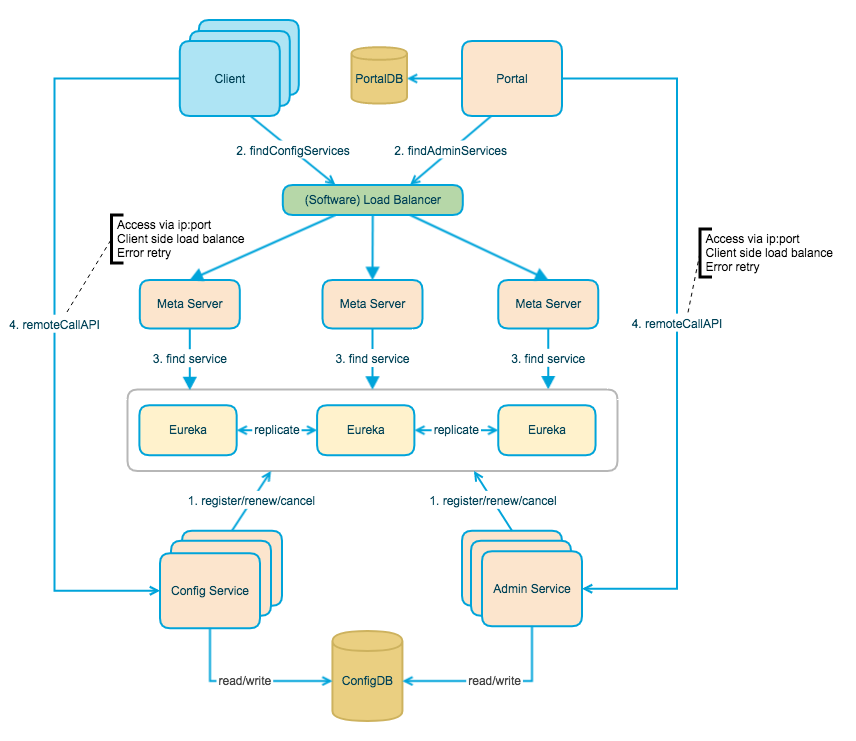
# Apollo设计和源码分析

## 架构模块介绍

### 架构设计图



### Apollo架构演进

https://mp.weixin.qq.com/s/-hUaQPzfsl9Lm3IqQW3VDQ

### 架构设计模块概要

### 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）
  + 服务端使用[Spring DeferredResult](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/context/request/async/DeferredResult.html)实现异步化，从而大大增加长连接数量
  + 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）。
* 接口服务对象为Apollo客户端

### Admin Service

* 提供配置管理接口
* 提供配置修改、发布等接口
* 接口服务对象为Portal

### Meta Server

* Portal通过域名访问Meta Server获取Admin Service服务列表（IP+Port）
* Client通过域名访问Meta Server获取Config Service服务列表（IP+Port）
* Meta Server从Eureka获取Config Service和Admin Service的服务信息，相当于是一个Eureka Client
* 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
* 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

### Eureka

* 基于[Eureka](https://github.com/Netflix/eureka)和[Spring Cloud Netflix](https://cloud.spring.io/spring-cloud-netflix/)提供服务注册和发现
* Config Service和Admin Service会向Eureka注册服务，并保持心跳
* 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）

### Portal

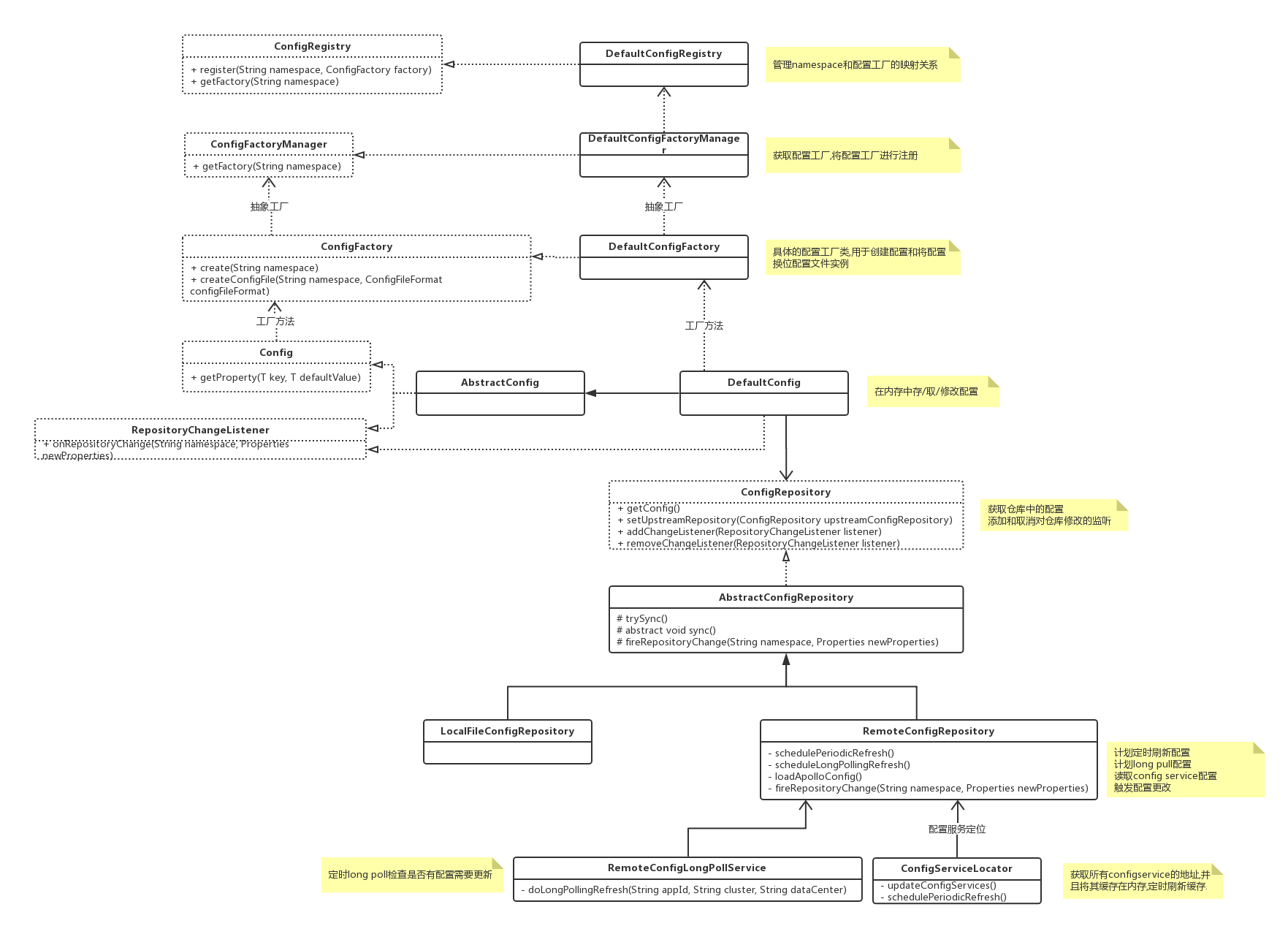
* 提供Web界面供用户管理配置
* 通过Meta Server获取Admin Service服务列表（IP+Port），通过IP+Port访问服务
* 在Portal侧做load balance、错误重试

### Client

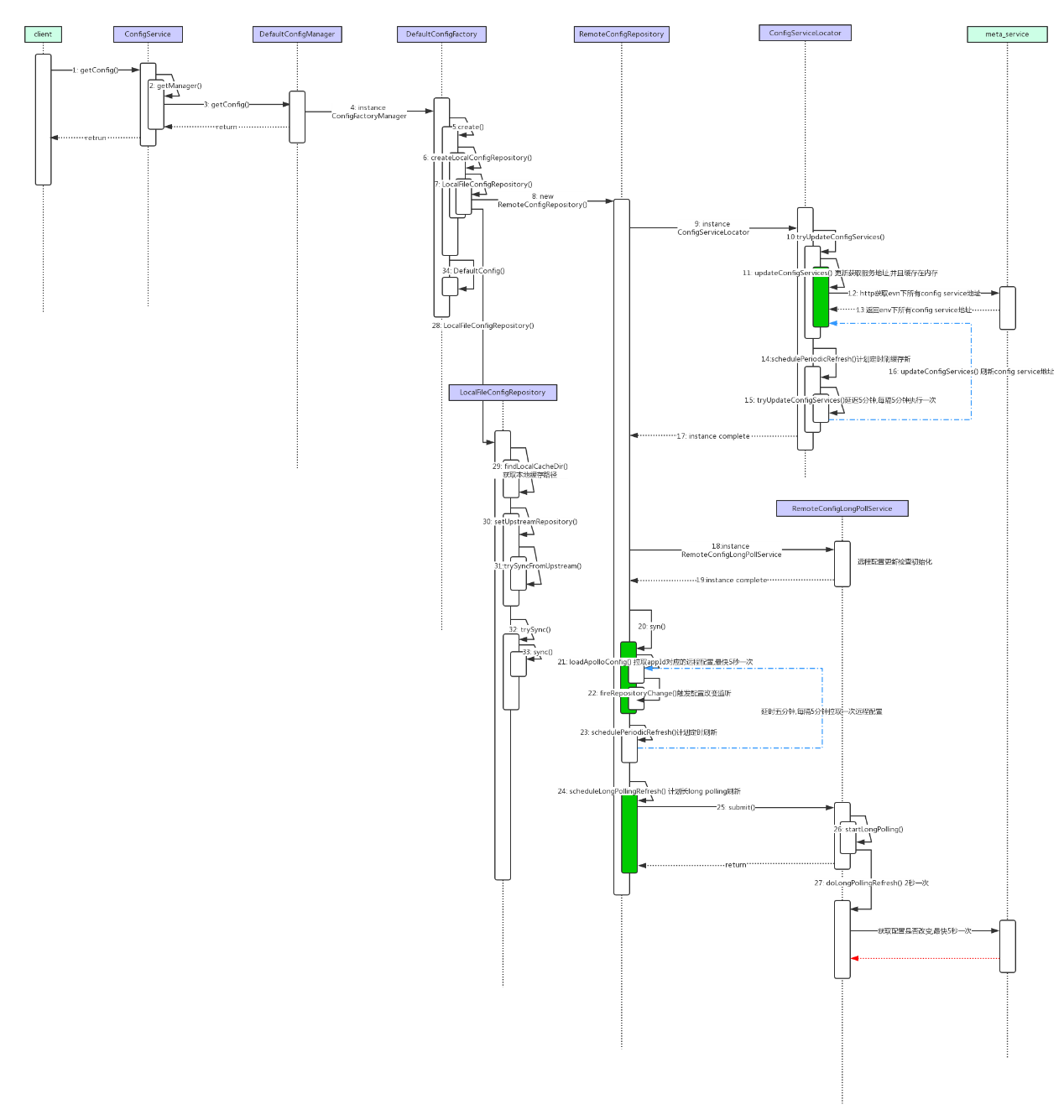
* Apollo提供的客户端程序，为应用提供配置获取、实时更新等功能
* 通过Meta Server获取Config Service服务列表（IP+Port），通过IP+Port访问服务
* 在Client侧做load balance、错误重试

## Apollo Client 核心代码review

### 关键类类图



### Java方式初始化加载时序图



### Spring Boot方式初始化加载关键代码review

类路径: [com](eclipse-javadoc:%E2%98%82=apollo-client/src%5C/main%5C/java%3Ccom).[ctrip](eclipse-javadoc:%E2%98%82=apollo-client/src%5C/main%5C/java%3Ccom.ctrip).[framework](eclipse-javadoc:%E2%98%82=apollo-client/src%5C/main%5C/java%3Ccom.ctrip.framework).[apollo](eclipse-javadoc:%E2%98%82=apollo-client/src%5C/main%5C/java%3Ccom.ctrip.framework.apollo).[spring](eclipse-javadoc:%E2%98%82=apollo-client/src%5C/main%5C/java%3Ccom.ctrip.framework.apollo.spring).[boot](eclipse-javadoc:%E2%98%82=apollo-client/src%5C/main%5C/java%3Ccom.ctrip.framework.apollo.spring.boot).ApolloApplicationContextInitializer

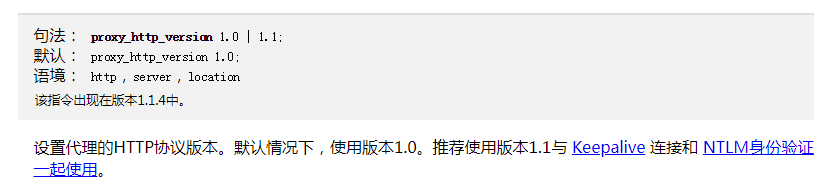
黄色背景标记代码备注:

* 实现ApplicationContextInitializer接口,在spring boot初始化执行initialize方法
* 读取环境中是否配置了apollo.bootstrap.enabled=true
* 读取环境中配置的apollo.bootstrap.namespaces=namespace1,namespace2…
* 初始化加载namespace,参考2.2

1. **public** **class** ApolloApplicationContextInitializer **implements**
2. ApplicationContextInitializer<ConfigurableApplicationContext> {
3. **private** **static** **final** Logger logger = LoggerFactory.getLogger(ApolloApplicationContextInitializer.**class**);
4. **private** **static** **final** Splitter NAMESPACE\_SPLITTER = Splitter.on(",").omitEmptyStrings().trimResults();
6. **private** **final** ConfigPropertySourceFactory configPropertySourceFactory = SpringInjector
7. .getInstance(ConfigPropertySourceFactory.**class**);
9. @Override
10. **public** **void** initialize(ConfigurableApplicationContext context) {
11. ConfigurableEnvironment environment = context.getEnvironment();
12. String enabled = environment.getProperty(PropertySourcesConstants.APOLLO\_BOOTSTRAP\_ENABLED, "false");
13. **if** (!Boolean.valueOf(enabled)) {
14. logger.debug("Apollo bootstrap config is not enabled for context {}, see property: ${{}}", context, PropertySourcesConstants.APOLLO\_BOOTSTRAP\_ENABLED);
15. **return**;
16. }
17. logger.debug("Apollo bootstrap config is enabled for context {}", context);
19. **if** (environment.getPropertySources().contains(PropertySourcesConstants.APOLLO\_BOOTSTRAP\_PROPERTY\_SOURCE\_NAME)) {
20. //already initialized
21. **return**;
22. }
24. String namespaces = environment.getProperty(PropertySourcesConstants.APOLLO\_BOOTSTRAP\_NAMESPACES, ConfigConsts.NAMESPACE\_APPLICATION);
25. logger.debug("Apollo bootstrap namespaces: {}", namespaces);
26. List<String> namespaceList = NAMESPACE\_SPLITTER.splitToList(namespaces);
28. CompositePropertySource composite = **new** CompositePropertySource(PropertySourcesConstants.APOLLO\_BOOTSTRAP\_PROPERTY\_SOURCE\_NAME);
29. **for** (String namespace : namespaceList) {
30. Config config = ConfigService.getConfig(namespace);
32. composite.addPropertySource(configPropertySourceFactory.getConfigPropertySource(namespace, config));
33. }
35. environment.getPropertySources().addFirst(composite);
36. }
37. }

### Apollo Client网络相关

* 协议类型:Http
* 协议版本:1.1
* 实现类: [java](eclipse-javadoc:%E2%98%82=apollo-client/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_102%5C/jre%5C/lib%5C/rt.jar%3Cjava).[net](eclipse-javadoc:%E2%98%82=apollo-client/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_102%5C/jre%5C/lib%5C/rt.jar%3Cjava.net).HttpURLConnection
* 设计上是否需要LSB负载转发:需要
* LSB(以Nginx为例)对该协议的版本支持



## Apollo configservice RESTful

主要包: [com](eclipse-javadoc:%E2%98%82=apollo-configservice/src%5C/main%5C/java%3Ccom).[ctrip](eclipse-javadoc:%E2%98%82=apollo-configservice/src%5C/main%5C/java%3Ccom.ctrip).[framework](eclipse-javadoc:%E2%98%82=apollo-configservice/src%5C/main%5C/java%3Ccom.ctrip.framework).[apollo](eclipse-javadoc:%E2%98%82=apollo-configservice/src%5C/main%5C/java%3Ccom.ctrip.framework.apollo).[configservice](eclipse-javadoc:%E2%98%82=apollo-configservice/src%5C/main%5C/java%3Ccom.ctrip.framework.apollo.configservice).[controller](eclipse-javadoc:%E2%98%82=apollo-configservice/src%5C/main%5C/java%3Ccom.ctrip.framework.apollo.configservice.controller).ConfigController

主要类:

