

服务注册与发现



扫码试看/订阅
《玩转 Spring 全家桶》

使用 Eureka 作为服务注册中心

认识 Eureka

什么是 Eureka

- Eureka 是在 AWS 上定位服务的 REST 服务

Netflix OSS 不在AWS上时候用发布在Netflix中。但是不在AWS上不建议用
现在2.0版本已经不再开源

- <https://netflix.github.io>

Spring 对 Netflix 套件的支持

- Spring Cloud Netflix

在本地启动一个简单的 Eureka 服务

Starter

- spring-cloud-dependencies
- spring-cloud-starter-netflix-eureka-starter

声明 在config类上

- @EnableEurekaServer 这是单机的，要部署一个集群来用！！！！

注意事项

- 默认端口8761 Localhost:8761 直接打开
- Eureka 自己不要注册到 Eureka 了

将服务注册到 Eureka Server

Starter 如果俩注解都不加，classpath里面有这个依赖的话也行

- spring-cloud-starter-netflix-eureka-client

声明

- @EnableDiscoveryClient 这是个集成的注解
- @EnableEurekaClient

一些配置项

- eureka.client.service-url.default-zone
- eureka.client.instance.prefer-ip-address 优先以IP地址

关于 Bootstrap 属性

跟配置application.properties一样的,这里配置bootstrap.properties

Bootstrap 属性

- 启动引导阶段加载的属性
- bootstrap.properties | .yml
- spring.cloud.bootstrap.name=bootstrap

常用配置

- spring.application.name=应用名
- 配置中心相关

注意jaxb依赖在JDK11里去掉了，需要自己加入，jdk8还有

“Talk is cheap, show me the code.”

Chapter 12 / eureka-server eureka-waiter-service

使用 Spring Cloud LoadBalancer 访问服务

`server.port = 0` ——随机选一个端口

如何获得服务地址

EurekaClient

- getNextServerFromEureka()

DiscoveryClient springCloud给的抽象，建议使用

- getInstances()

Load Balancer Client

增强了RestTemplate

RestTemplate 与 WebClient

- @LoadBalanced 为restTemplate或者WebClient增加负载均衡支持
- 实际是通过 ClientHttpRequestInterceptor 实现的 接口
- LoadBalancerInterceptor  ribbon是netflix套件提供的
- LoadBalancerClient  获取目标地址
- RibbonLoadBalancerClient

“Talk is cheap, show me the code.”

Chapter 12 / ribbon-customer-service

```

@Configuration
@ConditionalOnClass(RestTemplate.class)
@ConditionalOnBean(LoadBalancerClient.class)
@EnableConfigurationProperties(LoadBalancerRetryProperties.class)
public class LoadBalancerAutoConfiguration {

```

```

    @LoadBalanced
    @Autowired(required = false)
    private List<RestTemplate> restTemplatees = Collections.emptyList();

    @Autowired(required = false)
    private List<LoadBalancerRequestTransformer> transformers = Collections.emptyList();

```

```

@Configuration
@ConditionalOnMissingClass("org.springframework.retry.support.RetryTemplate")
static class LoadBalancerInterceptorConfig {

    @Bean
    public LoadBalancerInterceptor ribbonInterceptor(
        LoadBalancerClient loadBalancerClient,
        LoadBalancerRequestFactory requestFactory) {
        return new LoadBalancerInterceptor(loadBalancerClient, requestFactory);
    }

    @Bean
    @ConditionalOnMissingBean
    public RestTemplateCustomizer restTemplateCustomizer(
        final LoadBalancerInterceptor loadBalancerInterceptor) {
        return restTemplate -> {
            List<ClientHttpRequestInterceptor> list = new ArrayList<>(
                restTemplate.getInterceptors());
            list.add(loadBalancerInterceptor);
            restTemplate.setInterceptors(list);
        };
    }
}

```


Maven: org.springframework.boot:spring-boot-test-autoconfigure:2.1.1.RELEASE

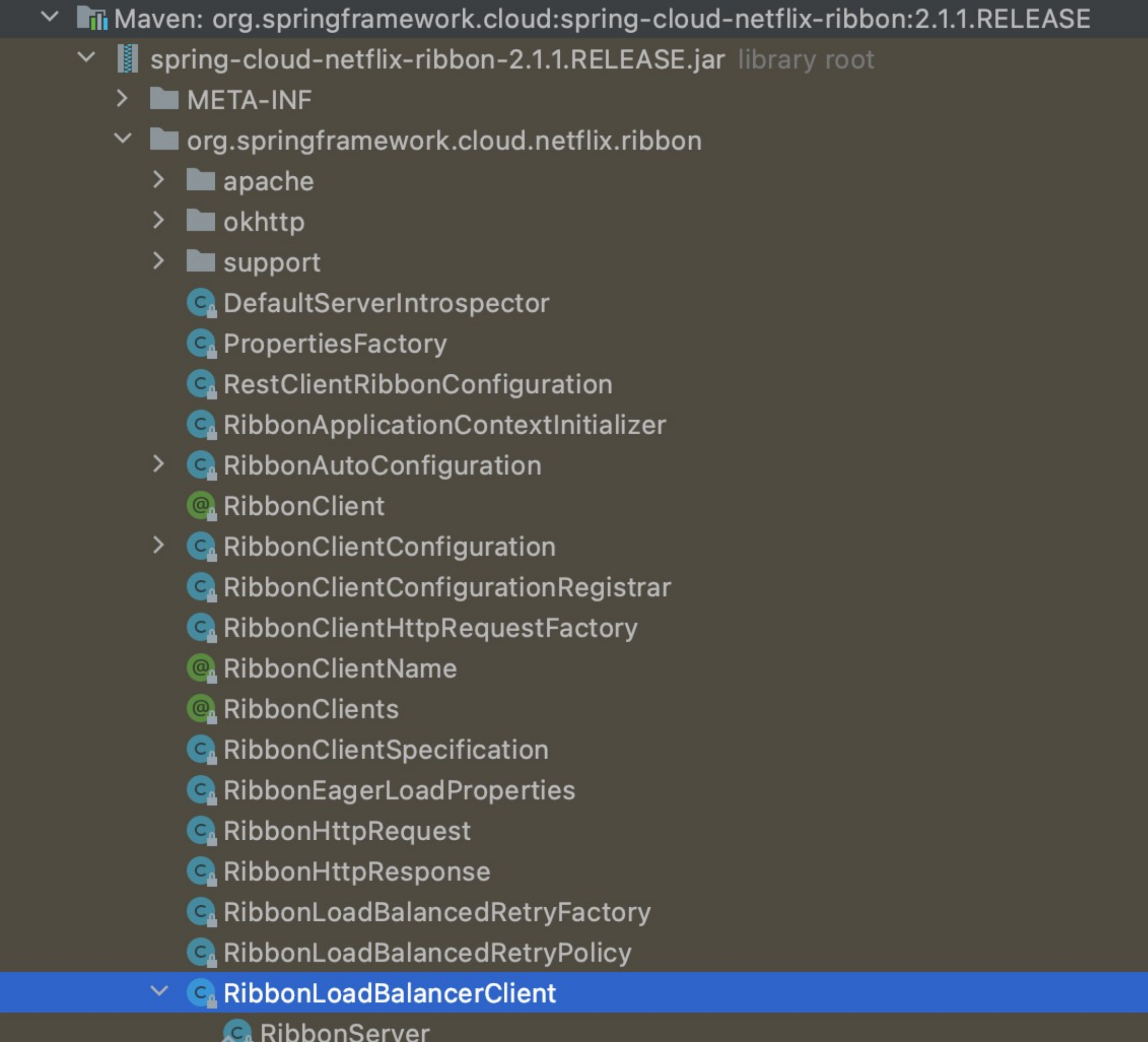
▼ Maven: org.springframework.cloud:spring-cloud-commons:2.1.1.RELEASE

▼ spring-cloud-commons-2.1.1.RELEASE.jar library root

- > META-INF
- ▼ org.springframework.cloud
 - ▼ client
 - > actuator
 - > circuitbreaker
 - > discovery
 - > hypermedia
 - ▼ loadbalancer
 - > reactive
 - > AsyncLoadBalancerAutoConfiguration
 - > AsyncLoadBalancerInterceptor
 - > AsyncRestTemplateCustomizer
 - > ClientHttpResponseStatusCodeException
 - > ClientHttpRequestPolicy
 - Read-only class
 - > LoadBalancedRecoveryCallback
 - > LoadBalancedRetryContext
 - > LoadBalancedRetryFactory
 - > LoadBalancedRetryPolicy
 - > LoadBalancerAutoConfiguration
 - > LoadBalancerClient
 - LoadBalancerInterceptor
 - > LoadBalancerRequest
 - > LoadBalancerRequestFactory
 - > LoadBalancerRequestTransformer
 - > LoadBalancerRetryProperties
 - > RestTemplateCustomizer
 - > RetryableStatusCodeException
 - > RetryLoadBalancerInterceptor
 - > ServiceInstanceChooser
 - > ServiceRequestWrapper

Author: Spencer Gibb, Dave Syer, Ryan Baxter, William Tran

```
34 public class LoadBalancerInterceptor implements ClientHttpRequestInterceptor {
35
36     private LoadBalancerClient loadBalancer;
37
38     private LoadBalancerRequestFactory requestFactory;
39
40     public LoadBalancerInterceptor(LoadBalancerClient loadBalancer,
41         LoadBalancerRequestFactory requestFactory) {
42         this.loadBalancer = loadBalancer;
43         this.requestFactory = requestFactory;
44     }
45
46     public LoadBalancerInterceptor(LoadBalancerClient loadBalancer) {
47         // for backwards compatibility
48         this(loadBalancer, new LoadBalancerRequestFactory(loadBalancer));
49     }
50
51     @Override
52     public ClientHttpResponse intercept(final HttpRequest request, final byte[] body,
53         final ClientHttpRequestExecution execution) throws IOException {
54         final URI originalUri = request.getURI();
55         String serviceName = originalUri.getHost();
56         Assert.state(expression: serviceName != null,
57             message: "Request URI does not contain a valid hostname: " + originalUri);
58         return this.loadBalancer.execute(serviceName,
59             this.requestFactory.createRequest(request, body, execution));
60     }
61
62 }
```

```
@Override
public <T> T execute(String serviceId, ServiceInstance serviceInstance,
    LoadBalancerRequest<T> request) throws IOException {
    Server server = null;
    if (serviceInstance instanceof RibbonServer) {
        server = ((RibbonServer) serviceInstance).getServer();
    }
    if (server == null) {
        throw new IllegalStateException("No instances available for " + serviceId);
    }

    RibbonLoadBalancerContext context = this.clientFactory
        .getLoadBalancerContext(serviceId);
    RibbonStatsRecorder statsRecorder = new RibbonStatsRecorder(context, server);

    try {
        T returnVal = request.apply(serviceInstance);
        statsRecorder.recordStats(returnVal);
        return returnVal;
    }
    // catch IOException and rethrow so RestTemplate behaves correctly
    catch (IOException ex) {
        statsRecorder.recordStats(ex);
        throw ex;
    }
    catch (Exception ex) {
        statsRecorder.recordStats(ex);
        ReflectionUtils.rethrowRuntimeException(ex);
    }
    return null;
}
```

使用 Feign 访问服务

认识 Feign

Feign

- 声明式 REST Web 服务客户端
- <https://github.com/OpenFeign/feign>

Spring Cloud OpenFeign

- `spring-cloud-starter-openfeign`

Feign 的简单使用

开启 Feign 支持

- `@EnableFeignClients`

定义 Feign 接口

- `@FeignClient` 添加了这个注解的接口会被实例化一个 Bean

简单配置

- `FeignClientsConfiguration`
- `Encoder` / `Decoder` / `Logger` / `Contract` / `Client` ...

不要把 `@RequestMapping` 加到接口上，直接用 `@FeignClient(path= “”)`

通过配置定制 Feign

yml方式

```
feign:
  client:
    config:
      feignName:
        connectTimeout: 5000
        readTimeout: 5000
        loggerLevel: full
        errorDecoder: com.example.SimpleErrorDecoder
        retryer: com.example.SimpleRetryer
        requestInterceptors:
          - com.example.FooRequestInterceptor
          - com.example.BarRequestInterceptor
        decode404: false
        encoder: com.example.SimpleEncoder
        decoder: com.example.SimpleDecoder
        contract: com.example.SimpleContract
```

Feign 的一些其他配置

使用application.properties配置

- feign.okhttp.enabled=true
- feign.httpClient.enabled=true
支持httpClient的话就要放进去httpClient的包
- feign.compression.response.enabled=true
- feign.compression.request.enabled=true
- feign.compression.request.mime-types=
text/xml,application/xml,application/json
对xml和
- feign.compression.request.min-request-size=2048

“Talk is cheap, show me the code.”

Chapter 12 / feign-customer-service

深入理解 DiscoveryClient

Spring Cloud Commons 提供的抽象

服务注册抽象

- 提供了 ServiceRegistry 抽象

客户发现抽象

- 提供了 DiscoveryClient 抽象
- @EnableDiscoveryClient
- 提供了 LoadBalancerClient 抽象

自动向 Eureka 服务端注册

ServiceRegistry

- EurekaServiceRegistry
- EurekaRegistration

自动配置

- EurekaClientAutoConfiguration
- EurekaAutoServiceRegistration
- SmartLifecycle

使用 Zookeeper 作为服务注册中心

认识 Zookeeper

Zookeeper

- A Distributed Coordination Service for Distributed Applications
- <http://zookeeper.apache.org>

设计目标

- 简单
- 多副本
- 有序
- 快



使用 Zookeeper 作为注册中心

Spring Cloud Zookeeper

- spring-cloud-starter-zookeeper-discovery
- Apache Curator

简单配置

- `spring.cloud.zookeeper.connect-string=localhost:2181`

提示

- 注意 Zookeeper 的版本
 - 3.5.x 还是 Beta, 但很多人在生产中使用它

使用 Zookeeper 作为注册中心的问题

两篇文章值得阅读

- 《阿里巴巴为什么不用 Zookeeper 做服务发现》
- 《Eureka! Why You Shouldn't Use ZooKeeper for Service Discovery》

核心思想

- 在实践中，注册中心不能因为自身的任何原因破坏服务之间本身的可连通性
- 注册中心需要 AP，而 Zookeeper 是 CP
 - CAP - 一致性、可用性、分区容忍性

通过 Docker 启动 Zookeeper

官方指引

- https://hub.docker.com/_/zookeeper

获取镜像

- `docker pull zookeeper:3.5`

运行 Zookeeper 镜像

- `docker run --name zookeeper -p 2181:2181 -d zookeeper:3.5`

“Talk is cheap, show me the code.”

Chapter 12 / zk-waiter-service zk-customer-service

使用 Consul 作为服务注册中心

“Consul is a distributed, highly available, and data center aware solution to connect and configure applications across dynamic, distributed infrastructure.”

– <https://github.com/hashicorp/consul>

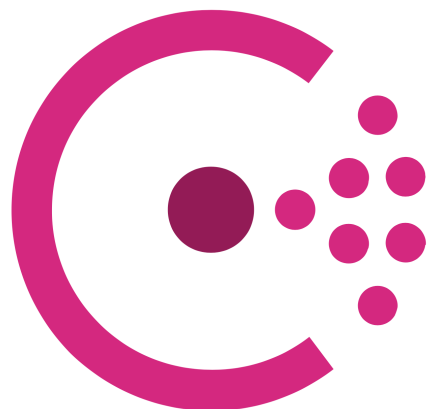
认识 HashiCorp Consul

Consul

- <https://www.consul.io>

关键特性

- 服务发现
- 健康检查
- KV 存储
- 多数据中心支持
- 安全的服务间通信



使用 Consul 提供服务发现能力

Consul 的能力

- Service registry, integrated health checks, and DNS and HTTP interfaces enable any service to discover and be discovered by other services

好用的功能

- HTTP API
- DNS (xxx.service.consul)
- 与 Nginx 联动, 比如 ngx_http_consul_backend_module

使用 Consul 作为注册中心

Spring Cloud Consul

- `spring-cloud-starter-consul-discovery`

简单配置

- `spring.cloud.consul.host=localhost`
- `spring.cloud.consul.port=8500`
- `spring.cloud.consul.discovery.prefer-ip-address=true`

通过 Docker 启动 Consul

官方指引

- https://hub.docker.com/_/consul

获取镜像

- `docker pull consul`

运行 Consul 镜像

- `docker run --name consul -d -p 8500:8500 -p 8600:8600/udp consul`

“Talk is cheap, show me the code.”

Chapter 12 / consul-waiter-service consul-customer-service

使用 Nacos 作为服务注册中心

认识 Nacos

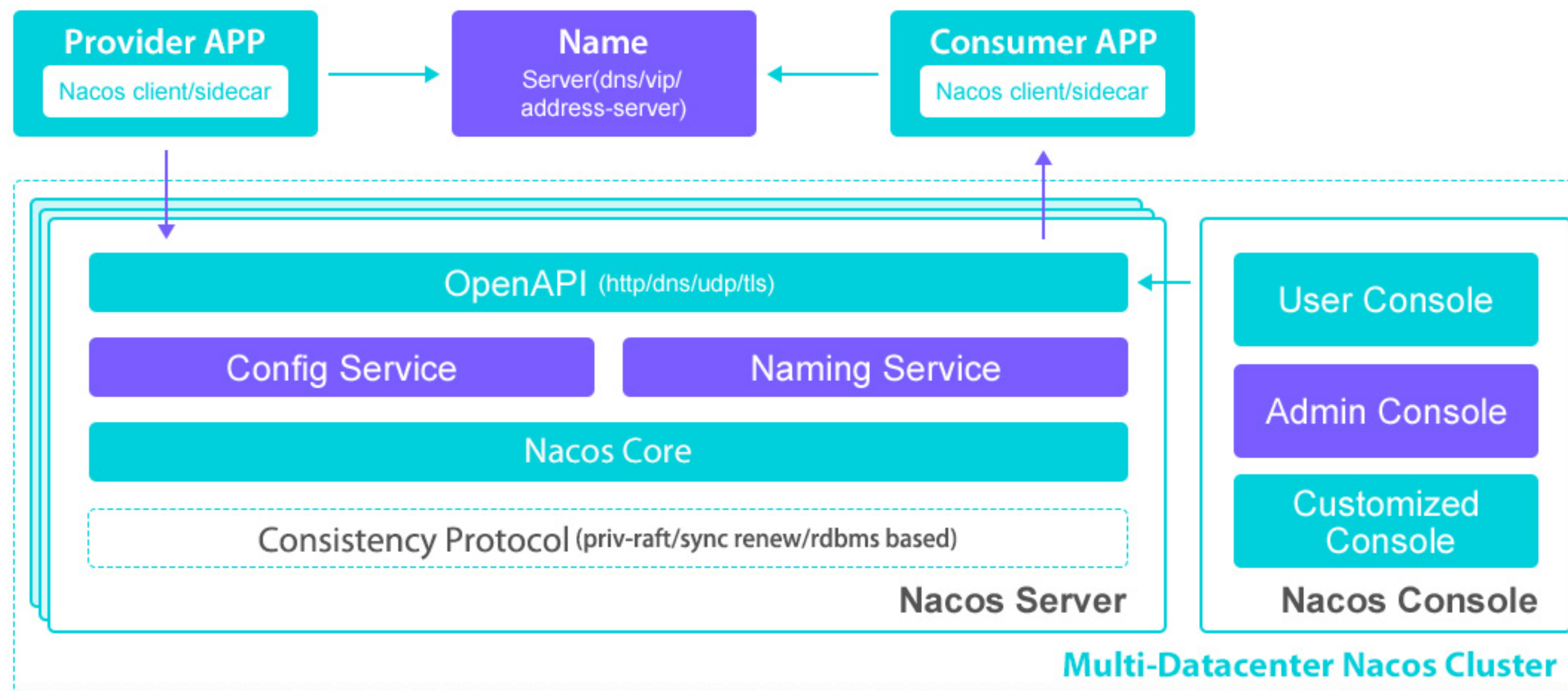
Nacos

- 一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。
- <https://nacos.io/zh-cn/index.html>

功能

- 动态服务配置
- 服务发现和管理
- 动态 DNS 服务

认识 Nacos



使用 Nacos 作为注册中心

Spring Cloud Alibaba

- `spring-cloud-alibaba-dependencies`
- `spring-cloud-starter-alibaba-nacos-discovery`

简单配置

- `spring.cloud.nacos.discovery.server-addr`

通过 Docker 启动 Nacos

官方指引

- <https://hub.docker.com/r/nacos/nacos-server>

获取镜像

- `docker pull nacos/nacos-server`

运行 Nacos 镜像

- `docker run --name nacos -d -p 8848:8848 -e MODE=standalone nacos/nacos-server`
- 用户名密码为 nacos

“Talk is cheap, show me the code.”

Chapter 12 / nacos-waiter-service nacos-customer-service

如何定制自己的 DiscoveryClient

已经接触过的 Spring Cloud 类

DiscoveryClient

- EurekaDiscoveryClient
- ZookeeperDiscoveryClient
- ConsulDiscoveryClient
- NacosDiscoveryClient

LoadBalancerClient

- RibbonLoadBalancerClient

实现自己的 DiscoveryClient

需要做的：

- 返回该 DiscoveryClient 能提供的服务名列表
- 返回指定服务对应的 ServiceInstance 列表
- 返回 DiscoveryClient 的顺序
- 返回 HealthIndicator 里显示的描述

实现自己的 RibbonClient 支持

需要做的：

- 实现自己的 `ServerList<T extends Server>`
- Ribbon 提供了 `AbstractServerList<T extends Server>`
- 提供一个配置类，声明 `ServerList Bean` 实例

“Talk is cheap, show me the code.”

Chapter 12 / fixed-discovery-client-demo

SpringBucks 实战项目进度小结

本章小结

各种服务注册中心

- Eureka、Zookeeper、Consul、Nacos

如何在服务间进行负载均衡

- Ribbon、OpenFeign

Spring Cloud 的服务注册与发现机制

- ServiceRegistry、DiscoveryClient
- LoadBalancerClient

SpringBucks 进度小结

waiter-service

- 使用多种服务注册中心注册服务
 - Eureka、Zookeeper、Consul、Nacos

customer-service

- 通过多种服务注册中心发现 waiter-service
 - Eureka、Zookeeper、Consul、Nacos



扫码试看/订阅
《玩转 Spring 全家桶》