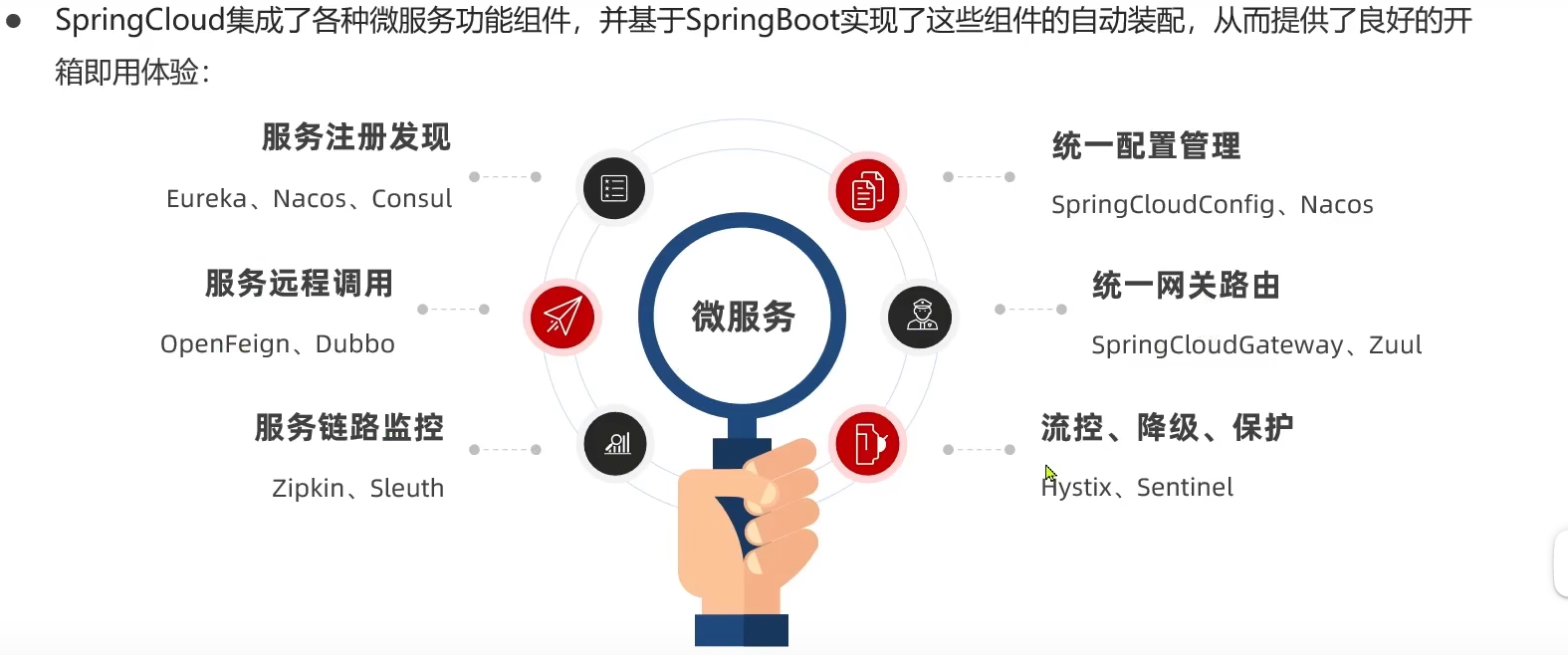
# SpringCloud微服务

1. **第一部分**
   1. **演变**
      1. **单体架构**：将业务的所有功能集中在一个项目中开发，打包成一个包部署（适合小型项目）
         1. 优点：机构简单、部署成本低
         2. 缺点：耦合度高
      2. **分布式架构**：根据业务对系统进行拆分，每个业务模块作为独立项目开发，称为一个服务（适合大型项目）
         1. 优点：降低服务耦合、有利于服务升级拓展
      3. **微服务**：一种经过良好架构设计的分布式架构方案
         1. 架构特征：
         2. 单一职责：微服务拆分粒度更小，每一个服务都对应唯一的业务能力，做到单一职责，避免重复业务开发
         3. 面向服务：微服务对外暴露业务接口
         4. 自治：团队独立、技术独立、数据独立、部署独立隔离性强：服务调用做好隔离，容错、降级、避免出现级联问题
      4. **微服务**
         1. 注册中心：拉取或注册服务信息（服务集群）
         2. 配置中心：拉取配置信息（服务集群）
         3. 服务网关：请求路由负载均衡（服务集群）
   2. **SpringCloud**





* 1. 微服务治理
     1. 同时启动不同端口的两个实例[(65条消息) 如何在IntelliJ Idea中同时启动不同端口的两个实例\_淸風是也的博客-CSDN博客](https://blog.csdn.net/u010999223/article/details/88547932)
     2. Eureka、Nacos
        1. **服务调用关系**（服务拆分）
           1. 服务提供者：暴露接口给其它微服务调用
           2. 服务消费者：调用其它微服务提供的接口
           3. 提供者与消费者角色其实是相对的
        2. Eureka
           1. EurekaServer：服务端，注册中心

记录服务信息

心跳监控

* + - * 1. EurekaClient：客户端

Provider：服务提供者

注册自己的信息到EurekaServer

每隔30秒向EurekaServer发送心跳

consumer：服务消费者

根据服务名称从EurekaServer拉取服务列表

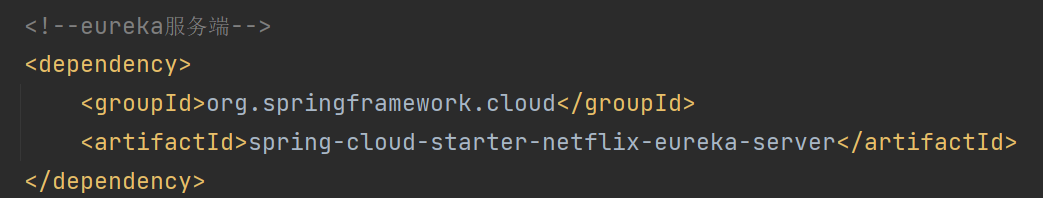
基于服务列表做负载均衡，选中一个微服务发起远程调用

* + - * 1. 实践

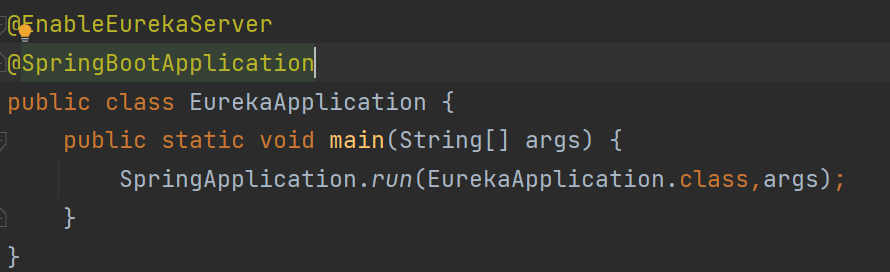
搭建注册中心

搭建EurekaServer

创建项目，导入坐标



编写启动类，添加@EnableEurekaServer注解



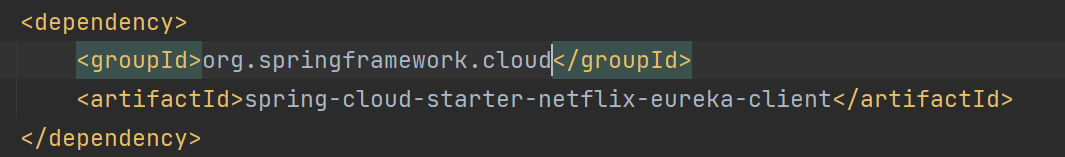
添加application.yml文件



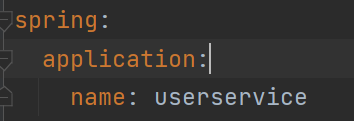
服务注册

将服务注册到Eureka

在某一个服务项目中引入依赖



书写配置文件application.yml配置eureka地址





服务发现

其中一个服务完成服务拉取，然后通过负载均衡挑选一个服务，实现远程调用

引入eureka-client依赖

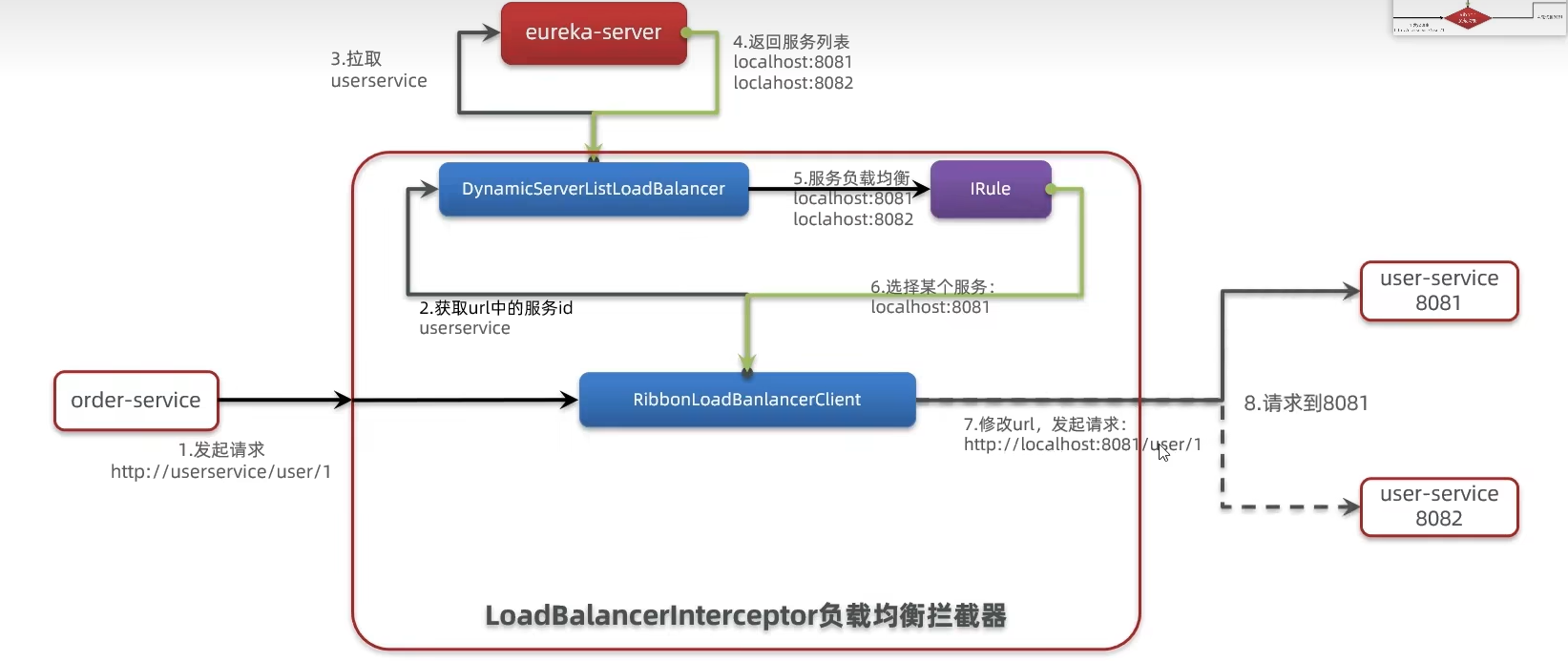
在application.yml中配置eureka地址

给Rest Template添加@LoadBalanced注解

用服务提供者的服务名称远程调用

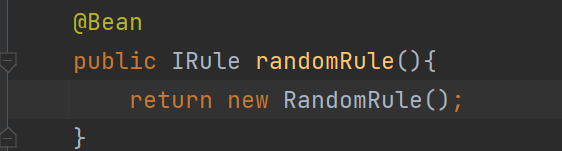
* + - * 1. Ribbon负载均衡

负载均衡流程

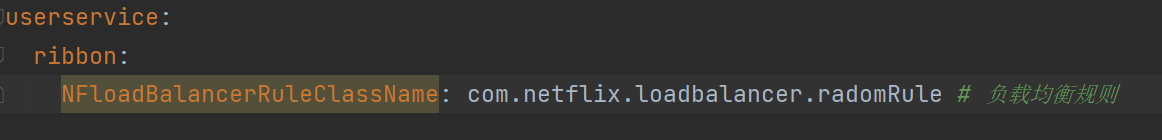


调整负载均衡的规则

代码方式：在Application类中，定义一个新的Rule



配置文件方式：在application.yml中配置



* + - 1. Nacos

阿里巴巴的产品，比Eureka功能丰富，再国内受欢迎程度高

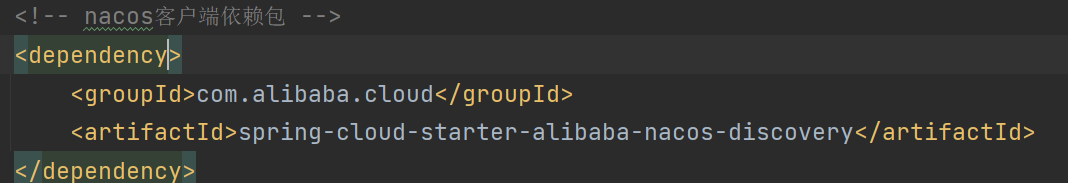
* + - * 1. Nacos注册中心

服务器搭建

下载安装包

解压

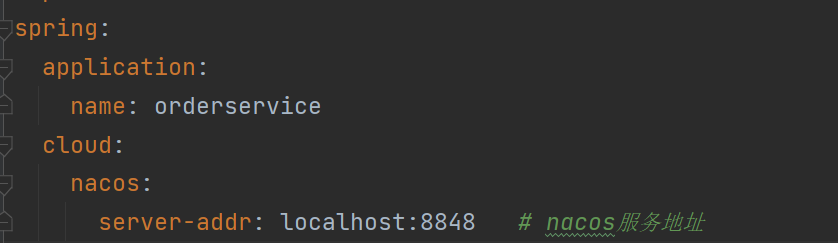
再Bin目录下运行指令startup.cmd -m standalone



Nacos服务注册或发现

引入Nacos.discovery依赖

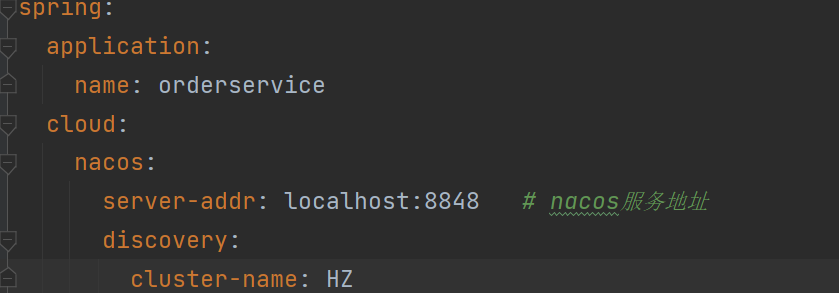
配置Nacos地址spring.cloud.nacos.server-addr



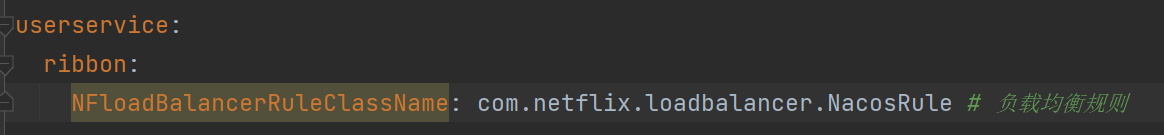
服务跨集群调用问题

服务调用尽可能选择本地集群的服务，跨集群调用延迟较高
本地集群不可访问时，再去访问其他集群

修改order-service中的application.yml设置集群为HZ



然后再order-service中设置负载均衡的Rule为NachosRule，这个规则优先会寻找与自己同集群的服务



NacosRule负载均衡策略

优先选择集群服务实例列表

本地集群找不到提供者，才去其他集群寻找，并且会报警告

确定了可用实例列表后，再采用随机负载均衡挑选实例

实例的权重控制

Nacos控制台可以设置实例的权重值，0~1之间

同集群内的多个实例，权重越高访问的频率越高

权重设置为0则完全不被访问

环境隔离-namespace

Nacos中服务存储和数据存储的最外层都是一个名为namespace的东西，用来做最外层隔离

在nacos控制台可以创建namespace，用来隔离不同环境

填写一个新的命名空间信息

保存后在控制台查看命名空间的id

修改配置文件，添加namespace



实现环境的隔离（不同namespace下的服务不可见）

nacos注册中心细节分析

nacos与eureka的共同点

都支持服务注册和服务拉取

都支持服务提供者心跳方式做健康检测

Nacos与Eureka的区别Nacos支持微服务端主动检测提供者状态：临时实例采用心跳模式，非临时实例采用主动检测模式

临时实例心跳不正常会被剔除，非临时实例则不会被剔除

nacos支持服务列表变更的消息推送模式，服务列表更新更及时

Nacos集群默认采用AP方式，当集群中存在非临时实例时，采用CP模式；Eureka采用AP模式

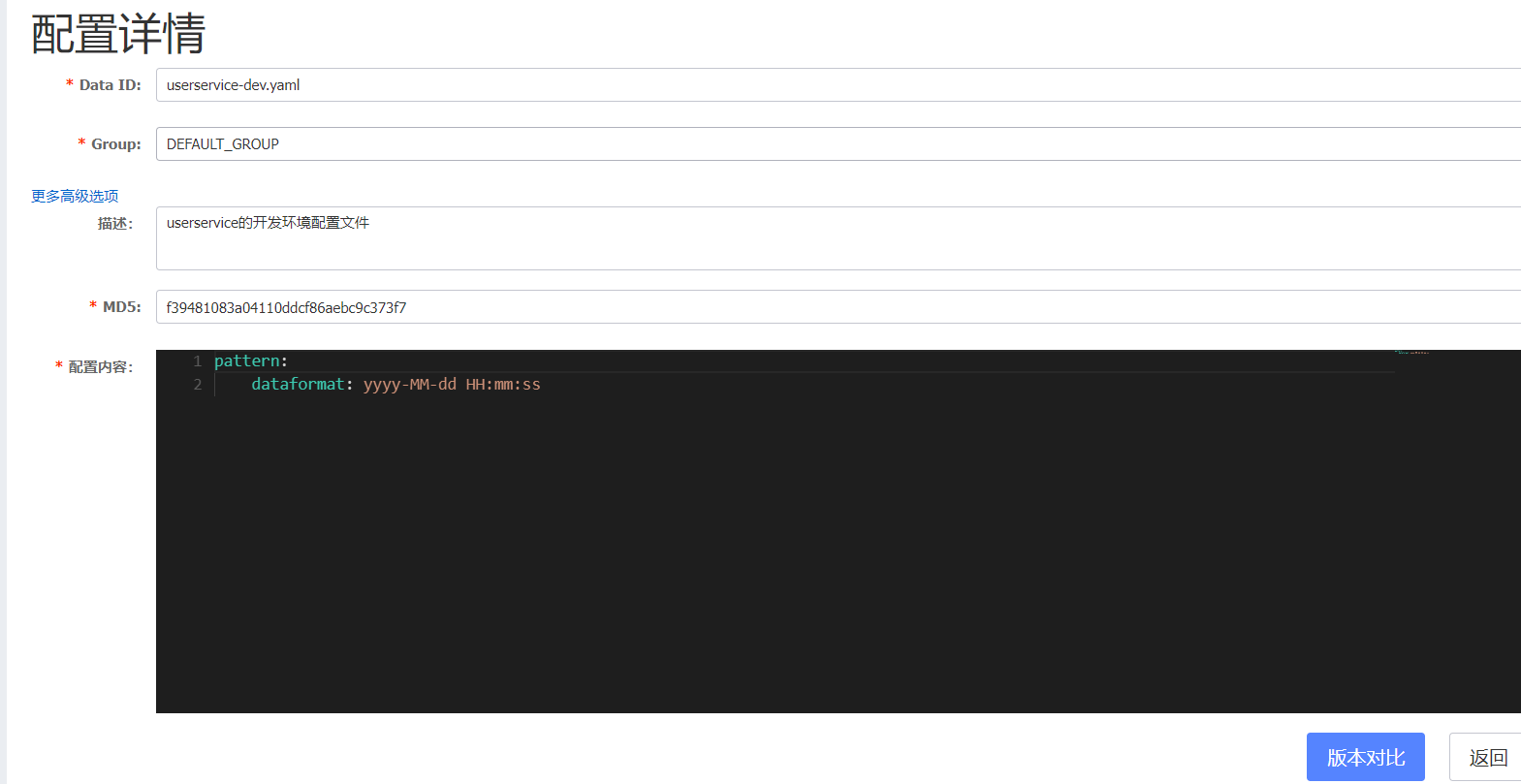
* + - 1. Nacos配置管理
         1. 统一配置管理

配置更改热更新

在nacos中添加配置信息

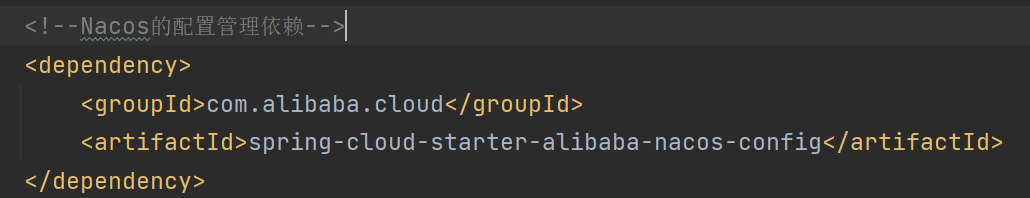


在弹出表单中填写配置信息



配置获取步骤：①项目启动==》（在bootstrap.yml中添加nacos地址==>bootstrap.yml的优先级比application.yml高，后读取地址在将nacos中的配置和本地配置文件结合）读取nacos中的配置文件==》读取本地配置文件application.yml==》闯将spring容器==》加载bean

①在服务中 引入nacos的配置管理客户端依赖

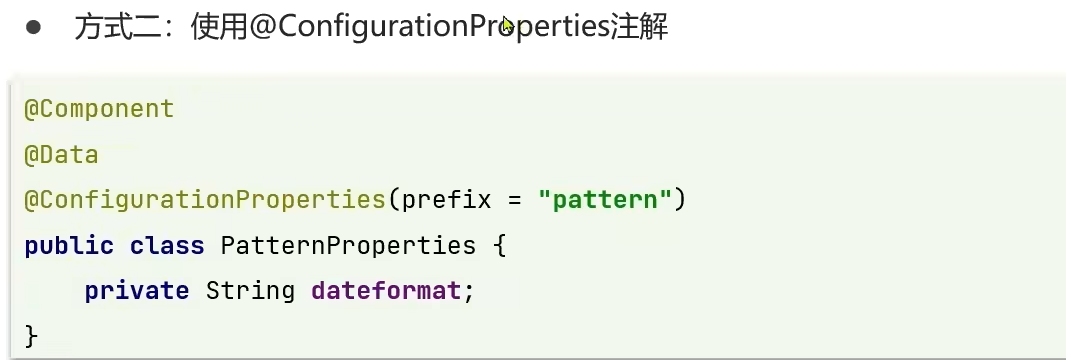


②在userservice中的resource目录添加一个bootstrap.yml文件，这个文件是引导文件，优先级高于application.yml



配置自动更新





注意事项：①不是所有的配置都适合放到配置中心，维护起来比较麻烦

建议将一些关键参数，需要运行时调整参数放到nacos配置中心，一般都是自定义配置

* + - * 1. 多环境配置共享

微服务启动时会从那cos读取多个配置文件

[[spring.application.name](http://spring.application.name/)]-[spring.profiles.active].yaml

[[spring.application.name](http://spring.application.name/)].yaml

无论profile如何变化,[[spring.application.name](http://spring.application.name/)].yaml这个配置文件一定会加载，因此多环境共享配置可以写入这个文件

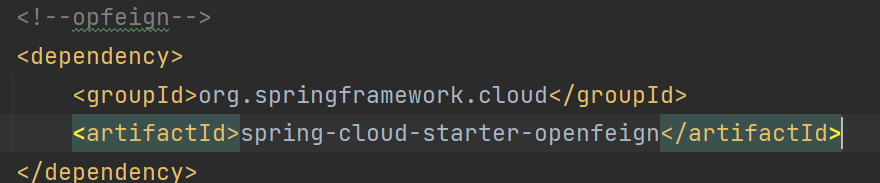
多种配置的优先级:

[服务名]-[环境].yaml(nacos中的配置)>[服务名].yaml（nacos中的配置）>本地配置

* + - 1. Nacos集群搭建
         1. 搭建MySQl集群并初始化数据库表
         2. 下载解压Nacos
         3. 修改集群配置（节点信息），数据库配置
         4. 分别启动多个nacos节点
         5. nginx反向代理
    1. OpenFeign

Feign是一个声明式的http客户端，其作用就是帮助使用者优雅的实现http请求的发送

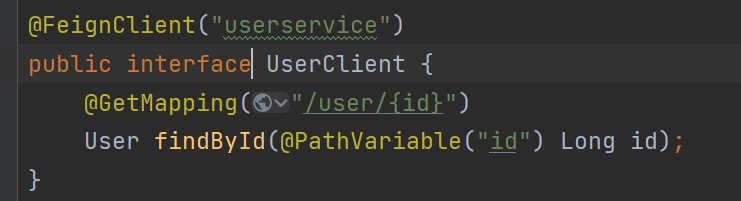
* + - 1. 定义和使用Feign
         1. 引入依赖



* + - * 1. 在order-service的启动类添加注解开启Feign的功能



* + - * 1. 编写Feign客户端



主要基于SpringMvc的注解来声明远程调用的信息，比如：
服务名称：userservice
请求方式：GEt
​请求路径：/user/{id}
请求参数：Long id
返回值类型：User

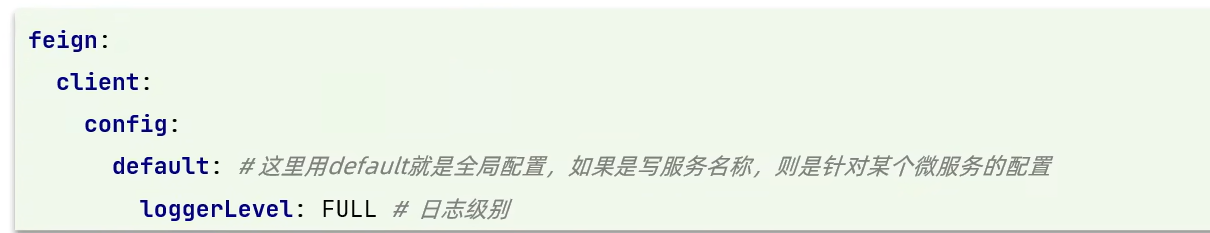
* + - 1. 自定义Feign配置



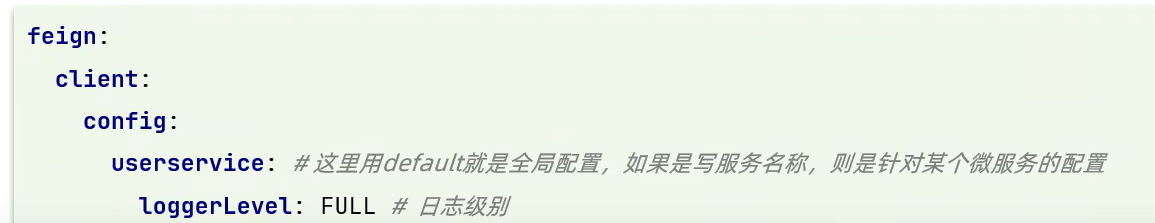
* + - * 1. 配置Feign方式

①配置文件方式

全局生效



局部生效



Java代码方式，需要先声明一个Bean



* + - * 1. Feign的性能优化

Feign底层的客户端实现：

URLConnection：默认实现，不支持连接池

Apache HttpClient：支持连接池

OKHttp：支持连接池

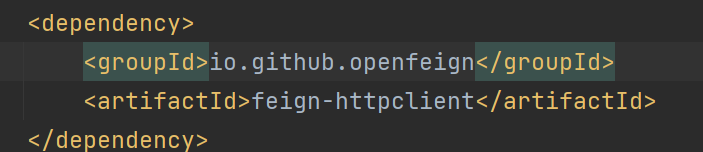
Feign的性能优主要包括：

使用连接词代替默认的URLConnection

日志级别，最好用basic或none

Feign添加HttpClient的支持

引入依赖



配置连接池



Feign最佳实践

方式一（继承）：给消费者的FeignClient和提供者的Controler定义统一父接口作为标准（对SPringMvc不起作用且从API上造成服务紧耦合）

方式二（抽取）：将FeignClient抽取为独立模块，并且把接口有关的POJO，默认的Feign配置都放到这个模块中，提供给所有消费者使用

①先创建一个module，命名为feign-api，然后引入feign的start依赖

②将order-service中编写的UserClient，User，DefaultFeignConfiguration都复制到feign-api项目中

③在order-service中引入feign-api的依赖

④修改irder-service中的所有与上述三个组件相关的import部分，改成导入feign-api中的包

重启测试



* + 1. 网关Gateway
       1. 网关认证
          1. 身份认证和权限校验
          2. 服务路由、负载均衡
          3. 请求限流
       2. 搭建网关服务
          1. 创建新的module，引入SpringCloudGateWay依赖和nacos的服务发现依赖



* + - * 1. 编写路由配置及nacos地址



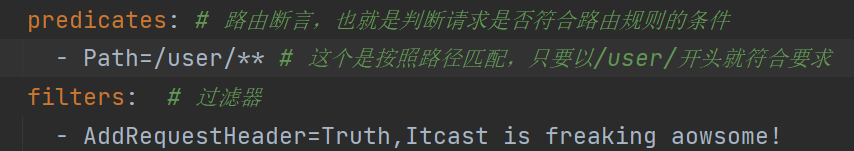
* + - * 1. 路由断言工厂Route Predicate Factory

在配置文件中写的断言规则只是字符串，这些字符串会被Predicate Factory读取处理，转变为路由判断的条件

在时间2031以前符合规则，否则报404



* + - * 1. 路由过滤器GatewayFilter



GateWayFilter是网关中提供的一种过滤器，可以对进入网关的请求和微服务返回的响应做处理

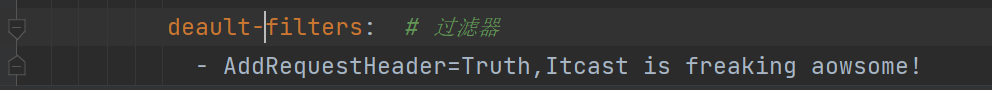
* + - * 1. 全局过滤器GlobalFilter





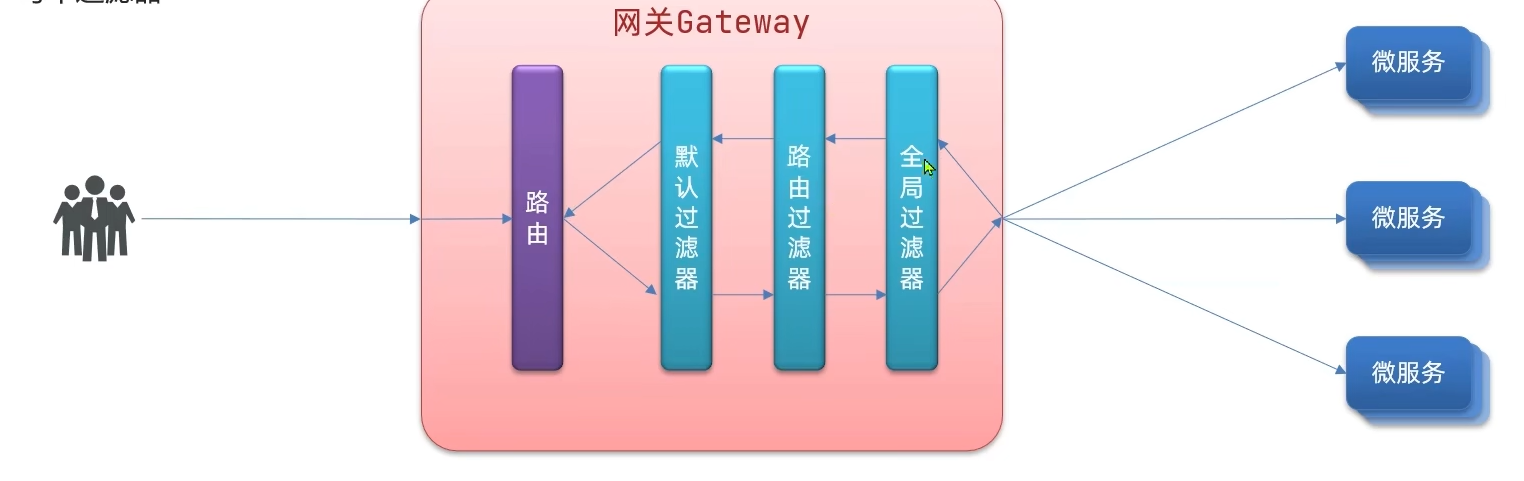
作用是处理一切进入网关的请求和微服务响应，与GatewayFilter的作用一样区别在于GatewayFilter通过配置定义，处理逻辑是固定的，而GlobalFilter的逻辑需要自己手写代码实现
定义方式是实现GlobalFilter接口

* + - * 1. 默认过滤器



对所有路由都生效的过滤器

* + - * 1. 过滤器执行顺序



请求进入网关会碰到三类过滤器：当前路由的过滤器、DefaultFilter、GlobalFilter
请求路由后，会将当前路由过滤器和DeFaultFilter，GlobalFilte，合并到一个过滤器链（集合）中，排序后一次执行每个过滤器

每一个多滤器都必须指定一个int类型的order值，**order值越小，优先级越高**，执行顺序越靠前

**GlobalFilter通过实现Ordered接口，或者添加@Order注解来指定order值，由自己指定**

**路由过滤器和defaultFilte的order的order由spring指定，默认是按照声明顺序从1递增**

**当过滤器的order值一样是，会按照defaultFilter > 路由过滤器 > GlobalFilter的顺序执行**

* + - * 1. 跨域问题处理

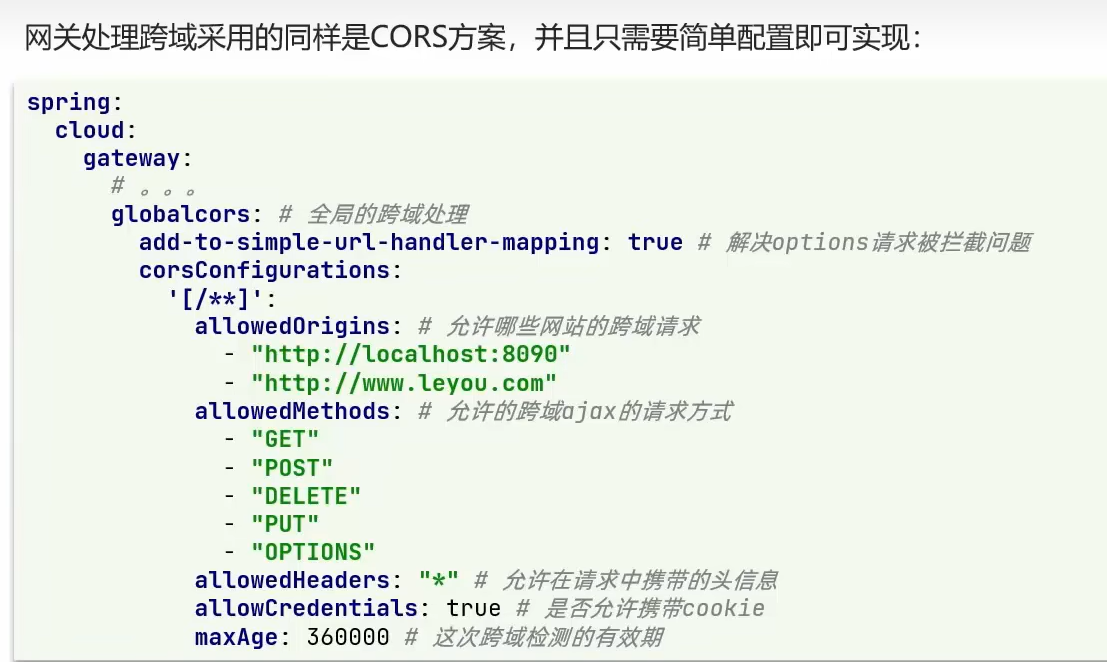
跨域：域名不一致就是跨域

域名不同

域名相同,端口不同

跨域问题：浏览器禁止请求的发起者与服务端发生跨域ajax请求，请求被浏览器拦截的问题

解决方案：CORS



* + 1. 配置中心Nacos
  1. Docker
     1. 认识Docker
        1. 项目部署问题

大型项目组件较多，运行环境复杂

* + - * 1. 依赖关系复杂，容易出现兼容性问题

Docker允许开发中将应用、依赖、库函数、配置一起打包，形成可移植镜像

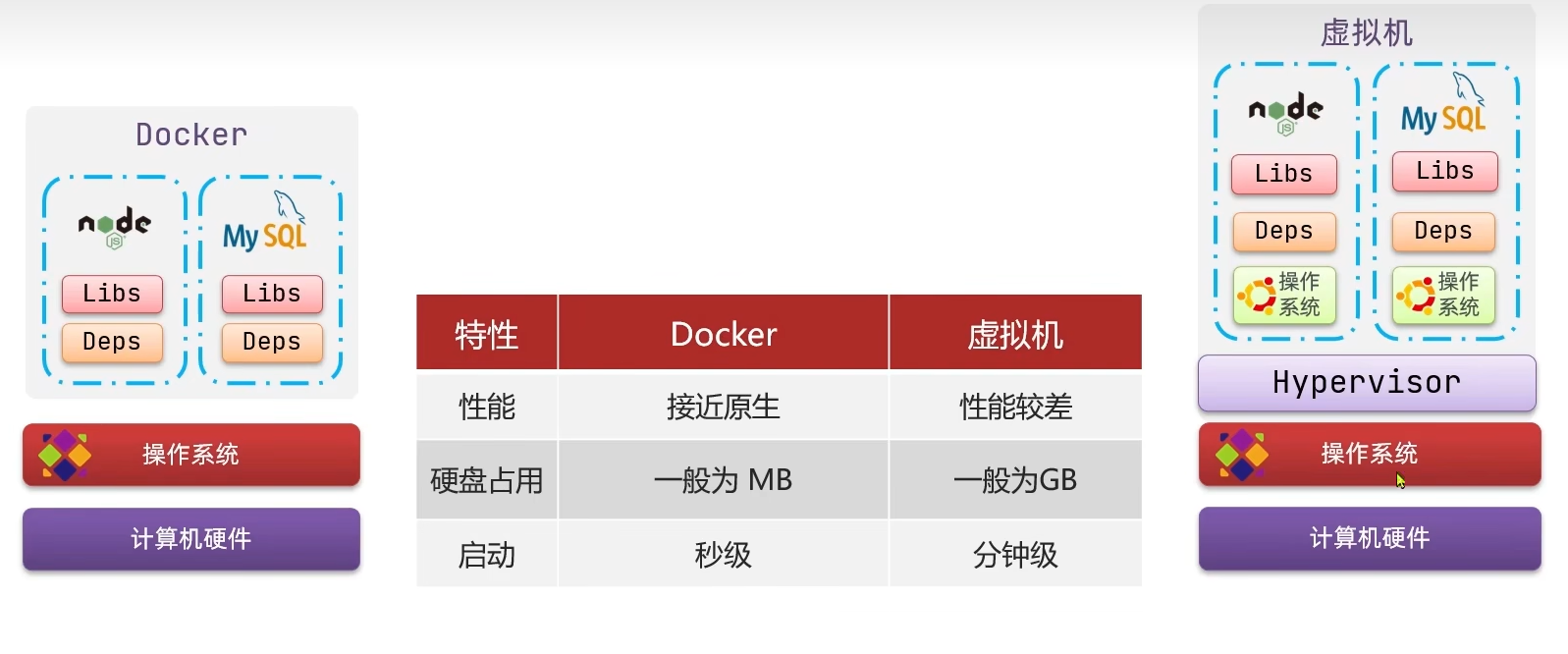
Docker应用运行在容器中，使用沙箱机制，相互隔离

启动，移除都可以通过一行命令完成，方便快捷

* + - * 1. 开发、测试、生产环境有差异

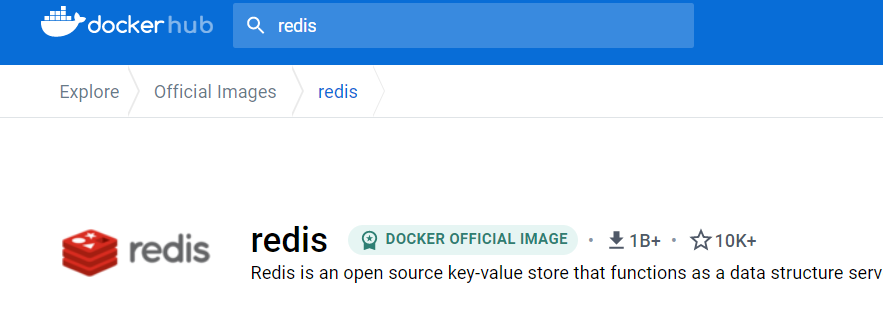
Dockers镜像中包含完整运行环境，包括系统函数库，仅依赖系统的Linux内核，因此可以在任意Linux操作系统上运行

* + - 1. Docker解决依赖兼容问题
         1. 将应用的Libs（函数库），Deps（依赖）、配置与应用一起打包
         2. 将每个应用放到一个兼容容器中运行，避免互相干扰
      2. Docker解决不同系统环境的问题
         1. Docker将用户程序与所需要调用的系统（比如Ubuntu）函数库一起打包
         2. Docker运行到不同操作系统时，直接基于打包的库函数，借助于操作系统的Linux内核来运行
      3. Docker和虚拟机
         1. Docker是一个系统进程；虚拟机是在操作系统中的操作系统



* + - 1. 镜像和容器
         1. 镜像：Docker将应用程序极其所需的依赖、函数库、环境、配置等文件打包在一起，称为镜像
         2. 容器：镜像中的应用程序运行后形成的进程就是容器，只是Docker会给容器做隔离，对外不可见
      2. DockerHub
         1. DockerHub是一个Docker镜像的托管平台。这样的平台称为Docker Registry
         2. 国内：阿里云镜像等
      3. Docker是一个Cs架构程序
         1. 服务端：Docker守护进程负责处理Docker指令、管理镜像，容器等
         2. 客户端：通过命令或RestAPI向Docker服务端发送指令，可以在本地或远程像服务端发送指令
    1. Dockers原理
    2. Docker使用
       1. 镜像操作命令
          1. 使用帮助文档操作命令（centos）
          2. docker build构建镜像
          3. docker pull从服务拉取镜像
          4. docker images查看镜像
          5. docker rmi删除镜像
          6. docker save 保存镜像为一个压缩包
          7. docker load加载压缩包为镜像
          8. 练习

到DockerHub搜索Redis镜像



查看Redis镜像的名称和版本

利用docker pull拉取镜像

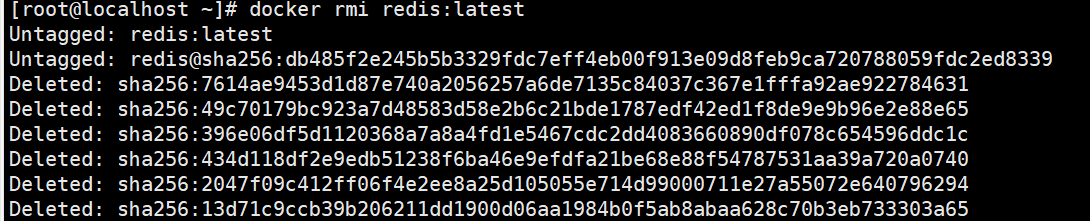


利用docker save命令将redis:latest打包为一个redis.tar包





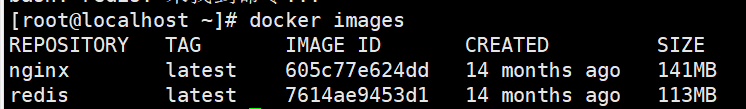
利用docker rmi 删除本地的redis:latest



利用docker load重新加载redis.tar文件



查看镜像

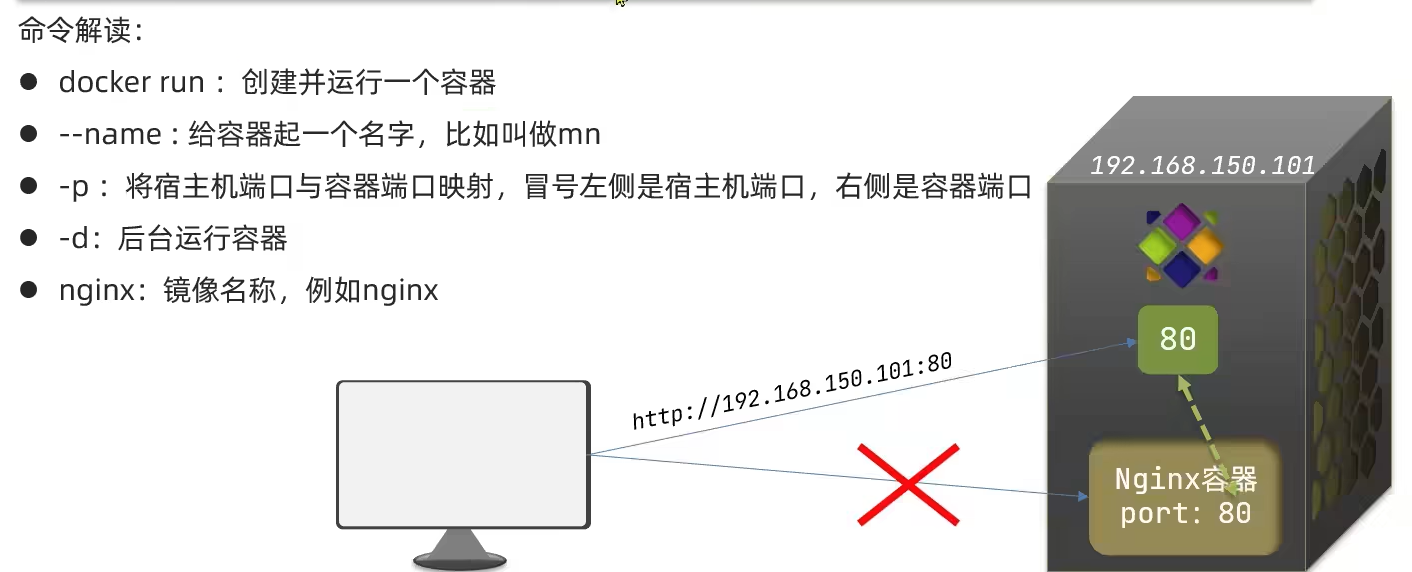


* + - 1. 容器相关命令
         1. docker run创建容器并运行
         2. docker pause容器从运行到暂停
         3. docker unpause容器从暂停到运行
         4. docker stop容器从运行到停止
         5. docker start容器从停止到运行
         6. docker ps查看所有运行的容器的状态
         7. docker logs查看容器运行日志
         8. docker exec进入容器执行命令
         9. docker rm 删除指定容器（彻底删除）
         10. 练习

创建运行一个Nginx容器

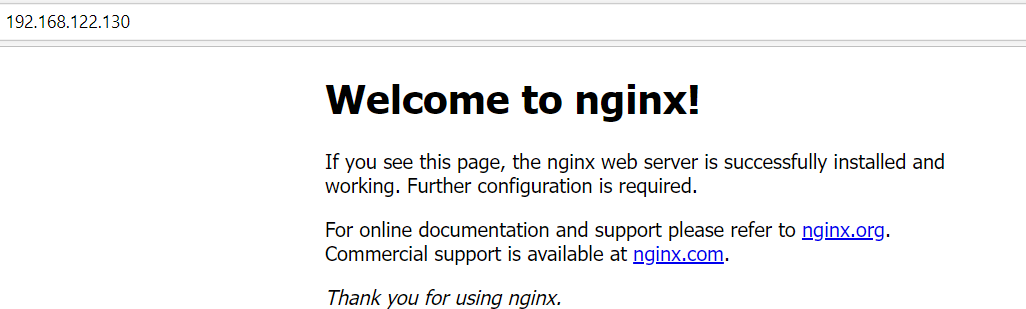
到dockerhub查看nginx的容器运行命令





外界不能访问容器，容器对外隔离。通过暴露宿主机端口进而访问容器，宿主机端口可以不和容器端口相同，-d：daemon

访问nginx 地址为虚拟机地址



查看日志（持续查看日志）



进入Nginx容器，修改HTML文件内容（不推荐）



停止容器



查看容器（默认查看运行中程序，-a查看全部）



删除容器



* + - * 1. Docker数据卷命令

数据卷（Volume）：一个虚拟目录，指向宿主机文件系统中的某个目录

作用：将容器与数据分离，解耦合，方便操作容器内数据，保证数据安全

基本语法：docker volume [COMMAND] ==> docker volume命令是数据卷操作，command是确定下一步操作

create 创建一个volume

inspect 显示一个或多个volume的信息

ls 列出所有的volume

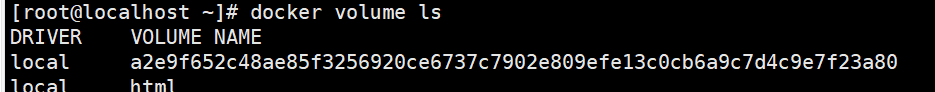
rm 删除一个或多个指定的volume

练习

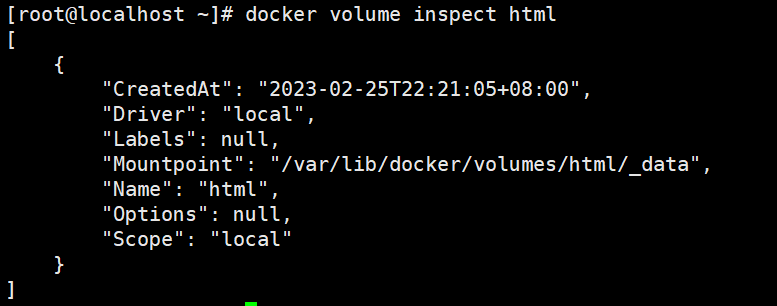
创建一个数据卷



列出所有的volume



查看详细信息



* + - * 1. Docker数据卷挂载案例

挂载数据卷

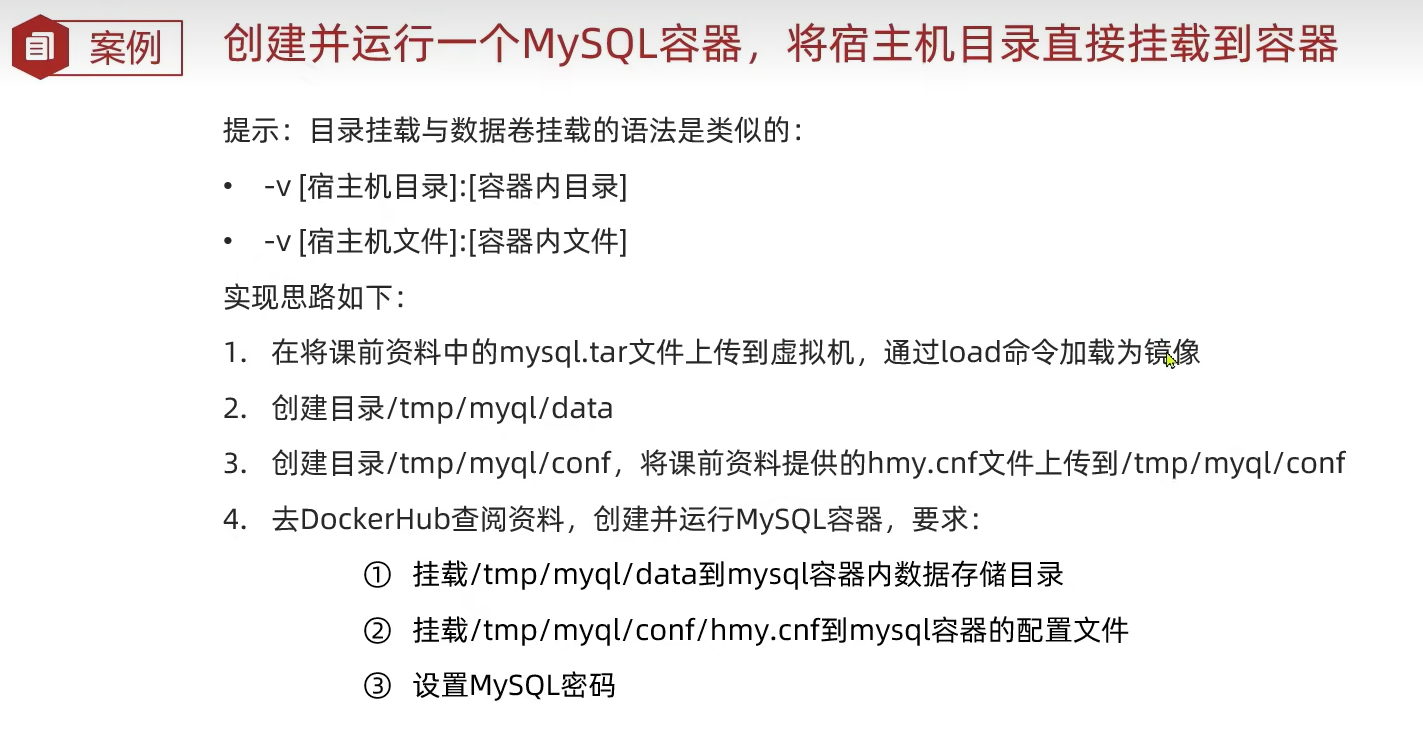




挂载的目录在dockerHub上有介绍



* + - * 1. 案例



* + 1. DockerFile自定义镜像
       1. 镜像结构
       2. Dockerfile语法
       3. 构建Java项目
    2. DockerComPose
  1. 异步通信
     1. 同步和异步
     2. MQ技术选型
     3. SpringAMQP
     4. 消费者限流
  2. 分布式搜索
     1. DSL语法
     2. HighLevelClient
     3. 拼音搜索
     4. 自动补全

1. **第二部分**
   1. 微服务保护
      1. 流量控制
      2. 系统保护
      3. 熔断降级
      4. 服务授权
   2. 分布式事务
      1. XA模式
      2. TCC模式
      3. AT模式
      4. Saga模式
   3. 分布式缓存
      1. 数据持久化
      2. Redis主从集群
      3. 哨兵机制
      4. Redis分片集群
   4. 多级缓存
      1. 多级缓存分层
      2. Nginx缓存
      3. Redis缓存
      4. Canal数据同步
   5. 可靠消息服务
      1. 消息三方确认
      2. 惰性队列
      3. 延迟队列
      4. 镜像集群
      5. 仲裁队列
2. 第三部分
   1. Nacos源码
   2. Sentinel源码
   3. Redis热点问题