

# Apollo 调研

## 目录

为什么 Apollo.....	3
1. 统一管理不同环境、不同集群的配置.....	3
1.1.管理不同的环境、不同集群、不同命名空间配置.....	3
1.2.不同集群，可以有不同的配置.....	3
1.3.命名空间支持支持不同的应用共享同一份配置.....	3
2. 配置修改实时发布(热发布).....	3
3. 版本发布管理.....	4
4. 灰度发布.....	4
5. 权限管理、发布审核、操作审计.....	4
1. 应用.....	4
2. 环境.....	5
3. 集群.....	5
4. 命名空间.....	6
1. 客户端获取“application” Namespace 的代码如下.....	6
2. 客户端获取非“application” Namespace 的代码如下.....	6
4.1. 命名空间格式.....	6
4.2. 命名空间获取权限分类.....	6
4.3. 命名空间类型.....	7
k1 = v1.....	8
k2 = v2.....	8
k1 = v3.....	8
k2 = v2.....	8
5. 权限控制.....	8
6. Apollo 设计介绍.....	8
6.1. Config Service.....	9
1. 服务于 Apollo 客户端对配置的操作.....	9
2. 提供配置更新推送接口.....	9
6.2. Admin Service.....	9
6.3. Meta Server.....	9
6.4. Eureka.....	9
6.5. Portal.....	9
6.6. Client.....	9
6.7. 基础设计.....	10
1. 用户在配置中心对配置进行修改并发布.....	10
2. 配置中心通知客户端有配置更新.....	10
3. Apollo 客户端从配置中心拉取最新的配置、更新本地配置并通知到应用.....	10
7. Java 中使用 apollo.....	11
7.1 导入依赖.....	11
7.2 使用 API 方式获取配置.....	11
7.3 Meta Server 配置.....	11

7. 4. AppId 的配置.....	11
7. 5. Environment 配置.....	12
7. 5. 1. 通过 Java System Property 来指定环境.....	12
7. 5. 2. 通过配置文件 server.properties.....	12
7. 5. 3. 通过代码设置环境.....	12
7. 6. 监听配置修改.....	12
7. 7. 代码结果.....	12
否则无法获取到监听修改配置.....	14
5. 修改配置监听结果.....	14
7. 8. 获取公共 NameSpace 的配置.....	14
7. 9. 获取非 properties 格式 namespace 的配置.....	14
7. 9. 1. yaml/yml 格式的 namespace.....	15
7. 9. 2. 非 yaml/yml 格式的 namespace.....	15
8. Apollo 使用指南.....	15
8. 1. 普通应用接入指南.....	15
8. 1. 1. 创建项目.....	15
8. 1. 2. 项目权限分配.....	16
1. 管理项目的权限分配.....	16
2. 可以创建集群.....	16
3. 可以创建 NameSpace.....	16
8. 2. 公共组件接入指南.....	19
8. 2. 1. 公共组件接入步骤.....	19
8. 2. 2. 应用覆盖公用组件配置步骤.....	20
8. 3. 集群独立配置说明.....	22
9. 4. 多个 AppId 使用同一份配置.....	23
9. 5. 灰度发布使用指南.....	23
9. 5. 1. 灰度发布.....	26
9. 5. 2. 全量发布.....	26
9. 5. 3. 放弃灰度.....	26
9. 5. 4. 发布历史.....	26
9. 6. 其它功能配置.....	26
9. 6. 1. 配置查看权限.....	26
9. spring boot 中使用 apollo.....	27
9. 1. 准备 spring boot 项目.....	27
9. 2. 加入 apollo-client 依赖.....	27
9. 3. 配置 apollo 信息，放在 application.yml 中.....	27
9. 4. Placeholder 注入配置.....	28
9. 5. 配置迁移到 apollo.....	30
10. Apollo 服务端设计.....	31
10. 1. 配置发布后的实时推送设计.....	31
10. 2. 发送 ReleaseMessage 的实现方式.....	31
10. 3. Config Service 通知客户端的实现方式.....	32
10. 4. 源码解析实时推送设计.....	32
11. Apollo 客户端设计.....	32

12.1. 设计原理.....	33
12.2. 和 Spring 集成的原理.....	33
12.3. 启动时初始化配置到 Spring .....	33
12.4. 运行中修改配置如何更新.....	33
13. Apollo 高可用设计.....	33
12. linux 与 docker 部署.....	34
12.1. linux 的部署.....	34
6.1.1. 环境准备.....	34
1. 确保 Java 已经正确安装.....	34
2. 关闭防火墙.....	34
3. 去 git 上下载 apollo 包.....	34
6.1.4. 启动 apollo.....	35
6.1.5. 验证 Apollo.....	36
12.2. Docker 的部署.....	36
13. Portal 实现用户登录功能.....	36
13.1. 使用 Apollo 提供的 Spring Security 简单认证.....	36
13.2. 接入公司的统一登录认证系统.....	37
14. 接入邮件.....	37
15. Apollo 开放平台.....	37
15.1. 申请 token.....	37
第三方应用未接入会提示创建第三方应用,创建如上.....	38
16. Apollo 使用场景和代码.....	39

# 为什么 Apollo

## 1. 统一管理不同环境、不同集群的配置

- 1.1. 管理不同的环境、不同集群、不同命名空间配置
- 1.2. 不同集群，可以有不同的配置
- 1.3. 命名空间支持支持不同的应用共享同一份配置

## 2. 配置修改实时发布(热发布)

修改完配置并发布后，客户端能实时（1 秒）接收到最新的配置，并通知到应用程序

### 3. 版本发布管理

配置发布有版本概念、支持回滚

### 4. 灰度发布

点击灰度发布，只对部分应用实例起作用，没有问题再推到所有的应用实例

### 5. 权限管理、发布审核、操作审计

权限

审计日志追踪问题

## 1. 应用

即项目，用 `appId` 标识指定，可根据此 `appId` 去配置中心获取对应的配置，一般在 `application.yml` 文件中

注意:`appId` 的配置方式

### 3. Spring Boot application.properties

Apollo 1.0.0+支持通过Spring Boot的application.properties文件配置，如

```
app.id=YOUR-APP-ID
```

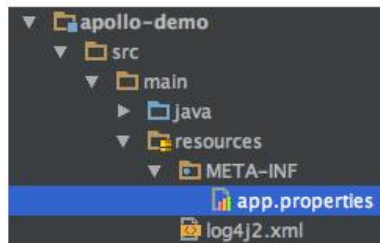
该配置方式不适用于多个war包部署在同一个tomcat的使用场景

### 4. app.properties

确保classpath:/META-INF/app.properties文件存在，并且其中内容形如：

```
app.id=YOUR-APP-ID
```

文件位置参考如下：



注：app.id是用来标识应用身份的唯一id，格式为string。

## 2. 环境

Local 本地环境

DEv 开发环境

FAT 测试环境

UAT 集成环境

PRO 生产环境

## 3. 集群

一个应用下不同实例的分组，比如把上海的机房应用实例分为一个集群，把北京机房的应用实例分为另一个集群

Apollo 支持配置按照集群划分，也就是说对于一个 appId 和一个环境，对不同的集群可以有不同的配置。

## 4. 命名空间

一个项目里有多个配置文件，则有多个命名空间，即多个配置文件，即多个 application.properties

命名空间可以用来对配置做分类，不同类型的配置放在在不同的命名空间

可以让多个项目拥有多个配置

注意:获取命名空间配置

1. 客户端获取“application” Namespace 的代码如下

```
Config config = ConfigService.getAppConfig();
```

2. 客户端获取非“application” Namespace 的代码如下

```
Config config = ConfigService.getConfig(namespaceName);
```

### 4.1. 命名空间格式

properties、xml、yaml、json 等



红色框的标明此“application”是 properties 格式的。

### 4.2. 命名空间获取权限分类

命名空间的权限分为两种:private 与 public

注意:获取权限是相对于 apollo 的客户端来说的

区别:

### 4.2.1. private 权限

private 权限的 Namespace，只能被所属的应用获取到。一个应用尝试获取其它应用 private 的 Namespace，Apollo 会报“404”异常。

### 4.2.2. public 权限

public 权限的 Namespace，能被任何应用获取

## 4.3. 命名空间类型

### 4.3.1. 私有类型

私有类型的 Namespace 具有 private 权限。例如上文提到的“application” Namespace 就是私有类型。

### 4.3.2. 公共类型

公共类型的 Namespace 具有 public 权限。公共类型的 Namespace 相当于游离于应用之外的配置，且通过 Namespace 的名称去标识公共 Namespace，所以公共的 Namespace 的名称必须全局唯一。

使用场景:

- 部门级别共享的配置
- 小组级别共享的配置
- 几个项目之间共享的配置
- 中间件客户端的配置

### 4.3.3. 关联类型

又可称为继承类型,关联类型具有 **private** 权限。关联类型的 **Namespace** 继承于公共类型的 **Namespace**，用于覆盖公共 **Namespace** 的某些配置。例如公共的 **Namespace** 有两个配置项:

k1 = v1

k2 = v2

然后应用 A 有一个关联类型的 **Namespace** 关联了此公共 **Namespace**，且覆盖了配置项 k1，新值为 v3。那么在应用 A 实际运行时，获取到的公共 **Namespace** 的配置为:

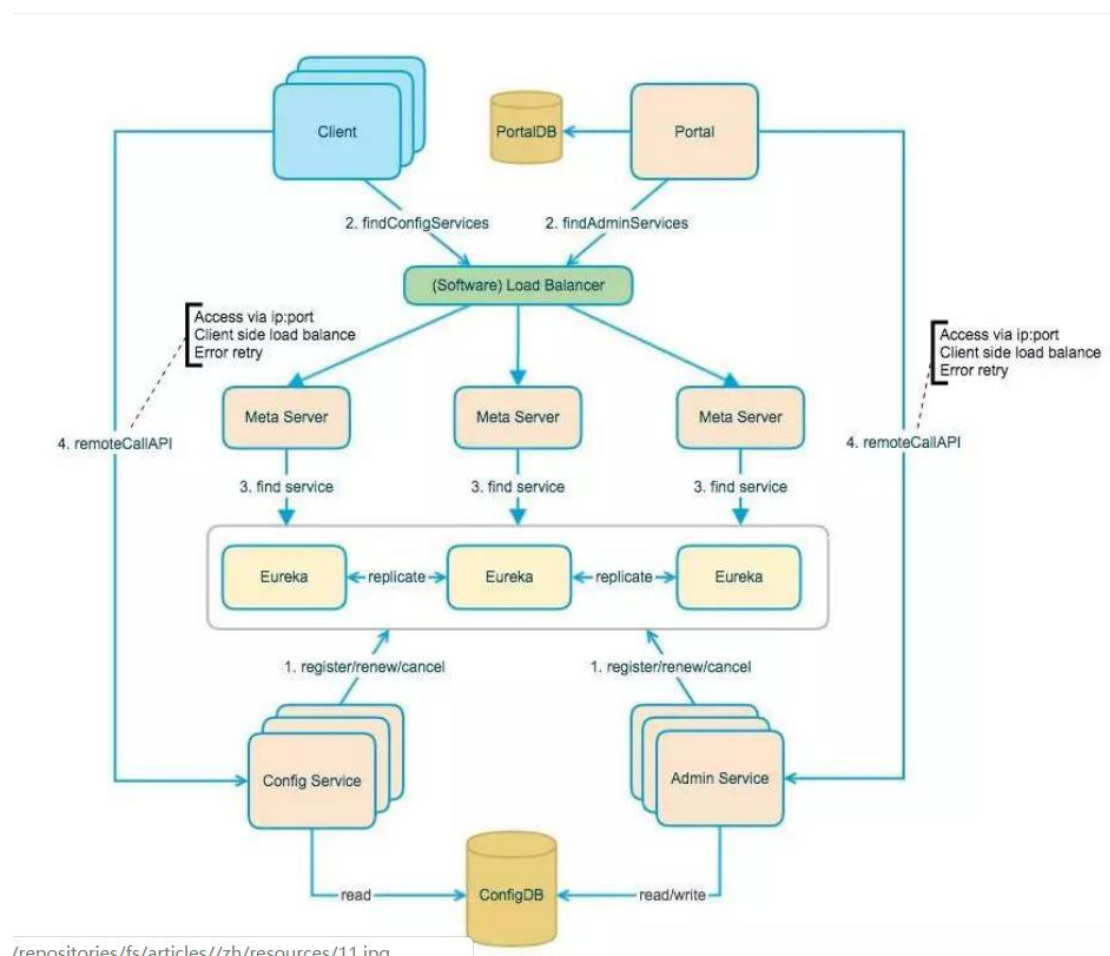
k1 = v3

k2 = v2

## 5. 权限控制

可以防止配置被不想干人误操作，对于开发人员，可以只分配测试环境的修改权限和发布权限，只有负责人才能有正式环境权限

## 6. Apollo 设计介绍





## 6.1. Config Service

1. 服务于 Apollo 客户端对配置的操作
2. 提供配置更新推送接口

## 6.2. Admin Service

服务于后台 Portal (web 管理端), 提供配置管理接口

## 6.3. Meta Server

Meta Service 是对 Eureka 的一个封装, 提供 Http 接口获取 Admin Service 和 Config Service 的服务信息

部署时与 Config Service 是在一个 JVM 进程, 故 IP、端口和 Config Service 一致

Apollo 支持应用在不同的环境有不同的配置

## 6.4. Eureka

用于提供服务的注册和发现

Config Service 和 Admin Service 会向 Eureka 注册服务

Config Service 同时包含了 Eureka 和 Meta Server

## 6.5. Portal

后端 Web 界面的管理配置

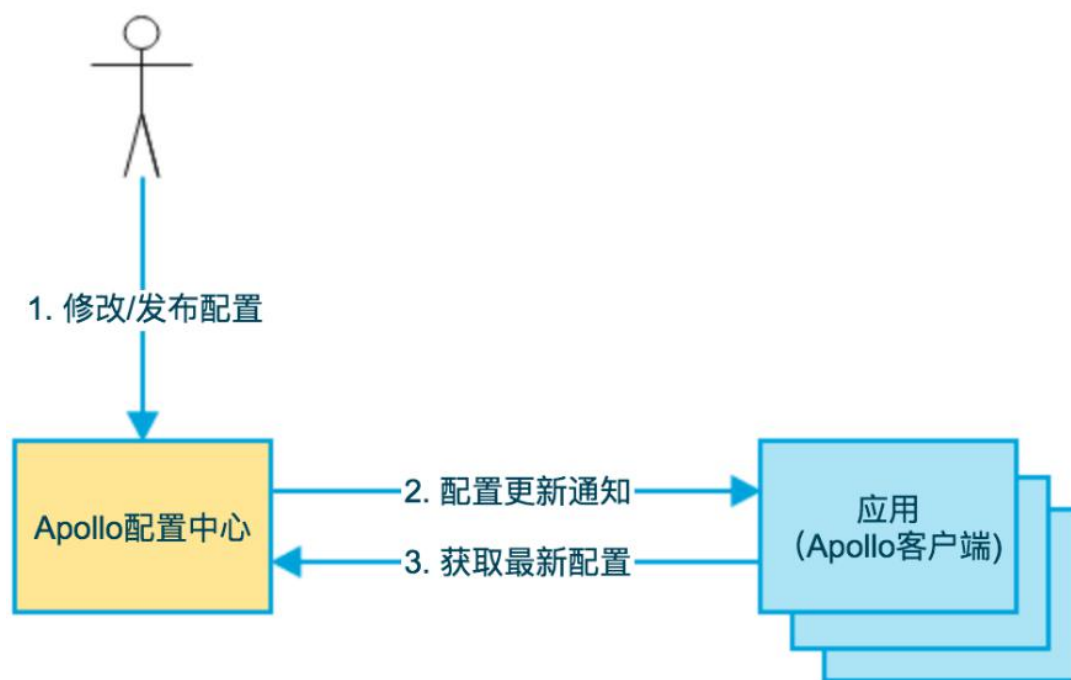
通过 Meta Server 获取 Admin Service 服务列表(IP+PORT)管理, 在客户端做负载均衡

## 6.6. Client

Apollo 提供的客户端, 用于项目中对配置的获取、更新

通过 Meta Server 获取 Config Service 服务列表(ip+port)进行配置管理, 在客户端内做负载均衡

## 6.7. 基础设计



1. 用户在配置中心对配置进行修改并发布
2. 配置中心通知客户端有配置更新
3. APollo 客户端从配置中心拉取最新的配置、更新本地配置并通知到应用

## 6.7. 设计流程

### 6.7.1. Portal 管理配置流程

Portal 连接 PortalDB,通过域名访问 Meta Server 获取 Admin Server 服务列表, 直接对 Admin Service 会对 ConfigDB 进行数据库操作

### 6.7.2. 客户端获取配置流程

Client 通过域名访问 Meta Server 获取 Config Service 服务列表, 直接对 Config Service 对 ConfigDB 进行数据库操作

### 6.7.3. Meta Server 获取服务列表流程

Meta Server 会去 Eureka 中获取对应服务的实例信息，Eureka 中的实例信息是 Admin Service 与 Config Service 自动注册到 Eureka 中保持心跳

## 7. Java 中使用 apollo

### 7.1 导入依赖

apollo-client,apollo 客户端依赖

### 7.2 使用 API 方式获取配置

通过 ConfigService 得到配置对象

```
Config config = ConfigService.getAppConfig();
```

定义 key 与 defaultValue

通过 config.getProperty()方法获取想获取的配置

```
String username = config.getProperty(key,defaultValue)
```

defaultValue 是防止无返回值时避免空指针

### 7.3 Meta Server 配置

由于 Apollo 支持应用在不同的环境有不同的配置，故需 apollo meta server 配置

### 7.4 AppId 的配置

Appid 是应用的身份信息，是从服务端获取配置的一个重要信息

在 application.yml 中配置

## 7.5. Environment 配置

### 7.5.1. 通过 Java System Property 来指定环境

```
// 指定环境（开发演示用，不能用于生产环境）
System.setProperty("env", "DEV");
Config config = ConfigService.getAppConfig();
//getValue(config);
addChangeListener(config);
try {
    Thread.sleep(Integer.MAX_VALUE);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

### 7.5.2. 通过配置文件 server.properties

### 7.5.3. 通过代码设置环境


```
System.setProperty("env","DEV")
```

## 7.6. 监听配置修改

```
config.addChangeListener(new ConfigChangeListener() {
    public void onChange(ConfigChangeEvent changeEvent) {
        System.out.println("发生修改数据的命名空间是: " + changeEvent.getNamespace());
        for (String key : changeEvent.changedKeys()) {
            ConfigChange change = changeEvent.getChange(key);
            System.out.println(String.format("发现修改 - 配置key: %s, 原来的值: %s, 修改后的值: %s, 操作类型: %s", change.getPr
        }
    }
});
```

## 7.7. 代码结果

1.创建命名空间 application 并新增配置 guo:guo-test

 Apollo

搜索项目(AppId, 项目名) 帮助 管理工具 apollo

环境列表

DEV

项目信息

AppId: SampleApp  
应用名: Sample App  
部门: 样例部门(TE ST1)  
负责人: apollo  
邮箱: apollo@acme.com

管理项目

添加集群

添加Namespace

私有 properties

application

发布 回滚 发布历史 授权 灰度 设置

表格 文本 更改历史 实例列表

过滤配置 同步配置 比较配置 新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	timeout	100	sample timeout配置		2019-08-22 19:13:04	编辑 删除

公共 properties

application

发布 回滚 发布历史 授权 灰度 设置

表格 文本 更改历史 实例列表

过滤配置 同步配置 比较配置 新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	guo	guo-test		apollo	2019-08-25 09:14:39	编辑 删除

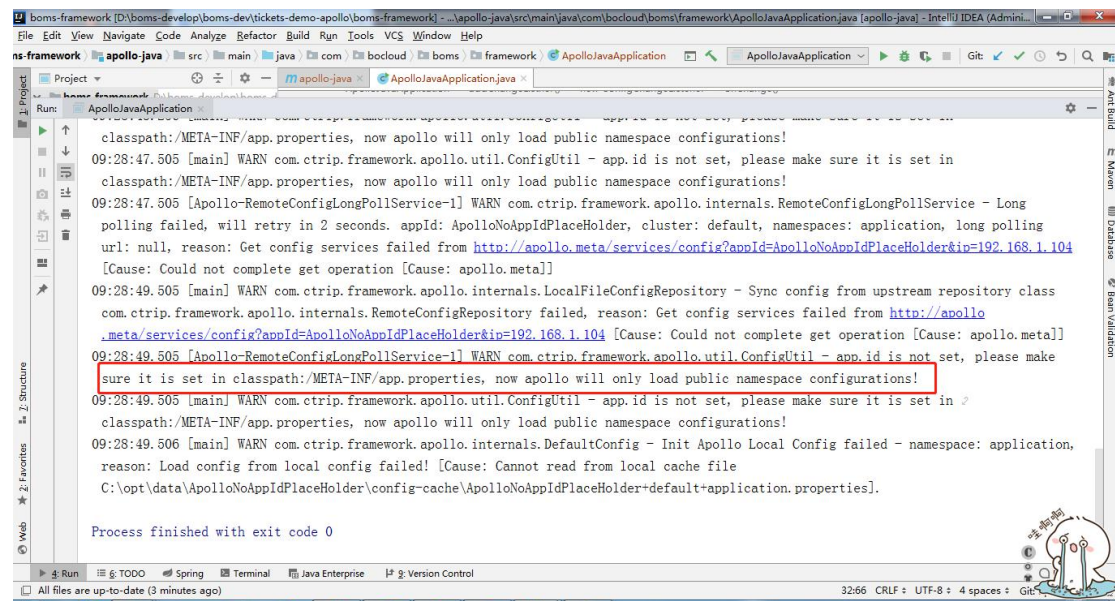
## 2. 运行 `getValue(config)` 获得 username, 即 `defaultValue`

```

boms-framework [D:\boms-develop\boms-dev\tickets-demo-apollo\boms-framework] - ...\apollo-java\src\main\java\com\bocloud\boms\framework\ApolloJavaApplication.java [apollo-java] - IntelliJ IDEA (Admini...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ns-framework apollo-java src main java com bocloud boms framework ApolloJavaApplication ApolloJavaApplication ApolloJavaApplication Git
Run ApolloJavaApplication
09:16:11.327 [Apollo-RemoteConfigLongPollService-1] WARN com.ctrip.framework.apollo.internals.RemoteConfigLongPollService - Long
polling failed, will retry in 2 seconds. appId: ApolloNoAppIdPlaceholder, cluster: default, namespaces: application, long polling
url: null, reason: Get config services failed from http://apollo.meta/services/config?appId=ApolloNoAppIdPlaceholder&ip=192.168.1.104
[Cause: Could not complete get operation [Cause: apollo.meta]]
09:16:13.326 [main] WARN com.ctrip.framework.apollo.internals.LocalFileConfigRepository - Sync config from upstream repository class
com.ctrip.framework.apollo.internals.RemoteConfigRepository failed, reason: Get config services failed from http://apollo
.meta/services/config?appId=ApolloNoAppIdPlaceholder&ip=192.168.1.104 [Cause: Could not complete get operation [Cause: apollo.meta]]
09:16:13.327 [main] WARN com.ctrip.framework.apollo.util.ConfigUtil - app.id is not set, please make sure it is set in
classpath:/META-INF/app.properties, now apollo will only load public namespace configurations!
09:16:13.327 [Apollo-RemoteConfigLongPollService-1] WARN com.ctrip.framework.apollo.util.ConfigUtil - app.id is not set, please make
sure it is set in classpath:/META-INF/app.properties, now apollo will only load public namespace configurations!
09:16:13.327 [main] WARN com.ctrip.framework.apollo.internals.DefaultConfig - Init Apollo Local Config failed - namespace: application,
reason: Load config from local config failed! [Cause: Cannot read from local cache file
C:\opt\data\ApolloNoAppIdPlaceholder\config-cache\ApolloNoAppIdPlaceholder+default+application.properties].
09:16:13.328 [main] WARN com.ctrip.framework.apollo.internals.DefaultConfig - Could not load config for namespace application from
Apollo, please check whether the configs are released in Apollo! Return default value now!
username: guo-test

Process finished with exit code 0
Compilation completed successfully in 7 s 626 ms (8 minutes ago)
12 chars 70:52 CRLF UTF-8 4 spaces Git
  
```

#### 4. 需要在 resource 目录下新建 META-INF 目录/app.properties 文件



```
classpath:/META-INF/app.properties, now apollo will only load public namespace configurations!
09:28:47.505 [main] WARN com.ctrip.framework.apollo.util.ConfigUtil - app.id is not set, please make sure it is set in
classpath:/META-INF/app.properties, now apollo will only load public namespace configurations!
09:28:47.505 [Apollo-RemoteConfigLongPollService-1] WARN com.ctrip.framework.apollo.internals.RemoteConfigLongPollService - Long
polling failed, will retry in 2 seconds. appId: ApolloNoAppIdPlaceholder, cluster: default, namespaces: application, long polling
url: null, reason: Get config services failed from http://apollo.meta/services/config?appId=ApolloNoAppIdPlaceholder&ip=192.168.1.104
[Cause: Could not complete get operation [Cause: apollo.meta]]
09:28:49.505 [main] WARN com.ctrip.framework.apollo.internals.LocalFileConfigRepository - Sync config from upstream repository class
com.ctrip.framework.apollo.internals.RemoteConfigRepository failed, reason: Get config services failed from http://apollo
.meta/services/config?appId=ApolloNoAppIdPlaceholder&ip=192.168.1.104 [Cause: Could not complete get operation [Cause: apollo.meta]]
09:28:49.505 [Apollo-RemoteConfigLongPollService-1] WARN com.ctrip.framework.apollo.util.ConfigUtil - app.id is not set, please make
sure it is set in classpath:/META-INF/app.properties, now apollo will only load public namespace configurations!
09:28:49.505 [main] WARN com.ctrip.framework.apollo.util.ConfigUtil - app.id is not set, please make sure it is set in
classpath:/META-INF/app.properties, now apollo will only load public namespace configurations!
09:28:49.506 [main] WARN com.ctrip.framework.apollo.internals.DefaultConfig - Init Apollo Local Config failed - namespace: application,
reason: Load config from local config failed! [Cause: Cannot read from local cache file
C:\opt\data\ApolloNoAppIdPlaceholder\config-cache\ApolloNoAppIdPlaceholder+default+application.properties].

Process finished with exit code 0
```

否则无法获取到监听修改配置

#### 5. 修改配置监听结果

### 7.8. 获取公共 NameSpace 的配置

```
//定义公共的命名空间
String somePublicNamespace = "CAT";
Config config = ConfigService.getConfig(somePublicNamespace);
//config instance is singleton for each namespace and is never null
//定义公共命名空间的 key
String someKey = "someKeyFromPublicNamespace";
//定义公共命名空间的 value
String someDefaultValue = "someDefaultValueForTheKey";
//获取最终相同的配置信息
String value = config.getProperty(someKey, someDefaultValue);
```

### 7.9. 获取非 properties 格式 namespace 的配置

注意:

apollo-client 1.3.0 版本开始对 **yaml/yml** 做了更好的支持, 使用起来和 **properties** 格式一致

### 7.9.1. yaml/yml 格式的 namespace

```
Config config = ConfigService.getConfig("application.yml");
String someKey = "someKeyFromYmlNamespace";
String someDefaultValue = "someDefaultValueForTheKey";
String value = config.getProperty(someKey, someDefaultValue);
```

### 7.9.2. 非 yaml/yml 格式的 namespace

```
String someNamespace = "test";
ConfigFile configFile = ConfigService.getConfigFile("test", ConfigFileFormat.XML);
String content = configFile.getContent();
```

## 8. Apollo 使用指南

参考地址:

<https://github.com/ctripcorp/apollo/wiki/Apollo%E4%BD%BF%E7%94%A8%E6%8C%87%E5%8D%97>

### 8.1. 普通应用接入指南

#### 8.1.1. 创建项目

点击红色框创建项目



可以先创建自己的用户

创建项目

\* 部门

\* 应用Id   
(应用唯一标识)

\* 应用名称   
(建议格式 xx-yy-zz 例:apollo-server)

\* 应用负责人

项目管理员   
(应用负责人默认具有项目管理员权限，  
项目管理员可以创建Namespace和集群、分配用户权限)

## 8.1.2. 项目权限分配

### 8.1.2.1. 项目管理员权限

1. 管理项目的权限分配
2. 可以创建集群
3. 可以创建 NameSpace



选择管理项目：

项目管理 ( AppId:boms )

返回到项目首页

管理员 (项目管理员具有以下权限: 1. 创建Namespace 2. 创建集群 3. 管理项目、Namespace权限)

添加

guo

✕

基本信息

\* AppId

boms

\* 部门

样例部门1(TEST1)

\* 项目名称

boms-guo

(建议格式 xx-yy-zz 例:apollo-server)

\* 项目负责人

guo

修改项目信息

8.1.2.2. 配置编辑、发布权限

项目创建完，默认没有分配配置的编辑和发布权限，需要项目管理员进行授权。

私有 properties

application

发布 回滚 发布历史 授权 灰度 设置

表格 文本 更改历史 实例列表

过滤配置 同步配置 比较配置 新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
------	-----	-------	----	-------	--------	----

选择红色框的授权按钮

Apollo

搜索项目(AppId、项目名)

添加成功

权限管理(AppId:boms Namespace:application)

返回到项目首页

修改权  
(可以修改配置)

所有环境

添加

所有环境

apollo

guo

DEV

发布权  
(可以发布配置)

所有环境

添加

所有环境

apollo

guo

8.1.3. 添加配置项



点击红色框的新增配置按钮

8.1.3.1. 通过表格模式添加配置

添加配置项 (温馨提示: 可以通过文本模式批量添加配置)

\* Key: portal.name

Value: guo

注意: 隐藏字符(空格、换行符、制表符Tab)容易导致配置出错, 如果需要检测Value中隐藏字符请点击 [检测隐藏字符](#)

Comment: test

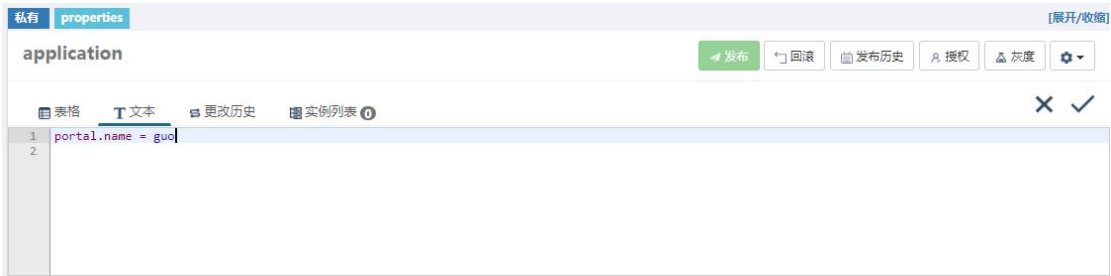
\* 选择集群

☐ 环境 集群

☒ DEV default

取消 提交

8.1.3.2. 通过文本模式编辑



通过文本模式编辑配置

8.1.4. 发布配置

点击发布按钮

### 8.1.5. 应用读取配置

具体读取配置参见 7.Java 中使用 apollo

### 8.1.6. 回滚已发布的配置

如果发现已发布的配置有问题,可以通过点击回滚按钮将客户端读取的配置回滚到上一个发布版本

回滚操作是将部署到机器上的安装包回滚到上一个部署的版本,但代码仓库中的代码是不会回滚的,从而开发可以在修复代码后重新发布

## 8.2. 公共组件接入指南

### 8.2.1. 公共组件接入步骤

#### 8.2.1.1. 创建项目

#### 8.2.1.2. 项目管理员权限

#### 8.2.1.3. 创建 Namespace

注意:公共组件的 Namespace 名称,需要注意的是 Namespace 名称全局唯一

关联公共Namespace

创建Namespace

AppId

boms

\* 名称

TEST1.

TEST1.

自动添加部门前缀

☒ TEST1.

(公共Namespace的名称需要全局唯一,添加部门前缀有助于保证全局唯一性)

\* 类型

☒ public

☐ private

备注

提交

- 8.2.1.4. 添加配置项
- 8.2.1.5. 应用读取配置

具体参见 7.Java 中使用 apollo

## 8.2.2. 应用覆盖公用组件配置步骤

### 8.2.2.1. 关联公共组件 Namespace

添加 Namespace

新建Namespace (点击了解更多Namespace相关知识)

返回到项目首页

Tips:

- 应用可以通过关联公共namespace来覆盖公共Namespace的配置
- 如果应用不需要覆盖公共Namespace的配置,那么无需关联公共Namespace

关联公共Namespace

创建Namespace

AppId

boms

\* 选择集群

☒ 环境

集群

☒ DEV

default

\* namespace

TEST1.publi-wq

提交

选择公共的 Namespace

test 应用先开始只有 application 这个命名空间, 关联之后的结果如下:

环境列表

DEV

项目信息

- AppId: SampleApp
- 应用名: Sample App
- 部门: 样例部门1(TE ST1)
- 负责人: apollo
- 邮箱: apollo@acme.com

管理项目

添加集群

添加Namespace

application

发布

回滚

发布历史

授权

灰度

更多

表格

文本

更改历史

实例列表

过滤配置

同步配置

比较配置

新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	timeout	109	sample timeout配置	apollo	2019-08-25 09:37:03	编辑

关联

properties

TEST1.publi-wq

发布

回滚

发布历史

授权

灰度

更多

同步配置

比较配置

覆盖的配置

filter by key ...

无覆盖的配置

公共的配置 (AppId:test, Cluster:default)

filter by key ...

Key	Value	备注	最后修改人	最后修改时间	操作
guo	guo-test		apollo	2019-08-25 13:02:52	编辑

### 8.2.2.2. 覆盖公用组件的配置

点击新增配置:注意要在创建关联的命名空间下新增配置,我是在 test 应用下

发布状态	Key ↑↓	Value	备注	最后修改人 ↑↓	最后修改时间 ↑↓	操作
已发布	guo	guo-test		apollo	2019-08-25 13:02:52	

点击新增配置，选择提交

添加配置项 (温馨提示: 可以通过文本模式批量添加配置)

\* Key: send.batcher

Value: 1200

注意: 隐藏字符(空格、换行符、制表符Tab)容易导致配置出错, 如果需要检测Value中隐藏字符请点击 [检测隐藏字符](#)

Comment: 测试覆盖公共组件配置

\* 选择集群

☒ 环境 ☐ 集群

☐ DEV default

取消 提交

选择点击发布按钮

### 8.2.2.3. 发布配置

发布 (只有发布过的配置才会被客户端获取到, 此次发布只会作用于当前环境:DEV)

Changes	Key	发布的值	未发布的值	修改人	修改时间
	send.batcher <span>新</span>		1200	apollo	2019-08-25 13:56:31

\* Release Name: 20190825135745-release

Comment: Add an optional extended description...

取消 发布

公共

properties

[展开/收缩]

TEST1.publi-wq

发布

回滚

发布历史

授权

灰度

表格

T 文本

更改历史

实例列表

过滤配置

同步配置

比较配置

+ 新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	guo	guo-test		apollo	2019-08-25 13:02:52	<div></div> <div></div>
已发布	send.batcher	1200	测试覆盖公共组件配置	apollo	2019-08-25 13:56:31	<div></div> <div></div>

此时在客户端可以看到此配置改变

## 8.3. 集群独立配置说明

点击集群按钮，弹出新增集群框，新增成功，会在环境列表看到新增的集群名称

Apollo

搜索项目(AppId, 项目名)

帮助 管理工具 apollo

环境列表

- DEV
- default 集群
- Jl-QUN 集群

项目信息

AppId: test  
应用名: test  
部门: 样例部门1(EST1)  
负责人: apollo  
邮箱: apollo@acme.com

管理项目 添加集群

注意: 所有不属于 Jl-QUN 集群的实例会使用default集群(当前页面)的配置, 属于 Jl-QUN 的实例会使用对应集群的配置!

私有 properties [展开/收缩]

application

发布 回滚 发布历史 授权 灰度 更多

表格 文本 更改历史 实例列表 过滤配置 同步配置 比较配置 新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	guo	guo-test9		apollo	2019-08-25 10:32:40	编辑 删除

公共 properties [展开/收缩]

TEST1.publi-wq

发布 回滚 发布历史 授权 灰度 更多

表格 文本 更改历史 实例列表 过滤配置 同步配置 比较配置 新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	guo	guo-test		apollo	2019-08-25 13:02:52	编辑 删除
已发布	send.batcher	1200	测试覆盖公共组件配置	apollo	2019-08-25 13:56:31	编辑 删除



不同的集群可以有不同的配置

## 9.4. 多个 AppId 使用同一份配置

尽管应用本身不是公共组件，但还是需要在多个 AppId 之间共用同一份配置，比如同一个产品的不同项目：XX-Web, XX-Service, XX-Job 等。

与添加公共组件一致

## 9.5. 灰度发布使用指南



新增灰度配置：

私有 properties		[展开/收缩]					
主版本 灰度版本							
application 有修改 1		灰度发布 全量发布 放弃灰度					
配置 灰度规则 灰度实例列表 更改历史							
灰度的配置		+ 新增灰度配置					
发布状态	Key ↑↓	主版本的值	灰度的值	备注	最后修改人 ↑↓	最后修改时间 ↑↓	操作
未发布	guo 改	3000	20000		apollo	2019-08-25 14:16:43	✎ ✕

## 新增规则

私有

properties

[展开/收缩]

主版本

灰度版本

application

有修改 1

灰度发布

全量发布

放弃灰度

配置

灰度规则

灰度实例列表 0

更改历史

灰度的AppId

灰度的IP列表

操作

新增规则

新增灰度规则，我的 test 应用下的 guo 这个配置机器 118 使用灰度版本 20000 测试

编辑灰度规则

\* 灰度的IP

从实例列表中选择

没找到你想要的IP？可以手动输入IP

10.20.11.118

添加

10.20.11.118 ✕

取消

完成



灰度列表展示

注意: 所有不属于 JI-QUN 集群的实例会使用default集群 (当前页面) 的配置, 属于 JI-QUN 的实例会使用对应集群的配置!

私有 properties [展开/收缩]

主版本 灰度版本

application 有修改

灰度发布 全量发布 放弃灰度

配置 灰度规则 灰度实例列表 更改历史

灰度的AppId	灰度的IP列表	操作
test	10.20.11.118	

如果没有问题则选择灰度发布

灰度发布 (灰度发布的配置只会作用于在灰度规则中配置的实例)

Changes

Key	主版本值	灰度版本发布的值	灰度版本未发布的值	修改人	修改时间
guo	3000	3000	20000	apollo	2019-08-25 14:16:43

\* Release Name

20190825142500-gray

Comment

Add an optional extended description...

取消 发布

发布后，切换到灰度实例列表 Tab，就能看到 10.20.11.118 已经使用了灰度发布的值。

切换到主版本的实例列表，会看到主版本配置只有 10.20.11.112 在使用了

选择全量发布发现主版本更改为灰度的版本的值已经在使用了

私有 properties		[展开/收缩]					
主版本 灰度版本							
application		灰度发布 全量发布 放弃灰度					
配置 灰度规则 灰度实例列表 更改历史							
灰度的配置		+ 新增灰度配置					
发布状态	Key ↑↓	主版本的值	灰度的值	备注	最后修改人 ↑↓	最后修改时间 ↑↓	操作
已发布	guo 改	20000	20000		apollo	2019-08-25 14:16:43	✎ ✕

### 9.5.1. 灰度发布

1.对程序影响较大的配置，可以在一个或者多个实例中生效，观察一段时间没问题再全量发布

### 9.5.2. 全量发布

### 9.5.3. 放弃灰度

### 9.5.4. 发布历史

## 9.6. 其它功能配置

### 9.6.1. 配置查看权限

支持配置某个环境只允许项目成员查看私有 Namespace 的配置。

这里的项目成员是指：

项目的管理员

具备该私有 Namespace 在该环境下的修改或发布权限

用超级管理员账号登录后，进入管理员工具 - 系统参数页面新增或修改 configView.memberOnly.envs 配置项即可，如下：

The screenshot shows the Apollo configuration management interface. At the top left is the Apollo logo. At the top right is a search bar labeled '搜索项目(AppId, 项目名)'. Below the search bar is a blue notification box with a warning icon and the text: 'Key: configView.memberOnly.envs 已存在，点击保存后会覆盖该配置项'. The main area contains a form titled '应用配置 (维护ApolloPortalDB.ServerConfig表数据, 如果已存在配置项则会覆盖, 否则创建配置项。配置更新后, 一分钟自动生效)'. The form has three fields: 'key' with the value 'configView.memberOnly.envs' and a '查询' button; 'value' with the value 'dev'; and 'comment' with the text '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'. At the bottom of the form is a '保存' button.

## 9. spring boot 中使用 apollo

注意:支持通过 `application.properties/bootstrap.properties` 来配置

配置示例:

### 1. 注入默认 `application` namespace的配置示例

```
# will inject 'application' namespace in bootstrap phase
apollo.bootstrap.enabled = true
```

### 2. 注入非默认 `application` namespace或多个namespace的配置示例

```
apollo.bootstrap.enabled = true
# will inject 'application', 'FX.apollo' and 'application.yml' namespaces in bootstrap p
apollo.bootstrap.namespaces = application,FX.apollo,application.yml
```

### 9.1. 准备 spring boot 项目

### 9.2. 加入 apollo-client 依赖

### 9.3. 配置 apollo 信息, 放在 `application.yml` 中

身份信息:app.id=

Meta Server: apollo.meat=

项目启动的 bootstrap 阶段，向 spring 容器注入配置信息  
apollo.bootstrap.enabled=true

注入命名空间:apollo.bootstrap.namespaces=

启动类可以指定环境:

```
public static void main(String[] args) {  
    // 指定环境（开发演示用，不能用于生产环境）  
    System.setProperty("env", "DEV");  
    SpringApplication.run(App.class, args);  
}
```

## 9.4. Placeholder 注入配置

### 1. Placeholder 注入配置

```
@Value("${username:guoshuang}")  
private String username;
```

### 2. Java Config 方式

```
@Data  
@Configuration  
public class UserConfig {  
  
    @Value("${username:guoshuang}")  
    private String username;  
  
}
```

### 3. Config 配置类注入

```
@Autowired
private UserConfig userConfig;
```

## 4. ConfigurationProperties 的使用方式

```
@Data
@Configuration
@ConfigurationProperties(prefix = "redis.cache")
public class RedisConfig {

    private String host;

}
```

配置中心只需要增加 redis.cache.host 配置项可实现注入，配置如下：

```
redis.cache.host = 10.20.11.113
```

缺点: **ConfigurationProperties** 方式有个缺点，当配置值发生变化时不会自动刷新，需手动刷新实现逻辑，一般用 **Java Config** 方式

## 5. Spring Anotation 注解的支持-@ApolloConfig

```
private Config config;

@GetMapping("/config/getUserName3")
public String getUserName3() {
    return config.getProperty("username", "guo shuang");
}
```

## 6. Spring Anotation 注解的支持-@ApolloConfigChangeListener

注解方式监听配置变化

此注解用来注册 ConfigChangeListener

代码如下：

```
@ApolloConfigChangeListener
```

```

private void someOnChange(ConfigChangeEvent changeEvent) {
    if(changeEvent.isChanged("username")) {
        System.out.println("username 发生修改了");
    }
}

```

## 7. Spring Anotation 注解的支持-@ApolloJsonValue

用来把配置的 JSON 字符串自动注入为对象

首先定义一个实体类

```

@Data
public class Student{

    private int id;

    private String name;

}

```

对象注入:

```

@ApolloJsonValue("${stus:[]}")
private List<Student> stus;

```

后台增加配置如下:

```

stus = [{"id":1,"name":"jason"}]

```

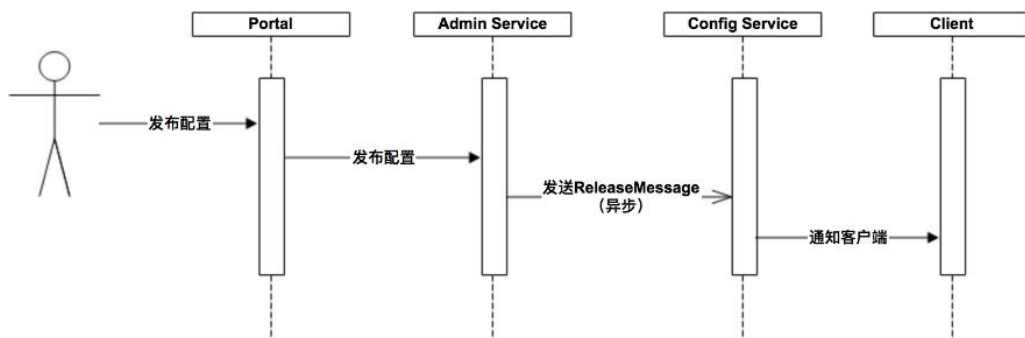
## 9.5. 配置迁移到 apollo

注意:对 apollo 版本有限制, 1.3.0 以上



## 10. Apollo 服务端设计

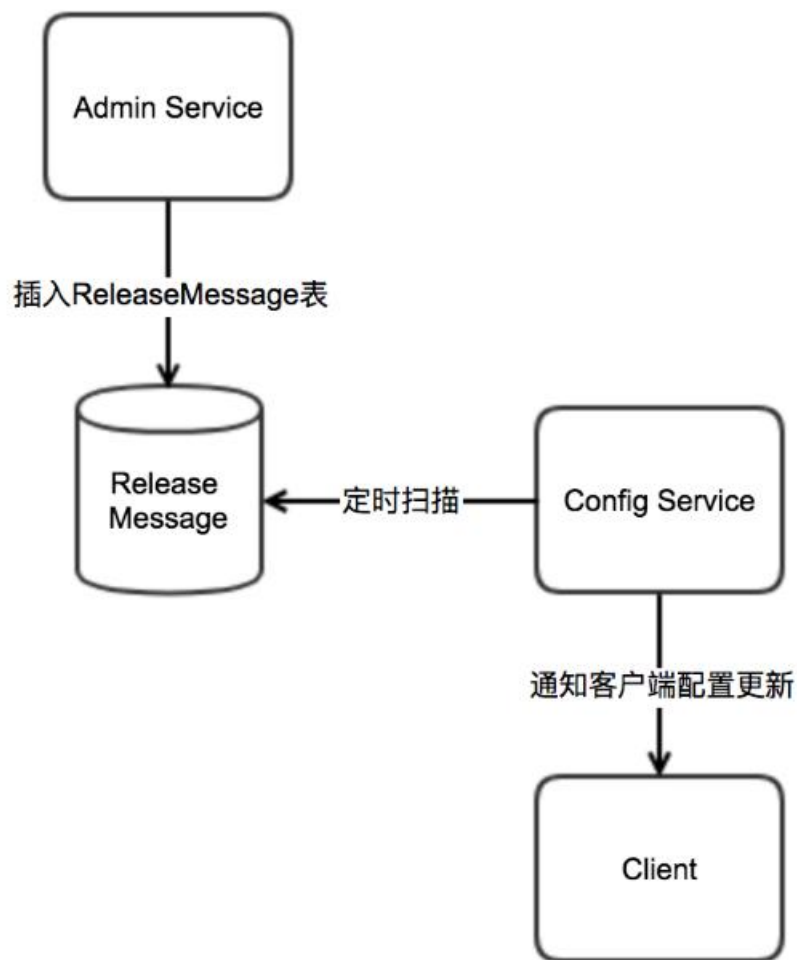
### 10.1. 配置发布后的实时推送设计



配置中心最重要的特性是实时推送，正因此，我们可以依赖配置中心做很多事情  
配置发布的大致过程如下：

1. 用户在 Portal 中进行配置发布的大致过程
2. Portal 会调用 Admin Service 提供的接口进行发布操作
3. Admin Service 收到请求后，发送 ReleaseMessage 给各个 Config Service,通知 Config Service 配置发生变化
4. Config Service 收到 ReleaseMessage，通知对应的客户端，基于 Http 长连接实现

### 10.2. 发送 ReleaseMessage 的实现方式



发送ReleaseMessage的大致过程如下：

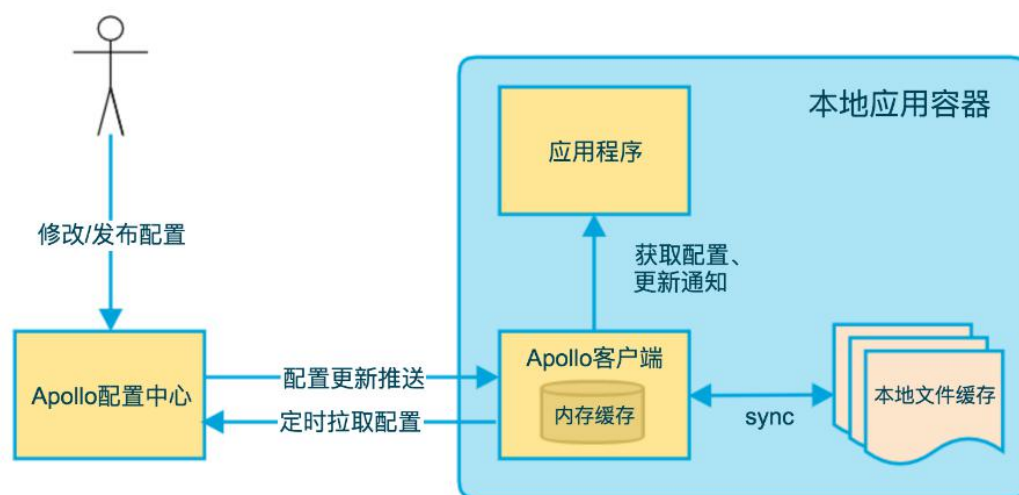
1. Admin Service在配置发布后会往ReleaseMessage表插入一条消息记录
2. Config Service会启动一个线程定时扫描ReleaseMessage表，来查看是否有新的消息记录
3. Config Service发现有新的消息记录，就会通知到所有的消息监听器
4. 消息监听器得到配置发布的信息后，就会通知对应的客户端

### 10.3. Config Service 通知客户端的实现方式

### 10.4. 源码解析实时推送设计

## 11. Apollo 客户端设计





## 1 2 . 1 设计原理

1. 客户端与服务端保持一个长连接，编译配置的实时更新推送
2. 定时拉取配置是客户端的本地的一个定时任务，默认为5分钟拉取一次，也可以通过在运行时指定`System.Property:apollo.refreshInterval`来覆盖，单位是分钟，推送+定时拉取
3. 客户端从Apollo配置中心服务端获取到应用的最新配置后，会保存在内存中
4. 客户端会把从服务端获取的配置在本地文件系统缓存一份，当服务或者网络不可用时可以使用本地配置，也就是我们的本地开发模式`env = Local`

## 1 2 . 2 . 和 S p r i n g 集成的原理

## 1 2 . 3 . 启动时初始化配置到 S p r i n g

## 1 2 . 4 . 运行中修改配置如何更新

# 13. Apollo 高可用设计

# 12. linux 与 docker 部署

## 12.1. linux 的部署

### 6.1.1. 环境准备

- 1.确保 Java 已经正确安装
- 2.关闭防火墙
- 3.去 git 上下载 apollo 包

### 6.1.2. 解压包

#### 1.创建四个目录

Apollo-admin:存放 apollo-adminservice-1.3.0-github.zip admin/  
Apollo-config: 存放 apollo-configservice-1.3.0-github.zip  
Apollo-portal: apollo-portal-1.3.0-github.zip  
Logs: 查看日志

#### 2. 创建目录的命令

mkdir xx 目录

#### 3. 解压命令

unzip xxx.zip 包

### 6.1.3. 修改配置

#### 1.apollo-config 配置文件，注意数据库路径修改为虚拟 Ip

```
# vim config/config/application-github.properties
spring.datasource.url=jdbc:mysql://10.20.11.114:3306/ApolloConfigDB?
characterEncoding=utf8
spring.datasource.username = root
spring.datasource.password = 123456
```

#### 2.apollo-admin 配置文件，注意数据库路径修改为虚拟 Ip# vim

```
admin/config/application-github.properties
```

```
spring.datasource.url=jdbc:mysql://10.20.11.114:3306/ApolloConfigDB?
characterEncoding=utf8
spring.datasource.username=root
spring.datasource.password=123456
```

### 3.apollo-portal 配置文件，注意数据库路径修改为虚拟 Ip

```
# vim portal/config/application-github.properties
spring.datasource.url=jdbc:mysql://10.20.11.114:3306/ApolloPortalDB?
characterEncoding=utf8
spring.datasource.username=root
spring.datasource.password=123456
```

4.这里的 apollo-env.properties 需要填写的是 apollo-config 的地址,因为我是在一台机器部署,所

以填写的是本机。如果部署同一台机器部署请修改 Ip 地址

```
vim portal/config/apollo-env.properties
local.meta=http://localhost:8080
dev.meta=http://localhost:8080
fat.meta=http://localhost:8080
uat.meta=http://localhost:8080
lpt.meta=${lpt_meta}
pro.meta=http://localhost:8080
```

## 6.1.4. 添加数据库

注意:添加的用户必须为系统用户

## 6.1.4.启动 apollo

注意:启动 apollo 必须按照 config-admin-portal 的顺序启动

### 1.启动 apollo-config

```
sh config/scripts/startup.sh
```

### 2.启动 apollo-admin

```
sh admin/scripts/startup.sh
```

#### 4. 启动 apollo-portal

sh portal/scripts/startup.sh

### 6.1.5.验证 Apollo

查看进程:

```
ps -ef | grep apollo
```

访问 web, 是否 apollo-admin 与 apollo-config 均已注册到 eureka

查看地址:

<http://10.20.11.118:8080/>

## 12.2. Docker 的部署

## 13. Portal 实现用户登录功能

### 13.1. 使用 Apollo 提供的 Spring Security 简单认证

Users 表中, 存在默认的用户名与密码分别是 apollo,admin

添加用户:超级管理员登录系统后打开管理员工具 - 用户管理即可添加用户。

用户管理 ( 仅对默认的Spring Security简单认证方式有效: -Dapollo\_profile=github,auth )

\* 用户名   
输入的用户名如果不存在, 则新建。若已存在, 则更新。

\* 密码

\* 邮箱

修改用户名密码:

管理工具 - 用户管理

## 13.2. 接入公司的统一登录认证系统

## 14. 接入邮件

## 15. Apollo 开放平台

目的:使第三方应用能够自己管理配置,在有些情景下,应用需要通过程序去管理配置

### 15.1. 申请 token

- 第三方应用的 AppId、应用名、部门
- 第三方应用负责人

创建第三方应用 (说明: 第三方应用可以通过Apollo开放平台来对配置进行管理)

\* 第三方应用ID   Token: App(boms)未创建, 请先创建  
(创建前请先查询第三方应用是否已经申请过)

\* 部门

\* 第三方应用名称   
(建议格式: xx-yy-zz 例apollo-server)

\* 项目负责人

第三方应用未接入会提示创建第三方应用,创建如上

创建第三方应用 (说明: 第三方应用可以通过Apollo开放平台来对配置进行管理)

\* 第三方应用ID

查询

(创建前请先查询第三方应用是否已经申请过)

\* 部门

样例部门1(TEST1)

\* 第三方应用名称

(建议格式 xx-yy-zz 例:apollo-server)

\* 项目负责人

guo | guo

创建

Token: fbaa19112e5d910d11d90ba32378bc13613880e8

赋权 (Namespace级别权限包括: 修改、发布Namespace, 应用级别权限包括: 创建Namespace、修改或发布应用下任何Namespace)

\* token

fbaa19112e5d910d11d90ba32378bc13613880e8

\* 被管理的AppId

被管理的Namespace

创建应用成功会出现 token

项目负责人

guo | guo

创建

赋权成功

赋权 (Namespace级别权限包括: 修改、发布Namespace, 应用级别权限包括: 创建Namespace、修改或发布应用下任何Namespace)

\* token

fbaa19112e5d910d11d90ba32378bc13613880e8

\* 被管理的AppId

bocloud

被管理的Namespace

application

(非properties类型的namespace请加上类型后缀, 例如apollo.xml)

授权类型

☒ Namespace

☐ App

环境

☒ DEV

(不选择则所有环境都有权限, 如果提示Namespace's role does not exist, 请先打开该Namespace的授权页面触发一下权限的初始化动作)

提交

赋权成功的条件是已经创建好 bocloud 这个项目,否则会报这个项目不存在

授权成功便可在第三方应用调用

首先引入 `apollo-openapi` 依赖：

```
<dependency>
  <groupId>com.ctrip.framework.apollo</groupId>
  <artifactId>apollo-openapi</artifactId>
  <version>1.1.0</version>
</dependency>
```

在程序中构造 `ApolloOpenApiClient`：

```
String portalUrl = "http://localhost:8070"; // portal url
String token = "e16e5cd903fd0c97a116c873b448544b9d086de9"; // 申请的token
ApolloOpenApiClient client = ApolloOpenApiClient.newBuilder()
    .withPortalUrl(portalUrl)
    .withToken(token)
    .build();
```

## 16. Apollo 使用场景和代码

学习地址:<https://github.com/ctripcorp/apollo-use-cases>