项目梳理-Apollo&RPC

RPC框架

> 为方法提供远程调用方式和接口

```
@SoaService //项目中注册方法使用的注解
@Slf4j
public class AssetBusinessServiceImpl implements AssetBusinessService {...}
```

- 通讯：客户端和服务器之间TCP连接（长连接）
- 寻址：基于Web服务协议栈的RPC，就要提供一个endpoint URI
- 远程调用：方法的参数需要通过底层的网络协议如TCP传递到B服务器，由于网络协议是基于二进制的，内存中的参数的值要序列化成二进制的形式，也就是序列化（Serialize）或编组（marshal）
- 本地调用：收到请求后，需要对参数进行反序列化（序列化的逆操作），恢复为内存中的表达方式，然后找到对应的方法（寻址的一部分）进行本地调用，然后得到返回值。
- 返回：经过序列化的方式发送，对方接到后，再反序列化，恢复为内存中的表达方式。

Apollo配置中心

集中化管理应用**不同环境、不同集群**的配置，配置修改后能够**实时推送到应用端，并且具备规范的权限、流程治理**等特性。

部署apollo

- 将配置文件上传至服务器
- 发布，执行脚本。启动注册中心->发布服务->启动web界面服务

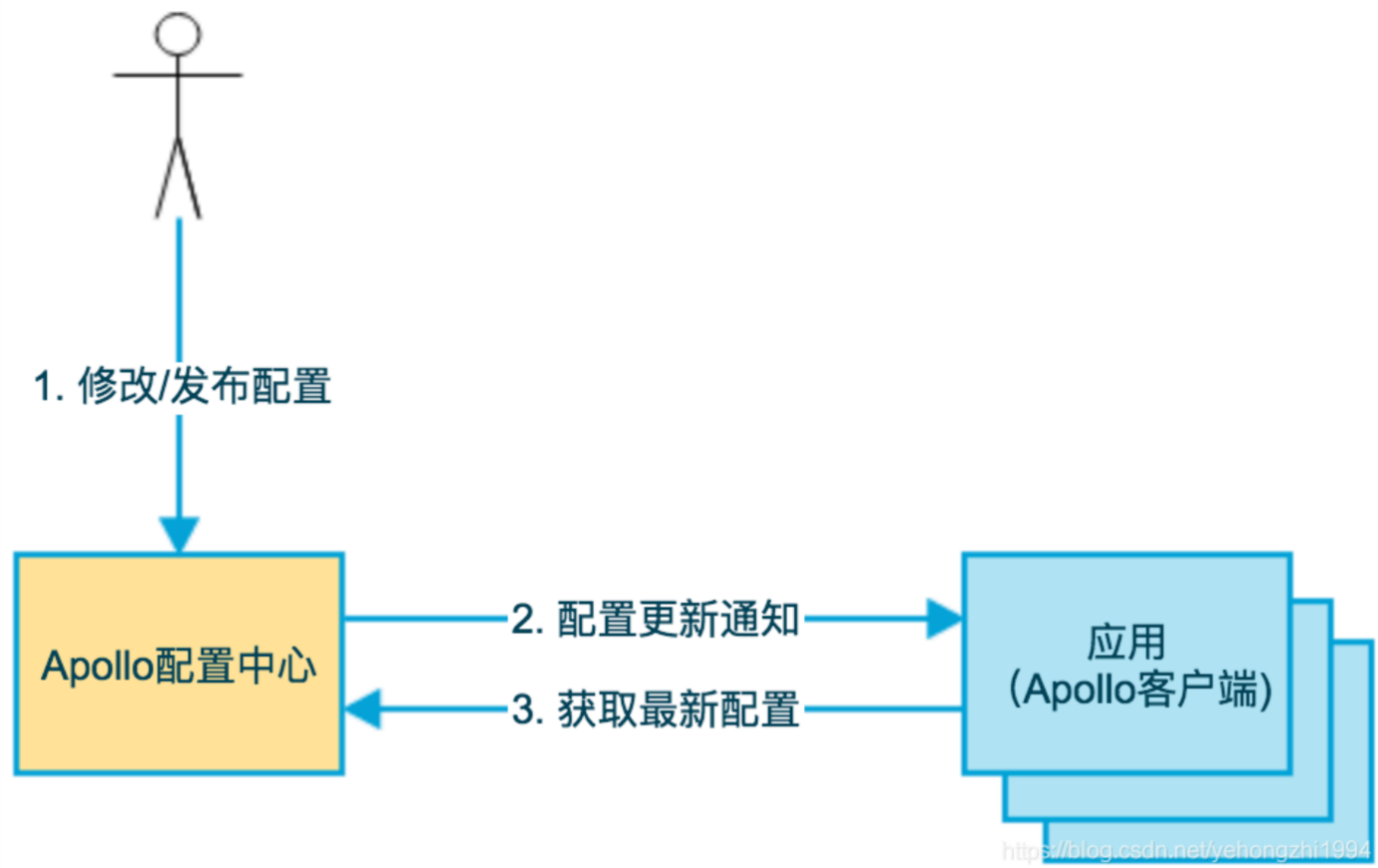
SpringBoot整合

- 引入maven依赖
- 客户端创建AppId（唯一标识
- 配置Apollo服务器地址

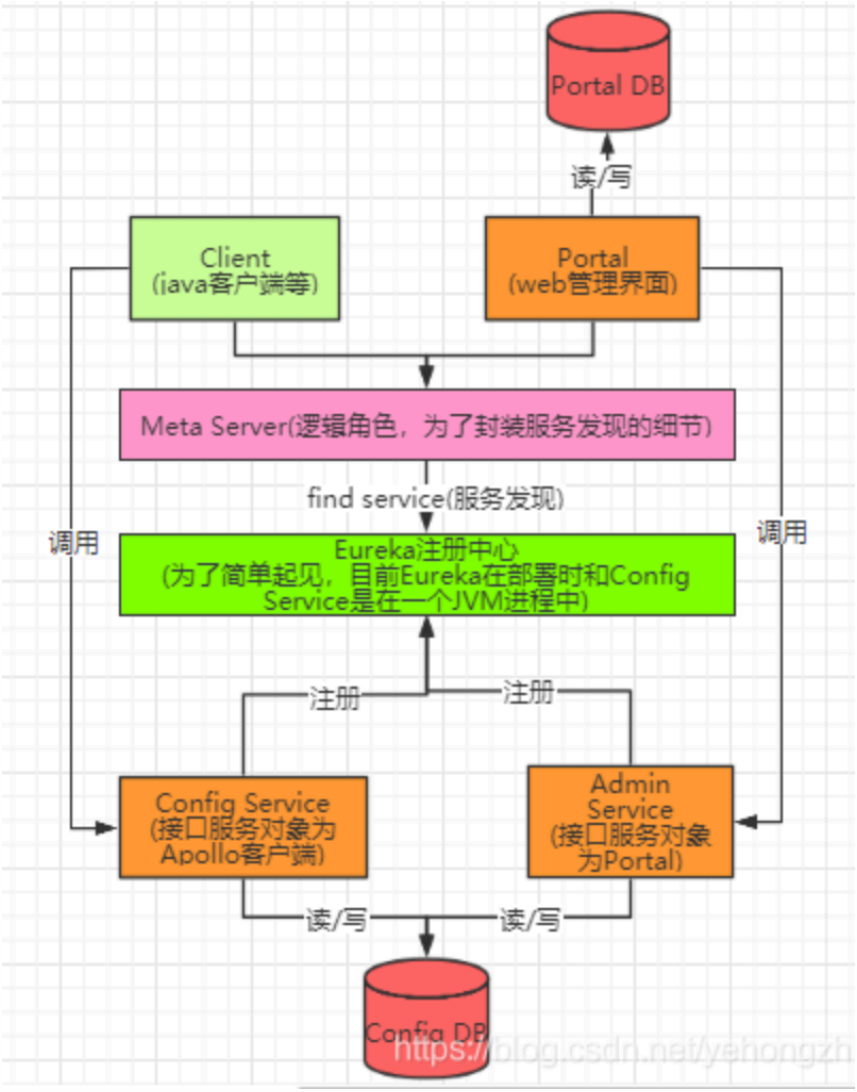
```
dev.meta=http://192.168.0.0:8080
# fat.meta=http://apollo.fat.xxx.com
# uat.meta=http://apollo.uat.xxx.com
# pro.meta=http://apollo.xxx.com
```

- 配置环境：Java-System-Property->VM options->配置
- 启动类上加注解@EnableApolloConfig
- 管理界面创建配置，则配置可以被请求

Apollo架构



可以参考如下架构：



Portal服务：提供Web界面供用户管理配置，通过MetaServer获取AdminService服务列表（IP+Port），通过IP+Port访问AdminService服务。

Client：实际上就是我们创建的SpringBoot项目，引入ApolloClient的maven依赖，为应用提供配置获取、实时更新等功能。

Meta Server：从Eureka获取Config Service和Admin Service的服务信息，相当于是一个Eureka Client。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。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，所以IP、端口和Config Service一致。

Eureka：注册中心。Config Service和Admin Service会向Eureka注册服务。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。

Config Service：提供配置获取接口。提供配置更新推送接口(基于Http long polling)。服务对象为Apollo客户端(Client)。

Admin Service：提供配置管理接口。提供配置发布、修改等接口。服务对象为Portal。

推送过程

1. 用户在Portal操作配置发布。
2. Portal调用Admin Service的接口操作发布。
3. Admin Service发布配置后，发送ReleaseMessage给各个Config Service。
4. Config Service收到ReleaseMessage后，通知对应的客户端(Client)。