# 第二十九节 Spring Cloud Alibaba

## 内容回顾

## Spring Cloud Bootstrap 应用上下文



## Spring Cloud 服务注册与发现

### Enable 模块驱动 - @EnableDiscoveryClient

字面意思，激活服务发现，不过也包含服务注册，比如 `autoRegister()` 方法。由于 EnableDiscoveryClientImportSelector 中的 `hasDefaultFactory()` 方法永远返回 `true`，所以 `@EnableDiscoveryClient` 实际上是一个可选 Enable 模块注解。

在 Spring Cloud 服务注册与发现，即可使用 `@EnableDiscoveryClient` 来配置实现 Configuration Class，也可以通过 `@EnableAutoConfiguration` 来实现。

服务注册与发现并非是 Spring Cloud 的专利，它可以降级到 Spring Boot，甚至到 Spring Framework。

# Spring Cloud 配置架构

## 主要特性

- 加载配置
- 配置变化通知



## Spring Cloud Bootstrap PropertySource 架构

基于 org.springframework.cloud.bootstrap.config.PropertySourceLocator 实现，它也是 Spring Cloud Config Client/Server 架构的底层

## Spring Cloud Config Client/Server 架构

- 加载配置 - HTTP 方法获取 Config Server 返回内容，并且将内容转化为 PropertySources 添加到本地 Environment 中
- 配置变化通知 - 基于 Spring Cloud Bus 事件：
  - org.springframework.cloud.bus.event.EnvironmentChangeRemoteApplicationEvent（配置变化）
    - 监听器 - org.springframework.cloud.bus.event.EnvironmentChangeListener
      - 修改本地 Environment，即 PropertySources
  - org.springframework.cloud.bus.event.RefreshRemoteApplicationEvent（刷新）
    - 监听器 - org.springframework.cloud.bus.event.RefreshListener
      - 调用 ContextRefresher#refresh() 方法



## @RefreshScope Bean 原理

当 Spring Bean 被标注 `@RefreshScope` 之后，相当于 BeanDefinition 中的 scope 设置为 "refresh"

Spring Cloud 自定义了一个 "refresh" 的 Scope 实现，即 RefreshScope，区分是否标注 `@RefreshScope` Bean。

### ContextRefresher 原理

refresh() 方法

- refreshEnvironment() : 比较配置前后变化，并且封装成 EnvironmentChangeEvent
- RefreshScope 是一个自定义 Spring Bean Scope 接口实现，调用 refreshAll()
    - 调用 destroy() 方法：将所有标注 @RefreshScope Bean 调用销毁方法
    - 发送 RefreshScopeRefreshedEvent

# Spring Cloud LoadBalancer



## 主要实现

- 随机
- 轮询
- 一致性Hash
- 粘性实现（Stick）

## 随机实现 - org.springframework.cloud.loadbalancer.core.RandomLoadBalancer

Random 和 ThreadLocalRandom

## 轮询实现 - org.springframework.cloud.loadbalancer.core.RoundRobinLoadBalancer

假设服务实例 N 个（N 个元素的数组），position 字段代表游标，从 0 开始，每次调用负载均衡，position ++

posistion >= N，posistion - N = 新的 index



第一轮：posisition = [0 , N - 1] , index = [0 , N - 1]

第二轮：posisition = [N , 2* N - 1], index = [ P - 1, P - N -1 ]

当 posistion 超过最大值时，取绝对值（Math.abs）

**缺点：缺少权重算法**

## 其他实现

- Netflix Ribbon
- Apache Dubbo

- Alibaba Nacos

# Spring Cloud Circuit Breaker

## Enable 模块驱动 - @EnableCircuitBreaker

### Spring Boot 自动装配实现 - @EnableAutoConfiguration

- org.springframework.cloud.circuitbreaker.resilience4j.Resilience4JAutoConfiguration

### Spring Cloud 标准实现 - @EnableCircuitBreaker

- com.alibaba.cloud.sentinel.custom.SentinelCircuitBreakerConfiguration



# Spring Cloud Stream

## Spring Cloud Stream Binder 实现

屏蔽底层 Message 中间件实现细节，比如 Kafka、RabbitMQ 以及 RocketMQ 等

Spring Cloud Stream Binder 三个大版本，并且每个版本不兼容

# Apache Dubbo

# Spring Cloud Alibaba 实现

## 服务注册与发现 - Spring Cloud Alibaba Nacos Discovery



### 基本特性

- 服务注册与发现
  - 传统
  - Reactive
- Spring Boot
  - 自动装配
    - com.alibaba.cloud.nacos.discovery.NacosDiscoveryAutoConfiguration
    - com.alibaba.cloud.nacos.registry.NacosAutoServiceRegistration
  - 外部化配置

        - `@ConfigurationProperties` - com.alibaba.cloud.nacos.NacosDiscoveryProperties
    - Actuator
        - Health 指标 - com.alibaba.cloud.nacos.discovery.actuate.health.NacosDiscoveryHealthIndicator

Spring Cloud Alibaba Nacos Discovery 同样未使用 `@EnableDiscoveryClient` 标准注解实现。

# Spring Cloud Alibaba Nacos Config



基于 Spring Cloud Bootstrap PropertySource 架构实现，利用 Nacos Config Client 来实现配置加载和监听。

## 服务端实现

扩展 org.springframework.cloud.config.server.environment.EnvironmentRepository

## 配置启动加载实现 - com.alibaba.cloud.nacos.client.NacosPropertySourceLocator

## 配置变化通知实现 - com.alibaba.cloud.nacos.refresh.NacosContextRefresher

Nacos 配置变化监听器 API - com.alibaba.nacos.api.config.listener.Listener

Nacos 配置服务 API - com.alibaba.nacos.api.config.ConfigService#addListener



Spring Cloud 刷新事件 - org.springframework.cloud.endpoint.event.RefreshEvent

Spring Cloud 刷新事件监听器 - org.springframework.cloud.endpoint.event.RefreshEventListener

RefreshEventListener 会监听 RefreshEvent，当 RefreshEvent 发布时，RefreshEventListener 关联的 ContextRefresher 会被调用 `refresh()`，促使标注 `@RescopeScope` Bean 属性更新。如 Nacos Config 实现 NacosContextRefresher 会注册 AbstractSharedListener。

Nacos Config 变化 -> RefreshEvent 发布 -> RefreshEventListener 监听 -> ContextRefresher#refresh() 方法 -> 所有标注 `@RescopeScope` Bean 属性更新

# Spring Cloud Alibaba Nacos Config Server

## EnvironmentRepository 实现 - com.alibaba.cloud.nacos.config.server.environment.NacosEnvironmentRepository

# Spring Cloud Alibaba Sentinel



# Spring Cloud Alibaba Dubbo

https://www.bilibili.com/video/BV1gx411X7yy

# Spring Cloud Alibaba RocketMQ

## 扩展接口 - org.springframework.cloud.stream.binder.AbstractMessageChannelBinder

核心特性

- 消息读取
- 消息发送

# Spring Cloud Alibaba Seata

## 相关问题

### 问：Spring、Spring Web MVC、Spring Boot 和 Spring Cloud ApplicationContext 之间的关系



答：

Spring 应用上下文即 ApplicationContext 接口实现，主要有三种实现：

- 标准 ApplicationContext 实现 - org.springframework.context.support.GenericApplicationContext
- Servlet Web 实现（WebApplicationContext）- org.springframework.web.context.support.GenericWebApplicationContext
- Reactive Web 实现（Spring Boot 2.0）- org.springframework.boot.web.reactive.context.GenericReactiveWebApplicationContext

Spring Web MVC 主要两大核心组件：

- ContextLoaderListener - 继承 ContextLoader，而 ContextLoader 它会创建 Web ROOT ApplicationContext
- DispatcherServlet - 继承 FrameworkServlet
  - 默认情况，FrameworkServlet 创建一个 WebApplicationContext，如果 Web ROOT WebApplicationContext 存在的话，它会将 Web ROOT WebApplicationContext 作为自己的 Parent
  - 特殊情况，当前 FrameworkServlet 已经关联了 WebApplicationContext，将不会创建 WebApplicationContext
    - 关联 WebApplicationContext 来自于 setApplicationContext(AppliactionContext) 方法



相关实现：

```
protected WebApplicationContext
initWebApplicationContext() {
        WebApplicationContext rootContext
=

WebApplicationContextUtils.getWebApplicati
onContext(getServletContext());
        WebApplicationContext wac = null;

        if (this.webApplicationContext !=
null) {
```

```
            wac =
this.webApplicationContext;
            if (wac instanceof
ConfigurableWebApplicationContext) {

ConfigurableWebApplicationContext cwac =
(ConfigurableWebApplicationContext) wac;
                if (!cwac.isActive()) {
                    if (cwac.getParent()
== null) {

cwac.setParent(rootContext);
                    }

configureAndRefreshWebApplicationContext(c
wac);
            }
            }
        }
        if (wac == null) {
            wac =
findWebApplicationContext();
        }
        if (wac == null) {
            wac =
createWebApplicationContext(rootContext);
        }
        ...

        return wac;
    }
```

假设 Spring Web MVC 创建了两个 WebApplicationContext，其中 ROOT 主要加载服务组件 Bean，这些 Bean 给 DispatcherServlet 的 WebApplicationContext 来复用。比如，通常配置 ContextLoaderListener 会在 web.xml 文件中关联一个`contextConfigLocation`的参数：

```
    <context-param>
        <param-
name>contextConfigLocation</param-name>
        <param-value>classpath:**/META-
INF/spring-context.xml</param-value>
    </context-param>

    <listener>
        <listener-
class>org.springframework.web.context.ContextLo
aderListener</listener-class>
    </listener>
```



特殊情况，复用 ROOT WebApplicationContext 中的 Bean，实际上来自于 org.springframework.web.servlet.FrameworkServlet，它的一个著名子类即：org.springframework.web.servlet.DispatcherServlet，然而在其他框架实现中，有可能有其他 FrameworkServlet 实现。

Spring Boot Web 功能激活后：

- DispatcherServlet 将会作为一个 Spring Bean 被自动装配，同时，将 SpringApplication 所创建 WebApplicationContext 主动注入到 DispatcherServlet#setApplicationContext(ApplicationContext) 方法中（因为 DispatcherServlet 继承了 FrameworkServlet，并且 FrameworkServlet 实现了 ApplicationContextAware 接口），所以，在这种情况下，DispatcherServlet 不会创建自己的 WebApplicationContext 对象（在 Servlet 引擎初始化当前实例）。因此，Spring Boot Web 应用，默认只有一个 WebApplicationContext 实例。

Spring Cloud 应用上下文：



- 默认情况，创建一个 Bootstrap ApplicationContext（非 Web），它将作为 Spring Boot 主 ApplicationContext 的 Parent

# 问：如何设计一个抢红包的功能?

1 亿红包，预计有 10 亿用户（机器人）来抢，红包金额 0.1 元到 1元（随机） [0 ,1]

# 实时实现

中心化：

负载均衡，计算成本（1000 机器），每台机器要抗住 1000000 并发流量

识别流量（人、机器）

分流和限流（数据来自于压测）

友好的用户提醒（当服务器发生异常，提示“抱歉，没有抢到”）

# 非实时实现



去中心化：提前把红包发出去，打开客户端（一个星期预告期），丢到客户端