一、搭建apollo环境

86：搭建portal

87：搭建config和admin 搭建两套dev跟pro

Apollo的配置会在本地有一份缓存，即使apollo挂了也一样高可用，但不能保证一致性

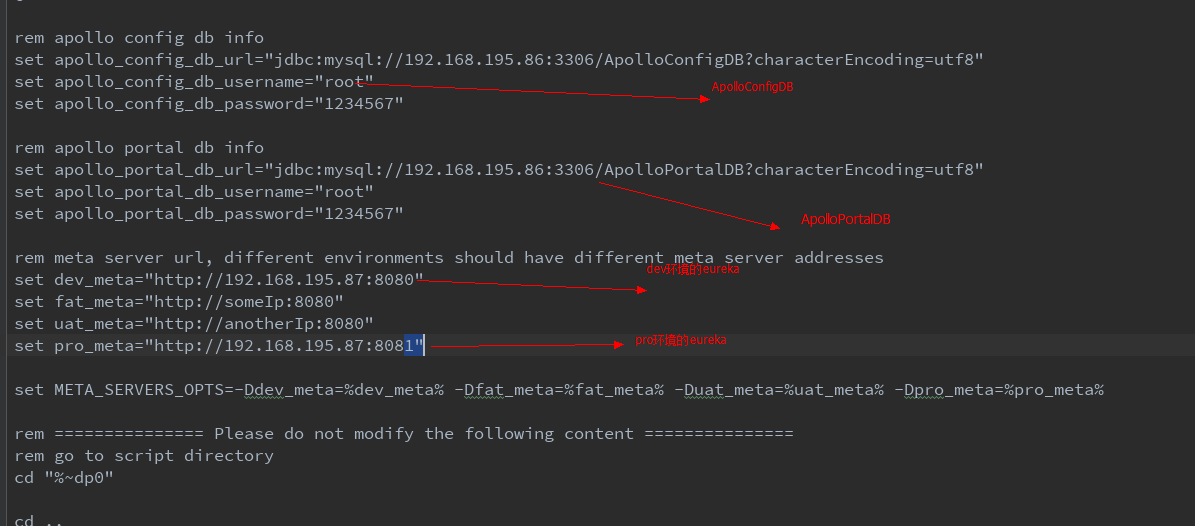
1. 下载apollo
2. 安装数据库

portaldb

configdb

脚本位置：apollo-master\scripts\db

3.构建配置文件：



1. 打包

执行build.bat

1. 上传zip到服务器

三个zip文件 portal，adminservice，configservice

1. 服务器解压

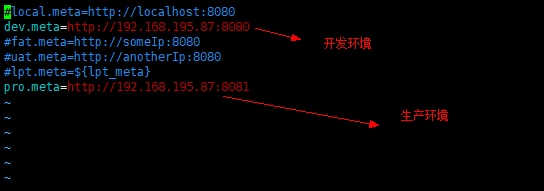
unzip apollo-portal-1.6.0-SNAPSHOT-github.zip

unzip apollo-adminservice-1.6.0-SNAPSHOT-github.zip -d adminservice

unzip apollo-configservice-1.6.0-SNAPSHOT-github.zip -d configservice

1. 修改portal的环境配置文件

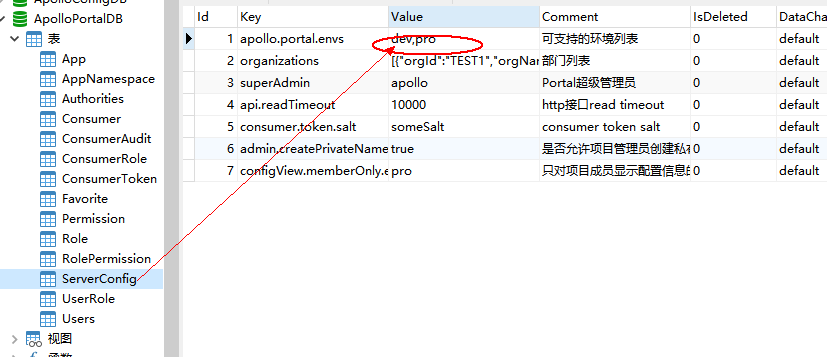
vim apollo-env.properties



 Meta Server只是一个逻辑角色，在部署时和Config Service是在一个JVM进程中的

为了简单起见，目前Eureka在部署时和Config Service是在一个JVM进程中的

因为是生存跟测试两环境，需在portal添加



1. 查看数据库连接配置

application-github.properties

1. 启动apollo

顺序:先启动 configservice,adminservice,portal

执行scripts目录下的startup.sh脚本

1. apollo日志是在opt目录下
2. portal

端口：8070

账号：apollo

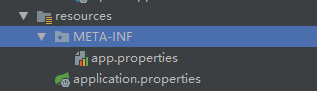
密码：admin

cnfigservice端口：8080

adminservice端口:8090

二、springboot整合apollo

1.需建META-INF目录



新建app.properties

app.id=pasda

pom文件添加依赖

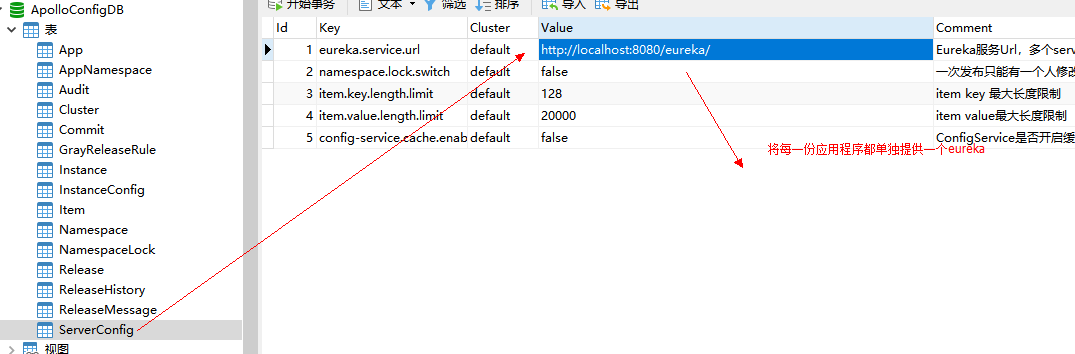


获取bean注入的值时，必须要使用配置刷新即：@RefreshScope注解

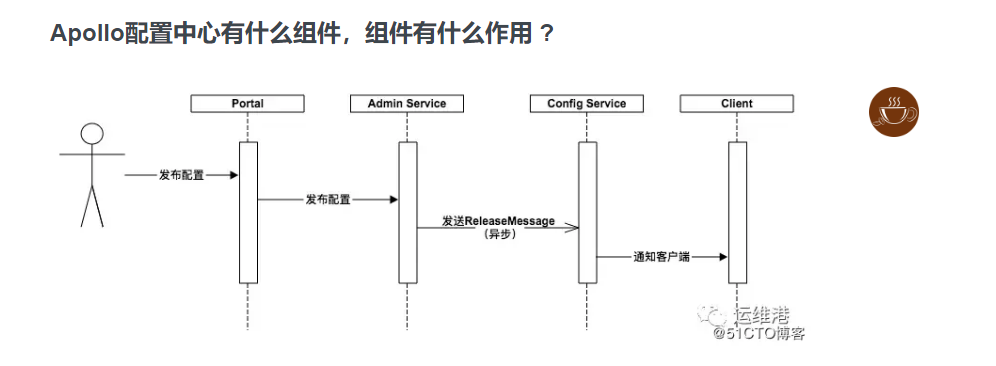
灰度发布：程序会先获取灰度的配置，当灰度不存在时才获取主配置

三、高可用

将多个 adminservice和configservice都指向同一个数据库，并将eureka的配置设置成多个



Apollo总结：<https://blog.csdn.net/qq_33320785/article/details/80772810>



**Portal**

1. 提供Web界面供用户管理配置
2. 通过Meta Server获取Admin Service服务列表（IP+Port），通过IP+Port访问服务
3. 在Portal侧做load balance、错误重试

**Admin Service**

1. 提供配置管理接口
2. 提供配置修改、发布等接口
3. 接口服务对象为Portal

**Config Service**

1. Config Service 包中包含了三个组件: config ,meta server,euraka  
   a. Config 组件  
   A. 提供配置获取接口  
   B. 提供配置更新推送接口（基于Http long polling）  
   i. 服务端使用Spring DeferredResult实现异步化，从而大大增加长连接数量  
   ii. 目前使用的tomcat embed默认配置是最多10000个连接（可以调整），使用了4C8G的虚拟机实测可以支撑10000个连接，所以满足需求（一个应用实例只会发起一个长连接）。  
   C. 接口服务对象为Apollo客户端

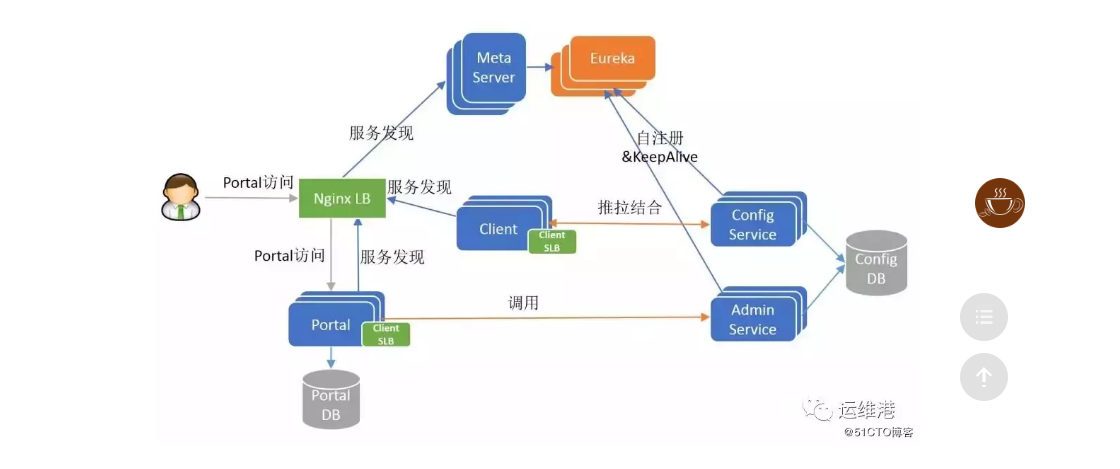
b. Meta Server组件   
A. Portal通过域名访问Meta Server获取Admin Service服务列表（IP+Port）  
B. Client通过域名访问Meta Server获取Config Service服务列表（IP+Port）  
C. Meta Server从Eureka获取Config Service和Admin Service的服务信息，相当于是一个Eureka Client  
D. 增设一个Meta Server的角色主要是为了封装服务发现的细节，对Portal和Client而言，永远通过一个Http接口获取Admin Service和Config Service的服务信息，而不需要关心背后实际的服务注册和发现组件  
E. Meta Server只是一个逻辑角色，在部署时和Config Service是在一个JVM进程中的

c. Eureka组件  
A. 基于Eureka和Spring Cloud Netflix提供服务注册和发现  
B. Config Service和Admin Service会向Eureka注册服务，并保持心跳  
C. 为了简单起见，目前Eureka在部署时和Config Service是在一个JVM进程中的（通过Spring Cloud Netflix）

**Client**

1. Apollo提供的客户端程序，为应用提供配置获取、实时更新等功能
2. 通过Meta Server获取Config Service服务列表（IP+Port），通过IP+Port访问服务
3. 在Client侧做load balance、错误重试

如果从用户访问数据流的关系图，大概类似如下的结构:



基本机制流程如下：

1. Config Service启动时候启动Config server，Meta server，Eureka三个服务 ，同时Config Server把自身注册到Eureka上面，并保持心跳。
2. Admin Server 启动时，把自身注册到Eureka 上面，并保持心跳。
3. Config Server，Admin Server都通过Meta Server连接Eureka，所以Meta Server相当于一个Eureka的客户端，提供功能接口。
4. Portal 启动时候，通过Meta Server获取Admin Service 的列表（IP:PORT）,同时通过软件实现LB，错误重试
5. Client 客户端添加到开发项目中，项目启动后，客户端会通过Meta Server获取Config service的列表，并在客户端内部实现了LB,错误重试