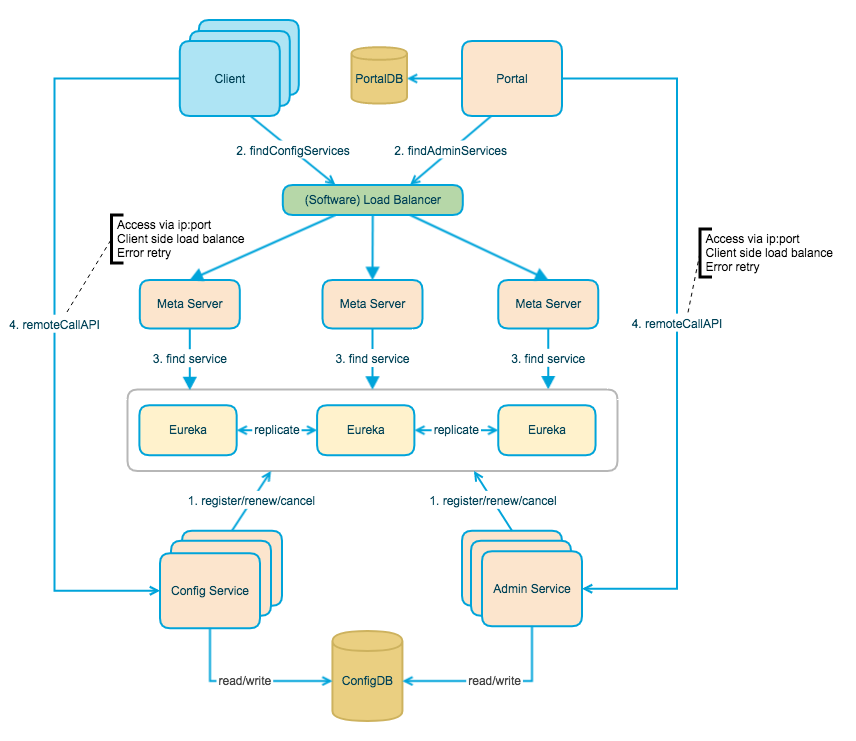
## 什么是分布式配置中心

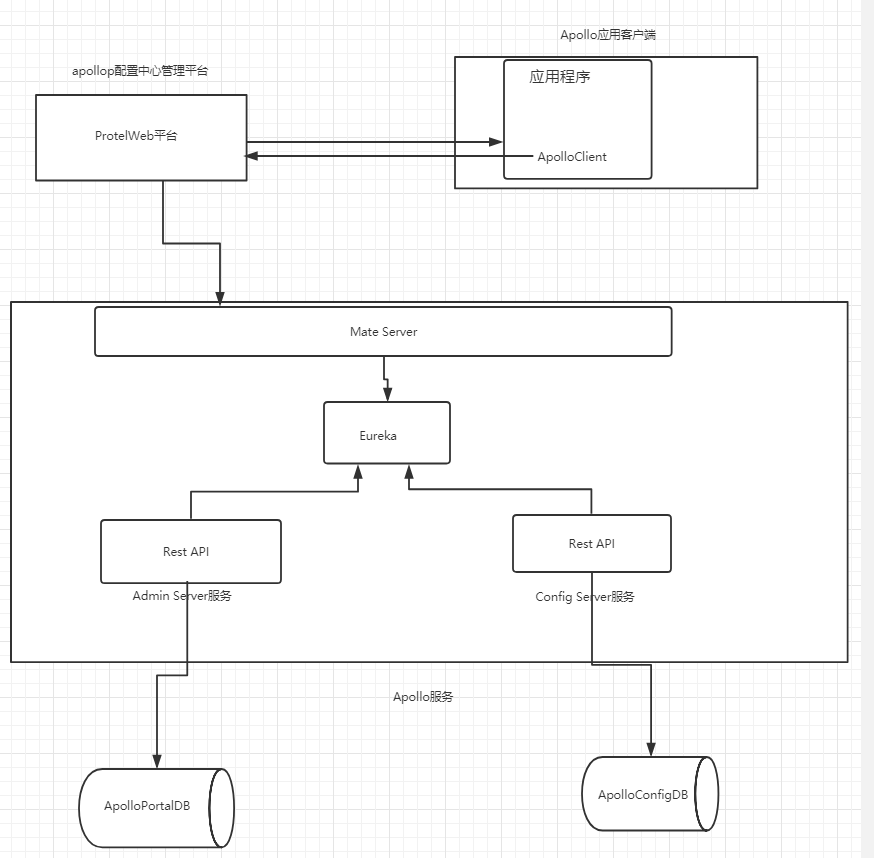
项目中配置文件比较繁杂，而且不同环境的不同配置修改相对频繁，每次发布都需要对应修改配置，如果配置出现错误，需要重新打包发布，时间成本较高，因此需要做统一的分布式注册中心，能做到自动更新配置文件信息，解决以上问题

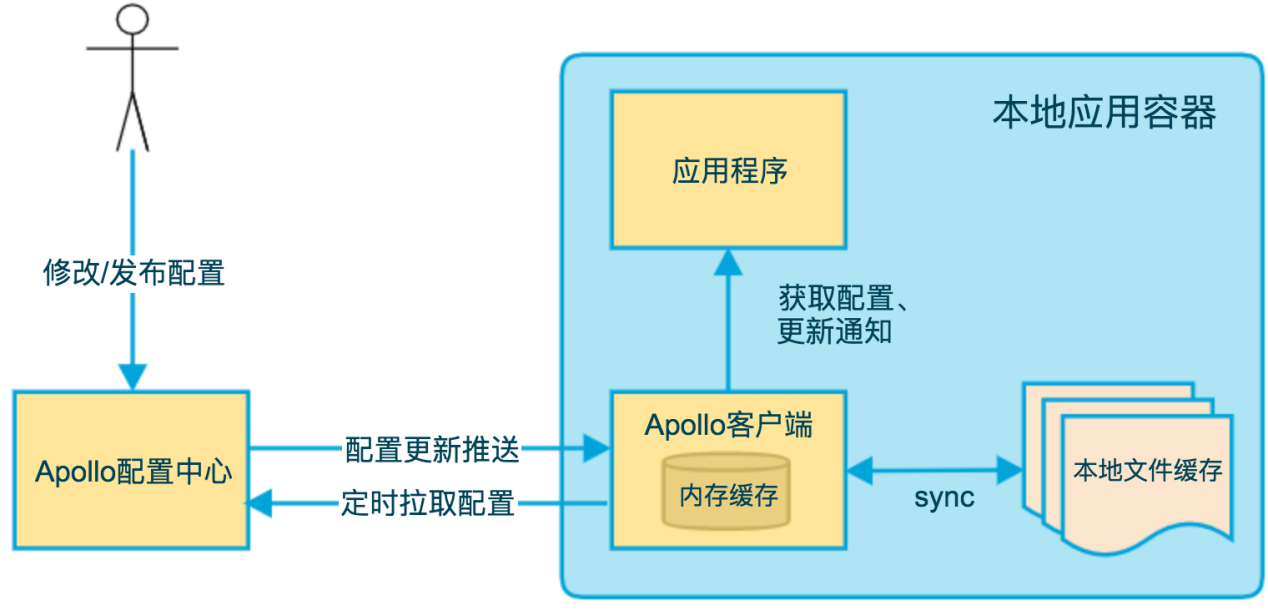
## 常用分布式配置中心框架

Disconf（依赖于zookpeer）、Zookpeer（）、Diamond、Apollo（阿波罗）、Redis、xxl-conf

## Apollo（阿波罗）：







详情见Apollo官网；https://github.com/ctripcorp/apollo

## Apollo阿波罗特点

用户在Apollo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。

### 统一管理不同环境、不同集群的配置

所有的配置发布都有版本概念，从而可以方便的支持配置的回滚。

### 配置修改实时生效（热发布）

用户在Apollo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

### 版本发布管理

所有的配置发布都有版本概念，从而可以方便的支持配置的回滚

### 灰度发布

支持配置的灰度发布，比如点了发布后，只对部分应用实例生效，等观察一段时间没问题后再推给所有应用实例。

### 权限管理、发布审核、操作审计

应用和配置的管理都有完善的权限管理机制，对配置的管理还分为了编辑和发布两个环节，从而减少人为的错误。

所有的操作都有审计日志，可以方便的追踪问题。

### 客户端配置信息监控

可以方便的看到配置在被哪些实例使用

### 提供Java和.Net原生客户端

提供了Java和.Net的原生客户端，方便应用集成

支持Spring Placeholder, Annotation和Spring Boot的ConfigurationProperties，方便应用使用（需要Spring 3.1.1+）

同时提供了Http接口，非Java和.Net应用也可以方便的使用

### 提供开放平台API

Apollo自身提供了比较完善的统一配置管理界面，支持多环境、多数据中心配置管理、权限、流程治理等特性。

不过Apollo出于通用性考虑，对配置的修改不会做过多限制，只要符合基本的格式就能够保存。

在我们的调研中发现，对于有些使用方，它们的配置可能会有比较复杂的格式，如xml, json，需要对格式做校验。

还有一些使用方如DAL，不仅有特定的格式，而且对输入的值也需要进行校验后方可保存，如检查数据库、用户名和密码是否匹配。

对于这类应用，Apollo支持应用方通过开放接口在Apollo进行配置的修改和发布，并且具备完善的授权和权限控制

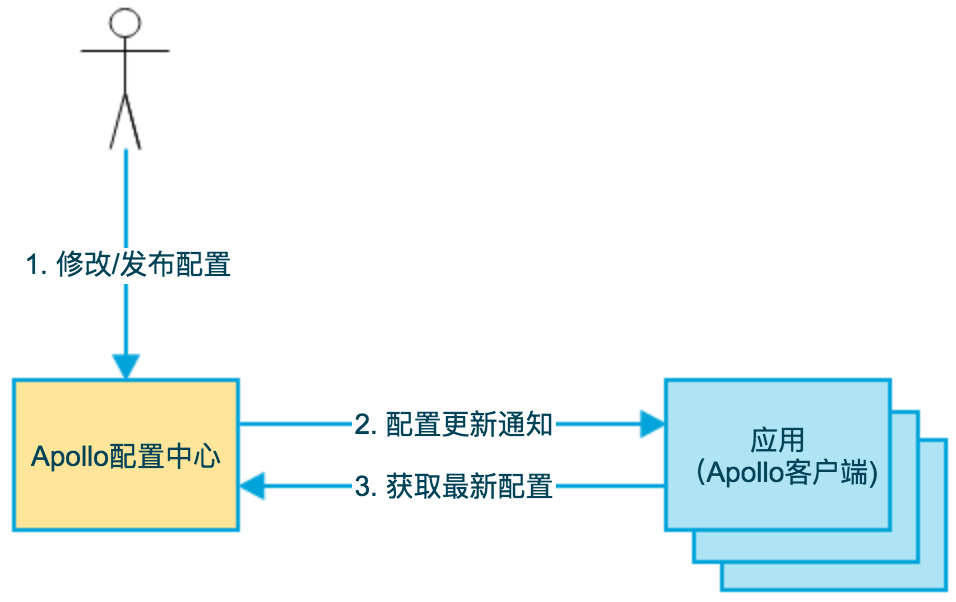
### 部署简单

配置中心作为基础服务，可用性要求非常高，这就要求Apollo对外部依赖尽可能地少

目前唯一的外部依赖是MySQL，所以部署非常简单，只要安装好Java和MySQL就可以让Apollo跑起来

Apollo还提供了打包脚本，一键就可以生成所有需要的安装包，并且支持自定义运行时参数

## Apollo整体架构原理

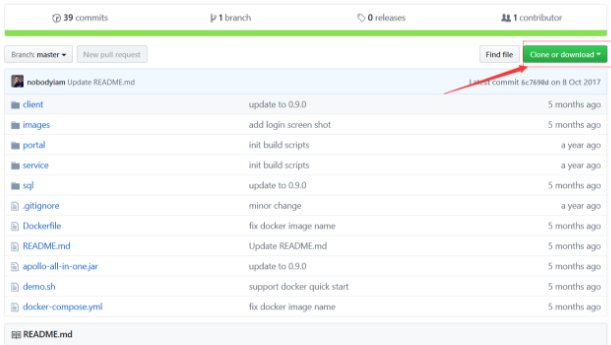


## Apollo(阿波罗)环境搭建

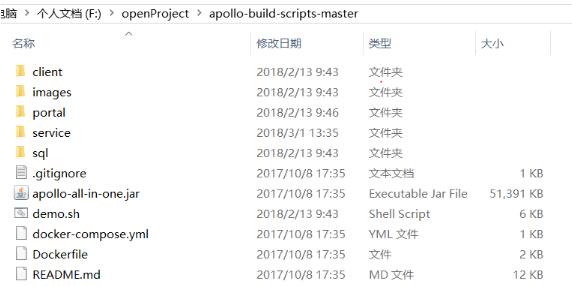
1. 下载apollo-client 打入到本地Maven仓库中
2. 部署

### 搭建Apollo配置中心

下载aploll配置中心 https://github.com/nobodyiam/apollo-build-scripts



下载好文件图



### 配置数据库参数配置

### 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。

**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。**

### 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，所以需要编辑demo.sh，修改ApolloPortalDB和ApolloConfigDB相关的数据库连接串信息。

注意：填入的用户需要具备对ApolloPortalDB和ApolloConfigDB数据的读写权限

|  |
| --- |
| #apollo config db info  apollo\_config\_db\_url=jdbc:mysql://localhost:3306/ApolloConfigDB?characterEncoding=utf8  apollo\_config\_db\_username=用户名  apollo\_config\_db\_password=密码（如果没有密码，留空即可）  # apollo portal db info  apollo\_portal\_db\_url=jdbc:mysql://localhost:3306/ApolloPortalDB?characterEncoding=utf8  apollo\_portal\_db\_username=用户名  apollo\_portal\_db\_password=密码（如果没有密码，留空即可） |

### 启动Apollo配置中心

#### 确保端口未被占用

Quick Start脚本会在本地启动3个服务，分别使用8070, 8080, 8090端口，请确保这3个端口当前没有被使用。

例如，在Linux/Mac下，可以通过如下命令检查：

#### 执行启动脚本

./demo.sh start

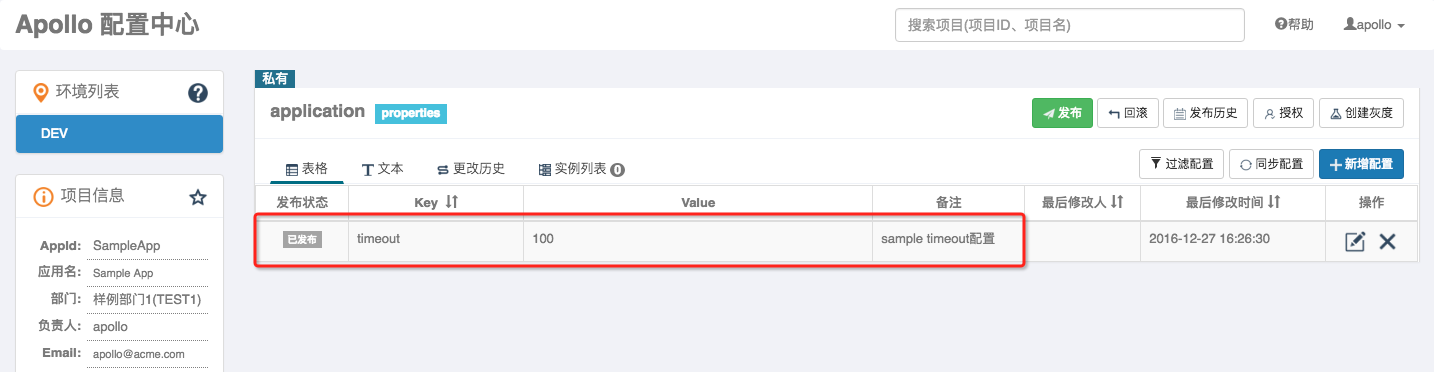
当看到如下输出后，就说明启动成功了！

### 使用Apollo配置中心

访问网址:打开http://localhost:8070



输入用户名apollo，密码admin后登录



### Apollo配置中心客户端集成

将apollo-master 打入到本地Maven仓库中

### Maven依赖信息

|  |
| --- |
| <!-- apollo 携程apollo配置中心框架 -->  <dependency>  <groupId>com.ctrip.framework.apollo</groupId>  <artifactId>apollo-client</artifactId>  <version>1.0.0</version>  </dependency>  <dependency>  <groupId>com.ctrip.framework.apollo</groupId>  <artifactId>apollo-core</artifactId>  <version>1.0.0</version>  </dependency> |

### application.yml配置文件信息

|  |
| --- |
| server:  port: 8001  spring:  application:  name: itmayiedu\_a  eureka:  client:  service-url:  defaultZone: http://192.168.212.162:8080/eureka |

### 修改环境

修改/opt/settings/server.properties（Mac/Linux）或C:\opt\settings\server.properties（Windows）文件，设置env为DEV：

env=DEV

### 创建apollo-env配置文件信息

apollo-env.properties

|  |
| --- |
| local.meta=http://192.168.212.162:8080  dev.meta=http://192.168.212.162:8080  fat.meta=${fat\_meta}  uat.meta=${uat\_meta}  lpt.meta=${lpt\_meta}  pro.meta=${pro\_meta} |

### App信息

在META-INF文件夹创建app.properties 指定appid

|  |
| --- |
| app.id=appid\_0064 |

### 项目启动

|  |
| --- |
| @Configuration  @EnableApolloConfig  @SpringBootApplication  @EnableAutoConfiguration  **public** **class** App {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(App.**class**, args);  }  } |

Apollo平台详细使用见官网：https://github.com/ctripcorp/apollo/wiki/Apollo使用指南

Java客户端具体接入见官网：https://github.com/ctripcorp/apollo/wiki/Java客服端使用指南