Apusic 金蝶天燕

生成时间：2026-07-13

安装手册

金蝶Apusic分布式配置中心 for Apollo V1.0

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 前言

本文档为金蝶Apusic分布式配置中心for Apollo（Apusic Distributed Config Center for Apollo，简称ADCC for Apollo）V1.0产品的安装手册，介绍了如何按照分布式部署的方式部署ADCC for Apollo配置中心，从而可以在开发、测试、生产等环境分别部署运行。

1.1 适用对象

本文档适用于IT信息化业务负责人、研发经理、软件项目经理、软件架构师、运维工程师。

1.2 相关文档

了解更多ADCC for Apollo v1.0产品相关的信息，请参阅以下ADCC for Apollo V1.0产品手册文档集：

序号	手册文档	说明
1	金蝶Apusic分布式配置中心for Apollo V1.0 安装手册	详细介绍如何在各操作系统上安装ADCC for Apollo，以及服务组件实例启停操作，产品的注册过程。
2	金蝶Apusic分布式配置中心for Apollo V1.0 用户手册	详细介绍 ADCC 相关功能的使用、配置、管理及配套工具的使用方法。
3	金蝶Apusic分布式配置中心for Apollo V1.0 开发手册	详细介绍基于各种开发语言进行ADCC for Apollo客户端应用开发的说明。
4	金蝶Apusic分布式配置中心for Apollo V1.0 API开放平台手册	详细介绍ADCC for Apollo的标准API。
5	金蝶Apusic分布式配置中心for Apollo V1.0 常见问题	详细介ADCC for Apollo的常见问题及处理方法。

1.3 技术支持

ADCC for Apollo产品提供全面的技术支持服务，您可以通过以下方式获得技术支持：

- 网址：www.apusic.com
- 电话：400-855-5800
- 邮箱：support@apusic.com

- 金蝶云社区：https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN

您在取得技术支持时，请提供如下信息：

- 您的姓名
- 公司信息与联系方式
- 操作系统及其版本
- 产品版本号
- 出现异常及错误的日志、截图等详细信息

2 准备工作

2.1 运行时环境

2.1.1 操作系统

服务端基于Spring Boot，启动脚本理论上支持所有Unix/Linux发行版以及国产服务器操作系统。

2.1.2 Java

- 服务端：1.8+
- 客户端：1.8+

在配置好后，可以通过如下命令检查：

```
java -version
```

样例输出：

```
java version "1.8.0_74"
Java(TM) SE Runtime Environment (build 1.8.0_74-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.74-b02, mixed mode)
```

2.2 MySQL

- 版本要求：5.6.5+

Apollo的表结构对 `timestamp` 使用了多个default声明，所以需要5.6.5以上版本。

连接上MySQL后，可以通过如下命令检查：

```
SHOW VARIABLES WHERE Variable_name = 'version';
```

Variable_name	Value
version	5.7.11

> *产品支持国产人大金仓、达梦等数据库。本手册以MySQL安装为例进行介绍。*

2.3 环境

分布式部署需要事先确定部署的环境以及部署方式。

Apollo目前支持以下环境：

- DEV
  - 开发环境
- FAT
  - 测试环境，相当于alpha环境(功能测试)
- UAT
  - 集成环境，相当于beta环境（回归测试）
- PRO
  - 生产环境

2.4 网络策略

分布式部署的时候，`apollo-configservice`和`apollo-adminservice`需要把自己的IP和端口注册到Meta Server（apollo-configservice本身）。

Apollo客户端和Portal会从Meta Server获取服务的地址（IP+端口），然后通过服务地址直接访问。

需要注意的是，`apollo-configservice`和`apollo-adminservice`是基于内网可信网络设计的，所以出于安全考虑，**请不要将`apollo-configservice`和`apollo-adminservice`直接暴露在公网。**

所以如果实际部署的机器有多块网卡（如docker），或者存在某些网卡的IP是Apollo客户端和Portal无法访问的（如网络安全限制），那么我们就需要在`apollo-configservice`和`apollo-adminservice`中做相关配置来解决连通性问题。

2.4.1 忽略某些网卡

可以分别修改`apollo-configservice`和`apollo-adminservice`的startup.sh，通过JVM System Property传入-D参数，也可以通过OS Environment Variable传入，下面的例子会把`docker0`和`veth`开头的网卡在注册到Eureka时忽略掉。

JVM System Property示例：

```
-Dspring.cloud.inetutils.ignoredInterfaces[0]=docker0
-Dspring.cloud.inetutils.ignoredInterfaces[1]=veth.*
```

OS Environment Variable示例：

```
SPRING_CLOUD_INETUTILS_IGNORED_INTERFACES[0]=docker0
SPRING_CLOUD_INETUTILS_IGNORED_INTERFACES[1]=veth.*
```

2.4.2 指定要注册的IP

可以分别修改 `apollo-configservice` 和 `apollo-adminservice` 的startup.sh，通过JVM System Property传入-D参数，也可以通过OS Environment Variable传入，下面的例子会指定注册的IP为 `1.2.3.4` 。

JVM System Property示例：

```
-Deureka.instance.ip-address=1.2.3.4
```

OS Environment Variable示例：

```
EUREKA_INSTANCE_IP_ADDRESS=1.2.3.4
```

2.4.3 指定要注册的URL

可以分别修改 `apollo-configservice` 和 `apollo-adminservice` 的startup.sh，通过JVM System Property传入-D参数，也可以通过OS Environment Variable传入，下面的例子会指定注册的URL为 `http://1.2.3.4:8080` 。

JVM System Property示例：

```
-Deureka.instance.homePageUrl=http://1.2.3.4:8080
-Deureka.instance.preferIpAddress=false
```

OS Environment Variable示例：

```
EUREKA_INSTANCE_HOME_PAGE_URL=http://1.2.3.4:8080
EUREKA_INSTANCE_PREFER_IP_ADDRESS=false
```

2.4.4 打通网络

在一些公司（例如金融行业的公司），存在很多防火墙和网络隔离，需要打通网络（让ip1可以访问ip2的某个端口）

2.4.4.1 打通客户端到配置中心的网络

对于使用配置中心的客户端，

client需要访问所有（或者相同机房内的）Meta Server和Config Service（默认都是8080端口），请不要打通Client到Admin Service的网络

```
flowchart LR
    subgraph servers[IP1:8080, IP2:8080, ..., IPn:8080]
        m[Meta Sever]
        c[Config Service]
    end
    client --> servers
```

如果某个应用需要使用openapi，还需要访问Portal（默认是8070端口）

```
flowchart LR
    subgraph servers[IP:8070]
        Portal
    end
    openapi-client --> servers
```

2.4.4.2 打通配置中心内部的网络

对于配置中心自己内部，由于各个服务之间需要互相访问，所以也要保证网络上的连通

```
flowchart LR
    subgraph config-service-servers[All Config Service's IP:8080]
        m[Meta Server]
        c[Config Service]
    end
    subgraph admin-service-servers[All Admin Service's IP:8090]
        a[Admin Service]
    end

    subgraph portal-servers[IP:8070]
        p[Portal]
    end
```

```
configdb[(ConfigDB)]
portaldb[(PortalDB)]

a --> config-service-servers

a --> configdb
c --> configdb

p --> config-service-servers
p --> admin-service-servers
p --> portaldb
```

3 部署步骤

3.1 创建数据库

Apollo服务端共需要两个数据库：`ApolloPortalDB`和`ApolloConfigDB`，我们把数据库、表的创建和样例数据都分别准备了sql文件，只需要导入数据库即可。

需要注意的是ApolloPortalDB只需要在生产环境部署一个即可，而ApolloConfigDB需要在每个环境部署一套，如fat、uat和pro分别部署3套ApolloConfigDB。

> *注意：如果你本地已经创建过Apollo数据库，请注意备份数据。我们准备的sql文件会清空Apollo相关的表。*

3.1.1 创建ApolloPortalDB

可以根据实际情况选择通过手动导入SQL或是通过Flyway自动导入SQL创建。

3.1.1.1 手动导入SQL创建

通过各种MySQL客户端导入apolloportaldb.sql即可。

以MySQL原生客户端为例：

```
source /your_local_path/scripts/sql/apolloportaldb.sql
```

3.1.1.2 验证

导入成功后，可以通过执行以下sql语句来验证：

```
select `Id`, `Key`, `Value`, `Comment` from
`ApolloPortalDB`.`ServerConfig` limit 1;
```

Id	Key	Value	Comment
1	apollo.portal.envs	dev	可支持的环境列表

> *注：ApolloPortalDB只需要在生产环境部署一个即可*

3.1.2 创建ApolloConfigDB

可以根据实际情况选择通过手动导入SQL或是通过Flyway自动导入SQL创建。

3.1.2.1 手动导入SQL

通过各种MySQL客户端导入apolloconfigdb.sql即可。

以MySQL原生客户端为例：

```
source /your_local_path/scripts/sql/apolloconfigdb.sql
```

3.1.2.2 验证

导入成功后，可以通过执行以下sql语句来验证：

```
select `Id`, `Key`, `Value`, `Comment` from
`ApolloConfigDB`.`ServerConfig` limit 1;
```

Id	Key	Value	Comment
1	eureka.service.url	http://127.0.0.1:8080/eureka/	Eureka服务Url

> *注：ApolloConfigDB需要在每个环境部署一套，如fat、uat和pro分别部署3套ApolloConfigDB*

3.1.2.3 从别的环境导入ApolloConfigDB的项目数据

如果是全新部署的Apollo配置中心，请忽略此步。

如果不是全新部署的Apollo配置中心，比如已经使用了一段时间，这时在Apollo配置中心已经创建了不少项目以及namespace等，那么在新环境中的ApolloConfigDB中需要从其它正常运行的环境中导入必要的项目数据。

主要涉及ApolloConfigDB的下面4张表，下面同时附上需要导入的数据查询语句：

1. App
   - 导入全部的App
   - 如：insert into **新环境的**`ApolloConfigDB`.`App` select * from **其它环境的** `ApolloConfigDB`.`App` where `IsDeleted` = 0;
2. AppNamespace
   - 导入全部的AppNamespace
   - 如：insert into **新环境的**`ApolloConfigDB`.`AppNamespace` select * from **其它环境的** `ApolloConfigDB`.`AppNamespace` where `IsDeleted` = 0;
3. Cluster
   - 导入默认的default集群
   - 如：insert into **新环境的**`ApolloConfigDB`.`Cluster` select * from **其它环境的** `ApolloConfigDB`.`Cluster` where `Name` = 'default' and `IsDeleted` = 0;

4. Namespace
    - 导入默认的default集群中的namespace
    - 如：insert into **新环境的**`ApolloConfigDB`.`Namespace` select * from **其它环境的**`ApolloConfigDB`.`Namespace` where `ClusterName` = 'default' and `IsDeleted` = 0;

同时也别忘了通知用户在新的环境给自己的项目设置正确的配置信息，尤其是一些影响面比较大的公共namespace配置。

> *如果是为正在运行的环境迁移数据，建议迁移完重启一下config service，因为config service中有appnamespace的缓存数据*

3.1.3 调整服务端配置

ADCC for Apollo自身的一些配置是放在数据库里面的，所以需要针对实际情况做一些调整，具体参数说明请参考服务端配置说明。

大部分配置可以先使用默认值，不过 apollo.portal.envs 和 eureka.service.url 请务必配置正确后再进行下面的部署步骤。

3.2 虚拟机/物理机部署

3.2.1 获取安装包

从知识中心下载最新版本的`adcc-configservice-x.x.x.tar.gz`、`adcc-adminservice-x.x.x.tag.gz`和`adcc-portal-x.x.x.tar.gz`产品包解压即可。

3.2.2 配置数据库连接信息

Apollo服务端需要知道如何连接到你前面创建的数据库，数据库连接串信息位于上一步下载的压缩包中的`config/application-github.properties`中。

3.2.2.1 配置apollo-configservice的数据库连接

1. 解压`adcc-configservice-x.x.x.tar.gz`
2. 用编辑器（如vim，notepad++，sublime等）打开`config`目录下的`application-github.properties`文件
3. 填写正确的ApolloConfigDB数据库连接串信息，注意用户名和密码后面不要有空格!
4. 修改完的效果如下：

```
# DataSource
spring.datasource.url =
jdbc:mysql://localhost:3306/ApolloConfigDB?
useSSL=false&characterEncoding=utf8
spring.datasource.username = someuser
spring.datasource.password = somepwd
```

> *注：由于ApolloConfigDB在每个环境都有部署，所以对不同的环境config-service需要配置对应环境的数据库参数*

3.2.2.2 配置apollo-adminservice的数据库连接

1. 解压 `apollo-adminservice-x.x.x-github.zip`
2. 用程序员专用编辑器（如vim，notepad++，sublime等）打开 `config` 目录下的 `application-github.properties` 文件
3. 填写正确的ApolloConfigDB数据库连接串信息，注意用户名和密码后面不要有空格!
4. 修改完的效果如下：

```
# DataSource
spring.datasource.url =
jdbc:mysql://localhost:3306/ApolloConfigDB?
useSSL=false&characterEncoding=utf8
spring.datasource.username = someuser
spring.datasource.password = somepwd
```

> *注：由于ApolloConfigDB在每个环境都有部署，所以对不同的环境admin-service需要配置对应环境的数据库参数*

3.2.2.3 配置apollo-portal的数据库连接

1. 解压 `apollo-portal-x.x.x-github.zip`
2. 用程序员专用编辑器（如vim，notepad++，sublime等）打开 `config` 目录下的 `application-github.properties` 文件
3. 填写正确的ApolloPortalDB数据库连接串信息，注意用户名和密码后面不要有空格!
4. 修改完的效果如下：

```
# DataSource
spring.datasource.url =
jdbc:mysql://localhost:3306/ApolloPortalDB?
useSSL=false&characterEncoding=utf8
spring.datasource.username = someuser
spring.datasource.password = somepwd
```

3.2.2.4 配置apollo-portal的meta service信息

Apollo Portal需要在不同的环境访问不同的meta service(apollo-configservice)地址，所以我们需要在配置中提供这些信息。默认情况下，meta service和config service是部署在同一个JVM进程，所以meta service的地址就是config service的地址。

> *可以通过ApolloPortalDB.ServerConfig中的配置项来配置Meta Service地址，详见apollo.portal.meta.servers - 各环境Meta Service列表*

使用编辑器（如vim，notepad++，sublime等）打开 `adcc-portal-x.x.x.zip` 中 `config` 目录下的 `apollo-env.properties` 文件。

假设DEV的apollo-configservice未绑定域名，地址是1.1.1.1:8080，FAT的apollo-configservice绑定了域名apollo.fat.xxx.com，UAT的apollo-configservice绑定了域名apollo.uat.xxx.com，PRO的apollo-configservice绑定了域名apollo.xxx.com，那么可以如下修改各环境meta service服务地址，格式为 `${env}.meta=http://${config-service-url:port}` ，如果某个环境不需要，也可以直接删除对应的配置项（如lpt.meta）：

```
dev.meta=http://1.1.1.1:8080
fat.meta=http://apollo.fat.xxx.com
uat.meta=http://apollo.uat.xxx.com
pro.meta=http://apollo.xxx.com
```

除了通过 `apollo-env.properties` 方式配置meta service以外，apollo也支持在运行时指定meta service（优先级比 `apollo-env.properties` 高）：

1. 通过Java System Property `${env}_meta`
    - 可以通过Java的System Property `${env}_meta` 来指定
    - 如 `java -Ddev_meta=http://config-service-url -jar xxx.jar`
    - 也可以通过程序指定，如 `System.setProperty("dev_meta", "http://config-service-url");`

2. 通过操作系统的System Environment `${ENV}_META`
    - 如 `DEV_META=http://config-service-url`
    - 注意key为全大写，且中间是 `_` 分隔

> *注1: 为了实现meta service的高可用，推荐通过SLB（Software Load Balancer）做动态负载均衡*

> *注2: meta service地址也可以填入IP，0.11.0版本之前只支持填入一个IP。从0.11.0版本开始支持填入以逗号分隔的多个地址（PR #1214），如 `http://1.1.1.1:8080,http://2.2.2.2:8080`，不过生产环境还是建议使用域名（走slb），因为机器扩容、缩容等都可能导致IP列表的变化。*

3.2.3 部署ADCC for Apollo服务端

3.2.3.1 部署apollo-configservice

将对应环境的 `apollo-configservice-x.x.x-github.zip` 上传到服务器上，解压后执行scripts/startup.sh即可。如需停止服务，执行scripts/shutdown.sh.

记得在scripts/startup.sh中按照实际的环境设置一个JVM内存，以下是我们的默认设置，供参考：

```
export JAVA_OPTS="-server -Xms6144m -Xmx6144m -Xss256k -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=384m -XX:NewSize=4096m -
XX:MaxNewSize=4096m -XX:SurvivorRatio=18"
```

> *注1：如果需要修改JVM参数，可以修改scripts/startup.sh的 `JAVA_OPTS` 部分。 注2：如要调整服务的日志输出路径，可以修改scripts/startup.sh和apollo-configservice.conf中的 `LOG_DIR` 。 注3：如要调整服务的监听端口，可以修改scripts/startup.sh中的 `SERVER_PORT` 。另外apollo-configservice同时承担meta server职责，如果要修改端口，注意要同时ApolloConfigDB.ServerConfig表中的 `eureka.service.url` 配置项以及apollo-portal和apollo-client中的使用到的meta server信息，详见：2.2.1.1.2.4 配置apollo-portal的meta service信息和1.2.2 Apollo Meta Server。*

> *注4：如果ApolloConfigDB.ServerConfig的eureka.service.url只配了当前正在启动的机器的话，在启动apollo-configservice的过程中会在日志中输出eureka注册失败的信息，如 `com.sun.jersey.api.client.ClientHandlerException: java.net.ConnectException: Connection refused` 。需要注意的是，这个是预期的情况，因为apollo-configservice需要向Meta Server（它自己）注册服务，但是因为在启动过程中，自己还没起来，所以会报这个错。后面会进行重试的动作，所以等自己服务起来后就会注册正常了。*

> *注5：如果你看到了这里，相信你一定是一个细心阅读文档的人，而且离成功就差一点点了，继续加油，应该很快就能完成Apollo的分布式部署了！不过你是否有感觉Apollo的分布式部署步骤有点繁琐？是否有啥建议想要和作者说？如果答案是肯定的话，请移步 #1424，期待你的建议！*

3.2.3.2 部署apollo-adminservice

将对应环境的`apollo-adminservice-x.x.x-github.zip`上传到服务器上，解压后执行scripts/startup.sh即可。如需停止服务，执行scripts/shutdown.sh.

记得在scripts/startup.sh中按照实际的环境设置一个JVM内存，以下是我们的默认设置，供参考：

```
export JAVA_OPTS="-server -Xms2560m -Xmx2560m -Xss256k -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=384m -XX:NewSize=1024m -
XX:MaxNewSize=1024m -XX:SurvivorRatio=22"
```

> *注1：如果需要修改JVM参数，可以修改scripts/startup.sh的`JAVA_OPTS`部分。*

> *注2：如要调整服务的日志输出路径，可以修改scripts/startup.sh和apollo-adminservice.conf中的`LOG_DIR`。*

> *注3：如要调整服务的监听端口，可以修改scripts/startup.sh中的`SERVER_PORT`。*

3.2.3.3 部署apollo-portal

将`apollo-portal-x.x.x-github.zip`上传到服务器上，解压后执行scripts/startup.sh即可。如需停止服务，执行scripts/shutdown.sh.

记得在startup.sh中按照实际的环境设置一个JVM内存，以下是我们的默认设置，供参考：

```
export JAVA_OPTS="-server -Xms4096m -Xmx4096m -Xss256k -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=384m -XX:NewSize=1536m -
XX:MaxNewSize=1536m -XX:SurvivorRatio=22"
```

> *注1：如果需要修改JVM参数，可以修改scripts/startup.sh的`JAVA_OPTS`部分。*

> *注2：如要调整服务的日志输出路径，可以修改scripts/startup.sh和apollo-portal.conf中的`LOG_DIR`。*

> *注3：如要调整服务的监听端口，可以修改scripts/startup.sh中的`SERVER_PORT`。*

3.2.4 使用其它服务注册中心替换内置eureka

3.2.4.1 nacos-discovery

启用外部nacos服务注册中心替换内置eureka

> *注意：需要重新打包*

1. 修改build.sh/build.bat，将config-service和admin-service的maven编译命令更改为

```
mvn clean package -Pgithub,nacos-discovery -DskipTests -pl
apollo-configservice,apollo-adminservice -am -
Dapollo_profile=github,nacos-discovery -
Dspring_datasource_url=$apollo_config_db_url -
Dspring_datasource_username=$apollo_config_db_username -
Dspring_datasource_password=$apollo_config_db_password
```

2. 分别修改apollo-configservice和apollo-adminservice安装包中config目录下的application-github.properties，配置nacos服务器地址

```
nacos.discovery.server-addr=127.0.0.1:8848
# 更多 nacos 配置
nacos.discovery.access-key=
nacos.discovery.username=
nacos.discovery.password=
nacos.discovery.secret-key=
nacos.discovery.namespace=
nacos.discovery.context-path=
```

3.2.4.2 consul-discovery

启用外部Consul服务注册中心替换内置eureka

1. 修改 `adcc-configservice-x.x.x.tar.gz` 和 `adcc-adminservice-x.x.x.tar.gz` 解压后的 `config/application.properties`，取消注释，把

```
#spring.profiles.active=github,consul-discovery
```

变成

```
spring.profiles.active=github,consul-discovery
```

2. 分别修改apollo-configservice和apollo-adminservice安装包中config目录下的application-github.properties，配置consul服务器地址

```
spring.cloud.consul.host=127.0.0.1
spring.cloud.consul.port=8500
```

3.2.4.3 zookeeper-discovery

启用外部Zookeeper服务注册中心替换内置eureka。

1. 修改 `adcc-configservice-x.x.x.tar.gz` 和 `adcc-adminservice-x.x.x.tar.gz` 解压后的 `config/application.properties`，取消注释，把

   ```
   #spring.profiles.active=github,zookeeper-discovery
   ```

   变成

   ```
   spring.profiles.active=github,zookeeper-discovery
   ```

2. 分别修改apollo-configservice和apollo-adminservice安装包中config目录下的application-github.properties，配置zookeeper服务器地址

   ```
   spring.cloud.zookeeper.connect-string=127.0.0.1:2181
   ```

3. Zookeeper版本说明

- 支持Zookeeper3.5.x以上的版本;
- 如果apollo-configservice应用启动报端口占用,请检查Zookeeper的如下配置;

> *注：Zookeeper3.5.0新增了内置的AdminServer*

```
admin.enableServer
admin.serverPort
```

3.2.4.4 custom-defined-discovery

启用custom-defined-discovery替换内置eureka

1. 修改 `adcc-configservice-x.x.x.tar.gz` 和 `adcc-adminservice-x.x.x.tar.gz` 解压后的 `config/application.properties`，取消注释，把

```
#spring.profiles.active=github,custom-defined-discovery
```

变成

```
spring.profiles.active=github,custom-defined-discovery
```

2. 配置自定义的 config-service 与 admin-service 的访问地址有两种方式：一种在mysql数据库ApolloConfigDB，表ServerConfig当中写入两条数据。

```
INSERT INTO `ApolloConfigDB`.`ServerConfig` (`Key`, `Value`, `Comment`) VALUES ('apollo.config-service.url', 'http://apollo-config-service', 'ConfigService 访问地址');
INSERT INTO `ApolloConfigDB`.`ServerConfig` (`Key`, `Value`, `Comment`) VALUES ('apollo.admin-service.url', 'http://apollo-admin-service', 'AdminService 访问地址');
```

另外一种修改apollo-configservice安装包中config目录下的application-github.properties

```
apollo.config-service.url=http://apollo-config-service
apollo.admin-service.url=http://apollo-admin-service
```

3.2.4.5 database-discovery

启用database-discovery替换内置eureka

Apollo支持使用内部的数据库表作为注册中心，不依赖第三方的注册中心

1. 修改 `apollo-configservice-x.x.x-github.zip` 和 `apollo-adminservice-x.x.x-github.zip` 解压后的 `config/application.properties`，取消注释，把

```
#spring.profiles.active=github,database-discovery
```

变成

```
spring.profiles.active=github,database-discovery
```

2. 在多机房部署时， 如果你需要apollo客户端只读取同机房内的Config Service， 你可以在Config Service和Admin Service安装包中 `config/application-github.properties` 新增一条配置

```
apollo.service.registry.cluster=与apollo的Cluster同名
```

4 服务端配置说明

> *以下配置除了支持在数据库中配置以外，也支持通过-D参数、application.properties等配置，且-D参数、application.properties等优先级高于数据库中的配置*

4.1 调整ApolloPortalDB配置

配置项统一存储在ApolloPortalDB.ServerConfig表中，也可以通过 **管理员工具 - 系统参数** 页面进行配置，无特殊说明则修改完一分钟实时生效。

4.1.1 apollo.portal.envs - 可支持的环境列表

默认值是dev，如果portal需要管理多个环境的话，以逗号分隔即可（大小写不敏感），如：

```
DEV,FAT,UAT,PRO
```

修改完需要重启生效。

> *注1：一套Portal可以管理多个环境，但是每个环境都需要独立部署一套Config Service、Admin Service和ApolloConfigDB。*

> *注2：只在数据库添加环境是不起作用的，还需要为apollo-portal添加新增环境对应的meta server地址。*

4.1.2 apollo.portal.meta.servers - 各环境Meta Service列表

Apollo Portal需要在不同的环境访问不同的meta service(apollo-configservice)地址，所以我们需要在配置中提供这些信息。默认情况下，meta service和config service是部署在同一个JVM进程，所以meta service的地址就是config service的地址。

样例如下：

```
{
    "DEV":"http://1.1.1.1:8080",
    "FAT":"http://apollo.fat.xxx.com",
    "UAT":"http://apollo.uat.xxx.com",
    "PRO":"http://apollo.xxx.com"
}
```

修改完需要重启生效。

> *该配置优先级高于其它方式设置的Meta Service地址，更多信息可以参考2.2.1.1.2.4 配置apollo-portal的meta service信息。*

4.1.3 organizations - 部门列表

Portal中新建的App都需要选择部门，所以需要在这里配置可选的部门信息，样例如下：

```
[{"orgId":"TEST1","orgName":"样例部门1"},
{"orgId":"TEST2","orgName":"样例部门2"}]
```

4.1.4 superAdmin - Portal超级管理员

超级管理员拥有所有权限，需要谨慎设置。

如果没有接入自己公司的SSO系统的话，可以先暂时使用默认值apollo（默认用户）。等接入后，修改为实际使用的账号，多个账号以英文逗号分隔(,)。

4.1.5 consumer.token.salt - consumer token salt

如果会使用开放平台API的话，可以设置一个token salt。如果不使用，可以忽略。

4.1.6 wiki.address

portal上"帮助"链接的地址，默认是Apollo github的wiki首页，可自行设置。

4.1.7 admin.createPrivateNamespace.switch

是否允许项目管理员创建private namespace。设置为`true`允许创建，设置为`false`则项目管理员在页面上看不到创建private namespace的选项。

4.1.8 emergencyPublish.supported.envs

配置允许紧急发布的环境列表，多个env以英文逗号分隔。

当config service开启一次发布只能有一个人修改开关(`namespace.lock.switch`)后，一次配置发布只能是一个人修改，另一个发布。为了避免遇到紧急情况时（如非工作时间、节假日）无法发布配置，可以配置此项以允许某些环境可以操作紧急发布，即同一个人可以修改并发布配置。

4.1.9 configView.memberOnly.envs

只对项目成员显示配置信息的环境列表，多个env以英文逗号分隔。

对设定了只对项目成员显示配置信息的环境，只有该项目的管理员或拥有该namespace的编辑或发布权限的用户才能看到该私有namespace的配置信息和发布历史。公共namespace始终对所有用户可见。

4.1.10 role.create-application.enabled - 是否开启创建项目权限控制

默认为false，所有用户都可以创建项目

如果设置为true，那么只有超级管理员和拥有创建项目权限的帐号可以创建项目，超级管理员可以通过 **管理员工具 - 系统权限管理** 给用户分配创建项目权限

4.1.11 role.manage-app-master.enabled - 是否开启项目管理员分配权限控制

默认为 `false`，所有项目的管理员可以为项目添加/删除管理员

如果设置为true，那么只有超级管理员和拥有项目管理员分配权限的帐号可以为特定项目添加/删除管理员，超级管理员可以通过 **管理员工具 - 系统权限管理** 给用户分配特定项目的管理员分配权限

4.1.12 admin-service.access.tokens - 设置apollo-portal访问各环境apollo-adminservice所需的access token

如果对应环境的apollo-adminservice开启了访问控制，那么需要在此配置apollo-portal访问该环境apollo-adminservice所需的access token，否则会访问失败

格式为json，如下所示：

```
{
    "dev" : "098f6bcd4621d373cade4e832627b4f6",
    "pro" : "ad0234829205b9033196ba818f7a872b"
}
```

4.1.13 searchByItem.switch - 控制台搜索框是否支持按配置项搜索

默认为 true，可以方便的按配置项快速搜索配置。如果设置为 false，则关闭此功能。

4.2 调整ApolloConfigDB配置

配置项统一存储在ApolloConfigDB.ServerConfig表中，需要注意每个环境的ApolloConfigDB.ServerConfig都需要单独配置，修改完一分钟实时生效。

4.2.1 eureka.service.url - Eureka服务Url

> *不适用于基于Kubernetes原生服务发现场景*

不管是apollo-configservice还是apollo-adminservice都需要向eureka服务注册，所以需要配置eureka服务地址。 按照目前的实现，apollo-configservice本身就是一个eureka服务，所以只需要填入apollo-configservice的地址即可，如有多个，用逗号分隔（注意不要忘了/eureka/后缀）。

需要注意的是每个环境只填入自己环境的eureka服务地址，比如FAT的apollo-configservice是1.1.1.1:8080和2.2.2.2:8080，UAT的apollo-configservice是3.3.3.3:8080和4.4.4.4:8080，PRO的apollo-configservice是5.5.5.5:8080和6.6.6.6:8080，那么：

1. 在FAT环境的ApolloConfigDB.ServerConfig表中设置eureka.service.url为：

```
http://1.1.1.1:8080/eureka/,http://2.2.2.2:8080/eureka/
```

2. 在UAT环境的ApolloConfigDB.ServerConfig表中设置eureka.service.url为：

```
http://3.3.3.3:8080/eureka/,http://4.4.4.4:8080/eureka/
```

3. 在PRO环境的ApolloConfigDB.ServerConfig表中设置eureka.service.url为：

```
http://5.5.5.5:8080/eureka/,http://6.6.6.6:8080/eureka/
```

> *注:这里需要填写本环境中全部的eureka服务地址，因为eureka需要互相复制注册信息*

> *注2：在多机房部署时，往往希望config service和admin service只向同机房的eureka注册，要实现这个效果，需要利用 `ServerConfig` 表中的cluster字段，config service和admin service会读取所在机器的 `/opt/settings/server.properties` （Mac/Linux）或 `C:\opt\settings\server.properties` （Windows）中的idc属性，如果该idc有对应的eureka.service.url配置，那么就只会向该机房的eureka注册。比如config service和admin service会部署到 `SHAOY` 和 `SHAJQ` 两个IDC，那么为了实现这两个机房中的服务只向该机房注册，那么可以在 `ServerConfig` 表中新增两条记录，分别填入 `SHAOY` 和 `SHAJQ` 两个机房的eureka地址即可， `default` cluster的记录可以保留，如果有config service和admin service不是部署在 `SHAOY` 和 `SHAJQ` 这两个机房的，就会使用这条默认配置。*

Key	Cluster	Value	Comment
eureka.service.url	default	http://1.1.1.1:8080/eureka/	默认的Eureka服务Url
eureka.service.url	SHAOY	http://2.2.2.2:8080/eureka/	SHAOY的Eureka服务Url
eureka.service.url	SHAJQ	http://3.3.3.3:8080/eureka/	SHAJQ的Eureka服务Url

4.2.2 namespace.lock.switch - 一次发布只能有一个人修改开关，用于发布审核

这是一个功能开关，如果配置为true的话，那么一次配置发布只能是一个人修改，另一个发布。

> *生产环境建议开启此选项*

4.2.3 config-service.cache.enabled - 是否开启配置缓存

这是一个功能开关，如果配置为true的话，config service会缓存加载过的配置信息，从而加快后续配置获取性能。

默认为false，开启前请先评估总配置大小并调整config service内存配置。

> *开启缓存后必须确保应用中配置的app.id大小写正确，否则将获取不到正确的配置*

4.2.4 item.key.length.limit - 配置项 key 最大长度限制

默认配置是128。

4.2.5 item.value.length.limit - 配置项 value 最大长度限制

默认配置是20000。

4.2.5.1 namespace.value.length.limit.override - namespace 的配置项 value 最大长度限制

此配置用来覆盖 `item.value.length.limit` 的配置，做到细粒度控制 namespace 的 value 最大长度限制，配置的值是一个 json 格式，json 的 key 为 namespace 在数据库中的 id 值，格式如下：

```
namespace.value.length.limit.override = {1:200,3:20}
```

以上配置指定了 ApolloConfigDB.Namespace 表中 id=1 的 namespace 的 value 最大长度限制为 200，id=3 的 namespace 的 value 最大长度限制为 20

4.2.6 admin-service.access.control.enabled - 配置apollo-adminservice是否开启访问控制

默认为false，如果配置为true，那么apollo-portal就需要正确配置访问该环境的access token，否则访问会被拒绝。

4.2.7 admin-service.access.tokens - 配置允许访问apollo-adminservice的access token列表

如果该配置项为空，那么访问控制不会生效。如果允许多个token，token 之间以英文逗号分隔

样例：

```
admin-service.access.tokens=098f6bcd4621d373cade4e832627b4f6
admin-
service.access.tokens=098f6bcd4621d373cade4e832627b4f6,ad0234829205b903
```

4.2.8 apollo.access-key.auth-time-diff-tolerance - 配置服务端AccessKey校验容忍的时间偏差

默认值为60，单位为秒。由于密钥认证时需要校验时间，客户端与服务端的时间可能存在时间偏差，如果偏差太大会导致认证失败，此配置可以配置容忍的时间偏差大小，默认为60秒。

4.2.9 apollo.eureka.server.security.enabled - 配置是否开启eureka server的登录认证

默认为`false`，如果希望提升安全性（比如公网可访问的场景），可以设置该配置项为`true`启用登录认证。

需要注意的是，开启登录认证后，eureka.service.url中的地址需要配置用户名和密码，如：

```
http://some-user-name:some-password@1.1.1.1:8080/eureka/,http://some-user-name:some-password@2.2.2.2:8080/eureka/
```

其中`some-user-name`和`some-password`需要和`apollo.eureka.server.security.username`以及`apollo.eureka.server.security.password`的配置项一致。

4.2.10 apollo.eureka.server.security.username - 配置eureka server的登录用户名

配置eureka server的登录用户名，需要和`apollo.eureka.server.security.enabled`一起使用。

> *注意：用户名不能配置为apollo*

4.2.11 apollo.eureka.server.security.password - 配置eureka server的登录密码

配置eureka server的登录密码，需要和`apollo.eureka.server.security.enabled`一起使用。

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com