出



```
3 System.out.println("Hello World");
4 }
5 }
```

## 列表

### 无序列表

只需要在文字前面加上-、\*、+就可以了，它们效果是一样的，例如：

#### 输入

```
1 - 项目1
2 - 项目11
3 * 项目2
4 + 项目21
5 * 项目3
6 * 项目4
```

复制代码

#### 输出

- 项目1
  - 项目11
- 项目2
  - 项目21
- 项目3
- 项目4

### 有序列表

只需在数字后面加上英文句点即可，这里面的数字不影响排序，例如：



## 输入

```
1 1. 项目1
2 2. 项目2
3 3. 项目3
4
5 - [] 计划任务
6 - [x] 完成任务
```

复制代码

## 输出

1. 项目1
2. 项目2
3. 项目3

- 计划任务
- 完成任务

## 表格

一个简单的表格是这么创建的：

| 项目 | Value  |
|----|--------|
| 电脑 | \$1600 |
| 手机 | \$12   |
| 导管 | \$1    |

## 设定内容居中、居左、居右

使用 :-----: 居中 使用 :-----: 居左 使用 -----: 居右

| 第一列     | 第二列     | 第三列     |
|---------|---------|---------|
| 第一列文本居中 | 第二列文本居右 | 第三列文本居左 |



# Markdown 拓展(Vue)

## 链接

Home

### 说明

[跳转](/)

跳转到根的 index.html

SUMMARY

### 说明

[跳转](/summary)

跳转到 /summary 文件夹的 index.html

appendix markdown

### 说明

[跳转](/appendix/markdown)

跳转到 /zh/appendix 文件夹的 markdown.html

appendix markdown

### 说明

[跳转](/zh/appendix/markdown.md)

跳转到 /zh/appendix 文件夹的 markdown.html 具体文件可以使用 .md  
结尾 ( 推荐 )

appendix markdown



跳转到 /zh/appendix 文件夹的 markdown.html

## 表格

### 输入

```
1 | Tables | Are | Cool |
2 | -----:|:-----:|-----|
3 | col 3 is | right-aligned | $1600 |
4 | col 2 is | centered | $12 |
5 | zebra stripes | are neat | $1 |
```

复制代码

### 输出

| Tables        | Are           | Cool   |
|---------------|---------------|--------|
| col 3 is      | right-aligned | \$1600 |
| col 2 is      | centered      | \$12   |
| zebra stripes | are neat      | \$1    |

## Emoji

□□

你可以在这个列表 [找到所有可用的 Emoji。](#)

## 目录

### 输入

```
1 | [[toc]]
```

复制代码



## 输出

- 标题
- 文本样式
- 链接与图片
- 代码片
- 列表
  - 无序列表
  - 有序列表
- 表格
- Markdown 拓展(Vue)
  - 链接
  - 表格
  - Emoji
  - 目录
  - 自定义容器
  - Badge beta 0.10.1+ 默认主题
  - 代码
  - 行号

## 自定义容器

### 输入

```
1 :::: tip
2 This is a tip
3 ::::
4
5 :::: warning
6 This is a warning
7 ::::
8
9 :::: danger
10 This is a dangerous warning
11 ::::
12
```

[复制代码](#)



16 | :::

## 输出

### 提示

This is a tip

### 注意

This is a warning

### 警告

This is a dangerous warning

你也可以自定义块中的标题：

### STOP

Danger zone, do not proceed

## Badge beta 0.10.1+ 默认主题

- Props:
  - text - string
  - type - string, 可选值： "tip"|"warn"|"error" , 默认值是： "tip"
  - vertical - string, 可选值： "top"|"middle" , 默认值是： "top"

## 输入



```
3 | 三千世界<Badge text="世" type="error"/>
4 | 三千世界<Badge text="界" type="tip" vertical="middle"/>
```

## 输出

```
三千世界 三
三千世界 千
三千世界 世
三千世界 界
```

## 代码

VuePress 使用了 Prism 来为 markdown 中的代码块实现语法高亮。Prism 支持大量的编程语言，你需要做的只是在代码块的开始倒勾中附加一个有效的语言别名：

在 Prism 的网站上查看合法的语言列表[。](#)

## 输入

```
1 | ``java {3}
2 | public class HelloWorld {
3 | public static void main(String[] args) {
4 | System.out.println("Hello World");
5 | }
6 | }
7 | ``
```

复制代码

## 输出

```
1 | public class HelloWorld {
2 | public static void main(String[] args) {
3 | System.out.println("Hello World");
4 | }
5 | }
```

java 复制代码



## 行号

你可以通过配置来为每个代码块显示行号：

```
1 module.exports = {
2 markdown: {
3 lineNumbers: true
4 }
5 }
```

js 复制代码

[在 GitHub 上编辑此页](#)

总字数: 1,471 字 上次更新: 2021-11-22 23:56:32

← 深入理解Java虚拟机（第3版）

Halo →



# Halo

<https://halo.run> ↗

## install openjdk11

```
1 # 安装
2 cd /opt/package/java
3 wget https://download.java.net/java/GA/jdk11/9/GPL/openjdk-
4 tar -zxvf openjdk-11.0.2_linux-x64_bin.tar.gz
5 # 检查
6 cd jdk-11.0.2/bin
7 java -version
```

复制代码

## 安装Mysql

[CentOS8 安装 MySQL8.0 \( RPM \)](#) ↗

```
1 wget https://repo.mysql.com//mysql80-community-release-el8-
2 # 创建数据库
3 create database halo character set utf8mb4 collate utf8mb4_|
```

## 下载halo

```
1 # 下载halo
2 wget https://dl.halo.run/release/halo-1.4.12.jar
3 # 下载配置文件
4 mkdir -p /opt/source/backend/halo
5 cd /opt/source/backend/halo
6 wget https://dl.halo.run/config/application-template-mysql.
7 # wget https://dl.halo.run/config/application-template-h2.y
8 mv application-template-mysql.yaml application.yaml
9
```

复制代码



## 启动halo

```
1 /opt/package/java/jdk-11.0.2/bin/java -jar halo-1.4.12.jarsh 复制代码
```

## 作为服务运行

```
1 wget https://dl.halo.run/config/halo.service -O /etc/systemd/system/halo.servicesh 复制代码
2 vi /etc/systemd/system/halo.service
3
4 systemctl daemon-reload
5 systemctl enable halo
6 systemctl start halo
7 # 查看服务日志检查启动状态
8 journalctl -n 20 -u halo
```

```
1 [Unit]
2 Description=Halo Service
3 Documentation=https://docs.halo.run
4 After=network-online.target
5 Wants=network-online.target
6
7 [Service]
8 Type=simple
9 ExecStart=/bin/sh -c 'cd /opt/source/backend/halo && /opt/p...
10 ExecStop=/bin/kill -s QUIT $MAINPID
11 Restart=always
12 StandOutput=syslog
13
14 StandError=inherit
15
16 [Install]
17 WantedBy=multi-user.targetsh 复制代码
```



---

← [Markdown基本语法](#)

[Git](#) →



# Git

## 参考

- Reference [↗](#)
- 一分钟玩转 Git [↗](#)

## config

```
1 git --version
2
3
4 git config --global user.name "zhouxiao"
5 git config --global user.email "764800230@qq.com"
6
7 git config --global core.autocrlf false
8 git config --global core.safecrlf true
9
10 # 查看所有
11 git config --list
12 # 查看user.name
13 git config user.name
```

[复制代码](#)

## AutoCRLF与SafeCRLF换行符问题

```
1 一、AutoCRLF
2 # 提交时转换为LF，检出时转换为CRLF
3 git config --global core.autocrlf true
4
5 # 提交时转换为LF，检出时不转换
6 git config --global core.autocrlf input
7
8 # 提交检出均不转换
```

[复制代码](#)



```
--
12
13 # 拒绝提交包含混合换行符的文件
14 git config --global core.safecrlf true
15
16 # 允许提交包含混合换行符的文件
17 git config --global core.safecrlf false
18
19 # 提交包含混合换行符的文件时给出警告
20 git config --global core.safecrlf warn
```

## 使用

### Git 创建仓库

```
1 git init
2
3 git clone <repo>
```

复制代码

### Git 基本操作



实际写代码的地方，临时存放的地方 存放历史版本信息  
比如 IDE

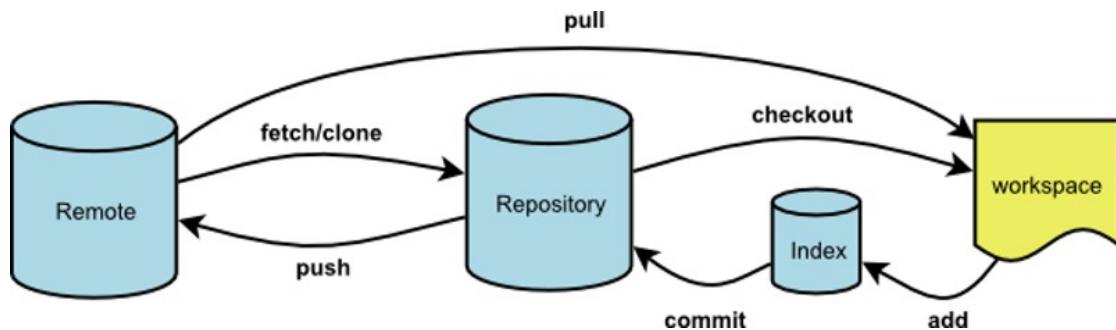
```
1 git add 文件名/文件夹/多个也可
2 git add .
3 git commit -m "comment"
4 git commit --amend
5
6 # 发现有不该提交的文件已经提交后，仅仅在.gitignore中加入忽略是不行的
7 git rm -r --cached 文件/文件夹名字
8
```

复制代码



12

git reset HEAD &lt;filename&gt;



```

1 git pull origin master
2 git merge
3
4 git push origin master
5

```

复制代码

## Git 分支管理

```

1 # 查看当前所有分支
2 git branch
3 # 创建分支
4 git branch <分支名字>
5 # 切换到分支
6 git checkout <分支名字>
7 # 删除分支, 有可能会删除失败, 因为Git会保护没有被合并的分支
8 git branch -d <分支名字>
9 # 强行删除, 丢弃没被合并的分支
10 git branch -D <分支名字>

```

复制代码

[在 GitHub 上编辑此页](#)

总字数: 415 字 上次更新: 2021-11-22 23:56:32





# 乱七八糟

C:\Windows\System32

```
1 %windir%\System32
2 %SystemRoot%\system32
```

sh 复制代码

## tcping

<https://www.elifulkerson.com/projects/tcping.php>

```
1 # help
2 tcping /?
3
4 tcping baidu.com 80
```

sh 复制代码

## nc

<https://eternallybored.org/misc/netcat>

```
1 # help
2 nc -h
3 # server
4 nc -l -p 9090
5 # client
6 nc localhost 9090
```

sh 复制代码

## ngrok

<https://ngrok.com/download>



```
3
4 ngrok http 8080
```

## PuTTY

<https://www.chiark.greenend.org.uk/>

### 快捷登录

```
1 -load "192.168.9.27" -ssh -l root -pw root
2
3 192.168.9.27 -l root -pw root
```

sh 复制代码

### 文件传输

```
1 # 1.运行psftp.exe并连接远程服务器
2 open 192.168.9.27
3
4 # 2.进入linux中对应存放文件的文件夹位置 ,
5 cd /usr/local/src
6
7 # 3.进入本地相应的文件夹位置
8 lcd e:/111
9
10 # 4.上传recerver.go到服务器上
11 put recerver.go
12
13 # 5.下载recerver.go到自己的电脑上
14 get recerver.go
```

sh 复制代码

## TreeSize

[https://www.jam-software.com/treesize\\_free/](https://www.jam-software.com/treesize_free/)



← Git

专有名词解释 →



# 专有名词解释

| 缩写  | 中文     | 英文                                 |
|-----|--------|------------------------------------|
| IDE | 集成开发环境 | Integrated Development Environment |

[在 GitHub 上编辑此页](#) ↗

总字数: 32 字 上次更新: 2021-11-22 23:56:32

---

← [乱七八糟](#)