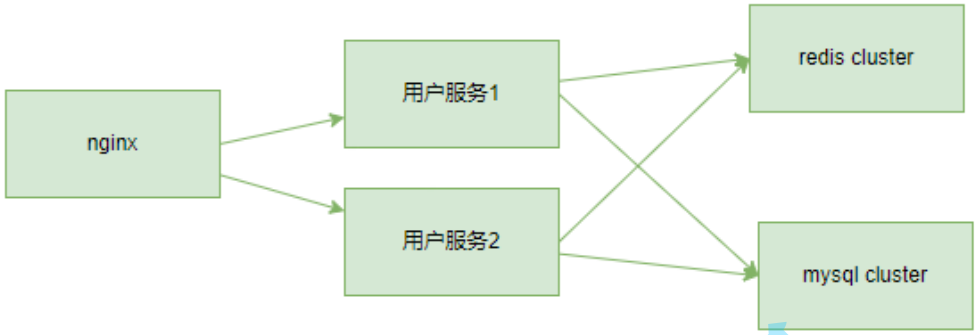
Dockerfile 配合 docker-compose 搭建微服务

版权说明

对比特项目感兴趣，可以联系这个微信。

实战目的

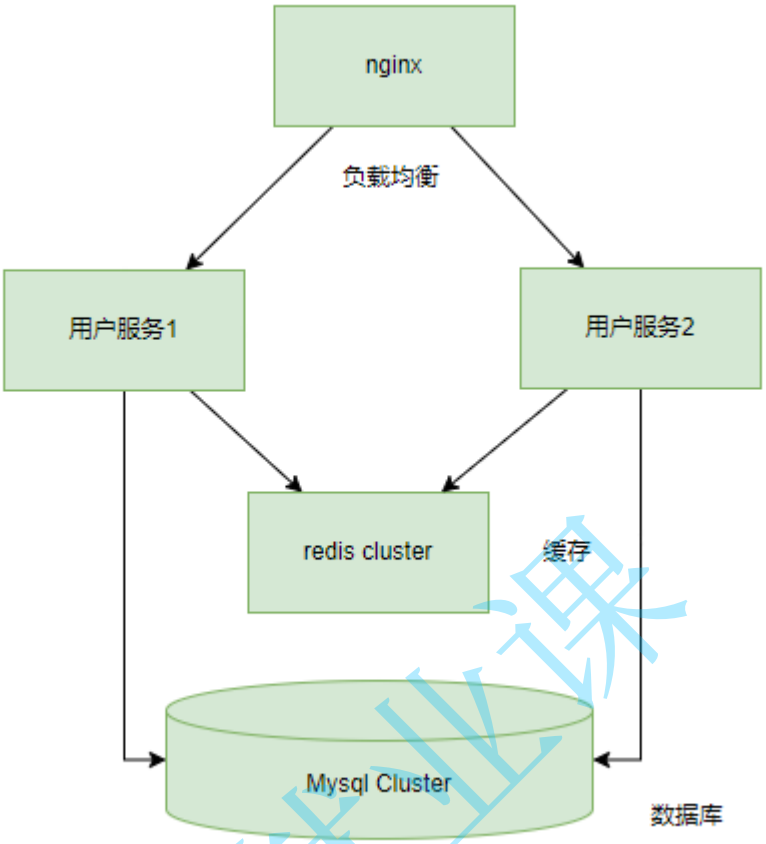
通过 dockerfile 和 docker-compose 完成微服务的搭建，核心功能是实现用户管理微服务，整体拓扑如下：



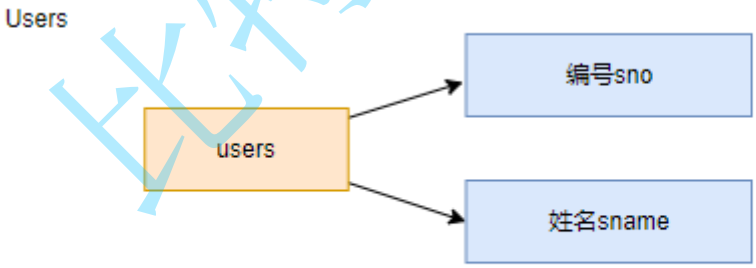
实战步骤

1. 方案设计
    a. 整体架构

    系统整体架构



b. 库表设计



对应脚本

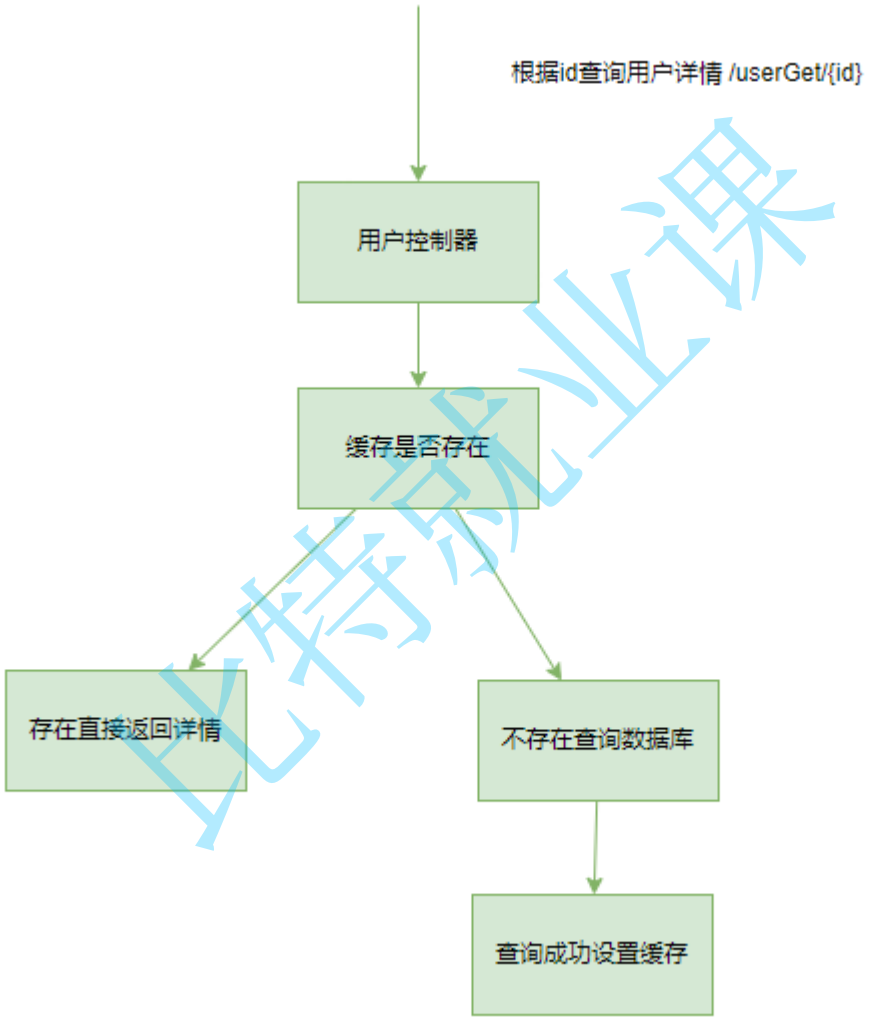
```
Shell
drop database if exists test;
CREATE DATABASE `test` DEFAULT CHARACTER SET utf8mb4 ;
use `test`;

CREATE TABLE `users` (
```

```
                          `sno` int(11)  PRIMARY KEY,
                          `sname` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO users (sno,sname) VALUES
                                  (1,'pony'),
                                  (2,'maxhou');
```

c. 接口设计，用户管理一般有用户的增加，用户编辑，用户查询，用户删除等功能，我们实现其中一个用户查询功能。用户查询功能的逻辑如下：



2. 准备目录

```Shell
mkdir -p /data/maxhou/mymicroservice
cd /data/maxhou/mymicroservice
```

```
#准备 nginx 目录
mkdir -p /data/maxhou/mymicroservice/nginx
#准备 java 应用目录
mkdir -p /data/maxhou/mymicroservice/app/user
#准备 mysql 目录
mkdir -p /data/maxhou/mymicroservice/mysql
#准备 redis 目录
mkdir -p /data/maxhou/mymicroservice/redis
```

3. 搭建 Mysql 集群和 redis 集群，mysql 和 redis 的相关配置我们在之前已经讲解过了，将对应内容拷贝过去

```
Shell
cp -r /data/maxhou/mysqlcluster/master
/data/maxhou/mymicroservice/mysql
cp -r /data/maxhou/mysqlcluster/slave
/data/maxhou/mymicroservice/mysql
cp -r /data/maxhou/mysqlcluster/docker-compose.yml
/data/maxhou/mymicroservice/mysql
cp -r /data/maxhou/rediscluster/redis/*
/data/maxhou/mymicroservice/redis
```

4. 编写进入目录/data/maxhou/mymicroservice, vi docker-compose.yml，考虑服务器资源，我们将 mysql 集群配置为 1 主 1 从,降低内存占用

```
Shell
version: "3"
services:
  mysql-master:
    build:
      context: ./mysql
      dockerfile: ./master/Dockerfile-master
    image: mysqlmaster:v2.0
    restart: always
    container_name: mysql-master
    volumes:
      - ./mastervarlib:/var/lib/mysql
    ports:
      - 9306:3306
    environment:
      MYSQL_ROOT_PASSWORD: root
    privileged: true
    command: ['--server-id=1',
              '--log-bin=master-bin',
```

```
            '--binlog-ignore-db=mysql',
            '--binlog_cache_size=256M',
            '--binlog_format=mixed',
            '--lower_case_table_names=1',
            '--character-set-server=utf8',
            '--collation-server=utf8_general_ci']
mysql-slave:
  build:
    context: ./mysql
    dockerfile: ./slave/Dockerfile-slave
  image: mysqlslave:v2.0
  restart: always
  container_name: mysql-slave
  volumes:
    - ./slavevarlib:/var/lib/mysql
  ports:
    - 9307:3306
  environment:
    MYSQL_ROOT_PASSWORD: root
  privileged: true
  command: ['--server-id=2',
            '--relay_log=slave-relay',
            '--lower_case_table_names=1',
            '--character-set-server=utf8',
            '--collation-server=utf8_general_ci']
  depends_on:
    - mysql-master
redis01:
    image: myredis:v1.0
    build: ./redis
    ports:
      - 6379:6379
    container_name: redis01
    healthcheck:
      test:  /redis/redis-cli ping
      interval: 10s
      timeout: 5s
      retries: 10
redis02:
    image: myredis:v1.0
    container_name: redis02
    healthcheck:
      test:  /redis/redis-cli ping
      interval: 10s
```

```
        timeout: 5s
        retries: 10
  redis03:
      image: myredis:v1.0
      container_name: redis03
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis04:
      image: myredis:v1.0
      container_name: redis04
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis05:
      image: myredis:v1.0
      container_name: redis05
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis06:
      image: myredis:v1.0
      container_name: redis06
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis07:
      image: myredis:v1.0
      container_name: redis07
      entrypoint: ["/redis/redis-cli","--
cluster","create","redis01:6379","redis02:6379","redis03:6379","re
dis04:6379","redis05:6379","redis06:6379","--cluster-
replicas","1","-a","123456","--cluster-yes"]
      depends_on:
        redis01:
         condition: service_healthy
```

```
        redis02:
         condition: service_healthy
        redis03:
         condition: service_healthy
        redis04:
         condition: service_healthy
        redis05:
         condition: service_healthy
        redis06:
         condition: service_healthy
```

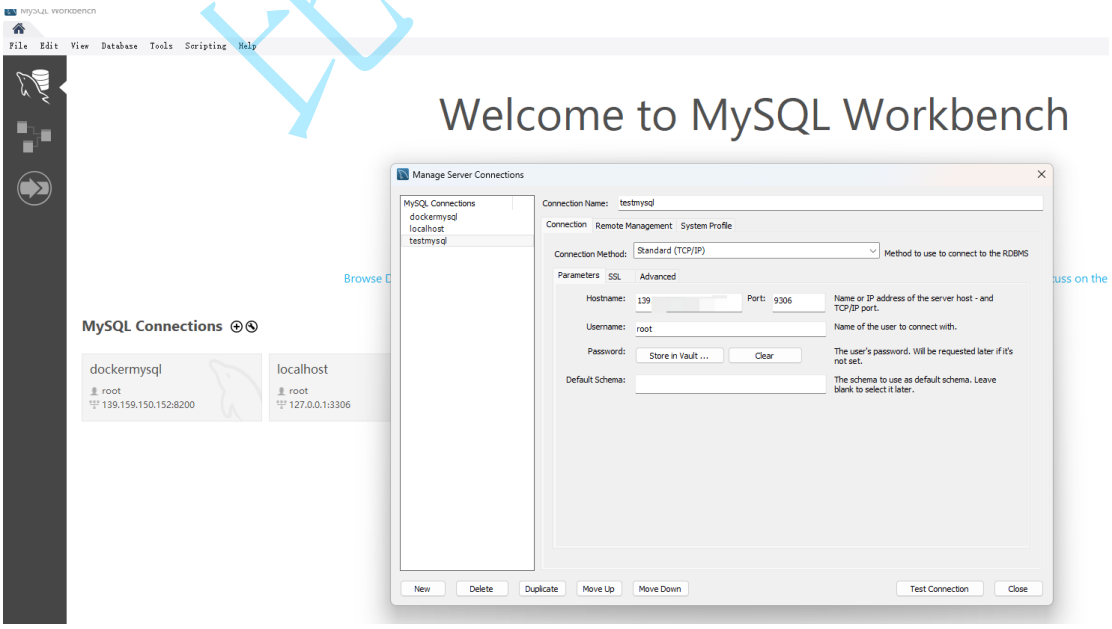
5. 构建镜像，启动 mysql 和 redis 集群。

```
Shell
[root@VM-0-6-centos mymicroservice]# docker compose build
[root@VM-0-6-centos mymicroservice]# docker compose up -d
[+] Running 10/10
 ✔ Network mymicroservice_default  Created
0.1s
 ✔ Container redis01               Healthy
0.1s
 ✔ Container redis06               Healthy
0.1s
 ✔ Container redis02               Healthy
0.1s
 ✔ Container redis03               Healthy
0.1s
 ✔ Container redis05               Healthy
0.1s
 ✔ Container mysql-master          Started
0.1s
 ✔ Container redis04               Healthy
0.1s
 ✔ Container redis07               Started
0.0s
 ✔ Container mysql-slave           Started
 [root@VM-0-6-centos mymicroservice]# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED
STATUS                    PORTS
NAMES
a36d805ad1da   mysqlslave:v2.0    "docker-entrypoint.s…"   58
seconds ago   Up 55 seconds             33060/tcp, 0.0.0.0:9307-
>3306/tcp, :::9307->3306/tcp   mysql-slave
```

```
d5bdeff4dd44   mysqlmaster:v2.0   "docker-entrypoint.s…"   59
seconds ago   Up 42 seconds              33060/tcp, 0.0.0.0:9306-
>3306/tcp, :::9306->3306/tcp   mysql-master
e3d680944470   myredis:v1.0       "/redis/redis-server…"   59
seconds ago   Up 57 seconds (healthy)   0.0.0.0:16379-
>6379/tcp, :::16379->6379/tcp            redis02
95719bccb465   myredis:v1.0       "/redis/redis-server…"   59
seconds ago   Up 55 seconds (healthy)   0.0.0.0:26379-
>6379/tcp, :::26379->6379/tcp            redis03
159e6772e291   myredis:v1.0       "/redis/redis-server…"   59
seconds ago   Up 55 seconds (healthy)   0.0.0.0:46379-
>6379/tcp, :::46379->6379/tcp            redis05
fec3ff0053f7   myredis:v1.0       "/redis/redis-server…"   59
seconds ago   Up 56 seconds (healthy)   0.0.0.0:36379-
>6379/tcp, :::36379->6379/tcp            redis04
438e12036c4a   myredis:v1.0       "/redis/redis-server…"   59
seconds ago   Up 56 seconds (healthy)   0.0.0.0:6379-
>6379/tcp, :::6379->6379/tcp             redis01
bfe8930d802d   myredis:v1.0       "/redis/redis-server…"   59
seconds ago   Up 57 seconds (healthy)   0.0.0.0:56379-
>6379/tcp, :::56379->6379/tcp            redis06
```

6. 我们使用本地客户端工具 workbench 或者 navicat 连接 mysql，redis 可以进入容器连接上我们创建的 redis 集群，注意如果是云服务器需要通过安全组放通对应的端口策略才能本地访问。

- mysql 检查：



- redis 检查：进入容器，测试 redis 功能

```Shell
[root@VM-0-6-centos mymicroservice]# docker exec -it redis01 bash
[root@9e6550e7c1bb /]# cd /redis
[root@9e6550e7c1bb redis]# ./redis-cli -c -a 123456
Warning: Using a password with '-a' or '-u' option on the command
line interface may not be safe.
127.0.0.1:6379> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:6
cluster_my_epoch:1
cluster_stats_messages_ping_sent:1539
cluster_stats_messages_pong_sent:1556
cluster_stats_messages_sent:3095
cluster_stats_messages_ping_received:1551
cluster_stats_messages_pong_received:1539
cluster_stats_messages_meet_received:5
cluster_stats_messages_received:3095
total_cluster_links_buffer_limit_exceeded:0
```

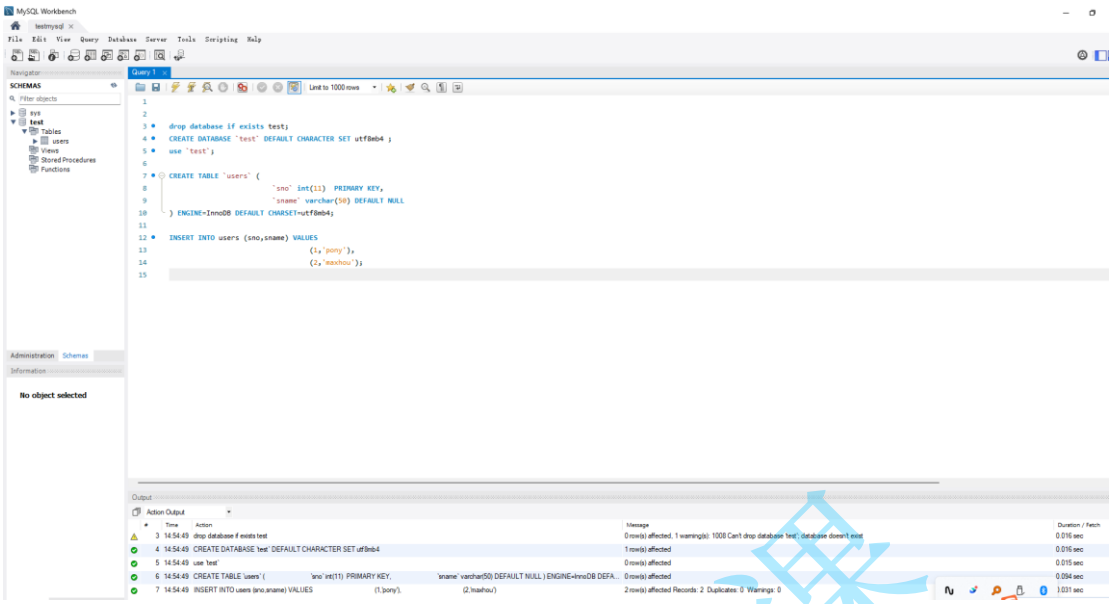
7. 使用可视化工具执行脚本，初始化数据库

```Shell


drop database if exists test;
CREATE DATABASE `test` DEFAULT CHARACTER SET utf8mb4 ;
use `test`;

CREATE TABLE `users` (
                         `sno` int(11)  PRIMARY KEY,
                         `sname` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO users (sno,sname) VALUES
                                    (1,'pony'),
                                    (2,'maxhou');
```



8. 创建 springboot 的 maven 工程，可以直接复用之前 docker-compose 实战中的工程，配置 pom.xml 添加 mysql 和 redis 对应的 pom 依赖信息

```Shell
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.11</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.bit</groupId>
    <artifactId>testmymysql</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>testmymysql</name>
    <description>testmymysql</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
```

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

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.49</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-
redis</artifactId>
        </dependency>

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

        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
            <version>2.11.1</version>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

9. 添加一个简单的用户 bean

```Shell
package com.bit.testmymysql.bean;

import lombok.Data;

@Data
public class UserBean {
    //用户 id
    private Integer sno;

    //用户名称
    private String sname;

}
```

10. 添加 redis 使用 json 来完成存储的配置

```Shell
package com.bit.testmymysql.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializer;

@Configuration
public class CustomRedis {
    @Bean
    public RedisTemplate<String,Object>
redisSerTemplate(RedisConnectionFactory redisConnectionFactory) {
```

```
        RedisTemplate<String, Object> redisTemplate = new
RedisTemplate<>();

        //设置工厂

redisTemplate.setConnectionFactory(redisConnectionFactory);

        //设置序列化器
        redisTemplate.setKeySerializer(RedisSerializer.string());

redisTemplate.setValueSerializer(RedisSerializer.string());


redisTemplate.setHashKeySerializer(RedisSerializer.string());

redisTemplate.setHashValueSerializer(RedisSerializer.string());

        return redisTemplate;
    }
}
```

11. 创建简单控制器

```
Shell
package com.bit.testmymysql;

import com.bit.testmymysql.bean.UserBean;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/user")
public class UserController {
    @Autowired
    UserService userService ;

    @GetMapping("/userGet/{id}")
    public Object getUsersById(@PathVariable(name = "id") Integer
sno){
```

```
        return userService.getUsersById(sno);
    }


}
```

12. 创建用户服务

```Shell
package com.bit.testmymysql;

import com.bit.testmymysql.bean.UserBean;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Service
public class UserService {

    @Autowired
    StringRedisTemplate stringRedisTemplate;

    @Autowired
    JdbcTemplate jdbcTemplate;

    ObjectMapper objectMapper = new ObjectMapper();

    @GetMapping("/userGet/{id}")
    public Object getUsersById(@PathVariable(name = "id") Integer
sno){
        String userKey = "u:"+sno;

        //先查询缓存
        if (stringRedisTemplate.hasKey(userKey)){
            try {
                return
objectMapper.readValue(stringRedisTemplate.opsForValue().get(userK
```

```
ey),UserBean.class) ;
            } catch (JsonProcessingException e) {
                throw new RuntimeException(e);
            }
        }

        //再查询数据库
        String sql = "select * from users where sno= ?";

        UserBean userBean= new UserBean();
        try{
             userBean= jdbcTemplate.queryForObject(sql, new
BeanPropertyRowMapper<>(UserBean.class),sno);
        }
        catch (Exception e){
            return userBean;
        }

        //查询数据库说明没缓存，添加下缓存下次直接查询redis
        try {
            if (userBean != null){

stringRedisTemplate.opsForValue().set(userKey,objectMapper.writeVa
lueAsString(userBean));
            }
            else {
                return new UserBean();
            }
        } catch (Exception e) {
            //缓存更新失败，可以再度重试，再度失败可以引入消息中间件记
录失败信息过段时间再重试
            throw new RuntimeException(e);
        }

        return userBean;
    }


}
```

13. 配置默认的应用

```Shell
package com.bit.testmymysql;
```

```
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TestmymysqlApplication {

    public static void main(String[] args) {
        SpringApplication.run(TestmymysqlApplication.class, args);
    }

}
```

14. 配置 application-docker.yml 来读取 mysql 和 redis

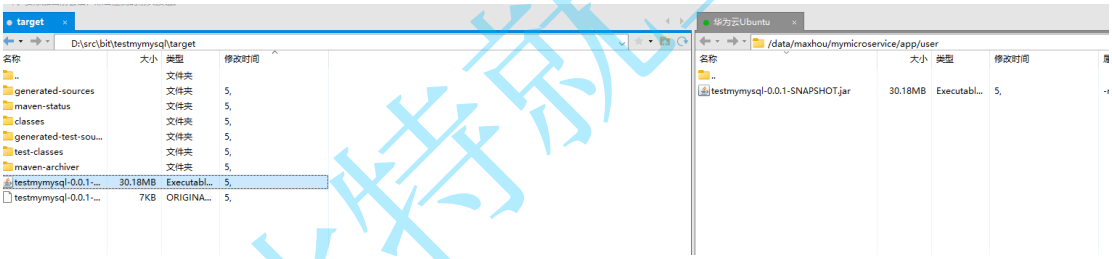
```
Shell
spring:
  datasource:
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://mysql-master:3306/test
    username: root
    password: root
  redis:
    password: 123456
    lettuce:
      pool:
        max-active: 50
        max-idle: 5
        min-idle: 0
        max-wait: 5000
    database: 0
    cluster:
      nodes:
redis01:6379,redis02:6379,redis03:6379,redis04:6379,redis05:6379,r
edis06:6379
    #host: 127.0.0.1
    connect-timeout: 5000
    timeout: 5000
```

15. 配置 application-local.yml 来测试，注意 ip 地址为我们集群所在服务器的 ip 地址。

```
Shell
spring:
```

```
  datasource:
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://172.27.0.6:9306/test
    username: root
    password: root
  redis:
    password: 123456
    lettuce:
      pool:
        max-active: 50
        max-idle: 5
        min-idle: 0
        max-wait: 5000
    database: 0
    cluster:
      nodes: 172.27.0.6:6379
    #host: 127.0.0.1
    connect-timeout: 5000
    timeout: 5000
```

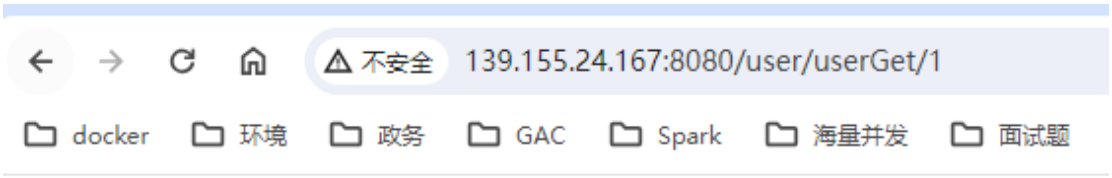
16. 上传 jar 包到 java/user 目录下面



17. 执行以下命令来测试我们的服务

```Shell
java -jar testmymysql-0.0.1-SNAPSHOT.jar --
spring.profiles.active=local
```

18. 通过浏览器访问，可以看到我们的功能正常，如果想本地调试，也可以在本地搭建一个 redis 或者 mysql 连完成本地的调试。



19. 查看 redis 中信息，可以看到 redis 中已经缓存我们的用户信息

```
Shell
[root@9e6550e7c1bb /]# /redis/redis-cli  -c -a 123456
Warning: Using a password with '-a' or '-u' option on the command
line interface may not be safe.
127.0.0.1:6379> get s1
172.25.0.7:6379> get u:1
"{\"sno\":1,\"sname\":\"pony\"}"
172.25.0.7:6379>
```

20. 下面开始制作我们的 java 镜像的 dockerfile

```
Shell
root@139-159-150-152:/data/maxhou/mymicroservice/app/user#
root@139-159-150-152:/data/maxhou/mymicroservice/app/user# vi
Dockerfile


# 指定 java 环境基础镜像
FROM openjdk:8
# 拷贝 jar 包到容器中
ADD ./testmymysql-0.0.1-SNAPSHOT.jar /app.jar
# 运行 jar 包
CMD ["java", "-jar", "/app.jar","--spring.profiles.active=docker"]
```

21. 制作 nginx 的配置文件 bit.conf 放到 /data/maxhou/mymicroservice/nginx

```
Shell

upstream myapi{
      server myuser:8080;
      server myuser2:8080;
}
```

```
server {
    listen 80;
    access_log off;


    location / {
        proxy_pass http://myapi/user/;
    }
}
```

22. 制作 nginx 的 dockerfile，进入目录 /data/maxhou/mymicroservice/nginx，vi Dockerfile

```Shell
FROM nginx:1.24.0
COPY ./bit.conf /etc/nginx/conf.d/
CMD ["nginx","-g","daemon off;"]
ENTRYPOINT ["/docker-entrypoint.sh"]
```

23. 进入目录/data/maxhou/mymicroservice，添加应用容器编排文件 docker-compose.yml

```Shell
version: "3"
services:
  mysql-master:
    build:
      context: ./mysql
      dockerfile: ./master/Dockerfile-master
    image: mysqlmaster:v2.0
    restart: always
    container_name: mysql-master
    volumes:
      - ./mastervarlib:/var/lib/mysql
    ports:
      - 9306:3306
    environment:
      MYSQL_ROOT_PASSWORD: root
    privileged: true
    command: ['--server-id=1',
              '--log-bin=master-bin',
              '--binlog-ignore-db=mysql',
              '--binlog_cache_size=256M',
```

```
            '--binlog_format=mixed',
            '--lower_case_table_names=1',
            '--character-set-server=utf8',
            '--collation-server=utf8_general_ci']
mysql-slave:
  build:
    context: ./mysql
    dockerfile: ./slave/Dockerfile-slave
  image: mysqlslave:v2.0
  restart: always
  container_name: mysql-slave
  volumes:
    - ./slavevarlib:/var/lib/mysql
  ports:
    - 9307:3306
  environment:
    MYSQL_ROOT_PASSWORD: root
  privileged: true
  command: ['--server-id=2',
            '--relay_log=slave-relay',
            '--lower_case_table_names=1',
            '--character-set-server=utf8',
            '--collation-server=utf8_general_ci']
  depends_on:
    - mysql-master
redis01:
    image: myredis:v1.0
    build: ./redis
    ports:
      - 6379:6379
    container_name: redis01
    healthcheck:
      test:  /redis/redis-cli ping
      interval: 10s
      timeout: 5s
      retries: 10
redis02:
    image: myredis:v1.0
    container_name: redis02
    healthcheck:
      test:  /redis/redis-cli ping
      interval: 10s
      timeout: 5s
      retries: 10
```

```
  redis03:
      image: myredis:v1.0
      container_name: redis03
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis04:
      image: myredis:v1.0
      container_name: redis04
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis05:
      image: myredis:v1.0
      container_name: redis05
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis06:
      image: myredis:v1.0
      container_name: redis06
      healthcheck:
        test:  /redis/redis-cli ping
        interval: 10s
        timeout: 5s
        retries: 10
  redis07:
      image: myredis:v1.0
      container_name: redis07
      entrypoint: ["/redis/redis-cli","--
cluster","create","redis01:6379","redis02:6379","redis03:6379","re
dis04:6379","redis05:6379","redis06:6379","--cluster-
replicas","1","-a","123456","--cluster-yes"]
      depends_on:
        redis01:
         condition: service_healthy
        redis02:
         condition: service_healthy
```

```
      redis03:
       condition: service_healthy
      redis04:
       condition: service_healthy
      redis05:
       condition: service_healthy
      redis06:
       condition: service_healthy
web:
  image: mynginx:v2.0
  build:
    context: ./nginx
  ports:
    - 80:80
  depends_on:
    myuser:
       condition: service_started
    myuser2:
       condition: service_started
myuser:
  image: myuser:v2.0
  build:
    context: ./app/user
  depends_on:
      redis01:
       condition: service_healthy
      redis02:
       condition: service_healthy
      redis03:
       condition: service_healthy
      redis04:
       condition: service_healthy
      redis05:
       condition: service_healthy
      redis06:
       condition: service_healthy
      mysql-master:
       condition: service_started
myuser2:
  image: myuser:v2.0
  build:
    context: ./app/user
  depends_on:
      redis01:
```

```
         condition: service_healthy
       redis02:
         condition: service_healthy
       redis03:
         condition: service_healthy
       redis04:
         condition: service_healthy
       redis05:
         condition: service_healthy
       redis06:
         condition: service_healthy
       mysql-master:
         condition: service_started
```

24. 批量构建镜像

```
Shell
[root@VM-0-6-centos mymicroservice]# docker compose build
[+] Building 2.4s (43/44)
docker:default
 => [redis01 internal] load build definition from Dockerfile
0.1s
 => => transferring dockerfile: 905B
0.0s
 => [redis01 internal] load .dockerignore
0.1s
 => => transferring context: 2B
0.0s
 => [mysql-master internal] load .dockerignore
0.1s
 => => transferring context: 2B
0.0s
 => [mysql-master internal] load build definition from Dockerfile-
master
0.1s
 => => transferring dockerfile: 174B
0.0s
 => [redis01 internal] load metadata for
docker.io/library/centos:7
0.0s
 => [redis01 buildstage  1/10] FROM docker.io/library/centos:7
0.0s
 => [redis01 internal] load build context
0.0s
```

```
 => => transferring context: 74B
0.0s
 => [mysql-slave internal] load metadata for
docker.io/library/mysql:5.7
0.0s
 => [mysql-slave 1/3] FROM docker.io/library/mysql:5.7
0.0s
 => [mysql-master internal] load build context
0.0s
 => => transferring context: 612B
0.0s
 => CACHED [redis01 stage-1 2/3] RUN mkdir -p /data/redis && mkdir
-p /redis
0.0s
 => CACHED [redis01 buildstage  2/10] RUN  sed -e
's|^mirrorlist=|#mirrorlist=|g'          -e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=ht  0.0s
 => CACHED [redis01 buildstage  3/10] RUN yum install -y centos-
release-scl
0.0s
 => CACHED [redis01 buildstage  4/10] RUN yum makecache
0.0s
 => CACHED [redis01 buildstage  5/10] RUN yum install -y
devtoolset-9-gcc devtoolset-9-gcc-c++ devtoolset-9-binutils make
0.0s
 => CACHED [redis01 buildstage  6/10] ADD redis-7.0.11.tar.gz /
0.0s
 => CACHED [redis01 buildstage  7/10] ADD redis.conf /redis/
0.0s
 => CACHED [redis01 buildstage  8/10] WORKDIR /redis-7.0.11
0.0s
 => CACHED [redis01 buildstage  9/10] RUN source
/opt/rh/devtoolset-9/enable&& make
0.0s
 => CACHED [redis01 buildstage 10/10] RUN mv /redis-
7.0.11/src/redis-server /redis/ && mv /redis-7.0.11/src/redis-cli
/redis/                           0.0s
 => CACHED [redis01 stage-1 3/3] COPY --from=buildstage /redis
/redis
0.0s
 => [web] exporting to image
0.6s
 => => exporting layers
0.0s
```

```
 => => writing image
sha256:ef35b827a806f689db217304605a971018fddf3c603fd902cd1d24d91f4
29eb7
0.0s
 => => naming to docker.io/library/myredis:v1.0
0.0s
 => => writing image
sha256:9014b08c00d5bd7cd2171df9c6b661910f3928d16312a830105b2d162cd
c59ed
0.0s
 => => naming to docker.io/library/mysqlmaster:v2.0
0.0s
 => => writing image
sha256:d3ea5affd842353709574b33b79d4facc4a12ab4fd258de7d241c07c976
3a2b9
0.0s
 => => naming to docker.io/library/mysqlslave:v2.0
0.0s
 => => writing image
sha256:bcd311cad9ece006171aff6b8f9e2f573f51967cabbf40a54e840a10363
8a911
0.0s
 => => naming to docker.io/library/myuser:v2.0
0.0s
 => => writing image
sha256:6a5ec8935a382ed88b019af0c9768c3dc2edcf08a0db6c26fdc4cc7f968
bdf5c
0.0s
 => => writing image
sha256:6d518aea7845ce566f57c8800a3cfb63a047a8e74f61f73f84ae1b13ba9
be13a
0.0s
 => => naming to docker.io/library/mynginx:v2.0
0.0s
 => CACHED [mysql-slave 2/3] RUN ln -sf
/usr/share/zoneinfo/Asia/Shanghai /etc/localtime
0.0s
 => CACHED [mysql-master 3/3] COPY ./master/master.sql /docker-
entrypoint-initdb.d
0.0s
 => [mysql-slave internal] load build definition from Dockerfile-
slave
0.0s
 => => transferring dockerfile: 169B
```

```
0.0s
 => [mysql-slave internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [myuser2 internal] load build definition from Dockerfile
0.1s
 => => transferring dockerfile: 240B
0.0s
 => [myuser2 internal] load .dockerignore
0.1s
 => => transferring context: 2B
0.0s
 => [myuser internal] load .dockerignore
0.2s
 => => transferring context: 2B
0.0s
 => [myuser internal] load build definition from Dockerfile
0.2s
 => => transferring dockerfile: 240B
0.0s
 => [mysql-slave internal] load build context
0.1s
 => => transferring context: 189B
0.0s
 => [myuser internal] load metadata for
docker.io/library/openjdk:8
0.0s
 => [myuser2 internal] load build context
0.7s
 => => transferring context: 31.65MB
0.5s
 => [myuser2 1/2] FROM docker.io/library/openjdk:8
0.9s
 => CACHED [mysql-slave 3/3] COPY ./slave/slave.sql /docker-
entrypoint-initdb.d
0.1s
 => [myuser internal] load build context
0.8s
 => => transferring context: 31.65MB
0.5s
 => [myuser2 2/2] ADD ./testmymysql-0.0.1-SNAPSHOT.jar /app.jar
0.2s
 => [web internal] load build definition from Dockerfile
```

```
0.1s
 => => transferring dockerfile: 160B
0.0s
 => [web internal] load .dockerignore
0.1s
 => => transferring context: 2B
0.0s
 => [web internal] load metadata for
docker.io/library/nginx:1.24.0
0.0s
 => [web internal] load build context
0.1s
 => => transferring context: 242B
0.0s
 => [web 1/2] FROM docker.io/library/nginx:1.24.0
0.2s
 => [web 2/2] COPY ./bit.conf /etc/nginx/conf.d/
0.0s
```

25. 启动服务

```
Shell
[root@VM-0-6-centos mymicroservice]# docker compose up -d
[+] Running 12/12
 ✔ Container redis05                 Healthy
0.9s
 ✔ Container redis02                 Healthy
0.9s
 ✔ Container mysql-master            Running
0.0s
 ✔ Container mysql-slave             Running
0.0s
 ✔ Container redis03                 Healthy
0.8s
 ✔ Container redis06                 Healthy
1.0s
 ✔ Container redis01                 Healthy
0.0s
 ✔ Container redis04                 Healthy
0.9s
 ✔ Container mymicroservice-myuser2-1  Started
0.1s
```

```
 ✔ Container redis07                    Started
0.1s
 ✔ Container mymicroservice-myuser-1    Started
0.1s
 ✔ Container mymicroservice-web-1       Started
0.0s
```

26. 测试服务，因为我们已经配置了 nginx 的反向代理，请求地址变为 http://服务器 ip/userGet/{用户 id}

27. 查看两个用户服务的日志，可以看到 2 个服务会轮流请求，也就是发生了负载均衡

```
Shell
[root@VM-0-6-centos mymicroservice]# docker logs -f 1d941d463134



  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.11)

2024-01-17 08:21:54.620  INFO 1 --- [           main]
c.b.testmymysql.TestmymysqlApplication   : Starting
TestmymysqlApplication v0.0.1-SNAPSHOT using Java 1.8.0_342 on
1d941d463134 with PID 1 (/app.jar started by root in /)
2024-01-17 08:21:54.639  INFO 1 --- [           main]
c.b.testmymysql.TestmymysqlApplication   : The following 1 profile
is active: "docker"
2024-01-17 08:21:58.425  INFO 1 ---
[           main] .s.d.r.c.RepositoryConfigurationDelegate :
Multiple Spring Data modules found, entering strict repository
configuration mode
```

```
2024-01-17 08:21:58.434  INFO 1 ---
[           main] .s.d.r.c.RepositoryConfigurationDelegate :
Bootstrapping Spring Data Redis repositories in DEFAULT mode.
2024-01-17 08:21:58.635  INFO 1 ---
[           main] .s.d.r.c.RepositoryConfigurationDelegate :
Finished Spring Data repository scanning in 30 ms. Found 0 Redis
repository interfaces.
2024-01-17 08:22:02.081  INFO 1 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with
port(s): 8080 (http)
2024-01-17 08:22:02.136  INFO 1 --- [           main]
o.apache.catalina.core.StandardService   : Starting service
[Tomcat]
2024-01-17 08:22:02.153  INFO 1 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet
engine: [Apache Tomcat/9.0.74]
2024-01-17 08:22:02.497  INFO 1 --- [           main]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
embedded WebApplicationContext
2024-01-17 08:22:02.509  INFO 1 --- [           main]
w.s.c.ServletWebServerApplicationContext : Root
WebApplicationContext: initialization completed in 7629 ms
2024-01-17 08:22:08.895  INFO 1 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on
port(s): 8080 (http) with context path ''
2024-01-17 08:22:08.934  INFO 1 --- [           main]
c.b.testmymysql.TestmymysqlApplication   : Started
TestmymysqlApplication in 16.723 seconds (JVM running for 19.249)
2024-01-17 08:22:37.316  INFO 1 --- [nio-8080-exec-1]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
DispatcherServlet 'dispatcherServlet'
2024-01-17 08:22:37.319  INFO 1 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet        : Initializing Servlet
'dispatcherServlet'
2024-01-17 08:22:37.340  INFO 1 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet        : Completed
initialization in 21 ms
query user1
query user1
query user1
query user1
query user1

[root@VM-0-6-centos mymicroservice]# docker logs -f 1d941d463134
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.11)

2024-01-17 08:21:54.620  INFO 1 --- [           main]
c.b.testmymysql.TestmymysqlApplication   : Starting
TestmymysqlApplication v0.0.1-SNAPSHOT using Java 1.8.0_342 on
1d941d463134 with PID 1 (/app.jar started by root in /)
2024-01-17 08:21:54.639  INFO 1 --- [           main]
c.b.testmymysql.TestmymysqlApplication   : The following 1 profile
is active: "docker"
2024-01-17 08:21:58.425  INFO 1 ---
[           main] .s.d.r.c.RepositoryConfigurationDelegate :
Multiple Spring Data modules found, entering strict repository
configuration mode
2024-01-17 08:21:58.434  INFO 1 ---
[           main] .s.d.r.c.RepositoryConfigurationDelegate :
Bootstrapping Spring Data Redis repositories in DEFAULT mode.
2024-01-17 08:21:58.635  INFO 1 ---
[           main] .s.d.r.c.RepositoryConfigurationDelegate :
Finished Spring Data repository scanning in 30 ms. Found 0 Redis
repository interfaces.
2024-01-17 08:22:02.081  INFO 1 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with
port(s): 8080 (http)
2024-01-17 08:22:02.136  INFO 1 --- [           main]
o.apache.catalina.core.StandardService   : Starting service
[Tomcat]
2024-01-17 08:22:02.153  INFO 1 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet
engine: [Apache Tomcat/9.0.74]
2024-01-17 08:22:02.497  INFO 1 --- [           main]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
embedded WebApplicationContext
2024-01-17 08:22:02.509  INFO 1 --- [           main]
w.s.c.ServletWebServerApplicationContext : Root
WebApplicationContext: initialization completed in 7629 ms
```

```
2024-01-17 08:22:08.895  INFO 1 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on
port(s): 8080 (http) with context path ''
2024-01-17 08:22:08.934  INFO 1 --- [           main]
c.b.testmymysql.TestmymysqlApplication   : Started
TestmymysqlApplication in 16.723 seconds (JVM running for 19.249)
2024-01-17 08:22:37.316  INFO 1 --- [nio-8080-exec-1]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring
DispatcherServlet 'dispatcherServlet'
2024-01-17 08:22:37.319  INFO 1 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet        : Initializing Servlet
'dispatcherServlet'
2024-01-17 08:22:37.340  INFO 1 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet        : Completed
initialization in 21 ms
query user1
query user1
query user1
query user1
query user1
```

28. 查看 redis 已经写入数据

```Shell
[root@VM-0-6-centos mymicroservice]# docker exec -it redis01 bash
[root@836fc79a6a0e /]# redis-cli -c -a 123456
bash: redis-cli: command not found
[root@836fc79a6a0e /]# /redis/redis-cli -c -a 123456
Warning: Using a password with '-a' or '-u' option on the command
line interface may not be safe.
127.0.0.1:6379> get u:1
-> Redirected to slot [16340] located at 172.28.0.2:6379
"{\"sno\":1,\"sname\":\"pony\"}"
172.28.0.2:6379> get u:2
-> Redirected to slot [4023] located at 172.28.0.8:6379
"{\"sno\":2,\"sname\":\"maxhou\"}"
172.28.0.8:6379>
```

29. 至此我们完整的一个微服务拓扑就开发完了，实际工作中，我们微服务往往是多个，而且有对应的注册中心，配置中心，网关等一系列微服务，最后我们释放资源

```Shell
root@139-159-150-152:/data/maxhou/mymicroservice# docker compose
down
[+] Running 12/12
 ✔ Container mymicroservice-web-1     Removed
0.3s
 ✔ Container mysql-slave              Removed
2.0s
 ✔ Container redis07                  Removed
0.0s
 ✔ Container mymicroservice-myuser-1  Removed
0.4s
 ✔ Container redis06                  Removed
0.8s
 ✔ Container redis01                  Removed
0.8s
 ✔ Container redis02                  Removed
1.0s
 ✔ Container redis03                  Removed
1.0s
 ✔ Container redis04                  Removed
1.0s
 ✔ Container redis05                  Removed
0.3s
 ✔ Container mysql-master             Removed
10.3s
 ✔ Network mymicroservice_default     Removed
0.1s
```