主讲老师：Fox

有道笔记地址：https://note.youdao.com/s/NCe9oaem

前期准备工作

初始脚本来源

> 第六期项目实战环境
>
> https://docs.qq.com/doc/DUUZBZFJQdGRuV1Rr

借助的AI生成工具

> 问小白的deepseek：https://www.wenxiaobai.com/

Docker环境

当前 Docker 环境是 **24.0.7**，支持 Docker Compose 的 version: '3.8' 配置文件

```
[root@192-168-65-162 redis-env]# docker version
Client: Docker Engine - Community
 Version:           24.0.7
 API version:       1.43
 Go version:        go1.20.10
 Git commit:        afdd53b
 Built:             Thu Oct 26 09:11:35 2023
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          24.0.2
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.20.4
  Git commit:       659604f
  Built:            Thu May 25 21:54:24 2023
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.21
  GitCommit:        3dce8eb055cbb6872793272b4f20ed16117344f8
 runc:
  Version:          1.1.7
  GitCommit:        v1.1.7-0-g860f061
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Nacos环境搭建

问题

Nacos 默认使用 Derby 作为嵌入式数据库，所有数据（包括配置信息）会存储在容器的文件系统中。如果未将 Derby 数据目录挂载到宿主机，重启容器时，容器内的文件系统会被重置，导致数据丢失

deepseek生成的结果

nacos用默认的derby，重启docker容器会导致配置中心的配置丢失吗

为您检索到100篇内容

使用 Nacos 默认的 Derby 数据库时，**重启 Docker 容器可能会导致配置中心的配置丢失**，原因如下：

1. Derby 数据库的持久化机制

Nacos 默认使用 **Derby** 作为嵌入式数据库，所有数据（包括配置信息）会存储在容器的文件系统中。如果未将 Derby 数据目录挂载到宿主机，重启容器时，容器内的文件系统会被重置，导致数据丢失 [1] [39]。

2. 数据丢失的根本原因

- **未挂载数据目录**：如果未通过 `volumes` 将 Derby 数据目录挂载到宿主机，容器重启后，Derby 数据库文件会被清除，导致配置丢失。
- **容器文件系统的临时性**：Docker 容器的文件系统是临时的，容器重启后会重建文件系统，未持久化的数据将丢失 [39]。

解决方法

为了避免配置丢失，需要将 Derby 数据目录挂载到宿主机，确保数据持久化。以下是优化后的 `docker-compose.yml` 文件：

```yaml
version: '3.8'
services:
  nacos:
    image: nacos/nacos-server:v2.1.0
    container_name: nacos
    ports:
      - "8848:8848"
      - "9848:9848"
    environment:
      - MODE=standalone
    volumes:
      - ./nacos-data:/home/nacos/data  # 挂载数据目录
      - ./nacos-logs:/home/nacos/logs  # 挂载日志目录
    restart: always
```

关键配置说明

1. `volumes` **挂载**：
   - `./nacos-data:/home/nacos/data`：将 Derby 数据目录挂载到宿主机的 `./nacos-data`，确保数据持久化。
   - `./nacos-logs:/home/nacos/logs`：将日志目录挂载到宿主机，方便排查问题 [39]。
2. `restart: always`：确保容器意外退出时自动重启，提高服务的可靠性。

操作步骤

1. 创建挂载目录：

```bash
mkdir -p ./nacos-data ./nacos-logs
```

2. 启动 Nacos：

```bash
docker-compose up -d
```

3. 验证配置持久化：
   - 在 Nacos 控制台添加配置。

- 重启 Nacos 容器，检查配置是否仍然存在。

总结

- **未挂载数据目录时**，重启容器会导致配置丢失 [1] [39]。
- **挂载数据目录后**，配置会持久化到宿主机，重启容器不会丢失配置。

如果需要更高的可靠性，建议将 Derby 替换为 MySQL 或其他外部数据库 [63]。

解决方法

为了避免配置丢失，需要将 Derby 数据目录挂载到宿主机，确保数据持久化。以下是优化后的 docker-compose.yml 文件：

```yaml
version: '3.8'
services:
  nacos:
    image: nacos/nacos-server:v2.1.0
    container_name: nacos
    ports:
      - "8848:8848"
      - "9848:9848"
    environment:
      - MODE=standalone
    volumes:
      - ./nacos-data:/home/nacos/data  # 挂载数据目录
      - ./nacos-logs:/home/nacos/logs  # 挂载日志目录
    restart: always
```

关键配置说明

1. volumes **挂载**：
   - ./nacos-data:/home/nacos/data：将 Derby 数据目录挂载到宿主机的 ./nacos-data，确保数据持久化。
   - ./nacos-logs:/home/nacos/logs：将日志目录挂载到宿主机，方便排查问题
2. restart: always：确保容器意外退出时自动重启，提高服务的可靠性。

操作步骤

1. 创建挂载目录：

```
mkdir -p ./nacos-data ./nacos-logs
```

1. 启动 Nacos：

```
docker compose up -d
```

2. 验证配置持久化：
   - 在 Nacos 控制台添加配置。
   - 重启 Nacos 容器，检查配置是否仍然存在。

Mysql的部署

需求

```
spring:
  datasource:
    dynamic:
      primary: user
      # 未匹配到数据源时，true抛异常，false使用默认数据源
      strict: false
      datasource:
        user:
          url: jdbc:mysql://tlmall-mysql:3306/tl_mall_user?characterEncoding=utf-8&useSSL=false
          driver-class-name: com.mysql.cj.jdbc.Driver
          username: tlmall
          password: tlmall123
        goods:
          url: jdbc:mysql://tlmall-mysql:3306/tl_mall_goods?characterEncoding=utf-8&useSSL=false
          driver-class-name: com.mysql.cj.jdbc.Driver
          username: tlmall
          password: tlmall123
        promotion:
          url: jdbc:mysql://tlmall-mysql:3306/tl_mall_promotion?characterEncoding=utf-8&useSSL=false
          driver-class-name: com.mysql.cj.jdbc.Driver
          username: tlmall
          password: tlmall123
        normal:
```

以下是基于 Docker 24.0.7 的 MySQL 配置方案，使用 mysql_native_password 认证方式创建用户 tlmall 并设置密码 tlmall123，同时将 root 用户密码设置为 root 的完整步骤和 docker-compose.yml 文件

1. 目录结构

创建以下目录结构：

```
 mysql-env/
├── docker-compose.yml
├── mysql
│   ├── data
│   └── my.cnf
```

2. 配置文件

配置文件 (mysql/my.cnf):

```
[mysqld]
server-id = 1
bind-address = 0.0.0.0
default-authentication-plugin = mysql_native_password  # 使用旧版认证方式
character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci

[client]
default-character-set = utf8mb4
```

3. docker-compose.yml 文件

```yaml
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root  # 设置 root 用户密码为 root
    volumes:
      - ./mysql/my.cnf:/etc/mysql/conf.d/my.cnf  # 挂载配置文件
      - ./mysql/data:/var/lib/mysql  # 挂载数据目录
    ports:
      - "3306:3306"  # 暴露端口
    networks:
      - mysql-net

networks:
  mysql-net:
    driver: bridge
```

4. 操作步骤

1. 创建目录并编辑配置文件：

```bash
mkdir -p mysql/data
touch mysql/my.cnf
```

将配置文件内容写入 mysql/my.cnf 。

2. 启动容器：

```bash
cd mysql-env
docker compose up -d
```

```
[root@192-168-65-162 mysql-env]# docker compose up -d
[+] Running 2/2
 ✔ Network mysql-env_mysql-net  Created                                    0.4s
 ✔ Container mysql              Started                                    1.1s
```

3. 验证 MySQL 服务状态：

```
Bashdocker logs -f mysql
```

确认日志中显示 mysqld: ready for connections，表示服务已正常启动。

4. 使用 root 用户和密码 root 登录 MySQL：

```
docker exec -it mysql mysql -uroot -proot
```

```
[root@192-168-65-162 mysql-env]# docker exec -it mysql mysql -uroot -proot
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.41 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

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

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

执行以下 SQL：

```
-- 创建应用用户 tlmall
CREATE USER 'tlmall'@'%' IDENTIFIED WITH mysql_native_password BY 'tlmall123';
GRANT ALL PRIVILEGES ON *.* TO 'tlmall'@'%';
FLUSH PRIVILEGES;

-- 验证用户创建
SELECT user, host, plugin FROM mysql.user WHERE user = 'tlmall';
```

```
mysql> CREATE USER 'tlmall'@'%' IDENTIFIED WITH mysql_native_password BY 'tlmall123';
Query OK, 0 rows affected (0.07 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'tlmall'@'%';
Query OK, 0 rows affected (0.18 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT user, host, plugin FROM mysql.user WHERE user = 'tlmall';
+--------+------+-----------------------+
| user   | host | plugin                |
+--------+------+-----------------------+
| tlmall | %    | mysql_native_password |
+--------+------+-----------------------+
1 row in set (0.00 sec)
```

常见错误

在 MySQL 8.0.32 中，root 用户默认使用 caching_sha2_password 认证插件，这可能导致在连接时出现 Public Key Retrieval is not allowed 错误。为了避免使用 allowPublicKeyRetrieval=true，可以通过以下方法解决：

修改 root 用户的认证插件

将 root 用户的认证插件从 caching_sha2_password 改为 mysql_native_password，这样可以避免公钥检索的需求。

操作步骤：

1. 使用 root 用户登录 MySQL：

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

输入 root 密码。

2. 修改 root 用户的认证插件：

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '你的新密码';
FLUSH PRIVILEGES;
```

   - 'root'@'localhost'：表示 root 用户只能从本地连接。如果需要远程连接，可以将 localhost 改为 %。
   - '你的新密码'：设置 root 用户的新密码。

3. 验证修改是否生效：

```
SELECT user, host, plugin FROM mysql.user;
```

确保 root 用户的 plugin 列显示为 mysql_native_password。

Mysql主从架构的搭建建议通过docker方法分别配置主从，可以参考mysql专题主从复制的课程笔记

Redis搭建

目录结构

```
redis-single/
├── docker-compose.yml
├── redis.conf
└── redis-data
```

以下是一个基于 redis:5.0 的 单节点 Redis 的 docker-compose.yml 配置文件，支持数据持久化和自定义配置：

docker-compose.yml 文件

```yaml
version: '3.8'

services:
  redis:
    image: redis:5.0  # 使用 Redis 5.0 镜像
    container_name: redis-server
    restart: unless-stopped
    ports:
      - "6379:6379"  # 宿主机端口:容器端口
    volumes:
      - ./redis-data:/data  # 持久化数据目录
      - ./redis.conf:/usr/local/etc/redis/redis.conf  # 挂载自定义配置文件
    command: redis-server /usr/local/etc/redis/redis.conf --requirepass yourpassword
# 设置密码（可选）
```

关键配置说明

1. 镜像版本：使用 redis:5.0，确保版本与需求一致。
2. 数据持久化：
   - volumes: ./redis-data:/data：将容器内的 /data 目录挂载到宿主机的 ./redis-data 目录，避免数据丢失。
3. 自定义配置：
   - 挂载 redis.conf 文件到容器内，并通过 command 指定配置文件。
   - 若无需自定义配置，可直接通过 command 参数启用持久化（--appendonly yes）。
4. 密码保护：通过 --requirepass yourpassword 设置 Redis 访问密码（替换 yourpassword）。
5. 端口映射：6379:6379 允许宿主机和外部应用访问 Redis。

自定义配置示例（redis.conf）

```
# 绑定所有网络接口
bind 0.0.0.0

# 关闭保护模式（允许外部访问）
protected-mode no

# 启用持久化
appendonly yes

# 设置密码
requirepass yourpassword

# 其他配置...
# 完整配置参考：https://redis.io/topics/config
```

使用方法

1. 创建目录和配置文件

```
mkdir -p ./redis-data  # 数据持久化目录
touch ./redis.conf     # 自定义配置文件（内容参考上方示例）
```

2. 启动 Redis

```
docker compose up -d
```

3. 停止 Redis

```
docker compose down
```

4. 连接 Redis CLI

```
docker exec -it redis-server redis-cli -a yourpassword
```

Redis三主三从集群搭建

问题

redis集群需要密码，否则tulingmall-member启动会报错

```
... 21 common frames omitted
Caused by: org.redisson.client.RedisException Create breakpoint Lingma : ERR invalid password. channel: [id:
 0x56d65394, L:/192.168.65.103:58923 - R:/192.168.65.162:7005] command: (AUTH), params: []
    at org.redisson.client.handler.CommandDecoder.decode(CommandDecoder.java:300) ~[redisson-3.6.5.jar:na]
    at org.redisson.client.handler.CommandDecoder.decode(CommandDecoder.java:110) ~[redisson-3.6.5.jar:na]
    <19 internal lines>
```

nacos配置中心中Redis集群的配置

```yaml
spring:
  redis:
    cluster:
      nodes: 192.168.65.162:7000,192.168.65.162:7001,192.168.65.162:7002,192.168.65.162:7003,192.168.
192.168.65.162:7005
    lettuce:
      pool:
        max-active: 8 # 连接池最大连接数（使用负值表示没有限制）
        max-wait: -1ms # 连接池最大阻塞等待时间（使用负值表示没有限制）
        max-idle: 8 # 连接池中的最大空闲连接
        min-idle: 0 # 连接池中的最小空闲连接
      timeout: 3000ms # 连接超时时间（毫秒）
    password: tlmall
```

借助AI生成三主三从的redis集群，集群密码是tlmall

deepseek生成的结果

根据你提供的 Docker 版本信息，你的 Docker 环境是 **24.0.7**，支持 Docker Compose 的 `version: '3.8'` 配置文件。以下是针对 6 个 Redis 节点（7000 到 7005）的完整 Docker Compose 脚本，适配你的 Docker 环境，并确保 Redis 集群能够正确启动和初始化。

完整 Docker Compose 脚本

Yaml

```yaml
version: '3.8'

services:
  redis-node1:
    image: redis:5.0
    container_name: redis-node1
    command: redis-server --port 7000 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7000:7000"
      - "17000:17000"
    volumes:
      - ./data/node1:/data
    networks:
      - redis-cluster

  redis-node2:
    image: redis:5.0
    container_name: redis-node2
    command: redis-server --port 7001 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7001:7001"
      - "17001:17001"
    volumes:
      - ./data/node2:/data
    networks:
      - redis-cluster

  redis-node3:
    image: redis:5.0
    container_name: redis-node3
    command: redis-server --port 7002 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7002:7002"
      - "17002:17002"
    volumes:
      - ./data/node3:/data
    networks:
      - redis-cluster

  redis-node4:
    image: redis:5.0
    container_name: redis-node4
    command: redis-server --port 7003 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7003:7003"
      - "17003:17003"
    volumes:
      - ./data/node4:/data
    networks:
      - redis-cluster

  redis-node5:
    image: redis:5.0
    container_name: redis-node5
    command: redis-server --port 7004 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7004:7004"
      - "17004:17004"
    volumes:
      - ./data/node5:/data
    networks:
```

```
      - redis-cluster

  redis-node6:
    image: redis:5.0
    container_name: redis-node6
    command: redis-server --port 7005 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --
requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7005:7005"
      - "17005:17005"
    volumes:
      - ./data/node6:/data
    networks:
      - redis-cluster

  redis-init:
    image: redis:5.0
    container_name: redis-init
    command: >
      sh -c "sleep 15 && redis-cli -a tlmall --cluster create 192.168.65.162:7000 192.168.65.162:7001 192.168.65.162:7002
192.168.65.162:7003 192.168.65.162:7004 192.168.65.162:7005 --cluster-replicas 1 --cluster-yes"
    depends_on:
      - redis-node1
      - redis-node2
      - redis-node3
      - redis-node4
      - redis-node5
      - redis-node6
    networks:
      - redis-cluster

networks:
  redis-cluster:
    driver: bridge
```

关键配置说明

1. **节点配置**：
   - 每个节点（`redis-node1` 到 `redis-node6`）使用独立的端口（7000 到 7005）和数据目录。
   - 启用集群模式（`--cluster-enabled yes`）并设置密码认证（`--requirepass tlmall` 和 `--masterauth tlmall`）。
   - 使用 `--cluster-announce-ip` 指定节点的公网 IP，确保节点间通信正常。
2. **集群初始化**：
   - `redis-init` 服务负责在节点启动后自动创建集群。
   - 使用 `redis-cli --cluster create` 命令初始化集群，并设置 `--cluster-replicas 1` 为每个主节点分配一个从节点。
3. **网络配置**：
   - 所有节点加入 `redis-cluster` 网络，确保节点间可以互相通信。
4. **适配 Docker 24.0.7**：
   - 使用 `version: '3.8'`，完全兼容你的 Docker 版本。

使用方法

1. 将上述脚本保存为 `docker-compose.yml`。
2. 在终端中运行以下命令启动集群：

```
Bash
docker-compose up -d
```

3. 检查集群状态：

```
Bash
docker exec -it redis-node1 redis-cli -a tlmall --cluster check 192.168.65.162:7000
```

验证集群状态

1. 查看集群节点信息：

Bash

```
docker exec -it redis-node1 redis-cli -a tlmall cluster nodes
```

2. 检查集群健康状态：

Bash

```
docker exec -it redis-node1 redis-cli -a tlmall cluster info
```

总结

- 该脚本适配 Docker 24.0.7，确保 Redis 集群能够正确启动和初始化。
- 通过 `redis-init` 服务自动创建集群，无需手动操作。
- 使用密码认证和哈希槽分配，确保集群的安全性和高可用性。

如果有任何问题，请随时告诉我！

以下是基于 6 个 Redis 节点（7000 到 7005）的完整 Docker Compose 脚本，支持集群模式、密码认证和哈希槽分配。每个节点都配置了独立的端口、数据目录和集群参数。

```
version: '3.8'

services:
  redis-node1:
    image: redis:5.0
    container_name: redis-node1
    command: redis-server --port 7000 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7000:7000"
      - "17000:17000"
    volumes:
      - ./data/node1:/data
    networks:
      - redis-cluster

  redis-node2:
    image: redis:5.0
    container_name: redis-node2
    command: redis-server --port 7001 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7001:7001"
      - "17001:17001"
    volumes:
      - ./data/node2:/data
    networks:
      - redis-cluster

  redis-node3:
    image: redis:5.0
    container_name: redis-node3
    command: redis-server --port 7002 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7002:7002"
      - "17002:17002"
    volumes:
```

```
      - ./data/node3:/data
    networks:
      - redis-cluster

  redis-node4:
    image: redis:5.0
    container_name: redis-node4
    command: redis-server --port 7003 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7003:7003"
      - "17003:17003"
    volumes:
      - ./data/node4:/data
    networks:
      - redis-cluster

  redis-node5:
    image: redis:5.0
    container_name: redis-node5
    command: redis-server --port 7004 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7004:7004"
      - "17004:17004"
    volumes:
      - ./data/node5:/data
    networks:
      - redis-cluster

  redis-node6:
    image: redis:5.0
    container_name: redis-node6
    command: redis-server --port 7005 --cluster-enabled yes --cluster-config-file /data/nodes.conf --appendonly yes --bind 0.0.0.0 --requirepass tlmall --masterauth tlmall --cluster-announce-ip 192.168.65.162
    ports:
      - "7005:7005"
      - "17005:17005"
    volumes:
```

```
      - ./data/node6:/data
    networks:
      - redis-cluster

  redis-init:
    image: redis:5.0
    container_name: redis-init
    command: >
      sh -c "sleep 10 && redis-cli -a tlmall --cluster create 192.168.65.162:7000
192.168.65.162:7001 192.168.65.162:7002 192.168.65.162:7003 192.168.65.162:7004
192.168.65.162:7005 --cluster-replicas 1 --cluster-yes"
    depends_on:
      - redis-node1
      - redis-node2
      - redis-node3
      - redis-node4
      - redis-node5
      - redis-node6
    networks:
      - redis-cluster

networks:
  redis-cluster:
    driver: bridge
```

关键配置说明

1. 节点配置
   - 每个节点（redis-node1 到 redis-node6）使用独立的端口（7000 到 7005）和数据目录。
   - 启用集群模式（--cluster-enabled yes）并设置密码认证（--requirepass tlmall 和 --masterauth tlmall）。
   - 使用 --cluster-announce-ip 指定节点的公网 IP，确保节点间通信正常。
2. 集群初始化
   - redis-init 服务负责在节点启动后自动创建集群。
   - 使用 redis-cli --cluster create 命令初始化集群，并设置 --cluster-replicas 1 为每个主节点分配一个从节点。
3. 网络配置
   - 所有节点加入 redis-cluster 网络，确保节点间可以互相通信。

使用方法

1. 将上述脚本保存为 docker-compose.yml。

2. 在终端中运行以下命令启动集群：

```
docker compose up -d
```

3. 检查集群状态：

```
docker exec -it redis-node1 redis-cli -a tlmall --cluster check 192.168.65.162:7000
```

注意事项

1. 密码一致性
   - 确保所有 Redis 节点的 --requirepass 和 --masterauth 参数的值一致。
   - 在创建集群时，--cluster-password 参数的值也需要与上述密码一致。
2. 安全性
   - 密码 tlmall 仅为示例，建议在生产环境中使用更强的密码。
3. 集群创建
   - redis-init 服务用于自动创建 Redis 集群。确保所有 Redis 节点启动后再执行此操作。
4. 网络配置
   - 如果 Redis 节点部署在云服务器上，确保安全组或防火墙放行相关端口（如 7000-7005 和 17000-17005）。

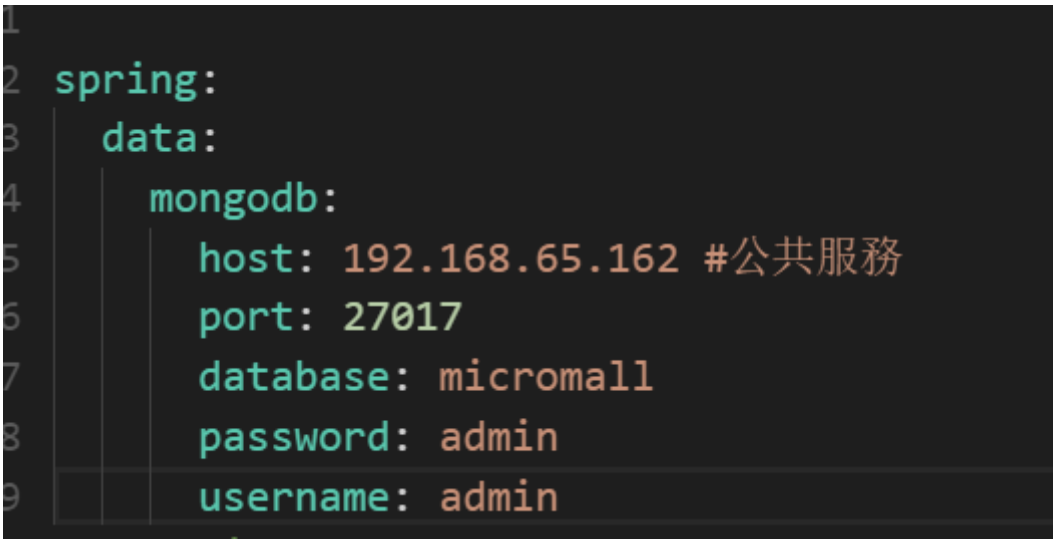
MongoDB环境搭建

问题

tulingmall-member服务启动报错，需要用到mongodb

```
com.mongodb.MongoSocketOpenException Create breakpoint Lingma → : Exception opening socket
    at com.mongodb.internal.connection.SocketStream.open(SocketStream.java:70) ~[mongodb-driver-core-4.0.6.jar:na]
    at com.mongodb.internal.connection.InternalStreamConnection.open(InternalStreamConnection.java:127) ~[mongodb-driver-core-4.0.6.jar:na]
    at com.mongodb.internal.connection.DefaultServerMonitor$ServerMonitorRunnable.run(DefaultServerMonitor.java:117) ~[mongodb-driver-core-4.0.6.jar:na] <1 internal line>
```

去nacos配置中心检查发现mongodb的配置



```yaml
spring:
  data:
    mongodb:
      host: 192.168.65.162 #公共服務
      port: 27017
      database: micromall
      password: admin
      username: admin
```

借助AI生成mongodb的docker-compose.yml脚本

deepseek生成的结果

```
version: '2.2'

services:
  mongodb:
    image: mongo:4.4
    container_name: mongodb
    ports:
      - "27017:27017"
#    volumes:
#      - ./mongodb-data:/data/db
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: admin
```

优化这个脚本，数据文件还有配置文件都需要挂载，同时要求这个配置spring:

```
  data:
    mongodb:
      host: 192.168.65.162 #公共服务
      port: 27017
      database: micromall
      password: admin
      username: admin
```

可以连上docker部署的mongodb， 生成完整的mongodb部署的docker compose文件，docker版本是24.0.7

为您检索到78篇内容

以下是优化后的 Docker Compose 文件，满足你的需求：

1. **挂载数据目录和配置文件**：将 MongoDB 的数据目录、日志目录和配置文件挂载到宿主机，确保数据持久化和配置可管理。
2. **支持 Spring Boot 连接**：通过环境变量和网络配置，确保 Spring Boot 应用能够连接到 Docker 部署的 MongoDB。

Yaml

```yaml
version: '3.8'  # 适配 Docker 24.0.7

services:
  mongodb:
    image: mongo:4.4  # 使用 MongoDB 4.4 版本
    container_name: mongodb
    ports:
      - "27017:27017"  # 映射宿主机端口
    volumes:
      - ./mongodb-data:/data/db  # 挂载数据目录
      - ./mongodb-log:/var/log/mongodb  # 挂载日志目录
      - ./mongodb-config:/etc/mongo  # 挂载配置文件目录
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin  # 初始化管理员用户名
      MONGO_INITDB_ROOT_PASSWORD: admin  # 初始化管理员密码
    networks:
      - mongodb-network

networks:
  mongodb-network:
    driver: bridge
```

Spring Boot 配置

在 Spring Boot 的 `application.yml` 中，配置 MongoDB 连接信息：

Yaml

```yaml
spring:
  data:
    mongodb:
      host: 192.168.65.162  # MongoDB 服务地址
```

```
host: 192.168.65.162  # MongoDB 服务地址
port: 27017  # MongoDB 端口
database: micromall  # 数据库名称
username: admin  # 用户名
password: admin  # 密码
```

关键配置说明

1. **数据持久化**：
   - `./mongodb-data:/data/db`：将 MongoDB 的数据目录挂载到宿主机的 `./mongodb-data` 目录，确保数据不会因容器重启而丢失 1 。
   - `./mongodb-log:/var/log/mongodb`：将日志目录挂载到宿主机，便于日志管理 1 。
2. **配置文件挂载**：
   - `./mongodb-config:/etc/mongo`：将配置文件目录挂载到宿主机，方便修改 MongoDB 配置 1 。
3. **网络配置**：
   - 使用 `mongodb-network` 网络，确保容器间通信正常 52 。
4. **Spring Boot 连接**：
   - 通过 `host: 192.168.65.162` 和 `port: 27017`，Spring Boot 应用可以连接到 Docker 部署的 MongoDB 52 。

使用方法

1. 创建挂载目录：

```bash
mkdir -p ./mongodb-data ./mongodb-log ./mongodb-config
```

2. 启动 MongoDB 服务：

```bash
docker-compose up -d
```

3. 验证 MongoDB 是否正常运行：

```bash
docker exec -it mongodb mongo -u admin -p admin
```

4. 在 Spring Boot 应用中配置 `application.yml`，并启动应用，确保能够连接 MongoDB。

参考来源

- 挂载数据目录和配置文件 1
- Docker Compose 网络配置 52
- Spring Boot 连接 MongoDB 52

复制 重新生成 ···

以下是优化后的 Docker Compose 文件，满足你的需求：

1. 挂载数据目录和配置文件：将 MongoDB 的数据目录、日志目录和配置文件挂载到宿主机，确保数据持久化和配置可管理。
2. 支持 Spring Boot 连接：通过环境变量和网络配置，确保 Spring Boot 应用能够连接到 Docker 部署的 MongoDB。

```yaml
version: '3.8'  # 适配 Docker 24.0.7

services:
  mongodb:
    image: mongo:4.4  # 使用 MongoDB 4.4 版本
    container_name: mongodb
    ports:
      - "27017:27017"  # 映射宿主机端口
    volumes:
      - ./mongodb-data:/data/db  # 挂载数据目录
      - ./mongodb-log:/var/log/mongodb  # 挂载日志目录
      - ./mongodb-config:/etc/mongo  # 挂载配置文件目录
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin  # 初始化管理员用户名
      MONGO_INITDB_ROOT_PASSWORD: admin  # 初始化管理员密码
    networks:
      - mongodb-network

networks:
  mongodb-network:
    driver: bridge
```

关键配置说明

1. 数据持久化
   - ./mongodb-data:/data/db：将 MongoDB 的数据目录挂载到宿主机的 ./mongodb-data 目录，确保数据不会因容器重启而丢失
   - ./mongodb-log:/var/log/mongodb：将日志目录挂载到宿主机，便于日志管理
2. 配置文件挂载
   - ./mongodb-config:/etc/mongo：将配置文件目录挂载到宿主机，方便修改 MongoDB 配置
3. 网络配置
   - 使用 mongodb-network 网络，确保容器间通信正常
4. Spring Boot 连接
   - 通过 host: 192.168.65.162 和 port: 27017，Spring Boot 应用可以连接到 Docker 部署的 MongoDB

使用方法

1. 创建挂载目录：

```
mkdir -p ./mongodb-data ./mongodb-log ./mongodb-config
```

2. 启动 MongoDB 服务：

```
docker compose up -d
```

3. 验证 MongoDB 是否正常运行：

```
docker exec -it mongodb mongo -u admin -p admin
```

```
[root@192-168-65-162 mongodb-env]# docker exec -it mongodb mongo -u admin -p admin
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("fb9ed97d-2bf9-416d-9faf-abb1e1c8a4a4") }
MongoDB server version: 4.4.29
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2025-03-21T04:59:47.509+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'
---
```

4. 在 Spring Boot 应用中配置 application.yml，并启动应用，确保能够连接 MongoDB。

```
spring:
  data:
    mongodb:
      host: 192.168.65.162  # MongoDB 服务地址
      port: 27017  # MongoDB 端口
      database: micromall  # 数据库名称
      username: admin  # 用户名
      password: admin  # 密码
```

Zookeeper环境搭建

问题

tulingmall-product服务用到了zookeeper

```yaml
#zk分布式锁相关配置
zk:
  curator:
    retryCount: 5 #重试次数
    elapsedTimeMs: 5000 #
    connectUrl: tlshop.com:2181 #zk地址
    sessionTimeOutMs: 60000 #会话超时时间
    connectionTimeOutMs: 5000 #连接超时时间
```

以下是一个基于 Docker Compose 的 Zookeeper 单节点快速部署脚本，适配您提供的配置参数。该脚本使用
zookeeper 官方镜像，并支持数据持久化和端口映射。

1. 目录结构

```
zookeeper-env/
├── docker-compose.yml
├── data
└── logs
```

2. Docker Compose 文件 (docker-compose.yml)

```yaml
version: '3.8'

services:
  zookeeper:
    image: zookeeper:3.7.0  # 使用 Zookeeper 官方镜像
    container_name: zookeeper
    restart: always  # 容器自动重启
    ports:
      - "2181:2181"  # 暴露 Zookeeper 客户端端口
    volumes:
      - ./data:/data  # 持久化数据目录
      - ./logs:/logs  # 持久化日志目录
    environment:
      ZOO_MY_ID: 1  # Zookeeper 节点 ID
      ZOO_TICK_TIME: 2000  # Zookeeper 心跳时间（毫秒）
      ZOO_INIT_LIMIT: 5  # 初始化连接的最大时间（心跳倍数）
      ZOO_SYNC_LIMIT: 2  # 同步连接的最大时间（心跳倍数）
```

配置详解

- image: zookeeper:3.7.0：使用 Zookeeper 官方镜像，版本为 3.7.0。
- ports: "2181:2181"：将宿主机的 2181 端口映射到容器的 2181 端口，用于 Zookeeper 客户端连接。
- volumes：挂载 data 和 logs 目录，确保数据持久化。
- environment：配置 Zookeeper 环境变量，包括节点 ID、心跳时间等。

3. 操作步骤

1. 启动 Zookeeper 容器

在 zookeeper-env 目录下执行以下命令：

```bash
docker compose up -d
```

2. 远程连接zookeeper

```bash
zkCli -server 192.168.65.162:2181
```

Rocketmq的环境搭建

1.目录结构

```
rocketmq-env/
├── data
│   ├── broker
│   │   ├── conf
│   │   │   └── broker.conf
│   │   ├── logs
│   │   └── store
│   └── namesrv
│       └── logs
└── docker-compose.yml
```

2.docker-compose.yml 脚本

```yaml
version: '3.8'

services:
  namesrv:
    image: apache/rocketmq:4.8.0
    container_name: rocketmq-namesrv
    ports:
      - "9876:9876"
    volumes:
      - ./data/namesrv/logs:/home/rocketmq/logs  # 挂载日志目录
    command: sh mqnamesrv

  broker:
    image: apache/rocketmq:4.8.0
    container_name: rocketmq-broker
    environment:
      - NAMESRV_ADDR=namesrv:9876
    ports:
      - "10911:10911"
      - "10909:10909"
    volumes:
      - ./data/broker/logs:/home/rocketmq/logs  # 挂载日志目录
      - ./data/broker/store:/home/rocketmq/store  # 挂载存储目录
      - ./data/broker/conf/broker.conf:/home/rocketmq/conf/broker.conf  # 挂载配置文件
    depends_on:
      - namesrv
    command: sh mqbroker -c /home/rocketmq/conf/broker.conf

  console:
    image: styletang/rocketmq-console-ng  # RocketMQ 控制台
    container_name: rocketmq-console
    ports:
      - "8080:8080"
    environment:
      - JAVA_OPTS=-Drocketmq.namesrv.addr=namesrv:9876 -
Dcom.rocketmq.sendMessageWithVIPChannel=false
    depends_on:
      - namesrv
      - broker
```

关键配置说明

1. NameServer
   - 使用 apache/rocketmq:4.8.0 镜像。
   - 挂载日志目录到本地 ./data/namesrv/logs，确保日志持久化
2. Broker
   - 依赖于 NameServer，通过 NAMESRV_ADDR 环境变量指定 NameServer 地址。
   - 挂载日志目录（logs）、存储目录（store）和配置文件（broker.conf）到本地，确保数据和配置持久化
   - 使用自定义配置文件 /home/rocketmq/conf/broker.conf，可以通过挂载本地文件进行配置。
3. Console（控制台）
   - 使用 styletang/rocketmq-console-ng 镜像，提供 RocketMQ 的 Web 管理界面
   - 通过 JAVA_OPTS 指定 NameServer 地址。
   - 映射端口 8080，访问地址为 http://localhost:8080。

Broker 配置文件示例 (broker.conf)

在 ./data/broker/conf/broker.conf 中添加以下内容：

```
brokerClusterName = DefaultCluster
brokerName = broker-a
brokerId = 0
deleteWhen = 04
fileReservedTime = 48
brokerRole = ASYNC_MASTER
flushDiskType = ASYNC_FLUSH
namesrvAddr = namesrv:9876
autoCreateTopicEnable = true
```

3.启动 RocketMQ 集群

1. 创建挂载目录：

```
mkdir -p ./data/namesrv/logs ./data/broker/logs ./data/broker/store ./data/broker/conf
```

2. 启动服务：

```
docker compose up -d
```

3. 验证服务：
   - 访问 RocketMQ 控制台：http://localhost:8080
   - 检查日志：tail -f ./data/namesrv/logs/rocketmqlogs/namesrv.log

Elasticsearch的环境搭建

需求

tulingmall-search用到了ES，需要搭建ES的环境

```yaml
data:
  elasticsearch:
    rest:
      #uris: 192.168.65.110:9300,192.168.65.113:9300,192.168.65.219:9300
      uris: 192.168.65.219:9300
      username: elastic
      password: 123456
```

以下是根据您的要求优化后的 Docker Compose 脚本，适配当前 Docker 环境，并配置了 Elasticsearch 和 Kibana 的安全认证（用户名 elastic，密码 123456）。同时，挂载了数据目录和配置文件，确保数据持久化和配置可管理。

1. 目录结构

```
elasticsearch-env/
├── docker-compose.yml
├── elasticsearch
│   ├── config
│   │   └── elasticsearch.yml
│   └── data
│   └── plugins
└── kibana
    └── config
        └── kibana.yml
```

2. 配置文件

Elasticsearch 配置文件 (elasticsearch/config/elasticsearch.yml)

```
cluster.name: "docker-cluster"
network.host: 0.0.0.0
discovery.type: single-node
xpack.security.enabled: true  # 启用安全认证
xpack.security.transport.ssl.enabled: true
```

Kibana 配置文件 (kibana/config/kibana.yml)

```
server.host: "0.0.0.0"
elasticsearch.hosts: ["http://elasticsearch:9200"]
elasticsearch.username: "elastic"  # Kibana 连接 Elasticsearch 的用户名
elasticsearch.password: "123456"  # Kibana 连接 Elasticsearch 的密码
xpack.security.enabled: true  # 启用安全认证
i18n.locale: "zh-CN"
```

3. Docker Compose 文件 (docker-compose.yml)

```yaml
version: '3.8'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.6.1
    container_name: elasticsearch
    user: "1000:1000"  # 指定容器内的用户和组
    environment:
      - discovery.type=single-node
      - ELASTIC_PASSWORD=123456  # 设置 elastic 用户的密码
      - ES_JAVA_OPTS=-Xms512m -Xmx512m  # 设置 JVM 内存
    ports:
      - "9200:9200"
      - "9300:9300"
    volumes:
      - ./elasticsearch/data:/usr/share/elasticsearch/data  # 挂载数据目录
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml  # 挂载配置文件
      - ./elasticsearch/plugins:/usr/share/elasticsearch/plugins  # 挂载插件目录
    networks:
      - elastic-net

  kibana:
    image: docker.elastic.co/kibana/kibana:7.6.1
    container_name: kibana
    environment:
      - ELASTICSEARCH_URL=http://elasticsearch:9200
    ports:
      - "5601:5601"
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml  # 挂载配置文件
    depends_on:
      - elasticsearch
    networks:
      - elastic-net

volumes:
  elasticsearch-data:
    driver: local
```

```
39
40 networks:
41   elastic-net:
42     driver: bridge
```

配置说明

- 安全认证：通过 xpack.security.enabled 启用了 Elasticsearch 和 Kibana 的安全认证，并设置了用户名 elastic 和密码 123456。
- 数据持久化：挂载了 Elasticsearch 的数据目录和配置文件，确保数据持久化和配置可管理。
- JVM 内存配置：通过 ES_JAVA_OPTS 设置了 Elasticsearch 的 JVM 内存，避免内存不足导致的启动失败。
- 权限问题

Elasticsearch 容器默认以 elasticsearch 用户（UID 1000）运行，因此需要确保宿主机上的目录具有适当的权限。可以通过以下命令设置权限：

```
1 mkdir -p ./elasticsearch/data
2 chmod -R 777 ./elasticsearch/data  # 确保目录可写
```

或者将目录的所有者设置为 UID 1000：

```
1 sudo chown -R 1000:1000 ./elasticsearch/data
```

4. 操作步骤

1. 启动容器

在 elasticsearch-env 目录下执行以下命令：

```
1 docker compose up -d
```

2. 验证 Elasticsearch 是否正常运行

访问 http://localhost:9200，输入用户名 elastic 和密码 123456，确认返回 Elasticsearch 的欢迎信息。

3. 验证 Kibana 是否正常运行

访问 http://localhost:5601，输入用户名 elastic 和密码 123456，确认 Kibana 界面正常加载。

5. 安装 IK 分词器

1. 下载 IK 分词器

从https://release.infinilabs.com/analysis-ik/stable/下载与 Elasticsearch 7.6.1 版本匹配的 IK 分词器插件

2. 解压到插件目录

将下载的 ZIP 文件解压到 ./elasticsearch/plugins/ik 目录中：

```
unzip elasticsearch-analysis-ik-7.6.1.zip -d ./elasticsearch/plugins/ik
```

3. 重启 Elasticsearch 容器

重启容器以加载插件：

```
docker restart elasticsearch
```

4. 测试分词器

在 Kibana 中执行以下命令测试 IK 分词器：

```
POST _analyze
{
  "analyzer": "ik_smart",
  "text": "中华人民共和国"
}
```

如果返回分词结果，则说明分词器已正常工作。